BABEŞ-BOLYAI TUDOMÁNYEGYETEM
MATEMATIKA-INFORMATIKA KAR

Felvételi verseny – 1. Tételsor

Informatika írásbeli

A versenyzők figyelmébe:

1. Minden tömböt 1-től kezdődően indexelünk.
2. A rácsteszteket (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 (30 pont)

A.1. Vajon mit csinál? (5p)

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

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

A.2. Számolás (5p)

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

A.3. 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 *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*;

A.4. Mit fog kiírni? (5p)

Legyen a következő program:

C változat

```
#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++ változat

```
#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 változat

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

function sum(n : integer; a : vector;
             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 : vector;
begin
  n := 3; a[1] := -1; a[2] := 0;
  a[3] := 3; p := 0;
  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

A.5. Szerencsés szám (5p)

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*** értéke -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)
```

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)
```

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)
```

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

A.6. Tegyél 'b' betűket (5p)

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

```
Algoritmus tegyélB(mat, n, i, j):
   Ha i ≤ 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
B. ugyanannyiszor, mint a következő programrészlet esetében:
```
n ← 9, i ← 3, j ← 5
tegyélB(mat, n, i, j)
```
C. 10-szer
D. 0-szor
E. végtelenszer

B rész (60 pont)

B.1. Számolás - karakterekkel (10 pont)

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

```
Algoritmus számolásKarakterekkel(s, n, p, q, szám):
   eredmény ← 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)
```

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

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

B.2. Periódus (25 pont)

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

Írjatok algoritmust, amely meghatározza az ***n*** elemű ***x*** karaktersorozat (***n*** – természetes szám, $2 \leq n \leq 200$) ***pm*** *maximális periódusát*. Ha az ***x*** sorozatnak nincs periódusa, ***pm*** értéke -1. Az algoritmus bemeneti paraméterei ***x*** és ***n***, kimeneti paramétere ***pm***.

1. *Példa:* ha ***n*** = 8 és ***x*** = `"abababab"`, akkor ***pm*** = 4.

2. *Példa:* ha ***n*** = 7 és ***x*** = `"abcxabc"`, akkor ***pm*** = -1.

B.3. Robi-kert (25 pont)

Egy modern műszaki megoldásokat kedvelő kertész elhatározta, hogy a kert ágyásainak öntözéséhez egy robotokból álló „hadsereget” fog használni. A vizet a kertet átszelő 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. 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.
   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.

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.

Megjegyzések:

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

BAREM
OFICIU ..... 10 puncte

Partea A ..... 30 puncte
A. 1. Oare ce face? Răspunsul A ..... 5 puncte
A. 2. Calcul. Răspunsurile B, D..... 5 puncte
A. 3. Expresie logică. Răspunsurile A, B, D, E ..... 5 puncte
A. 4. Ce se afișează? Răspunsul D..... 5 puncte
A. 5. Număr norocos. Răspunsurile B, C..... 5 puncte
A. 6. Pune b. Răspunsurile A, B ..... 5 puncte

Partea B ..... 60 puncte
B. 1. Calcul..... 10 puncte
- respectarea parametrilor de intrare și ieșire..... 2 puncte
- condiția de oprire din recursivitate..... 1 puncte
- valoarea returnată la oprirea recursivității..... 1 puncte
- condiția pentru caracter diferit de cifră ..... 2 puncte
- valoarea returnată în cazul unui caracter diferit de cifră ..... 2 puncte
- valoarea returnată în cazul unui caracter cifră..... 2 puncte

```
Subalgoritm calculCuCaractere(s, n, p, q, nr):
   Dacă p > q atunci
      returnează nr
   altfel
      Dacă s[p] < '0' sau s[p] > '9' atunci
         returnează nr + calculCuCaractere(s, n, p + 1, q, 0)
      altfel
         returnează calculCuCaractere(s, n, p + 1, q, 10 * nr + s[p] - '0')
      SfDacă
   SfDacă
SfSubalgoritm
```

B. 2. Perioadă ..... 25 puncte
- respectarea parametrilor de intrare și ieșire..... 2 puncte
- parcurgerea valorilor posibile ale perioadei ..... maxim 10 puncte

  Notă: punctajul acordat depinde și de următoarele aspecte:
  i. parcurgerea valorilor posibile ale perioadei
  ii. considerarea ca perioadă a divizorilor lui ***n***
- verificarea periodicității ..... maxim 13 puncte

  Notă: punctajul acordat depinde și de numărul de structuri repetitive folosite

B. 3. Robi grădină ..... 25 puncte
a. la un anumit moment de timp ***mt*** ($1 \leq mt \leq t$) se întâlnesc la izvor roboții care au valoarea ***q*** (egală cu suma dintre timpul necesar deplasării până la strat și înapoi, timpul necesar udării stratului și timpul necesar umplerii rezervorului) multiplu de ***mt*** ..... 5 puncte
b. numărul minim de robinete suplimentare este egal cu maximul vectorului ***v*** - 1, unde vectorul v reține, pentru fiecare moment de timp, câți roboți se întâlnesc la izvor în momentul respectiv ..... 5 puncte
c. Dezvoltare subalgoritm
- V1: folosirea unui vector de frecvență pentru multiplii timpilor de lucru ai fiecărui robot..... 15 puncte
  a. respectarea parametrilor de intrare și ieșire ..... 2 puncte
  b. calcul timp de lucru (***q*** = 2 * deplasare + udare + încărcare)..... 2 puncte
  c. prelucrarea vectorului de frecvențe..... 5 puncte
  d. stabilirea frecvenței maximale..... 4 puncte
  e. determinarea numărului de robinete suplimentare..... 2 puncte
- V2: simulare..... 10 puncte
  a. respectarea parametrilor de intrare și ieșire ..... 2 puncte
  b. calcul timp de lucru (***q*** = 2 * deplasare + udare + încărcare)..... 2 puncte
  c. structura repetitivă pentru timp..... 1 puncte
  d. structura repetitivă pentru roboți..... 1 puncte
  e. stabilirea numărului de robinete necesare la un anumit moment de timp..... 1 punct
  f. stabilirea numărului maxim de robinete ..... 1 punct
  g. determinarea numărului de robinete suplimentare..... 2 puncte

Partea B (soluții) ..... 60 puncte
B. 1. Calcul..... 10 puncte

```
Subalgoritm calculCuCaractere(s, n, i, nr):
   Dacă i > n atunci
      returnează nr
    altfel
      Dacă s[i] < '0' sau s[i] > '9' atunci
         returnează nr + calculCuCaractere(s, n, i + 1, 0)
       altfel
         returnează calculCuCaractere(s, n, i + 1, 10 * nr + s[i] - '0')
      SfDacă
   SfDacă
SfSubalgoritm
```

B. 2. Perioadă maximală..... 25 puncte

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

int perioadaMax(int n, char x[]){
    int perioada = -1;
    for (int per = 2; per * per <= n; ++per){
         if (n % per == 0){ // perioada trebuie sa fie printre divizorii lui n
                 if (verif(n, x, n / per))
                         return n / per;

                 if ((per * per < n) && verif(n, x, per)) {
                         perioada = per;
                 }
         }
    }
    return perioada;
}
```

B. 3. Robi grădină ..... 25 puncte

```
int robiGradina(int n, int d[], int u[], int t){
    int aux[200000];      //aux(i) va retine cate robinete sunt necesare la momentul i
    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;
                                        //se marchează multiplii lui q in vectorul aux
         for (int i = q; i <= t; i = i + q){
                 aux[i]++;
                 if (max < aux[i])                      //se determina maximul din aux
                         max = aux[i];
         }
    }
    return max - 1;
}
```