

Păţcaş Csaba György

Ionescu Klára

**Hogyan
válhatunk
profi
programozókká?**

– Megoldott és kitűzött feladatok –

Păţcaş Csaba György

Ionescu Klára

**Hogyan
válhatunk
profí
programozókká?**

Presa Universitară Clujeană / Kolozsvári Egyetemi Kiadó

2019



A kiadvány megjelenését az NTP-HTTSZ-M-19-0005 kódszámú pályázaton keresztül az Emberi Erőforrások Minisztériuma támogatta. A pályázatot az Emberi Erőforrások Minisztériuma megbízásából az Emberi Erőforrás Támogatáskezelő hirdette meg.

© Pătaș Csaba György és Ionescu Klára 2019

Minden jog fenntartva. Jelen könyvet vagy annak részleteit tilos reprodukálni, adatrendszerben tárolni, bármely formában vagy eszközzel – elektronikus, fényképeseti úton vagy más módon – a kiadó engedélye nélkül közölni.

Szaklektorok:

Dr. Darvay Zsolt egyetemi docens

Dr. Gaskó Noémi egyetemi docens

Borítóterv: Bodó Zalán

Tartalomjegyzék

Tartalomjegyzék	5
1. Bevezetés	7
2. Forrásaink	8
3. 2019	9
3.1. Általánosságok – 2019	9
3.2. Matek–Infó verseny, 2019	11
3.3. Felvételi vizsga, 2019	27
4. 2018	45
4.1. Általánosságok – 2018	45
4.2. Matek–Infó verseny, 2018	46
4.3. Felvételi vizsga (július), 2018	60
4.4. Felvételi vizsga (szeptember), 2018	73
5. 2017	86
5.1. Általánosságok – 2017	86
5.2. Matek–Infó verseny, 2017	88
5.3. Felvételi vizsga, 2017	99
6. 2016	109
6.1. Általánosságok – 2016	109
6.2. Matek–Infó verseny, 2016	111
6.3. Felvételi vizsga, 2016	120
7. Kitűzött feladatok	132

1. Bevezetés

Versenyfeladatokat készíteni mindig komoly kihívás. A feladatkészítőnek feltétlenül ismernie kell a válaszokat a következő kérdésekre: milyen céllal szervezik az illető versenyt? Gépen vagy papíron, esetleg mindkettőn dolgoznak majd a versenyzők? Mennyire magas/alacsony a jelentkezők színvonala? Kik, és milyen körülmények között fogják az értékelést végezni?

A Babeş–Bolyai Tudományegyetem Matematika és Informatika Kara néhány éve felvételi versenyt szervez az érettségi után azoknak, akik egyetemünk diákjaivá szeretnének válni – vagyis „profi programozókká”. A felvételizőnek mind a felvételi vizsgán, mind a tavasszal megrendezésre kerülő Matek–Infó versenyen több feladatot kell megoldania.

Napjainkban a legtöbb informatikaverseny számítógépen történik. De mit lehet tenni, ha 1700-nál több a jelentkező? Ráadásul, itt nem az arany-, ezüst-, bronzérmeseket keressük, hanem egy bizonyos tudásszint fölöttieket. De ez nem lehet egyszerűen az eddig tanult anyag visszakérdezése. Készségeket, képességeket és alkotó fantáziát szeretnénk felmérni. Szeretnénk tudni, hogy képes-e megfelelően figyelni, koncentrálni a versenyző? Rájön-e a kérdés lényegére, és így a valóban feltett kérdésre válaszol, nem pedig egy másikra? Tud-e logikusan gondolkodni? Felismeri több (javasolt) megoldás közül a helyeset? Kiemelt szerepet kap a modellezési feladat, amely az egyetlen, ahol a versenyzőnek algoritmust/programot kell írnia.

Az utóbbi négy év feladatait vesszük sorra, elemezzük, elmagyarázzuk a megoldásokat. Ezeken kívül hasonló jellegűeket is bemutatunk, kitűzött feladatokként, mivel szeretnénk egy kevés munkát az olvasónak is hagyni. 😊

A bizottság elnöke 2016-tól 2019-ig *dr. Laura Dioşan* (egyetemi tanár)
Tagjai:

- *Dr. Ionescu Klára* (egyetemi adjunktus): 2016–2019
- *Dr. Dana Lupşa* (egyetemi adjunktus): 2016
- *Dr. Adriana Guran* (egyetemi adjunktus): 2017–2019
- *Dr. Diana Cristea* (egyetemi adjunktus): 2018–2019

Köszönjük munkájukat és az általuk javasolt feladatokat!

Forrásaink

1. A Babeş–Bolyai Tudományegyetem honlapján belül:
<http://www.cs.ubbcluj.ro/felveteli/alapkepzes/tematika-felveteli-kar/>
2. Érettségi mintafeladatok:
[http://www.cnszalai.ro/wp-content/uploads/2014/03/2009_Variante Informatica.pdf](http://www.cnszalai.ro/wp-content/uploads/2014/03/2009_Variante_Informatica.pdf)
3. Ionescu Klára: *Bevezetés az algoritmikába*, Egyetemi Könyvkiadó Kolozsvár, 2005.
4. *Gazeta de Informatică*, Libris / Agora Media / Computer Libris Agora, 1993–2008

3.1. Általánosságok – 2019

2019-ben a versenyfeladatokat – mind a *Matek–Infó versenyen*, mind a *felvételin* – a következő figyelmeztető szöveg előzte meg:

A versenyzők figyelmébe:

1. Minden tömböt 1-től kezdődően indexelünk.
2. Az **A** rész rácstesztjeire egy vagy több helyes válasz lehetséges, ezeket a vizsgadolgozat megfelelő formanyomtatványába írjátok. A rácsteszték pontozása a verseny szabályzatában leírt részleges pontozással történik.
3. A **B** rész feladatainak teljes megoldását a vizsgadolgozatba írjátok. Ezek részletes értékelése a javítókulcs alapján történik.
 - a. A **B1-B4** követelmények megoldásait beszélt nyelven/matematikai eszközökkel, illetve *pszeudokódban* vagy egy programozási nyelvben (*Pascal/C/C++*) kell megadnotok.
 - b. A megoldások értékelésekor az első szempont az algoritmus *helyessége*, majd a *hatékonysága*, ami a *végrehajtási időt* és a *felhasznált memória méretét* illeti.
 - c. A tulajdonképpeni megoldás *előtt*, *kötelezően írjátok le szavakkal az alprogramokat, és indokoljátok meg a megoldások lépéseit*. Írjátok *megjegyzéseket* (kommenteket), amelyek segítik az adott megoldás technikai részleteinek megértését, az azonosítók jelentését és a fölhasznált adatszerkezeteket stb. Ha ez hiányzik, a tételre kapható pontszámotok 10% -kal csökken.
 - d. Ne használjatok fejállományokat, előre definiált függvényeket (pl. *STL*, karakterláncokat feldolgozó sajátos függvények stb.).

A feladatlap néhány hasznos információval záródik:

Megjegyzések:

1. Minden tétel kidolgozása kötelező.
2. A piszkozatokat nem vesszük figyelembe.
3. Hivatalból jár 10 pont.
4. Rendelkezésekre áll 3 óra.

3.2. Matek–Infó verseny – 2019. április 6.

A Rész (60 pont)

A.1. Mit ír ki? (6 pont)

Legyen a következő program. Állapítsátok meg, mit ír ki a program a végrehajtás eredményeként.

A. 7;11

B. 6;9

C. 7;9

D. 7;12

C	
<pre>#include <stdio.h> int feldVektor(int v[], int *n){ int s = 0; int i = 2; while (i <= *n) { s = s + v[i] - v[i - 1]; if (v[i] == v[i - 1]) *n = *n - 1; i++; } return s; }</pre>	<pre>int main(){ int v[8]; v[1] = 1; v[2] = 4; v[3] = 2; v[4] = 3; v[5] = 3; v[6] = 10; v[7] = 12; int n = 7; int eredmeny = feldVektor(v, &n); printf("%d;%d", n, eredmeny); return 0; }</pre>
C++	
<pre>#include <iostream> using namespace std; int feldVektor(int v[], int &n){ int s = 0; int i = 2; while (i <= n) { s = s + v[i] - v[i - 1]; if (v[i] == v[i - 1]) n--; i++; } return s; }</pre>	<pre>int main(){ int v[8]; v[1] = 1; v[2] = 4; v[3] = 2; v[4] = 3; v[5] = 3; v[6] = 10; v[7] = 12; int n = 7; int eredmeny = feldVektor(v, n); cout << n << ";" << eredmeny; return 0; }</pre>
Pascal	
<pre>type vector = array[1..10] of integer; function feldVektor(v: vector; var n: integer): integer; var s, i: integer; begin s := 0; i := 2; while (i <= n) do begin s := s + v[i] - v[i - 1]; if (v[i] = v[i - 1]) then n := n - 1; i := i + 1; feldVektor := s; end; end;</pre>	<pre>var n, eredmeny: integer; v: vector; Begin n := 7; v[1] := 1; v[2] := 4; v[3] := 2; v[4] := 3; v[5] := 3; v[6] := 10; v[7] := 12; eredmeny := feldVektor(v, n); write(n, ';', eredmeny); End.</pre>

Megoldás

Adott egy alprogram és az alprogramot meghívó programegység. A kérdés egyszerű, hiszen csak azt kell kiválasztanunk (a lehetséges változatok közül), hogy mit ír ki a hívó programegység. Ez a feladattípus valóban könnyű, de figyelni, koncentrálni elengedhetetlen: észre kell vennünk a paraméterátadás módját (érték szerint, cím szerint, esetleg referencia szerint), és az értékek változását is figyelmesen kell követnünk. Érdemes táblázatot készíteni. A versenyző a három programozási nyelvből azt választja ki, amelyiket a legjobban ismeri.

A program rögzíti a bemenet értékeit: $n = 7$, $v = (1, 4, 2, 3, 3, 10, 12)$. Észrevesszük, hogy az alprogram kiszámítja a tömb elemeinek összegét a másodiktól kezdve, de az összegből minden lépésben kivonja az aktuális elem előtti elem értékét. Tehát: $(4 - 1) + (2 - 4) + (3 - 2) + (3 - 3) + (10 - 3) + (12 - 10) = 3 - 2 + 1 + 0 + 7 + 2 = 11$. Ugyanakkor azt is fontos észrevennünk, hogy a sorozatban létezik két egymás után elhelyezkedő azonos értékű elem, tehát n értéke egyszer csökken. Így (mivel csökken a sorozat azon része, amit fel kell dolgoznunk) az utolsó tagot (2) nem adjuk az összeghez. A program kiírja n megváltozott értékét (6) és a kiszámolt eredményt (9). Tehát a helyes válasz: **B**.

A.2. Logikai kifejezés (6 pont)

Állapítsátok meg, hogy a következő kifejezések közül melyiknek lesz az értéke akkor és csakis akkor igaz, ha az n természetes szám osztható 3-mal és az utolsó számjegye 4 vagy 6:

- A. $n \text{ DIV } 3 = 0$ és $(n \text{ MOD } 10 = 4 \text{ vagy } n \text{ MOD } 10 = 6)$
- B. $n \text{ MOD } 3 = 0$ és $(n \text{ MOD } 10 = 4 \text{ vagy } n \text{ MOD } 10 = 6)$
- C. $(n \text{ MOD } 3 = 0 \text{ és } n \text{ MOD } 10 = 4)$ vagy $(n \text{ MOD } 3 = 0 \text{ és } n \text{ MOD } 10 = 6)$
- D. $(n \text{ MOD } 3 = 0 \text{ és } n \text{ MOD } 10 = 4)$ vagy $n \text{ MOD } 10 = 6$

Megoldás

Adott több logikai kifejezés, amelyekben a részkifejezések relációs, illetve aritmetikai kifejezések. A megoldáshoz szükséges ismerni a logikai és aritmetikai operátorokat, valamint az operátorok precedenciáját. Ahhoz, hogy a feltett kérdésre válaszolhassunk, kiértékeljük a kifejezéseket, hiszen több jó megoldás is előfordulhat.

Választunk – tetszőlegesen – egy természetes számot, amely osztható 3-mal és az utolsó számjegye 4 vagy 6. Elvégezzük az aritmetikai műveleteket, majd a kapott részeredményekre elvégezzük a logikai műveleteket. A zárójelekben található kifejezéseket kiértékeljük, és a kapott részeredményekkel elvégezzük a logikai műveleteket.

Legyen, például $n = 24$ (utolsó számjegye 4) $\Rightarrow 24 \text{ DIV } 3 = 8$, tehát nem 0 \Rightarrow a $(24 \text{ DIV } 3 = 0)$ kifejezés értéke *hamis*. Mivel a következő operátor az „és”, nincs értelme kiértékelni a következő részkifejezést (tudjuk, hogy $(\text{hamis és hamis}) = \text{hamis}$, valamint $(\text{hamis és igaz}) = \text{hamis}$). Az **A.** választ kizárjuk a jó válaszok közül. Ha $n = 36$ (utolsó számjegye 6), ugyanerre az eredményre jutunk.

A **B.** kifejezés esetében az első részkifejezést 24-re és 36-ra is igaznak találjuk, mivel $24 \text{ MOD } 3 = 0$ és $36 \text{ MOD } 3 = 0$. Ebben az esetben ki kell számolnunk a második részkifejezés értékét is: $(24 \text{ MOD } 10 = 4 \text{ vagy } 24 \text{ MOD } 10 = 6)$. Ha $n = 24$, az első részkifejezés *igaz*, és ha $n = 36$, *igaz* a második részkifejezés. Mivel az őket összekötő operátor a „vagy”, és tudjuk, hogy $(\text{igaz vagy hamis} = \text{igaz})$, illetve $\text{hamis vagy igaz} = \text{igaz}$, végrehajtjuk a zárójel előtti és műveletet: $(\text{igaz és igaz} = \text{igaz})$, így a **B.** választ jónak minősítjük.

Hasonlóképpen kiértékeljük a **C.** kifejezést: ha $n = 24$, az első zárójelben található részkifejezés értéke *igaz*, a második *hamis*, de a részeredmény *igaz*. Mivel a következő művelet „vagy”, a második kifejezést nem értékeljük ki, és levonjuk a következtetést, hogy az eredmény *igaz*. Ha $n = 36$, a második részkifejezés értéke *hamis*, tehát az első zárójelben levő részkifejezés értéke *hamis*. Most fontos kiértékelni a második zárójelben levő részkifejezés értékét is. A részeredmény *igaz*, tehát a végeredmény is *igaz*. A **C.** választ is jónak találjuk. A megfelelő számolásokkal a **D.** választ *hamis*-nak találjuk, tehát a jó válaszok: **B.** és **C.**

A.3. Ackermann (6 pont)

Legyenek az m és n természetes számok ($0 \leq m \leq 10, 0 \leq n \leq 10$), valamint az $\text{Ack}(m, n)$ algoritmus, amely kiszámítja az Ackermann-függvény értékét m és n esetében. Állapítsátok meg, hányszor hívja meg önmagát az $\text{Ack}(m, n)$ algoritmus a következő utasítások végrehajtásának következtében.

$m \leftarrow 1, n \leftarrow 2$ $\text{Ack}(m, n)$
--

<p>Algoritmus $\text{Ack}(m, n)$</p> <p>Ha $m = 0$ akkor térítsd $n + 1$</p> <p>különbén Ha $m > 0$ és $n = 0$ akkor térítsd $\text{Ack}(m - 1, 1)$</p> <p>különbén térítsd $\text{Ack}(m - 1, \text{Ack}(m, n - 1))$</p> <p> vége(ha)</p> <p> vége(ha)</p> <p>Vége(algoritmus)</p>
--

- A. 7-szer
- B. 10-szer
- C. 5-ször
- D. ugyanannyiszor, mint az alábbi utasítások végrehajtásának következtében:

$m \leftarrow 1, n \leftarrow 3$ $\text{Ack}(m, n)$
--

Megoldás

Adott egy rekurzív alprogram, és meg kell állapítanunk, hogy a paraméterek adott értékére hányszor hívja meg önmagát. A megoldáshoz követnünk kell a paraméterek értékeit a hívássorozaton belül, – esetleg lerajzoljuk a végrehajtási verem tartalmát és megszámloljuk a rekurzív hívásokat:

$\text{Ack}(1, 2) \Rightarrow$ (1. hívás):

$\text{Ack}(0, \text{Ack}(1, 1)) \Rightarrow$ (2. hívás):

$\text{Ack}(1, \text{Ack}(1, 0)) \Rightarrow$ (3. hívás):

$\text{Ack}(0, 1) \Rightarrow$ (4. hívás):

$\text{Ack}(0, 2) \Rightarrow$ (5. hívás)

$\text{Ack}(0, 3)$: vége.

		Hívás sorszáma
3	n	5
0	m	
2	n	4
0	m	
1	n	3
0	m	
0	n	2
1	m	
1	n	1
1	m	
2	n	0 (első hívás)
1	m	

Tehát, a rekurzív hívások száma 5, vagyis a **C.** válasz jó, így azt is tudjuk bizonyosan, hogy az **A.** és **B.** válaszok nem jók. De maradt még egy válasz, amelyről még nem tudjuk, hogy helyes vagy sem. A fent leírt módon megvizsgáljuk ezt az esetet is, és megállapítjuk, hogy a rekurzív hívások száma 7, ami különbözik 5-től. Tehát a **D.** válasz sem jó.

A.4. Csonkított számok összege (6 pont)

Definiáljuk a $\overline{c_1 c_2 \dots c_k}$ számjegyekből álló k természetes számra a *csonkítás* műveletet: $\text{csonkít}(\overline{c_1 c_2 \dots c_k}) = \begin{cases} 0, & \text{ha } k < 2 \\ \overline{c_1 c_2}, & \text{különben} \end{cases}$

Állapítsátok meg, hogy az alábbi algoritmusok közül melyik határozza meg az n elemű x sorozat *csonkított elemeinek összegét*. Az elemek természetes számok és kisebbek, mint 1 000 000 (n – természetes szám és $1 \leq n \leq 1\,000$). Például, ha $n = 4$ és $x = (213, 7, 78347, 22)$, akkor a csonkított elemek összege $21 + 0 + 78 + 22 = 121$.

A. Algoritmus csontkítás(n, x) $s \leftarrow 0$ Amíg $n > 0$ végezd el Ha $x[n] > 9$ akkor Amíg $x[n] > 99$ végezd el $x[n] \leftarrow x[n] \text{ DIV } 10$ vége(amíg) $s \leftarrow s + x[n]$ vége(ha) $n \leftarrow n - 1$ vége(amíg) térítsd s Vége(algoritmus)	B. Algoritmus csontkítás(n, x) $s \leftarrow n$ Amíg $n > 0$ végezd el Ha $x[n] > 9$ akkor Amíg $x[n] > 99$ végezd el $x[n] \leftarrow x[n] \text{ DIV } 10$ vége(amíg) $s \leftarrow s + x[n]$ vége(ha) $n \leftarrow n - 1$ vége(amíg) térítsd s Vége(algoritmus)
C. Algoritmus csontkítás(n, x) $s \leftarrow 0$ Amíg $n > 0$ végezd el Ha $x[n] > 9$ akkor Amíg $x[n] > 99$ végezd el $x[n] \leftarrow x[n] \text{ DIV } 10$ $s \leftarrow s + x[n]$ vége(amíg) vége(ha) $n \leftarrow n - 1$ vége(amíg) térítsd s Vége(algoritmus)	D. Algoritmus csontkítás(n, x) $s \leftarrow 0$ Amíg $x[n] > 99$ végezd el $x[n] \leftarrow x[n] \text{ DIV } 10$ vége(amíg) $s \leftarrow s + x[n]$ térítsd s Vége(algoritmus)

Megoldás

Adott egy feladatszöveg és több algoritmus, amelyek közül ki kell választanunk a helyes algoritmust, esetleg algoritmusokat. Azt gondolhatnánk, hogy ez nem lehetséges csak úgy, ha a papíron „végrehajtjuk” mindegyiket. De erre most nincs szükség: ha figyelmesen tanulmányozzuk az algoritmusokat, észrevevessük, hogy nincs nagy különbség köztük, és el lehet különíteni a helyeseket anélkül, hogy táblázatban követnénk az értékeket.

Ha összehasonlítjuk az **A.** és **B.** algoritmusokat, azonnal észrevevessük, hogy az egyetlen különbség a két algoritmus első sorában van: az **A.** algoritmus s -nek 0 kezdőértéket ad, míg a **B.** algoritmusban a kezdőérték n . Természetesen, az n nincs mit keresen itt, hiszen n az x sorozat mérete! Így a **B.** algoritmust hibásnak minősítjük.

A továbbiakban az **A.** algoritmust bejárjuk gondolatban, és belátjuk, hogy helyes.

Mivel van egy jónak talált algoritmusunk, most ezt összehasonlítjuk a **C.** és **D.** algoritmusokkal. Észrevevessük, hogy a **C.**-ben az aktuális elem „túl korán” kerül be az összegbe, míg a **D.** algoritmus nem figyeli a sorozat végét, sőt az n változó értéke egyáltalán nem változik. Tehát a helyes válasz: **A.**

A.5. Mely értékek szükségesek? (6 pont)

Legyen a különbség(a , n) algoritmus, ahol a egy n elemű ($0 < n < 100$) sorozat, amely egész számokat tárol:

```

Algoritmus különbség( $a$ ,  $n$ )
  Ha  $n = 0$  akkor térítsd 0
  vége(ha)
  Ha  $|a[n]| \bmod 2 = 0$  akkor      //  $|a[n]|$  az  $a[n]$  szám abszolút értékét jelöli
    térítsd különbség( $a$ ,  $n - 1$ ) +  $a[n]$ 
  különben
    térítsd különbség( $a$ ,  $n - 1$ ) -  $a[n]$ 
  vége(ha)
Vége(algoritmus)

```

Az n és a változók mely értékeire térít a fenti algoritmus 0-át?

- A. $n = 4$ és $a = (6, 4, 5, 5)$
- B. $n = 4$ és $a = (-6, 5, 4, -7)$
- C. $n = 8$ és $a = (-6, 5, -1, -4, 1, 4, -7, 6)$
- D. $n = 8$ és $a = (-6, -3, 0, 1, 2, 3, -1, 4)$

Megoldás

Könnyű belátni, hogy a páros számokat összeadja az algoritmus, míg a páratlanokat kivonja az aktuális összegből, amit nem tárolunk egy változóban. Mivel az utolsó híváskor 0 kezdőértéket térítünk, minden rekurzív hívásból való visszatéréskor az aktuális összeghez hozzáadjuk a soron következő számot, ha a páros, illetve kivonjuk, ha páratlan. Tehát azokat a bemeneti adatokat választjuk ki, amelyeknek esetében a páros számok összege egyenlő a páratlanok összegével. Helyes válaszok: **A.**, **B.** és **D.**

A.6. Különleges számok sorozatának generálása (6 pont)

Legyen az s egy természetes számokat tároló sorozat, ahol

$$s_i = \begin{cases} x & \text{ha } i = 1 \\ x + 1 & \text{ha } i = 2, (i = 1, 2, \dots). \\ s_{i-1} @ s_{i-2} & \text{ha } i > 2 \end{cases}$$

A $@$ művelet a bal és a jobb operandus

számjegyeit konkatenálja (egymás után ragasztja) ebben a sorrendben, az x pedig egy természetes szám ($1 \leq x \leq 99$). Például, ha $x = 3$, az s sorozat elemei a következők: 3, 4, 43, 434, 43443, ...

Állapítsátok meg, hány számjegye van az s sorozat azon elemének, amely a k számjegyű elem előtt található ($1 \leq k \leq 30$).

- A. ha $x = 15$ és $k = 6$, az s sorozatban a k számjegyű elem előtt levő elem számjegyeinek száma 5.
- B. ha $x = 2$ és $k = 8$, az s sorozatban a k számjegyű elem előtt levő elem számjegyeinek száma 5.
- C. ha $x = 14$ és $k = 26$, az s sorozatban a k számjegyű elem előtt levő elem számjegyeinek száma 16.
- D. ha $x = 5$ és $k = 13$, az s sorozatban a k számjegyű elem előtt levő elem számjegyeinek száma 10.

Megoldás

Számolunk:

	A.	B.	C.	D.
<i>Elem sorszáma</i>	k	k	k	k
1	2	1	2	1
2	2	1	2	1
3	$4 \neq 5$	2	4	2
4	6	3	6	3
5		5	10	5
6		8	16	$8 \neq 10$
7			26	13

Mivel a feladat nem kéri a számsorozatot – elég, ha az elemek hosszával foglalkozunk. Észrevevesszük, hogy egy bizonyos elem hossza egyenlő a közvetlenül előtte levő két elem hosszának összegével (számjegyeiknek darabszámával).

Az **A.** válasz $x = 15$ -ből indul, a $k = 6$ számjegyű elemre hivatkozik, és feltételezi, hogy az s sorozatban a k számjegyű elem előtt levő elem számjegyeinek száma 5. Mivel a 6 számjegyű előtti szám négy számjegyű, az **A.** válasz nem helyes.

A **B.** válasz $x = 2$ -ből indul, a $k = 8$ számjegyű elemre hivatkozik, és feltételezi, hogy a sorozatban a k számjegyű elem előtt levő elem számjegyeinek száma 5. Mivel a 8 számjegyű előtti elem valóban ötszámjegyű, a **B.** válasz helyes.

A **C.** válasz 14-ből indul, a $k = 26$ számjegyű elemre hivatkozik, és feltételezi, hogy a k számjegyű elem előtt levő elem számjegyeinek száma 16, ami igaznak bizonyul (**C.** helyes).

A **D.** válasz 5-ből indul, a $k = 13$ számjegyű elemre hivatkozik, és feltételezi, hogy a k számjegyű elem előtt levő elem számjegyeinek száma 10, ami hamisnak bizonyul, tehát **D.** nem helyes.

A.7. Körkörös permutációk (6 pont)

Legyen az n ($3 \leq n \leq 10\,000$) elemű x sorozat, amely természetes számokat tárol és a k természetes szám ($1 \leq k < n$). A $\text{körkörösPerm}(n, k, x)$ algoritmusnak az x sorozat körkörös permutációját kellene generálnia k pozícióval balra. Például, a $(4, 5, 2, 1, 3)$ sorozat az $(1, 3, 4, 5, 2)$ sorozat körkörös permutációja két pozícióval balra. Sajnos, a $\text{körkörösPerm}(n, k, x)$ algoritmus nem helyes, mivel n és k bizonyos értékeire hibás eredményt generál.

```

Algoritmus körkörösPerm( $n, k, x$ )
   $c \leftarrow k$ 
  Minden  $j = 1, c$  végezd el
     $hova \leftarrow j$ ;  $s\acute{a}m \leftarrow x[hova]$ 
    Minden  $i = 1, n / c - 1$  végezd el
       $honna\!n \leftarrow hova + k$ 
      Ha  $honna\!n > n$  akkor
         $honna\!n \leftarrow honna\!n - n$ 
      vége(ha)
       $x[hova] \leftarrow x[honna\!n]$ 
       $hova \leftarrow honna\!n$ 
    vége(minden)
     $x[hova] \leftarrow s\acute{a}m$ 
  vége(minden)
Vége(algoritmus)

```

Válasszátok ki n , k és x értékeit, melyekre a $\text{körkörösPerm}(n, k, x)$ algoritmus az x sorozat körkörös permutációját generálja, k pozícióval balra:

A. $n = 6, k = 2, x = (1, 2, 3, 4, 5, 6)$

B. $n = 8, k = 3, x = (1, 2, 3, 4, 5, 6, 7, 8)$

C. $n = 5, k = 3, x = (1, 2, 3, 4, 5)$

D. $n = 8, k = 4, x = (1, 2, 3, 4, 5, 6, 7, 8)$

Megoldás

A legértékesebb észrevétel a bemeneti adatokra vonatkozik, ugyanis n és k értékei a **B.** és **C.** esetekben relatív prímek $(8, 3)$; és $(5, 3)$, az **A.** és **D.** esetekben pedig n -nek és k -nak van közös osztója $(6, 2)$; és $(8, 4)$. Sejtjük, hogy ez lehet az adatok közti különbség, amelynek következtében egyes bemeneti adatokra nem működik helyesen az algoritmus. Készítünk táblázatot $n = 6$ -ra és $k = 2$ -re:

	n	k	c	j	$hova$	$s\acute{a}m$	i	$honna\!n$	x
	6	2							1, 2, 3, 4, 5, 6
1. <i>bejárás</i> ($j = 1$)			2	1	1	1	1	3	3, 2, 3, 4, 5, 6
				1	3		2	5	3, 2, 5, 4, 5, 6
				1	5				3, 2, 5, 4, 1, 6
2. <i>bejárás</i> ($j = 1$)				2	2	2	1	4	3, 4, 5, 4, 1, 6
				2	4		2	6	3, 4, 5, 6, 1, 6
				2	6				3, 4, 5, 6, 1, 2

A fenti esetben az algoritmus helyesen működik. Lássuk mi történik, ha $n = 5$ és $k = 3$? Mivel $c = k$ és az i által vezérelt **Minden** ciklus 1-től $(n / c - 1)$ -ig dolgozik, ahol a végsőérték a példánk esetében $5 / 3 - 1 = 0$, a ciklus törzsét az algoritmus egyszer sem hajtja végre. Ennek következtében a sorozat nem változik. Tehát a **C.** válasz nem helyes. Innen tovább, végre lehetne hajtani az algoritmust a **B.** és a **D.** válaszok esetében is. De kiindulhatunk mindkét esetben a fentiekből, vagyis **A.** és **D.** helyes, **B.** és **C.** nem helyes.

A.8. Kiegészítés (6 pont)

Legyen a $\text{kizárPáratlan}(n)$ algoritmus, ahol n ($1 \leq n \leq 100\,000$) természetes szám. Állapítsátok meg, melyik utasítást kellene a „...” helyére írni, ahhoz, hogy az algoritmus zárja ki az n számból a páratlan értékű számjegyeket.

```

Algoritmus   $\text{kizárPáratlan}(n)$ 
  Ha  $n = 0$  akkor
    térítsd 0
  vége(ha)
  Ha  $n \bmod 2 = 1$  akkor
    térítsd  $\text{kizárPáratlan}(n \text{ DIV } 10)$ 
  vége(ha)
  térítsd ...
Vége(algoritmus)

```

- A. $\text{kizárPáratlan}(n \bmod 10) * 10 + n \text{ DIV } 10$
- B. $\text{kizárPáratlan}(n) * 10 + n \bmod 10$
- C. $\text{kizárPáratlan}(n \text{ DIV } 10) * 10 + n \bmod 10$
- D. $\text{kizárPáratlan}((n \text{ DIV } 10) \bmod 10) * 10$

Megoldás

Tulajdonképpen egy új számot kell felépítenünk az adott n szám páros számjegyeiből. Látjuk, hogy amíg n páratlan szám, megtörténik a rekurzív hívás, anélkül, hogy módosítanánk az új számnak megfelelő térítendő értéket. A páros számjegyek a C. válaszban leírt utasítás eredményeként épülnek majd a térítendő értékbe. Az eddig kiszámolt értéket szorozzuk 10-zel és hozzáadjuk a páros számjegy értékét. Az n paraméter új értéke $n / 10$. Tehát a helyes válasz: C. A többi válasz közül – ilyen körülmények között egy sem lehet helyes.

A.9. Vajon mit csinál? (6 pont)

Az m és n hosszú oldalakkal rendelkező téglalap fel van osztva 1 oldalhosszúságú négyzetecskékre (m, n – természetes számok, $0 < m < 101$, $0 < n < 101$). Adott a $\text{téglalap}(m, n)$ algoritmus. Mi a hatása ennek az algoritmusnak?

```

Algoritmus   $\text{téglalap}(m, n)$ 
   $d \leftarrow m$ 
   $c \leftarrow n$ 
  Amíg  $d \neq c$  végezd el
    Ha  $d > c$  akkor
       $d \leftarrow d - c$ 
    különben
       $c \leftarrow c - d$ 
    vége(ha)
  vége(amíg)
  térítsd  $m + n - d$ 
Vége(algoritmus)

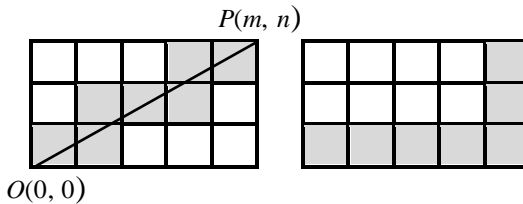
```

- A. Kiszámítja és téríti azoknak az 1 oldalhosszúságú négyzetecskéknek a számát, amelyeket átvág a téglalap egy átlója.
- B. Meghatározza d -ben a téglalap oldalainak legnagyobb közös osztóját és téríti az oldalak összegének és d -nek a különbségét.
- C. Ha $m = 8$ és $n = 12$, a térített érték 16.
- D. Ha $m = 6$ és $n = 11$, a térített érték 15.

Megoldás

A **B.** válasz biztos nem jó, hiszen „az oldalak összege” mindkét oldalt kétszer kellene tartalmazza. Könnyen belátjuk, hogy a **C.** helyes ($8 + 12 - 4 = 16$), de a **D.** nem ($6 + 11 - 1 = 16 \neq 15$). A legérdekesebb az **A.** válasz. A feladat nem kéri, hogy bebizonyítsuk az állítás helyességét, itt mégis leírjuk a gondolatmenetet, amelynek alapján eldönthető, hogy az **A.** válasz is helyes.

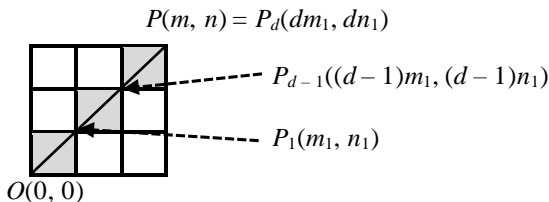
Előbb megvizsgáljuk azt az esetet, amikor $\text{lnko}(m, n) = 1$. Ekkor az átló nem tartalmaz egyetlen rácspontot sem a végpontokon kívül (és így könnyű megszámlolni az átvágott kis négyzeteket).



Valóban, ha a téglalapot egy olyan koordináta-rendszerbe helyezzük, amelyben $O(0, 0)$ a téglalap bal alsó sarka és $P(m, n)$ a jobb felső sarka, akkor az OP egyenes egyenlete $y = \frac{n}{m} * x$. Mivel az $\frac{n}{m}$ tört irreducibilis, ezért az $\frac{n}{m} * x$ egyetlen $x \in \{1, 2, \dots, m-1\}$ értékre sem lesz egész szám, ami pontosan azt jelenti, hogy nincs több rácspont az OP szakaszon a végpontokon kívül (és ez hasonlóan igazolható a másik átlóra is).

A bal alsó, O sarokkal rendelkező kis négyzetből indulunk a jobb felső, P sarokkal rendelkező kis négyzet felé úgy, hogy minden lépésben vagy felfele, vagy jobbra lépünk. Ahhoz, hogy megszámoljuk az OP átló által átvágott kis négyzeteket, meg kell számolnunk, hogy hány kis négyzetet érintünk az út során. Ez a szám egyenlő $m + n - 1$ -gyel.

Általános esetben legyen $\text{lnko}(m, n) = d$, tehát létezik $m_1, n_1 \in \mathbf{N}$ úgy, hogy $m = m_1 * d$ és $n = n_1 * d$, valamint $\text{lnko}(m_1, n_1) = 1$. Ez viszont azt jelenti, hogy az $m \times n$ -es téglalap felbomlik kis $m_1 \times n_1$ -es téglalapokra, az alábbi ábra szerint.



Könnyen észrevehető, hogy

- minden sorban és minden oszlopban d darab $m_1 \times n_1$ -es téglalap van;

- a főátló mentén összesen d darab kis téglalap helyezkedik el, és ezeknek a megfelelő csúcsai is a főátlón vannak (ez kijön a következő hasonlóságból: $\frac{m}{n} = \frac{d \cdot m_1}{d \cdot n_1} = \frac{m_1}{n_1}$, tehát a $P_1(m_1, n_1)$ pont is a nagy téglalap főátlóján van, hasonlóan a $P_2(2 \cdot m_1, 2 \cdot n_1)$ is stb.)

A fentiek alapján az általános esetben az OP főátló annyi kis négyzetet vág ketté, mint amennyit a főátlón elhelyezkedő kisebb, $m_1 \times n_1$ -es téglalapokban kettévág összesen, vagyis pontosan $d \cdot (m_1 + n_1 - 1) = m + n - d$ darabot.

A.10. Dominók elhelyezése az átlóra (6 pont)

Legyen egy téglalap alakú tábla, amely föl van osztva $n \times m$ cellára (n – a sorok száma és $2 \leq n \leq 100$, m – az oszlopok száma és $2 \leq m \leq 100$, ahol n és m természetes számok). Két játékos, A és B, felváltva lépéseket hajtanak végre: minden lépésnél a soron levő játékos megjelöl egyetlen cellát, amely átlósan szomszédos az előző lépésben, a másik játékos által megjelölt cellával és amely még nincs megjelölve. Az a játékos, akinek nincs hova lépnie, veszít. Az A játékos lép először, és megjelöl egy cellát a táblán. Határozzátok meg, milyen feltételek mellett van A-nak biztos nyerési stratégiája, (vagyis nyerni fog, B lépéseitől függetlenül) és mi lehet A első lépése ahhoz, hogy nyerjen.

a	b	c	d	e

Példa: a) eredeti állapot ($n = 5$ és $m = 4$), b) az első lépés utáni helyzet (lépett A), c) a második lépés után (lépett B), d) a harmadik lépés után (lépett A), e) a negyedik lépés után (lépett B)

A. feltétel: m páratlan szám;

az A játékos először a fenti első sorban (1-es sor) és egy páratlan sorszámu oszlopban levő cellára lép.

B. feltétel: n páratlan szám;

az A játékos először egy páros sorszámu sorban és a tábla bal első oszlopában (1-es oszlop) levő cellára lép.

C. feltétel: mindkét szám (n és m) páros szám;

az A játékos először a tábla bal felső sarkában (1-es sor és 1-es oszlop) levő cellára lép.

D. feltétel: n és m közül legalább az egyik páratlan szám;

az A játékos először a tábla bal felső sarkában (1-es sor és 1-es oszlop) levő cellára lép.

Megoldás

Könnyű belátni, hogy legalább egy számnak az m és n közül páratlannak kell lennie, ahhoz, hogy az „ellenfél” leszorítható legyen a tábláról egy adott pillanatban. Így a **C.** választ helytelennek minősítjük. Mivel a lépések átlósan szomszédos cellára történnek, az **A.** válasz biztos helyes. Ahhoz, hogy vizsgálhassuk, mi történik a **D.** esetben, feltételezzük, hogy n páratlan és m páros. Ha B átlósan lefele lép, egy adott pillanatban a szélre ér, de A léphet az előzőtől átlósan lefele. De ez fordítva is lehetséges. Ugyanígy ellenőrizhetnénk a többi lépést is, de egyszerű számolással is eljuthatunk a következtetésre, hogy **A.** és **D.** helyes válaszok.

B rész (30 pont)**Kalitkák**

Az állatkertben a papagájok 1-től n -ig számozott kalitkákban élnek ($1 \leq n \leq 10\,000$). Egy adott pillanatban egy játékos majom kinyit minden kalitkát. Megijed a következményektől, visszatér az első kalitkához és bezár minden második kalitkát (így bezárja a 2, 4, 6, ... sorszámúakat). A majomnak megtetszik ez a játék. Ezért újra elindul az elejéről és meglátogat minden harmadik kalitkát (vagyis a 3, 6, 9, ... sorszámúakat) és bezárja a kalitkát, ha az nyitva van, illetve kinyitja, ha azt zárva találja. A negyedik bejáráskor meglátogat minden negyedik kalitkát, és hasonlóan jár el (megváltoztatva a meglátogatott kalitka állapotát). A majom megismétli a játékot, míg az utolsó bejáráskor (az n . bejárás) bezárja az n . kalitkát, ha ez nyitva van vagy kinyitja, ha zárva van.

Követelmények:

- B.1.** Hány kalitka marad nyitva az utolsó bejárás után, ha $n = 10$? (2 pont)
- B.2.** Mely sorszámú kalitkák maradnak nyitva az utolsó bejárás után, ha $n = 10$? (2 pont)
- B.3.** Összesen hányszor látogatja meg a majom a k sorszámú kalitkát ($1 \leq k \leq n$) az n bejárás során? Indokoljátok meg a választ. (4 pont)
- B.4.** Mi annak szükséges és elégséges feltétele, hogy a k sorszámú kalitka ($1 \leq k \leq n$) nyitva maradjon az n kalitka utolsó bejárása után? Indokoljátok meg a választ. (4 pont)
- B.5.** Hány kalitka marad nyitva az n kalitka utolsó bejárása után? Indokoljátok meg a választ. (4 pont)

B.6. Írjatok algoritmust, amely kiszámítja az utolsó bejárás után *nyitva maradt kalitkák számát* (*nyitvaSz*). Az algoritmus bemeneti paramétere a kalitkák n ($1 \leq n \leq 10\,000$) száma, kimeneti paramétere a *nyitvaSz* szám. (14 pont)

1. Példa: ha $n = 5$, akkor *nyitvaSz* = 2 (nyitva marad az 1-es és a 4-es sorszámú kalitka).

2. Példa: ha $n = 12$, akkor *nyitvaSz* = 3.

Megoldás

A majom egy bejárás alkalmával, akkor és csakis akkor látogat meg egy kalitkát, ha a kalitka sorszáma a bejárás sorszámának többszöröse. Következik, hogy egy adott kalitka meglátogatásának darabszáma egyenlő a kalitka sorszáma különböző osztóinak darabszámával. A kalitka akkor és csakis akkor marad nyitva, ha a sorszámának *páratlan* darabszámú osztója van.

Lássuk, mi történik, ha $n = 5$. Jelöljük a *bejárásSorszáma* változóval a kalitkasor aktuális bejárásának sorszámát. Ha a kalitkák száma n , a *bejárásSorszáma* az $\{1, 2, \dots, n\}$ halmazból kap értéket.

Kövessük a példát:

1. Az első bejáráskor (*bejárásSorszáma* = 1) a majom kinyitja az 1, 2, 3, 4, 5 sorszámú kalitkákat. Tehát most nyitva vannak az **1 2 3 4 5** sorszámú kalitkák. Észrevevessük, hogy az 1, 2, 3, 4, 5 számoknak *egyetlen* osztójuk van az $\{1, \dots, \textit{bejárásSorszáma}\}$ halmazban, ami most $\{1\}$. Így az egyetlen osztó az 1.
2. A második bejárás során (*bejárásSorszáma* = 2) a majom bezár minden második kalitkát, vagyis a 2 és a 4 sorszámúakat. Nyitva maradnak az **1 3 5** sorszámúak, ahol mindegyiknek *egyetlen* osztója van az $\{1, \dots, \textit{bejárásSorszáma}\} = \{1, 2\}$ halmazban és pedig az 1, miközben a 2-es és a 4-es sorszámúak zárva vannak. Ennek a két számnak egyenként *két* osztója van az $\{1, 2\}$ halmazban, és pedig az 1 és a 2.
3. A harmadik bejárás során (*bejárásSorszáma* = 3) a majom kinyit minden harmadik kalitkát, ha ez zárva van és bezár minden harmadik kalitkát, ha ez nyitva van. Így bezárja a 3. kalitkát, miközben nyitva maradnak: **1 és 5**. Az 1-nek és az 5-nek *egyetlen* osztója van az $\{1, \dots, \textit{bejárásSorszáma}\} = \{1, 2, 3\}$ halmazban, miközben a 2, 3, 4 sorszámú kalitkák zárva vannak. Észrevevessük, hogy ezeknek a számoknak egyenként *két* osztója van az $\{1, 2, 3\}$ halmazban: 2 osztói az 1 és a 2, a 3 osztói az 1 és a 3, a 4 osztói pedig az 1 és a 2).
4. A negyedik bejárás során (*bejárásSorszáma* = 4) a majom kinyitja a 4-es sorszámú kalitkát (zárva volt). Most nyitva vannak az **1, 4 és 5** sorszámú kalitkák, ahol 1-nek egy osztója van az $\{1, 2, 3, 4\}$ halmazban, 4-nek három osztója van

(1, 2 és 4), 5-nek pedig egy osztója van (*az osztók száma páratlan*). A 2-es és 3-as kalitkák zárva vannak (2-nek és 3-nak két-két osztója van a megfelelő halmazban).

5. Az ötödik bejárás (*bejárásSorszáma* = 5) végén a majom bezárja az ötödik kalitkát. Nyitva vannak az **1** és a **4** (*páratlan* számú osztók vannak az {1, 2, 3, 4, 5} halmazban) és zárva vannak a 2-es, 3-as és 5-ös kalitkák (ahol 2-nek, 3-nak és 5-nek *két-két* osztója van).

Állítás

Egy természetes számnak akkor és csakis akkor van páratlan számú osztója, ha *négyzetszám*.

Bizonyítás

Ha d az m természetes szám osztója, akkor m/d is osztója m -nek, mivel:

$$d * (m/d) = m \quad (1)$$

Következik, hogy egy természetes szám osztói párba állíthatók, vagyis úgy gondolhatnánk, hogy bármely természetes számnak páros darabszámú osztója van. Másfelől, ha van olyan pár, amelyben a két szám egyenlő, következik, hogy az m szám négyzetszám, mivel, ha behelyettesítjük a d -t (m/d -vel az (1)-es egyenletben, kapjuk, hogy $d * d = m$. Ezekben az esetekben, csökkentjük a páros számú osztók darabszámát, amiből következik, hogy az osztók száma páratlan.

Ha $d \neq m/d$, marad a páros darabszámú osztószám (a nem négyzetszámok esetében).

Ilyen körülmények között a feladat megoldása visszavezethető azoknak a kalitkáknak a megszámlálására, amelyeknek sorszáma négyzetszám. Egy n természetes szám négyzetszám, ha $\sqrt{n} * \sqrt{n} = n$. Másfelől az n -nél kisebb (vagy egyenlő) négyzetszámok darabszáma egyenlő $\lfloor \sqrt{n} \rfloor$ -nel. Ez az érték meghatározható a *bináris keresés módszerével*, amikor keressük a $\lfloor \sqrt{n} \rfloor$ -et az 1, 2, ..., $n/2$ rendezett számsorozatban.

A kalitkákat „zárhatjuk/nyithatjuk” a következő táblázatban:

<i>Bejárás-sorszáma</i>	<i>Kalitkák</i>					<i>Osztók halmaza</i>
	1	2	3	4	5	
kiindulópont	zárva	zárva	zárva	zárva	zárva	
1	nyitva	nyitva	nyitva	nyitva	nyitva	1
2	nyitva	zárva	nyitva	zárva	nyitva	1, 2
3	nyitva	zárva	zárva	zárva	nyitva	1, 2, 3
4	nyitva	zárva	zárva	nyitva	nyitva	1, 2, 3, 4
5	nyitva	zárva	zárva	nyitva	zárva	1, 2, 3, 4, 5

Lássuk a válaszokat a feladatban feltett kérdésekre:

- B.1.** Ha $n = 10$, nyitva marad 3 kalitka.
- B.2.** Ha $n = 10$, nyitva maradnak az 1, 4, 9 sorszámú kalitkák.
- B.3.** A k sorszámú kalitkát összesen annyszor látogatja meg a majom az n bejárás során, ahány osztója van k -nak az $\{1, 2, \dots, n\}$ halmazban.
- B.4.** Egy k sorszámú kalitka akkor és csakis akkor marad nyitva az n kalitka utolsó bejárása után, ha:
- k -nak *páratlan darabszámú osztója* van az $\{1, 2, \dots, n\}$ halmazban vagy – másképp kifejezve:
 - k *négyzetszám*, mivel minden olyan szám, amelynek páratlan darabszámú osztója van négyzetszám.
- B.5.** A nyitva maradt kalitkák darabszáma egyenlő $\lfloor \sqrt{n} \rfloor$ -nel, mivel minden olyan szám, amelynek páratlan darabszámú osztója van, négyzetszám. Ugyanakkor az n -nél kisebb négyzetszámok darabszáma: $\lfloor \sqrt{n} \rfloor$.
- B.6.** Az algoritmus több elképzelés alapján is megtervezhető. A következőkben bemutatjuk az egyik legkézenfekvőbbet (**V1**), majd egy hatékonyabbat (**V2**).
- V1:** a nyitva maradt kalitkák számát meghatározzuk az n -nél kisebb négyzetszámok megszámlálása által (egyetlen ismétlődő struktúrával);
- V2:** a nyitva maradt kalitkák számát a $\lfloor \sqrt{n} \rfloor$ alapján határozzuk meg; az optimális algoritmus ezt az értéket a bináris keresés algoritmusával számítja ki:

```
int kalitkakV1(int n){
    int nyitvaSz = 0;
    for (int i = 1; i * i <= n; i++){
        nyitvaSz++;
    }
    return nyitvaSz;
}
```

```
int kalitkakV2(int n){
    int bal = 1;
    int jobb = n / 2;
    while (bal < jobb){
        int k = ((bal + jobb) / 2);
        if (k * k >= n)
            jobb = k;
        else
            bal = k + 1;
    }
    if (bal * bal <= n)
        return bal;
    else
        return bal - 1;
}
```

V3: ha a fenti ötletek közül egyik sem „ugrik be”, szimulálhatjuk a majom tevékenységét: bejárunk minden kalitkát n -szer, kinyitjuk/bezárjuk a kalitkát a bejárás során, majd a nyitva maradt kalitkákat megszámláljuk.

```
int kalitkakV3(int n){
    const int MAX = 10001;
    bool ajtok[MAX];
    for (int i = 1; i <= n; i++){
        ajtok[i] = true;
    }
    for (int k = 2; k <= n; k++){
        for (int i = k; i <= n; i = i + k){
            if (ajtok[i])
                ajtok[i] = false;
            else
                ajtok[i] = true;
        }
    }
    int nyitvaSz = 0;
    for (int i = 1; i <= n; i++){
        if (ajtok[i])
            nyitvaSz++;
    }
    return nyitvaSz;
}
```

3.3. Felvételi verseny – 2019. július 21.

A Rész (60 pont)

A.1. Mit ír ki? (6 pont)

Adott az alábbi program. Állapítsátok meg, mit ír ki a program a végrehajtás eredményeként.

A. P(a, b) = 253;
a = 11; b = 11;

C. P(a, b) = 22;
a = 11; b = 11;

B. P(a, b) = 132;
a = 11; b = 121;

D. P(a, b) = 253;
a = 11; b = 121;

C++	C
<pre>#include<iostream> using namespace std; int P(int x, int &y){ y = y * x; x = x + y; return x + y; } int main(){ int a = 11; int b = a; cout << "P(a, b) = " << P(a, b); cout << endl << "; a = " << a; cout << "; b = " << b << endl; return 0; }</pre>	<pre>#include<stdio.h> int P(int x, int *y){ *y = (*y) * x; x = x + (*y); return x + (*y); } int main(){ int a = 11; int b = a; printf("P(a, b) = %d", P(a, &b)); printf("; \na = %d", a); printf("; b = %d\n", b); return 0; }</pre>
Pascal	
<pre>function P(x : Integer; var y : Integer) : Integer; begin y := y * x; x := x + y; P := x + y; end; var a, b : Integer; Begin a := 11; b := a; Write('P(a, b) = ', P(a, b)) WriteLn('; '); Write('a = ', a); WriteLn('b = ', b); End.</pre>	

Megoldás

Hasonlóan járunk el, mint a Mat–Infó verseny **A.1.** feladatának megoldásakor.

A program itt is rögzíti a bemenet értékeit: $a = 11$, $b = a$, tehát $b = 11$. Fontos észrevennünk, hogy x érték szerint átadott, míg y cím (illetve *referencia*) szerint átadott paraméter. Következik, hogy x értéke a függvényen belül 132 lesz, de a hívás helyén megőrzi értékét (11). Az y paraméter értéke a függvényen belül 121 lesz, ez a hívás helyén is érvényes lesz, a függvény által térített érték pedig $x + y = 132 + 121 = 253$. Tehát a helyes válasz: **D**.

A.2. Kiértékelés (6 pont)

Adott az n elemű ($1 \leq n \leq 10\,000$), 30 000-nél kisebb természetes számokat tároló v sorozat. Állapítsátok meg, hogy a következő programrészletek közül melyiket kellene a „...” helyére írni, ahhoz, hogy a $\text{feldolgoz}(v, n, er, m)$ algoritmus ismétlődő struktúrájának végrehajtása után az er sorozat, a v sorozat azon elemeit tárolja, amelyek 5-nek azon többszörösei, amelyek páros indexű helyeken találhatók. Az er sorozat hosszát az m változó tárolja.

```

Algoritmus feldolgoz(v, n, er, m)
  m ← 0
  Minden i = 1, n végezd el
    ...
  vége(minden)
Vége(algoritmus)

```

A.	Ha $i \bmod 2 = 0$ és $v[i] \bmod 5 = 0$ akkor $m \leftarrow m + 1$ $er[m] \leftarrow v[i]$ vége(ha)	B.	Ha $i \bmod 2 = 0$ akkor Ha $v[i] \bmod 10 = 5$ akkor $m \leftarrow m + 1$ $er[m] \leftarrow v[i]$ vége(ha) Ha $v[i] \bmod 10 = 0$ akkor $m \leftarrow m + 1$; $er[m] \leftarrow v[i]$ vége(ha) vége(ha)
C.	Ha $v[i] \bmod 10 = 0$ akkor Ha $i \bmod 2 = 0$ akkor $m \leftarrow m + 1$ $er[m] \leftarrow v[i]$ különben Ha $v[i] \bmod 10 = 5$ akkor $m \leftarrow m + 1$; $er[m] \leftarrow v[i]$ vége(ha) vége(ha) vége(ha)	D.	Ha $i \bmod 2 = 0$ és $v[i] \bmod 10 = 5$ és $v[i] \bmod 10 = 0$ akkor $m \leftarrow m + 1$ $er[m] \leftarrow v[i]$ vége(ha)

Megoldás

A **Ha** utasítások feltételeit (a logikai kifejezéseket) fogjuk vizsgálni. Az **A.** változatban az $(i \bmod 2 = 0 \text{ és } v[i] \bmod 5 = 0)$ kifejezés akkor térít *igaz*-at, ha az aktuális elem indexe páros és az elem 5-nek többszöröse. Tehát **A.** helyes.

A **B.** algoritmusrészlet csak akkor foglalkozik egy adott elemmel, ha az indexe páros, majd két egymás után írt **Ha** utasításban foglalkozik az elem értékével. Mindkét logikai kifejezés helyes és szükséges, hiszen 5 többszöröseinek utolsó számjegye vagy 0 vagy 5. Így **B.** is helyes.

A **C.** algoritmus csak 10 többszöröseit dolgozza fel, tehát hibás.

A **D.** szintén hibás, mivel a logikai kifejezés második és harmadik részkifejezése nem lehet *igaz* egyszerre $(v[i] \bmod 10 = 5 \text{ és } v[i] \bmod 10 = 0)$.

A.3. Mely értékek szükségesek? (6 pont)

Legyen a $\text{számol}(n, v)$ algoritmus, ahol v egy n egész számot tároló sorozat (n – természetes szám, $1 \leq n \leq 1\,000$).

Állapítsátok meg, hogy a paraméterek mely értékeire térít az algoritmus 0-t.

```

Algoritmus számol(n, v)
  a ← 0
  b ← 0
  Minden i = 1, n végezd el
    a ← a + v[i]
  vége(minden)
  Minden i = 1, n végezd el
    a ← a - v[i]
    Ha a = b akkor
      térítsd v[i]
    vége(ha)
    b ← b + v[i]
  vége(minden)
  térítsd -1
Vége(algoritmus)

```

- A.** $n = 5, v = (4, 5, 7, 3, 6)$
B. $n = 7, v = (-3, 1, 2, 0, 5, -2, -3)$
C. $n = 4, v = (-2, 2, 5, -5)$
D. $n = 8, v = (1, -7, 3, 0, -2, 1, -2, 0)$

Megoldás

Az algoritmus leáll, és téríti az aktuális elem értékét, ha a két lokális változó (a és b) értéke egyenlő. Mivel csak a **B.** és **D.** sorozatokban található 0 értékű elem, ezért csak ezeket fogjuk vizsgálni, hiszen az algoritmus egy sorozatelemet térít, amelynek a követelmény szerint 0-nak kell lennie. A kezdőértékadások után ($a = 0$ és $b = 0$), az algoritmus előbb összeadja a sorozat elemeit a -ban. A **B.**-ben megadott sorozat esetében ez az összeg 0. A sorozat második bejárásakor, ebből az összegből (vagyis a -ból) rendre kivonjuk az aktuális elem értékét és ugyanezt az elemet hozzáadjuk b -hez. Ezeket a műveleteket követjük a következő táblázatban.

<i>Elem sorszáma</i>	<i>a</i>	<i>b</i>	<i>v_i</i>	<i>a-t b-vel a kivonás után hasonlítja össze az algoritmus</i>
	0	0		
1	$0 - (-3) = 3$	$0 + (-3) = -3$	-3	$a = 3, b = 0$
2	$3 - 1 = 2$	$-3 + 1 = -2$	1	$a = 2, b = -3$
3	$2 - 2 = 0$	$-2 + 2 = 0$	2	$a = 0, b = -2$
4	$0 - 0 = 0$		0	$a = b = 0$, és $v_i = 0$

Következik a táblázatból, hogy a **B.** sorozat esetében, az algoritmus 0-t térít.

<i>Elem sorszáma</i>	<i>a</i>	<i>b</i>	<i>v_i</i>	<i>a-t b-vel a kivonás után hasonlítja össze az algoritmus</i>
	-6	0		
1	$-6 - 1 = -7$	$0 + 1 = 1$	1	$a = 3, b = 0$
2	$-7 - (-7) = 0$	$1 + (-7) = -6$	-7	$a = 2, b = -3$
3	$0 - 3 = -3$	$-6 + 3 = -3$	3	$a = 0, b = -2$
4	$-3 - 0 = -3$		0	$a = b = 0$, és $v_i = 0$

Tehát, a **D.** pontban megadott sorozat esetében szintén 0-t térít az algoritmus.

A.4. Vajon mit csinál? (6 pont)

Legyen a $\text{találdKi}(n)$ algoritmus, ahol n természetes szám ($1 \leq n \leq 10000$).
Állapítsátok meg a $\text{találdKi}(n)$ algoritmus hatását.

```

Algoritmus találdKi(n)
  f ← 0
  p ← -1
  Minden c = 0, 9 végezd el
    x ← n
    k ← 0
    Amíg x > 0 végezd el
      Ha x MOD 10 = c akkor
        k ← k + 1
      vége(ha)
    x ← x DIV 10
  vége(amíg)
  Ha k > f akkor
    p ← c
    f ← k
  vége(ha)
  vége(minden)
  térítsd p
Vége(algoritmus)

```

- Kiszámítja és visszatéríti a 10-es számrendszerben megadott n szám számjegyeinek darabszámát.
- Kiszámítja és visszatéríti a 10-es számrendszerben megadott n szám legnagyobb számjegyének darabszámát.
- Kiszámítja és visszatéríti a 10-es számrendszerben megadott n szám egyik olyan számjegyét, amelynek előfordulási száma maximális.
- Kiszámítja és visszatéríti a 10-es számrendszerben megadott n szám, c -vel egyenlő számjegyeinek darabszámát.

Megoldás

Az algoritmus generálja c -ben a számjegyeket 0-tól 9-ig, majd megszámlolja, minden számjegy esetében, hogy ezek hányszor fordulnak elő az n számban. Az előfordulások k számát felhasználja ahhoz, hogy aktualizálja f -ben az előfordulások maximumát és p -ben az ehhez az előforduláshoz tartozó számjegyet. Tehát a helyes válasz **C**. Itt fölösleges a többi válasszal foglalkozni, a tartalmuk nem lehet helyes, tekintve, hogy a kiválasztott helyes választól eltérő hatást jelölnek meg.

A.5. Sajátos sorozat generálása (6 pont)

Ismert, hogy egy k elemű x sorozat ($x = (x_1, x_2, x_3, \dots, x_k)$) p hosszúságú tömbszakasa az x sorozat p darab eleméből áll, amelyek egymás utáni helyeket foglalnak el. Például, $y = (x_3, x_4, x_5, x_6)$ egy $p = 4$ hosszúságú tömbszakasz.

Legyen az $M = \{1, 2, \dots, n\}$ számjegyeket tároló halmaz, és az M halmaz $n!$ darab permutációja (n – természetes szám, $n \leq 9$). Létre lehet hozni azt a legrövidebb ss számjegysorozatot, amely M számjegyeit tárolja és az M halmaz összes $n!$ permutációja megtalálható az ss -ben, n hosszúságú tömbszakaszok formájában.

Például, ha $n = 3$, a permutációk száma 6, az ss sorozatot 9 számjegy alkotja és lehetne, például $ss = (1, 2, 3, 1, 2, 1, 3, 2, 1)$. A $perm_1 = (1, 2, 3)$ permutációnak felhasználjuk az utolsó két számjegyet és hozzáfűzünk egy harmadik számjegyet, ahhoz, hogy egy másik permutációt kapjunk, vagyis a $perm_2 = (2, 3, 1)$ permutációt; majd a $perm_2$ két utolsó számjegye után helyezhetjük a 2-es számjegyet és megkapjuk a $perm_3 = (3, 1, 2)$ permutációt. Ha az $(1, 2)$ után íránk a 3-as számjegyet, megkapnánk a $perm_1$ permutációt, amely már létezik az eddig generált számjegysorozatban. Ekkor $perm_3$ -nak csak az utolsó számjegyet használjuk fel és egy olyan permutációt keresünk, amely a 2-es számjeggyel kezdődik és még nincs benne a sorozatban stb. Így, a létrehozott sorozatban egyetlen olyan három számjegyből álló tömbszakasz található, amely nem permutáció: $(1, 2, 1)$; a többi tömbszakasz, (vagyis $(2, 1, 3)$, $(1, 3, 2)$ és $(3, 2, 1)$) helyesek.

Állapítsátok meg, hogy az $M = \{1, 2, 3, 4\}$ halmaz esetében legkevesebb hány számjegy típusú eleme lesz az ss sorozatnak.

$perm_1$									
1	2	3	1	2	1	3	2	1	
	$perm_2$								

A. 55

B. 16

C. 33

D. 37

Megoldás

Generáljuk a kért *ss* sorozatot. Ahhoz, hogy ne veszítsünk el egyetlen permutációt sem, kénytelenek vagyunk leírni ezeket. Amint egy permutációt elhelyeztünk *ss*-be, kihúzzuk, hogy ne használjuk fel még egyszer. A feladat szövege tartalmazza a számjegysorozat létrehozatalának stratégiáját, tehát most csak arra kell vigyáznunk, hogy figyelmesen dolgozzunk és ne tévedjünk.

Permutáció sorszáma	Perm.	Sorsz.	Perm.	Sorsz.	Perm.	Sorsz.	Perm.
1.	1234	7.	2134	13.	3124	19.	4123
2.	1243	8.	2143	14.	3142	20.	4132
3.	1324	9.	2314	15.	3214	21.	4213
4.	1342	10.	2341	16.	3241	22.	4231
5.	1423	11.	2413	17.	3412	23.	4312
6.	1432	12.	2431	18.	3421	24.	4321

Elkezdjük az „építkezést”. Minden lépésben keresünk egy olyan permutációt, amely az aktuális permutációnak utolsó három számjegyével kezdődik. Ez csak a 19. permutáció beszúrásáig sikerül. Ekkor egy olyan permutációt keresünk, amely az eddig generált számjegysorozat utolsó két számjegyével kezdődik. Ez a 9. Folytatjuk a számjegysorozat építését az eddig alkalmazott stratégia alapján.

Sorsz.	1.	10.	17.	19.	9.	14.	5.	22.
Perm.	1234	2341	3412	4123	2314	3142	1423	4231
Sorsz.	13.	2.	12.	23.	7.	4.	18.	21.
Perm.	3124	1243	2431	4312	2134	1342	3421	4213
Sorsz.	3.	16.	11.	20.	15.	8.	6.	24.
Perm.	1324	3241	2413	4132	3214	2143	1432	4321

Ha most megszámloljuk a táblázatban fehérén maradt számjegyeket (vagyis azokat, amelyekre nem csúszott rá más számjegy az ideragasztott permutációból) megkapjuk a számjegysorozat hosszát. Ha generáltuk a számjegysorozatot, – természetesen – megszámlolhatjuk az elemeit. A számjegyek száma 33, tehát a **C.** válasz a helyes.

A.6.Számjegysorozat (6 pont)

A számJegyek(*n*, *d*) algoritmus (*n* és *d* természetes számok, $10 \leq n \leq 100\,000$, $1 \leq d \leq 9$), meghatározza és visszatéríti azt a legkisebb természetes számot, amelynek *d*-nél kisebb vagy *d*-vel egyenlő, nem nulla számjegyei vannak és amely számjegyeknek a szorzata egyenlő *n*-nel. Például, ha *n* = 108 és *d* = 9, az algoritmus 269-et térít vissza. Ha ilyen szám nem létezik, az algoritmus -1-et térít.

Állapítsátok meg, hányszor hívja meg önmagát a számJegyek(*n*, *d*) algoritmus az alábbi programrészlet végrehajtásának következtében:


```

Algoritmus számJegyek(n, d)
  Ha d = 1 akkor
    Ha n = 1 akkor
      térítsd 0
    különben
      térítsd -1
    vége(ha)
  különben
    Ha n MOD d = 0 akkor
      érték ← számJegyek(n DIV d, d)
    Ha érték < 0 akkor
      térítsd -1
    különben
      térítsd érték * 10 + d
    vége(ha)
  különben
    térítsd számJegyek(n, d - 1)
  vége(ha)
vége(ha)
Vége(algoritmus)

```

```

beOlvas n
érték ← számJegyek(n, 9)

```

- A. Ha $n = 108$, az algoritmus 11-szer hívja meg önmagát.
- B. Ha $n = 109$, az algoritmus 8-szor hívja meg önmagát.
- C. Ha $n = 13$, az algoritmus egyszer sem hívja meg önmagát.
- D. Ha $n = 100$, az algoritmus 10-szer hívja meg önmagát.

Megoldás

A rekurzív hívások számát megszámlálhatjuk, ha hívássorozatot készítünk vagy, ha követjük a paraméterek értékeit a végrehajtási veremben:

számjegyek(108, 9) \Rightarrow (0. hívás):
 számjegyek(12, 9) \Rightarrow (1. hívás):
 számjegyek(12, 8) \Rightarrow (2. hívás):
 számjegyek(12, 7) \Rightarrow (3. hívás):
 számjegyek(12, 6) \Rightarrow (4. hívás):
 számjegyek(2, 6) \Rightarrow (5. hívás):
 számjegyek(2, 5) \Rightarrow (6. hívás):
 számjegyek(2, 4) \Rightarrow (7. hívás):
 számjegyek(2, 3) \Rightarrow (8. hívás):
 számjegyek(2, 2) \Rightarrow (9. hívás):
 számjegyek(1, 2) \Rightarrow (10. hívás):
 számjegyek(1, 1) \Rightarrow (11. hívás)

		Hívás sorszáma
1	d	11
1	n	
2	d	10
1	n	
2	d	9
2	n	
3	d	8
2	n	
4	d	7
2	n	
5	d	6
2	n	
6	d	5
2	n	
6	d	4
12	n	
7	d	3
12	n	
8	d	2
12	n	
9	d	1
12	n	
9	d	0 (első hívás)
108	n	

A hívások száma 11, tehát **A.** helyes.

A **B.** válasz esetében, valóban 8 rekurzív hívás után áll le a hívások sorozata. Az n paraméter értéke nem változik, miközben d csökken 9-től 1-ig, amikor -1-et térít, mivel nem talál megoldást. Így a **B.** válasz is helyes.

A **C.** válasz nem helyes, mivel, ha $n = 13$, az algoritmusnak „nincs oka”, hogy egyszer se hívja meg önmagát. Ha készítünk hívássorozatot, látni fogjuk, hogy nem talál megoldást, de a **B.** esethez hasonlóan szintén 8-szor hívja meg önmagát, miközben d csökken 9-től 1-ig, amikor leáll, tehát **C.** nem helyes.

A **D.** válasz esetében a rekurzív hívások száma 8, tehát **D.** sem helyes.

A.7.Sorozat feldolgozása (6 pont)

Adott a $\text{feldolgoz}(n, x)$ algoritmus, ahol x egy $n - 1$ elemű, különböző természetes számokat tároló sorozat, amely számok az $\{1, 2, \dots, n\}$ halmazhoz tartoznak. Állapítsátok meg az algoritmus által visszatérített érték jelentését.

```
Algoritmus feldolgoz(n, x)
  s ← 0
  Minden i = 1, n - 1 végezd el
    s ← s + x[i]
  vége(minden)
  térítsd n * (n + 1) / 2 - s
Vége(algoritmus)
```

- A. Az algoritmus az első n nem nulla természetes szám összegének és az x sorozat elemei összegének különbségét téríti vissza.
- B. Az algoritmus az első n nem nulla természetes szám összegének és az x sorozat elemei összegének (kivéve az utolsó elemet) különbségét téríti vissza.
- C. Az algoritmus annak az $\{1, 2, \dots, n\}$ halmazhoz tartozó természetes számnak az értékét téríti, amely nem szerepel az x sorozatban.
- D. Az algoritmus 0-t térít vissza.

Megoldás

Az **A.** kijelentés egyszerű és könnyen ellenőrizhető, hogy helyes.

A **B.** esetében azonnal észrevevessük, hogy az algoritmus nem a feltételezett funkciót teljesíti. Egyrészt az x sorozat elemeinek darabszáma $n - 1$, így a zárójelben található kitélből („kivéve az utolsó elemet”) az következik, hogy csak $n - 2$ elem összegét számolja ki az algoritmus. De ez nem igaz, hiszen a **Minden** ciklus i ciklusváltozójának értékei: 1, 2, ..., $n - 1$, vagyis $n - 1$ elemet adunk össze (az x sorozat minden elemét).

A **C.** kijelentés meglepő, de miután észrevevessük, hogy az első n természetes szám összegéből $n - 1$ különböző természetes szám összegét vonjuk ki, amely

számok az $\{1, 2, \dots, n\}$ halmazhoz tartoznak, egyértelműen következik, hogy a sorozatból hiányzik ennek a halmaznak egy eleme. Az algoritmus pontosan ennek az értékét számolja ki. Például, ha $n = 4$, a halmaz $= \{1, 2, 3, 4\}$. Legyen $x = (1, 2, 4)$. A halmaz elemeinek összege 10, a sorozat elemeinek összege pedig $s = 7$. A kettő különbsége 3, amely pontosan az x -ből hiányzó elem értéke. Tehát, **C.** helyes.

A **D.** válasz szintén megtévesztő. Az algoritmus által térített érték csak akkor lehetne 0, ha az x sorozat elemeinek összege egyenlő lenne az első n természetes szám összegével. De ez lehetetlen, mivel az x sorozatnak csak $n - 1$ eleme van az $\{1, 2, \dots, n\}$ halmazból és ezek az elemek különbözők.

A.8. Varázslat (6 pont)

Egy számjegymágus olyan varázslatot végez, amelynek eredményeképpen egy x természetes szám ($100 < x < 1\,000\,000$, amelynek a 10-es számrendszerben van legkevesebb két 0-tól különböző számjegye) szétválik két pozitív természetes számra: a **bal** és **jobb** számokra, amelyek egymás után ragasztva megadják az x számot. Ugyanakkor a **bal** és **jobb** számok szorzata a lehető legnagyobb. Például, ha $x = 1\,092$, a varázslat szétválasztja a **bal** = 10 és **jobb** = 92 számokra.

Az alábbi algoritmusok közül melyik alkalmazza a varázslatot az x természetes számra, amelynek 10-es számrendszerben van legkevesebb két 0-tól különböző számjegye ($100 \leq x \leq 1\,000\,000$)? Az algoritmus meghatározza a z természetes számban ($0 \leq z \leq 1\,000\,000$) az x szám **jobb** részét. Az alábbi algoritmusok léteznek:

- $\text{hatvány}(b, p)$ – meghatározza a b^p értéket (b a p . hatványon), b, p – természetes számok ($1 \leq b \leq 20, 1 \leq p \leq 20$);
- $\text{szjSzám}(sz)$ – meghatározza az sz szám ($0 \leq sz \leq 1\,000\,000$) számjegyeinek darabszámát;

A.	<pre> Algoritmus varázslat(x, z) maxSzorzat \leftarrow -1 eredmény \leftarrow 0 Amíg $x > 0$ végezd el $z \leftarrow (x \bmod 10) * \text{hatvány}(10, \text{szjSzám}(z)) + z$ $x \leftarrow x \text{ DIV } 10$ Ha $x * z > \text{maxSzorzat}$ akkor maxSzorzat \leftarrow $x * z$ eredmény \leftarrow z vége(ha) vége(amíg) térítsd maxSzorzat Vége(algoritmus) </pre>
-----------	--

B.	<pre> Algoritmus varázslat(x, z) t ← 0 Ha x > 0 akkor y ← (x MOD 10) * hatvány(10, szjSzám(z)) + z t ← x DIV 10 Ha x * z < y * t akkor térítsd varázslat(y, t) különben térítsd t vége(ha) különben térítsd t vége(ha) Vége(algoritmus) </pre>
C.	<pre> Algoritmus varázslat(x, z) maxSzorzat ← -1 eredmény ← 0 Amíg x > 0 végezd el z ← (x MOD 10) * hatvány(10, szjSzám(z)) + z x ← x DIV 10 Ha x * z > maxSzorzat akkor maxSzorzat ← x * z eredmény ← z vége(ha) vége(amíg) térítsd eredmény Vége(algoritmus) </pre>
D.	<pre> Algoritmus varázslat(x, z) Ha x > 0 akkor y ← (x MOD 10) * hatvány(10, szjSzám(z)) + z t ← x DIV 10 Ha x * z < y * t akkor térítsd varázslat(y, t) különben térítsd z vége(ha) különben térítsd z vége(ha) Vége(algoritmus) </pre>

Megoldás

Az **A.** algoritmus inicializálja az *eredmény* változó értékét, de sehol nem használja fel. Ez az észrevétel arra ösztönöz, hogy keressünk a változatok között egy másikat, amely szintén iteratív és nagyvonalakban hasonlít az **A.** algoritmushoz, de nem tartalmazza az előbb említett hibát, vagy ehhez hasonlót. Ez a **C.** algoritmus, de más eltérést is tartalmaz. Nem a *maxSzorzat* változó értékét téríti, hanem

az **eredmény** változót. A két algoritmus közül egyértelmű, hogy az **A.** nem helyes, mivel a feladat nem a maximális szorzatot kéri, hanem az **x** szám **jobb** részét. A **C.** algoritmus a **z** változóban építi a szám **jobb** részét **x** számjegyeiből jobbról balra haladva, és minden lépésben aktualizálja, ha szükséges, a **maxSzorzat** értéket is. Ha ez megtörtént, megjegyzi az **eredmény** változóban azt a **z** értéket (a szám **jobb** részét), amely nagyobb **maxSzorzat** értéket eredményezett. Végül a **maxSzorzat** legnagyobb értékéhez „tartozó” **eredmény** változó értékét téríti. Tehát a **C.** algoritmus helyes.

A **B.** algoritmus két paramétere, hasonlóan a **C.** algoritmushoz **x** és **z**, ahol **x**-ben az **x** szám **bal** részét, **z**-ben a **jobb** részét szeretné felépíteni az algoritmus. Ez az algoritmus a **t** változó értékét téríti, vagy meghívja önmagát az **y** és **t** aktuális paraméterekkel. De **t** az **x** változónak a **bal** része, hiszen a 10-zel való osztások eredménye. Ebből következik, hogy a **varázslat(y, t)** rekurzív hívás helyett **varázslat(t, y)** lett volna a helyes, ezáltal **x** bal része **t**-től kapna értéket, jobb része **y**-től. A **térítsd t** utasítás is hibás, hiszen a feladat a szám **jobb** részét kéri, de **t** a **bal** része.

A továbbiakban összehasonlítjuk a **B.** rekurzív megoldást a **D.**, szintén rekurzív megoldással és azonnal látjuk, hogy ugyanaz a gond a rekurzív hívás aktuális paramétereinek sorrendjével: **varázslat(y, t)** helyett **varázslat(t, y)** lett volna a helyes sorrend. Tehát csak egy helyes megoldást találtunk, a **C**-t.

A.9. Egészítsétek ki (6 pont)

Adott az **n** elemű ($3 \leq n \leq 100$), növekvően rendezett **x** sorozat, amely 30 000-nél kisebb különböző természetes számokat tartalmaz. A **legközelebbi(x, bal, jobb, ér)** algoritmus meghatározza az **x** sorozat legnagyobb értékű elemének pozícióját, amely a **bal** és **jobb** pozíciók között helyezkedik el ($1 \leq \text{bal} < \text{jobb} \leq n$) és, amelynek az értéke kisebb, vagy egyenlő **ér**-rel. Ha nem létezik ilyen elem, a **legközelebbi(x, bal, jobb, ér)** algoritmus 0-t térít vissza.

A **modulusz(a)** algoritmus az **a** egész szám abszolút-értékét téríti vissza.

A **számol(n, x, adottSz)** algoritmus meghatározza azt az elemét az **x** sorozatnak, amely a legközelebb áll **adottSz**-hoz. Ha két elem azonosan közel van **adottSz** értékéhez, az algoritmus a nagyobb számot határozza meg.

Legyen **n** = 5, **x** = (5, 9, 11, 15, 99) és **adottSz** = 12. Állapítsátok meg melyik kifejezéssel helyettesíthető a „...” a **legközelebbi(x, bal, jobb, ér)** algoritmusban, ahhoz, hogy a **számol(n, x, adottSz)** algoritmus 11-et térítsen vissza.

```
Algoritmus legközelebbi(x, bal, jobb, ér)
  Ha ér > x[jobb] akkor
    térítsd jobb
  vége(ha)
  Ha ér < x[bal] akkor
    térítsd bal - 1
  vége(ha)
  közép ← (bal + jobb) DIV 2
  Ha ... akkor
    térítsd közép - 1
  különben
    Ha ér < x[közép] akkor
      térítsd legközelebbi(x, bal, közép - 1, ér)
    különben
      térítsd legközelebbi(x, közép + 1, jobb, ér)
    vége(ha)
  vége(ha)
Vége(algoritmus)
```

```
Algoritmus számol(n, x, adottSz)
  i ← legközelebbi(x, 1, n, adottSz)
  Ha i = 0 akkor
    térítsd x[i + 1]
  különben
    Ha modulusz(x[i] - adottSz) < modulusz(x[i + 1] - adottSz) akkor
      térítsd x[i]
    különben
      térítsd x[i + 1]
    vége(ha)
  vége(ha)
Vége(algoritmus)
```

- A. $x[\text{közép} - 1] \leq \text{ér}$ és $\text{ér} < x[\text{közép}]$
B. $x[\text{közép} - 1] \leq \text{ér}$ vagy $\text{ér} < x[\text{közép}]$
C. $x[\text{közép} - 1] < \text{ér}$ és $\text{ér} \leq x[\text{közép}]$
D. $x[\text{közép}] \leq \text{ér}$ és $\text{ér} < x[\text{közép} - 1]$

Megoldás

Az algoritmusban felismerjük a bináris keresést, azzal a különbséggel, hogy itt nem egy adott értékű elemet keresünk, hanem egy másikat, amelynek az értéke a legközelebb áll az adott számhoz. A `számol(n, x, adottSz)` algoritmus meghívja a `legközelebbi(x, bal, jobb, ér)` algoritmust, amely téríti az adott szám valamelyik szomszédos értékű elemének indexét. A továbbiakban, a térített indexnek megfelelő elem és az adott szám különbsége alapján eldönti, hogy melyik érték a legközelebbi, így ezzel a gonddal a `legközelebbi(x, bal, jobb, ér)` algoritmus nem foglalkozik. Következik, hogy azt az indexet térítjük, amelytől balra egy olyan elem található, amely kisebb, mint az adott szám, miközben jobbra egy nagyobb.

Ugyanakkor, a hiányzó feltételt tartalmazó utasítás fölöslegesnek tűnik, mivel sejtjük, hogy az algoritmus működése során, vagy valamelyik önmeghívás, vagy a két leállási feltétel közül valamelyik hajtódik majd végre. Megvizsgáljuk, mi történne, ha a **közép** kiszámítása utáni **Ha** (a felkínált feltételektől függetlenül) és ennek a **Ha**-nak az **akkor** ága hiányozna. Lássuk, hogyan hajtódik végre az algoritmus a fenti példa esetében ($n = 5$, $x = (5, 9, 11, 15, 99)$ és **adottSz** = 12):

<i>bal</i>	<i>jobb</i>	<i>közép</i>	<i>ér</i>	<i>Végrehajtott utasítás</i>	<i>Magyarázat</i>
1	5	3	12	Ha $ér > x[jobb]$ akkor	Mivel $12 < 99$ a Ha -nak vége
				Ha $ér < x[bal]$ akkor	Mivel $12 > 5$ a Ha -nak vége
				Ha $ér < x[közép]$ akkor	Mivel $12 > 11$, a különb ág következik (újra hívás)
4	5	4		Ha $ér > x[jobb]$ akkor	Mivel $12 < 99$ a Ha -nak vége
				Ha $ér < x[bal]$ akkor	Mivel $12 < 15$, téríti bal – 1-et vagyis 3-at ($x_3 = 11$), vége

Az egyetlen eset, amelynek megfelelően a tárgyalt **Ha** utasítást végrehajtja a program, a **B.** eset. Ekkor a térített érték 2, vagyis $x_2 = 9$. De mégsem ez lesz a végeredmény, erről gondoskodik a $szo(n, x, adottSz)$ alprogram.

Mivel beláttuk, hogy a hiányzó feltételt tartalmazó utasítás fölösleges, válasz lehetne **A.**, **B.**, **C.**, **D.**, tehát, mindegyiket helyesnek nyilváníthatjuk.

A.10. Lovagok és hazugok (6 pont)

Egy szigeten csak lovagok élnek, akik mindig igazat mondanak és hazugok, akik mindig hazudnak. Egy látogató, aki a szigetre érkezett szeretné eldönteni két helybéli lakos természetét, akikkel találkozna a szigeten. Így, amikor találkozik két lakossal, A-val és B-vel, felteszi a Q_1 kérdést A-nak: „Mindketten lovagok vagytok?” de a kapott válasz (V_1) alapján nem tudja eldönteni, hogy milyen természetű a két lakos. Ezért, a látogató feltesz egy újabb Q_2 kérdést A-nak: „Egyformák vagytok, mindketten lovagok vagy mindketten hazugok?”; ezúttal, a kapott válasz (V_2) alapján a látogató már megvilágosodik (vagyis, most már tudja, hogy melyik lakos lovag és melyik hazug).

Az alábbi változatok közül, melyik felel meg az A lakos válaszainak, tudva azt, hogy a látogató pontosan meg tudta állapítani a két helybéli lakos természetét.

A. V_1 : Igen, V_2 : Igen

B. V_1 : Igen, V_2 : Nem

C. V_1 : Nem

D. A **B.** és **C.** válaszok helyesek

Megoldás

Mivel a **C.** csak a V_1 választ adja meg, ellentmondásba kerültünk a kijelentéssel, miszerint az első válasz a látogatónak nem volt elegendő. A **D.** pont szerint a **B.** és **C.** válaszok helyesek. Ez megint nem lehetséges, hiszen a **C.**-ről már eldöntöttük, hogy nem helyes. Ahhoz, hogy az **A.** és **B.** válaszokat vizsgálhassuk, készítettünk egy táblázatot. A fejlécben feltüntetünk minden lehetséges esetet, ami a két lakos természetét illeti. A következő sor A válaszait tartalmazza a megfelelő esetekben:

A / B	L / L	L / H	H / L	H / H
V_1	igen	nem	igen	igen
V_2	igen	nem	igen	nem

Ha feltételezzük, hogy A és B is lovag, a válaszok „igen” + „igen”. Sajnos, ugyanezt a választ kapjuk akkor is, ha A hazug és B lovag. Tehát *nem* tudjuk, hogy az A lakos lovag vagy hazug.

A „nem” + „nem” válaszokat nem értékeljük, mivel nem szerepelnek a lehetséges válaszok között a feladatban.

Marad az A lakos „igen” + „nem” válasza. Ha A lovag lenne, az első válaszból az derülne ki, hogy lovagok mindketten, de a második válasz ennek ellentmond. Marad a feltételezés, hogy A hazug, de akkor mindkét válasza hamis, vagyis *nem* mindketten lovagok, de *egyformák*, vagyis hazugok. Így a **B.** válasz a helyes.

B Rész (30 pont)

Banánok



Egy hajótörés után a tengerészeknek sikerült a mentőcsónakkal megmenekülni. Minden csónakban volt három tengerész és minden csónak más-más szigeten vetődött partra. Élelem után kellett nézniük és minden i . szigeten a tengerészek összegyűjtöttek b_i banánt, amelyeket elraktározták a csónakjukba, mivel úgy döntöttek, hogy csak másnap osztják szét egymás között. Bármely csónakban legtöbb k banán fér el. Az éjszaka folyamán, az egyik szigeten, az egyik tengerész felkelt és szétosztotta a csónakban levő banánokat három részre, minden halomba ugyanannyi banánt tett, de maradt még egy banán, amit nem tehetett egyik halomba sem, így ezt megette. Ezután, az egyik részt elrejtette, a másik két részt visszatette a csónakba és lefeküdt aludni. Reggelig, minden tengerész, mindegyik szigeten elvégezte ugyanezt a titkos beavatkozást (elosztotta a banánokat három egyenlő részre és megette az egyetlen megmaradt banánt).

Reggel, mindegyik szigeten, a megmaradt banánokat három egyenlő (nem üres) részre osztották és megint maradt egy banán, amit a tengerészek egy majomnak adtak. Igen, mindegyik szigeten élt egy-egy majom!☺

Követelmények:

- B.1.** Ha az egyik szigeten, az éjszaka beállta előtt, a tengerészek összegyűjtöttek 241 banánt, hány banán maradt egy-egy halomban az osztozkodás végén (ezen a szigeten)? (2 pont)
- B.2.** Ha az egyik szigeten, az osztozkodás végén minden halom 15 banánt tartalmazott, és egy másik szigeten minden halom 31 banánt, hány banánt gyűjtöttek összesen a két szigeten? (2 pont)
- B.3.** Írjatok alprogramot, amely egy adott k számra kiszámítja azt a lehetséges legnagyobb $bmax$ értéket, amely azoknak a banánoknak a száma, amelyeket a tengerészek egy bizonyos szigeten összegyűjtöttek. Bemeneti paraméter: k . Kimeneti paraméter: $bmax$. Keressetek képletet, amely kiszámítja $bmax$ legnagyobb értékét k függvényében. Magyarazzátok el az alkalmazott gondolatmenetet. ($bmax, k$ – természetes számok, $1 \leq bmax \leq k$, $100 \leq k \leq 10\,000\,000$). (14 pont)

Példa: ha $k = 200$, akkor $bmax = 160$.

- B.4.** Írjatok alprogramot, amely egy adott k számra kiszámítja az összes szigeten összegyűjtött maximális banánszámok összegét (*összegMax*). Tudjuk, hogy minden szigeten a tengerészek különböző b_i számú banánt gyűjtöttek (b_i – természetes szám, $i = 1, 2, \dots, sz$, $1 \leq b_i \leq k$, $100 \leq k \leq 10\,000\,000$). Bemeneti paraméterek: k és sz – a szigetek száma (sz – természetes szám, $2 \leq sz \leq 10$). Kimeneti paraméter: *összegMax* – az összes szigeten összegyűjtött maximális banánszámok összege. A k és sz változók értékei úgy vannak megadva, hogy a feladatnak legyen megoldása. (12 pont)

Példa: ha $k = 400$, $sz = 3$, akkor a három szigeten rendre 322, 241 és 160 banánt gyűjtöttek. Következik, hogy az összes szigeten összegyűjtött maximális banánszámok összege *összegMax* = $322 + 241 + 160 = 723$.

Megoldás

Bevezetünk néhány jelölést, ahol n az összegyűjtött banánok száma.

- $n = 3 * p + 1$ (eredetileg összegyűjtött banánok)
- $2 * p = 3t + 1$ (az első éjszaka után maradt banánok)
- $2 * t = 3q + 1$, (a második éjszaka után)
- $2 * q = 3r + 1$ (a harmadik éjszaka után)
- reggel $3 * r + 1$ banánt osztanak szét, (egy végleges halomban r banán van)

Elvégezzük a behelyettesítéseket és megkapjuk, hogy $r = (8 * n - 65) / 81$ (*), ami az osztzkodás reggelén található banánok száma (n , vagyis az eredetileg összegyűjtött banánok számának függvényében). Innen következik, hogy 81-esével haladunk ahhoz, hogy végeredményhez jussunk.

Mivel egész számok halmazán dolgozunk, következik, hogy r páratlan szám és arra is rájövünk (az alábbiak alapján), hogy r legkisebb lehetséges értéke 7, és n legkisebb lehetséges értéke 79:

- ha $r = 1$, $2 * q = 4$, $2 * t = 7$ – lehetetlen,
- ha $r = 3$, $2 * q = 10$, $2 * t = 16$, $2 * p = 25$ – lehetetlen
- ha $r = 5$, $2 * q = 16$, $2 * t = 25$ – lehetetlen
- ha $r = 7$, $2 * q = 22$, $2 * t = 34$, $2 * p = 52$, $n = 79$

Erre az eredményre jutunk a következőképpen is:

A fent kiszámolt (*) reláció alapján $\Rightarrow n = (81 * r + 65) / 8$ (**). Ezt fel lehet írni így is: $n = (80 * r + 64) / 8 + (r + 1) / 8$. Ebből következik, hogy $r + 1$ a 8 többszöröse, így r legkisebb lehetséges értéke 7.

Válaszolunk a feladatban megfogalmazott kérdésekre:

B.1. (*) $\Rightarrow r = (8 * n - 65) / 81$. Behelyettesítve n -et: $(241 * 8 - 65) / 81 = 23$.

B.2. (**) $\Rightarrow n = (81 * r + 65) / 8$. Behelyettesítve a két szigeten összegyűjtött banánok számát: $(81 * 15 + 65) / 8 + (81 * 31 + 65) / 8 = 482$ banán.

B.3. Az algoritmust is megírjuk a képleteinkre támaszkodva: megkeressük azt a 79-nél nagyobb számot, 81-esével haladva, amelyre: $n_{\max} = 79 + d * 81$, ahol n_{\max} -ot helyettesítjük k -val (lehetséges legtöbb banán), és meghatározzuk d -t: $d = (k - 79) / 81$.

Innen következik, hogy: $b_{\max} = 79 + [(k - 79) / 81] * 81$.

```
int bananokV1(int k){
    return 79 + (k - 79) / 81 * 81;
}
```

Az algoritmus dolgozhat egy „okos” ismétlő struktúrával (felhasználva a (*) relációt):

```
int bananokV2(int k){
    while ((8 * k - 65) % 81 != 0){
        k--;
    }
    return k;
}
```

Ha nincs jobb ötletünk, olyan algoritmust írunk, amely szimulálja az eseményeket:

```

int bananokV3(int k){
    int n = k;
    while (n >= 4){
        if (n % 3 == 1){
            int n1 = 2 * n / 3;
            if (n1 % 3 == 1){
                int n2 = 2 * n1 / 3;
                if (n2 % 3 == 1){
                    int n3 = 2 * n2 / 3;
                    if (n3 % 3 == 1)
                        return n;
                }
            }
        }
        n--;
    }
}

```

B.4. Fel kell dolgoznunk minden sziget esetében az eseményeket, tehát az algoritmusnak tartalmaznia kell egy ismétlő struktúrát a szigetek feldolgozására és ki kell számolnia a különböző szigetekeken összegyűjtött banánok számát, amit összegeznie kell. Vigyáznunk kell arra is, hogy az egyes szigetekeken begyűjtött banánok összege különböző legyen:

```

int bananokMindenSzigetV1(int k, int sz){
    int osszegMax = 0;
    for(int i = 1; i <= sz; i++){ // vesszük sorra a szigeteket
        int ni = bananokV2(k);    // az i. szigeten összegyűjtött banánok száma
        osszegMax += ni;          // az összes szigeten összegyűjtött banánok
        k = ni - 1;               // a következő szigeten már kisebb a k
    }                             // így különböző ni értékeket generálunk
    return osszegMax;
}

```

Ha a két követelményt összefűzzük:

```

int bananokV4(int k, int b[], int & nb){
    int n = 1;
    nb = 0;
    while (n <= k){
        if ((8 * n - 65) % 81 == 0){
            nb++;
            b[nb] = n;
        }
        n++;
    }
    return b[nb];
}

```

```
int mindenSzigetV2(int k, int sz){
    int nb = 0;
    int b[100000];
    bananokV4(k, b, nb);
    int s = 0;
    for(int i = nb; (i >= 1 && sz > 0); i--){
        s += b[i];
        sz--;
    }
    return s;
}
```

4.1. Általánosságok – 2018

2018-ban a rácsteszték száma hat volt, egyenként öt lehetséges válasszal. A helyes válaszáért/válaszokért minden teszt esetében öt pontot lehetett szerezni. A **B** rész feladatainak megoldása egy-egy algoritmus vagy alprogram volt, amelyeket a versenyzőnek a megadott bemeneti és kimeneti paraméterekkel kellett megírnia. A versenyfeladatokat – mind a *Matek–Infó versenyen*, mind a *felvételin* – a következő figyelmeztető szöveg előzte meg:

A versenyzők figyelmébe:

1. A tömböket 1-től kezdődően indexeljük.
2. A rácstesztékre (**A** rész) egy vagy több helyes válasz lehetséges. A válaszokat a vizsgadolgozatba írástok (nem a feladatlapra). Ahhoz, hogy a feltüntetett pontszámot megkapjátok, elengedhetetlenül szükséges, hogy minden helyes választ megadjatok, és kizárólag csak ezeket.
3. A **B** részben szereplő feladatok megoldásait részletesen kidolgozva a vizsgadolgozatba írástok.
 - a. A feladatok megoldásait *pszeudokódban* vagy egy *programozási nyelvben* (*Pascal/C/C++*) kell megadnotok.
 - b. A megoldások értékelésekor az első szempont az algoritmus **helyessége**, majd a **hatékonysága**, ami a *végrehajtási időt* és a *felhasznált memória méretét* illeti.
 - c. A tulajdonképpeni megoldások előtt, **kötelezően leírástok szavakkal az alprogramokat, és megindokoljátok a megoldásotok lépéseit**. Feltétlenül írástok **megjegyzéseket** (kommenteket), amelyek segítik az adott megoldás technikai részleteinek megértését. Adjátok meg az azonosítók jelentését és a fölhasznált adatszerkezeteket. Ha ez hiányzik, a tételre kapható pontszámotok 10%-kal csökken.
 - d. Ne használjátok különleges fejlálmányokat, előredefiniált függvényeket (például STL, karakterláncokat feldolgozó sajátos függvények stb.).

A feladatlap néhány hasznos információval záródik:

Megjegyzések:

1. Minden tétel kidolgozása kötelező.
2. A piszkozatokat nem vesszük figyelembe.
3. Hivatalból jár 10 pont.
4. Rendelkezesetekre áll 3 óra.

4.2. Matek-Infó verseny – 2018. március 25.

A rész (30 pont)

A.1. Vajon mit csinál? (5p)

Adott az $\text{alg}(x, b)$ algoritmus, amelynek bemeneti paraméterei az x és b természetes számok ($1 \leq x \leq 1000, 1 < b \leq 10$). Határozzátok meg az algoritmus hatását.

```
Algoritmus alg(x, b):  
  s ← 0  
  Amíg x > 0 végezd el  
    s ← s + x MOD b  
    x ← x DIV b  
  vége(amíg)  
  térítsd s MOD (b - 1) = 0  
Vége(algoritmus)
```

- A. kiszámítja az x természetes szám számjegyeinek összegét a b számrendszerben
- B. vizsgálja, hogy az x szám számjegyeinek összege a $b - 1$ számrendszerben osztható-e $(b - 1)$ -gyel
- C. vizsgálja, hogy az x szám osztható-e $(b - 1)$ -gyel
- D. vizsgálja, hogy a b számrendszerben felírt x szám számjegyeinek összege osztható-e $(b - 1)$ -gyel
- E. vizsgálja, hogy az x szám számjegyeinek összege osztható-e $(b - 1)$ -gyel

Megoldás

A feladat egy összeg értékét számolja ki. Ha figyelmesen vizsgáljuk az utasításokat, belátjuk, hogy az x bemeneti paramétert a ciklus minden lépésében (amíg x nullává nem válik) osztjuk b -vel, de előbb a b -vel való egész osztás maradékát az összegbe helyezzük.

A **térítsd s MOD (b - 1) = 0** utasítást „magyarul” a következőképpen olvassuk ki: ha s maradék nélkül osztható $(b - 1)$ -gyel, akkor *igaz*-at térít, különben *hamis*-at. Tehát az algoritmus eldönti, hogy az x szám b számrendszerben felírt számjegyeinek összege osztható-e $(b - 1)$ -gyel. Ezt a funkciót tartalmazza a **D.** válasz.

A **B.** választ máris elvetjük, mivel a szövegben az áll, hogy az x szám $b - 1$ számrendszerben felírt számjegyeinek összegét téríti az algoritmus. Ez nem igaz,

hiszen az algoritmus egy döntés eredményét téríti. Úgyisintén, az **A.** válasz sem helyes, hiszen ez a kijelentés is az összeg térítését feltételezi.

Az **E.** választ is helytelennek – jobban mondva – pontatlannak minősítjük, mivel nem tartalmazza, hogy az x szám számjegyeit milyen számrendszerben határozza meg az algoritmus.

Maradt még a **C.** válasz, amelyet egy példa segítségével vizsgálunk.

Ha $x = 12$ és $b = 5$, akkor $b - 1 = 4$ és 12 osztható 4-gyel. Lássuk, mit térít az algoritmus? Kiszámítjuk s értékét (4) majd meghatározzuk az $s \bmod (b - 1) = 0$ relációs kifejezés értékét. Mivel 4 osztható 4-gyel, az algoritmus *igaz*-at térít. Lássunk egy másik példát, ahol az $s \bmod (b - 1) = 0$ kifejezés értéke *hamis*. Legyen $x = 13$ és $b = 5$, akkor $b - 1 = 4$ és 13 nem osztható 4-gyel. Az s értéke most 5, és valóban, 5 nem osztható 4-gyel. Tehát a **C.** válasz is helyes.

A.2. Mit fog kiírni? (5p)

Legyen a következő program:

C++/C	
<pre>int sum(int n, int a[], int s){ s = 0; int i = 1; while (i <= n){ if(a[i] != 0) s += a[i]; ++i; } return s; }</pre>	<pre>int main(){ int n = 3, p = 0, a[10]; a[1] = -1; a[2] = 0; a[3] = 3; int s = sum(n, a, p); cout << s << " "; // printf("%d;%d", s, p); return 0; }</pre>
Pascal	
<pre>type vektor = array[1..10] of integer; function sum(n: integer; a: vektor; s: integer): integer; var i : integer; begin s := 0; i := 1; while i <= n do begin if a[i] <> 0 then s := s + a[i]; i := i + 1; end; sum := s; end;</pre>	<pre>var n, p, s: integer; a: vektor; begin n := 3; a[1] := -1; a[2] := 0; a[3] := 3; p := 0; s := sum(n, a, p); writeln(s, ' '); end.</pre>

Mit fog kiírni a program?

- A.** 0;0 **B.** 2;0 **C.** 2;2 **D.** Egyik válasz sem helyes **E.** 0;2

Megoldás

A számolást fejben végezzük, hiszen a sorozatnak csak három eleme van, és a program ezeket adja össze (a 0 értékűt kivéve). Az összeg egyenlő 2-vel, így ezt írja ki a program és a 0-t, mivel *p* érték szerint átadott paraméter volt, így a hívás helyén megmarad a hívás előtt érvényes értéke. Helyes válasz: **B**.

A.3. Logikai kifejezés (5p)

Legyen a következő logikai kifejezés: $(X \text{ OR } Z) \text{ AND } (\text{NOT } X \text{ OR } Y)$. Válaszátok ki *X*, *Y*, *Z* értékeit úgy, hogy a kifejezés értéke legyen *igaz*:

- A. *X* = hamis; *Y* = hamis; *Z* = igaz;
- B. *X* = igaz; *Y* = hamis; *Z* = hamis;
- C. *X* = hamis; *Y* = igaz; *Z* = hamis;
- D. *X* = igaz; *Y* = igaz; *Z* = igaz;
- E. *X* = hamis; *Y* = hamis; *Z* = hamis;

Megoldás

Kiértékeljük a kifejezést, rendre a megadott *X*, *Y*, *Z* értékekkel:

- A. (hamis vagy igaz) és (nem hamis vagy hamis) = igaz és igaz = igaz
- B. (igaz vagy hamis) és (nem igaz vagy hamis) = igaz és hamis = hamis
- C. (hamis vagy hamis) és (nem hamis vagy igaz) = hamis és igaz = hamis
- D. (igaz vagy igaz) és (nem igaz vagy igaz) = igaz és igaz = igaz
- E. (hamis vagy hamis) és (nem hamis vagy hamis) = hamis és igaz = hamis

A fentiek alapján a kifejezés helyes az **A.** és **D.** esetekben megadott adatokkal.

A.4. Számolás (5p)

Legyen a számol(*a*, *b*) algoritmus, amelynek bemeneti paraméterei az *a* és *b* pozitív természetes számok, ahol $1 \leq a \leq 100$, $1 \leq b \leq 100$.

```
1. Algoritmus számol(a, b):
2.   Ha a ≠ 0 akkor
3.     térítsd számol(a DIV 2, b + b) + b * (a MOD 2)
4.   vége(ha)
5.   térítsd 0
6. Vége(algoritmus)
```

Az alábbi válaszok közül melyek hamisak?

- A. ha *a* és *b* egyenlők, az algoritmus *a* értékét téríti
- B. ha *a* = 1000 és *b* = 2, az algoritmus 10-szer hívja meg önmagát
- C. az algoritmus által kiszámított és térített érték egyenlő $(a / 2 + 2 * b)$ -vel
- D. az 5. sorban található utasítás egyszer sem hajtódik végre
- E. az 5. sorban található utasítás egyszer hajtódik végre

Megoldás

Megjegyezzük, hogy a *hamis* állításokat kell megadnunk!

Remélhetően sok feladatmegoldó felismeri az algoritmust, hiszen ez az úgynevezett „orosz szorzás”. Az sem kizárt, hogy van olyan feladatmegoldó is, aki tudja, hogy ez egy logaritmikus bonyolultságú algoritmus, tehát a **B.** válasz helyes, hiszen $\log_2 1000 \approx 10$ (mivel $2^{10} = 1024$). Tehát a **B.**-t nem jelöljük meg.

Ha bárki hiányolná a szorzandó vizsgálatát (páros vagy páratlan?), ajánlatos figyelmesen tanulmányoznia, például az algoritmus 3. sorát:

térítsd számol(**a DIV 2, b + b**) + **b * (a MOD 2)**.

A rekurzív hívás paraméterei – a várakozásnak megfelelően – a szorzandó fele és a szorzó kétszerese. Az utasítást a következőképpen értékelhetjük ki: ha **a** páratlan szám, **a MOD 2 = 1**, tehát a térített értékhez hozzáadjuk **b * 1 = b**-t, különben a **b**-t 0-val szorozza az algoritmus, tehát nem adja hozzá.

Ha valaki nem ismeri az algoritmust, az sem okoz gondot, hiszen, például az **A.** választ illetően, azonnal észre lehet venni, hogy az algoritmus sehol nem vizsgálja (még „burkoltan” sem) **a** és **b** egyenlőségét. Másfelől, az eredmény bizonyos aktuálisan érvényes **b** értékek összege, így az algoritmus soha nem térítheti **a**-t. Megtaláltuk az első hamis állítást.

A **C.** pontban olyan állítás található, amely szintén nem lehet igaz; a fent leírt érveléshez hasonlóan rájövünk, hogy az összeg, amit térít az algoritmus nem lehet egyenlő $(a / 2 + 2 * b)$ -vel. Ezek szerint a **C.** is hamis.

A **D.** és **E.** állításokat együttesen vizsgáljuk. Hamar belátjuk, hogy **E.** helyes, hiszen a rekurzív hívások addig ismétlődnek, amíg **a** > 0, de ezután végrehajtódik az 5. sorban található utasítás. Tulajdonképpen ekkor kap kezdőértéket az összeg, ami végül **a** és **b** szorzata. Tehát a hamis állítások: **A.**, **C.** és **D.**

A.5. Elem beazonosítása(5p)

Legyen az (1, 2, 3, 2, 5, 2, 3, 7, 2, 4, 3, 2, 5, 11, ...) sorozat, amelyet a következőképpen hoztunk létre: kiindulva a természetes számok sorozatából, azokat a számokat, amelyek nem prímszámok helyettesítettük a saját osztóikkal úgy, hogy minden **d** osztót csak egyszer használtunk minden szám esetében. Az alábbi algoritmusok közül melyik határozza meg a sorozat **n**-dik elemét (**n** természetes szám, $1 \leq n \leq 1000$)?

A.	<p>Algoritmus beazonosítás(n):</p> <pre> a ← 1; b ← 1; c ← 1 Amíg c < n végezd el a ← a + 1 b ← a c ← c + 1 d ← 2 Amíg c ≤ n és d ≤ a DIV 2 végezd el Ha a MOD d = 0 akkor c ← c + 1 b ← d vége(ha) d ← d + 1 vége(amíg) vége(amíg) térítsd b Vége(algoritmus) </pre>
B.	<p>Algoritmus beazonosítás(n):</p> <pre> a ← 1; b ← 1; c ← 1 Amíg c < n végezd el c ← c + 1 d ← 2 Amíg c ≤ n és d ≤ a DIV 2 végezd el Ha a MOD d = 0 akkor c ← c + 1 b ← d vége(ha) d ← d + 1 vége(amíg) a ← a + 1 b ← a vége(amíg) térítsd b Vége(algoritmus) </pre>
C.	<p>Algoritmus beazonosítás(n):</p> <pre> a ← 1; b ← 1; c ← 1 Amíg c < n végezd el a ← a + 1 d ← 2 Amíg c < n és d ≤ a végezd el Ha a MOD d = 0 akkor c ← c + 1 b ← d vége(ha) d ← d + 1 vége(amíg) vége(amíg) térítsd b Vége(algoritmus) </pre>

D.	<p>Algoritmus beazonosítás(n):</p> <pre> a ← 1; b ← 1; c ← 1 Amíg c < n végezd el b ← a a ← a + 1 c ← c + 1 d ← 2 Amíg c ≤ n és d ≤ a DIV 2 végezd el Ha a MOD d = 0 akkor c ← c + 1 b ← d vége(ha) d ← d + 1 vége(amíg) vége(amíg) térítsd b Vége(algoritmus) </pre>
E.	<p>Algoritmus beazonosítás(n):</p> <pre> a ← 1; b ← 1; c ← 1 Amíg c < n végezd el a ← a + 1 b ← a c ← c + 1 d ← 2 f ← hamis Amíg c ≤ n és d ≤ a DIV 2 végezd el Ha a MOD d = 0 akkor c ← c + 1 b ← d f ← igaz vége(ha) d ← d + 1 vége(amíg) Ha f akkor c ← c - 1 vége(ha) vége(amíg) térítsd b Vége(algoritmus) </pre>

Megoldás

Bevezetésként megjegyezzük, hogy a feladat nem kéri, hogy generáljuk a sorozatot, így nincs szükség az elemek tárolására. Ami a helyes algoritmus/algoritmusok kiválasztását illeti, sajnos nincs idő lépésenként futtatni a papíron mind az öt algoritmust. Megpróbáljuk elemezni, megérteni és összehasonlítani őket.

Hasonlóságokat több helyen is látunk: mindegyik algoritmus a **b** változó értékét téríti, tehát a sorozat n -edik elemét **b**-ben kapjuk meg. A kezdőértékadások is azonosak, úgymint a külső **Amíg** struktúra feltétele, amelynek egyértelműen az a szerepe, hogy addig dolgoztassa az algoritmust, amíg a **c** számláló egyenlővé nem válik n -nel.

A külső **Amíg** cikluson belül már sok különbség létezik az algoritmusok között. Például, észrevesszük, hogy csak az **E.** algoritmusban jelenik meg az f logikai változó. Lássuk, mi a szerepe? Az f a *hamis* kezdőértékről akkor vált *igaz*-ra, ha az aktuálisan vizsgált a számról eldönt, hogy nem prím, vagyis -2 -től indulva – találtunk egy d osztót. Mivel a feladat azt kéri, hogy az aktuális a elemet *őrizzük meg a sorozatban* **vagy** helyettesítsük az osztójával/osztóival, most ezt a d -t el kellene helyeznünk a sorozatba, de – ugyebár – csak gondolatban, hiszen a legfontosabb a c változó értékét figyelni, amely a képzeletbeli sorozat elemeit számlálja. Mivel nem lehet tudni pontosan, hogy mikor áll le az **Amíg**, a b változó rendre megkapja a sorozatba aktuálisan elhelyezendő elem értékét. Igen, de az **E.** algoritmus minden aktuális iteráció elején az a aktuális értékét „megszámolja” (c értéke nő 1 -gyel). Ha találunk osztót, azt is „megszámoljuk”. Most már világos az f változó szerepe: ha volt osztó, amit gondolatban a sorozatba tettünk, ez az osztó tulajdonképpen „felülírja” a értékét, tehát a c változó értékének csökkennie kell 1 -gyel.

Az **E.** algoritmus belső **Amíg** ciklusának szerepe az aktuális a szám feldolgozása. A folytatási feltétel helyes, a d osztót csak egyszer számolja meg (miután felhasználta, vagy nem, az értéke nő 1 -gyel). A külső **Amíg** ciklus minden iterációja arra készül, hogy rendre feldolgozza a természetes számokat, amelyeket az a változó tárol. A d és f változókat helyes kezdőértékekkel látja el.

Úgy tűnik, hogy az **E.** algoritmus helyes. Mindenképpen van egy fontos észrevételünk, ami az f változó szerepét illeti. Vesszük sorra a többi négy algoritmust, és vizsgáljuk, hogy ezek az algoritmusok hogyan számlálják a sorozat elemeit.

Az **A.** algoritmus megszámlálja az a számot, de utána az osztókat is, tehát nem helyes.

A **B.** algoritmus megszámlálja a -t, de nem teszi be b -be, hanem folytatja az a osztóinak keresésével. Az a értékét az osztók után teszi b -be, függetlenül attól, hogy volt-e osztó vagy sem. Így több szempontból sem teljesíti a feladat követelményeit.

A **C.** algoritmus nem számlálja meg a -t, függetlenül attól, hogy volt-e osztója vagy sem. Ugyanakkor a b változóba sem helyezi el. Következik, hogy a **C.** algoritmus sem helyes.

Ha $n > 1$, a **D.** algoritmus az 1 értéket kétszer teszi a sorozatba (kétszer számlálja meg). Ráadásul, a többi természetes számot és az osztókat is megszámlálja. Így a **D.** algoritmus szintén hibás.

Mivel van négy hibás algoritmusunk, visszatérünk az **E.** algoritmushoz, még egyszer, figyelmesen követjük az utasítások hatását, és eldöntjük, hogy helyes.

A.6. Törzstényezők (5p)

Legyen a törzsTényezők(n , d , k , x) algoritmus, amely meghatározza az n természetes szám k darab törzstényezőjét, a törzstényezők keresését a d értéktől kezdve. Bemeneti paraméterek az n , d és k számok, kimeneti paraméterek az x sorozat, amely a k darab törzstényezőt tartalmazza ($1 \leq n \leq 10000$, $2 \leq d \leq 10000$, $0 \leq k \leq 10000$).

```

Algoritmus törzsTényezők( $n$ ,  $d$ ,  $k$ ,  $x$ ):
  Ha  $n \bmod d = 0$  akkor
     $k \leftarrow k + 1$ 
     $x[k] \leftarrow d$ 
  vége(ha)
  Amíg  $n \bmod d \neq 0$  végezd el
     $n \leftarrow n \operatorname{DIV} d$ 
  vége(amíg)
  Ha  $n > 1$  akkor
    törzsTényezők( $n$ ,  $d + 1$ ,  $k$ ,  $x$ )
  vége(ha)
Vége(algoritmus)

```

Határozzátok meg, hányszor hívja meg önmagát a törzsTényezők(n , d , k , x) algoritmus a következő programrészlet végrehajtásának következtében:

```

 $n \leftarrow 120$ 
 $d \leftarrow 2$ 
 $k \leftarrow 0$ 
törzsTényezők( $n$ ,  $d$ ,  $k$ ,  $x$ )

```

A. 3-szor

B. 5-ször

C. 9-szer

D. 6-szor

E. ugyanannyiszor, mint a következő programrészlet esetében:

```

 $n \leftarrow 750$ 
 $d \leftarrow 2$ 
 $k \leftarrow 0$ 
törzsTényezők( $n$ ,  $d$ ,  $k$ ,  $x$ )

```

Megoldás

A rekurzív algoritmus akkor hívja meg önmagát, ha n (még) nagyobb, mint 1. A folytatási feltételt magába foglaló **Ha** utasítás előtt az n szám aktuális d osztóját elhelyezi az x sorozatba, de előbb aktualizálja a sorozat k hosszát, majd addig osztja az n számot d -vel, amíg ez maradék nélkül lehetséges. Így eléri azt, hogy egy törzstényező csak egyszer kerüljön az x sorozatba. Könnyen belátható, hogy a rekurzív hívások száma egyenlő azoknak a d értékeknek a számával, amelyek nagyobbak, mint 2 és kisebbek vagy egyenlők n legnagyobb prímosztójával. Az **A.** és **E.** pontoknak megfelelően az újrahívások száma $5 - 2 = 3$. A törzstényezők:

120	2	750	2
60	2	375	3
30	2	125	5
15	3	25	5
5	5	5	5
1		1	

Újrahívás mindig egy új d értékkel történik. Például, ha $n = 120$, az algoritmus meghívja önmagát $d = 3$ -ra, 4-re és 5-re, vagyis 3-szor, tehát a rekurzív hívások száma nem lehet 5 vagy 9, esetleg 6. Ha $n = 750$, a rekurzív hívások száma szintén 3, tehát a helyes válaszok: **A.** és **E.**

B rész (60 pont)

B.1. Konverzió (10 pont)

Legyen a konverzió(s , hossz) algoritmus, amely átalakítja az s karakterláncot, amely egy 16-os számrendszerben ábrázolt szám, a megfelelő 10-es számrendszerben érvényes alakjára. Az s karakterlánc **hossz** darab karaktert tartalmaz, ahol a karakter értéke lehet egy '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' számjegy, vagy egy 'A', 'B', 'C', 'D', 'E', 'F' nagybetű (a **hossz** természetes szám, $1 \leq \text{hossz} \leq 10$).

Írjátok le a konverzió(s , hossz) algoritmus *rekurzív* változatát úgy, hogy a fejléce és a hatása legyen azonos az alábbi algoritmuséval:

```

Algoritmus konverzió( $s$ , hossz):
    szám  $\leftarrow$  0
    Minden  $i = 1$ , hossz végezd el
        Ha  $s[i] \geq \text{'A'}$  akkor
            szám  $\leftarrow$  szám * 16 +  $s[i] - \text{'A'} + 10$ 
        különben
            szám  $\leftarrow$  szám * 16 +  $s[i] - \text{'0'}$ 
        vége(ha)
    vége(minden)
    térítsd szám
Vége(algoritmus)

```

Megoldás

Az értékelés a következő szempontok szerint történik:

- A rekurzív algoritmus fejléce azonos-e az iteratív fejlécével? (2 pont)
- A rekurzív hívások leállási feltétele helyes-e? (1 pont)
- A leálláskor térített érték helyes-e? (1 pont)

- Helyesen vizsgálja, hogy a karakter nem számjegy? (2 pont)
- Helyesen határozza meg a nem számjegy karakter értékét? (2 pont)
- Helyesen határozza meg a számjegy karakter értékét? (2 pont)

Egy helyes rekurzív algoritmus:

```

Algoritmus konverzió(s, hossz):
  Ha hossz > 0 akkor
    Ha s[hossz] ≥ 'A' akkor
      térítsd konverzió(s, hossz - 1) * 16 + s[hossz] - 'A' + 10
    különben
      térítsd konverzió(s, hossz - 1) * 16 + s[hossz] - '0'
    vége(ha)
  különben
    térítsd 0
  vége(ha)
Vége(algoritmus)

```

B.2. Azonos számjegyek (20 pont)

Adott két természetes szám: a és b , $1 \leq a \leq 1\,000\,000$ és $1 \leq b \leq 1\,000\,000$.

Írjatok algoritmust, amely meghatározza a k elemű x sorozatot, (k – természetes szám, $0 \leq k \leq 1000$), amely minden olyan természetes számot tárol, amelyek az $[a, b]$ intervallumhoz tartoznak és azonos számjegyekből állnak. Ha ilyen szám nem létezik, k értéke 0 lesz. Az algoritmus bemeneti paraméterei a és b , kimeneti paraméterek pedig k és x .

1. Példa: ha $a = 8$ és $b = 120$, akkor $k = 12$ és $x = (8, 9, 11, 22, 33, 44, 55, 66, 77, 88, 99, 111)$.

2. Példa: ha $a = 590$ és $b = 623$, akkor $k = 0$ és az x sorozat üres.

Megoldás

A következőkben bemutatunk két különböző megoldást. A naiv algoritmus, venné rendre az adott intervallumhoz tartozó számokat, és ellenőrizné a kért tulajdonságot. De a feladatmegoldó „érzi”, hogy valószínű létezik ennél hatékonyabb megoldás.

V1: A hatékony algoritmus (20 pontot ért), nem dolgozik fölöslegesen, hanem *számolásokkal generálja* a kért sorozat elemeit.

Az értékelés a következő szempontok szerint történt:

- A bemeneti és kimeneti paraméterek megfelelnek a követelményekben leírtaknak? (2 pont)
- A számok generálása az egyszámjegyű számokkal kezdődik és/vagy a 11, 111, 1111 stb. számok többszöröseivel folytatódik) (16 pont)
- Helyesen menti a kimeneti sorozatba az azonos számjegyű számokat? (2 pont)

Íme egy lehetséges implementáció:

```
void azonosSzamjegyek_V1(int a, int b, int &k, int x[]){
    k = 0;
    for(int i = a; ((i < 10) && (i <= b)); i++){
        x[++k] = i;
    }
    int akt = 11;
    int leptek = 11;
    while(akt <= b){
        if(akt >= a)
            x[++k] = akt;
        akt = akt + leptek;
        if(akt > 9 * leptek){
            akt = leptek * 10 + 1;
            leptek = akt;
        }
    }
}
```

V2: A második algoritmus bejárja az $[a, b]$ intervallumot számról számra és vizsgálja, hogy azonos számjegyűek-e. (10 pont)

Az értékelés szempontjai:

- A bemeneti és kimeneti paraméterek megfelelnek a követelményekben leírtaknak? (2 pont)
- Helyesen dönti el, hogy a szám számjegyei azonosak vagy sem? (4 pont)
- Az ellenőrzés céljából helyesen járja be az $[a, b]$ intervallumhoz tartozó számokat? (2 pont)
- Helyesen menti a kimeneti sorozatba az azonos számjegyű számokat? (2 pont)

```
bool ok(int szam){ // eldöntjük, hogy a "szam"-ot azonos számjegyek alkotják-e
    int szamjegy = szam % 10;
    szam /= 10;
    while(szam > 0 && szam % 10 == szamjegy){
        szam /= 10;
    }
    return szam == 0;
}
```

```
void azonosSzamjegyek_V2(int a, int b, int &k, int x[]){
// generáljuk az összes számot a-tól b-ig és ellenőrizzük a kért tulajdonságot
    k = 0;
    for (int szam = a; szam <= b; szam++){
        if (ok(szam)){
            k++;
            x[k] = szam;
        }
    }
}
```


B.3. Sétáló robot (30 pont)

Egy robot egy négyzetes tömb alakú térképen mozog, amelynek a mérete páratlan számú, és a térkép minden celláján letesz bizonyos számú tárgyat. A robot a következő szabályok alapján mozog:

- a cellában, ahonnan indul letesz egy tárgyat, a második cellában, ahova lép, letesz kettőt, a harmadik cellában, ahova eljut, hármat és így tovább;
- a robot az utolsó oszlop közepéről indul, és lép egyet átlósan a felfele és jobbra szomszédos szabad cellára (párhuzamosan a mellékátlóval) ha ez a cella létezik és szabad a hely; ha ilyen cella nincs, akkor:
 - ha a robot az utolsó oszlopban található, akkor „átugrik” az első oszlopba, a robot felett levő sorba, ha ez a hely szabad;
 - ha a robot az első sorban található, akkor „átugrik” az utolsó sorba a robottól jobbra levő oszlopba, ha ez a hely szabad;
 - ha a robot a térkép jobb-felső sarkában található. akkor átugrik az utolsó sor első oszlopában levő cellába, ha ez a hely szabad.
- ha a cella, ahova lépni szeretne foglalt, a robot balra lép egyet a sorban, ahol éppén található, a szomszédos szabad cellába.

Ezek a szabályok biztosítják, hogy a robot a térkép minden celláját egyszer látogatja meg (és azt is, hogy nem fog elakadni sehol). Miután a robot a térkép minden cellájában elhelyezte a tárgyakat, megáll.

Például, egy 5×5 cellából álló térképen a robot első 22 lépése a következő:

9	3	22	16	15
2	21	20	14	8
	19	13	7	1
18	12	6	5	
11	10	4		17

Írjatok algoritmust, amely meghatározza azoknak a tárgyaknak a *szám* számát, amelyeket a robot a térkép főátlóján levő cellákba helyez. Az algoritmus bemeneti paramétere a térkép n mérete (n – páratlan természetes szám, $3 \leq n \leq 100$), kimeneti paramétere a *szám* (*szám* – természetes szám).

1. Példa: ha $n = 5$, akkor *szám* = 65; **2. Példa:** ha $n = 11$, akkor *szám* = 671.

Megoldás

V1: Az optimális megoldást egy képlettel valósítjuk meg. Természetesen, akkor járt érte 30 pont, ha helyes indoklás előzte meg, esetleg követte:

$$\text{szám} = (n * n * n + n) / 2.$$

Az értékelés szempontjai:

- A bemeneti és kimeneti paraméterek megfelelnek a követelményekben leírtaknak? (2 pont)
- Helyes képlet (14 pont)
- A számolások (a képlet) részletes megindoklása (14 pont)

```
int setaloRobot_1(int n){
    return (n * n * n + n) / 2;
}
```

Indoklás:

Mozgatjuk a robotot a térképen:

9	3			
2				8
		13	7	1
	12	6	5	
11	10	4		

17	23			
24				18
		13	19	25
	14	20	21	
15	16	22		

		22	16	15
	21	20	14	
25	19	13		
18				24
			23	17

9	3	22	16	15
2	21	20	14	8
25	19	13	7	1
18	12	6	5	
11	10	4		17

A robot útvonala szimmetrikus a négyzet középhez viszonyítva. Másképp fogalmazva, a négyzet közepét a robot pontosan az útnak a közepén érinti ($(n * n - 1) / 2$ lépés után). Az útvonal második része ugyanaz, mint az első, de a bejárást fordított sorrendben végezzük és elforgatjuk 180 fokkal a négyzet középhez viszonyítva (például, az utolsó négyzet a bal szél közepén lesz). Következik, hogy bármely négyzetpárra, amelyek szimmetrikusak a középponthoz viszonyítva fennáll, hogy a tárgyak számának összege $n * n + 1$. Ha ezt az észrevételt alkalmazzuk a főátló elemeire, majd párokat alkotunk belőlük, megkapjuk a kért összeget az $n * (n * n + 1) / 2$ kifejezést.

V2: Ha a feladatmegoldó nem jön rá a fenti megoldásra, a kért eredményt meghatározhatja egy helyes szimulálással (25 pont)

Az értékelés szempontjai:

- A bemeneti és kimeneti paraméterek megfelelnek a követelményekben leírtaknak? (2 pont)
- Helyesen járja be az összes $n \times n$ cellát? (4 pont)
- Helyesen mozgatja a robotot a 4 lehetséges esetben? (4*4 pont)
- Helyesen határozza meg a főátlón található tárgyak számát? (3 pont)

Mielőtt elkezdődne a mozgás, kezdőértékkel látjuk el a térképet ábrázoló tömböt, majd kiválasztjuk az indulási pontot. „Menet” közben, vigyázunk, hogy amikor valamelyik szélre értünk, helyesen változtassuk meg a sor és oszlopindexeket.

Végül, kiszámítjuk a főátló elemeinek összegét.

```
int setaloRobot_2(int n){
    int a[100][100];
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            a[i][j] = 0;
    int sor = n / 2 + 1;
    int oszlop = n;
    for (int i = 1; i <= n * n; i++){
        a[sor][oszlop] = i;
        int ujOszlop = oszlop + 1;
        int ujSor = sor - 1;
        if (ujOszlop > n){
            if (ujSor >= 1){
                ujOszlop = 1;
            }
            else{
                ujOszlop = 1;
                ujSor = n;
            }
        }
        else{
            if (ujSor < 1){
                ujSor = n;
            }
        }
        if (a[ujSor][ujOszlop] > 0){
            ujOszlop = oszlop - 1;
            ujSor = sor;
        }
        sor = ujSor;
        oszlop = ujOszlop;
    }
    int szam = 0;
    for (int i = 1; i <= n; i++){
        szam += a[i][i];
    }
    return szam;
}
```

4.3. Felvételi verseny – 2018. július 15.

A rész (30 pont)

A.1. Vajon mit csinál? (5p)

Adott a kifejezés(n) algoritmus, ahol n ($1 \leq n \leq 10000$) természetes szám.

```
Algoritmus kifejezés(n):  
  Ha  $n > 0$  akkor  
    Ha  $n \bmod 2 = 0$  akkor  
      térítsd  $-n * (n + 1) + \text{kifejezés}(n - 1)$   
    különben  
      térítsd  $n * (n + 1) + \text{kifejezés}(n - 1)$   
    vége(ha)  
  különben  
    térítsd 0  
  vége(ha)  
Vége(algoritmus)
```

Állapítsátok meg az $E(n)$ kifejezésnek azt a matematikai alakját, amelyet a fenti algoritmus számít ki:

- A. $E(n) = 1 * 2 - 2 * 3 + 3 * 4 + \dots + (-1)^{n+1} * n * (n + 1)$
- B. $E(n) = 1 * 2 - 2 * 3 + 3 * 4 + \dots + (-1)^n * n * (n + 1)$
- C. $E(n) = 1 * 2 + 2 * 3 + 3 * 4 + \dots + (-1)^{n+1} * n * (n + 1)$
- D. $E(n) = 1 * 2 + 2 * 3 + 3 * 4 + \dots + (-1)^n * n * (n + 1)$
- E. $E(n) = 1 * 2 - 2 * 3 - 3 * 4 - \dots - (-1)^n * n * (n + 1)$

Megoldás

Az algoritmus nem hívja meg önmagát, ha n értéke 0-ra csökkent, ekkor kap az összeg, amit kiszámít 0 kezdőértéket. Látjuk, hogy az n aktuális értékének párosságától függően az rekurzív hívás különböző értékeket térít. Ha n páros, az aktuális tagot kivonja az összeg aktuális értékéből, ha n páratlan, hozzáadja. Ekkor már kizárhatjuk a lehetséges válaszok közül a C.-t és a D.-t, mivel ezekben a kifejezésekben nincs kivonás. A továbbiakban vizsgáljuk a kifejezések utolsó tagját. Az A. kifejezésben a hatványkitevő $(n + 1)$, amiből következik, hogy ha n páros (tehát $(n + 1)$ páratlan), ezt a tagot kivonjuk a kifejezés eddigi értékéből. Ez megfelel az algoritmus előbbi elemzésének, tehát az A. válasz helyes. A B. kifejezésben a (-1) -et az n . hatványra emeljük, de így a tagok előjele, nem az algoritmusnak megfelelően váltakozik, tehát a B. válasz sem helyes. Az E. kifejezésben túl sok tag előjele „–”, tehát nem ezt a kifejezést értékeli az algoritmus.

A.2.Számolás (5p)

Adott a számol(n) algoritmus, ahol n egy természetes szám ($1 \leq n \leq 10000$).

```

Algoritmus számol( $n$ ):
   $x \leftarrow 0$ 
   $z \leftarrow 1$ 
  Amíg  $z \leq n$  végezd el
     $x \leftarrow x + 1$ 
     $z \leftarrow z + 2 * x$ 
     $z \leftarrow z + 1$ 
  vége(amíg)
  térítsd  $x$ 
Vége(algoritmus)

```

Az alábbi válaszok közül melyek **hamisak**?

- A. Ha $n = 25$ vagy $n = 35$, akkor a számol(n) 5-öt térít vissza
- B. Ha $n < 8$, akkor a számol(n) 3-at térít vissza
- C. Ha $n \geq 85$ és $n < 100$, akkor a számol(n) 9-et térít vissza
- D. Az algoritmus kiszámítja és visszatéríti az n -nél kisebb, szigorúan pozitív négyzetszámok darabszámát
- E. Az algoritmus kiszámítja és visszatéríti az n szám négyzetgyökének egész részét

Megoldás

Megjegyezzük, hogy a hamis válaszokat kell megjelölnünk!

A legegyszerűbb előbb az **A.**, **B.** és **C.** válaszokat vizsgálni. Ebből a célból ellenőrző táblázatot készítünk:

$n = 25$, majd 35	
x	z
0	1
1	3
	4
2	8
	9
3	15
	16
4	24
	25
5	35
	36

$n < 8$	
x	z
0	1
1	3
	4
2	8

$n \geq 85$ és $n < 100$	
x	z
...	...
5	35
	36
6	48
	49
7	63
	64
8	80
	81
9	99

A táblázat kitöltése közben rájöttünk, hogy miről is van szó. Az x értéke egyesevel nő 1-től és amikor az **Amíg** struktúra végrehajtása véget ér, az algoritmus az x aktuális értékét téríti. Egy-egy iteráció végén z aktuális értéke egyenlő az x aktuális értékének a négyzetével. Amikor z értéke egyenlővé válik az adott n szám értékével, még végrehajtunk egy lépést, így az algoritmus azt az x értéket téríti, amelynek a négyzete kisebb, mint az adott szám, vagyis az adott szám négyzetgyökének egész részét.

A példánk esetében, ahol $n = 25$, és $z = 36$, akkor $x = 5$. Amikor $n = 35$, a z értéke szintén 36-ig nő és x értéke most is 5. Az **A.** válasz helyes, tehát nem jelöljük meg. De arra is rájöttünk, hogy az **E.** is helyes, tehát ezt sem jelöljük meg.

A **B.** válasz azt állítja hogy, ha $n < 8$, az algoritmus 3-at térít, de látjuk a táblázatban, hogy ez nem igaz, hiszen ha például $n = 2$, az algoritmus 1-et térít, ha $n = 5$, az algoritmus 2-t térít stb. Tehát **B.** nem helyes, így megjelöljük.

A **C.** helyes (látjuk a táblázatban), nem jelöljük meg.

Ahhoz, hogy a **D.** válaszról eldönthessük, hogy helyes-e, előbb meg kell vizsgálnunk, hogy ha az algoritmusról már eldöntöttük, hogy az adott szám négyzetgyökének egész részét határozza meg, ez vajon egyenlő-e, az n -nél kisebb, szigorúan pozitív négyzetszámok darabszámával? Ha ez a válasz az „ n -nél kisebb vagy n -nel egyenlő” négyzetszámok darabszámát tartalmazta volna, akkor az állítás igaz lenne, de így hiányos. Például, ha $n = 25$, az algoritmus 5-öt térít, miközben a 25-nél kisebb négyzetszámok darabszáma 4. Tehát a **D.**-t is megjelöljük.

A.3. Logikai kifejezés (5p)

Legyen a következő logikai kifejezés: $(\text{NOT } Y \text{ OR } Z) \text{ OR } (X \text{ AND } Y)$. Válasszátok ki X, Y, Z értékeit úgy, hogy a kifejezés kiértékelésének eredménye legyen igaz:

- A. $X = \text{hamis}; Y = \text{hamis}; Z = \text{hamis};$
- B. $X = \text{hamis}; Y = \text{hamis}; Z = \text{igaz};$
- C. $X = \text{hamis}; Y = \text{igaz}; Z = \text{hamis};$
- D. $X = \text{igaz}; Y = \text{hamis}; Z = \text{igaz};$
- E. $X = \text{hamis}; Y = \text{igaz}; Z = \text{igaz};$

Megoldás

Kiértékeljük a kifejezést, rendre a megadott X, Y, Z értékekkel:

- A. $(\text{not hamis or hamis}) \text{ or } (\text{hamis and hamis}) =$
 $= (\text{igaz or hamis}) \text{ or hamis} = \text{igaz or hamis} = \text{igaz}$
- B. $(\text{not hamis or igaz}) \text{ or } (\text{hamis and hamis}) = (\text{igaz or igaz}) \text{ or hamis} =$
 $= \text{igaz or hamis} = \text{igaz}$
- C. $(\text{not igaz or hamis}) \text{ or } (\text{hamis and igaz}) = (\text{hamis or hamis}) \text{ or hamis} =$
 $= \text{hamis or hamis} = \text{hamis}$

- D. (not hamis or igaz) or (igaz and hamis) = (igaz or igaz) or hamis =
= igaz or hamis = igaz
- E. (not igaz or igaz) or (hamis and igaz) = (hamis or igaz) or hamis =
= igaz or hamis = igaz

Tehát a kifejezés igaz az A., B., D. és E. esetekben.

A.4. Mit fog kiírni? (5p)

Legyen a következő program:

C	C++
<pre>#include <stdio.h> int sum(int n, int a[], int* s){ *s = 0; int i = 1; while(i <= n){ if(a[i] != 0) *s += a[i]; ++i; } return *s; } int main(){ int n = 3; int p = 0; int a[10]; a[1] = -1; a[2] = 0; a[3] = 3; int s = sum(n, a, &p); printf("%d;%d", s, p); return 0; }</pre>	<pre>#include <iostream> using namespace std; int sum(int n, int a[], int& s){ s = 0; int i = 1; while(i <= n){ if(a[i] != 0) s += a[i]; ++i; } return s; } int main(){ int n = 3; int p = 0; int a[10]; a[1] = -1; a[2] = 0; a[3] = 3; int s = sum(n, a, p); cout << s << ";" << p; return 0; }</pre>
Pascal	
<pre>type vektor = array[1..10] of integer; function sum(n: integer; a: vektor; var s: integer): integer; var i: integer; begin s := 0; i := 1; while(i <= n) do begin if(a[i] <> 0) then s := s + a[i]; i := i + 1; end; sum := s; end; var n, p, s : integer; a : vektor; begin n := 3; p := 0; a[1] := -1; a[2] := 0; a[3] := 3; s := sum(n, a, p); write(s, ';', p); end.</pre>	

A végrehajtás eredményeként mit fog kiírni a program?

- A. 3;0 B. 2;0 C. 0;2 D. 2;2 E. Egyik válasz sem helyes

Megoldás

Mivel a sorozatnak csak három eleme van, az ellenőrzés könnyű. A program összeadja a sorozat elemeit (a 0 értékűt kivéve). Az összeg egyenlő 2-vel, így ezt írja ki a program és a p paraméter megváltozott értékét (2), mivel *cím* szerint átadott paraméter volt. Helyes válasz: **D**.

A.5. Szerencsés szám (5p)

Egy nullától különböző x természetes számot *szerencsésnek* nevezzük, ha a négyzete felírható x darab egymás utáni természetes szám összegeként. Például, a 7 szerencsés szám, mivel $7^2 = 4 + 5 + 6 + 7 + 8 + 9 + 10$.

A következő algoritmusok közül, melyik dönti el az x természetes számról ($2 \leq x \leq 1000$), hogy *szerencsés szám*?

Minden algoritmus bemeneti paramétere az x szám, kimeneti paraméterei pedig a nullától különböző *start* természetes szám és a *szerencsés* logikai változó. Ha az x szerencsés szám, akkor *szerencsés* = *igaz* és *start* értéke az összeg első tagjának értéke (például, ha $x = 7$, akkor *start* = 4); ha x nem szerencsés szám, akkor *szerencsés* = *hamis* és *start* értéke -1.

A.	<p>Algoritmus szerencsésSzám(x, $start$, <i>szerencsés</i>):</p> <pre> xNégyzet $\leftarrow x * x$ szerencsés $\leftarrow hamis$ start $\leftarrow -1$, $k \leftarrow 1$, $s \leftarrow 0$ Amíg $k \leq xNégyzet - x$ és nem szerencsés végezd el Minden $i = k$, $k + x - 1$ végezd el $s \leftarrow s + i$ vége(minden) Ha $s = xNégyzet$ akkor szerencsés $\leftarrow igaz$ start $\leftarrow k$ vége(ha) vége(amíg) Vége(algoritmus)</pre>
B.	<p>Algoritmus szerencsésSzám(x, $start$, <i>szerencsés</i>):</p> <pre> xNégyzet $\leftarrow x * x$ szerencsés $\leftarrow hamis$ start $\leftarrow -1$, $k \leftarrow 1$ Amíg $k \leq xNégyzet - x$ és nem szerencsés végezd el $s \leftarrow 0$ Minden $i = k$, $k + x - 1$ végezd el $s \leftarrow s + i$ vége(minden) Ha $s = xNégyzet$ akkor szerencsés $\leftarrow igaz$ start $\leftarrow k$ vége(ha) $k \leftarrow k + 1$ vége(amíg) Vége(algoritmus)</pre>

C.	Algoritmus szerencsésSzám(x, start, szerencsés): Ha $x \bmod 2 = 0$ akkor szerencsés \leftarrow hamis start $\leftarrow -1$ különben szerencsés \leftarrow igaz start $\leftarrow (x + 1) \text{ DIV } 2$ vége(ha) Vége(algoritmus)
D.	Algoritmus szerencsésSzám(x, start, szerencsés): Ha $x \bmod 2 = 0$ akkor szerencsés \leftarrow hamis start $\leftarrow -1$ különben szerencsés \leftarrow igaz start $\leftarrow x \text{ DIV } 2$ vége(ha) Vége(algoritmus)
E.	Algoritmus szerencsésSzám(x, start, szerencsés): szerencsés \leftarrow igaz start $\leftarrow (x + 1) \text{ DIV } 2$ Vége(algoritmus)

Megoldás

Reménytelen munka lenne mindegyik algoritmust papíron futtatni. Inkább elemezzük és összehasonlítjuk őket. Például, az **A.** és **B.** algoritmusok között két különbséget találunk:

1. az **A.** algoritmusban az s változó a külső **Amíg** előtt kap kezdőértéket, és ezáltal túl sok számot fog összeadni. A **B.** algoritmusban a kezdőértékadás helyes, így minden négyzet vizsgálatakor az összeg kiszámolása 0 kezdőértéktől indul.
2. Az **A.** algoritmusban a k változó értéke nem változik, így a valós futtatáskor végtelen ciklust eredményezne.

Tehát az **A.** algoritmusról eldöntöttük, hogy hibás. A **B.** algoritmus esetében, főleg a fent említett észrevételek alapján és az algoritmus további utasításainak figyelmes ellenőrzése után, kijelenthetjük, hogy helyes.

Összehasonlítjuk a **C.** és **D.** algoritmusokat. Látjuk, hogy mindkét algoritmus mindenekelőtt kizárja a páros számokat. Vajon miért nem lehet *szerencsés* egy páros szám? Felírjuk x^2 -et összeg formájában, a feladat követelményeinek megfelelően: $x^2 = (k + 1) + (k + 2) + \dots + (k + x)$. (Az utolsó tag azért $k + x$, mert pontosan x tagból kellene előállítanunk az x^2 -et.)

Alkalmazzuk az első n természetes szám összegének képletét:

$$x^2 = \frac{(k+x)*(k+x+1)}{2} - \frac{k*(k+1)}{2}. \text{ Következik, hogy } x^2 = \frac{2*k*x + x^2 + x}{2} \text{ vagyis}$$

$2 * x^2 = 2 * k * x + x^2 + x$, amit, ha elosztunk x -szel: $2 * x = 2 * k + x + 1$, ahonnan $x = 2 * k + 1$, vagyis x -nek páratlannak kell lennie. Tehát, a páros számokat, valóban ki lehet zárni.

A **C.** és **D.** algoritmusok között az egyetlen különbség a **start** változó értékének kiszámításában fedezhető fel. A **C.** algoritmus a **start** változónak az $(x + 1) \text{ DIV } 2$ értéket adja, míg a **D.** algoritmus az $x \text{ DIV } 2$ értéket. Tekintve, hogy x páratlan, ez azt jelenti, hogy a **C.** algoritmus x 2-vel végzett osztási hányadosát felfele, míg a **D.** algoritmus lefele kerekíti. A feladat szövegében láttuk a példát: ha $x = 7$, **start** = 4, vagyis felfele kerekítünk, tehát a **C.** algoritmus a helyes.

Az **E.** algoritmus nem zárja ki a páros számokat, tehát nem helyes.

A.6. Tegyel 'b' betűket (5p)

Legyen az $n \times n$ méretű négyzetes **mat** tömb (n – páratlan természetes szám, $3 \leq n \leq 100$). A **tegyélB(mat, n, i, j)** algoritmus 'b' betűket tesz a **mat** tömb bizonyos pozícióira. Az i és j paraméterek természetes számok ($1 \leq i \leq n, 1 \leq j \leq n$).

```

Algoritmus tegyélB(mat, n, i, j):
  Ha  $i \leq n \text{ DIV } 2$  akkor
    Ha  $j \leq n - i$  akkor
       $\text{mat}[i][j] \leftarrow 'b'$ 
       $\text{tegyélB}(\text{mat}, n, i, j + 1)$ 
    különben
       $\text{tegyélB}(\text{mat}, n, i + 1, i + 2)$ 
  vége(ha)
vége(ha)
Vége(algoritmus)

```

Határozzátok meg, hányszor hívja meg önmagát a **tegyélB(mat, n, i, j)** algoritmus a következő programrészlet végrehajtásának következtében:

```

n ← 7
i ← 2
j ← 4
tegyélB(mat, n, i, j)

```

A. 5-ször

C. 10-szer

D. 0-szor

E. végtelenszer

B. ugyanannyiszor, mint a mellékelt programrészlet esetében:

```

n ← 9, i ← 3, j ← 5
tegyélB(mat, n, i, j)

```

Megoldás

Adott egy rekurzív alprogram, és meg kell állapítanunk, hogy a paraméterek adott értékeire hányszor hívja meg önmagát. A megoldáshoz követnünk kell a paraméterek értékeit a hívássorozaton belül.

$\text{tegyélB}(\text{mat}, 7, 2, 4) \Rightarrow$ (1. hívás):
 $\text{tegyélB}(\text{mat}, 7, 2, 5) \Rightarrow$ (2. hívás):
 $\text{tegyélB}(\text{mat}, 7, 2, 6) \Rightarrow$ (3. hívás):
 $\text{tegyélB}(\text{mat}, 7, 3, 4) \Rightarrow$ (4. hívás):
 $\text{tegyélB}(\text{mat}, 7, 3, 5) \Rightarrow$ (5. hívás):
 $\text{tegyélB}(\text{mat}, 7, 4, 5)$: vége.

		Hívás sorszáma
5	<i>j</i>	5
4	<i>i</i>	
5	<i>j</i>	4
3	<i>i</i>	
4	<i>j</i>	3
3	<i>i</i>	
6	<i>j</i>	2
2	<i>i</i>	
5	<i>j</i>	1
2	<i>i</i>	
4	<i>j</i>	0 (első hívás)
2	<i>i</i>	

Tehát, a rekurzív hívások száma 5, vagyis az **A.** válasz jó, így azt is tudjuk bizonyosan, hogy a **C.**, **D.** és **E.** válaszok nem jók. Megvizsgáljuk a fent leírt módon a **B.** esetet is, és megállapítjuk, hogy a rekurzív hívások száma szintén 5, tehát a **B.** válasz is jó.

B rész (60 pont)

B.1. Számolás - karakterekkel (10 pont)

Legyen a számolásKarakterekkel(s , n , p , q , szám) algoritmus, ahol s egy n karakterből álló sorozat (n természetes szám, $1 \leq n \leq 9$), p , q és *szám* természetes számok ($1 \leq p \leq n$, $1 \leq q \leq n$, $p \leq q$).

```

Algoritmus számolásKarakterekkel( $s$ ,  $n$ ,  $p$ ,  $q$ , szám):
    eredmény  $\leftarrow$  0
     $i \leftarrow p$ 
    Amíg  $i \leq q$  végezd el
        Amíg  $i \leq q$  és  $s[i] \geq '0'$  és  $s[i] \leq '9'$  végezd el
            szám  $\leftarrow$  szám * 10 +  $s[i] - '0'$ 
             $i \leftarrow i + 1$ 
        vége(amíg)
        eredmény  $\leftarrow$  eredmény + szám
        szám  $\leftarrow$  0
         $i \leftarrow i + 1$ 
    vége(amíg)
    térítsd eredmény
Vége(algoritmus)

```

Írjátok le a számolásKarakterekkel(s , n , p , q , szám) algoritmus *rekurzív* változatát úgy, hogy a fejléce és a hatása legyen azonos a fenti algoritmuséval. Az alábbi programrészletből hívjuk meg:

Beolvas: n , s , p , q
Kiír: számolásKarakterekkel(s , n , p , q , 0)

Megoldás

Az értékelés szempontjai:

- A rekurzív algoritmus fejléce azonos-e az iteratív fejlécével? (2 pont)
- A rekurzív hívások leállási feltétele helyes-e? (1 pont)
- A leálláskor térített érték helyes-e? (1 pont)
- Helyesen vizsgálja, hogy a karakter nem számjegy? (2 pont)
- Helyesen határozza meg a nem számjegy karakter értékét? (2 pont)
- Helyesen határozza meg a számjegy karakter értékét? (2 pont)

Egy helyes rekurzív algoritmus:

```

Algoritmus számolásKarakterekkel(s, n, p, q, szám):
    Ha p > q akkor
        térítsd szám
    különben
        Ha s[p] < '0' vagy s[p] > '9' akkor
            térítsd szám + számolásKarakterekkel(s, n, p + 1, q, 0)
        különben
            térítsd számolásKarakterekkel(s, n, p + 1, q, 10 * szám + s[p] - '0')
        vége(ha)
    vége(ha)
Vége(algoritmus)

```

B.2. Periódus (25 pont)

Azt mondjuk, hogy egy n karakterből álló sorozatnak a *periódusa* k , ha az adott sorozat előállítható egy k elemű karaktersorozat ismételt egymás után ragasztása révén ($2 \leq n \leq 200$, $1 \leq k \leq 100$, $2 * k \leq n$). Az "abcabcabcabc" sorozatnak a *periódusa* 3, mivel előállítható úgy, hogy az "abc" karaktersorozatot 4-szer egymás után ragasztjuk; ugyanakkor a sorozatnak a *periódusa* 6, ha úgy tekintjük, hogy az "abcabc" karaktersorozatot 2-szer egymás után ragasztjuk. Az "abcxabc" sorozatnak nincs periódusa. *Maximális periódus*nak nevezzük a sorozat legnagyobb *periódusát*.

Írjatok algoritmust, amely meghatározza az n elemű x karaktersorozat (n – természetes szám, $2 \leq n \leq 100$) *pm* *maximális periódusát*. Ha az x sorozatnak nincs periódusa, *pm* értéke -1. Az algoritmus bemeneti paraméterei x és n , kimeneti paramétere *pm*.

1. Példa: ha $n = 8$ és $x = \text{"abababab"}$, akkor *pm* = 4.

2. Példa: ha $n = 7$ és $x = \text{"abcxabc"}$, akkor *pm* = -1.

Megoldás

Az értékelés szempontjai:

- A bemeneti és kimeneti paraméterek megfelelnek a követelményekben leírtaknak? (2 pont)
- Fölhasználja-e az észrevételt, miszerint a periódus hosszát csak n osztói között érdemes keresni? (legtöbb 10 pont)
- Helyesen ellenőrzi a periodicitást? (legtöbb 13 pont, az alkalmazott ismétlődő struktúrák számától függően)

```

bool ok(int n, char const x[], int periodus){
    for (int i = periodus; i < n; ++i) {
        if (x[i + 1] != x[i % periodus + 1])
            return false;
    }
    return true;
}

```

```

int periodusMax(int n, char x[]){
    int periodus = -1;
    for (int per = 2; per * per <= n; ++per){
        if (n % per == 0){ // a periódus hosszát n osztói között keressük
            if (ok(n, x, n / per))
                return n / per;
            if ((per * per < n) && ok(n, x, per))
                periodus = per;
        }
    }
    return periodus;
}

```

B.3. Robi-kert (25 pont)

Egy modern műszaki megoldásokat kedvelő kertész elhatározta, hogy a kert ágyásainak öntözéséhez egy robotokból álló „hadsereget” fog használni. A vizet a kertet átszelő főcső végénél található forrásból szeretné venni. Minden ágyáshoz kioszt egy robotot úgy, hogy minden robotnak egyetlen ágyást kell megöntöznie. Minden robot egyszerre indul öntözni a forrástól reggel ugyanabban az időpontban (például, reggel 5:00:00 órakor) és párhuzamosan dolgoznak, megállás nélkül azonos időn át. A robotok haladnak a főcsőn, amíg a saját ágyásukhoz érnek, az ágyást megöntözik és visszatérnek a forráshoz, hogy a víztartályukat újból megtöltsék. A tevékenységre szánt idő lejártá után, minden robot egyszerre leáll, függetlenül attól, hogy mely állapotban vannak. Eredetileg, a forrásnál egyetlen vízcsap volt. A kertész észrevette, hogy öntözés közben késségek adódtak, mivel a robotoknak sorba kellett állniuk a vízcsapnál, hogy feltölthessék a víztartályukat. Ebből kifolyólag úgy döntött, hogy fel fog szerelni több vízcsapot. A robotok reggel tele víztartállyal indulnak. Két robot nem töltheti fel a víztartályát ugyanabban a pillanatban egyazon vízcsapnál.

Ismert adatok: a t időintervallum (másodpercekben) amennyi ideig az n robot dolgozik, a d_i a másodpercek száma, amennyi idő alatt a robotok eljutnak a forrástól a számukra kiosztott ágyásig, az u_i a másodpercek száma, amennyi idő alatt a robotok megöntözik a saját ágyásukat. Még ismert, hogy mindegyik robot a saját víztartályát egy másodperc alatt tölti meg. (t, n, d_i, u_i – természetes számok, $1 \leq t \leq 20000, 1 \leq n \leq 100, 1 \leq d_i \leq 1000, 1 \leq u_i \leq 1000, i = 1, 2, \dots, n$).

Követelmények:

- a. Soroljátok fel azokat a robotokat, amelyek találkoznak a forrásnál egy adott mt időpillanatban ($1 \leq mt \leq t$). Indokoljátok meg a választ (a robotokat a sorszámaik azonosítják)

- b. Legkevesebb hány ***minÚjVízcsap*** új vízcsapot kell felszerelnie a kertésznek ahhoz, hogy a robotoknak egyáltalán ne kelljen várakozniuk egymás után, amikor meg kell tölteniük a víztartályukat? Indokoljátok meg a választ.
- c. Írjatok algoritmust, amely meghatározza, hogy legkevesebb hány ***minÚjVízcsap*** újabb vízcsapot kell még felszerelnie a kertésznek. Bemeneti paraméterek n , t , valamint a d és u sorozatok, mindkettő n elemmel, kimeneti paraméter a ***minÚjVízcsap***.

1. Példa: ha $t = 32$ és $n = 5$, $d = (1, 2, 1, 2, 1)$, $u = (1, 3, 2, 1, 3)$, akkor ***minÚjVízcsap*** = 3.

Magyarázat: az első ágyással foglalkozó robotnak szüksége van egy másodpercre, hogy az ágyáshoz érjen, egy másodpercre az öntözéshez és még egy másodpercre ahhoz, hogy visszatérjen a forráshoz; így, ez a robot $1 + 1 + 1 = 3$ másodperc után tér vissza a forráshoz, hogy újra megtöltse a tartályát az indulási időponttól számítva (5:00:00), tehát 5:00:03-kor; megtölti a víztartályát egy másodperc alatt és 5:00:04-kor indul vissza az ágyáshoz; visszatér 5:00:07-kor a forráshoz, és így tovább; tehát az első, a második, a negyedik és az ötödik robot találkoznak a forrásnál 05:00:23-kor; következik, hogy 3 új vízcsapra van szükség.

2. Példa:

ha $t = 37$, $n = 3$, $d = (1, 2, 1)$, $u = (1, 3, 2)$, akkor ***minÚjVízcsapok*** = 1.

Megoldás

Elvégezzük a következő előfeldolgozást egy példa esetében:

Ha $n = 5$, $d = (1, 2, 1, 2, 1)$, $u = (1, 3, 2, 1, 3)$ és $t = 32$, kiszámítjuk és tároljuk egy tömbben azokat a q értékeket, amelyek minden robot esetében azt az időt jelentik, amely szükséges ahhoz, hogy a robot az ágyáshoz érjen, onnan visszatérjen, megöntözze az ágyást és megtöltse a tartályt.

1. robot	1. robot	1. robot	1. robot	1. robot
$2*1 + 1 + 1 = 4$	$2*2 + 3 + 1 = 8$	$2*1 + 2 + 1 = 5$	$2*2 + 1 + 1 = 6$	$2*1 + 3 + 1 = 6$

Így a q számok tömbje: (4, 8, 5, 6, 6)

Minden létező q esetében létrehozunk egy v sorozatot (a példánk esetében) $t = 32$ elemmel. Vesszük sorra az előfordult q értékeket és a v sorozat azon elemének értékét növeljük 1-gyel, amelynek indexe az adott q többszöröse

Amikor $q = 4$ (első robot számára szükséges munkaidő):

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
v	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0

	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
v	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1

Amikor $q = 8$ (a második robot számára szükséges munkaidő):

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
v	0	0	0	1	0	0	0	2	0	0	0	1	0	0	0	2	0

	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
v	0	0	1	0	0	0	2	0	0	0	1	0	0	0	2

Amikor $q = 5$ (a harmadik robot számára szükséges munkaidő):

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
v	0	0	0	1	1	0	0	2	0	1	0	1	0	0	1	2	0

	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
v	0	0	2	0	0	0	2	1	0	0	1	0	1	0	2

Amikor $q = 6$ (a negyedik robot számára szükséges munkaidő):

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
v	0	0	0	1	1	1	0	2	0	1	0	2	0	0	1	2	0

	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
v	1	0	2	0	0	0	3	1	0	0	1	0	2	0	2

Amikor $q = 6$ (az ötödik robot számára szükséges munkaidő):

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
v	0	0	0	1	1	2	0	2	0	1	0	3	0	0	1	2	0

	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
v	2	0	2	0	0	0	4	1	0	0	1	0	3	0	2

A következőkben meghatározzuk a v tömb maximumát. A példánk esetében ez 4 (a 24. időpillanatban), tehát szükséges még $4 - 1 = 3$ új vízcsap.

Most válaszolunk a feladat kijelentésében megfogalmazott kérdésekre:

- Azok a robotok találkoznak a forrásnál egy adott mt pillanatban ($1 \leq mt \leq t$), amelyeknek a megfelelő q értékük mt osztója. (5 pont)
- A legkevesebb új vízcsap száma, amit a kertésznek fel kell szerelnie egyenlő a v tömb maximumával, amiből kivonunk 1-et (a már létező vízcsapot). A v tömb elemeinek értéke azoknak a robotoknak a száma, amelyek az egyes időpillanatokban találkoznak a forrásnál. (5 pont)

Több lehetséges helyes algoritmusból bemutatunk kettőt.

V1: Létrehozunk egy gyakoriságtömböt, amelyben a munkaidők többszöröseit tároljuk, majd itt megkeressük a maximumot. (15 pont)

Az értékelés szempontjai:

- A bemeneti és kimeneti paraméterek megfelelnek a követelményekben leírtaknak? (2 pont)
- A munkaidők kiszámítása ($q = 2 * \text{utazás} + \text{öntözés} + \text{újratöltés}$) (2 pont)
- A gyakoriságtömb feldolgozása (5 pont)
- A maximum meghatározása (4 pont)
- A felszerelendő új vízcsapok számának meghatározása (2 pont)

```
int robikert_1(int n, int d[], int u[], int t){
    int v[200000]; //v[i] = az i. időpillanatban szükséges vízcsapok száma
    int max = 1;
    for (int i = 1; i <= t; i++){
        v[i] = 0;
        for (int j = 1; j <= n; j++){
            int q = d[j] * 2 + u[j] + 1;
            for (int i = q; i <= t; i = i + q){
                v[i]++;
                if (max < v[i])
                    max = v[i];
            }
        }
    }
    return max - 1;
}
```

V2: A második algoritmusban szimuláljuk a robotok munkáját. (10 pont)

```
int robikert_2(int n, int d[], int u[], int t){
    int minVizcsap = 1;
    int aktIdo = 1;
    while (aktIdo <= t){
        int aktVizcsap = 0;
        for (int j = 1; j <= n; j++){
            int q = 2 * d[j] + u[j] + 1;
            if (aktIdo % q == 0)
                // ha az aktIdo időpillanatban az i. robot a forrásnál van
                aktVizcsap++;
        }
        if (aktVizcsap > minVizcsap)
            minVizcsap = aktVizcsap;
        aktIdo++;
    }
    return minVizcsap - 1;
}
```


4.4. Felvételi verseny – 2018. szeptember 12.

A rész (30 pont)

A.1. Vajon mit csinál? (5p)

A generál(n) algoritmus egy n természetes számot dolgoz fel ($0 < n < 100$).

```
Algoritmus generál( $n$ ):  
    szám  $\leftarrow 0$   
    Minden  $i \leftarrow 1, 1801$  végezd el  
        használt[ $i$ ]  $\leftarrow hamis$   
    vége(minden)  
    Amíg nem használt[ $n$ ] végezd el  
        összeg  $\leftarrow 0$ ;  
        használt[ $n$ ]  $\leftarrow igaz$   
        Amíg  $n \neq 0$  végezd el  
            számjegy  $\leftarrow n \text{ MOD } 10$   
            összeg  $\leftarrow összeg + \text{számjegy} * \text{számjegy} * \text{számjegy}$   
             $n \leftarrow n \text{ DIV } 10$   
        vége(amíg)  
         $n \leftarrow összeg$   
        szám  $\leftarrow szám + 1$   
    vége(amíg)  
    térítsd szám  
Vége(algoritmus)
```

Állapítsátok meg a fenti algoritmus hatását.

- A. ismételten kiszámítja az n szám számjegyei köbének összegét ameddig az összeg egyenlővé válik az n számmal és visszatéríti a végrehajtott ismétlések számát
- B. kiszámítja az n szám számjegyei köbének összegét és visszatéríti ezt az összeget
- C. kiszámítja az n szám számjegyei köbének összegét, felülírja n értékét ezzel az összeggel, és visszatéríti ezt az összeget
- D. ismételten kiszámítja az n szám számjegyei köbének összegét ameddig egy összeget másodsorra kap meg, és visszatéríti a végrehajtott ismétlések számát
- E. meghatározza, hogy hányszor lesz felülírva az n szám a számjegyei köbének összegével, ameddig egy már kiszámolt értéket vagy magát a számot kapja, és visszatéríti ezt a számot

Megoldás

Az algoritmus elemzése során „tudomásul vesszük”, hogy a generált számokat az előfordulásokat tároló tömbben tartja nyilván az algoritmus. Egy kissé furcsának tűnhet a tömb mérete, de valószínűnek tartjuk, hogy a 100-nál kisebb számok esetében létezik legalább egy, ahol a számjegyek köbének összege pontosan 1801. Az előfordulások tömbjében az n -edik elem *igaz* értéket kap, ha a generálás során megjelenik az n szám. Mivel az első érték, amelyet nyilvántartásba vesz az algoritmus, maga az adott n szám, amelyet később felülír az aktuális n érték számjegyei köbének összegével, könnyű belátni, hogy az **E.** válasz helyes.

Sorra vesszük a többi választ: az **A.** nem lehet jó, mert abból a meghatározásból, hogy „ismételten kiszámítja az n szám számjegyei köbének összegét ameddig az összeg egyenlővé válik az n számmal” úgy tűnik, hogy n nem változik. Ugyanakkor az algoritmus tartalmazza az $n \leftarrow \text{összeg}$ utasítást, tehát n értéke felülíródik, vagyis változik.

A **B.** és **C.** válaszok biztos nem jók: az algoritmus nem az összeget téríti.

A **D.** válasz „majdnem jó”, de nem tesz említést az adott számról, tehát hiányos, így nem tekinthető helyesnek.

A.2. Mely értékekre van szükség? (5p)

Adott a $\text{feldolgoz}(v, k)$ algoritmus, ahol v egy k elemű, természetes számokat tároló sorozat ($1 \leq k \leq 1\,000$).

```

Algoritmus feldolgoz(v, k)
  i ← 1; n ← 0
  Amíg i ≤ k és v[i] ≠ 0 végezd el
    y ← v[i]; c ← 0
    Amíg y > 0 végezd el
      Ha y MOD 10 > c akkor
        c ← y MOD 10
      vége(ha)
    y ← y DIV 10
  vége(amíg)
  n ← n * 10 + c
  i ← i + 1
vége(amíg)
térítsd n
Vége(algoritmus)

```

Állapítsátok meg, v és k mely értékeire térít vissza az algoritmus 928-at.

- A.** $v = (194, 121, 782, 0)$ és $k = 4$
- B.** $v = (928)$ és $k = 1$
- C.** $v = (9, 2, 8, 0)$ és $k = 4$
- D.** $v = (8, 2, 9)$ és $k = 3$
- E.** $v = (912, 0, 120, 8, 0)$ és $k = 5$

Megoldás

Fárasztó és időigényes munka lenne papíron futtatni a programot. Táblázatkészítés helyett inkább elemezzük az algoritmust. A k elemű v sorozat feldolgozása akkor ér véget, ha az algoritmus bejárta mind a k elemet, vagy akkor, ha a feldolgozás során 0 értékű elem következne. Az algoritmus az aktuális elem értékét számjegyekre bontja, kiválasztja közülük a maximális értékűt és ezt a számjegyet az n számba építi balról jobbra haladva.

Ha már világos az algoritmus funkciója, könnyű kiválasztani a megfelelő bemeneti adatokat, amelyeknek feldolgozása 928-at eredményez. Ezek az **A.** és **C.** pontokban megadott adatok. Az **A.** esetben a sorozat (194, 121, 782, 0), és a legnagyobb számjegyeket a sorozat elemeinek sorrendjében, balról jobbra haladva tartalmazó szám értéke pontosan 928. A **C.** pontban a sorozat a 928 számjegyeit tartalmazza, tehát jó eredményt kapunk.

A **B.** esetben a sorozat egyetlen számot tartalmaz, ennek maximális számjegye 9, amelyből csak a 9-es számot lehet felépíteni, tehát nem jutunk el a 928-hoz.

A **D.** esetben az algoritmus a 829-es számot építi fel, az **E.** esetben a feldolgozás leáll az első szám után, mivel a második szám értéke 0.

Tehát a helyes válaszok: **A.** és **C.**

A.3. Logikai kifejezés kiértékelése (5p)

Adott a k elemű s sorozat, amelynek elemei logikai (boolean) típusúak és a kiértékelés(s , k , i) algoritmus, ahol k és i ($0 \leq i \leq k \leq 100$) természetes számok.

```

Algoritmus kiértékelés( $s$ ,  $k$ ,  $i$ )
  Ha  $i \leq k$  akkor
    Ha  $s[i]$  akkor
      térítsd  $s[i]$ 
    különben
      térítsd ( $s[i]$  vagy kiértékelés( $s$ ,  $k$ ,  $i + 1$ ))
    vége(ha)
  különben
    térítsd hamis
  vége(ha)
Vége(algoritmus)

```

Határozzátok meg, hányszor hívja meg önmagát a kiértékelés(s , k , i) algoritmus a következő programrészlet végrehajtásának következtében:

```

 $s \leftarrow$  (hamis, hamis, hamis, hamis, hamis, hamis, igaz, hamis, hamis, hamis)
 $k \leftarrow 10$ 
 $i \leftarrow 3$ 
kiértékelés( $s$ ,  $k$ ,  $i$ )

```

A. 3-szor

B. ugyanannyiszor, mint a következő programrészlet esetében

```

 $s \leftarrow$  (hamis, hamis, hamis, hamis, hamis, hamis, hamis, igaz)
 $k \leftarrow 8$ 
 $i \leftarrow 4$ 
kiértékelés( $s$ ,  $k$ ,  $i$ )

```

C. 6-szor

D. egyszer sem

E. végtelenszer

Megoldás

Adott egy rekurzív alprogram, és meg kell állapítanunk, hogy a paraméterek adott értékeire hányszor hívja meg önmagát. A megoldáshoz követnünk kell a paraméterek értékeit a hívássorozaton belül, – esetleg lerajzoljuk a végrehajtási verem tartalmát és megszámloljuk a rekurzív hívásokat:

kiértékelés($s, 10, 3$) \Rightarrow (1. hívás):

kiértékelés($s, 10, 4$) \Rightarrow (2. hívás):

kiértékelés($s, 10, 5$) \Rightarrow (3. hívás):

kiértékelés($s, 10, 6$) \Rightarrow (4. hívás):

kiértékelés($s, 10, 7$): vége.

		Hívás sorszáma
7	<i>i</i>	4
10	<i>k</i>	
6	<i>i</i>	3
10	<i>k</i>	
5	<i>i</i>	2
10	<i>k</i>	
4	<i>i</i>	1
10	<i>k</i>	
3	<i>i</i>	0 (első hívás)
10	<i>k</i>	

Tehát, a rekurzív hívások száma 4, vagyis nem 3, tehát az **A.**, **C.**, **D.** és **E.** válaszok nem jók. De maradt még egy válasz, amelyről még nem tudjuk, hogy helyes vagy sem. Megvizsgáljuk a fent leírt módon a **B.** esetet is, és megállapítjuk, hogy a rekurzív hívások száma 4, vagyis ugyanannyi, mint az **A.** pontban érvényes adatok esetében. Tehát a **B.** válasz jó.

A.4. Egyesítés (5p)

Adottnak tekintjük az $\text{eleme}(x, a, n)$ algoritmust, amely eldönti, hogy az x természetes szám eleme-e az n elemű a halmaznak; a egy n elemű sorozat, amely egy természetes számokat tartalmazó halmazt ábrázol ($1 \leq n \leq 200, 1 \leq x \leq 1000$).

Legyen az alább megadott egyesítés(a, n, b, m, c, p) algoritmus, ahol a , b és c sorozatok, amelyek természetes számokat tároló és rendre n , m és p elemű halmazokat ábrázolnak ($1 \leq n \leq 200, 1 \leq m \leq 200, 1 \leq p \leq 400$). A bemeneti paraméterek a, n, b és m , kimeneti paraméterek pedig c és p .

```

1.  Algoritmus egyesítés( $a, n, b, m, c, p$ ):
2.      Ha  $n = 0$  akkor
3.          Minden  $i = 1, m$  végezd el
4.               $p \leftarrow p + 1$ 
5.               $c[p] \leftarrow b[i]$ 
6.          vége(minden)
7.      különben
8.          Ha nem  $\text{eleme}(a[n], b, m)$  akkor
9.               $p \leftarrow p + 1$ 
10.              $c[p] \leftarrow a[n]$ 
11.          vége(ha)
12.          egyesítés( $a, n - 1, b, m, c, p$ )
13.      vége(ha)
14.  Vége(algoritmus)
```

A következő állítások közül melyek bizonyulnak mindig igaznak?

- A. ha az a halmaz egy elemet tartalmaz, az egyesítés(a, n, b, m, c, p) algoritmus végtelen ciklusba kerül
- B. ha az a halmaznak négy eleme van, az egyesítés(a, n, b, m, c, p) algoritmus első meghívása maga után vonja a 12. sorban található utasítás végrehajtását négyszer
- C. ha az a halmaznak öt eleme van, az egyesítés(a, n, b, m, c, p) algoritmus első meghívása maga után vonja a második sorban található utasítás végrehajtását ötször
- D. ha az a halmaznak ugyanannyi eleme van, mint a b halmaznak, az egyesítés(a, n, b, m, c, p) algoritmus végrehajtása után a c halmaznak ugyanannyi eleme lesz, mint az a halmaznak
- E. ha az a és b halmazok elemei azonosak, az egyesítés(a, n, b, m, c, p) algoritmus végrehajtása után a c halmaznak ugyanannyi eleme lesz, mint az a halmaznak

Megoldás

Az A. válasz nem lehet helyes. Ha az a halmaznak egy eleme van, $n = 1$ (vagyis nem 0) az algoritmus előbb eldönti, hogy ez az elem megtalálható-e a b halmazban. Ha a_1 értéke nem eleme a b halmaznak, elhelyezi a_1 -et a c halmazba, majd meghívja önmagát $n = 0$ -ra, különben csak a rekurzív hívást hajtja végre. A következő végrehajtáskor megtörténik a kezdőértékdadás (az algoritmus bemásolja a b halmaz elemeit a c halmazba), kilép az aktuális hívásból és visszatér a hívás helyére. Ezúttal kilép az algoritmusból és visszatér abba a programegységbe, amely meghívta eredetileg. Tehát nem kerül végtelen ciklusba.

A B. állítást egyszerű ellenőrizni, mivel a 12. sorban a rekurzív hívás található, amely pontosan annyszor kerül végrehajtásra, ahány eleme van az a halmaznak. Tehát a B. válasz helyes.

A C. pontban leírtaknak megfelelően az n változó értékének 0-val történő összehasonlításainak száma egyenlő n -nel. Ez hibás állítás, mivel az összehasonlítást előbb az eredeti n -re, majd $(n - 1)$ -re stb., végül 0-ra végzi az algoritmus (például, ha $n = 5$, a második sorban található utasítást 6-szor hajtja végre az algoritmus).

A D. feltételezés sem lehet helyes. Ha például, az a halmaznak és a b -nek is két eleme van: $a = (1, 2)$ és $b = (3, 4)$, az egyesítés eredménye $c = (1, 2, 3, 4)$, vagyis négy elem, míg az a halmaznak két eleme van.

Az E. feltételezés helyes, hiszen egy érték csak egyszer szerepelhet az egyesített halmazban. Ha például, $a = (1, 2)$ és $b = (1, 2)$, az egyesítés eredménye $c = (1, 2)$.

A.5. Hatványra emelés (5p)

Melyik alábbi algoritmus számítja ki helyesen a^b értékét, ahol a és b természetes számok ($1 \leq a \leq 11, 0 \leq b \leq 11$)?

A. Algoritmus hatvány(a, b): eredmény \leftarrow 1 Amíg $b > 0$ végezd el Ha $b \bmod 2 = 1$ akkor eredmény \leftarrow eredmény * a vége(ha) $b \leftarrow b \text{ DIV } 2$ $a \leftarrow a * a$ vége(amíg) térítsd eredmény Vége(algoritmus)	B. Algoritmus hatvány(a, b): Ha $b \neq 0$ akkor Ha $b \bmod 2 = 1$ akkor térítsd hatvány($a * a, b / 2$) * a különben térítsd hatvány($a * a, b / 2$) vége(ha) különben térítsd 1 vége(ha) Vége(algoritmus)
C. Algoritmus hatvány(a, b): eredmény \leftarrow 1 Amíg $b > 0$ végezd el eredmény \leftarrow eredmény * a $b \leftarrow b - 1$ vége(amíg) térítsd eredmény Vége(algoritmus)	D. Algoritmus hatvány(a, b): Ha $b = 0$ akkor térítsd 1 vége(ha) aux \leftarrow hatvány(a, $b \text{ DIV } 2$) Ha $b \bmod 2 = 0$ akkor térítsd aux * aux különben térítsd $a * \text{aux} * \text{aux}$ vége(ha) Vége(algoritmus)
E. Algoritmus hatvány(a, b): Ha $b = 0$ akkor térítsd 1 vége(ha) térítsd $a * \text{hatvány}(a, b - 1)$ Vége(algoritmus)	

Megoldás

Nagy előnyt jelent, ha ismerjük *al-Kwarizmi*, „gyorshatvány” néven ismert algoritmusát. Ennek iteratív változatát az **A.** pontban látjuk, a rekurzívat pedig a **B.** pontban. Hasonlóan párba állítható a két – hatványt számoló – naiv algoritmus: A **C.** az iteratív, az **E.** a rekurzív változat. Ezek mind helyesek. A **D.** algoritmust összehasonlítjuk a **B.** – szintén rekurzív – algoritmussal és belátjuk, hogy ugyanaz a hatásuk, csak **D.** felhasznál egy segédváltozót, amelybe ideiglenesen elmenti az aktuális hívás által térített értéket. Tehát, mind az öt algoritmus helyes.

A.6. Legnagyobb többszörös (5p)

Melyik alábbi algoritmus téríti az a számnak azt a legnagyobb többszörösét, amely kisebb vagy egyenlő b -vel ($0 < a < 10\,000, 0 < b < 10\,000, a < b$)?

A. Algoritmus $f(a, b)$: $c \leftarrow b$ Amíg $c \bmod a = 0$ végezd el $c \leftarrow c - 1$ vége(amíg) térítsd c Vége(algoritmus)	B. Algoritmus $f(a, b)$: Ha $a < b$ akkor térítsd $f(2 * a, b)$ különben Ha $a = b$ akkor térítsd a különben térítsd b vége(ha) vége(ha) Vége(algoritmus)
C. Algoritmus $f(a, b)$: térítsd $(b \text{ DIV } a) * a$ Vége(algoritmus)	E. Algoritmus $f(a, b)$: $c \leftarrow a$ Amíg $c < b$ végezd el $c \leftarrow c + a$ vége(amíg) Ha $c = b$ akkor térítsd c különben térítsd $c - a$ vége(ha) Vége(algoritmus)
D. Algoritmus $f(a, b)$: Ha $b \bmod a = 0$ akkor térítsd b vége(ha) térítsd $f(a, b-1)$ Vége(algoritmus)	

Megoldás

A legkönnyebb a **C.** algoritmust ellenőrizni. Kiszámoljuk a térített értéket egy-két példára és azonnal belátjuk, hogy helyes.

A **D.** algoritmus elindul a b értékétől. Ha ennek osztója a , máris megvan az eredmény, különben a $b - 1$ értékkel próbálkozik. Az algoritmus helyes.

Az **A.** algoritmus azt tervezi, hogy mindaddig csökkentse c értékét (kezdőértéke b), amíg a c -t maradék nélkül osztja a . De ez lehetetlen, mivel előfordulhat ugyan, hogy az **Amíg** törzse végrehajtódik egyszer, de többször biztos nem. Ha például, egy x számnak van egy y szám osztója, az $x - 1$ számot y már nem osztja maradék nélkül. Így az **A.** választ nem jelöljük meg.

A **B.** algoritmus megpróbálja a többszöröseit generálni, de ezt hibásan végzi. Az $a, 2a, 3a, 4a, 5a, 6a, \dots$ értékek helyett az $a, 2a, 4a, 8a, 16a, 32a \dots$ értékeket generálja. Így kihagy egy sor többszöröst, amelyek között biztos lett volna olyan, amely helyes eredményhez vezetett volna. Tehát a **B.** algoritmus is hibás.

Az **E.** algoritmus addig generálja a többszöröseit, amíg a többszörös kisebb, mint b . Amikor az algoritmus kilép az **Amíg** utasításból a nagyobb vagy egyenlő, mint b , tehát megvizsgálja az egyenlőséget. Amennyiben a többszörös egyenlő b -vel, téríti b értékét, különben (a többszörös b -nél nagyobb) kivon belőle a -t. Az algoritmus helyes.

Tehát összesen három helyes algoritmust találtunk: **C.**-t, **D.**-t, és **E.**-t

B rész (60 pont)**B.1. Polinom értéke (10 pont)**

Adott a kiértékelés(n , egyh, x) algoritmus, ahol **egyh** egy $n + 1$ elemű, valós számokat tároló sorozat, amelynek értékei a $[-100, 100]$ intervallumhoz tartoznak és amelyek az n -ed fokú $P(x) = \text{egy}h_1 * x^n + \text{egy}h_2 * x^{n-1} + \dots + \text{egy}h_n * x + \text{egy}h_{n+1}$ polinom együtthatói, x hatványainak csökkenő sorrendjében (n természetes szám, $1 \leq n \leq 10$). Az algoritmus meghatározza a polinom értékét egy adott x pontban (x valós szám, amely a $[-10, 10]$ intervallumhoz tartozik).

```
Algoritmus kiértékelés( $n$ , egyh,  $x$ ):  
    érték  $\leftarrow 0.0$   
    Minden  $i \leftarrow 1, n + 1$  végezd el  
        érték  $\leftarrow$  érték  $\cdot x + \text{egy}h[i]$   
    vége(minden)  
    térítsd érték  
Vége(algoritmus)
```

Írjátok le a kiértékelés(n , egyh, x) algoritmus *rekurzív* változatát úgy, hogy a fejléce és a hatása legyen azonos a fenti algoritmuséval.

Megoldás

Az algoritmusban felismerjük a *Horner séma* néven ismert híres módszert, amely optimálisan számítja ki a polinom értékét az x érték esetében.

Az értékelés a következő szempontok szerint történt:

- A rekurzív algoritmus fejléce azonos az iteratív fejlécével? (2 pont)
- Rekurzív hívás (2 pont)
 - A rekurzív hívás a megfelelő helyen található? (1 pont)
 - A paraméterezés helyes-e? (1 pont)
- Helyes értéket térít leálláskor? (2 pont)
- A folytatáskor térített érték helyes-e? (2 pont)

Egy lehetséges algoritmus:

```
Algoritmus kiértékelésRek( $n$ , egyh,  $x$ ):  
    Ha  $n = 0$  akkor  
        térítsd egyh[1]  
    különben  
        térítsd egyh[ $n + 1$ ] +  $x * \text{kiértékelésRek}(n - 1, \text{egy}h, x)$   
    vége(ha)  
Vége(algoritmus)
```


B.2. Keresztmetszet (25 pont)

Adott két sorozat, amelyek $-30\,000$ és $30\,000$ közötti egész számokat tárolnak, amelyek az egyes sorozatokon belül *különbözők*. Az **a** sorozatnak n eleme, ($0 < n \leq 10\,000$) a **b** sorozatnak m eleme van ($0 < m \leq 10\,000$), és *növekvően rendezett*.

Írjatok algoritmust, amely meghatározza azt a k elemű ($0 \leq k \leq 10\,000$) **c** sorozatot, amely az adott két sorozat közös elemeit tartalmazza, minden elemet csak egyszer, tetszőleges sorrendben. Az algoritmus bemeneti paraméterei a két sorozat (**a** és **b**) valamint a hosszúságaik (n és m). Kimeneti paraméterek a **c** sorozat és ennek k hosszúsága. Ha nincsenek közös elemek, k értéke 0 .

Példa: ha $n = 4$, $a = (5, -7, -2, 3)$, $m = 5$ és $b = (-2, 3, 5, 7, 8)$, a **c** sorozatnak $k = 3$ eleme van és $c = (5, -2, 3)$.

Megoldás

Adott két sorozat, amelyek nem rendezettek, ugyanakkor halmazokat tárolnak, mivel az elemek különbözők. Természetesen, alkalmazhatnánk a *keresztmetszet programozási tétel* néven ismert algoritmust, de így nem használnánk ki a **b** sorozat tulajdonságát, vagyis azt, hogy *növekvően rendezett*. Ebből azonnal következik, hogy ez a megközelítés nem vezet optimális megoldáshoz.

Mivel több lehetséges és hatékony megközelítés is számításba jöhet, ezek közül bemutatunk kettőt.

V1: A sorozatokat az adott formában dolgozzuk fel és az **a** sorozat aktuális értékét a bináris keresés algoritmusával keressük a **b** sorozatban. Ha megtaláltuk, elhelyezzük a **c** sorozatba (25 pont)

Az értékelés a következő szempontok szerint történt:

- A bemeneti és kimeneti paraméterek megfelelnek a követelményekben leírtaknak? (2 pont)
- Helyesen járja be az **a** sorozatot? (5 pont)
- Hogyan történik az **a** sorozat aktuális elemének megkeresése a **b** sorozatban?
 - szekvenciális kereséssel:
 - A keresés leáll, amint az algoritmus megtalálta a keresett elemet? (8 pont)
 - A keresés nem áll le, amint az algoritmus megtalálta a keresett elemet (4 pont)
 - bináris kereséssel (ha az alábbiak helyesek)
 - Helyesen inicializálja a **bal** és **jobb** indexeket? (2 pont)
 - Az **Amíg** struktúra helyes-e? (2 pont)
 - Helyesen határozza meg a sorozat középső indexét? (2 pont)
 - Helyesen azonosítja be a keresett elemet? (3 pont)
 - Helyesen módosítja a **bal** és **jobb** indexértékeket? (4 pont)
- Helyesen hozza létre a **c** sorozatot? (5 pont)

```

void keresztmetszet_1(int n, int a[], int m, int b[], int &k, int c[]){
    k = 0;
    for (int i = 1; i <= n; i++){
        bool megvan = false;
        int bal = 1; int jobb = m;
        while (!megvan && (bal <= jobb)){
            int kozep = (bal + jobb) / 2;
            if (a[i] == b[kozep]){
                c[++k] = a[i];
                megvan = true;
            }
            else{
                if (a[i] < b[kozep]) jobb = kozep - 1;
                else bal = kozep + 1;
            }
        }
    }
}

```

V2: Előbb rendezzük az **a** sorozatot, majd egy, az összefésüléshez hasonló algoritmust alkalmazunk (legtöbb 22 pont, lineáris rendezés esetében 25 pont)

Az értékelés szempontjai:

- Helyesen rendezi növekvő sorrendbe az **a** sorozatot? (5 pont)
 - hatékony rendezés (például, gyorsrendezés (*quicksort*), összefésülő rendezés (*mergesort*), kupacrendezés (*heapsort*) stb.) (5 pont) Lineáris rendezés (*binsort*) alkalmazásával: 8 pont.
 - kevésbé hatékony rendezés (például buborékredezés (*bubblesort*), beszűrő rendezés (*insertsort*) stb.) (3 pont)
- Helyesen járja be az **a** sorozatot? (5 pont)
- Az **a** sorozat aktuális elemét a **b** sorozatban az összefésülés algoritmusának mintájára keresi meg? (5 pont)
- Helyesen hozza létre a **c** sorozatot? (5 pont)

```

void keresztmetszet_2(int n, int a[], int m, int b[], int &k, int c[]){
    rendez(n, a);
    int i = 1; int j = 1;
    k = 0;
    a[n + 1] = b[m] + 1; b[m + 1] = a[n] + 1;
    while ((i <= n) && (j <= m)){
        if (a[i] < b[j])
            i++;
        else{
            if (a[i] > b[j])
                j++;
            else{
                c[++k] = a[i];
                i++; j++;
            }
        }
    }
}

```

B.3. Tömbszakasz – testvérszámok nélkül (25 pont)

Adott az n elemű ($0 < n \leq 10\,000$) a sorozat, amely 30 000-nél kisebb, nem nulla *különböző* természetes számokat tárol. Két számot *testvéreknek* nevezünk, ha *nem* azonosak és *van* legkevesebb két *különböző* közös számjegyük. Például, 5867 és 17526 *testvérek*, miközben 5867 és 152 *nem testvérek*. Hasonlóképpen, 131 és 114 *nem testvérek*.

Követelmények:

- Írjatok algoritmust, amely eldönti, hogy az a természetes szám testvére-e a b természetes számnak ($0 < a \leq 30\,000$, $0 < b \leq 30\,000$). Az algoritmus bemeneti paraméterei az a és b számok. Kimeneti paraméter a *testvér* logikai változó, amelynek értéke *igaz* ha a testvére b -nek, különben *hamis*.
- Írjatok algoritmust, amely meghatározza az a sorozatnak azt a leghosszabb tömbszakaszát, amely az a sorozat azon elemeit tartalmazza, amelyeknek *nincs egyetlen testvérük sem* az a sorozatban. Egy sorozat tömbszakasza a sorozat egymás utáni pozícióin található elemeit tartalmazza. Az algoritmus bemeneti paraméterei az a sorozat és az n hosszúsága. Kimeneti paraméterek a leghosszabb tömbszakasz kezdeti pozíciója (*start*) és a tömbszakasz k hosszúsága. Ha több azonos hosszúságú leghosszabb tömbszakasz létezik, a legutolsót kell megadni. Ha nem létezik egyetlen, adott tulajdonságú tömbszakasz sem, *start* értéke -1 és k értéke 0.

Példa: Legyen $n = 10$ és $a = (12345, 9, 100, 567, 5678, 345, 123, 8989, 222, 11, 78)$. Az x sorozat elemei, amelyeknek nincs testvérük: 9, 100, 8989, 222, 11, a keresett tömbszakasz pedig: (8989, 222, 11), tehát *start* = 8 és $k = 3$.

Megoldás

Természetesen több lehetséges megoldás létezik. A továbbiakban bemutatjuk azt, amelyre maximális pontszámot lehetett szerezni.

Az értékelés a következő szempontok szerint történt:

- A bemeneti és kimeneti paraméterek megfelelnek-e? (2 pont)
- Helyesen és optimálisan vizsgálja a „testvér” tulajdonságot? (10 pont)
- Helyesen határozza meg a testvérek tömbjét? (9 pont)
- Helyesen és optimálisan határozza meg a leghosszabb tömbszakaszt? (4 pont)

Az alábbi megoldásban alkalmazunk egy logikai tömböt, amelyben az adott szám számjegyeinek előfordulását vesszük nyilvántartásba. Ahhoz, hogy két számról eldöntsük, hogy testvérek-e, elégséges egy szám esetében létrehozni az előfordulások tömbjét, amelyben a másik szám utolsó számjegyének megfelelő elemet kérdezzük le. A feldolgozás leáll, ha találtunk két közös számjegyet.

```

bool testverek(int a, int b){    // eldöntjük, hogy a és b testvérek vagy sem
    bool szJegyekA[10];
    for (int i = 0; i <= 9; i++){
        szJegyekA[i] = false;
    }
    while (a > 0){
        szJegyekA[a % 10] = true;
        a /= 10;
    }
    int kozosSzjDb = 0;
    while (b > 0 && kozosSzjDb < 2){
        if (szJegyekA[b % 10]){
            kozosSzjDb++;
            szJegyekA[b % 10] = false;
        }
        b /= 10;
    }
    return kozosSzjDb == 2;
}

```

A következő alprogram létrehozza a *vanTestvére* tömböt a „testvér” tulajdonság alapján. Vigyázunk, hogy egy adott számot ne tekintsünk saját maga testvéreinek. Az adott sorozat elemeinek megfelelően a *vanTestvére* tömb azon elemei kapnak *igaz* értéket, amelyeknek van legalább egy testvérük az *a* tömbben.

```

void testverekTomb(int n, int a[], int vanTestvere[]){
    // ha a vanTestvere tömbben az i. elem értéke true, ez azt jelenti, hogy az i.
    // személynek van legalább egy testvére az a sorozatban
    for (int i = 1; i <= n; i++){
        int j = 1;
        vanTestvere[i] = false;
        while (!vanTestvere[i] && j <= n){
            if (i != j)
                vanTestvere[i] = testverek(a[i], a[j]);
            j++;
        }
    }
}

```

A leghosszabb tömbszakaszt a *vanTestvére* tömb értékeinek alapján határozzuk meg, de a tömbszakasz széleinek indexei érvényesek az *a* sorozatban is.

Az *aktEleje* változóban az aktuális tömbszakasz első elemének indexét, az *aktHossz* változóban a hosszát tartjuk nyilván. Amikor megjelenik az első olyan elem, amelynek van testvére, meghatározzuk az aktuális tömbszakasz hosszát, és aktualizáljuk a *k* értékét, amely a leghosszabb tömbszakasz hossza.

A *start* változó értékét csak akkor aktualizáljuk, ha találtunk egy, az eddigi tömbszakaszoknál hosszabbat.

```
void leghosszabbTombszakasz(int n, int a[], int &start, int &k){
    int vanTestvere[1000];
    testverekTomb(n, a, vanTestvere); // létrehozzuk a vanTestvere tömböt
    start = -1;
    k = 0;                               // még nem találtunk egyetlen tömbszakaszt sem
    int i = 1;                           // bejárjuk a sorozat elemeit
    while (i <= n){
        int aktEleje = i;                // az aktuális tömbszakasz első elemének indexe
        int aktHossz = 0;                // az aktuális tömbszakasz hossza
        while (i <= n && ! vanTestvere[i])
            i++;                          // haladunk, amíg az aktuális elemnek nincs testvére
        if (i >= aktEleje)
            aktHossz = i - aktEleje;
        if (aktHossz >= k){
            start = aktEleje; // aktualizáljuk a leghosszabb tömbszakasz első
                               // elemének indexét
            k = aktHossz;      // aktualizáljuk a leghosszabb tömbszakasz hosszát
        }
        i++;
    }
}
```

5.1. Általánosságok – 2017

2017-ben nem voltak rácsteszték. Az **I.** részben két feladatot tűzött ki a bizottság, a **II.** és **III.** részben egy-egy feladatot. Az **I.** rész első feladata egy úgynevezett modellezési feladat volt, amelynek megoldása – általában – rövid volt, de talán a legnehezebb. Ha a versenyzőnek nem sikerült ötletesen és – természetesen – hatékonyan megoldani a feladatot, volt még egy esélye, és pedig írt egy algoritmust, amely szimulálta a leírt eseményeket. A második feladat megoldása egy olyan algoritmus vagy alprogram volt, amelynek esetében szintén számított a megoldás hatékonysága. A **II.** részben egy iteratív algoritmus rekurzív változatát, vagy egy rekurzívnak az iteratív változatát kellett megírnia a versenyzőknek. A **III.** részben a feladat összetettebb volt, több alprogramot kért.

A versenyfeladatokat – mind a *Matek–Infó versenyen*, mind a *felvételin* – a következő figyelmeztető szöveg előzte meg:

A versenyzők figyelmébe:

1. A feladatok megoldásait *pseudokódban* vagy egy *programozási nyelvben* (*Pascal/C/C++*) kell megadni.
2. A megoldások értékelésekor az első szempont az algoritmus **helyessége**, majd a **hatékonysága**, ami a *végrehajtási időt* és a *felhasznált memória méretét* illeti.
3. A tulajdonképpeni megoldások előtt, **kötelezően leírástok szavakkal az alprogramokat, és megindoklójátok a megoldások lépéseit**. Feltétlenül írjátok **megjegyzéseket** (kommenteket), amelyek segítik az adott megoldás technikai részleteinek megértését. Adjátok meg az azonosítók jelentését és a felhasznált adatszerkezeteket. Ha ez hiányzik, a tételre kapható pontszámotok 10%-kal csökken.
4. Ne használjatok különleges fejláblományokat, előredefiniált függvényeket (például STL, karakterláncokat feldolgozó sajátos függvények stb.).

Az értékelő/javítóbizottság számára is készült egy felhívó szöveg:

1. Az értékeléskor az algoritmus helyességére fektetjük a hangsúlyt, nem az esetleges szintaxishibákra.
2. Ha egy versenyző C/C++ programot írt, nem vonunk le a pontszámából, ha hiányzik valahonnan egy '{', '}', ';', '(', ') ' karakter, de az esetlegesen hiányzó deklaráció, vagy redundáns deklaráció sem jár penalizációval.
3. Ha egy versenyző Pascal programot írt, nem vonunk le a pontszámából, ha hiányzik valahonnan egy ';', '(', ') ' karakter, de az esetlegesen hiányzó **begin/end** kulcsszó, deklaráció, vagy redundáns deklaráció sem jár pontlevonással.
4. **Ha több ilyen hibát találunk**, és nyilvánvaló, hogy ezek nem csak azért jelentek meg, mert papíron nehéz ezekre a részletekre figyelni, a pontlevonást illetően az értékelést végző személy dönt.
5. A sorozatok indexelése kezdődhet 0-tól vagy 1-től.
6. Ha a feladat specifikációja egy alprogramot kér, és a versenyző több alprogram segítségével oldja meg a feladatot, helyesnek minősítjük, ha a versenyző megindokolta ennek szükségességét.

A feladatlap néhány hasznos információval záródik:

Megjegyzések:

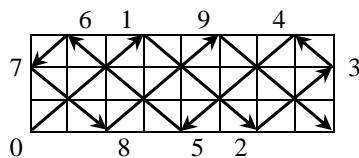
1. Minden tétel kidolgozása kötelező.
2. A megoldásokat a vizsgalapokra írjátok, (a piszkozatokat nem vesszük figyelembe).
3. Hivatalból jár 10 pont.
4. Rendelkezésekre áll 3 óra.

5.2. Matek–Infó verseny – 2017. április 1.

I. Tétel (35 pont)

1. Fénysugár (20 pont)

Legyen egy tükrökből kialakított, téglalap alakú keret. Egy fénysugár elindul a téglalap bal alsó sarkából, 45° fokos szöget alkotva a téglalap alsó oldalával, és nekiütközik a téglalap felső vagy jobboldali oldalának.



Itt tükröződik (elindul egy másik oldal felé, szintén 45° fokos szöget alkotva azzal az oldallal, amelybe beleütközött). Így folytatja az útját, amíg a keret valamelyik sarkába nem ér.

Írjatok alprogramot, amely kiszámítja, hogy hányszor (*váltSz*) változtatja a tükröződés irányát a fénysugár, amíg leáll valamelyik sarokban. A kiindulási pontot nem számítjuk be ebbe a számba. Az alprogram bemeneti paraméterei a téglalap hossza ($1 < a < 10\,000$) és szélessége ($1 < b < 10\,000$), míg a *váltSz* kimeneti paraméter.

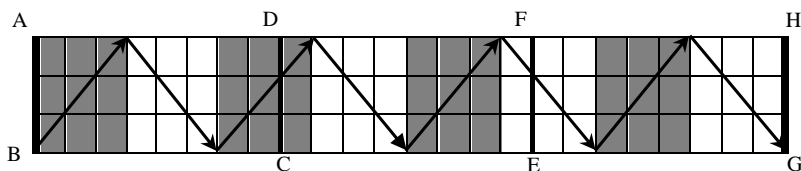
1. Példa: ha $a = 8$ és $b = 3$, akkor *váltSz* = 9.

2. Példa: ha $a = 8$ és $b = 4$, akkor *váltSz* = 1.

Megoldás

Két lehetséges megoldást mutatunk be. Az első, amelyre maximális pontszámot lehetett szerezni, komoly elemzést igényelt és egy kevés matematikai tudást.

Legyen például egy ABCD téglalap, amelyet 1×1 méretű négyzetecskékre osztottunk. A téglalap hossza $a = 8$ (négyzetecske), a szélessége $b = 3$ (négyzetecske). Ha tükrözzük a téglalapot a CD oldal mentén, megkapjuk a CDFE téglalapot. Hasonlóan járunk el a CDFE téglalappal (most az EF oldal mentén tükrözzünk).



Ha a fénysugarat az ABGH téglalapon belül mozgatjuk, látható, hogy $k - 1$ alkalommal változtatja az irányát, ahol k azoknak a négyzeteknek a száma, amelyeknek oldalhosszúsága $b = 3$ négyzetecske, tehát a BG oldal hossza egyenlő $lkkt(a, b)$ négyzetecskével.

Kiszámítjuk a példánk esetében: $k = \frac{lkkt(a,b)}{b} = \frac{lkkt(8,3)}{3} = \frac{24}{3} = 8$ és az ábrán is látható, hogy az ABGH téglalapon belül a fénysugár 7-szer változtat irányt.

Visszatérünk az eredeti ABCD téglalaphoz. Amikor a fénysugár nekiütközik a CD oldalnak, majd az AB oldalnak, $p - 1$ -szer változtatja meg az irányát, ahol p a tükrözött (a példában 8 oldalhosszúságú) téglalapok száma. Valóban, $p = \frac{lkkt(a,b)}{a} = \frac{24}{8} = 3$. Az ábrán látjuk, hogy a fénysugár kétszer ütközik függőlegesen.

Tehát, az irányváltások összes száma:

$$\text{váltSzám} = (k - 1) + (p - 1) = \frac{lkkt(a,b)}{b} - 1 + \frac{lkkt(a,b)}{a} - 1.$$

Tudjuk, hogy $lkkt(a, b) = \frac{a \cdot b}{lnko(a,b)}$. Elvégezzük a behelyettesítéseket:

$$\text{váltSzám} = \frac{a \cdot b}{lnko(a,b) \cdot b} - 1 + \frac{a \cdot b}{lnko(a,b) \cdot a} - 1 = \frac{a}{lnko(a,b)} - 1 + \frac{b}{lnko(a,b)} - 1.$$

A példánk esetében: $lnko(a, b) = lnko(8, 3) = 1$, tehát behelyettesítve:

$$\text{váltSzám} = 8 / 1 - 1 + 3 / 1 - 1 = 9.$$

Következik, hogy a fénysugár a és b értékeinek függvényében változtatja az irányt, de látható a fenti levezetésből, hogy tulajdonképpen az $lnko(a, b)$ függvényében.

V1: A legnagyobb közös osztó alkalmazása (20 pont)

- Az $lnko(a, b)$ -t helyes algoritmussal számolja ki. (5 pont)
- A váltSz értékét a fenti képlettel számítja ki. (15 pont)

<pre>int lnko(int a, int b){ if ((a == b) && (a != 0)){ return 1; } if (a * b == 0){ return a + b; } int mar = a % b; while (mar != 0){ a = b; b = mar; mar = a % b; } return b; }</pre>	<pre>int feny sugar_1(int a, int b){ int d = lnko(a, b); if (d == a) valtSz = b / d - 1; else{ if (d == b) valtSz = a / d - 1; else valtSz = b / d + a / d - 2; } return valtSz; }</pre>
--	--

V2: A második megoldás a fénysugár útját szimulálja (15 pont)

Ahhoz, hogy a szimulálást megvalósíthassuk, a téglalapot egy koordináta-rendszerbe helyezzük, amelynek az origója a téglalap bal alsó sarka, az Ox tengelye a téglalap alsó oldala (az ábrán a BC oldal) és az Oy tengelye a téglalap bal oldala (az ábrán a téglalap AB oldala). A következő alprogram ebben a koordináta-rendszerben mozgatja a fénysugarat a szabályoknak megfelelően, amíg egy sarokba nem ér.

```
int fénysugar_2(int a, int b){
    valtSz = 0;
    int pos_x = 0;
    int pos_y = 0;
    enum irány {jobbraFel, jobbraLe, balraLe, balraFel};
    irány aktIrány = jobbraFel;
    bool sarok = false;
    while (!sarok)
        switch(aktIrány){
            case jobbraFel:
                if (a - posY == b - posX)           // a fénysugár egy sarokba ért
                    sarok = true;
                else
                    if (a - posY < b - posX){// a fénysugár a felső oldalba ütközött
                        posX += a - posY;
                        posY = a;
                        aktIrány = jobbraLe;
                        ++valtSz;
                    }
                    else{                          // a fénysugár a jobb oldalba ütközött
                        posY += b - posX;
                        posX = b;
                        aktIrány = balraFel;
                        ++valtSz;
                    }
                break;
            case jobbraLe:
                if (posY == b - posX)               // a fénysugár egy sarokba ért
                    sarok = true;
                else
                    if (posY < b - posX){ // a fénysugár az alsó oldalba ütközött
                        posX += posY;
                        posY = 0;
                        aktIrány = jobbraFel;
                        ++valtSz;
                    }
                    else{                          // a fénysugár a jobb oldalba ütközött
                        posY -= b - posX;
                        posX = b;
                        aktIrány = balraLe;
                        ++valtSz;
                    }
                break;
        }
```

```

case balraLe:
    if (posY == posX)                // a fénysugár egy sarokba ért
        sarok = true;
    else
        if (posY < posX){            // a fénysugár az alsó oldalba ütközött
            posX -= posY;
            posY = 0;
            aktIrany = balraFel;
            ++valtSz;
        }
        else{                        // a fénysugár a bal oldalba ütközött
            posY -= posX;
            posX = 0;
            aktIrany = jobbraLe;
            ++valtSz;
        }
        break;
case balraFel:
    if (a - posY == posX)            // a fénysugár egy sarokba ért
        sarok = true;
    else
        if (a - posY < posX){        // a fénysugár a felső oldalba ütközött
            posX -= a - posY;
            posY = a;
            aktIrany = balraLe;
            ++valtSz;
        }
        else{                        // a fénysugár a bal oldalba ütközött
            posY += posX;
            posX = 0;
            aktIrany = jobbraFel;
            ++valtSz;
        }
        break;
    }
return valtSz;
}

```

2. „Erős” számok (15 pont)

Egy nullától különböző *sz* természetes számnak az *erőssége* *k*, ha bináris alakjában pontosan *k* darab 1-es számjegy található. Például, a 23 erőssége 4 (kettes számrendszerben felírva, 4 darab 1-es számjegye van). Adott számsorozat *k* *erősségű csoportjának* nevezzük azt a részsorozatot, amely a sorozat *k* erősségű elemeit tartalmazza, az elemek eredeti sorrendjében. Például, az *s* = (7, 12, 3, 13, 24, 19) sorozat *k* = 2 *erősségű csoportja* a (12, 3, 24) részsorozat.

Írjatok alprogramot, amely meghatároz minden *erősségi csoportot*, amelyek az adott *x* sorozat elemeiből létrehozhatók. Bemeneti paraméterek: a nem nulla, 30 000-nél kisebb, különböző természetes számokból álló *x* sorozat és a sorozat *n* hossza ($1 < n < 100$). Kimeneti paraméterek: a *csSzáma* (a csoportok száma) és a *csoportok* (a létrehozott csoportok, erősségük szerint növekvően rendezve; a csoporton belül az elemek sorrendje tetszőleges).

Példa: ha $n = 6$ és $x = (12, 3, 24, 16, 15, 32)$, akkor $csSz\acute{a}ma = 3$, és a *csoportok*: $(16, 32), (12, 3, 24), (15)$.

Megoldás

Az adott feladat részfeladatokra bontása a versenyzőre hárul. Egyértelmű, hogy ahhoz, hogy a számokat erősségük alapján csoportosíthassuk, előbb meg kell határoznunk ezeket az erősségeket.

a. A szám erősségének meghatározása

Meg szeretnénk számolni a szám 1-es bitjeit. Ezt többféleképpen is megvalósíthatjuk. A megoldás „szépségének” és hatékonyságának megfelelően, a pontszámok csökkenő sorrendet alkottak. Megjegyezzük, hogy 2017-ben a *bitenkénti* műveletek még a tematikához tartoztak.

Va1: Alkalmazzuk az & (*bitenkénti és* műveletet) (5 pont)

- Ha *szám* páratlan (utolsó bitje 1), akkor $szám \& (szám - 1) = szám - 1$.

<i>szám</i>	9	1001	11	1011	13	1101	15	1111
<i>szám - 1</i>	8	1000	10	1010	12	1100	14	1110
<i>szám & (szám - 1)</i>	8	1000	10	1010	12	1100	14	1110

- Ha *szám* páros (utolsó bitje 0) és a bináris alakja k darab 1-es számjegyet tartalmaz, akkor $szám \& (szám - 1)$ megőrzi a *szám* bináris alakjának $k - 1$ darab legjelentősebb 1-es bitjét.

<i>szám</i>	8	1000	10	1010	12	1100	14	1110
<i>szám - 1</i>	7	0111	9	1001	11	1011	13	1101
<i>szám & (szám - 1)</i>	0	0000	8	1000	8	1000	12	1100

```
int eroSzamolas_1(int szam){
    int ero = 0;
    do{
        szam &= szam - 1; // bitenkénti „és” szam és szam-1 között
        ero++;           // minden lépés „kizár” szam-ból egy 1-es bitet
    } while (szam);
    return ero;
}
```

Va2: Alkalmazzuk a *bitenkénti és* műveletet (&) és a jobbra tolást (1 pozícióval), (>>). Ha *szám* legkevésbé jelentős bitje 1-es, a művelet eredménye 1. (4 pont)

```
int eroSzamolas_2(int szam){
    int ero = 0;
    while (szam > 0){
        if (szam & 1) // ha az utolsó bit 1
            ero++;
        szam >>= 1;   // 2-vel osztunk 1 jobbra tolással
    }
    return ero;
}
```

Va3: Kettes számrendszerbe alakítjuk a számot és megszámloljuk az ábrázolásban található 1-es biteket. (3 pont)

```
int eroSzamolas_3(int szam){
    int ero = 0;
    while (szam > 0){
        if (szam % 2 == 1)
            ero++;
        szam /= 2;
    }
    return ero;
}
```

b. Azonos erősségű számok csoportosítása

A megoldás lehetséges stratégiáit csak vázoljuk, majd kiválasztjuk azt a megoldást, amelyről az elemzésünk eldönti, hogy hatékonyabb, mint a többi. Az első gondunk annak az adatszerkezetnek a kiválasztása, amelyben a csoportokat tárolni fogjuk.

Vb1: a csoportokat egy kétdimenziós tömbbe helyezhetjük, ahol az első oszlop az egyes csoportok hosszát tartalmazza.

Vb2_a: a csoportokat egy olyan sorozatba helyezhetjük, amelynek minden eleme egy sorozat.

Vb2_b: a csoportokat tároló sorozat minden eleme egy verem típusú dinamikusan láncolt lista.

Vb3: ha nem akarjuk nyilvántartani a csoportok hosszát, tehetjük a csoportokat egy kétdimenziós tömbbe, ahol a „fölösleges” elemek értéke -1.

Eldöntjük, hogy a legkedvezőbb tárolási mód az, amelyet a **Vb1** pontban írtunk le. Most a csoportok kialakításának mikéntjén gondolkodunk. Elvetjük a számok (erősségük alapján történő) rendezésének ötletét, még akkor is, ha a feladat szövegében ott a követelmény, hogy a csoportokat a számok erősségének növekvő sorrendjében kell megadnunk. Egy ilyen rendezés csak terhelné az algoritmusunkat, miközben a kiválasztott tárolási módban az *i* erősségű számokat az *i*. sorba helyezzük, tehát a csoportokat erősségük szerint növekvő sorrendben kapjuk meg.

A következőkben $\text{maxSzjSz} = 16$ és $\text{maxDim} = 100$.

Vb1: A kétdimenziós tömbnek maxSzjSz sora és *n* oszlopa lesz (előfordulhat, hogy a sorozat minden egyes eleme más-más csoportot alkot). Az *i*. sorba azok a számok kerülnek, amelyeknek az erőssége *i*. A csoport számosságát a 0 oszlopindexű elembe tároljuk. A csoport elemeit a megfelelő sor további pozícióin tároljuk. Miután meghatároztuk a sorozat aktuális elemének erősségét, megvizsgáljuk, hogy az erősségnek megfelelő sor hossza nulla-e. Ha igen, növeljük a csoportok darabszámát (*csoportokDb*), hiszen új csoportot indítunk, majd növeljük a sor hosszát és elhelyezzük a sorba az új elemet.

```
int erossegek_v1(int n, int x[], int f[][maxDim], int &csoportokDb){
    for (int i = 1; i < maxSzjSz; i++)
        f[i][0] = 0;
    csoportokDb = 0;
    for (int i = 0; i < n; i++){
        int ero = eroSzamolas_v1(x[i]);
        if (f[ero][0] == 0)
            csoportokDb++;
        int pos = f[ero][0] + 1;
        f[ero][pos] = x[i];
        f[ero][0]++;
    }
}
```

II. Tétel (15 pont)

Adott a következő algoritmus, amelynek három, nem nulla természetes szám bemeneti paramétere van: a , b és sz , amelyeknek értékei kisebbek, mint 10 000:

```
Algoritmus f(a, b, sz):
    k ← 0
    Amíg b < sz végezd el:
        k ← k + 1
        b ← a + b
        a ← b - a
    vége(amíg)
    térítsd k
Vége(algoritmus)
```

- Adjátok meg a feladat szövegét, amelyet ez az algoritmus old meg, ha $a = 1$ és $b = 0$ értékekre hívjuk meg.
- Mit térít az $f(1, 0, 10)$ hívás?
- Írjátok le a fenti algoritmusnak egy *rekurzív* változatát, megőrizve az iteratív (nem rekurzív) változat fejlődését.

Megoldás

Adva van egy algoritmus és meg kell fogalmaznunk a feladat kijelentését, amelyet ezzel az algoritmussal oldunk meg. Ha a versenyző találkozott ezzel az algoritmussal, azonnal tudni fogja, hogy az adott sz számnál kisebb *Fibonacci* számokat generálja, megszámlolja ezeket k -ban és téríti ezt a számot. Ha nem ismeri fel az algoritmus funkcióját, ellenőrző táblázatot készít.

A következő táblázatban látható, hogy az a , illetve a b értékei, rendre *Fibonacci* számok. A k értéke a b -ben generált, sz -nél kisebb számok darabszáma.

a	b	sz	k
1	0	10	0
0	1		1
1	1		2
1	2		3
2	3		4
3	5		5
5	8		6

Így a válaszok:

- Az algoritmus az sz -nél szigorúan kisebb *Fibonacci* számok darabszámát határozza meg.
- Ha az algoritmust az $f(1, 0, 10)$ utasítással hívjuk meg, a térített érték 7.
- Egy helyes rekurzív algoritmus:

```
int f_Rek(int a, int b, int szam){
    if (b < szam)
        return f_Rek(b, a + b, szam) + 1;
    else
        return 0;
}
```

III. Tétel (40 pont)

Előszólet

Sors-számjegynek hívjuk azt a természetes számot, amelyet adott természetes számra a következőképpen számítunk ki: összeadjuk a szám számjegyeit, majd a kapott összeg számjegyeit, és így tovább, amíg a kapott összeg nem válik egy-számjegyű számmá. Például, a 182 *sors-számjegye* 2 ($1 + 8 + 2 = 11$, $1 + 1 = 2$).

Egy pontosan k számjegyű p számot egy legkevesebb k számjegyű q szám *előszóletének* nevezünk, ha a q szám első k számjegyéből alkotott szám (balról jobbra tekintve) egyenlő p -vel. Például, 17 előszólete 174-nek, és 1713 előszólete 1 713 242-nek.

Legyen az sz természetes szám ($0 < sz \leq 10\,000$) és az m sorral és n oszloppal ($0 < m \leq 100$, $0 < n \leq 100$) rendezhető A mátrix (kétdimenziós tömb), amelynek elemei 30 000-nél kisebb természetes számok.

Írjatok programot, amely meghatározza és kiírja az sz szám *leghosszabb előszóletét*, amelyet az adott mátrix elemeinek megfelelő *sors-számjegyeiből* fel lehet építeni. Egy ilyen sors-számjegyet akárhányszor fel lehet használni. Ha nem építhető fel előszólet, a program írja ki a „nem létezik előszólet” üzenetet.

Példa: ha $sz = 12319$, $m = 3$, $n = 4$ és $A = \begin{pmatrix} 182 & 12 & 274 & 22 \\ 22 & 1 & 98 & 56 \\ 5 & 301 & 51 & 94 \end{pmatrix}$, akkor a mátrix:

leghosszabb előszólet 1231, a megfelelő sors-számjegyek pedig:

Mátrixelem értéke	182	12	274	22	1	98	56	5	301	51	94
Sors-számjegy	2	3	4	4	1	8	2	5	4	6	4

A megoldásban föl kell használnotok a következő alprogramokat:

- a bemeneti adatok beolvasása billentyűzetről;
- adott számhoz rendelhető sors-számjegy meghatározása;
- a leghosszabb előszelet meghatározása;
- a leghosszabb előszelet kiírása a képernyőre; ha nincs előszelet, a megfelelő üzenet kiírása.

Megoldás

A feladat szövegében megadták a részfeladatokat. Így az egyetlen feladatunk ezeknek kidolgozása.

a. A bemeneti adatok beolvasása (mátrix és szám), *maxDim* = 100 (3 pont)

```
void beOlvas(int &szam, int &m, int &n, int A[][maxDim]){
    cin >> szam >> m >> n;
    for (int i = 0; i < m; i++){
        for (int j = 0; j < n; j++){
            cin >> A[i][j];
        }
    }
}
```

b. Adott számhoz rendelhető sors-számjegy meghatározása (5 pont)

A sors-számjegy kiszámításának módja megtalálható a feladat szövegében: összeadjuk az aktuális szám számjegyeit, majd a kapott összeg számjegyeit, és így tovább, amíg a kapott összeg egyszámjegyű számmá nem válik:

```
int sorsSzamjegy(int x){
    while(x > 9){
        int y = x;
        int s = 0;
        while (y > 0){
            s += y % 10;
            y /= 10;
        }
        x = s;
    }
    return x;
}
```

c. A leghosszabb előszelet meghatározása (15 pont)

Az *sz* szám leghosszabb előszeletét, az adott mátrix elemeinek megfelelő sors-számjegyeiből kell felépítenünk. Előbb meg kell határoznunk a mátrix elemeinek sorsszámjegyeit, de mivel a további feldolgozás érdekében csak azt kell tudnunk, hogy mely számjegyek fordulnak elő a mátrix sorsszámjegyei között, előfordulási tömbben tartjuk nyilván ezeket. Szükségünk lesz még az adott *sz* számjegyeinek tömbjére. Végül, bejárjuk ezt a tömböt (a legértékesebb számjegytől kezdve) és megnézzük, hogy szerepel-e a sorsszámjegyek között.


```

int eloszelet_1(int sz, int m, int n, int A[][maxDim], int szamjegyek[],
               int & szamjegyekSz){
    bool elof[maxSzjSz];
    int i = 0;
    for (i = 0; i < maxSzjSz; i++) elof[i] = 0;
    for (i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            elof[sorsSzamjegy(A[i][j])] = 1;
    szamjegyekSz = 0;
    while (sz > 0){
        szamjegyek[szamjegyekSz++] = sz % 10;
        sz /= 10;      // az sz számjegyei előfordulásuk fordított sorrendjében
    }                // kerülnek a szamjegyek sorozatba
    i = szamjegyekSz - 1; // a feldolgozást a legértékesebb számjegytől kezdjük
    while ((i >= 0) && (elof[szamjegyek[i]]))
        i--;
    return szamjegyekSz - i - 1;
}

```

Ha nem az előfordulások tömbjét használjuk, hanem magát a sorsszámjegyek tömbjét, ezeket rendre meg kell keresnünk a mátrixban. (8 pont)

```

bool keresSzj(int szamjegy, int m, int n, int A[][maxDim]){
    for (int i = 0; i < m; i++){
        for (int j = 0; j < n; j++){
            if (szamjegy == sorsSzamjegy(A[i][j]))
                return true;
        }
    }
    return false;
}

int eloszelet_2(int sz, int m, int n, int A[][maxDim],
               int szamjegyek[], int &szamjegyekSz){
    szamjegyekSz = 0;
    while (sz > 0){
        szamjegyek[szamjegyekSz++] = sz % 10;
        sz /= 10;
    }
    int i = szamjegyekSz - 1;
    int p = 0;
    while (i >= 0){
        if (keresSzj(szamjegyek[i], m, n, A)){
            p++;
            i--;
        }
        else
            return p;
    }
    return p;
}

```

d. A leghosszabb előszolet kiírása/üzenet (3 pont)

```
void kiIrEloszelet(int hossz, int szamjegyek[], int szamjegyekSz){
    if (hossz == 0)
        cout << "nem letezik eloszelet";
    for (int i = 0; (i < hossz && i < szamjegyekSz); i++)
        cout << szamjegyek[szamjegyekSz - i - 1];
    cout << endl;
}
```

e. Hívó programegység (main függvény/főprogram): (3 pont)

```
void main (){
    int szam = -1;
    int m = -1;
    int n = -1;
    int A[maxDim][maxDim];
    beOlvas(szam, m, n, A);
    int szamjegyek[maxDim];
    int szamjegyekSz = 0;
    int hossz = eloszelet_1(szam, m, n, A, szamjegyek, szamjegyekSz);
    kiIrEloszelet (hossz, szamjegyek, szamjegyekSz);
}
```

A helyes paraméterezésért járt még 5 pont, a megjegyzésekért, az indentálásért, a beszédes azonosítókért egyenként 2-2 pontot lehetett még szerezni.

5.3. Felvételi verseny – 2017. július 17.

I. Tétel (35 pont)

1. Csokik (20 pont)

Egy reklámcég egy új csokoládét szeretne népszerűsíteni és ebből a célból azt tervezi, hogy kioszt egy-egy csokit n gyermeknek ($10 \leq n \leq 10\,000\,000$), akiket előbb körbeállítottak. A cég alkalmazottai rájöttek, hogy túl nagy költség lenne, ha minden gyermeknek adnak majd egy csokit. Következésképpen, úgy döntenek, hogy az n gyermek közül csak minden k -adik fog csokit kapni ($0 < k < n$). Elkezdődik a számolás k -ig, majd újból k -ig (amikor az utolsó gyermekhez érnek, a kiszámolás folytatódik az első gyermekkel és így tovább). Számoláskor minden gyermeket figyelembe vesznek, függetlenül attól, hogy kapott már csokit vagy sem. A kiszámolás *leáll, amikor a soron levő csokit egy olyan gyermeknek kellene adni, aki már kapott.*

Írjatok alprogramot, amely kiszámítja azt az sz számot, amely azoknak a gyermekeknek a száma, akik *nem* kapnak csokit. Bemeneti paraméterek az n és k természetes számok, kimeneti paraméter az sz természetes szám.

1. Példa: ha $n = 12$ és $k = 9$, akkor $sz = 8$ (nem kap csokit az első, a második, a 4., az 5., a 7., a 8., a 10. és a 11.).

2. Példa: ha $n = 15$ és $k = 7$, akkor $sz = 0$ (minden gyermek kap csokit).

Megoldás

Tudjuk, hogy a szimuláláshoz csak akkor folyamodunk, ha nem ugrik be valami jobb ötlet. Tehát vizsgáljuk annak a lehetőségét, hogy valamilyen számolással/képlettel jussunk eredményhez.

Feltételezzük, hogy $n = 5$ és $k = 4$. A számolást, amely a feladat szövege szerint körkörösén történik, elképzelhetjük lineárisan több „kicsi” sorozatban, mindegyikben n gyerekkel. Ha a sok kicsi sorozatot, egymás után ragasztjuk, kapunk egy „nagy” sorozatot, amelyhez p gyerek tartozik, ahol p n -nek többszöröse. A számolást akkor fejezzük be, amikor az n . gyermek (egy kicsi sorozatban) megkapja a csokiját, mivel a következő gyermek, akinek csokit kellene kapnia a következő kicsi sorozatban a k . lenne, aki viszont már kapott csokit. Tehát p nemcsak n többszöröse, hanem a k számé is, vagyis $p = lkkt(n, k)$.

Az összes p gyerek közül pontosan p / k gyerek kap csokit, a többi $n - p / k$ nem kap.

$$sz = n - \frac{p}{k} = n - \frac{lkkt(n,k)}{k} = n - \frac{\frac{n \cdot k}{lnko(n,k)}}{k} = n - \frac{n}{lnko(n,k)}.$$

Hely	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Sorszám	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
Csoki																				

V1a: az sz értékének helyes meghatározása ($sz = n - n / lnko(n, k)$) (15 pont)

<pre>int lnko(int a, int b){ if ((a == b) && (a == 0)) return 1; if (a * b == 0) return a + b; int mar = a % b; while (mar){ a = b; b = mar; mar = a % b; } return a; }</pre>	<pre>int csokik_v1a(int n, int k){ return n - n / lnko(n, k); }</pre>
---	---

V1b: optimalizált szimulálás (15 pont)

A számolást a sorozat első bejárásával kezdjük, amelynek során n / k gyerek kap csokit. Ahhoz, hogy folytassuk, gondolatban létrehozuk a „nagy” sorozatot (több „kicsi” sorozatból, amelyeknek a hossza egyenként n). Ennek a „nagy” sorozatnak a mérete p .

A következő számolás az $(n - n \bmod k + 1)$ pozíción levő gyerekekkel kezdődik és a következő gyerek, aki csokit kap, a $(k - n \bmod k)$ pozíción található. A „nagy” sorozatban addig lépünk előre $(p \bmod k)$ -val amíg a sorozat p hossza k -nak többszöröse nem lesz. Minden lépésben az utolsó $(n \bmod k)$ gyerek nem kap csokit, tehát beszámítanak a következő számolásba. Minden lépésben p / k gyerek kap csokit.

A *csokitKapnak* változóban megszámoljuk azokat a gyerekeket, akik kapnak csokit, tehát az $n - kapnakCsokit$ kifejezés értékét térítjük.

```
int csokik_v1b(int n, int k){
    int csokitKapnak = n / k;
    int p = n;
    while (p % k > 0){
        p = n + p % k;
        csokitKapnak += p / k;
    }
    return n - csokitKapnak;
}
```

V2: klasszikus szimulálás (8 pont)

Két k . gyerek közöttiek nem kapnak csokit, tehát k lépésszámmal járjuk be a sorozatot. A kör bejárását úgy oldjuk meg, hogy vigyázunk arra a pillanatra, amikor meghaladtuk az n . gyereket és ilyenkor a (*poz*) számláló értékét – nem 1-re – hanem *poz mod n*-re állítjuk.

A kör bejárása véget ér, amikor *poz* egyenlővé válik k -val, ami azt jelenti, hogy az első gyerek, aki csokit kapott, másodszor kerülne sorra.

```
int csokit_v2(int n, int k){
    int poz = k;
    int csokitKapnak = 0;
    do{
        poz += k;
        if (poz > n){
            poz = poz % n;
        }
        csokitKapnak ++;
    } while (poz != k);
    return n - csokitKapnak;
}
```

2. Megfeleltetés (15 pont)

Legyen az n elemű ($1 \leq n \leq 10\,000$) a és az m elemű ($m \leq 10\,000$) b sorozat, amelyeknek elemei 30 000-nél kisebb természetes számok.

Tömbszakaszt alkot egy sorozat egy vagy több eleme, amelyek az eredeti sorozatban egymás utáni pozíciókon találhatók.

Az a sorozat „*megfeleltethető*” a b sorozatnak, ha az a sorozatot fel tudjuk osztani diszjunkt tömbszakaszokra úgy, hogy teljesüljenek a következő tulajdonságok:

- ha az összes tömbszakaszt, a felosztás sorrendjében, egymás után ragasztjuk, megkapjuk az a sorozatot;
- ha az összes tömbszakaszt, a felosztás sorrendjében, behelyettesítjük a megfelelő tömbszakasz elemeinek összegével, megkapjuk, rendre a b sorozat elemeit.

Írjatok alprogramot, amely eldönti, hogy az a sorozat *megfeleltethető* vagy sem a b sorozatnak. Ha igen, találjátok meg azt az elemét a b sorozatnak (és ennek az elemnek a k indexét), amely egyenlő az a sorozat leghosszabb tömbszakaszához tartozó elemek összegével. Bemeneti paraméterek az a és b sorozat, valamint ezeknek a hossza: n és m . Kimeneti paraméterek a *válasz*, a k és a *maxHossz*, ahol: *válasz* értéke *igaz*, ha a megfeleltetés lehetséges, ellenkező esetben *hamis*; k a b sorozat azon elemének indexe, amely egyenlő a legnagyobb (*maxHossz*)

elemszámú tömbszakasz elemeinek összegével. Ha több *maxHossz* elemszámú tömbszakasz létezik, az elsőt vesszük figyelembe. Ha *válasz* értéke *hamis*, *k* és *maxHossz* értéke -1.

1. Példa: ha

$n = 12$, $a = (6, 3, 4, 1, 6, 4, 6, 1, 7, 1, 8, 7)$,

$m = 4$ és $b = (13, 7, 18, 16)$, akkor *válasz* = *igaz*, mivel:

$6 + 3 + 4 = 13$, $1 + 6 = 7$, $4 + 6 + 1 + 7 = 18$, $1 + 8 + 7 = 16$.

Ezek szerint $k = 3$ és *maxHossz* = 4.

2. Példa: ha

$n = 17$, $a = (10, 12, 11, 2, 2, 3, 2, 3, 13, 3, 41, 5, 4, 5, 6, 5, 2)$,

$m = 6$ és $b = (33, 4, 15, 41, 25, 2)$, akkor *válasz* = *hamis*, mivel:

$10 + 12 + 11 = 33$, $2 + 2 = 4$, de $3 + 2 + 3 < 15$, és $3 + 2 + 3 + 13 > 15$.

Tehát a $b_3 = 15$ értéket nem tudjuk megfeleltetni az a sorozat egyik tömbszakaszának sem, mivel nem egyenlő az a sorozat egymás utáni elemeinek összegével. *Megjegyzés:* A példában a sorozatot 1-től kezdődően indexeltük.

Megoldás

A legelső észrevételünk arra az esetre vonatkozik, amikor nem lehetséges megvalósítani a megfeleltetéseket. Ha az a sorozat mérete kisebb, mint a b sorozaté, nincs megoldás, hiszen – bármilyen értékei is lennének a két sorozat elemeinek – a b sorozatban maradnának elemek, amelyeknek nincs mit megfeleltetni.

Az elemzés során rájövünk, hogy célszerű a két sorozatot párhuzamosan bejárni. Az a sorozat elemeit addig adjuk össze az *összeg* változóban amíg ez az összeg kisebb vagy egyenlő a b sorozat aktuális elemének értékével. Ha a parciális összeg egy adott elem összeadása után egyenlő a b sorozat aktuális elemével, haladunk tovább mindkét sorozatban. Ha a parciális összeg nagyobb, mint a b sorozat aktuális eleme, az algoritmus *hamis*-at fog téríteni, mivel nem lehet elvégezni a megfeleltetést.

Abban a különleges esetben, amikor az a sorozat bejárása véget ér, de a b sorozatban még vannak elemek, szintén *hamis*-at térítünk.

Az a sorozat maximális hosszúságú tömbszakaszának *maxHossz* hosszát meghatározzuk a két sorozat bejárása során, anélkül, hogy újra bejárnánk a sorozatokat vagy a tömbszakaszokat. A maximális hosszúságú tömbszakasz méretét akkor aktualizáljuk, amikor sikerült „előállítani” a b sorozat aktuális elemét.

```
bool megfeleltetes(int n, int a[], int m, int b[], int& k, int & maxHossz){
    if (n < m)
        return false;
    int i = 0;
    int j = 0;
```

```

int aktHossz = 0;
maxHossz = aktHossz;
bool ok = true;
int aktHossz = 0;
while (ok && i < m){
    int osszeg = 0;
    while (osszeg < b[i] && j < n){
        osszeg += a[j];
        j++;
        aktHossz++;
    }
    if (osszeg == b[i]) {
        i++;
        if (j - aktHossz > maxHossz) {
            maxHossz = j - aktHossz;
            k = i;
        }
        if (osszeg > b[i])
            ok = false;
        if (j == n && i < m)
            ok = false;
    }
}
return ok;
}

```

II. Tétel (15 pont)

Legyen a következő alprogram, ahol n bemeneti paraméter, p és i kimeneti paraméterek (n, p, i – természetes számok, $1 \leq n \leq 1\,000\,000$, $(0 \leq p \leq 1\,000\,000$, $0 \leq i \leq 1\,000\,000)$):

```

Algoritmus f(n, p, i):
  Ha  $n \leq 9$  akkor
    Ha  $n \bmod 2 = 0$  akkor
       $p \leftarrow n$ 
       $i \leftarrow 0$ 
    különben
       $p \leftarrow 0$ 
       $i \leftarrow n$ 
  vége(ha)
  különben
     $f(n \text{ DIV } 10, p, i)$ 
    Ha  $n \bmod 2 = 0$  akkor
       $p \leftarrow p * 10 + n \bmod 10$ 
    különben
       $i \leftarrow i * 10 + n \bmod 10$ 
    vége(ha)
  vége(ha)
Vége(algoritmus)

```

- Adjátok meg annak a feladatnak a szövegét, amelyet ez az algoritmus old meg.
- Mi lesz p és i értéke az $f(205609, p, i)$ hívás után?
- Írjátok le egy iteratív változatát az adott algoritmusnak, megőrizve a rekurzív változat fejlécét.

Megoldás

Ahhoz, hogy megírjuk az algoritmus iteratív változatát, előbb meg kell értenünk az adott algoritmus funkcióját. Azt, hogy számjegyekre bontja az adott n számot, hamar belátjuk, de azt is meg kell értenünk, hogy mit ábrázol a p és az i változó.

A rekurzív hívások leállnak, amikor az n szám egyszámjegyű lett. Ekkor adunk kezdőértéket p -nek és i -nek. Ha n páros, p értéke n , i értéke pedig 0 lesz. Más szóval, p kezdőértéke az a páros értékű számjegy, amely az n szám ismételt osztásai után maradt belőle. Ha n páratlan, a két kezdőértékadás máris segít a feladat funkciójának megállapításában, hiszen most p kezdőértéke 0, és i kezdőértéke n . Így megállapítjuk, hogy p az n szám páros számjegyeiből, i a páratlanokból álló számok értéke lesz. A számjegyek az eredeti sorrendben épülnek a p -be és az i -be.

Válaszolunk a kérdésekre:

- Az algoritmus meghatározza az n szám páros, illetve páratlan számjegyeit – az eredeti sorrendben – tartalmazó számokat.
- Az $f(205609, p, i)$ hívás eredményeként $p = 2060$, $i = 59$ lesz.
- Egy lehetséges helyes iteratív algoritmus, amelynek a fejléce azonos a rekurzív algoritmus fejlécével:

```
void fiterativ_1(int n, int & paros, int & paratlan){
    paros = 0;
    paratlan = 0;
    int hatvanyParatlan = 1;
    int hatvanyParos = 1;
    while (n){
        if (n & 1){
            paratlan = paratlan + hatvanyParatlan * (n % 10);
            n /= 10;
            hatvanyParatlan *= 10;
        }
        else{
            paros = paros + hatvanyParos * (n % 10);
            n /= 10;
            hatvanyParos *= 10;
        }
    }
}
```


A következő, kicsit hosszabb algoritmus is helyes:

```
void fiterativ_2(int n, int & paros, int & paratlan){
    paros = 0;
    paratlan = 0;
    int hatvanyParos = 0;
    int hatvanyParatlan = 0;
    while (n > 0){
        int utolsoSzj = n % 10;
        if (utolsoSzj % 2 == 0){
            if (paros == 0 && hatvanyParos == 0){
                paros = utolsoSzj;
                hatvanyParos = 10;
            }
            else {
                paros = paros + utolsoSzj * hatvanyParos;
                hatvanyParos *= 10;
            }
        }
        else {
            if (paratlan == 0){
                paratlan = utolsoSzj;
                hatvanyParatlan = 10;
            }
            else {
                paratlan = paratlan + utolsoSzj * hatvanyParatlan;
                hatvanyParatlan *= 10;
            }
        }
        n /= 10;
    }
}
```

III. Tétel (40 pont)

Képfeldolgozás

Egy fehér-fekete képet egy olyan négyzetes tömbbel kódolunk, amelynek elemei nullák (0 = fehér pixel) és egyesek (1 = fekete pixel). A képet a következő műveletekkel alakíthatjuk át:

- **Megfordítás (M)**, vagyis a 0-t 1-gyé, az 1-et 0-ává változtatjuk;
- **Forgatás 90 fokkal (F)**, az óramutatójárásával megegyező irányban;
- **Zoom (Z)**, vagyis minden pixel helyére négy új, az eredetivel azonos értékű pixel kerül.

Egy feldolgozás-sorozatot egy M, F és Z betűkből álló karaktersorozattal írunk le, ahol a betűk sorrendje tetszőleges.

Írjatok programot, amely adott, m sorral és m oszloppal rendelkező **kép** kétdimenziós tömb (m – természetes szám, $2 \leq m \leq 10$) és s , legtöbb öt képfeldolgozó műveletet tartalmazó betűsor esetében végrehajtja ezeket a műveleteket és kiírja az átalakított képet.

Példa: ha $m = 3$, $\text{kép} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}$ és $s = (\text{F}, \text{M}, \text{F}, \text{Z})$, akkor a feldolgozás

eredménye: $\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$.

Az egyes feldolgozások eredményei, rendre:

$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$.

A megoldásban írjatok egy-egy alprogramot, amely:

- beolvassa a bemeneti adatokat a billentyűzetről (ezek garantáltan megfelelnek a követelményeknek);
- megfordít egy képet;
- elforgat 90 fokkal egy képet;
- végrehajtja a zoom műveletet egy képre;
- kiír a képernyőre egy képet.

Megoldás

Nem szükséges különleges ötleteket keresnünk a megoldáshoz, mivel a feladat szövege majdnem mindent részletesen leírt. A részfeladatokra bontást is elvégezték helyettünk, most csak a részfeladatok megoldásait kell kidolgoznunk.

a. Adatok beolvasása a billentyűzetről (4 pont);

- A kép beolvasása (2 pont)

A képet a **kép** kétdimenziós négyzetes tömb tárolja, amelynek legtöbb **dimMax** = 10 sora és oszlopa van.

```
void beOlvasKep(int &m, int kep[][dimMax]){
    cout << "m = "; cin >> m;
    for (int i = 0; i < m; i++){
        for (int j = 0; j < m; j++){
            cin >> kep[i][j];
        }
    }
}
```

- A feldolgozás-sorozat beolvasása (2 pont)

```
void beFeldolgozasSorozat(int &k, char felDolgozas[]){
    cout << " k = "; cin >> k;
    for (int i = 0; i < k; i++){
        cin >> felDolgozas[i];
    }
}
```

b. Egy kép megfordítása (M művelet) (4 pont)

```
void megfordit(int m, int kep[][dimMax]){
    for (int i = 0; i < m; i++){
        for (int j = 0; j < m; j++){
            kep[i][j] = 1 - kep[i][j];
        }
    }
}
```

c. Elforgatás 90 fokkal (F művelet) (8 pont)

- a kép másolatának elkészítése (3 pont) és a kép forgatása (5 pont)

```
void forgatas(int m, int kep[][dimMax]){
    int masolat[dimMax][dimMax];
    for (int i = 0; i < m; i++){
        for (int j = 0; j < m; j++){
            masolat[i][j] = kep[i][j];
        }
    }
    for (int i = 0; i < m; i++){
        for (int j = 0; j < m; j++){
            kep[j][m - i - 1] = masolat[i][j];
        }
    }
}
```

d. A zoom művelet (Z) megvalósítása (8 pont)

- egy pixel bővítése (6 pont) és az új kép rámásolása a régre (2 pont)

```
void zoom(int &m, int kep[][dimMax]){
    int ujKep[dimMax][dimMax];
    int uj_I = 0;
    for (int i = 0; i < m; i++){
        uj_J = 0;
        for (int j = 0; j < m; j++){
            ujKep[uj_I][uj_J] = kep[i][j];
            ujKep[uj_I][uj_J + 1] = kep[i][j];
            ujKep[uj_I + 1][uj_J++] = kep[i][j];
            ujKep[uj_I + 1][uj_J++] = kep[i][j];
        }
        uj_I += 2;
    }
    m *= 2;
    for (int i = 0; i < m; i++){
        for (int j = 0; j < m; j++){
            kep[i][j] = ujKep[i][j];
        }
    }
}
```

e. A kép kiírása (2 pont)

```
void kiIrKep(int m, int kep[][dimMax]){
    for (int i = 0; i < m; i++){
        for (int j = 0; j < m; j++){
            cout << kep[i][j] << " ";
        }
        cout << endl;
    }
}
```

f. A kép átalakítása (3 pont)

```
void atalakitasok(int k, char felDolgozas[], int &m, int kep[][dimMax]){
    for (int i = 0; i < k; i++){
        if (felDolgozas[i] == 'I')
            megfordit(m, kep);
        if (felDolgozas[i] == 'R')
            forgatas(m, kep);
        if (felDolgozas[i] == 'Z')
            zoom(m, kep);
    }
}
```

g. Hívó programegység (main függvény/főprogram) (3 pont)

```
int main(){
    int m = 0;
    int kep[dimMax][dimMax];
    beOlvasKep(m, kep);
    int k = 0;
    char felDolgozas[dimMax];
    beFeldolgozasSorozat(k, felDolgozas);
    atalakitasok(k, felDolgozas, m, kep);
    kiIrKep(m, kep);
    return 0;
}
```

A helyes paraméterezésért járt még 5 pont, a megjegyzésekért, az indentálásért, a beszédes azonosítókért egyenként 2-2 pontot lehetett még szerezni.

6.1. Általánosságok – 2016

2016-ben nem voltak rácstesztetek. Az **I.** részben három feladatot tűzött ki a bizottság, a **II.** és **III.** részben egy-egy feladatot. Az **I.** rész első feladata egy úgynevezett modellezési feladat volt, amelynek megoldása – általában – rövid volt, de talán a legnehezebb. Ha a versenyzőnek nem sikerült ötletesen és – természetesen – hatékonyan megoldani a feladatot, volt még egy esélye, éspedig írt egy algoritmust, amely szimulálta a leírt eseményeket. A második és harmadik feladat megoldása egy-egy olyan algoritmus vagy alprogram volt, amelynek esetében szintén számított a megoldás hatékonysága. A **II.** részben egy iteratív algoritmus rekurzív változatát, vagy egy rekurzívnak az iteratív változatát kellett megírnia a versenyzőnek. A **III.** részben a feladat összetettebb volt, több alprogramot kért.

A versenyfeladatokat – mind a *Matek–Infó versenyen*, mind a *felvételin* – a következő figyelmeztető szöveg előzte meg:

A versenyzők figyelmébe:

1. Adjátok meg az alábbi feladatok megoldásait *pseudokódban* vagy a *líceumban tanult bármely programozási nyelvben (Pascal/C/C++)*!
2. A megoldásokra kapható pontszám első sorban az algoritmusok **helyességétől** függ, majd az algoritmusok **hatékonyságától** (ami a *végrehajtási időt* és a *felhasznált memória méretét* illeti).
3. Feltétlenül írjatok **megjegyzéseket** (kommenteket) amelyek segítik az adott megoldás megértését (írjátok le az azonosítók jelentését és a **megoldás során alkalmazott ötleteiteket**).
4. Ne használjatok különleges fejláblományokban definiált függvényeket (például STL, karakterláncokat feldolgozó sajátos függvények stb.).

Az értékelő/javítóbizottság számára is készült egy felhívó szöveg:

1. Az értékeléskor az algoritmus helyességére fektetjük a hangsúlyt, nem az esetleges szintaxishibákra.
2. Ha egy versenyző C/C++ programot írt, nem vonunk le a pontszámából, ha hiányzik valahonnan egy '{', '}', ';', '(', ')', ' ' karakter, de az esetlegesen hiányzó deklaráció, vagy redundáns deklaráció sem jár penalizációval.

3. Ha egy versenyző Pascal programot írt, nem vonunk le a pontszámából, ha hiányzik valahonnan egy ';', '(', ')', ' ' karakter, de az esetlegesen hiányzó **begin/end** kulcsszó, deklaráció, vagy redundáns deklaráció sem jár pontlevonással.
4. **Ha több ilyen hibát találunk**, és nyilvánvaló, hogy ezek nem csak azért jelentek meg, mert papíron nehéz ezekre a részletekre figyelni, a pontlevonást illetően az értékelést végző személy dönt.
5. A sorozatok indexelhetők 0-tól vagy 1-től kezdődően.
6. Ha a felvételiző egy alprogramot, amelyet a feladat kijelentése kér, több alprogramra bont, helyesnek tekintendő, amennyiben a felbontás indokolt.

A feladatlap néhány hasznos információval záródik:

Megjegyzések:

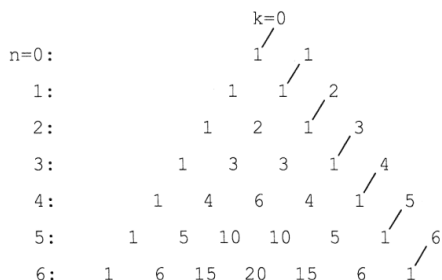
1. Minden tétel kidolgozása kötelező.
2. A megoldásokat a vizsgalapokra íjátok, (a piszkozatokat nem vesszük figyelembe).
3. Hivatalból jár 10 pont.
4. Rendelkezésekre áll 3 óra.

6.2. Matek–Infó verseny – 2016. április 16.

I. Tétel (50 pont)

1. Pascal háromszög (20 pont)

A *Pascal háromszög* egy olyan egyenlő szárú háromszög, amelynek több, természetes számokat tartalmazó vízszintes sora van: a két egyenlő száron az 1-es számjegy található. Egy adott n . sorban található minden érték a felette levő ($n - 1$). sor két szomszédos elemének összege, ha $n > 1$. A sorokat fentről lefele, 0-val kezdődően számozzuk, ahogy a mellékelt ábrán látható:



Írjatok alprogramot, amely generálja az r . sor elemeit ($2 \leq r \leq 32$), anélkül, hogy *kétdimenziós tömböt használhatok*. Bemeneti paraméter az r természetes szám, kimeneti paraméter az r . sor elemeinek sorozata.

Megoldás

A mellékelt ábrát tekintve, máris van több ötletünk, amelyeknek alapján generálhatnánk a Pascal háromszög sorait. De vigyáznunk kell a megkötésre: leírták a feladat szövegében, hogy ne használjunk kétdimenziós tömböt!

Az is kiderül a szövegből, hogy az n . sor elemeit az $(n - 1)$. sor elemeit használva lehet kiszámítani. Így máris megvan a megoldás: két sorozatot alkalmazunk, egy aktuálisat és egy újat, majd, minden lépés végén az „új”-ból „aktuális” lesz, és a következő lépésben generáljuk a következő „új”-at. (*maxPascal* = 33)

```
void PascalHaromszog(int r, int sor[]){
    int ujSor[maxPascal];           // maxPascal = 33
    sor[0] = 1;
    sor[1] = 1;
    for (int ri = 2; ri <= r; ri++){
        ujSor[0] = 1;
        ujSor[ri] = 1;
        for (int i = 1; i < ri; i++)
            ujSor[i] = sor[i - 1] + sor[i];
        for (int i = 0; i <= ri; i++)
            sor[i] = ujSor[i];
    }
}
```

2. Vírusok (10 pont)

Egy kísérlet során, egy n létszámú ($3 \leq n \leq 1\,000$) víruspopuláció a következőképpen változik:

- ha egy bizonyos óra kezdetekor a vírusok létszáma *páros* szám, akkor az óra végére a létszám 50%-kal csökken;
- ha egy bizonyos óra kezdetekor a vírusok létszáma *páratlan* szám, akkor az óra végére a létszám 1-gyel növekszik;
- ha egy bizonyos óra végén, a vírusok létszáma *szigorúan kisebb, mint a túléléshez szükséges kritikus szám*, akkor a vírusok populációja megsemmisül.

Írjatok alprogramot, amely meghatározza hány óra telik el, míg az n létszámú víruspopuláció megsemmisül. A túléléshez szükséges kritikus számot k -val jelöljük ($2 \leq k < n$), a megsemmisülésig eltelt órák számát **órákSz** jelöli. Az alprogram bemeneti paraméterei n és k , a kimeneti paramétere **órákSz** lesz.

Példa: ha $n = 11$ és $k = 3$, a populáció **órákSz** = 5 óra alatt semmisül meg.

Megoldás

A vírusok populációjának alakulását követhetjük iteratív vagy rekurzív algoritmussal. Semmi egyébbe nincs szükség, mint az algoritmus utasításaival szimulálni a vírusok sorsát, a leírásnak megfelelően.

```
int virusok(int n, int k){
    bool megsemmisult = n < k;
    int orakSz = 0;
    while(!megsemmisult){
        if(!(n & 1)) // n páros szám
            n /= 2;
        else
            n++;
        orakSz++;
        megsemmisult = n < k;
    }
    return orakSz;
}
```

A rekurzív algoritmus tömörebb, és nincs szükségünk a **megsemmisult** változóra. Az eredmény (amit a fenti algoritmusban az **órákSz** változóban számolt ki) a leállási feltétel **akkor** ágán kap 0 kezdőértéket. A rekurzív hívásokban az n aktuális paraméter értékét az n párosságának függvényében módosítjuk.

```
int virusokRek(int n, int k){
    if(n < k)
        return 0;
    else
        if(!(n & 1))
            return virusokRek(n / 2, k) + 1;
        else
            return virusokRek(n + 1, k) + 1;
}
```


3. Maximális szorzat (20 pont)

Adott az n elemű ($3 \leq n \leq 10\,000$), egész számokat tároló x sorozat, ahol az elemek értéke nagyobb, mint $-30\,000$ és kisebb, mint $30\,000$.

Írjatok alprogramot, amely meghatározza *három* számot az x sorozatból, amelyeknek a szorzata *maximális*. Az alprogram bemeneti paraméterei n és x , kimeneti paraméterei az a , b és c számok, amelyek az x sorozat elemei és rendelkeznek a kért tulajdonsággal. Ha létezik több megoldás, csak egyet kell megadni.

Példa: ha $n = 10$ és $a = (3, -5, 0, 5, 2, -1, 0, 1, 6, 8)$, a három szám: $a = 5$, $b = 6$, $c = 8$.

Megoldás

Az elemzés során először arra gondolunk, hogy hasznos lenne, ha rendeznénk a sorozatot. De ezt a gondolatot hamar elvetjük, hiszen ezáltal az algoritmusunk veszítené a hatékonyságából. Arra is gondolnunk kell, hogy a keresett három szám nem biztos, hogy a csökkenő sorrendben tekintett értékek közül az első három, hiszen előfordulhat, hogy a két – abszolút értékben vett – legkisebb negatív elem szorzata nagyobb, mint a két legnagyobb elem szorzata.

A fenti elmélkedésnek megfelelően, meghatározzuk a sorozat három legnagyobb értékét és a két legkisebbet. Ahhoz, hogy ne találjuk ezeket újra meg, minden kiválasztáskor, töröljük a sorozatból (felülírjuk a sorozat utolsó elemével) a megfelelő minimum- illetve maximumértéket, és n értékét minden kiválasztás után csökkentjük 1-gyel.

```
int minimum(int &n, int a[]){
    int m = a[0];
    int minind = 0;
    for (int i = 1; i < n; i++){
        if (a[i] < m){
            m = a[i];
            minind = i;
        }
    }
    a[minind] = a[n - 1];
    n--;
    return m;
}
```

```
int maximum(int &n, int a[]){
    int m = a[0];
    int maxind = 0;
    for (int i = 1; i < n; i++){
        if (a[i] > m){
            m = a[i];
            maxind = i;
        }
    }
    a[maxind] = a[n - 1];
    n--;
    return m;
}
```

Ezt az öt elemet a következőképpen vizsgáljuk:

```
int maxSzorzat(int n, int x[], int &a, int &b, int &c){
    int min1 = minimum(n, x);
    int min2 = minimum(n, x);
    int max1 = maximum(n, x);
    int max2 = maximum(n, x);
    int max3 = maximum(n, x);
}
```

```
if ((max1 > 0) && (min1 * min2 > max2 * max3)){  
    a = max1;  
    b = min1;  
    c = min2;  
}  
else {  
    a = max1;  
    b = max2;  
    c = max3;  
}  
}
```

II. Tétel (15 pont)

Legyen a következő alprogram, ahol az a bemeneti paraméter természetes szám ($0 < a \leq 30\,000$):

```
Algoritmus F(a):  
    b ← 0  
    p ← 1  
    Amíg a > 0 végezd el:  
        c ← a MOD 10  
        Ha c MOD 2 ≠ 0 akkor  
            b ← b + p * c  
            p ← p * 10  
        vége(ha)  
        a ← a DIV 10  
    vége(amíg)  
    térítsd b  
Vége(algoritmus)
```

- Adjátok meg annak a feladatnak a szövegét, amelyet ez az algoritmus old meg.
- Mit térít az $F(2103)$ hívás?
- Írjátok le az adott algoritmus *rekurzív* változatát, amelynek fejléce azonos az iteratív algoritmus fejlécével.

Megoldás

Előbb nézzük meg, mi a hatása az algoritmusnak? Látható, hogy a b változó értékét téríti, amely az algoritmus elején 0 kezdőértéket kap. A feldolgozás során b értékébe rendre beépülnek az a szám páratlan számjegyei balról jobbra, az eredeti sorrendjükben. Ennek érdekében az algoritmus felhasználja a p változót, amely a 10-nek különböző hatványait tárolja, a hatványkitevő növekvő sorrendjében. Így a b első számjegye az a szám első páratlan számjegye lesz. Ha az a szám nem tartalmaz egyetlen páratlan számjegyet sem, b értéke 0 marad.

Válaszolunk a kérdésekre:

a. Az algoritmus az a szám páratlan számjegyeiből felépíti a b számot, megőrizve a számjegyek eredeti sorrendjét. Ha az a számnak nincs páratlan számjegye, b értéke 0 lesz.

b. $F(2103) = 13$

c. Egy lehetséges, helyes rekurzív algoritmus, amelynek fejléce azonos az iteratív algoritmus fejlécével:

```
int FRek(int a){
    if (a < 1)
        return a;
    else{
        int c = a % 10;
        if (c % 2 != 0) // páratlan számjegy esete
            return c + 10 * FRek(a / 10); // beépítjük a térítendő értékbe
        else
            return FRek(a / 10); // páros számjegy, figyelmen kívül hagyjuk
    }
}
```

III. Tétel (25 pont)

Ciklikus palindrom

Egy természetes számokat tartalmazó sorozatot *palindrom*nak nevezünk, ha balról jobbra olvasva, ugyanazt a sorozatot kapjuk, mintha jobbról balra haladva olvasnánk. Például az (1, 2, 3, 2, 1) sorozat *palindrom*, míg az (1, 2, 3, 2, 4) nem *palindrom*. Egy természetes számokat tartalmazó sorozat *ciklikus palindrom*, ha az elemeinek néhány körkörös permutációjával *palindrommá* alakítható. Az elemek körkörös permutációja alatt a sorozat elemeinek egy pozícióval balra tolását értjük (kivételet képez az első elem, amely a sorozat utolsó pozíciójára kerül).

Írjatok programot, amely eldönti, hogy az n elemű ($1 \leq n \leq 1\,000$), természetes számokat tartalmazó a sorozat *ciklikus palindrom-e* vagy *sem*, és kiír egy megfelelő üzenetet (*Igen/Nem*). Ha a döntés eredménye *Igen*, a program meghatározza azoknak a körkörös permutációknak a számát, amelyekkel a sorozat palindrommá alakítható.

1. Példa: az $a = (1, 1, 2, 2)$ sorozat az $(1, 2, 2, 1)$ egyetlen körkörös permutációval palindrommá alakítható.

2. Példa: az $a = (3, 4, 3, 2, 1, 1, 2)$ sorozat öt körkörös permutációval palindrommá alakítható: $(4, 3, 2, 1, 1, 2, 3)$; $(3, 2, 1, 1, 2, 3, 4)$; $(2, 1, 1, 2, 3, 4, 3)$; $(1, 1, 2, 3, 4, 3, 2)$; $(1, 2, 3, 4, 3, 2, 1)$.

3. Példa: az $a = (1, 2, 3)$ sorozat nem alakítható palindrommá körkörös permutációkkal.

Írjatok egy-egy alprogramot, amely:

- beolvassa az **a** sorozatot a billentyűzetről;
- kiírja az *Igen/Nem* üzenetet a képernyőre és *Igen* esetében az elvégzendő körkörös permutációk számát;
- elönti egy sorozatról, hogy ciklikus palindrom-e vagy sem;
- meghatározza a szükséges körkörös permutációk számát.

Megoldás

A feladat szövegében meghatározták a részfeladatokat, nekünk csak a megfelelő alprogramokat kell megvalósítanunk.

a. A sorozat hosszának és elemeinek beolvasása (1 pont)

```
void beOlvas(int & n, int a[]){
    // beolvassuk a sorozat hosszát és az elemeit
    cin >> n;                // az a sorozat hossza
    for (int i = 0; i < n; i++)
        f >> a[i];          // az a sorozat elemei
}
```

b. Az eredmény kiírása (2 pont)

```
void kiIr(int n, int a[]){
    // ha az a sorozatot, lehetséges palindrommá alakítani
    // kiírjuk a szükséges permutációk számát
    int permSz = ciklikusPalindrom_v1(n, a);
    if (permSz != -1)
        cout << "Igen " << permSz << endl;
    else
        cout << "Nem" << endl;
}
```

c. A „ciklikus palindrom sorozat” tulajdonság ellenőrzése (4 pont) és

d. A szükséges körkörös permutációk számának meghatározása (9 pont)

A két követelmény teljesítését együtt mutatjuk be, mivel a részfeladatok egymásra épülnek. Bemutatunk két lehetséges megoldást.

V1: Az első megoldásban – eltérően a másodiktól – nem generáljuk a sorozat körkörös permutációit. Ehelyett az **a** sorozatot meghosszabbítjuk úgy, hogy a végére másoljuk a teljes **a** sorozatot. Ebben a sorozatban megtalálhatók az **a** sorozat körkörös permutációi, amelyek **n** hosszúságú tömbszakaszai ennek a megkétszerezett sorozatnak. Minden lépésben eldöntjük egy-egy ilyen tömbszakaszcól, hogy palindrom tulajdonságú, vagy sem. A tömbszakaszok első elemének indexét az *eleje* változóban, az utolsóét a *vége* változóban tároljuk. Két alprogramunk lesz:

V1_1: A `palindrom(n, a, eleje, vege)` algoritmus eldönti, hogy az ***a*** sorozat aktuális tömbszakasza palindrom tulajdonságú vagy sem. A bemeneti ***eleje*** paraméter a vizsgálandó tömbszakasz első elemének, a ***vége*** paraméter az utolsó elemének indexe. Az algoritmusban rendre összehasonlítjuk a sorozat (tömbszakasz) két végén, illetve a végektől egyenlő távolságra levő elemeket. A sorozat bejárása addig tart, amíg az elemek egyenlőek, illetve a két összehasonlítandó elem indexe azonosná nem válik. Ha sikerült bejárni a sorozatot az ***eleje*** és a ***vége*** között, az alprogram ***igaz***-at térít, különben ***hamis***-at.

```
bool palindrom(int n, int a[], int eleje, int vege){
// eldöntjük, hogy az [eleje..vege] tömbszakasz palindrom-e vagy sem
    int i = eleje; int j = vege;
    while (a[i] == a[j] && i < j){
        i++;
        j--;
    }
    return i == j;
}
```

V1_2: A `palindrom(n, a, eleje, vege)` alprogramot meghívja a ciklikus-Palindrom_v1(`n, a`) alprogram, amely implementálja az előbb leírt trükköt. Így elérjük, hogy nem kell generálnunk a körkörös permutációkat és rendre vizsgáljuk (legtöbb ***n***-szer), hogy az ***eleje*** és ***vége*** közti tömbszakasz palindrom-e vagy sem.

Amikor a `palindrom(n, a, eleje, vege)` algoritmus ***igaz***-at térít, leállunk, hiszen megtaláltuk az ***a*** sorozat azon körkörös permutációját, amely palindrom tulajdonságú. Térítjük az ***(i - 1)*** értéket, mivel az **Amíg** ciklus akkor talált palindrom tulajdonságú részt, amikor az ***i***. lépésben hívta meg a `palindrom(n, a, eleje, vege)` alprogramot (de utána ***i*** még nőtt eggyel). Ez az a szám, ahányszor a körkörös permutálást el kellett volna végeznünk. Ha az **Amíg**-ből való kilépés azután történik, miután az ***n***. körkörös permutációnak megfelelő lépést is végrehajtottuk, (következne az ***(n + 1)***.), levonjuk a következtetést, hogy az ***a*** sorozat nem alakítható palindrommá körkörös permutálásokkal, és **-1**-et térítünk.

```
int ciklikusPalindrom_v1(int n, int a[]){
// eldöntjük, hogy az a sorozat ciklikus palindrom vagy sem
    for (int i = 0; i < n; i++){
        a[n + i] = a[i];
        bool talalt = false;
        int i = 0;
        while (!talalt && (i < n)){
            talalt = palindrom(n, a, i, i + n - 1);
            i++;
        }
        if (talalt)
            return i - 1;
        else
            return -1;
    }
}
```

V2: A második megközelítés is helyes, de kevésbé hatékony, mivel elvégzi a körkörös permutálásokat. Másfelől nyer az olvashatóságában. (Egy-egy „trükkös” megoldás elég sok rizikót is jelent, mivel előfordulhat, hogy értékeléskor nem értik meg a trükk szükségességét vagy – mint itt is – a szépségét.)

Ezt a megoldást három alprogrammal valósítjuk meg:

V2_1: A `pali(n, a)` alprogram eldönti, hogy az **a** sorozat palindrom tulajdonságú vagy sem:

```
bool pali(int n, int a[]){
    // eldöntjük, hogy az a sorozat palindrom tulajdonságú vagy sem
    int i = 0;
    int fele = n / 2;
    bool ok = true;
    while(ok && (i < fele)){
        if (a[i] != a[n - i - 1])
            ok = false;
        i++;
    }
    return ok;
}
```

V2_2: Meghatározzuk a sorozat elemeinek egy körkörös permutációját:

```
void permutacio(int n, int a[]){
    // elvégzünk egy körkörös permutációt az a sorozaton (1 pozícióval balra)
    int aux = a[0];
    for (int i = 0; i < n - 1; i++)
        a[i] = a[i + 1];
    a[n - 1] = aux;
}
```

V2_3: Az alábbi algoritmus előbb megvizsgálja az eredeti sorozatot, mivel az is lehetséges, hogy ez palindrom tulajdonságú és akkor máris vége van a feldolgozásnak. Ekkor a szükséges körkörös permutálások száma 0. Ha a sorozat nem palindrom tulajdonságú, akkor az algoritmus elvéggez legtöbb $(n - 1)$ körkörös permutálást, és vizsgálja minden új **a** sorozatra a palindrom tulajdonságot. A *permSz* változó a szükséges körkörös permutálások száma.

```
int ciklikusPalindrom_v2(int n, int a[]){
    // eldöntjük, hogy az a sorozat ciklikus palindrom vagy sem
    bool ok = false;
    int permSz = 0;
    if (pali(n, a)){ // előfordulhat, hogy az eredeti sorozat palindrom
        permSz = 0;
        ok = true;
    }
    else {
```

```
while (!ok && permSz < n){
    permutacio(n, a);
    ok = pali(n, a);        // ha a körkörös permutáció végrehajtása
    permSz++;               // palindromot eredményezett
}
}
if (ok)
    return permSz;
else
    return -1; // ha nincs palindrom tulajdonságú permutáció
}
```

e. Hívó programegység (main függvény/főprogram) (2 pont)

```
int main(){
    int n;
    int a[maxDim];
    beOlvas(n, a);
    kiir(n, a); // a kiir alprogram hívja meg a sorozat feldolgozását
               // végző alprogramot
}
```

A helyes paraméterezésért járt még 4 pont, a megjegyzésekért, az indentálásért, a beszédes azonosítókért egyenként 1-1 pontot lehetett még szerezni.

6.3. Felvételi verseny – 2016. július 22.

I. Tétel (50 pont)

1. Tornyok (10 pont)

Legyen megfelelő darabszámú azonos méretű érme, amelyekből tornyok építendők a következő szabályok alapján:



1. a legmagasabb torony magassága n ($0 < n \leq 13$), a legkisebbnek a magassága 1;
2. a tornyok úgy kerülnek egymás mellé, hogy bármely két, azonos magasságú torony között létezik legalább egy magasabb torony, mint ez a kettő.

Írjatok alprogramot, amely kiszámítja azt a *legnagyobb toronyszámot* (*tornyokSzáma*), amelyek felépíthetők az adott szabályok alapján és az építkezéshez szükséges *érmék számát* (*érmékSzáma*). A legnagyobb toronymagasság n értéke az alprogram bemeneti paramétere, a *tornyokSzáma* és az *érmékSzáma* kimeneti paraméterek.

Példa: ha $n = 3$, *tornyokSzáma* = 7 és *érmékSzáma* = 11.

Megoldás

Rajzolgatunk és rájövünk, hogy ha lerajzoltuk azokat a tornyokat, amelyek megfelelnek egy bizonyos n értéknek, majd növeljük az n értékét, hogy lerajzoljuk a beszúrandó tornyokat, a tornyok száma 2^{n-1} -gyel nő.

Tehát, ahhoz, hogy megadhassuk a tornyok számát, generáljuk a kettőhatványokat (2^{n-1} -ig), majd összeadjuk őket. Vigyáznunk kell a hatékonyságra, tehát egy-egy kettőhatvány kiszámítását nem kezdjük mindig előlről, hanem felhasználjuk az előző lépésben kiszámítottak az értékét.

V1: A *tornyokSzáma* és *érmékSzáma* értékeket egyetlen ismétlő struktúrával számítjuk ki: (10 pont)

```
Algoritmus tornyok_v1a(n, tornyokSzáma, érmékSzáma):
    érmékSzáma ← 0
    tornyokSzáma ← 0
    hatvány ← 1
    Minden magasság = n, 1, -1 végezd el:
        tornyokSzáma ← tornyokSzáma + hatvány
        érmékSzáma ← érmékSzáma + hatvány * magasság
        // minden toronyban „magasság” darab érme van
        hatvány ← hatvány * 2
    vége(minden)
Vége(algoritmus)
```


V2: A következő algoritmus is helyes és a hatékonysága azonos az előbbi algoritmus hatékonyságával:

```

Algoritmus tornyok_v1b(n, tornyokSzama, érmekSzama):
    érmekSzama  $\leftarrow$  n
    tornyokSzama  $\leftarrow$  1
    magasság  $\leftarrow$  n
    hatvány  $\leftarrow$  1
    Minden i = 1, n-1 végezd el:
        hatvány  $\leftarrow$  hatvány * 2
        magasság  $\leftarrow$  magasság - 1
        tornyokSzama  $\leftarrow$  tornyokSzama + hatvány
        érmekSzama  $\leftarrow$  érmekSzama + hatvány * magasság
    vége(minden)
Vége(algoritmus)

```

2. Bűvös számok (20 pont)

Legyen két természetes szám p és q ($2 \leq p \leq 10$, $2 \leq q \leq 10$). Egy természetes számot *bűvösnek* nevezünk, ha a p számrendszerben felírt alakjában szereplő számjegyek halmaza azonos a q számrendszerben felírt alakjában szereplő számjegyek halmazával. Például, ha $p = 9$ és $q = 7$, $(31)_{10}$ *bűvös szám*, mivel $(34)_9 = (43)_7$; ha $p = 3$ és $q = 9$, $(9)_{10}$ *bűvös szám*, mivel $(100)_3 = (10)_9$.

Írjatok alprogramot, amely adott p és q számrendszerek ismeretében, meghatározza azt a *bűvös* számokból álló x sorozatot, amely minden 0-nál szigorúan nagyobb és adott n ($1 < n \leq 10\,000$) természetes számnál szigorúan kisebb számot tárol. Az alprogram bemeneti paraméterei p és q (a két alap) és az n szám. Kimeneti paraméter az x sorozat és ennek k hossza.

Példa: ha $p = 9$, $q = 7$ és $n = 500$, az x sorozatnak $k = 11$ eleme lesz: (1, 2, 3, 4, 5, 6, 31, 99, 198, 248, 297).

Megoldás

Adott egy szám, amelynél kisebb és – adott tulajdonsággal rendelkező számokból – létre kell hoznunk egy új sorozatot. A feladat szövegében megtaláljuk a tulajdonság (*bűvös szám*) definícióját és megjegyezzük, hogy a tulajdonság ellenőrzéséhez a szám adott számrendszer szerinti számjegyeinek halmazát kell összehasonlítani ugyanannak a számnak egy másik számrendszerben érvényes számjegyhalmazával. Az észrevétel máris sugalmazza, hogy – valószínűleg – hasznos lesz a nyilvántartást az előfordulások tömbjeivel végezni. (20 pont)

V1_1: A feladat megoldása két tömb használatával rendkívül egyszerű:

- meghatározzuk az adott szám számjegyeinek *előfordulási* tömbjét (mindkét számrendszerben);
- összehasonlítjuk a két tömböt, és eldöntjük, hogy a két tömb azonos-e.

```

void elofordul(int szam, int alap, bool elof[]){
    // létrehozuk a szam számjegyeinek előfordulástömbjét
    // (az alap számrendszerben)
    for (int i = 0; i <= 9; i++){
        elof[i] = false;
    }
    while (szam > 0){
        elof[szam % alap] = true;
        szam /= alap;
    }
}

```

A két halmaz azonosságának vizsgálatát a két előfordulási tömb elemeinek összehasonlítása révén valósítjuk meg.

```

bool azonos_1(int szam, int p, int q){
    // vizsgáljuk, hogy a két előfordulási tömb azonos-e
    bool elofP[10], elofQ[10];
    elofordul(szam, p, elofP);
    elofordul(szam, q, elofQ);
    int i = 0;
    while (i <= 9 && elofP[i] == elofQ[i])
        i++;
    return i > 9;
}

```

A bűvös számok sorozatát létrehozó alprogram:

```

void buvosSzamok_v1(int p, int q, int n, int & k, int x[]){
    // generáljuk a bűvös számok sorozatát
    k = 0; // még nem találtunk egyetlen bűvös számot sem
    for (int szam = 1; szam < n; szam++){
        // megvizsgálunk minden, n-nél kisebb számot
        if (azonos_1(szam, p, q)){
            x[k] = szam; // a számot elmentjük az x sorozatba
            k++;
        }
    }
}

```

V1_2: A feladat megoldása egy tömb használatával szintén egyszerű:

- csak a **p** számrendszer szerinti *előfordulási* tömböt hozzuk létre;
- meghatározzuk rendre a szám számjegyeit a **q** számrendszerben, és vizsgáljuk, hogy ez a számjegy előfordult-e a **p** számrendszerben felírt számokban; ha minden számjegyet megtaláltunk, következik, hogy a két halmaz azonos, vagyis a szám bűvös.

```

bool ellenoriz(int szam, int p, int q){
    // az elofP tömbben keressük a szám számjegyeit, amelyeket
    // a q számrendszerben felírt alakja tartalmaz
    bool elofP[10];
    elofordul(szam, p, elofP);
    while (szam > 0 && elofP[szam % q])
        szam /= q;
    return szam == 0; // minden számjegyet megtaláltunk
}

```

Ahhoz, hogy helyes eredményhez jussunk, ezt az algoritmust szükséges kétszer is meghívni, hiszen ahhoz, hogy eldönthessük, hogy a két halmaz (amelyek közül ez az algoritmus csak egyet hoz létre) azonos, meg kell néznünk, fordítva is: a szám számjegyei között, amelyeket a p számrendszerben tartalmaz, megtalálható-e minden számjegy, amely előfordult a q számrendszerben felírt számban.

```
bool azonos_2(int szam, int p, int q){
    return ellenoriz(szam, p, q) && ellenoriz(szam, q, p);
}
```

V1_3 A harmadik megoldás nem használ egyetlen tömböt sem, de más nehézségbe ütközik. A *szám* értékére egymás után többször is szükségünk lesz, de az algoritmus osztja ezt a számot, amíg 0-vá nem válik. Ebből kifolyólag másolatot készítünk róla. Ráadásul két másolatra van szükségünk, mivel a számot mindkét számrendszerben osztjuk, abból a célból, hogy *szám* számjegyeit a p számrendszerben összehasonlíthassuk minden számjegyével a q számrendszerben.

```
bool szamjegyekPQ(int szam, int p, int q){
    // vizsgáljuk, hogy a szám minden számjegye (p számrendszerben)
    // előfordul-e a számban (q számrendszerben)
    int szamQ = szam; // másolat szam-ról
    int szamP = szam; // másolat szam-ról
    while (szamP > 0){ // vizsgáljuk, hogy a szám számjegyei p számrendszerben
        // megtalálható-e a q számrendszerben felírt számban is
        int utolsoSzj = szamP % p; // utolsó számjegy a p számrendszerben
        while (szamQ > 0 && utolsoSzj != szamQ % q)
            szamQ /= q;
        if (szamQ == 0)
            // ha szamQ 0 lett, a szám aktuális számjegye (a p számrendszerben)
            // nincs meg a szám q számrendszerben felírt alakjában
            return false;
        szamQ = szam; // visszaállítjuk szamQ értékét
        szamP /= p; // levágjuk szamP utolsó számjegyét
    }
    return true;
}
```

Ezt az algoritmust szintén kétszer hívjuk meg, hiszen ahhoz, hogy eldönthessük, hogy a két halmaz (amelyeket ez az algoritmus nem hoz létre) azonos, meg kell néznünk, fordítva is: a szám számjegyei között, amelyeket a p számrendszerben tartalmaz, megtalálható-e minden számjegy, amely előfordult a q számrendszerben felírt számban;

```
bool azonos_3(int szam, int p, int q){
    return (szamjegyekPQ(szam, p, q) && szamjegyekPQ(szam, q, p));
}
```

3. Beszúrás (20 pont)

Adott az n elemű ($0 < n \leq 10\,000$) a sorozat, amelynek elemei 30 000-nél kisebb szigorúan pozitív természetes számok.

Írjatok alprogramot, amely a sorozat minden eleme után beszúr egy új számot, amely 2-nek az a legnagyobb hatványa, amely kisebb vagy egyenlő az adott elem értékével. Az a sorozat és az n értéke az alprogram bemeneti és egyben kimeneti paraméterei.

Példa: ha $n = 4$ és $a = (3, 1, 24, 9)$, akkor az új a sorozat: $(3, 2, 1, 1, 24, 16, 9, 8)$, valamint $n = 8$.

Megoldás

Több lehetséges megoldásra is gondolunk, de előre látjuk, hogy egy hatékony algoritmus egyszer fogja bejárni a sorozatot és nem használ segéd adatszerkezeteket. (20 pont)

Előbb lássunk egy kettőhatványokat generáló alprogramot. Az alábbi alprogramban a balra tolás művelete 2-vel történő szorzással helyettesíthető:

```
int kettoHatvany(int x){  
    // meghatározzuk az x-nél kisebb, legnagyobb kettőhatványt  
    int hatvany = 1;  
    while (hatvany < x)  
        hatvany = hatvany << 1;  
    // a hatvány változó értékének ábrázolását 1-gyel balra toljuk  
    if (hatvany > x)  
        return hatvany / 2;  
    else  
        return hatvany;  
}
```

V1: Abból a célból, hogy a sorozatot ne kelljen minden lépésben eltolni egy-elemmel jobbra (ezáltal helyet szorítva a kettőhatványnak), a sorozatot a végétől az eleje felé haladva dolgozzuk fel. Egy bizonyos lépésben elhelyezzük az aktuális elemet a végleges helyére, majd a kettőhatványt utána. A sorozatot kettőszel járjuk be. Minden elemet egyszer dolgozunk fel, segéd adatszerkezet nélkül.

```
void beszurHatvanyokat_1(int n, int a[]){  
    // minden elem után beszúrjuk a legközelebbi, kisebb kettőhatványt  
    int i = 2 * n - 1; // az utolsó elem indexe a megváltozott sorozatban  
    int j = n - 1; // a sorozat utolsó elemének eredeti indexe  
    while (i > 0) {  
        a[i - 1] = a[j]; // az eredeti utolsó elemet a végleges helyére tesszük  
        a[i] = kettoHatvany(a[j]); // a következő elem a kettőhatvány  
        i -= 2; // a következő feldolgozandó elem indexe  
        j--;  
    }  
}
```

V2: Ha az előbbi megoldásra nem jövünk rá, bejárjuk a sorozatot balról jobbra és a kért (megváltoztatott) sorozatot egy másik sorozatban hozzuk létre. Ezt rá-másoljuk az eredetire és az eredeti sorozat hosszának értékét megkétszerezzük:

```
void beszurHatvanyokat_2(int &n, int a[]){
    // minden elem után beszúrjuk a legközelebbi, kisebb kettőhatványt
    int b[maxDim];
    int i = 0;
    int j = 0;
    while (i < n){
        b[j] = a[i]; b[j + 1] = kettoHatvany(a[i]);
        i++;
        j += 2;
    }
    n = 2 * n; // a sorozat n hossza megváltozik, n bemenet és kimeneti paraméter
    for (int i = 0; i < n; i++)
        a[i] = b[i];
}
```

V3: Ha előbb kiszámítjuk a kettőhatványokat és tároljuk őket egy sorozatban, majd „összefésüljük” az eredetivel egy harmadikba, az algoritmus szintén nem dolgozik helyben, de nem szükséges négyzetes algoritmust írunk. Az új sorozat elemeit rámásoljuk az eredetire és az eredeti sorozat hosszát megduplázzuk.

```
void beszurHatvanyokat_3(int &n, int a[]){
    int b[maxDim], c[maxDim];
    for (int i = 0; i < n; i++)
        b[i] = hatvany(a[i]);
    int i = 0;
    int j = 0;
    while (j < n){
        c[i] = a[j]; c[i + 1] = b[j];
        i += 2;
        j++;
    }
    n = 2 * n; // a sorozat n hossza megváltozik, n bemenet és kimeneti paraméter
    for (int i = 0; i < n; i++)
        a[i] = c[i];
}
```

V4: Lássunk egy négyzetes bonyolultságú algoritmust is:

```
void beszurHatvanyokat_3(int &n, int a[]){
    int i = 1;
    while (i <= n) {
        int j = n;
        while (j > i){
            a[j] = a[j - 1];
            j--;
        }
        a[i] = kettoHatvany(a[i - 1]);
        i += 2;
        n++;
    }
}
```

II. Tétel (15 pont)

Adott a következő alprogram, ahol az a és b ($0 < a \leq 10\,000, 0 \leq b \leq 10\,000$) természetes számok bemeneti paraméterek.

```

Algoritmus F(a, b):
  c ← 1
  Amíg b > 0 végezd el:
    Ha b MOD 2 = 1 akkor
      c ← (c * a) MOD 10
    vége(ha)
    a ← (a * a) MOD 10
    b ← b DIV 2
  vége(amíg)
  térít c
Vége(algoritmus)

```

- Adjátok meg annak a feladatnak a szövegét, amelyet ez az algoritmus old meg.
- Mit térít az $F(1002, 6)$ hívás?
- Írjátok le egy *rekurzív* változatát a fenti iteratív (nem rekurzív) algoritmusnak. A fejléce legyen azonos a fenti algoritmus fejlécével.

Megoldás

- Az algoritmus ismerősnek tűnhet sok feladatmegoldó számára, mivel hasonlít *al-Kwarizmi* módszeréhez, amely hatékonyan számítja ki a^b értékét. De van egy furcsa különbség: a térített érték, amely itt a c változóban jön létre, a „gyorshatvány” néven ismert algoritmusban, a **Ha** utasítás akkor ágán a $c * a$ új értéket kapja. Itt viszont $(c * a) \bmod 10$ található. A másik különbség a **Ha** utasítás után látható, ahol $a * a$ helyett $(a * a) \bmod 10$ található. Ha nem jövünk rá, hogy az algoritmus mit számol ki, végrehajthatjuk egy-két egyszerű példára, amíg világossá nem válik, hogy a térített érték az a^b érték utolsó számjegye. (5 pont)
- $F(1002, 6) = 4$ (3 pont)
- Az algoritmusnak egy lehetséges, helyes rekurzív változata, amelynek a fejléce azonos az iteratív algoritmuséval:

```

Algoritmus FRek(a, b):
  Ha b = 0 akkor
    térítsd 1
  különben
    Ha b MOD 2 = 1 akkor
      térítsd a MOD 10 * FRek(a*a MOD 10, b DIV 2) MOD 10
    különben
      térítsd FRek(a*a MOD 10, b DIV 2) MOD 10
    vége(ha)
  vége(ha)
Vége(algoritmus)

```

III. Tétel (25 pont)

Szigetek

Egy légitársaság az utasok rendelkezésére bocsátotta azoknak a földrajzi pontoknak a magasságait tartalmazó sorozatot, amelyek fölött a repülőgép száll majd Kolozsvár és New York között. Az a sorozatnak n darab ($3 \leq n \leq 10\,000$), 30 000-nél szigorúan kisebb természetes szám eleme van. A szárazföldnek megfelelő pontok magasságai 0-tól különböznek, míg az óceánnak megfelelő pontok magasságai egyenlők 0-val. *Sziget*-nek olyan egymás utáni szárazföldnek megfelelő pontok sorozatát nevezzük, amely előtt és után víz található.

Írjatok programot, amely:

1. Meghatározza és kiírja a leghosszabb sziget kezdetét, valamint a végét jelző pont sorszámát. Ha több megoldás létezik, csak egyet kell meghatároznotok. Ha nem létezik egyetlen sziget sem, akkor kiírja „*Nem létezik sziget*”.
2. Eldönti, hogy a leghosszabb sziget *hegy* típusú-e vagy sem és kiírja a „*Hegy*”, illetve a „*Nem hegy*” üzenetet. Egy sziget *hegy* típusú, ha a felszínén található magasságok egy adott elemig szigorúan növekvő – nem üres – sorozatot alkotnak, majd szigorúan csökkenőt, amely nem üres.
3. Eldönti, hogy a szárazföldnek megfelelő pontok magasságai páronként *különböző értékek* vagy sem és kiírja a „*A magasságok különbözők*”, illetve a „*A magasságok nem különbözők*” üzenetet.
4. Ha a 3. pontban feltett kérdésre a válasz „*nem*”, meghatározza és kiírja a leggyakoribb magasság értékét és az előfordulásainak számát. Ha több ilyen magasság létezik, csak egyet kell megadnotok.

1. Példa: ha $n = 15$ és $a = (10, 2, 1, 0, 7, 0, 1, 2, 13, 5, 0, 0, 8, 5, 2)$, összesen 2 sziget van, a leghosszabb sziget a 7. és 10. sorszámú magasságok között található. A legnagyobb sziget *hegy* típusú. A magasságok értékei nem különbözők, és a leggyakoribb magasság értéke 2, amely 3-szor fordul elő.

2. Példa: ha $n = 10$ és $a = (1, 2, 0, 1, 2, 13, 0, 0, 1, 2)$, egyetlen sziget létezik, amely a 4. és 6. sorszámú pontok között található és *nem hegy* típusú. A magasságok értékei nem különbözők, az egyik leggyakoribb magasság értéke 1 és 3-szor fordul elő.

Megjegyzés: A példákban, az a sorozatot 1-től kezdve indexeltük.

Írjatok egy-egy alprogramot, amely:

- a. beolvassa az a sorozat hosszát, és az a sorozatot a billentyűzetről;
- b. meghatározza a leghosszabb sziget kezdetének, valamint a végének megfelelő pont sorszámát;

- c. eldönti, hogy egy sziget *hegy* típusú vagy sem;
- d. eldönti, hogy a szárazföldnek megfelelő pontok magasságai páronként *különböző értékek* vagy sem;
- e. meghatározza a szárazföldön található leggyakoribb magasságot és ennek a magasságnak megfelelő előfordulások számát.

Megoldás

A feladat szövegében meghatározták a részfeladatokat, nekünk csak a megfelelő alprogramokat kell megvalósítanunk.

a. Adatok beolvasása (1 pont)

```
void beOlvas(int & n, int a[]){  
    // adatok beolvasása  
    cin >> n;  
    for (int i = 1; i <= n; i++)  
        f >> a[i];  
}
```

b. Eredmények kiírása (1 pont)

```
void kiirEredmeny(int szigetekSzama, int eleje, int vege,  
                  bool hegy, bool kulonbozok, int magassag, int k){  
    // az eredmények kiírása  
    if (szigetekSzama == 0)  
        cout << "Nincs sziget!" << endl;  
    else{  
        cout << "Szigetek szama: " << szigetekSzama << endl;  
        cout << "A legnagyobb sziget a(z) " << eleje << ". es a(z) " << vege;  
        cout << ". pontok kozott talalhato." << endl;  
        if (hegy)  
            cout << "Hegy." << endl;  
        else  
            cout << "Nem hegy." << endl;  
    }  
    if (kulonbozok)  
        cout << "A magassagok kulonboznek." << endl;  
    else{  
        cout << "A magassagok nem kulonboznek." << endl;  
        cout << "A leggyakoribb magassag: " << magassag;  
        cout << " es " << k << "-szor fordul elo." << endl;  
    }  
}
```

c. A szigetek számának és a legnagyobb szigetnek meghatározása (5 pont)

A szigetek számának és a legnagyobb szigetnek meghatározására tervezett algoritmus alapötlete arra az algoritmusra támaszkodik, amellyel azt a leghosszabb tömbszakaszt határozzuk meg, amely csak szigorúan pozitív számokat tartalmaz. Vigyázunk arra is, hogy a kontinenseket ne tévesszük össze a szigetekkel. A hatékonyság érdekében minden elemet csak egyszer dolgozunk fel, és nem használunk segéd adatszerkezeteket:


```

void repulesSzimulalasa(int n, int a[], int & szigetekSzama,
                        int & maxKezd, int & maxVeg){
    // meghatározzuk a leghosszabb pozitív számokból álló tömbszakaszt
    int eleje, vege, i;
    maxKezd = 0;
    maxVeg = 0;
    szigetekSzama = 0;           // még nem találtunk egy szigetet sem
    i = 1;
    while (i <= n && a[i] > 0)    // repülünk Európa fölött
        i++;
    while (i <= n){               // ha nem ért véget az utazás
        while (i <= n && a[i] == 0) // repülünk az óceán fölött
            i++;
        eleje = i;               // egy sziget kezdete, ha valóban sziget
        while (i <= n && a[i] > 0) // repülünk a sziget fölött
            i++;
        if (i < n){              // ha nem ért véget az utazás,
                                // vagyis legutoljára nem Amerika fölött repültünk
            szigetekSzama++;     // nő a szigetek száma
            vege = i - 1;        // az aktuális sziget vége
            if(maxVeg - maxKezd <= vege - eleje){
                                // aktualizáljuk a legnagyobb szigetet
                maxKezd = eleje;
                maxVeg = vege;
            }
        }
    }
}

```

d. Annak eldöntése, hogy a sziget *hegy* típusú-e (3 pont)

Ahhoz, hogy egy sorozat *hegy* típusú legyen, két olyan tömbszakaszból kell állnia, ahol az első szigorúan növekvő, a második szigorúan csökkenő. Mivel csak a legnagyobb szigetet kell megvizsgálnunk, a magasságok *a* sorozatában csak az *eleje* és *vége* sorszámú pontok közé eső tömbszakaszt vizsgáljuk.

```

bool hegyTulajdonsag(int eleje, int vege, int a[]){
    // vizsgáljuk, hogy a sziget "hegy"-típusú-e
    bool hegy;
    int i = eleje;
    while (i < vege && a[i] < a[i+1]) // vizsgáljuk, hogy a sorozat növekvő-e
        i++;
    hegy = i < vege;                 // ha nincs vége a szigetnek
    if (hegy)
        while(i < vege && a[i] > a[i+1]) // vizsgáljuk, hogy a sorozat növekvő-e
            i++;
    hegy = hegy && (i == vege);      // ha a sorozat előbb növekvő, majd csökkenő
    // és véget ért a sziget, megvan a "hegy" tulajdonság
    return hegy;
}

```

e. A szárazföldi magasságok értékei különbözők vagy sem? (3 pont)

Összehasonlítsuk rendre a 0-tól különböző elemeket az összes többi nem 0 elemmel és leállunk azonnal, ha találtunk két azonos értéket. Ha sikerült bejárni a sorozatot, a térített érték *igaz*, különben *hamis*.

```
bool kulonbozok_e(int n, int a[]){
    int i = 1;                // a magasságok különbözőségének vizsgálata
    bool kulonbozok = true;   // feltételezzük, hogy a magasságok különbözők
    while (kulonbozok && (i < n)){
        while (a[i] == 0)      // csak a szárazföld pontjai érdekelnek
            i++;              // az "óceán" fölött repülünk
        int j = i + 1;         // az i. magasság szárazföldön van
        while (j <= n && a[i] != a[j]) // a különböző magasságok fölött repülünk
            j++;
        kulonbozok = j > n;    // ha nem találtunk két azonos magasságot
        i++;                  // következő magassági pont
    }
    return kulonbozok;
}
```

f. A leggyakoribb magasság meghatározása és előfordulásainak száma (3 pont)

Ha a szárazföldön található magasságok *nem különbözők*, meghatározzuk a leggyakoribbat és előfordulásainak számát. Ha több eredmény létezik, csak az elsőt kell megadnunk.

```
void leggyakoribbMagassag(int n, int a[], int &magassag, int &k){
    // a leggyakoribb magasság meghatározása (k-szor fordul elő)
    int elofoordulas[maxDim2]; // a maximális magasság (30000)
    for (int i = 1; i < maxDim2; i++)
        // gyakoriságtömb kezdőértékei
        elofoordulas[i] = 0;
    for (int i = 1; i <= n; i++) // gyakoriságtömb aktualizálása
        elofoordulas[a[i]]++;
    int max = 1;                // a maximális gyakoriság
    for (int i = 1; i < maxDim2; i++) // a 0 magasságot kihagyjuk
        if (elofoordulas[i] > elofoordulas[max])
            max = i;
    magassag = max;
    k = elofoordulas[max];     // "k" a maximális gyakoriság előfordulásának száma
}
```

g. Hívó programegység (main függvény/főprogram)(1 pont)

```
int main(){
    int a[maxDim];           // a magasságok sorozata (maximális elemszám = 10000)
    int n,                   // a magasságok száma
    szigetekSzama,           // szigetek száma
}
```

```
eleje,                                // a legnagyobb sziget kezdetének sorszáma
vege,                                // a legnagyobb sziget végének sorszáma
magassag,                            // a leggyakoribb magasság
k;                                   // a leggyakoribb magasság előfordulásainak száma
bool hegy,                           // ha az értéke true: a leghosszabb sziget hegy típusú
kulonbozok; // ha az értéke true: a leghosszabb sziget magasságai különbözők
beOlvas(n, a);
repulesSzimulalasa(n, a, szigetekSzama, eleje, vege);
if (szigetekSzama != 0)                // ha van legalább egy sziget
    hegy = hegyTulajdonsag(eleje, vege, a);
kulonbozok = kulonbozok_e(n, a);
if (!kulonbozok)                      // a leggyakoribb magasság meghatározása
    leggyakoribbMagassag(n, a, magassag, k);
kiirEredmeny(szigetekSzama, eleje, vege, hegy, kulonbozok, magassag, k);
}
```

7. Kitűzött feladatok

7.1. Rácsteszték

7.1.1. Mit ír ki?

Legyen a következő program:

C	C++
<pre>#include <stdio.h> int P(int *x){ int s = 0; while (*x != 0){ s = s + (*x) % 2; (*x) = (*x) / 2; } return s; } int main(){ int n = 1234; printf("%d", P(&n) + P(&n) + 1); return 0; }</pre>	<pre>#include <iostream> using namespace std; int P(int & x){ int s = 0; while (x != 0){ s = s + x % 2; x = x / 2; } return s; } int main(){ int n = 1234; cout << P(n) + P(n) + 1; return 0; }</pre>
Pascal	
<pre>function P(var x: integer): integer; var s: integer; begin s := 0; while x <> 0 do begin s := s + x MOD 2; x := x DIV 2; end; P := s end; var n: integer; begin n := 1234; write(P(n) + P(n) + 1); end.</pre>	

Állapítsátok meg, mit ír ki a program a végrehajtás eredményeként.

E. 5

F. 6

G. 7

H. 10

7.1.2. Vajon mit csinál?

Legyen a $miEz(n, x, szám)$ algoritmus, ahol n ($2 \leq n \leq 10000$) természetes szám, x egy n elemű számjegysorozat, $szám$ természetes szám ($1 \leq szám < n$).

```

Algoritmus miEz(n, x, szám)
  Amíg szám ≠ 0 végezd el
    k ← 1
    Amíg k < n és x[k] ≥ x[k + 1] végezd el
      k ← k + 1
    vége(amíg)
    Minden i = k, n - 1 végezd el
      x[i] ← x[i + 1]
    vége(minden)
    n ← n - 1
    szám ← szám - 1
  vége(amíg)
Vége(algoritmus)

```

Állapítsátok meg a $\text{miEz}(n, x, \text{szám})$ algoritmus hatását.

- A. Az algoritmus kizár **szám** darab elemet az x sorozatból úgy, hogy ez csökkenően rendezetté válik.
- B. Az algoritmus kizár **szám** darab elemet az x sorozatból úgy, hogy ez növekvően rendezetté válik.
- C. Az algoritmus kizár **szám** darab elemet az x sorozatból úgy, hogy az x -ben maradt számjegyek eredeti sorrendben történő konkatenálása által azt a lehetséges legnagyobb számot kapjuk, amelyet $n - \text{szám}$ számjegy alkot.
- D. Az algoritmus kizár **szám** darab elemet az x sorozatból úgy, hogy az x -ben maradt számjegyek eredeti sorrendben történő konkatenálása által azt a lehetséges legkisebb számot kapjuk, amelyet $n - \text{szám}$ számjegy alkot.

7.1.3. Különleges számok sorozatának generálása

Legyen a következő, természetes számokat tároló sorozat: 1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211, Állapítsátok meg a sorozat 10. elemét.

- A. 141122221
- B. 13211311123113112211
- C. 411211131221
- D. 1114122112132113212221

7.1.4. Adattípus

A következő intervallumok közül melyikhez tartozhat egy x biten ábrázolt egész adattípussal rendelkező érték (x – szigorúan pozitív természetes szám)?

- A. $[0, 2^x]$
- B. $[0, 2^{x-1} - 1]$
- C. $[-2^{x-1}, 2^{x-1} - 1]$
- D. $[-2^x, 2^x - 1]$

7.1.5. A legkisebb n -nél nagyobb szám

Legyen egy pontosan kilenc különböző, nem nulla számjegyű n természetes szám és a következő két algoritmus:

- $\text{felcserél}(a, b)$ – felcseréli az a és b természetes számokat;
- $\text{épít}(n, x, \text{érték})$ – az *érték* természetes szám után ragasztja az n elemű x számjegysorozat elemeit (például, ha $n = 4$, $x = (2, 6, 0, 4)$ és *érték* = 71, az $\text{épít}(n, x, \text{érték})$ végrehajtása után az *érték* = 712604).

Döntsék el, hogy az alábbi algoritmusok közül melyik határozza meg azt a legkisebb m értéket, amely nagyobb, mint n , és pontosan *azokból a számjegyekből áll, mint n* ? Ha nincs ilyen érték, az eredmény -1 lesz.

A.	<p>Algoritmus meghatároz(n)</p> <pre> szám ← 1 v[szám] ← n MOD 10 n ← n DIV 10 kész ← hamis Amíg nem kész és n > 0 végezd el szám ← szám + 1 v[szám] ← n MOD 10 n ← n DIV 10 Ha v[szám] < v[szám - 1] akkor kész ← igaz vége(ha) vége(amíg) Ha n = 0 akkor térítsd -1 vége(ha) felcserél(v[szám], v[szám - 1]) n ← n * 10 + v[szám] épít(szám - 1, v, n) térítsd n Vége(algoritmus) </pre>
B.	<p>Algoritmus meghatároz(n)</p> <pre> szám ← 0 Amíg n > 0 végezd el szám ← szám + 1 v[szám] ← n MOD 10 n ← n DIV 10 vége(amíg) kész ← hamis i ← szám Amíg nem kész és i > 0 végezd el Ha v[i] < v[i - 1] akkor kész ← igaz vége(ha) i ← i - 1 vége(amíg) felcserél(v[i], v[i - 1]) n ← n * 10 + v[szám] épít(szám, v, n) térítsd n Vége(algoritmus) </pre>

C.	<p>Algoritmus meghatároz(n)</p> <pre> szám ← 0 Amíg n DIV 10 ≠ 0 és n MOD 10 < n DIV 10 MOD 10 végezd el szám ← szám + 1 v[szám] ← n MOD 10 n ← n DIV 10 vége(amíg) Ha n < 10 akkor térítsd -1 különben szám ← szám + 1 v[szám] ← n MOD 10 n ← n DIV 10 c ← n MOD 10 n ← n DIV 10 felcserél(v[szám], c) n ← n * 10 + c épít(szám, v, n) térítsd n vége(ha) Vége(algoritmus) </pre>
D.	<p>Algoritmus meghatároz(n)</p> <pre> szám ← 0; másolat ← n Amíg n > 0 végezd el szám ← szám + szám v[szám] ← n MOD 10 n ← n DIV 10 vége(amíg) kész ← hamis i ← szám Amíg nem kész és i > 1 végezd el Ha v[i] > v[i - 1] akkor kész ← igaz különben i ← i - 1 vége(ha) vége(amíg) Ha i = 0 akkor térítsd -1 vége(ha) felcserél(v[i - 2], v[i - 1]) p ← 1 Minden k = 1, i - 1 végezd el p ← p * 10 vége(minden) n ← másolat / p épít(i - 1, v, n) térítsd n Vége(algoritmus) </pre>

7.1.6. Sorozatok

Legyen minden $h \in \{1, 2, 3\}$ hosszú sorozat, amelyeknek elemei az $\{a, b, c, d, e\}$ halmazhoz tartoznak. Hány olyan sorozat található ezek között, amelyeknek elemei növekvően rendezettek és páratlan darab magánhangzót tárolnak?

A. 14

B. 7

C. 81

D. 78

7.1.7. Pozitív számok

Legyen a pozitivSzamok(m , a , n , b) alprogram:

C/C++
<pre>void pozitivSzamok(int m, int a[], int &n, int b[]){ n = 0; for (int i = 1; i <= n; i++){ if (a[i] > 0){ n = n + 1; b[n] = a[i]; } } }</pre>
Pascal
<pre>type sor = array[1..100] of integer; procedure pozitivSzamok(m: integer; a: sor; var n: integer; var b: sor) begin n := 0; for i := 1 to n do if a[i] > 0 then begin n := n + 1; b[n] := a[i]; end; end; end;</pre>

Mi lesz a hatása a pozitivSzamok(m , a , n , b) alprogram hívásának, ha $m = 4$ és $a = (-1, 2, -3, 4)$?

- A. $n = 3$ és $b = (2, 4)$
- B. $n = 0$ és $b = (2, 4)$
- C. $n = 0$ és $b = ()$
- D. Függ m értékétől

7.1.8. Hatvány

A következő rekurzív algoritmus minimális számú szorzással számítja ki az x^n hatványértékét (x – valós szám, $0 < x \leq 10$, n – természetes szám, $1 \leq n \leq 40$).

Állapítsátok meg, hány szorzást végez el az algoritmus, ha x és n alább megadott értékeire hívjuk meg.

<pre>Algoritmus f(x, n) Ha n = 0 akkor térítsd 1 különben Ha n páratlan szám akkor térítsd f(x * x, n DIV 2) * x különben térítsd f(x * x, n DIV 2) vége(ha) vége(ha) Vége(algoritmus)</pre>
--

- A. Ha $x = 1.5$ és $n = 10$,
akkor a szorzások száma = 6
- B. Ha $x = 2.55$ és $n = 1$,
akkor a szorzások száma = 1
- C. Ha $x = 3.14$ és $n = 32$,
akkor a szorzások száma = 7
- D. Ha $x = 0.5$ és $n = 1$,
akkor a szorzások száma = 2

7.1.9. A legnagyobbérték

Adott a számol(n) algoritmus, ahol n egy 4-jegyű természetes szám.

```

Algoritmus számol( $n$ )
   $m \leftarrow n$ 
   $s \leftarrow -1$ 
  Amíg  $n > 0$  végezd el
    Ha  $n \bmod 10 > s$  akkor
       $s \leftarrow n \bmod 10$ 
    különben
       $s \leftarrow (s + m - n) \bmod 123$ 
    vége(ha)
     $n \leftarrow n \text{ DIV } 10$ 
  vége(amíg)
  térítsd  $s$ 
Vége(algoritmus)

```

Állapítsátok meg n -nek azt a legnagyobb értékét, amelyre az algoritmus 101-et térít vissza.

- A. 9016
- B. 9941
- C. 9613
- D. 9981

7.1.10. A bemeneti paraméter értékei

Legyen a következő algoritmus:

```

Algoritmus paraméter( $a$ ):
  Ha  $a < 50$  akkor
    Ha  $a \bmod 3 = 0$  akkor
      térítsd paraméter( $2 * a - 3$ )
    különben
      térítsd paraméter( $2 * a - 1$ )
    vége(ha)
  különben
    térítsd  $a$ 
  vége(ha)
Vége(algoritmus)

```

Az a bemeneti paraméter mely értékeire térít a fenti algoritmus 61-et?

- A. 16
- B. 61
- C. 4
- D. 31

7.1.11. Igaz vagy hamis?

Tekintsük a következő alprogramot:

```

Algoritmus  $f(a)$ :
  Ha  $a \neq 0$  akkor
    térítsd  $a + f(a - 1)$ 
  különben
    térítsd 0
  vége(ha)
Vége(algoritmus)

```

Válasszátok ki a hamis állításokat:

- A. az $f(a)$ egy rekurzívan definiált algoritmus
- B. ha a negatív szám, az algoritmus 0-t térít vissza
- C. az algoritmus az $a * (a+1) / 4$ értéket számítja ki
- D. az algoritmus az a -nál kisebb vagy egyenlő természetes számok összegét számolja ki
- E. ha az algoritmust $a = -5$ -re hívjuk meg, az algoritmus hibaüzenettel áll le

7.1.12. Számolás

Adott a $\text{számol}(n, x, a, b)$ algoritmus. Bemeneti paraméterek az n természetes szám ($1 \leq n \leq 1\,000$) és az n elemű x sorozat, amely különböző, $1\,000$ -nél kisebb természetes számokat tárol. Kimeneti paraméterek az a és b természetes szám ($0 \leq a < 1\,000, 0 \leq b < 1\,000, a > b$).

```

Algoritmus számol( $n, x, a, b$ )
  határ  $\leftarrow 999$ ; lépés  $\leftarrow 100$ 
   $a \leftarrow 0$ ;  $b \leftarrow 0$ ;  $k \leftarrow 0$ 
  Minden  $i = 1$ , határ - lépés végezd el
    előfordul[ $i$ ]  $\leftarrow$  hamis
  vége(minden)
  Minden  $i = 1, n$  végezd el
    Ha  $x[i] \geq$  lépés és  $x[i] \leq$  határ akkor
      előfordul[ $x[i] -$  lépés]  $\leftarrow$  igaz
       $k \leftarrow k + 1$ 
    vége(ha)
  vége(minden)
  Ha  $k <$  határ - lépés akkor
     $i \leftarrow$  határ - lépés
    Amíg előfordul[ $i$ ] és  $i \geq 0$  végezd el
       $i \leftarrow i - 1$ 
    vége(amíg)
    Ha  $i \geq 0$  akkor
       $a \leftarrow i +$  lépés
       $i \leftarrow i - 1$ 
      Amíg előfordul[ $i$ ] és  $i \geq 0$  végezd el
         $i \leftarrow i - 1$ 
      vége(amíg)
       $b \leftarrow i +$  lépés
    vége(ha)
  vége(ha)
Vége(algoritmus)

```

Állapítsátok meg, melyek igazak az alábbi állítások közül.

- A. Ha $n = 7$ és $x = (12, 345, 123, 67, 989, 6, 997)$, akkor $a = 999$ és $b = 998$.
- B. Ha $n = 5$ és $x = (996, 998, 995, 997, 999)$, akkor $a = 999$ és $b = 998$.
- C. Az algoritmus a -ban és b -ben azt a két legnagyobb háromjegyű számot határozza meg, amelyek nem találhatók meg az x sorozatban. Ha nem lehet két ilyen számot meghatározni, $a = b = 0$.
- D. Az algoritmus a -ban és b -ben az x sorozat azon két legnagyobb háromjegyű számát határozza meg, amelyek kisebbek, mint 1000 . Ha nem lehet két ilyen számot meghatározni, $a = b = 0$.

7.1.13. Logikai kifejezés értéke

Legyen x egy egész típusú szám, amelynek az értéke az a legkisebb 0 -tól különböző természetes szám, amely 36 -nak többszöröse és osztható minden 10 -nél kisebb prímszámmal.

Állapítsátok meg, hogy mely kijelentések igazak az alábbiak közül:

- A. $(x < 1000)$ és $((x * x * x) \bmod 1000 = 0)$
- B. $(x \bmod 100 = 0)$ vagy $(x \text{ DIV } 100 = 0)$
- C. $(x > 1000)$ és $(x \bmod 7 = 0)$
- D. $((x * x) \text{ DIV } 16) \bmod 2 = 1$
- E. $((x * x) \text{ DIV } 16) \bmod 2 = 0$

7.1.14. Hívások száma

Legyen az $f(a, b)$ algoritmus:

```

Algoritmus f(a, b):
  Ha a > 1 akkor
    térítsd b * f(a - 1, b)
  különben
    térítsd b * f(a + 1, b)
  vége(ha)
Vége(algoritmus)

```

Hányszor hívja meg önmagát az $f(a, b)$ algoritmus a mel-
lélt programrészletben található hívás következtében?

```

a ← 4
b ← 3
c ← f(a, b)

```

- A. 4-szer
- B. 3-szor
- C. végtelenszer
- D. egyszer sem

7.1.15. Módosítás

Legyen a következő algoritmus:

```

1: Algoritmus keresés(x, n, érték):
2:   Ha n = 1 akkor
3:     térítsd x[1] = érték
4:   különben
5:     térítsd keresés(x, n - 1, érték)
6:   vége(ha)
7:   Vége(algoritmus)

```

Az algoritmusnak el kellene döntenie, hogy *érték* megtalálható vagy sem az n (0-nál szigorúan nagyobb természetes szám) elemű x sorozat elemei között, de hibásan működik. Mely módosítást kellene elvégezni ahhoz, hogy helyessé váljon?

- A. Az 5. sort módosítjuk: **térítsd** $((x[n] = \text{érték})$ és **keresés**($x - 1, n, \text{érték}$))
- B. Az 5. sort módosítjuk: **térítsd** $((x[n] = \text{érték})$ vagy **keresés**($x, n - 1, \text{érték}$))
- C. Az 5. sort módosítjuk: **Ha** $(x[n] = \text{érték})$ **akkor** **térítsd** true
különben **térítsd** **keresés**($x, n - 1, \text{érték}$)
- D. nem kell módosítani egyetlen utasítást sem

7.1.16. Kiegészítés

Egy számokkal bűvészkedő varázsló szeretné az x ($1 \leq x \leq 1\,000\,000$) természetes számot szétválasztani a **bal** és **jobb** számra úgy, hogy a **bal** és **jobb** számjegyeinek konkatenálásával kapja meg az x számot, miközben a **bal** és a **jobb** szorzata maximális. Például, ha $x = 1092$, a varázslat eredményeként x szétválik **bal** = 10-re és **jobb** = 92-re. A varázsló felhasználja a következő algoritmusokat:

- **hatvány(alap, kitevő)** – meghatározza az **alap**^{**kitevő**} értékét (**alap** felemelve a **kitevő** hatványra), ahol **alap** és **kitevő** természetes számok ($1 \leq \text{alap} \leq 20$, $1 \leq \text{kitevő} \leq 20$);
- **szjSzám(szám)** – meghatározza a **szám** természetes szám számjegyeinek darabszámát ($1 \leq \text{szám} \leq 1\,000\,000$);
- **szorzat(bal, jobb)**.

```

Algoritmus szorzat(bal, jobb)
  Ha bal > 0 akkor
    aktJobb ← ...
    aktBal ← bal DIV 10
    Ha bal * jobb < aktBal * aktJobb akkor
      térítsd szorzat(aktBal, aktJobb)
    különben
      térítsd bal * jobb
  vége(ha)
különben
  térítsd bal * jobb
vége(ha)
Vége(algoritmus)

```

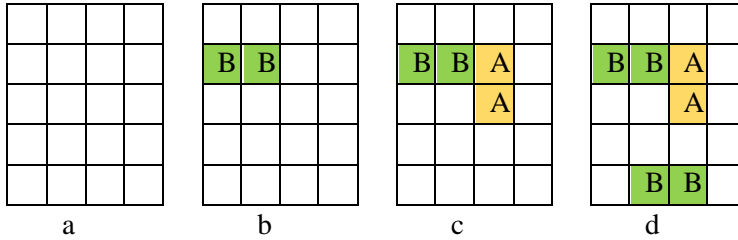
Állapítsátok meg, mivel helyettesíthető a „...” ahhoz, hogy az alábbi utasítások végrehajtásának eredménye 920 és 73 575 legyen.

Ki: szorzat(1092, 0)
Ki: szorzat(75981, 0)

- A. $(\text{bal} \bmod 10) * \text{hatvány}(10, \text{szjSzám}(\text{jobb})) + \text{jobb}$
 B. $(\text{bal} \bmod 10) * \text{hatvány}(10, \text{jobb}) + \text{jobb}$
 C. $(\text{bal} \bmod 10) * \text{hatvány}(10, \text{szjSzám}(\text{jobb}))$
 D. $(\text{bal} \bmod 10) * \text{szjSzám}(\text{jobb})$

7.1.17. Dominó

Legyen egy kétdimenziós tábla, amely föl van osztva $n \times m$ cellára (n – a sorok száma, m – az oszlopok száma, n, m – természetes számok, $2 \leq n \leq 100$, $2 \leq m \leq 100$). Két játékos, A és B, felváltva lépéseket hajtanak végre: minden lépésnél a soron levő játékos megjelöl két szomszédos cellát (megfelelnek egy dominónak), amelyek még nincsenek megjelölve. A két cella vízszintesen, vagy függőlegesen szomszédos. Az a játékos, aki nincs hova lépjen, veszít. A B játékos lép először.



Példa: a) eredeti állapot ($n = 5$ és $m = 4$), b) az első lépés utáni helyzet (lépett B), c) a második lépés után (lépett A), d) a harmadik lépés után (lépett B)

Határozzátok meg, milyen feltételeknek kell teljesülniük ahhoz, hogy A-nak biztos nyerési stratégiája legyen, (vagyis nyerni fog, függetlenül B lépéseitől) és legtöbb hány lépést fog végrehajtani A ahhoz, hogy nyerjen.

- A.** feltétel: az n páratlan és az m páros szám;
az A játékos maximális lépésszáma egyenlő $n \times (m \text{ DIV } 2)$ -vel.
- B.** feltétel: az összes cella darabszáma páros szám;
az A játékos maximális lépésszáma egyenlő a táblán levő cellák számának egynegyedével.
- C.** feltétel: n páros szám és m páratlan szám;
az A játékos maximális lépésszáma egyenlő a táblán levő cellák számának felével.
- D.** feltétel: az összes cella darabszáma 4-nek többszöröse;
az A játékos maximális lépésszáma egyenlő a táblán levő cellák számának egynegyedével.

7.1.18. Misszionáriusok és kannibálok

Egy folyó bal partján m misszionárius és k kannibál található (k, m – természetes számok, $k < m$, $1 \leq k < 10$, $1 \leq m \leq 10$). Mindannyian át szeretnének kelni a jobb partra egy kétszemélyes csónakkal. Tudjuk, hogy ha egyik parton több kannibál marad, mint misszionárius, a kannibálok megeszik azokat a misszionáriusokat, akik azon a parton vannak.

Melyik lesz sikeres a következő költöztetési stratégiák közül, vagyis, az összes misszionárius és az összes kannibál átjut a jobb partra úgy, hogy a kannibálok egyetlen misszionáriust se egyenek meg?

- A.**
- ha a bal parton csak misszionáriusok vannak, akkor a jobb partra átkel két misszionárius, különben egy misszionárius és egy kannibál;
 - ha a jobb parton van legalább egy misszionárius és a bal parton a misszionáriusok száma egyenlő a kannibálok számával, akkor a bal partra átkel egy misszionárius, különben egy kannibál.

- B.**
- ha a bal parton misszionáriusok is és kannibálok is vannak, akkor a jobb partra átkel egy misszionárius és egy kannibál, különben két misszionárius;
 - ha a jobb parton van legalább egy kannibál és a bal parton a misszionáriusok száma szigorúan nagyobb, mint a kannibáloké, akkor a bal partra átkel egy kannibál, különben egy misszionárius.
- C.**
- ha a bal parton a misszionáriusok száma egyenlő a kannibálok számával, akkor a jobb partra átkel egy kannibál, különben egy misszionárius;
 - ha a jobb parton van legalább egy kannibál és a bal parton a misszionáriusok száma szigorúan nagyobb, mint a kannibáloké, akkor a bal partra átkel egy misszionárius, különben egy kannibál.
- D.**
- ha bal parton van egy kannibál és több misszionárius, akkor a jobb partra átkel egy misszionárius és egy kannibál, különben két misszionárius;
 - a jobb parton a kannibálok száma egyenlő a misszionáriusok számával, akkor a bal partra átkel egy kannibál, különben egy misszionárius.

7.2. Alprogramok

7.2.1. Legnagyobb palindrom

Egy természetes szám *palindrom*, ha egyenlő azzal a számmal, amelyet úgy kapunk, hogy számjegyeit fordított sorrendben írjuk fel.

Írjatok alprogramot, amely adott n ($0 < n < 2^{31}$) természetes szám esetében, meghatározza azt a **maxPal** számot, amely a legnagyobb *palindrom*, amelyet az n szám minden számjegyének átrendezése által kaphatunk. Ha nem lehet kialakítani a palindromot, amelyben az n szám minden számjegye szerepeljen, akkor **maxPal** értéke -1 lesz. Az alprogram bemeneti paramétere az n szám, kimeneti paramétere pedig a **maxPal**.

1. Példa: ha $n = 21523531$, akkor **maxPal** = 53211235.

2. Példa: ha $n = 12272351$, akkor **maxPal** = -1.

7.2.2. Alszámok

Egy pontosan kétjegyű x természetes számot az y természetes szám „alszám”-ának nevezzük, ha x számjegyei megjelennek, eredeti sorrendjükben, egymás után az y számban. Például a 41 alszáma 14121-nek, 413-nak és 41-nek, de nem alszáma 143-nak és 431-nek, 77 alszáma 77757-nek, ahol kétszer fordul elő.

Legyen az n elemű a , természetes számokat tároló sorozat ($0 \leq n \leq 10^6$). Az a sorozat elemei nagyobbak, mint 0 és kisebbek, mint 10^9 .

Írjatok algoritmust, amely meghatározza azt a k elemű b sorozatot, amely az a sorozat elemeinek azokat az alszámaikat tartalmazza, amelyek a *legtöbbször* fordulnak elő. A b sorozat elemeinek sorrendje tetszőleges. Az algoritmus bemeneti paramétere n és a , kimeneti paramétere k és b .

1. Példa: ha $n = 5$ és $a = (733325, 698787, 127898, 613373, 212612)$, akkor $k = 2$ és $b = (12, 33)$. Megjegyzés: a 12 és 33 alszámok, mindkettő háromszor fordul elő az a sorozatban.

2. Példa: ha $n = 2$ és $a = (5907, 23)$, akkor $k = 3$ és $b = (59, 90, 23)$.

7.2.3. Összeg

Legyen az n ($0 < n < 10\,000$) természetes szám és a következő összeg:
$$S = 1^1 + 2^2 + 3^3 + \dots + n^n.$$

Írjatok alprogramot, amely meghatározza az S összeg *utolsó számjegyét* (*utolsóSzj*). Az n értékét bemeneti paraméterként adjuk meg az alprogramnak, *utolsóSzj* kimeneti paraméter lesz.

Példa: ha $n = 3$, *utolsóSzj* = 4; ha $n = 9\,989$, akkor *utolsóSzj* = 3.

7.2.4. Tükrök

Adott az x és az y sorozat, melyeknek elemei 0 és 1. Mindkét sorozat (egyenként) n ($3 \leq n \leq 10\,000$) elemet tartalmaz, de *nem azonosak*. Definiáljuk a „tükrözés” műveletet, amelynek végrehajtása következtében az x sorozat azonosá válhat az y sorozattal: elhelyezünk egy „tükröt” az x sorozatba az a -dik elem baloldalára és egy másik „tükröt” a b -dik elem jobboldalára. A tükrözés eredményeképpen az x sorozatnak azok az elemei, amelyek a és b között találhatók, beleértve a -t és b -t is, *tükröződnek* (vagyis az x sorozat a -dik eleme felcserélődik az x sorozat b -dik elemével, az $(a + 1)$. elem felcserélődik a $(b - 1)$ -kel stb.).

Írjatok alprogramot, amely meghatározza az a és b pozíciókat, ahova a tükröket el kellene helyezni az x sorozatba, és eldönti, hogy egyetlen „tükrözés” művelet végrehajtása következtében az x sorozat átalakul, vagy sem úgy, hogy azonosá váljon az y sorozattal. Az alprogram bemeneti paraméterei n , x és y , kimeneti paraméterek *válasz*, a és b . A *válasz* paraméter értéke *true*, ha az x sorozat, a tükrözés hatására átalakul az y sorozattá, különben *false*. Ha az x sorozat nem alakítható át a tükrözéssel az y sorozattá, a és b értéke -1 lesz.

1. Példa: ha $n = 8$, $x = (0, 0, 1, 1, 0, 1, 1, 0)$ és $y = (0, 1, 0, 1, 1, 0, 1, 0)$, a tükröket az $a = 2$ és $b = 6$ pozíciókra kellene helyezni. Így az x sorozat átalakul úgy, hogy azonos lesz az y sorozattal, tehát *válasz* = *true*.

2. Példa: ha $n = 5$, $x = (1, 0, 1, 1, 1)$ és $y = (1, 1, 1, 1, 1)$, $a = -1$ és $b = -1$, mivel az x sorozat nem alakul át az y sorozattá, tehát *válasz* = *false*.

Megjegyzés: A példákban a sorozatokat 1-től kezdődően indexeltük.

7.2.5. Trükkös sorozat

Egy n elemű, természetes számokat tartalmazó a sorozatot *trükkösnek* nevezünk, ha néhány balra végzett körkörös permutációval előállítható belőle az $a_1 + 1, a_2, a_3, \dots, a_{n-2}, a_{n-1}, a_n - 1$ sorozat. Az elemek körkörös permutációja alatt a sorozat elemeinek egy pozícióval balra tolását értjük (kivételt képez az első elem, amely a sorozat utolsó pozíciójára kerül).

Írjatok alprogramot, amely eldönti, hogy az n elemű ($3 \leq n \leq 1\,000$), természetes számokat tartalmazó a sorozat *trükkös-e* vagy sem. Az alprogram bemeneti paraméterei n és a . Kimeneti paramétere a *trükkös* nevű változó lesz, amelynek értéke *true*, ha az a sorozat *trükkös*, különben *trükkös* értéke *false*.

Példák: az $a = (2, 2, 2, 3)$ sorozat trükkös; az $(1, 2, 1, 2)$ nem trükkös.

7.2.6. Maximális összeg

Legyen n és m két természetes szám ($0 < n \leq 1\,000$ és $0 < m \leq 1\,000$, $n \geq m$) és egy n elemű, egész számokat tároló sorozat a sorozat, amelyeknek az értékei nagyobbak, mint -30 000 és kisebbek, mint 30 000.

Írjatok alprogramot, amely meghatározza az m elemű b részsorozatot, amely az a sorozat azon elemeit tartalmazza, amelyeknek az *összege maximális*. Az alprogram bemeneti paraméterei n, m és a , kimeneti paraméter a b lesz.

Példa: ha $n = 6$, $a = (5, -3, 12, 8, 2, 2)$ és $m = 3$, a kért sorozat $b = (5, 12, 8)$.

7.2.7. Minimális különbség

Legyen két *rendezett sorozat*: az n elemű a és az m elemű b sorozat. A sorozatok elemei -30 000-nél nagyobbak és 30 000-nél kisebbek ($1 \leq n \leq 100$ és $1 \leq m \leq 100$).

Írjatok alprogramot, amely meghatározza a két sorozat *bármely két eleme különbségének abszolút-értékei közül a legkisebbet*, ahol az első szám az a sorozat eleme, a második pedig a b sorozathoz tartozik.

Példa: ha $n = 6$ és $a = (2, 4, 5, 10, 21, 29)$, $m = 4$ és $b = (13, 14, 15, 15)$, a *minKülönbség* = 3. Ez az a sorozathoz tartozó 10 és a b sorozathoz tartozó 13 különbségének abszolút-értéke (modulusa): $3 = |10 - 13|$.

7.2.8. Átrendezés

Adott az n természetes szám ($1 \leq n \leq 100$) és az n elemű x sorozat, amelynek elemei nagyobbak, mint 1 és kisebbek, mint 500. Írjatok alprogramot, amely átrendezi az x sorozatot a következőképpen: az elemek számjegyeinek összege alapján csökkenő sorrendbe rendezi a páros szám értékű elemeket (ha két páros szám számjegyeinek összege azonos, megőrzi az eredeti sorrendet). A páratlan számok az eredeti helyükön maradnak. Az alprogram bemeneti paramétere n , bemeneti és kimeneti paramétere x .

Példa: ha $n = 5$ és $x = (123, 2244, 5282, 4679, 548)$, akkor a megváltoztatott sorrendű $x = (123, 5282, 548, 4679, 2244)$.

7.2.9. Különleges szám

Egy n természetes számot *különlegesnek* nevezünk, ha létezik olyan m természetes szám, amelyre $n = m + S(m)$, ahol $S(m)$ az m szám számjegyeinek összege. Például, $n=15$ különleges szám, mivel létezik $m=12$ és $15 = 12 + 3$.

Írjatok algoritmust, amely eldönti a bemeneti paraméterként megadott n természetes számról, hogy különleges-e.

7.3. Rekurzió

7.3.1. Számolás – karakterekkel

Legyen a számolásKarakterekkel(s , n , i , szám) algoritmus, ahol s egy n karakterből álló sorozat (n természetes szám, $1 \leq n \leq 200$), i és **szám** természetes számok ($1 \leq i \leq n$).

```

Algoritmus számolásKarakterekkel( $s$ ,  $n$ ,  $i$ , szám):
    eredmény  $\leftarrow 0$ 
    Amíg  $i \leq n$  végezd el
        Amíg  $i \leq n$  és  $s[i] \geq '0'$  és  $s[i] \leq '9'$  végezd el
            szám  $\leftarrow$  szám * 10 +  $s[i] - '0'$ 
             $i \leftarrow i + 1$ 
        vége(amíg)
        eredmény  $\leftarrow$  eredmény + szám
        szám  $\leftarrow 0$ 
         $i \leftarrow i + 1$ 
    vége(amíg)
    térítsd eredmény
Vége(algoritmus)

```

Írjátok le a számolásKarakterekkel(s , n , i , szám) algoritmus *rekurzív* változatát úgy, hogy a fejléce és a hatása legyen azonos a fenti algoritmuséval. Az alábbi programrészletből hívjuk meg:

```

Beolvas:  $n$ ,  $s$ 
Kiír: számolásKarakterekkel( $s$ ,  $n$ , 1, 0)

```

7.3.2. Iteratívól rekurzív (1)

Legyen a következő alprogram, ahol az a és b bemeneti paraméterek természetes számok ($0 < a \leq 1\,000$ és $0 < b \leq 1\,000$):

```

Algoritmus  $f(a, b)$ :
    eredmény  $\leftarrow 0$ 
    Amíg  $a > 0$  végezd el:
        Ha  $a \bmod 2 = 1$  akkor
            eredmény  $\leftarrow$  eredmény +  $b$ 
        vége(ha)
         $a \leftarrow a \text{ DIV } 2$ 
         $b \leftarrow b + b$ 
    vége(amíg)
    térít eredmény
Vége(algoritmus)

```

- Adjátok meg annak a feladatnak a szövegét, amelyet ez az algoritmus old meg.
- Mit térít az $f(5, 15)$ hívás?
- Írjátok le az adott algoritmus *rekurzív* változatát, amelynek fejléce azonos az iteratív (nem rekurzív) algoritmus fejlécével.

7.3.3. Iteratívból rekurzív (2)

Adott a következő alprogram, ahol az sz tetszőleges természetes szám bemeneti paraméter ($1 < sz \leq 10\,000$):

```

Algoritmus F( $sz$ ):
   $a \leftarrow 1$ 
   $b \leftarrow 0$ 
  Amíg  $sz > a + b$  végezd el:
     $a \leftarrow a + b$ 
     $b \leftarrow a - b$ 
  vége(amíg)
  Ha  $sz = a + b$  akkor
    térítsd true
  különben
    térítsd false
  vége(ha)
Vége(algoritmus)

```

- Adjátok meg annak a feladatnak a szövegét, amelyet ez az algoritmus old meg.
- Mit térít az $F(17)$ hívás?
- Írjátok le az adott algoritmus *rekurzív* változatát, amelynek fejléce azonos az iteratív (nem rekurzív) algoritmus fejlécével.

7.3.4. Iteratívból rekurzív (3)

Adott a következő alprogram, ahol az n és k bemeneti paraméterek ($1 < n \leq 30\,000$ és $1 < k \leq 30\,000$) és természetes számok:

```

Algoritmus F( $n, k$ ):
   $\text{érték} \leftarrow 0$ 
  Amíg  $n \geq k$  végezd el:
     $n \leftarrow n \text{ DIV } k$ 
     $\text{érték} \leftarrow \text{érték} + 1$ 
  vége(amíg)
  térít érték
Vége(algoritmus)

```

- Adjátok meg annak a feladatnak a szövegét, amelyet ez az algoritmus old meg.
- Mit térít az $F(98, 2)$ hívás?
- Írjátok le az adott algoritmus *rekurzív* változatát, amelynek fejléce azonos az iteratív (nem rekurzív) algoritmus fejlécével.

7.3.5. Mi a hatása?

Adott a következő algoritmus, amelynek két természetes szám paramétere van, n és m ($m \leq n$). Az algoritmus egy természetes szám értékét téríti vissza.

```

Algoritmus F( $n, m$ ):
  Ha  $m = 0$  vagy  $m = n$  akkor
    térítsd 1
  különben térítsd  $F(n - 1, m - 1) + F(n - 1, m)$ 
  vége(ha)
Vége(algoritmus)

```

- Mit térít az $F(15, 13)$ hívás? Indokoljátok meg a választ!
- Adjátok meg egy-egy értéket n és m számára úgy, hogy az $F(n, m)$ algoritmus által visszatérített érték legyen 243. Indokoljátok meg a választ!
- Adjátok meg az algoritmus által megoldott feladat szövegét.

7.3.6. Különböző elemek

Írjatok *rekurzív* algoritmust, amely az adott n természetes szám és az n elemű x sorozat esetében eldönti, hogy a sorozat elemei különbözők-e.

7.3.7. Mi a hatása?

Adott a következő algoritmus:

```

Algoritmus F(n):
  x ← 1
  y ← n DIV 2
  Amíg x ≠ 0 és y > 0 végezd el:
    z ← n
    Amíg z ≥ y végezd el:
      z ← z - y
    vége(amíg)
    x ← z
    x ← 1
    y ← y - 1
  vége(amíg)
  y ← y + 1
  térítsd y
Vége(algoritmus)

```

- Mit térít az $F(91)$ hívás? Indokoljátok meg a választ!
- Adjátok meg n értéket úgy, hogy az $F(n)$ algoritmus által visszatérített érték legyen 11. Indokoljátok meg a választ!
- Adjátok meg az algoritmus által megoldott feladat szövegét (n 0-tól különböző természetes szám).

7.3.8. Mi a hatása?

Adott a következő algoritmus:

```

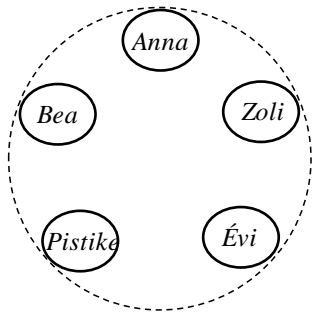
Algoritmus F(a):
  s ← 0
  Minden i = 1, 4 végezd el:
    Be b
    x ← a; y ← b; z ← 0
    Amíg x ≠ 0 végezd el:
      Ha x MOD 2 = 1 akkor
        z ← x + y
      vége(ha)
      x ← x DIV 2; y ← y * 2
    vége(amíg)
    s ← s + z; a ← b
  vége(minden)
  térítsd s
Vége(algoritmus)

```

- Mit térít az $F(4)$ hívás, ha a b értékei rendre: 16, 40, 15, 8? Indokoljátok meg a választ!
- Adjátok adatokat a b változónak úgy, hogy ha $a = 4$, az $F(a)$ algoritmus által visszatérített érték legyen 63. Indokoljátok meg a választ!
- Adjátok meg az algoritmus által megoldott feladat szövegét. Tudjuk, hogy az adatok 0-tól különböző természetes számok.

7.4. Modellezési feladatok

7.4.1. Játék



Az óvoda udvarán n gyerek körben áll, és kiszámoló játékot játszanak. A kiszámolást egy bizonyos gyerektől kezdik (*start*), számolnak m -ig – az óramutató járásának megfelelő irányban – amikor az m -dik gyerek kilép a körből. Aki utolsónak marad a körben, az a nyertes. Pistike sír, mert eddig nem nyert egyszer sem. Az óvónéni megsajnálja, és szeretné tudni, mely sorszámú gyerektől kellene kezdeni a kiszámolást ahhoz, hogy Pistike nyerjen.

Követelmények:

1. Adjátok meg a körből kilépő gyerekek sorszámait, ha $n = 7$, $m = 4$ és *start* = 3? (A gyerekeket a sorszámaik azonosítják.)
2. Összesen hányszor érinti a számolás Pistikét, ha $n = 8$, $m = 5$ és *start* = 3?
3. Mi annak szükséges és elégséges feltétele, hogy Pistike nyerjen? Indokoljátok meg a választ.
4. Írjatok programot, amely meghatározza azt a távolságot, amelynek lennie kell Pistike és a között a gyerek között – az óramutató járásával megegyező irányban – akitől a számolást el kell kezdeni ahhoz, hogy Pistike nyerjen. Tudjuk, hogy $3 \leq n \leq 10\,000$, $3 \leq m \leq 10\,000$.

1. Példa: ha $n = 5$ és $m = 3$, *táv* = 2. *Magyarázat:* mivel Pistike a negyedik, a kiszámolást a hatodik gyerektől kezdjük, vagyis az 1-es sorszámú gyerektől, mivel körben állnak. Tehát Annával kezdünk, így rendre kilép a körből Évi, Anna, Bea és Zoli. Nyer Pistike.

2. Példa: ha $n = 5$ és $m = 4$, akkor *táv* = 0. *Magyarázat:* a számolást Pistikével kezdjük, így rendre kilép Zoli, Anna, Évi és Bea. Pistike nyer.

7.4.2. Forgolódás

Testnevelés órán a gyermekek a tanárunkkal szemben állnak egy sorban, egymás mellett, amikor a tanár azt kéri, hogy mindenki forduljon jobbra. A gyermekek közül egyesek balra fordulnak meg, mások jobbra. Ha egy gyermek, szemtől szemben találja magát egy másikkal, mindketten sarkon fordulnak (vagyis megfordulnak 180° -kal). Egy időegység alatt több gyermek is végezhet ilyen mozgulattal, de egy gyermek csak egyszer fordul meg az adott időegységen belül.

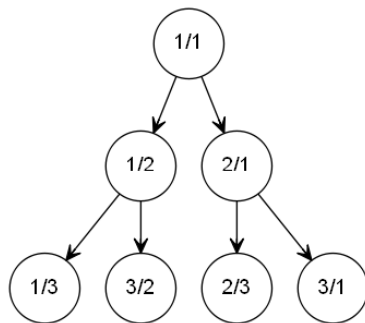
Írjatok alprogramot, amely meghatározza azoknak az időegységeknek a számát, amelyek eltelnek, amíg a sor megnyugszik. Az alprogram bemeneti paramétere a gyermekek n száma ($1 \leq n \leq 10\,000$) és a gyermekek sorozata (**gyermekek**), amely 'b' és 'j' karakterekből áll. A 'b' karakter a gyermek baloldalát, a 'j' a jobboldalát jelenti, annak függvényében, hogy milyen irányba fordult az illető gyermek, miután a tanár kiadta a parancsot. Kimeneti paraméter az **idő**, amely azoknak az időegységeknek a száma, amelyek eltelnek, amíg a sor megnyugszik (egy gyermek sem mozdul meg többet).

Példa: ha $n = 5$ és **gyermekek** = ('b', 'j', 'b', 'j', 'b'), **idő** = 2.

7.4.3. Törtek fája

Ismert, hogy bármely tört elhelyezhető egy fába a következő szabályok alapján:

- az első szinten az $\frac{1}{1}$ tört található;
- a második szinten, az első szinten levő $\frac{1}{1}$ törttől balra az $\frac{1}{2}$ tört található, tőle jobbra a $\frac{2}{1}$ tört;
- ...
- a k . szinten a $(k-1)$. szinten levő $\frac{i}{j}$ törttől balra az $\frac{i}{i+j}$ tört, tőle jobbra az $\frac{i+j}{j}$ tört található.



Követelmények:

1. Állapítsátok meg, mely szinten található az $\frac{1}{7}$ tört.
2. Soroljátok fel az 5. szinten található törteket.
3. Írjatok algoritmust, amely meghatározza az **szintSz** számot, amely annak a szintnek a sorszáma, amelyen az $\frac{a}{b}$ tört található, ahol a és b relatív prímek ($\text{lnko}(a, b) = 1$). Az algoritmus bemeneti paramétere a tört a számlálója és b nevezője, kimeneti paramétere az **szintSz** ($a, b, \text{szintSz}$ – természetes számok, $1 \leq a \leq 2 \cdot 10^9$, $1 \leq b \leq 2 \cdot 10^9$, $1 \leq \text{szintSz} \leq 2 \cdot 10^9$).

Példa: ha $a = 13$, $b = 8$, akkor **szintSz** = 6.

4. **Törtek felbontása.** Ismert, hogy két nem nulla természetes szám a és b esetében, amelyek relatív prímek ($\text{lnko}(a, b) = 1$) és $a < b$, az $\frac{a}{b}$ tört felírható k darab $\frac{1}{q_i}$ alakú tört összegeként, ahol $i = 1, 2, \dots, k$, és q_1, q_2, \dots, q_k természetes számok.

5. Írjatok algoritmust, amely adott $\frac{a}{b}$ tört esetében meghatároz egy ilyen felbontást. Az algoritmus bemeneti paraméterei a tört a számlálója és b nevezője, kimeneti paraméterek pedig: k, q_1, q_2, \dots, q_k ($a, b, k, q_1, q_2, \dots, q_k$ – természetes számok, ahol $1 \leq a \leq 2 \cdot 10^9, 1 \leq b \leq 2 \cdot 10^9, a < b, a$ és b relatív prímek, $2 \leq k$).

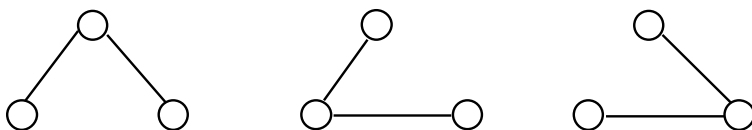
Példa: ha $a = 7$ és $b = 50$, akkor a következő felbontások helyesek:

- a $\frac{7}{50} = \frac{1}{8} + \frac{1}{67} + \frac{1}{13400}$ felbontásban három tag szerepel, így $k = 3, q = (8, 67, 13400)$, míg
- a $\frac{7}{50} = \frac{1}{10} + \frac{1}{25}$ felbontásban két tag szerepel, így $k = 2, q = (10, 25)$.

7.4.4. Eszkimók

A távoli északon az eszkimók jégsátrakban laknak. Télen naponta ki kell ásnuk a hóból az ösvényeket, hogy iglutól igluig haladva eljuthassanak az ismerőseikhez. A Nagy Eszkimó egy szép nap kitalálta, hogy olyan stratégiát szeretne, aminek eredményeképpen minimális számú ösvényt kell kiszabadítani a hóból. Tehát, a cél az, hogy bárki bárkit meg tudjon látogatni, két iglu között legfeljebb egy ösvény legyen és az ösvények ne metsszék egymást. Szeretné tudni, hogy hányféleképpen lehet ezt a stratégiát megvalósítani. A településen n iglu egy egyenes vonal mentén helyezkedik el a Nagy Eszkimó sátrával szemben.

Példa: ha $n = 1$, van egy iglu + a Nagy Eszkimó sátra. Nyilvánvaló, hogy egyetlen ösvény lesz. Ha $n = 2$ a következő három lehetőség közül lehet válogatni:



Ha $n = 3$, összesen 8 lehetőség van.

Írjatok alprogramot, amely adott n esetében meghatározza a k változóban, hogy hányféleképpen lehetséges kialakítani az ösvények karbantartásának módját. Bemeneti paraméter az n (természetes szám $1 \leq n \leq 35$), kimeneti paraméter k , az ösvények a feladatban leírt jelentéssel.

7.5. Összetett feladatok

7.5.1. „Majdnem prímszámok”

Egy természetes számot *majdnem prím*nek nevezünk, ha egyenlő két különböző prímszám szorzatával. Például, a 15 *majdnem prím*, mivel egyenlő a 3 és 5 prímszámok szorzatával.

Legyen egy n természetes szám ($1 \leq n \leq 1\,000$) és egy n elemű x sorozat, amelyben az elemek 1-nél szigorúan nagyobb és 30 000-nél kisebb természetes számok.

Írjatok programot, amely meghatározza az adott sorozat leghosszabb tömbszakaszát, amely csak *majdnem prím*eket tartalmaz, és kiírja az illető tömbszakasz kezdőindexét (*balMax*) és végsőindexét (*jobbMax*). Ha több ilyen tömbszakasz létezik, a *legelső* leghosszabb tömbszakasz kezdőindexét és végsőindexét kell kiírnotok. Ha a sorozatban nem létezik egyetlen *majdnem prím* sem, *balMax* és *jobbMax* értéke egyenlő lesz -1-gyel. Egy tömbszakasz egy adott sorozat egy vagy több elemét tartalmazza, amelyek az eredeti sorozatban egymás utáni pozíciókon találhatók.

1. Példa: ha $n = 8$ és $x = (24, 34, 35, 11, 8, 77, 35, 26)$, akkor *balMax* = 6 és *jobbMax* = 8. ($x_6 = 77 = 7 * 11$, $x_7 = 35 = 5 * 7$, $x_8 = 26 = 2 * 13$). A példában a sorozatot 1-től kezdődően indexeltük.

2. Példa: ha $n = 3$ és $x = (24, 11, 8)$, akkor *balMax* = -1 és *jobbMax* = -1.

A megoldásban fölhasználjátok a következő alprogramokat:

- bemeneti adatok beolvasása billentyűzetről;
- annak eldöntése, hogy egy szám *majdnem prím*-e vagy sem;
- a kért tulajdonsággal rendelkező leghosszabb tömbszakasz kezdő- és végső-indexének meghatározása;
- a *balMax* és a *jobbMax* értékek kiírása.

7.5.2. Stabilszélsőértékek

Egy természetes számot *stabil*nak nevezünk, ha egyetlenegy számjegye van vagy, ha bármely két egymás után elhelyezkedő számjegyének összege nagyobb vagy egyenlő 10-zel. Például, 291 *stabil*, mivel $2 + 9 \geq 10$ és $9 + 1 \geq 10$, ugyanakkor 9278 nem *stabil*, mivel $2 + 7 < 10$.

Bármely nem *stabil* szám feldolgozható a következőképpen: bármely két egymás után elhelyezkedő számjegyet, amelyeknek összege szigorúan kisebb, mint 10, behelyettesíthetjük azzal a számjeggyel, amely ennek a kettőnek az összege. A behelyettesítések tovább alkalmazhatók, ugyanezen feltételek mellett, a behelyettesítések után kapott számra, akárhányszor, amíg egy *stabil* számot nem kapunk. A 2453-ból, egyetlen behelyettesítéssel a 653, a 293 vagy a 248 számokat kapjuk. Ezek közül csak a 293 *stabil* szám. A 653-ból, mivel nem *stabil*, egy újabb behelyettesítés révén megkapjuk a 68 *stabil* számot. Hasonlóan, a 248-ból kinyerjük a 68 *stabil* számot.

Írjatok programot, amely az adott, n elemű ($1 \leq n \leq 10\,000$), 30 000-nél kisebb természetes számokat tároló a sorozat alapján felépíti az alább leírt módon, és kiírja az m elemű b sorozatot, amely *stabil* számokat tartalmaz:

- ha az a sorozat aktuális eleme *stabil*, elhelyezzük a b sorozatba;
- ha az a sorozat aktuális eleme *nem stabil*, meghatározzuk és betesszük a b sorozatba azt a legnagyobb *stabil* számot (**max**) és azt a legkisebb *stabil* számot (**min**), amelyeket előállíthatunk az a sorozat eleméből egy, esetleg több, fent leírt behelyettesítéssel.

Példa: ha $n = 3$ és $a = (2367, 12, 5689)$, akkor $m = 5$ és $b = (567, 297, 3, 3, 5689)$. *Magyarázat:* 2367 nem *stabil*, belőle megkaphatjuk a **max** = 567 és **min** = 297 *stabil* számokat, amelyeket elhelyezünk a b sorozatba; a 12 nem *stabil*; kinyerhetjük belőle a *stabil* 3-at (mivel ez az egyetlen *stabil* szám, amit generálhatunk, a 3 egyszerre minimum is és maximum is, és elhelyezzük a b sorozatba); 5689 *stabil* szám, betesszük a b sorozatba.

A megoldásban fölhasználjátok a következő alprogramokat:

- a bemeneti adatok beolvasása billentyűzetről;
- annak eldöntése, hogy egy szám *stabil*-e vagy sem;
- adott, nem *stabil* szám esetében a **min** és **max** *stabil* számok generálása;
- a b sorozat felépítése;
- az eredmények kiírása.

7.5.3. Mellékfolyók

Egy ország felszínén több folyó található. Egy folyónak lehet egy vagy több mellékfolyója, de előfordulhat, hogy nincs egy sem. Egy bizonyos folyó csak egyetlen másik folyó mellékfolyója lehet. Egy adott r folyó vízhozama a forrásánál az a vízmennyiség, amely egy időegység alatt folyik át rajta a forrásánál. Egy adott r folyó vízhozama a torkolatánál az a vízmennyiség, amely egy időegység

alatt folyik át rajta a torkolatánál. Ahhoz, hogy megkapjuk az r folyó vízhozamát a torkolatánál, összeadjuk az r folyó vízhozamát a forrásánál és az r folyó minden mellékfolyójának vízhozamát a torkolatuknál.

Írjatok programot, amely feldolgoz n folyót. Adottak a folyók mellékfolyói (**mellékFolyók**) és a folyók vízhozamai a forrásuknál (**vízForrás**). A program határozza meg és írjon ki minden folyót, amelyek nem mellékfolyók (**nemMellék**), minden folyót, amelyeknek nincs mellékfolyójuk (**nincsMellék**), valamint az egyes folyók vízhozamait a torkolatuknál (**vízTorkolat**). Tudjuk, hogy n természetes szám, $0 < n < 1000$; **mellékFolyók** egy n sort és n oszlopot tartalmazó kétdimenziós tömb, ahol:

$$\text{mellékFolyók}_{ij} = \begin{cases} 1, & \text{ha az } i \text{ folyó mellékfolyója a } j \text{ folyónak} \\ 0, & \text{különben} \end{cases}, \quad 1 \leq i \leq n, 1 \leq j \leq n$$

n és **vízForrás**, **vízTorkolat**, **nemMellék** valamint **nincsMellék** n elemű, 1000-nél kisebb természetes pozitív számokat tároló sorozatok.

1. Példa: ha $n = 3$, **mellékFolyók** = $\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ és **vízForrás** = (5, 3, 6), akkor **nemMellék** = (2), **nincsMellék** = (1, 3), **vízTorkolat** = (5, 14, 6).

2. Példa: ha $n = 4$, **mellékFolyók** = $\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$ és **vízForrás** = (5, 3, 6, 1), akkor **nemMellék** = (1, 4), **nincsMellék** = (1, 2), **vízTorkolat** = (5, 3, 9, 13).

A megoldásban írjatok egy-egy alprogramot, amely:

- beolvassa a bemeneti adatokat a billentyűzetről (a bemeneti adatok garantáltan megfelelnek a követelményeknek);
- eldönti egy folyóról, hogy nem mellékfolyója egyetlen más folyónak sem;
- eldönti egy folyóról, hogy nincs egyetlen mellékfolyója sem;
- meghatározza egy folyó vízhozamát a torkolatánál;
- kiír a képernyőre egy természetes számokat tároló egydimenziós tömböt.

7.5.4. Törzstényezők

Adott az n elemű, természetes számokat tartalmazó x sorozat ($1 \leq n \leq 500$, $1 \leq X_i \leq 1000$). Írjatok programot, amely:

- Törzstényezőkre bontja a sorozat elemeit, és meghatározza azoknak a különböző törzstényezőeknek a k elemű y sorozatát, amelyek az első hatványon fordulnak elő. Ha az x sorozat egyetlen elemének sincs olyan törzstényezője, amely az első hatványon fordulna elő, a program írjon ki megfelelő üzenetet.

2. Meghatározza azt a h elemű p sorozatot, amely azokat a prímszámokat tartalmazza, amelyek nagyobbak, mint az y sorozatban található legkisebb érték, kisebbek, mint az y sorozatban található legnagyobb érték, és hiányoznak az y sorozatból. Ha az y sorozat üres, a p sorozat is üres lesz, és a program kiír egy megfelelő üzenetet. Ha az y sorozat egyetlen elemet tartalmaz, a p sorozat szintén üres lesz.

1. *Példa:* ha $n = 4$ és $x = (77, 58, 77, 31)$, az y sorozat $(2, 7, 11, 29, 31)$ és a p sorozat $(3, 5, 13, 17, 19, 23)$.

2. *Példa:* ha $n = 4$ és $x = (64, 36, 100, 125)$, az y és p sorozatok üresek.

3. *Példa:* ha $n = 4$ és $x = (5, 25, 125, 625)$, az y sorozat (5) és a p sorozat üres.

7.5.5. Előszélet (2)

Sors-számjegynek hívjuk azt a természetes számot, amelyet adott természetes számra a következőképpen számítunk ki: összeadjuk a szám számjegyeit, majd a kapott összeg számjegyeit, és így tovább, amíg a kapott összeg nem válik egy-számjegyű számmá. Például, a 182 *sors-számjegye* 2 ($1 + 8 + 2 = 11$, $1 + 1 = 2$).

Egy pontosan k számjegyű p számot egy legkevesebb k számjegyű q szám *előszéletének* nevezünk, ha a q szám első k számjegyéből alkotott szám (balról jobbra tekintve) egyenlő p -vel. Például, 17 előszete 174-nek, és 1713 előszete 1 713 242-nek.

Legyen az sz természetes szám ($0 < sz \leq 30\,000$) és egy m soros és n oszlopos ($0 < m \leq 100$, $0 < n \leq 100$) A mátrix (kétdimenziós tömb), amelynek elemei 30 000-nél kisebb természetes számok. Adva van a *sorsSzámjegy(x)* algoritmus, amely meghatározza az x számhoz rendelt sors-számjegyet:

Algoritmus sorsSzámjegy(x):

$s \leftarrow 0$

Amíg $x > 0$ **végezd el**

$s \leftarrow s + x \text{ MOD } 10$

$x \leftarrow x \text{ DIV } 10$

Ha $x = 0$ **akkor**

Ha $s < 10$ **akkor**

térítsd s

különben

$x \leftarrow s$

$s \leftarrow 0$

vége(ha)

vége(ha)

vége(amíg)

térítsd s

Vége(algoritmus)

Követelmények:

- Írjátok le egy *rekurzív* változatát (ismétlő struktúrák nélkül) a *sorsSzámjegy(x)* algoritmusnak. A fejléce és a hatása legyen azonos a fenti algoritmus fejlécével és hatásával.
- Írjátok le a *sorsSzámjegy(x)* algoritmus rekurzív változatának (amelyet kidolgoztatok az **a.** pontnál) a matematikai modelljét (vagyis, írjátok le a rekurzív függvényt matematikai képlet formájában).
- Írjátok alprogramot, amely – felhasználva a *sorsSzámjegy(x)* alprogramot – meghatározza az *sz* szám leghosszabb előszeletét (**prefix**), amelyet az adott tömb elemeinek megfelelő *sors-számjegyeiből* fel lehet építeni. Egy ilyen sors-számjegyet akárhányszor fel lehet használni. Ha nem építhető fel előszelet, **prefix** = -1. Az alprogram bemeneti paraméterei: *sz*, **m**, **n** és az **A** mátrix, kimeneti paramétere: **prefix**.

Példa: ha *sz* = 12319, **m** = 3, **n** = 4 és a mátrix: $A = \begin{pmatrix} 182 & 12 & 274 & 22 \\ 22 & 1 & 98 & 56 \\ 5 & 301 & 51 & 94 \end{pmatrix}$, akkor a leghosszabb előszelet **prefix** = 1231, a megfelelő sors-számjegyek pedig:

Mátrixelem értéke	182	12	274	22	1	98	56	5	301	51	94
Sors-számjegy	2	3	4	4	1	8	2	5	4	6	4

7.5.6. Bűvös számok (2)

Legyen két természetes szám **p** és **q** ($2 \leq p \leq 10$, $2 \leq q \leq 10$). Egy természetes számot *bűvösnek* nevezünk, ha a **p** számrendszerben felírt alakjában szereplő számjegyek halmaza azonos a **q** számrendszerben felírt alakjában szereplő számjegyek halmazával. Például, ha **p** = 9 és **q** = 7, $(31)_{10}$ *bűvös szám*, mivel $(34)_9 = (43)_7$; ha **p** = 3 és **q** = 9, $(9)_{10}$ *bűvös szám*, mivel $(100)_3 = (10)_9$. Adott még a *számjegyek(x, b, c)* alprogram, amely meghatározza az **x** szám számjegyeit a **b** számrendszerben (a **c** sorozatban):

```

Algoritmus számjegyek(x, b, c):
  Amíg x > 0 végezd el
    c[x MOD b] ← 1
    x ← x DIV b
  vége(amíg)
Vége(algoritmus)

```

Követelmények:

- Írjátok le egy *rekurzív* változatát (ismétlő struktúrák nélkül) a *számjegyek(x, b, c)* algoritmusnak. A fejléce és a hatása legyen azonos a fenti algoritmus fejlécével és hatásával.

- b. Írjátok le a számjegyek(x , b , c) algoritmus rekurzív változatának (amelyet kidolgoztatok az **a.** pontnál) matematikai modelljét, (vagyis, írjátok le a rekurzív függvényt matematikai képlet formájában).
- c. Írjatok alprogramot, amely – felhasználva a számjegyek(x , b , c) alprogramot – adott p és q számrendszerek ismeretében, meghatározza azt a *bűvös számokból álló a sorozatot*, amely minden 0-nál szigorúan nagyobb és adott n ($1 < n \leq 10\,000$) természetes számnál szigorúan kisebb számot tárol. Az alprogram bemeneti paraméterei p és q (a két alap) és az n szám. Kimeneti paraméter az a sorozat és ennek k hossza.
- Példa:** ha $p = 9$, $q = 7$ és $n = 500$, az a sorozatnak $k = 11$ eleme lesz: (1, 2, 3, 4, 5, 6, 31, 99, 198, 248, 297).

7.5.7. Ritka mátrix

Az $A(n, m)$, egész számokat tároló kétdimenziós tömböt *ritka mátrix*nak nevezzük, ha az elemeinek többsége 0 értékű. Egy k darab nem nulla elemmel rendelkező $A(n, m)$ ritka mátrixot egy olyan k elemű egydimenziós x tömb formájában ábrázolunk, amelyben az elemek leírják a 0-tól különböző értékeket a (*sor, oszlop, érték*) jellemzők alapján. Így nincs szükség a kétdimenziós tömböt tárolni a memóriában. Az x tömb elemei a *sor*, majd, ezen belül *oszlop* szerint lexikográfikus sorrendben találhatók.

Példa: ha $n = 3$ és $m = 3$, az $A(3, 3) = \begin{pmatrix} 0 & 5 & 2 \\ 0 & 2 & 0 \\ 2 & 0 & 3 \end{pmatrix}$ mátrixot a $k = 5$ elemű x tömbben tároljuk: $x = ((1, 2, 5), (1, 3, 2), (2, 2, 2), (3, 1, 2), (3, 3, 3))$.

Írjatok programot, amely beolvassa a billentyűzetről az n , m számokat és az $A(n, m)$ valamint $B(n, m)$ ritka mátrixokat, majd kiszámítja $A(n, m)$ és $B(n, m)$ összegét a $C(n, m)$ ritka mátrixban és kiírja ezt kétdimenziós tömb alakban.

A ritka mátrixokat a megfelelő egydimenziós tömbök formájában olvassuk be, ameddig az egyes tömbök esetében a (-1, -1, -1) elem meg nem jelenik.

Az (i_1, j_1) értékpár lexikográfikusan kisebb az (i_2, j_2) értékpárnál, ha $i_1 < i_2$ vagy $i_1 = i_2$ és $j_1 < j_2$.

Írjatok alprogramokat a következő részfeladatok megoldásának érdekében:

- Annak ellenőrzése, hogy az (i_1, j_1) értékpár lexikográfikusan kisebb-e az (i_2, j_2) értékpárnál;
- A (*sor, oszlop, érték*) elem beszúrása az $A(n, m)$ ritka mátrixot ábrázoló x sorozatba;
- Adott *sor* és *oszlop* indexű elem beazonosítása az $A(n, m)$ tömbnek megfelelő x sorozatban;

- d. Az $A(n, m)$ tömb beolvasása az előbb leírt módon;
- e. A $C(n, m)$ ritka mátrix elemeinek kiszámítása, ahol $C(n, m)$ az $A(n, m)$ és $B(n, m)$ ritka mátrixok összege;
- f. Az $A(n, m)$ ritka mátrix kiírása kétdimenziós tömb formájában.

Példa: ha $n = 3$, $m = 3$, és az A ritka mátrix elemei az x sorozatban ábrázolva:

$((2, 2, 2), (3, 3, 3), (1, 2, 5), (3, 1, 2), (1, 3, 5), (-1, -1, -1))$.

A B mátrix elemei: $((3, 2, 4), (1, 2, -5), (2, 2, 1), (-1, -1, -1))$.

A program kiírja a $\begin{pmatrix} 0 & 0 & 5 \\ 0 & 3 & 0 \\ 2 & 4 & 3 \end{pmatrix}$ ritka mátrixot.

7.5.8. Tökéletes számok

Adott az n elemű x sorozat, amely természetes számokat tárol ($1 \leq n \leq 200$, $1 \leq x_i \leq 30000$). Az x_i ($1 \leq i \leq n - 1$) elemet *tökéletesnek* nevezzük, ha van legalább egy olyan számjegye, amely megtalálható az összes, a sorozatban utána következő elem értékében.

Írjatok programot, amely beszűrja az x sorozatba, minden tökéletes elem után az illető elem összes valódi osztóját, értékeik szerint csökkenő sorrendben. Az x sorozat módosítása után, hozzátok létre azt az $o = (o_1, o_2, \dots, o_n)$ sorozatot, amely tárolja azt a sorrendet, amely szerint az x sorozatot kiírhatjuk csökkenő sorrendben. Végül a program kiírja az x sorozatot abban a sorrendben, amelyet az o sorozat ad meg.

1. Példa: ha $n = 4$ és $x = (24, 5, 8, 218)$, a módosított $x = (24, 5, 8, 4, 2, 218)$. Mivel $o = (5, 4, 2, 3, 1, 6)$, kiírjuk a $(2, 4, 5, 8, 24, 218)$ sorozatot.

2. Példa: ha $n = 5$ és $x = (24, 5, 4, 42, 6)$, a módosított $x = (24, 5, 4, 42, 6)$. Mivel $o = (3, 2, 5, 1, 4)$, kiírjuk a $(4, 5, 6, 24, 42)$ sorozatot.

Írjatok alprogramokat a következő részfeladatok megoldása érdekében:

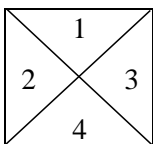
- a. egy sorozat beolvasása;
- b. annak ellenőrzése, hogy két számnak van legalább egy közös számjegye;
- c. a *tökéletes* szám tulajdonság ellenőrzése;
- d. egy új elem beszűrése az x sorozat adott pozíciójára;
- e. az osztók beszűrése az x sorozatba minden tökéletes szám után;
- f. az x sorozatnak megfelelő o sorozat létrehozása;
- g. egy x sorozat kiírása az o sorozat által megadott sorrendben.

7.5.9. Szuperprímek

Egy számot *szuperprím*nek nevezünk, ha minden előszelete prímszám (például, 239 *szuperprím*, mivel 2, 23 és 239 prímszámok, miközben 241 nem *szuperprím*, mivel 24 nem prím).

Adott egy n sorral és n oszloppal rendelkező négyzetes A tömb, amelynek elemei természetes számok ($3 \leq n \leq 50$, $1 \leq a_{ij} \leq 20000$).

Írjatok programot, amely meghatározza és kiírja azt az x sorozatot, amely növekvő sorrendben tartalmazza azokat a különböző értékű szuperprímeket, amelyek az A tömb bal, illetve jobb háromszögében találhatók. Ha az x sorozatnak nincs egyetlen eleme sem, a program írjon ki egy megfelelő üzenetet.



A mellékelt ábrán a mátrix bal háromszögét 2-sel, a jobb háromszögét 3-sal jelöltük meg.

Írjatok alprogramokat a következő részfeladatok megoldásának érdekében:

- egy négyzetes mátrix beolvasása;
- egy sorozat kiírása;
- a prímszám tulajdonság ellenőrzése;
- a *szuperprím*-szám tulajdonság ellenőrzése;
- új elem beszúrása egy rendezett sorozatba;
- az x sorozat felépítése.

Példa: ha $n = 4$ és $A = \begin{pmatrix} 16 & 241 & 15 & 8 \\ 239 & 3 & 2 & 79 \\ 241 & 100 & 5 & 239 \\ 12 & 92 & 241 & 79 \end{pmatrix}$, $x = (239, 79)$. A tömb bal

háromszögéhez az a_{21} és a_{31} elemek, míg a jobb háromszögéhez az a_{24} és a_{34} elemek tartoznak. Az átlókhöz tartozó elemeket nem vesszük figyelembe.