I. Rácstesztek

Mintatételek a 2021-es BBTE Matek-Infó versenyre | Matematika és Informatika Kar (ubbcluj.ro)

A.1. Vajon mit csinál? Mintatételben a 3.

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

```
Algoritmus kifejezés(n):
    Ha n > 0 akkor
        Ha n MOD 2 = 0 akkor
            térítsd -n*(n+1)+kifejezés(n-1)
        különben
            térítsd n*(n+1)+kifejezés(n-1)
        vége(ha)
    különben
        térítsd 0
    vége(ha)
Vége(algoritmus)
```

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

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

Elemzés

1. Az algoritmus nem hívja meg önmagát, ha ***n*** értéke 0-ra csökkent ⇒
   ⇒ ekkor kap az összeg (amit kiszámít) 0 kezdőértéket.
2. Az ***n*** aktuális értékének párosságától függően a 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.

3. 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. 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 nem helyes.
   - Az **E.** kifejezésben túl sok tag előjele „–”, tehát nem az **E.** kifejezést értékeli az algoritmus.

A.2. Számolás Mintatételben a 17.

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

```
Algoritmus számol(n):
    x ← 0
    z ← 1
    Amíg z ≤ n végezd el
        x ← x + 1
        z ← z + 2 * x
        z ← 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*** ≥ 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. 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 ≥ 85 és ***n*** < 100	
x	***z***
…	…
5	35
	36
6	48
	49
7	63
	64
8	80
	81
9	99

Észrevételek:

1. Az ***x*** értéke egyesével nő 1-től
2. Amikor az `Amíg` struktúra végrehajtása véget ér, az algoritmus az ***x*** aktuális értékét téríti.
3. Egy-egy iteráció végén ***z*** aktuális értéke egyenlő az ***x*** aktuális értékének a négyzetével.
4. 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. Ha ***n*** = 25, ***z*** 36-ig nő és ***x*** = 5. Ha ***n*** = 35, a ***z*** értéke szintén 36-ig nő és ***x*** értéke most is 5. (helyes, tehát nem jelöljük meg)

E. Rájöttünk hogy helyes, tehát nem jelöljük meg.

B. 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.

C. Helyes (látjuk a táblázatban), nem jelöljük meg.

D. Eldöntöttük, hogy az algoritmus az adott szám négyzetgyökének egész részét határozza meg, vajon ez 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 Mintatételben a 6.

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

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

Megoldás

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

A. (**not** *hamis* **or** *hamis*) **or** (*hamis* **and** *hamis*) = = (*igaz* **or** *hamis*) **or** *hamis* = *igaz* **or** *hamis* = *igaz*

B. (**not** *hamis* **or** *igaz*) **or** (*hamis* **and** *hamis*) = (*igaz* **or** *igaz*) **or** *hamis* = = *igaz* **or** *hamis* = *igaz*

C. (**not** *igaz* **or** *hamis*) **or** (*hamis* **and** *igaz*) = (*hamis* **or** *hamis*) **or** *hamis* = = *hamis* **or** *hamis* = *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?

Legyen a következő program:

C

```c
#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;
}
```

C++

```cpp
#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;
}
```

Pascal	

`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.`
	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: A sorozatnak csak három eleme van, 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 Mintatételben a 25.

Egy nullától különböző ***x*** természetes számot *szerencsés*nek 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*** = -1.

A.	`Algoritmus szerencsésSzám(x, start, szerencsés):`
	`  xNégyzet ← x * x`
	`  szerencsés ← hamis`
	`  start ← -1, k ← 1, s ← 0`
	`  Amíg k ≤ xNégyzet - x és nem szerencsés végezd el`
	`  Minden i = k, k + x - 1 végezd el`
	`  s ← s + i`
	`  vége(minden)`
	`  Ha s = xNégyzet akkor`
	`  szerencsés ← igaz, start ← k`
	`  vége(ha)`
	`  vége(amíg)`
	`Vége(algoritmus)`
		Megoldás
		Nem „futtatjuk” papíron az algoritmusokat. Elemezzük és összehasonlítjuk őket. Például, az **A.** és **B.** algoritmusok közötti különbségek:
		1. az **A.** algoritmusban az ***s*** változó a külső `Amíg` előtt kap kezdőértéket, és ezáltal túl sok számot fog összeadni.

B.

```
Algoritmus szerencsésSzám(x, start, szerencsés):
    xNégyzet ← x * x
    szerencsés ← hamis
    start ← -1, k ← 1
    Amíg k ≤ xNégyzet - x és nem szerencsés végezd el
        s ← 0
        Minden i = k, k + x - 1 végezd el
            s ← s + i
        vége(minden)
        Ha s = xNégyzet akkor
            szerencsés ← igaz, start ← k
        vége(ha)
        k ← k + 1
    vége(amíg)
Vége(algoritmus)
```

2. A **B.** algoritmusban a kezdőértékadás helyes, így a négyzetek vizsgálatakor az összeg kiszámolása 0 kezdőértéktől indul.

3. Az **A.** algoritmusban a ***k*** változó értéke nem változik, így futtatáskor végtelen ciklust eredményez.

⇒ az **A.** algoritmus hibás.

⇒ a **B.** algoritmus helyes.

C.

```
Algoritmus szerencsésSzám(x, start, szerencsés):
    Ha x MOD 2 = 0 akkor
        szerencsés ← hamis, start ← -1
    különben
        szerencsés ← igaz
        start ← (x + 1) DIV 2
    vége(ha)
Vége(algoritmus)
```

D.

```
Algoritmus szerencsésSzám(x, start, szerencsés):
    Ha x MOD 2 = 0 akkor
        szerencsés ← hamis, start ← -1
    különben
        szerencsés ← igaz, start ← x DIV 2
    vége(ha)
Vége(algoritmus)
```

A **C.** és **D.** algoritmusok előbb kizárják a páros számokat. Vajon miért nem lehet *szerencsés* egy páros szám?

Felírjuk x^2-et összeg formájában:

$x^2 = (k + 1) + (k + 2) + ... + (k + x)$. (Az utolsó tag azért $k + x$, mert pontosan x tagból kellene előállítanunk az x^2-et.)

Alkalmazzuk az első n természetes szám összegének képletét (lásd a táblázat alatt)

E.

```
Algoritmus szerencsésSzám(x, start, szerencsés):
    szerencsés ← igaz
    start ← (x + 1) DIV 2
Vége(algoritmus)
```

$x^2 = \frac{(k+x)(k+x+1)}{2} - \frac{k(k+1)}{2}$. Következik, hogy $x^2 = \frac{2kx + x^2 + x}{2}$ vagyis $2x^2 = 2kx + x^2 + x$, amit, ha elosztunk x-szel: $2x = 2k + x + 1$, ahonnan $x = 2k + 1$, vagyis x-nek páratlannak kell lennie. Tehát, a páros számokat, valóban ki lehet zárni.

A **C.** és **D.** algoritmusok közötti különbség a ***start*** változó értékének kiszámításában fedezhető fel. A **C.** algoritmus a ***start*** változónak az `(x + 1) DIV 2` értéket adja, míg a **D.** algoritmus az `x 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. Tegyél 'b' betűket Mintatételben a 18.

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 ≤ n DIV 2 akkor
      Ha j ≤ n - i akkor
         mat[i][j] ← 'b'
         tegyélB(mat, n, i, j + 1)
      különben
         tegyélB(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 értékére hányszor hívja meg önmagát. Követjük a paraméterek értékeit a hívássorozaton belül.

`tegyélB(mat, 7, 2, 4)` ⇒ (1. hívás):
`tegyélB(mat, 7, 2, 5)` ⇒ (2. hívás):
`tegyélB(mat, 7, 2, 6)` ⇒ (3. hívás):
`tegyélB(mat, 7, 3, 4)` ⇒ (4. hívás):
`tegyélB(mat, 7, 3, 5)` ⇒ (5. hívás)
`tegyélB(mat, 7, 4, 5)`: vége.

Tehát, a rekurzív hívások száma 5 ⇒ az **A.** válasz jó és a **C.**, **D.** és **E.** válaszok nem jók. A **B.** esetben a hívások száma szintén 5, tehát a **B.** válasz is jó.

		Hívás sorszáma
5	***j***	5
4	***i***	
5	***j***	4
3	***i***	
4	***j***	3
3	***i***	
6	***j***	2
2	***i***	
5	***j***	1
2	***i***	
4	***j***	0 (első hívás)
2	***i***	

A.7. Vajon mit csinál? Mintatételben az 1.

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

```
Algoritmus generál(n):
   szám ← 0
   Minden i ← 1, 1801 végezd el
      használt[i] ← hamis
   vége(minden)
   Amíg nem használt[n] végezd el
      összeg ← 0
      használt[n] ← igaz
      Amíg n ≠ 0 végezd el
         számjegy ← n MOD 10
         összeg ← összeg+számjegy*számjegy*számjegy
         n ← n DIV 10
      vége(amíg)
      n ← összeg
      szám ← szám + 1
   vége(amíg)
   térítsd szám
Vége(algoritmus)
```

Állapítsátok meg, mi a hatása?

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 másodszorra 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

Észrevételek:

1. Az algoritmus a generált számokat az előfordulásokat tároló tömbben tartja nyilván.
2. Egy kissé furcsának tűnik a tömb mérete: valószínű, hogy a 100-nál kisebb számok esetében létezik legalább egy, ahol a számjegyek köbének összege pontosan 1801.
3. Az előfordulások tömbjében az ***n***-edik elem *igaz* értéket kap, ha a generálás során megjelenik az ***n*** szám.
4. 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.
5. 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 ← összeg` utasítást, tehát ***n*** értéke felülíródik, vagyis változik.
6. A **B.** és **C.** válaszok biztos nem jók: az algoritmus nem az összeget téríti.
7. A **D.** válasz „majdnem jó”, de nem tesz említést az adott számról, tehát hiányos, így nem tekinthető helyesnek.

A.8.Mely értékekre van szükség? Mintatételben a 10.

Adott a `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

Észrevételek:

1. 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.
2. Az algoritmus az aktuális elem értékét számjegyekre bontja, kiválasztja közülük a maximális értékűt és ezt a számjegyet beépíti az ***n*** számba balról jobbra haladva.

Kiválasztjuk a megfelelő bemeneti adatokat, amelyeknek feldolgozása 928-at eredményez:
A: a sorozat (194, 121, 782, 0) és a legnagyobb számjegyeket a sorozat elemeinek sorrendjében, balról jobbra haladva tartalmazó szám értéke pontosan 928.
C: a sorozat a 928 számjegyeit tartalmazza, tehát jó eredményt kapunk.
B: a sorozat egyetlen számot tartalmaz, ennek maximális számjegye 9, amelyből csak a 9-es számot lehet felépíteni, tehát nem jutunk el a 928-hoz.
D: az algoritmus a 829-es számot építi fel.
E: esetében a feldolgozás leáll az első szám után, mivel a második szám értéke 0.
Helyes válaszok: **A.** és **C.**

A.9.Logikai kifejezés kiértékelése Mintatételben a 2.

Adott a ***k*** elemű ***s*** sorozat, amelynek elemei boolean típusúak és a `kiértékelés(s, k, i)` algoritmus, ahol ***k*** és ***i*** ($0 \leq i \leq k \leq 100$) természetes számok. Határozzátok meg, hányszor hívja meg önmagát a `kiértékelés(s,k,i)` algoritmus a jobboldali programrészlet végrehajtása következtében:

```
Algoritmus kiértékelés(s, k, i)
    Ha i ≤ 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)
```

```
s ← (hamis, hamis, hamis, hamis, hamis,
hamis, igaz, hamis, hamis, hamis)
k ← 10
i ← 3
kiértékelés(s, k, i)
```

A. 3-szor
C. 6-szor
D. egyszer sem
E. végtelenszer

B. ugyanannyiszor, mint a következő programrészlet esetében

```
s ← (hamis, hamis, hamis, hamis, hamis, hamis, hamis, igaz)
k ← 8, i ← 4
kiértékelés(s, k, i)
```

Megoldás

Adott egy rekurzív alprogram, és meg kell állapítanunk, hogy a paraméterek adott értékére hányszor hívja meg önmagát. Követük a paraméterek értékeit a hívássorozaton belül, vagy lerajzoljuk a végrehajtási verem tartalmát és megszámoljuk a rekurzív hívásokat:

`kiértékelés(s, 10, 3)` ⇒ (1. hívás):
`kiértékelés(s, 10, 4)` ⇒ (2. hívás):
`kiértékelés(s, 10, 5)` ⇒ (3. hívás):
`kiértékelés(s, 10, 6)` ⇒ (4. hívás):
`kiértékelés(s, 10, 7)`: vége.

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

A rekurzív hívások száma 4, vagyis nem 3, tehát az **A.**, **C.**, **D.** és **E.** válaszok nem jók. A **B.** esetben a rekurzív hívások száma 4, vagyis ugyanannyi, mint az **A.** esetében, tehát a **B.** válasz jó.

A.10. Egyesítés Mintatételben a 16.

Adottnak tekintjük az `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 ($1 \le n \le 200$, $1 \le x \le 1000$).

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 ($1 \le n \le 200$, $1 \le m \le 200$, $1 \le p \le 400$). A bemeneti paraméterek ***a***, ***n***, ***b*** és ***m***, kimeneti paraméterek pedig ***c*** és ***p***. A következő állítások közül melyek bizonyulnak mindig igaznak?

```
Algoritmus egyesítés(a, n, b, m, c, p):
   Ha n = 0 akkor
      Minden i = 1, m végezd el
         p ← p + 1
         c[p] ← b[i]
      vége(minden)
   különben
      Ha nem eleme(a[n], b, m) akkor
         p ← p + 1
         c[p] ← a[n]
      vége(ha)
      egyesítés(a, n - 1, b, m, c, p)
   vége(ha)
Vége(algoritmus)
```

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

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

Megoldás

1. 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 kezdőértékadás (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.
2. A **B.** állítást egyszerű ellenőrizni, mivel a 12. sorban a rekurzív hívás található, amely pontosan annyiszor kerül végrehajtásra, ahány eleme van az ***a*** halmaznak.Tehát a **B.** válasz helyes.
3. 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).
4. A **D.** feltételezés sem lehet helyes. Ha például, az ***a*** halmaznak és a ***b***-nek is két eleme van: ***a*** = (1, 2) és a ***b*** = (3, 4), az egyesítés eredménye ***c*** = (1, 2, 3, 4), vagyis négy elem, míg az ***a*** halmaznak két eleme van.

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

A.11. Hatványra emelés

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 ← 1
   Amíg b > 0 végezd el
      Ha b MOD 2 = 1 akkor
         eredmény ← eredmény * a
      vége(ha)
      b ← b DIV 2
      a ← a * a
   vége(amíg)
   térítsd eredmény
Vége(algoritmus)
```

B.

```
Algoritmus hatvány(a, b):
   Ha b ≠ 0 akkor
      Ha b MOD 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 ← 1
   Amíg b > 0 végezd el
      eredmény ← eredmény * a
      b ← b - 1
   vége(amíg)
   térítsd eredmény
Vége(algoritmus)
```

D.

```
Algoritmus hatvány(a, b):
   Ha b = 0 akkor
      térítsd 1
   vége(ha)
   aux ← hatvány(a, b DIV 2)
   Ha b MOD 2 = 0 akkor
      térítsd aux * aux
   különben
      térítsd a * aux * aux
   vége(ha)
Vége(algoritmus)
```

E.

```
Algoritmus hatvány(a, b):
   Ha b = 0 akkor
      térítsd 1
   vége(ha)
   térítsd a * hatvány(a, b - 1)
Vége(algoritmus)
```

Megoldás

Előny, ha ismerjük *al-Kwarîzmi*, „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.12. Legnagyobb többszörös Mintatételben a 14.

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 ← b
   Amíg c MOD a = 0 végezd el
      c ← 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 DIV a) * a
Vége(algoritmus)
```

D.

```
Algoritmus f(a, b):
   Ha b MOD a = 0 akkor
      térítsd b
   vége(ha)
   térítsd f(a, b-1)
Vége(algoritmus)
```

E.

```
Algoritmus f(a, b):
   c ← a
   Amíg c < b végezd el
      c ← c + a
   vége(amíg)
   Ha c = b akkor
      térítsd c
   különben
      térítsd c – a
   vége(ha)
Vége(algoritmus)
```

Megoldás

C: Kiszámoljuk a térített értéket egy-két példára és azonnal belátjuk, hogy helyes.

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

A: Az 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 oszthatja maradék nélkül. Tehát nem helyes.

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

E. Az 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.

II. Rekurzió

1. Számolás - karakterekkel

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 ($1 \leq p \leq n$, $1 \leq q \leq n$, $p \leq q$). Írjátok le a `számolásKarakterekkel(s, n, p, q, szám)` algoritmus *rekurzív* változatát úgy, hogy a fejléce és a hatása legyen azonos a fenti algoritmuséval.

```
Algoritmus számolásKarakterekkel(s, n, p, q, szám):
   eredmény ← 0
   i ← p
   Amíg i ≤ q végezd el
      Amíg i ≤ q és s[i] ≥ '0' és s[i] ≤ '9' végezd el
         szám ← szám * 10 + s[i] - '0'
         i ← i + 1
      vége(amíg)
      eredmény ← eredmény + szám
      szám ← 0
      i ← i + 1
   vége(amíg)
   térítsd eredmény
Vége(algoritmus)
```

A rekurzív algoritmust az alábbi programrészletből hívjuk meg:

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

Megoldás

```
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)
```

2. Polinom értéke

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 $\boldsymbol{P(x) = egyh_1 * x^n + egyh_2 * x^{n-1} + ... + egyh_n * x + egyh_{n+1}}$ 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 ← 0.0
   Minden i ← 1, n + 1 végezd el
      érték ← érték * x + egyh[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.

```
Algoritmus kiértékelésRek(n, egyh, x):
   Ha n = 0 akkor
      térítsd egyh[1]
   különben
      térítsd egyh[n + 1] + x * kiértékelésRek(n - 1, egyh, x)
   vége(ha)
Vége(algoritmus)
```

3. Mi a hatása?

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)
```

a. Adjátok meg a feladat szövegét, amelyet ez az algoritmus old meg, ha ***a*** = 1 és ***b*** = 0 értékekre hívjuk meg.

b. Mit térít az `f(1,0,10)` hívás?

c. Írjátok le az algoritmusnak egy rekurzív változatát, megőrizve az iteratív (nem rekurzív) változat fejlécét.

Megoldás

Ha találkoztunk már ezzel az algoritmussal, azonnal tudni fogjuk, hogy az adott ***sz*** számnál kisebb *Fibonacci* számokat generálja, megszámolja ezeket ***k***-ban és téríti ezt a számot. Ha nem ismerjük fel az algoritmus funkcióját, ellenőrző táblázatot készítünk. 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:

a. Az algoritmus az ***sz***-nél szigorúan kisebb *Fibonacci* számok darabszámát határozza meg.

b. Ha az algoritmust az `f(1,0,10)` utasítással hívjuk meg, a térített érték 7.

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

4. Mi a hatása?

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

```
Algoritmus f(n, p, i):
   Ha n ≤ 9 akkor
      Ha n MOD 2 = 0 akkor
         p ← n
         i ← 0
      különben
         p ← 0
         i ← n
      vége(ha)
   különben
      f(n DIV 10, p, i)
      Ha n MOD 2 = 0 akkor
         p ← p * 10 + n MOD 10
      különben
         i ← i * 10 + n MOD 10
      vége(ha)
   vége(ha)
Vége(algoritmus)
```

a. Adjátok meg annak a feladatnak a szövegét, amelyet ez az algoritmus old meg.

b. Mi lesz ***p*** és ***i*** értéke az `f(205609, p, i)` hívás után?

c. Írjátok le egy iteratív változatát az adott algoritmusnak, megőrizve a rekurzív változat fejlécét.

Megoldás

Előbb meg kell értenünk az adott algoritmus funkcióját. Számjegyekre bontja az adott ***n*** számot, és rekurzív hívások leállnak, amikor az ***n*** szám egyszámjegyű lett. Ekkor kap kezdőértéket ***p*** és ***i***. 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, ***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.

A válaszok:

a. Az algoritmus meghatározza az ***n*** szám páros, illetve páratlan számjegyeit – az eredeti sorrendben – tartalmazó számokat.

b. Az `f(205609, p, i)` hívás eredményeként ***p*** = 2060, ***i*** = 59 lesz.

```
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;
      }
   }
}
```

5. Vírusok

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

a. ha egy bizonyos óra kezdetekor a vírusok létszáma *páros* szám, akkor az óra végére a létszám 50%-kal csökken;

b. ha egy bizonyos óra kezdetekor a vírusok létszáma *páratlan* szám, akkor az óra végére a létszám 1-gyel növekszik;

c. 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 egyébre nincs szükség, mint az algoritmus utasításaival szimulálni a vírusok sorsát, a leírásnak megfelelően.

A rekurzív algoritmus tömörebb, és nincs szükségünk a ***megsemmisült*** változóra. Az eredmény (amit az iteratív algoritmus 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 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;
}
```

```
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;
}
```

6. Mi a hatása?

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

```
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)
```

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

Megoldás

Látható, hogy az algoritmus 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.

Válaszok:

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`

```
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. Modellezési feladatok

1. Tornyok

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

1. a legmagasabb torony magassága ***n*** ($0 < \boldsymbol{n} \leq 13$), 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.

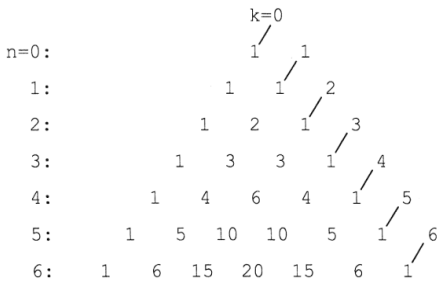
Tehát, ahhoz, hogy megadhassuk a tornyok számát, generáljuk a kettőhatványokat (2^{n-1}-ig), majd összeadjuk őket. Vigyáznunk kell a hatékonyságra, tehát egy-egy kettőhatvány kiszámítását nem kezdjük mindig elölről, hanem felhasználjuk az előző lépésben kiszámítottnak az értékét.

```
Algoritmus tornyok (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)
```

```
// a következő algoritmus rekurzívan
// számolja ki a tornyok számát
Algoritmus tornyokSzáma(n):
   Ha n = 1 akkor
      térítsd 1
   vége(ha)
   térítsd 2 * tornyokSzama(n - 1) + 1
Vége(algoritmus)
```

2. Pascal háromszög

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 ($2 \leq \boldsymbol{r} \leq 32$), anélkül, hogy *kétdimenziós tömböt használnátok*. Bemeneti paraméter az ***r*** természetes szám, kimeneti paraméter az ***r.*** sor elemeinek sorozata.

Megoldás

Az 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];
   }
}
```

3. Csokik

Egy reklámcég egy új csokoládét szeretne népszerűsíteni és ebből a célból azt tervezi, hogy kioszt egy-egy csokit ***n*** gyermeknek ($10 \leq \boldsymbol{n} \leq 10\,000\,000$), 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 ($0 < \boldsymbol{k} < \boldsymbol{n}$). 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, hogy valamilyen számolással/képlettel jussunk eredményhez.

Feltételezzük, hogy $n = 5$ és $k = 4$. A számolást, amely a feladat szövege szerint körkörösen 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.

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

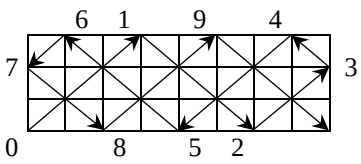
Hely	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Sorszám	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
Csoki																				

Az sz értékének helyes meghatározása ($sz = n - n / lnko(n, k)$)

```
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;
}
```

```
int csokik_v1a(int n, int k){
   return n - n / lnko(n, k);
}
```

4. Fénysugár



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 ($1 < \boldsymbol{a} < 10\,000$) és szélessége ($1 < \boldsymbol{b} < 10\,000$), míg a ***váltSz*** kimeneti paraméter.

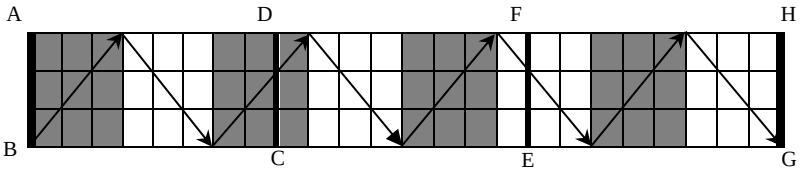
1. *Példa:* ha $\boldsymbol{a} = 8$ és $\boldsymbol{b} = 3$, akkor ***váltSz*** = 9.

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

Megoldás

Két lehetséges megoldást mutatunk be. Az első komoly elemzést igényel és egy kevés matematikai tudást.

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



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

Kiszámítjuk a példánk esetében: $\boldsymbol{k} = \frac{lkkt(\boldsymbol{a},\boldsymbol{b})}{\boldsymbol{b}} = \frac{lkkt(8,3)}{3} = \frac{24}{3} = 8$ 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, $\boldsymbol{p} - 1$-szer változtatja meg az irányát, ahol $\boldsymbol{p}$ a tükrözött (a példában 8 oldalhosszúságú) téglalapok száma. Valóban, $\boldsymbol{p} = \frac{lkkt(\boldsymbol{a},\boldsymbol{b})}{\boldsymbol{a}} = \frac{24}{8} = 3$. Az ábrán látjuk, hogy a fénysugár kétszer ütközik függőlegesen.

Tehát, az irányváltások összes száma: $\boldsymbol{váltSzám} = (\boldsymbol{k} - 1) + (\boldsymbol{p} - 1) = \frac{lkkt(\boldsymbol{a},\boldsymbol{b})}{\boldsymbol{b}} - 1 + \frac{lkkt(\boldsymbol{a},\boldsymbol{b})}{\boldsymbol{a}} - 1$.

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

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

A példánk esetében: $lnko(\boldsymbol{a}, \boldsymbol{b}) = lnko(8, 3) = 1$, tehát $\boldsymbol{váltSzám} = 8 / 1 - 1 + 3 / 1 - 1 = 9$.

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

```
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;
}
```

```
int fenysugar_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;
}
```

5. Robi-kert

Egy modern műszaki megoldásokat kedvelő kertész elhatározta, hogy a kert ágyásainak öntözéséhez egy robotokból álló „hadsereget” fog használni. A vizet a kertet átszelő fősétány 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 fősétányon, amíg a saját ágyásukhoz érnek, az ágyást megöntözik és visszatérnek a forráshoz, hogy a víztartályukat újból megtöltsék. A tevékenységre szánt idő lejárta után, minden robot egyszerre leáll, függetlenül attól, hogy mely állapotban vannak. Eredetileg, a forrásnál egyetlen vízcsap volt. A kertész észrevette, hogy öntözés közben késések 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. Mindegyik robot a saját víztartályát egy másodperc alatt tölti meg. (t, n, d_i, u_i – természetes számok, $1 \leq t \leq 20000$, $1 \leq n \leq 100$, $1 \leq d_i \leq 1000$, $1 \leq u_i \leq 1000$, $i = 1, 2, ..., n$).

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.

Válaszolunk a feladat kijelentésében megfogalmazott kérdésekre:

a. Azok a robotok találkoznak a forrásnál egy adott ***mt*** pillanatban ($1 \leq mt \leq t$), amelyeknek a megfelelő ***q*** értékük ***mt*** osztója.

b. 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.

Algoritmus: létrehozunk egy gyakoriságtömböt, amelyben a munkaidők többszöröseit tároljuk, majd itt megkeressük a maximumot.

```
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;
}
```

IV. Mintatétel

Mintatételben 4.

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. $[-2^x, 2^x - 1]$

Megoldás

Legyen ***x*** = 2. A 0 természetesen ábrázolható akárhány biten, így 2 biten is. Kiszámítjuk a 2^x értékét, ha ***x*** = 2: $2^2 = 4 = 100_{(2)}$, vagyis 3 biten ábrázolható. Tehát az **A.** nem helyes. Ha a **B.** válaszban szereplő ($2^{x-1} - 1$) értékét számítjuk ki ***x*** = 2-re: $2^{2-1} - 1 = 1 = 1_{(2)}$-et kapunk, ami 1 biten ábrázolható, tehát a **B.** változat helyes. A **C.** pontban az intervallum végpontja, ugyanaz, mint a **B.** pontban, de kezdőpontja -2^{x-1}. Ha ***x*** = 2, $-2^{2-1} = -2 = -10_{(2)}$, vagyis 2 bit. A két egész érték között található -1 és 0, amelyek megfelelnek. Tehát a **C.** is helyes. A **D.** válasz kezdőpontja -2^x, ami ***x*** = 2-re -4 = -100(2), tehát a bináris ábrázoláshoz 3 bit szükséges. Így a **D.** nem helyes.

Mintatételben 5.

Legyen az `f(a, b)` algoritmus. Határozzátok meg, hányszor hívja meg önmagát az `f(a, b)` algoritmus a következő programrészletben található hívás következtében:

```
a ← 4
b ← 3
c ← f(a, b)
```

```
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)
```

A. 4-szer
B. 3-szor
C. végtelenszer
D. egyszer sem

Megoldás

Nem szükséges hívássorozatot írni, hiszen könnyen belátható, hogy ha ***a*** = 4, az első három rekurzív hívás csökkenti ***a***-t, míg 1 lesz, amikor a **`különben`** ágon ***a*** értéke 1-gyel nő. Ekkor ***a*** = 2 lesz, tehát az **`akkor`** ágon folytatódik a végrehajtás. Az ***a*** értéke megint csökken, majd a **`különben`** ágon megint nő, és így tovább. Tehát a **C.** válasz helyes.

Mintatételben 7.

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 DIV 3 = 0 és(n MOD 10 = 4 vagy n MOD 10 = 6)`

B. `n MOD 3 = 0 és (n MOD 10 = 4 vagy n MOD 10 = 6)`

C. `(n MOD 3 = 0 és n MOD 10 = 4) vagy (n MOD 3 = 0 és n MOD 10 = 6)`

D. `(n MOD 3 = 0 és n MOD 10 = 4) vagy n 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) ⇒ `24 DIV 3 = 8`, tehát nem 0 ⇒ a (`24 DIV 3 = 0`) kifejezés értéke *hamis*. Mivel a következő operátor az „**és**”, nincs értelme kiértékelni a következő részkifejezést (tudjuk, hogy (*hamis* **és** *hamis*) = *hamis*, valamint (*hamis* **és** *igaz*) = *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 MOD 3 = 0` és `36 MOD 3 = 0`. Ebben az esetben ki kell számolnunk a második részkifejezés értékét is: `(24 MOD 10 = 4 vagy 24 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 (*igaz* **vagy** *hamis* = *igaz*, illetve *hamis* **vagy** *igaz* = *igaz*), végrehajtjuk a zárójel előtti **és** műveletet: (*igaz* **és** *igaz* = *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**.

Mintatételben 8.

Adott a következő alprogram.

Algoritmus f(a):
  Ha a ≠ 0 **akkor**
  térítsd a + f(a - 1)
  különben
  térítsd 0
  vége(ha)
Vége(algoritmus)
	Az alábbi kijelentések közül melyik **hamis**?
	A. ha ***a*** negatív, az alprogram 0-át térít vissza
	B. az ***f*** által kiszámított érték ***a*** * (***a*** + 1) / 4
	C. az alprogram kiszámolja az ***a***-nál kisebb vagy egyenlő természetes számok összegét
	D. az ***f***(-5) hívás végtelen ciklust eredményez

Megoldás

Fontos megjegyezni, hogy a **hamis** válaszokat fogjuk megjelölni. Az **A.** válasz biztos nem helyes, hiszen, ha ***a*** negatív (vagyis nem 0), a rekurzív hívások során tovább csökken, így ***a*** soha nem lesz 0, tehát az algoritmus végtelen ciklust eredményez. A **B.**-t is megjelöljük (hamis), hiszen az algoritmus az első ***a*** természetes szám összegét számolja ki, vagyis ***a*** * (***a*** + 1) / 2-t. A **D.** válasz helyes (lásd az **A.** pont magyarázatát), így nem jelöljük meg. A **C.** nem helyes, mivel, ha ***a*** negatív szám, nincs nála kisebb természetes szám és a **D.** alapján ekkor végtelen ciklust eredményez, tehát nem igaz az állítás.

Mintatételben 9.

Legyen a következő algoritmus:

Algoritmus SA9(a):
  Ha a < 50 **akkor**
  Ha a **MOD** 3 = 0 **akkor**
  térítsd SA9(2 * a - 3)
  különben
  térítsd SA9(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

Megoldás

Első észrevétel: az algoritmusnak akkor lesz vége, ha ***a*** értéke nagyobb lesz, mint 50, amikor téríti ***a***-nak az értékét. Így máris látjuk, hogy a **B.** válasz helyes. Amikor ***a*** kisebb, mint 50, és nem osztható 3-mal, például 16, a belső **Ha** utasítás **különben** ágán folytatódik a végrehajtás, tehát az alprogram meghívja önmagát ***a*** = 31-gyel. Ez az érték még mindig kisebb, mint 50, nem osztható 3-mal, tehát az újrahívás ***a*** = 61-re történik. Ez nagyobb, mint 50, tehát az algoritmus téríti a 61-et (**A.** helyes). Hasonlóan járunk el ***a*** = 31 esetében, amikor egyetlen újrahívás lesz ***a*** = 61-gyel (**D.** helyes). Ha ***a*** = 4, az újrahívások rendre a 7, 13, 25, 49, 97 értékekkel történnek és a térített érték 97, tehát nem 61 **(C.** nem helyes). Következik, hogy helyesek az **A.**, **B.** és **D.** változatok.

Mintatételben 11.

Legyen a következő logikai kifejezés: (***X*** OR ***Z***) AND (NOT ***X*** OR ***Y***). Válasszátok ki ***X***, ***Y***, ***Z*** értékeit úgy, hogy a kifejezés értéke legyen TRUE:

A. ***X*** ← FALSE; ***Y*** ← FALSE; ***Z*** ← TRUE;
B. ***X*** ← TRUE; ***Y*** ← FALSE; ***Z*** ← FALSE;
C. ***X*** ← FALSE; ***Y*** ← TRUE; ***Z*** ← FALSE;
D. ***X*** ← TRUE; ***Y*** ← TRUE; ***Z*** ← TRUE;

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.

Mintatételben 13.

Adott az alábbi pszeudokódban írt algoritmus:

```
beolvas a
Minden i = 1, a - 1 végezd el:
   Minden j = i + 2, a végezd el:
      Ha i + j > a - 1 akkor
         kiír a, ‘ ‘, i, ‘ ‘, j
         új sorba lépés
      vége(ha)
   vége(minden)
vége(minden)
```

Hány megoldáspárt ír ki az algoritmus, ha ***a*** = 8?

A. 13
B. 15
C. 20
D. egyik válasz sem helyes

Megoldás

Ellenőrző táblázatot készítünk. A helyes válasz a **B.**

Mintatételben 15.

Adott az összes ***l*** ∈ {1, 2, 3} hosszúságú sorozat, mely az {*a*, *b*, *c*, *d*, *e*} halmazból való betűket tartalmaz. Ezek közül a sorozatok közül, hány olyan van, melynek elemei szigorúan csökkenő sorrendbe rendezettek és páratlan számú magánhangzót tartalmaznak? (*a* és *e* a magánhangzók)

A. 14
B. 7
C. 81
D. 78

Megoldás

Egy kicsit átfogalmazzuk a feladatot. Az {1, 2, …, 5} halmaz minden ***k*** = 1, 2, 3 elemű részhalmazaira gondolunk, amely az elemeket csökkenő sorrendben tartalmazza és a 2-sek, 3-sok és 4-esek összdarabszáma páratlan szám. Előbb felírjuk a halmazokat (az egyszerűség kedvéért, írhatjuk az elemek növekvő sorrendjében is, hiszen a sorozatok darabszámát ez a tulajdonság nem

befolyásolja). Ezután megszámoljuk azokat, amelyek páratlan számú „mássalhangzót” tartalmaznak.

{1, 2, 3}, {1, 2, 4}, {1, 2, 5}, {1, 3, 4}, {1, 3, 5}, {1, 4, 5}, {2, 3, 4}, {2, 3, 5}, {2, 4, 5}, {3, 4, 5}
{1, 2}, {1, 3}, {1, 4}, {1, 5}, {2, 3}, {2, 4}, {2, 5}, {3, 4}, {3, 5}, {4, 5}

A fenti táblázatban 11 halmaz/sorozat található, ehhez hozzáadjuk a {2}, {3}, {4} egyelemű halmazt/sorozatot. Így összesen 14 halmazunk/sorozatunk van. A helyes válasz: **A.**

Mintatételben 23.

Legyen az ***s*** egy természetes számokat tároló sorozat, ahol

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

(***i*** = 1, 2, ...). 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.**
Elem sorszáma	***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. Észrevesszük, 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.

Mintatételben 30.

Egy 8 sorból álló mátrix csak 0 és 1 elemeket tartalmaz és a következő három tulajdonsággal rendelkezik:
a. az első sor egyetlen 1-es elemet tartalmaz,
b. a j. sor kétszer annyi nullától különböző elemet tartalmaz, mint a $j - 1$. sor, bármely $j \in \{2, 3, \ldots, 8\}$ esetén,
c. az utolsó sor egyetlen 0-val egyenlő elemet tartalmaz.
Összesen hány 0-val egyenlő elemet tartalmaz a mátrix?
A. 777
B. 769
C. 528
D. nem létezik ilyen mátrix

Megoldás

Vegyük észre, hogy nincs precíz információ az oszlopok számát illetően. Számolunk:
Az első sorban $n - 1$ darab 0 található, ahol n a mátrix oszlopainak darabszáma.
A második sorban $n - 2$ darab 0 található
A harmadik sorban $n - 4$ darab 0 található
…
A k. sorban $n - 2^{k-1}$ darab 0 található
A 8. sorban 1 darab 0 található
Ha figyelembe vesszük, hogy a 8. sorban $n - 2^{8-1} = 1$ darab 0 található, meghatározható az oszlopok darabszáma: $n - 2^7 - 1 = 0$, $n = 2^7 + 1 = 129$.
Akkor a 0-ák darabszáma a mátrixban:
$n - 2^0 + n - 2^1 + \ldots + n - 2^6 + 1 = 7 * n + 1 - (2^0 + 2^1 + \ldots + 2^6) = 7 * 129 + 1 - (2^7 - 1) = 777$.
Tehát az **A.** válasz helyes.

Eddig már létezik:

Mintatétel 12.: a Múlt heti házik megoldásai.docx-ben
Mintatétel 19.: a 3. Házi Megoldásokkal.docx-ben
Mintatétel 20.: a 3. Házi Megoldásokkal.docx-ben
Mintatétel 21.: a Rekurzió melletti Feladatok.pptx-ben
Mintatétel 22.: a Múlt heti házik megoldásai.docx-ben
Mintatétel 24.: a Múlt heti házik megoldásai.docx-ben
Mintatétel 26.: a 3. Házi Megoldásokkal.docx-ben
Mintatétel 27.: a 3. Házi Megoldásokkal.docx-ben
Mintatétel 28.: a Múlt heti házik megoldásai.docx-ben
Mintatétel 29.: a Múlt heti házik megoldásai.docx-ben

Vigyázzatok:

A honlapra feltöltött pdf tartalmaz néhány elírást (szövegben is, de főleg a javítókulcsban).