

CUV. 2018

Concurs admitere - varianta 1
Proba scrisă la Informatică

În atenția concurenților:

- Se consideră că indexarea tuturor șirurilor începe de la 1.
- Problemele tip grilă (Partea A) pot avea unul sau mai multe răspunsuri corecte. Răspunsurile trebuie scrise de candidat pe foaia de concurs (nu pe foaia cu enunțuri). Obținerea punctajului aferent problemei este condiționată de identificarea tuturor variantelor de răspuns corecte și numai a acestora.
- Pentru problemele din Partea B se cer rezolvări complete pe foaia de concurs.
 - Rezolvările se vor scrie în *pseudocod* sau *într-un limbaj de programare (Pascal/C/C++)*.
 - Primul criteriu în evaluarea rezolvărilor va fi *corectitudinea* algoritmului, iar apoi *performanța* din punct de vedere al *timpului de execuție* și al *spațiului de memorie utilizat*.
 - Este obligatorie descrierea și justificarea* (sub) algoritmilor înaintea rezolvărilor. Se vor scrie, de asemenea, *comentarii* pentru a ușura înțelegerea detaliilor tehnice ale soluției date, a semnificației identificărilor, a structurilor de date folosite etc. Nedeplinirea acestor cerințe duce la pierderea a 10% din punctajul aferent subiectului.
- Nu se vor folosi funcții sau biblioteci predefinite (de exemplu, STL, funcții predefinite pe șiruri de caractere).

Partea A (30 puncte)

A.1. Oare ce face? (5 puncte)

Se consideră subalgoritmul expresie(n), unde n este un număr natural ($1 \leq n \leq 10000$).

Subalgoritm expresie(n):
Dacă $n > 0$ atunci
 returnează $-n * (n + 1) + expresie(n - 1)$
altfel
 returnează $n * (n + 1) + expresie(n - 1)$
SfDacă
altfel
 returnează 0
SfSubalgoritm

A. 3:0
B. 2:0
C. 0:2
D. 2:2
E. Niciun răspuns nu este corect

Precizați forma matematică a expresiei $E(n)$ calculată de acest subalgoritm:

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

A.2. Calcul (5 puncte)

Se consideră subalgoritmul calcul(n) unde n este un număr natural ($1 \leq n \leq 10000$).

Subalgoritm calcul(n):
 $x \leftarrow 0$; $z \leftarrow 1$
Cât timp $z \leq n$ execută
 $x \leftarrow x + 1$
 $z \leftarrow z + 2 * x$
SfCât timp
 returnează x
SfSubalgoritm

A. Dacă $n = 25$ sau $n = 35$, atunci calcul(n) returnează 5.
B. Dacă $n < 8$, atunci calcul(n) returnează 3.
C. Dacă $n \geq 85$ și $n < 100$, atunci calcul(n) returnează 9.
D. Subalgoritmul calculează și returnează numărul pătratelor perfecte strict pozitive și strict mai mici decât n.
E. Subalgoritmul calculează și returnează partea întreagă a radicalului numărului n.

Care dintre afirmațiile de mai jos sunt false?

- ☒ A. Dacă $n = 25$ sau $n = 35$, atunci calcul(n) returnează 5.
☒ B. Dacă $n < 8$, atunci calcul(n) returnează 3.
☒ C. Dacă $n \geq 85$ și $n < 100$, atunci calcul(n) returnează 9.
☐ D. Subalgoritmul calculează și returnează numărul pătratelor perfecte strict pozitive și strict mai mici decât n.
☐ E. Subalgoritmul calculează și returnează partea întreagă a radicalului numărului n.

A.3. Expresie logică (5 puncte)

Se consideră următoarea expresie logică: (NOT Y OR Z) OR (X AND Y). Alegeți valorile pentru X, Y, Z astfel încât rezultatul evaluării expresiei să fie adevărat.

- ☐ A. $X = \text{fals}$; $Y = \text{fals}$; $Z = \text{fals}$;
☒ B. $X = \text{fals}$; $Y = \text{fals}$; $Z = \text{adevărat}$;
☐ C. $X = \text{fals}$; $Y = \text{adevărat}$; $Z = \text{fals}$;
☐ D. $X = \text{adevărat}$; $Y = \text{fals}$; $Z = \text{adevărat}$;
☒ E. $X = \text{fals}$; $Y = \text{adevărat}$; $Z = \text{adevărat}$;

A.4. Ce se afișează? (5 puncte)

Se consideră următorul program:

Variantă C++	Variantă Pascal
<pre>#include <iostream> using namespace std; int sum(int n, int a[], int s){ s = 0; 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, 0); cout << s << '\n'; return 0; }</pre>	<pre>type vector=array[1..10] of integer; function sum(n:integer; a:vector; var s:integer):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; begin n := 3; a[1] := -1; a[2] := 0; a[3] := 3; p := 0; s := sum(n, a, p); write(s, '\n'); end.</pre>

Care este rezultatul afișat în urma execuției programului?

- ☐ A. 3:0
☒ B. 2:0
☐ C. 0:2
☐ D. 2:2
☐ E. Niciun răspuns nu este corect

A.5. Număr norocos (5 puncte)

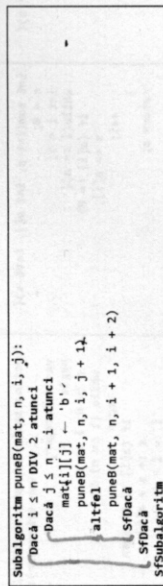
Un număr natural nenul x se numește *norocos* dacă pătratul său se poate scrie ca sumă de x numere naturale consecutive. De exemplu, 7 este număr norocos pentru că $7^2 = 4 + 5 + 6 + 7 + 8 + 9 + 10$.

Care dintre următorii subalgoritmi verifică dacă un număr natural x ($2 \leq x \leq 1000$) este *norocos*? Fiecare subalgoritm are ca parametru de intrare numărul x, iar ca parametri de ieșire numărul nenumit *start* și variabila de tip boolean *esteNorocos*. Dacă numărul x este norocos, atunci *esteNorocos* = *adevărat* și *start* va reține primul termen din sumă (de ex., dacă x = 7, atunci *start* = 4); dacă x nu este norocos, atunci *esteNorocos* = *fals* și *start* va reține valoarea -1.

<p>A. Subalgoritm norocos(x, start, esteNorocos): xpatrat := x * x estelnorocos := fals start := -1; k := 1; Cât timp k ≤ xpatrat - x și nu esteNorocos execută s := s + i Pentru i ← k, k + x - 1 execută s := s + i SfPentru Dacă s = xpatrat atunci estelnorocos := adevărat start := k SfDacă SfCât timp SfSubalgoritm</p>	<p>B. Subalgoritm norocos(x, start, esteNorocos): xpatrat := x * x estelnorocos := fals Cât timp k ≤ xpatrat - x și nu esteNorocos execută s := s + i Pentru i ← k, k + x - 1 execută s := s + i SfPentru Dacă s = xpatrat atunci estelnorocos := adevărat start := k SfDacă SfCât timp SfSubalgoritm</p>
<p>C. Subalgoritm norocos(x, start, esteNorocos): Dacă x MOD 2 = 0 atunci estelnorocos := fals altfel start := -1 estelnorocos := adevărat start := (x + 1) DIV 2 SfDacă SfSubalgoritm</p>	<p>D. Subalgoritm norocos(x, start, esteNorocos): Dacă x MOD 2 = 0 atunci estelnorocos := fals altfel start := -1 estelnorocos := adevărat start := x DIV 2 SfDacă SfSubalgoritm</p>
<p>E. Subalgoritm norocos(x, start, esteNorocos): estelnorocos := adevărat start := (x + 1) DIV 2 SfSubalgoritm</p>	<p>F. Subalgoritm norocos(x, start, esteNorocos): estelnorocos := adevărat start := x DIV 2 SfSubalgoritm</p>

A.6. Pune 'b' (5 puncte)

Se consideră o matrice pătratică mat de dimensiune $n \times n$ (n - număr natural impar, $3 \leq n \leq 100$) și subalgoritmul $puneB(mat, n, i, j)$ care pune caracterul 'b' pe anumite poziții în matricea mat . Parametrii i și j sunt numere naturale ($1 \leq i \leq n, 1 \leq j \leq n$).



Precizați de câte ori se autoapelează subalgoritmul $puneB(mat, n, i, j)$ dacă avem secvența de instrucțiuni:

```

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

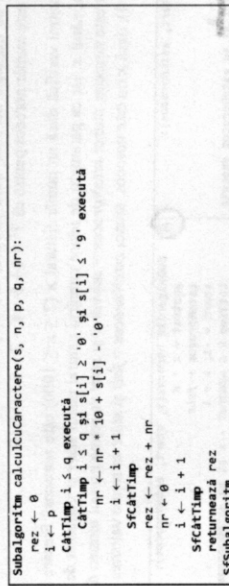
```

- A. de 5 ori
 B. de același număr de ori ca și în cazul secvenței de instrucțiuni
 $n \leftarrow 9, i \leftarrow 3, j \leftarrow 5$
 $puneB(mat, n, i, j)$
 C. de 10 ori
 D. de 0 ori
 E. de o infinitate de ori

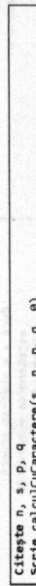
Partea B (60 puncte)

B.1. Calcul cu caractere (10 puncte)

Se consideră subalgoritmul $calculCuCaractere(s, n, p, q, nr)$, unde s este un șir cu n caractere (n este număr natural, $1 \leq n \leq 9$), iar p, q și nr sunt numere naturale ($1 \leq p \leq n, 1 \leq q \leq n, p \leq q$).



Scrieți o variantă recursivă a subalgoritmului $calculCuCaractere(s, n, p, q, nr)$ care are același antet cu acesta și care are același efect în secvența de instrucțiuni:



B.2. Perioadă (25 puncte)

Se spune că un șir de n caractere are *perioadă* k dacă șirul respectiv poate fi format prin concatenarea repetată a unui șir de caractere de lungime k ($2 \leq n \leq 200, 1 \leq k \leq n$). Șirul "abcabcabcabc" are *perioadă* 3 deoarece poate fi considerat ca 4 apariții concatenate ale șirului "abc"; el are de asemenea *perioadă* 6 dacă remarcăm că este format din 2 apariții concatenate ale șirului "abcabc". Șirul "abcabc" nu are perioadă. Se numește *perioadă maximă* cea mai mare *perioadă* a unui șir.

Scrieți un subalgoritm care determină *perioada maximă* pm a unui șir de caractere x cu n elemente (n - număr natural, $2 \leq n \leq 200$). Dacă șirul x nu are perioadă, pm va avea valoarea -1. Parametrii de intrare ai subalgoritmului sunt x și n , iar parametrul de ieșire este pm .

Exemplu 1: dacă $n = 8$ și $x = "abababab"$, atunci $pm = 4$.

Exemplu 2: dacă $n = 7$ și $x = "ababcabc"$, atunci $pm = -1$.

B.3. Robi-grădina (25 puncte)

Un grădinar pasionat de tehnologie decide să folosească o "armată" de roboți pentru a uda straturile din grădina sa. El dorește să folosească apă de la izvorul situat la capătul cărării principale care străbate grădina. Fiecărui strat îi este asociat un robot și fiecare robot are de dat un singur strat. Toți roboții pornesc de la izvor în misunea de udare a straturilor la aceeași oră a dimineții (spre exemplu 5:00:00) și lucrează în paralel și neîntrerupt un același interval de timp. Ei parcurg cărarea principală până la stratul lor, pe care îl udă și revin la izvor pentru a-și reumple rezervorul de apă. La finalul intervalului de timp aferent activității, toți roboții se opresc simultan, indiferent de starea lor curentă. Inițial, la izvor este amplasat un singur robinet. Grădinarul constată însă că apar înălzieri în programul de udare a plantelor deoarece roboții trebuie să aștepte la rând pentru reumplerea rezervoarelor cu apă, astfel încât consideră că cea mai bună soluție este să instaleze mai multe robinete pentru alimentarea roboților. Roboții pornesc dimineața cu rezervoarele umplute. Doi roboți nu își pot umple rezervorul în același moment de la același robinet.

Se cunosc: intervalul de timp t cât cei n roboți lucrează (exprimat în secunde), numărul de secunde d_i necesare pentru a parcurge distanța de la izvor la stratul asociat, numărul de secunde u_i necesar pentru udarea acestui strat și faptul că umplerea rezervorului propriu cu apă durează exact o secundă pentru fiecare robot (t, n, d_i, u_i - numere naturale, $1 \leq t \leq 20000, 1 \leq n \leq 100, 1 \leq d_i \leq 1000, 1 \leq u_i \leq 1000, i = 1, 2, \dots, n$).

Cerințe

- Enumerați roboții care se întâlnesc la izvor la un anumit moment de timp mt ($1 \leq mt \leq t$). Justificați răspunsul.
 Notă: roboții se identifică prin numărul lor de ordine.
- Care este numărul minim de robinete suplimentare $minRobineteSuplim$ pe care trebuie să le instaleze grădinarul astfel încât roboții să nu aștepte deloc unul după altul pentru reumplerea rezervorului? Justificați răspunsul.
- Scrieți un subalgoritm care determină numărul minim de robinete suplimentare $minRobineteSuplim$. Parametrii de intrare sunt numerele n și t , și șirurile d și u cu câte n elemente fiecare, iar parametrul de ieșire este $minRobineteSuplim$.

Exemplu 1: dacă $t = 32, n = 5, d = (1, 2, 1, 2, 1), u = (1, 3, 2, 1, 3)$ atunci $minRobineteSuplim = 3$. Explicație: robotul care se ocupă de stratul 1 are nevoie de o secundă pentru a ajunge la strat, o secundă pentru a uda stratul și de încă o secundă pentru a se întoarce la izvor; el se întoarce la izvor pentru a-și reumple rezervorul după $1 + 1 + 1 = 3$ secunde de la ora de plecare (5:00:00), deci la ora 5:00:03; el își reumple rezervorul într-o secundă și pornește înapoi spre strat la ora 5:00:04; revine la ora 5:00:07 pentru a-și reumple rezervorul, continuând ritmul de udare a straturilor, ș.a.m.d.; deci, primul, al doilea, al patrulea și al cincilea robot se întâlnesc la izvor la ora 05:00:23; în consecință, este nevoie de 3 robinete suplimentare.

Exemplu 2: dacă $t = 37, n = 3, d = (1, 2, 1), u = (1, 3, 2)$, atunci $minRobineteSuplim = 1$.

Notă:

- Toate subiectele sunt obligatorii.
- Ciornele nu se iau în considerare.
- Se acordă 10 puncte din oficiu.
- Timpul efectiv de lucru este de 3 ore.

BAREM	
OFICIU	
Partea A	10 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 ¹	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 nr + calculCuCaractere(s, n, p + 1, q, 10 * nr + s[p] - '0')
    Sfîrșit
Sfîrșit

```

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 grădina	25 puncte
a. la un anumit moment de timp mt ($1 \leq mt \leq \ell$) se întâlnesc la izvor roboți 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) divizor al lui mt	5 puncte
b. numărul minim de robinete suplimentare este egal cu maximul vectorului $aux - 1$, unde vectorul aux 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 ($q = 2 * \text{deplasare} + \text{udare} + \text{încărcare}$)	2 puncte
c. prelucrarea vectorului de frecvențe	5 puncte
d. stabilirea frecvenței maximele	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 ($q = 2 * \text{deplasare} + \text{udare} + \text{încărcare}$)	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 puncte
f. stabilirea numărului maxim de robinete	1 puncte
g. determinarea numărului de robinete suplimentare	2 puncte

¹ În varianta subiectului în limba engleză, datorită traducerii ambigue a termenului *auto-apel*, a fost considerată corectă atât varianta cu răspunsurile A și B, cât și varianta cu răspunsul B.

Partea B (soluții)	60 puncte
B. 1. Calcul	10 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 nr + calculCuCaractere(s, n, p + 1, q, 10 * nr + s[p] - '0')
    Sfîrșit
Sfîrșit

```

B. 2. Perioadă maximă	25 puncte
------------------------------	------------------

```

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

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

```

B. 3. Robi grădina	25 puncte
---------------------------	------------------

```

int robinetGradina(int n, int d[], int d[], int t){
    int aux[200000]; //aux[i] va reține câte 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 + d[j] + 1; //se marchează multiplii lui q în vectorul aux
        for (int i = q; i <= t; i = i + q){
            aux[i]++;
            if (max < aux[i])
                max = aux[i];
        }
    }
    return max - 1;
}

```