BABEŞ-BOLYAI TUDOMÁNYEGYETEM
MATEMATIKA-INFORMATIKA KAR

Matek-Info verseny - minta

Informatika írásbeli

A versenyzők figyelmébe:

1. Minden tömböt 1-től kezdődően indexelünk.
2. A rácstesztekre (A rész) egy vagy több helyes válasz lehetséges. A válaszokat a vizsgadolgozatba írjátok (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 írjátok.
   - **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 leírjátok szavakkal az alprogramokat (algoritmusokat), és megindokoljátok a megoldásotok lépéseit***. Feltétlenül írjatok ***megjegyzéseket*** (kommenteket), amelyek segítik az adott megoldás technikai részleteinek megértését. Adjátok meg az azonosítok jelentését és a fölhasznált adatszerkezeteket stb. Ha ez hiányzik, a tételre kapható pontszámotok 10%-kal csökken.
   - **d.** Ne használjatok különleges fejállományokat, előredefiniált függvényeket (például *STL*, karakterláncokat feldolgozó sajátos függvények stb.).

A RÉSZ (60 pont)

A.1. Vajon mit csinál? (6 pont)

Adott az `alg(x, b)` algoritmus (alprogram), amelynek bemeneti paraméterei az x és b természetes számok ($1 \leq x \leq 1000$, $1 < b \leq 10$).

```
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)
```

Állapítsátok meg a fenti algoritmus hatását.

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

A.2. Mit fog kiírni? (6 pont)

Legyen a következő program:

Varianta C++/C

```
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, p = 0, a[10];
   a[1] = -1; a[2] = 0; a[3] = 3;
   int s = sum(n, a, p);
   cout << s << ";" << p; // printf("%d;%d", s, p);
   return 0;
}
```

Varianta Pascal

```
type vector = array[1..10] of integer;

function sum(n:integer; a:vector; 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 : vector;
Begin
    n := 3; a[1] := -1; a[2] := 0; a[3] := 3; p := 0;
    s := sum(n, a, p);
    writeln(s, ';', p);
End.
```

Mit fog kiírni a program?

A. `0;0`
B. `2;0`
C. `2;2`
D. Egyik válasz sem helyes

A.3. Logikai kifejezés (6 pont)

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;

A.4. Számolás (6 pont)

Legyen a számol(a, b) algoritmus, amelynek bemeneti paraméterei az ***a*** és ***b*** pozitív természetes számok, ahol $1 \leq \boldsymbol{a} \leq 1000$, $1 \leq \boldsymbol{b} \leq 1000$.

```
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 kerül végrehajtásra

A.5. Elem beazonosítása(6 pont)

Legyen az (1, 2, 3, 2, 5, 2, 3, 7, 2, 4, 3, 2, 5, 11, ...) sorozat, amelyet a következőképpen hoztunk létre: kiindulva 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, $1 \leq \boldsymbol{n} \leq 1000$)?

A.
```
Algoritmus beazonosítás(n):
   a ← 1, b ← 1, c ← 1
   Amíg c < n végezd el
      a ← a + 1, b ← a, c ← c + 1, d ← 2
      f ← false
      Amíg c ≤ n és d ≤ a DIV 2 végezd el
         Ha a MOD d = 0 akkor
            c ← c + 1, b ← d, f ← true
         vége(ha)
         d ← d + 1
      vége(amíg)
      Ha f akkor c ← c - 1
      vége(ha)
   vége(amíg)
   térítsd b
Vége(algoritmus)
```

B.
```
Algoritmus beazonosítás(n):
   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)
```

C.
```
Algoritmus beazonosítás(n):
   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)
```

D.
```
Algoritmus beazonosítás(n):
   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)
```

A.6. Törzstényezők (6 pont)

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 ($1 \leq \boldsymbol{n} \leq 10000$, $2 \leq \boldsymbol{d} \leq 10000$, $0 \leq \boldsymbol{k} \leq 10000$).

```
Algoritmus törzsTényezők(n, d, k, x):
   Ha n MOD d = 0 akkor
      k ← k + 1
      x[k] ← d
   vége(ha)
   Amíg n MOD d = 0 végezd el
      n ← n 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 programrészletben található hívás következtében:

```
n ← 120
d ← 2
k ← 0
törzsTényezők(n, d, k, x)
```

A. 3-szor
B. 5-ször
C. 9-szer
D. ugyanannyiszor, mint a következő programrészlet esetében

```
n ← 750
d ← 2
k ← 0
törzsTényezők(n, d, k, x)
```

A.7. Vajon mit csinál? (6 pont)

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

```
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 kiszámít a fenti algoritmus:

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

A.8. Logikai kifejezés (5p)

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 `TRUE`:

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

A.9. A bemeneti paraméter értékei (6 pont)

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

A.10. Hiányzó utasítások (6 pont)

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)
```

Mely módosítást kellene elvégezni a `keresés(x, n, érték)` algoritmusban ahhoz, hogy ha meghívjuk, ez határozza meg, hogy az ***érték*** megtalálható vagy sem az ***n*** elemű ***x*** sorozatban (***n*** – egy 0-nál szigorúan nagyobb természetes szám)?

A. Az 5. sort módosítjuk: `térítsd ((x[n] = érték) és keresés(x - 1, n, érték))`
B. Az 5. sort módosítjuk: `térítsd ((x[n] = érték) vagy keresés(x, n - 1, érték))`
C. Az 5. sort módosítjuk: `Ha (x[n] = érték) akkor térítsd` *true* `különben térítsd keresés(x, n - 1, érték)`
D. nem kell módosítani egyetlen utasítást sem

Partea B (30 pont)

1. Előszelet (15 pont)

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 egyszámjegyű 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 *előszelet*ének nevezünk, ha a ***q*** szám első ***k*** számjegyéből alkotott szám (balról jobbra tekintve) egyenlő ***p***-vel. Például, 17 előszelete 174-nek, és 1713 előszelete 1713242-nek.

Legyen az ***sz*** természetes szám ($0 < sz \leq 30000$) és egy ***m*** soros és ***n*** oszlopos ($0 < m \leq 100$, $0 < n \leq 100$) ***A*** mátrix (kétdimenziós tömb), amelynek elemei 30000-nél kisebb természetes számok. Adva van a `sorsSzámjegy(x)` algoritmus, amely meghatározza az ***x*** számhoz rendelt sors-számjegyet:

```
Algoritmus sorsSzámjegy(x):
    s ← 0
   Amíg x > 0 végezd el
      s ← s + x MOD 10
      x ← x DIV 10
      Ha x = 0 akkor
         Ha s < 10 akkor
            térítsd s
         különben
            x ← s
            s ← 0
         vége(ha)
      vége(ha)
   vége(amíg)
   térítsd s
Vége(algoritmus)
```

Követelmények:

a. 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. (**5 pont**)
b. Írjátok le a `sorsSzámjegy(x)` algoritmus rekurzív változatának (amelyet kidolgoztatok az a. pontnál) a matematikai modelljét. (**3 pont**)
c. Írjatok alprogramot, amely – felhasználva a `sorsSzámjegy(x)` alprogramot – meghatározza az ***sz*** szám leghosszabb előszeletét (***prefix***), amelyet az adott mátrix 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***. (**7 pont**).

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

2. Robi-kert (15 pont)

Egy modern műszaki megoldásokat kedvelő kertész elhatározta, hogy a kert ágyásainak öntözéséhez egy robotokból álló „hadsereget” fog használni. A vizet a kertet átszelő fő sétány 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 fő sétányon, 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. Még ismert, hogy mindegyik robot a saját víztartályát egy másodperc alatt tölti meg. (t, n, d_i, u_i – természetes számok, $1 \leq t \leq 20000$, $1 \leq n \leq 100$, $1 \leq d_i \leq 1000$, $1 \leq u_i \leq 1000$, $i = 1, 2, ..., n$).

Követelmények:

a. Soroljátok fel azokat a robotokat, amelyek találkoznak a forrásnál egy adott mt időpillanatban ($1 \leq mt \leq t$). Indokoljátok meg a választ. (**3 pont**)
Megjegyzés: a robotokat a sorszámaik alapján azonosítjuk be.

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. (**2 pont**)

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***. (**10 pont**)

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.

Megjegyzések:

1. Minden tétel kidolgozása kötelező.
2. A piszkozatokat nem vesszük figyelembe.
3. Hivatalból jár 10 pont.
4. Rendelkezésetekre áll 3.5 óra.

JAVÍTÁSI KULCS & MEGOLDÁSOK

HIVATALBÓL ..... **10 pont**

A RÉSZ ..... **60 pont**
A. 1. Vajon mit csinál? Válasz: C, D ..... **6 pont**
A. 2. Mit fog kiírni? Válasz: B ..... **6 pont**
A. 3. Logikai kifejezés Válasz: A, D ..... **6 pont**
A. 4. Számolás Válasz: A, C, D ..... **6 pont**
A. 5. Elem beazonosítása Válasz: A ..... **6 pont**
A. 6. Törzsténeyezők Válasz: A, D ..... **6 pont**
A. 7. Vajon mit csinál? Válasz: A ..... **6 pont**
A. 8. Logikai kifejezés Válasz: A, B, D ..... **6 pont**
A.9. A bemeneti paraméter értékei Válasz: A, B, D ..... **6 pont**
A.10. Hiányzó utasítások Válasz: B, C ..... **6 pont**

B RÉSZ ..... **30 pont**
B. 1. Előszelet ..... **15 pont**
B.1.a. A `sorsSzámjegy(x)` algoritmus rekurzív változata ..... **5 pont**
– a fejléc tiszteletben tartása ..... 1 pont
– a rekurzív hívás leállási feltétele ..... 1 pont
– újrahívás (logika, paraméterek) ..... 2 pont
– térített értékek ..... 1 pont

```
Algoritmus sorsSzámjegy(x):
   Ha x > 9 akkor
      s ← x MOD 10 + sorsSzámjegy(x DIV 10)
      Ha s < 10 akkor
         térítsd s
      különben
         térítsd sorsSzámjegy(s)
      vége(ha)
   különben
      térítsd x
   vége(ha)
Vége(algoritmus)
```

B.1.b. A `sorsSzámjegy(x)` algoritmus matematikai modellje ..... **3 pont**

$$sorsSzámjegy(x) = \begin{cases} x, & ha\ x < 10 \\ sorsSzámjegy(x\ MOD\ 10) + sorsSzámjegy(x\ DIV\ 10)), & különben \end{cases}$$

B.1.c. Algoritmus, amely meghatározza a leghosszabb előszeletet ..... **7 pont**

1 változat: a mátrixot egyszer járjuk be és felépítjük a `megjelent` vektort, amelyben tároljuk, hogy egy bizonyos számjegy megjelent-e vagy sem a mátrix elemeinek sorsszámjegyei között. Tároljuk az ***sz*** szám számjegyeit egy tömbben. Bejárjuk ezt a tömböt (a legnagyobb nagyságrendűvel kezdődően) és vizsgáljuk, hogy az előzőleg meghatározott tömbben megjelenik-e. **7 pont**
Megjegyzés: az ötlet megvalósítható egy statisztikatömbbel is (a `megjelent` tömb helyett).

```
const int MAXSZAMJEGYDB = 10;
int legnagyobbEloszelet(int nr, int m, int n, int A[][MAX]){
   bool megjelent[MAXSZAMJEGYDB];
   int szJ[MAX]; int szamjegyekSzama;
   for (int i = 0; i < MAXSZAMJEGYDB; i++) ..... // inicializálás
      megjelent[i] = 0;
   for (i = 0; i < m; i++){
      for (int j = 0; j < n; j++){
         megjelent[sorsSzamjegy(A[i][j])] = 1; ..... // megjelent A[i][j]) sors-számjegye
      }
   }
   szamjegyekSzama = 0; ..... // az adott szám számjegyeinek darabszáma
   while (nr > 0){
      szJ[szamjegyekSzama++] = nr % 10; ..... // az adott szám számjegyeinek sorozata
      nr /= 10;
   }
```

```
   i = szamjegyekSzama - 1; ..... // bejárjuk az sz szám számjegyeinek szJ tömbjét
   while ((i >= 0) && (megjelent[szJ[i]])).. // létezik az aktuális számjeggyel egyenlő sors-számjegy
      i--;
   return szamjegyekSzama - i - 1; ..............................// a leghosszabb előszelet hossza
}
```

3. <u>változat:</u> Tároljuk az ***sz*** szám számjegyeinek tömbjét. Megkeresünk minden számjegyet a mátrixban (így a mátrixot többször is bejárjuk) .....**5 pont**

```
bool szamjegyetKeres(int szJ, int m, int n, int A[][MAX]){
   for(int i = 0; i < m; i++){
      for(int j = 0; j < n; j++){
         if(szJ == sorsSzamjegy(A[i][j]))
                              // létezik A[i][j], amelynek sors-számjegye egyenlő a keresett szJ-vel
            return true;
      }
   }
   return false;                                                    // sikertelen keresés
}
int prefixMaxSzJ_v2(int nr, int m, int n, int A[][MAX]){
   int szJ[MAX];
   int szamjegyekSzama = 0;                                  // az adott szám számjegyeinek darabszáma
   while(nr > 0){
      szJ[szamjegyekSzama++] = nr % 10;                       // az adott szám számjegyeinek sorozata
      nr /= 10;
   }
   int i = szamjegyekSzama - 1;                                      // bejárjuk a számjegyek sorozatát
   int p = 0;
   while(i >= 0){
      if(szamjegyetKeres(szJ[i], m, n, A)){                  // keressük a számjegyeket az A mátrixban
         p++;
         i--;
      }
      else
         return p;
   }
   return p;                  // egymás után megtalált számjegyek az A mátrixban (az előszelet hossza)
}
```

B. 2. Robi kert .....15 pont

Ha ***n*** = 5, ***d*** = (1, 2, 1, 2, 1), ***u*** = (1, 3, 2, 1, 3), ***t*** = 32, kiszámítjuk ***q*** értékeit:

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 => (4, 8, 5, 6, 6)

Inicializáljuk a ***t*** = 32 elemű ***v*** vektort 0 értékekkel. Vesszük rendre a ***q*** értékeit, és minden robot esetében növeljük a ***q*** többszöröseinek megfelelő elemeket 1-gyel.

q = 4

	1	2	3	**4**	5	6	7	**8**	9	10	11	**12**	13	14	15	**16**	17	18	19	**20**
v	0	0	0	**1**	0	0	0	**1**	0	0	0	**1**	0	0	0	**1**	0	0	0	**1**

	21	22	23	**24**	25	26	27	**28**	29	30	31	**32**
v	0	0	0	**1**	0	0	0	**1**	0	0	0	**1**

q = 8

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
v	0	0	0	1	0	0	0	2	0	0	0	1	0	0	0	2	0	0	0	1

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

q = 5

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
v	0	0	0	1	1	0	0	2	0	1	0	1	0	0	1	2	0	0	0	2

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

q = 6

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
v	0	0	0	1	1	1	0	2	0	1	0	2	0	0	1	2	0	1	0	2

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

q = 6

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
v	0	0	0	1	1	2	0	2	0	1	0	3	0	0	1	2	0	2	0	2

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

Meghatározzuk a ***v*** tömb maximumát. A példa esetében a maximum 4 (a 24. időpillanatban), tehát szükséges még 4 – 1 = 3 új vízcsap.

B.2.a. Egy adott ***mt*** ($1 \leq mt \leq t$) időpillanatban azok a robotok találkoznak a forrásnál, amelyeknek esetében ***q*** értéke az ***mt*** többszöröse (egyenlő az ágyáshoz vezető út megtételéhez, a vissza út megtételéhez, az öntözéshez és a tartály megtöltéséhez szükséges idővel) ..... 3 pont

B.2.b. a legkevesebb vízcsap, amit a kertésznek fel kell szerelnie egyenlő a ***v*** tömb maximumával, amiből kivonunk egyet (a már létező vízcsapot), ahol ***v*** elemei azoknak a robotoknak a darabszámát tárolják, amelyek az egyes időpillanatokban találkoznak a forrásnál ..... 2 pont

B.2.c. Az algoritmus

- V1: egy gyakoriság-tömb használata minden robot munkaidejének többszöröseire vonatkozóan ..... 10 pont
  - c.1. a bemeneti és kimeneti paraméterek tiszteletben tartása ..... **2 pont**
  - c.2. munkaidő kiszámítása (***q*** = 2 * útidő + öntözés + tartály feltöltése) ..... **1 pont**
  - c.3. a gyakoriság-tömb feldolgozása ..... **4 pont**
    - c.3.1. inicializálás ..... 1 pont
    - c.3.2. aktualizálás ..... 3 pont
      1. v1a vagy v1b: bejárás ***q***-ról ***q***-ra ..... 3 pont
      2. v2a sau v2b: bejárás egyesével és annak vizsgálata, hogy ***q*** többszöröse vagy sem .. 2 pont
  - c.4. a maximális vízcsapszám meghatározása ..... 2 **pont**
      3. párhuzamosan az aktualizálással vagy az aktualizálások után ..... 1 pont
  - c.5. a felszerelendő új vízcsapok számának meghatározása (max - 1) ..... **1 pont**

```
int robiV1a(int n, int d[], int u[] int t){
    int aux[200000];
    int max = 1;
    for (int i = 1; i <= t; i++)
         aux[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){
                 aux[i]++;
                 if (max < aux[i])
                         max = aux[i];
         }       //for i
    }    //for j
    return max - 1;
}
```

```
int robiV1b(int n, int d[], int u[], int t){
    int aux[200000];
    int max = 1;
    for (int i = 1; i <= t; i++)
         aux[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){
                 aux[i]++;
         }       //for i
    }    //for j
    for (int i = 1; i <= t; i++){
         if (max < aux[i])
                 max = aux[i];
    }    //for i
    return max - 1;
}
```

```
int robiV2a(int n, int d[], int u[], int t){
    int aux[200000];
    int max = 1;
    for (int i = 1; i <= t; i++)
         aux[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 + 1){
                 if (i % q == 0){
                         aux[i]++;
                         if (max < aux[i])
                                 max = aux[i];
                 }       //if (i % q)
         }       //for i
    }    //for j
    return max - 1;
```

```
int robiV2b(int n, int d[], int u[], int t){
    int aux[200000];
    int max = 1;
    for (int i = 1; i <= t; i++)
         aux[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 + 1){
                 if (i % q == 0){
                         aux[i]++;
                 }       //if (i % q)
         }       //for i
    }    //for j
    for (int i = 1; i <= t; i++){
         if (max < aux[i])
                 max = aux[i];
```

```
}
```

```
    }    //for i
    return max - 1;
}
```

- V2: szimulálás ..... 8 pont
  - c.1. a bemeneti és kimeneti paraméterek tiszteletben tartása ..... 2 pont
  - c.2. munkaidő kiszámítása (***q*** = 2 * útidő + öntözés + tartály feltöltése) ..... 1 pont
  - c.3. ismétlő struktúra, amely az időre vonatkozik ..... 0.5 pont
  - c.4. ismétlő struktúra, amely a robotokra vonatkozik ..... 0.5 pont
  - c.5. egy adott időpillanatban szükséges vízcsapok számának meghatározása ..... 1 pont
  - c.6 a maximális vízcsapszám meghatározása ..... 2 pont
  - c.7. a felszerelendő új vízcsapok számának meghatározása ..... 1 pont

```
int robiV3(int n, int d[], int u[], int t){
    int minVizcsapSzam = 1;
    int idoPillanat = 1;
    while (idoPillanat <= t){
       int hanyVizcsap = 0;
       for (int j = 1; j <= n; j++){
           int q = 2 * d[j] + u[j] + 1;
           if (idoPillanat % q == 0)           // ha az aktuális időpillanatban a j-dik robot a forrásnál van
              hanyVizcsap++;
       }
       if (hanyVizcsap > minVizcsapSzam)
           minVizcsapSzam = hanyVizcsap;
       idoPillanat++;
    }
    return minVizcsapSzam - 1;
}
```