INFORMATIKA – tétel
2026

ELIGAZÍTÁS:

- 1 pont hivatalból;
- Az 1-4 feladatokban (a pszeudokód programrészletekben): (1) a kiírások szóközzel elválasztva történnek; (2) a „/” operátor osztási hányadost ad meg, a „%” operátor pedig osztási maradékot; (3) a tömbök 0-tól indexeltek.
- Az 5-9 feladatok esetében használj alprogramot, valahányszor célszerűnek találod. Maximális pontszám **hatékony megoldásra** kapható (elsődleges szempont az algoritmusok időigénye). Lásd el **beszédes kommentekkel** programjaidat.
- A bemeneti adatok helyesnek tekinthetők, és minden bemeneti egész szám 32 bites egész típusban ábrázolható.

FELADATOK:

1. Legyen az alábbi pszeudokód programrészlet:

```
n = ?
s = 0
┌amíg n > 0 végezd
|  s = s + (n % 10)
|  n = n / 10
└■
kiír s
```

Melyik természetes szám szerepelhetett, például, a `?` helyén, ha `77` került kiírásra? [1 pont]

`599999999` (számjegyösszege 77)

2. Legyen az alábbi pszeudokód programrészlet:

```
A = [3, 1, 4, 1, 5]
i = 0
s = 0
┌amíg i < 5 végezd
|  ┌ha A[i] % 2 == 1 akkor
|  |  s = s + A[i]
|  |különben
|  |  s = s - A[i]
|  └■
|  i = i + 2
└■
kiír s
```

Mit ír ki a fenti programrészlet? [1 pont]

4

3. Legyen az alábbi pszeudokód programrészlet:

```
M = [[1, 2, 3],
     [4, ?, 6],
     [7, 8, 9]]
s = 0
┌minden i = 0, 2 végezd
|  s = s + M[i][2 - i]
```

```
└■
kiír s
```

Melyik természetes szám szerepelhetett a `?` helyén, ha `15` került kiírásra? [1 pont]

`5` (a mellékátló elemeinek összegét számolja ki)

4. Legyen az alábbi pszeudokód programrészlet:

```
x = 3
y = 1
┌minden i = 1, 5 végezd
|  x = x + y
|  y = x - y
└■
kiír x, ' ', y
```

Mi kerül kiírásra? [1 pont]

`29 18`

5. Egy okos üvegházban a rendszer figyeli a hőmérsékletet (Celsius fok, egész szám) és a páratartalmat (százalék, egész szám). Írj Pascal vagy C/C++ *programot*, amely bekéri ezt a két értéket, majd kiírja, hogy `RIASZTAS` vagy `MINDEN RENDBEN`. A riasztónak akkor kell bekapcsolnia, ha a hőmérséklet meghaladja a `35` fokot, VAGY ha a hőmérséklet `30` fok felett van ÉS a páratartalom eléri a `80%`-ot. Minden más esetben a rendszer rendben működik. [1 pont]

Példa bement:
```
32 85
```
Kimenet:
```
RIASZTAS
```

```
int main() {
    int homerseklet, paratartalom;
    cin >> homerseklet >> paratartalom;
    if (homerseklet > 35 || (homerseklet > 30 && paratartalom >= 80)) {
        cout << "RIASZTAS";
    } else {
        cout << "MINDEN RENDBEN";
    }
    return 0;
}
```

6. Adott egy `n`-hosszú (`2≤n≤10000`), egész számokat tartalmazó `nums` tömb (`1≤nums[i]≤n`), amely eredetileg az `1`, `2`, ..., `n` számokat tartalmazta pontosan egyszer, nem feltétlenül rendezett formában! Egy hiba következtében egy szám kétszer szerepel a tömbben, egy másik szám pedig hiányzik. Írj *hatékony* C/C++/Pascal *függvényt*, amely meghatározza melyik szám szerepel kétszer, és melyik szám hiányzik, majd írd ki őket ilyen sorrendben a képernyőre szóközzel elválasztva. [1 pont]

Lehetséges C/C++ függvényfejléc:
```
void findErrorNums(int nums[], int n)
```

Példa bemenet:
```
n = 4
nums = [1, 2, 2, 4]
```
Kimenet:
```
2 3
```

Példa bemenet:
```
n = 6
nums = [4, 3, 6, 2, 1, 1]
```

Kimenet:
`1 5`

```
void findErrorNums(int nums[], int n){
    int duplikalt = -1;
    int hianyzik = -1;
    // A módszer lényege:
    // A számokat indexként használjuk (érték x --> index x-1).
    // Végigmegyünk a tömbön, és az adott indexen lévő elemet negatívvá tesszük,
    // ezzel jelezzük, hogy ezt a számot már láttuk.
    // Ha egy indexhez tartozó elem már negatív, akkor az a szám duplikált.
    // A végén az az index marad pozitív, amelyikhez tartozó szám hiányzik.
    for (int i = 0; i < n; i++) {
        int index = abs(nums[i]) - 1;
        if (nums[index] < 0) {
            duplikalt = abs(nums[i]);
        }
        else {
            nums[index] = -nums[index];
        }
    }
    // Hiányzó szám keresése
    for (int i = 0; i < n; i++) {
        if (nums[i] > 0) {
            hianyzik = i + 1;
        }
    }
    // Eredmény kiírása
    cout << duplikalt << " " << hianyzik << endl;
}
```

7. Az `Emter` versenyen `N` versenyző vesz részt. Minden versenyzőnek `9` feladatot kell megoldania, és megoldásaikat pontozva rangsoroljuk őket, így minden feladatra egy helyezést érnek el `1` és `N` között. Azonban minden feladat esetén csak az első `K` helyezett kap pontot úgy, hogy a `K`. helyezett `1` pontot kap, és a nála jobb helyezettek pontszáma helyezésenként `2` ponttal növekszik.

A verseny végén minden versenyző összpontszámát kell meghatározni. Az egyetemre a pontszám alapján a legjobb `K` versenyző jut be. Holtverseny esetén ez a szám `K`-nál nagyobb is lehet, mert a `K`. helyen álló versenyzővel azonos pontszámot elért versenyzők mind bejutnak.

Írj Pascal vagy C/C++ *függvényt*, amely paraméterként megkapja `N` értékét (`1<N≤60`), `K` értékét (`1<K≤10`), valamint egy `N×9` méretű kétdimenziós tömböt, amely az `N` versenyző helyezéseit tartalmazza. A függvény számolja ki és írja ki minden versenyző összpontszámát, majd térjen vissza az első helyen végzett versenyző sorszámával; több első helyezett esetén a legnagyobb sorszámút adja vissza. [1 pont]

Lehetséges C/C++ függvényfejléc:
`int eredmeny(int N, int K, int helyezesek[][9])`

Példa (részletek):
ha `K = 4` és `N = 50`, akkor a pontozás a következő: *1. hely*: `7` pont, *2. hely*: `5` pont, *3. hely*: `3` pont, *4. hely*: `1` pont.
ha egy versenyző helyezései feladatonként: `1 4 1 3 2 1 4 5 46`
akkor az összpontszáma: `7+1+7+3+5+7+1+0+0 = 31`.

```
int eredmeny(int N, int K, int helyezesek[][9]) {
    int pont[60] = {0};
    int maxPont = -1;
    int elso = -1;
    // Minden versenyző összpontszámának kiszámítása
    for (int i = 0; i < N; i++) {
        pont[i] = 0;
        for (int j = 0; j < 9; j++) {
```

```
            int h = helyezesek[i][j];
            if (h <= K) {
                pont[i] += 2 * (K - h) + 1;
            }
        }
        // Kiírás
        cout << pont[i] << endl;
        // Legjobb versenyző keresése
        // Holtversenynél a nagyobb sorszámú kell
        if (pont[i] >= maxPont) {
            maxPont = pont[i];
            elso = i + 1;   // sorszám 1-től
        }
    }
    return elso;
}
```

8. Adott n ($1<n<10000$) darab valós számpár (a_i, b_i), amelyek az $y_i=a_ix+b_i$ egyeneseket határozzák meg. Tegyük fel, hogy ezek az egyenesek a $[0,1]$ intervallumban rendezettek, azaz minden $x \in [0,1]$-re: $y_1(x)<y_2(x)<\cdots<y_n(x)$ vagyis az egyenesek nem metszik egymást ebben az intervallumban.

Adott egy (x,y) pont, ahol $0 \le x \le 1$. Írjunk *hatékony függvényt*, amely eldönti:

- ha a pont *pontosan egy egyenesre esik*, akkor írja ki, hogy melyik egyenesre,
- ha a pont *két egyenes közé esik*, akkor írja ki azt a két egyenest,
- ha a pont *az összes egyenes alatt vagy felett van*, akkor ezt is jelezze. [1 pont]

Lehetséges C/C++ függvényfejléc:

```
void holVanAPont(int n, double a[], double b[], double x, double y)
```

/* A kulcs az, hogy az egyenesek minden $x \in [0,1]$-re rendezettek, tehát adott x-re az $y_i(x)=a_ix+b_i$ értékek szigorúan növekvő sorozatot adnak. Ez azt jelenti, hogy az adott ponthoz tartozó y értéket bináris kereséssel tudjuk elhelyezni az egyenesek között. */

```
void holVanAPont(int n, double a[], double b[], double x, double y) {
   const double EPS = 1e-9;
   int left = 0, right = n - 1;
   // Az összes alatt vagy felett
   double y0 = a[0] * x + b[0];
   double yn = a[n - 1] * x + b[n - 1];
   if (y < y0 - EPS) {
      cout << "A pont az osszes egyenes alatt van.\n";
      return;
   }
   if (y > yn + EPS) {
      cout << "A pont az osszes egyenes felett van.\n";
      return;
   }
   // Bináris keresés
   while (left <= right) {
      int mid = (left + right) / 2;
      double ymid = a[mid] * x + b[mid];
      if (fabs(ymid - y) < EPS) {
         cout << "A pont a(z) " << mid + 1 << ". egyenesen van.\n";
         return;
      }
      if (ymid < y)
         left = mid + 1;
      else
         right = mid - 1;
   }
   // Ha nem találtuk meg pontosan, akkor két egyenes között van
   cout << "A pont a(z) " << right + 1 << ". es a(z) "
        << left + 1 << ". egyenes kozott van.\n";
```

}

9. A távoli Eldoria Királyságában különleges szabályok szerint határozzák meg a trónöröklés rendjét. Ismert a királyi család élő tagjainak családfája, amely egyetlen összefüggő fát alkot (mindenki egyetlen közös ősre vezethető vissza). Ez alapján a család minden tagjához egy trónörökös-súlyt rendelnek; a legkisebb súlyú személy lesz a trón várományosa.

A súly kiszámítása az alábbi szabályok szerint történik:

- Királyi viszonyítás: A király súlya `0`. Minden szint a családfában (felfelé az ősök, lefelé az utódok irányába) `1`-gyel növeli a súlyt. Más szóval: valamely személy súlyának kezdeti értéke a királytól mért távolsága a fában.
- Születési sorrend: A gyermekek születési sorszáma hozzáadódik a súlyhoz (első gyerek `+1`, második `+2`, stb.). A családfa egyetlen közös ősénél (a gyökérnél, akinek nincs szülője a rendszerben) ez a sorszám 0.
- Nemi szorzó: A leánygyermekek végső súlyát megduplázzuk.

A családfát szülők tömbjével ábrázoljuk (indexek: `1…N`). Az `ID` megegyezik az indexszel. A bemenet garantálja, hogy az `ID`-k születési sorrendben vannak ($ID_A < ID_B$, ha `A` idősebb testvér).

Holtverseny feloldása: Amennyiben több jelöltnek azonos a súlya, a fiatalabb tag (a nagyobb `ID`-val rendelkező) válik a trónörökössé.

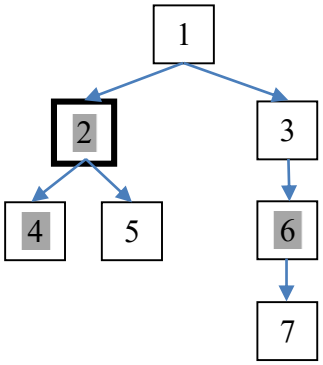
Írj Pascal vagy C/C++ *programot*, amely a `be.txt` állományból beolvassa az `N` értékét (`1<N<100`), a király `ID`-ját, valamint két `N`-elemű számsorozatot, a szülők és a nemek (`1`-férfi, `2`-nő) tömbjét, és kiírja a képernyőre a trónörökös `ID`-ját. [1 pont]

Példa bemenet (`be.txt`):

```
7 2
-1 1 1 2 2 3 6
1 2 1 2 1 2 1
```

Kimenet:

```
1
```



Magyarázat: a példában `7`-tagú a királyi család, a `2`-es `ID`-jű a király(nő), a szülők tömbje az ábra szerinti hierarchiát kódolja (a közös ősnek megfelelő érték `-1`, mert neki nincs szülője a megadott családfa részletben). A családtagok trónörökös-súlya: `1`:**1**, `2`:**0**, `3`:**4**, `4`:**4**, `5`:**3**, `6`:**8**, `7`:**5**. A legkisebb súlyértékű várományosa a trónnak az `1`-es `ID`-jű, azaz a királynő apja a trónörökös.

```c
#include <stdio.h>
#define MAX_N 105

int main() {
    FILE *f = fopen("be.txt", "r");
    int n, kiraly;
    int szulok[MAX_N], nem[MAX_N];
    int tav[MAX_N];
    int sorszam[MAX_N] = {0};
    int gyerek_szamlalo[MAX_N] = {0};
    fscanf(f, "%d %d", &n, &kiraly);
    for (int i = 1; i <= n; i++) {
        fscanf(f, "%d", &szulok[i]);
    }
    for (int i = 1; i <= n; i++) {
        fscanf(f, "%d", &nem[i]);
    }
    fclose(f);
    /* 1. Gyerekek sorszámának meghatározása */
    for (int i = 1; i <= n; i++) {
        int sz = szulok[i];
        if (sz != -1) {
            gyerek_szamlalo[sz]++;
            sorszam[i] = gyerek_szamlalo[sz];
        }
    }
    /* 2. Távolságok inicializálása */
```

```
    for (int i = 1; i <= n; i++) {
        tav[i] = -1;
    }
    tav[kiraly] = 0;
    /* 3. Távolságok számítása ismételt terjesztéssel */
    int valtozott = 1;
    while (valtozott) {
        valtozott = 0;
        for (int i = 1; i <= n; i++) {
            int a = i;            /* gyerek */
            int b = szulok[i];    /* szülő */
            if (b == -1) continue;
            if (tav[a] != -1 && tav[b] == -1) {
                tav[b] = tav[a] + 1;
                valtozott = 1;
            }
            else if (tav[b] != -1 && tav[a] == -1) {
                tav[a] = tav[b] + 1;
                valtozott = 1;
            }
        }
    }
    /* 4. Trónörökös meghatározása */
    int legjobb_id = -1;
    int legjobb_suly = 1000000;
    for (int i = 1; i <= n; i++) {
        if (i == kiraly) continue;    /* a királyt nem vizsgáljuk */
        int suly = tav[i] + sorszam[i];
        if (nem[i] == 2) {
            suly *= 2;
        }
        if (suly < legjobb_suly || (suly == legjobb_suly && i > legjobb_id)) {
            legjobb_suly = suly;
            legjobb_id = i;
        }
    }
    printf("%d\n", legjobb_id);
    return 0;
}
```

Alább megadunk egy C++ kódot, amely egy lineáris bonyolultságú algoritmust implementál

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, kiraly;
    cin >> n >> kiraly;
    vector<int> szulok(n + 1);
    vector<int> nem(n + 1);
    for (int i = 1; i <= n; i++) {
        cin >> szulok[i];
    }
    for (int i = 1; i <= n; i++) {
        cin >> nem[i];
    }
    // sorszam[i] = i-edik személy születési sorszáma a saját testvérei között
    // gyerek_szamlalo[p] = eddig hány gyerekét láttuk p szülőnek
    vector<int> sorszam(n + 1, 0);
    vector<int> gyerek_szamlalo(n + 1, 0);
```

```
    // A feladat szerint az ID-k születési sorrendben vannak,
    // ezért egy balról jobbra bejárás elegendő a sorszámok meghatározásához.
    for (int i = 1; i <= n; i++) {
        int sz = szulok[i];
        if (sz != -1) {
            gyerek_szamlalo[sz]++;
            sorszam[i] = gyerek_szamlalo[sz];
        }
    }
    // Szomszédsági lista az irányítatlan fához:
    // minden él összeköti a gyereket a szülőjével
    vector<vector<int>> szomszedok(n + 1);
    for (int i = 1; i <= n; i++) {
        int sz = szulok[i];
        if (sz != -1) {
            szomszedok[i].push_back(sz);
            szomszedok[sz].push_back(i);
        }
    }
    // tav[i] = i-edik személy távolsága a királytól
    vector<int> tav(n + 1, -1);
    // BFS a királyból, így minden távolság O(n)-ben kiszámolható
    queue<int> q;
    tav[kiraly] = 0;
    q.push(kiraly);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int v : szomszedok[u]) {
            if (tav[v] == -1) {
                tav[v] = tav[u] + 1;
                q.push(v);
            }
        }
    }
    // A legjobb trónörökös kiválasztása
    int legjobb_id = -1;
    int legjobb_suly = 1000000000;
    for (int i = 1; i <= n; i++) {
        if (i == kiraly) continue;  // a királyt nem vizsgáljuk
        int suly = tav[i] + sorszam[i];
        // Nő esetén a végső súlyt duplázzuk
        if (nem[i] == 2) {
            suly *= 2;
        }
        // Kisebb súly a jobb;
        // holtversenyben a fiatalabb (nagyobb ID) nyer
        if (suly < legjobb_suly || (suly == legjobb_suly && i > legjobb_id)) {
            legjobb_suly = suly;
            legjobb_id = i;
        }
    }
    cout << legjobb_id << '\n';
    return 0;
}
```