Algoritmusok Bonyolultsága

Egyetemi Jegyzet

Lovász László

Eötvös Loránd Tudományegyetem

Matematikai Intézet

Szerkesztette, javította:
Király Zoltán[1]

[1] 2009. május 27.

Tartalomjegyzék

Előszó

Az az igény, hogy egy feladat, algoritmus vagy struktúra bonyolultságát számszerűen mérni tudjuk, és ennek alapján e bonyolultságra korlátokat és számszerű összefüggéseket nyerjünk, egyre több tudományág területén vetődik fel: a számítógéptudományon kívül a matematika hagyományos ágai, a statisztikus fizika, a biológia, az orvostudomány, a társadalomtudományok és a mérnöki tudományok is egyre gyakrabban kerülnek szembe ezzel a kérdéssel. A számítógéptudomány ezt a problémát úgy közelíti meg, hogy egy feladat elvégzéséhez szükséges számítástechnikai erőforrások (idő, tár, program, kommunikáció) mennyiségével méri a feladat bonyolultságát. Ennek az elméletnek az alapjaival foglalkozik ez a jegyzet.

A bonyolultságelmélet alapvetően három különböző jellegű részre oszlik. Először is, be kell vezetni az algoritmus, idő, tár stb. pontos fogalmát. Ehhez a matematikai gép különböző modelljeit kell definiálni, és az ezeken elvégzett számítások tár- és időigényét tisztázni (ezt általában a bemenet méretének függvényében mérjük). Az erőforrások korlátozásával a megoldható feladatok köre is szűkül; így jutunk a különböző bonyolultsági osztályokhoz. A legalapvetőbb bonyolultsági osztályok a matematika klasszikus területein felvetődő problémáknak is fontos, és a gyakorlati és elméleti nehézséget jól tükröző osztályozását adják. Ide tartozik a különböző modellek egymáshoz való viszonyának vizsgálata is.

Másodszor, meg kell vizsgálni, hogy a legfontosabb algoritmusok a matematika különböző területein milyen erőforrás-igényűek, ill. hatékony algoritmusokat kell megadni annak igazolására, hogy egyes fontos feladatok milyen bonyolultsági osztályokba esnek. Ebben a jegyzetben a konkrét algoritmusok ill. feladatok vizsgálatában nem törekszünk teljességre; ez az egyes matematikai tárgyak (kombinatorika, operációkutatás, numerikus analízis, számelmélet) dolga.

Harmadszor, módszereket kell találni „negatív eredmények" bizonyítására, vagyis annak igazolására, hogy egyes feladatok nem is oldhatók meg bizonyos erőforrás-korlátozások mellett. Ezek a kérdések gyakran úgy is fogalmazhatók, hogy a bevezetett bonyolultsági osztályok különbözőek-e ill. nem üresek-e. Ennek a problémakörnek része annak a vizsgálata, hogy egy feladat megoldható-e egyáltalán algoritmikusan; ez ma már klasszikus kérdésnek tekinthető, és sok fontos eredmény van vele kapcsolatban. A gyakorlatban felvetődő algoritmikus problémák többsége azonban olyan, hogy algoritmikus megoldhatósága önmagában nem kérdéses, csak az a kérdés, hogy milyen erőforrásokat kell ehhez felhasználni. Az ilyen alsó korlátokra vonatkozó vizsgálatok igen nehezek, és még gyerekcipőben járnak. Ebben a jegyzetben is csak ízelítőül tudunk bemutatni néhány ilyen jellegű eredményt.

Végül érdemes még megjegyezni, hogy ha egy feladatról kiderül, hogy csak „nehezen" oldható meg, ez nem szükségképpen negatív eredmény. Egyre több területen (véletlen

számok generálása, kommunikációs protokollok, titkosírások, adatvédelem) van szükség garantáltan bonyolult problémákra és struktúrákra. Ezek a bonyolultságelmélet fontos alkalmazási területei; közülük a titkosírások elméletével, a kriptográfiával foglalkozunk a 11. fejezetben.

A jegyzetben felhasználjuk a számelmélet, lineáris algebra, gráfelmélet és (kisebb mértékben) a valószínűségelmélet alapfogalmait. Ezek azonban főleg példákban szerepelnek, az elméleti eredmények — kevés kivétellel — ezek nélkül is érthetők.

Köszönettel tartozom BABAI LÁSZLÓNAK, ELEKES GYÖRGYNEK, FRANK ANDRÁSNAK, KATONA GYULÁNAK, KIRÁLY ZOLTÁNNAK, RÁCZ ANDRÁSNAK és SIMONOVITS MIKLÓSNAK a kézirattal kapcsolatos tanácsaikért, és MIKLÓS DEZSŐNEK a MATEX programcsomag használatában nyújtott segítségéért. GÁCS PÉTER jegyzetemet angolra fordította, és eközben számos lényeges kiegészítést, javítást tett hozzá; ezekből is többet felhasználtam a jelen változat elkészítésekor.

Lovász László

A csak angolul meglévő részek magyarra fordításában, a csak papíron megtalálható részek begépelésében, valamint a hibakeresésben a segítségemre voltak: KESZEGH BALÁZS, PÁLVÖLGYI DÖMÖTÖR és KIRÁLY CSABA.

a szerkesztő

Néhány jelölés és definíció

Egy tetszőleges véges halmazt *ábécének* is nevezünk. A Σ ábécé elemeiből alkotott véges sorozatot Σ fölötti *szónak* hívjuk. Ezek közé tartozik az üres szó is, melyet $\emptyset$ jelöl. A szó *hossza* a benne szereplő betűk száma. A Σ ábécé fölötti n hosszúságú szavak halmazát Σ^n-nel, az összes Σ fölötti szó halmazát $\Sigma*$-gal jelöljük. $\Sigma*$ egy részhalmazát (vagyis szavak egy tetszőleges halmazát) *nyelvnek* hívjuk.

A Σ fölötti szavaknak többféle sorbarendezésére is szükségünk lesz. Feltesszük, hogy a Σ elemeinek adott egy sorrendje. A *lexikografikus rendezésben* egy α szó megelőz egy β szót, ha vagy kezdőszelete (prefixe), vagy az első olyan betű, amely nem azonos a két szóban, az α szóban kisebb (az ábécé rendezése szerint). A lexikografikus rendezés nem rendezi egyetlen sorozatba a szavakat; pl. a $\{0,1\}$ ábécé fölött az „1” szót minden 0-val kezdődő szó megelőzi. Ezért sokszor jobban használható a *növekvő rendezés*: ebben minden rövidebb szó megelőz minden hosszabb szót, az azonos hosszúságú szavak pedig lexikografikusan vannak rendezve. Ha a pozitív egész számokat növekvő sorrendben, kettes számrendszerben írjuk le, majd a kezdő 1-est levágjuk, a $\{0,1\}*$ növekvő rendezését kapjuk.

A valós számok halmazát R, az egész számokét Z, a racionális számokét Q jelöli. A nemnegatív valós (egész, racionális) számok halmazának jele R_+. (Z_+, Q_+.). A logaritmus, ha alapja nincs külön föltüntetve, mindig 2-es alapú logaritmust jelent.

Legyen f és g két, természetes számokon értelmezett, komplex értékű függvény. Azt írjuk, hogy

$$f = O(g),$$

ha van olyan $c > 0$ konstans és olyan $n_0 \in Z_+$ küszöb, hogy minden $n > n_0$ esetén $|f(n) \leq c|g(n)|$. Azt írjuk, hogy

$$f = o(g),$$

ha $g(n)$ csak véges sok helyen nulla, és $f(n)/g(n) \to 0$ ha $n \to \infty$. A nagy-O jelölés kevésbé általánosan használt megfordítása:

$$f = \Omega(g)$$

ha $g = O(f)$. Az

$$f = \Theta(g)$$

jelölés azt jelenti, hogy $f = O(g)$ és $f = \Omega(g)$, vagyis vannak olyan $c_1, c_2 > 0$ konstansok és olyan $n_0 \in Z_+$ küszöb, hogy minden $n > n_0$ esetén $c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|$.

Ezeket a jelöléseket formulákon belül is használjuk. Például

$$(n+1)^2 = n^2 + O(n)$$

azt jelenti, hogy $(n+l)^2$ felírható $n^2+R(n)$ alakban, ahol az $R(n)$ függvényre $R(n) = O(n)$ teljesül. (Az ilyen formulákban az egyenlőség nem szimmetrikus! Pl. $O(n) = O(n^2)$, de nem áll az, hogy $O(n^2) = O(n)$.)

1. fejezet

Számítási modellek

Ebben a fejezetben az algoritmus fogalmát tárgyaljuk. Ez a fogalom témánk szempontjából alapvető, mégsem definiáljuk. Inkább olyan intuitív fogalomnak tekintjük, melynek formalizálására (és ezzel matematikai szempontból való vizsgálhatóságára) különféle lehetőségek vannak. Az algoritmus olyan matematikai eljárást jelent, mely valamely számítás vagy konstrukció elvégzésére — valamely függvény kiszámítására — szolgál, és melyet gondolkodás nélkül, gépiesen lehet végrehajtani. Ezért az algoritmus fogalma helyett a *matematikai gép* különböző fogalmait vezetjük be.

Minden matematikai gép valamilyen *bemenetből* valamilyen *kimenetet* számít ki. A bemenet és kimenet lehet pl. egy rögzített ábécé feletti szó (véges sorozat), vagy számok egy sorozata. A gép a számításhoz különböző erőforrásokat (pl. idő, tár, kommunikáció) vesz igénybe, és a számítás bonyolultságát azzal mérjük, hogy az egyes erőforrásokból mennyit használ fel.

Talán a legegyszerűbb gép a *véges automata*. Ezzel itt nem foglalkozunk, részben mert külön tárgy foglalkozik ezzel a modellel, részben pedig, mert a bonyolultságelmélet céljaira túlságosan primitív: csak nagyon egyszerű függvények kiszámítására alkalmas.

A számítások legrégibb, legismertebb és matematikai szempontból „legtisztább" modellje a *Turing-gép*. Ezt a fogalmat A. Turing angol matematikus vezette be 1936-ban, tehát még a programvezérlésű számítógépek megalkotása előtt. Lényege egy korlátos (bemenettől független szerkezetű) központi rész, és egy végtelen tár. A Turing-gépeken minden olyan számítás elvégezhető, melyet akár előtte, akár azóta bármilyen más matematikai gép-modellen el tudtak végezni. E gépfogalmat főleg elméleti vizsgálatokban használjuk. Konkrét algoritmusok megadására kevésbé alkalmas, mert leírása nehézkes, és főleg, mert a létező számítógépektől több fontos vonatkozásban eltér.

A Turing-gép nehézkes, a valódi számítógépekétől eltérő vonásai közül a leglényegesebb, hogy memóriáját nem lehet közvetlenül címezni, egy „távoli" memória-rekesz kiolvasásához minden korábbi rekeszt is el kell olvasni. Ezt hidalja át a *RAM* (Random Access Machine, Közvetlen Elérésű Gép = KEG) fogalma. Ez a gép a memória tetszőleges rekeszét egy lépésben el tudja érni. A RAM a valódi számítógépek egy leegyszerűsített modelljének tekinthető, azzal az absztrakcióval, hogy a memóriája korlátlan. A RAM-ot tetszőleges programnyelven programozhatjuk. Algoritmusok leírására a RAM-ot célszerű használni (már amikor az informális leírás nem elegendő), mert ez áll legközelebb a valódi programíráshoz. Látni fogjuk azonban, hogy a Turing-gép és a RAM igen sok szempontból egyenértékű; legfontosabb, hogy ugyanazok a függvények számíthatók ki Turing-gépen,

mint RAM-on.

Sajnos a RAM-nak ezért az előnyös tulajdonságáért fizetni kell: ahhoz, hogy közvetlenül el tudjunk érne egy memóriarekeszt, ezt meg kell címezni. Mivel nem korlátos a memóriarekeszek száma, a cím sem korlátos, és így a címet tartalmazó rekeszben akármilyen nagy természetes számot meg kell engednünk. Ezzel viszont ismét eltávolodunk a létező számítógépektől; ha nem vigyázunk, a RAM-on a nagy számokkal végzett műveletekkel visszaélve olyan algoritmusokat programozhatunk be, melyek létező számítógépeken csak sokkal nehezebben, lassabban valósíthatók meg.

Harmadikként foglalkozunk még egy számítási modellel, a *logikai hálózattal*. Ez a modell már nem ekvivalens a másik kettővel; egy adott logikai hálózat csak adott nagyságú bemenetet enged meg. Így egy logikai hálózat csak véges számú feladatot tud megoldani; az viszont nyilvánvaló lesz, hogy rögzített méretű bemenet esetén minden függvény kiszámítható logikai hálózattal. Azonban ha megszorítást teszünk pl. a kiszámítás idejére, akkor a logikai hálózatra és a Turing-gépre vagy RAM-ra vonatkozó problémák már nem különböznek ilyen lényegesen egymástól. Mivel a logikai hálózatok szerkezete, működése a legáttekinthetőbb, elméleti vizsgálatokban (főleg a bonyolultságra vonatkozó alsó korlátok bizonyításában) a logikai hálózatok igen fontos szerepet játszanak.

1.1. Véges automata

A *véges automata* egy nagyon egyszerű és általános számítási modell. Mindössze annyit teszünk fel, hogy ha kap egy bemenetet, akkor megváltoztatja a belső állapotát és kiad egy eredményt. Precízebben fogalmazva, egy véges automata rendelkezik

- egy *bemeneti ábécével*, amely egy Σ véges halmaz,
- egy *kimeneti ábécével*, amely egy másik Σ' véges halmaz, és
- a belső állapotok ugyancsak véges Γ halmazával.

Hogy teljesen leírjunk egy véges automatát, meg kell határoznunk minden $s \in \Gamma$ állapot és $a \in \Sigma$ bemeneti betű esetén a $\beta(s,a) \in \Sigma'$ kimenetet és az $\alpha(s,a) \in \Gamma$ új állapotot. Hogy az automata működése jól meghatározott legyen, az egyik állapotot kinevezzük START *kezdőállapotnak*.

A számítás kezdetén az automata az $s_0 = \text{START}$ állapotban van. A számítás bemenete egy $a_1 a_2 \dots a_n \in \Sigma^*$ szóval van megadva. A bemenet első a_1 betűje az automatát az $s_1 = \alpha(s_0, a_1)$ állapotba viszi, a következő a_2 betű az $s_2 = \alpha(s_1, a_2)$ állapotba, stb. A számítás eredménye a $b_1 b_2 \dots b_n$ szó, ahol $b_k = \beta(s_{k-1}, a_k)$ a k. lépés kimenete.

Így a véges automata leírható a $\langle \Sigma, \Sigma', \Gamma, \alpha, \beta, s_0 \rangle$ hatossal, ahol Σ, Σ', Γ véges halmazok, $\alpha : \Gamma \times \Sigma \to \Gamma$ és $\beta : \Gamma \times \Sigma \to \Sigma'$ tetszőleges leképezések, és $s_0 \in \Gamma$.

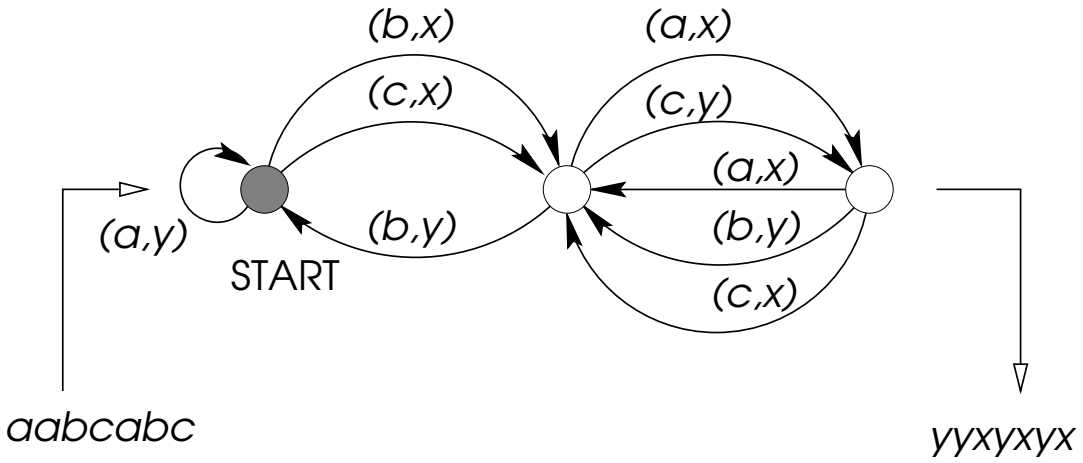
Megjegyzések. 1. Sok különböző változata van ennek a fogalomnak, melyek lényegében ekvivalensek. Gyakran nincs kimenet, és így kimeneti ábécé sem. Ebben az esetben az eredményt abból határozzuk meg, hogy az automata melyik állapotban van, mikor vége a számításnak. Tehát partícionáljuk az automata állapotainak halmazát két részre, ELGODADÓ és ELUTASÍTÓ állapotokra. Az automata elfogad egy szót, ha a számítás

végén ELFOGADÓ állapotban van. Egy $\mathcal{L}$ nyelvet *regulárisnak* nevezünk, ha van olyan véges automata, amely pont az $x \in \mathcal{L}$ bemeneteket fogadja el.

Abban az esetben, ha van kimenet, akkor gyakran kényelmes feltételezni, hogy Σ' tartalmazza a $*$ *üres szimbólumot*. Más szóval megengedjük az automatának, hogy bizonyos lépések során ne adjon kimenetet.

2. A kedvenc személyi számítógépünk is modellezhető véges automatával, ahol a bemeneti ábécé tartalmazza az összes lehetséges billentyűleütést és a kimeneti ábécé pedig az összes szöveget, amit egy billentyűleütést követően ki tud írni a képernyőre a számítógép (az egeret, lemezmeghajtót stb. figyelmen kívül hagyjuk). Figyeljük meg, hogy az állapotok halmaza csillagászati méretű (egy gigabájt tárhely esetén nagyságrendileg legalább $2^{10^{10}}$ állapot van). Ilyen nagy állapothalmazt megengedve szinte bármi modellezhető véges automatával. Minket viszont az olyan automaták érdekelnek, ahol az állapothalmaz sokkal kisebb, többnyire feltesszük hogy ez felülről korlátos, viszont a bemeneti szó hossza általában nem korlátozott.

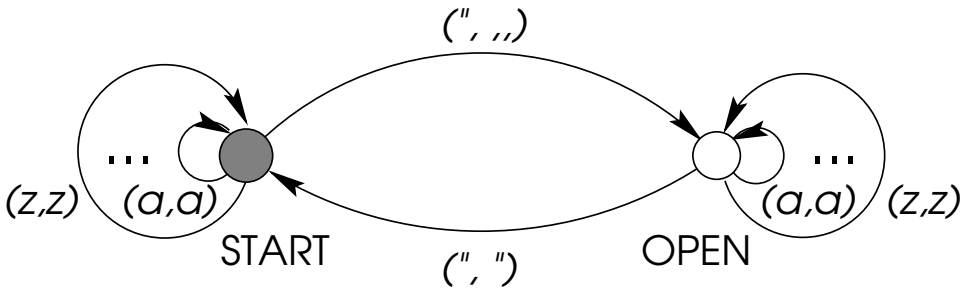
Minden véges automata leírható egy irányított gráffal. A gráf pontjai Γ elemei és amennyiben $\beta(s,a) = b$ és $\alpha(s,a) = s'$, akkor megy egy (a,b)-vel címkézett irányított él s-ből s'-be. Egy $a_1 a_2 \dots a_n$ bemenet esetén végzett számítás megfelel a gráfban egy olyan START-ban kezdődő útnak, melyben az élek címkéinek első tagjai sorra $a_1, a_2, \dots, a_n$. A második tagok adják az eredményt (lásd 1.1. ábra).



1.1. ábra. Véges automata

1. Példa. Konstruáljunk olyan automatát, mely kijavítja az idézőjeleket egy szövegben, azaz beolvas egy szöveget betűnként, és ha talál egy olyat, hogy "...", akkor azt kicseréli „..."-re. Az automatának mindössze annyit kell megjegyeznie, hogy eddig páros vagy páratlan darab idézőjel volt-e. Tehát két állapota lesz, egy START és egy NYITOTT (ez annak felel meg, hogy éppen egy idézeten belül vagyunk). A bemeneti ábécé tartalmaz minden karaktert, ami előfordulhat a szövegben, így az "-et is. A kimeneti ábécé ugyanez, csak " helyett most „ és " van. Az automata minden "-től eltérő karakter esetén ugyanazt adja kimenetnek, mint ami a bemenet volt és marad ugyanabban az állapotban. Amennyiben "-et olvas be, akkor „-t ad ki, ha éppen a START állapotban volt és "-t ha a

NYITOTT-ban, továbbá állapotot is vált (lásd 1.2. ábra).



1.2. ábra. Idézőjeleket kijavító automata

Feladatok.

1. Konstruáljunk egy korlátos számú állapottal rendelkező véges automatát, melynek megadva két kettes számrendszerben felírt számot, kiszámítja az összegüket. Az automata felváltva kapja meg a két szám egy-egy bitjét, mindkettőben jobbról kezdve. Ha valamelyik szám első bitjén túljutunk, akkor a bemenet egy speciális • szimbólum. A bemenet megáll, ha egymásután két • következik.

2. Konstruáljunk minél kevesebb állapotú véges automatát, mely egy szám tízes számrendszerbeli számjegyeit kapja balról kezdve és az utolsó kimenet legyen IGEN, ha a szám osztható 7-tel és NEM ellenkező esetben.

3. (a) Rögzített n pozitív egész esetén konstruáljunk egy véges automatát mely beolvas egy $2n$ hosszú szót és az utolsó kimenete IGEN, ha a szó első fele és második fele megegyezik és a kimenete NEM egyéb esetben.

(b) Bizonyítsuk be, hogy egy ilyen automatának legalább 2^n állapota kell, hogy legyen.

Az alábbi egyszerű lemma, ill. különböző változatai központi szerepet játszanak a bonyolultságelméletben. Ha $a, b, c \in \Sigma^*$ szavak, akkor ab^ic jelölje azt a szót, amikor egymás után írjuk az a szót, majd i-szer a b szót, végül a c szót.

1.1.1 Lemma [Pumpálási lemma] *Minden reguláris $\mathcal{L}$ nyelvhez létezik olyan pozitív egész k szám, hogy minden $x \in \mathcal{L}$ szó, melyre $|x| \geq k$, felírható $x = abc$ alakban, ahol $|ab| \leq k$, $|b| > 0$, hogy minden i természetes számra $ab^ic \in \mathcal{L}$.*

Feladatok.

4. Bizonyítsuk be a pumpálási lemmát.

5. Bizonyítsuk be, hogy az $\mathcal{L} = \{0^n1^n \mid n \in \mathbb{N}\}$ nyelv nem reguláris.

6. Bizonyítsuk be, hogy a palindrómák nyelve: $\mathcal{L} = \{x_1 \dots x_n x_n \dots x_1 : \ x_1 \dots x_n \in \Sigma^*\}$ nem reguláris.

1.2. A Turing-gép

Egy *Turing-gép* a következőkből áll:

- $k \geq 1$ két irányban végtelen szalagból. A szalagok mindkét irányban végtelen sok mezőre vannak osztva. Minden szalagnak van egy kitüntetett *kezdőmezeje*, melyet 0-ik mezőnek is hívunk. Minden szalag minden mezejére egy adott véges Σ ábécéből lehet jelet írni. Véges sok mező kivételével ez a jel az ábécé egy speciális „$*$" jele kell, hogy legyen, mely az „üres mezőt" jelöli.
- Minden szalaghoz tartozik egy *író-olvasófej*, mely minden lépésben a szalag egy mezején áll.
- Van még egy vezérlőegység. Ennek lehetséges állapotai egy véges Γ halmazt alkotnak. Ki van tüntetve egy „START" kezdőállapot és egy „STOP" végállapot.

Kezdetben a vezérlőegység START állapotban van, és a fejek a szalagok kezdőmezején állnak. Minden lépésben minden fej leolvassa a szalagjának adott mezején álló jelet; a vezérlőegység a leolvasott jelektől és a saját állapotától függően 3 dolgot csinál:

- átmegy egy új állapotba;
- minden fejnek utasítást ad, hogy azon a mezőn, melyen áll, a jelet írja fölül;
- minden fejnek utasítást ad, hogy lépjen jobbra vagy balra egyet, vagy maradjon helyben.

A gép megáll, ha a vezérlőegység a STOP állapotba jut.

Matematikailag a Turing-gépet az alábbi adatok írják le: $T = \langle k, \Sigma, \Gamma, \alpha, \beta, \gamma \rangle$, ahol $kr1$ egy természetes szám, Σ és Γ véges halmazok, $* \in \Sigma$, START, STOP $\in \Gamma$, és

$$\begin{aligned} \alpha: &\quad \Gamma \times \Sigma^k \to \Gamma, \\ \beta: &\quad \Gamma \times \Sigma^k \to \Sigma^k, \\ \gamma: &\quad \Gamma \times \Sigma^k \to \{-1, 0, 1\} \end{aligned}$$

tetszőleges leképezések. α adja meg az új állapotot, β a szalagokra írt jeleket, γ azt, hogy mennyit lép a fej. A Σ ábécét a továbbiakban rögzítjük, és feltesszük, hogy a „$*$" jelen kívül legalább két jelből áll, mondjuk tartalmazza 0-t és 1-et (a legtöbb esetben elegendő volna erre a két jelre szorítkozni).

Megjegyzés: A Turing-gépeknek nagyon sok különböző, de minden lényeges szempontból egyenértékű definíciója található a különböző könyvekben. Gyakran a szalagok csak egyirányban végtelenek; számuk szinte mindig könnyen korlátozható volna kettőre, és igen sok vonatkozásban egyre is; feltehetnénk, hogy „$*$" jelen kívül (amit ebben az esetben 0-val azonosítunk) csak az „1" jel van az ábécében; bizonyos szalagokról kiköthetnénk, hogy azokra csak írhat vagy csak olvashat róluk (de legalább egy szalagnak írásra és olvasásra is alkalmasnak kell lenni) stb. Ezeknek a megfogalmazásoknak az ekvivalenciája a velük végezhető számítások szempontjából több-kevesebb fáradsággal, de nagyobb nehézség nélkül igazolható. Mi csak annyit bizonyítunk ilyen irányban, amire szükségünk lesz.

Egy Turing-gép *bemenetén* az induláskor a szalagokra írt szavakat értjük. Mindig feltesszük, hogy ezek a 0-dik mezőktől kezdődően vannak a szalagokra írva. Egy k-szalagos

Turing-gép bemenete tehát egy rendezett k-as, melynek minden eleme egy Σ^*-beli szó. Leggyakrabban csak a gép első szalagjára írunk nem-üres szót bemenet gyanánt. Ha azt mondjuk, hogy a bemenet egy x szó, akkor azt értjük alatta, hogy az $(x, \emptyset, \ldots, \emptyset)$ k-as a bemenet.

Célszerű lesz föltenni, hogy a bemeneti szavak a „$*$" jelet nem tartalmazzák. Különben nem lehetne tudni, hogy hol van a bemenet vége: egy olyan egyszerű feladat, mint „határozzuk meg a bemenet hosszát", nem volna megoldható, a fej hiába lépkedne jobbra, nem tudná, hogy vége van-e már a bemenetnek. A $\Sigma - \{*\}$ ábécét Σ_0-lal jelöljük. Feltesszük azt is, hogy a Turing-gép az egész bemenetét elolvassa működése során; ezzel csat triviális eseteket zárunk ki.

A gép *kimenete* a megálláskor a szalagokon levő szavakból álló rendezett k-as. Gyakran azonban egyetlen szóra vagyunk kíváncsiak, a többi „szemét". Ha egyetlen szóra, mint kimenetre hivatkozunk, akkor az utolsó szalagon levő szót értjük ez alatt.

Feladatok:

7. Konstruáljunk olyan Turing-gépet, mely a következő függvényeket számolja ki:

a) $x_1 \ldots x_m \mapsto x_m \ldots x_1$.

b) $x_1 \ldots x_m \mapsto x_1 \ldots x_m x_1 \ldots x_m$.

c) $x_1 \ldots x_m \mapsto x_1 x_1 \ldots x_m x_m$.

d) m hosszú csupa 1-esből álló bemenetre az m szám 2-es számrendszerbeli alakját; egyéb bemenetre azt, hogy „MICIMACKÓ".

e) ha a bemenet az m szám 2-es számrendszerbeli alakja, a kimenet legyen m darab 1-es (különben „MICIMACKÓ").

f) A d) és e) feladatokat úgy is oldjuk meg, hogy ha kimenet nem „MICIMACKÓ", akkor a gép csak $O(m)$ lépést tegyen.

8. Tegyük föl, hogy van két Turing-gépünk, melyek az $f : \Sigma_0^* \mapsto \Sigma_0^*$ ill. a $g : \Sigma_0^* \mapsto \Sigma_0^*$ függvényt számolják ki. Konstruáljunk olyan Turing-gépet, mely az $f \circ g$ függvényt számolja ki.

9. Konstruáljunk olyan Turing-gépet, mely egy x bemenetre pontosan $2^{|x|}$ lépést tesz.

10. Konstruáljunk olyan Turing-gépet, mely egy x bemenetre akkor és csak akkor áll meg véges számú lépés után, ha x-ben előfordul a 0 jel.

Az eddigiek alapján egy lényeges különbséget vehetünk észre a Turing-gépek és a valódi számítógépek között: Minden függvény kiszámításához külön-külön Turing-gépet konstruáltunk, míg a valódi programvezérlésű számítógépeken elegendő megfelelő programot írni. Megmutatjuk most, hogy lehet a Turing-gépet is így kezelni: lehet olyan Turing-gépet konstruálni, melyen alkalmas „programmal" minden kiszámítható, ami bármely Turing-gépen kiszámítható. Az ilyen Turing-gépek nem csak azért érdekesek, mert jobban hasonlítanak a programvezérlésű számítógépekhez, hanem fontos szerepet fognak játszani sok bizonyításban is.

Legyen $T = \langle k+1, \Sigma, \Gamma_T, \alpha_T, \beta_T, \gamma_T \rangle$ és $S = \langle k, \Sigma, \Gamma_S, \alpha_S, \beta_S, \gamma_S \rangle$ két Turing-gép ($k \geq 1$). Legyen $p \in \Sigma_0^*$. Azt mondjuk, hogy T a p programmal *szimulálja* S-et, ha tetszőleges $x_1, \ldots, x_k \in \Sigma_0^*$ szavakra T az $(x_1, \ldots, x_k, p)$ bemeneten akkor és csak akkor áll meg véges számú lépésben, ha S az $(x_1, \ldots, x_k)$ bemeneten megáll, és megálláskor T első

k szalagján rendre ugyanaz áll, mint S szalagjain.

Akkor mondjuk, hogy a $k+1$ szalagos T Turing-gép *univerzális* (a k szalagos Turing-gépekre nézve), ha bármely k szalagos Σ fölötti S Turing-géphez létezik olyan p szó (program), mellyel a T szimulálja S-et.

1.1.1 Tétel. *Minden $k \geq 1$ számhoz és minden Σ ábécéhez létezik $k+1$ szalagú univerzális Turing-gép.*

Bizonyítás: Az univerzális Turing-gép konstrukciójának alapgondolata az, hogy a $(k+1)$-edik szalagra a szimulálandó S Turing-gép működését leíró táblázatot írunk. Az univerzális T Turing-gép ezenkívül még fölírja magának, hogy a szimulált S gépnek melyik állapotában van éppen (hiába van csak véges sok állapot, a rögzített T gépnek minden S gépet szimulálnia kell, így az S állapotait „nem tudja fejben tartani"). Minden lépésben ennek, és a többi szalagon olvasott jelnek az alapján a táblázatból kikeresi, hogy S milyen állapotba megy át, mit ír a szalagokra, és merre mozdulnak a fejek.

A pontos konstrukciót először úgy adjuk meg, hogy $k+2$ szalagot használunk. Egyszerűség kedvéért tegyük föl, hogy Σ tartalmazza a „0", „1" és „-1" jeleket.

Legyen $S = \langle k, \Sigma, \Gamma_S, \alpha_S, \beta_S, \gamma_S \rangle$ tetszőleges k szalagos Turing-gép. $\Gamma_S - \{\text{STOP}\}$ minden elemét azonosítsuk egy-egy r hosszúságú Σ_0^*-beli szóval. Az S gép egy adott helyzetének „kódja" az alábbi szó legyen:

$$gh_1 \dots h_k \alpha_S(g, h_1, \dots, h_k) \beta_S(g, h_1, \dots, h_k) \gamma_S(g, h_1, \dots, h_k),$$

ahol $g \in \Gamma_S$ a verérlőautomata adott állapota, és $h_1, \dots, h_k \in \Sigma$ az egyes fejek által olvasott jelek. Az ilyen szavakat tetszőleges sorrendben összefűzzük; így kapjuk a p_S szót. Ezt fogjuk majd a $(k+1)$-edik szalagra írni; a $(k+2)$-edikre pedig az S gép egy állapotát, kiinduláskor az S gép START állapotának a nevét.

Ezekután a T' Turing-gépet konstruáljuk meg. Ez az S gép egy lépését úgy szimulálja, hogy a $(k+1)$-edik szalagon kikeresi, hogy hol van a $(k+2)$-edik szalagon följegyzett állapotnak és az első k fej által olvasott jeleknek megfelelő feljegyzés, majd onnan leolvassa a teendőket: fölírja a $(k+2)$-edik szalagra az új állapotot, az első k fejjel pedig a megfelelő jeleket íratja és a megfelelő irányban lép.

Teljesség kedvéért formálisan is leírjuk a T' gépet, de az egyszerűségnek is teszünk annyi engedményt, hogy csak a $k = 1$ esetben. A gépnek tehát három feje van. A kötelező „START" és „STOP" állapotokon kívül legyenek még ITTVAN-VISSZA, NEMITTVAN-VISSZA, NEMITTVAN-TOVÁBB, KÖVETKEZŐ, ÁLLAPOT-FÖLÍRÁS, MOZGATÁS, és ÚJRA állapotai. Jelölje $h(i)$ az i-edik fej által olvasott betűt ($1 \leq i \leq 3$). Az α, β, γ függvényeket az alábbi táblázattal írjuk le (ha nem mondunk külön új állapotot, akkor a vezérlőegység marad a régiben).

START:

ha $h(2) = h(3) \neq *$, akkor 2 és 3 jobbra lép;
ha $h(3) \neq *$ és $h(2) = *$, akkor STOP;
ha $h(3) = *$ és $h(2) = h(1)$, akkor „ITTVAN-VISSZA", 2 jobbra lép és 3 balra lép;
ha $h(3) = *$ és $h(2) \neq h(1)$, akkor „NEMITTVAN-VISSZA", 2 jobbra, 3 balra lép;
egyébként „NEMITTVAN-TOVÁBB", és 2, 3 jobbra lép.

ITTVAN-VISSZA:

ha $h(3) \neq *$, akkor 3 balra lép;
ha $h(3) = *$, akkor „ÁLLAPOT-FÖLÍRÁS”, és 3 jobbra lép.

NEMITTVAN-TOVÁBB:
ha $h(3) \neq *$, akkor 2 és 3 jobbra lép;
ha $h(3) = *$, akkor „NEMITTVAN-VISSZA”, 2 jobbra lép és 3 balra lép.

NEMITTVAN-VISSZA:
ha $h(3) \neq *$, akkor 2 jobbra lép, 3 balra lép;
ha $h(3) = *$, akkor „KÖVETKEZŐ”, 2 és 3 jobbra lép.

KÖVETKEZŐ:
„START”, és 2 jobbra lép.

ÁLLAPOT-FÖLÍRÁS:
ha $h(3) \neq *$, akkor 3 a $h(2)$ jelet írja, és 2, 3 jobbra lép;
ha $h(3) = *$, akkor „MOZGATÁS”, az 1 fej $h(2)$-t ír, 2 jobbra lép.

MOZGATÁS:
„ÚJRA”, az 1 fej $h(2)$-t lép.

ÚJRA:
ha $h(2) \neq *$ és $h(3) \neq *$, akkor 2 és 3 balra lép;
ha $h(2) \neq *$, de $h(3) = *$, akkor 2 balra lép;
ha $h(2) = h(3) = *$, akkor „START”, és 2, 3 jobbra lép.

A $(k+2)$-edik szalagtól könnyen megszabadulhatunk: tartalmát (ami mindig csak r mező), a $(k+1)$-edik szalag (-2)-edik, (-3)-adik, ..., $(-r-1)$-edik mezején helyezzük el (a -1. mezőn hagyunk egy $*$-ot határolójelnek). Problémát okoz azonban, hogy még mindig két fejre van szükségünk ezen a szalagon: egyik a szalag pozitív felén, a másik a negatív felén mozog. Ezt úgy oldjuk meg, hogy minden mezőt megduplázunk; a bal felébe marad írva az eredetileg is oda írt jel, a jobb felén pedig 1-es áll, ha ott állna a fej, ha két fej volna (a többi jobb oldali félmező üresen marad. Könnyű leírni, hogy hogyan mozog az egyetlen fej ezen a szalagon úgy, hogy szimulálni tudja mindkét eredeti fej mozgását. □

Feladatok.
11. Mutassuk meg, hogy ha a fent konstruált univerzális $(k+1)$-szalagos Turing-gépen szimuláljuk a k-szalagosat, akkor tetszőleges bemeneten a lépésszám csak a szimuláló program hosszával arányos (tehát konstans) szorzótényezővel növekszik meg.
12. Legyenek T és S egyszalagos Turing-gépek. Azt mondjuk, hogy T az S működését a p programmal szimulálja ($p \in \Sigma_0^*$), ha minden $x \in \Sigma_0^*$ szóra a T gép a $p*x$ bemeneten akkor és csak akkor áll meg véges számú lépésben, ha S az x bemeneten megáll, és megálláskor T szalagján ugyanaz áll, mint S szalagján. Bizonyítsuk be, hogy van olyan egyszalagos T Turing-gép, mely minden más egyszalagos Turing-gép működését ebben az értelemben szimulálni tudja.

Következő tételünk azt mutatja, hogy nem lényeges, hogy hány szalagja van egy Turing-gépnek.

1.1.2 Tétel. *Minden k szalagos S Turing-géphez van olyan egy szalagos T Turing-gép, amely S-et helyettesíti a következő értelemben: minden $*$-ot nem tartalmazó x szóra, S akkor és csak akkor áll meg véges sok lépésben az x bemeneten, ha T megáll, és megálláskor S utolsó szalagjára ugyanaz lesz írva, mint T szalagjára. Továbbá, ha S N lépést tesz, akkor T $O(N^2)$ lépést tesz.*

Bizonyítás: Az S gép szalagjainak tartalmát a T gép egyetlen szalagján kell tárolnunk. Ehhez először is a T szalagjára írt bemenetet „széthúzzuk": az i-edik mezőn álló jelet átmásoljuk a $(2ki)$-edik mezőre. Ezt úgy lehet megcsinálni, hogy először 1-től indulva jobbra lépegetve minden jelet $2k$ hellyel odébb másolunk (ehhez egyszerre csak 2k, vagyis rögzített számú jelre kell a vezérlőegységnek emlékeznie). Közben az $1, 2, \ldots, 2k-1$ helyekre $*$-ot írunk. Majd a fej visszajön az első $*$-ig (a $(2k-1)$-edik helyig), és a $(2k+1)$-edik helyen álló jeltől kezdve minden jelet újabb $2k$ hellyel jobbra visz stb.

Ezekután $(2ki + 2j - 2)$-edik mező $(1 \leq j \leq k)$ fog megfelelni a j-edik szalag i-edik mezejének, a $(2ki+2j-1)$-edik mezőn pedig 1 vagy $*$ fog állni aszerint, hogy az S megfelelő feje az S számításának megfelelő lépésénél azon a mezőn áll-e vagy sem. A szalagunk két végét jelöljük meg egy-egy 0-val, az első olyan páratlan sorszámú mezőben, amelyben még soha nem volt 1-es. Így az S számításának minden helyzetének megfeleltettünk T-nek egy helyzetét.

Megmutatjuk most, hogy S lépéseit T hogyan tudja utánozni. Mindenekelőtt, T „fejben tartja" azt, hogy az S gép melyik állapotban van. Azt is mindig tudja, hogy modulo $2k$ milyen sorszámú az a mező, amin a saját feje éppen tartózkodik, valamint hogy az S fejeit jelző 1-esek közül hány darab van a saját fejétől jobbra. A legjobboldalibb S-fejtől indulva, haladjon most végig a fej a szalagon. Ha egy páratlan sorszámú mezőn 1-est talál, akkor a következő mezőt leolvassa, és megjegyzi hozzá, hogy modulo $2k$ mi volt a mező sorszáma. Mire végigér, tudja, hogy milyen jeleket olvasnak ennél a lépésnél az S gép fejei. Innen ki tudja számítani, hogy mi lesz S új állapota, mit írnak és merre lépnek a fejei. Visszafelé indulva, minden páratlan mezőn álló 1-esnél át tudja írni megfelelően az előtte levő mezőt, és el tudja mozdítani az 1-est szükség esetén $2k$ hellyel balra vagy jobbra. (Persze jobbra mozgatáskor átlépheti S néhány fejét, így ezekhez vissza kell mennie. Ha közben túlszaladna a kezdő vagy záró 0-n, akkor azt is mozgassuk kijjebb.)

Ha az S gép számításának szimulációja befejeződött, akkor az eredményt „tömöríteni" kell: a $2ki$ mező tartalmát át kell másolni az i mezőre. Ez a kezdő „széthúzáshoz" hasonlóan tehető meg, itt használjuk a szalvégeket jelző 0-kat.

Nyilvánvaló, hogy az így leírt T gép ugyanazt fogja kiszámítani, mint S. A lépésszám három részből tevődik össze: a „széthúzás", a szimulálás, és a tömörítés idejéből. Legyen M a T gépen azon mezők száma, melyekre a gép valaha is lép; nyilvánvaló, hogy $M = O(N)$. A „széthúzás"-hoz és a „tömörítés"-hez $O(M^2)$ idő kell. Az S gép egy lépésének szimulálásához $O(M)$ lépés kell, így a szimulációhoz $O(MN)$ lépés. Ez összesen is csak $O(N^2)$ lépés. □

Feladatok.

13*. Mutassuk meg, hogy ha az 1.1.2 Tételben 1-szalagos helyett kétszalagos Turing-gépet engedünk meg, akkor a lépésszám kevesebbet növekszik: minden k szalagos Turing-gépet olymódon helyettesíthetünk kétszalagossal, hogy ha egy bemeneten a k szalagos lépésszáma N, akkor a kétszalagosé legfeljebb $O(N \log N)$.

14. Álljon az $\mathcal{L}$ nyelv a „palindrómákból”:

$$\mathcal{L} = \{x_1 \dots x_n x_n \dots x_1 : \ x_1, \dots, x_n \in \Sigma_0^*\}.$$

a) Van olyan 2-szalagos Turing-gép, mely egy n hosszú szóról $O(n)$ lépésben eldönti, hogy $\mathcal{L}$-ben van-e.

b) Bármely 1 szalagos Turing-gépnek $\Omega(n^2)$ lépésre van szüksége ahhoz, hogy ugyanezt eldöntse. [Erre a feladatra a 10. fejezetben visszatérünk.]

1.3. A RAM

A RAM (Random Access Machine = Közvetlen Elérésű Gép = KEG)) a Turing-gépnél bonyolultabb, de a valódi számítógépekhez közelebb álló matematikai modell. Mint a neve is mutatja, a legfőbb pont, ahol a Turing-gépnél többet „tud”: memóriarekeszeit közvetlenül lehet elérni (bele írni vagy belőle kiolvasni).

Sajnos a RAM-nak ezért az előnyös tulajdonságáért fizetni kell: ahhoz, hogy közvetlenül el tudjunk érne egy memóriarekeszt, ezt meg kell címezni; a címet valamelyik másik rekeszben tároljuk. Mivel nem korlátos a memóriarekeszek száma, a cím sem korlátos, és így a címet tartalmazó rekeszben akármilyen nagy természetes számot meg kell engednünk. Ennek a rekesznek a tartalma maga is változhat a program futása során (indirekt címzés). Ez a RAM erejét tovább növeli; viszont azzal, hogy az egy rekeszben tárolható számot nem korlátozzuk, ismét eltávolodunk a létező számítógépektől. Ha nem vigyázunk, a RAM-on a nagy számokkal végzett műveletekkel „visszaélve” olyan algoritmusokat programozhatunk be, melyek létező számítógépeken csak sokkal nehezebben, lassabban valósíthatók meg.

A RAM-ban van egy programtár és egy memória. A memória végtelen sok memóriarekeszből áll, melyek az egész számokkal vannak címezve. Minden i memóriarekesz egy $x[i]$ egész számot tartalmaz (ezek közül mindig csak véges sok nem 0). A programtár ugyancsak végtelen sok, $0, 1, 2, \dots$-vel címzett rekeszből, *sorból* áll; ebbe olyan (véges hosszúságú) programot írhatunk, mely valamely gépi kód-szerű programnyelven van írva. Például elegendő a következő utasításokat megengedni:

$x[i] := 0;$ $x[i] := x[i] + 1;$ $x[i] := x[i] - 1;$
$x[i] := x[i] + x[j];$ $x[i] := x[i] - x[j];$ $x[x[i]] := x[j];$ $x[i] := x[x[j]];$
IF $x[i] \leq 0$ THEN GOTO p.

Itt i és j valamely memóriarekesz sorszáma (tehát tetszőleges egész szám), p pedig valamelyik programsor sorszáma (tehát tetszőleges természetes szám). Az utolsó előtti két utasítás biztosítja közvetett címzés lehetőségét. Elméleti szempontból nincs azonban különösebb jelentősége, hogy mik ezek az utasítások; csak annyi lényeges, hogy mindegyikük könnyen megvalósítható műveletet fejezzen ki, és eléggé kifejezőek legyenek ahhoz, hogy a kívánt számításokat el tudjuk végezni; másrészt véges legyen a számuk. Például i és j értékére elég volna csak a $-1, -2, -3$ értékeket megengedni. Másrészről bevehetnénk a szorzást, stb.

A RAM bemenete egy természetes számokból álló véges sorozat, melyek hosszát az $x[0]$ memóriarekeszbe, elemeit pedig rendre az $x[1], x[2], \dots$ memóriarekeszekbe írjuk be. Ha az inputban nem engedjük meg a 0 értéket, akkor a hosszra az $x[0]$ rekeszben nincs szükségünk.

A RAM a fenti utasításokból álló tetszőleges véges programot értelemszerűen végrehajt; akkor áll meg, ha olyan programsorhoz ér, melyben nincsen utasítás. A *kimeneten* az $x[i]$ rekeszek tartalmát értjük.

A RAM lépésszáma nem a legjobb mértéke annak, hogy „mennyi ideig dolgozik". Amiatt ugyanis, hogy egy lépésben akármilyen nagy természetes számokon végezhetünk műveleteket, olyan trükköket lehet csinálni, amik a gyakorlati számításoktól igen messze vannak. Pl. két igen hosszú természetes szám összeadásával vektorműveleteket szimulálhatnánk. Ezért szokásosabb a RAM-ok lépésszáma helyett a *futási idejükről* beszélni. Ezt úgy definiáljuk, hogy egy lépés idejét nem egységnyinek vesszük, hanem annyinak, mint a benne fellépő természetes számok (rekeszcímek és tartalmak) kettes számrendszerbeli jegyeinek száma. (Mivel ez lényegében a kettes alapú logaritmusuk, szokás ezt a modellt *logaritmikus költségű RAM-nak* is nevezni.) Meg kell jegyeznünk, hogy ha a RAM utasításkészletét kibővítjük, pl. a szorzással, akkor ezen fajta lépések idejét úgy kell definiálni, hogy hosszú számokra a lépés pl. Turing-gépen ennyi idő alatt tényleg szimulálható legyen.

Szokás két paraméterrel is jellemezni a futási időt, hogy „a gép legfeljebb n lépést végez (kettes számrendszerben) legfeljebb k jegyű számokon"; ez tehát $O(nk)$ futási időt ad.

Feladat. 15. Írjunk olyan programot a RAM-ra, mely adott a pozitív egész számra

a) meghatározza azt a legnagyobb m számot, melyre $2^m \leq a$;

b) kiszámítja a kettes számrendszerbeli alakját (az a szám i. bitjét írja az $x[i]$ rekeszbe);

c) adott a és b pozitív egész számra kiszámítja a szorzatukat.

Ha a és b számjegyeinek száma k, akkor a program $O(k)$ lépést tegyen $O(k)$ jegyű számokkal.

Most megmutatjuk, hogy a RAM és a Turing-gép lényegében ugyanazt tudják kiszámítani, és a futási idejük sem különbözik túlságosan. Tekintsünk egy (egyszerűség kedvéért) 1 szalagos T Turing-gépet, melynek ábécéje $\{0, 1, 2\}$, ahol (a korábbiaktól eltérően, de itt célszerűbben) a 0 legyen az üres-mező jel.

A Turing-gép minden $x_1 \dots x_n$ bemenete (mely egy 1-2 sorozat) kétféleképpen is tekinthető a RAM egy bemenetének: beírhatjuk az $x_1, \dots, x_n$ számokat rendre az $x[1], \dots, x[n]$ rekeszekbe, vagy megfeleltethetünk az $x_1 \dots x_n$ sorozatnak egyetlen természetes számot pl. úgy, hogy a ketteseket 0-ra cseréljük és az elejére egy egyest írunk; és ezt a számot írjuk be az $x[0]$ rekeszbe. A Turing-gép kimenetét is hasonlóképpen értelmezhetjük, mint a RAM kimenetét.

Csak az első értelmezéssel foglalkozunk, a második szerinti input a 15. b) feladat alapján átalakítható az első értelmezés szerintivé.

1.2.2 Tétel. *Minden $\{0, 1, 2\}$ fölötti Turing-géphez konstruálható olyan program a RAM-on, mely minden bemenetre ugyanazt a kimenetet számítja ki, mint a Turing-gép, és ha a Turing-gép lépésszáma N, akkor a* RAM *$O(N)$ lépést végez $O(\log N)$ jegyű számokkal.*

Bizonyítás: Legyen $T = \{1, \{0, 1, 2\}, \Gamma, \alpha, \beta, \gamma\}$. Legyen $\Gamma = \{1, \dots, r\}$, ahol $1 =$ START és $r =$ STOP. A Turing-gép számolásának utánzása során a RAM $2i$-edik rekeszében ugyanaz a szám (0, 1, vagy 2) fog állni, mint a Turing-gép szalagjának i-edik mezején. Az $x[1]$ rekeszben tároljuk, hogy hol van a fej a szalagon, a vezérlőegység állapotát pedig az fogja meghatározni, hogy hol vagyunk a programban.

Programunk P_i $(1 \leq i \leq r)$ és Q_{ij} $(1 \leq i \leq r-1,\ 0 \leq j \leq 2)$ részekből fog összetevődni. Az alábbi P_i programrész $(1 \leq i \leq r-1)$ azt utánozza, amikor a Turing-gép vezérlőegysége

i állapotban van, és a gép kiolvassa, hogy a szalag $x[i]/2$-edik mezején milyen szám áll. Ettől függően fog más-más részre ugrani a programban:

$x[3] := x[x[1]]$;
IF $x[3] \leq 0$ THEN GOTO [Q_{i0} címe];
$x[3] := x[3] - 1$;
IF $x[3] \leq 0$ THEN GOTO [Q_{i1} címe];
$x[3] := x[3] - 1$;
IF $x[3] \leq 0$ THEN GOTO [Q_{i2} címe];

A P_r programrész álljon egyetlen üres programsorból. Az alábbi Q_{ij} programrész átírja az $x[1]$-edik rekeszt a Turing-gép szabályának megfelelően, módosítja $x[1]$-et a fej mozgásának megfelelően, és az új a állapotnak megfelelő P_a programrészre ugrik:

$x[3] := 0$;
$\left.\begin{array}{l} x[3] := x[3] + 1; \\ \vdots \\ x[3] := x[3] + 1; \end{array}\right\}$ $\beta(i,j)$-szer
$x[x[1]] := x[3]$;
$x[1] := x[1] + \gamma(i,j)$;
$x[1] := x[1] + \gamma(i,j)$;
$x[3] := 0$;
IF $x[3] \leq 0$ THEN GOTO [$P_{\alpha(i,j)}$ címe];

(Itt az $x[1] := x[1] + \gamma(i,j)$ utasítás úgy értendő, hogy az $x[1] := x[1] + 1$ ill. $x[1] := x[1] - 1$ utasítást vesszük, ha $\gamma(i,j) = 1$ vagy -1, és elhagyjuk, ha $\gamma(i,j) = 0$.) Maga a program így néz ki:

$x[1] := 0$;
P_1
P_2
$\vdots$
P_r
Q_{00}
$\vdots$
$Q_{r-1,2}$

Ezzel a Turing-gép „utánzását” leírtuk. A futási idő megbecsléséhez elég azt megjegyezni, hogy N lépésben a Turing-gép legfeljebb egy $-N$ és $+N$ közötti sorszámú mezőbe ír bármit is, így a RAM egyes lépéseiben legfeljebb $O(\log N)$ hosszúságú számokkal dolgozunk. □

1.3.1. Megjegyzés. *Az 1.2.2 Tétel bizonyításában nem használtuk fel az $y := y + z$ utasítást; erre az utasításra csak a 15. feladat megoldásában van szükség. Sőt ez a feladat is megoldható volna, ha ejtenénk a lépésszámra vonatkozó kikötést. Azonban ha a RAM-on tetszőleges egész számokat megengedünk bemenetként, akkor enélkül az utasítás nélkül exponenciális futási időt, sőt lépésszámot kapnánk igen egyszerű problémákra is. Például tekintsük azt a feladatot, hogy az $x[1]$ regiszter a tartalmát hozzá kell adni az $x[0]$ regiszter*

b tartalmához. Ez a RAM-on könnyen elvégezhető néhány lépésben; ennek futási ideje még logaritmikus költségek esetén is csak kb. $\log_2 |a| + \log_2 |b|$. *De ha kizárjuk az* $y := y+z$ *utasítást, akkor legalább* $min\{|a|, |b|\}$ *idő kell hozzá (ugyanis minden más utasítás a maximális tárolt szám abszolút értékét legfeljebb 1-gyel növeli).*

Legyen most adott egy RAM program. Ennek bemenetét és kimenetét egy-egy $\{0, 1, -, \#\}^*$-beli szónak tekinthetjük (a szereplő egész számokat kettes számrendszerben, ha kell előjellel fölírva, a közöket # jellel jelölve). Ebben az értelemben igaz az alábbi tétel:

1.2.3 Tétel. *Minden a* RAM *programhoz van olyan Turing-gép, mely minden bemenetre ugyanazt a kimenetet számítja ki, mint a* RAM*, és ha a* RAM *futási ideje* N*, akkor a Turing-gép lépésszáma* $O(N^2)$.

Bizonyítás: A RAM számolását négyszalagos Turing-géppel fogjuk szimulálni. Turing-gépünk első szalagjára írjuk az $x[i]$ memóriarekeszek tartalmát (kettes számrendszerben, ha negatív, előjellel ellátva). Megtehetnénk, hogy minden rekesz tartalmát sorra föltüntetjük (a 0 tartalom mondjuk a „$*$” jelnek felelne meg). Problémát jelent azonban, hogy a RAM akár a 2^{N-1} sorszámú rekeszbe is írhat csak N időt véve igénybe, a logaritmikus költség szerint. Természetesen ekkor a kisebb indexű rekeszek túlnyomó többségének a tartalma 0 marad az egész számítás folyamán; ezeknek a tartalmát nem célszerű a Turing-gép szalagján tárolni, mert akkor a szalag nagyon hosszú részét használjuk, és exponenciális időt vesz igénybe csak amíg a fej ellépeget oda, ahová írnia kell. Ezért csak azoknak a rekeszeknek a tartalmát tároljuk a Turing-gép szalagján, melyekbe ténylegesen ír a RAM. Persze ekkor azt is fel kell tüntetni, hogy mi a szóbanforgó rekesz sorszáma.

Azt tesszük tehát, hogy valahányszor a RAM egy $x[z]$ rekeszbe egy y számot ír, a Turing-gép ezt úgy szimulálja, hogy az első szalagja végére a $\#\#y\#z$ jelsorozatot írja. (Átírni soha nem ír át ezen a szalagon!) Ha a RAM egy $x[z]$ rekesz tartalmát olvassa ki, akkor a Turing-gép első szalagján a fej hátulról indulva megkeresi az első $\#\#u\#z$ alakú sorozatot; ez az u érték adja meg, hogy mi volt utoljára a z-edik rekeszbe írva. Ha ilyen sorozatot nem talál, akkor $x[z]$-t 0-nak tekinti. Könnyű az elején RAM bemenetét is ilyen formátumúra átírni.

A RAM „programnyelvének” minden egyes utasítását könnyű szimulálni egy-egy alkalmas Turing-géppel, mely csak a másik három szalagot használja.

Turing-gépünk olyan „szupergép” lesz, melyben minden programsornak megfelel állapotok egy halmaza. Ezek az állapotok egy olyan Turing-gépet alkotnak, mely az illető utasítást végrehajtja, és a végén a fejeket az első szalag végére (utolsó nem-üres mezejére), a többi szalagnak pedig a 0-dik mezejére mezejére viszi vissza. Minden ilyen Turing-gép STOP állapota azonosítva van a következő sornak megfelelő Turing-gép START állapotával. (A feltételes ugrás esetén, ha $x[i] \leq 0$ teljesül, a p sornak megfelelő Turing-gép kezdőállapotába megy át a „szupergép”.) A 0-dik programsornak megfelelő Turing-gép START-ja lesz a szupergép START-ja is. Ezenkívül lesz még egy STOP állapot; ez felel meg minden üres programsornak.

Könnyű belátni, hogy az így megkonstruált Turing-gép lépésről lépésre szimulálja a RAM működését. A legtöbb programsort a Turing-gép a benne szereplő számok számjegyeinek számával, vagyis éppen a RAM-on erre fordított idejével arányos lépésszámban hajtja végre. Kivétel a kiolvasás, melyhez esetleg (egy lépésnél legfeljebb kétszer) végig kell keresni az egész szalagot. Mivel a szalag hossza legfeljebb $\frac{5}{2}N$, a teljes lépésszám $O(N^2)$. □

1.4. Boole-függvények és logikai hálózatok

Boole-függvénynek nevezünk egy $f : \{0,1\}^n \to \{0,1\}$ leképezést. Szokás az 1 értéket az „IGAZ", a 0 értéket a „HAMIS" logikai értékkel azonosítani, a függvény változóit, melyek ezeket az értékeket vehetik fel, *logikai változóknak* (vagy *Boole-változóknak*) nevezni. Igen sok algoritmikus feladat bemenete n logikai változó, kimenete pedig egyetlen bit. Például: adott egy N pontú G gráf, döntsük el, hogy van-e benne Hamilton kör. Ekkor a gráfot $\binom{N}{2}$ logikai változóval írhatjuk le: a pontokat 1-től N-ig megszámozzuk, és x_{ij} $(1 \le i < j \le N)$ legyen 1, ha i és j össze vannak kötve, és 0, ha nem. Az $f(x_{12}, x_{13}, \ldots, x_{n-1,n})$ függvény értéke legyen 1, ha G-ben van Hamilton-kör, és 0, ha nincs. Problémánk ekkor ezen (implicite megadott) Boole-függvény értékének a kiszámítása.

Egyváltozós Boole-függvény csak 4 van: az azonosan 0, az azonosan 1, az identitás és a *tagadás* vagy *negáció*: $x \mapsto \overline{x} = 1 - x$. A kétváltozós Boole-függvények közül itt csak hármat említünk: a *konjunkciót* vagy logikai „ÉS" műveletét:

$$x \wedge y = \begin{cases} 1, & \text{ha } x = y = 1, \\ 0, & \text{egyébként,} \end{cases}$$

(ez tekinthető volna közönséges vagy modulo 2 szorzásnak is), a *diszjunkciót* vagy logikai „VAGY" műveletét:

$$x \vee y = \begin{cases} 0, & \text{ha } x = y = 0, \\ 1, & \text{egyébként,} \end{cases}$$

és a *bináris összeadást*:

$$x \oplus y \equiv x + y \pmod 2.$$

E műveleteket számos azonosság kapcsolja össze. Mindhárom említett kétváltozós művelet *asszociatív* és *kommutatív*. Fontos még a disztributivitás, melynek ebben a struktúrában több változata is van:

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z),$$

$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z),$$

és

$$x \wedge (y \oplus z) = (x \wedge y) \oplus (x \wedge z).$$

Végül idézzük még fel a *de Morgan azonosságokat:*

$$\overline{x \wedge y} = \overline{x} \vee \overline{y},$$

és

$$\overline{x \vee y} = \overline{x} \wedge \overline{y}.$$

A konjunkció, diszjunkció és negáció műveleteivel fölírt kifejezéseket *Boole-polinomoknak* nevezzük.

1.3.1 Lemma. *Minden Boole-függvény kifejezhető Boole-polinommal.*

Bizonyítás: Legyen $(a_1, \ldots, a_n) \in \{0,1\}^n$. Legyen

$$z_i = \begin{cases} x_i, & \text{ha } a_i = 1, \\ \overline{x}_i, & \text{ha } a_i = 0, \end{cases}$$

és $E_{a_1 \ldots a_n}(x_1, \ldots, x_n) = z_1 \wedge \ldots \wedge z_n$. Vegyük észre, hogy $E_{a_1 \ldots a_n}(x_1, \ldots, x_n) = 1$ akkor és csak akkor áll, ha $(x_1, \ldots, x_n) = (a_1, \ldots, a_n)$. Ezért

$$f(x_1, \ldots, x_n) = \bigvee_{f(a_1, \ldots, a_n)=1} E_{a_1 \ldots a_n}(x_1, \ldots, x_n).$$

A most megkonstruált Boole-polinom speciális alakú. Az egyetlen (negált vagy nem negált) változóból álló Boole-polinomot *literálnak* nevezzük. *Elemi konjunkciónak* nevezünk egy olyan Boole-polinomot, mely $\wedge$ művelettel összekapcsolt literálokból áll. (Elfajuló esetként minden literált és az 1 konstanst is elemi konjunkciónak tekintjük.) *Diszjunktív normálformának* nevezzük az olyan Boole-polinomot, mely $\vee$ művelettel összekapcsolt elemi konjunkciókból áll (ezeket a normálforma *tényezőinek* nevezzük). Megengedjük itt az üres diszjunkciót is, amikoris a diszjunktív normálformának nincsen egyetlen tényezője sem. Ekkor az általa definiált Boole-függvény azonosan 0. *Diszjunktív k-normálformán* olyan diszjunktív normálformát értünk, melyben minden elemi konjunkció legfeljebb k literált tartalmaz.

Az $\wedge$ és $\vee$ műveletek szerepét fölcserélve definiálhatjuk az *elemi diszjunkciót* és a *konjunktív normálformát*.

A fentiek szerint tehát minden Boole-függvény kifejezhető diszjunktív normálformával. A diszjunktív normálformából a disztributivitást alkalmazva konjunktív normálformát kaphatunk.

Egyazon Boole-függvényt általában igen sokféleképpen ki lehet fejezni Boole-polinomként. Ha egy ilyen kifejezést megtaláltunk, a függvény értékét már könnyű kiszámolni. Általában azonban egy Boole-függvényt csak igen nagyméretű Boole-polinommal lehet kifejezni, még akkor is, ha gyorsan kiszámolható. Ezért bevezetünk egy általánosabb kifejezési módot.

Legyen G egy olyan irányított gráf, mely nem tartalmaz irányított kört (röviden: aciklikus). A gráf forrásait, vagyis azon csúcsait, melyekbe nem fut bele él, *bemeneti csúcsoknak* nevezzük. Minden bemeneti csúcshoz hozzá van rendelve egy változó vagy a negáltja.

A gráf nyelőit, vagyis azon csúcsait, melyekből nem fut ki él, *kimeneti csúcsoknak* is hívjuk. (A továbbiakban leggyakrabban olyan logikai hálózatokkal lesz dolgunk, melyeknek egyetlen kimeneti csúcsuk van.)

A gráf minden olyan v csúcsához, mely nem forrás, tehát melynek be-foka valamely $d = d_+(v) > 0$, adjunk meg egy „kaput”, vagyis egy $F_v : \{0,1\}^d \to \{0,1\}$ Boole-függvényt. (A függvény változói feleljenek meg a v-be befutó éleknek.) Az ilyen függvényekkel ellátott irányított gráfot *logikai hálózatnak* nevezzük.

A hálózat *mérete* a kapuk száma, a *mélysége* pedig egy bemenet-csúcstól egy kimenet-csúcsig vezető út maximális hossza.

Minden H logikai hálózatot felhasználhatunk egy Boole-függvény kiszámítására. Pontosabban, minden logikai hálózat meghatároz egy Boole-függvényt a következőképpen. Adjuk minden bemeneti pontnak a hozzárendelt literál értékét, ez lesz a számítás *bemenete*. Ebből minden más v csúcshoz ki tudunk számítani egy-egy $x(v) \in \{0,1\}$ értéket, éspedig úgy, hogy ha egy v csúcsra a bele befutó élek $u_1, \ldots, u_d$ $(d = d_+(v))$ kezdőpontjainak már

ki van számítva az értéke, akkor a v-be kerüljön az $F_v(x(u_1), \ldots, x(u_d))$ érték. Az nyelőkhöz rendelt értékek adják a számítás *kimenetét*. Az így definiált Boole-függvényről azt mondjuk, hogy az adott H hálózat *számolja ki*.

Feladat. 16. Bizonyítsuk be, hogy a fenti módon a logikai hálózat minden bemenethez egyértelműen számít ki egy kimenetet.

Természetesen minden Boole-függvényt ki tudunk számítani egy olyan triviális (1 mélységű) logikai hálózattal, melyben egyetlen kapu számítja ki a kimenetet közvetlenül a bemenetből. Akkor lesz hasznos ez a fogalom számunkra, ha olyan logikai hálózatokat tekintünk, melyekben a kapuk maguk csak valamilyen egyszerű művelet kiszámítására alkalmasak (ÉS, VAGY, kizáró VAGY, implikáció, tagadás, stb.). Leggyakrabban minden kapu a bemenő értékek konjunkcióját vagy diszjunkcióját számítja ki; az ilyen logikai hálózatot *Boole-hálózatnak* nevezzük. Másik természetes megszorítás az, hogy minden kapu be-foka legfeljebb kettő. (Néha azt is célszerű föltenni, hogy a kapunk ki-foka is korlátos, vagyis egy csúcs az általa kiszámított bitet nem tudja „ingyen" akárhány helyre szétosztani.)

1.4.1. Megjegyzés. *A logikai hálózatok (speciálisan a Boole-hálózatok) két dolgot is modelleznek. Egyrészt kombinatorikus leírását nyújtják bizonyos egyszerű (visszacsatolás nélküli) elektronikus hálózatoknak. Másrészt – és a mi szempontunkból ez a fontosabb – az algoritmusok logikai struktúráját írják le. Erre vonatkozóan egy általános tételt is be fogunk bizonyítani (1.3.1. tétel), de igen sokszor egy algoritmus közvetlenül jól, áttekinthetően leírható logikai hálózattal.*

Egy logikai hálózat csúcsai felelnek meg egyes műveleteknek. Ezek végrehajtási sorrendje csak annyiban kötött, amennyire ezt az irányított gráf meghatározza: egy él végpontjában szereplő műveletet nem végezhetjük el előbb, mint a kezdőpontjában szereplőt. Így jól írják le algoritmusok párhuzamosíthatóságát: pl. ha egy függvényt h mélységű, n csúcsú Boole-hálózattal tudunk kiszámítani, akkor egy processzorral $O(n)$ időben, de sok (legfeljebb n) processzorral $O(h)$ időben is ki tudjuk számítani (feltéve, hogy a processzorok összekapcsolását, kommunikációját jól oldjuk meg; párhuzamos algoritmusokkal a 8. fejezetben fogunk foglalkozni).

További előnye ennek a modellnek, hogy egyszerűsége révén erős alsó korlátokat lehet benne adni, tehát konkrét függvényekre bizonyítani, hogy nem számíthatók ki kis hálózattal; ilyenekkel a 12. fejezetben foglalkozunk.

Feladatok. 17. Mutassuk meg, hogy minden N méretű Boole-hálózathoz van olyan legfeljebb N^2 méretű, 2 befokú Boole-hálózat, mely ugyanazt a Boole-függvényt számolja ki.

18. Mutassuk meg, hogy minden N méretű, legfeljebb 2 befokú *logikai* hálózathoz van olyan $O(N)$ méretű, legfeljebb 2 befokú Boole-hálózat, mely ugyanazt a Boole-függvényt számolja ki.

19. Mutassuk meg, hogy minden N méretű, legfeljebb 2 befokú Boole-hálózat átalakítható olyanná, mely egyváltozós kapukat nem használ, ugyanazt számolja ki, és mérete legfeljebb kétszeres.

Legyen $f: \{0,1\}^n \to \{0,1\}$ tetszőleges Boole-függvény, és legyen

$$f(x_1,\ldots,x_n) = E_1 \vee \ldots \vee E_N$$

egy előállítása diszjunktív normálformaként. Ennek az előállításnak megfelel egy 2 mélységű Boole-hálózat a következő módon: bemeneti pontjai feleljenek meg az $x_1,\ldots,x_n$ változóknak és az $\overline{x}_1,\ldots,\overline{x}_n$ negált változóknak. Minden E_i elemi konjunkciónak feleljen meg egy csúcs, melybe az E_i-ben fellépő literáloknak megfelelő bemeneti pontokból vezet él, és amely ezek konjunkcióját számítja ki. Végül ezekből a csúcsokból él vezet a t kimeneti pontba, mely ezek diszjunkcióját számítja ki.

Feladat. 20. Mutassuk meg, hogy a Boole-polinomok kölcsönösen egyértelműen megfelelnek azoknak a Boole-hálózatoknak, melyek fák.

Minden Boole-hálózatot tekinthetünk úgy, mint egy Boole-függvény kiszámítására szolgáló algoritmust. Azonnal látható azonban, hogy a logikai hálózatok kevesebbet „tudnak", mint pl. a Turing-gépek: egy Boole-hálózat csak adott hosszúságú bemenettel (és kimenettel) tud foglalkozni. Az is világos, hogy (mivel a gráf aciklikus) az elvégezhető lépések száma is korlátozott. Azonban ha a bemenet hosszát és a lépések számát rögzítjük, akkor alkalmas Boole-hálózattal már minden olyan Turing-gép működését utánozni tudjuk, mely egyetlen bitet számít ki. Úgy is fogalmazhatunk, hogy minden olyan Boole-függvényt, melyet Turing-géppel bizonyos számú lépésben ki tudunk számítani, egy alkalmas nem túl nagy Boole-hálózattal is ki lehet számolni:

1.3.1 Tétel: *Minden T $\Sigma = \{0,1,*\}$ feletti Turing-géphez és minden $N \geq n \geq 1$ számpárhoz van olyan n bemenetű, $O(N^2)$ méretű, $O(N)$ mélységű, legfeljebb 2 befokú Boole-hálózat, mely egy $(x_0,\ldots,x_{n-1}) \in \{0,1\}^n$ bemenetre akkor és csak akkor számol ki* 1*-et, ha az $x_0\ldots x_{n-1}$ bemenetre a T Turing gép N lépése után az utolsó szalag 0-ik mezején* 1 *áll.*

(A Boole-hálózat méretére ill. mélységére tett megszorítások nélkül az állítás triviális volna, hiszen minden Boole-függvény kifejezhető Boole-hálózattal.)

Bizonyítás: Legyen adva egy $T = \langle k, \Sigma, \Gamma, \alpha, \beta, \gamma\rangle$ Turing-gép és $n, N \geq 1$. Az egyszerűség kedvéért tegyük föl, hogy $k = 1$. Szerkesszünk meg egy irányított gráfot, melynek csúcsai a $v(t,g,p)$ és $w(t,p,h)$ pontok, ahol $0 \leq t \leq N$, $g \in \Gamma$, és $-N \leq p \leq N$. Minden $v[t+1,g,p]$ ill. $w[t+1,p,h]$ pontba vezessen él a $v[t,g',p+\epsilon]$ és $w[t,p+\epsilon,h']$ ($g' \in \Gamma$, $h' \in \Sigma$, $\epsilon \in \{-1,0,1\}$) pontokból. Vegyünk fel n bemeneti pontot, $s_0,\ldots,s_{n-1}$-et, és húzzunk s_i-ből élt a $w[0,i,h]$ ($h \in \Sigma$) pontokhoz. Az s_i bemeneti pontra az x_i változót írjuk. A kimeneti pont legyen $w[N,0,1]$. (Ezután, amíg van másik nyelő, azokat törölhetjük.)

A gráf csúcsaiban a Boole-hálózat kiértékelése során kiszámítandó logikai értékek (melyeket egyszerűség kedvéért ugyanúgy jelölünk, mint a megfelelő csúcsot) a T gép egy számolását az $x_1, x_2, \ldots, x_n$ bemeneten írják le a következőképpen: a $v[t,g,p]$ csúcs értéke igaz, ha a t-edik lépés után a vezérlőegység a g állapotban van és a fej a szalag p-edik mezején tartózkodik. A $w[t,p,h]$ csúcs értéke igaz, ha a t-edik lépés után a szalag p-edik mezején a h jel áll.

E logikai értékek közül bizonyosak adottak. A gép kezdetben a START állapotban van, és a fej a 0 mezőről indul:

$$v[0,g,p] = \begin{cases} 1, & \text{ha } g = \text{START és } p = 0, \\ 0, & \text{egyébként,} \end{cases}$$

továbbá a bemenet a szalag $0, \ldots, (n-1)$-ik mezejére van írva:

$$w[0,p,h] = \begin{cases} 1, & \text{ha } (p<0 \text{ vagy } p \geq n) \text{ és } h = *, \\ & \text{vagy ha } 0 \leq p \leq n-1 \text{ és } h = x_p, \\ 0, & \text{egyébként.} \end{cases}$$

A Turing-gép szabályai megmondják, hogy hogyan kell a többi csúcsnak megfelelő logikai értéket kiszámítani:

$$v[t+1,g,p] = \bigvee_{\substack{g' \in \Gamma \\ h' \in \Sigma \\ \alpha(g',h')=g}} \Big(v[t,g',p-\gamma(g',h')] \wedge w[t,p-\gamma(g',h'),h'] \Big)$$

$$w[t+1,p,h] = \Big(w[t,p,h] \wedge \bigwedge_{g' \in \Gamma} \overline{v[t,g',p]} \Big) \vee \Big(\bigvee_{\substack{g' \in \Gamma \\ h' \in \Sigma \\ \beta(g',h')=h}} (v[t,g',p] \wedge w[t,p,h']) \Big)$$

Látható, hogy ezek a rekurziók olyan logikai függvényeknek tekinthetők, melyekkel ellátva a G gráf olyan logikai hálózat lesz, mely a kívánt függvényt számolja ki. A hálózat mérete $O(N^2)$, mélysége $O(N)$. Mivel minden pont be-foka legfeljebb $3|\Sigma| \cdot |\Gamma| = O(1)$, a hálózatot átalakíthatjuk hasonló méretű és mélységű Boole-hálózattá. □

1.4.2. Megjegyzés. *Érdekes módon a másik irány, amikor Boole-hálózatot akarunk szimulálni Turing-géppel, nem megy ilyen egyszerűen. Tegyük fel, hogy minden n-re adott egy n bemenetű $O(n^c)$ méretű Boole-hálózat. Szeretnénk, hogy ekkor van egy olyan Turing-gép, amely minden x bemenetre ugyanazt számolja ki, mint az $|x|$ bemenetű Boole-hálózat, legfeljebb $O(|x|^c)$ lépésben. Ez így nem igaz, mivel a különböző n-ekre a Boole-hálózatok nagyon különbözőek lehetnek. Az állítás csak akkor igaz, ha van egy olyan másik Turing-gép, amely az n bemenetre $O(n^c)$ időben elő tudja állítani a fenti n bemenetű Boole-hálózat egy leírását.*

2. fejezet

Algoritmikus eldönthetőség

Századunk 30-as éveiig az volt a — többnyire nem pontosan kimondott — vélemény a matematikusok körében, hogy minden olyan matematikai kérdést, melyet pontosan meg tudunk fogalmazni, el is tudunk dönteni. Ez igazából kétféleképpen érthető. Lehet egyetlen igen-nem-kérdésről szó, és ekkor az eldöntés azt jelenti, hogy a halmazelmélet (vagy más elmélet) axiómáiból vagy az állítást, vagy az ellenkezőjét be tudjuk bizonyítani. 1931-ben publikálta Gödel azt híres eredményét, mely szerint ez nem így van, sőt az is kiderült, hogy akárhogyan is bővítenénk a halmazelmélet axiómarendszerét (bizonyos ésszerű kikötéseknek megfelelően, pl. hogy ne lehessen ellentmondást levezetni, és egy adott állításról el lehessen dönteni, hogy az axióma-e), mindig maradna megoldatlan probléma.

Az eldönthetőség kérdésének egy másik formája az, amikor egy probléma-seregről van szó, és olyan algoritmust keresünk, mely ezek mindegyikét eldönti. Church 1936-ban fogalmazott meg olyan probléma-sereget, melyről be tudta azt is bizonyítani, hogy algoritmussal nem dönthető el. Ahhoz, hogy egy ilyen bizonyításnak értelme legyen, meg kellett alkotni az algoritmus matematikai fogalmát. Church erre logikai eszközöket alkalmazott. Természetesen elképzelhető lenne, hogy valaki az algoritmusok eszköztárát olyan eszközökkel bővíti, melyek újabb problémák eldöntésére teszik azokat alkalmassá. Church azonban megfogalmazta az ún. **Church-tézist**, mely szerint *minden „számítás" az általa megadott rendszerben formalizálható.*

Ugyanebben az évben Turing megalkotta a Turing-gép fogalmát; azt nevezzük algoritmikusan kiszámíthatónak, ami Turing-gépen kiszámítható. Láttuk az előző fejezetben, hogy nem változtatna ezen, ha a Turing-gép helyett a RAM-ból indulnánk ki. Church eredeti modelljéről és igen sok egyéb számítási modellről is kiderült, hogy ebben az értelemben ekvivalens a Turing-géppel. Olyan modellt, amely (legalábbis determinisztikus, véletlent nem használó módon) több mindent tudna kiszámolni, mint a Turing-gép, senki sem talált. Mindezek alátámasztják a Church-tézist.

2.1. Rekurzív és rekurzíve fölsorolható nyelvek

Legyen Σ egy véges ábécé, mely tartalmazza a „$*$" szimbólumot. Mint már korábban megjegyeztük, Turing-gépek bemeneteként olyan szavakat fogunk megengedni, melyek ezt a speciális jelet nem tartalmazzák, vagyis melyek a $\Sigma_0 = \Sigma - \{*\}$ ábécéből állnak.

Egy $f : \Sigma_0^* \to \Sigma_0^*$ függvényt *kiszámíthatónak* vagy *rekurzívnak* nevezünk, ha van olyan T Turing-gép (tetszőleges k számú szalaggal), mely bármely $x \in \Sigma_0^*$ bemenettel (vagyis első szalagjára az x szót, a többire az üres szót írva), véges idő után megáll, és az utolsó szalagjára az $f(x)$ szó lesz írva.

Megjegyzés: Az 1. fejezetben láttuk, hogy nem változna egy függvény kiszámíthatósága, ha a definícióban feltennénk, hogy $k = 1$.

Legyen $\mathcal{L} \subseteq \Sigma_0^*$ egy nyelv. Az $\mathcal{L}$ nyelvet *rekurzívnak* hívjuk, ha karakterisztikus függvénye:

$$f(x) = \begin{cases} 1, & \text{ha } x \in \mathcal{L}; \\ 0, & \text{ha } x \in \Sigma_0^* - \mathcal{L}. \end{cases}$$

kiszámítható. Ha egy T Turing-gép ezt az f függvényt számítja ki, azt mondjuk, hogy T *eldönti* az $\mathcal{L}$ nyelvet. Nyilvánvaló, hogy minden véges nyelv rekurzív. Az is világos, hogy ha az $\mathcal{L}$ nyelv rekurzív, akkor a komplementere: $\Sigma_0^* - \mathcal{L}$ is az.

Megjegyzés: Nyilvánvaló, hogy kontinuum sok nyelv van, míg a Turing-gépek száma megszámlálható. Így kell lennie nem rekurzív nyelvnek is. Látni fogjuk, hogy vannak konkrét nyelvek, melyekről be lehet bizonyítani, hogy nem rekurzívok.

Az $\mathcal{L}$ nyelvet *rekurzíve fölsorolhatónak* nevezzük, ha vagy $\mathcal{L} = \emptyset$, vagy van olyan kiszámítható $f : \Sigma_0^* \to \Sigma_0^*$ függvény, melynek értékkészlete $\mathcal{L}$. (Másszóval, az $\mathcal{L}$-be tartozó szavakat fel lehet sorolni: $x_1, x_2, \ldots$ – ismétléseket is megengedve – úgy, hogy a $k \mapsto x_k$ függvény rekurzív.) Mint látni fogjuk, rekurzíve fölsorolható nyelvek sok szempontból másként viselkednek, mint a rekurzívak. Például egy rekurzíve fölsorolható nyelv komplementere már nem szükségképpen rekurzíve fölsorolható.

Rekurzíve fölsorolható nyelveknek egy másik fontos definiálási lehetőségét mutatja az alábbi lemma. Rendezzük Σ_0^* elemeit úgy, hogy a rövidebb szavak előzzék meg a hosszabbakat, az egyforma hosszúakat pedig rendezzük lexikografikusan. Könnyű csinálni olyan Turing-gépet, amely Σ_0^* szavait ebben a sorrendben sorba előállítja, és olyat is, mely adott j-hez kiszámítja a j. szót.

2.1.1 Lemma: *Egy $\mathcal{L}$ nyelv akkor és csak akkor rekurzíve fölsorolható, ha van olyan T Turing gép, melynek első szalagjára x-et írva, a gép akkor és csak akkor áll le véges idő múlva, ha $x \in \mathcal{L}$.*

Bizonyítás: Legyen $\mathcal{L}$ rekurzíve fölsorolható; feltehetjük, hogy nem üres. Legyen $\mathcal{L}$ az f függvény értékkészlete. Készítünk egy Turing-gépet, mely egy x bemeneten akkor és csak akkor áll meg véges sok lépésben, ha $x \in \mathcal{L}$. A Turing-gép minden adott x bemenethez sorra veszi az $y \in \Sigma_0^*$ szavakat, kiszámítja $f(y)$-t, és megáll, ha $x = f(y)$.

Megfordítva, tegyük föl, hogy $\mathcal{L}$ azokból a szavakból áll, melyekre egy T Turing-gép véges sok lépésben megáll. Föltehetjük, hogy $\mathcal{L}$ nem üres, és legyen $a \in \mathcal{L}$. Csináljunk egy T_0 Turing-gépet, melynek első szalagjára egy i természetes számot írva, a következőket csinálja: a T első szalagjára (amely T_0-nak, mondjuk, a második szalagja) az $(i - \lfloor\sqrt{i}\rfloor^2)$ -edik szót írja (legyen ez x; így minden szó végtelen sok különböző i bemenetre kerül felírásra). Ezután a T géppel ezen a bemeneten i lépést próbál végeztetni. Ha a T gép ezalatt leáll, akkor T_0 az utolsó szalagjára x-et ír, és leáll. Ha a T gép ezalatt nem áll le, akkor T_0 az utolsó szalagjára a-t ír, és leáll. A T által számított függvény értékkészlete éppen $\mathcal{L}$. □

A rekurzív és rekurzíve felsorolható nyelvek kapcsolatát a logikával a 2.3. alfejezetben tárgyaljuk. Most megvizsgáljuk a rekurzív és rekurzíve fölsorolható nyelvek kapcsolatát. Kezdjük egy egyszerű észrevétellel:

2.1.2 Lemma: *Minden rekurzív nyelv rekurzíve felsorolható.*

Bizonyítás: Legyen $\mathcal{L}$ rekurzív nyelv. Ha $\mathcal{L} = \emptyset$, akkor $\mathcal{L}$ definíció szerint rekurzíve fölsorolható, így föltehetjük, hogy $\mathcal{L}$ nem üres; legyen $a \in \mathcal{L}$. Tekintsük a következő függvényt:

$$f(x) = \begin{cases} x, & \text{ha } x \in \mathcal{L} \\ a, & \text{ha } x \notin \mathcal{L} \end{cases}$$

Nyilvánvaló, hogy ez rekurzív, és értékkészlete éppen $\mathcal{L}$. □

A következő tétel mutatja a pontos kapcsolatot rekurzív és rekurzíve fölsorolható nyelvek között:

2.1.3 Tétel: *Egy $\mathcal{L}$ nyelv akkor és csak akkor rekurzív, ha mind az $\mathcal{L}$ nyelv, mind a $\Sigma_0^* - \mathcal{L}$ nyelv rekurzíve fölsorolható.*

Bizonyítás: Ha $\mathcal{L}$ rekurzív, akkor a komplementere is az, és így az előző lemma szerint mindkettő rekurzíve fölsorolható.

Megfordítva, tegyük föl, hogy $\mathcal{L}$ is és a komplementere is rekurzíve fölsorolható. El akarjuk dönteni, hogy egy x szó $\mathcal{L}$-ben van-e. Csináljunk két gépet, az egyik pontosan az $\mathcal{L}$ szavaira, a másik pontosan a $\Sigma_0^* - \mathcal{L}$ szavaira álljon meg véges időben; valamint egy harmadikat, amelyik azt figyeli, hogy melyik gép áll le. Valamelyik véges sok lépés után biztosan leáll, és akkor tudjuk, hogy melyik nyelvben van x. □

Most megmutatjuk, hogy van rekurzíve fölsorolható, de nem rekurzív nyelv. Legyen T egy k szalagos Turing-gép. Álljon $\mathcal{L}_T$ mindazon $x \in \Sigma_0^*$ szavakból, melyekre fennáll, hogy T minden szalagjára x-et írva, a gép véges sok lépésben megáll.

2.1.4 Tétel: *Az $\mathcal{L}_T$ nyelv rekurzíve fölsorolható. Ha T univerzális Turing-gép, akkor $\mathcal{L}_T$ nem rekurzív.*

Röviden szólva: algoritmikusan nem lehet eldönteni, hogy egy univerzális Turing gép egy adott bemenettel véges időn belül leáll-e. Ezt a feladatot *megállási feladatnak* (halting problem) nevezik.

Bizonyítás: Az első állítás a 2.1.1. lemmából következik. A második állítás bizonyításához az egyszerűség kedvéért tegyük föl, hogy T-nek két szalagja van. Ha $\mathcal{L}_T$ rekurzív volna, akkor $\Sigma_0^* - \mathcal{L}$ rekurzíve fölsorolható volna, és így megadható volna olyan (mondjuk 1 szalagos) T_1 Turing-gép, hogy az x bemeneten T_1 akkor és csak akkor áll le, ha $x \notin \mathcal{L}_T$. A T_1 Turing-gép szimulálható T-n úgy, hogy második szalagjára egy alkalmas p „programot” írunk. Ekkor T mindkét szalagjára p-t írva, akkor és csak akkor áll le, ha T_1 leállna (a szimuláció miatt). T_1 viszont akkor és csak akkor áll le, ha T *nem* áll le ezzel a bemenettel (vagyis ha $p \notin \mathcal{L}_T$). Ellentmondás. □

E tétel bizonyítása emlékeztet annak a ténynek az elemi halmazelméletből ismert bizonyítására, hogy a valós számok halmaza nem megszámlálható. Valójában ez a módszer, az ún. átlós módszer vagy diagonalizálás, igen sok logikai, halmazelméleti és bonyolultságelméleti bizonyításnak az alapja. Ezek közül többet látni is fogunk a továbbiakban.

Az előző tételnek számos változata van, melyek hasonló feladatok eldönthetetlenségét mondják ki. Ahelyett, hogy egy $\mathcal{L}$ nyelv nem rekurzív, szemléletesebben azt fogjuk mondani, hogy az $\mathcal{L}$-et definiáló tulajdonság algoritmikusan eldönthetetlen.

2.1.5 Tétel: *Van olyan 1-szalagos Turing-gép, melyre algoritmikusan eldönthetetlen, hogy egy x bemenettel véges időn belül megáll-e.*

Bizonyítás: Legyen T 2-szalagos univerzális Turing-gép, és konstruáljunk egy 1-szalagos T_0 gépet az 1.1.2 Tétel bizonyításához hasonló módon ($k = 2$-vel), azzal a különbséggel, hogy induláskor az x szó i-edik betűjét ne csak a $(4i)$-edik, hanem a $(4i-2)$-edik mezőre is átmásoljuk. Ekkor T_0 egy x bemeneten szimulálni fogja T működését mindkét szalagján x-szel indulva. Mivel az utóbbiról eldönthetetlen, hogy adott x-re véges időn belül megáll-e, T_0-ról is eldönthetetlen, hogy adott x bemenetre megáll-e. □

E tételnek érdemes megemlíteni még néhány következményét. Egy Turing-gép *leírásának* nevezzük a Σ és Γ halmazok felsorolását (ahol, mint eddig, Γ elemeit Σ_0 fölötti szavak kódolják) és az α, β, γ függvények táblázatát.

2.1.6 Következmény: *Algoritmikusan eldönthetelen, hogy egy (leírásával adott) Turing-gép az üres bemeneten véges időben megáll-e.*

Bizonyítás: Minden x szóra módosíthatjuk úgy az előző tételbeli T_0 gépet, hogy az először az adott x szót írja a szalagra, és utána már úgy működjön, mint T_0. Az így kapott T_x Turing-gép persze függ az x szótól. Ha T_x-ről el tudnánk dönteni (leírása alapján), hogy véges időben megáll-e, akkor azt is tudnánk, hogy T_0 az x bemeneten megáll-e. □

2.1.7 Következmény: *Algoritmikusan eldönthetetlen, hogy egy (leírásával adott) 1-szalagos T Turing-gépre az $\mathcal{L}_T$ nyelv üres-e.*

Bizonyítás: Adott S Turing-géphez konstruáljunk meg egy T Turing-gépet, mely a következőt csinálja: először letöröl mindent a szalagról, utána pedig átalakul az S géppé. Nyilvánvaló, hogy S leírásából a T leírása könnyen megkonstruálható. Így ha S az üres bemeneten véges sok lépésben megáll, akkor T minden bemeneten véges sok lépésben megáll, és ezért $\mathcal{L}_T = \Sigma_0^*$ nem üres. Ha S az üres bemeneten végtelen ideig dolgozik, akkor T minden bemeneten végtelen ideig dolgozik, és így $\mathcal{L}_T$ üres. Így, ha el tudnánk dönteni, hogy $\mathcal{L}_T$ üres-e, akkor azt is el tudnánk dönteni, hogy S az üres bemeneten megáll-e, ami pedig eldönthetetlen. □

Nyilvánvaló, hogy az $\mathcal{L}_T$ nyelv üres volta helyett semmilyen más P tulajdonságot sem tudunk eldönteni, ha P az üres nyelvnek megvan és Σ_0^*-nak nincs meg, vagy megfordítva. Ennél még „negatívabb” eredmény is igaz. Nyelvek egy tulajdonságát *triviálisnak* nevezünk, ha vagy minden $\mathcal{L}_T$ típusú (ahol T tetszőleges Turing-gép) nyelvnek megvan, vagy egyiknek

sem.

2.1.8 Rice Tétele: *Bármely nem-triviális nyelv-tulajdonságra algoritmikusan eldönthetetlen, hogy egy adott $\mathcal{L}_T$ nyelvnek megvan-e.*

Így tehát eldönthetetlen a T leírása alapján, hogy $\mathcal{L}_T$ véges-e, reguláris-e, tartalmaz-e egy adott szót stb.

Bizonyítás: Feltehetjük, hogy az üres nyelvnek nincs meg a P tulajdonsága (különben a tulajdonság tagadását tekinthetjük). Legyen T_1 olyan Turing-gép, melyre $\mathcal{L}_{T_1}$-nek megvan a P tulajdonsága. Adott S Turing-géphez készítsünk el egy T gépet a következőképpen: egy x bemeneten dolgozzon T úgy, mint T_1, de ezzel párhuzamosan dolgozzon úgy is, mint S az üres bemeneten. Így ha S nem áll meg az üres bemeneten, akkor T semmilyen bemeneten nem áll meg, tehát $\mathcal{L}_T$ az üres nyelv. Ha S megáll az üres bemeneten, akkor T pontosan azokon a bemeneteken áll meg, mint T_1, és így $\mathcal{L}_{T_1} = \mathcal{L}_T$. Így ha el tudnánk dönteni, hogy $\mathcal{L}_T$-nek megvan-e a P tulajdonsága, akkor azt is el tudnánk dönteni, hogy S megáll-e az üres bemeneten. □

2.2. Egyéb algoritmikusan eldönthetetlen problémák

Az előző pontban megfogalmazott eldönthetetlen probléma (a megállási probléma) kissé mesterkélt, és a bizonyítás azon múlik, hogy Turing-gépekről akarunk valamit eldönteni Turing-gépekkel. Azt gondolhatnánk, hogy a „valódi életben" fölvetődő matematikai problémák nem lesznek eldönthetetlenek. Ez azonban nem így van! A matematika számos problémájáról derült ki, hogy algoritmikusan eldönthetetlen; ezek között sok olyan is van, mely egyáltalán nem logikai jellegű.

Először egy geometriai jellegű problémát említünk. Tekintsünk egy „dominókészletet" melyben minden „dominó" négyzetalakú, és minden oldalára egy természetes szám van írva. Csak véges sok különböző fajta dominónk van, de mindegyikből végtelen sok példány áll rendelkezésre. Ki van tüntetve továbbá egy „kezdődominó".

2.2.1. Dominó-probléma: *Ki lehet-e rakni ezekkel az adott négyzetekkel a síkot úgy, hogy a kezdődominó kell, hogy szerepeljen, és az egymáshoz illeszkedő oldalakon mindig ugyanaz a szám legyen?*

(Hogy triviális megoldásokat elkerüljünk, kikötjük, hogy a négyzeteket nem szabad elforgatni, olyan állásban kell elhelyezni őket, ahogyan adva vannak.)

Könnyű olyan készletet megadni, amellyel a síkot ki lehet rakni (pl. egyetlen négyzet, melynek minden oldala ugyanazt a számot viseli) és olyat is, mellyel nem lehet (pl. egyetlen négyzet, melynek minden oldala különböző számot visel). Az a meglepő tény igaz azonban, hogy a dominóprobléma algoritmikusan eldönthetetlen!

A pontos megfogalmazáshoz írjunk le minden dominókészletet egy-egy $\Sigma_0 = \{0, 1, +\}$ fölötti szóval, pl. úgy, hogy az egyes dominók oldalaira írt számokat kettes számrendszerben „+" jellel elválasztva leírjuk, a felső oldalon kezdve, az óramutató járásának megfelelő sorrendben, majd a kapott számnégyeseket összefűzzük, a kezdődominóval kezdve. (A kódolás részletei nem lényegesek.) Jelölje $\mathcal{L}_{\text{KIRAK}}$ [ill. $\mathcal{L}_{\text{NEMRAK}}$] azon készletek kódjainak halmazát, melyekkel a sík kirakható [ill. nem rakható ki].

2.2.2. Tétel *Az $\mathcal{L}_{KIRAK}$ nyelv nem rekurzív.*

Elfogadva egyelőre bizonyítás nélkül ezt az állítást, a 2.1.3 tétel szerint vagy a kirakható készletek, vagy a nem kirakható készletek olyan nyelvet kell, hogy alkossanak, mely rekurzíve nem fölsorolható. Vajon melyik? Első pillanatra azt gondolhatnánk, hogy $\mathcal{L}_{\mathrm{KIRAK}}$ rekurzíve felsorolható: az, hogy egy készlettel a sík kirakható, bebizonyítható úgy, hogy megadjuk a kirakást. Ez azonban nem véges bizonyítás, és valójában éppen az ellenkezője igaz:

2.2.3. Tétel *Az $\mathcal{L}_{NEMRAK}$ nyelv rekurzíve felsorolható.*

A 2.2.2 tétellel egybevetve látjuk, hogy $\mathcal{L}_{\mathrm{KIRAK}}$ nem lehet rekurzíve fölsorolható sem. A 2.2.3 tétel bizonyításában fontos szerepet fog játszani az alábbi lemma.

2.2.4. Lemma *Egy készlettel akkor és csak akkor rakható ki a sík, ha minden n-re a $(2n+1)\times(2n+1)$-es négyzet kirakható úgy, hogy közepén a kezdődominó van.*

Bizonyítás: Az állítás „csak akkor" fele triviális. Az „akkor" felének bizonyításához tekintsük négyzeteknek egy $N_1, N_2, \ldots$ sorozatát, melyek mindegyike páratlan oldalhosszúságú, oldalhosszuk végtelenhez tart, és melyek kirakhatók a készlettel. Meg fogjuk konstruálni az egész sík egy kirakását. Az általánosság megszorítása nélkül feltehetjük, hogy minden egyes négyzet középpontja az origó.

Tekintsük először az origó középpontú 3×3-as négyzetet. Ez minden egyes N_i-ben valahogyan ki van rakva a készlettel. Mivel csak véges sokféleképpen rakható ki, lesz végtelen sok olyan N_i, melyben ugyanúgy van kirakva. Az N_i sorozat alkalmas ritkításával feltehetjük, hogy ez a négyzet minden N_i-ben ugyanúgy van kirakva. Ezt a kilenc dominót már rögzíthetjük is.

Továbbmenve, tegyük föl, hogy a sorozatot kiritkítottuk úgy, hogy minden egyes megmaradó N_i az origó középpontú $(2k+1)\times(2k+1)$-es négyzetet ugyanúgy rakja ki, és ezt a $(2k+1)^2$ dominót rögzítettük. Ekkor a megmaradó N_i négyzetekben az origó középpontú $(2k+3)\times(2k+3)$-as négyzet csak véges sokféleképpen van kirakva, így ezek valamelyike végtelen sokszor fordul elő. Ha csak ezeket az N_i négyzeteket őrizzük meg, akkor minden megmaradó négyzet az origó középpontú $(2k+3)\times(2k+3)$-as négyzetet ugyanúgy rakja ki, és ez a kirakás a már rögzített dominókat tartalmazza. Így rögzíthetjük a nagyobb négyzet peremén a dominókat.

A sík bármely egész csúcsú egységnégyzetét fedő dominó előbb-utóbb rögzítve lesz, vagyis az egész sík egy lefedését kapjuk. Mivel a fedésre kirótt feltétel „lokális", azaz csak két szomszédos dominóra vonatkozik, ezek csatlakozása szabályszerű lesz a végső fedésben is. □

A 2.2.3 tétel bizonyítása: Konstruáljunk egy Turing-gépet, mely a következőt csinálja. Adott $x\in\Sigma_0^*$ szóról először is eldönti, hogy az egy dominókészlet kódja-e (ez könnyű); ha nem, akkor végtelen ciklusba megy. Ha igen, akkor ezzel a készlettel megpróbálja rendre az 1×1-es, 3×3-as, stb. négyzeteket kirakni (a kezdődominóval a közepén). Minden egyes konkrét négyzetre véges sok lépésben eldönthető, hogy az kirakható-e. Ha a gép olyan négyzetet talál, mely nem rakható ki az adott készlettel, akkor megáll.

Nyilvánvaló, hogy ha $x\notin\mathcal{L}_{\mathrm{NEMRAK}}$, vagyis x nem kódol készletet, vagy olyan kész-

letet kódol, mellyel a sík kirakható, akkor ez a Turing-gép nem áll meg. Másrészt ha $x \in \mathcal{L}_{\text{NEMRAK}}$, vagyis x olyan készletet kódol, mellyel a sík nem rakható ki, akkor a 2.2.4 lemma szerint elég nagy k-ra már a $k \times k$-as négyzet sem rakható ki, és ezért a Turing-gép véges sok lépés után megáll. Így a 2.1.1 lemma szerint $\mathcal{L}_{\text{NEMRAK}}$ rekurzíve fölsorolható. □

A 2.2.2 tétel bizonyítása: Legyen $T = \langle K, \Sigma, \Gamma, \alpha, \beta, \gamma \rangle$, tetszőleges Turing-gép; konstruálunk hozzá (a leírásából kiindulva) olyan K készletet, mellyel a sík akkor és csak akkor rakható ki, ha T az üres bemeneten nem áll meg. A 2.1.6 következmény miatt azonban az utóbbi algoritmikusan eldönthetetlen, így az is, hogy a megkonstruált készlettel kirakható-e a sík.

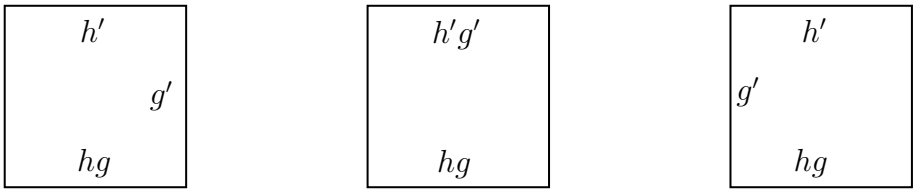
A K készletet úgy adjuk meg, hogy a dominók oldalaira nem számokat, hanem egyéb jeleket írunk; ezt könnyen helyettesíthetjük számokkal. Egyszerűség kedvéért tegyük föl, hogy $k = 1$. Kényelmes lesz azt is föltenni (amit mindig elérhetünk T triviális módosításával), hogy a T gép csak az első lépés előtt van a START állapotban.

	$*$ $*$	1 1	2 2	1 $*\ g_2$	$*\ g_1$ g_1 $*$
	$*$ $*$	1 1	2 $*\ g_1$	$*\ g_2$ $*$	$*$ $*$
	$*$ $*$	1 $*START$	$*\ g_1$ $*$	$*$ $*$	$*$ $*$
	$*$ N N $*$	$*START$ N P $START*$	$*$ P P $*$	$*$ P P $*$	$*$ P P $*$

2.1. ábra. Egy számítás elejének megfelelő kirakás

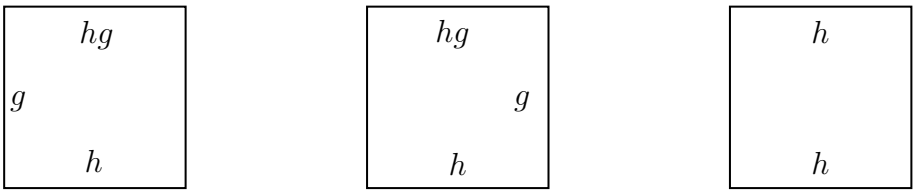
Osszuk fel a síkot olyan egységnégyzetekre, melyeknek középpontja egész koordinátájú pont. Tegyük föl, hogy T az üres bemeneten nem áll meg. Ekkor a gép számolásából

konstruáljunk egy dominókkal való kirakást a következőképpen: ha a szalag p-edik mezejének tartalma q lépés után a h jel, akkor írjuk rá a h jelet annak a négyzetnek a felső oldalára, melynek középpontja a (p, q) pont, és annak a négyzetnek az alsó oldalára, melynek a középpontja a $(p, q+1)$ pont. Ha a q-adik lépés után a fej a p-edik mezőn áll, és a vezérlőegység g állapotban van, akkor írjuk a (p, q) középpontú négyzet felső, és a $(p, q+1)$ középpontú négyzet alsó oldalára a g jelet. Ha a fej a q-adik lépésben jobbra [balra] lép, mondjuk a $(p-1)$-edik [$(p+1)$-edik] négyzetről a p-edikre, és a lépés után a g állapotban van, akkor írjuk rá a g jelet a (p, q) középpontú négyzet bal [jobb] oldalára és a $(p-1, q)$ [$(p+1, q)$] középpontú négyzet jobb [bal] oldalára. A legalsó sorbeli négyzetek függőleges éleire írjunk egy „N” jelet, ha az él az origótól balra van, és egy „P” jelet, ha az él az origótól jobbra van. Tükrözzük a kapott címkézést az x-tengelyre, megfordítva az egy élen levő címkék sorrendjét is. A 2.1 ábra arra az egyszerű Turing-gépre mutatja be a konstrukciót, mely az üres szalagon indulva jobbra lépeget és felváltva 1-est és 2-est ír a szalagra.



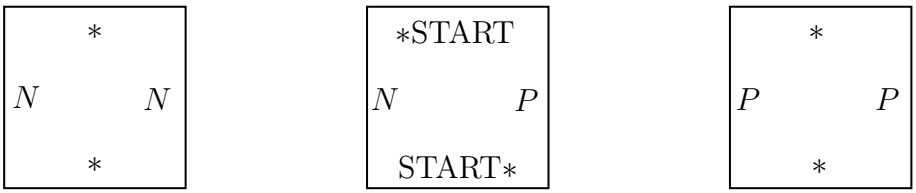
$\alpha(g,h) = g'$	$\alpha(g,h) = g'$	$\alpha(g,h) = g'$
$\beta(g,h) = h'$	$\beta(g,h) = h'$	$\beta(g,h) = h'$
$\gamma(g,h) = 1$	$\gamma(g,h) = 0$	$\gamma(g,h) = -1$

2.2. ábra. A fej alatti cellának megfelelő dominó-típusok



$a)$ $b)$ $c)$

2.3. ábra. Egy cella, a) ahová a fej belép balról, b) ill. jobbról, c) amelyet nem érint a fej



2.4. ábra. Cellák a kezdősorban

Határozzuk meg, hogy az így kapott kirakásban milyen dominók szerepelnek. A felső

félsíkban alapvetően négy fajta van. Ha $q > 0$ és a q-adik lépés után a fej a p-edik helyen áll, akkor a (p, q) középpontú négyzet a 2.2 ábrán látható dominók valamelyike. Ha $q > 0$ és a $(q-1)$-edik lépés után a fej a p-edik helyen áll, akkor a (p, q) középpontú négyzet a 2.3.a)-b) ábrán látható dominók valamelyike. Ha $q > 0$ és a fej sem a q-adik, sem a $q-1$-edik lépés után nem áll a p-edik mezőn, akkor a (p, q) középpontú négyzet egyszerűen a 2.3.c) ábrán látható alakú. Végül az alsó sor négyzeteit a 2.4 ábrán láthatjuk. Az alsó félsíkban szereplő dominókat úgy kapjuk, hogy a fentieket vízszintes tengelyre tükrözzük.

Mármost a 2.2-2.4 ábrák a T Turing-gép leírása alapján megszerkeszthetők; így egy K_T véges készletet kapunk, melynek kezdődominója a 2.4 ábra középső dominója. A fenti gondolatmenet azt mutatja, hogy ha T az üres bemeneten végtelen sok lépésig működik, akkor ezzel a készlettel a sík kirakható. Megfordítva, ha a sík kirakható a K_T készlettel, akkor (mondjuk) a $(0, 0)$ pontot a kezdődominó fedi le; ettől balra ill. jobbra csak a 2.4 ábrán látható másik két dominó állhat. Innen sorról sorra haladva láthatjuk, hogy a lefedés egyértelmű, és a T gép üres bemenetű számolásának felel meg. Mivel az egész síkot lefedtük, ez a számolás végtelen. □

Feladatok. 1. Mutassuk meg, hogy van olyan dominókészlet, mellyel a sík kirakható, de nem rakható ki kétszeresen periodikusan (vagyis úgy, hogy alkalmas lineárisan független egész koordinátájú (p, q) és (r, s)vektorokra bármely (x, y) pontot ugyanolyan dominó fed le, mint az $(x + p, y + q)$ és $(x + r, y + s)$ pontot).

2. Igazoljuk, hogy azok a készletek, melyekkel a sík kétszeresen periodikusan kirakható, rekurzívan fölsorolhatóak.

3. Bizonyítsuk be a következőket:

(a) Van olyan $F : \mathbb{Z}_+ \to \mathbb{Z}_+$ függvény, melyre a következő igaz: ha egy dominókészlet kódjának hossza n, és a $(2F(n) + 1) \times (2F(n) + 1)$-es négyzet kirakható a készlettel úgy, hogy közepén a kezdődominó van, akkor az egész sík kirakható.

(b) Az ilyen tulajdonságú F függvény nem lehet rekurzív.

Megjegyzés: A dominóprobléma akkor is eldönthetetlen, ha nem jelölünk ki kezdődominót. Azonban a bizonyítás lényegesen nehezebb.

Néhány további algoritmikusan eldönthetetlen problémát említünk, az eldönthetetlenség bizonyítása nélkül. Hilbert 1900-ban megfogalmazta az akkori matematika 23 általa legizgalmasabbnak tartott problémáját. Ezek a problémák a század matematikájának fejlődésére igen nagy hatást gyakoroltak. (Érdekes megjegyezni, hogy Hilbert úgy gondolta: problémáival évszázadokig nem fog boldogulni a tudomány; mára mindet lényegében megoldották.) Ezen problémák egyike volt a következő:

2.2.5 Diophantoszi egyenlet: *Adott egy egész együtthatós n változós $p(x_1, ..., x_n)$ polinom, döntsük el, hogy van-e a $p = 0$ egyenletnek egész számokból álló megoldása?*

(Diophantoszinak nevezzük az olyan egyenletet, melynek megoldását egész számokban keressük.)

Hilbert idejében az algoritmus fogalma még nem volt ugyan tisztázva, de az volt az elképzelése, hogy lehet találni olyan, mindenki számára elfogadható és mindig végrehajtható eljárást, mely adott Diophantoszi egyenletről eldönti, hogy megoldható-e. Az algoritmus

fogalmának tisztázása, és az első algoritmikusan eldönthetetlen problémák megtalálása után egyre inkább az vált valószínűvé, hogy a probléma algoritmikusan eldönthetetlen. Ezt a sejtést Davis, Robinson és Myhill egy számelméleti feladatra vezették vissza, melyet végül Matyijaszevics 1970-ben megoldott. Kiderült tehát, hogy a diophantoszi egyenletek megoldhatóságának problémája algoritmikusan eldönthetetlen.

Említünk egy fontos algebrai problémát is. Legyen adva n szimbólum: $a_1, \ldots, a_n$. Az általuk generál *szabad csoporton* az $a_1, \ldots, a_n, a_1^{-1}, \ldots, a_n^{-1}$ jelekből alkotott mindazon (véges) szavak halmazát értjük, melyekben nem fordul elő közvetlenül egymásután a_i és a_i^{-1} (semmilyen sorrendben). Két ilyen szót úgy szorzunk össze, hogy egymásután írjuk, és az esetleg egymás mellé kerülő a_i és a_i^{-1} szimbólumokat ismételten eltöröljük. Meg kell gondolni, de ez nem nehéz, hogy az így definiált szorzás asszociatív. Az üres szót is megengedjük, ez lesz a csoport egységeleme. Egy szót megfordítva és minden a_i kicserélve a_i^{-1}-re (és viszont), kapjuk a szó inverzét. Ebben a nagyon egyszerű struktúrában eldönthetetlen az alábbi probléma:

2.2.6 Csoportok szóproblémája: *Adott az $a_1, \ldots, a_n$ szimbólumok által generált szabad csoportban $n+1$ szó: $\alpha_1, \ldots, \alpha_n$ és β. Benne van-e β az $\alpha_1, \ldots, \alpha_n$ által generált részcsoportban?*

Egy probléma a topológia területéről. Legyenek $e_1 \ldots, e_n$ az n-dimenziós euklideszi tér egységvektorai. A $0, e_1, \ldots, e_n$ pontok konvex burkát *standard szimplexnek* nevezzük. A szimplex *lapjai* a $\{0, e_1, \ldots, e_n\}$ halmaz részhalmazainak konvex burkai. *Poliédernek* nevezzük a standard szimplex lapjai tetszőleges halmazának az egyesítését. Alapvető topológiai kérdés egy P poliéderrel kapcsolatban a következő:

2.2.7 Poliéderek összehúzhatósága: *Összehúzható-e egy adott poliéder (folytonosan, mindig önmagán belül maradva) egy ponttá.*

Ezt pontosan úgy definiáljuk, hogy kijelölünk a poliéderben egy p pontot, és úgy akarjuk a poliéder minden pontját (mondjuk a 0 időponttól az 1 időpontig) mozgatni a poliéderen belül, hogy az végül a p pontba jusson, és közben a poliéder „ne szakadjon szét". Jelölje $F(x,t)$ az x pont helyzetét a t időpontban ($0 \leq t \leq 1$). Ekkor tehát $F : P \times [0,1] \longrightarrow P$ olyan (két változójában együtt) folytonos leképezés, melyre $F(x,0) = x$ és $F(x,1) = p$ minden x-re. Ha ilyen F létezik, akkor azt mondjuk, hogy P *összehúzható*. Ez a tulajdonság azonban eldönthetetlen.

Végül még egy, a dominóproblémához hasonlóan egyszerűnek látszó probléma. Szótárnak nevezünk véges sok (u_i, v_i); $1 \leq i \leq N$ párt, ahol minden i-re $u_i \in \Sigma_0^*$ és $v_i \in \Sigma_0^*$.

2.2.8 Post szóproblémája: *A bemenet egy szótár. Van-e olyan mondat, ami mindkét nyelven ugyanazt jelenti (ha a betűközöktől eltekintünk)? Azaz van-e az indexek olyan $i_1, i_2, \ldots, i_K$ sorozata, hogy $u_{i_1} u_{i_2} \ldots u_{i_K} = v_{i_1} v_{i_2} \ldots v_{i_K}$?*

Meglepő módon ez a tulajdonág is eldönthetetlen.

2.3. Kiszámíthatóság a logikában

2.3.1. Gödel nem-teljességi tétele

A matematikusoknak mindig is az volt a meggyőződésük, hogy egy tökéletes részletességgel leírt bizonyítás helyességét minden kétséget kizáróan le lehet ellenőrizni. Már Arisztotelész megkísérelte formalizálni a levezetés szabályait, de a helyes formalizmust csak a tizenkilencedik század végén találta meg Frege és Russell. Ezt Hilbertnek köszönhetően ismerték el, mint a matematika alapját. Ebben a fejezetben megpróbáljuk ismertetni a logikai eldönthetőség legfontosabb eredményeit.

A matematika *mondatokat* használ, állításokat bizonyos matematikai objektumokról. A mondatok egy véges ábécéből alkotott betűsorozatok. Fel fogjuk tenni, hogy a mondatok halmaza (amit *nyelvnek* szokás hívni) eldönthető, hiszen meg kell tudnunk megkülönböztetni a (formálisan) értelmes mondatokat az értelmetlen karaktersorozatoktól. Ezenkívül feltesszük, hogy létezik egy algoritmus, mely minden φ mondatból kiszámít egy ψ mondatot, amit a φ *negáltjának* hívunk.

1. Példa: Álljon az L_1 nyelv az „$l(a,b)$” és „$l'(a,b)$” alakú mondatokból, ahol a és b természetes számok (tízes számrendszerben írva). Az $l(a,b)$ és az $l'(a,b)$ mondatok legyenek egymás negáltjai.

Egy *bizonyítása* egy T mondatnak egy P karaktersorozat, ami azt mutatja, hogy T igaz. Egy **F** *formális rendszer* vagy más néven *elmélet* egy algoritmus, mely eldönti egy (P,T) párról, hogy P helyes bizonyítása-e T-nek. Az olyan T mondatokat, melyekre létezik **F** szerint helyes bizonyítás, az **F** elmélet *tételeinek* hívjuk.

2. Példa: Bemutatunk egy lehetséges T_1 elméletet az imént definiált L_1 nyelvhez. Hívjuk *axiómáknak* azon „$l(a,b)$” alakú mondatokat, melyekre $b = a+1$. Egy *bizonyítás* egy $S_1,\ldots,S_n$ mondatsorozat mely teljesíti az alábbi feltételt: Ha az S_i a sorozat i. mondata, akkor vagy axióma vagy pedig léteznek $1 \le j,k < i$ és a,b,c egészek, melyekre S_j =„$l(a,b)$”, S_k=„$l(b,c)$” és S_i=„$l(a,c)$”. Ebben az elméletben minden olyan $l(a,b)$ alakú mondat tétel, melyre $a < b$.

Egy elméletet *konzisztensnek* hívunk, ha nincs olyan mondat, hogy ő is és a negáltja is tétel. Az inkonzisztens (nem konzisztens) elméletek érdektelenek, de néha nem lehet eldönteni egy elméletről, hogy konzisztens-e.

Egy S mondatot a $\mathcal{T}$ elmélettől *függetlennek* hívunk, ha sem S, sem negáltja nem tétel $\mathcal{T}$-ben. Egy konzisztens elmélet *teljes*, ha nincsen tőle független mondat.

Például a T_1 nem teljes, mert nem bizonyítható sem $l(5,3)$, sem $l'(5,3)$. De könnyen teljessé tehetjük, ha hozzávesszük axiómának azon $l'(a,b)$ alakú mondatokat, melyekre $a \ge b$, valamint hozzávesszük bizonyításnak a megfelelő mondatsorozatokat.

A nem-teljesség egyszerűen annyit jelent, hogy az elmélet a vizsgált rendszernek csak bizonyos tulajdonságait határozza meg; a többi függ attól, hogy melyik (az elmélet által meghatározott tulajdonságokkal rendelkező) rendszert tekintjük. Tehát bizonyos elméleteknél, ahol fenn akarunk tartani némi szabadságot, nem is akarjuk, hogy teljesek legyenek. De ha egy bizonyos rendszer tulajdonságait akarjuk minél pontosabban leírni, akkor teljes elméletre törekszünk. Például a természetes számokhoz szeretnénk egy teljes rendszert, melyben így minden róluk szóló igaz állítás bizonyítható. A teljes elméleteknek megvan az a kellemes tulajdonsága, hogy bármely állítás igazságát el tudjuk dönteni egy egyszerű

algoritmussal:

2.3.1 Tétel: Ha egy $\mathcal{T}$ elmélet teljes, akkor létezik egy algoritmus, mely minden S mondathoz vagy S-re, vagy S negáltjára ad egy bizonyítást.

Bizonyítás: Az algoritmus sorban felsorolja az összes véges P karaktersorozatot és mindegyikről megnézi, hogy nem bizonyítása-e S-nek vagy a negáltjának. Előbb-utóbb fel fogja sorolni valamelyiknek a bizonyítását, hiszen a teljesség miatt tudjuk, hogy létezik valamelyik. Egyébként a konzisztencia garantálja, hogy csak az egyikre létezik bizonyítás. □

Tegyük fel, hogy a természetes számokhoz akarunk csinálni egy teljes elméletet. Mivel az összes karaktersorozatokról, mátrixokról, Turing-gépekről szóló állítás belekódolható a természetes számokba, ezért az elméletnek az ezekről szóló mondatokat is el kell tudnia döntenie, hogy igazak-e.

Legyen L a természetes számok egy rekurzíve felsorolható részhalmaza, mely nem rekurzív. (Az előző fejezetben megmutattuk, hogy ilyen nyelv létezik.) L a továbbiakban fix. A $\mathcal{T}$ aritmetikai elmélet *minimálisan megfelelő*, ha minden n természetes számra az elméletben benne van egy φ_n mondat, mely azt az állítást fejezi ki, hogy „$n \in L$". Ezenkívül még megköveteljük, hogy ez az állítás akkor és csak akkor legyen tétel $\mathcal{T}$-ben, ha igaz.

Egy természetes követelmény, hogy a természetes számok egy teljes elmélete legyen minimálisan megfelelő, azaz hogy az „$n \in L$" típusú egyszerű állítások megfogalmazhatók és bizonyíthatók legyenek benne. (A következő fejezetben majd adunk egy ilyen minimálisan megfelelő elméletet.) Most már be tudjuk látni a matematika egyik leghíresebb tételét, mely a filozófusok kedvence:

2.3.2 Gödel nem-teljességi tétele: Minden minimálisan megfelelő elmélet nem-teljes.

Bizonyítás: Ha az elmélet teljes lenne, akkor a 2.3.1 Tétel szerint minden $n \in L$ alakú mondat eldönthető lenne benne, de ez ellentmond annak, hogy L nem rekurzív.

□

Megjegyzés: Az előző bizonyításból az is következik, hogy minden minimálisan megfelelő elmélethez létezik olyan n szám, melyre az „$n \notin L$" állítás igaz és megfogalmazható, de nem bizonyítható. Ha további, erősebb feltételeket is megkövetelünk a $\mathcal{T}$ elmélettől, akkor más mondatokról is belátható, hogy nem bizonyíthatók; Gödel megmutatta, hogy alkalmas feltételek mellett a $\mathcal{T}$ elmélet konzisztenciáját állító mondat sem bizonyítható a $\mathcal{T}$ elméletben. Ezt Gödel második nem-teljességi tételének hívják, de itt ennek bizonyításával nem foglalkozunk.

Megjegyzés: Gödel nem-teljességi tételei 3-4 évvel a kiszámíthatóság fogalma előtt születtek.

2.3.2. Elsőrendű logika

Formulák Bemutatunk egy formálist rendszert, melyet a legalkalmasabbnak vélnek a matematika leírására. Egy elsőrendű nyelv az alábbi szimbólumokból építkezik:

- Megszámlálható sok változójel: $x, y, z, x_1, x_2, \ldots$, melyek a leírni kívánt univerzum elemeit jelölhetik.

- Néhány függvényjel, mint $f, g, h, +, \cdot, f_1, f_2, \ldots$, melyek mindegyikéhez hozzá van rendelve egy természetes szám, mely meghatározza az argumentumainak számát. Amelyekre ez a szám 0, azokat a függvényeket *konstansnak* hívjuk. Ezek az univerzum valamely fix elemét jelölik. Néhány függvénynél, mint például a +-nál, az argumentumokat nem a függvényjel után, hanem a hagyományos módon a függvényjel köré írjuk.

- Néhány relációjel, mint $<, >, \subset, \supset, P, Q, R, P_1, P_2, \ldots$, melyeknek szintén meg van határozva az argumentumaik száma. Néhány relációjelnél, mint például a <-nél, az argumentumokat nem a relációjel után, hanem a hagyományos módon a relációjel köré írjuk. Az *egyenlőség* („=") egy kitüntetett relációjel.

- Logikai műveletek: $\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow, \ldots$.

- Kvantorok: $\forall, \exists$.

- Zárójelek: (,).

Egy *kifejezést* úgy kapunk, hogy néhány konstansjelre és változójelre alkalmazunk néhány függvényt. Pl: $(x + 2) + y$ vagy $f(f(x, y), g(c))$ kifejezések (itt 2 egy konstans).

Primformulának hívjuk a $P(t_1, \ldots, t_k)$ alakú állításokat, ahol P relációjel és t_i-k kifejezések. Pl: $x + y < (x \cdot x) + 1$ egy primformula.

Egy *formula* primformulákból áll, melyeket logikai műveletek kapcsolnak össze, melyek elé még tetszés szerint írhatunk $\forall x$ és $\exists x$ kvantorokat: Pl: $\forall x(x < y) \Rightarrow \exists z g(c, z)$ vagy $x = x \vee y = y$ formulák. A $\exists y(\forall x(F) \Rightarrow G)$ formulában az F részformulát az x kvantor *hatáskörének hívjuk*. Egy x változó előfordulását egy formulában kötöttnek mondjuk, ha benne van valamely x kvantor hatáskörében, egyébként pedig szabadnak. Egy formulát, melyben nincsenek szabad (előfordulású) változók, *mondatoknak* hívjuk; ezek a változók kiértékelésétől függetlenül vagy igazak, vagy hamisak.

Egy t kifejezés *helyettesíthető* az x változóval az A formulában, ha nincs a t-ben olyan y változó, mely szerepel A-ban kvantorként és a hatáskörében benne van x egy szabad előfordulása. Ha t helyettesíthető x-szel A-ban, akkor jelölje $A[t/x]$ a helyettesítést, ami azt jelenti, hogy x minden szabad előfordulásakor x helyett t-t írunk. Pl: Ha $A = (x < 3 - x)$ és $t = (y^2)$, akkor $A[t/x] = (y^2 < 3 - y^2)$.

Mostantól minden vizsgált formális rendszer elsőrendű nyelv lesz, tehát csak abban térnek el egymástól, hogy mik bennük a függvény- és változójelek.

Több természetes mód is van, hogy egy elsőrendű nyelv kifejezéseit és formuláit interpretáljuk. Egy interpretáció után minden mondat vagy igaz lesz, vagy hamis. Egy interpretáció egy adott halmaz, az *univerzum* elemeit, függvényeit és relációit rendeli hozzá az elsőrendű nyelv konstansjeleihez, függvényjeleihez és relációjeleihez.

Példa: Tekintsük azt a nyelvet, melynek a konstansjelei c_0 és c_1, ezenkívül pedig még van benne egy kétváltozós f függvényjel. Egy lehetséges interpretáció a természetes számok halmaza, $c_0 = 0$, $c_1 = 0$ és $f(a, b) = a + b$. Egy másik lehetséges interpretációban a halmaz $\{0, 1\}$, $c_0 = 0$, $c_1 = 1$ és $f(a, b) = a \cdot b$. Vannak olyan mondatok melyek mindkét esetben

igazak, de vannak, amik nem. Például mindkét interpretációban igaz a $\forall x \forall y\, f(x,y) = f(y,x)$ mondat, de az $f(c_1,c_1) = c_1$ csak az egyikben igaz.

Egy adott T elmélet nyelvének interpretációját T *modelljének* hívjuk, ha az összes T-beli axióma (és emiatt az összes tétel is) igaz benne. A fenti példa mindkét interpretációja modellje volt az egyetlen axiómából álló $T_1 = \{\forall x \forall y\, f(x,y) = f(y,x)\}$ elméletnek.

Régóta ismert, hogy a bizonyítás helyességét ellenőrző algoritmusnak nem kell függenie az általa vizsgált elmélettől; csak az elmélet axiómáit kell betáplálnunk neki. Ezt az algoritmust szokás „józan paraszti észnek" hívni. Az elsőrendű logikában ezt először Russell és Whitehead formalizálta a Principia Mathematica című könyvében a huszadik század elején. Ennek a fejezetnek a végén vázolni fogunk egy ilyen algoritmust. GÖDEL 1930-ban bebizonyította, hogy ha $\mathcal{B}$-ből következik T az összes lehetséges interpretáció esetén, akkor T be is bizonyítható $\mathcal{B}$-ből (a Principia Mathematica szerinti definíció szerint). Ennek a következménye az alábbi tétel:

2.3.3 Gödel teljességi tétele: Legyen $\mathcal{P}$ az összes olyan $(\mathcal{B}, T)$ pár halmaza, hogy $\mathcal{B}$ véges sok mondat és a T mondat minden olyan interpretációban igaz, melyben a $\mathcal{B}$-beli mondatok igazak. Ekkor $\mathcal{P}$ rekurzív felsorolható.

Tarski bebizonyította, hogy a valós számok algebrai elmélete (és ennek következményeként az Euklideszi geometria is) teljes. Ezzel szemben a természetes számok elméletei, köztük a minimálisan megfelelők, nem-teljesek. (A valós számok algebrai elméletében nem beszélhetünk „tetszőleges egész számról", csak „tetszőleges valós számról".) A 2.3.1 Tétel garantálja, hogy létezik algoritmus, mely eldönti, hogy egy a valós számokról szóló mondat igaz-e. Az eddig ismert algoritmusok a gyakorlatban használhatatlanok lassúságuk miatt, de még fejlődnek.

Bizonyítások A *bizonyítás* $F_1, \ldots, F_n$ formulák sorozata, ahol mindegyik formula vagy axióma, vagy a korábbi formulákból kapható az alábbi szabályok valamelyikével. Ezekben a szabályokban A, B és C tetszőleges formulák, x pedig egy változó.

Egy végtelen sok formulából álló halmazról meg fogjuk követelni, hogy minden axiómarendszerben benne legyenek; ezeket fogjuk *logikai axiómáknak* hívni. Ezek nem feltétlenül mondatok, lehetnek bennük szabad változók is. Ezeknek megadása előtt, szükségünk van még néhány definícióra.

Legyen $F(X_1, \ldots, X_n)$ olyan Boole-formula, mely az $X_1, \ldots, X_n$ változók tetszőleges kiértékelése mellet igaz. Legyenek $\varphi_1, \ldots, \varphi_n$ tetszőleges formulák. Ekkor az $F(\varphi_1, \ldots, \varphi_n)$ alakú formulákat *tautológiáknak* hívjuk.

Rendszerünk logikai axiómái az alábbi csoportokba tartoznak:

Tautológiák: Minden tautológia axióma.

Egyenlőségi axiómák: Legyenek $t_1, \ldots, t_n$ és $u_1, \ldots, u_n$ kifejezések, f függvényjel és P relációjel, melyek argumentumainak száma n. Ekkor

$$\begin{aligned} (t_1 = u_1 \wedge \cdots \wedge t_n = u_n) &\Rightarrow f(t_1, \ldots, t_n) = f(u_1, \ldots, u_n), \\ (t_1 = u_1 \wedge \cdots \wedge t_n = u_n) &\Rightarrow (P(t_1, \ldots, t_n) \Leftrightarrow P(u_1, \ldots, u_n)) \end{aligned}$$

axiómák.

A $\exists$ definíciója: Minden A formulára és x axiómára a $\exists x\, A \Leftrightarrow \neg\forall x\, \neg A$ formula axióma.

Specializáció: Ha a t kifejezés helyettesítheti az x változót az A formulában, akkor $\forall x\, A \Rightarrow A[t/x]$ axióma.

A rendszernek két levezetési szabálya van:

Modus ponens: $A \Rightarrow B$-ből és $B \Rightarrow C$-ből, következik $A \Rightarrow C$.

Általánosítás: Ha az x változónak nincs szabad előfordulása az A formulában, akkor $A \Rightarrow B$-ből következik $A \Rightarrow \forall x\, B$.

Megjegyzés: Az általánosítási szabály azt mondja ki, hogyha B igaz anélkül, hogy bármit is kikötnénk x-ről, akkor igaz bármely x-re. Ez *nem* ugyanaz, mint hogy $B \Rightarrow \forall x B$ igaz.

A fenti rendszerre a Gödel teljességi tétel egy erősebb verziója is igaz.

2.3.4 Tétel: Tegyük fel, hogy $\mathcal{B}$ mondatok egy halmaza és T egy mondat, mely minden interpretációban igaz, melyben a $\mathcal{B}$-beli mondatok mind igazak. Ekkor T bebizonyítható, ha a $\mathcal{B}$-beli mondatokat is hozzávesszük a fenti axiómákhoz.

Egy egyszerű aritmetikai elmélet és a Church-tézis Ez az N elmélet két konstansjelet, a 0-t és az 1-et tartalmazza, valamint két függvényjelet, a $+$-t és a $\cdot$-t, ezenkívül pedig még a $<$ relációjelet. Csak véges sok nemlogikai axiómát tartalmaz, melyek mind kvantormentesek.

$$\begin{aligned}
\neg(x+1 &= 0).\\
1+x=1+y &\Rightarrow x=y.\\
x+0 &= x.\\
x+(1+y) &= 1+(x+y).\\
x\cdot 0 &= 0.\\
x\cdot(1+y) &= (x\cdot y)+x.\\
\neg(x &< 0).\\
x<(1+y) &\Leftrightarrow x<y\vee x=y.\\
x<y\vee x=y\vee y<x.&
\end{aligned}$$

2.3.5 Tétel: Az N elmélet minimálisan megfelelő. Tehát létezik az aritmetikának egy végesen axiomatizált minimálisan megfelelő konzisztens elmélete.

Ennek következménye Church alábbi tétele, mely azt mutatja, hogy nincs olyan algoritmus, mely eldönti egy formuláról, hogy igaz-e.

2.3.6 Az igazságok eldönthetetlenségi tétele: Az üres axiómarendszerből levezethető $\mathcal{P}$ mondatok halmaza eldönthetetlen.

Bizonyítás: Legyen $\mathcal{N}$ az aritmetika egy minimálisan megfelelő konzisztens elméletének véges axiómahalmaza, és legyen N az a mondat, melyet úgy kapunk, hogy ezt a véges sok axiómát össze-éseljük és az univerzális kvantorral az összes szabad változóját lekötjük. Emlékeztetünk, hogy a „minimálisan megfelelő" definíciójában használtuk a természetes számok egy L nem rekurzív, de rekurzíve felsorolható halmazát. Az aritmetikánkban írjuk fel azt a $Q(n)$ formulát, mely azt mondja, hogy $N \Rightarrow (n \in L)$. Akkor és csak akkor lesz „$n \in L$" bizonyítható N-ben, ha $Q(n)$ bizonyítható az üres axiómarendszerből. Márpedig a 2.3.2 Tétel utáni megjegyzésből következik, hogy van olyan n, melyre „$n \in L$" nem bizonyítható N-ben, tehát $Q(n)$ sem az az üres axiómarendszerből.

□

3. fejezet

Tár és idő

Az egyes feladatok algoritmikus megoldhatósága igen messze lehet a *gyakorlati* megoldhatóságtól. Mint látni fogjuk, van olyan algoritmikusan megoldható feladat, amelyet n hosszúságú bemenet esetén nem lehet pl. 2^{2^n}-nél kevesebb lépésben megoldani. Kialakult ezért az algoritmusok elméletének egyik fő ága, mely az egyes feladatok algoritmikus megoldhatóságát bizonyos erőforrás-korlátozások mellett vizsgálja. A legfontosabb erőforrás-korlátozások az *idő* és a *tár*.

Rögzítsünk egy m betűből álló Σ ábécét, és mint korábban, legyen $\Sigma_0 = \Sigma - \{*\}$. Ebben a fejezetben a Turing-gépekről feltesszük, hogy van egy bemenet-szalagjuk (erről csak olvasnak), egy kimenet-szalagjuk (erre csak írnak), és $k \geq 1$ további munkaszalagjuk. Induláskor csak a bemenet-szalagra van írva egy Σ_0 fölötti szó.

Egy T Turing-gép *időigénye* az a $\mathrm{time}_T(n)$ függvény, mely a gép lépésszámának maximumát adja meg n hosszúságú bemenet esetén. Föltesszük, hogy $\mathrm{time}_T(n) \geq n$ (a gépnek el kell olvasnia a bemenetet; ez nem okvetlenül van így, de csak triviális eseteket zárunk ki ezzel a feltevéssel). A $\mathrm{space}_T(n)$ *tárigény*-függvényt úgy definiáljuk, mint a gép szalagjain azon különböző mezők maximális számát az n hosszúságú bemenetek esetén, melyekre a gép ír. (Így a bemenet által elfoglalt mezőket nem számítjuk a tárba.) Nyilván $\mathrm{space}_T(n) \geq 1$.

Azt mondjuk, hogy a T Turing-gép *polinomiális,* ha időigénye $O(f)$ valamely f polinomra, vagyis van olyan $c > 0$ konstans, hogy T időigénye $O(n^c)$. Hasonlóan definiálhatjuk az exponenciális algoritmusokat ($O(2^{n^c})$ időigényű valamely $c > 0$-ra), polinomiális tárigényű algoritmusokat (Turing-gépeket), stb.

Azt mondjuk, hogy egy $\mathcal{L} \subseteq \Sigma_0^*$ nyelv *időbonyolultsága* legfeljebb $f(n)$, ha a nyelv egy legfeljebb $f(n)$ időigényű Turing-géppel eldönthető. A legfeljebb $f(n)$ időbonyolultságú nyelvek osztályát $\mathrm{DTIME}(f(n))$-nel jelöljük. (A „D” betű arra utal, hogy determinisztikus algoritmusokat tekintünk; a későbbiekben nem-determinisztikus és véletlent használó használó algoritmusokat is fogunk tárgyalni.) Mindazon nyelvek osztályát, melyek polinomiális Turing-géppel eldönthetők, PTIME-mal vagy egyszerűen P-vel jelöljük. Hasonlóan definiáljuk egy nyelv *tárbonyolultságát*, és a $\mathrm{DSPACE}(f(n))$ nyelvosztályokat és a PSPACE (polinomiális tárral eldönthető) nyelvosztályt.

A Turing-géppel definiált idő- és tárigény elméleti vizsgálatokra alkalmas; gyakorlatban jobban használható (jobban közelíti a valóságot), ha erre a RAM-ot használjuk. Az 1.2.2 és 11.2.3 tételekből következik azonban, hogy a legfontosabb bonyolultsági osztályok (polinomiális, exponenciális idő és tár, stb.) szempontjából mindegy, hogy melyik gépet használjuk a definícióban.

3.1. Polinomiális idő

A gyakorlatban fontos algoritmusok közül igen sok polinomiális idejű (röviden polinomiális). A polinomiális algoritmusok gyakran matematikailag is igen érdekesek, mély eszközöket használnak. Ebben a könyvben nem célunk ezek áttekintése; csak néhány egyszerű, de fontos speciális algoritmust tárgyalunk, részben a fogalom illusztrálása, részben néhány fontos alapgondolat bevezetése céljából. Megjegyzendő, hogy az 1.2.2 és 1.2.3 tételek szerint a polinomiális algoritmus fogalma nem változik, ha a Turing-gép helyett a RAM-ot tekintjük.

a) Kombinatorikai algoritmusok. Polinomiálisak a gráfelméletben használt legfontosabb algoritmusok: összefüggőség-teszt, legrövidebb út keresése, maximális folyam keresése (Edmonds–Karp vagy Dinic–Karzanov módszerrel), „magyar módszer", Edmonds párosítás algoritmusa, stb. Ezek egyike-másika meglehetősen nehéz, és ezek az algoritmusok más tárgyak (gráfelmélet, operációkutatás) anyagának fontos részei. Ezért itt nem foglalkozunk velük részletesen.

b) Aritmetikai algoritmusok. Polinomiálisak az alapvető aritmetikai műveletek: egész számok összeadása, kivonása, szorzása, maradékos osztása. (Emlékezzünk rá, hogy egy n egész szám, mint bemenet hossza az n bináris jegyeinek száma, vagyis $\log_2 n$.) Mindezekre polinomiális idejű (az összeadás és kivonás esetén lineáris, a szorzás és osztás esetén kvadratikus idejű) algoritmust az általános iskolában tanulunk. Triviális, de alapvető aritmetikai műveletként tartjuk számon két szám nagyság szerinti összehasonlítását is. Természetesen ez is elvégezhető lineáris időben.

Számelméleti és algebrai algoritmusoknál kényelmes néha az *aritmetikai műveleteket* számolni; a RAM-on ez annak felel meg, hogy a programnyelvet a szorzással és a maradékos osztással bővítjük, és nem a futási időt, hanem a lépésszámot nézzük. Ha (a bemenet hosszában mérve) polinomiális számú műveletet végzünk, és ezeket legfeljebb polinomiális sok jegyű számokon, akkor algoritmusunk (futási időben mérve is) polinomiális lesz.

Az alapvető, polinomiális idejű aritmetikai algoritmusok között kell még megemlíteni az *euklideszi algoritmust* két természetes szám legnagyobb közös osztójának megkeresésére:

Euklideszi algoritmus. Adott két természetes szám, a és b. Válasszuk ki a nemnagyobbikat, legyen ez, mondjuk, a. Ha $a = 0$, akkor a és b legnagyobb közös osztója $\text{lnko}(a, b) = b$. Ha $a \neq 0$, akkor osszuk el b-t maradékosan a-val, és legyen a maradék r. Ekkor $\text{lnko}(a, b) = \text{lnko}(r, a)$, és így elegendő a és r legnagyobb közös osztóját meghatározni.

3.1.1 Lemma. *Az euklideszi algoritmus polinomiális idejű. Pontosabban, $O(\log a + \log b)$ aritmetikai műveletből áll, melyeket a, b-nél nem nagyobb természetes számokon kell végezni.*

Bizonyítás: Mivel $0 \leq r < b$, az euklideszi algoritmus előbb-utóbb véget ér. Belátjuk, hogy polinomiális időben ér véget. Ehhez azt vegyük észre, hogy $b \geq a + r > 2r$ és így $r < b/2$. Ezért $ar < ab/2$. Tehát $\lceil \log(ab) \rceil$ iteráció után a két szám szorzata kisebb lesz, mint 1, és így valamelyikük 0, vagyis az algoritmus véget ér. Nyilvánvaló, hogy minden

iteráció polinomiális időben elvégezhető (sőt, egy darab maradékos osztással). □

Érdemes megjegyezni, hogy az euklideszi algoritmus nemcsak a legnagyobb közös osztó értékét adja meg, hanem olyan p, q egész számokat is szolgáltat, melyekre $\text{lnko}(a, b) = pa + qb$. Ehhez egyszerűen az algoritmus során kiszámított számok mindegyének egy ilyen előállítását is nyilvántartjuk. Ha $a' = p_1a + q_1b$ és $b' = p_2a + q_2b$, és mondjuk b'-t osztjuk maradékosan a'-vel: $b' = ha' + r'$, akkor $r' = (p_2 - hp_1)a + (q_2 - hq_1)b$, így megkapjuk az új r' szám előállítását is $p'a + q'b$ alakban.

Feladat. 1. Legyen $1 \leq a \leq b$, és jelölje F_k a b-nél nem nagyobb Fibonacci számok közül a legnagyobbat (a Fibonacci számokat az $F_0 = 0$, $F_1 = 1$, $F_{k+1} = F_k + F_{k-l}$ rekurzió definiálja). Bizonyítsuk be, hogy az (a, b) párra alkalmazott euklideszi algoritmus legfeljebb k aritmetikai lépés után véget ér.

Megjegyzés. Az euklideszi algoritmust néha a következő iterációval adják meg: Ha $a = 0$, akkor készen vagyunk. Ha $a > b$, akkor cseréljük meg a számokat. Ha $0 < a < b$, akkor legyen $b := b - a$. Bár matematikailag lényegében ugyanaz történik, ez az algoritmus *nem polinomiális*: Már az lnko(1,b) kiszámításához b iterációra van szükség, ami a bemenet méretében ($\log b$) exponenciálisan nagy.

Az összeadás, kivonás, szorzás műveletek a modulo m vett maradékosztályok gyűrűjében is elvégezhetők polinomiális időben. A maradékosztályokat a legkisebb nem-negatív maradékkal reprezentáljuk. Ezeken, mint egész számokon elvégezzük a műveletet, majd egy maradékos osztást kell még elvégezni.

Ha m prímszám, akkor az osztást is elvégezhetjük a modulo m vett maradékosztályok testében polinomiális idő alatt (ez nem maradékos osztás!). Általánosabban, a modulo m maradékosztálygyűrűben el tudjuk végezni egy m-hez relatív prím számmal az osztást polinomiális időben. Legyen $0 \leq a, b < m$, $\text{lnko}(m, b) = 1$. Az $a : b$ osztás elvégzése azt jelenti, hogy olyan $0 \leq x < m$ egész számot keresünk, melyre

$$bx \equiv a \pmod{m}.$$

Az euklideszi algoritmust a b és m számok legnagyobb közös osztójának kiszámítására alkalmazva olyan p és q egész számokat kapunk, melyekre $bp + mq = 1$. Így $bp \equiv 1 \pmod{m}$, vagyis $b(ap) \equiv a \pmod{m}$. Így a keresett x hányados az ap szorzat m-mel vett osztási maradéka.

Az euklideszi algoritmusnak még egy alkalmazását említjük. Tegyük föl, hogy egy x egész számnak (melyet nem ismerünk) ismerjük az egymáshoz relatív prím $m_1, \ldots, m_k$ modulusokra vett $x_1, \ldots, x_k$ maradékát. A Kínai Maradéktétel szerint ez egyértelműen meghatározza x-nek az $m_1 \ldots m_k$ szorzatra vett osztási maradékát. De hogyan lehet ezt kiszámítani?

Elegendő a $k = 2$ esettel foglalkozni, mert általános k-ra az algoritmus ebből teljes indukcióval adódik. Van olyan q_1 és q_2 egész szám, melyre $x = x_1 + q_1m_1$ és $x = x_2 + q_2m_2$. Így $x_2 - x_1 = q_1m_1 - q_2m_2$. Ez az egyenlet természetesen nem határozza meg egyértelműen a q_1 és q_2 számokat, de az is elegendő lesz számunkra, hogy az előzőekhez hasonlóan, az euklideszi algoritmus segítségével tudunk olyan q_1' és q_2' egész számokat találni, melyekre $x_2 - x_1 = q_1'm_1 - q_2'm_2$. Legyen ugyanis $x' = x_1 + q_1'm_1 = x_2 + q_2'm_2$. Ekkor $x' \equiv x$

(mod m_1) és $x' \equiv x \pmod{m_2}$, és ezért $x' \equiv x \pmod{m}$. Azonban ilyen q_1' és q_2' egész számokat az euklideszi algoritmus valóban ad, ha lefuttatjuk az lnko$(m_1, m_2) = 1$ feladatra.

Érdemes a hatványozás műveletéről is foglalkozni. Mivel a 2^n számnak már a leírásához is exponenciálisan sok jegy kell (ha a bemenet hosszát, mint n kettes számrendszerbeli jegyeinek számát értelmezzük), így ez természetesen nem számítható ki polinomiális időben. Más a helyzet azonban, ha modulo m akarjuk a hatványozást elvégezni: ekkor a^b (modulo m) is egy maradékosztály, tehát leírható $\log m$ jellel. Megmutatjuk, hogy nemcsak, hogy leírható, hanem ki is számítható polinomiálisan.

3.1.2 Lemma. *Legyen a, b és m három természetes szám. Ekkor a^b (modulo m) kiszámítható polinomiális időben, pontosabban $O(\log b)$ aritmetikai művelettel, melyeket $O(\log m + \log a)$ jegyű természetes számokon végzünk.*

Algoritmus. Írjuk fel b-t 2-es számrendszerben: $b = 2^{r_1} + \ldots + 2^{r_k}$, ahol $0 \le r_1 < \ldots < r_k$. Nyilvánvaló, hogy $r_k \le \log b$ és ezért $k \le \log b$. Mármost az $a^{2^t} \pmod{m}$ maradékosztályokat ($0 \le t \le \log b$) ismételt négyzetreemeléssel könnyen meg tudjuk kapni, majd ezek közül a k megfelelő számot szorozzuk össze. Természetesen minden műveletet modulo m végzünk, vagyis minden szorzás után egy maradékos osztást is elvégzünk. □

Megjegyzés. Nem ismeretes, hogy kiszámítható-e polinomiális időben $a!$ modulo m, vagy $\binom{a}{b}$ modulo m.

c) Lineáris algebrai algoritmusok. Polinomiálisak az alapvető lineáris algebrai algoritmusok: vektorok összeadása, skaláris szorzása, mátrixok szorozása, invertálása, determinánsok kiszámítása. E két utóbbi feladat esetében ez nem triviális, így ezekkel külön foglalkozunk.

Legyen $A = (a_{ij})$ tetszőleges $n \times n$-es egész számokból álló mátrix. Először is gondoljuk meg, hogy $\det(A)$ polinomiális kiszámítása nem eleve lehetetlen, azaz az eredmény felírható polinomiális sok számjeggyel. Legyen $K = \max |a_{ij}|$, ekkor nyilván A leírásához $\langle A \rangle \ge n^2 + \log K$ bit kell. Másrészről nyilván

$$|\det(A)| \le n! K^n,$$

így

$$\langle \det(A) \rangle \le \log(n! K^n) + O(1) \le n(\log n + \log K) + O(1) < \langle A \rangle^2.$$

Így tehát $\det(A)$ felírható polinomiálisan sok számjeggyel. Mivel A^{-1} minden eleme A két aldeterminánsának a hányadosa, így A_{-1} is felírható polinomiálisan sok számjeggyel.

Feladat. 2. Mutassuk meg, hogy $\langle \det(A) \rangle \le \langle A \rangle$.

A determináns kiszámítására az ún. Gauss-elimináció a szokásos eljárás. Ezt úgy tekintjük, mint a mátrix alsó háromszögmátrixszá transzformálását oszlopműveletekkel. Ezek a determinánst nem változtatják meg, és a kapott háromszögmátrixra csak a főátlóban levő elemeket kell összeszorozni, hogy a determinánst megkapjuk. (Az inverz mátrixot is könnyű megkapni ebből az alakból; ezzel külön nem foglalkozunk.)

Gauss-elimináció. Tegyük föl, hogy már elértük, hogy az i-edik sorban csak az első i pozícióban van nem-0 elem ($1 \le i \le t$). Szemeljük ki az utolsó $n-t$ oszlop egy nem-0 elemét

(ha ilyen nincs, megállunk). Rendezzük át a sorokat és oszlopokat úgy, hogy ez az elem a $(t+1, t+1)$ pozícióba kerüljön. Vonjuk ki a $(t+1)$-edik oszlop $(a_{t+1,i}/a_{t+1,t+1})$-szeresét az i-edik oszlopból $(i = t+1, \ldots, n)$.

Mivel a Gauss-elimináció egy iterációja $O(n^2)$ aritmetikai művelettel jár, és n iterációt kell végrehajtani, ez $O(n^3)$ aritmetikai műveletet jelent. Probléma azonban az, hogy osztanunk is kell, és pedig nem maradékosan. Véges test fölött ez nem jelent problémát, de a racionális test esetén igen. Azt föltettük, hogy az eredeti A mátrix elemei egészek; de az algoritmus menetközben racionális számokból álló mátrixokat produkál. Milyen alakban tároljuk ezeket a mátrixelemeket? A természetes válasz az, hogy egy egész számokból álló számpárokként (ezek hányadosa a racionális szám).

De megköveteljük-e, hogy a törtek egyszerűsített alakban legyenek, vagyis a számlálójuk és nevezőjük relatív prím legyen? Ezt megtehetjük, de akkor minden mátrixelemet minden iteráció után egyszerűsítenünk kell, amihez egy euklideszi algoritmust kellene végrehajtani. Ez ugyan polinomiális időben elvégezhető, de igen sok többletmunka, jó volna elkerülni. (Természetesen be kell még látnunk, hogy egyszerűsített alakban a szereplő számlálóknak és nevezőknek csak polinomiálisan sok jegye lesz.)

Megtehetnénk azt is, hogy nem kötjük ki, hogy a mátrixelemek egyszerűsített alakban legyenek. Ekkor két racionális szám, a/b és c/d összegét ill. szorzatát egyszerűen a következő formulával definiáljuk: $(ad+bc)/(bd)$, $(ac)/(bd)$. Ezzel a megállapodással az a probléma, hogy a menet közben előálló számlálók és nevezők igen nagyok (nem polinomiális jegyszámúak) lehetnek!

Feladat. 3. Legyen A az az $(n \times n)$-es mátrix, melynek főátlójában 2-esek, alatta 1-esek, fölötte 0-k állnak. Mutassuk meg, hogy ha a Gauss-eliminációt egyszerűsítés nélkül alkalmazzuk erre a mátrixra, akkor k iteráció után a nevezők mindegyike 2^{2^k-1} lesz.

Meg tudunk azonban olyan eljárást is adni, mely a törteket részben egyszerűsített alakban tárolja, és így mind az egyszerűsítést, mind a jegyek számának túlzott növekedését elkerüli. Evégett elemezzük kissé a Gauss-elimináció közben előálló mátrixokat. Feltehetjük, hogy a kiszemelt elemek rendre a $(1,1), \ldots, (n,n)$ pozícióban vannak, vagyis nem kell sorokat, oszlopokat permutálnunk. Jelölje $(a_{ij}^{(k)}$ $(1 \leq i, j \leq n)$ a k iteráció után kapott mátrixot. Egyszerűség kedvéért a végül kapott mátrix főátlójában levő elemek legyenek $d_1, \ldots, d_n$ (így $d_i = a_{ii}^n$). Jelölje $D^{(k)}$ az A mátrix első k sora és oszlopa által meghatározott részmátrixot, és jelölje $D_{ij}^{(k)}$ $(k+1 \leq i, j \leq n)$ az első k és az i-edik sor, valamint az első k és a j-edik oszlop által meghatározott részmátrixot. Nyilván $D^{(k)} = D_{kk}^{(k-1)}$.

3.1.3 Lemma.

$$a_{ij}^{(k)} = \frac{\det(D_{ij}^{(k)})}{\det(D^{(k)})}.$$

Bizonyítás: Ha $\det(D_{ij}^{(k)})$-t számítjuk ki Gauss-eliminációval, akkor a főátlójában rendre a $d_1, \ldots, d_k, a_{ij}^{(k)}$ elemeket kapjuk. Így tehát

$$\det D_{ij}^{(k)} = d_1 \cdot \ldots \cdot d_k \cdot a_{ij}^{(k)}.$$

Hasonló meggondolással adódik, hogy

$$\det D^{(k)} = d_1 \cdot \ldots \cdot d_k.$$

E két egyenletet egymással elosztva a lemma adódik. □

E lemma szerint a Gauss-eliminációban fellépő minden tört nevezője és számlálója (alkalmas egyszerűsítés után) az eredeti A mátrix egy-egy részmátrixának a determinánsa. Így a menetközben kapott törtek leírásához (egyszerűsítés után) biztosan elegendő polinomiálisan sok jegy.

De nem szükséges menet közben törtek egyszerűsítéseit számolni. A Gauss-elimináció alapján fennáll, hogy $i, j > k$ esetén

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{i,k+1}^{(k)} a_{k+1,j}^{(k)}}{a_{k+1,k+1}^{(k)}},$$

és ebből

$$\det(D_{ij}^{(k+1)}) = \frac{\det(D_{ij}^{(k)}) \det(D_{k+1,k+1}^{(k)}) - \det(D_{i,k+1}^{(k)}) \det(D_{k+1,j}^{(k)})}{\det(D_{k,k}^{(k-1)})}.$$

Ez a formula úgy tekinthető, mint egy rekurzió a $\det(D^(k)_{ij})$ számok kiszámítására. Mivel a baloldalon egész szám áll, ezért az osztást el tudjuk végezni. A fenti meggondolások szerint a hányados jegyeinek száma polinomiális a bemenet méretéhez képest.

Megjegyzések. 1. A Gauss-eliminációban föllépő törtekkel kapcsolatos probléma orvoslására két további lehetőséget is érdemes megemlíteni. Megtehetjük (sőt a számítógépes realizáció szempontjából ez a természetes megoldás), hogy a számokat korlátos pontosságú „kettedes törtekkel" közelítjük; mondjuk p „kettedes jegyet" engedve meg a „kettedes vessző" után. Ekkor az eredmény csak közelítő, de mivel előre tudjuk, hogy a determináns egész szám, ezért elegendő azt $1/2$-nél kisebb hibával meghatározni. A numerikus analízis eszközeivel meghatározható, hogy mekkorának kell p-t választani ahhoz, hogy a végeredményben levő hiba kisebb legyen, mint $1/2$. Kiderül, hogy $O(n\langle A\rangle)$ jegy elegendő, és ezért ez az eljárás is polinomiális algoritmushoz vezet.

A harmadik lehetőség azon az észrevételen alapszik, hogy ha $m > 2 \cdot |\det(A)|$, akkor elegendő $\det(A)$ értékét modulo m meghatározni. Mivel tudjuk, hogy $|\det(A)| < 2^{\langle A\rangle}$, elegendő m gyanánt olyan prímszámot választani, mely nagyobb, mint $2^{1+\langle A\rangle}$. Ez azonban nem könnyű (v.ö. a „Randomizált algoritmusok" című fejezettel), ezért ehelyett m-et különböző kis prímek szorzatának választhatjuk: $m = 2 \cdot 3 \cdot \ldots \cdot p_k$, ahol $k = 1 + \langle A\rangle$ megfelel. Ekkor $\det(A)$ maradékát modulo p_i minden i-re könnyen kiszámíthatjuk a maradékosztálytestben elvégzett Gauss-eliminációval, majd $\det(A)$ maradékát modulo m a Kínai Maradéktétellel számíthatjuk ki.

Ez az utóbbi módszer számos más esetben is jól alkalmazható. Úgy is tekinthetjük, mint az egész számoknak egy, a kettes (vagy tízes) számrendszertől különböző kódolását: az n természetes számot a 2, 3, stb. prímekkel vett osztási maradékával kódoljuk (ez ugyan végtelen sok bit, de ha előre tudjuk, hogy a számolás közben nem fordul elő N-nél nagyobb természetes szám, akkor elegendő az első k prímet tekinteni, melyek szorzata nagyobb, mint

N). Ebben a kódolásban az aritmetikai alapműveletek igen egyszerűen (párhuzamosan) elvégezhetők, de a nagyság szerinti összehasonlítás nehézkes.

2. A polinomiális idejű algoritmusok különösen fontos szerepét magyarázza az is, hogy megadhatók olyan természetes szintaktai feltételek, melyek egy algoritmus polinomiális voltával ekvivalensek. Polinomiálisak az olyan programok pl. Pascal-ban, melyekben nincsen „goto", „repeat" és „while" utasítás, és a „for" utasítások felső határa a bemenet mérete (megfordítva is igaz: minden polinomiális algoritmus programozható így).

3.2. Egyéb bonyolultsági osztályok

a) Exponenciális idő. Az „összes eset" végignézése gyakran vezet exponenciális idejű algoritmusokhoz (vagyis olyan algoritmusokhoz, melyeknek időigénye 2^{n^a} és 2^{n^b} közé esik, ahol $a, b > 0$ konstansok). Például ha meg akarjuk határozni, hogy egy G gráf csúcsai kiszínezhetők-e 3 színnel (úgy, hogy szomszédos csúcsok különböző színűek legyenek), akkor a triviális algoritmus az, hogy végignézzük az összes színezéseket. Ez 3^n eset végignézését jelenti, ahol n a gráf csúcsainak a száma; egy eset magában $O(n^2)$ időt igényel. (Sajnos erre a feladatra lényegesen jobb — nem exponenciális idejű — algoritmus nem ismeretes, sőt bizonyos értelemben nem is várható, mint azt látni fogjuk a következő fejezetben. Azt is látni fogjuk azonban, hogy a gráfszínezés az exponenciális idejű problémák egy igen speciális osztályába esik.)

b) Lineáris idő. Több alapvető aritmetikai algoritmus (két szám összeadása, összehasonlítása) lineáris idejű.

A lineáris idejű algoritmusok főleg ott fontosak, ahol nagyméretű bemeneteken viszonylag egyszerű feladatokat kell elvégezni. így számos adatkezelési algoritmus lineáris idejű. Fontos lineáris idejű gráfalgoritmus a mélységi keresés. Ennek segítségével több nem-triviális gráfelméleti probléma (pl. síkbarajzolás) is megoldható lineáris időben.

Kvázilineáris idejűnek nevezik az olyan algoritmust, melynek időigénye $O(n(\log n)^c)$, ahol c konstans. A legfontosabb kvázilineáris időben megoldható probléma a sorbarendezés, melyre több $O(n \log n)$ idejű algoritmus is ismert. Fontos kvázilineáris algoritmusokat találhatunk a geometriai algoritmusok és a képfeldologozás területén (pl. egy n elemű síkbeli ponthalmaz konvex burka ugyancsak $O(n \log n)$ lépésben határozható meg).

c) Polinomiális tár. Nyilvánvalóan polinomiális tárigényű minden polinomiális idejű algoritmus, de a polinomiális tár lényegesen általánosabb. Az a) pontban tárgyalt triviális gráfszínezési algoritmus tárigénye csak polinomiális (sőt lineáris): ha a színezéseket lexikografikus sorrendben nézzük végig, akkor elegendő azt nyilvántartani, hogy melyik színezést vizsgáljuk, és azt, hogy mely élekről ellenőriztük már, hogy nem kötnek-e össze azonos színű pontokat.

Polinomiális tárigényű algoritmusra a legtipikusabb példa egy játékban az optimális lépés meghatározása az összes lehetséges folytatás végigvizsgálásával. Feltesszük, hogy a játék bármely adott helyzete egy n hosszúságú x szóval írható le (tipikusan a figurák helyzetével a táblán, beleértve, hogy ki lép, és esetleg egyéb információt, pl. a sakknál azt, hogy lépett-e már a király stb.). Ki van tüntetve egy kezdő helyzet. Azt is feltesszük, hogy a két játékos felváltva lép, és van arra algoritmusunk, mely két adott helyzetre megmondja,

hogy az elsőből legális-e a másodikba lépni, mondjuk polinomiális időben (elegendő volna azt föltenni, hogy polinomiális tárral, de meglehetősen unalmas az olyan játék, melyben annak eldöntése, hogy egy lépés legális-e, több, mint polinomiális időt igényel). Ha egy helyzetben nincs legális lépés, akkor eldönthető polinomiális időben, hogy ki nyert (a döntetlentől egyszerűség kedvéért eltekintünk). Végül föltesszük, hogy a játék legfeljebb n^c lépésben mindig véget ér.

Ekkor az összes lehetséges lejátszások végignézése, és minden helyzetre annak meghatározása, hogy ki nyer, csak polinomiális tárat (bár általában exponenciális időt) vesz igénybe. Tegyük föl, hogy egy adott x_0 helyzetről akarjuk eldönteni, hogy nyerő vagy vesztő helyzet-e (a lépésen következő szempontjából). Helyzeteknek egy $x_0, x_1, ..., x_k$ sorozatát *részjátéknak* nevezzük, ha minden x_i helyzetből x_{i+1}-be legális lépés vezet. A gép egy adott pillanatban egy részjáték lehetséges folytatásait elemzi. Fenn fog mindig állni, hogy $x_0, x_1, ..., x_i$ $(0 \leq i < k)$ lehetséges folytatásai közül azok, melyek v_{i+1}-nél kisebbek (az n hosszúságú szavak lexikografikus rendezésére nézve), „rossz lépések", azaz az $x_0, x_1, .., x_i$ után lépésen lévő játékos veszít, ha ide lép (vagy a lépés illegális).

Az algoritmus végignézi az összes n hosszúságú szót lexikografikus sorrendben, hogy legális folytatásai-e x_k-nak. Ha ilyet talál, az lesz x_{k+1}, és az így kapott eggyel hosszabb részjátékot vizsgáljuk. Ha nem talál ilyet, akkor a vizsgált részjáték bizonyos helyzeteit „nyerő helyzetnek" jelöli meg (a lépésen levő játékos számára), az alábbiak szerint: eldönti, hogy x_k-ban ki nyer. Ha a lépésen levő játékos nyer, akkor x_k nyerő helyzet; ha a másik, akkor $x_k - 1$ nyerő helyzet. Legyen i az a legkisebb index, melyre már tudjuk, hogy nyerő helyzet. Ha $i = 0$, akkor tudjuk, hogy a játékban a kezdő nyer. Ha $t > 1$, akkor x_{i-1}-ből ide lépni rossz lépés volt. Megnézi ezért az algoritmus, hogy lehet-e x_{i-1}-ből legálisan olyan helyzetbe lépni, mely x_i-nél lexikografikusan nagyobb. Ha igen, legyen y az első ilyen; az algoritmus az $x_0, x_1, ..., x_{i-1}, y$ részjáték vizsgálatával folytatódik. Ha semelyik x_i-nél lexikografikusan nagyobb helyzet sem legális lépés x_{i-1}-ből, akkor x_{i-1}-ből minden legális lépés „rossz". Így ha $i = 1$, akkor a kezdőállapotban a kezdő veszít, és készen vagyunk. Ha $i > 2$, akkor x_{i-2}-t nyerő helyzetként jelölhetjük meg.

Ugyancsak polinomiális tárral, de exponenciális idővel működik a „triviális összeszámlálás" a legtöbb leszámlálási feladat esetén. Például adott G gráfhoz meg akarjuk határozni a G három színnel való színezéseinek a számát. Ekkor végignézhetjük az összes színezéseket, és valahányszor egy színezés legális, 1-et adunk egy számlálóhoz.

d) Lineáris tár. A tárigény szempontjából a polinomiális időhöz hasonlóan alapvető osztály. Ilyenek a gráfalgoritmusok közül azok, melyek leírhatók a pontokhoz és élekhez rendelt címkék változtatásával: pl. összefüggőség, magyar módszer, legrövidebb út, optimális folyam keresése. Ilyen a korlátos pontosságú numerikus algoritmusok többsége.

Példaként írjuk le Turing-gépen a *szélességi keresést.* Ennek az algoritmusnak bemenete egy (irányítatlan) G gráf és egy $v \in V(G)$ csúcs. Kimenete G-nek egy olyan F feszítőfája, melyre minden x pontra az x-et v-vel összekötő F-beli út az összes G-beli utak közül a legrövidebb. Feltesszük, hogy a G gráf úgy van megadva, hogy rendre minden csúcsához meg van adva a csúcs szomszédainak listája. Az algoritmus során minden x ponthoz minden lépésben vagy a „címkézett" vagy az „átvizsgált" címke van hozzárendelve. Kezdetben nincs átvizsgált pont, és csak v címkézett. Menet közben a címkézett, de nem átvizsgált pontok neveit egy külön szalagon, a „sor" szalagon tároljuk, egy másik szalagon (az „F" szalagon) pedig a címkézett pontokat, mindegyik után zárójelben fölírva azt is, hogy melyik pontból

jutottunk el hozzá. A gép a címkézett, de nem átvizsgált pontok közül elolvassa az első pont nevét (legyen ez u), majd ennek szomszédai közül keres olyat, amely még nincsen címkézve. Ha talál, akkor ennek a nevét fölírja a „sor" szalag és az „F" szalag végére, az utóbbin utána írva zárójelben, hogy „u". Ha végigvizsgálta u összes szomszédját, akkor u-t letörli a „sor" szalag elejéről. Az algoritmus megáll, ha a „sor" szalag üres. Ekkor az „F" szalagon szereplő párok a keresett F fa éleit adják.

Nyilvánvaló, hogy ennek az algoritmusnak a tárigénye csak $O(n)$ darab log n jegyű szám, ennyi tár (konstans tényezőtől eltekintve) pedig már a csúcsok neveinek leírásához is kell.

3.3. Általános tételek a tár- és időbonyolultságról

Ha egy L nyelvhez van olyan L-et eldöntő Turing-gép, melyre minden elég nagy n-re $\text{time}_T(n) \leq f(n)$, (ahol $f(n) \geq n$ minden n-re), akkor olyan L-et felismerő Turing-gép is van, melyre ez az egyenlőtlenség minden n-re fönnáll. Kis n-ekre ugyanis a nyelv eldöntését a vezérlőegységre bízhatjuk.

Várható, hogy a gép további bonyolítása árán az időigényeket csökkenteni lehet. A következő tétel mutatja, hogy valóban tetszőleges konstans faktorral lehet a gépet gyorsítani, legalábbis, ha az időigénye elég nagy (a beolvasásra fordított időt nem lehet „megtakarítani").

3.3.1 Lineáris Gyorsítási Tétel. *Minden T Turing-géphez és $c > 0$-hoz található olyan S Turing-gép, mely ugyanazt a nyelvet dönti el, és melyre $\text{time}_S(n) \leq c \cdot \text{time}_T(n) + n$.*

Bizonyítás: Egyszerűség kedvéért tegyük föl, hogy T-nek egy munkaszalagja van (a bizonyítás hasonlóan szólna k szalagra is). Feltehetjük, hogy $c = 1/p$, ahol p egész szám.

Legyen az S Turing-gépnek egy bemenet-szalagja. Ezenkívül vegyünk föl $2p-1$ „induló" szalagot és $2p-1$ munkaszalagot. Számozzuk ezeket külön-külön $(1-p)$-től $(p-1)$-ig. Az i sorszámú (induló vagy munka-) szalag j-edik mezejének *indexe* legyen a $j(2p-1)+i$ szám. A t indexű induló ill. munkamező majd a T gép bemenet- ill. munkaszalagján a t-edik mezőnak fog megfelelni. Legyen még S-nek egy kimenet-szalagja is.

Az S gép működését azzal kezdi, hogy bemenet-szalagjáról az x bemenet minden betűjét átmásolja az induló szalagok megfelelő indexű mezejére, majd minden fejet visszaállít a 0-dik mezőre. Ettől kezdve az „igazi" bemenet-szalaggal nem törődik.

Az S gép minden további lépése a T gép p egymásutáni lépésének fog megfelelni. A T gép pk lépése után a bemenet-szalag és a munkaszalag olvasó-feje álljon a t-edik ill. s-edik mezőn. Úgy fogjuk tervezni az S gépet, hogy ilyenkor az S gép induló ill. munkaszalagjainak minden mezején ugyanaz a jel álljon, mint a T gép megfelelő szalagjának megfelelő mezején, a fejek pedig a T-beli $t-p+1, \ldots, t+p-1$ indexű induló ill. $s-p+1, \ldots, s+p-1$ indexű munkamezőkön állnak. Feltesszük, hogy az S gép vezérlőegysége azt is „tudja", hogy melyik fej áll a t-nek ill. s-nek megfelelő mezőn. Tudja továbbá, hogy mi a T vezérlőegységének belső állapota.

Mivel S vezérlőegysége nemcsak azt látja, hogy T vezérlőegysége ebben a pillanatban mit olvas az induló és munkaszalagján, hanem az ettől legfeljebb $p-1$ távolságra levő mezőket is, ki tudja számítani, hogy a következő p lépés során T fejei merre lépnek és mit írnak. Mondjuk p lépés után T fejei a $t+i$, ill. $s+j$ pozícióban lesznek (ahol

mondjuk $i, j \geq 0$). Nyilván $i, j \leq p$. Vegyük észre, hogy közben a „munkafej" csak az $[s-p+1, s+p-1]$ intervallumban változtathatta meg a munkaszalagra írt jeleket.

Ezekután S vezérlőegysége a következőket tegye: számítsa ki és jegyezze meg, hogy mi lesz T vezérlőegységének belső állapota p lépéssel később. Jegyezze meg, hogy melyik fej áll a $(t+i)$ ill. $(s+j)$ pozíciónak megfelelő mezőn. A munkaszalagokon írassa át a jeleket a p lépéssel későbbi helyzetnek megfelelően (ez lehetséges, mert a minden, az $[s-p+1, s+p-1]$ intervallumba eső indexű munkamezőn áll fej). Végül a $[t-p+1, t+i-p]$ intervallumba eső indexű induló fejeket és a $[s-p+1, s+j-p]$ intervallumba eső indexű munkafejeket léptesse egy hellyel jobbra; így az azok által elfoglalt mezők indexei a $[t+p, t+i+p-1]$ ill. $[s+p, s+j+p-1]$ intervallumot fogják kitölteni, ami a helyben maradó fejekkel együtt kiadja a $[t+i-p+1, t+i+p-1]$ ill. $[s+j-p+1, s+j+p-1]$ intervallumot.

Ha T a tekintett p lépés alatt a kimenet-szalagra ír (0-t vagy 1-et) és leáll, akkor tegye S is ezt. Ezzel megkonstruáltuk az S gépet, mely (a kiindulási másolástól eltekintve) p-edannyit lép, mint T, és nyilván ugyanazt a nyelvet dönti el. □

Megjegyzések. 1. Ha nem engedjük meg a szalagok számának növelését, de ehelyett az ábécéhez vehetünk hozzá új jeleket, akkor hasonló tétel lesz igaz. Ez a fentihez hasonló konstrukcióval látható be; lásd [HU].

2. A tárra vonatkozóan hasonló tétel csak akkor ismert, ha megengedjük az ábécé bővítését. Lásd [HU].

Triviális, hogy egy k-szalagos Turing-gép tárigénye nem nagyobb, mint időigényének k-szorosa (hiszen egy lépésben legfeljebb k mezőre történik írás). Tehát ha egy nyelvre $\mathcal{L} \in \mathrm{DTIME}(f(n))$, akkor van olyan (a nyelvtől függő) k konstans, hogy $\mathcal{L} \in \mathrm{DSPACE}(k \cdot f(n))$. (Ha megengedjük az ábécé bővítését, akkor $\mathrm{DSPACE}(k \cdot f(n)) = \mathrm{DSPACE}(f(n))$, és így következik, hogy $\mathrm{DTIME}(f(n)) \subseteq \mathrm{DSPACE}(f(n))$. Másrészt az időigény nem nagyobb, mint a tárigény egy exponenciális függvénye (mivel nem állhat elő pontosan ugyanaz a memória-helyzet, a fejek helyzetét és a vezérlőegység állapotát is figyelembe véve, egynél többször ciklizálás nélkül). Pontosabban számolva, a különböző memória-helyzetek száma legfeljebb $c \cdot f(n)^k m^{f(n)}$.

Mivel a fentiek szerint egy nyelv időbonyolultsága konstans szorzótól nem függ, és itt a felső korlátban c, k és m konstansok, következik, hogy ha $f(n) > \log n$ és $L \in \mathrm{DSPACE}(f(n))$ akkor $L \in \mathrm{DTIME}((m+1)^{f(n)})$. Tehát $\mathrm{DSPACE}(f(n)) \subseteq \bigcup_{c=1}^{\infty} \mathrm{DTIME}(c^{f(n)})$.

3.3.2 Tétel: *Van olyan $f(n)$ függvény, hogy minden rekurzív nyelv benne van a* $\mathrm{DTIME}(f(n))$ *osztályban.*

Bizonyítás: Összesen csak megszámlálható sok Turing-gép van, így persze az olyan Turing-gépek halmaza is megszámlálható, melyek minden bemenetre megállnak. Legyen ezek egy felsorolása $T_1, T_2, \ldots$. Definiáljuk f-et a következőképpen:

$$f(n) := \max_{i \leq n} \mathrm{time}_{T_i}(n).$$

Ha $\mathcal{L}$ rekurzív, akkor van olyan j, hogy T_j eldönti $\mathcal{L}$-et, $n \geq j$ esetén $\mathrm{time}_{T_j}(n) \leq f(n)$ időben. De ekkor (az alfejezet első megjegyzése szerint) van olyan T Turing-gép is, mely

minden n-re legfeljebb $f(n)$ időben dönti el $\mathcal{L}$-et. □

Egy rekurzív nyelv idő-bonyolultsága (és ezért, a fentiek miatt, a tár-bonyolultsága is) akármilyen nagy lehet. Pontosabban:

3.3.3 Tétel: *Minden rekurzív $f(n)$ függvényhez van olyan rekurzív $\mathcal{L}$ nyelv, mely nem eleme* DTIME($f(n)$)*-nek.*

Megjegyzés: Ebből persze rögtön következik, hogy az előző tételben konstruált f függvény nem rekurzív.

Bizonyítás: A bizonyítás hasonló annak bizonyításához, hogy a megállási probléma eldönthetetlen. Feltehetjük, hogy $f(n) > n$. Legyen T egy 2 szalagos univerzális Turing-gép és álljon $\mathcal{L}$ mindazokból az x szavakból, melyekre igaz az, hogy T két szalagjára x-et írva, T legfeljebb $f(|x|)^3$ lépésben megáll. Nyilvánvaló, hogy $\mathcal{L}$ rekurzív.

Tegyük föl mármost, hogy $\mathcal{L} \in \mathrm{DTIME}(f(n))$. Ekkor van olyan Turing-gép (valahány, mondjuk k szalaggal), mely $\mathcal{L}$-et $f(n)$ időben eldönti. Ebből a korábban említett módon csinálhatunk olyan 1 szalagos Turing-gépet, mely $\mathcal{L}$-et $cf(n)^2$ időben eldönti. Módosítsuk a gépet úgy, hogy ha egy x szó $\mathcal{L}$-ben van, akkor működjön a végtelenségig, míg ha $x \in \Sigma_0^* - \mathcal{L}$, akkor álljon meg. Legyen ez a gép S. S szimulálható T-n valamely p_S (csak S-től függő konstans hosszú) programmal úgy, hogy T az (x, p_S) bemenettel akkor és csak akkor áll meg, ha S az x bemenettel megáll, és ilyenkor $c_S \cdot c \cdot f(|x|)^2$ lépésen belül áll meg. Nyilván van olyan n_0 szám, hogy $n \geq n_0$ esetén $c_S \cdot c \cdot f(n)^2 < f(n)^3$. A p_S program végére egy kis elkülöníthető szemetet írva, kapunk egy p programot, mellyel T szintén szimulálja S-et (ugyanannyi időben, mivel a szemetet sose olvassa el), de $|p| > n_0$.

Mármost 2 eset lehetséges. Ha $p \in \mathcal{L}$, akkor – $\mathcal{L}$ definíciója miatt – a T gép mindkét szalagjára p-t írva, az $f(|p|)^3$ időn belül megáll. Mivel a p program S-et szimulálja, következik, hogy S a p bemenettel $f(|p|)^3$ időn belül megáll. Ez azonban lehetetlen, mert S $\mathcal{L}$-beli bemenetre egyáltalában nem áll meg.

Másrészt ha $p \notin \mathcal{L}$, akkor – S konstrukciója miatt – az S gép első szalagjára p-t írva, az $cf(|p|)^2$ idő alatt megáll. Így T is megáll $f(|p|)^3$ idő alatt. De akkor $p \in \mathcal{L}$ az $\mathcal{L}$ nyelv definíciója miatt.

Ez az ellentmondás mutatja, hogy $\mathcal{L} \notin \mathrm{DTIME}(f(n))$. □

Egy $f : \mathbb{Z}_+ \to \mathbb{Z}_+$ függvényt *teljesen időkonstruálhatónak* nevezünk, ha van olyan T Turing-gép, mely minden n hosszú bemeneten pontosan $f(n)$ lépést végez. Ennek a furcsa definíciónak értelme az, hogy teljesen időkonstruálható függvényekkel könnyen lehet Turing-gépek lépésszámát korlátozni: Ha van egy Turing-gépünk, mely minden n hosszúságú bemeneten pontosan $f(n)$ lépést végez, akkor ezt beépíthetjük bármely más Turing-gépbe, mint órát: szalagjaik a bemenet-szalag kivételével különbözőek, és az egyesített Turing-gép minden lépésben mindkét gép működését végrehajtja.

Nyilvánvaló, hogy minden teljesen időkonstruálható függvény rekurzív. Másrészt könnyen látható, hogy $n^2, 2^n, n!$ és minden „ésszerű” függvény teljesen időkonstruálható. Az alábbi lemma általában is garantálja sok teljesen időkonstruálható függvények létezését.

Nevezzük az $f : \mathbb{Z}_+ \to \mathbb{Z}_+$ függvényt *jól számolhatónak*, ha van olyan Turing-gép, mely $f(n)$-et az n bemeneten $O(f(n))$ idő alatt kiszámítja. Ezekután könnyen bizonyítható az

alábbi lemma:

3.3.4 Lemma: (a) *Minden jól számolható $f(n)$ függvényhez van olyan teljesen időkonstruálható $g(n)$ függvény és c konstans, melyre $f(n) \leq g(n) \leq c \cdot f(n)$.*

(b) *Minden teljesen időkonstruálható $g(n)$ függvényhez van olyan jól számolható $f(n)$ függvényés c konstans, melyre $g(n) \leq f(n) \leq c \cdot g(n)$.*

(c) *Minden rekurzív f függvényhez van olyan teljesen időkonstruálható g függvény, melyre $f \leq g$.* □

E lemma alapján a legtöbb esetben egyformán használhatjuk a teljesen időkonstruálható és a jól számolható függvényeket. A szokást követve az előbbit használjuk.

A 3.3.3 Tétel bizonyításának további finomításával (felhasználva az 1.2 alfejezet 13. feladatának állítását) igazolható a következő:

3.3.5 Idő-hierarchia Tétel: *ha $f(n)$ teljesen időkonstruálható és $g(n)(\log g(n)) = o(f(n))$, akkor van olyan nyelv* DTIME$(f(n))$*-ben mely nem tartozik* DTIME$(g(n))$*-be.* □

Azt mondja ez a tétel, hogy „több” időben többet tudunk kiszámolni, ha a „több” fogalmát jól definiáljuk. Megjegyezzük, hogy az analóg Tár-hierarchia tétel még élesebben igaz, elég a $g(n) = o(f(n))$ feltevés.

Az $f(n)$ függvény teljesen időkonstruálható volta igen fontos szerepet játszik az előző tételben. Ha ezt elejtjük, akkor tetszőlegesen nagy „hézag” lehet $f(n)$ „alatt”, amibe nem esik semmilyen nyelv idő-bonyolultsága:

3.3.6 Hézag Tétel: *Minden rekurzív $\phi(n) \geq n$ függvényhez van olyan rekurzív $f(n)$ függvény, hogy*

$$\text{DTIME}(\phi(f(n))) = \text{DTIME}(f(n)).$$

Van tehát olyan rekurzív f függvény, melyre

$$\text{DTIME}(f(n)^2) = \text{DTIME}(f(n)),$$

sőt olyan is, melyre

$$\text{DTIME}(2^{2^{f(n)}}) = \text{DTIME}(f(n)).$$

Érdemes megjegyezni, hogy ezek szerint olyan rekurzív f függvény is van, melyre

$$\text{DTIME}(f(n)) = \text{DSPACE}(f(n)).$$

Bizonyítás: Rögzítsünk egy 2 szalagos univerzális Turing-gépet. Jelölje $\tau(x, y)$ azt az időt, amennyit T az első szalagra x-et, a másodikra y-t írva számol. (Ez lehet végtelen is.) A bizonyítás lelke az alábbi konstrukció.

Állítás: Van olyan h rekurzív függvény, hogy minden $n > 0$-ra és minden $x, y \in \Sigma_0^*$-ra, ha $|x|, |y| \leq n$ akkor vagy $\tau(x, y) \leq h(n)$ vagy $\tau(x, y) \geq (\phi(h(n)))^3$.

Ha a

$$\psi(n) = \max\{\tau(x,y) : |x|, |y| \leq n, \tau(x,y) \text{ véges}\}$$

függvény rekurzív volna, akkor ez triviálisan megfelelne a feltételeknek. Ez a függvény azonban nem rekurzív (feladat: bizonyítsuk be!). Ezért a következő „konstruktív változatát" vezetjük be: adott n-re induljunk ki a $t = n + 1$ időkorlátból. Rendezzük az összes $(x, y) \in \Sigma_0^*$, $|x|, |y| \leq n$ párt egy sorba. Vegyük a sor első (x, y) elemét, és futtassuk a gépet ezzel a bemenettel. Ha t időn belül megáll, akkor az (x, y) párt dobjuk ki. Ha s lépésben megáll, ahol $t < s \leq \phi(t)^3$, akkor legyen $t := s$ és az (x, y) párt megint csak dobjuk ki. (Itt kihasználtuk, hogy $\phi(n)$ rekurzív.) Ha a gép még $\phi(t)$ lépés után sem állt le, akkor állítsuk meg és tegyük az (x, y) párt a sor végére. Ha a soron egyszer végigmentünk anélkül, hogy egyetlen párt is kidobtunk volna, akkor álljunk meg, és legyen $h(n) := t$. Ez a függvény nyilván rendelkezik az Állításban megfogalmazott tulajdonsággal.

Megmutatjuk, hogy az így definiált $h(n)$ függvényre

$$\text{DTIME}(h(n)) = \text{DTIME}(\phi(h(n)).$$

Tekintsünk evégett egy tetszőleges $\mathcal{L} \in \text{DTIME}(\phi(h(n)))$ nyelvet (a másik irányú tartalmazás triviális). Megadható ehhez tehát egy olyan Turing-gép, mely $\mathcal{L}$-et $\phi(h(n))$ időben eldönti. Megadható ezért olyan egy szalagos Turing-gép, mely $\mathcal{L}$-et $\phi(h(n))^2$ időben eldönti. Ez az utóbbi Turing gép szimulálható az adott T univerzális gépen, ha annak második szalagjára valamilyen p programot írunk, $|p| \cdot \phi(h(n))^2$ időben. Így ha n elég nagy, akkor T minden $(y, p)(|y| \leq n)$ alakú bemeneten legfeljebb $\phi(h(n))^3$ ideig dolgozik. Ekkor azonban $h(n)$ definíciója miatt minden ilyen bemeneten legfeljebb $h(n)$ ideig dolgozik. Tehát ez a gép az adott programmal (amit beletehetünk a vezérlőegységbe, ha akarjuk) $h(n)$ idő alatt eldönti az L nyelvet, vagyis $L \in \text{DTIME}(h(n))$. □

A tétel következményeként látjuk, hogy van olyan rekurzív $f(n)$ függvény, hogy

$$\text{DTIME}((m+1)^{f(n)}) = \text{DTIME}(f(n)),$$

és így

$$\text{DTIME}(f(n)) = \text{DSPACE}(f(n)).$$

Egy adott feladatra általában nincsen „legjobb" algoritmus, sőt a következő meglepő tétel igaz.

3.3.7 Gyorsítási Tétel: *Bármely rekurzív $g(n)$ függvényhez létezik olyan rekurzív $\mathcal{L}$ nyelv, hogy minden $\mathcal{L}$-et eldöntő T Turing-géphez létezik olyan $\mathcal{L}$-et eldöntő S Turing-gép, melyre $g(\text{time}_S(n)) < \text{time}_T(n)$.*

A Lineáris Gyorsítási Tétel minden nyelvre vonatkozott; ez a tétel csak a létezését mondja ki tetszőlegesen „gyorsítható" nyelvnek. Általában egy tetszőleges nyelvre lineárisnál jobb gyorsítás nem várható.

Bizonyítás: A bizonyítás lényege az, hogy minnél bonyolultabb gépeket engedünk meg, annál több információt „beégethetünk" a vezérlőegységbe. Így a gépnek csak a hosszabb

bemenetekkel kell „érdemben” foglalkoznia, és a nyelvet úgy akarjuk megcsinálni, hogy ezeken egyre könnyebb legyen. Nem lesz elég azonban csak a „rövid” szavak $\mathcal{L}$-hez tartozását vagy nem tartozását beégetni, hanem ezekről több információra is szükség lesz.

Az általánosság megszorítása nélkül feltehetjük, hogy $g(n) > n$, és hogy g teljesen időkonstruálható függvény. Definiáljunk egy h függvényt a

$$h(0) = 1,\ h(n) = (g(h(n-1)))^3$$

rekurzióval. Könnyen látható, hogy $h(n)$ monoton növekvő (sőt nagyon gyorsan növekvő), teljesen időkonstruálható függvény. Rögzítsünk egy univerzális T_0 Turing-gépet mondjuk két szalaggal. Jelölje $\tau(x, y)$ azt az időt, amit T_0 az (x, y) bemeneten dolgozik (ez lehet végtelen is). Nevezzük az (x, y) párt ($x, y \in \Sigma^*$) „gyorsnak”, ha $|y| \leq |x|$ és $\tau(x, y) \leq h(|x| - |y|)$.

Legyen $(x_1, x_2, \ldots)$ a szavak pl. lexikografikus sorbarendezése; kiválasztunk bizonyos i indexekre egy-egy y_i szót a következőképpen. Az $i = 1, 2, \ldots$ indexekre sorba nézzük meg, hogy van-e olyan y szó, melyre (x_i, y) gyors, és melyet még nem választottunk ki; ha van ilyen, legyen y_i egy legrövidebb. Álljon $\mathcal{L}$ mindazon x_i szavakból, melyekre y_i létezik, és a T_0 Turing-gép az (x_i, y_i) bemeneten azzal áll le, hogy első szalagján a „0” szó áll. (Vagyis melyeket T_0 az y_i programmal nem fogad el.)

Először is belátjuk, hogy $\mathcal{L}$ rekurzív, sőt minden k természetes számra $x \in \mathcal{L}$ eldönthető $h(n-k)$ lépésben (ahol $n = |x|$), ha n elég nagy. Az x_i szóról eldönthetjük azt, hogy benne van-e, ha eldöntjük, hogy y_i létezik-e, megtaláljuk y_i-t (ha létezik), és az (x_i, y_i) bemeneten futtatjuk a T_0 Turing-gépet $h(|x_i| - |y_i|)$ ideig.

Ez az utóbbi már magában túl sok, ha $|y_i| \leq k$; ezért azokról az (x_i, y_i) párokról, melyekre $|y_i| \leq k$, listát készítünk (ez rögzített k-re rögzített véges lista), és ezt elhelyezzük a vezérlőegységben. Ez tehát azzal kezdi, hogy az adott x szót megnézi, nincs-e ebben a listában egy pár első elemeként, és ha benne van, elfogadja (ez az x elolvasásán felül csak 1 lépés!).

Tegyük fel tehát, hogy x_i nincs a listában. Ekkor y_i, ha létezik, k-nál hosszabb. Kipróbálhatjuk az összes (x, y) bemenetet ($k < |y| \leq |x|$), hogy „gyors”-e, és ehhez csak $|\Sigma|^{2n+1} \cdot h(n-k-1)$ idő kell (beleértve $h(|x| - |y|)$ kiszámítását is). Ha ezt ismerjük, akkor y_i is megvan, és azt is látjuk, hogy T_0 elfogadja-e az (x_i, y_i) párt. A $h(n)$ függvény olyan gyorsan nő, hogy ez kisebb, mint $h(n-k)$.

Másodszor azt igazoljuk, hogy ha egy y program a T_0 gépen eldönti az $\mathcal{L}$ nyelvet (vagyis minden $x \in \Sigma^*$-ra megáll, és első szalagjára 1-et vagy 0-t ír aszerint, hogy x a $\mathcal{L}$ nyelvben van-e), akkor y nem lehet egyenlő egyik kiválasztott y_i szóval sem. Ez a szokásos „diagonalizálás” gondolatmenettel következik: ha $y_i = y$, akkor vizsgáljuk meg, hogy x_i a $\mathcal{L}$ nyelvben van-e. Ha igen, akkor T_0-nak az (x_i, y_i) párra „1”-et kell eredményül adnia (mert $y = y_i$ eldönti $\mathcal{L}$-et). De ekkor $\mathcal{L}$ definíciója szerint x_i-t nem tesszük bele. Megfordítva, ha $x_i \notin \mathcal{L}$, akkor azért maradt ki, mert T_0 az (x_i, y_i) bemenetre „1”-et ad válaszul; de ekkor $x_i \in \mathcal{L}$, mert az $y = y_i$ program eldönti $\mathcal{L}$-et. Mindkét esetben ellentmondást kapunk.

Harmadszor belátjuk, hogy ha az y program a T_0 gépen eldönti $\mathcal{L}$-et, akkor csak véges sok x szóra lehet (x, y) „gyors”. Legyen ugyanis (x, y) „gyors”, ahol $x = x_i$. Mivel az y_i választásakor y rendelkezésre állt (nem volt korábban kiválasztva), ezért erre az i-re kellett, hogy válasszunk y_i-t, és a ténylegesen kiválasztott y_i nem lehetett hosszabb, mint y. Így ha x különbözik az x_j ($|y_j| \leq |y|$) szavaktól, akkor (x, y) nem „gyors”.

Ezekután tekintsünk egy tetszőleges $\mathcal{L}$-et eldöntő T Turing-gépet. Ehhez csinálhatunk olyan egy szalagos T_1 Turing-gépet, mely ugyancsak $\mathcal{L}$-et dönti el, és melyre $\mathrm{time}_{T_1}(n) \leq (\mathrm{time}_T(n))^2$. Mivel T_0 univerzális gép, van olyan y program, mellyel T_0 a T_1-t szimulálja, úgy, hogy (legyünk bőkezűek) $\tau(x,y) \leq (\mathrm{time}_T(|x|))^3$ minden elég hosszú x szóra. A fent bizonyítottak szerint azonban véges sok x kivételével $\tau(x,y) \geq h(|x| - |y|)$, és így $\mathrm{time}_T(n) \geq (h(n - |y|))^{1/3}$.

Így a fent megkonstruált S Turing-gépre, mely L-et $h(n - |y| - 1)$ lépésben eldönti, fennáll

$$\mathrm{time}_T(n) \geq (h(n - |y|))^{1/3} \geq g(h(n - |y| - 1) \geq g(\mathrm{time}_S(n)).$$

□

4. fejezet

Nem-determinisztikus algoritmusok

Ha egy algoritmus megold egy problémát, akkor (implicite) arra is bizonyítékot szolgáltat, hogy a válasza helyes. Néha ez a bizonyíték sokkal egyszerűbb (rövidebb, áttekinthetőbb), mint az algoritmus figyelemmel kísérése volna. Például a magyar módszer az általa megtalált párosítás maximalitására a Kőnig-tételen keresztül szolgáltat egyszerű bizonyítékot: egy, a párosítással azonos elemszámú lefogó ponthalmazt (v.ö. a 4.3.3 tétellel).

Meg is fordíthatjuk ezt, és vizsgálhatjuk a bizonyítékot anélkül, hogy törődnénk vele, hogy hogyan lehet megtalálni. Ennek a szemléletnek több irányban van haszna. Egyrészt, ha már tudjuk, hogy a kívánt algoritmusnak milyen jellegű bizonyítékot kell szolgáltatnia, ez segíthet az algoritmus megalkotásában is. Másrészt, ha tudjuk, hogy egy feladat olyan, hogy a válaszra már a helyesség bizonyítéka sem adható meg pl. adott időn (vagy táron) belül, akkor algoritmus bonyolultságára is alsó becslést kapunk. Harmadszor, aszerint osztályozva a feladatokat, hogy mennyire könnyű a válaszra a helyességét rábizonyítani, igen érdekes és alapvető bonyolultsági osztályokat kapunk.

Ezzel a gondolattal, melyet *nem-determinizmusnak* neveznek, több fejezetben is fogunk találkozni.

4.1. Nem-determinisztikus Turing-gépek

Egy nem-determinisztikus Turing-gép csak annyiban különbözik egy determinisztikustól, hogy minden helyzetben a vezérlőegység állapota és a fejek által leolvasott jelek több lehetséges lépést is megengedhetnek. Pontosabban: egy nem-determinisztikus Turing-gép egy $T = \langle k, \Sigma, \Gamma, \alpha, \beta, \gamma \rangle$ rendezett 6-os, ahol $k \geq 1$ egy természetes szám, Σ és Γ véges halmazok, $* \in \Sigma$, START, STOP$\in \Gamma$, (eddig ugyanúgy, mint a determinisztikus Turing gépnél), és

$$\alpha \subseteq (\Gamma \times \Sigma^k) \times \Gamma,$$

$$\beta \subseteq (\Gamma \times \Sigma^k) \times \Sigma^k,$$

$$\gamma \subseteq (\Gamma \times \Sigma^k) \times \{-1, 0, 1\}^k$$

tetszőleges relációk. A gép egy *legális számolása* lépéseknek egy sorozata, ahol minden lépésben (ugyanúgy, mint a determinisztikus Turing gépnél) a vezérlőegység új állapotban

megy át, a fejek új jeleket írnak a szalagokra, és legföljebb egyet lépnek jobbra vagy balra. Eközben fenn kell állni a következőknek: ha a vezérlőegység állapota a lépés előtt $g \in \Gamma$ volt, és a fejek a szalagokról rendre a $h_1, \ldots, h_k \in \Sigma$ jeleket olvasták, akkor az új g' állapotra, a leírt $h'_1, ..., h'_k$ jelekre és a fejek $\epsilon_1, \ldots, \epsilon_k \in \{-1, 0, 1\}$ lépéseire teljesül:

$$((g, h_1, \ldots, h_k), g') \in \alpha,$$

$$((g, h_1, \ldots, h_k), h'_1, \ldots, h'_k) \in \beta,$$

$$((g, h_1, \ldots, h_k), \epsilon_1, \ldots, \epsilon_k) \in \gamma.$$

Egy nem-determinisztikus Turing-gépnek ugyanazzal a bemenettel tehát sok különböző legális számolása lehet.

Akkor mondjuk, hogy a T nem-determinisztikus Turing-gép t időben *elfogadja* az $x \in \Sigma_0^*$ szót, ha az első szalagjára x-et, a többire az üres szót írva, van a gépnek ezzel a bemenettel olyan legális számolása, mely legfeljebb t lépésből áll, és megálláskor az első szalag 0 pozíciójában az „1” jel áll. (Lehetnek tehát más legális számolások, melyek sokkal tovább tartanak, esetleg meg sem állnak, vagy a szót elutasítják.) Hasonlóan definiáljuk azt, ha a gép az $\mathcal{L}$ nyelvet s tár fölhasználásával fogadja el.

Azt mondjuk, hogy a T nem-determinisztikus Turing-gép *fölismeri* az $\mathcal{L} \subseteq \Sigma^*$ nyelvet, ha $\mathcal{L}$ pontosan azokból a szavakból áll, melyeket T elfogad (akármekkora véges időben). Ha ezenfelül a gép minden $x \in \mathcal{L}$ szót $f(|x|)$ időben elfogad (ahol $f : \mathbb{Z}_+ \to \mathbb{Z}_+$), akkor azt mondjuk, hogy a gép $\mathcal{L}$-et $f(n)$ időben ismeri föl (hasonlóan definiáljuk az $f(n)$ tárral való fölismerhetőséget). Az $f(n)$ időben [tárral] nem-determinisztikus Turing-géppel fölismerhető nyelvek osztályát NTIME($f(n)$) [NSPACE($f(n)$)] jelöli.

A nem-determinisztikus osztályoktól eltérően, egy $\mathcal{L}$ nyelv nem-determinisztikus fölismerhetősége nem jelenti, hogy a komplementer nyelv ($\Sigma_0^* - \mathcal{L}$) is fölismerhető (alább látni fogjuk, hogy erre minden rekurzíve fölsorolható, de nem rekurzív nyelv példa). Ezért bevezetjük a co-NTIME($f(n)$) és co-NSPACE($f(n)$) osztályokat: egy $\mathcal{L}$ nyelv akkor és csak akkor tartozik a co-NTIME($f(n)$) [co-NSPACE($f(n)$)] osztályba, ha a komplementer nyelv $\Sigma_0^* - \mathcal{L}$ az NTIME($f(n)$) [NSPACE($f(n)$)] osztályba tartozik.

Megjegyzések: 1. Természetesen a determinisztikus Turing-gépek speciális nem-determinisztikus Turing-gépeknek is tekinthetők.

2. A nem-determinisztikus Turing-gépekkel nem kívánunk semmilyen valódi számoló-eszközt sem modellezni; látni fogjuk, hogy ezek a gépek a feladatok *kitűzésének* és nem *megoldásának* eszközei.

3. A nem-determinisztikus Turing-gép egy szituációban többféle lépést tehet; ezeken nem tételeztünk föl semmilyen valószínűségeloszlást, tehát nem beszélhetük egy számolás valószínűségéről. Ha ezt tennénk, akkor a *randomizált Turing-gépekről* beszélnénk, melyekről az 5. fejezetben lesz szó. Ezek, a nem-determinisztikus gépekkel ellentétben, gyakorlatilag is fontos számítási eljárásokat modelleznek.

4. Említettük, hogy ha egy nem-determinisztikus Turing-gép egy adott bemenetet t lépésben elfogad, akkor lehetnek sokkal hosszabb, akár végtelen legális számolásai is. Ha a t időkorlát a bemenet hosszának jól számolható függvénye, akkor betehetünk a gépbe egy „órát”, mely a számolást t lépés után leállítja. Így módosíthatnánk a definíciót úgy, hogy minden legális számolás legfeljebb t idejű, és ezek közül legalább egy elfogadja a szót.

4.1.1 Tétel: *A nem-determinisztikus Turing-géppel fölismerhető nyelvek pontosan a rekurzíve fölsorolható nyelvek.*

Bizonyítás: Tegyük föl először, hogy az $\mathcal{L}$ nyelv rekurzíve fölsorolható. Ekkor a 2.1.1 Lemma szerint van olyan T Turing-gép, mely akkor és csak akkor áll meg véges idő múlva egy x bemeneten, ha $x \in \mathcal{L}$. Módosítsuk T-t úgy, hogy ha megáll, akkor előtte írjon az első szalag 0 mezejére 1-est. Nyilván T-nek akkor és csak akkor van x-et elfogadó legális számolása, ha $x \in \mathcal{L}$.

Megfordítva, tegyük föl, hogy $\mathcal{L}$ fölismerhető egy nem-determinisztikus T Turing-géppel, megmutatjuk, hogy rekurzíve fölsorolható. Föltehetjük, hogy $\mathcal{L}$ nem üres, és legyen $a \in \mathcal{L}$. Álljon $\mathcal{L}^{\#}$ a T Turing-gép összes véges legális számolásaiból (lépésről-lépésre leírva alkalmas kódolásban, hogy mi a vezérlőegység állapota, melyik fej hol van, mit olvas, mit ír, merre lép). Nyilvánvaló, hogy $\mathcal{L}^{\#}$ rekurzív. Legyen S egy olyan Turing-gép, mely egy y bemenetre eldönti, hogy az $\mathcal{L}^{\#}$-ben van-e, és ha igen, akkor olyan legális számolást ír-e le, mely egy x szót elfogad. Ha igen, nyomtassa ki az x szót; ha nem, nyomtassa ki az a szót. Nyilvánvaló, hogy az S által definiált rekurzív függvény értékkészlete éppen $\mathcal{L}$. □

4.2. Nem-determinisztikus algoritmusok bonyolultsága

Rögzítsünk egy véges Σ_0 ábécét, és tekintsünk egy efölötti $\mathcal{L}$ nyelvet. Először azt vizsgáljuk meg, hogy mit is jelent, ha $\mathcal{L}$ nem-determinisztikus Turing-géppel bizonyos időben fölismerhető. Megmutatjuk, hogy ez azzal függ össze, hogy mennyire egyszerű „rábizonyítani" egy szóra, hogy $\mathcal{L}$-ben van.

Azt mondjuk, hogy $\mathcal{L}$-nek *tanúja* az $\mathcal{L}_0$ nyelv, ha $x \in \mathcal{L}$ akkor és csak akkor, ha van olyan $y \in \Sigma_0^*$ szó, hogy $x\&y \in \mathcal{L}_0$ (itt & egy új jel, mely az x és y szavak elválasztására szolgál).

Pontosabban, az $\mathcal{L}$ nyelvnek $f(n)$ *hosszúságú*, $g(n)$ *idejű tanúja* az $\mathcal{L}_0$ nyelv, ha tanúja, és egyrészt $\mathcal{L}_0 \in \mathrm{DTIME}(g(n))$, másrészt minden $x \in \mathcal{L}$-re van olyan $y \in \Sigma_0^*$ szó, hogy $|y| \leq f(|x|)$ és $x\&y \in \mathcal{L}_0$.

4.2.1 Tétel: Legyen f jól számolható függvény és $g(n) \geq n$.

(a) *Minden* $\mathcal{L} \in \mathrm{NTIME}(f(n))$ *nyelvnek van* $O(f(n))$ *hosszúságú, lineáris idejű tanúja.*

(b) *Ha egy* $\mathcal{L}$ *nyelvnek van* $f(n)$ *hosszúságú,* $g(n)$ *idejű tanúja, akkor*

$$\mathcal{L} \in \mathrm{NTIME}(g(n+1+f(n))).$$

Bizonyítás: (a) Legyen T az $\mathcal{L}$ nyelvet $f(n)$ időben fölismerő nem-determinisztikus Turing-gép, mondjuk két szalaggal. A 4.1.1 tétel bizonyításának mintájára, rendeljük hozzá minden $x \in \mathcal{L}$ szóhoz a T egy olyan legális számolásának a leírását, mely x-et $f(|x|)$ időben elfogadja. Nem nehéz olyan Turing-gépet csinálni, mely egy N hosszúságú leírásról $O(N)$ lépésben eldönti, hogy az egy legális számolás leírása-e, és ha igen, akkor ez a számolás az x szót fogadja-e el. Így az x-et elfogadó legális számolások leírásai alkotják a tanút.

(b) Legyen $\mathcal{L}_0$ az $\mathcal{L}$ nyelv $f(n)$ hosszúságú, $g(n)$ idejű tanúja, és tekintsünk egy olyan S determinisztikus Turing-gépet, mely $\mathcal{L}_0$-t $g(n)$ időben eldönti. Konstruáljunk egy nem-determinisztikus T Turing-gépet, mely a következőt teszi. Ha első szalagjára x van írva, akkor először (determinisztikusan) kiszámítja $f(|x|)$ értékét, és ennyi 1-est ír a második szalagra. Ezután x végére ír egy & jelet, majd átmegy az egyetlen olyan állapotba, melyben működése nem-determinisztikus. Ennek során egy legfeljebb $f(|x|)$ hosszúságú y szót ír az x& szó után. Ezt úgy teszi, hogy amíg a második szalagon 1-est olvas, $|\Sigma_0|+1$ legális lépése van: vagy leírja az ábécé valamely betűjét az első szalagra, jobbra lép az első szalagon, és balra a második szalagon, vagy nem ír semmit, hanem átmegy egy új START2 állapotba. Ha a második szalagon üres mezőt olvas, mindenképpen a START2 állapotba megy át.

A START2 állapotban a gép az első szalagon a kezdőmezőre állítja a fejet, letörli a második szalagot, majd az S Turing-gépnek megfelelően működik.

Ennek a T gépnek akkor és csak akkor van x-et elfogadó legális számolása, ha van olyan legfeljebb $f(|x|)$ hosszúságú $y \in \Sigma_0$ szó, melyre S az $x\&y$ szót elfogadja, vagyis ha $x \in \mathcal{L}$. Nyilvánvaló, hogy ennek a legális számolásnak a lépésszáma legfeljebb $O\left(f(|x|)\right) + g\left(|x| + 1 + f(|x|)\right) = O\left(g\left(|x| + 1 + f(|x|)\right)\right)$. □

4.2.2 Következmény: *Tetszőleges $\mathcal{L} \subseteq \Sigma_0^*$ nyelvre az alábbi tulajdonságok ekvivalensek:*

(i) *$\mathcal{L}$ fölismerhető nem-determinisztikus Turin-gépen polinomiális időben.*

(ii) *$\mathcal{L}$-nek van polinomiális hosszúságú és idejű tanúja.* □

Megjegyzés: Bizonyítás, sőt pontos megfogalmazás nélkül megemlítjük, hogy ezek a tulajdonságok azzal is ekvivalensek, hogy az $\mathcal{L}$ nyelvnek adható a halmazelmélet axiómarendszerében olyan definíciója, hogy minden $x \in \mathcal{L}$ szóra az az állítás, hogy „$x \in \mathcal{L}$”, bebizonyítható a halmazelmélet axiómáiból $|x|$-ben polinomiális számú lépésben.

A 4.2.2 Következményben kimondott tulajdonsággal rendelkező nyelvek osztályát NP-vel jelöljük. Azon $\mathcal{L}$ nyelvek, melyekre $\Sigma_0^* - \mathcal{L} \in$ NP, alkotják a co-NP osztályt. Mint említettük, ezeknél a feladatosztályoknál nem a feladat megoldása, hanem kitűzése könnyű. A következő szakaszban látni fogjuk, ezek az osztályok alapvetőek, a gyakorlat szempontjából fontos algoritmikus problémák nagy részét tartalmazzák.

Számos fontos nyelv tanújával van megadva, pontosabban, a tanú definíciójában szereplő $\mathcal{L}_0$ nyelv és $f(n)$ függvény által (erre sok példát fogunk látni a következő pontban). Ilyen esetben nemcsak azt kérdezhetjük, hogy egy adott x szó $\mathcal{L}$-ben van-e (vagyis, hogy létezik-e olyan y, hogy $|y| \leq f(n)$ és $x\&y \in \mathcal{L}_0$), hanem igen gyakran szeretnénk egy ilyen y-t elő is állítani. Ezt a feladatot az $\mathcal{L}$ nyelvhez tartozó *kereső feladatnak* nevezzük. Természetesen egy nyelvhez sok különböző kereső feladat tartozhat. A kereső feladatnak akkor is lehet értelme, ha a megfelelő döntési feladat triviális. Például minden számnak létezik prímfelbontása, de ezt nem könnyű megtalálni.

Mivel minden determinisztikus Turing-gép tekinthető nem-determinisztikusnak, nyilvánvaló, hogy

$$\mathrm{DTIME}(f(n)) \subseteq \mathrm{NTIME}(f(n)).$$

Annak az analógiájára, hogy van rekurzíve fölsorolható, de nem rekurzív nyelv (vagyis idő- és tárkorlátozás nélkül a nem-determinisztikus Turing-gépek „erősebbek”), azt várjuk,

hogy itt szigorú tartalmazás érvényes. Ez azonban csak nagyon speciális esetekben van bebizonyítva (pl. lineáris függvények esetén Paul, Pippinger, Trotter és Szemerédi igazolták). A következő szakaszokban részletesen tárgyalni fogjuk a legfontosabb speciális esetet, a P és NP osztályok viszonyát.

A nem-determinisztikus idő- és tárbonyolultsági osztályok között az alábbi egyszerű összefüggések állnak fenn:

4.2.3 Tétel: *Legyen f jól számolható függvény. Ekkor*

(a) $\mathrm{NTIME}(f(n)) \subseteq \mathrm{DSPACE}(f(n))$

(b) $\mathrm{NSPACE}(f(n)) \subseteq \cup_{c>0}\mathrm{DTIME}(2^{cf(n)})$

(c) $\mathrm{NTIME}(f(n)) \subseteq \cup_{c>0}\mathrm{DTIME}(2^{cf(n)})$.

Bizonyítás: (a) A konstrukció lényege, hogy egy nem-determinisztikus Turing-gép összes legális számolását egymásután kipróbálhatjuk úgy, hogy mindig csak annyi helyet használunk, amennyi egy számoláshoz kell; ezenfelül kell még némi hely annak nyilvántartásához, hogy hol tartunk az esetek kipróbálásában.

Pontosabban ez így írható le: Legyen T egy olyan nem-determinisztikus Turing-gép, mely az $\mathcal{L}$ nyelvet $f(n)$ idő alatt fölismeri. Mint említettük, föltehetjük, hogy T bármely legális számolása legfeljebb $f(n)$ lépésből áll, ahol n a bemenet hossza. Módosítsuk T működését úgy, hogy (valamely x bemenetre) először mindig a lexikografikusan legelső legális lépést válassza (rögzítjük Γ és Σ egy-egy rendezését, ekkor jól definiált a legális lépések lexikografikusan rendezése). Adunk a gépnek egy új „nyilvántartó" szalagot, melyre fölírja, hogy milyen legális lépéseket választott. Ha ez a számolás nem végződik x elfogadásával, akkor gép ne álljon meg, hanem a nyilvántartó szalagon keresse meg az utolsó olyan lépést (mondjuk ez a j-edik), amit megváltoztathat lexikografikusan nagyobbra, és hajtson végre egy legális számolást úgy, hogy a j-edik lépésig a nyilvántartó szalagon fölírt legális lépéseket teszi, a j-edik lépésben a lexikografikusan rákövetkezőt, ezekután pedig a lexikografikusan legelsőt (és persze ennek megfelelően írja át a nyilvántartó szalagot). A visszalépéseknél persze a T szalagjainak a tartalmát is vissza kell állítani, ez pl. könnyű, ha a nyilvántartó szalagra azt is felírjuk, hogy egy-egy lépésben milyen betűket írt át a gép.

A módosított, immár determinisztikus Turing-gép az eredeti gép minden legális számolását végigpróbálja, és közben csak annyi tárat használ, mint az eredeti gép (ami legfeljebb $f(n)$), plusz a nyilvántartó szalagon használt hely (ami ismét csak $O(f(n))$.

(b) Legyen $T = \langle k, \Sigma, \Gamma, \Phi\rangle$ egy olyan nem-determinisztikus Turing-gép, mely $\mathcal{L}$-et $f(n)$ tárral fölismeri. Föltehetjük, hogy T-nek csak egy szalagja van. Végig akarjuk próbálni T összes legális számolását. Kicsit vigyázni kell, mert T-nek egy legális számolása akár $2^{f(n)}$ lépésig tarthat, és így akár $2^{2^{f(n)}}$ legális számolás lehet; ennyi számolás végigvizsgálására már nincs időnk.

Hogy a végigvizsgálást jobban szervezzük, a helyzetet egy gráffal szemléltetjük a következőképpen. Rögzítsük a bemenetek n hosszát. A gép egy *helyzetén* egy (g, p, h) hármast értünk, ahol $g \in \Gamma$, $-f(n) \leq p \leq f(n)$ és $h \in \Sigma^{2f(n)+1}$. A g állapot a vezérlőegység állapota az adott lépésben, p szám mondja meg, hogy hol van a fej, h pedig megadja a szalagon levő jeleket (mivel csak legfeljebb $f(n)$ tárigényű számítások érdekelnek minket, ezért elég $2f(n)+1$ mezőt tekinteni). Látható, hogy a helyzetek száma legföljebb $|\Gamma|f(n)|\Sigma|^{2f(n)+1} = 2^{O(f(n))}$ Minden helyzetet kódolhatunk egy-egy Σ fölötti $O(f(n))$

hosszúságú szóval.

Készítsük el mármost azt az irányított G gráfot, melynek csúcsai a helyzetek, és az u csúcsból a v csúcsba rajzoljunk nyilat, ha az u helyzetből a v helyzetbe a gép egy legális lépése vezet. Vegyünk föl egy v_0 csúcsot, és húzzunk v_0-hoz élt minden olyan „helyzetből", melyben a gép a STOP állapotban van, és a szalagja 0-dik mezején 1 áll. Jelölje u_x az x bemenetnek megfelelő induló helyzetet. Az x szó akkor és csak akkor van $\mathcal{L}$-ben, ha ebben az irányított gráfban vezet u_x-ből v_0-ba irányított út.

A G gráfot $2^{O(f(n))}$ idő alatt meg tudjuk konstruálni, majd (pl. szélességi kereséssel) $O(|V(G)|^2) = 2^{O(f(n))}$ idő alatt el tudjuk dönteni, hogy tartalmaz-e u_x-ből v_0-ba irányított utat.

(c) Következik a (b) részből, mivel NTIME$(f(n)) \subseteq$NSPACE$(f(n))$. □

Az alábbi érdekes tétel azt mutatja, hogy a tárigényt nem csökkenti nagyon lényegesen, ha nem-determinisztikus Turing-gépeket is igénybe veszünk:

4.2.4 Savitch Tétele: *Ha $f(n)$ jól számolható függvény, és $f(n) \geq \log n$, akkor minden $\mathcal{L} \in$ NSPACE$(f(n))$ nyelvhez van olyan $c > 0$, hogy $\mathcal{L} \in$ DSPACE$(cf(n)^2)$.*

Bizonyítás: Legyen $T = \langle 1, \Sigma, \Gamma, \Phi \rangle$ egy olyan nem-determinisztikus Turing-gép, mely $\mathcal{L}$-et $f(n)$ tárral fölismeri. Tekintsük a fenti G gráfot; el akarjuk dönteni, hogy tartalmaz-e u_x-ből v_0-ba irányított utat. Természetesen ezt a gráfot nem akarjuk teljes egészében megkonstruálni, mert igen nagy. Ezért úgy tekintjük, hogy egy „orákulummal" van megadva. Ez itt most annyit tesz, hogy két pontról egyetlen lépésben el tudjuk dönteni, hogy össze vannak-e kötve. Pontosan ez így fogalmazható meg: Bővítsük ki a Turing-gépek definícióját. Egy *orákulumos Turing-gépen* olyan Turing-gépet értünk, melynek egy külön speciális szalagját az orákulum számára foglaljuk le. A gépnek van még három különleges állapota, „ORÁKULUM-KÉRDÉS", „ORÁKULUM-IGEN" és „ORÁKULUM-NEM". Ha a gép az „ORÁKULUM-KÉRDÉS" állapotba jut, akkor a következő állapota vagy az „ORÁKULUM-IGEN", vagy az „ORÁKULUM-NEM" lesz attól függően, hogy az orákulum-szalagján levő u, v csúcsnevekre vezet-e él u-ból v-be.

4.2.5 Lemma: *Tegyük föl, hogy orákulummal adott egy G irányított gráf a t hosszúságú szavak halmazán. Ekkor létezik olyan orákulumos Turing-gép, mely adott u, v csúcsokhoz és q természetes számhoz legfeljebb $qt + q \log q$ tár fölhasználásával eldönti, hogy vezet-e G-ben u-ból v-be legfeljebb 2^q hosszúságú irányított út.*

A megszerkesztendő Turing-gépnek két szalagja lesz az orákulum-szalagon kívül. Induláskor az első szalag az (u, q) párt, a második a (v, q) párt tartalmazza. A Turing-gép működése során mindkét szalag néhány (x, r) párt fog tartalmazni, ahol x egy csúcs neve, $r \leq q$ pedig egy természetes szám.

Legyen (x, r) és (y, s) a két szalagon lévő utolsó pár. A gép arra keres választ, hogy x-ből y-ba vezet-e legfeljebb 2^p hosszúságú út, ahol $p = \min\{r, s\}$. Ha $p = 0$, akkor a válasz egy orákulum-kérdéssel megválaszolható. Egyébként az első szalag végére egy (w, m) párt írunk, ahol $m = p - 1$, w pedig egy csúcs neve, és rekurzíve meghatározzuk, hogy van-e w-ből y-ba legfeljebb 2^m hosszúságú út. Ha van, felírjuk (w, m)-et a második szalag végére, letöröljük az első szalag végéről, és meghatározzuk, hogy van-e x-ből w-be legfeljebb 2^m hosszúságú út. Ha van, akkor letöröljük (w, m)-et a második szalag végéről:

tudjuk, hogy van x-ből y-ba legfeljebb 2^p hosszúságú út. Ha akár x és w, akár w és y között nincsen legfeljebb 2^m hosszúságú út, akkor a lexikografikusan következő w-vel próbálkozunk. Ha már minden w-t végigpróbáltunk, tudjuk, hogy x-ből y-ba nem vezet legfeljebb 2^p hosszúságú út.

Könnyű belátni, hogy mindkét szalagon a felírt párok második elemei balról jobbra csökkennek, és így a szalagokra legfeljebb q pár kerül. Egy pár $O(t + \log q)$ jelet igényel, így a felhasznált tár csak $O(qt + q \log q)$. Ezzel a lemmát bebizonyítottuk.

Visszatérve a tétel bizonyításához, vegyük észre, hogy az, hogy a G gráf két csúcsa között vezet-e él, tár igénybevétele nélkül könnyen eldönthető; ezt az eldöntést úgy is tekinthetjük, mint egy orákulumot. Így a Lemma alkalmazható $t, q = O(f(n))$ értékekkel, és azt kapjuk, hogy legfeljebb $qt + q \log q = O(f(n)^2)$ tárral eldönthető, hogy egy adott u_x pontból vezet-e irányított út v_0-ba, vagyis, hogy az x szó $\mathcal{L}$-ben van-e. □

Mint megjegyeztük, igen a fontos a polinomiális tárral determinisztikus Turing-gépen eldönthető nyelvek PSPACE osztálya. Kézenfekvő volna bevezetni az NPSPACE osztályt, mely a polinomiális tárral nem-determinisztikus Turing-gépen fölismerhető nyelvek osztálya. Savitch tételének alábbi következménye mutatja azonban, hogy ez nem vezetne új fogalomhoz:

4.2.6 Következmény: PSPACE = NPSPACE.

4.3. Példák NP-beli nyelvekre

A továbbiakban, ha gráfot mondunk, akkor többszörös és hurokél nélküli, ún. *egyszerű* gráfot értünk alatta. Egy ilyen gráf egyértelműen leírható adjacencia-mátrixának főátló feletti részével, mely sorfolytonosan írva egy $\{0,1\}^*$-beli szót alkot. Így gráfok egy tulajdonságát úgy tekinthetjük, mint egy $\{0,1\}$ fölötti nyelvet. Beszélhetünk tehát arról, hogy egy gráf-tulajdonság NP-ben van. (Vegyük észre, hogy ha a gráfot más szokásos módon adnánk meg, például úgy, hogy minden pontra megadjuk szomszédainak a listáját, akkor a gráf-tulajdonságok NP-beli volta nem változna. Ezeket a reprezentációkat ugyanis könnyű egymásból polinomiális időben kiszámítani.) Ilyen módon NP-ben vannak az alábbi gráf-tulajdonságok:

a) Összefüggőség. Tanú: $\binom{n}{2}$ út, minden pontpárra egy-egy.

b) Nem-összefüggőség. Tanú: a ponthalmaz egy valódi részhalmaza, melyet nem köt össze él a többi ponttal.

c) Síkbarajzolhatóság. A természetes tanú egy konkrét lerajzolás; a gráfelméletből ismert tétel, hogy ez mindig megvalósítható úgy, hogy az élek egyenes szakaszok, és így elegendő a csúcsok koordinátáit megadni. Vigyázni kell azonban, mert a gráf lerajzolásában a csúcsok koordinátái esetleg igen sokjegyű számok lesznek, így a tanú hosszára tett kikötés nem teljesül. (Be lehet bizonyítani, hogy minden síkbarajzolható gráf úgy is síkbarajzolható, hogy minden éle egyenes szakasz, és minden csúcs koordinátái olyan egész számok, melyeknek n-ben polinomiális a jegyeinek a száma.)

Megadható azonban a síkbarajzolás kombinatorikusan is. Legyen G n pontú, m élű, és az egyszerűség kedvéért összefüggő gráf. Minden országra megadjuk a határoló zárt élsorozatot. Ebben az esetben a megadott élsorozat-rendszerről elegendő azt ellenőrizni, hogy minden él pontosan kettőben van benne, és az élsorozatok száma $m-n+2$. Az, hogy ilyen élsorozat-rendszer létezése szükséges feltétele a síkbarajzolhatóságnak, az *Euler-féle formulából* következik:

4.3.1 Tétel: *Ha egy összefüggő síkgráf pontjainak száma n, éleinek száma m, akkor országainak száma $m-n+2$.* □

Az elégségességhez kicsit nehezebb topológiai eszközök kellenek, ezeket itt nem részletezzük.

d) Síkba nem rajzolhatóság. Az alábbi alapvető gráfelméleti tételt lehet felhasználni:

4.3.2 Kuratowski tétele: *Egy gráf akkor és csak akkor rajzolható síkba, ha nem tartalmaz olyan részgráfot, mely élek felosztásával jön létre a teljes 5-szögből vagy a három-ház-három-kút gráfból.* □

Ha a gráf nem síkbarajzolható, akkor erre ez a részgráf szolgálhat tanúként.

e) Teljes párosítás létezése. Tanú: a párosítás.

f) Teljes párosítás nem létezése. A tanút páros gráfok esetén a következő alapvető tétel alapján adhatjuk meg:

4.3.3 Kőnig–Frobenius tétel: *Egy G páros gráfban akkor és csak akkor van teljes párosítás, ha ugyanannyi „alsó" és „felső" pontja van, és az „alsó" pontjai közül választott bármely X ponthalmaznak legalább $|X|$ szomszédja van.* □

Így ha a páros gráfban nincsen teljes párosítás, akkor arra a tételbeli feltételt megsértő X halmaz a tanú.

Általános gráfra egy ugyancsak alapvető, bár némileg bonyolultabb tételt használhatunk:

4.3.4 Tutte tétele: *Egy G gráfban akkor és csak akkor van teljes párosítás, ha a gráf pontjainak bármely X halmazát elhagyva, a maradék gráf páratlan pontszámú összefüggő komponenseinek száma legfeljebb $|X|$.* □

Így ha a gráfban nincsen teljes párosítás, akkor arra egy olyan X ponthalmaz a tanú, melyet elhagyva túl sok páratlan komponens keletkezik.

g) Hamilton kör létezése. Tanú: a Hamilton-kör.

h) Három színnel való színezhetőség. Tanú: a színezés.

Ezek közül az a)-f) tulajdonságok P-ben vannak. Ez a) és b) esetében egyszerűen belátható (széltében vagy mélységben keresés). c)-d)-re polinomiális idejű síkbarajzolási algoritmust Hopcroft és Tarjan adott meg. e)-f)-re páros gráf esetén „magyar módszer"

alkalmazható (ezt Kőnig és Egerváry munkái alapján Kuhn fogalmazta meg), nem-páros gráfokra pedig Edmonds algoritmusa. A g)-h) tulajdonságokra nem ismeretes polinomiális idejű algoritmus (erre a következő szakaszban még visszatérünk).

A számelméletben és algebrában is igen sok probléma tartozik az NP osztályba. Minden természetes számot tekinthetünk úgy, mint egy $\{0,1\}^*$-beli szót (kettes számrendszerben felírva a számot). Ebben az értelemben NP-ben vannak az alábbi tulajdonságok:

i) Összetettség. Tanú: egy valódi osztó.

j) Prímség. Itt lényegesen nehezebb a megfelelő tanút megtalálni. Az alábbi alapvető számelméleti tételt használjuk:

4.3.5 Tétel. *Egy $n \geq 2$ egész szám akkor és csak akkor prím, ha van olyan a természetes szám, melyre $a^{n-1} \equiv 1 \pmod{n}$ de $a^m \not\equiv 1 \pmod{n}$ ha $1 \leq m < n-1$.* □

E tétel alapján azt szeretnénk, hogy arra, hogy n prím, az a szám legyen a tanú. Mivel nyilvánvaló, hogy az a számnak csak n-nel vett osztási maradéka játszik szerepet, mindig lesz olyan a tanú is, melyre $0 \leq a < n$. Így tehát a tanú hosszára tett kikötésünk rendben is volna: a-nak nincsen több számjegye, mint k, az n jegyeinek száma. A 3.1.2 Lemma alapján ellenőrizni lehet polinomiális időben a

$$(1)\ a^{n-1} \equiv 1 \pmod{n}$$

feltételt is. Sokkal nehezebb kérdés azonban, hogy hogyan ellenőrizzük a további feltételeket:

$$(2) a^m \not\equiv 1 \quad (1 \leq m < n-1) \pmod{n}.$$

Minden konkrét m-re ez ugyanúgy megtehető polinomiális időben, mint (1) ellenőrzése, de ezt (látszólag) $n-2$-szer, tehát k-ban exponenciálisan sokszor kell megtennünk. Felhasználjuk azonban azt az elemi számelméleti tényt, hogy ha (1) teljesül, akkor a legkisebb olyan $m = m_0$, mely (2)-t megsérti (ha van ilyen egyáltalán), $(n-1)$-nek osztója. Azt is könnyű belátni, hogy ekkor (2)-t m_0 minden $(n-1)$-nél kisebb többszöröse is megsérti. Így ha $n-1$ prímfelbontása $n-1 = p_1^{r_1} p_2^{r_2} \dots p_t^{r_t}$, akkor valamely $m = (n-1)/p_i$ is megsérti (2)-t. Elegendő tehát az ellenőrizni, hogy minden $1 \leq i \leq t$-re

$$a^{(n-1)/p_i} \not\equiv 1 (\mathrm{mod}\ n).$$

Mármost nyilvánvaló, hogy $t \leq k$, és így legfeljebb k értékre kell (2)-t ellenőrizni, melyet az előzőekben leírt módon összesen polinomiális időben meg lehet tenni.

Van azonban még egy nehézség: hogyan számítjuk ki $n-1$ prímfelbontását? Ez magában nehezebb probléma, mint eldönteni, hogy egy szám prím-e. Megtehetjük azonban, hogy magát a prímfelbontást is a „tanúhoz" soroljuk; ez tehát az a számon kívül a $p_1, r_1, \dots, p_t, r_t$ számokból áll (ez könnyen láthatóan legfeljebb $3k$ bit). Ekkor már csak az a probléma, hogy ellenőrizni kell, hogy ez valóban prímfelbontás-e, vagyis hogy $n-1 = p_1^{r_1} \dots p_t^{r_t}$ (ez könnyű), és hogy $p_1, \dots, p_t$ valóban prímek. Ehhez rekurzíve igénybe vehetjük magát az itt tárgyalt eljárást.

Ellenőrizni kell azonban, hogy ez a rekurzió polinomiálisan hosszú tanúkat ad és polinomiális időben eldönthető, hogy ezek tényleg tanúk. Jelölje $L(k)$ az így definiált tanú maximális hosszát k jegyű számok esetén. Ekkor a fenti rekurzió szerint

$$L(k) \leq 3k + \sum_{i=1}^{t} L(k_i),$$

ahol k_i a p_i prím jegyeinek száma. Mivel $p_1 \dots p_t \leq n-1 < n$, következik, hogy

$$k_1 + \ldots + k_t \leq k.$$

Az is nyilvánvaló, hogy $k_i \leq k-1$. Ezek alapján a fenti rekurzióból következik, hogy $L(k) \leq 3k^2$. Ez ugyanis nyilvánvaló $k = 1$-re, és ha már tudjuk minden k-nál kisebb számra, akkor

$$\begin{aligned} L(k) &\leq 3k + \sum_{i=1}^{t} L(k_i) \leq 3k + \sum_{i=1}^{t} 3k_i^2 \\ &\leq 3k + 3(k-1)\sum_{i=1}^{t} k_i \leq 3k + 3(k-1)\cdot k \leq 3k^2 \end{aligned}$$

Hasonlóan lehet igazolni, hogy egy jelsorozatról polinomiális időben eldönthető, hogy a fenti módon definiált tanú-e.

k) Korlátos osztó létezése. Az n számról nem elég eldönteni, hogy prím-e, hanem ha azt találjuk, hogy nem prím, akkor egy valódi osztóját is szeretnénk megtalálni. (Ha ezt a feladatot meg tudjuk oldani, akkor a teljes prímfelbontást ennek ismétlésével meg tudjuk kapni.) Ez a feladat nem igen-nem probléma, de nem nehéz átfogalmazni ilyen feladattá: *Adott két természetes szám: n és k; van-e n-nek k-nál nem nagyobb valódi osztója?* Ez a feladat nyilvánvalóan NP-ben van (tanú: az osztó).

NP-ben van azonban a komplementer nyelv is, vagyis mindazon (n,k) párok halmaza, melyekre az áll, hogy n minden valódi osztója nagyobb, mint k. Erre tanú ugyanis n-nek a prímfelbontása, mellékelve minden prímtényezőről annak a tanúját, hogy az prím.

Nem ismeretes, hogy az összetettség (még kevésbé a korlátos osztó létezése) P-ben van-e. Az algoritmus fogalmának bővítésével, véletlen számok felhasználásával egy számról eldönthető polinomiális időben hogy prím-e (lásd a „Randomizált algoritmusok" c. fejezetet). Ugyanakkor a megfelelő keresési feladat (valódi osztó keresése), ill. a korlátos osztó létezésének eldöntése lényegesen nehezebb, erre még véletlent használva sem sikerült polinomiális idejű algoritmust adni.

l) Polinom reducibilitása a racionális test fölött. Tanú: egy valódi osztó. Meg kell itt azonban gondolni, hogy egy valódi osztó felírásához szükséges bitek száma korlátozható az eredeti polinom felírásában szereplő bitek számának egy polinomjával. (Ennek bizonyítását nem részletezzük.) Megmutatható az is, hogy ez a nyelv P-ben is benne van.

Egy $Ax \leq b$ lineáris egyenlőtlenségrendszert (ahol A m sorú és n oszlopú egész mátrix, b pedig m elemű oszlopvektor) tekinthetünk a „0", „1", „," és „;" jelekből álló ábécé fölötti szónak pl. úgy, hogy az elemeit kettes számrendszerben adjuk meg, és sorfolytonosan írjuk le, minden szám után vesszőt, minden sor után pontosvesszőt írva. Lineáris egyenlőtlenségrendszerek alábbi tulajdonságai NP-ben vannak:

m) Megoldás létezése. Itt nyilvánvalóan adódik tanúnak a megoldás, de vigyázni kell: be kell látni, hogy ha egy egész együtthatós lineáris egyenlőtlenségrendszer megoldható, akkor a racionális számok körében is megoldható, éspedig úgy, hogy a megoldás számlálóinak és nevezőinek csak polinomiális számú számjegye van. Ezek a tények a lineáris programozás elméletének alapjaiból következnek.

n) Megoldás nem létezése. A lineáris programozásból ismert alapvető tételt használjuk fel:

4.3.6 Farkas-lemma: *$Ax \leq b$ akkor és csakis akkor nem oldható meg, ha az $yA = 0$, $yb = -1$, $y \geq 0$ egyenlőtlenségrendszer megoldható.* □

Ennek alapján a megoldás nem létezésére a lemmában szereplő másik egyenlőtlenségrendszer megoldásának megadásával tanúsíthatjuk.

o) Egész megoldás létezése. Itt is maga a megoldás a tanú, de a)-hoz hasonló meggondolásokra van szükség, melyek itt bonyolultabbak (Votyakov és Frumkin eredménye).

Érdemes megjegyezni, hogy a *lineáris programozás* alapproblémája, t.i. lineáris feltételek mellett lineáris célfüggvény optimumának megkeresése, könnyen visszavezethető a lineáris egyenlőtlenségrendszerek megoldhatóságának kérdésére. Hasonlóan, optimális egész megoldás keresése visszavezethető egész megoldás létezésének az eldöntésére.

Hosszú ideig nem volt ismeretes, hogy a lineáris egyenlőtlenségrendszerek megoldhatóságának problémája P-ben van-e (a közismert szimplex-módszer nem polinomiális). Az első polinomiális algoritmus erre a feladatra Hacsiján ellipszoid-módszere volt. Ennek a módszernek az ideje azonban igen magas fokú polinomra vezetett, ezért gyakorlatban nem versenyezhetett a szimplex-módszerrel, mely ugyan a legrosszabb esetben exponenciális, de átlagban (tapasztalat szerint) sokkal gyorsabb, mint az ellipszoid-módszer. Azóta több polinomiális idejű lineáris programozási algoritmust is találtak, ezek közül Karmarkar módszere a szimplex módszerrel a gyakorlatban is felveszi a versenyt.

Lineáris egyenlőtlenségrendszerek egész számokban való megoldására nem ismeretes polinomiális algoritmus, sőt ilyen algoritmus nem is várható (megint csak lásd a következő pontot).

Az előző példasort vizsgálva, érdemes az alábbi megállapításokat tenni:

- Sok NP-beli tulajdonság tagadása is NP-beli (vagyis a megfelelő nyelv komplementere is NP-ben van). Ez a tény azonban általában nem triviális, sőt a matematika különböző ágaiban sokszor a legalapvetőbb tételek mondják ki ezt egyes nyelvekről (Kuratowski, Frobenius-Kőnig, Tutte tétele, Farkas lemmája).

- Igen sokszor az a helyzet, hogy ha egy tulajdonságról (nyelvről) kiderül, hogy NP∩co-NP-ben van, akkor előbb-utóbb az is kiderül, hogy P-ben van. Például ez történt teljes párosítások létezésével, síkbarajzolhatósággal, lineáris egyenlőtlenségrendszer megoldásával. Nagy erővel folytak a vizsgálatok a prímteszteléssel kapcsolatban. Végül 2002-ben Agrawal, Kayal, és Saxena bebizonyították, hogy ez a probléma is P-ben van.

- Ha NP-t a „rekurzíve fölsorolható”, P-t pedig a „rekurzív” analogonjának tekintjük, akkor azt várhatjuk, hogy mindig ez a helyzet. Erre azonban nincsen bizonyítás, sőt

nem is igen várható, hogy igaz legyen teljes általánosságban.

- Más NP-beli problémákkal az a helyzet, hogy megoldásuk polinomiális időben reménytelennek tűnik, igen nehezen kezelhetők (Hamilton-kör, gráf-színezés, lineáris egyenlőtlenségrendszer egészértékű megoldása). Nem tudjuk bebizonyítani, hogy ezek nincsenek P-ben (nem tudjuk, hogy P=NP fennáll-e); azonban mégis bebizonyítható olyan egzakt tulajdonságuk, mely mutatja, hogy nehezek. Ezzel foglalkozunk a következő pontban.

- Sok olyan NP-beli probléma van, melyet ha meg tudunk oldani, akkor a (természetes módon) hozzárendelhető kereső feladatot is meg tudjuk oldani. Például ha polinomiális időben el tudjuk dönteni, hogy egy gráfban van-e teljes párosítás, akkor a következőképpen tudunk ugyancsak polinomiális időben teljes párosítást keresni: addig hagyjunk el éleket a gráfból, amíg csak marad benne teljes párosítás. Amikor megakadunk, a maradék gráf éppen egy teljes párosítás kell, hogy legyen. Hasonló egyszerű fogásokkal vezethető vissza a Hamilton-kör létezésének, a 3 színnel való színezhetőségnek stb. megfelelő kereső feladat a döntési feladatra. Ez azonban nem mindig van így. Például hiába tudjuk (legalábbis bizonyos értelemben) polinomiális időben eldönteni, hogy egy szám prím-e, nem sikerült ezt egy valódi osztó megtalálásának problémájára alkalmazni.

Természetesen vannak érdekes nyelvek más nem-determinisztikus bonyolultsági osztályokban is. A *nem-determinisztikus exponenciális idő* (NEXPTIME) osztály úgy definiálható, mint az NTIME(2^{n^c}) osztályok uniója minden $c > 0$-ra. Egy példát Ramsey tételével kapcsolatban fogalmazhatunk meg. Legyen G egy gráf; a G-hez tartozó $R(G)$ *Ramsey-szám* az a legkisebb $N > 0$, melyre fennáll, hogy akárhogyan is színezzük az N csúcsú teljes gráf éleit két színnel, valamelyik szín tartalmazza G-nek egy példányát. Álljon $\mathcal{L}$ azokból a (G, N) párokból, melyekre $R(G) > N$. A (G, N) bemenet mérete (ha G-t mondjuk adjacencia-mátrixa írja le) $O(|V(G)|^2 + \log N)$.

Mármost $\mathcal{L} \in$ NEXPTIME, mert annak, hogy $(G, N) \in \mathcal{L}$, tanúja a G éleinek egy 2 színnel színezése, melyben nincs egyszínű G, és ez a tulajdonság $O(N^{|V(G)|})$ időben ellenőrizhető, ami exponenciális a bemenet méretében (de nem rosszabb). Másrészről determinisztikusan nem tudunk jobb algoritmust $(G, N) \in \mathcal{L}$ eldöntésére, mint duplán exponenciálist. A triviális algoritmus, aminél lényegesen jobb sajnos nem ismeretes, végignézi az N csúcsú teljes gráf éleinek minden 2 színnel való színezését, és ezek száma $2^{N(N-1)/2}$.

4.4. NP-teljesség

Azt mondjuk, hogy az $\mathcal{L}_1 \subseteq \Sigma_1^*$ nyelv *polinomiálisan visszavezethető* az $\mathcal{L}_2 \subseteq \Sigma_2^*$ nyelvre, ha van olyan polinomiális időben kiszámítható $f : \Sigma_1^* \to \Sigma_2^*$ függvény, hogy minden $x \in \Sigma_1^*$ szóra

$$x \in \mathcal{L}_1 \iff f(x) \in \mathcal{L}_2.$$

Közvetlenül ellenőrizhető a definíció alapján, hogy ez a reláció tranzitív:

4.4.1 Állítás: *Ha $\mathcal{L}_1$ polinomiálisan visszavezethető $\mathcal{L}_2$-re, és $\mathcal{L}_2$ polinomiálisan visszavezethető $\mathcal{L}_3$-ra, akkor $\mathcal{L}_1$ is polinomiálisan visszavezethető $\mathcal{L}_3$-ra.* □

Azt, hogy egy nyelv P-ben van, úgy is kifejezhetjük, hogy polinomiálisan visszavezethető az {1} nyelvre. Valamint megfogalmazhatjuk az alábbit:

4.4.2 Állítás: *Ha egy nyelv P-ben van, akkor minden rá polinomiálisan visszavezethető nyelv is P-ben van. Ha egy nyelv NP-ben van, akkor minden rá polinomiálisan visszavezethető nyelv is NP-ben van.* □

Egy NP-beli $\mathcal{L}$ nyelvet **NP-teljesnek** nevezünk, ha minden NP-beli nyelv polinomiálisan visszavezethető $\mathcal{L}$-re. Ezek tehát a „legnehezebb" NP-beli nyelvek. Ha egyetlen NP-teljes nyelvről meg tudnánk mutatni, hogy P-ben van, akkor következne, hogy P=NP. Nyilvánvaló az alábbi észrevétel is:

4.4.3 Állítás: *Ha egy NP-teljes $\mathcal{L}_1$ nyelv polinomiálisan visszavezethető egy $\mathcal{L}_2 \in$NP nyelvre, akkor $\mathcal{L}_2$ is NP-teljes.* □

Egyáltalán nem nyilvánvaló, hogy létezik NP-teljes nyelv. Első célunk, hogy megadjunk egy NP-teljes nyelvet; utána (ennek más nyelvekre való polinomiális visszavezetésével, a 4.4.3 állítás alapján) sok más problémáról is bebizonyítjuk, hogy NP-teljes.

Egy Boole-polinomot *kielégíthetőnek* nevezünk, ha az általa definiált Boole-függvény nem azonosan 0. A *Kielégíthetőség Probléma* az, hogy adott f Boole-polinomról döntsük el, hogy kielégíthető-e. A problémát általában abban az esetben tekintjük, amikor a Boole-polinom egy konjunktív normálformával van megadva.

Feladatok. 1. Mikor kielégíthető egy diszjunktív normálforma?

2. Adott egy G gráf és három szín, 1, 2 és 3. Minden v csúcshoz és i színhez vezessünk be egy $x[v,i]$ logikai értéket. Írjunk föl olyan B konjunktív normálformát az $x[v,i]$ változókra, mely akkor és csak akkor igaz, ha

a) van olyan színezése a csúcsoknak az adott 3 színnel, hogy $x[v,i]$ akkor és csak akkor igaz, ha v színe i;

b) az előző színezésről még azt is megköveteljük, hogy szomszédos csúcsok színe különböző legyen.

c) Írjunk föl olyan konjunktív normálformát, mely akkor és csak akkor kielégíthető, ha a G gráf 3 színnel színezhető.

Minden konjunktív normálformát úgy is tekinthetünk, mint az „x", „0", „1", „(", „)", „$\neg$" „$\wedge$" és „$\vee$" jelekből álló ábécé fölötti szót: a változók indexeit kettes számrendszerben írjuk föl, pl. az $(x_1 \wedge \overline{x}_3 \wedge x_6) \vee (\overline{x}_1 \wedge x_2)$ konjunktív normálformának a következő szó felel meg: $(x1 \wedge \neg x11 \wedge x110) \vee (\neg x1 \wedge x10)$. Jelölje SAT a kielégíthető konjunktív normálformák által alkotott nyelvet.

4.4.4 Cook Tétele: *A* SAT *nyelv* NP*-teljes.*

Bizonyítás: Legyen $\mathcal{L}$ tetszőleges NP-beli nyelv. Ekkor létezik olyan $T = \langle k, \Sigma, \Gamma, \alpha, \beta, \gamma \rangle$ nem-determinisztikus Turing-gép és léteznek olyan $c, c_1 > 0$ egész számok, hogy T $\mathcal{L}$-et $c_1 \cdot n^c$ időben fölismeri. Feltehetjük, hogy $k = 1$. Tekintsünk egy tetszőleges $h_1 \dots h_n \in \Sigma_0^*$ szót. Legyen $N = \lceil c_1 \cdot n^c \rceil$. Vezessük be az alábbi változókat:

$$\begin{array}{ll} x[t,g] & (0 \le t \le N,\ g \in \Gamma), \\ y[t,p] & (0 \le t \le N,\ -N \le p \le N), \\ z[t,p,h] & (0 \le t \le N, -N \le p \le N, h \in \Sigma). \end{array}$$

Ha adott a T gép egy legális számolása, akkor adjuk ezeknek a változóknak a következő értéket: $x[t,g]$ igaz, ha a t-edik lépés után a vezérlőegység a g állapotban van; $y[t,p]$ igaz, ha a t-edik lépés után a fej a szalag p-edik mezején tartózkodik; $z[t,p,h]$ igaz, ha a t-edik lépés után a szalag p-edik mezején a h jel áll. Nyilvánvaló, hogy az x, y, z változók a Turing-gép számolását egyértelműen meghatározzák (nem fog azonban a változók minden lehetséges értéke a Turing-gép egy számolásának megfelelni).

Könnyen fölírhatók azonban olyan logikai összefüggések e változók között, amelyek együttvéve azt fejezik ki, hogy ez egy olyan legális számolás, mely $h_1 \dots h_n$-et elfogadja. Föl kell írni, hogy minden lépésben a vezérlőegység valamelyik állapotban van:

$$\bigvee_{g \in \Gamma} x[t,g] \qquad (0 \le t \le N);$$

és nincsen két állapotban:

$$\overline{x}[t,g] \vee \overline{x}[t,g'] \qquad (g \ne g' \in \Gamma;\ 0 \le t \le N).$$

Hasonlóan fölírhatjuk, hogy a fej minden lépésben egy és csakis egy helyen van, és a szalag minden mezején egy és csakis egy jel áll. Fölírjuk, hogy a gép kezdetben a START, a számolás végén a STOP állapotban van, és a fej a 0 mezőről indul:

$$x[0, START] = 1, \qquad x[N, STOP] = 1, \qquad y[0,0] = 1.$$

és hasonlóan, hogy kezdetben a szalagon a $h_1 \dots h_n$ bemenet, a végén a 0 mezőn az 1 jel áll:

$$z[0, i-1, h_i] = 1 \qquad (1 \le i \le n)$$

$$z[0, i-1, *] = 1 \qquad (i < 0 \text{ vagy } i > n)$$

$$z[N, 0, 1] = 1.$$

Ki kell továbbá fejeznünk a gép számolási szabályait, vagyis, hogy minden $g, g' \in \Gamma$, $h, h' \in \Sigma$, $\varepsilon \in \{-1, 0, 1\}$, $-N \le p \le N$ és $0 \le t < N$ esetén, ha $(g, h, g', h', \varepsilon) \notin \Phi$, akkor

$$(\overline{x[t,g]} \vee \overline{y[t,p]} \vee \overline{z[t,p,h]} \vee \overline{x[t+1,g']} \vee \overline{y[t+1,p+\varepsilon]} \vee \overline{z[t+1,p,h']}),$$

és hogy ahol nem áll a fej, ott a szalag tartalma nem változik:

$$(y[t,p] \vee z[t,p,h] \vee \overline{z[t+1,p,h]}).$$

Ezeket az összefüggéseket $\wedge$ jellel összekapcsolva olyan konjunktív normálformát kapunk, mely akkor és csak akkor elégíthető ki, ha a T Turing gépnek van legfeljebb N idejű

$h_1 \dots h_n$-et elfogadó számolása. Könnyű ellenőrizni, hogy a leírt konstrukció adott $h_1 \dots h_n$ esetén polinomiális időben elvégezhető. □

Kényelmes lesz a továbbiakban, ha a kielégíthetőség probléma két speciális esetére is megmutatjuk, hogy NP-teljes. Nevezzük röviden *k-formának* a konjunktív k-normálformát, vagyis az olyan konjunktív normálformát, melynek minden tényezőjében legfeljebb k literál fordul elő. Jelölje k-SAT a kielégíthető k-formák által alkotott nyelvet. Jelölje továbbá SAT-k azt a nyelvet, mely azon kielégíthető konjunktív normálformákból áll, melyekben minden változó legfeljebb k elemi diszjunkcióban fordul elő.

4.4.5 Tétel. *A* 3-SAT *nyelv* NP*-teljes.*

Bizonyítás: Új vátozók bevezetésével a SAT nyelvet visszavezetjük a 3-SAT nyelvre. Legyen adva egy B konjunktív normálforma, és tekintsük ennek egy elemi diszjunkcióját. Ez fölírható $E = z_1 \vee \ldots \vee z_k$ alakban, ahol minden z_i egy literál. Vezessünk be be k új változót, $y_1^E, \dots y_k^E$-t, és írjuk föl a

$$y_1^E \vee \overline{z_1}, \qquad \overline{y}_1^E \vee z_1$$

és

$$y_i^E \vee \overline{z}_i, \qquad y_i^E \vee \overline{y}_{i-1}^E, \qquad \overline{y}_i^E \vee y_{i-1}^E \vee z_i \qquad (i = 2, \dots k)$$

elemi diszjunkciókat (ezek azt „fejezik ki”, hogy $y_1^E = z_1$ és $y_i^E = y_{i-1}^E \vee z_i$, vagyis hogy $y_i^E = z_1 \vee \ldots \vee z_i$). Ezeket és az y_k^E egytagú elemi diszjunkciókat minden E-re $\wedge$ jelekkel egymáshoz fűzve egy olyan 3-normálformát kapunk, mely akkor és csak akkor elégíthető ki, ha a kiindulásul vett B konjunktív normálforma kielégíthető. □

Természetesen vetődik föl a kérdés, hogy miért éppen a 3-SAT problémát tekintettük. A 4-SAT, 5-SAT stb. problémák nehezebbek, mint a 3-SAT, így természetesen ezek is NP-teljesek. Másrészt azonban az alábbi tétel mutatja, hogy a 2-SAT probléma már nem NP-teljes (legalábbis, ha P≠NP). (Azt is illusztrálja ez, hogy gyakran a feladatok feltételeinek kis módosítása polinomiálisan megoldható feladatból NP-teljes feladathoz vezet.)

4.4.6 Tétel. *A* 2-SAT *nyelv* P*-ben van.*

Bizonyítás: Legyen B egy 2-normálforma az $x_1, \dots, x_n$ változókon. Konstruáljunk meg egy G irányított gráfot a következőképpen. Csúcsai legyenek az összes literálok, és kössük össze a z_i literált a z_j literállal, ha $\overline{z_i} \vee z_j$ egy elemi diszjunkció B-ben. Vegyük észre, hogy ekkor ebben a gráfban $\overline{z_j}$-ből is vezet él $\overline{z_i}$-be.

Tekintsük ennek az irányított gráfnak az *erősen összefüggő komponenseit*; ezek azok a pontosztályok, melyeket úgy kapunk, hogy két pontot egy osztályba sorolunk, ha köztük mindkét irányban vezet irányított út.

4.4.7 Lemma: *A B formula akkor és csak akkor kielégíthető, ha G egyetlen erősen összefüggő komponense sem tartalmaz egy változót és a negáltját.*

Ebből a Lemmából a tétel állítása már következik, mert egy irányított gráf erősen összefüggő komponensei könnyen megkereshetők polinomiális időben.

A lemma bizonyításához először is jegyezzük meg, hogy ha egy értékadás a B formulát kielégíti, és ebben az értékadásban egy z_i literál „igaz”, akkor minden olyan z_j literál is „igaz”, melybe z_i-ből él vezet: ellenkező esetben az $\overline{z_i} \vee z_j$ elemi diszjunkció nem volna kielégítve. Ebből következik, hogy egy erősen összefüggő komponens pontjai vagy mind „igazak”, vagy egy sem. De ekkor nem szerepelhet egy komponensben egy változó és a negáltja.

Megfordítva, tegyük föl, hogy semelyik erősen összefüggő komponens sem tartalmaz együtt egy változót és a negáltját. Tekintsünk egy x_i változót. A feltétel szerint x_i és $\overline{x}_i$ között nem vezethet mindkét irányban irányított út. Tegyük föl, hogy egyik irányban sem vezet. Ekkor húzzunk egy új élt x_i-ből $\overline{x}_i$-ba. Ettől nem sérül meg az a feltevésünk, hogy egyetlen erősen összefüggő komponens sem tartalmaz egy pontot a negáltjával együtt. Ugyanis ha létrejönne egy ilyen erősen összefüggő komponens, akkor ennek tartalmaznia kellene az új élt, de ekkor x_i és $\overline{x}_i$ is ebbe a komponensbe tartoznának, és ezért lenne a gráfban $\overline{x}_i$-ból x_i-be vezető irányított út. De ekkor ez az út az eredeti gráfban is megvolna, ami lehetetlen.

Ezt az eljárást ismételve behúzhatunk tehát új éleket (méghozzá egy-egy változóból a negáltjához) úgy, hogy a kapott gráfban minden változó és a negáltja között pontosan egy irányban vezet irányított út. Legyen mármost $x_i = 1$ akkor és csak akkor, ha $\overline{x}_i$-ból x_i-be vezet irányított út. Azt állítjuk, hogy ez az értékadás minden elemi diszjunkciót kielégít. Tekintsünk ugyanis egy elemi diszjunkciót, mondjuk $z_i \vee z_j$-t. Ha ennek mindkét tagja hamis volna, akkor – definíció szerint – vezetne irányított út z_i-ből $\overline{z_i}$-ba és z_j-ből $\overline{z_j}$-ba. Továbbá a gráf definíciója szerint, $\overline{z_i}$-ból él vezet z_j-be és $\overline{z_j}$-ból él vezet z_i-be. Ekkor viszont z_i és $\overline{z_i}$ egy összefüggő komponensben vannak, ami ellentmondás. □

4.4.8 Tétel. *A* SAT-3 *nyelv* NP*-teljes.*

Bizonyítás: Legyen $B = E_1 \wedge E_2 \wedge \ldots \wedge E_m$ tetszőleges konjunktív normálforma. Feltehetjük, hogy az elemi diszjunkciókban csak különböző változók vannak. Helyettesítsük az E_i diszjunkcióban szereplő x_j változót egy új y_j^i változóval, és vegyük hozzá az így kapott kifejezéshez új elemi diszjunkcióként a következőket minden j-re:

$$y_j^1 \vee \overline{y_j^2},\ y_j^2 \vee \overline{y_j^3}, \ldots,\ y_j^{m-1} \vee \overline{y_j^m},\ y_j^m \vee \overline{y_j^1}.$$

Nyilvánvaló, hogy így olyan konjunktív normálformát kapunk, melyben minden változó legfeljebb háromszor fordul elő, és mely akkor és csak akkor elégíthető ki, ha B kielégíthető. □

Feladatok. 3. Definiáljuk a 3-SAT-3 nyelvet, és bizonyítsuk be, hogy NP-teljes.
4. Mutassuk meg, hogy a SAT-2 nyelv P-ben van.

4.5. További NP-teljes problémák

A továbbiakban különböző fontos nyelvekről fogjuk megmutatni, hogy NP-teljesek. Ezek többsége nem logikai jellegű, hanem „mindennapos” kombinatorikai, algebrai stb. problé-

mákat írnak le. Ha egy $\mathcal{L}$ nyelvről megmutatjuk, hogy NP-teljes, akkor következik, hogy $\mathcal{L}$ csak akkor lehet P-ben, ha P=NP. Bár ez az utóbbi egyenlőség nincs megcáfolva, eléggé általánosan elfogadott az a hipotézis, hogy nem áll. Ezért egy nyelv NP-teljességét úgy tekinthetjük, mint annak bizonyítékát, hogy az nem dönthető el polinomiális időben.

Fogalmazzunk meg egy alapvető kombinatorikai feladatot:

Lefogási feladat: Adott egy véges S halmaz részhalmazainak egy $\{A_1, \ldots, A_m\}$ rendszere, és egy k természetes szám. Van-e olyan legfeljebb k elemű részhalmaza S-nek, mely minden A_i-t metsz?

4.5.1 Tétel. *A* Lefogási feladat NP-*teljes*

Bizonyítás: Visszavezetjük a 3-SAT-ot erre a problémára. Adott B konjunktív 3-normálformához megkonstruálunk egy halmazrendszert a következőképpen: az alaphalmaz legyen a B-beli változójelek és negáltjaik halmaza: $\{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$. B minden tényezőjéhez tekintsük a benne fellépő változójelek és negált változójelek halmazát, és ezenkívül az $\{x_i, \overline{x}_i\}$ halmazokat. Ennek a halmazrendszernek az elemei akkor és csak akkor foghatók le legfeljebb n ponttal, ha a normálforma kielégíthető. □

A lefogási feladat NP-teljes marad akkor is, ha a halmazrendszerre különböző megszorításokat teszünk. A fenti konstrukcióból látható, hogy a lefogási feladat már olyan halmazrendszerre is NP-teljes, mely legfeljebb háromelemű halmazokból áll. (Látni fogjuk kicsit később, hogy azt is elérhetjük, hogy csak kételemű halmazok – vagyis egy gráf élei – szerepeljenek.) Ha a SAT nyelvet először a SAT-3 nyelvre vezetjük vissza a 4.4.8 tétel szerint, és erre alkalmazzuk a fenti konstrukciót, olyan halmazrendszert kapunk, melyre az alaphalmaz minden eleme legfeljebb 4 halmazban van benne. Kissé bonyolultabban visszavezethetnénk a feladatot olyan halmazrendszer lefogására is, melyben minden elem legfeljebb 3 halmazban van benne. Ennél tovább már nem mehetünk: ha minden elem legfeljebb 2 halmazban van benne, akkor a halmazlefogási probléma polinomiális időben megoldható (lásd a 3. feladatot).

A lefogási feladattal könnyen láthatóan ekvivalens az alábbi probléma (csak az „elemek” és „részhalmazok” szerepét kell fölcserélni):

Lefedési feladat: Adott egy véges S halmaz részhalmazainak egy $\{A_1, ..., A_m\}$ rendszere és egy k természetes szám. Kiválasztható-e k halmaz úgy, hogy egyesítésük az egész S halmaz legyen?

A fentiek szerint ez már akkor is NP-teljes, ha az adott részhalmazok mindegyike legfeljebb 4 elemű.

További fontos feladat halmazrendszerekre az alábbi feladatpár:

k-Partíció feladat: Adott egy véges S halmaz részhalmazainak egy $\{A_1, ..., A_m\}$ rendszere és egy k természetes szám. Kiválasztható-e olyan $\{A_{i_1}, ..., A_{i_k}\}$ részrendszer, mely az alaphalmaz egy partícióját adja (vagyis diszjunkt halmazokból áll, és egyesítése az egész S halmaz)?

Partíció feladat: Adott egy véges S halmaz részhalmazainak egy $\{A_1, ..., A_m\}$ rendszere. Kiválasztható-e olyan $\{A_{i_1}, ..., A_{i_k}\}$ részrendszer, mely az alaphalmaz egy partícióját adja?

4.5.2 Tétel: *A* k*-*Partíció feladat *és a* Partíció feladat *NP-teljes.*

Bizonyítás: A legfeljebb 4 elemű halmazokkal való fedés problémáját (melyről már tudjuk, hogy NP-teljes) vezetjük vissza a k-Partíció problémára. Adott tehát egy véges S halmaz legfeljebb 4 elemű részhalmazainak egy rendszere és egy k természetes szám. El akarjuk dönteni, hogy kiválasztható-e k darab az adott halmazok közül úgy, hogy egyesítésük az egész S legyen. Csapjuk hozzá a rendszerhez az adott halmazok összes részhalmazait (itt használjuk ki, hogy az adott halmazok korlátosak: ettől a halmazok száma legfeljebb $2^4 = 16$-szorosára nő). Nyilvánvaló, hogy ha az eredeti rendszerből k darab lefedi S-et, akkor a bővített rendszerből alkalmas k darab S-nek egy partícióját adja, és viszont. Ezzel beláttuk, hogy a k-partíció feladat NP-teljes.

Másodszorra a k-Partíció feladatot a Partíció feladatra vezetjük vissza. Legyen U egy S-től diszjunkt k elemű halmaz. Új alaphalmazunk legyen $S \cup U$, a halmazrendszer halmazai pedig legyenek az $A_i \cup \{u\}$ alakú halmazok, ahol $u \in U$. Nyilvánvaló, hogy ha ebből az új halmazrendszerből kiválaszthatók az alaphalmaz egy partícióját alkotó halmazok, akkor ezek száma k, és S-be eső részeik S-nek egy partícióját adják k halmazra. Megfordítva, S minden k darab A_i halmazra történő partíciója az $S \cup U$ halmaznak az új halmazrendszerbeli halmazokra történő partícióját szolgáltatja. így a Partíció feladat is NP-teljes. □

Ha az adott halmazok kételeműek, akkor a Partíció feladat éppen a teljes párosítás létezésének problémája, és így polinomiális időben megoldható. De megmutatható, hogy már 3 elemű halmazokra a Partíció feladat NP-teljes.

Most egy alapvető gráfelméleti feladattal, a színezési problémával foglalkozunk. Említettük, hogy ez a feladat polinomiális időben megoldható, ha két színünk van. Ezzel szemben:

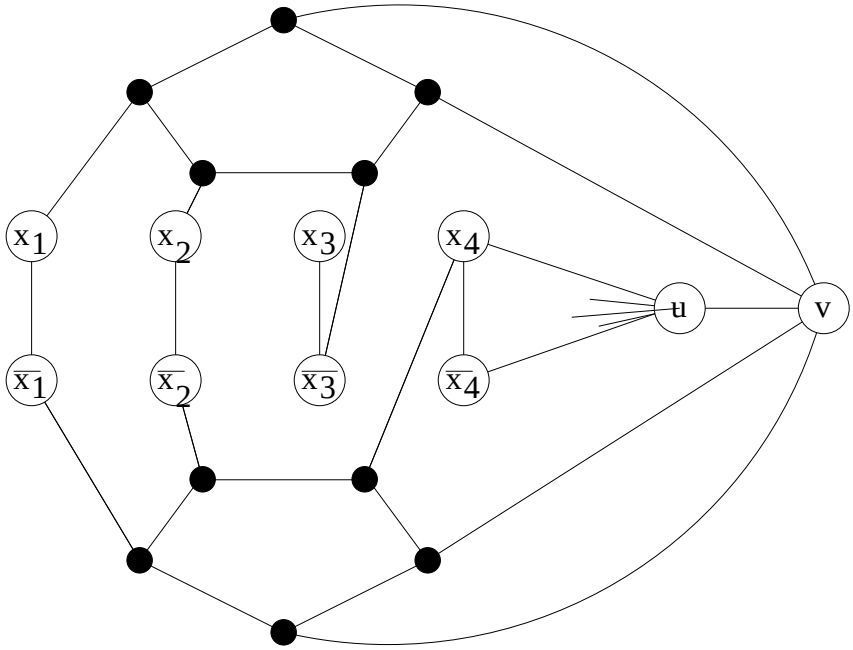
Látható, hogy a lefogási feladat már olyan halmazrendszerre is NP-teljes, mely legfeljebb háromelemű halmazokból áll. Látni fogjuk, hogy azt is elérhetjük, hogy csak kételemű halmazok (vagyis egy gráf élei) szerepeljenek. Előbb azonban egy másik alapvető gráfelméleti feladattal, a színezési problémával foglalkozunk. Említettük, hogy ez a feladat polinomiális időben megoldható, ha két színünk van. Ezzel szemben:

4.5.2 Tétel. *Gráfok* 3 *színnel való színezhetősége* NP*-teljes probléma.*

Bizonyítás: Legyen adva egy B konjunktív 3-forma; megkonstruálunk hozzá egy G gráfot, mely akkorés csak akkor színezhető ki három színnel, ha B kielégíthető.

A G gráf pontjai közé először is felvesszük a literálokat, és minden változót összekötünk a negáltjával. Felveszünk még két pontot: u-t és v-t, és összekötjük őket egymással, valamint u-t összekötjük az összes negálatlan és negált változóval. Végül, minden $z_{i_1} \vee z_{i_2} \vee z_{i_3}$ elemi diszjunkcióhoz felveszünk még egy-egy ötszöget; ennek két szomszédos csúcsát v-vel kötjük össze, a másik három csúcsát pedig rendre z_{i_1}-gyel, z_{i_2}-vel és z_{i_3}-mal. Azt állítjuk, hogy az így megkonstruált G gráf akkor és csakis akkor színezhető ki három színnel, ha B kielégíthető (4.1 ábra).

A bizonyításban kulcsszerepet játszik az alábbi könnyen belátható észrevétel: ha valamely $z_{i_1} \vee z_{i_2} \vee z_{i_3}$ elemi diszjunkcióra a z_{i_1}, z_{i_2}, z_{i_3} és v pontok ki vannak színezve három



4.1. ábra. Visszavezetési konstrukció pl. az $(\overline{x}_1 \vee \overline{x}_2 \vee x_4) \wedge (x_1 \vee x_2 \vee \overline{x}_3)$ formulához

színnel, akkor ez a színezés akkor és csakis akkor terjeszthető ki a megfelelő ötszögre legális színezésként, ha a négy pont színe nem azonos.

Tegyük föl először is, hogy B kielégíthető, és tekintsünk egy megfelelő értékadást. Színezzük pirosra azokat a literálokat, melyek „igazak", és kékre a többit. Színezzük u-t sárgára, v-t pedig kékre. Mivel minden elemi diszjunkcióban kell, hogy legyen egy piros pont, ez a színezés kiterjeszthető az ötszögek pontjaira legális színezésként.

Megfordítva, tegyük föl, hogy a G gráf három színnel színezhető, és tekintsük egy „legális" színezését pirossal, kékkel és sárgával. Feltehetjük, hogy a v pont kék, az u pont pedig sárga. Ekkor a literáloknak megfelelő pontok csak kékek és pirosak lehetnek, és minden változó és a negáltja közül az egyik piros, a másik kék. Abból, hogy az ötszögek is ki vannak színezve, következik, hogy minden elemi diszjunkcióban van egy piros pont. De ez azt is jelenti, hogy „igaznak" véve a piros pontokat, egy olyan értékadást kapunk, mely B-t kielégíti. □

Könnyen következik az előző tételből, hogy bármely $k \geq 3$ számra a gráfok k színnel való színezhetősége is NP-teljes.

A 4.5.1 tétel bizonyításánál megkonstruált halmazrendszerben legfeljebb három elemű halmazok voltak, éspedig azért, mert a 3-SAT problémát vezettük vissza lefogási feladatra. Mivel a 2-SAT probléma P-ben van, azt várhatnánk, hogy kételemű halmazokra a lefogási probléma is P-ben van. Megjegyezzük, hogy ez a speciális eset különösen érdekes, mert itt gráfok éleinek lefogásáról van szó. Észrevehetjük, hogy egy lefogó ponthalmazból kimaradó pontok függetlenek (nem megy köztük él), és megfordítva. Ezért minimális

lefogó halmaz helyett maximális független halmazt is kereshetünk, ami szintén alapvető gráfelméleti feladat. Igen-nem kérdésként megfogalmazva:

Független ponthalmaz feladat: *Adott egy G gráf és egy k természetes szám, van-e G-ben k független pont?*

Sajnos azonban ez a probléma sem könnyebb lényegesen, mint az általános lefogási feladat:

4.5.3 Tétel: *A* Független ponthalmaz feladat NP*-teljes.*

Bizonyítás: Visszavezetjük rá a 3 színnel való színezhetőség problémáját. Legyen G tetszőleges n pontú gráf, és konstruáljuk meg a H gráfot a következőképpen: Vegyük G-nek három diszjunkt példányát, G_1-et, G_2-t és G_3-at, és kössük össze a három példány egymásnak megfelelő pontjait. Legyen H a kapott gráf, ennek tehát $3n$ pontja van.

Állítjuk, hogy a H gráfban akkor és csak akkor van n független pont, ha G három színnel színezhető. Valóban, ha G három színnel, mondjuk pirossal, kékkel és sárgával kiszínezhető, akkor a piros pontoknak megfelelő G_1-beli, a kék pontoknak megfelelő G_2-beli, és a sárga pontoknak megfelelő G_3-beli pontok együttvéve is függetlenek a H gráfban, és számuk éppen n. A megfordítás ugyanígy látható be. □

Megjegyzés: A független ponthalmaz probléma (és ugyanígy a halmazrendszer lefogásának feladata) csak akkor NP-teljes, ha a k szám is része a bemenetnek. Nyilvánvaló ugyanis, hogy ha k-t rögzítjük (pl. k=137), akkor egy n pontú gráfra polinomiális időben (az adott példában $O(n^{137})$ időben) eldönthető, hogy van-e k független pontja. Más a helyzet a színezhetőséggel, ahol már a 3 színnel való színezhetőség is NP-teljes.

Feladatok. 5. Igazoljuk, hogy annak eldöntése is NP-teljes, hogy egy adott $2n$ pontú gráfban van-e n pontú független ponthalmaz.

6. Igazoljuk, hogy annak eldöntése is NP-teljes, hogy egy G gráf kromatikus száma egyenlő a legnagyobb teljes részgráfjának pontszámával.

7. Mutassuk meg, hogy ha egy halmazrendszer olyan, hogy az alaphalmaz minden eleme legfeljebb 2 halmazban van benne, akkor a rá vonatkozó Lefogási feladat polinomiálisan visszavezethető a párosítás-problémára.

8. Igazoljuk, hogy „hipergráfokra" már a 2 színnel való színezhetőség is NP-teljes: Adott egy véges S alaphalmaz és részhalmazainak egy $\{A_1, \ldots, A_m\}$ rendszere. Kiszínezhető-e S 2 színnel úgy, hogy minden A_i mindkét színű pontot tartalmazzon?

Igen sok más fontos kombinatorikai és gráfelméleti probléma is NP-teljes: Hamilton kör létezése, a pontok diszjunkt háromszögekkel való lefedhetősége (2-szögekre ez a párosítás-probléma!), adott pontpárokat összekötő pontdiszjunkt utak létezése stb. Garey és Johnson (1979) könyve százával sorol föl NP-teljes problémákat.

A kombinatorikán kívül is számos NP-teljes probléma ismert. Ezek közül talán a legfontosabb a következő:

Diophantoszi egyenlőlenségrendszer. Adott egy $Ax \leq b$ egész együtthatós lineáris egyenlőtlenségrendszer, el akarjuk dönteni, hogy van-e megoldása egész számokban? (A matematikában a „diophantoszi" jelző arra utal, hogy egész számokban keressük a megol-

dást.)

4.5.5 Tétel: *A* Diophantoszi egyenlőlenségrendszer *megoldhatósága* NP*-teljes probléma.*

Bizonyítás: Legyen adva egy B 3-forma az $x_1, \ldots x_n$ változókon. Írjuk föl az alábbi egyenlőtlenségeket:

$0 \leq x_i \leq 1$	minden i-re,
$x_{i_1} + x_{i_2} + x_{i_3} \geq 1$	minden $x_{i_1} \vee x_{i_2} \vee x_{i_3}$ elemi diszjunkcióra,
$x_{i_1} + x_{i_2} + (1 - x_{i_3}) \geq 1$	minden $x_{i_1} \vee x_{i_2} \vee \overline{x}_{i_3}$ elemi diszjunkcióra,
$x_{i_1} + (1 - x_{i_2}) + (1 - x_{i_3}) \geq 1$	minden $x_{i_1} \vee \overline{x}_{i_2} \vee \overline{x}_{i_3}$ elemi diszjunkcióra,
$(1 - x_{i_1}) + (1 - x_{i_2}) + (1 - x_{i_3}) \geq 1$	minden $\overline{x}_{i_1} \vee \overline{x}_{i_2} \vee \overline{x}_{i_3}$ elemi diszjunkcióra.

Nyilvánvaló, hogy ennek a lineáris egyenlőtlenségrendszernek a megoldásai pontosan a B-t kielégítő értékadások, így a 3-SAT problémát visszavezettük a lineáris egyenlőtlenségrendszerek egész számokban való megoldhatóságának problémájára. □

A bizonyításból kitűnik, hogy lineáris egyenlőtlenségrendszerekre már a 0-1 értékű megoldás létezésének problémája is NP-teljes. Valójában még ennek egy igen speciális esete is NP-teljes. (Így a fenti bizonyítás fölösleges volt; mégis leírtuk azért, hogy lássuk, milyen természetesen lehet logikai feltételeket egész számokra vonatkozó egyenlőtlenségekké átfogalmazni. Az ilyen átfogalmazás az egész értékű lineáris programozás sok gyakorlati alkalmazásának az alapja.)

Részletösszeg probléma: Adottak az $a_1, \ldots, a_k$ és b természetes számok. Van-e az $\{a_1, \ldots, a_m\}$ halmaznak olyan részhalmaza, melynek az összege b?

4.5.6 Tétel: *A* Részletösszeg probléma *NP-teljes.*

Bizonyítás: A Partíció problémát vezetjük vissza a Részletösszeg problémára. Legyen $\{A_1, \ldots, A_m\}$ az $S = \{0, \ldots, n-1\}$ halmaz részhalmazainak egy rendszere, el akarjuk dönteni, hogy van-e olyan részrendszere, mely S-nek egy partícióját adja. Legyen $q = m + 1$, és rendeljük hozzá minden A_i halmazhoz az $a_i = \sum_{j \in A_i} q^j$ számot. Legyen továbbá $b = 1 + q + \ldots + q^{n-1}$. Azt állítjuk, hogy $A_{i_1} \cup \ldots \cup A_{i_k}$ akkor és csak akkor partíciója az S halmaznak, ha

$$a_{i_1} + \ldots + a_{i_k} = b.$$

A „csak akkor” triviális. Megfordítva, tegyük föl, hogy $a_{i_1} + \ldots + a_{i_k} = b$. Legyen c_j azon A_{i_r} halmazok száma, melyek a j elemet tartalmazzák ($0 \leq j \leq n-1$). Ekkor

$$a_{i_1} + \ldots + a_{i_k} = \sum_j d_j q^j.$$

Mivel b egyértelműen írható föl q alapú számrendszerben, következik, hogy $d_j = 1$, vagyis $A_{i_1} \cup \ldots \cup A_{i_k}$ partíciója S-nek. □

Ez az utóbbi probléma jó példa arra, hogy a számok kódolása jelentősen tudja befolyásolni az eredményeket. Tegyük föl ugyanis, hogy minden a_i szám úgy van megadva, hogy ez a_i bitet igényel (pl. egy a_i hosszú $1 \dots 1$ sorozattal). Ezt röviden úgy mondjuk, hogy *unáris* jelölést használunk. Ezzel a bemenet hossza természetesen megnő, így az algoritmusok lépésszáma viszonylag kisebb lesz.

4.5.7 Tétel: *Unáris jelölés esetén a részletösszeg probléma polinomiálisan megoldható.*

(A lineáris egyenlőtlenség egész számokban való megoldhatóságának általános problémája unáris jelölés esetén is NP-teljes; ezt a fenti visszavezetés mutatja.)

Bizonyítás: Minden $1 \leq p \leq m$-re meghatározzuk mindazon t természetes számok T_p halmazát, melyek előállnak $a_{i_1} + \ldots + a_{i_k}$ alakban, ahol $1 \leq i_1 < \ldots < i_k \leq p$. Ezt a következő triviális rekurzióval tehetjük meg:

$$T_0 = \{0\}, \qquad T_{p+1} = T_p \cup \{t + a_{p+1} : \ t \in T_p\}.$$

Ha T_m meg van határozva, akkor csak azt kell megnéznünk, hogy $b \in T_p$ teljesül-e.

Be kell látnunk, hogy ez az egyszerű algoritmus polinomiális. Ez azonnal adódik abból az észrevételből, hogy $T_p \subseteq \{0, \ldots, \sum_i a_i\}$, így a T_p halmazok mérete polinomiális a bemenet méretében, ami most $\sum_i a_i$. □

Megjegyzések: 1. NP-*nehéznek* nevezik az olyan f függvényt, mely nem szükségképpen van NP-ben, de melyre minden NP-beli probléma visszavezethető abban az értelemben, hogy ha az f függvény értékének a kiszámítását hozzávesszük a RAM utasításaihoz (s így egyetlen lépésnek tekintjük), akkor bármely NP-beli probléma polinomiális időben megoldható. Minden NP-teljes nyelv karakterisztikus függvénye NP-nehéz, de vannak olyan NP-nehéz karakterisztikus függvényű nyelvek is, melyek határozottan nehezebbek, mint bármely NP-beli probléma (pl. az $n \times n$-es GO táblán egy állásról annak eldöntése, hogy ki nyer). Van sok fontos NP-nehéz függvény, mely nem 0-1 értékű. Az operációkutatás igen sok diszkrét optimalizálási problémájáról derült ki, hogy NP-nehéz. Így NP-nehéz az utazóügynök-probléma (egy gráf minden éléhez egy „költség" van hozzárendelve, és keressünk minimális költségű Hamilton-kört), a Steiner-probléma (az előző feltételek mellett keressünk minimális költségű összefüggő részgráfot, mely adott csúcspontokat tartalmaz), a hátizsák-probléma, az ütemezési problémák nagy része stb. Sok leszámlálási probléma is NP-nehéz (pl. egy gráf teljes párosításai számának, Hamilton-körei számának, vagy legális színezései számának megállapítása).

2. Tapasztalati tény, hogy a legtöbb fölvetődő NP-beli problémáról vagy az derül ki, hogy NP-teljes, vagy az, hogy P-beli. Nem sikerült eddig sem P-be, sem az NP-teljesek közé elhelyezni az alábbi problémákat:

a) *Adott n természetes számnak van-e k-nál nem nagyobb osztója?*

b) *Két adott gráf izomorf-e?*

3. Ha egy problémáról kiderül, hogy NP-teljes, akkor nem remélhetjük, hogy olyan hatékony, polinomiális idejű algoritmust tudunk találni rá, mint pl. a párosítás-problémára. Mivel ilyen problémák gyakorlatilag igen fontosak lehetnek, nem adhatjuk föl őket egy ilyen

negatív eredmény miatt. Egy-egy NP-teljes probléma körül különböző típusú részeredmények tömege születik: speciális osztályok, melyekre polinomiálisan megoldható; exponenciális, de nem túl nagy bemenetekre jól használható algoritmusok; heurisztikák, melyek nem adnak pontos eredményt, de (bizonyíthatóan vagy tapasztalatilag) jó közelítést adnak.

Néha azonban a feladatnak éppen a bonyolultsága az, ami kihasználható: lásd a Kriptográfiáról szóló fejezetet.

5. fejezet

Randomizált algoritmusok

A 2. fejezetben idéztük a Church-tézist: minden „algoritmus” (a szó heurisztikus értelmében) megvalósítható pl. egy Turing-gépen. Ez azonban nem teljesen igaz: ha egy algoritmusban megengedjük a „forint feldobást”, vagyis véletlen szám generálását elemi lépésként, akkor bizonyíthatóan képesek leszünk olyan feladatok megoldására, melyekre a Turing-gép nem (ezt a következő fejezetben mutatjuk meg). Más esetekben a véletlen választás az algoritmusokat jelentősen meggyorsítja. Erre látunk példákat ebben a fejezetben.

Mivel így új, erősebb matematikai gépfogalmat nyerünk, ennek megfelelő bonyolultsági osztályok is bevezethetők. Ezek közül néhány legfontosabbat tárgyalunk a fejezet végén.

5.1. Polinomazonosság ellenőrzése

Legyen $f(x_1, ..., x_n)$ egy n-változós racionális együtthatós polinom, mely minden változójában legfeljebb k-adfokú. El szeretnénk dönteni, hogy f azonosan 0-e (mint n-változós függvény). A klasszikus algebrából tudjuk, hogy egy polinom akkor és csak akkor azonosan 0, ha zárójeleit felbontva, minden tag „kiesik”. Ez a kritérium azonban nem mindig alkalmazható jól. Például elképzelhető, hogy a polinom zárójeles alakban van adva, és a zárójelek fölbontása exponenciálisan sok taghoz vezet. Jó volna olyan polinomokra is mondani valamit, melyek megadásában az alapműveleteken kívül más műveletek, pl. determináns kiszámítása is szerepelhet.

Az alapgondolat az, hogy a változók helyébe véletlenszerűen választott számokat írunk, és kiszámítjuk a polinom értékét. Ha ez nem 0, akkor természetesen a polinom nem lehet azonosan 0. Ha a kiszámított érték 0, akkor lehet ugyan, hogy a polinom nem azonosan 0, az azonban igen kis valószínűségű, hogy „beletaláltunk” egy gyökébe; így ilyen esetben megállapíthatjuk, hogy a polinom azonosan 0; kicsi a valószínűsége, hogy tévedünk.

Ha a változóknak pl. a $[0, 1]$ intervallumban egyenletes eloszlás szerint választott valós értékeket tudnánk adni, akkor a hiba valószínűsége 0 volna. Valójában azonban diszkrét értékekkel kell számolnunk; ezért azt tesszük fel, hogy a változók értékei egy $[0, N]$ intervallum egész számai közül vannak választva egyenletes eloszlás szerint. Ekkor a tévedés valószínűsége már nem lesz 0, de „kicsi” lesz, ha N elég nagy. Erre vonatkozik a következő alapvető eredmény:

5.1.1 Schwartz Lemmája: Ha f nem azonosan 0 n-változós, minden változójában legfeljebb k-adfokú polinom, és a ξ_i $(i = 1, \ldots, n)$ értékek a $[0, N-1]$ intervallumban egyenletes

eloszlás szerint véletlenszerűen és egymástól függetlenül választott egész számok, akkor

$$\text{Prob}(f(\xi_1,\ldots,\xi_n)=0) \le \frac{kn}{N}.$$

Bizonyítás: Az állítást n szerinti teljes indukcióval bizonyítjuk be. $n=1$-re az állítás igaz, mert egy k-adfokú polinomnak legfeljebb k gyöke lehet. Legyen $n>1$, és rendezzük f-et x_1 hatványai szerint:

$$f = f_0 + f_1x_1 + f_2x^2 + \ldots f_tx_1^t,$$

ahol $f_0,\ldots,f_t$ az $x_2,\ldots,x_n$ változók polinomjai, f_t nem azonosan 0, és $t\le k$. Mármost

$$\text{Prob}(f(\xi_1,\ldots,\xi_n)=0) \le$$

$$\text{Prob}(f(\xi_1,\ldots,\xi_n)=0 \mid f_n(\xi_2,\ldots,x_n)=0)\text{Prob}(f_n(\xi_2,\ldots,\xi_n)=0)+$$

$$+\text{Prob}(f(\xi_1,\ldots,\xi_n)=0 \mid f_n(\xi_2,\ldots,\xi_n)\neq 0)\text{Prob}(f_n(\xi_2,\ldots,\xi_n)\neq 0)$$

$$\le \text{Prob}(f_n(\xi_2,\ldots,\xi_n)=0) + \text{Prob}(f(\xi_1,\ldots,\xi_n)=0 \mid f_n(\xi_2,\ldots,\xi_n)\neq 0).$$

Itt az első tagot az indukciós feltevésből tudjuk megbecsülni, a második tag pedig legfeljebb k/N (mivel ξ_1 független a $\xi_2,\ldots,\xi_n$ változóktól, így ha azokat úgy rögzítjük, hogy $f_n\neq 0$ és így f mint x_1 polinomja nem azonosan 0, akkor legfeljebb k/N annak a valószínűsége, hogy ξ_1 gyöke legyen). Így

$$\text{Prob}(f(\xi_1,\ldots,\xi_n)=0) \le \frac{k(n-1)}{N} + \frac{k}{n} \le \frac{kn}{N}.$$

□

Ezek alapján a polinom azonosan 0 voltának eldöntésére a következő véletlent használó, *randomizált* algoritmus adódik:

5.1.2 Algoritmus: *Számítsuk ki $f(\xi_1,\ldots,\xi_n)$-t a $[0,2kn]$ intervallumban egyenletes eloszlás szerint véletlenszerűen és egymástól függetlenül választott egész ξ_i értékekkel. Ha nem 0 értéket kapunk, megállunk: f nem azonosan 0. Ha 0 értéket kapunk, megismételjük a számítást. Ha 100-szor egymásután 0-t kapunk, megállunk, és azt mondjuk, hogy f azonosan 0.*

Ha f azonosan 0, ez az algoritmus ezt is állapítja meg. Ha f nem azonosan 0, akkor minden egyes iterációnál — Schwartz lemmája szerint — 1/2-nél kisebb annak a valószínűsége, hogy 0-t kapunk eredményül. 100 függetlenül megismételt kísérletnél 2^{-100}-nál kisebb annak a valószínűsége, hogy ez mindig bekövetkezik, vagyis, hogy az algoritmus hibásan azt állítja, hogy f azonosan 0.

Ahhoz, hogy az algoritmust valóban végre tudjuk hajtani, két dolog kell: egyrészt független véletlen számokat kell tudni generálni (ezt most feltesszük, hogy megvalósítható, éspedig a generálandó számok bitjeinek számában polinomiális időben), másrészt f-et ki kell tudni értékelni polinomiális időben (az input mérete az f „definíciójának” hossza; ilyen lehet pl. explicit zárójeles kifejezés, de más is, pl. determináns alak).

A módszer alkalmazására meglepő példaként egy párosítás-algoritmust mutatunk be. (A párosítás-problémát a 4. fejezetben már tárgyaltuk.) Legyen G páros gráf A és B színosztályokkal, $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_n\}$. Minden a_ib_j élhez rendeljünk hozzá egy x_{ij} változót. Konstruáljuk meg az $n \times n$-es $M = (m_{ij})$ mátrixot a következőképpen:

$$m_{ij} = \begin{cases} x_{ij}, & \text{ha } a_ib_j \in E(G); \\ 0, & \text{egyébként.} \end{cases}$$

Ennek a mátrixnak a determinánsa szoros kapcsolatban áll a G gráf párosításaival, mint azt Kőnig Dénes észrevette Frobenius egy munkáját elemezve (v.ö a 4.3.3 tétellel):

5.1.3 Tétel: *A G páros gráfban akkor és csak akkor van teljes párosítás, ha* $\det(M) \not\equiv 0$.

Bizonyítás: Tekintsük a determináns egy kifejtési tagját:

$$\pm m_{1\pi(1)} m_{2\pi(2)} \ldots m_{n\pi(n)},$$

ahol π az $1, \ldots, n$ számok egy permutációja. Hogy ez ne legyen 0, ahhoz az kell, hogy a_i és $b_{\pi(i)}$ minden i-re össze legyenek kötve; másszóval, hogy $\{a_1b_{\pi(1)}, \ldots a_nb_{\pi(n)}\}$ teljes párosítás legyen G-ben. Így ha G-ben nincs teljes párosítás, a determináns azonosan 0. Ha G-ben van teljes párosítás, akkor minden ilyennek megfelel egy-egy nem-0 kifejtési tag. Mivel ezek a tagok nem ejtik ki egymást (bármely kettő tartalmaz két különböző változót), a determináns nem azonosan 0. □

Mivel $\det(M)$ az M mátrix elemeinek olyan polinomja, mely polinomiális időben kiszámítható (pl. Gauss-eliminációval), ebből a tételből polinomiális idejű randomizált algoritmus adódik a párosításproblémára páros gráfokban. Azt korábban már említettük, hogy erre a problémára létezik polinomiális idejű determinisztikus algoritmus is (a „magyar módszer"). Az itt tárgyalt algoritmusnak az az egyik előnye, hogy igen egyszerűen programozható (determináns-számítás általában van a könyvtárban). Ha „gyors" mátrixszorzási eljárásokat alkalmazunk, ez a randomizált algoritmus aszimptotikusan kicsit gyorsabb, mint a leggyorsabb ismert determinisztikus: $O(n^{2.5})$ helyett $O(n^{2.4})$ idő alatt végrehajtható. Legfőbb érdeme azonban az, hogy jól párhuzamosítható, mint azt a következő fejezetben látni fogjuk.

Hasonló, de kicsit bonyolultabb módszerrel nem-páros gráfokban is eldönthető, hogy van-e teljes párosítás. Legyen $V = \{v_1, \ldots, v_n\}$ a G gráf ponthalmaza. Rendeljünk megint minden v_iv_j élhez (ahol $i < j$) egy x_{ij} változót, és konstruálunk meg egy antiszimmetrikus $n \times n$-es $T = (t_{ij})$ mátrixot a következőképpen:

$$t_{ij} = \begin{cases} x, & \text{ha } v_iv_j \in E(G), \text{ és } i < j; \\ -x, & \text{ha } v_iv_j \in E(G), \text{ és } i > j; \\ 0, & \text{egyébként.} \end{cases}$$

Tutte-tól származik a fent idézett Frobenius–Kőnig-féle tétel következő analogonja, melyet bizonyítás nélkül idézünk:

5.1.4 Tétel: *A G gráfban akkor és csak akkor van teljes párosítás, ha $det(T) \not\equiv 0$.* □

Ebből a tételből, a páros gráf esetéhez hasonlóan egy polinomiális idejű randomizált algoritmus adódik annak eldöntésére, hogy G-ben van-e teljes párosítás.

5.2. Prímtesztelés

Legyen m páratlan természetes szám, el akarjuk dönteni, hogy prím-e. Az előző fejezetben láttuk, hogy ez a probléma NP∩co-NP-ben van. Az ott leírt tanúk azonban (legalábbis egyelőre) nem vezettek polinomiális idejű prímteszthez. Ezért most először az összetettségnek egy új, bonyolultabb módon való NP-leírását (tanúsítását) adjuk meg.

Idézzük föl az ún. „Kis" Fermat Tételt: Ha m prím, akkor minden $1 \leq a \leq m-1$ természetes számra $a^{m-1}-1$ osztható m-mel. Ha — adott m mellett — egy a számra $a^{m-1}-1$ osztható m-mel, akkor azt mondjuk, hogy *a kielégíti a Fermat-feltételt.*

A minden $1 \leq a \leq m-1$ számra megkövetelt Fermat-feltétel jellemzi is prímeket, mert ha m összetett szám, és a-nak bármely olyan számot választunk, mely m-hez nem relatív prím, akkor $a^{m-1}-1$ nyilvánvalóan nem osztható m-mel. Természetesen nem tudjuk a Fermat-feltételt minden a-ra ellenőrizni; ez exponenciális időt igényelne. Kérdés tehát, hogy milyen a-kra alkalmazzuk? Vannak olyan m összetett számok (az ún. pszeudoprímek), melyekre a Fermat-feltétel minden m-hez relatív prím a számra teljesül; ezekre különösen nehéz lesz a feltételt megsértő a számot találni. (Ilyen pszeudoprím például az $561 = 3 \cdot 11 \cdot 17$.)

Idézzük föl, hogy egy m számmal azonos osztási maradékot adó egész számok halmazát *modulo m maradékosztálynak* nevezzük. A maradékosztály *primitív*, ha elemei m-hez relatív prímek (ez nyilván egy maradékosztály minden elemére egyszerre teljesül, mint ahogyan a Fermat-feltétel is).

Érdemes megjegyezni, hogy ha m nem pszeudoprím, akkor a modulo m primitív maradékosztályoknak legfeljebb a felére teljesül a Fermat-feltétel. (A nem primitív maradékosztályok egyikére sem). Az olyan a primitív maradékosztályok ugyanis, melyek a Fermat-feltételt kielégítik, a szorzásra nézve részcsoportot alkotnak. Ha ez a részcsoport valódi, akkor indexe legalább 2, így a primitív maradékosztályoknak legfeljebb a felét tartalmazza.

Így ha m nem pszeudoprím, akkor a következő egyszerű randomizált prímteszt működik: ellenőrizzük, hogy egy véletlenszerűen választott $1 \leq a \leq m-1$ szám kielégíti-e a „Kis" Fermat Tételt. Ha nem, tudjuk, hogy m nem prím. Ha igen, akkor ismételjük meg az eljárást. Ha 100-szor egymástól függetlenül azt találtuk, hogy a „Kis"-Fermat-tétel teljesül, akkor azt mondjuk, hogy m prím. Előfordulhat ugyan, hogy m összetett, de ha nem pszeudoprím, akkor minden lépésben $1/2$-nél kisebb volt annak a valószínűsége, hogy a feltételt kielégítő a-t találtunk, és így 2^{-100}-nál kisebb annak a valószínűsége, hogy egymásután 100-szor ez bekövetkezzen.

Sajnos ez a módszer pszeudoprímekre csődöt mond (azokat nagy valószínűséggel prímeknek találja). Így a Fermat-feltételt kissé módosítjuk. Írjuk fel az $m-1$ számot $2^k M$ alakban, ahol M páratlan. Azt mondjuk, hogy egy a szám *kielégíti a Rabin-feltételt*, ha az

$$a^M - 1,\ a^M + 1,\ a^{2M} + 1,\ a^{4M} + 1, \ldots,\ a^{2^{k-1}M} + 1$$

számok egyike sem osztható m-mel. Mivel ezeknek a szorzata éppen $a^{m-1}-1$, minden olyan a szám, mely a Rabin-feltételt megsérti, megsérti a Fermat-feltételt is (de nem megfordítva, mert m lehet összetett, és így lehet osztója egy szorzatnak anélkül, hogy bármelyik tényezéjőnek is osztója volna).

5.2.1 Lemma: *Akkor és csak akkor elégíti ki minden $1 \leq a \leq m-1$ egész szám a Rabin-feltételt, ha m prímszám.*

Bizonyítás: I. Ha m összetett, akkor bármely valódi osztója megsérti a Rabin-feltételt.

II. Tegyük föl, hogy m prím. Ekkor a Fermat-feltétel szerint bármely $1 < a < m$ természetes számra $a^{m-1} - 1$ osztható m-mel. Ez a szám azonban szorzattá bontható:

$$a^{m-1} - 1 = (a^M - 1)(a^M + 1)(a^{2M} + 1)(a^{4m} + 1) \dots (a^{2^{k-1}M} - 1).$$

Így (ismét felhasználva, hogy m prím) ezen tényezők valamelyike is osztható kell, hogy legyen m-mel, vagyis a kielégíti a Rabin-feltételt. □

Az algoritmus kulcslépése az az eredmény, hogy - a Fermat-feltétellel ellentétben - a Rabin-feltételt összetett számokra a maradékosztályok többsége megsérti:

5.2.2 Lemma: *Ha m összetett szám, akkor modulo m a primitív maradékosztályoknak legalább fele megsérti a Rabin-feltételt.*

Bizonyítás: Mivel a lemma igazságát nem-pszeudoprímekre már beláttuk, így a továbbiakban feltehetjük, hogy m pszeudoprím. Legyen m prímfelbontása $p_1^{r_1} \dots, p_t^{r_t}$ $(t \geq 2)$. Állítjuk, hogy minden i-re $p_i - 1$ osztója $(m-1)$-nek. Tegyük föl, hogy ez nem áll, akkor $m - 1 = (p_i - 1)q + r$, ahol $1 \leq r < p_i - 1$. Mint az előző fejezetben már fölhasználtuk, van olyan p_i-vel nem osztható a szám, melyre $a^r - 1$ nem osztható p_i-vel. De ekkor $a^{m-1} = (a^{p_i-1})^q a^r \equiv a^r \not\equiv 1 (\mathrm{mod}\ p_i)$, és így $a^{m-1} - 1$ nem osztható p_i-vel, tehát m-mel sem. Ez ellentmond annak, hogy m pszeudoprím.

Legyen l az a legnagyobb kitevő a $0 \leq l \leq k$ intervallumban, melyre a $p_i - 1$ számok egyike sem osztója $2^l M$-nek. Mivel a $p_i - 1$ számok párosak, M pedig páratlan, $l \geq 1$. Mármost ha $l < k$, van olyan i, hogy $p_i - 1$ osztója $2^{l+1}M$-nek és ezért p_i osztója $a^{2^s M} - 1$-nek minden primitív a maradékosztályra és minden $l < s \leq k$-ra (ez triviálisan igaz akkor is, ha $l = k$). Ekkor viszont p_i nem lehet osztója $a^{2^s M} + 1$-nek és így m sem osztója $a^{2^s M} + 1$-nek. Tehát ha a olyan primitív maradékosztály, mely megsérti a Rabin-feltételt, akkor m osztója kell, hogy legyen az $a^M - 1$, $a^M + 1$, $a^{2M} + 1$, …, $a^{2^l M} + 1$ maradékosztályok valamelyikének. Ezért minden ilyen a-ra m vagy $(a^{2^l M} - 1)$-nek vagy $(a^{2^l M} + 1)$-nek osztója. Nevezzük a modulo m primitív a maradékosztályt „első fajtának”, $a^{2^l M} \equiv 1 (\mathrm{mod}\ m)$, és „második fajtának”, ha $a^{2^l M} \equiv -1 (\mathrm{mod}\ m)$.

Becsüljük meg először az első fajta maradékosztályok számát. Tekintsünk egy i, $1 \leq i \leq t$ indexet. Mivel $p_i - 1$ nem osztója a $2^l M$ kitevőnek, $2^l = (p_i - 1)q + r$, ahol $1 \leq r < p_i - 1$. Mint az előző fejezetben már fölhasználtuk, van olyan p_i-vel nem osztható c szám, melyre $c^r - 1$ nem osztható p_i-vel. De ekkor $c^{m-1} = (c^{p_i-1})^q c^r \equiv c^r \not\equiv 1 (\mathrm{mod}\ p_i)$, és így $c^{m-1} - 1$ nem osztható p_i-vel. Ugyancsak használtuk már azt a gondolatmenetet, hogy ekkor a modulo p_i primitív maradékosztályoknak legfeljebb a fele olyan, hogy $a^{m-1} - 1$ osztható p_i-vel. A „Kínai Maradéktétel” szerint kölcsönösen egyértelmű megfeleltetés van a $p_1 \dots p_t$ szorzatra mint modulusra vett primitív maradékosztályok és a $p_1, \dots, p_t$ prímekre vett primitív maradékosztály-t-esek között. Így modulo $p_1 \dots p_t$ a primitív maradékosztályoknak legfeljebb 2^t-ed része olyan, hogy mindegyik p_i osztója $(a^{2^l M} - 1)$-nek. Ezért a modulo m vett primitív maradékosztályoknak legfeljebb (2^t)- edrésze olyan, hogy m osztója $(a^{2^l M} - 1)$-nek.

Könnyű látni, hogy két második fajta maradékosztály szorzata első fajta. Így a második fajta maradékosztályokat egy rögzített maradékosztállyal végigszorozva különböző első

fajta maradékosztályokat kapunk, tehát a második fajta maradékosztályok száma legfeljebb akkora, mint az első fajtájúaké. Így a kettő együtt is a primitív maradékosztályoknak legfeljebb 2^{t-1}-edrészét, és így legfeljebb felét teszi ki. □

5.2.3 Lemma: Adott m-ről és a-ról polinomiális időben eldönthető, hogy a kielégíti-e a Rabin-feltételt.

Ehhez csak a 3.1.2 lemmát kell emlékezetbe idézni: a^b maradéka modulo c kiszámítható polinomiális időben. E három lemma alapján a következő randomizált algoritmus adható prímtesztelésre:

5.2.4 Algoritmus: Véletlenszerűen választunk egy a számot 1 és $m-1$ között, és megnézzük, hogy kielégíti-e a Rabin-feltételt. Ha nem, akkor m összetett. Ha igen, új a-t választunk. Ha 100-szor egymásután teljesül a Rabin-feltétel, akkor azt mondjuk, hogy m prím.

Ha m prím, akkor az algoritmus biztosan ezt mondja. Ha m összetett, akkor a véletlenszerűen választott a szám legalább $1/2$ valószínűséggel megsérti a Rabin-feltételt. Így száz független kísérlet során 2^{-100}-nál kisebb annak a valószínűsége, hogy egyszer sem sérül meg a Rabin-feltétel, vagyis hogy az algoritmus azt mondja, hogy m prím.

Megjegyzések: 1. Ha az algoritmus azt találja, hogy m összetett, akkor — érdekes módon — nem abból látjuk ezt, hogy egy osztóját találja meg, hanem (lényegében) arról, hogy megsérti a Rabin-feltételt. Ha a szám a Fermat-feltételt nem sérti meg, akkor m nem lehet a $a^M-1, a^M+1, a^{2M}+1, a^{4M}+1, \ldots, a^{2^{k-1}M}+1$ számok mindegyikéhez relatív prím, így ezekkel vett legnagyobb közös osztóit kiszámítva, valamelyik egy valódi osztó lesz. Nem ismeretes (sem determinisztikus, sem randomizált) polinomiális algoritmus arra, hogy ha a Fermat-feltétel is megsérül, egy valódi osztót találjunk. Ez a feladat gyakorlatban is lényegesen nehezebb, mint a prímség eldöntése. A kriptográfiáról szóló fejezetben látni fogjuk, hogy ennek a tapasztalati ténynek fontos alkalmazásai vannak.

2. Megpróbálhatunk adott m-hez a Rabin-feltételt megsértő a-t nem véletlen választással, hanem az 1, 2, stb. számok kipróbálásával keresni. Nem ismeretes, hogy ha m összetett, milyen kicsi az első ilyen szám. Felhasználva azonban az analitikus számelmélet egy évszázados sejtését, az ún. Általánosított Riemann Hipotézist, meg lehet mutatni, hogy nem nagyobb, mint $\log^2 m$. Így ez a determinisztikus prímtesztelés polinomiális idejű, ha az Általánosított Riemann Hipotézis igaz.

Az előzőekben megismert prímtesztelési algoritmust felhasználhatjuk arra, hogy adott n számjegyű prímszámot keressünk (mondjuk kettes számrendszerben). Válasszunk ugyanis véletlenszerűen egy k számot a $[2^{n-1}, 2^n-1]$ intervallumból, és ellenőrizzük, hogy prím-e, mondjuk legfeljebb 2^{-100} valószínűségű hibával. Ha igen, megállunk. Ha nem, új k számot választunk. Mármost a prímszámok elméletéből következik, hogy ebben az intervallumban nemcsak, hogy van prímszám, de a prímszámok száma elég nagy: aszimptotikusan $(\log e)2^{n-1}/n$, vagyis egy véletlenszerűen választott n jegyű szám kb. $(\log e)/n$ valószínűséggel lesz prím. Ezért a kísérletet $O(n)$-szer ismételve már igen nagy valószínűséggel találunk prímszámot.

Ugyanígy választhatunk véletlen prímet minden elég hosszú intervallumból, pl. az $[1, 2^n]$ intervallumból.

5.3. Randomizált komplexitási osztályok

Az előző két pontban olyan algoritmusokat tárgyaltunk, melyek véletlen számokat használtak fel. Most az ilyen algoritmusokkal megoldható feladatoknak definiáljuk egy osztályát.

Először a megfelelő gépet definiáljuk. Legyen $T = \langle k, \Sigma, \Gamma, \alpha, \beta, \gamma \rangle$ egy nemdeterminisztikus Turing-gép, és legyen minden $g \in \Gamma$, $h_1, \ldots, h_k \in \Sigma$-ra az $\alpha(g, h_1, \ldots, h_k)$, $\beta(g, h_1, \ldots, h_k)$, $\gamma(g, h_1, \ldots, h_k)$ halmazokon egy-egy valószínűségeloszlás megadva. (Célszerű föltenni, hogy az egyes események valószínűségei racionális számok, így ilyen valószínűségű események könnyen generálhatók, föltéve, hogy egymástól független bitek generálhatók.) A nem-determinisztikus Turing-gépet ezekkel a valószínűségeloszlásokkal együtt *randomizált Turing-gépnek* nevezzük.

Egy randomizált Turing-gép minden legális számolásának van bizonyos valószínűsége. Azt mondjuk, hogy a randomizált Turing-gép *gyengén eldönt* (vagy *Monte-Carlo értelemben eldönt*) egy $\mathcal{L}$ nyelvet, ha minden $x \in \Sigma^*$ bemenetre legalább $3/4$ valószínűséggel úgy áll meg, hogy $x \in \mathcal{L}$ esetén az eredményszalagra 1-et, $x \notin \mathcal{L}$ esetén az eredményszalagra 0-t ír. Röviden: legfeljebb $1/4$ a valószínűsége annak, hogy hibás választ ad.

Példáinkben ennél erősebb értelemben használtunk randomizált algoritmusokat: azok legfeljebb az egyik irányban tévedhettek. Azt mondjuk, hogy a randomizált Turing-gép *elfogad* egy $\mathcal{L}$ nyelvet, ha minden x bemenetre $x \notin \mathcal{L}$ esetén az x szót mindig elutasítja, $x \in \mathcal{L}$ esetén pedig legalább $1/2$ annak a valószínűsége, hogy az x szót elfogadja.

Azt mondjuk, hogy a randomizált Turing-gép *erősen eldönt* (vagy *Las Vegas értelemben eldönt*) egy $\mathcal{L}$ nyelvet, ha minden $x \in \Sigma^*$ szóra 1 valószínűséggel helyes választ ad. (Mivel minden véges hosszúságú konkrét számolás valószínűsége pozitív, így itt a 0 valószínűségű kivétel nem lehet az, hogy a gép rossz válasszal áll meg, hanem csak az, hogy végtelen ideig működik.)

Randomizált Turing-gépnél meg lehet különböztetni minden bemenetre a legrosszabb számolás lépésszámát, és a várható lépésszámot. Mindazon nyelvek osztályát, melyet randomizált Turing-gépen polinomiális várható időben gyengén el lehet dönteni, BPP-vel (Bounded Probability Polynomial) jelöljük. Mindazon nyelvek osztályát, melyet randomizált Turing-gépen polinomiális várható időben föl lehet ismerni, RP-vel (Random Polynomial) jelöljük. Mindazon Turing-gépek osztályát, melyeket randomizált Turing-gépen polinomiális várható időben erősen el lehet dönteni, ΔRP-rel jelöljük. Nyilvánvaló, hogy $\mathrm{BPP} \supseteq \mathrm{RP} \supseteq \Delta\mathrm{RP} \supseteq \mathrm{P}$.

A gyenge eldöntés definíciójában szereplő $3/4$ konstans önkényes: ehelyett bármilyen 1-nél kisebb, de $1/2$-nél nagyobb számot is mondhatnánk anélkül, hogy pl. a BPP osztály változna ($1/2$-et már nem: ekkora valószínűségű helyes választ már pénzfeldobással adhatunk). Ugyanis ha a gép $1/2 < c < 1$ valószínűséggel ad helyes választ, akkor ismételjük meg az x bemeneten a számolást függetlenül t-szer, és a legtöbbször adott választ tekintsük válasznak. A Nagy Számok Törvényéből könnyen látható, hogy annak a valószínűsége, hogy ez a válasz hibás, kisebb, mint c_1^t, ahol c_1 csak a c-től függő 1-nél kisebb konstans. Elég nagy t-re ez tetszőlegesen kicsivé tehető, és ez a várható lépésszámot csak konstans

szorzóval növeli meg.

Hasonlóan belátható, hogy az elfogadás definíciójában szereplő 1/2 konstans is helyettesíthető volna bármilyen 1-nél kisebb pozitív számmal.

Végül még azt jegyezzük meg, hogy a BPP és RP osztályok definíciójában szereplő várható lépésszám helyett tekinthetnénk a legrosszabb lépésszámot is; ez sem változtatná meg az osztályokat. Nyilvánvaló, hogy ha a legrosszabb lépésszám polinomiális, akkor a várható lépésszám is az. Megfordítva, ha a várható lépésszám polinomiális, mondjuk legfeljebb $|x|^d$, akkor a Markov-egyenlőtlenség szerint annak a valószínűsége, hogy egy számolás több, mint $8|x|^d$ ideig tart, legfeljebb 1/8. Így beépíthetünk egy számlálót, mely a gépet $8|x|^d$ lépés után leállítja, és az eredményszalagra 0-t ír. Ez a hiba valószínűségét legfeljebb 1/8-dal növeli meg.

Ugyanez a ΔRP osztályra már nem ismeretes: itt a legrosszabb futási idő korlátozása már determinisztikus algoritmushoz vezetne, és nem ismeretes, hogy ΔRP egyenlő-e P-vel (sőt inkább azt lehet várni, hogy nem; vannak példák olyan polinomiális Las Vegas algoritmussal megoldható problémákra, melyekre polinomiális determinisztikus algoritmus nem ismeretes).

Megjegyzés: Definiálhatnánk a véletlent használó Turing-gépet máshogy is: tekinthetnénk egy olyan determinisztikus Turing-gépet, melynek a szokásos (bemenet-, munka-, és eredmény-) szalagjain kívül van egy olyan szalagja is, melynek minden mezejére egy véletlenül, 1/2 valószínűséggel választott bit (mondjuk 0 vagy 1) van írva. A különböző mezőkön álló bitek egymástól függetlenek. A gép maga determinisztikusan működik, de számolása természetesen függ a véletlentől (a véletlen szalagra írt jelektől). Könnyű belátni, hogy az így definiált (determinisztikus, véletlen szalaggal ellátott) Turing-gép és a nemdeterminisztikus, de valószínűségeloszlással ellátott Turing-gép egymással helyettesíthetők volnának.

Definiálhatjuk a randomizált RAM-ot is: ennek van egy külön w rekesze, melyben mindig 1/2 valószínűséggel 0 vagy 1 áll. A programnyelvhez hozzá kell még venni az $y := w$ utasítást. Valahányszor ez végre van hajtva, a w rekeszben új véletlen bit jelenik meg, mely az előző bitektől teljesen független. Ismét nem nehéz belátni, hogy ez nem jelentene lényeges különbséget.

Látható, hogy minden RP-beli nyelv NP-ben van. Triviális, hogy a BPP és ΔRP osztályok komplementálásra zártak: ha minden $\mathcal{L}$ nyelvvel együtt tartalmazzák a $\Sigma_0^* - \mathcal{L}$ nyelvet is. Az RP osztály definíciója nem ilyen, és nem is ismeretes, hogy ez az osztály zárt-e a komplementálásra. Ezért érdemes definiálni a co-RP osztály: Egy $\mathcal{L}$ nyelv co-RP-ben van, ha $\Sigma_0^* - \mathcal{L}$ RP-ben van.

Az NP osztálynak hasznos jellemzését adták a „tanúk”. Egy analóg tétel az RP osztályra is áll:

5.3.1. Tétel: *Egy $\mathcal{L}$ nyelv akkor és csak akkor van RP-ben, ha létezik olyan $\mathcal{L}' \in P$ nyelv és olyan $f(n)$ polinom, hogy*

*(*a*)* $\mathcal{L} = \{x \in \Sigma^* : \exists y \in \Sigma^*, |y| = f(|x|), x\&y \in \mathcal{L}'\}$

és

*(*b*) ha $x \in \mathcal{L}$, akkor az $f(|x|)$ hosszúságú y szavaknak legalább fele olyan, hogy $x\&y \in \mathcal{L}'$.*

Bizonyítás: Hasonló az NP-re vonatkozó tétel bizonyításához. □

Az RP és ΔRP osztályok kapcsolata szorosabb, mint ahogyan - az NP és P osztályok analógiája alapján - várnánk:

5.3.2 Tétel: *Egy $\mathcal{L}$ nyelvre az alábbiak ekvivalensek:*

(i) $\mathcal{L} \in \Delta\mathrm{RP}$*;*

(ii) $\mathcal{L} \in \mathrm{RP} \cap \text{co-RP}$*;*

(iii) Van olyan polinomiális (legrosszabb) idejű randomizált Turing-gép, mely az eredményszalagra az „1" és „0" jeleken kívül a „FÖLADOM" szót is írhatja; az „1" és „0" válaszok soha nem hibásak, vagyis $x \in \mathcal{L}$ esetén vagy „1" vagy „FÖLADOM", $x \notin \mathcal{L}$ esetén pedig vagy „0" vagy „FÖLADOM" az eredmény. A „FÖLADOM" válasz valószínűsége legfeljebb 1/2 lehet.

Bizonyítás: Nyilvánvaló, hogy (i)$\Longrightarrow$(ii). Ugyancsak könnyen látható, hogy (ii)$\Longrightarrow$(iii): Adjuk be x-et egy olyan randomizált Turing-gépnek, mely $\mathcal{L}$-et polinomiális időben elfogadja, és egy olyannak is, mely $\Sigma^* - \mathcal{L}$-et fogadja el polinomiális időben. Ha a kettő ugyanazt a választ adja, akkor az biztosan korrekt. Ha eltérő választ adnak, akkor „föladjuk". Ekkor valamelyik tévedett, és így ennek a valószínűsége legfeljebb $1/2$.

Végül (iii)$\Longrightarrow$(i) belátásához nem kell mást tenni, mint a (iii)-beli T_0 Turing gépet úgy módosítani, hogy a „FÖLADOM" válasz helyett induljon újra. Ha T_0 lépésszáma egy x bemeneten τ, és a föladás valószínűsége p, akkor ugyanezen a bemeneten a módosított gép várható lépésszáma

$$\sum_{t=1}^{\infty} c^{t-1}(1-c)t\tau = \frac{\tau}{1-c} \leq 2\tau.$$

□

A korábbi szakaszokban láttuk, hogy az összetett számok „nyelve" RP-ben van. Ennél több is igaz: legújabban Adleman és Huang megmutatták, hogy ez a nyelv ΔRP-ben is benne van. A másik fontos példánkra, a nem azonosan 0 polinomokra csak annyi ismeretes, hogy RP-ben vannak. Az algebrai (elsősorban csoportelméleti) problémák között is sok olyan van, mely RP-ben ill. ΔRP-ben van, de polinomiális algoritmus nem ismeretes a megoldására.

Megjegyzés: A véletlent használó algoritmusok nem tévesztendők össze az olyan algoritmusokkal, melyeknek teljesítményét (pl. lépésszámának várható értékét) véletlen bemenetre vizsgáljuk. Mi itt a bemenetek halmazán nem tételeztünk föl valószínűségeloszlást, hanem a legrosszabb esetet tekintettük. Algoritmusoknak bizonyos eloszlásból származó véletlen bemeneteken várható viselkedésének vizsgálata igen fontos, de nehéz és meglehetősen gyermekcipőben járó terület, amivel itt nem foglalkozunk.

6. fejezet

Információs bonyolultság (a véletlen komplexitáselméleti fogalma)

A valószínűségszámítás matematikai megalapozása HILBERT már említett híres problémái között szerepel. Erre az első fontos kísérletet VON MISES tette, aki egy 0-1 sorozat véletlen voltát akarta definiálni, azzal, hogy a 0-k és 1-ek gyakorisága megközelítőleg azonos, és ez minden pl. számtani sorozat szerint választott részsorozatra is igaz. Ez a megközelítés akkor nem bizonyult elég hatékonynak. Más irányban indult el KOLMOGOROV, aki a véletlen fogalmát mértékelméletileg alapozta meg. Elmélete a valószínűségszámítás szempontjából igen sikeres volt, de voltak olyan kérdések, melyeket nem tudott megfogni. Így pl. egyetlen 0-1 sorozat véletlen voltáról a mértékelméletre alapozott valószínűségszámításban nem beszélhetünk, csak sorozatok egy halmazának valószínűségéről, holott hétköznapi értelemben pl. a FejFejFejFej... sorozatról magában is „nyilvánvaló", hogy nem lehet véletlen pénzdobálás eredménye. KOLMOGOROV és CHAITIN a 60-as években feleleventették VON MISES gondolatát, bonyolultságelméleti eszközöket használva. Eredményeik érdekessége túlmutat a valószínűségszámítás megalapozásán; az adattárolás alapvető fogalmainak tisztázásához is hozzájárul.

6.1. Információs bonyolultság

Rögzítsünk egy Σ ábécét, és legyen, mint korábban, $\Sigma_0 = \Sigma - \{*\}$. Kényelmes lesz Σ_0-t a $\{0, 1, \ldots, m-1\}$ halmazzal azonosítani. Tekintsünk egy Σ fölötti 2 szalagos univerzális T Turing-gépet. Azt mondjuk, hogy egy Σ_0 fölötti q szó (program) a T gépen *kinyomatja* az x szót, ha a T gép második szalagjára q-t írva, az elsőt üresen hagyva, a gép véges sok lépésben megáll úgy, hogy az első szalagján az x szó áll. Mindjárt jegyezzük meg, hogy minden x szó kinyomatható T-n. Ugyanis van olyan egy szalagos (eléggé triviális) S_x Turing-gép, mely üres szalaggal indulva nem csinál mást, mint erre az x szót írja. Ez a Turing-gép szimulálható T-n egy q_x programmal, mely ezek szerint x-et nyomatja ki.

Egy $x \in \Sigma_0^*$ szó *bonyolultságán* a legrövidebb olyan szó (program) hosszát értjük, mely T-n az x szót nyomatja ki. Az x szó bonyolultságát $\mathbf{K}_T(x)$-szel jelöljük.

Az x-et kinyomató programot úgy is tekinthetjük, mint az x szó egy „kódját", ahol maga

a T Turing-gép a dekódolást végzi. Egy ilyen programot az x szó *Kolmogorov-kódjának* nevezzük. Egyelőre nem teszünk feltevést arra vonatkozóan, hogy ez a dekódolás (vagy a kódolás, a megfelelő program megtalálása) mennyi ideig tarthat.

Azt szeretnénk, ha ez a bonyolultság az x szó egy jellemző tulajdonsága lenne, és minnél kevésbé függne a T géptől. Sajnos könnyű olyan univerzális T gépet csinálni, ami nyilvánvalóan „ügyetlen". Például minden programnak csak minden második jelét használja föl, a közbülső betűkön „átsiklik": ekkor kétszer olyan bonyolultnak definiál minden x szót, mint ha ezeket a betűket le se kellene írni.

Megmutatjuk azonban, hogy ha bizonyos — eléggé egyszerű — feltevéseket teszünk a T gépre, akkor már nem lesz lényeges, hogy melyik univerzális Turing-gépet használjuk a bonyolultság definiálására. Durván szólva elég azt feltennünk, hogy minden T-n végrehajtható számítás bemenetét beadhatjuk a program részeként is. Ennek pontosításaként feltesszük, hogy van olyan szó (mondjuk ADATOK), melyre az áll, hogy

(a) minden egyszalagos Turing-gép szimulálható olyan programmal, mely az ADATOK szót rész-szóként nem tartalmazza, és

(b) ha a gép második szalagjára olyan szót írunk, mely az ADATOK szót rész-szóként tartalmazza, tehát ami így írható: xADATOKy, ahol az x szó már nem tartalmazza az ADATOK rész-szót, akkor a gép akkor és csak akkor áll meg, mint ha y-t az első szalagra és x-et a második szalagra írva indítottuk volna el, és megálláskor az első szalagon ugyanaz áll.

Könnyű látni, hogy minden univerzális Turing-gép módosítható úgy, hogy az (a), (b) feltevéseknek eleget tegyen. A továbbiakban mindig feltesszük, hogy univerzális Turing-gépünk ilyen tulajdonságú.

6.1.1 Lemma: *Létezik olyan (csak T-től függő) c_T konstans, hogy $\mathbf{K}_T(x) \leq |x| + c_T$.*

Bizonyítás: T univerzális, tehát az a (triviális) egyszalagos Turing-gép, mely semmit sem csinál (azonnal megáll), szimulálható rajta egy p_0 programmal. Ekkor viszont bármely $x \in \Sigma_0^*$ szóra, a p_0ADATOKx program az x szót fogja kinyomatni. Így $c_T = |p_0| + 6$ megfelel a feltételeknek. □

Egyszerűség kedvéért a továbbiakban feltesszük, hogy $c_T \leq 100$.

Megjegyzés: Kissé vigyázni kellett, mert nem akartuk megszorítani, hogy milyen jelek fordulhatnak elő az x szóban. Pl. BASIC-ben a `PRINT ,,x''` utasítás nem jó az olyan x szavak kinyomatására, melyek tartalmazzák a „ jelet. Az érdekel bennünket, hogy az x szót az *adott ábécében* mennyire lehet tömören kódolni, és így nem engedjük meg az ábécé bővítését.

Most bebizonyítunk egy alapvető lemmát, mely azt mutatja, hogy a bonyolultság (a fenti feltételek mellett) nem függ nagyon az alapul vett géptől.

6.1.2 Tétel (Invariancia Tétel): *Legyen T és S az* (a), (b) *feltételeknek eleget tevő univerzális Turing-gép. Ekkor van olyan c_{TS} konstans, hogy bármely x szóra $|\mathbf{K}_T(x) - \mathbf{K}_S(x)| \leq c_{TS}$.*

Bizonyítás: Az S kétszalagos Turing-gép működését szimulálhatjuk egy 1-szalagos S_1 Turing-géppel úgy, hogy ha S-en valamely q program egy x szót nyomtat ki, akkor S_1 szalagjára q-t írva, az is megáll véges sok lépésben, éspedig úgy, hogy a szalagjára x van írva. Továbbmenve, az S_1 Turing-gép működését szimulálhatjuk T-n egy olyan p_{S_1} programmal, mely az ADATOK rész-szót nem tartalmazza.

Legyen mármost x tetszőleges Σ_0^*-beli szó, és legyen q_x egy legrövidebb olyan program, mely S-en x-et kinyomatja. Tekintsük T-n a p_{S_1}ADATOKq_x programot: ez nyilván x-et nyomatja ki, és hossza csak $|q_x| + c_{TS}$, ahol $c_{TS} = |p_{S_1}| + 6$. Így tehát

$$\mathbf{K}_T(x) \leq \mathbf{K}_S(x) + c_{TS}.$$

Az ellenkező irányú egyenlőtlenség hasonlóan adódik. □

Ennek a lemmának az alapján most már T-t rögzítettnek tekintjük, és a továbbiakban nem írjuk ki a T indexet.

A következő tétel mutatja, hogy az optimális kód algoritmikusan nem kereshető meg.

6.1.3 Tétel: *A* $\mathbf{K}(x)$ *függvény nem rekurzív.*

Bizonyítás: A bizonyítás lényege egy klasszikus logikai paradoxon, az ún. írógép-paradoxon. (Ez egyszerűen így fogalmazható: „legyen n a legkisebb 100-nál kevesebb jellel nem definiálható szám". Akkor éppen most definiáltuk n-et 100-nál kevesebb jellel!)

Tegyük fel mármost, hogy $\mathbf{K}(x)$ kiszámítható. Legyen c alkalmasan megválasztandó természetes szám. Rendezzük Σ_0^* elemeit növekvő sorrendben. Jelölje $x(k)$ a k-adik szót e szerint a rendezés szerint, és legyen x_0 a legelső olyan szó, melyre $\mathbf{K}(x_0) \geq c$. Föltéve, hogy gépünk Pascal nyelven programozható, tekintsük az alábbi egyszerű programot:

```
var k : integer;
function x(k : integer) : integer;
⋮
function Kolm(k : integer) : integer;
⋮
begin
  k := 0;
  while Kolm(k) < c do k := k + 1;
  print(x(k));
end.
```

Ez a program nyilván x_0-t nyomatja ki. A hosszának a meghatározásánál hozzá kell venni az $x(k)$ és $\mathrm{Kolm}(k) = \mathbf{K}(x(k))$ függvények kiszámítását; ez azonban összesen is csak $\log c +$konstans számú jel. Ha c-t elég nagynak vesszük, ez a program c-nél kevesebb jelből áll, és x_0-t nyomtatja ki, ami ellentmondás. □

A tétel egyszerű alkalmazásaként új bizonyítást nyerünk a megállási probléma eldönthetetlenségére. Miért is nem lehet ugyanis $\mathbf{K}(x)$-et a következőképpen kiszámítani? Vegyük

sorban a szavakat, és nézzük meg, hogy T második szalagjára ezeket írva, úgy áll-e le, hogy az első szalagra x van írva. Tegyük föl, hogy van olyan program, mely egy adott programról eldönti, hogy azt írva a második szalagra, T véges sok lépésben megáll-e. Ekkor azokat a szavakat, melyekkel T „elszáll", eleve ki tudjuk szűrni, ezekkel nem is próbálkozunk. A maradék szavak közül a legelsőnek a hossza, mellyel T az x szót kinyomatja, lesz $\mathbf{K}(x)$.

Az előző tétel szerint ez az „algoritmus" nem működhet; csak az lehet azonban a baja, hogy nem tudjuk kiszűrni a végtelen ideig futó programokat, vagyis a megállási probléma nem dönthető el.

Feladatok. 1. Mutassuk meg, hogy a $\mathbf{K}(x)$ függvényt még megközelítőleg sem tudjuk kiszámítani a következő értelemben: Ha f rekurzív függvény, akkor nincsen olyan algoritmus, mely minden x szóhoz egy $\gamma(x)$ természetes számot számol ki úgy, hogy minden x-re

$$\mathbf{K}(x) \leq \gamma(x) \leq f(\mathbf{K}(x)).$$

2. Bizonyítsuk be, hogy nincs olyan algoritmus, mely minden adott n számhoz olyan n hosszúságú x 0-1 sorozatot konstruál, melyre $\mathbf{K}(x) > 2\log n$.

3. Ha egy $f : \Sigma_0^* \to \mathbf{Z}_+$ rekurzív függvényre $f \leq \mathbf{K}$, akkor f korlátos.

A 6.1.3 tétellel és az 1. feladattal szembeállíthatóan megmutatjuk, hogy a $\mathbf{K}(x)$ bonyolultság *majdnem minden* x-re igen jól megközelíthető. Ehhez először is pontosítanunk kell, hogy mit értünk „majdnem minden" x-en. Tegyük fel, hogy a bemenő szókat véletlenszerűen kapjuk; másszóval, minden $x \in \Sigma_0^*$ szónak van egy $p(x)$ valószínűsége. Erről tehát annyit tudunk, hogy

$$p(x) \geq 0, \qquad \sum_{x \in \Sigma_0^*} p(x) = 1.$$

Ezen felül csak annyit kell feltennünk, hogy *$p(x)$ algoritmikusan kiszámítható.* Egy ilyen tulajdonságú p függvényt *kiszámítható valószínűségeloszlásnak* nevezünk. Egyszerű példát ad ilyen valószínűségeloszlásra a $p(x_k) = 2^{-k}$, ahol x_k a növekvő rendezés szerinti k-adik szó; vagy a $p(x) = (m+1)^{-|x|-1}$.

6.1.4 Tétel: *Minden kiszámítható p valószínűségeloszláshoz van olyan algoritmus, mely minden x szóhoz kiszámítja x egy $f(x)$ Kolmogorov-kódját, és erre a kódra $|f(x)| - \mathbf{K}(x)$ várható értéke véges.*

Bizonyítás: Legyen $x_1, x_2, \ldots,$ a Σ_0^*beli szavaknak az a sorbarendezése, melyre $p(x_1) \geq p(x_2) \geq \ldots$, és az azonos valószínűségű szavak mondjuk növekvő sorrendben vannak.

Állítás: *Az i indexhez az x_i szó algoritmikusan kiszámítható.*

Az állítás bizonyítása: Legyenek $y_1, y_2, \ldots$ a szavak növekvően rendezve. Legyen k rendre $i,\ i+1, \ldots$; adott k-hoz számítsuk ki a $p(y_1), \ldots, p(y_k)$ számokat és legyen t_k ezek közül az i-edik legnagyobb. Nyilván $t_i \leq t_{i+1} \leq \ldots$ és $t_k \leq p(x_i)$ minden $k \geq i$-re. Továbbá, ha

$$(*) \qquad p(y_1) + \ldots + p(y_k) \geq 1 - t_k,$$

akkor a további szavak között t_k-nál nagyobb valószínűségű már nem lehet, így $t_k = p(x_i)$ és x_i az első olyan y_j szó ($1 \le j \le k$), melyre $p(y_j) = t_k$.

Így sorra véve a $k = i,\ i+1, \ldots$ értékeket, megállhatunk, ha $(*)$ teljesül. Mivel $(*)$ baloldala 1-hez tart, a jobboldal pedig monoton nem-növekvő, ez előbb-utóbb bekövetkezik. Ezzel az állítást bebizonyítottuk.

Visszatérve a tétel bizonyítására, az Állításban szereplő algoritmus programmja az i számmal együtt az x_i szó egy $f(x_i)$ Kolmogorov-kódját szolgáltatja. Megmutatjuk, hogy ez a kód kielégíti a tétel követelményeit.

Nyilván $|f(x)| \ge \mathbf{K}(x)$. Továbbá $|f(x)| - \mathbf{K}(x)$ várható értéke

$$\sum_{i=1}^{\infty} p(x_i)\,(|f(x_i)| - \mathbf{K}(x_i)) = \sum_{i=1}^{\infty} p(x_i)(|f(x_i)| - \log_m i) + \\ + \sum_{i=1}^{\infty} p(x_i)(\log_m i - \mathbf{K}(x_i)).$$

Itt $|f(x_i)| = \log_m i + c$ a konstrukció miatt (c konstans), és $\mathbf{K}(x_i) \le |f(x_i)|$, hiszen $f(x_i)$ egy speciális Kolmogorov-kód, tehát a várható érték nem negatív. Tekintsük a fenti összeg első tagját:

$$\sum_{i=1}^{\infty} p(x_i)(|f(x_i)| - \log_m i) = \sum_{i=1}^{\infty} p(x_i)c = c.$$

Másrészt a második tagban az összeget növeljük, ha a $\mathbf{K}(x_i)$ számokat növekvő sorrendbe rendezzük át (mivel a $p(x_i)$ együtthatók csökkennek). Legyen z_i a szavak egy ilyen rendezése, tehát ahol $\mathbf{K}(z_1) \le \mathbf{K}(z_2) \le \ldots$. Ekkor a megnövelt második tag: $\sum_{i=1}^{\infty} p(x_i)(\log_m i - \mathbf{K}(z_i))$. Azon x szavak száma, melyekre $\mathbf{K}(x) = k$, legfeljebb m^k. Tehát azon x szavak száma, melyekre $\mathbf{K}(x) \le k$, legfeljebb $1 + m + m^2 + \ldots + m^k \le m^{k+1}$. De ez pont azt jelenti, hogy $i \le m^{\mathbf{K}(z_i)+1}$, azaz $\mathbf{K}(z_i) \ge \log_m i - 1$.

$$\sum_{i=1}^{\infty} p(x_i)(|f(x_i)| - \mathbf{K}(x_i)) \le c + \sum_{i=1}^{\infty} p(x_i)(\log_m i - \mathbf{K}(z_i)) \le c + \sum_{i=1}^{\infty} p(x_i) = c + 1. \ \square$$

6.2. Önkorlátozó információs bonyolultság

A Kolmogorov-kód, ha szigorúan vesszük, kihasznál a Σ_0 ábécén kívül is egy jelet: a programszalag olvasásakor a program végét arról ismeri föl, hogy a „$*$” jelet olvassa. Módosíthatjuk a fogalmat úgy, hogy ez ne legyen lehetséges: a programot olvasó fejnek nem szabad túlszaladnia a programon (pl. fenntartunk egy jelet a „program vége” jelzésére). Egy olyan szót, melyet kétszalagos univerzális T Turing-gépünk programszalagjára írva, a fej soha nem is próbál a programon kívüli mezőt olvasni, *önkorlátozónak* nevezünk. A legrövidebb x-et kinyomtató önkorlátozó program hosszát $\mathbf{H}_T(x)$-szel jelöljük. Ezt a módosított információs bonyolultságfogalmat Levin és Chaitin vezették be. Könnyű belátni, hogy az Invarianciatétel most is érvényes, és ezért ismét elég az index nélküli $\mathbf{H}(x)$ jelölést

használni. A $\mathbf{K}$ és $\mathbf{H}$ függvények nem térnek el nagyon, ahogy azt a következő lemma mutatja:

6.2.1 Lemma: $\mathbf{K}(x) \leq \mathbf{H}(x) \leq \mathbf{K}(x) + 2\log_m \mathbf{K}(x) + O(1)$.

Bizonyítás: Az első egyenlőtlenség triviális. A második bizonyításához tekintsünk egy p programot, mely x-et kétszalagos univerzális T Turing-gépünkön kinyomtatja. Legyen $n = |p|$, és legyen az n szám m alapú számrendszerbeli alakja $u_1...u_k$, valamint legyen $u = u_1 0 u_2 0 ... u_k 011$. Ekkor az up szóban az u prefix egyértelműen rekonstruálható, és ebből a szó hossza megállapítható anélkül, hogy túlszaladnánk a végén. Könnyű ezek alapján olyan önkorlátozó programot írni, mely $2k+n+O(1)$ hosszúságú, és x-et nyomtatja ki.□

Az eddigiek alapján úgy tűnhet, hogy a $\mathbf{H}$ függvény a Kolmogorov-bonyolultság egy apró technikai változata. A következő lemma egy lényeges különbséget mutat közöttük.

6.2.2 Lemma:

(a) $\sum_x m^{-\mathbf{K}(x)} = +\infty$.

(b) $\sum_x m^{-\mathbf{H}(x)} \leq 1$.

Bizonyítás: Az (a) állítás a 6.1.1 Lemmából azonnal következik. A (b) állítás bizonyítása végett tekintsük minden x szónak egy optimális $f(x)$ kódját. Az önkorlátozás miatt ezek egyike sem lehet egy másiknak kezdőszelete (prefixe); így (b) azonnal adódik a következő egyszerű, de fontos információelméleti lemmából:

6.2.3 Lemma: *Legyen $\mathcal{L} \subseteq \Sigma_0^*$ olyan nyelv, melynek egyik szava sem prefixe a másiknak, és legyen $m = |\Sigma_0|$. Ekkor* $\sum_{y \in \mathcal{L}} m^{-|y|} \leq 1$.

Bizonyítás: Válasszunk a Σ_0 ábécéből véletlenszerűen, függetlenül és egyenletes eloszlás szerint $a_1, a_2, ...$ betűket; álljunk meg, ha a kapott szó $\mathcal{L}$-ben van. Annak a valószínűsége, hogy egy $y \in \mathcal{L}$ szót kapunk, éppen $m^{-|y|}$ (hiszen y kezdőszeletén nem állunk meg a feltevés szerint). Mivel ezek egymást kizáró események, a lemma állítása következik. □

A 6.2.1 és 6.2.2 lemmák néhány következményét fogalmazzák meg az alábbi feladatok:

Feladatok. 4. Mutassuk meg, hogy a 6.2.1 lemma alábbi élesítése már nem igaz: $\mathbf{H}(x) \leq \mathbf{K}(x) + \log_m \mathbf{K}(x) + O(1)$.

5. A $\mathbf{H}(x)$ függvény nem rekurzív.

A következő tétel azt mutatja, hogy a $\mathbf{H}(x)$ függvény jól közelíthető.

6.2.4 Tétel (Levin kódolási tétele): Legyen p egy kiszámítható valószínűségeloszlás Σ_0^*-on. Ekkor minden x szóra $\mathbf{H}(x) \leq -\log_m p(x) + O(1)$.

Bizonyítás: Nevezzük *m-adikusan racionálisnak* azokat a racionális számokat, melyek felírhatók úgy, hogy nevezőjük m-nek hatványa. A $[0,1)$ intervallum m-adikusan racionális számai m alapú számrendszerben $0,a_1...a_k$ alakban írhatók, ahol $0 \le a_i \le m-1$.

Osszuk fel a $[0,1)$ intervallumot balról zárt, jobbról nyílt, rendre $p(x_1), p(x_2), ...$ hosszúságú $J(x_1), J(x_2), ...$ intervallumokra (ahol $x_1, x_2, ...$ a Σ_0^* növekvő rendezése). Minden olyan $x \in \Sigma_0^*$-hoz, melyre $p(x) > 0$, lesz olyan $0,a_1...a_k$ m-adikusan racionális szám, melyre $0,a_1...a_k \in J(x)$ és $0,a_1...a_k(m-1) \in J(x)$. Egy legrövidebb ilyen tulajdonságú $a_1...a_k$ sorozatot nevezzünk az x *Levin kódjának*.

Azt állítjuk, hogy minden x szó a Levin-kódjából könnyen kiszámítható. Valóban, adott $a_1, ..., a_k$ sorozathoz $i = 0, 1, 2, ...$ értékekre rendre megnézzük, hogy $0,a_1...a_i$ és $0,a_1...a_i(m-1)$ ugyanabba a $J(x)$ intervallumba esnek-e; ha igen, kinyomtatjuk x-et és megállunk. Vegyük észre, hogy ez a program önkorlátozó: nem kell tudnunk előre, hogy milyen hosszú a kód, és ha $a_1...a_k$ egy x szó Levin-kódja, akkor soha nem fogunk az $a_1...a_k$ sorozat végén túl olvasni. Így tehát $(\mathrm{H})(x)$ nem nagyobb, mint a fenti algoritmus (konstans hosszúságú) programjának és az x Levin-kódjának együttes hossza; erről könnyű látni, hogy legfeljebb $\log_m p(x) + 1$. □

E tételből következik, hogy $\mathbf{H}(x)$ és $-log_m p(x)$ eltérésének várható értéke korlátos (v.ö. a 6.1.4 tétellel).

6.2.5 Következmény: *A 6.2.4 tétel feltételeivel*

$$\sum_x p(x)|\mathbf{H}(x) + \log_m p(x)| = O(1).$$

Bizonyítás:

$$\sum_x p(x)|\mathbf{H}(x) + \log_m p(x)| =$$

$$= \sum_x p(x)|\mathbf{H}(x) + \log_m p(x)|_+ + \sum_x p(x)|\mathbf{H}(x) + \log_m p(x)|_-.$$

Itt az első összeg a 6.2.4 tétel szerint becsülhető:

$$\sum_x p(x)|\mathbf{H}(x) + \log_m p(x)|_+ \le \sum_x p(x)O(1) = O(1).$$

A második összeget így becsüljük:

$$|\mathbf{H}(x) + \log_m p(x)|_- \le m^{-\mathbf{H}(x)-\log_m p(x)} = \frac{1}{p(x)} m^{-\mathbf{H}(x)},$$

és így a 6.2.2 lemma szerint

$$\sum_x p(x)|\mathbf{H}(x) + \log_m p(x)|_- \le \sum_x m^{-\mathbf{H}(x)} \le 1.$$

□

6.3. A véletlen sorozat fogalma

Ebben a szakaszban föltesszük, hogy $\Sigma_0 = \{0, 1\}$, vagyis 0-1 sorozatok bonyolultságával foglalkozunk.

Durván szólva, akkor akarunk egy sorozatot véletlennek tekinteni, ha nincs benne szabályosság. Itt a szabályosságot azzal fogjuk meg, hogy az a sorozat gazdaságosabb kódolására adna lehetőséget, tehát a sorozat bonyolultsága kicsi volna.

Nézzük először, hogy mekkora az „átlagos" 0-1 sorozat bonyolultsága.

6.3.1 Lemma: *Azon n hosszúságú x 0-1 sorozatok száma, melyekre $\mathbf{K}(x) \leq n-k$, kisebb, mint 2^{n-k+1}.*

Bizonyítás: A legfeljebb $n-k$ hosszúságú „kódok" száma legfeljebb $1+2+\ldots+2^{n-k} < 2^{n-k+1}$, így csak 2^{n-k+1}-nél kevesebb sorozatnak lehet csak ilyen kódja. □

6.3.2 Következmény: *Az n hosszúságú 0-1 sorozatok 99%-ának a bonyolultsága nagyobb, mint $n-8$. Ha véletlenszerűen választunk egy n hosszúságú x 0-1 sorozatot, akkor $1-2^{-100}$ valószínűséggel $|\mathbf{K}(x) - n| \leq 100$.*

Az előző fejezet utolsó tétele szerint algoritmikusan lehetetlen megtalálni a legjobb kódot. Vannak azonban olyan, könnyen felismerhető tulajdonságok, melyek egy szóról azt mutatják, hogy az a hosszánál hatékonyabban kódolható. Egy ilyen tulajdonságot mutat a következő lemma.

6.3.3 Lemma: *Ha egy n hosszúságú x 0-1 sorozatban az 1-ek száma k, akkor*

$$\mathbf{K}(x) \leq \log_2 \binom{n}{k} + \log_2 n + \log_2 k + \text{konstans}.$$

Legyen $k = pn$ $(0<p<1)$, akkor ez így becsülhető:

$$\mathbf{K}(x) \leq (-p \log p + (1-p)\log(1-p))n + O(\log n).$$

Speciálisan, ha $m > (1/2+\epsilon)n$ vagy $m < (1/2-\epsilon)n$, akkor

$$\mathbf{K}(x) \leq cn + O(\log n),$$

ahol $c = (1/2+\epsilon)\cdot\log(1/2+\epsilon) + (1/2-\epsilon)\cdot\log(1/2-\epsilon)$ csak ϵ-tól függő 1-nél kisebb pozitív konstans.

Bizonyítás: x-et megadhatjuk úgy, mint „lexikografikusan a t-edik olyan n hosszúságú 0-1 sorozat, mely pontosan m 1-est tartalmaz". Mivel az m 1-est tartalmazó, n hosszúságú 0-1 sorozatok száma $\binom{n}{m}$, a t, n és m számok leírásához csak $\log_2 \binom{n}{m} + \log_2 n + \log_2 m$ bit kell, a megfelelő sorozatot kiválasztó programhoz pedig csak konstans számú.

A binomiális együttható becslése a valószínűségszámításból ismert módon történik. □

Legyen x végtelen 0-1 sorozat, és jelölje x_n az első n eleme által alkotott kezdőszeletét. Az x sorozatot *informatikusan véletlennek* nevezzük, ha $\mathbf{K}(x_n)/n \longrightarrow 1$ ha $n \longrightarrow \infty$.

Megmutatható, hogy minden informatikusan véletlen sorozat kielégíti a nagy számok törvényeit, a különböző statisztikai teszteket stb. Itt most csak a legegyszerűbb ilyen eredményt tekintjük. Jelölje a_n az x_n sorozatban az 1-esek számát, ekkor az előző lemmából rögtön adódik a következő tétel:

6.3.4 Tétel: *Ha x informatikusan véletlen, akkor $a_n/n \longrightarrow 1/2$ $(n \longrightarrow \infty)$.*

Kérdés, hogy a „véletlen" sorazat definíciója nem túl szigorú-e, van-e egyáltalán informatikusan véletlen sorozat. Megmutatjuk, hogy nem csak, hogy van, de majdnem minden sorozat ilyen. Pontosabban a következő igaz:

6.3.5 Tétel: *Legyenek a végtelen x $0-1$ sorozat elemei egymástól függetlenül $1/2$ valószínűséggel 0-k vagy 1-ek. Ekkor x 1 valószínűséggel informatikusan véletlen.*

Bizonyítás: Jelölje A_n azt az eseményt, hogy $\mathbf{K}(x_n) \leq n - 2\log n$. A 6.3.1 Lemma szerint $\text{Prob}(A_n) < 2^{1-2\log n} = 2/n^2$ és így a $\sum_{n=1}^{\infty} Prob(A_n)$ összeg konvergens. De ekkor a Borel-Cantelli Lemma szerint 1 valószínűséggel csak véges sok A_n esemény következik be. 1 tehát annak a valószínűsége, hogy van olyan n_0 természetes szám, hogy minden $n > n_0$-ra $\mathbf{K}(x_n) > n - 2\log n$. Minden ilyen esetben $\mathbf{K}(x_n)/n \longrightarrow 1$. □

Megjegyzés: Ha az x sorozat elemeit egy algoritmus generálja, akkor x_n-t megadhatjuk ennek a programjával (ami konstans hosszúságú) és az n számmal (amihez $\log n$ bit kell). Ilyen sorozatra tehát $\mathbf{K}(x_n)$ igen lassan nő.

6.4. Kolmogorov-bonyolultság, entrópia és kódolás

Legyen $p = (p_1, \ldots, p_m)$ egy *valószínűségeloszlás*, vagyis olyan nem-negatív vektor, melyre $\sum_i p_i = 1$. Ennek az *entrópiája* a

$$H(p) = \sum_{i=1}^{n} -p_i \log p_i$$

mennyiség (ha $p_i = 0$, akkor a $p_i \log p_i$ tagot is 0-nak tekintjük). Vegyük észre, hogy ebben az összegben minden tag nem-negatív, így $H(p) \geq 0$; egyenlőség akkor és csak akkor áll, ha valamelyik p_i értéke 1, a többi pedig 0. Könnyű belátni, hogy rögzített m-re a maximális entrópiájú valószínűségeloszlás az $(1/m, \ldots, 1/m)$ eloszlás, és ennek entrópiája $\log m$.

Az entrópia az információelmélet alapfogalma, és ebben a jegyzetben nem foglalkozunk vele részletesen, csak a Kolmogorov-bonyolultsággal való kapcsolatát tárgyaljuk. Az $m = 2$ esetben már találkoztunk az entrópiával a 6.3.3 Lemmában. Ez a lemma könnyen általánosítható tetszőleges ábécére:

6.4.1 Lemma. *Legyen $x \in \Sigma_0^*$ $|x| = n$, és jelölje p_h a $h \in \Sigma_0$ betű relatív gyakoriságát az x szóban. Legyen $p = (p_h : h \in \Sigma_0)$. Ekkor*

$$\mathbf{K}(x) \leq \frac{H(p)}{\log m} n + O(\log n).$$

□

Legyen $\mathcal{L} \subseteq \Sigma^*$ rekurzív nyelv, és tegyük föl, hogy csak az $\mathcal{L}$-beli szavakhoz akarunk őket kinyomató rövid programot, „kódot" találni. Keresünk tehát minden $x \in \mathcal{L}$ szóhoz olyan $f(x) \in \{0,1\}^*$ programot, mely őt kinyomtatja. Az $f : \mathcal{L} \to \Sigma^*$ függvényt *kódnak* nevezzük. A kód *tömörsége* az $\eta(n) = \max\{|f(x)| : x \in \mathcal{L}, |x| \leq n\}$ függvény. Nyilvánvaló, hogy $\eta(n) \geq \max\{\mathbf{K}(x) : x \in \mathcal{L}, |x| \leq n\}$.

Könnyen nyerhetünk alsó korlátot bármely kód tömörségére. Jelölje $\mathcal{L}_n$ az $\mathcal{L}$ nyelv legfeljebb n hosszúságú szavainak halmazát. Ekkor nyilvánvaló, hogy

$$\eta(n) \geq \log_2 |\mathcal{L}_n|.$$

Ezt a becslést *információelméleti korlátnak* nevezzük.

Ez az alsó becslés (additív állandótól eltekintve) éles. Egyszerűen kódolhatunk minden $x \in \mathcal{L}$ szót azzal, hogy az $\mathcal{L}$ nyelv hányadik eleme a növekvő rendezésben. Ha az n hosszúságú x szó a t-edik elem, akkor ehhez $\log_2 t \leq \log_2 |\mathcal{L}_n|$ bit kell, meg még konstans számú bit (annak a programnak a leírása, mely Σ^* elemeit lexikografikusan sorba veszi, megnézi, hogy melyik tartozik $\mathcal{L}$-be, és ezek közül a t-ediket kinyomatja).

Érdekesebb kérdésekhez jutunk, ha kikötjük, hogy a szóból a kód, és megfordítva, a kódból a kódolt szó polinomiálisan kiszámítható legyen. Másszóval: keresünk olyan $\mathcal{L}'$ nyelvet, és két polinomiálisan kiszámítható függvényt:

$$f : \mathcal{L} \longrightarrow \mathcal{L}', \qquad g : \mathcal{L}' \longrightarrow \mathcal{L},$$

hogy $f \circ g = id_{\mathcal{L}}$ és melyre minden $x \in \mathcal{L}$-re $|f(x)|$ az $|x|$-hez képest „rövid". Egy ilyen függvény-párt *polinomiális idejű kódnak* nevezünk. (Természetesen tekinthetnénk a polinomiális időkorlát helyett más bonyolultsági megszorítást is.)

Bemutatunk néhány példát arra, amikor polinomiális idejű kód is közel tud kerülni a információelméleti korláthoz.

6.4.1 Példa: Korábban a 6.3.3 Lemma bizonyításában használtuk azon n hosszúságú 0-1 sorozatoknak, melyekben pontosan m 1-es van, azt az egyszerű kódolását, melynél egy sorozat kódja az, hogy lexikografikusan hányadik. Belátjuk, hogy ez a kódolás polinomiális.

0-1 sorozatok helyett egy n elemű halmaz részhalmazait tekintjük. Legyen $a_1 > a_2 > \ldots > a_m$ a $\{0, \ldots, n-1\}$ halmaz lexikografikusan t-edik m elemű részhalmaza, ahol minden részhalmazt csökkenően rendezünk a lexikografikus rendezéshez. Ekkor

$$t = 1 + \binom{a_1}{m} + \binom{a_2}{m-1} + \ldots + \binom{a_m}{1} (*)$$

Valóban, ha $\{b_1, \ldots, b_m\}$ az $\{a_1, \ldots, a_m\}$ halmazt lexikografikusan megelőzi, akkor valamely i-re $b_i < a_i$ de minden $j < i$ esetén $b_j = a_j$. Az ilyen tulajdonságú $\{b_1, \ldots, b_m\}$ halmazok száma éppen $\binom{a_i}{m-i+1}$. Ha ezt minden i-re összegezzük, megkapjuk a $(*)$ formulát.

A $(*)$ formula n-ben polinomiális időben könnyen kiszámítható. Megfordítva, ha $t \leq \binom{n}{m}$ adott, akkor t könnyen felírható $(*)$ alakban: bináris kereséssel megkeressük a legnagyobb a_1 természetes számot, melyre $\binom{a_1}{m} \leq t-1$, majd a legnagyobb a_2-t, melyre $\binom{a_2}{m-1} \leq t - 1 - \binom{a_1}{m}$ stb. Ezt csináljuk m lépésig. Az így kapott számokra fennáll, hogy

$a_1 > a_2 \ldots$; valóban, például a_1 definíciója szerint $\binom{a_1+1}{m} = \binom{a_1}{m} + \binom{a_1}{m-1} > t-1$, így $\binom{a_1}{m-1} > t-1-\binom{a_1}{m}$, ahonnan $a_1 > a_2$. Hasonlóan adódik, hogy $a_m \geq 0$ és hogy m lépés után nincs „maradék", vagyis $(*)$ fönnáll. Így tehát adott t-re polinomiális időben kiszámítható, hogy melyik a lexikografikusan t-edik m elemű részhalmaz.

6.4.2 Példa: Tekintsük a fákat, pl. adjacencia-mátrixukkal megadva (de lehetne más „értelmes" leírás is). Így a fa pontjainak adott sorrendje van, amit úgy is fogalmazhatunk, hogy a fa pontjai 0-tól $(n-1)$-ig meg vannak számozva. Két fát akkor tekintünk azonosnak, ha valahányszor az i-edik és j-edik pontok össze vannak kötve az elsőben, akkor össze vannak kötve a másodikban is és viszont (így ha egy fa pontjait átszámozzuk, esetleg különböző fához jutunk). Az ilyen fákat *számozott fáknak* nevezzük.

Nézzük meg először, mit mond az információelméleti alsó becslés, vagyis hány fa van. Erre vonatkozik a következő klasszikus eredmény:

6.4.3 Cayley Tétele: *Az n pontú számozott fák száma n^{n-2}.*

Így az információelméleti becslés szerint bármilyen kódolásnál legalább egy n pontú fának legalább $\lceil \log(n^{n-2}) \rceil = \lceil (n-2)\log n \rceil$ hosszú kódja kell, hogy legyen. Vizsgáljuk meg, hogy ez az alsó korlát elérhető-e polinomiális idejű kóddal.

(a) Ha a fákat adjacencia-mátrixukkal kódoljuk, az n^2 bit.

(b) Jobban járunk, ha minden fát az éleinek a fölsorolásával adunk meg. Ekkor minden csúcsnak egy „nevet" kell adni; mivel n csúcs van, adhatunk minden csúcsnak egy-egy $\lceil \log n \rceil$ hosszúságú 0–1 sorozatot név gyanánt. Minden élt két végpontjával adunk meg. Így az élek fölsorolásához kb. $2(n-1)\log_2 n$ bit kell.

(c) Megtakaríthatunk egy 2-es faktort (b)-ben, ha a fában kitüntetünk egy gyökeret, mondjuk a 0 pontot, és a fát azzal az $(\alpha(1), \ldots, \alpha(n-1))$ sorozattal adjuk meg, melyben $\alpha(i)$ az i pontból a gyökérhez vezető út első belső pontja (az i „apja"). Ez $(n-1)\lceil \log_2 n \rceil$ bit, ami már majdnem optimális.

(d) Ismeretes azonban olyan eljárás is, az ún. Prüfer-kód, mely bijekciót létesít az n pontú számozott fák és a $0, \ldots, n-1$ számok $n-2$ hosszúságú sorozatai között. (Ezzel Cayley tételét is bizonyítja.) Minden ilyen sorozatot tekinthetünk egy természetes szám n alapú számrendszerbeli alakjának; így az n pontú számozott fákhoz egy-egy 0 és n^{n-2} közötti „sorszámot" rendelünk. Ezeket a „sorszámokat kettes számrendszerben kifejezve olyan kódolást kapunk, mely minden fa kódja legfeljebb $\lceil (n-2)\log n \rceil$ hosszúságú.

A Prüfer-kód a (c) eljárás finomításának tekinthető. Az ötlet az, hogy az $[i, \alpha(i)]$ éleket nem i nagysága szerint rendezzük, hanem kicsit másként. Definiáljuk az $(i_1, \ldots, i_n)$ permutációt a következőképpen: legyen i_1 a fa legkisebb végpontja; ha $i_1, \ldots, i_k$ már definiálva vannak, akkor legyen i_{k+1} az $i_1, \ldots, i_k$ pontok elhagyása után visszamaradó gráf legkisebb végpontja. (A 0-t nem tekintjük végpontnak.) Legyen $i_n = 0$. Az így definiált i_k-kal tekintsük az $(\alpha(i_1), \ldots, \alpha(i_{n-1}))$ sorozatot. Ennek az utolsó eleme 0 (ugyanis az i_{n-1} pont „apja" csak az i_n lehet), így ez nem érdekes. A maradék $(\alpha(i_1), \ldots, \alpha(i_{n-2}))$ sorozatot nevezzük a fa *Prüfer kódjának.*

Állítás: *A fa Prüfer-kódja meghatározza a fát.*

Ehhez elegendő belátni, hogy a Prüfer-kód az $i_1, \ldots, i_n$ sorozatot meghatározza; tudniillik akkor már ismerjük a fa éleit (az $[i_k, \alpha(i_k)]$ párokat). Az i_1 a fa legkisebb végpontja, így meghatározásához elegendő azt megmutatni, hogy a Prüfer-kódról leolvasható, hogy melyek a végpontok. De ez nyilvánvaló: pontosan azok a pontok végpontok, melyek nem „apjai" más pontnak, tehát melyek nem fordulnak elő az $\alpha(i_1), \ldots, \alpha(i_{n-2}), 0$ számok között. Így tehát i_1 egyértelműen meg van határozva.

Feltéve, hogy $i_1, \ldots, i_{k-1}$-ről már tudjuk, hogy a Prüfer-kód egyértelműen meghatározza őket, az előző meggondoláshoz hasonlóan adódik, hogy i_k *a legkisebb olyan szám, mely nem fordul elő sem az* $i_1, \ldots, i_{k-1}$*, sem az* $\alpha(i_k), \ldots, \alpha(i_n)$ *számok között.* Így i_k is egyértelműen meg van határozva.

Állítás: *Minden* $(b_1, \ldots, b_{n-2})$ *sorozat* ($1 \leq a_i \leq n$) *fellép, mint egy fa Prüfer-kódja.*

Az előző bizonyítás ötletét felhasználva, legyen $b_{n-1} = 0$, és definiáljuk az $i_1, \ldots, i_n$ permutációt azzal a rekurzióval, hogy legyen i_k a legkisebb olyan szám, mely sem az $i_1, \ldots, i_{k-1}$, sem a $b_k, \ldots, b_n$ számok között nem fordul elő ($1 \leq k \leq n-1$), i_n pedig legyen 0. Kössük össze i_k-t b_k-val minden $1 \leq k \leq n-1$-re, és legyen $\gamma(i_k) = b_k$. Így egy $n-1$ élű G gráfot kapunk az $1, \ldots, n$ pontokon. Ez a gráf összefüggő, mert minden i-re $\gamma(i)$ később van az $i_1, \ldots, i_n$ sorozatban, mint i, és ezért az $(i, \gamma(i), \gamma(\gamma(i)), \ldots)$ sorozat egy olyan út, mely i-t a 0 ponttal köti össze. De ekkor G $n-1$ élű összefüggő gráf, tehát fa. Az, hogy G Prüfer-kódja éppen az adott $(b_1, \ldots, b_{n-2})$ sorozat, a konstrukcióból nyilvánvaló.

6.4.4 Példa: Tekintsük most a számozatlan fákat. Ezeket úgy definiálhatjuk, mint számozott fák ekvivalencia-osztályait, ahol két számozott fát ekvivalensnek tekintünk, ha *izomorfak*, vagyis alkalmas átszámozással ugyanaz a számozott fa válik belőlük. Feltesszük, hogy egy-egy ilyen ekvivalencia-osztályt egy elemével, vagyis egy számozott fával adunk meg (hogy konkrétan melyikkel, az most nem lényeges). Mivel minden számozatlan fa legfeljebb $n!$ féleképpen számozható (számozásai nem biztos, hogy mind különbözők, mint számozott fák!), ezért a számozatlan fák száma legalább $n^{n-2}/n! \leq 2^{n-2}$. (Pólya György egy nehéz eredménye szerint az n pontú fák száma aszimptotikusan $c_1 c_2^n n^{3/2}$, ahol c_1 és c_2 bonyolultan definiálható konstansok.) Így az információelméleti alsó korlát legalább $n-2$.

Másrészt a következő kódolási eljárást alkalmazhatjuk. Legyen adott egy n pontú F fa. Járjuk be F-et a „hosszában keresés" szabálya szerint: Legyen x_0 a 0 számú pont, és definiáljuk az $x_1, x_2, \ldots$ pontokat a következőképpen: Ha van x_i-nek olyan szomszédja, mely még nem szerepel a sorozatban ($i \geq 0$), akkor legyen x_{i+1} ezek közül a legkisebb számú. Ha nincs, és $x_i \neq x_0$, akkor legyen x_{i+1} az x_i-ből x_0-ban vezető úton az x_i szomszédja. Végül ha $x_i = x_0$ és minden szomszédja szerepelt már a sorozatban, akkor megállunk.

Könnyű belátni, hogy az így definiált sorozatra az $[x_i, x_{i+1}]$ párok között a fa minden éle szerepel, méghozzá mindkét irányban pontosan egyszer. Ebből következik, hogy a sorozat hossza pontosan $2n-1$. Legyen mármost $\epsilon_i = 1$, ha x_{i+1} messzebb van a gyökértől, mint x_i, és $\epsilon_i = 0$ egyébként. Könnyű meggondolni, hogy az $\epsilon_0 \epsilon_1 \ldots \epsilon_{2n-3}$ sorozat egyértelműen meghatározza a fát; a sorozaton végighaladva, lépésről lépésre megrajzolhatjuk a gráfot, és megkonstruálhatjuk az $x_1 \ldots x_i$ sorozatot. Az $(i+1)$-edik lépésben, ha $\epsilon_i = 1$, akkor egy új pontot veszünk föl (ez lesz x_{i+1}), és összekötjük x_i-vel; ha $\epsilon_i = 0$, akkor x_{i+1} legyen az x_i-nek az x_0 „felé" eső szomszédja.

Megjegyzések: 1. Ennél a kódolásnál egy fához rendelt kód függ a fa számozásától, de nem határozza azt meg egyértelműen (csak a számozatlan fát határozza meg).

2. A kódolás nem bijektív: nem minden 0-1 sorozat lesz egy számozatlan fa kódja. Észrevehetjük, hogy

(a) minden kódban ugyanannyi 1-es van, mint 0, továbbá

(b) minden kód minden kezdőszeletében legalább annyi 1-es van, mint 0

(az 1-esek és 0-k számának különbsége az első i szám között az x_i pontnak a 0-tól való távolságát adja meg). Könnyű belátni, hogy megfordítva, minden (a)-(b) tulajdonságú 0-1 sorozathoz található olyan számozott fa, melynek ez a kódja. Nem biztos azonban, hogy ez a fa, mint számozatlan fa, éppen ezzel a számozással van adva. Ezért a kód még az (a)-(b) tulajdonságú szavakat sem használja mind fel (attól függően, hogy az egyes számozatlan fákat mely számozásukkal adtuk meg).

3. Az (a)-(b) tulajdonságú $2n-2$ hosszúságú 0-1 sorozatok száma ismert kombinatorikai tétel szerint $\frac{1}{n}\binom{2n-2}{n-1}$. Megfogalmazhatunk olyan fa-fogalmat, melynek éppen az (a)-(b) tulajdonságú sorozatok felelnek meg: ezek a *gyökeres síkfák*, melyek kereszteződés nélkül vannak lerajzolva a síkban úgy, hogy egy speciális csúcsuk — a gyökerük — a lap bal szélén van. Ez a lerajzolás minden csúcs fiai (a gyökértől távolabb levő szomszédjai) között egy rendezést ad meg „fölülről lefelé"; a lerajzolást ezekkel a rendezésekkel jellemezzük. A fent leírt kódolás gyökeres síkfákban is elvégezhető és bijekciót hoz létre köztük és az (a)-(b) tulajdonságú sorozatok között.

7. fejezet

Pszeudo-véletlen számok

7.1. Bevezetés

Mint láthattuk, számos fontos algoritmus használ véletlen számokat (vagy ezzel ekvivalensen független véletlen biteket). De vajon hogyan tehetünk szert ilyen bitekre?

Egyik lehetséges forrás a számítógépen kívülről jövő. Valódi véletlen számsorozatokat kaphatunk például a radioaktív bomlásból. Ám a legtöbb esetben ez nem működik, mert nincs olyan gyors fizikai eszközünk ami a számítógépek sebességével generál az érmefeldobásnak megfelelő véletlen biteket.

Így kénytelenek vagyunk számítógéppel előállítani ezeket. Viszont a véletlen információelméleti fogalmából következik, hogy egy hosszú sorozat, melyet egy rövid program generál, sosem lehet valódi véletlen sorozat. Ezért kénytelenek vagyunk olyan algoritmusokat használni melyek véletlen-szerű sorozatokat állítanak elő. Ugyanakkor Neumann (az első matematikusok egyike aki ezek használatát javasolta) megjegyezte, hogy bárki aki használja ezen véletlennek kinéző sorozatokat, szükségszerűen „bűnt" követ el. Ebben a fejezetben megértjük, hogy miként védhetjük ki ennek a súlyosabb következményeit.

A gyakorlati célokon kívül más okok miatt is vizsgálják a pszeudo-véletlen szám generátorokat. Gyakran meg akarunk ismételni valamilyen számítást. Ennek különböző okai lehetnek, egyikük a hibák ellenőrzése. Ebben az esetben, ha a véletlen számaink forrása valódi véletlen volt, akkor egyetlen lehetőségünk hogy újra ugyanazt a számítást elvégezhessük, hogy eltároljuk ezeket. Ez viszont esetleg sok tárhelyet igényel. Pszeudo-véletlen számok esetén nem ez az eset, hiszen elég a „magot" eltárolnunk, ami sokkal kisebb helyen elfér. Egy másik sokkal fontosabb indok, hogy bizonyos alkalmazásoknál csak annyit akarunk elérni, hogy a sorozat egy olyan valaki számára „látsszon véletlennek", aki nem tudja hogyan generáltuk. Ezen alkalmazások összességét hívjuk kriptográfiának, melyet egy későbbi fejezetben tárgyalunk.

A *pszeudo-véletlen bitgenerátorok* azon az elven működnek, hogy egy „magnak" nevezett rövid sorozatból csinálnak egy hosszabb pszeudo-véletlen sorozatot. Megköveteljük, hogy polinom időben működjön az algoritmus. Az eredményül kapott sorozatnak „véletlenszerűnek" kell lennie, és ami a legfontosabb, hogy ez pontosan definiálható. Durván fogalmazva, nem szabad léteznie olyan polinom idejű algoritmusnak, amely megkülönböztetné egy valódi véletlen sorozattól. Egy másik tulajdonság, melyet gyakran könnyebb ellenőrizni, hogy ne tudja egy algoritmus se megjósolni egy bitjét sem az előzőek ismeretében. Be fogjuk

látni, hogy ez a két feltétel ekvivalens.

Miként készíthetünk véletlen-generátort? Több különböző ad hoc algoritmusról (pl. hogy egy adott egyenlet megoldásának kettes számrendszerbeli alakjában vesszük a biteket) kiderül, hogy az általuk generált sorozatok nem teljesítik az elvárt feltételeinket. Egy általános módszer, mely ilyen sorozatokat ad, az *egyirányú függvényeken* alapul. Ezek olyan függvények melyeket könnyű kiszámítani, de nehéz megfordítani. Miközben ilyen függvények létezése nem bizonyított (ebből következne, hogy P különbözik NP-től), számos jelölt van, mely biztonságos, legalábbis a jelenlegi technikákkal szemben.

7.2. Klasszikus módszerek

Számos „véletlenszerű" bitsorozatot előállító klasszikus módszert ismerünk. Ezek egyike sem teljesíti elvárásainkat, melyeket a következő fejezetben fogalmazunk meg pontosan. Ennek ellenére, hatékonyságuk és egyszerűségük folytán (főként a lineáris kongruencia generátorok, lásd 2. példa lent) jól használhatók a gyakorlatban. Rengeteg praktikus információ létezik ezek paramétereinek legmegfelelőbb megválasztásáról, melyekbe nem mélyedünk bele, hanem Knuth könyvének 2. kötetére hivatkozunk.

1. Példa. Az eltolás regiszterek a következő módon vannak definiálva. Legyen $f : \{0,1\}^n \to \{0,1\}$ egy könnyen számolható függvény. Egy n bites $a_0, a_1, \ldots, a_{n-1}$ *magból* kiindulva, kiszámoljuk a $a_n, a_{n+1}, a_{n+2}, \ldots$ biteket az

$$a_k = f(a_{k-1}, a_{k-2}, \ldots, a_{k-n})$$

rekurzióval. Az „eltolás regiszter" név abból a tulajdonságból származik, hogy csak $n+1$ bitet kell eltárolnunk: miután eltároltuk $f(a_0, \ldots, a_{n-1})$-t a_n-ben, nincs szükségünk többé a_0-ra és eltolhatjuk a_1-t a_0-ba, a_2-t a_1-be, stb. Legfontosabb eset, mikor f egy lineáris függvény a 2-elemű test felett, a továbbiakban ezzel foglalkozunk.

Bizonyos esetekben a lineáris eltolás regiszter által generált bitek véletlennek látszanak, legalábbis egy darabig. Természetesen az $a_0, a_1, \ldots$ sorozatban egy idő után ismétlődik valamely n egymásutáni bit, és innentől periodikus lesz a sorozat. Ennek viszont nem kell a_{2^n} előtt bekövetkeznie, és valóban, választható olyan lineáris függvény, melyre a sorozat periódusa 2^n.

A problémát az jelenti, hogy a sorozatnak van más rejtett szerkezete is a periodicitáson kívül. Valóban, legyen

$$f(x_0, \ldots, x_{n-1}) = b_0 x_0 + b_1 x_1 + \ldots b_{n-1} x_{n-1}$$

(ahol $b_i \in \{0,1\}$). Tegyük fel nem ismerjük a $b_0, \ldots, b_{n-1}$ együtthatókat, viszont az eredményül kapott sorozatnak ismerjük az első n bitjét $(a_n, \ldots, a_{2n-1})$. Ekkor a következő lineáris egyenletrendszer meghatározza a b_i-ket:

$$\begin{aligned} b_0 a_0 + b_1 a_1 + \ldots b_{n-1} a_{n-1} &= a_n \\ b_0 a_1 + b_1 a_2 + \ldots b_{n-1} a_n &= a_{n+1} \\ &\vdots \\ b_0 a_{n-1} + b_1 a_n + \ldots b_n a_{2n-2} &= a_{2n-1} \end{aligned}$$

Van n egyenletünk n ismeretlennel (az egyenletek a 2-elemű test felett vannak). Miután meghatároztuk a b_i-ket, meg tudjuk mondani minden további $a_{2n}, a_{2n+1}, \ldots$ elemét a sorozatnak.

Előfordulhat persze, hogy az egyenletrendszer megoldása nem egyértelmű, mert az egyenletek összefüggenek. Például, ha a $0, 0, \ldots, 0$ magból indultunk, akkor az egyenletek nem mondanak semmit. Megmutatható viszont, hogy egy véletlen magból indulva az egyenletek pozitív valószínűséggel meghatározzák a b_i-ket. Tehát a sorozat első $2n$ elemének ismeretében a többi „nem látszik véletlennek" egy olyan megfigyelő számára, aki egy viszonylag egyszerű (polinom idejű) számítást akar csak elvégezni.

2. Példa. A gyakorlatban legfontosabb pszeudo-véletlen szám generátorok a *lineáris kongruencia generátorok*. Egy ilyen generátor három pozitív egész paraméterrel van megadva (a, b és m). Az X_0 magból kiindulva, melyre $0 \leq X_0 \leq m-1$, a generátor az $X_1, X_2, \ldots$ egészeket a

$$X_i = aX_{i-1} + b \pmod{m}.$$

rekurzióval számolja. Eredményként használhatjuk az X_i-ket, vagy pl. a középső bitjeiket.

Az derül ki, hogy ezen generátorok által kapott sorozatok is megjósolhatók polinom idejű számolással, polinom darab sorozatelem felhasználásával. Ám ezen algoritmusok nagyon bonyolultak, így gyorsaságuk és egyszerűségük miatt a lineáris kongruencia generátorok jók a legtöbb gyakorlati alkalmazásban.

3. Példa. Utolsó példaként nézzük a kettes számrendszerbeli alakját pl. a $\sqrt{5}$-nek:

$$\sqrt{5} = 10.001111000110111\ldots$$

Ez a sorozat eléggé véletlenszerűnek látszik. Természetesen nem használhatjuk mindig ugyanazt a számot, de választhatunk mondjuk egy n-bites a egészet és ekkor legyen a kimenet $\sqrt{a} - \lfloor\sqrt{a}\rfloor$. Sajnos ez a módszer is „feltörhető" nagyon bonyolult (viszont polinom idejű) algoritmikus számelméleti módszerekkel.

7.3. A pszeudo-véletlen szám generátor fogalma

Általánosságban, egy pszeudo-véletlen bitgenerátor átalakít egy rövid, valóban véletlen s sorozatot (a „magot") egy hosszabb $g(s)$ sorozattá, amelyik még mindig „véletlenszerű". Hogy mennyire jól használható a $g(s)$ egy valódi véletlen sorozat helyett, attól függ, hogy mennyire szigorúan teszteli az alkalmazás $g(s)$ véletlen mivoltát. Ha az alkalmazás képes tesztelni minden lehetséges magot, mely $g(s)$-t generálhatta, akkor megtalálja az igazit is, és nem sok véletlenszerűség marad. Ennek eléréséhez viszont az alkalmazás esetleg túl sokáig kéne hogy fusson. Akkor akarunk egy g-t pszeudo-véletlen bitgenerátornak hívni, ha semely polinom időben futó alkalmazás nem tudja megkülönböztetni $g(s)$-t egy valódi véletlen sorozattól.

A pontos definíció kimondásához előkészületekre van szükség. Azt mondjuk, hogy egy $f : \mathbb{Z}_+ \to \mathbb{R}$ függvény *elhanyagolható*, ha minden fix k-ra $n \to \infty$ esetén $n^k f(n) \to 0$. Szavakkal, f gyorsabban tart 0-hoz mint bármelyik polinom reciproka. Jelöljük ezt (a nagy O-hoz hasonlóan) úgy, hogy

$$f(n) = \mathrm{NEGL}(n).$$

Figyeljük meg, hogy egy elhanyagolható függvény bármilyen polinommal megszorozva elhanyagolható marad, azaz

$$n^r \mathrm{NEGL}(n) = \mathrm{NEGL}(n)$$

minden fix r esetén.

Tekintsünk egy polinom időben számítható $G : \{0,1\}^* \to \{0,1\}^*$ függvényt, melyre feltesszük, hogy $|G(x)|$ csak $|x|$-től függ és $|x| \leq |G(x)| < |x|^c$ egy c konstansra. Egy ilyen függvényt hívunk *generátornak*. Legyen $\mathcal{A}$ egy polinom idejű véletlen algoritmus (Turing-gép), amely bármely x 0-1 sorozatot elfogadja bemenetként és kiszámít belőle egy $\mathcal{A}(x)$ bitet (a kimenetnél a 0 jelentése, hogy „nem véletlen", míg az 1 jelentése, hogy „véletlen" volt a bemenet). Rögzítsünk le egy $n \geq 1$ számot. Válasszuk x-et egyenletesen $\{0,1\}^n$-ből és y-t egyenletesen $\{0,1\}^N$-ből, ahol $N = |G(x)|$. Feldobunk egy érmét és az eredménytől függően vagy $G(x)$-et vagy y-t adjuk meg $\mathcal{A}$-nak bemenetként. Akkor hívjuk $\mathcal{A}$-t *sikeresnek*, ha vagy $G(x)$ volt a bemenet és 0 a kimenet vagy y volt a bemenet és 1 a kimenet.

Egy G generátor *(biztonságos) véletlen szám generátor*, ha minden polinom idejű véletlen $\mathcal{A}$ algoritmusra, melynek bemenete egy x 0-1 sorozat és kimenete egy $\mathcal{A}(x)$ bit, annak valószínűsége, hogy $\mathcal{A}$ sikeres, legfeljebb $1/2 + \mathrm{NEGL}(n)$. Ez a feltétel azt jelenti, hogy $G(x)$ átmegy minden „értelmes" (polinom időben számolható) teszten abban az értelemben, hogy annak valószínűsége, hogy a teszt felismeri, hogy $G(x)$ nem valódi véletlen, elhanyagolhatóan nagyobb csak $1/2$-nél (ami persze elérhető véletlen találgatással is). A definícióban a valószínűséget x és y véletlen választása, az érmefeldobás mely eldönti melyik lesz $\mathcal{A}$ bemenete és végül $\mathcal{A}$ belső érmefeldobásai szerint értjük.

Ez a feltétel annyira erős, hogy nem is tudjuk, hogy biztonságos véletlen szám generátor létezik-e egyáltalán (ha igen, akkor P $\neq$ NP, lásd a feladatokat). A következő fejezetben látni fogjuk, hogy bizonyos bonyolultság-elméleti feltételek esetén léteznek ilyen generátorok.

Ez a feltétel nagyon általános és nehéz ellenőrizni. Yao következő tétele megad egy gyakran kényelmesebb módszert, mellyel eldönthető, hogy egy függvény biztonságos véletlen szám generátor-e. Azt állítja, hogy ha a jóslásra használt algoritmus nem használ túl sok időt, akkor $G(x)$ bármely bitje szinte teljesen megjósolhatatlan az előző bitekből.

Azt mondjuk, hogy a g generátor *megjósolhatatlan*, ha a következő teljesül. Legyen $n \geq 1$ és válasszuk x-et egyenletesen $\{0,1\}^n$-ből. Legyen $g(x) = G_1 G_2 \dots G_N$. Ekkor minden polinom idejű véletlen $\mathcal{A}$ algoritmus esetén, mely egy $x \in \{0,1\}^i$ $(i \leq N)$ sorozatot fogad el bemenetként és egy bitet ad kimenetként, teljesül, hogy

$$\max_i \mathsf{P}\Big(\mathcal{A}(G_1 \dots G_i) = G_{i+1}\Big) = \frac{1}{2} + \mathrm{NEGL}(n). \tag{7.1}$$

Tehát $\mathcal{A}$-t arra próbáljuk használni, hogy megjósoljuk $G_1 \dots G_N$ mindegyik bitjét az előző bitekből. Ekkor csak elhanyagolhatóan nagyobb $1/2$-nél (amit véletlen találgatással is elérhetünk) annak az esélye, hogy ez sikerül.

7.3.1 Tétel: (A. Yao) *Egy g generátor akkor és csak akkor biztonságos véletlen szám generátor, ha megjósolhatatlan.*

Bizonyítás: I. Tegyük fel, hogy g nem megjósolhatatlan. Ekkor van egy polinom idejű véletlen $\mathcal{A}$ algoritmus és egy $k > 0$ konstans, és végtelen sok n értékre egy-egy $i < N$, hogy

$$\mathsf{P}\Big(\mathcal{A}(G_1 \dots G_i) = G_{i+1}\Big) > \frac{1}{2} + \frac{1}{n^k}.$$

(ahol $x \in \{0,1\}^n$ egy egyenletesen véletlenül választott sorozat és $g(x) = G_1 \dots G_N$).

Ekkor viszont a következő véletlenséget ellenőrző $\mathcal{B}$ tesztet tudjuk elvégezni az $y = y_1 \dots y_N$ sorozatra: ha $\mathcal{A}(y_1 \dots y_i) = y_{i+1}$, akkor y-t „nem véletlennek" mondjuk, ellenkező esetben pedig „véletlennek". Így $\mathcal{B}$-nek vagy egy $R_1 \dots R_N$ valódi véletlen sorozatot adva vagy pedig $g(x)$-et (mindkettőt 1/2-1/2 valószínűséggel), a sikeres válasz valószínűsége

$$\frac{1}{2}\mathsf{P}\Big(\mathcal{A}(R_1 \dots R_i) \neq R_{i+1}\Big) + \frac{1}{2}\mathsf{P}\Big(\mathcal{A}(G_1 \dots G_i) = G_{i+1}\Big)$$
$$\geq \frac{1}{2}\frac{1}{2} + \frac{1}{2}\left(\frac{1}{2} + \frac{1}{n^k}\right) = \frac{1}{2} + \frac{1}{2n^k}.$$

Miután ez nem elhanyagolhatóan nagyobb 1/2-nél, a generátor nem biztonságos.

II. Tegyük fel, hogy létezik egy olyan $\mathcal{A}$ algoritmus, mely végtelen sok n értékre megkülönbözteti a pszeudo-véletlen $g(x) = G_1 \dots G_N$ sorozatot a valódi véletlen $r = R_1 \dots R_N$ sorozattól, azaz az algoritmus sikerességének valószínűsége $1/2+n^{-k}$ valamilyen $k > 0$ konstanssal. Megmutatjuk, hogy ebben az esetben megjósolhatjuk néhány bitjét a $G_1 \dots G_N$ sorozatnak.

Az $\mathcal{A}$ algoritmus sikerességének valószínűsége

$$\frac{1}{2}\mathsf{P}(\mathcal{A}(r) = 1) + \frac{1}{2}\mathsf{P}(\mathcal{A}(g(x)) = 0)$$

(mivel a siker azt jelenti, hogy $\mathcal{A}$ elfogadja r-t, ha r volt a bemenete és elutasítja $g(x)$-et, ha $g(x)$ volt a bemenete). Ez egyenlő

$$\frac{1}{2} + \frac{1}{2}\Big(\mathsf{P}(\mathcal{A}(r) = 1) - \mathsf{P}(\mathcal{A}(g(x)) = 1)\Big),$$

és így

$$\mathsf{P}(\mathcal{A}(r) = 1) - \mathsf{P}(\mathcal{A}(g(x)) = 1) > \frac{2}{n^c}. \tag{7.2}$$

Most tekintsük az

$$y^i = G_1 \dots G_i R_{i+1} \dots R_N$$

kevert sorozatokat és adjuk ezeket $\mathcal{A}$-nak bemenetként. Mivel $y^0 = r$ és $y^N = g(x)$, ezért

$$\mathsf{P}(\mathcal{A}(y^0) = 1) - \mathsf{P}(\mathcal{A}(y^N) = 1) > \frac{2}{n^k}.$$

Ebből következik, hogy van egy olyan i index ($1 \leq i \leq N$), melyre

$$\mathsf{P}(\mathcal{A}(y^{i-1}) = 1) - \mathsf{P}(\mathcal{A}(y^i) = 1) > \frac{2}{Nn^k}. \tag{7.3}$$

Megmutatjuk, hogy i 1/2-nél nem elhanyagolhatóan nagyobb valószínűséggel megjósolható. A jósló algoritmus a következő: azt tippeljük, hogy

$$X = R_{i+1} \oplus \mathcal{A}(y^i)$$

a G_{i+1} értéke (ez az érték valóban egy véletlen polinom idejű algoritmussal számolható $G_1, \dots, G_i$-ből). A jelölés egyszerűsége kedvéért a továbbiakban $G = G_i$, $R = R_i$, és

$$A_0 = \mathcal{A}(G_1 \dots G_{i-1} 0 R_{i+1} \dots R_N), A_1 = \mathcal{A}(G_1 \dots G_{i-1} 1 R_{i+1} \dots R_N).$$

Vegyük észre, hogy G, X, R, A_0 és A_1 véletlen bitek, melyek általában nem függetlenek, de R független a többitől. Sőt, könnyen látszik, hogy

$$X = R(1 - A_1) + (1 - R)A_0.$$

Így $G = X$ (azaz sikeres volt a jóslatunk) ekvivalens azzal, hogy

$$G(R(1 - A_1) + (1 - R)A_0) + (1 - G)(1 - R(1 - A_1) - (1 - R)A_0) = 1,$$

vagy

$$\begin{aligned} GR(1 - A_1) \quad &+ \quad G(1 - R)A_0 \; + \; (1 - G) \\ &- \quad (1 - G)R(1 - A_1) - (1 - G)(1 - R)A_0 = 1. \end{aligned}$$

Miután a bal oldal mindig 0 vagy 1, ezért annak valószínűsége, hogy teljesül az egyenlőség, megegyezik a bal oldal várható értékével. Ez viszont

$$\begin{aligned} &\mathsf{E}\Big(GR(1 - A_1) \quad + \quad G(1 - R)A_0 + (1 - G) - (1 - G)R(1 - A_1) \\ &\qquad\qquad\qquad - \quad (1 - G)(1 - R)A_0\Big) \\ &= \mathsf{E}(GR(1 - A_1)) \quad + \quad \mathsf{E}(G(1 - R)A_0) \\ &\qquad\qquad\quad + \quad \mathsf{E}(1 - G) - \mathsf{E}((1 - G)R(1 - A_1)) \\ &\qquad\qquad\quad - \quad \mathsf{E}((1 - G)(1 - R)A_0). \end{aligned}$$

mivel R független a többi változótól, ezért helyettesíthetünk $\mathsf{E}(R) = 1/2$-t. így azt kapjuk, hogy

$$\begin{aligned} &\frac{1}{2}\Big(\mathsf{E}(G(1 - A_1)) + \mathsf{E}(GA_0) + 2\mathsf{E}(1 - G) - \mathsf{E}((1 - G)(1 - A_1)) - \mathsf{E}((1 - G)A_0)\Big) \\ &= \frac{1}{2}\mathsf{E}(1 + 2GA_0 - 2GA_1 - A_0 + A_1). \end{aligned}$$

Viszont (7.3) is kifejezhető ezekkel a bitekkel. Valójában $\mathcal{A}(y^i) = RA_1 + (1 - R)A_0$, és így

$$\mathsf{P}(\mathcal{A}(y^i) = 1) = \frac{1}{2}\mathsf{E}(A_0 + A_1).$$

Hasonlóan, $\mathcal{A}(y^{i+1}) = GA_1 + (1 - G)A_0$, és így

$$\mathsf{P}(\mathcal{A}(y^{i+1}) = 1) = \mathsf{E}(GA_1 + (1 - G)A_0).$$

Tehát (7.3)-ból következik, hogy

$$\frac{1}{2}\mathsf{E}(A_0 + A_1 - 2GA_1 - 2(1 - G)A_0) = \frac{1}{2}\mathsf{E}(A_1 - A_0 - 2GA_1 + 2GA_0) > \frac{2}{Nn^k}.$$

Ekkor viszont

$$\mathsf{P}(G = X) = \frac{1}{2}\mathsf{E}(1 + 2GA_0 - 2GA_1 - A_0 + A_1) > \frac{1}{2} + \frac{1}{Nn^k} > \frac{1}{2} + \frac{1}{n^{k+c}},$$

amiből látszik, hogy a jóslatunk sikerességének valószínűsége nemelhanyagolhatóan nagyobb a triviálisnál. □

Pszeudo-véletlen bitgenerátorok létezése következik bizonyos (nem bizonyított, de valószínűnek tűnő) bonyolultságelméleti feltételezésekből. Ezeket tárgyaljuk a következő fejezetben.

7.4. Egyirányú függvények

Az egyirányú függvény olyan függvény, melyet könnyű számolni, de nehéz megfordítani. A pontos definíció a következő:

Definíció. Egy $f : \{0,1\}^* \to \{0,1\}^*$ függvényt *egyirányúnak* nevezünk, ha

1. létezik egy $c \geq 1$ konstans, melyre $|x|^{1/c} < |f(x)| < |x|^c$;
2. $f(x)$ számolható polinom időben;
3. minden $\mathcal{A}$ polinom idejű véletlen algoritmusra, mely $0-1$ sorozatokat számol $0-1$ sorozatokból, és egy véletlen y sorozatra, mely egyenletesen véletlenül van választva $\{0,1\}^n$-ből, teljesül:

$$\mathsf{P}(f(\mathcal{A}(f(y))) = f(y)) = NEGL(n). \tag{7.4}$$

A 3. feltétel magyarázatra szorul. Értelmezhetjük úgy, hogy amennyiben egy n hosszú véletlen y sorozatra kiszámítjuk $f(y)$-t és ezután $\mathcal{A}$ segítségével megpróbáljuk kiszámítani $f(y)$ ősét, akkor a siker valószínűsége elhanyagolható. Vegyük észre, hogy nem tettük fel, hogy f invertálható, így nem írhatjuk azt, hogy $\mathcal{A}(f(y)) = y$.

De miért nem írhatjuk egyszerűen azt, hogy

$$\mathsf{P}(f(\mathcal{A}(z)) = z) = NEGL(n) \tag{7.5}$$

egy egyenletesen véletlenül választott z-re? A lényeg, hogy mivel f nem feltétlenül ráképezés, ezért lehet hogy a legtöbb z sorozatot nem is veszi fel f. Ekkor ez a valószínűség kicsi lenne még akkor is, ha minden z-re amit felvesz f, az ős könnyen számolható. Ezért (7.4) azokat az eseteket nézi, amikor van ős, és azt írja elő, hogy ezekre nehéz az ős meghatározása.

Az *egyirányú permutáció* egy olyan egyirányú függvény, mely egy-egy értelmű és $|f(x)| = |x|$ minden x esetén. Világos, hogy ebben az esetben (7.5) és (7.4) ekvivalensek.

7.5.1 Tétel: *Legyen g egy biztonságos véletlen szám generátor és tegyük fel, hogy $|g(x)| \geq 2|x|$.* Ekkor g egyirányú.

Bizonyítás: Tegyük fel, hogy g mégsem egyirányú. Ekkor létezik egy $k > 0$ konstans, egy polinom idejű véletlen $\mathcal{A}$ algoritmus és végtelen sok n érték, melyre teljesül, hogy egy egyenletesen véletlenül választott $y \in \{0,1\}^n$-ra

$$\mathsf{P}(g(\mathcal{A}(g(y))) = g(y)) > \frac{1}{n^k}.$$

Tekintsük a következő véletlenséget ellenőrző $\mathcal{B}$ algoritmust: egy $z \in \{0,1\}^N$ sorozatot „nem véletlennek” nevezünk, ha $g(\mathcal{A}(z)) = z$, egyébként pedig „véletlennek”. Ha $\mathcal{B}$-nek bemenetként megadunk egy valódi véletlen $r = R_1 \ldots R_N$ véletlen sorozatot vagy pedig $g(x)$-et (mindkettőt 1/2-1/2 valószínűséggel), akkor a sikeresség valószínűsége

$$\frac{1}{2}\mathsf{P}(g(\mathcal{A}(r)) \neq r) + \frac{1}{2}\mathsf{P}(g(\mathcal{A}(g(x))) = g(x)).$$

Az első tag nagyon közel van 1/2-hez. Valóban, az összes $g(x)$-szel megegyező hosszú sorozatok száma 2^n, tehát annak valószínűsége, hogy r ezek egyike, $2^{n-N} \leq 2^{-n}$, amennyiben $N \geq 2n$ (és persze ha r hossza megfelelő, még akkor sem biztos, hogy egyenlő $g(\mathcal{A}(r))$-rel). A második tag legalább $1/n^k$ a feltétel alapján. A sikeresség valószínűsége tehát

$$\frac{1}{2}\left(1 - \frac{1}{2^n}\right) + \frac{1}{2}\frac{1}{n^k} > \frac{1}{2} + \frac{1}{4n^k},$$

ami nemelhanyagolhatóan nagyobb 1/2-nél. □

A tétel megfordításaként megmutatjuk, miként lehet egy egyirányú permutáció segítségével konstruálni egy biztonságos véletlen szám generátort. Ehhez szükségünk lesz egy Goldreichnek és Levinnek köszönhető lemmára. Két $u = u_1 \dots u_n$ és $v = v_1 \dots v_n$ 0-1 sorozathoz definiáljuk $b(u, v) = u_1 v_1 \oplus \dots \oplus u_n v_n$-t.

7.5.2 Lemma: *Legyen f egy egyirányú permutáció. Ekkor minden polinom idejű véletlen $\mathcal{A}$ algoritmus és $\{0,1\}^n$-ből egyenletesen véletlenül választott x, y sorozat esetén*

$$\mathsf{P}\Big(\mathcal{A}(x, y) = b(f^{-1}(x), y)\Big) = \frac{1}{2} + \mathrm{NEGL}(n).$$

Tehát a lemma szerint ha kiválasztjuk az f^{-1} bitjeinek egy véletlen részhalmazát és vesszük ezen bitek modulo 2 összegét, akkor ez nem számolható sikeresen 1/2-nél nemelhanyagolhatóan nagyobb valószínűséggel. Mivel a lemma bizonyítása elég bonyolult, ezért ezt elhagyjuk.

7.5.3 Tétel: *Tegyük fel, hogy létezik egy egyirányú permutáció. Ekkor létezik biztonságos véletlen szám generátor is.*

Bizonyítás: Konstruálunk egy véletlen szám generátort. Legyen f egy egyirányú függvény és $k > 0$ konstans. Húzzunk szét egy $x = (x_1, \dots, x_{2n})$ $2n$ hosszú véletlen sorozatot (a sorozat hossza páros a kényelem kedvéért) egy $N = n^k$ hosszú pszeudo-véletlen sorozattá a következőképpen. Számoljuk ki a

$$y_1^t \dots y_n^t = f^t(x_1, \dots, x_n)$$

sorozatokat $t = 1, \dots, N$-re (itt f^t az f t-szeres iterációja) és legyen

$$g_t = y_1^t x_{n+1} \oplus \dots \oplus y_n^t x_{2n}.$$

Megmutatjuk, hogy $g(x) = g_N g_{N-1} \dots g_1$ egy biztonságos véletlen szám generátort definiál (figyeljük meg, hogy a biteket fordított sorrendben írtuk!). A 7.3.1 Tételt felhasználva elég annyit bizonyítanunk, hogy minden $1 \leq i \leq N$ és minden polinom idejű véletlen $\mathcal{A}$ algoritmus esetén, 1/2-nél csak elhanyagolhatóan nagyobb annak valószínűsége, hogy $\mathcal{A}$ kimenete éppen g_i, ha a bemenete $g_N \dots g_{i+1}$. Legyünk nagylelkűek és engedjük meg az algoritmusnak, hogy ne csak ezen biteket használhassa, hanem az $f^t(x_1, \dots, x_n)$ sorozatot is minden $t \geq i + 1$-re, sőt $x_{n+1} \dots x_{2n}$-t is (amiből $g_N \dots g_{i+1}$ könnyen számolható). Ekkor viszont nem kell megadnunk az $f^{i+2}(x_1, \dots, x_n)$, …, $f^N(x_1, \dots, x_n)$ sorozatokat, mert ezek könnyen számolhatók $f^{i+1}(x_1, \dots, x_n)$-ből. Így feltehetjük, hogy az

algoritmus $z = f^{i+1}(x_1, \ldots, x_n)$-t és $u = x_{n+1} \ldots x_{2n}$-t kapja meg és ebből kell kitalálnia $g_i = b(f^{-1}(z), u)$-t. Ekkor viszont a sikeresség valószínűsége az előző lemma miatt 1/2-nél elhanyagolhatóan nagyobb. □

7.5. Egyirányú függvény jelöltek

A számelmélet több egyirányú függvény jelöltet is a rendelkezésünkre bocsát. A továbbiak során a bemenetek és kimenetek hossza nem pontosan n, csak polinomiális n-ben.

A faktorizációs probléma. Jelöljön x két n hosszú prímet (mondjuk a prímségük bizonyításával együtt). Legyen $f(n, x)$ ezen két prím szorzata. Ez a probléma sok speciális esetben megoldható polinom időben, de az esetek többsége továbbra is nehéz.

A diszkrét logaritmus probléma. Adott egy p prím, p-nek egy g primitív gyöke és egy $i < p$ pozitív egész, a kimenet pedig p, g és $y = g^i \bmod p$. Ennek a megfordítását nevezik diszkrét logaritmus problémának, mivel adott p, g, y esetén i-t keressük, amit y-nak p-re vonatkozó *diszkrét logaritmusának* vagy *indexének* is hívnak.

A diszkrét négyzetgyök probléma. Adottak az m és $x < m$ pozitív egészek, a függvény kimenete m és $y = x^2 \bmod m$. Ennek megfordítása, hogy találjuk meg azt az x számot, melyre $x^2 \equiv y \pmod{m}$. Ez polinom időben megoldható egy véletlen algoritmussal, amennyiben m prím, általában viszont nehéz megoldani.

7.6. Diszkrét négyzetgyökök

Ebben a fejezetben a négyzetgyökvonás számelméleti algoritmusát tárgyaljuk.

A $0, 1, \ldots, p-1$ egészeket (modulo p) *maradékoknak* hívjuk. Legyen p páratlan prím. Egy y-t akkor hívunk az x (modulo p) *négyzetgyökének*, ha

$$y^2 \equiv x \pmod{p}.$$

Ha x-nek van négyzetgyöke, akkor *kvadratikus maradéknak* hívjuk. Ha nincs, akkor pedig *kvadratikus nemmaradéknak*.

Világos, hogy a 0-nak csak egy négyzetgyöke van modulo p, hiszen ha $y^2 \equiv 0 \pmod{p}$, akkor $p|y^2$ és mivel p prím, ezért $p|y$ is teljesül. Minden más egész x esetén, ha y az x négyzetgyöke, akkor $p - y \equiv -y \pmod{p}$ is az és nincsen más. Valóban, ha $z^2 \equiv x$ valamely z maradékkal, akkor $p|y^2 - z^2 = (y - z)(y + z)$ és így $p|y - z$ vagy $p|y + z$. Tehát $z \equiv y$ vagy $z \equiv -y$.

A fentiekből következik, hogy nem minden egésznek van négyzetgyöke modulo p, hiszen a négyzetre emelés a nemnulla maradékokat egy $(p-1)/2$ elemű részhalmazra képezi, így a többi $(p-1)/2$ maradéknak nincs négyzetgyöke.

A következő lemma megad egy egyszerű módszert annak eldöntésére, hogy egy maradéknak van-e négyzetgyöke.

7.6.1 Lemma: *Egy x maradéknak akkor és csak akkor van négyzetgyöke, ha*

$$x^{(p-1)/2} \equiv 1 \pmod{p}. \tag{7.6}$$

Bizonyítás: Ha x-nek van egy y négyzetgyöke, akkor

$$x^{(p-1)/2} \equiv y^{p-1} \equiv 1 \pmod{p}$$

következik a Kis Fermat Tételből. A fordított irány bizonyításához vegyük észre, hogy $x^{(p-1)/2} - 1$ foka $(p-1)/2$, ezért legfeljebb $(p-1)/2$ „gyöke" van modulo p (ezt hasonlóképpen bizonyíthatjuk, mint azt a jól ismert tételt, miszerint egy n-ed fokú polinomnak legfeljebb n valós gyöke van). Mivel minden kvadratikus maradék gyöke $x^{(p-1)/2} - 1$-nek, ezért egyetlen kvadratikus nemmaradék sem lehet az. □

De vajon hogyan találhatjuk meg ezt a négyzetgyököt? Némelyik prímre nem nehéz:

7.6.2 Lemma: *Tegyük fel, hogy $p \equiv 3 \pmod 4$. Ekkor minden x kvadratikus maradék esetén $x^{(p+1)/4}$ az x négyzetgyöke.*

Valóban,

$$\left(x^{(p+1)/4}\right)^2 = x^{(p+1)/2} = x \cdot x^{(p-1)/2} \equiv x \pmod{p}.$$

A $p \equiv 1 \pmod 4$ eset ennél nehezebb és a megoldás véletlent használ. Igazából a véletlenre csak a következő segédalgoritmusban van szükség.

7.6.3 Lemma: *Legyen p egy páratlan prím. Ekkor van olyan véletlen algoritmus mely polinom időben talál egy kvadratikus nemmaradékot modulo p.*

Ilyet úgy tudunk csinálni, hogy választunk egy véletlen $z \neq 0$ maradékot és teszteljük a 7.6.1 Lemma segítségével, hogy kvadratikus nemmaradék-e. Ha nem, akkor választunk egy új z-t. Mivel a találat valószínűsége $1/2$, ezért átlagosan 2 próbálkozással fogunk találni egyet.

Természetesen elkerülhetjük a véletlen használatát, ha sorban teszteljük a $2, 3, 5, 7, \ldots$ számokat. Előbb vagy utóbb találunk egy kvadratikus nemmaradékot. Azt viszont nem tudjuk, hogy a legkisebb kvadratikus nemmaradék megtalálható-e ily módon polinom időben. Sejtés, hogy $O(\log^2 p)$-nél több lépés nem kell ezzel a módszerrel sem.

Térjünk vissza arra a problémára, hogy miként találhatjuk meg egy x maradék négyzetgyökét, ha a p prímre $p \equiv 1 \pmod 4$. Ekkor van olyan q páratlan és $k \geq 2$ egész, hogy $p - 1 = 2^k q$.

Először keresünk egy z kvadratikus nemmaradékot. Az ötlet az, hogy keresünk egy páros hatvány z^{2t}-t, amire $x^q z^{2t} \equiv 1 \pmod p$. Ekkor $y \equiv x^{(q+1)/2} z^t \pmod p$ négyzetgyöke x-nek, hiszen

$$y^2 \equiv x^{q+1} z^{2t} \equiv x \pmod{p}.$$

Egy ilyen z-hatványt úgy találunk, hogy keresünk minden $j \leq k-1$ egészhez egy $t_j > 0$ egészet úgy, hogy

$$x^{2^j q} z^{2^{j+1} t_j} \equiv 1 \pmod{p}. \tag{7.7}$$

Ez $j = 0$-ra éppen megadja amit keresünk. Ha $j = k - 1$, akkor jó lesz a $t_{k-1} = q$:

$$x^{2^{k-1} q} z^{2^k q} = x^{(p-1)/2} z^{p-1} \equiv 1 \pmod{p},$$

miután x kvadratikus maradék és a Kis Fermat Tételből következően $z^{p-1} \equiv 1 \pmod{p}$. Ez azt sugallja, hogy visszafelé konstruáljuk meg a t_j-ket, azaz $j = k-2, k-3, \ldots$ sorrendben.

Tegyük fel, hogy már tudjuk t_j-t ($j > 0$), és t_{j-1}-t akarjuk megtalálni. Tudjuk, hogy

$$p \left| x^{2^j q} z^{2^{j+1} t_j} - 1 = \left(x^{2^{j-1} q} z^{2^j t_j} - 1 \right) \left(x^{2^{j-1} q} z^{2^j t_j} + 1 \right) \right.$$

Megnézzük, hogy a két tényező közül melyik osztható p-vel. Ha az első, akkor $t_{j-1} = t_j$ jó, ha a második, akkor legyen

$$t_{j-1} = t_j + 2^{k-j-1} q.$$

Ez jó választás, hiszen

$$x^{2^{j-1} q} z^{2^j t_{j-1}} = x^{2^{j-1} q} z^{2^j t_j + 2^{k-1} q} = x^{2^{j-1} q} z^{2^j t_j} z^{(p-1)/2} \equiv (-1)(-1) = 1,$$

mivel z kvadratikus nemmaradék.

Ezzel teljesen leírtuk az algoritmust.

Feladatok. 1. Mutassuk meg, hogy egy egész négyzetre emelése nem véletlen szám generátor.

2. Egy x sorozatra jelölje $rev(x)$ az x megfordítását. Mutassuk meg, hogy ha $g(s)$ biztonságos véletlen szám generátor, akkor $rev(g(s))$ is az.

3. Ha P = NP, akkor nem létezik egyirányú függvény.

4. Tekintsük a következő véletlen szám generátort. A generátor egy n bites x egészet vesz magnak és a kimenete $\lfloor x^3 / 10^n \rfloor$. Mutassuk meg, hogy ez a véletlen szám generátor nem biztonságos.

8. fejezet

Párhuzamos algoritmusok

A technológia fejlődése egyre nagyobb súllyal veti föl a párhuzamos számítások matematikai alapjainak kidolgozását. A nagy energiával folytatott kutatások dacára sincs azonban a párhuzamos számításoknak teljesen kiforrott modellje. A legfőbb probléma az egyes processzorok ill. részprogramok közötti kommunikáció modellezése: ez történhet közvetlen csatornákon, előre adott összeköttetések mentén, „rádióadásszerűen" stb.

Egy másik többféleképpen modellezhető kérdés az egyes processzorok óráinak összehangolása: ez történhet közös jelekkel, vagy egyáltalán nem.

Ebben a fejezetben csak egy modellt tárgyalunk, az ún. párhuzamos RAM-ot, mely bonyolultságelméleti szempontból talán a „legtisztább", és a legjobban ki van dolgozva. Az erre a speciális esetre elért eredmények mindazonáltal a számítások párhuzamosíthatóságának néhány alapkérdését exponálják. Az ismertetett algoritmusok pedig úgy tekinthetők, mint egy magas szintű nyelven megírt programok: ezeket a konkrét technológiai megoldásoknak megfelelően implementálni kell.

8.1. Párhuzamos RAM

Párhuzamos számítást végző gépek elméleti szempontból legtöbbet vizsgált matematikai modellje a párhuzamos RAM (PRAM). Ez $p \geq 1$ azonos RAM-ból (processzorból) áll. A gépek programtára közös, és van egy közös memóriájuk is, mely mondjuk az $x[i]$ rekeszekből áll (ahol i az egész számokon fut végig). Kényelmes lesz feltenni (bár nem volna okvetlenül szükséges), hogy saját tulajdonukban van végtelen sok $u[i]$ ($i \in \mathbb{Z}$) memóriarekeszük. Induláskor minden processzor $u[0]$ rekeszében a processzor sorszáma áll (ha ez nem volna, minden processzor ugyanazt csinálná!). A processzor írhat és olvashat a saját $u[i]$ rekeszeibe/-ből és a közös $x[i]$ memóriarekeszekbe/-ből. Másszóval, a RAM-nál megengedett utasításokhoz hozzá kell még venni az

$u[i] := 0;$ $u[i] := u[i] + 1;$ $u[i] := u[i] - 1;$
$u[i] := u[i] + u[j];$ $u[i] := u[i] - u[j];$ $u[u[i]] := u[j];$ $u[i] := u[u[j]];$
$u[i] := x[u[j]];$ $x[u[i]] := u[j];$ IF $u[i] \leq 0$ THEN GOTO p

utasításokat.

A *bemenetet* az $x[0], x[1], \ldots$ rekeszekbe írjuk. A bemenet és a (közös) program mellé meg kell adni azt is, hogy hány processzorral dolgozunk; ezt írhatjuk pl. az $x[-1]$ rekeszbe.

A processzorok párhuzamosan, de ütemre hajtják végre a programot. (Mivel a saját nevük is szerepet játszik, előbb-utóbb nem ugyanazt fogják számolni.) Logaritmikus költségfüggvényt használunk: pl. egy k egész számnak az $x[t]$ memóriarekeszbe való beírásának vagy kiolvasásának ideje a k és t jegyeinek együttes száma, vagyis kb. $\log|k| + \log|t|$. A következő ütem akkor kezdődik, ha az előző műveletet minden processzor végrehajtotta. Egy ütem idejét tehát a legtöbb időt igénybevevő processzor határozza meg. A gép akkor áll meg, ha minden processzor olyan programsorhoz ér, melyben nincsen utasítás. A *kimenet* az $x[i]$ rekeszek tartalma.

Egy fontos tisztázandó kérdés, hogy hogyan szabályozzuk a közös memória használatát. Mi történik, ha több processzor ugyanabba a rekeszbe akar írni vagy ugyanonnan akar olvasni? Az ilyen konfliktusok elkerülésére különböző konvenciók léteznek. Ezek közül hármat említünk:

- Nem szabad két processzornak ugyanabból a rekeszből olvasni vagy ugyanabba a rekeszbe írni. Ezt *teljesen konfliktusmentes* modellnek nevezzük (angolul *exclusive-read-exclusive-write*, rövidítve EREW). Ez úgy értendő, hogy a programozónak kell gondoskodnia arról, hogy ez ne következzen be; ha mégis megtörténne, a gép programhibát jelez.

- Talán legtermészetesebb az a modell, melyben megengedjük, hogy több processzor is ugyanazt a rekeszt olvassa, de ha ugyanoda akarnak írni, az már programhiba. Ezt *félig konfliktusmentes* modellnek nevezzük (*concurrent-read-exclusive-write*, CREW).

- Szabad több processzornak ugyanabból a rekeszből olvasni és ugyanabba a rekeszbe írni is, de csak akkor, ha ugyanazt akarják beírni. (A gép akkor áll le programhibával, ha két processzor ugyanabba a rekeszbe más-más számot akar beírni). Ezt *konfliktuskorlátozó* modellnek nevezzük (*concurrent-read-concurrent-write*, CRCW).

- Szabad több processzornak ugyanabból a rekeszből olvasni és ugyanabba a rekeszbe írni. Ha többen akarnak ugyanoda írni, a legkisebb sorszámúnak sikerül. Ezt a modellt *elsőbbségi modellnek* nevezzük (*priority concurrent-read-concurrent-write*, P-CRCW).

Feladatok.

1. a) Mutassuk meg, hogy két n hosszúságú 0-1 sorozatról n processzorral az elsőbbségi modellben $O(1)$ lépésben, a teljesen konfliktusmentes modellben $O(\log n)$ lépésben megállapítható, hogy lexikografikusan melyik a nagyobb.

b*) Mutassuk meg, hogy a teljesen konfliktusmentes modellben ehhez $\Omega(\log n)$ lépés kell is.

c*) Hány lépés kell a másik két modellben?

2. Mutassuk meg, hogy két n hosszúságú 0-1 sorozatnak, mint kettes számrendszerbeli számnak az összegét n^2 processzorral $O(1)$ lépésben ki lehet számítani az elsőbbségi modellben.

3. a) Mutassuk meg, hogy n darab legfeljebb n hosszúságú 0-1 sorozatnak, mint kettes számrendszerbeli számnak az összegét n^3 processzorral $O(\log n)$ lépésben ki lehet számítani az elsőbbségi modellben.

b*) Mutassuk meg, hogy ehhez n^2 processzor is elég.

c*) Csináljuk meg ugyanezt a teljesen konfliktusmentes modellben is.

Nyilvánvaló, hogy a fenti modellek egyre erősebbek, hiszen egyre többet engednek meg. Megmutatható azonban, hogy — legalábbis ha a processzorok száma nem túl nagy — a legerősebb elsőbbségi modellben elvégezhető számítások sem sokkal gyorsabbak a leggyöngébb teljesen konfliktusmentes modellben végezhetőknél. Erre vonatkozik a következő lemma:

8.1.1 Lemma: *Minden $\mathcal{P}$ programhoz létezik olyan $\mathcal{Q}$ program, hogy ha $\mathcal{P}$ egy bemenetből p processzorral t időben kiszámít egy kimenetet az elsőbbségi modellben, akkor a $\mathcal{Q}$ program $O(p^2)$ processzorral $O(t \cdot \log^2 p)$ időben a teljesen konfliktusmentes modellben számítja ki ugyanazt.*

(A PRAM-nál a processzorok számát nem csak azért kell megadni, mert ettől függ a számítás, hanem azért is, mert ez — az idő és tár mellett — a számítás igen fontos bonyolultsági mértéke. Ha ezt nem korlátozzuk, akkor igen nehéz feladatokat is nagyon röviden meg tudunk oldani. pl. egy gráf 3 színnel való színezhetőségét el tudjuk dönteni úgy, hogy az alaphalmaz minden színezéséhez és a gráf minden éléhez csinálunk egy processzort, mely megnézi, hogy az adott színezésben az adott él két végpontja különböző színű-e. Az eredményeket persze még összesíteni kell, de ez is megoldható $O(\log\ n)$ lépésben.)

Bizonyítás: A $\mathcal{P}$ programot végrehajtó minden processzornak meg fog felelni egy processzor a $\mathcal{Q}$ program végrehajtása során. Ezeket főnök-processzoroknak nevezzük. Ezenfelül a $\mathcal{Q}$ programban minden főnökprocesszornak p további „segédprocesszora" is lesz.

Az a konstrukció alapgondolata, hogy ami a $\mathcal{P}$ program futása során egy adott ütem után a z című memóriarekeszben van, az a $\mathcal{Q}$ program futása során a megfelelő ütemben a $2pz, 2pz+1, \ldots, 2pz+p-1$ című rekeszek mindegyikében meglegyen. Ha a $\mathcal{P}$ következő ütemében az i-edik processzor z című rekeszből kell, hogy olvasson, akkor a $\mathcal{Q}$ program következő ütemében az ennek megfelelő processzor olvashatja az $2pz+i$ című rekeszt. Ha pedig ebben az ütemben ($\mathcal{P}$ szerint) a z című rekeszbe kell, hogy írjon, akkor $\mathcal{Q}$ szerint az $2pz+i$ rekeszbe írhat. Ez biztosan elkerüli az összes konfliktust, mert a különböző processzorok modulo p különböző rekeszeket használnak.

Gondoskodni kell azonban arról, hogy a $\mathcal{Q}$ program futása során a $2pz, 2pz+1, \ldots, 2pz+p-1$ rekeszekbe mindegyikébe az a szám kerüljön, amit az elsőbbségi szabály a $\mathcal{P}$ program futásának megfelelő ütemében z-be ír. Ehhez a $\mathcal{P}$ futását szimuláló minden ütem után beiktatunk egy $O(\log\ p)$ „lépésből" álló fázist, amely ezt megvalósítja.

Először is minden olyan processzor, mely az előző ($\mathcal{P}$-t szimuláló) ütemben írt, mondjuk az $2pz+i$ című rekeszbe, ír egy 1-est az $2pz+p+i$ című rekeszbe. A következő „első lépésben" megnézi, hogy az $2pz+p+i-1$ rekeszben 1-es áll-e. Ha igen, elalszik ennek a fázisnak a hátralevő idejére. Ha nem, akkor ő ír oda 1-est, majd „fölébreszti" egy segédjét. Általában, a k-adik lépésben az i processzornak 2^{k-1} segédje lesz ébren (magát is beleértve), és ezek rendre az $2pz+p+i-2^{k-1}, \ldots, 2pz+p+i-(2^k-1)$ című rekeszeket olvassák le. Amelyik 1-est talál, elalszik. Amelyik nem talál 1-est, az 1-est ír, fölébreszt egy új segédet, azt 2^{k-1} hellyel „balra" küldi, míg maga 2^k hellyel „balra" lép. Amelyik segéd kiér a $[2pz+p, 2pz+2p-1]$ intervallumból, elalszik; ha egy főnök kiér ebből az intervallumból, akkor már tudja, hogy ő „győzött".

Könnyű meggondolni, hogy ha a $\mathcal{P}$ program megfelelő ütemében több processzor is írni akart a z rekeszbe, akkor az ezeknek megfelelő főnökök és segédeik nem kerülnek konfliktusba, mialatt az $[2pz+p, 2pz+2p-1]$ intervallumban mozognak. A k-adik lépésre

ugyanis az i processzor és segédei $2pz + p + i$-től lefelé 2^{k-1} egymásutáni helyre 1-t írtak. Minden tőlük jobbra induló processzor és annak minden segédje ebbe szükségképpen bele fog lépni, és ezért elalszik, mielőtt konfliktusba kerülhetne az i-edik processzor segédeivel. Azt is mutatja ez, hogy mindig egyetlenegy főnök fog győzni, éspedig a legkisebb sorszámú.

A „győzőnek" még az a dolga, hogy amit a megfelelő $2pz + i$ rekeszbe írt, azt most az egész $[2pz, 2pz + p - 1]$ intervallum minden rekeszébe beírassa. Ezt könnyű megtenni az előzőhöz nagyon hasonló eljárással: ő maga beírja a kívánt értéket az $2pz$ című rekeszbe, majd fölébreszt egy segédet; beírják a kívánt értéket az $2pz + 1$ és $2pz + 2$ rekeszekbe, majd fölébresztenek egy-egy segédet stb. Ha mindannyian kiértek a $[2pz, 2pz + p - 1]$ intervallumból, jöhet a következő szimuláló ütem.

A segédek fölébresztésének megtervezését az olvasóra bízzuk.

A fenti „lépések" végrehajtása több programsor végrehajtását jelenti, de könnyű látni, hogy már csak korlátos számúét, melyek költsége logaritmikus költségű modell esetén is csak $O(\log\ p{+}\log\ z)$. Így két szimuláló ütem között csak $O((\log\ p)(\log\ p{+}\log\ z))$ idő telik el. Mivel maga a szimulálandó ütem is beletelik $\log\ z$ időbe, ez csak $O(\log\ p)^2$-szeresére növeli a futási időt. □

A továbbiakban, ha külön nem mondjuk, a konfliktusmentes modellt (EREW) használjuk. Az előző lemma miatt nem jelentene lényeges különbséget az sem, ha bármilyen más megállapodást tennénk.

Az alábbi állítás könnyen belátható:

8.1.2 Állítás: *Ha egy számítás elvégezhető p processzorral t lépésben legfeljebb s jegyű számokkal, akkor minden $q \leq p$-re elvégezhető q processzorral $O(tp/q)$ lépésben legfeljebb $O(s + \log(p/q))$ jegyű számokkal. Speciálisan, elvégezhető szekvenciális (nem-párhuzamos)* RAM-*on $O(tp)$ lépésben $O(s + \log p)$ jegyű számokkal.* □

A párhuzamos algoritmusok komplexitáselméletének alapkérdése ennek a megfordítottja: tekintünk egy N idejű szekvenciális algoritmust, és szeretnénk ezt p processzoron „lényegében" N/p (mondjuk, $O(N/p)$) idő alatt megvalósítani. A következő pontban az e kérdés által motivált bonyolultsági osztályokat fogjuk áttekinteni.

A randomizálás, mint látni fogjuk, párhuzamos számítások esetén még fontosabb eszköz, mint a szekvenciális esetben. A *randomizált párhuzamos* RAM csak abban különbözik a fentebb bevezetett párhuzamos RAM-tól, hogy minden processzornak van egy további w rekesze, melyben mindig $1/2$ valószínűséggel 0 vagy 1 áll. Ha ezt kiolvassa a megfelelő processzor, akkor új véletlen bit jelenik meg benne. A véletlen bitek (egy processzoron belül is, meg a különböző processzoroknál is) teljesen függetlenek.

8.2. Az NC osztály

Egy $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ függvényt NC-*kiszámíthatónak* nevezünk, ha van olyan program a párhuzamos RAM-ra, és vannak olyan $c_1, c_2 > 0$ konstansok, hogy minden x bemenetre a program $f(x)$-et $O(|x|^{c_1})$ processzorral teljesen konfliktusmentesen $O((\log\ |x|)^{c_2})$ időben kiszámítja. (A 8.1.1 Lemma szerint nem változtatna ezen a definíción, ha pl. az elsőbbségi

modellt használnánk.)

Az NC nyelvosztály azokból az $\mathcal{L} \subseteq \{0,1\}^*$ nyelvekből áll, melyek karakterisztikus függvénye NC-algoritmussal kiszámítható. (Nem változtatna ezen a definíción, ha pl. az elsőbbségi modellt használnánk.)

Megjegyzés: Az NC osztály bevezetésének nem az a célja, hogy gyakorlatban elvégezhető párhuzamos számításokat modellezzen. Nyilvánvaló, hogy a gyakorlatban a logaritmikus időnél sokkal többet használhatunk föl, de (legalábbis a belátható jövőben) a polinomiálisnál sokkal kevesebb processzoron. A fogalom célja az, hogy leírja azokat a feladatokat, melyeket polinomiális számú művelettel ki lehet számolni, éspedig úgy, hogy ezeket szinte a lehető legjobban párhuzamosítjuk (n hosszúságú bemenet esetén a teljesen konfliktusmentes gépen már ahhoz is $\log\ n$ idő kell, hogy az adatokat kapcsolatba hozzuk egymással).

Nyilvánvaló, hogy NC$\subseteq$P. Nem ismeretes, hogy itt az egyenlőség áll-e, de az a valószínű, hogy nem.

A *randomizált* NC, rövidítve RNC nyelvosztályt a BPP osztály mintájára definiáljuk. Ez azokból az $\mathcal{L}$ nyelvekből áll, melyekhez van olyan program, mely a randomizált PRAM-on minden $x \in \{0,1\}^*$ bemenetre $O(|x|^{konst})$ processzorral (mondjuk, teljesen konfliktusmentesen) $O((log|x|)^{konst})$ időben 0-t vagy 1-et számít ki. Ha $x \in \mathcal{L}$, akkor a 0 eredmény valószínűsége kisebb, mint $1/4$, ha $x \notin \mathcal{L}$, akkor az 1 eredmény valószínűsége kisebb, mint $1/4$.

Az NC osztály körül kiépíthető egy hasonló bonyolultságelmélet, mint a P osztály körül. Definiálható egy nyelv NC-*visszavezetése* egy másik nyelvre, és pl. a P osztályon belül megmutatható, hogy vannak P-teljes nyelvek, vagyis olyanok, melyekre minden más P-beli nyelv NC-visszavezethető. Ennek részleteivel itt nem foglalkozunk, csak néhány fontos példára szorítkozunk.

8.2.1 Állítás: *A háromszöget tartalmazó gráfok adjacencia-mátrixai* NC*-beli nyelvet alkotnak.*

Algoritmus: Lényegében triviális. Először meghatározzuk a gráf n pontszámát. Majd az $i+jn+kn^2$ sorszámú processzort megbízzuk, hogy nézze meg, hogy az (i,j,k) ponthármas háromszöget alkot-e. A válaszokat bináris fa segítségével gyűjtjük össze.

Következő példánk kevésbé triviális, sőt talán első pillanatra meglepő: gráfok összefüggősége. A szokásos algoritmus (széltében vagy hosszában keresés) ugyanis igen erősen szekvenciális, minden lépés a korábbi lépések eredményétől erősen függ. A párhuzamosításhoz hasonló trükköt alkalmazunk, mint amelyet a 4. fejezetben Savitch tételének bizonyításánál használtunk.

8.2.2 Állítás: *Az összefüggő gráfok adjacencia-mátrixai* NC*-beli nyelvet alkotnak.*

Algoritmus: Az algoritmust az elsőbbségi modellben írjuk le. Az $i + jn + kn^2$ sorszámú processzort ismét a (i,j,k) ponthármas figyelésével bízzuk meg. Ha a ponthármasban két élt lát, akkor a harmadikat is behúzza. Ha ezt t-szer ismételjük, akkor nyilván pontosan azok a pontpárok lesznek összekötve, melyek távolsága az eredeti gráfban legfeljebb 2^t. Így $O(\log\ n)$-szer ismételve akkor és csak akkor kapunk teljes gráfot, ha az eredeti gráf összefüggő volt.

Nyilván hasonlóan dönthető el, hogy két pontot összeköt-e út, sőt a két pont távolsága is meghatározható a fenti algoritmus alkalmas módosításával.

A következő algoritmus párhuzamos számítások talán legfontosabb eszköze.

8.2.3 Csánky tétele: *Bármely egész mátrix determinánsa* NC*-kiszámítható. Következésképpen az invertálható mátrixok* NC*-beli nyelvet alkotnak.*

Algoritmus: Egy Csisztov-tól származó algoritmust ismertetünk. Előrebocsátjuk, hogy egy mátrix hatványait könnyen ki tudjuk számítani NC-értelemben. Először is két vektor skaláris szorzatát ki tudjuk számítani úgy, hogy a megfelelő elemeiket — párhuzamosan — összeszorozzuk, majd az így kapott szorzatokat párokba osztva összeadjuk, a kapott összegeket párokba osztva összeadjuk stb. Ezekután két mátrix szorzatát is ki tudjuk számítani, hiszen a szorzat minden eleme két vektor skaláris szorzata, és ezek párhuzamosan számolhatók. Ugyanezt a módszert még egyszer alkalmazva ki tudjuk számítani egy $(n\times n)$-es mátrix k-adik hatványát is minden $k \leq n$-re.

Az ötlet ezekután az, hogy a determinánst alkalmas mátrix-hatványsorként próbáljuk előállítani. Legyen B egy $(n\times n)$-es mátrix, és jelölje B_k a B bal felső sarkában álló $(k\times k)$-as részmátrixot. Tegyük föl egyelőre, hogy ezek a B_k részmátrixok nem szingulárisak, vagyis a determinánsuk nem 0. Ekkor B mátrix invertálható, és az inverzére vonatkozó ismert formula szerint

$$(B^{-1})_{nn} = \det\ B_{n-1}\ /\ \det\ B,$$

ahol $(B^{-1})_{nn}$ a B^{-1} mátrix jobb alsó sarkában álló elemet jelöli. Innen

$$\det\ B = \frac{\det\ B_{n-1}}{(B^{-1})_{nn}}.$$

Ezt folytatva kapjuk, hogy

$$\det\ B = \frac{1}{(B^{-1})_{nn}\cdot(B^{-1}_{n-1})_{n-1,n-1}\cdot\ldots\cdot(B^{-1}_1)_{11}}.$$

Írjuk B-t $B = I - A$ alakba, ahol $I = I_n$ az $(n\times n)$-es egységmátrix. Feltéve egy pillanatra, hogy A elemei elég kicsik, érvényes az alábbi sorfejtés:

$$B_k^{-1} = I_k + A_k + A_k^2 + \ldots$$

és így

$$(B_k^{-1})_{kk} = 1 + (A_k)_{kk} + (A_k^2)_{kk} + \ldots$$

Innen

$$\begin{aligned}((B_k^{-1})_{kk})^{-1} &= (1 + (A_k)_{kk} + (A_k^2)_{kk} + \ldots)^{-1}\\ &= 1 - [(A_k)_{kk} + (A_k^2)_{kk} + \ldots] + [(A_k)_{kk} + (A_k^2)_{kk} + \ldots]^2 + \ldots,\end{aligned}$$

és így

$$\det\ B = \prod_{k=1}^{n}\Big(1 - [(A_k)_{kk} + (A_k^2)_{kk} + \ldots] + [(A_k)_{kk} + (A_k^2)_{kk} + \ldots]^2 + \ldots\Big).$$

Persze, ezt a végtelen sorokból alkotott végtelen sort nem tudjuk így kiszámítani. Állítjuk azonban, hogy elég minden sorból az első n tagot kiszámítani. Pontosabban, helyettesítsünk A helyébe tA-t, ahol t valós változó. Elég kis t-re az $I_k + tA_k$ mátrixok biztosan

nem-szinguláris, így a fenti sorfejtések érvényesek. Ennél azonban többet is nyerünk. A helyettesítés után a formula így néz ki:

$$\begin{aligned}\det(I - tA) &= \prod_{k=1}^{n}\Big(1 - [t(A_k)_{kk} + t^2(A_k^2)_{kk} + \ldots] \\ &\quad + [t(A_k)_{kk} + t^2(A_k^2)_{kk} + \ldots]^2 - \ldots\Big).\end{aligned}$$

Most jön a döntő ötlet: a baloldalon t-nek egy legfeljebb n-edfokú polinomja áll, ezért a jobboldali hatványsornak is elég a legfeljebb t-edfokú tagjait kiszámítani! Így $\det(I - tA)$ a következő polinom legfeljebb n-edfokútagjaiból áll:

$$F(t) = \prod_{k=1}^{n}\Big[1 - \sum_{j=0}^{n}\Big(-\sum_{m=1}^{n} t^m (A_k^m)_{kk}\Big)^j\Big].$$

Mármost az $F(t)$ polinomot definiáló formula, bármennyire is bonyolult, könnyen kiszámítható NC-értelemben. Elhagyva belőle az n-nél magasabb fokú tagokat, megkapjuk $\det(I - tA)$-t. Ebben t=1-et helyettesítve, megkapjuk (det B)-t. □

Az 5.1.4 Tételt alkalmazva véletlen helyettesítéssel, a következő fontos alkalmazáshoz jutunk:

8.2.4 Következmény: *A teljes párosítást tartalmazó gráfok adjacencia-mátrixai* RNC-*beli nyelvet alkotnak.*

Ennek a ténynek nem ismeretes kombinatorikai (Csánky tételét nem használó) bizonyítása. Megjegyzendő, hogy ez a bizonyítás csak azt állapítja meg, hogy a gráfban van-e teljes párosítás, de nem adja meg a teljes párosítást, ha az létezik. Ez a jelentősen nehezebb probléma is megoldható azonban RNC-értelemben (Karp, Upfal és Wigderson).

9. fejezet

Döntési fák

Igen sok algoritmus logikai váza egy fával írható le: a gyökérből indulunk, és minden elágazási pontban az határozza meg, hogy melyik élen megyünk tovább, hogy egy bizonyos „teszt” milyen eredményt ad. Például a legtöbb sorbarendezési algoritmus időnként összehasonlításokat tesz bizonyos elempárok között, és aszerint folytatja a munkát, hogy az összehasonlításnak mi az eredménye. Föltesszük, hogy az elvégzett tesztek minden szükséges információt tartalmaznak a bemenetről, vagyis ha egy végponthoz érkeztünk, akkor a kimenetet már csak le kell olvasnunk végpontról. Az algoritmus bonyolultságáról képet ad a fa bonyolultsága; pl. a fa mélysége (a gyökérből kiinduló leghosszabb út élszáma) azt adja meg, hogy a legrosszabb esetben hány tesztet kell elvégeznünk a futás során. Természetesen minden algoritmust leírhatunk triviális, 1 mélységű fával (a gyökérben elvégzett teszt a végeredmény kiszámítása); ezért ennek az algoritmus-sémának csak akkor van értelme, ha megszorítjuk, hogy az egyes csúcsokban milyen teszt végezhető el.

Látni fogjuk, hogy a döntési fák nemcsak egyes algoritmusok szerkezetét teszik szemléletessé, hanem arra is alkalmasak, hogy alsó korlátot bizonyítsunk be a mélységükre. Egy ilyen alsó korlát úgy interpretálható, hogy a feladat nem oldható meg (a legrosszabb bemenetre) ennél kevesebb lépésben, ha feltesszük, hogy a bemenetről csak a megengedett teszteken keresztül nyerhető információ (pl. sorbarendezésnél az adott számokat csak egymással hasonlíthatjuk össze, nem végezhetünk velük aritmetikai műveleteket).

9.1. Döntési fákat használó algoritmusok

Tekintsünk néhány egyszerű példát.

a) Hamis pénz megtalálása egykarú mérleggel. Adott n külsőre egyforma pénzdarabunk. Tudjuk, hogy mindegyiknek 1g súlyúnak kellene lenni; de azt is tudjuk, hogy egy hamis van közöttük, mely könnyebb a többinél. Van egy egykarú mérlegünk; ezen megmérhetjük a pénzdarabok tetszőleges halmazának a súlyát. Hány méréssel tudjuk eldönteni, hogy melyik pénzdarab a hamis?

A megoldás egyszerű: egy méréssel a pénzdarabok tetszőleges halmazáról el tudjuk dönteni, hogy köztük van-e a hamis. Ha $\lceil n/2 \rceil$ pénzdarabot teszünk föl a mérlegre, akkor egy mérés után már csak legfeljebb $\lceil n/2 \rceil$ pénzdarab közül kell kiválasztani a hamisat. Ez a rekurzió $\lceil \log_2 n \rceil$ lépésben ér véget.

Az algoritmust egy gyökeres bináris fával jellemezhetjük. Minden v csúcsnak megfelel a pénzdaraboknak egy X_v részhalmaza; ebbe a csúcsba érve már tudjuk azt, hogy a hamis pénz ebbe a részhalmazba esik. (A gyökérnek az egész halmaz, a végpontoknak 1 elemű halmazok felelnek meg.) Minden v elágazási pontra az X_v részhalmazt két részre osztjuk; ezek elemszáma $\lceil |X_v|/2 \rceil$ és $\lfloor |X_v|/2 \rfloor$. Ezek felelnek meg a v fiainak. Az elsőt lemérve megtudjuk, hogy a hamis pénz melyikben van.

b) Hamis pénz megtalálása kétkarú mérleggel. Ismét csak adott n külsőre egyforma pénzdarabunk. Tudjuk, hogy egy hamis van közöttük, mely könnyebb a többinél. Ezúttal egy kétkarú mérlegünk van, de súlyok nélkül. Ezen összehasonlíthatjuk, hogy pénzdarabok két (diszjunkt) halmaza közül melyik a könnyebb, ill., hogy egyenlőek-e. Hány méréssel tudjuk eldönteni, hogy melyik pénzdarab a hamis?

Egy mérés abból áll, hogy mindkét serpenyőbe azonos számú pénzdarabot teszünk. Ha az egyik oldal könnyebb, akkor abban a serpenyőben van a hamis pénz. Ha a két oldal egyforma súlyú, akkor a kimaradó pénzek között van a hamis. Legcélszerűbb, ha mindkét serpenyőbe $\lceil n/3 \rceil$ pénzdarabot teszünk; ekkor egy mérés után már csak legföljebb $\lfloor n/3 \rfloor$ pénzdarab közül kell kiválasztani a hamisat. Ez a rekurzió $\lceil \log_3 n \rceil$ lépésben ér véget.

Mivel egy mérésnek 3 lehetséges kimenete van, az algoritmust egy olyan gyökeres fával jellemezhetjük, melyben az elágazási pontoknak 3 fiuk van. Minden v csúcsnak megfelel a pénzdaraboknak egy X_v részhalmaza; ebbe a csúcsba érve már tudjuk azt, hogy a hamis pénz ebbe a részhalmazba esik. (Mint fent, most is a gyökérnek az egész halmaz, a végpontoknak 1 elemű halmazok felelnek meg.) Minden v elágazási pontra az X_v részhalmazt három részre osztjuk; ezek elemszáma rendre $\lceil |X_v|/3 \rceil$, $\lceil |X_v|/3 \rceil$ és $|X_v| - 2\lceil |X_v|/3 \rceil$. Ezek felelnek meg a v fiainak. Az első kettőt összehasonlítva megtudjuk, hogy a hamis pénz melyikben van.

Feladat. 1. Mutassuk meg, hogy kevesebb méréssel sem az a), sem a b) feladatban nem lehet célt érni.

c) Sorbarendezés. Adott n elem, melyek (számunkra ismeretlen módon) rendezve vannak. Arra tudunk eljárást, hogy két elem sorrendjét eldöntsük; ezt egy *összehasonlításnak* nevezzük és elemi lépésnek tekintjük. Minnél kevesebb ilyen összehasonlítás árán szeretnénk a teljes rendezést meghatározni. Erre az adatkezelési alapfeladatra igen sok algoritmus ismeretes; itt csak olyan mélységig foglalkozunk a kérdéssel, amennyi a döntési fák illusztrálásához szükséges.

Nyilvánvaló, hogy $\binom{n}{2}$ összehasonlítás elég: ezzel megtudhatjuk bármely két elemről, hogy melyik a nagyobb, és ez meghatározza a rendezést. Ezek az összehasonlítások azonban nem függetlenek: a tranzitivitás alapján bizonyos párok sorrendjére gyakran következtethetünk. Valóban, elegendő $\sum_{k=1}^{n} \lceil \log_2 n \rceil \sim n \log_2 n$ összehasonlítást tenni. Ez a legegyszerűbben így látható be: tegyük föl, hogy az első $n-1$ elem rendezését már meghatároztuk. Ekkor csak az n-edik elemet kell „beszúrni”, ami nyilvánvaló módon megtehető $\lceil \log_2 n \rceil$ összehasonlítással.

Ez az algoritmus, de bármely más olyan sorbarendező algoritmus is, mely összehasonlításokkal dolgozik, leírható egy bináris fával. A gyökér megfelel az első összehasonlításnak; ennek eredményétől függően az algoritmus elágazik a gyökér valamelyik fiába. Itt ismét egy összehasonlítást végzünk stb. Minden végpont egy teljes rendezésnek felel meg.

Megjegyzés. A fenti sorbarendező algoritmusnál csak az összehasonlításokat számoltuk.

Egy igazi programmnál figyelembe kellene venni az egyéb műveleteket is, például az adatok mozgatását stb. A fenti algoritmus ebből a szempontból nem jó, mert minden egyes beszúrásnál akár az összes korábban elhelyezett elemet meg kell mozgatni, és ez $\text{const} \cdot n^2$ további lépést jelenthet. Léteznek azonban olyan sorbarendező algoritmusok, melyek összesen is csak $O(n \log n)$ lépést igényelnek.

d) Konvex burok. Ami az adatkezeléseknél a sorbarendezés, az a geometriai algoritmusok körében n síkbeli pont konvex burkának a meghatározása. A pontok koordinátáikkal vannak adva: $p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n)$. Egyszerűség kedvéért föltesszük, hogy a pontok *általános helyzetűek*, vagyis nincsen 3 egy egyenesen. Meg akarjuk határozni azokat az $i_0, \ldots, i_{k-1}, i_k = i_0$ indexeket, melyekre $p_{i_0}, \ldots, p_{i_{k-1}}, p_{i_k}$ az adott ponthalmaz konvex burkának a csúcsai, ebben a sorrendben a konvex burok mentén (mondjuk, a legkisebb abszcisszájú csúcsból indulva).

A „beszúrás" ötlete itt is ad egy egyszerű algoritmust. Rendezzük sorba az elemeket az x_i koordináták szerint; ez $O(n \log n)$ időben megtehető. Tegyük föl, hogy $p_1, \ldots, p_n$ már ebben a sorrendben vannak indexelve. Hagyjuk el a p_n pontot, és határozzuk meg a $p_1, \ldots, p_{n-1}$ pontok konvex burkát: legyenek ezek rendre a $p_{j_0}, \ldots, p_{j_{m-1}}, p_{j_m}$ pontok, ahol $j_0 = j_m = 1$.

Mármost p_n hozzávétele abból áll, hogy a $p_{j_0} \ldots p_{j_m}$ poligon p_n-ből „látható" ívét elhagyjuk, és helyette a p_n pontot tesszük be. Határozzuk meg a $p_{j_0} \ldots p_{j_m}$ sorozat legelső és legutolsó p_n-ből látható elemét, legyen ez p_{j_a} és p_{j_b}. Ekkor a keresett konvex burok $p_{j_0} \ldots p_{j_a} p_n p_{j_b} \ldots p_{j_m}$.

Hogyan lehet meghatározni, hogy egy p_{j_s} csúcs látható-e p_n-ből? A p_{n-1} pont nyilván a poligon csúcsai között szerepel, és látható p_n-ből; legyen $j_t = n-1$. Ha $s < t$, akkor nyilván p_{j_s} akkor és csak akkor látható p_n-ből, ha p_n a $p_{j_s} p_{j_{s+1}}$ egyenes fölött van. Hasonlóan, ha $s > t$, akkor p_{j_s} akkor és csak akkor látható p_n-ből, ha p_n a $p_{j_s} p_{j_{s-1}}$ egyenes fölött van. Így minden p_s-ről $O(1)$ lépésben el lehet dönteni, hogy látszik-e p_n-ből.

Ezek alapján bináris kereséssel $O(\log n)$ lépésben meg tudjuk határozni a-t és b-t, és elvégezni a p_n pont „beszúrását". Ebből a rekurzióból $O(n \log n)$ lépésszámú algoritmus adódik.

Érdemes itt elkülöníteni azokat a lépéseket, melyekben a pontok koordinátáival számításokat is végzünk, a többi (kombinatorikus jellegű) lépéstől. Nem tudjuk ugyanis, hogy a pontok koordinátái mekkorák, nem kell-e többszörös pontosságú számolás stb. A leírt algoritmust elemezve láthatjuk, hogy a koordinátákat csak két formában kellett figyelembe venni: a sorbarakásnál, melynél csak összehasonlításokat kellett tenni az abszcisszák között, és annak meghatározásánál, hogy a p_n pont a p_i és p_j pontok által meghatározott egyenes fölött vagy alatt van-e. Ez utóbbit úgy is fogalmazhatjuk, hogy meg kell határoznunk a $p_i p_j p_k$ háromszög körüljárását. Ez az analitikus geometria eszközeivel többféleképpen is megtehető.

A leírt algoritmust ismét egy bináris döntési fával írhatjuk le: minden csúcsa vagy két adott pont abszcisszái összehasonlításának, vagy három adott pont által alkotott háromszög körüljárása meghatározásának felel meg. A leírt algoritmus $O(n \log n)$ mélységű fát ad. (Sok más konvex burkot kereső algoritmus is hasonló mélységű döntési fára vezet.)

Feladatok. 2. Mutassuk meg, hogy n valós szám sorbarendezésének problémája lineáris számú lépésben visszavezethető n síkbeli pont konvex burkának a meghatározására.

3*. Mutassuk meg, hogy a fenti algoritmus második fázisa, vagyis a $p_1, \ldots, p_i$ pontok

konvex burkának a meghatározása rendre $i = 2, \ldots, n$ esetére $O(N)$ lépésben is elvégezhető.

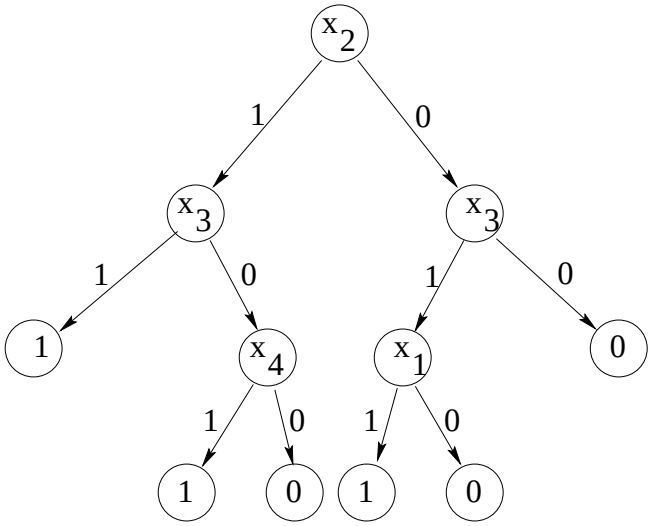
A döntési fa fogalmának formalizálása végett legyen adott A, a lehetséges bemenetek halmaza, B, a lehetséges kimenetek halmaza, és A-n értelmezett, $\{1, \ldots, k\}$ értékű függvényeknek, a *megengedett teszt-függvényeknek* egy $\mathcal{F}$ halmaza. Egy *döntési fa* olyan gyökeres fa, melynek belső pontjainak (gyökerét is beleértve) k fiuk van, végpontjai B elemeivel, többi pontja pedig $\mathcal{F}$-beli függvényekkel van címkézve. Feltesszük, hogy minden csúcsra a belőle kiinduló élek meg vannak számozva valamilyen sorrendben.

Minden döntési fa meghatároz egy $f : A \to B$ függvényt. Legyen ugyanis $a \in A$. A gyökérből kiindulva lesétálunk egy végpontba a következőképpen. Ha a v belső pontban vagyunk, akkor kiszámítjuk a v-hez rendelt tesztfüggvényt az a helyen; ha értéke i, akkor a v csúcs i-edik fiára lépünk tovább. Így eljutunk egy w végpontba; $f(a)$ értéke a w címkéje.

A kérdés az, hogy adott f függvényhez mi a legkevésbé mély döntési fa, mely f-et számítja ki.

A legegyszerűbb esetben egy $f(x_1, \ldots, x_n)$ Boole-függvényt akarunk kiszámítani, és minden teszt, amit a döntési fa csúcsaiban elvégezhetünk, valamelyik változó értékének a leolvasása. Ekkor a döntési fát *egyszerűnek* nevezzük. Minden egyszerű döntési fa bináris, az elágazási pontok a változókkal, a végpontok pedig 0-val és 1-gyel vannak címkézve. Vegyük észre, hogy a sorbarendezésre vonatkozó döntési fa nem ilyen: ott a tesztek (összehasonlítások) nem függetlenek, hiszen a rendezés tranzitív. Az f Boole-függvényt kiszámító egyszerű döntési fa minimális mélységét $D(f)$-fel jelöljük.

Példa: Tekintsük az $f(x_1, x_2, x_3.x_4) = (x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4)$ Boole-függvényt. Ezt kiszámíthatjuk pl. a 9.1 ábrán látható egyszerű döntési fával, így $D(f) \leq 3$. Könnyű meggondolni, hogy $D(f) = 3$.



9.1. ábra. Egyszerű döntési fa

Minden döntési fa úgy is tekinthető, mint egy kétszemélyes Barkochba–szerű játék. Az egyik játékos (Xavér) gondol egy $a \in A$ elemre, a másik játékos (Yvette) feladata, hogy meghatározza $f(a)$ értékét. Ehhez kérdéseket tehet föl Xavérnak. Kérdései azonban nem lehetnek tetszőlegesek, hanem csak valamely $\mathcal{F}$-beli tesztfüggvény értékét kérdezheti meg.

Hány kérdéssel tudja kiszámítani a választ? Yvette stratégiája egy döntési fának felel meg, Xavér pedig akkor játszik optimálisan, ha válaszaival a gyökértől legtávolabbi végpontba tereli Yvette-et. (Xavér csalhat, csak rajt' ne kapják — vagyis válaszaihoz kell lennie olyan $a \in A$ elemnek, amelyre mindegyik korrekt. Egyszerű döntési fa esetén ez automatikusan teljesül.)

9.2. Nem-determinisztikus döntési fák

A 4. fejezetben megismert gondolat, a *nem-determinizmus*, más bonyolultságelméleti vizsgálatokban is segít. A döntési fa modellben ez így fogalmazható meg (csak egyszerű döntési fák esetével foglalkozunk). Legyen a kiszámítandó függvény $f : \{0,1\}^n \to \{0,1\}$. A nem-determinisztikus döntési-fa-bonyolultságot két szám jellemzi (ahhoz hasonlóan, hogy a két nem-determinisztikus polinomiális osztály van: NP és co-NP). Minden x bemenetre, jelölje $D(f,x)$ azon változók minimális számát, melyek értéke az $f(x)$ értékét már meghatározza. Legyen

$$D_0(f) = \max\{D(f,x) : f(x) = 0\}, \qquad D_1(f) = \max\{D(f,x) : f(x) = 1\}.$$

Másszóval $D_0(f)$ a legkisebb olyan szám, melyre fennáll, hogy minden olyan x bemenetre, melyre $f(x) = 0$, lekérdezhető $D_0(f)$ változó úgy, hogy ezek ismeretében a függvény értéke meghatározható (az, hogy mely változókat kérdezzük le, függhet az x-től). A $D_1(f)$ szám hasonlóan jellemezhető. Nyilvánvaló, hogy

$$D(f) \geq \max\{D_0(f), D_1(f)\}.$$

Az alábbi példákból látható, hogy itt nem szükségképpen áll egyenlőség.

9.2.1 Példa: Legyen a teljes K_n gráf minden e éléhez egy-egy x_e Boole-változó hozzárendelve. Ekkor minden értékadás egy n pontú gráfnak felel meg (azokat a párokat kötjük össze éllel, melyekhez tartozó változó értéke 1). Legyen f az az $\binom{n}{2}$ vátozós Boole-függvény, melynek értéke 1, ha a bemenetnek megfelelő gráfban minden csúcs foka legalább egy, és 0, ha nem (vagyis, ha van izolált pont). Ekkor $D_0(f) \leq n-1$, hiszen ha a gráfban van izolált pont, az ebből kiinduló $n-1$ élről elég tudni, hogy nincsenek a gráfban. Azt is könnyű meggondolni, hogy $n-2$ pár összekötött, ill. nem-összekötött voltából még nem következtethetünk izolált pontra, és így

$$D_0(f) = n-1.$$

Hasonlóan, ha egy gráfban nincs izolált pont, akkor ezt már $n-1$ él megléte is bizonyítja (minden csúcsból elég egy-egy belőle kiinduló élt tudni, és az egyik él 2 csúcsot is lefed). Ha a bemenet gráf egy $n-1$ ágú csillag, akkor $(n-1)$-nél kevesebb él nem is elég. Ezért

$$D_1(f) = n-1.$$

Tehát bármelyik eset áll is fenn, $n-1$ szerencsés kérdés után már tudhatjuk a választ. Ugyanakkor, ha el akarjuk dönteni, hogy melyik eset áll fenn, akkor nem tudhatjuk előre, hogy melyik éleket kérdezzük; megmutatható, hogy a helyzet annyira rossz, amennyire csak lehet, vagyis

$$D(f) = \binom{n}{2}.$$

Ennek bizonyítására a következő alfejezetben visszatérünk (7. feladat).

9.2.2 Példa: Legyen most G tetszőleges, de rögzített n pontú gráf, és minden csúcsához rendeljünk hozzá egy-egy változót. A változók egy értékadása a csúcsok egy részhalmazának felel meg. Az f függvény értéke legyen 0, ha ez a halmaz független a gráfban, és 1 egyébként. Ez a tulajdonság Boole-formulával is egyszerűen kifejezhető:

$$f(x_1, ..., x_n) = \bigvee_{ij \in E(G)} (x_i \wedge x_j).$$

Ha ennek a Boole-függvénynek az értéke 1 (vagyis a halmaz nem független), akkor ez már 2 csúcs lekérdezéséből kiderül, de egyetlen pontéból persze nem, vagyis

$$D_1(f) = 2.$$

Másrészt ha bizonyos pontok lekérdezése után biztosak vagyunk abban, hogy a halmaz független, akkor azok a pontok, melyeket nem kérdeztünk meg, független halmazt kell, hogy alkossanak. így

$$D_0(f) \geq n - \alpha,$$

ahol α a gráf független pontjainak maximális száma. Az is bizonyítható (lásd 9.3.7 tétel), hogy ha n prím, és a gráf pontjainak ciklikus cseréje a gráfot önmagára képezi le, és a gráfnak van éle, de nem teljes, akkor

$$D(f) = n.$$

Látjuk tehát, hogy $D(f)$ lényegesen nagyobb lehet, mint $D_0(f)$ és $D_1(f)$ maximuma, sőt lehet, hogy $D_1(f) = 2$ és $D(f) = n$. Fennáll azonban a következő szép összefüggés:

9.2.3 Tétel: *Minden nem konstans f Boole-függvényre*

$$D(f) \leq D_0(f)D_1(f).$$

Bizonyítás: Teljes indukciót alkalmazunk a változók n számára vonatkozólag. Ha $n = 1$, az egyenlőtlenség triviális.

Legyen (mondjuk) $f(0, ..., 0) = 0$; ekkor kiválasztható $k \leq D_0(f)$ változó úgy, hogy azok értékét 0-nak rögzítve, a a függvény a többi változótól függetlenül 0. Feltehetjük, hogy az első k változó ilyen tulajdonságú.

Ezekután a következő determinisztikus döntési fát tekintjük: megkérdezzük az első k változó értékét, legyenek a kapott válaszok $a_1, ..., a_k$. Ezek rögzítésével egy

$$g(x_{k+1}, ..., x_n) = f(a_1, ..., a_k, x_{k+1}, ..., x_n)$$

Boole-függvényt kapunk. Nyilvánvaló, hogy $D_0(g) \leq D_0(f)$ és $D_1(g) \leq D_1(f)$. Azt állítjuk, hogy az utóbbi egyenlőtlenség élesíthető:

$$D_1(g) \leq D_1(f) - 1.$$

Tekintsük g-nek egy $(a_{k+1}, ..., a_n)$ bemenetét, melyre $g(a_{k+1}, ..., a_n) = 1$. Ez az $a_1, ..., a_k$ bitekkel együtt az f Boole-függvény egy olyan bemenetét adja, melyre $f(a_l, ..., a_n) = 1$. A $D_1(f)$ mennyiség definíciója szerint kiválasztható f-nek $m \leq D_1(f)$ változója, mondjuk

$x_{i_1}, ..., x_{i_m}$ úgy, hogy azokat az a_i értéken rögzítve, f értéke a többi változótól függetlenül 1. Ezen m változó között elő kell, hogy forduljon az első k változó valamelyike; ellenkező esetben $f(0, ..., 0, a_{k+1}, ..., a_n)$ értéke 0 kellene, hogy legyen (az első k változó rögzítése miatt), de egyben 1 is (az $x_{i_1}, ..., x_{i_m}$ rögzítése miatt), ami ellentmondás. Tehát a g függvénynek a $a_{k+1}, ..., a_n$ helyen már csak $m-1$ változóját kell rögzíteni ahhoz, hogy az azonosan 1 függvényt kapjuk. Ebből az állítás következik. Az indukciós feltevés szerint

$$D(g) \leq D_0(g)D_1(g) \leq D_0(f)(D_1(f) - 1),$$

és így

$$D(f) \leq k + D(g) \leq D_0(f) + D(g) \leq D_0(f)D_1(f). \qquad \square$$

A 9.2.1 példában egy diszjunktív 2-normálformával tudtuk megadni a függvényt, és $D_1(f) = 2$ állt fönn. Ez az egybeesés nem véletlen:

9.2.4 Állítás: Ha f kifejezhető diszjunktív k-normálformával, akkor $D_1(f) \leq k$. Ha f kifejezhető konjunktív k-normálformával, akkor $D_0(f) \leq k$.

Bizonyítás: Elég az első állítást igazolni. Legyen $(a_1, ..., a_n)$ egy olyan bemenet, amelyre a függvény értéke 1. Ekkor van olyan elemi konjunkció a diszjunktív normálforma-alakban, melynek értéke 1. Ha azokat a változókat rögzítjük, melyek ebben a konjunkcióban fellépnek, a többi változó értékétől függetlenül a függvény értéke 1 lesz. $\square$

Monoton függvényekre az előző Állításban kifejezett kapcsolat még szorosabb:

9.2.5 Állítás: Egy monoton növekvő f Boole-fiiggvény akkor és csak akkor fejezhető ki diszjunktív [konjunktív] k-normálformával, ha $D_1(f) \leq k$ $[D_0(f) \leq k]$.

Bizonyítás: A 9.2.4 Állítás alapján elegendő azt belátni, hogy ha $D_1(f) = k$, akkor f kifejezhető diszjunktív k-normálformával. Legyen $\{x_1, ..., x_{i_m}\}$ változók egy tartalmazásra nézve minimális olyan halmaza, melyet rögzíthetünk úgy, hogy a kapott függvény azonosan 1 legyen. (Egy ilyen részhalmazt *mintagnak* nevezünk.) Vegyük észre, hogy ekkor szükségképpen minden x_{i_j} változót l-nek kell rögzítenünk: a monotonitás miatt ez a rögzítés mindenképpen az azonosan 1 függvényt adja, és ha valamelyik változót 0-nak is rögzíthetnének, akkor ezt a változót egyáltalán nem is kellene rögzíteni.

Megmutatjuk, hogy $m \leq k$. Adjuk ugyanis az $x_{i_1}, ..., x_{i_m}$ változóknak az 1, a többinek a 0 értéket. Az előzőek szerint a függvény értéke 1. A $D_1(f)$ mennyiség definíciója szerint rögzíthetünk ebből az értékadásból legfeljebb k értéket úgy, hogy a kapott függvény azonosan 1 legyen. A fenti megjegyzések miatt feltehetjük, hogy csak 1-est rögzítünk, azaz csak az $x_{i_1}, ..., x_{i_m}$ változók közül rögzítünk néhányat. De az $\{x_{i_1}, ..., x_{i_m}\}$ halmaz minimalitása miatt ekkor mindegyiket rögzítenünk kellett, és ezért $m \leq k$.

Minden S mintagra készítsük el az $E_S = \bigwedge_{x_i \in S} x_i$ elemi konjunkciót, és vegyük ezek diszjunkcióját. A fentiek szerint így egy F diszjunktív k-normálformát kapunk. Triviálisan belátható, hogy ez az f függvényt definiálja. $\square$

Feladat. 4. Adjunk példát arra, hogy a 9.2.5 Állításban a monotonitás feltétele nem hagyható el.

9.3. Alsó korlátok döntési fák mélységére

Említettük, hogy a döntési fáknak, mint számítási modelleknek az az érdemük, hogy nemtriviális alsó korlátok adhatók a mélységükre. Először azonban egy majdnem triviális alsó korlátot említünk, melyet *információelméleti becslésnek* is szokás nevezni.

9.3.1 Lemma. *Ha f értékkészlete t elemű, akkor minden f-et kiszámoló k-adfokú döntési fa mélysége legalább* $\log_k t$.

Bizonyítás: Egy k-adfokú reguláris h mélységű gyökeres fának legfeljebb k^h végpontja van. Mivel f értékkészletének minden eleme kell, hogy szerepeljen, mint egy csúcs címkéje, következik, hogy $t \geq k^h$. □

Alkalmazásként tekintsünk egy tetszőleges sorbarendezési algoritmust. Ennek bemenete egész számok egy $a_1, \ldots, a_n$ sorozata, kimenete az $1, 2, \ldots, n$ számok egy $i_1, i_2, \ldots, i_n$ permutációja, melyre igaz, hogy $a_{i_1} \leq a_{i_2} \leq \ldots \leq a_{i_n}$. A tesztfüggvények pedig két elemet hasonlítanak össze:

$$\phi_{ij}(a_1, \ldots, a_n) = \begin{cases} 1, & \text{ha } a_i < a_j, \text{ és} \\ 0, & \text{ha } a_i > a_j. \end{cases}$$

Mivel $n!$ lehetséges kimenet van, ezért bármely olyan bináris döntési fa mélysége, mely kiszámítja a teljes sorrendet, legalább $\log_2(n!) \sim n \log_2 n$. A bevezetőben említett sorbarendezési algoritmus legfeljebb $\lceil \log_2 n \rceil + \lceil \log_2(n-1) + \ldots + \lceil \log_2 1 \rceil \sim n \log n$ összehasonlítást végez. (Megadhatók olyan sorbarendezési algoritmusok is, pl. az ún. Heapsort, melyek a többi munkát – adatok mozgatása stb. – figyelembe véve is csak $O(n \log n$ lépést végeznek.)

Ez a korlát sokszor igen gyönge; pl. ha egyetlen bitet kell kiszámítani, akkor semmitmondó. Egy másik egyszerű trükk alsó korlát bizonyítására a következő észrevétel.

9.3.2 Lemma. *Tegyük föl, hogy van olyan $a \in A$ bemenet, hogy akárhogyan is választunk ki K tesztfüggvényt, mondjuk $\phi_1, \ldots, \phi_K$-t, van olyan $a' \in A$, melyre $f(a') \neq f(a)$, de $\phi_i(a') = \phi_i(a)$ minden $1 \leq i \leq K$ esetén. Ekkor bármely f-et kiszámító döntési fa mélysége nagyobb, mint K.* □

Alkalmazásként nézzük meg, hogy hány összehasonlítással lehet n elemből a legnagyobbat kiválasztani. Tudjuk (kieséses bajnokság!), hogy erre $n-1$ összehasonlítás elég. A 9.3.1 Lemma csak $\log n$-et ad alsó korlátnak; de a 9.3.2 lemmát alkalmazhatjuk a következőképpen. Legyen $a = (a_1, \ldots, a_n)$ tetszőleges permutációja az $1, 2, \ldots, n$ számoknak, és tekintsünk $K < n-1$ összehasonlítás-tesztet. Azok az $\{i, j\}$ párok, melyekre a_i és a_j összehasonlításra került, egy G gráfot alkotnak az $\{1, \ldots, n\}$ alaphalmazon. Mivel kevesebb, mint $n-1$ éle van, ez a gráf nem összefüggő. Legyen G_1 az az összefüggő komponense, mely a maximális elemet ($a_j = n$-et) tartalmazza, és jelölje p a G_1 csúcsainak számát.

Legyen $a' = (a'_1, \ldots, a'_n)$ az a permutáció, melyben a G_1 csúcsainak megfelelő pozíciókban az $1, \ldots, p$ számok, a többi helyen a $p+1, \ldots, n$ számok állnak, külön-külön azonban ugyanolyan sorrendben, mint az a permutációban. Ekkor a-ban és a'-ben máshol van a maximális elem, de az adott K teszt ugyanazt adja mindkét permutációra.

Feladat. 5. Mutassuk meg, hogy $2n+1$ elem közül a nagyság szerint középső kiválasztásához legalább $n-1$ összehasonlítás kell, és (*) $O(n)$ összehasonlítás elegendő.

A következőkben speciálisabb döntési fák mélységére mutatunk néhány becslést, melyek azonban érdekesebb módszerekkel történnek. Az elsőnek Best, Schrijver és van Emde-Boas, valamint Rivest és Vuillemin egy eredményét említjük, mely egyszerű döntési fák mélységére ad szokatlan jellegű alsó becslést.

9.3.3 Tétel: *Legyen $f : \{0,1\}^n \to \{0,1\}$ tetszőleges Boole-függvény. Jelölje N azon helyettesítések számát, melyekre a függvény értéke „1", és legyen 2^t a legnagyobb 2-hatvány, mely osztója N-nek. Ekkor bármely f-et kiszámító egyszerű döntési fa mélysége legalább $n-t$.*

Bizonyítás: Tekintsünk egy tetszőleges h mélységű egyszerű döntési fát, mely kiszámítja az f függvényt és ennek egy levelét. Itt $m \leq h$ változó van rögzítve, ezért 2^{n-m} olyan bemenet van, mely ehhez a levélhez vezet. Ezekhez ugyanaz a függvényérték tartozik, így az ehhez a levélhez vezető bemenetek közül vagy 0, vagy 2^{n-m} adja az „1" függvényértéket. Ezek száma tehát osztható 2^{n-h}-val. Mivel ez minden levélre igaz, az „1" értéket adó bemenetek száma osztható 2^{n-h}-val, és ezért $t \geq n-h$. □

A fenti bizonyítás megfelelő kiterjesztésével igazolható a 9.3.3 tétel következő általánosítása; ennek részleteit feladatként az olvasóra hagyjuk.

9.3.4 Tétel. *Adott f n-változós Boole-függvényhez készítsük el a következő polinomot: $\Psi_f(t) = \sum f(x_1, \ldots, x_n) t^{x_1 + \ldots + x_n}$, ahol az összegezés minden $(x_1, \ldots, x_n) \in \{0,1\}^n$ értékrendszerre kiterjed. Ekkor, ha f kiszámítható h mélységű egyszerű döntési fával, akkor a $\Psi_f(t)$ polinom osztható $(t+1)^{n-h}$-val.* □

Egy n változós f Boole-függvényt *zárkózottnak* nevezünk, ha nem számítható ki n-nél kisebb mélységű egyszerű döntési fával. A 9.3.3 Tételből következik, hogy *ha egy Boole-függvényhez páratlan számú olyan helyettesítés van, melyre az értéke „1", akkor a függvény zárkózott.* Ennek a ténynek egy alkalmazását a 7. feladatban láthatjuk. A 9.3.4 tétel ennél többet is mond, ha $t = -1$, $h = n-1$ helyettesítéssel alkalmazzuk: *ha egy Bolle-függvény nem zárkózott, akkor azon bemeneteknek, melyekre a függvény értéke „1", pontosan a fele tartalmaz páratlan számú 1-est.*

Zárkózott függvények egy másik fontos osztályát szimmetria-feltételek adják. Egy n-változós Boole-függvényt *szimmetrikusnak* nevezünk, ha változóinak bármely permutációja a függvény értékét változatlanul hagyja. Szimmetrikus pl. az $x_1 \oplus x_2 \oplus \ldots \oplus x_n$, $x_1 \vee x_2 \vee \ldots \vee x_n$ vagy az $x_1 \wedge x_2 \wedge \ldots \wedge x_n$ függvény. Egy Boole-függvény akkor és csak akkor szimmetrikus, ha értéke csak attól függ, hogy hány változója 0 ill. 1.

9.3.5 Állítás. *Minden nem-konstans szimmetrikus Boole-függvény zárkózott.*

Bizonyítás: Legyen $f : \{0,1\}^n \to \{0,1\}$ a szóbanforgó Boole-függvény. Mivel f nem konstans, van olyan $1 \leq j \leq n$, hogy ha $j-1$ változó értéke 1, akkor a függvény értéke 0, de ha j változó értéke 1, akkor a függvény értéke 1 (vagy megfordítva).

Ezek alapján Xavérnak az alábbi stratégiát ajánlhatjuk: az első $j-1$ (különböző x_i változó értékét kérdező) kérdésre válaszoljon 1-essel, a többire 0-val. Így $n-1$ kérdés után Yvette nem tudhatja, hogy $j-1$ vagy j az egyesek száma, vagyis nem tudhatja a függvény értékét. □

A szimmetrikus Boole-függvények igen speciálisak; jelentősen általánosabb a következő osztály. Egy n-változós Boole-függvényt *gyengén szimmetrikusnak* nevezünk, ha bármely két x_i és x_j változójához van a változóknak olyan permutációja, mely x_i-t x_j-be viszi, de nem változtatja meg a függvény értékét. Gyengén szimmetrikus, de nem szimmetrikus például az

$$(x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee \ldots \vee (x_{n-1} \wedge x_n) \vee (x_n \wedge x_1)$$

függvény.

Az alábbi kérdés (az ún. általánosított Aanderaa–Rosenberg–Karp sejtés) nyitott:

9.3.6 Sejtés. *Ha egy nem-konstans monoton Boole-függvény gyengén szimmetrikus, akkor zárkózott.*

A 9.3.4 Tétel alkalmazásaként megmutatjuk, hogy a sejtés igaz egy fontos speciális esetben.

9.3.7 Tétel. *Ha egy nem-konstans monoton Boole-függvény gyengén szimmetrikus, és változóinak száma prímszám, akkor zárkózott.*

Bizonyítás: Elegendő ehhez azt megmutatni, hogy $\Psi_f(n-1)$ nem osztható n-nel. Felhasználjuk ehhez először is azt a csoportelméleti eredményt, hogy (a változók alkalmas indexezése esetén) az $x_1 \to x_2 \to \ldots \to x_n \to x_1$ ciklikus csere nem változtatja meg a függvény értékét. Ezért $\Psi_f(n-1)$ definíciójában, ha egy tagban $x_1, \ldots, x_n$ nem mind azonos, akkor belőle ciklikus cserével n további tag állítható elő, melyek mind azonosak. Így az ilyen tagok adaléka osztható n-nel. Mivel a függvény nem-konstans és monoton, következik, hogy $f(0,\ldots,0) = 0$ és $f(1,\ldots,1) = 1$, amiből látható, hogy $\Psi_f(n-1)$ n-nel osztva $(-1)^n$-et ad maradékul. □

Gyengén szimmetrikus Boole-függvényekre fontos példa bármely *gráftulajdonság*. Tekintsük egyszerű (vagyis többszörös- és hurokélek nélküli) gráfoknak egy tetszőleges tulajdonságát, pl. a síkbarajzolhatóságot; erről csak annyit teszünk föl, hogy ha egy gráfnak megvan ez a tulajdonsága, akkor minden vele izomorf gráfnak is megvan. Egy n pontú gráfot úgy adhatunk meg, hogy rögzítjük a csúcsait (legyenek ezek $1,\ldots,n$), és minden $\{i,j\} \subseteq \{1,\ldots,n\}$ párhoz bevezetünk egy x_{ij} Boole-változót, melynek az értéke 1, ha i és j össze vannak kötve, és 0, ha nem. Ílymódon n pontú gráfok síkbarajzolhatósága úgy tekinthető, mint egy $\binom{n}{2}$ változós Boole-függvény. Mármost ez a Boole-függvény gyengén szimmetrikus: bármely két párhoz, mondjuk $\{i,j\}$-hez és $\{u,v\}$-hez van a csúcsoknak olyan permutációja, mely i-t u-ba és j-t v-be viszi. Ez a permutáció a pontpárok hal-

mazán is indukál egy permutációt, mely az első adott pontpárt a másodikba viszi, és a síkbarajzolhatóságot nem változtatja meg.

A gráftulajdonságot *triviálisnak* nevezzük, ha vagy minden gráfnak megvan, vagy egyiknek sincs meg. A gráftulajdonság *monoton*, ha valahányszor egy gráfnak megvan, akkor minden olyan egyszerű gráfnak is megvan, mely élek behúzásával áll elő. A legtöbb gráftulajdonság, melyet vizsgálunk (összefüggőség, Hamilton kör létezése, teljes párosítás létezése, színezhetőség, stb.) monoton, vagy a tagadása monoton.

Az Aanderaa-Rosenberg-Karp sejtés eredeti formájában gráftulajdonságokra vonatkozott:

9.3.8 Sejtés. *Minden nem-triviális monoton gráftulajdonság zárkózott, vagyis minden olyan egyszerű döntési fa, mely egy ilyen gráftulajdonságot dönt el,* $\binom{n}{2}$ *mélységű.*

Ez a sejtés számos gráftulajdonságra be van bizonyítva (lásd az alábbi feladatokat); általános tulajdonságra annyi ismert, hogy a döntési fa $\Omega(n^2)$ mélységű (Rivest és Vuillemin), és hogy a sejtés igaz, ha a pontok száma prímhatvány (Kahn, Saks és Sturtevant). Be van bizonyítva az analóg sejtés páros gráfokra (Yao).

Feladatok. 6. Bizonyítsuk be, hogy egy gráf összefüggő volta zárkózott tulajdonság.

7. (a) Bizonyítsuk be, hogy ha n páros szám, akkor n rögzített ponton azon gráfok száma, melyekben nincs izolált pont, páratlan.

(b) Ha n páros, akkor az a gráftulajdonság, hogy egy n pontú gráfban nincsen izolált pont, zárkózott.

(c*) Ez az állítás páratlan n-re is igaz.

8. *Turnamentnek* nevezünk egy teljes gráfot, melynek minden éle irányítva van. Minden turnamentet leírhatunk $\binom{n}{2}$ bittel, melyek megmondják, hogy a teljes gráf egyes élei hogyan vannak irányítva. Így minden turnament-tulajdonság egy $\binom{n}{2}$ változós Boole-függvénynek tekinthető.

Bizonyítsuk be, hogy az a turnament-tulajdonság, hogy van 0 befokú csúcs, nem zárkózott.

A bonyolultabb döntési fák közül fontosak az *algebrai döntési fák*. Ezek esetében a bemenet n darab valós szám, $x_1, \ldots, x_n$, és minden teszt-függvényt egy polinom ír le; az elágazási pontokban háromfelé mehetünk aszerint, hogy a polinom értéke negatív, 0, vagy pozitív (néha ezek közül csak kettőt különböztetünk meg, és így csak kétfelé ágazik a fa). Példa ilyen döntési fa használatára a sorbarendezés, ahol a bemenet n valós számnak tekinthető, és a tesztfüggvényeket az $x_i - x_j$ polinomok adják.

Kevésbé triviális példa n síkbeli pont konvex burkának a meghatározása. Emlékezzünk rá, hogy itt a bemenet $2n$ valós szám (a pontok koordinátái), és a tesztfüggvényeket vagy két koordináta összehasonlítása, vagy egy háromszög körüljárásának meghatározása jelenti. Az (x_1, y_1), (x_2, y_2) és (x_3, y_3) pontok akkor és csakis akkor alkotnak pozitív körüljárású háromszöget, ha

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} > 0.$$

Így ez egy másodfokú polinom előjelének meghatározásának is tekinthető. A 9.1 pontban

leírt algoritmus tehát olyan algebrai döntési fát ad, melyben a tesztfüggvényeket legfeljebb másodfokú polinomok adják meg, és melynek mélysége $O(n \log n)$.

Algebrai döntési fák mélységére általános alsó korlátot ad Ben-Or következő tétele. A tétel kimondásához előrebocsátunk egy elemi topológiai fogalmat. Legyen $U \subseteq \mathbb{R}^n$. Az U halmaznak két pontját *ekvivalensnek* nevezzük, ha összeköthetők U-ban haladó folytonos görbével. Ennek az ekvivalencia-relációnak az ekvivalencia-osztályait az U halmaz (ívszerűen) *összefüggő komponenseinek* nevezzük.

9.3.9 Tétel: *Tegyük fel, hogy az $U \subseteq \mathbb{R}^n$ halmaznak legalább N összefüggő komponense van. Ekkor minden olyan algebrai döntési fának, mely eldönti, hogy $x \in U$ igaze, és melynek minden teszt-függvénye legfeljebb d-ed fokú polinom, a mélysége legalább* $(\log N)/(\log(6d)) - n$. *Ha $d = 1$, akkor minden ilyen fa mélysége legalább* $\log_3 N$.

Bizonyítás: A bizonyítást először az $d = 1$ esetre adjuk meg. Tekintsünk egy h mélységű algebrai döntési fát. Ennek legfeljebb 3^h végpontja van. Tekintsünk egy olyan végpontot, mely az $x \in U$ konklúzióra jut. Az ide vezető úton elvégzett tesztek eredményei legyenek mondjuk

$$p_1(x) = 0, \ \ldots, \ p_j(x) = 0, \ p_{j+1}(x) > 0, \ \ldots p_h(x) > 0.$$

Ennek az egyenlet- és egyenlőtlenség-rendszernek a megoldáshalmaza legyen K. Ekkor bármely $x \in K$ bemenet ugyanehhez a végponthoz vezet, és így $K \subseteq U$. Mivel most minden egyes p_i tesztfüggvény lineáris, K konvex, és így összefüggő. Ezért K az U halmaznak egyetlen összefüggő komponensébe esik. Tehát U különböző komponenseihez tartozó bemenetek a döntési fa különböző végpontjaihoz vezetnek. Ezért $N \leq 3^h$, ami az $d = 1$ esetre vonatkozó állítást bizonyítja.

Az általános esetben a bizonyítás ott módosul, hogy K nem szükségképpen konvex, és így nem is szükségképpen összefüggő. Ehelyett felhasználhatunk egy fontos eredményt az algebrai geometriából (Milnor és Thom tételét), melyből következik, hogy K összefüggő komponenseinek száma legfeljebb $(2d)^{n+h}$. Ebből az első részhez hasonlóan adódik, hogy

$$N \leq 3^h(2d)^{n+h} \leq (6d)^{n+h},$$

amiből a tétel állítása következik. □

Alkalmazásként tekintsük az alábbi feladatot: adott n valós szám, $x_1, \ldots, x_n$; döntsük el, hogy mind különbözőek-e. Elemi lépésnek tekintjük ismét két adott szám, x_i és x_j összehasonlítását. Ennek három kimenetele lehet: $x_i < x_j$, $x_i = x_j$, $x_i > x_j$. Mi a legkisebb mélységű döntési fa, mely ezt a problémát megoldja?

Igen egyszerűen meg tudunk adni $n \log n$ mélységű fát. Alkalmazzunk ugyanis az adott elemekre tetszőleges sorbarendezési algoritmust. Ha eközben bármikor két összehasonlítandó elemre azt a választ kapjuk, hogy egyenlők, akkor megállhatunk, mert tudjuk a választ. Ha nem, akkor $n \log n$ lépés után teljesen rendezni tudjuk az elemeket, és így mind különbözőek.

Lássuk be, hogy $\Omega(n \log n)$ összehasonlítás kell is. Tekintsük a következő halmazt:

$$U = \{(x_1, \ldots, x_n) : \ x_1, \ldots, x_n \text{ mind különböznek } \}.$$

Ennek a halmaznak pontosan $n!$ összefüggő komponense van (egy komponensbe tartozik két n-es, ha ugyanúgy vannak rendezve). Így a 9.3.9 Tétel szerint minden olyan algebrai döntési fa, mely eldönti, hogy $x \in U$, és melyben a tesztfüggvények lineárisak, legalább $\log_3(n!) = \Omega(n \log n)$ mélységű.

Látható, hogy nem tudnánk ehhez képest nyerni akkor sem, ha kvadratikus, vagy bármely más korlátos fokú polinomot megengednénk teszt-polinomként. Mivel bármely sorbarendező algebrai döntési fa használható n szám különbözőségének eldöntésére, következik, hogy n elem sorbarendezéséhez korlátos fokú teszt-polinomokat használva $\Omega(n \log n)$ mélységű algebrai döntési fa kell.

Láttuk, hogy n általános helyzetű síkbeli pont konvex burkát meg lehet határozni olyan $n \log n$ mélységű algebrai döntési fával, melyben a teszt-polinomok legfeljebb másodfokúak. Mivel a sorbarakás problémája visszavezethető a konvex burok meghatározására, következik, hogy ez lényegében a legjobb.

Feladatok. 9. (a) Ha n^2 fokú polinomot is megengedünk tesztfüggvény gyanánt, akkor 1 mélységű döntési fa is adható annak eldöntésére, hogy n szám különböző-e.

(b) Ha n-edfokú polinomot engedünk meg tesztfüggvény gyanánt, akkor n mélységű döntési fa adható annak eldöntésére, hogy n szám különböző-e.

10. Adott $2n$ különböző valós szám: $x_1, \ldots, x_n, y_1, \ldots, y_n$. El akarjuk dönteni, hogy igaz-e: nagyság szerint rendezve, bármely két y_i között van egy x_j. Bizonyítandó, hogy ehhez $\Omega(n \log n)$ összehasonlítás kell.

10. fejezet

A kommunikáció bonyolultsága

Sok algoritmikus és adatkezelési problémánál a fő nehézséget az adatok mozgatása jelenti a különböző processzorok között. Most egy olyan modellt tárgyalunk, mely — a legegyszerűbb esetben, 2 processzor részvételével — az adatmozgatás bonyolultságát igyekszik jellemezni.

Adott tehát két processzor, és tegyük föl, hogy mindegyik a bemenetnek csak egy részét ismeri. Feladatuk, hogy ebből valamit kiszámítsanak; itt csak azzal foglalkozunk, ha ez a kiszámítandó dolog egyetlen bit, tehát csak a (teljes) bemenetnek egy tulajdonságát akarják meghatározni. Elvonatkoztatunk attól az idő- és egyéb költségtől, amit a processzorok helyi számolása igényel; tehát csak a kettő közti kommunikációval foglalkozunk. Azt szeretnénk elérni, hogy minnél kevesebb bit kommunikációja árán meg tudják oldani feladatukat. Kívülről nézve azt fogjuk látni, hogy a egyik processzor egy ε_1 bitet üzen a másiknak; majd valamelyik (lehet a másik, lehet ugyanaz) egy ε_2 bitet üzen és így tovább. A végén mindkét processzornak „tudnia" kell, hogy mi a kiszámítandó bit.

Hogy szemléletesebb legyen a dolog, két processzor helyett két játékosról, Adélról és Béláról fogunk beszélni. Képzeljük el, hogy az Adél Európában, Béla Új-Zeelandban van; ekkor eléggé reális az a feltevés, hogy a helyi számítások költsége a kommunikáció költségéhez képest eltörpül.

Ami az algoritmikus bonyolultság területén az algoritmus, az a kommunikációs bonyolultság területén a *protokoll*. Ez azt jelenti, hogy mindegyik játékos számára előírjuk, hogy ha az ő bemenete x, és eddig adott $\varepsilon_1, \ldots, \varepsilon_k$ biteket üzenték (belértve azt is, hogy ki üzente), akkor ő van-e soron[1], és ha igen, akkor — ezektől függően — milyen bitet kell üzennie. Ezt a protokollt mindkét játékos ismeri, tehát tudja azt is, hogy a másik üzenete „mit jelent" (milyen bemenetek esetén üzenhette ezt a másik). Feltesszük, hogy a protokollt mindkét játékos betartja.

Könnyű megadni egy triviális protokollt: Adél küldje el a bemenet általa ismert részét Bélának. Ekkor Béla már ki tudja számítani a végeredményt, és ezt egyetlen további bittel tudatja Adéllal. Látni fogjuk, hogy ez általában igen messze lehet az optimumtól. Azt is látni fogjuk, hogy a kommunikációs bonyolultság területén hasonló fogalmak alkothatók, mint az algoritmikus bonyolultság területén, és ezek gyakran jobban kezelhetők.

[1]Az, hogy ő van-e soron, csak az $\varepsilon_1, \ldots, \varepsilon_k$ üzenetektől függhet, és nem az x-től; ezt ugyanis a másik játékosnak is tudnia kell, hogy ne legyen konfliktus afölött, hogy ki üzenheti a következő bitet.

10.1. A kommunikációs mátrix és a protokoll-fa

Legyenek Adél lehetséges bemenetei $a_1, \ldots, a_n$, Béla lehetséges bemenetei $b_1, \ldots, b_m$ (mivel a lokális számolás ingyen van, nem lényeges számunkra, hogy ezek hogyan vannak kódolva). Legyen c_{ij} a kiszámítandó érték az a_i és b_j bemenetekre. A $C = (c_{ij})_{i=1}^{n}{}_{j=1}^{m}$ mátrixot a szóbanforgó feladat *kommunikációs mátrixának* nevezzük. Ez a mátrix teljesen leírja a feladatot: mindkét játékos ismeri a teljes C mátrixot. Adél tudja C egy sorának i indexét, Béla pedig C egy oszlopának j indexét. Feladatuk, hogy a c_{ij} elemet meghatározzák. A triviális protokoll az, hogy pl. Adél megküldi Bélának az i számot; ez $\lceil \log n \rceil$ bitet jelent. (Persze ha $m < n$, akkor fordítva jobb eljárni.)

Nézzük meg, hogy mit is jelent egy protokoll erre a mátrixra nézve. Először is a protokollnak meg kell határoznia, ki kezd. Tegyük föl, hogy először Adél üzen egy ε_1 bitet. Ezt a bitet az kell, hogy meghatározza, hogy az Adél által ismert i index mi; másszóval, C sorait két osztályba lehet osztani aszerint, hogy $\varepsilon_1 = 0$ vagy 1. A C mátrix tehát két almátrixra: C_0-ra és C_1-re bomlik. Ezt a fölbontást a protokoll határozza meg, tehát mindkét játékos ismeri. Adél üzenete azt határozza meg, hogy C_0 és C_1 közül melyikben van az ő sora. Ettől kezdve tehát a probléma leszűkült a megfelelő kisebb mátrixra.

A következő üzenet kettébontja C_0-t és C_1-et. Ha az üzenő Béla, akkor az oszlopokat osztja két osztályba; ha az üzenő Adél, akkor a sorokat. Nem kell, hogy ugyanazt „jelentse", vagyis ugyanúgy ossza két részre a sorokat [oszlopokat] a második üzenet a C_0 és C_1 mátrixokon; sőt az is lehetséges, hogy mondjuk C_0-nek a sorait, C_1-nek az oszlopait osztotta ketté (Adél „0" üzenete azt jelenti, hogy „még mondok valamit", „1" üzenete azt, hogy „te jössz").

Továbbmenve látjuk, hogy a protokoll a mátrix egy egyre kisebb részmátrixokra való felbontásának felel meg. Minden „fordulóban" minden aktuális részmátrixot két kisebb részmátrixra bontunk vagy vízszintes, vagy függőleges hasítással. Egy ilyen részmátrixokra bontást *guillotine-fölbontásnak* nevezünk. (Fontos megjegyezni, hogy a mátrix sorait ill oszlopait tetszés szerint oszthatjuk két részre; tehát annak, hogy eredetileg milyen sorrendben vannak, semmi szerepe nincsen.)

Mikor áll meg ez a protokoll? Ha a játékosok már leszűkítették a lehetőségeket egy C' részmátrixra, az azt jelenti, hogy mindketten tudják, hogy a másik sora ill. oszlopa ehhez a részmátrixhoz tartozik. Ha ebből már minden esetben meg tudják mondani az eredményt, akkor ennek a részmátrixnak vagy minden eleme 0 vagy minden eleme 1.

Így a kommunikációs bonyolultság meghatározása az alábbi kombinatorikus jellegű feladatra vezet: hány fordulóban tudunk egy adott 0-1 mátrixot csupa-0 és csupa-1 mátrixokra fölhasogatni, ha minden fordulóban minden addig nyert részmátrixot vagy vízszintesen vagy függőlegesen vághatunk két részre? (Ha előbb kapunk egy csupa-0 vagy csupa-1 mátrixot, azt már nem hasogatjuk tovább. De néha hasznosabb lesz úgy tekinteni, hogy még ezt is hasogatjuk: ezt úgy fogjuk tekinteni, hogy egy csupa-1 mátrixról lehasíthatunk egy 0 sorból álló csupa-0 mátrixot.)

Még szemléletesebbé tehetjük a protokollt egy bináris fa segítségével. A fa minden pontja C egy részmátrixa. A gyökér a C mátrix, bal fia C_0, jobb fia C_1. Minden mátrix két fia úgy jön létre belőle, hogy vagy a sorait, vagy az oszlopait két osztályba osztjuk. A fa levelei mind csupa-0 vagy csupa-1 mátrixok.

A protokollt követve a játékosok ezen a fán mozognak a gyökértől valamelyik levélig. Ha egy csúcsban vannak, akkor az, hogy ennek fiai függőleges vagy vízszintes hasítással

jönnek létre, meghatározza azt, hogy ki küldi a következő bitet. Maga a bit aszerint 0 vagy 1, hogy a küldő sora [oszlopa] a csúcs jobb fiában vagy a bal fiában szerepel-e. Ha egy levélhez érnek, akkor ennek a mátrixnak minden eleme azonos, és ez a válasz a kommunikációs feladatra. A protokoll *időigénye* éppen ennek a fának a mélysége. A C mátrix *kommunikációs bonyolultsága* az összes őt megoldó protokollok lehető legkisebb időigénye. Ezt $\kappa(C)$-vel jelöljük.

Jegyezzük meg, hogy ha minden fordulóban minden mátrixot hasítunk (vagyis a fa teljes bináris fa), akkor a leveleinek pontosan a fele csupa-0, fele csupa-1. Ez abból következik, hogy a leveleket megelőző „generáció" minden mátrixát egy csupa-0-ra és egy csupa-1-re hasítottuk föl. Így ha a fa mélysége t, akkor levelei között 2^{t-1} csupa-1-es (és ugyanennyi csupa-0-ás) van. Ha azokon az ágakon, ahol előbb jutunk csupa-0 vagy csupa-1 mátrixhoz, előbb megállunk, akkor is igaz lesz annyi, hogy a csupa-1-es levelek száma legfeljebb 2^{t-1}, hiszen formálisan folytathatnánk az ágat úgy, hogy az egyik lehasított mátrix „üres".

Ebből az észrevételből egy egyszerű, de fontos alsó korlátot nyerhetünk a C mátrix kommunikációs bonyolultságára. Jelölje $rk(C)$ a C mátrix rangját; csak triviális eseteket zárunk ki, ha feltesszük, hogy C nem a csupa-0 vagy a csupa-1 mátrix. Ekkor $rk(C) > 0$.

10.1.1 Lemma:

$$\kappa(C) \geq 1 + \log rk(C).$$

Bizonyítás: Tekintsünk egy $\kappa(C)$ mélységű protokoll-fát, és legyenek $L_1, \ldots, L_N$ a levelei. Ezek tehát C részmátrixai. Jelölje M_i azt a (C-vel azonos méretű) mátrixot, melyet úgy kapunk, hogy C-ben az L_i-hez nem tartozó elemek helyébe 0-t írunk. Az előző megjegyzés alapján látjuk, hogy az M_i mátrixok között legfeljebb $2^{\kappa(C)-1}$ nem-0 van; az is könnyen látható, hogy ezek 1 rangúak.

Mármost

$$C = M_1 + M_2 + \ldots + M_N,$$

és így

$$rk(C) \leq rk(M_1) + \ldots + rk(M_N) \leq 2^{\kappa(C)-1}.$$

Ebből a lemma következik. □

10.1.2 Következmény: *Ha a C mátrix sorai lineárisan függetlenek, akkor a triviális protokoll optimális.* □

Tekintsünk egy egyszerű, de fontos kommunikációs feladatot, melyre ez az eredmény alkalmazható és mely több más szempontból is fontos példa lesz:

10.1.3 Példa: Adél is, Béla is ismer egy-egy n hosszúságú 0-1 sorozatot; azt akarják eldönteni, hogy a két sorozat egyenlő-e.

A feladathoz tartozó kommunikációs mátrix nyilván a $(2^n \times 2^n)$-szeres egységmátrix. Mivel ennek a rangja 2^n, erre a feladatra nincsen a triviálisnál ($n+1$ bitnél) jobb protokoll.

Egy másik, ugyancsak egyszerű meggondolással azt is meg tudjuk mutatni, hogy nemcsak a legrosszabb bemenetre, hanem majdnem minden bemenetre majdnem ilyen sok bitet kell kommunikálni:

10.1.4 Tétel. *Tekintsünk egy tetszőleges olyan kommunikációs protokollt, mely két n hosszúságú 0–1 sorozatról eldönti, hogy azonosak-e, és legyen $h \geq 0$. Ekkor azon $a \in \{0,1\}^n$ sorozatok száma, melyekre az (a,a) bemeneten a protokoll kevesebb, mint h bitet használ, legfeljebb 2^h.*

Bizonyítás. Minden (a,b) bemenetre jelölje $J(a,b)$ a protokoll „jegyzőkönyvét", vagyis azt a 0–1 sorozatot, melyet az egymásnak küldött bitek alkotnak. Azt állítjuk, hogy ha $a \neq b$, akkor $J(a,a) \neq J(b,b)$; ebből a tétel triviálisan következik, hiszen a h-nál kisebb hosszúságú jegyzőkönyvek száma legfeljebb 2^h.

Tegyük föl, hogy $J(a,a) = J(b,b)$, és tekintsük a $J(a,b)$ jegyzőkönyvet. Megmutatjuk, hogy ez is egyenlő $J(a,a)$-val.

Tegyük föl, hogy ez nem áll, és legyen az i-edik bit az első, ahol különböznek. Az (a,a), (b,b) és (a,b) bemeneteken nemcsak, hogy ugyanaz az első $i-1$ bit, hanem ugyanaz a kommunikáció iránya is. Adél ugyanis nem tudja megmondani az első $i-1$ lépésben, hogy Bélánál az a vagy b sorozat van-e, és mivel azt a protokoll meghatározza számára, hogy ő kell-e hogy üzenjen, ezt az (a,a) és (a,b) bemenetre ugyanúgy határozza meg. Hasonlóan, az i-edik bitet is ugyanabba az irányba küldik mindhárom bemeneten, mondjuk Adél küldi Bélának. De ebben az időpontban az (a,a) és (a,b) bemenetek Adél számára ugyanolyannak látszanak, és ezért az i-edik bit is ugyanaz lesz, ami ellentmondás. Tehát $J(a,b) = J(a,a)$.

Az (a,b) bemeneten a protokoll azzal végződik, hogy mindkét játékos tudja, hogy különböző a két sorozat. De Adél szempontjából a saját bemenete is, és a kommunkáció is ugyanaz, mint az (a,a) bemeneten, így azon a bemeneten protokoll rossz következtetésre jut. Ez az ellentmondás bizonyítja, hogy $J(a,a) \neq J(b,b)$. □

A kommunikációs komplexitás fogalmának egyik fő alkalmazása az, hogy néha algoritmusok lépésszámára alsó korlátot lehet nyerni abból, hogy bizonyos adatrészek közötti kommunikáció mennyiségét becsüljük meg. Ennek a módszernek az illusztrálására bizonyítást adunk az 1. fejezet 14.b) feladatára:

10.1.5 Tétel: *Bármely 1 szalagos Turing-gépnek $\Omega(n^2)$ lépésre van szüksége ahhoz, hogy egy $2n$ hosszúságú 0–1 sorozatról eldöntse, hogy palindróma-e.*

Bizonyítás: Tekintsünk egy tetszőleges 1 szalagos Turing-gépet, mely ezt a kérdést eldönti. Ültessük le Adélt és Bélát úgy, hogy Adél a szalag $n, n-1, \ldots, 0, -1, \ldots$ sorszámú mezőit lássa, Béla pedig a szalag $n+1, n+2, \ldots$ sorszámú mezőit; a Turing-gép szerkezetét mindkettőjüknek megmutatjuk. Induláskor tehát mindketten egy n hosszúságú 0–1 sorozatot látnak, és azt kell eldönteniük, hogy ez a két sorozat egyenlő-e (Adél sorozatát mondjuk megfordítva tekintve).

A Turing-gép működése Adélnak és Bélának egyszerű protokollt kínál: Adél gondolatban addig működteti a gépet, míg a fej az ő térfelén van, majd üzenetet küld Bélának: „Most megy át hozzád ... állapotban". Ekkor Béla működteti gondolatban addig, míg a

fej az ő térfelén van, majd megüzeni Adélnak, hogy milyen állapotban van a gép, amikor ismét visszamegy Adél térfelére stb. Így ha a gép feje k-szor megy át egyik térfélről a másikra, akkor $k \log |\Gamma|$ bitet üzennek egymásnak. A végén a Turing-gép a 0-adik mezőre írja a választ, és így Adél tudni fogja, hogy a szó palindróma-e. Ezt 1 bit árán tudathatja Bélával is.

A 10.1.4 tétel szerint ezért legföljebb $2^{n/2}$ olyan palindróma van, melyre $k \log |\Gamma| < n/2$, vagyis a legtöbb bemenetre a fej az n-edik és $(n+1)$-edik mezők között legalább cn-szer halad át, ahol $c = 1/(2 \log |\Gamma|)$.

Ez még csak $\Omega(n)$ lépés, de hasonló gondolatmenet mutatja, hogy minden $h \geq 0$-ra, legfeljebb $2^h \cdot 2^{n/2}$ bemenet kivételével a gép az $(n-h)$-adik és $(n-h+1)$-edik mezők között legalább cn-szer halad át. Ennek bizonyítása végett tekintsünk egy $2h$ hosszúságú α palindrómát, és írjunk elé egy $n-h$ jegyű β és mögé $n-h$ jegyű γ sorozatot. Az így kapott sorozat akkor és csak akkor palindróma, ha $\beta = \gamma^{-1}$, ahol γ^{-1}-gyel jelöltük a γ megfordítását. A 10.1.4 tétel és a fenti gondolatmenet szerint, minden α-hoz legfeljebb $2^{n/2}$ olyan β van, amelyre a $\beta\alpha\beta^{-1}$ bemeneten a fej az $n-h$-adik és $n-h+1$-edik mező között kevesebb, mint cn-szer halad át. Mivel az α-k száma 2^h, az állítás adódik.

Ha ezt a becslést minden $0 \leq h \leq n/2$-re összeadjuk, a kivételek száma legfeljebb

$$2^{n/2} + 2 \cdot 2^{n/2} + 4 \cdot 2^{n/2} + ... + 2^{n/2-1} \cdot 2^{n/2} < 2^n,$$

így van olyan bemenet, melyre a lépések száma legalább $(n/2) \cdot (cn) = \Omega(n^2)$. □

Feladat. 1. Mutassuk meg, hogy az alábbi kommunikációs feladatokat nem lehet a triviálisnál ($(n+1)$-nél) kevesebb bittel megoldani. Adél és Béla bemenete egy n elemű halmaz egy-egy részhalmaza, X és Y. El kell dönteniük, hogy

a) X és Y diszjunkt-e;

b) $|X \cap Y|$ páros-e.

10.2. Néhány protokoll

Eddigi példáinkban a legravaszabb kommunikációs protokoll sem tudta túlteljesíteni a triviális protokollt (azt, hogy Adél a teljes információt elküldi Bélának). Ebben az alfejezetben néhány olyan protokollt mutatunk be, mely a kommunikáció trükkös szervezésével meglepően olcsón oldja meg feladatát.

10.2.1 Példa: Adél és Béla egy (előre rögzített) n csúcsú T fa egy részfáját ismerik: Adélé a T_A részfa, Béláé a T_B részfa. Azt akarják eldönteni, hogy van-e a részfáknak közös pontjuk.

A triviális protokoll nyilván $\log M$ bitet használ, ahol M a T fa részfáinak a száma; M akár 2^{n-1}-nél is nagyobb lehet, pl. ha T egy csillag. (Különböző részfák esetén Adél üzenete különböző kell, hogy legyen. Ha a T_A és T'_A részfákra Adél ugyanazt üzeni, és mondjuk $T_A \not\subseteq T'_A$, akkor van T_A-nak olyan v csúcsa, mely nincs benne T_A-ban; ha Béla részfája az egyetlen v pontból áll, akkor ezen üzenet alapján nem tudhatja, hogy mi a válasz.)

Nézzük azonban a következő protokollt: Adél kiválaszt egy $x \in V(T_A)$ csúcsot, és elküldi Bélának (fenntartunk egy üzenetet arra az esetre, ha T_A üres; ebben az esetben készen is vannak). Ha x csúcsa a T_B fának is, készen vannak (erre az esetre Bélának van egy speciális üzenete). Ha nem, Béla megkeresi a T_B fa x-hez legközelebbi pontját (legyen ez y), és elküldi Adélnak. Ha ez T_A-ban van, akkor Adél tudja, hogy a két fának van közös pontja; ha y nincs a T_A fában, akkor a két fának egyáltalán nincs közös pontja. Ez a protokoll csak $1 + 2\lceil\log(n+1)\rceil$ bitet használ.

Feladatok. 2. Bizonyítsuk be, hogy a 10.2.1 példában bármely protokollhoz legalább $\log n$ bit kell.

3*. Finomítsuk a fenti protokollt úgy, hogy az csak $\log n + \log\log n + 1$ bitet használjon.

10.2.2 Példa: Adott egy n pontú G egyszerű gráf. Adél a G gráf egy teljes részgráfot feszítő S_A ponthalmazát, Béla egy független S_B ponthalmazát ismeri. El akarják dönteni, van-e a két ponthalmaznak közös pontja.

Ha Adél teljes információt akarna adni Bélának az általa ismert ponthalmazról, akkor $\log M$ bitre volna szükség, ahol M a teljes részgráfok száma. Ez azonban lehet akár $2^{n/2}$ is, vagyis (a legrosszabb esetben) Adél $\Omega(n)$ bitet kell, hogy használjon. Hasonló a helyzet Bélával.

A következő protokoll lényegesen gazdaságosabb. Adél megnézi, hogy van-e az S_A halmazban olyan csúcs, melynek foka legfeljebb $n/2 - 1$. Ha van, akkor egy 1-est és utána egy ilyen v csúcs nevét küldi el Bélának. Ekkor mindketten tudják, hogy Adél halmaza csak v-ből és bizonyos szomszédaiból áll, vagyis a feladatot egy legfeljebb $n/2$ csúcsú gráfra vezették vissza.

Ha S_A-ban minden csúcs foka nagyobb, mint $n/2 - 1$, akkor csak egy 0-t küld Bélának. Ekkor Béla megnézi, hogy az S_B halmazban van-e olyan csúcs, melynek foka nagyobb, mint $n/2 - 1$. Ha van, akkor egy l-est és egy ilyen w csúcs nevét elküldi Adélnak. Az előzőekhez hasonlóan ezután már mindketten tudják, hogy S_B w-n kívül csak a w-vel nem szomszédos csúcsok közül kerülhet ki, és ezzel a feladatot ismét egy legfeljebb $(n+1)/2$ csúcsú gráfra vezették vissza.

Végül ha S_B-ben minden csúcs foka legfeljebb $n/2 - 1$, Béla egy 0-t küld Adélnak. Ezután már tudják, hogy halmazaik diszjunktak.

A fenti forduló legfeljebb $O(\log n)$ bitet használ és mivel a gráf csúcsszámát felére csökkenti, legfeljebb $\log n$-szer lesz ismételve. Így a teljes protokoll csak $O(\log^2 n)$. Gondosabb számolás mutatja, hogy a felhasznált bitek száma legfeljebb $\lceil\log n\rceil(2 + \lceil\log n\rceil)/2$.

10.3. Nem-determinisztikus kommunikációs bonyolultság

Ahogyan az algoritmusoknál, a protokolloknál is fontos szerepet játszik a nem-determinisztikus változat. Ez — némileg a „tanúsítvány" fogalmával analóg módon — a következőképpen definiálható. Azt akarjuk, hogy Adél és Béla bármely olyan bemenetére, melyre a válasz 1, egy „szuperlény" kinyilatkoztathat egy rövid 0-1 sorozatot, mely mind Adélt, mind Bélát meggyőzi arról, hogy a válasz valóban 1. A „szuperlény" nyilatkozatát nem kell elhinniük, de mindketten csak annyit jelezhetnek, hogy a maguk részéről elfogadják a bizonyítékot. Egy nem-determinisztikus protokoll tehát bizonyos lehetséges $z_1, \ldots, z_n \in \{0,1\}^t$ „kinyilatkoztatásokból" áll (valamilyen előre lerögzített t hosszúsággal), melyek mindegyike Adél bi-

zonyos bemenetei és Béla bizonyos bemenetei számára elfogadhatók. Egy bemenet-párhoz akkor és csak akkor van olyan z_i, mely mindkettő számára elfogadható, ha a bemenet-párra a kommunikációs feladatra 1 a válasz. A t paraméter, azaz a z_i kinyilatkoztatások hossza a protokoll *bonyolultsága*. Végül a C mátrix *nem-determinisztikus kommunikációs bonyolultsága* a rá alkalmazható nem-determinisztikus protokollok minimális bonyolultsága; ezt $\kappa_{ND}(C)$-vel jelöljük.

10.3.1 Példa: A 10.1.3 Példában, ha azt akarja bizonyítani a szuperlény, hogy a két sorozat különböző, elég azt nyilatkoznia: „Adél i-edik bitje 0, míg Béláé nem az". Ez — a szöveges résztől eltekintve, ami része a protokollnak — csak $\lceil \log n \rceil + 1$ bit, tehát jóval kevesebb, mint az optimális determinisztikus protokoll bonyolultsága.

Megjegyezzük, hogy annak a bizonyítását, hogy a két szó azonos, még a szuperlény sem tudja elvégezni kevesebb, mint n bittel, mint azt mindjárt látni fogjuk.

10.3.2 Példa: Tegyük föl, hogy Adél és Béla egy-egy sokszöget ismer a síkon, és azt akarják eldönteni, hogy van-e a két sokszögnek közös pontja.

Ha a szuperlény arról akarja meggyőzni a játékosokat, hogy sokszögeik nem diszjunktak, akkor ezt könnyen megteheti azzal, hogy kinyilatkoztat egy közös pontot. Mindkét játékos ellenőrizheti, hogy a kinyilatkoztatott pont valóban hozzátartozik az ő sokszögéhez.

Észrevehetjük, hogy ebben a példában a tagadó választ is könnyen tudja bizonyítani a szuperlény: ha a két sokszög diszjunkt, akkor elegendő egy olyan egyenest kinyilatkoztatnia, melynek Adél sokszöge a jobbpartján, Béla sokszöge a balpartján van. (A példa bemeneteinek és a nyilatkozatoknak a pontos bitszámát most nem tárgyaljuk.)

Legyen z a szuperlény egy lehetséges kinyilatkoztatása, és legyen H_z mindazon (i, j) bemenet-párok halmaza, melyekre z „meggyőzi" a játékosokat arról, hogy $c_{ij} = 1$. Észrevehetjük, hogy ha $(i_1, j_1) \in H_z$ és $(i_2, j_2) \in H_z$, akkor (i_1, j_2) és (i_2, j_1) is H_z-be tartozik: mivel $(i_1, j_1) \in H_z$, Adél az i_1 bemenet birtokában elfogadja a z kinyilatkoztatást; mivel $(i_2, j_2) \in H_z$, Béla a j_2 bemenet birtokában elfogadja a z kinyilatkoztatást; így tehát az (i_1, j_2) bemenet birtokában mindketten elfogadják z-t, és ezért $(i_1, j_2) \in H_z$.

Ezek szerint H_z-t úgy is tekinthetjük, mint a C mátrix egy részmátrixát, mely nyilvánvalóan csupa 1-esből áll. Egy nem-determinisztikus protokoll lehetséges z kinyilatkoztatásaihoz tartozó H_z részmátrixok lefedik a C mátrix 1-eseit, mert a protokoll minden olyan bemenetre kell, hogy vonatkozzon, melyre a válasz 1 (lehetséges, hogy egy mátrixelem több ilyen részmátrixhoz is hozzátartozik). Tehát C 1-esei lefedhetők legfeljebb $2^{\kappa_{ND}(C)}$ csupa-1-es részmátrixszal.

Megfordítva, ha a C mátrix 1-esei lefedhetők 2^t csupa-1-es részmátrixszal, akkor könnyű t bonyolultságú nem-determinisztikus protokollt megadni: a szuperlény annak a részmátrixnak a sorszámát nyilatkoztatja ki, mely az adott bemenet-párt lefedi. Mindkét játékos ellenőrzi, hogy az ő bemenete a kinyilatkoztatott részmátrix sora ill. oszlopa-e. Ha igen, akkor meg lehetnek győződve, hogy a megfelelő mátrixelem 1. Ezzel tehát beláttuk a következő állítást:

10.3.3 Lemma: *$\kappa_{ND}(C)$ a legkisebb olyan t természetes szám, melyre a mátrix 1-esei lefedhetők 2^t csupa-1-es részmátrixszal.* □

A 10.1.3 Példa tagadásában (ha a két bemenet egyenlőségét kell bizonyítani), a C mátrix a $2^n \times 2^n$-szeres egységmátrix. Ennek nyilván csak 1×1-es részmátrixai csupa-1-

esek, így az 1-esek lefedéséhez 2^n ilyen részmátrix kell. Így ennek a feladatnak a nem-determinisztikus kommunikációs bonyolultsága n.

Legyen a $\kappa(C) = s$. Ekkor C felbontható 2^s részmátrixra, melyek fele csupa-0, fele csupa-1. Ezért a 10.3.3 Lemma szerint C nem-determinisztikus kommunikációs bonyolultsága legfeljebb $s - 1$. Tehát

$$\kappa_{ND}(C) \leq \kappa(C) - 1.$$

A 10.3.1 Példa mutatja, hogy a két mennyiség között igen nagy eltérés lehet.

Jelölje $\bar{C}$ azt a mátrixot, melyet C-ből úgy kapunk, hogy minden 1-est 0-ra, minden 0-t 1-esre cserélünk. Nyilvánvaló, hogy $\kappa(C) = \kappa(\bar{C})$. A 10.3.1 Példa azt is mutatja, hogy $\kappa_{ND}(C)$ és $\kappa_{ND}(\bar{C})$ nagyon különböző lehet. Az előző megjegyzések alapján

$$\max\{1 + \kappa_{ND}(C),\ 1 + \kappa_{ND}(\bar{C})\} \leq \kappa(C).$$

A következő fontos tétel (Aho, Ullman és Yannakakis) azt mutatja, hogy itt már nem lehet túl nagy az eltérés az egyenlőtlenség két oldala között:

10.3.4 Tétel:

$$\kappa(C) \leq (\kappa_{ND}(C) + 2)(\cdot\, \kappa_{ND}(\bar{C}) + 2).$$

Élesebb egyenlőtlenséget bizonyítunk be. Jelölje tetszőleges C 0–1 mátrix esetén $\varrho(C)$ a legnagyobb olyan t számot, melyre C-nek van olyan $t \times t$-es részmátrixa, melyben — a sorok és oszlopok alkalmas átrendezése után — a főátlóban 1-esek, a főátló fölött pedig 0-k állnak. Nyilvánvaló, hogy

$$\varrho(C) \leq rk(C),$$

és a 10.3.3 lemmából következik, hogy

$$\log \varrho(C) \leq \kappa_{ND}(C).$$

Ezért a következő egyenlőtlenségből a 10.3.4 tétel következik:

10.3.5 Tétel: *Ha C nem csupa-0 mátrix, akkor*

$$\kappa(C) \leq 2 + (\log \varrho(C))(2 + \kappa_{ND}(\bar{C})).$$

Bizonyítás: Teljes indukciót használunk $\varrho(C)$-re. Ha $\varrho(C) \leq 1$ akkor a protokoll triviális. Legyen $\varrho(C) > 1$ és $p = \kappa_{ND}(\bar{C})$. Ekkor a C mátrix 0-sai lefedhetők 2^p csupa-0 részmátrixszal, mondjuk $M_1, \ldots, M_{2^p}$-vel. Meg akarunk egy olyan protokollt, mely legfeljebb $(p + 2) \log \varrho(C)$ bittel eldönti a kommunikációs feladatot. A protokoll rögzíti az M_i mátrixokat, ezt tehát a játékosok is ismerik.

Minden M_i részmátrixhoz tekintsük azt az A_i mátrixot, mely C-nek M_i-t metsző sorai, és azt a B_i mátrixot, melyet C-nek M_i-t metsző oszlopai alkotnak. A protokoll alapja a következő, igen könnyen belátható állítás:

Állítás:

$$\varrho(A_i) + \varrho(B_i) \leq \varrho(C). \qquad \square$$

Ezekután a következő protokollt írhatjuk elő:

Adél megnézi, hogy van-e olyan i index, melyre M_i metszi az ő sorát, és melyre $\varrho(A_i) \leq \frac{1}{2}\varrho(C)$. Ha igen, akkor elküld egy 1-est és az i indexet Bélának, és lezárult a protokoll első fázisa. Ha nem, akkor azt üzeni, hogy „0”. Ekkor Béla megnézi, hogy van-e olyan i index, melyre M_i metszi az ő oszlopát, és melyre $\varrho(B_i) \leq \frac{1}{2}\varrho(C)$. Ha igen, akkor elküld egy 1-est és az i indexet Adélnak. Ha nem, akkor 0-t küld. Ezzel az első fázis mindenképpen befejeződött.

Ha az első fázisban akár Adél, akár Béla talál megfelelő i indexet, akkor legfeljebb $p+2$ bit kommunikációja árán a feladatot egy olyan C' ($= A_i$ vagy B_i) mátrixra szorították meg, melyre $\varrho(C') \leq \frac{1}{2}\varrho(C)$. Innen a tétel indukcióval adódik.

Ha az első fázisban mindkét játékos 0-t üzent, akkor be is fejezhetik a protokollt: a válasz „1”. Valóban, ha Adél sorának és Béla oszlopának a metszéspontjában „0” állna, akkor ez beletartozna valamelyik M_i részmátrixba. De erre a részmátrixra egyrészt

$$\varrho(A_i) > \frac{1}{2}\varrho(C)$$

(mert Adélnak nem felelt meg), másrészt

$$\varrho(B_i) > \frac{1}{2}\varrho(C)$$

(mert Bélának nem felelt meg). De ez ellentmond a fenti Állításnak. □

Érdemes megfogalmazni a fenti tételnek egy másik következményét (v.ö. a 10.1.1 lemmával):

10.3.6 Következmény:

$$\kappa(C) \leq 2 + (\log rk(C))(\kappa_{ND}(\overline{C}) + 2).$$ □

Megjegyzés: Kicsit tovább is boncolgathatjuk az algoritmusok és protokollok analógiáját. Legyen adva (az egyszerűség kedvéért négyzetes) 0-1 mátrixoknak egy $\mathcal{H}$ halmaza. Azt mondjuk, hogy $\mathcal{H} \in \mathrm{P}^{komm}$, ha bármely $C \in \mathcal{H}$ mátrix kommunikációs bonyolultsága $(\log\log n)$-nek egy polinomjánál nem nagyobb, ahol n a C mátrix sorainak száma. (Tehát ha a bonyolultság a triviális $1 + \log n$-nél lényegesen kisebb.) Azt mondjuk, hogy $\mathcal{H} \in \mathrm{NP}^{komm}$, ha bármely $C \in \mathcal{H}$ mátrix nem-determinisztikus kommunikációs bonyolultsága $(\log\log n)$-nek egy polinomjánál nem nagyobb. Azt mondjuk, hogy $\mathcal{H} \in \text{co-NP}^{komm}$, ha a $\{\overline{C} : C \in \mathcal{H}\}$ mátrixhalmaz NP^{komm}-ban van. Ekkor a 10.3.1 példa mutatja, hogy

$$\mathrm{P}^{komm} \neq \mathrm{NP}^{komm},$$

a 10.3.4 tételből pedig az következik, hogy

$$\mathrm{P}^{komm} = \mathrm{NP}^{komm} \cap \text{co-NP}^{komm}.$$

10.4. Randomizált protokollok

Ebben a pontban példát mutatunk arra, hogy a randomizáció igen jelentősen csökkentheti a protokollok bonyolultságát. Ismét azt a feladatot tekintjük, hogy a két játékos bemenetei azonosak-e. Mindkét bemenet egy-egy n hosszúságú 0-1 sorozat, mondjuk x és y. Ezeket tekinthetjük 0 és $2^n - 1$ közötti természetes számoknak is. Mint láttuk, ennek a feladatnak a kommunikációs bonyolultsága $n + 1$.

Ha a játékosok véletlen számokat is választhatnak, akkor sokkal hatékonyabban elintézhető a kérdés. Modellünkön csak annyi a változtatás, hogy mindkét játékosnak van egy-egy véletlenszám-generátora, melyek független biteket generálnak (nem jelenti az általánosság megszorítását, ha feltesszük, hogy a két játékos bitjei egymástól is függetlenek). A két játékos által kiszámolt eredmény-bit egy valószínűségi változó lesz; akkor jó a protokoll, ha ez legalább $2/3$ valószínűséggel az „igazi" értékkel egyenlő.

Protokoll: Adél választ egy véletlen p prímszámot az $1 < p < N$ intervallumból, és elosztja x-et p-vel maradékosan. Legyen a maradék r; ekkor Adél elküldi Bélának a p és r számokat. Béla ellenőrzi, hogy $y \equiv r \pmod{p}$. Ha nem, akkor azt állapítja meg, hogy $x \neq y$. Ha igen, akkor azt állapítja meg, hogy $x = y$. (A válaszbitet elküldi Adélnak.)

Először is megjegyezzük, hogy ez a protokoll csak $2 \log N + 1$ bitet használ, mivel $0 \leq r < p < N$. A probléma az, hogy tévedhet; nézzük meg, milyen irányban és milyen valószínűséggel. Ha $x = y$, akkor mindig a helyes eredményt adja. Ha $x \neq y$, akkor elképzelhető, hogy x és y ugyanazt az osztási maradékot adja p-vel osztva, és így a protokoll hibás következtetésre jut. Ez akkor következik be, ha p osztója a $d = |x - y|$ különbségnek. Legyenek d prímosztói $p_1, \ldots, p_k$, akkor

$$d \geq p_1 \cdot \ldots \cdot p_k \geq 2 \cdot 3 \cdot 5 \cdot \ldots \cdot q,$$

ahol q a k-adik prímszám. Ismert számelméleti tény, hogy

$$2 \cdot 3 \cdot 5 \cdot \ldots \cdot q > 2^{q-1}.$$

Mivel $d < 2^n$, ebből adódik, hogy $q \leq n$ és ezért $k \leq \pi(n)$ (ahol $\pi(n)$ a prímek száma n-ig). Így annak a valószínűsége, hogy éppen a d egy prímosztóját választottuk, így becsülhető:

$$\mathrm{Prob}(p|d) = \frac{k}{\pi(N)} \leq \frac{\pi(n)}{\pi(N)}.$$

Mármost a prímszámtétel szerint $\pi(n) \approx n/\log n$, és így ha N-et cn-nek választjuk, akkor a fenti korlát aszimptotikusan $1/c$, vagyis c választásával tetszőlegesen kicsivé tehető. Ugyanakkor az átküldendő bitek száma csak $2 \log N + 1 \ = 2 \log n \ +$ konstans.

Megjegyzés. Nem minden kommunikációs problémában segít ennyit a véletlen használata. Az 1. feladatban láttuk, hogy két halmaz diszjunkt voltának, ill. metszetük paritásának a meghatározása determinisztikus protokollok szempontjából ugyanúgy viselkedik, mint a 0-1 sorozatok azonos voltának eldöntése. Ezek a problémák azonban a véletlent is megengedő protokollok szempontjából már másképp viselkednek, mint az azonosság: Chor és Goldreich megmutatták, hogy a metszet paritásának a kiszámításához véletlent felhasználva is $\Omega(n)$ bit kell, Kalyanasundaram és Schnitger pedig két halmaz diszjunkt voltának eldöntésére bizonyította be ugyanezt az alsó becslést.

11. fejezet

A bonyolultság alkalmazása: kriptográfia

Egy jelenség bonyolult volta a megismerésének fő akadálya lehet. Könyvünk — reméljük — bizonyítja, hogy a bonyolultság nem csak akadálya a tudományos kutatásnak, hanem fontos és izgalmas tárgya is. Ezen azonban túlmegy: alkalmazásai is vannak, vagyis olyan példák, melyekben a jelenség bonyolultságát használjuk ki. Ebben a fejezetben egy ilyen példát tárgyalunk, a *kriptográfiát*, vagyis a titkosírások tudományát. Látni fogjuk, hogy éppen a bonyolultságelmélet eredményeinek felhasználásával, a titkosírások túllépnek a jólismert (katonai, hírszerzési) alkalmazásokon, és a polgári életben is számos alkalmazásra találnak (pl. internetes bankszámla-kezelés).

11.1. A klasszikus probléma

A Feladó egy x üzenetet akar elküldeni a Címzettnek (ahol x pl. egy n hosszúságú 0-1 sorozat). A cél az, hogy ha az üzenet egy illetéktelen harmadik kezébe kerül, az ne értse meg. Ehhez az üzenetet „kódoljuk”, ami azt jelenti, hogy a Feladó az üzenet helyett annak egy y kódját küldi el, melyből a címzett vissza tudja számolni az eredeti üzenetet, de az illetéktelen nem. Ehhez egy d kulcsot használunk, mely (mondjuk) ugyancsak egy n hosszúságú 0-1 sorozat. Ezt a kulcsot csak a Feladó és a Címzett ismerik.

A Feladó tehát kiszámít egy $y = f(x, d)$ „kódot”, mely ugyancsak egy n hosszúságú 0-1 sorozat. Feltesszük, hogy minden rögzített d-re $f(\cdot, d)$ bijektíven képezi le $\{0,1\}^n$-t önmagára. Ekkor $f^{-1}(\cdot, d)$ létezik, és így a Címzett, a d kulcs ismeretében, rekonstruálhatja az x üzenetet. A legegyszerűbb, gyakran használt f függvény az $f(x, d) = x \oplus d$ (bitenkénti összeadás modulo 2).

11.2. Egy egyszerű bonyolultságelméleti modell

Nézzünk most egy olyan feladatot, melynek — látszólag — semmi köze az előzőhöz. Egy bankautomatából lehet pl. pénzt kivenni. A számlatulajdonos begépeli a nevét vagy számlaszámát (gyakorlatban egy kártyát dug be, melyen ezek az adatok megtalálhatók) és egy jelszót. A bank számítógépe ellenőrzi, hogy valóban az ügyfél jelszava-e az. Ha ezt rendben találja, az automata kiadja a kívánt pénzmennyiséget. Ezt a jelszót elvileg csak a számla

tulajdonosa ismeri (az a kártyájára sincs ráírva), így ha ő vigyáz rá, hogy más meg ne ismerhesse, akkor ez a rendszer teljes biztonságot nyújt.

A probléma az, hogy a jelszót a banknak is ismerni kell, és így a bank alkalmazottja visszaélhet vele. Lehet-e olyan rendszert kialakítani, hogy a jelszót ellenőrző program teljes ismeretében se lehessen a jelszót kikövetkeztetni? Ez a látszólag önellentmondó követelmény kielégíthető!

Megoldás: az ügyfél fölvesz n pontot 1-től n-ig számozva, berajzol véletlenszerűen egy Hamilton kört, és hozzávesz tetszőlegesen további éleket. Ő maga megjegyzi a Hamilton-kört; ez lesz a jelszava. A gráfot (a Hamilton-kör kitüntetése nélkül!) átadja a banknak.

Ha a banknál valaki jelentkezik az ügyfél nevében, a bank a beadott jelszóról ellenőrzi, hogy a nála tárolt gráfnak Hamilton-köre-e. Ha igen, elfogadja; ha nem, elutasítja. Látható, hogy még ha a gráfot magát illetéktelenül meg is ismeri valaki, akkor is meg kell oldania azt a feladatot, hogy az adott gráfban Hamilton-kört keressen. Ez pedig NP-nehéz!

Megjegyzések: 1. Természetesen a Hamilton-kör probléma helyett bármely más NP-teljes problémára is alapozhattuk volna a rendszert.

2. Átsiklottunk egy nehéz kérdésen: hány további élt vegyen hozzá az ügyfél a gráfhoz, és hogyan? A probléma az, hogy a Hamilton-kör probléma NP-teljessége csak azt jelenti, hogy megoldása a *legrosszabb esetben* nehéz. Azt, hogy hogyan lehet *egy* olyan gráfot konstruálni, melyben van Hamilton-kör, de ezt nehéz megtalálni, nem tudjuk.

Természetes ötlet, hogy próbáljuk meg ezt a gráfot véletlen választással csinálni. Ha az összes n pontú gráfok közül választanánk egyet véletlenszerűen, akkor megmutatható, hogy ebben igen nagy valószínűséggel könnyen lehet Hamilton-kört találni. Ha az n pontú és m élű gráfok közül választunk egyet véletlenszerűen, akkor túl nagy m vagy túl kicsi m esetén hasonló a helyzet. Az $m = n \log n$ eset legalábbis nehéznek látszik.

11.3. Nyilvános kulcsú kriptográfia

Ebben a pontban egy olyan rendszert ismertetünk, mely számos ponton jelentősen megjavítja a klasszikus kriptográfia módszereit. Először is megjegyezzük, hogy a rendszer nem annyira katonai, mint polgári célokat kíván megoldani. Az elektronikus posta használatához a hagyományos levelezés olyan eszközeit kell elektronikusan megoldani, mint a boríték, aláírás, cégjelzés stb.

A rendszernek $N \geq 2$ szereplője van. Minden i szereplőnek van egy nyilvános e_i kulcsa (ezt pl. egy „telefonkönyvben” közzéteszi) és egy titkos d_i kulcsa, melyet csak ő ismer. Van továbbá egy mindenki által ismert kódoló/dekódoló függvény, mely minden x üzenetből és (titkos vagy nyilvános) e kulcsból kiszámít egy $f(x, e)$ üzenetet. (Az x üzenet és kódja valamely egyszerűen megadható H halmazból legyen; ez lehet pl. $\{0, 1\}^n$, de lehet akár a modulo m maradékosztályok halmaza is. Feltesszük, hogy az x üzenet maga tartalmazza a feladó és a címzett nevét, valamint a küldési időpontot „emberi nyelven” is.) Teljesül, hogy minden $x \in H$-ra és minden $1 \leq i \leq N$-re

$$f(f(x, e_i), d_i) = f(f(x, d_i), e_i) = x.$$

Ha az i szereplő egy x üzenetet akar elküldeni j-nek, akkor ehelyett az $y = f(f(x, d_i), e_j)$ üzenetet küldi el. Ebből j az eredeti üzenetet az $x = f(f(y, d_j), e_i)$ formulával ki tudja számítani.

Ahhoz, hogy ez a rendszer működjön, az alábbi bonyolultsági feltételeknek kell teljesülniük:

(B1) Adott x-re és e-re $f(x,e)$ hatékonyan kiszámítható.

(B2) Az x, e_i és tetszőleges számú $d_{j_1},\dots,d_{j_h}$ $(j_r \neq i)$ ismeretében $f(x,d_i)$ nem számítható ki hatékonyan.

A továbbiakban a „hatékonyan" alatt polinomiális időt értünk, de a rendszer értelmezhető más erőforrás-korlátozás mellett is.

A (B1) feltevés biztosítja, hogy ha az i szereplő a j szereplőnek üzenetet küld, azt ki tudja számítani polinomiális időben, és a címzett is meg tudja fejteni polinomiális időben. A (B2) feltétel úgy is fogalmazható, hogy ha valaki egy x üzenetet az i szereplő nyilvános kulcsával kódolt, és utána az eredetit elvesztette, akkor a kódolt üzenetből a résztvevők semmilyen koalíciója sem tudja az eredetit (hatékonyan) rekonstruálni, ha i nincsen közöttük. Ez a feltétel nyújtja a rendszer „biztonságát". Ehhez a (B2) klasszikus követelmény mellett egy sor más biztonsági feltétel is hozzátartozik.

11.3.1 *Egy i-nek szóló üzenetet csak i tud megfejteni.*

Bizonyítás: Tegyük föl, hogy illetéktelen szereplők egy $k_1,\dots,k_h$ bandája megtudja az $f(f(x,d_j),e_i)$ üzenetet (akár azt is, hogy ki küldte kinek), és ebből ki tudják hatékonyan számítani x-et. Ekkor $k_1,\dots,k_h$ és j együtt az $y=f(x,e_i)$-ből is ki tudná számítani $x=f(y,d_i)$-t. Legyen ugyanis $z=f(x,e_j)$, ekkor ismerik az $y=f(x,e_i)=f(f(z,d_j),e_i)$ üzenetet, és így $k_1,\dots,k_h$ eljárását alkalmazva ki tudják számítani z-t. De ebből j segítségével ki tudják számítani x-et is az $x=f(z,d_j)$ képlettel, ami ellentmond (B2)-nek (ha i nem volt a szereplők között). □

Hasonló meggondolásokkal igazolhatók az alábbiak:

11.3.2 *Nem tud senki az i nevében üzenetet hamisítani, vagyis j biztos lehet benne, hogy az üzenetet csak i küldhette.*

11.3.3 *j be tudja bizonyítani egy harmadik személynek (pl. bíróságnak) hogy i az adott üzenetet küldte; eközben a rendszer titkos elemeit (a d_i és d_j kulcsokat) nem kell felfedni.*

11.3.4 *j nem tudja az üzenetet megmásítani (és azt, mint i üzenetét pl. a bíróságon elfogadtatni), vagy i nevében másnak elküldeni.*

Egyáltalában nem világos persze, hogy van-e a fenti (B1) és (B2) feltételeknek eleget tevő rendszer. Ilyen rendszert csak bizonyos számelméleti bonyolultsági feltevések mellett sikerült megadni. (Javasoltak olyan rendszert is, melyről később kiderült, hogy nem biztonságos — a felhasznált bonyolultsági feltételek nem voltak igazak.) A következő pontban egy gyakran használt, és a mai tudásunk szerint biztonságos rendszert írunk le.

11.4. A Rivest-Shamir-Adleman kód (RSA kód)

Ennek a rendszernek (rövidített nevén *RSA kódnak*) egyszerűbb változatában a „posta" generál magának két n jegyű prímszámot, p-t és $q-$t, és kiszámítja az $m=pq$ számot. Ezt az m számot közzéteszi (de a prímfelbontása titokban marad!). Ezekután generál

minden előfizetőnek egy olyan $1 \leq e_i \leq m$ számot, mely $(p-1)$-hez és $(q-1)$-hez relatív prím. (Ezt megteheti úgy, hogy egy véletlen e_i-t generál 0 és $(p-1)(q-1)$ között, és az euklideszi algoritmussal ellenőrzi, hogy relatív prím-e $(p-1)(q-1)$-hez. Ha nem, új számmal próbálkozik. Könnyű belátni, hogy $\log n$ ismétlés alatt nagy valószínűséggel kap egy jó e_i számot.) Ezekután az euklideszi algoritmus segítségével talál olyan $1 \leq d_i \leq m-1$ számot, melyre

$$e_i d_i \equiv 1 \pmod{(p-1)(q-1)}$$

(itt $(p-1)(q-1) = \varphi(m)$, az m-nél kisebb, hozzá relatív prím számok száma). A nyilvános kulcs az e_i szám, a titkos kulcs a d_i szám. Magát az üzenetet egy $0 \leq x < m$ természetes számnak tekintjük (ha ennél hosszabb, akkor megfelelő darabokra vágjuk). A kódoló függvényt az

$$f(x,e) \equiv x^e \pmod{m}, \qquad 0 \leq f(x,e) < m$$

formulával definiáljuk. A dekódolásra ugyanez a formula szolgál, csak e helyett d-vel.

A kódolás/dekódolás inverz volta a „kis" Fermat-tételből következik. Definíció szerint $e_i d_i = 1 + \varphi(m)r = 1 + r(p-1)(q-1)$, ahol r természetes szám. Így ha $(x,p) = 1$, akkor

$$f(f(x,e_i),d_i) \equiv (x^{e_i})^{d_i} = x^{e_i d_i} = x(x^{p-1})^{r(q-1)} \equiv x \pmod{p}.$$

Másrészt ha $p|x$, akkor nyilván

$$x^{e_i d_i} \equiv 0 \equiv x \pmod{p}.$$

Így tehát

$$x^{e_i d_i} \equiv x \pmod{p}$$

minden x-re fennáll. Hasonlóan adódik, hogy

$$x^{e_i d_i} \equiv x \pmod{q},$$

és ezért

$$x^{e_i d_i} \equiv x \pmod{m}.$$

Mivel az első és utolsó szám egyaránt 0 és $m-1$ közé esik, következik, hogy egyenlőek, vagyis $f(f(x,e_i),d_i) = f(f(x,d_i),e_i) = x$.

A (B1) feltétel könnyen ellenőrizhető: x, e_i és m ismeretében x^{e_i}-nek m-mel vett osztási maradéka polinomiális időben kiszámítható, mint azt a 4. fejezetben láttuk. A (B2) feltétel legfeljebb csak az alábbi, gyengébb formában áll fenn:

(B2') Az x és e_i ismeretében $f(x,d_i)$ nem számítható ki hatékonyan.

Ezt feltételt úgy is fogalmazhatjuk, hogy összetett modulusra nézve e_i-edik gyök vonása a modulus prímfelbontásának ismerete nélkül nem végezhető el polinomiális időben. Ezt a feltevést nem tudjuk bebizonyítani (még a P≠NP hipotézissel sem), de mindenesetre a számelmélet jelenlegi állása szerint igaznak látszik.

Az RSA kód fenti egyszerű változata ellen több kifogás is felhozható. Először is, a „posta" minden üzenetet meg tud fejteni, hiszen ismeri a p, q számokat és a d_i titkos kulcsokat. De még ha fel is tételezzük, hogy a rendszer felállása után ezt az információt megsemmisítik, akkor is nyerhetnek illetéktelenek információt. A legnagyobb baj a következő: *A*

rendszer bármely résztvevője meg tud fejteni bármely más résztvevőnek küldött üzenetet. (Ez nem mond ellent a (B2') feltételnek, mert a rendszer j résztvevője x-en és e_i-n kívül ismeri a d_j kulcsot is.)

Tekintsük tehát a j résztvevőt, és tegyük fel, hogy kezébe jut a k résztvevőnek küldött $z = f(f(x, d_i), e_k)$ üzenet. Legyen $y = f(x, d_i)$. A j résztvevő a következőképpen fejti meg a nem neki szóló üzenetet. Kiszámítja $(e_j d_j - 1)$-nek egy $u \cdot v$ szorzattá bontását, ahol $(u, e_k) = 1$, míg v minden prímosztója e_k-nak is osztója. Ez úgy történhet, hogy kiszámítja az euklideszi algoritmussal e_k és $e_j d_j - 1$ legnagyobb közös osztóját, v_1-et, majd e_k és $(e_j d_j - 1)/v_1$ legnagyobb közös osztóját, v_2-t, majd e_k és $(e_j d_j - 1)/(v_1 v_2)$ legnagyobb közös osztóját, v_3-at stb. Ez az eljárás legfeljebb $t = \lceil \log(e_j d_j - 1) \rceil$ lépésben véget ér, vagyis $v_t = 1$. Ekkor $v = v_1 \cdots v_t$ és $u = (e_j d_j - 1)/v$ adja a kívánt felbontást.

Vegyük észre, hogy mivel $(\varphi(m), e_k) = 1$, ezért $(\varphi(m), v) = 1$. Mivel $\varphi(m) | e_j d_j - 1 = uv$, következik, hogy $\varphi(m) | u$. Mivel $(u, e_k) = 1$, az euklideszi algoritmus segítségével j ki tud számítani olyan s és t természetes számokat, melyekre $se_k = tu + 1$. Ekkor

$$z^s \equiv y^{se_k} = y^{1+tu} \equiv y(y^u)^t \equiv y \pmod{m},$$

és így

$$x \equiv y^{e_i} \equiv z^{e_i s}.$$

Így x-et a j résztvevő is ki tudja számítani.

Feladat: Mutassuk meg, hogy ha a rendszer minden résztvevője becsületes is, még mindig okozhat külső személy bajt a következőképpen. Tegyük föl, hogy egy illetéktelennek kezébe jut egy azonos szövegű levélnek két különböző személyhez küldött változata, mondjuk $f(f(x, d_i), e_j)$ és $f(f(x, d_i), e_k)$, ahol $(e_j, e_k) = 1$ (ez kis szerencsével fönnáll). Ekkor az x szöveget rekonstruálni tudja.

Most leírjuk az RSA kód egy jobb változatát. Minden i szereplő generál magának két n jegyű prímszámot, p_i-t és q_i-t, és kiszámítja az $m_i = p_i q_i$ számot. Ezekután generál magának egy olyan $1 \leq e_i < m_i$ számot, mely $(p_i - 1)$-hez és $(q_i - 1)$-hez relatív prím. Az euklideszi algoritmus segítségével talál olyan $1 \leq d_i \leq m_i - 1$ számot, melyre

$$e_i d_i \equiv 1 \pmod{(p_i - 1)(q_i - 1)}$$

(itt $(p_i - 1)(q_i - 1) = \varphi(m_i)$, az m_i-nél kisebb, hozzá relatív prím számok száma). A nyilvános kulcs az (e_i, m_i) párból, a titkos kulcs a (d_i, m_i) párból áll.

Magát az üzenetet egy x természetes számnak tekintjük. Ha $0 \leq x < m_i$, akkor a kódoló függvényt úgy, mint korábban, az

$$f(x, e_i, m_i) = x^{e_i} \pmod{m_i}, \qquad 0 \leq f(x, e_i, m_i) < m_i$$

formulával definiáljuk. Mivel azonban a különböző résztvevők különböző m_i modulust használnak, célszerű lesz kiterjeszteni az értelmezést közös értelmezési tartományra, amit akár a természetes számok halmazának is választhatunk. Írjuk fel x-et m_i alapú számrendszerben: $x = \sum_j x_j m_i^j$, és számítsuk ki a függvényt az

$$f(x, e_i, m_i) = \sum_j f(x_j, e_i, m_i) m_i^j$$

formulával. A dekódoló függvényt hasonlóan definiáljuk, e_i helyett d_i-t használva.

Az egyszerűbb változatra elmondottakhoz hasonlóan következik, hogy ezek a függvények egymás inverzei, hogy (B1) teljesül, és (B2)-ről is sejthető, hogy igaz. Ebben a változatban a „posta" semmilyen olyan információval nem rendelkezik, ami nem nyilvános, és az egyes d_j kulcsok természetesen semmilyen információt nem hordoznak a többi kulcsra vonatkozólag. Így itt a fent említett hibák nem lépnek föl.

12. fejezet

Hálózatok bonyolultsága

Az egyik legtöbbet vizsgált és legnehezebb feladat a számítástudományban különböző alsó korlátokat bizonyítani számítási problémák idő- és tárigényére. A legfontosabb kérdés az, hogy $P \stackrel{?}{=} NP$, de ennél egyszerűbb kérdésekre sem tudjuk még a választ. A klasszikus logikai megközelítés, mely a 2. fejezetben látott módszereket próbálja erre az esetre alkalmazni, úgy tűnik csődöt mond.

Létezik egy másik, kombinatorikai módszer, mellyel alsó korlátokat lehet bizonyítani számítási feladatok bonyolultságára. Ez a megközelítés a Boole-hálózatokkal való kiszámíthatóságra működik szépen és elsősorban a számítás lépései közti információcserét próbálja vizsgálni. Ezt a módszert egy szép (és elég bonyolult) bizonyítással mutatjuk be. (A kérdések nehézségét pedig jól mutatja, hogy ez a legegyszerűbb bizonyítás ilyen típusú kérdésre!)

Először két egyszerű függvényt fogunk vizsgálni, melyek:

A Többség függvény:

$$\text{TÖBBSÉG}(x_1, \ldots, x_n) = \begin{cases} 1, & \text{ha legalább } n/2 \text{ változó 1;} \\ 0, & \text{egyébként.} \end{cases}$$

A Paritás (vagy röviden XOR) függvény:

$$\text{PARITÁS}(x_1, \ldots, x_n) = x_1 + x_2 + \ldots + x_n \pmod 2.$$

Ezeket a függvényeket persze nem nehéz kiszámítani, de mi a helyzet, ha párhuzamos számítással akarjuk őket nagyon gyorsan kiszámolni? Ahelyett, hogy a nehezen kezelhető PRAM gépekkel dolgoznánk, inkább a párhuzamos számítás egy általánosabb modelljét, a kis mélységű Boole-hálózatokat fogjuk vizsgálni.

Itt meg kell jegyeznünk, hogy ha korlátozzuk a hálózat befokát például 2-re, akkor triviális, hogy legalább $\log n$ mélységű hálózatra lesz szükségünk; hiszen különben nem is függhetne a kimenet az összes bemenettől (lásd az erre vonatkozó feladatot). Tehát a hálózatok most nem-korlátozott be- és kifokúak lesznek (ez a CRCW PRAM gépek megfelelője).

Emlékeztetünk, hogy az 1. fejezetben láttuk, hogy bármely Boole-függvény kiszámítható 2 mélységű hálózattal. De olyan egyszerű függvények, mint a Többség esetén is exponenciálisan nagy hálózatokat kapunk így. Ha polinomiális algoritmussal számítunk ki

ilyen típusú függvényt, akkor, mint ahogy szintén az 1. fejezetben láttuk, polinomiális méretű Boole-hálózatot lehet belőlük csinálni, azonban ezek nem feltétlenül lesznek sekélyek (tipikusan lineáris, ha óvatosak vagyunk, logaritmikus mélységűek lesznek).

Lehet olyan hálózatot csinálni, ami egyszerre lesz kicsi (polinomiális méretű) és sekély (logaritmikus mélységű)? A válasz néhány egyszerűnek tűnő függvény esetén is nemleges. Furst–Saxe–Sipser, Ajtai majd Yao és Hastad cikkeiben egyre erősebb eredmények születtek, melyek azt bizonyították, hogy a Paritást kiszámoló bármely konstans mélységű hálózat exponenciális méretű és minden polinomiális méretű hálózat (szinte) logaritmikus mélységű. A bizonyítás túl bonyolult ahhoz, hogy itt ismertessük, de a tételt kimondjuk pontosan:

12.1 Tétel: Minden n bemenetű és d mélységű hálózat, mely a Paritást számítja ki legalább $2^{(1/10)n^{1/(d-1)}}$ méretű.

Nem sokkal később Razborov hasonló eredményt bizonyított a Többségről. Ő még Paritás (XOR) kapukat is megengedett a hálózatban a szokásos ÉS, VAGY és NEGÁCIÓ kapuk mellett (ahol a Paritás kapu kimenete természetesen a bemeneteinek mod 2 összege). Az itt következő korántsem könnyű bizonyítást csak lelkesebb olvasóinknak ajánljuk.

12.1. Alsó korlát a Többségre

Először is lássuk pontosan a tételt, amit be szeretnénk látni.

12.1.1 Tétel: Ha egy ÉS, VAGY, XOR, és NEGÁCIÓ kapukat tartalmazó, Többséget kiszámító, n bemenetű hálózat mélysége d, akkor a mérete legalább $2^{n^{(1/2d)}}/10\sqrt{n}$.

Nem lesz szükségünk mindegyik kapura (mint ahogy láttuk, hogy Boole-hálózatoknál sem kell ÉS és VAGY kapu egyszerre). Feltehetjük, hogy hálózatunk csak NEGÁCIÓ, XOR és VAGY kapukat használ, mert ezekkel konstans mélységben és polinom méretben szimulálható az ÉS kapu (lásd az erre vonatkozó feladatot).

Az ötlet az, hogy az igazi kapukat „approximálni" fogjuk. Ezek az approximáló kapuk nem a Többséget, hanem annak csak egy approximációját fogják persze kiszámítani. Az approximáció pontosságát úgy fogjuk mérni, hogy megszámoljuk hány bemenetre fog különbözni a mi approximáló hálózatunk kimenete a Többség kimenetétől. Arra vigyázunk, hogy az approximáló függvény, melyet így kapunk, ne legyen túl bonyolult, hanem a bemenetnek valamilyen „egyszerű" függvénye legyen. Végül megmutatjuk, hogy bármely „egyszerű" függvény a bemenetek jelentős részén el kell, hogy térjen a Többségtől, tehát nem approximálhatja jól. Mivel minden kapu csak egy kicsit ronthat az approximáción, ezért azt kapjuk, hogy sok kapunk kell, hogy legyen.

A bizonyításhoz először is bevezetjük az f_k ún. *k-küszöbfüggvényeket.* Egy k-küszöbfüggvény definíció szerint akkor 1, ha a bemenetek közül legalább k darab 1. Könnyen látható, hogy ha van egy Többséget kiszámító, $2n-1$ bemenetű, d mélységű, s méretű hálózatunk, akkor mindegyik $1 \leq k \leq n$-re van n bemenetű, d mélységű, s méretű hálózatunk, mely a f_k-t számítja ki (lásd az erre vonatkozó feladatot). Tehát ha f_k-ra sikerül adnunk egy jó alsó korlátot, az a Többségre is használható lesz. Mi $k = \lceil (n+h+1)/2 \rceil$-őt fogjuk vizsgálni egy megfelelően választott h-ra.

Minden Boole-függvényt ki lehet fejezni a két elemű test feletti polinomként. Ez a meg-

feleltetés szorosan összefügg a kiszámító hálózattal. Ha p_1 és p_2 felel meg két hálózatnak, akkor $p_1 + p_2$ fog megfelelni a hálózatok XORjának, $p_1 p_2$ az ÉSének és $(p_1 + 1)(p_2 + 1) + 1$ a VAGYának. A p hálózatának NEGÁCIÓját $1 - p$ fejezi ki.

A Boole-függvények egyszerűségét most a reprezentáló polinomok fokával fogjuk mérni, ezt fogjuk egyszerűen a függvény *fokának* hívni. Egy input foka például 1, tehát ezek valóban egyszerűek. De a fokszám gyorsan nőhet, mivel nem korlátoztuk egy kapu bemeneteinek számát. Már egyetlen VAGY kapu is tetszőlegesen nagy fokú függvényt adhat!

Ezért használjuk azt a trükköt, hogy a függvényeket alacsony fokú polinomokkal approximáljuk. A következő lemma biztosítja a közelítés pontosságát.

12.1.2 Lemma: Legyenek $g_1, \ldots, g_m$ legfeljebb h fokú Boole-függvények, legyen $r \geq 1$ és $f = \vee_{i=1}^{m} g_i$. Ekkor létezik egy f' legfeljebb rh fokú Boole-függvény, mely f-től legfeljebb csak 2^{n-r} darab bemenet esetén különbözik.

Bizonyítás: Válasszuk ki az $\{1, \ldots, m\}$ halmaz egy véletlen részhalmazát (minden elemét 1/2 eséllyel vesszük be a halmazunkba)! Ezt ismételjük meg r-szer, az így kapott véletlen halmazokat jelölje $I_1, \ldots, I_r$. Legyen

$$f_j = \sum_{i \in I_j} g_i,$$

és tekintsük az $f' = \vee_{j=1}^{r} f_j$ függvényt. Ennek a foka legfeljebb rh lehet (mert, mint láttuk, a VAGY kapuval való összekapcsolás polinomja kifejezhető az eredeti polinomokkal, az összeadás pedig nem növeli a fokot). Azt állítjuk, hogy f' pozitív valószínűséggel ki fogja elégíteni az approximációs követelményt.

Tekintsünk egy tetszőleges α inputot; azt állítjuk, hogy annak az esélye, hogy $f'(\alpha) \neq f(\alpha)$ legfeljebb 2^{-r}. Ehhez két esetet választunk szét. Ha minden i-re $g_i(\alpha) = 0$, akkor $f(\alpha) = 0$ és $f'(\alpha) = 0$, tehát megegyeznek. Ha nem, akkor rögzítsünk egy i-t, amire $g_i(\alpha) = 1$. Ekkor minden $f_j(\alpha)$ pontosan 1/2 valószínűséggel lesz 0 és 1, hiszen g_i-t vagy belevesszük, vagy nem. De $f'(\alpha) = 0$ csak akkor lehet, ha minden j-re $f_j(\alpha) = 0$, tehát ennek az esélye 2^{-r}. Vagyis azt kaptuk, hogy azon inputok számának várható értéke, ahol $f' \neq f$ legfeljebb 2^{n-r}. Tehát lesz egy olyan f', amire ez legfeljebb 2^{n-r}, ezt akartuk bizonyítani. □

Most megmutatjuk, hogy bármely alacsony fokú függvény rosszul közelíti az f_k küszöbfüggvényt.

12.1.3 Lemma: Legyen $n/2 \leq k \leq n$. Minden n változós, legfeljebb $h = 2k - n - 1$ fokú függvény legalább $\binom{n}{k}$ inputon különbözik f_k-tól.

Bizonyítás: Legyen g egy h fokú polinom és legyen $\mathcal{B}$ azon inputok karakterisztikus vektorainak halmaza, melyeken különbözik f_k-tól. Legyen $\mathcal{A}$ a pontosan k darab 1-est tartalmazó vektorok halmaza.

Egy f Boole-függvényre legyen $\hat{f}(x) = \sum_{y \leq x} f(y)$, ahol a $\leq$ azt jelenti, hogy y koordinátánként kisebb x-nél (azaz az y-hoz tartozó halmaz az x-hez tartozó halmaz részhalmaza). Világos, hogy az $x_{i_1} \ldots x_{i_r}$ monomhoz tartozó függvény kalapja éppen önmaga. Ebből következik, hogy egy f függvény foka akkor és csak akkor legfeljebb h, ha $\hat{f}$ eltűnik minden vektoron, amiben több, mint h darab 1-es van. Ezzel szemben $\hat{f}_k$-ról azt tudjuk, hogy 0

az összes legfeljebb $k-1$ darab 1-est tartalmazó vektoron és 1 az összes pontosan k darab 1-est tartalmazó vektoron.

Tekintsük azt az $M = (m_{ab})$ mátrixot, melynek soraihoz $\mathcal{A}$, oszlopaihoz pedig $\mathcal{B}$ elemei vannak rendelve és

$$m_{ab} = \begin{cases} 1, & \text{ha } a \geq b, \\ 0, & \text{egyébként.} \end{cases}$$

Azt akarjuk megmutatni, hogy a mátrix oszlopai generálják a teljes $GF(2)^{\mathcal{A}}$ teret. Ebből persze következik, hogy $|\mathcal{B}| \geq |\mathcal{A}| = \binom{n}{k}$.

Legyen $a_1, a_2 \in \mathcal{A}$ és jelölje $a_1 \wedge a_2$ a koordinátánkénti minimumukat. Ekkor, $\mathcal{B}$ definíciója szerint,

$$\sum_{\substack{b \leq a_1 \\ b \in \mathcal{B}}} m_{a_2 b} = \sum_{\substack{b \leq a_1 \wedge a_2 \\ b \in \mathcal{B}}} 1 = \sum_{u \leq a_1 \wedge a_2} (f_k(u) + g(u)) = \sum_{u \leq a_1 \wedge a_2} f_k(u) + \sum_{u \leq a_1 \wedge a_2} g(u).$$

A jobb oldal második összeadandója 0, mivel $a_1 \wedge a_2$ legalább $h+1$ darab 1-est tartalmaz h választása miatt. Az első összeadandó pedig csak akkor nem nulla, ha $a_1 = a_2$. Tehát ha összeadjuk az a_1-nél kisebb vektornak megfelelő oszlopokat, akkor épp az a_1-nek megfelelő koordinátához tartozó egységvektort kapjuk, ezek pedig nyilván generálják az egész teret. □

Innen már nem nehéz befejezni a 12.1.1 Tétel bizonyítását. Tegyük fel, hogy adott egy d mélységű, s méretű hálózat, mely az n bemenetű f_k függvényt akarja kiszámítani. Alkalmazzuk a 12.1.2 Lemmát $r = \lfloor n^{1/(2d)} \rfloor$-re, hogy közelítsük a hálózat VAGY kapuit (a másik két kapu nem növeli a fokot). Így az i. szintig kiszámított polinomok foka legfeljebb r^i lesz, azaz a végén kapott p_k közelítő polinom foka legfeljebb r^d lesz. A 12.1.2 Lemmából következik ($k = \lceil (n + r^d + 1)/2 \rceil$ választással), hogy a p_k polinomunk legalább $\binom{n}{k}$ helyen különbözik f_k-tól. Tehát a két lemmát összerakva azt kapjuk az eltérések számának megbecsléséből, hogy $s 2^{n-r} \geq \binom{n}{k}$. Ebből egyszerű számolással

$$s \geq \binom{n}{k} 2^{r-n} > \frac{2^r}{\sqrt{\pi n}},$$

ami épp a kívánt exponenciális alsó korlátot adja.

12.2. Monoton hálózatok

A hálózati bonyolultság talán egyik legkomolyabb eredménye Razborov nevéhez fűződik, 1985-ből. Ez azért különbözik a többi eredménytől, mert semmilyen megkötés nincs a hálózat mélységére; sajnos azonban szükség van egy igen komoly megszorításra, nevezetesen arra, hogy a hálózat *monoton*. Akkor hívunk egy Boole-hálózatot monotonnak, ha egyik bemenete sem negált és nem tartalmaz a hálózat egy NEGÁCIÓ kaput sem. Nyilvánvaló, hogy ilyen hálózattal csak monoton függvényeket lehet kiszámítani; azt sem nehéz látni, hogy bármely monoton függvény kiszámítható monoton hálózattal. Elég sok nevezetes függvény monoton, például az NP-beli párosítás, klikk, színezhetőség stb. feladatok mind monotonak. Például a k-klikk függvény definíciója az alábbi: $\binom{n}{2}$ bemenete van, melyeket x_{ij}-vel jelölünk ($1 \leq i < j \leq n$) és ezek egy gráf éleinek felelnek meg. A kimenet 1,

ha a gráfban van k méretű klikk. Mivel ez a feladat NP-teljes, ezért ebből következik, hogy minden NP-beli probléma polinomiális időben visszavezethető egy monoton függvény kiszámítására.

Razborov szuperpolinomiális alsó korlátot adott a klikket kiszámító monoton hálózat méretére anélkül, hogy bármilyen megszorítást tett volna a hálózat mélységére. Ezt az eredményt javította meg később Andreev, Alon és Boppana, akik exponenciális alsó korlátot bizonyítottak a k-klikkre.

De ezekből az eredményekből nem következik semmilyen alsó becslés a klikket kiszámító nem-monoton hálózatok méretére. Tardos Éva konstruált egy olyan monoton Boole-függvény családot, melyeket polinomiális időben ki lehet számítani, de egy őket kiszámító monoton Boole-hálózatnak exponenciálisan sok kaput kell tartalmaznia.

Feladatok.

1. Ha egy Boole-hálózat befoka 2 és n inputja van, melyek mindegyikétől függ a kimenet, akkor a mélysége legalább $\log n$.

2. (a) Mutassuk meg, hogy a NEGÁCIÓ kaput helyettesíthetjük XOR kapuval, ha bemenetként használhatjuk a konstans 1-et. (b) Mutassuk meg, hogy az n változós ÉS kapu helyettesíthető polinomiálisan sok NEGÁCIÓ, VAGY és XOR kapuval konstans mélységben.

3. Tegyük fel, hogy egy s méretű, d mélységű, $2n-1$ bemenetű hálózat kiszámítja a TÖBBSÉGet. Mutassuk meg, hogy ekkor minden $1 \leq k \leq n$-re létezik egy s méretű, d mélységű hálózat, mely az n bemenetű k-küszöbfüggvényt számítja ki.