UNIVERSITATEA BABEŞ-BOLYAI
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

Model de subiect pentru Concursul Mate-Info UBB si Admiterea la facultate 2021

Proba scrisă la Informatică

1. Subalgoritmul `generare(n)` prelucrează un număr natural ***n*** (0 < ***n*** < 100).

```
Subalgoritm generare(n):
   nr ← 0
   Pentru i ← 1, 1801 execută
      folosit_i ← fals
   SfPentru
   CâtTimp nu folosit_n execută
      suma ← 0, folosit_n ← adevărat
      CâtTimp (n ≠ 0) execută
         cifra ← n MOD 10, n ← n DIV 10
         suma ← suma + cifra * cifra * cifra
      SfCâtTimp
      n ← suma, nr ← nr + 1
   SfCâtTimp
   returnează nr
SfSubalgoritm
```

Precizați care este efectul acestui subalgoritm.

A. Calculează, în mod repetat, suma cuburilor cifrelor numărului ***n*** până când suma egalează numărul ***n*** și returnează numărul repetărilor efectuate
B. Calculează suma cuburilor cifrelor numărului ***n*** și returnează această sumă
C. Calculează suma cuburilor cifrelor numărului ***n***, înlocuiește numărul ***n*** cu suma obținută și returnează această sumă
D. Calculează numărul înlocuirilor lui ***n*** cu suma cuburilor cifrelor sale până când se obține o valoare calculată anterior sau numărul însuși și returnează acest număr

2. Fie ***s*** un şir cu ***k*** elemente de tip boolean și subalgoritmul `evaluare(s, k, i)`, unde ***k*** și ***i*** sunt numere naturale (0 ≤ ***i*** ≤ ***k*** ≤ 100).

```
Subalgoritm evaluare(s, k, i)
   Dacă i ≤ k atunci
      Dacă s_i atunci
         returnează s_i
      altfel
         returnează (s_i sau evaluare(s, k, i + 1))
      SfDacă
   altfel
      returnează fals
   SfDacă
SfSubalgoritm
```

Precizați de câte ori se autoapelează subalgoritmul `evaluare(s, k, i)` în urma execuției următoarei secvențe de instrucțiuni:

```
s ← (fals, fals, fals, fals, fals, fals, adevărat, fals, fals, fals)
k ← 10, i ← 3
evaluare(s, k, i)
```

A. de 3 ori
B. de același număr de ori ca în următoarea secvență de instrucțiuni:

```
s ← (fals, fals, fals, fals, fals, fals, fals, adevărat)
k ← 8, i ← 4
evaluare(s, k, i)
```

C. de 6 ori
D. Niciodată

3. Se consideră subalgoritmul `expresie(n)`, unde ***n*** este un număr natural ($1 \le n \le 10000$).

```
Subalgoritm expresie(n):
   Dacă n > 0 atunci
      Dacă n MOD 2 = 0 atunci
         returnează -n * (n + 1) + expresie(n - 1)
      altfel
         returnează n * (n + 1) + expresie(n - 1)
      SfDacă
   altfel
      returnează 0
   SfDacă
SfSubalgoritm
```

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

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

4. Un tip de date întreg reprezentat pe ***x*** biți (***x*** este număr natural strict pozitiv) va putea reține valori întregi din:

A. $[0, 2^x]$
B. $[0, 2^{x-1}-1]$
C. $[-2^{x-1}, 2^{x-1}-1]$
D. $[-2^x, 2^x-1]$

5. Se dă subalgoritmul `f(a, b)`:

```
Subalgoritm f(a, b):
       Dacă a > 1 atunci
              returnează b * f(a - 1, b)
       altfel
              returnează b * f(a + 1, b)
       SfDacă
SfSubalgoritm
```

Precizați de câte ori se auto apelează subalgoritmul `f(a, b)` în urma execuției următoarei secvențe de instrucțiuni:

```
a ← 4, b ← 3
c ← f(a, b)
```

A. de 4 ori
B. de 3 ori
C. de o infinitate de ori
D. niciodată

6. 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*** ← *fals*; ***Y*** ← *fals*; ***Z*** ← *fals*;
B. ***X*** ← *fals*; ***Y*** ← *adevărat*; ***Z*** ← *fals*;
C. ***X*** ← *adevărat*; ***Y*** ← *fals*; ***Z*** ← *adevărat*;
D. ***X*** ← *fals*; ***Y*** ← *adevărat*; ***Z*** ← *adevărat*;

7. Precizați care dintre următoarele expresii are valoarea adevărat dacă și numai dacă numărul natural ***n*** este divizibil cu 3 și are ultima cifră 4 sau 6:

A. `n DIV 3 = 0 și (n MOD 10 = 4 sau n MOD 10 = 6)`
B. `n MOD 3 = 0 și (n MOD 10 = 4 sau n MOD 10 = 6)`
C. `(n MOD 3 = 0 și n MOD 10 = 4) sau (n MOD 3 = 0 și n MOD 10 = 6)`
D. `(n MOD 3 = 0 și n MOD 10 = 4) sau n MOD 10 = 6`

8. Fie următorul subalgoritm:

```
Subalgoritm f(a):
    Dacă a ≠ 0 atunci
        returnează a + f(a - 1)
    altfel
        returnează 0
    SfDacă
SfSubalgoritm
```

Care din afirmatiile de mai jos sunt false?

A. daca ***a*** este negativ, subalgoritmul întoarce 0
B. valoarea calculată de ***f*** este ***a*** * (***a*** + 1) / 4
C. subalgoritmul calculează suma numerelor naturale mai mici sau egale cu ***a***
D. apelul ***f***(-5) intră în ciclu infinit.

9. Se consideră următorul subalgoritm:

```
Subalgoritm SA9(a):
        Dacă a < 50 atunci
                Dacă a MOD 3 = 0 atunci
                        returnează SA9(2 * a - 3)
                altfel
                        returnează SA9(2 * a - 1)
                SfDacă
        altfel
                returnează a
        SfDacă
SfSubalgoritm
```

Pentru care dintre valorile parametrului de intrare ***a*** subalgoritmul va returna valoarea 61?

A. 16
B. 61
C. 4
D. 31

10. Se consideră subalgoritmul `prelucreaza(v, k)`, unde ***v*** este un șir cu ***k*** numere naturale (1 ≤ ***k*** ≤ 1 000).

```
Subalgoritm prelucreaza(v, k)
   i ← 1, n ← 0
   CâtTimp i ≤ k și vᵢ ≠ 0 execută
      y ← vᵢ, c ← 0
      CâtTimp y > 0 execută
         Dacă y MOD 10 > c atunci
            c ← y MOD 10
         SfDacă
         y ← y DIV 10
      SfCâtTimp
      n ← n * 10 + c
      i ← i + 1
   SfCâtTimp
   returnează n
SfSubalgoritm
```

Precizați pentru care valori ale lui ***v*** și **k** subalgoritmul returnează valoarea 928.

A. ***v*** = (194, 121, 782, 0) și ***k*** = 4
B. ***v*** = (928) și ***k*** = 1
C. ***v*** = (9, 2, 8, 0) și ***k*** = 4
D. ***v*** = (8, 2, 9) și ***k*** = 3

11. Se consideră următoarea expresie logică `(X OR Z) AND (NOT X OR Y)`. Alegeți valorile pentru ***X***, ***Y***, ***Z*** astfel încât evaluarea expresiei să dea rezultatul `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;

12. Se consideră următorul program:

Varianta C

```c
#include <stdio.h>

int prelVector(int v[], int *n) {
   int s = 0; int i = 2;
   while (i <= *n) {
      s = s + v[i] - v[i - 1];
      if (v[i] == v[i - 1])
         *n = *n - 1;
      i++;
   }
   return s;
}


int main(){
  int v[8];
  v[1] = 1; v[2] = 4; v[3] = 2;
  v[4] = 3; v[5] = 3; v[6] = 10;
  v[7] = 12;
  int n = 7;
  int rezultat = prelVector(v,
&n);
  printf("%d;%d", n, rezultat);
  return 0;
}
```

Varianta C++

```cpp
#include <iostream>
using namespace std;
int prelVector(int v[], int&n) {
   int s = 0; int i = 2;
   while (i <= n) {
      s = s + v[i] - v[i - 1];
      if (v[i] == v[i - 1])
         n--;
      i++;
   }
   return s;
}


int main(){
   int v[8];
   v[1] = 1; v[2] = 4; v[3] = 2;
   v[4] = 3; v[5] = 3; v[6] = 10;
   v[7] = 12;
   int n = 7;
   int rezultat = prelVector(v, n);
   cout << n <<";" << rezultat;
   return 0;
}
```

Varianta Pascal

```pascal
type vector=array [1..10] of integer;
function prelVector(v: vector;
                var n: integer): integer;
   var s, i: integer;
   begin
      s := 0; i := 2;
      while (i <= n) do
       begin
        s := s + v[i] - v[i - 1];
        if (v[i] = v[i - 1]) then
           n := n - 1;
        i := i + 1;
      end;
      prelVector := s;
   end;
var n, rezultat:integer; v:vector;
begin
   n := 7;
   v[1] := 1; v[2] := 4; v[3] := 2;
   v[4] := 3; v[5] := 3; v[6] := 10;
   v[7] := 12;
   rezultat := prelVector(v,n);
   write(n, ';', rezultat);
end.
```

Precizați care este rezultatul afișat în urma executării programului.

A. 7;11
B. 6;9
C. 7;9
D. 7;12

13. Se consideră următorul algoritm în pseudocod:

```
citește a
Pentru i=1, a-1 execută
       Pentru j=i+2, a execută
              Dacă i+j>a-1 atunci
                     scrie a, ‘ ‘, i, ‘ ‘, j
                     trecere la rând nou
              SfDacă
       SfPentru
SfPentru
```

Câte perechi de soluții se vor afișa în urma execuției algoritmului pentru a=8?

A. 13
B. 15
C. 20
D. nici un răspuns nu e corect

14. Care dintre subalgoritmii de mai jos returnează cel mai mare multiplu al numărului natural ***a***, multiplu care este mai mic sau egal cu numărul natural ***b*** (0 < ***a*** < 10 000, 0 < ***b*** < 10 000, ***a*** < ***b***)?

A.
```
Subalgoritm f(a, b):
  c ← b
  CâtTimp c MOD a = 0 execută
    c ← c – 1
  SfCâtTimp
  returnează c
SfSubalgoritm
```

B.
```
Subalgoritm f(a, b):
  Dacă a < b atunci
    returnează f(2 * a, b)
  altfel
    Dacă a = b atunci
      returnează a
    altfel
      returnează b
    SfDacă
  SfDacă
SfSubalgoritm
```

C.
```
Subalgoritm f(a, b):
  returnează (b DIV a) * a
SfSubalgoritm
```

D.
```
Subalgoritm f(a, b):
  Dacă b MOD a = 0 atunci
    returnează b
  SfDacă
  returnează f(a, b - 1)
SfSubalgoritm
```

15. Se consideră toate șirurile de lungime ***l*** ϵ {1, 2, 3} formate din litere din mulțimea {*a*, *b*, *c*, *d*, *e*}. Câte dintre aceste șiruri au elementele ordonate strict descrescător și un număr impar de vocale? (*a* și *e* sunt vocale)

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

16. Se consideră dat subalgoritmul `aparține(x, a, n)` care verifică dacă un număr natural ***x*** aparține mulțimii ***a*** cu ***n*** elemente; ***a*** este un șir cu ***n*** elemente și reprezintă o mulțime de numere naturale (1 ≤ ***n*** ≤ 200, 1 ≤ ***x*** ≤ 1000).
Fie subalgoritmii `reuniune(a, n, b, m, c, p)` și `calcul(a, n, b, m, c, p)`, descriși mai jos, unde ***a***, ***b*** și ***c*** sunt șiruri care reprezintă mulțimi de numere naturale cu ***n***, ***m*** și respectiv ***p*** elemente (1 ≤ ***n*** ≤ 200, 1 ≤ ***m*** ≤ 200, 1 ≤ ***p*** ≤ 400). Parametrii de intrare sunt ***a***, ***n***, ***b***, ***m*** și ***p***, iar parametrii de ieșire sunt ***c*** și ***p***.

```
1.     Subalgoritm reuniune(a, n, b, m, c, p):
2.       Dacă n = 0 atunci
3.         Pentru i ← 1, m execută
4.           p ← p + 1, c_p ← b_i
5.         SfPentru
6.       altfel
7.         Dacă nu aparține(a_n, b, m) atunci
8.           p ← p + 1, c_p ← a_n
9.         SfDacă
10.        reuniune(a, n - 1, b, m, c, p)
11.      SfDacă
12.    SfSubalgoritm
```

```
Subalgoritm calcul(a, n, b, m, c, p):
     p ← 0
     reuniune(a, n, b, m, c, p)
SfSubalgoritm
```

Precizați care dintre afirmațiile de mai jos sunt întotdeauna adevărate:

A. când mulțimea ***a*** conține un singur element, apelul subalgoritmului `calcul(a, n, b, m, c, p)` provoacă apariția unui ciclu infinit
B. când mulțimea ***a*** conține 4 elemente, apelul subalgoritmului `calcul(a, n, b, m, c, p)` provoacă executarea instrucțiunii de pe linia 10 a subalgoritmului `reuniune` de 4 ori
C. când mulțimea ***a*** conține 5 elemente, apelul subalgoritmului `calcul(a, n, b, m, c, p)` provoacă executarea instrucțiunii de pe linia 2 a subalgoritmului `reuniune` de 5 ori
D. când mulțimea ***a*** are aceleași elemente ca și mulțimea ***b***, în urma execuției subalgoritmului `calcul(a, n, b, m, c, p)` mulțimea ***c*** va avea același număr de elemente ca și mulțimea ***a***

17. Se consideră subalgoritmul `calcul(n)` unde ***n*** este un număr natural ($1 \le n \le 10000$).

```
Subalgoritm calcul(n):
    x ← 0, z ← 1
    CâtTimp z ≤ n execută
        x ← x + 1
        z ← z + 2 * x
        z ← z + 1
    SfCâtTimp
    returnează x
SfSubalgoritm
```

Care dintre afirmațiile de mai jos sunt **false**?

A. Dacă ***n*** < 8, atunci `calcul(n)` returnează 3.
B. Dacă ***n*** ≥ 85 și ***n*** < 100, atunci `calcul(n)` returnează 9.
C. Subalgoritmul calculează și returnează numărul pătratelor perfecte strict pozitive și strict mai mici decât ***n***.
D. Subalgoritmul calculează și returnează partea întreagă a radicalului numărului ***n***.

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

```
Subalgoritm puneB(mat, n, i, j):
   Dacă i ≤ n DIV 2 atunci
      Dacă j ≤ n - i atunci
         mat[i][j] ← 'b'
         puneB(mat, n, i, j + 1)
      altfel
         puneB(mat, n, i + 1, i + 2)
      SfDacă
   SfDacă
SfSubalgoritm
```

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 ← 9, i ← 3, j ← 5
puneB(mat, n, i, j)
```
C. de 10 ori
D. de o infinitate de ori

19. Fie subalgoritmul `calcul(a, b)` cu parametrii de intrare ***a*** și ***b*** numere naturale, $1 \le a \le 1000$, $1 \le b \le 1000$.

```
Subalgoritm calcul(a, b):
   Dacă a ≠ 0 atunci
      returnează calcul(a DIV 2, b + b) + b * (a MOD 2)
   SfDacă
   returnează 0
SfSubalgoritm
```

Care din afirmațiile de mai jos sunt false?

A. dacă ***a*** și ***b*** sunt egale, subalgoritmul returnează valoarea lui ***a***
B. dacă ***a*** = 1000 și ***b*** = 2, subalgoritmul se autoapelează de 10 ori
C. valoarea calculată și returnată de subalgoritm este ***a*** / 2 + 2 * ***b***
D. instrucțiunea de pe linia 5 se execută o singură dată

20. Fie subalgoritmul `factoriPrimi(n, d, k, x)` care determină cei ***k*** factori primi ai unui număr natural ***n***, începând căutarea factorilor primi de la valoarea ***d***. Parametrii de intrare sunt numerele naturale ***n***, ***d*** și ***k***, iar parametrii de ieșire sunt șirul ***x*** cu cei ***k*** factori primi (1 ≤ ***n*** ≤ 10000, 2 ≤ ***d*** ≤ 10000, 0 ≤ ***k*** ≤ 10000).

```
Subalgoritm factoriPrimi(n, d, k, x):
   Dacă n MOD d = 0 atunci
      k ← k + 1
      x[k] ← d
   SfDacă
   CâtTimp n MOD d = 0 execută
      n ← n DIV d
   SfCâtTimp
   Dacă n > 1 atunci
      factoriPrimi(n, d + 1, k, x)
   SfDacă
SfSubalgoritm
```

Stabiliți de câte ori se autoapelează subalgoritmul `factoriPrimi(n, d, k, x)` în următoarea secvență de instrucțiuni:

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

A. de 3 ori
B. de 5 ori
C. de 6 ori
D. de același număr de ori ca și în cadrul secvenței de instrucțiuni:

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

21. Se consideră numerele naturale ***m*** și ***n*** (0 ≤ ***m*** ≤ 10, 0 ≤ ***n*** ≤ 10) și subalgoritmul `Ack(m, n)` care calculează valoarea funcției Ackermann pentru valorile ***m*** și ***n***.

```
Subalgoritm Ack(m, n)
  Dacă m = 0 atunci
       returnează n + 1
  altfel
    Dacă m > 0 și n = 0 atunci
      returnează Ack(m - 1, 1)
    altfel
      returnează Ack(m - 1, Ack(m, n - 1))
    SfDacă
  SfDacă
SfSubalgoritm
```

Precizați de câte ori se autoapelează subalgoritmul `Ack(m, n)` prin executarea secvenței de instrucțiuni:

```
m ← 1, n ← 2
Ack(m, n)
```

A. de 7 ori
B. de 5 ori
C. de 10 ori
D. de același număr de ori ca și în cazul executării secvenței de instrucțiuni:

```
m ← 1, n ← 3
Ack(m, n)
```

22. Definim operația de *trunchiere* a unui număr natural cu ***k*** cifre $\overline{c_1 c_2 \dots c_k}$ astfel:
$$trunchiere(\overline{c_1 c_2 \dots c_k}) = \begin{cases} 0, & \text{dacă } k < 2; \\ \overline{c_1 c_2}, & \text{altfel} \end{cases}.$$
Precizați care dintre următorii subalgoritmi calculează *suma trunchierilor* elementelor unui șir ***x*** cu ***n*** numere naturale mai mici decât 1 000 000 (***n*** – număr natural, 1 ≤ ***n*** ≤ 1 000)? De exemplu, dacă ***n*** = 4 și ***x*** = (213, 7, 78 347, 22), atunci suma trunchierilor este 21 + 0 + 78 + 22 = 121.

A.

```
Subalgoritm sumăTrunchiată(n, x)
s ← 0
CâtTimp n > 0 execută
    Dacă x[n] > 9 atunci
    CâtTimp x[n] > 99 execută
       x[n] ← x[n] DIV 10
    SfCâtTimp
       s ← s + x[n]
    SfDacă
    n ← n - 1
 SfCâtTimp
 returnează s
SfSubalgoritm
```

B.

```
Subalgoritm sumăTrunchiată(n, x)
    s ← n
    CâtTimp n > 0 execută
       Dacă x[n] > 9 atunci
       CâtTimp x[n] > 99 execută
                x[n] ← x[n] DIV 10
       SfCâtTimp
       s ← s + x[n]
       SfDacă
       n ← n - 1
    SfCâtTimp
    returnează s
 SfSubalgoritm
```

C.

```
Subalgoritm sumăTrunchiată(n, x)
    s ← 0
    CâtTimp n > 0 execută
       Dacă x[n] > 9 atunci
         CâtTimp x[n] > 99 execută
              x[n] ← x[n] DIV 10
              s ← s + x[n]
         SfCâtTimp
      SfDacă
      n ← n - 1
```

```
    SfCâtTimp
    returnează s
SfSubalgoritm
```

D.

```
Subalgoritm sumăTrunchiată(n, x)
    s ← 0
    CâtTimp x[n] > 99 execută
       x[n] ← x[n] DIV 10
    SfCâtTimp
       s ← s + x[n]
    returnează s
SfSubalgoritm
```

23. Fie ***s*** un şir de numere naturale unde elementele $\boldsymbol{s_i}$ sunt de forma $\boldsymbol{s_i} = \begin{cases} x, & \text{dacă } i = 1 \\ x + 1, & \text{dacă } i = 2 \\ s_{(i-1)}@s_{(i-2)} & \text{dacă } i > 2 \end{cases}$,

(***i*** = 1, 2, ...). Operatorul @ concatenează cifrele operandului stâng cu cifrele operandului drept, în această ordine (cifre aferente reprezentării în baza 10), iar ***x*** este un număr natural (1 ≤ ***x*** ≤ 99). De exemplu, dacă ***x*** = 3, șirul ***s*** va conține valorile 3, 4, 43, 434, 43443, ... Precizați numărul cifrelor acelui termen din șirul ***s*** care precede termenul format din ***k*** (1 ≤ ***k*** ≤ 30) cifre.

A. dacă ***x*** = 15 și ***k*** = 6, numărul cifrelor termenului aflat în șirul ***s*** în fața termenului format din ***k*** cifre este 5.
B. dacă ***x*** = 2 și ***k*** = 8, numărul cifrelor termenului aflat în șirul ***s*** în fața termenului format din ***k*** cifre este 5.
C. dacă ***x*** = 14 și ***k*** = 26, numărul cifrelor termenului aflat în șirul ***s*** în fața termenului format din ***k*** cifre este 16.
D. dacă ***x*** = 5 și ***k*** = 13, numărul cifrelor termenului aflat în șirul ***s*** în fața termenului format din ***k*** cifre este 10.

24. Fie un șir ***x*** cu ***n*** elemente numere naturale (3 ≤ ***n*** ≤ 10 000) și numărul natural ***k*** (1 ≤ ***k*** < ***n***). Subalgoritmul `permCirc(n, k, x)` ar trebui să genereze permutarea circulară a șirului ***x*** cu ***k*** poziții la stânga (de exemplu, șirul (4, 5, 2, 1, 3) este o permutare circulară cu 2 poziții la stânga pentru șirul (1, 3, 4, 5, 2)). Din păcate subalgoritmul `permCirc(n, k, x)` nu este corect, deoarece pentru anumite valori ale lui ***n*** și ***k*** nu determină rezultat corect.

```
Subalgoritm permCirc(n, k, x)
   c ← k
   Pentru j = 1, c execută
      unde ← j
      nr ← x[unde]
      Pentru i = 1, n / c – 1 execută
         deUnde ← unde + k
         Dacă deUnde > n atunci
            deUnde ←deUnde - n
         SfDacă
         x[unde] ← x[deUnde]
         unde ← deUnde
      SfPentru
      x[unde] ← nr
   SfPentru
SfSubalgoritm
```

Alegeți valorile lui ***n***, ***k*** și ***x*** pentru care algoritmul `permCirc(n, k, x)` generează o permutare circulară a șirului ***x*** cu ***k*** poziții la stânga:

A. ***n*** = 6, ***k*** = 2, ***x*** = (1, 2, 3, 4, 5, 6)
B. ***n*** = 8, ***k*** = 3, ***x*** = (1, 2, 3, 4, 5, 6, 7, 8)
C. ***n*** = 5, ***k*** = 3, ***x*** = (1, 2, 3, 4, 5)
D. ***n*** = 8, ***k*** = 4, ***x*** = (1, 2, 3, 4, 5, 6, 7, 8)

25. 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*** ($2 \le x \le 1000$) este *norocos*? Fiecare subalgoritm are ca parametru de intrare numărul ***x***, iar ca parametri de ieșire numărul natural nenul ***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.

A.
```
Subalgoritm norocos(x, start, esteNorocos):
    xpatrat ← x * x
    esteNorocos ← fals
    start ← -1, k ← 1, s ← 0
    CâtTimp k ≤ xpatrat - x și nu esteNorocos execută
        Pentru i ← k, k + x - 1 execută
            s ← s + i
        SfPentru
        Dacă s = xpatrat atunci
            esteNorocos ← adevărat
            start ← k
        SfDacă
    SfCâtTimp
SfSubalgoritm
```

B.
```
Subalgoritm norocos(x, start, esteNorocos):
    xpatrat ← x * x
    esteNorocos ← fals
    start ← -1, k ← 1
    CâtTimp k ≤ xpatrat - x și nu esteNorocos execută
        s ← 0
        Pentru i ← k, k + x - 1 execută
            s ← s + i
        SfPentru
        Dacă s = xpatrat atunci
            esteNorocos ← adevărat
            start ← k
        SfDacă
        k ← k + 1
    SfCâtTimp
SfSubalgoritm
```

C.
```
Subalgoritm norocos(x, start, esteNorocos):
    Dacă x MOD 2 = 0 atunci
        esteNorocos ← fals
        start ← -1
    altfel
        esteNorocos ← adevărat
        start ← (x + 1) DIV 2
    SfDacă
SfSubalgoritm
```

D.
```
Subalgoritm norocos(x, start, esteNorocos):
    Dacă x MOD 2 = 0 atunci
        esteNorocos ← fals
       start ← -1
    altfel
        esteNorocos ← adevărat
        start ← x DIV 2
    SfDacă
SfSubalgoritm
```

26. Se consideră subalgoritmul `alg(x, b)` cu parametrii de intrare două numere naturale ***x*** și ***b*** (1 ≤ ***x*** ≤ 1000, 1 < ***b*** ≤ 10).

```
Subalgoritm alg(x, b):
   s ← 0
   CâtTimp x > 0 execută
      s ← s + x MOD b
      x ← x DIV b
   SfCâtTimp
   returnează s MOD (b - 1) = 0
SfSubalgoritm
```

Precizați efectul acestui subalgoritm.

A. verifică dacă suma cifrelor reprezentării în baza ***b*** - 1 a numărului ***x*** este divizibilă cu ***b*** - 1
B. verifică dacă numărul natural ***x*** este divizibil cu ***b*** - 1
C. verifică dacă suma cifrelor reprezentării în baza ***b*** a numărului ***x*** este divizibilă cu ***b*** - 1
D. verifică dacă suma cifrelor numărului ***x*** este divizibilă cu ***b*** - 1

27. Se consideră șirul (1, 2, 3, 2, 5, 2, 3, 7, 2, 4, 3, 2, 5, 11, ...) format astfel: plecând de la șirul numerelor naturale, se înlocuiesc numerele care nu sunt prime cu divizorii lor proprii, fiecare divizor ***d*** fiind considerat o singură dată pentru fiecare număr. Care dintre subalgoritmi determină al ***n***-lea element al acestui șir (***n*** - număr natural, 1 ≤ ***n*** ≤ 1000)?

A.
```
Subalgoritm identificare(n):
    a ← 1, b ← 1, c ← 1
    CâtTimp c < n execută
        a ← a + 1, b ← a, c ← c + 1, d ← 2
        f ← false
        CâtTimp c ≤ n și d ≤ a DIV 2 execută
            Dacă a MOD d = 0 atunci
                c ← c + 1, b ← d, f ← true
            SfDacă
            d ← d + 1
        SfCâtTimp
        Dacă f atunci
            c ← c - 1
        SfDacă
    SfCâtTimp
    returnează b
SfSubalgoritm
```

B.

```
Subalgoritm identificare(n):
    a ← 1, b ← 1, c ← 1
    CâtTimp c < n execută
        c ← c + 1, d ← 2
        CâtTimp c ≤ n și d ≤ a DIV 2 execută
            Dacă a MOD d = 0 atunci
                c ← c + 1, b ← d
            SfDacă
            d ← d + 1
        SfCâtTimp
        a ← a + 1, b ← a
    SfCâtTimp
    returnează b
SfSubalgoritm
```

C.

```
Subalgoritm identificare(n):
    a ← 1, b ← 1, c ← 1
    CâtTimp c < n execută
        a ← a + 1, d ← 2
        CâtTimp c < n și d ≤ a execută
            Dacă a MOD d = 0 atunci
                c ← c + 1, b ← d
            SfDacă
            d ← d + 1
       SfCâtTimp
    SfCâtTimp
    returnează b
SfSubalgoritm
```

D.

```
Subalgoritm identificare(n):
    a ← 1, b ← 1, c ← 1
    CâtTimp c < n execută
        b ← a, a ← a + 1, c ← c + 1, d ← 2
        CâtTimp c ≤ n și d ≤ a DIV 2 execută
            Dacă a MOD d = 0 atunci
                c ← c + 1, b ← d
            SfDacă
            d ← d + 1
        SfCâtTimp
    SfCâtTimp
    returnează b
SfSubalgoritm
```

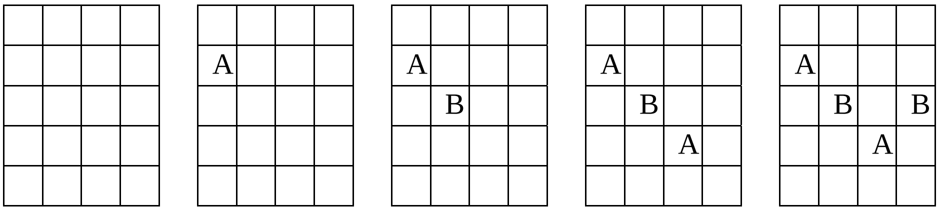
28. Dreptunghiul cu laturile de lungimi ***m*** și ***n*** (***m***, ***n*** – numere naturale, $0 < \boldsymbol{m} < 101$, $0 < \boldsymbol{n} < 101$) este împărțit în pătrățele cu latura de lungime 1. Se consideră subalgoritmul `dreptunghi(m, n)`:

```
Subalgoritm dreptunghi(m, n)
   d ← m
   c ← n
   CâtTimp d ≠ c execută
      Dacă d > c atunci
         d ← d – c
      altfel
         c ← c – d
      SfDacă
   SfCâtTimp
   returnează m + n – d
SfSubalgoritm
```

Precizați efectul acestui subalgoritm.

A. Calculează și returnează numărul pătrățelelor cu latura de lungime 1 traversate de o diagonală a dreptunghiului.
B. Determină în ***d*** cel mai mare divizor comun al laturilor dreptunghiului și returnează diferența dintre suma laturilor dreptunghiului și ***d***.
C. Dacă ***m*** = 8 și ***n*** = 12, returnează 16.
D. Dacă ***m*** = 6 și ***n*** = 11, returnează 15.

29. Fie o tablă dreptunghiulară împărțită în ***n*** × ***m*** căsuțe (***n*** – numărul liniilor, ***m*** – numărul coloanelor, ***n***, ***m*** – numere naturale, 2 ≤ ***n*** ≤ 100, 2 ≤ ***m*** ≤ 100). Pe rând, doi jucători A și B execută mutări alternative astfel: la fiecare mutare un jucător hașurează o singură căsuță care este vecină pe diagonală cu căsuța hașurată la pasul anterior de către celălalt jucător și care este nehașurată până în acel moment. Jucătorul care nu mai poate muta, pierde. Jucătorul A face prima mutare, hașurând o căsuță de pe tablă.



a b c d e

Exemplu de tablă de joc: a) inițială (*n* = 5 și *m* = 4), b) după prima mutare (mutarea lui A), c) după a 2-a mutare (mutarea lui B), d) după a 3-a mutare (mutarea lui A), e) după a 4-a mutare (mutarea lui B)

Determinați condiția în care jucătorul A are strategie sigură de câștig (adică va câștiga jocul, oricare ar fi mutările jucătorului B) și care poate fi prima mutare efectuată de jucătorul A pentru a câștiga jocul.

A. condiția: numărul ***m*** este impar;
prima mutare a jucătorului A: o căsuță aflată pe prima linie de sus a tablei (linia 1) și pe o coloană de indice impar
B. condiția: numărul ***n*** este impar;
prima mutare a jucătorului A: o căsuță aflată pe o linie de indice par și pe prima coloană din stânga tablei (coloana 1);
C. condiția: ambele numere ***n*** și ***m*** sunt pare;
prima mutare a jucătorului A: căsuța din colțul stânga sus (de pe linia 1, coloana 1);
D. condiția: cel puțin unul dintre numerele ***n*** și ***m*** este impar;
prima mutare a jucătorului A: căsuța din colțul stânga sus (de pe linia 1, coloana 1).

30. O matrice cu 8 linii, formată doar din elemente 0 și 1, are următoarele trei proprietăți:

a. prima linie conține un singur element cu valoarea 1,
b. linia $\boldsymbol{j}$ conține de două ori mai multe elemente nenule decât linia $\boldsymbol{j} - 1$, pentru orice $\boldsymbol{j} \in \{2, 3, \ldots, 8\}$,
c. ultima linie conține un singur element cu valoarea 0.

Care este numărul total de elemente cu valoarea 0 din matrice?

A. 777
B. 769
C. 528
D. nu există o astfel de matrice

Răspunsuri corecte:

1.	**D**	**11.**	**A, D**	**21.**	**B**
2.	**B**	**12.**	**B**	**22.**	**A**
3.	**A**	**13.**	**B**	**23.**	**B, C**
4.	**B, C**	**14.**	**C, D**	**24.**	**A, D**
5.	**C**	**15.**	**A**	**25.**	**B, C**
6.	**A, C, D**	**16.**	**B, D**	**26.**	**B, C**
7.	**B, C**	**17.**	**A, C**	**27.**	**A**
8.	**A, B, C**	**18.**	**A, B**	**28.**	**A, C**
9.	**A, B, D**	**19.**	**A, C**	**29.**	**A, D**
10.	**A, C**	**20.**	**A, D**	**30.**	**A**