2.5. A lineáris kongruencia egyenlet.

Definíció: Kongruencia

Az *a* és *b* egész számokat ***kongruens***nek mondjuk az *n* ***modulus*** szerint, ha az *n* szerinti osztás utáni maradékaik megegyeznek, vagy ami ugyanaz: ha $n|(a-b)$. Jelölésben: $a \equiv b \bmod n$.

1. Tétel: A kongruenciákon végezhető műveletek tétele

Legyen $a \equiv b \bmod n$ és $c \equiv d \bmod n$! Akkor igazak az alábbi állítások:

1. $a \pm c \equiv b \pm d \mod n$, (1)
2. $a \cdot c \equiv b \cdot d \mod n$, (2)
3. $\frac{a}{k} \equiv \frac{b}{k} \mod n$, $ha\ k|a, k|b\ és\ lnko(k,n)=1$ (3)
4. $a \equiv b \mod m$, $ha\ m|n$ (4)

A tétel bizonyítását az olvasóra bízzuk.

Definíció: A lineáris kongruencia egyenlet

Az

$$a \cdot x \equiv b \bmod n,\ a,b \in Z,\ n \in Z^+ \tag{5}$$

egyenletet, melyben $x \in Z$ az ismeretlen, ***lineáris kongruencia egyenlet***nek nevezzük.

Tétel: A lineáris kongruencia egyenlet megoldhatósági tétele

Legyen az (5) egyenletre $d^* = lnko(a,n) = a \cdot x^* + n \cdot y^*$. Az (5) lineáris kongruencia egyenletnek akkor és csak akkor van megoldása, ha $d^*|b$. Ha van megoldás, akkor végtelen sok van, de ezeket egy d^* számú megoldást tartalmazó úgynevezett *megoldás alaprendszer*ből megkaphatjuk az *n* egész számú többszöröseinek a hozzáadásával. Az alaprendszer elemeit a $0 \le x < n$ intervallumból választjuk ki. Az alaprendszer megoldásai az alábbi módon írhatók fel:

$$\boxed{\begin{aligned} x_0 &= x^* \cdot (b/d^*) \bmod n, \\ x_i &= x_0 + i \cdot (n/d^*) \bmod n,\quad i = 1,2,\ldots,d^*-1 \end{aligned}} \qquad \begin{matrix}(6)\\(7)\end{matrix}$$

Bizonyítás

Legyen $q_1 = \left\lfloor \frac{ax}{n} \right\rfloor$, $q_2 = \left\lfloor \frac{b}{n} \right\rfloor$, $q = q_2 - q_1$. Akkor a lineáris kongruencia egyenlet $ax - q_1 n = b - q_2 n$ alakra írható át, amiből az $ax + qn = b$ egyenlet adódik, vagyis hogy a b az a és az n lineáris kombinációja. Ha azt akarjuk, hogy legyen megoldás, akkor $b \in L(a,n)$ fenn kell álljon, ahol $L(a,n)$ az a és n lineáris kombinációinak a halmaza. Ha ez nem áll fenn, akkor nincs megoldás.

A lineáris kombinációban lévő elemeket viszont a $d^* = lnko(a,n)$ legnagyobb közös osztó osztja, és csak azokat osztja a lineáris kombinációk halmazának jellemzési tétele szerint. Legyen most b olyan, hogy $d^* | b$. Akkor van olyan k egész szám, hogy $b = k \cdot d^*$. A legnagyobb közös osztó viszont az a és az n lineáris kombinációja, azaz van olyan x^* és y^* egész, hogy $d^* = a \cdot x^* + n \cdot y^*$. Ez a formula viszont egyenértékű az $a \cdot x^* \equiv d^* \bmod n$ lineáris kongruencia egyenlettel, ha az n szerinti maradékokat nézzük. Beszorozva itt k-val $a \cdot x^* k \equiv d^* k \bmod n$ adódik, amiből azonnal látható, hogy az $x_0 = x^* k = x^* \left(b / d^*\right) \bmod n$ megoldás. További megoldásokat kapunk, hogyha képezzük az $x_i = x_0 + i \cdot \left(n / d^*\right) \bmod n$, $i = 1,2,\ldots,d^* - 1$ számokat, ugyanis a lineáris kongruencia egyenletbe történő behelyettesítés után az $ax_0 + a \cdot i \cdot \left(n / d^*\right)$, $i = 1,2,\ldots,d^* - 1$ jelenik meg a baloldalon, ahol a második tag osztható n-nel, mert a d^* az a-t osztja, így az n megmarad, tehát ez a tag nem módosítja az első tag általi maradékot. Ezeket a megoldásokat *alapmegoldások*nak nevezzük. Nyílvánvaló, hogy ha n egész többszörösét hozzáadom az alapmegoldásokhoz, akkor újra megoldást kapok, csak az már nem lesz alapmegoldás (nem viselkedik maradékként.)

■

A lineáris kongruencia egyenlet megoldására algoritmus konstruálható, ugyanis a kívánt x^* a kibővített euklideszi algoritmusból megkapható.

2.5.1. algoritmus **Lineáris kongruencia megoldó**	
1	Lineáris_kongruencia_megoldó (a, b, n, X)
2	// Input paraméterek: a,b,n∈Z, n>0
3	// Output paraméter : X – egyindexes tömb
4	// indexelés 0-tól
5	Kibővített_Euklidesz (a, n, d^*, x^*, y^*)
6	Hossz[X]← *0*
7	**IF** $d^* \mid b$
8	**THEN** $x_0 \leftarrow x^* \cdot \left(b / d^*\right) \bmod n$
9	Hossz[X]← d^*
10	**FOR** $i \leftarrow 1$ **TO** $d^* - 1$ **DO**
11	$x_i \leftarrow x_0 + i \cdot \left(n / d^*\right) \bmod n$
12	**RETURN** (X)
13	// Hossz[X]=0 jelenti, hogy nincs megoldás

1. Példa: $3604 \cdot x \equiv 136 \bmod 3332$

Láttuk, hogy $lnko(3604,3332) = 68 = -12 \cdot 3604 + 13 \cdot 3332$.

136 osztható 68-cal, így az egyenletnek van megoldása. Az alaprendszer 68 különböző elemet tartalmaz.

Most $b/d^* = 136/68 = 2$, $n/d^* = 3332/68 = 49$, $x^* = -12 + 68 = 56$.

A megoldások: $x_0 = 56 \cdot 2 = 112$, $x_1 = 112 + 49 = 161$, $x_2 = 112 + 2 \cdot 49 = 210$, …, $x_{67} = 112 + 67 \cdot 49 = 3395 \equiv 63 \bmod 3332$.

Definíció: A multiplikatív inverz

Legyen a lineáris kongruencia egyenlet

$$ax \equiv 1 \bmod n,\ a \in Z,\ n \in Z^+,\ lnko(a,n) = 1 \qquad (8)$$

alakú (azaz a és n legyenek relatív prímek). Az egyenlet egyetlen alapmegoldását az a szám n szerinti *multiplikatív inverzének* nevezzük. Jelölése:

$$x = a^{-1} \bmod n. \qquad (9)$$

A multiplikatív inverz meghatározása történhet a lineáris kongruencia megoldó 2.5.1. algoritmus segítségével. Természetesen a FOR ciklus alkalmazására az eljárásban nem lesz szükség.

2. Példa: $5^{-1} = ? \bmod 8$

$5x \equiv 1 \bmod 8$ megoldását keressük.

Lépésszám	n	a	q	r	d^*	x^*	y^*
0	8	5	1	3	1	2	-1-1·2 = -3
1	5	3	1	2	1	-1	1-1·(-1) = 2
2	3	2	1	1	1	1	0-1·1 = -1
3	2	1	2	0	1	0	1-2·0 = 1
4	1	0	-	1	1	1	0

Láthatóan *lnko(5,8)=1*, tehát van multiplikatív inverz. *1=2·8+(-3)·5=16-15*.

Az *a* együtthatója *–3*, aminek a *8* szerinti maradéka *–3+8=5*. Tehát az *5* multiplikatív inverze *8*-ra nézve éppen saját maga. Ellenőrzés: *5·5=25=3·8+1*.

2.6. RSA

Sok esetben – többek között a majd ismertetésre kerülő RSA algoritmusban – szükség van egészek hatványa valamely modulus szerinti maradékának meghatározására. Legyen $a, b, n \in Z^+$. A feladat $c = a^b \bmod n$ meghatározása lehetőleg elfogadható idő alatt. Ilyennek bizonyul a *moduláris hatványozás* algoritmusa. Ötlete a b szám bináris felírásából jön. Legyenek a b bitjei: $b_k, b_{k-1}, \ldots, b_1, b_0$. A legmagasabb helyiértékű bit 1-es. Ha b-nek ki akarjuk számítani az értékét, akkor ezt megtehetjük a *2* hatványaival történő számítással, $b = b_k \cdot 2^k + b_{k-1} \cdot 2^{k-1} + \ldots + b_1 \cdot 2^1 + b_0 \cdot 2^0$. Ugyanezt az eredményt megkaphatjuk a gazdaságosabb Horner séma szerint:

$$b = \left(\ldots\left(\left(b_k\right) \cdot 2 + b_{k-1}\right) \cdot 2 + \ldots + b_1\right) \cdot 2 + b_0 . \qquad (1)$$

Itt láthatóan csak kettővel való szorzást és egy nulla vagy egy hozzáadását kell végezni, melyek számítástechnikailag hatékony műveletek. Ez annál inkább hasznos, mivel még a *b* értékét sem kell kiszámítani az algoritmusban, hiszen az adott, hanem csak az egyes bitjeit kell elérni, ami eltolásokkal hatékonyan megvalósítható. A b szám a kitevőben van, ezért a hatványozás során a kettővel való szorzásnak a négyzetreemelés az egy hozzáadásának pedig az alappal történő szorzás felel meg. Minden lépés után vehető a modulo n szerinti maradék, így a használt számtartomány mérete mérsékelt marad. (Mekkora?) A megfelelő algoritmus pszeudokódja:

2.6.1. algoritmus Moduláris hatványozó	
1	Moduláris_hatványozó (a, b, n, c)
2	// Input paraméterek: a,b,n∈Z, a,b,n>0
3	// Output paraméter: c∈Z, c≥0
4	$p \leftarrow 0$
5	$c \leftarrow 1$
6	**FOR** $i \leftarrow k$ **DOWNTO** *0* **DO**
7	$p \leftarrow 2p$
8	$c \leftarrow c^2 \bmod n$
9	**IF** $b_i = 1$
10	**THEN** $p \leftarrow p + 1$
11	$c \leftarrow (c \cdot a) \bmod n$
12	**RETURN** (c)

Az algoritmusban ténylegesen a p értékét nem kell kiszámítani, mert az végül a b értékét adja majd.

1. Példa: $118^{2005} \bmod 137$ b $=2005_{10}= (111\ 1101\ 0101_2)$, a=118, n=137.

k	b_k	$c^2 \bmod n$	$(c \cdot a) \bmod n$
10	1	$1^2 = 1 \equiv 1$	$1 \cdot 118 = 118 \equiv 118$
9	1	$118^2 = 13924 \equiv 87$	$87 \cdot 118 = 10266 \equiv 128$
8	1	$128^2 = 16384 \equiv 81$	$81 \cdot 118 = 9558 \equiv 105$
7	1	$105^2 = 11025 \equiv 65$	$65 \cdot 118 = 7670 \equiv 135$
6	1	$135^2 = 18225 \equiv 4$	$4 \cdot 118 = 472 \equiv 61$
5	0	$61^2 = 3721 \equiv 22$	
4	1	$22^2 = 484 \equiv 73$	$73 \cdot 118 = 8614 \equiv 120$
3	0	$120^2 = 14400 \equiv 15$	
2	1	$15^2 = 225 \equiv 88$	$88 \cdot 118 = 10384 \equiv 109$
1	0	$109^2 = 11881 \equiv 99$	
0	1	$99^2 = 9801 \equiv 74$	$74 \cdot 118 = 8732 \equiv 101$

Az RSA algoritmus fel fogja tételezni, hogy nagy prímszámaink vannak. Ilyenek keresésére egy eszköz lehet (nem a leghatékonyabb és nem abszolút biztos) az alábbi tételen alapuló algoritmus.

1. Tétel: A Fermat tétel
Ha p prím, akkor

$$a^{p-1} \equiv 1 \bmod p, \quad a = 1,2,\ldots, p-1. \tag{2}$$

.

A tételt nem bizonyítjuk.

A tételre épülő prímszám ellenőrzési algoritmus egy egyszerű, de nem teljesen megbízható változatának a pszeudokódja:

	2.6.2. algoritmus Fermat féle álprímteszt
1	Fermat_teszt (n, p)
2	// Input paraméter: n∈Z, n>1
3	// Output paraméter: p logikai érték
4	// igaz – lehet prím
5	// hamis – nem prím
6	
7	Moduláris_hatványozó (2, n-1, n, c)
3	p←(c =1)
4	**RETURN** (p)

Ha ez az algoritmus azt mondja, hogy a szám összetett, akkor az biztosan nem lesz prím. Ha azt mondja, hogy lehet, hogy prím, akkor nagy eséllyel valóban prímet vizsgált, ugyanis 10000-ig terjedően a számok között csak 22 olyan van, amely nem prím és a teszt esetlegesen prímnek minősíti. Ilyenek a 341, 561, 645, 1105, … . Ötven bites számok esetén már csak a számok egy

milliomod része lehet ilyen, 100 biteseknél pedig ez az arány $1:10^{13}$. Ezen hibák egy része kiszűrhető azzal, hogy a 2 helyett más alapot is beveszünk a moduláris hatványozásba, például a 3-at, stb. Sajnos azonban vannak olyan számok, amelyek mindegyik alap esetén prímnek maszkírozzák magukat ennél az algoritmusnál. Ezek az úgynevezett *Carmichael számok*. A Carmichael számok relatíve nagyon kevesen vannak. (Valójában végtelen sok ilyen szám van. Ilyenek: 561, 1105, 1729,… Az első egy milliárd szám között csak 255 ilyen van.)

2. Példa: Döntsük el, hogy a 11 és a 12 prímek-e?

<table>
<tr><td>2^{10}=? mod 11, 10 = (1010)</td><td>2^{11}=? mod 12, 11=(1011)</td></tr>
<tr><td>
<table>
<tr><td>3</td><td>1</td><td>$1^2 = 1$</td><td>$1\cdot 2 = 2$</td></tr>
<tr><td>2</td><td>0</td><td>$2^2 = 4$</td><td></td></tr>
<tr><td>1</td><td>1</td><td>$4^2 = 16 \equiv 5$</td><td>$5\cdot 2 = 10$</td></tr>
<tr><td>0</td><td>0</td><td>$10^2 = 100 \equiv 1$</td><td></td></tr>
</table>
</td><td>
<table>
<tr><td>3</td><td>1</td><td>$1^2 = 1$</td><td>$1\cdot 2 = 2$</td></tr>
<tr><td>2</td><td>0</td><td>$2^2 = 4$</td><td></td></tr>
<tr><td>1</td><td>1</td><td>$4^2 = 16 \equiv 4$</td><td>$4\cdot 2 = 8$</td></tr>
<tr><td>0</td><td>1</td><td>$8^2 = 64 \equiv 4$</td><td>$4\cdot 2 = 8$</td></tr>
</table>
</td></tr>
<tr><td>$2^{10}\equiv 1$ mod 11 Tehát a 11 nagy eséllyel prím.</td><td>$2^{11}\equiv 8$ mod 12 Tehát a 12 nem prím.</td></tr>
</table>

Ezen előkészületek után térjünk rá a fejezet céljára a nyilvános kulcsú titkosításra A titkosítás alapja az eredeti szöveg átalakítása, kódolása. A nyílvános kulcsok használata azt jelenti, hogy minden résztvevőnek van egy nyílvános, mindenki számára hozzáférhető kulcsa (P, személyes, Private) és egy titkos, más által nem ismert kulcsa (S, titkos, Secret). Legyen M az üzenet. Legyen a két résztvevő A és B. A küldi B-nek az M üzenetet titkosítva. Az elküldött titkosított szöveg $C=P_B(M)$, B megkapja a C üzenetet és a titkos kulcsával dekódolja $M=S_B(C)$. A kulcsok egymás inverzei, és úgy vannak kialakítva, hogy a P kulcs révén könnyű legyen titkosítani, de a kulcs ismeretében nagyon nehezen lehessen - praktikusan lehetetlen legyen - az S kulcsot meghatározni. A digitális aláírás ilyenkor történhet úgy, hogy a küldő a titkosított C szöveg mellé akár nyíltan odaírja a saját Q azonosítóját (aláírását), majd annak az $R=S_A(Q)$ titkosítottját. Ezután B a Q alapján tudva, hogy kit nevez meg az aláírás, annak privát kulcsával dekódolja R-et. $Q^*=P_A(R)$. Ha $Q^*=Q$, akkor nem történt átviteli hiba, vagy hamisítás, egyébként igen. Persze Q az M-mel együtt is kódolható. Ez annak felel meg, mintha az első esetben nyílt levelezőlapon lenne az aláírásunk, a másodikban pedig mintha borítékba tettük volna.

Alább közöljük az RSA (Rivest – Shamir - Adleman) nyílvános kulcsú titkosítás algoritmusát. Az algoritmus feltételez két nagy prímszámot. (A gyakorlatban legalább 100-200 jegyűekre van szükség, hogy a titkosítás praktikusan feltörhetetlen legyen.) A P kulcs felépítése $P=(e,n)$, ahol n a két prím szorzata, e pedig egy kis páratlan szám. Az S kulcs $S=(d,n)$.

	2.6.3. algoritmus **RSA kulcsok meghatározása**
1	RSA_kulcsok_meghatározása (p, q, e, P, S)
2	// Input paraméterek: p, q, e
3	// Output paraméterek: P, S
4	**IF** *p* vagy *q* nem prím vagy *e*<*3* vagy e páros
5	**THEN RETURN** („Nincs kulcs”)
6	$n \leftarrow p \cdot q$
7	$f \leftarrow (p-1)\cdot(q-1)$
8	**IF** $lnko(e, f) \neq 1$
9	**THEN RETURN** („Nincs kulcs”)
10	$d \leftarrow e^{-1} \bmod f$
11	**RETURN** $(P=(e,n), S=(d,n))$

A szöveg titkosítása a $C = P(M) = M^e \bmod n$ alapján történik. Dekódolása pedig az $M = S(C) = C^d \bmod n$ alapján. A szöveg darabolásának bitméretét az n szabja meg.

Az eljárás helyességét nem bizonyítjuk.

3. Példa: Számpélda RSA algoritmusra (nem életszerű, mivel a prímek kicsik)

Legyen a titkos választás:
$p = 11,\ q = 29,\ n = p \cdot q = 11 \cdot 29 = 319,\ e = 3,\ f = (p-1)\cdot(q-1) = 10 \cdot 28 = 280$
A kibővített euklideszi algoritmust alkalmazzuk.

f	e	$\lfloor f/e \rfloor$	$f \bmod e$	d^*	x	y
280	3	93	1	1	1	- 93
3	1	3	1	1	0	1
1	0	-	1	1	1	0

Láthatóan $Lnko(f, e) = 1$ és e multiplikatív inverze $d = e^{-1} = -93$. Ez utóbbi helyett 280-at hozzáadva vesszük a 187-et. Ezek után akkor

$P = (3;319)$	közölhető kulcs	$P(M) = M^3 \bmod 319$
$S = (187;319)$	titkos kulcs	$S(C) = C^{187} \bmod 319$

Legyen az üzenetünk 100. Egy darabban titkosítható, mivel ez kisebb, mint 319. Titkosítsuk, majd fejtsük meg az üzenetet.

Titkosítás: C= $100^3 \bmod 319$ $\qquad 3_{10}=11_2$

1	1	$1^2 = 1 \equiv 1$	$1 \cdot 100 = 100 \equiv 100$
0	1	$100^2 = 10000 \equiv 111$	$111 \cdot 100 = 11100 \equiv 254$

Tehát a titkosított érték: $C = P(M) = 254$

Megfejtés: $M = 254^{187} \bmod 319$ $\qquad 187_{10} = 10111011_2$

7	1	$1^2 = 1 \equiv 1$	$1 \cdot 254 = 254 \equiv 254$
6	0	$254^2 = 64516 \equiv 78$	
5	1	$78^2 = 6084 \equiv 23$	$23 \cdot 254 = 5842 \equiv 100$
4	1	$100^2 = 10000 \equiv 111$	$111 \cdot 254 = 28194 \equiv 122$
3	1	$122^2 = 14884 \equiv 210$	$210 \cdot 254 = 53340 \equiv 67$
2	0	$67^2 = 4489 \equiv 23$	
1	1	$23^2 = 529 \equiv 210$	$210 \cdot 254 = 53340 \equiv 67$
0	1	$67^2 = 4489 \equiv 23$	$23 \cdot 254 = 5842 \equiv 100$

Tehát a megfejtés: $M = S(C) = 100$

FELADATOK

1. Bizonyítsuk be, hogy ha $a \equiv b \bmod n$ és k közös osztója a és b-nek, akkor $\frac{a}{k} \equiv \frac{b}{k} \bmod \frac{n}{lnko(k,n)}$!
2. Oldjuk meg az alábbi lineáris kongruencia egyenleteket! Adjuk meg a megoldások alaprendszerét! Írjuk fel a teljes megoldásrendszert!
   a. $2x \equiv 6 \pmod 8$
   b. $4x \equiv 4 \pmod 4$
   c. $18x \equiv 24 \pmod{60}$
   d. $63x \equiv 81 \pmod{72}$
   e. $2006x \equiv 2005 \pmod{2007}$
3. Határozzuk meg az alábbi számokat, ha értelmezve vannak! Az alapértelmezett megoldást adjuk meg!
   a. $x \equiv 5^{-1} \pmod 9$
   b. $x \equiv 2006^{-1} \pmod{2007}$
   c. $x \equiv 511^{-1} \pmod{1023}$
4. Számítsuk ki az alábbi számokat!
   a. $100^{100} \pmod{101}$
   b. $123^{2006} \pmod{2007}$
   c. $2^{65536} \pmod{255}$
5. Mit mond a Fermat féle álprímteszt az alábbi számok esetén?
   123, 234, 345, 511, 1023, 1105, 2047, 65535
6. A üzen B-nek. RSA kódolással kódoljuk, majd dekódoljuk az alábbi üzeneteket és a hozzátartozó aláírást! Maximum hány bites egységekre lehet tördelni az üzenetet?
   a. $p_A=29$, $q_A=31$, $e_A=11$, $p_B=97$, $q_B=101$, $e_B=7$, M=”x”, Q=”A”

3. Elemi dinamikus halmazok

3.1. A tömb adatstruktúra

Egy adastruktúra számtalan adatot tartalmazhat. Mondhatjuk, hogy egy adathalmazt tárolunk egy struktúrában. Számunkra a dinamikus halmazok lesznek fontosak.

Definíció: Dinamikus halmaz
Az olyan halmazt, amely az őt felhasználó algoritmus során változik (bővül, szűkül, módosul) *dinamikus halmaz*nak nevezzük.

A dinamikus halmazok elemei tartalmazhatnak az információs adatmezőiken felül *kulcsmező*t, és *mutató*kat (pointereket), amelyek a dinamikus halmaz más elemeire mutatnak. (pl: a következő elemre). Felsorolunk a dinamikus halmazokon néhány általánosságban értelmezett műveletet. Konkrét esetekben ezek közül egyesek el is maradhatnak, vagy továbbiak is megjelenhetnek. Az *S* jelöli a szóban forgó halmazt, *k* kulcsot ad meg és *x* mutató a halmaz valamely elemére. Feltételezzük, hogy a kulcsok között értelmezett a kisebb, nagyobb, egyenlő reláció.

Lekérdező műveletek	
KERES (*S, k, x*)	adott *k* kulcsú elem *x* mutatóját adja vissza, vagy NIL, ha nincs.
MINIMUM (*S, x*)	A legkisebb kulcsú elem mutatóját adja vissza
MAXIMUM (*S, x*)	A legnagyobb kulcsú elem mutatóját adja vissza
KÖVETKEZŐ (*S, x, y*)	az *x* elem kulcsa utáni kulcsú elem mutatóját adja vissza, NIL, ha *x* után nincs elem
ELŐZŐ (*S, x, y*)	az *x* elem kulcsa előtti kulcsú elem mutatóját adja vissza, NIL, ha *x* előtt nincs elem

Módosító műveletek	
BESZÚR (*S, x*)	az *S* bővítése az *x* mutatójú elemmel
TÖRÖL (*S, x*)	az *x* mutatójú elemet eltávolítja *S*-ből

Az egyes műveletek végrehajtásukat tekintve lehetnek statikusak (passzívak), vagy dinamikusak (aktívak) aszerint, hogy a struktúrát változatlannak hagyják-e vagy sem. A módosító műveletek alapvetően dinamikusak, a lekérdezők általában statikusak, de nem ritkán lehetnek szintén dinamikusak. (A dinamikus lekérdezés olyan szempontból érdekes és fontos, hogy ha egy elemet a többitől gyakrabban keresnek, akkor azt a struktúrában a keresés folyamán a megtalálási útvonalon közelebbi helyre helyezi át a művelet, ezzel megrövidíti a későbbi keresési időt erre az elemre, vagyis a művelet változást eredményez a struktúrában.)

Definíció: A sorozat adatstruktúra
Sorozatnak nevezzük az objektumok (elemek) olyan tárolási módját (adatstruktúráját), amikor az elemek a műveletek által kijelölt lineáris sorrendben követik egymást. Tipikus műveletek: keresés, beszúrás, törlés.

A sorozat egyik lehetséges *implementációja* - gyakorlati megvalósítása, megvalósítási eszköze – a tömb. A *tömb* azonos felépítésű (típusú) egymást fizikailag követő memóriarekeszeket jelent. Egy rekeszben egy elemet, *adatrekord*ot helyezünk el. Az egyes tömbelemek helyét az indexük határozza meg. Az elemek fontos része a kulcsmező, melyet **kulcs**[A_x] révén kérdezhetünk le az A tömb *x* indexű eleme esetén. Számunkra lényegtelen lesz, de a gyakorlat szempontjából alapvetően fontos része az adatrekordnak az információs (adat) mezőkből álló

rész. A tömböt szokás *vektor*nak is nevezni. Ha a lineáris elhelyezésen kívül egyéb szempontokat is figyelembe veszünk, akkor ezt az egyszerű szerkezetet el lehet bonyolítani. Ha például az elemek azonosítására indexpárt használunk, akkor *mátrix*ról vagy *táblázat*ról beszélünk. Ilyen esetben az első index a sort, a második az oszlopot adja meg. (Itt tulajdonképpen olyan vektorról van szó, amelynek elemei maguk is vektorok.)

A struktúrának és így az implementációnak is lehetnek *attributuma*i – jellemzői, hozzákapcsolt tulajdonságai. A tömb esetében ezeket az alábbi táblázatban adjuk meg.

Attributum	Leírás
***fej**[A]*	A tömb első elemének indexe. NIL, ha a tömbnek nincs eleme.
***vége**[A]*,	A tömb utolsó elemének indexe. NIL, ha a tömbnek nincs eleme.
***hossz**[A]*	A tömbelemek száma. Zérus, ha a tömbnek nincs eleme.
***tömbméret**[A]*	annak a memóriaterületnek a nagysága tömbelem egységben mérve, ahová a tömböt elhelyezhetjük. A tömb ezen terület elején kezdődik.

Vizsgáljuk meg most a műveletek algoritmusait!

A keresési algoritmus. Az *A* tömbben egy *k* kulcsú elem keresési algoritmusa pszeudokóddal lejegyezve következik alább. Az algoritmus NIL-t ad vissza, ha a tömb üres, vagy a tömbben nincs benne a keresett kulcsú elem. A tömb elejétől indul a keresés. Ha a vizsgált elem egyezik a keresett elemmel, akkor azonnal viszatérünk az indexével. (Realizáció szempontjából úgy is elképzelhetjük a dolgot, hogy a tömb elemeinek indexelése 1-gyel kezdődik és a NIL eredményt a 0 indexszel jelezzük.) Ha nem egyezik, akkor az INC függvénnyel növeljük eggyel az index értékét (rátérünk a következő elemre) és újra vizsgálunk. Addig növeljük az indexet, míg az érvényes indextartományból ki nem lépünk vagy meg nem találjuk a keresett kulcsú elemet. A legrosszabb eset az, ha az elem nincs benne a tömbben, – ekkor ugyanis az összes elemet meg kell vizsgálni – így az algoritmus időigénye: $T(n)=\Theta(n)$, ahol $n=$***hossz**[A]*, a tömbelemek száma.

3.1.1. algoritmus
Keresés tömbben

// $T(n)=\Theta(n)$

```
1  KERESÉS_TÖMBBEN (A, k, x )
2  // Input paraméter: A - a tömb
3  //                  k – a keresett kulcs
4  // Output paraméter: x - a k kulcsú elem pointere (indexe), ha van ilyen elem
                          vagy NIL, ha nincs
5  // Lineárisan keresi a k kulcsot.
6  //
7  x ← fej[A]
8  IF  hossz[A] ≠ 0
9      THEN    WHILE    x ≤ vége[A] és kulcs[A_x] ≠ k DO
10                      INC(x)
11             IF       x> vége[A]
12                      THEN x ← NIL
13 RETURN (x)
```

Az új elem beszúrásának algoritmusa az A tömb adott *x* indexű helyére szúrja be az új elemet. Az ott lévőt és a mögötte állókat egy hellyel hátrább kell tolni. Emiatt az időigény $T(n)=\Theta(n)$.

	3.1.2. algoritmus **Beszúrás tömbbe**
	// $T(n)=\Theta(n)$
1	BESZÚRÁS_TÖMBBE (*A, x, r, hibajelzés*)
2	// Input paraméter: *A* - a tömb
3	// *x* – a tömbelem indexe, amely elé történik a beszúrás, ha a tömb nem üres és az *x* index létező elemre mutat. Üres tömb esetén az *x* indexnek nincs szerepe, a beszúrandó elem az első helyre kerül.
4	// *r* a beszúrandó elem (rekord)
5	// Output paraméter: *hibajelzés* - a beszúrás eredményességét jelzi
6	//
7	**IF** ***hossz****[A]* ≠ 0
8	**THEN IF** (***fej****[A]* ≤ *x* ≤ ***vége****[A]*) és (***tömbméret****[A]* > ***hossz****[A]*)
9	**THEN FOR** *i* ← ***vége****[A]* **DOWNTO** *x* **DO**
	$A_{i+1} \leftarrow A_i$
	$A_x \leftarrow r$
	INC(***hossz****[A]*)
	INC(***vége****[A]*)
	hibajelzés: ← „sikeres beszúrás”
	ELSE *hibajelzés*: ← „nem létező elem,vagy nincs az új elemnek hely”
16	**ELSE** ***fej****[A]* ← ***vége****[A]* ← ***hossz****[A]* ← 1
17	$A_1 \leftarrow r$
18	**RETURN (** *hibajelzés* **)**

Ezzel az algoritmussal nem tudunk az utolsó elem után beszúrni. A problémát egy erre a célra megírt külön algoritmussal is megoldhatjuk. Legyen ennek CSATOL_TÖMBHÖZ a neve. Az olvasóra bízzuk pszeudokódjának megírását. Írjunk pszeudokódot arra az esetre, amikor a beszúrás az adott indexű elem mögé történik! Ennek is van egy szépséghibája! Micsoda? Hogyan korrigálható?