UNIVERSITATEA BABEŞ-BOLYAI
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

Concurs Mate-Info – martie 2021

Proba scrisă la Informatică

NOTĂ IMPORTANTĂ:
În lipsa altor precizări, presupuneți că toate operatiile aritmetice se efectuează pe tipuri de date nelimitate (nu există *overflow* / *underflow*).
De asemenea, numerotarea indicilor tuturor șirurilor începe de la 1.

1. Se consideră expresia următoare, în care *a* este un număr natural.

```
((a < 4) SAU (a < 5)) ȘI (a > 2)
```

Pentru ce valori ale lui *a* va avea expresia valoarea ***ADEVĂRAT***?

A. a = 3
B. a = 4
C. a = 2
D. Expresia nu va avea niciodată valoarea ***ADEVĂRAT***

2. Subalgoritmul de mai jos are ca parametri de intrare un șir **v** cu **n** numere naturale nenule (v[1], v[2], …, v[n]) și numărul întreg **n** ($1 \leq n \leq 10000$).

```
Subalgoritm f(v, n):
    x ← 0
    Pentru i ← 1, n execută
        c ← v[i]
        Câttimp c MOD 3 = 0 execută
            x ← x + 1
            c ← c DIV 3
        SfCâttimp
    SfPentru
    returnează x
SfSubalgoritm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul returnează numărul numerelor divizibile cu 3 din șirul ***v***
B. Subalgoritmul returnează cel mai mare număr ***k*** astfel încât v[1] * v[2] * … * v[n] este divizibil cu 3^k
C. Subalgoritmul returnează cel mai mare număr ***k*** astfel încât v[1] + v[2] + … + v[n] este divizibil cu 3^k
D. Subalgoritmul returnează suma numerelor divizibile cu 3 din șirul ***v***

3. Se consideră expresia următoare, în care *x* este un număr natural pozitiv.

```
(x MOD 2) + ((x + 1) MOD 2)
```

Care din afirmațiile de mai jos sunt adevărate?

A. Expresia are valoarea 1 pentru orice număr natural pozitiv x.
B. Expresia are valoarea 1 dacă şi numai dacă x este un număr par.
C. Expresia are valoarea 1 dacă şi numai dacă x este un număr impar.
D. Există număr natural x pentru care expresia are o valoare strict mai mare decât 1.

4. Fie subalgoritmul `prelucrare(x, n)` definit mai jos, care primeşte ca și parametru un șir **x** cu **n** numere reale nenule (x[1], x[2], …, x[n]) și numărul întreg **n** ($1 \leq n \leq 10000$).
Operatorul `/` reprezintă împărțirea reală (ex. `3/2=1,5`).

```
Subalgoritm prelucrare(x, n):
    p ← 1
    Pentru k ← 1, n - 1 execută
        p ← p + 1
        Pentru i ← 1, n - 1 execută
            Dacă x[i] > x[i + 1] atunci
                x[i] ← x[i] * x[i + 1]
                x[i + 1] ← x[i] / x[i + 1]
                x[i] ← x[i] / x[i + 1]
            SfDacă
        SfPentru
    SfPentru
    n ← p
SfSubalgoritm
```

Care dintre următoarele afirmații descriu modificarea aplicată șirului **x** în urma apelului subalgoritmului `prelucrare(x, n)`?

A. Elementele șirului **x** vor rămâne nemodificate
B. Elementele șirului **x** vor fi în ordine descrescătoare
C. Elementele șirului **x** vor fi în ordine crescătoare
D. Numărul **n** este decrementat cu o unitate

5. Se consideră subalgoritmul `calcul(a, n)`, care primește ca parametru un șir **a** cu **n** numere naturale (a[1], a[2], …, a[n]) și numărul întreg **n** ($1 \leq n \leq 10000$).

```
Subalgoritm calcul(a, n):
    Dacă n = 0 atunci
        returnează 0
    altfel
        returnează a[n] * (a[n] MOD 2) + calcul(a, n - 1)
    SfDacă
SfSubalgoritm
```

Pentru ce valori a numărului **n** și a șirului **a** funcția `calcul(a,n)` va returna valoarea 10?

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

6. Se consideră subalgoritmul `calcul(v, n)`, care primește ca parametru un șir **v** cu **n** numere naturale (v[1], v[2], …, v[n]) și numărul întreg n ($1 \leq n \leq 10000$).

```
Subalgoritm calcul(v, n):
    m ← 0
    x ← 0
    s ← 0
    Pentru i ← 1, n execută
        s ← s + v[i]
        m ← m + (s MOD 2 + x) MOD 2
        x ← s MOD 2
    SfPentru
    returnează m
SfSubalgoritm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul calculează și returnează suma numerelor impare din șirul v
B. Subalgoritmul calculează și returnează suma numerelor pare din șirul v
C. Subalgoritmul calculează și returnează numărul de numere impare din șirul v
D. Subalgoritmul calculează și returnează numărul de numere pare din șirul v

7. Se consideră subalgoritmul `magic(x)`, unde **x** este un număr natural ($1 \leq x \leq 32000$).

```
Subalgoritm magic(x):
    st ← 1
    dr ← x
    Câttimp st ≤ dr execută
        mj ← (st + dr) DIV 2
        Dacă mj * mj = x atunci
            returnează adevărat
        SfDacă
        Dacă mj * mj < x atunci
            st ← mj + 1
        altfel
            dr ← mj – 1
        SfDacă
    SfCâttimp
    returnează fals
SfSubalgoritm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul verifică dacă există un pătrat perfect mai mic decât x.
B. Subalgoritmul numără divizorii primi ai numărului x.
C. Subalgoritmul verifică dacă numărul x este prim.
D. Subalgoritmul verifică dacă numărul x este pătrat perfect.

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

```
Subalgoritm ceFace(n):
    a ← n
    b ← 0
    Câttimp a ≠ 0 execută
      b ← b * 10 + a MOD 10
      a ← a DIV 10
    SfCâttimp
    Dacă n = b atunci
      returnează adevărat
    altfel
      returnează fals
    SfDacă
SfSubalgoritm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul verifică dacă numărul n este prim.
B. Subalgoritmul verifică dacă numărul n este palindrom.
C. Subalgoritmul returnează întotdeauna adevărat.
D. Subalgoritmul verifică dacă numărul n este divizibil cu 10.

9. Se consideră subalgoritmul `calculeaza(a,b)`, unde **a** și **b** sunt numere naturale ($1 \leq a, b \leq 10000$).

```
Subalgoritm calculeaza(a, b):
    x ← 1
    Pentru i ← 1, b execută
        x ← (x MOD 10) * a
    SfPentru
    returnează x
SfSubalgoritm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Dacă **a** = 2021 și **b** = 2021, valoarea returnată de subalgoritm este 2021.
B. Pentru toate apelurile subalgoritmului cu **a** = 2021 și $1 \leq$ **b** ≤ 10000, valoarea returnată este 2021.
C. Dacă **a** = 7777 și **b** = 2021, valoarea returnată este 7777.
D. Pentru toate apelurile subalgoritmului cu $1 \leq$ **a** ≤ 10000 și **b** = 2021, valoarea returnată este valoarea lui a.

10. Câte elemente se găsesc pe cele două diagonale ale unei matrice pătratice cu **n** linii și **n** coloane ($10 \leq n \leq 1000$)? Se numără elementele de pe poziții distincte.

A. 2 * n
B. n * n
C. 2 * n - 1
D. 2 * n – (n MOD 2)

11. Care dintre expresiile logice următoare au valoarea `ADEVĂRAT` pentru ***a*** = 1 și ***b*** = 0?

A. `NU (((a > 0) ȘI (b < 1)) SAU (a > 1))`
B. `((b > 0) ȘI (b < 1)) SAU ((a > 0) ȘI (a < 2))`
C. `(NU (a > b)) SAU (NU (b > 0))`
D. `(a > 0) SAU ((b > 0) ȘI (b < 0)) SAU (a < 1)`

12. Subalgoritmii $\texttt{calcul}_i\texttt{(e, n)}$, $1 \leq$ **i** ≤ 4, primesc ca parametri o matrice **e** de **n** linii și **n** coloane (e[1][1], …, e[1][n], e[2][1], …, e[n][n]) și un număr natural **n** ($1 \leq$ **n** ≤ 1000). Alegeți variantele de răspuns care conțin definiția subalgoritmului $\texttt{calcul}_i\texttt{(e, n)}$, care are rezultat diferit față de toate celelalte trei variante, adică $\texttt{calcul}_i\texttt{(e, n)} \neq \texttt{calcul}_j\texttt{(e, n)}$ $\forall$ **e**, **n**, **j**, $1 \leq$ **j** ≤ 4, $i \neq j$ (**e** și **n** conform specificației anterioare).

A.

```
Subalgoritm calcul₁(e, n):
    s ← 0
    Pentru i ← 1, n execută
        s ← s + e[i][i]
    SfPentru
    returnează s
SfSubalgoritm
```

B.

```
Subalgoritm calcul₂(e, n):
    s ← 0
    Pentru i ← 1, n execută
        Pentru j ← 1, n, execută
            Dacă i = j atunci
                s ← s + e[i][j]
            SfDacă
        SfPentru
    SfPentru
    returnează s
SfSubalgoritm
```

C.

```
Subalgoritm calcul₃(e, n):
    s ← 0
    i ← 1
    Câttimp i ≤ n execută
        s ← s + e[i][i]
        i ← i + 1
    SfCâttimp
    returnează s
SfSubalgoritm
```

D.

```
Subalgoritm calcul₄(e, n):
    s ← 0
    Pentru i ← 1, n execută
        Pentru j ← i + 1, n, execută
            Dacă i = j atunci
                s ← s + e[i][j]
            SfDacă
        SfPentru
    SfPentru
    returnează s
SfSubalgoritm
```

13. Se consideră subalgoritmul `ceFace(a,b)`, unde **a** și **b** sunt numere naturale ($1 \leq$ **a** $<$ **b** ≤ 10000).

```
Subalgoritm ceFace(a, b):
    m ← a
    Câttimp b MOD m > 0 execută
        m ← m + 1
    SfCâttimp
    returnează m
SfSubalgoritm
```

Ce va returna apelul `ceFace(47, 100)`?

A. 48
B. 50
C. 3
D. 100

14. Se consideră subalgoritmul `afis(n)`, unde **n** este un număr natural ($0 \leq n \leq 10000$).

```
Subalgoritm afis(n):
    Scrie n
    Dacă n > 0 atunci
        afis (n - 1)
        Scrie n
    SfDacă
SfSubalgoritm
```

Ce se va afișa la apelul `afis(4)`?

A. 432100123
B. 123401234
C. 1234004321
D. 432101234

15. Care dintre următoarele baze de numerație ***x*** satisfac condiția $232_{(x)} \leq 67_{(10)}$?

A. x = 5
B. x = 3
C. x = 4
D. x = 6

16. Subalgoritmul `mutaZero(a, n)` primește ca și parametru un șir **a** de numere întregi, (a[1], a[2], …, a[n]) și numărul întreg **n** ($1 \leq n \leq 10000$). Subalgoritmul mută valorile de zero la finalul șirului, păstrând ordinea relativă a elementelor diferite de zero. De exemplu, dacă a este `[4, 0, 2, 5, 1, 0, 7, 11, 0, 3]`, după apelarea subalgoritmului, elementele lui a sunt `[4, 2, 5, 1, 7, 11, 3, 0, 0, 0]`. Care din implementările următoare ale subalgoritmului `mutaZero(a, n)` sunt corecte?

A.
```
Subalgoritm mutaZero(a, n):
    s ← ADEVĂRAT
    Câttimp s = ADEVĂRAT execută
        s ← FALS
        Pentru i ← 1, n - 1 execută
            Dacă a[i] = 0 atunci
                tmp ← a[i]
                a[i] ← a[i + 1]
                a[i + 1] ← tmp
                s ← ADEVĂRAT
            SfDacă
        SfPentru
    SfCâttimp
SfSubalgoritm
```

B.
```
Subalgoritm mutaZero(a, n):
    c ← 0
    Pentru i ← 0, n execută
        Dacă a[i] = 0 atunci
            c ← c + 1
        SfDacă
    SfPentru
    i ← n
    Câttimp c > 0 execută
        a[i] ← 0
        i ← i - 1
        c ← c - 1
    SfCâttimp
SfSubalgoritm
```

C.
```
Subalgoritm mutaZero(a, n):
    d ← 0
    i ← 1
    Câttimp i + d ≤ n execută
        Câttimp (i + d ≤ n) ȘI (a[i + d] = 0) execută
            d ← d + 1
        SfCâttimp
        Dacă i + d ≤ n atunci
            a[i] ← a[i + d]
            i ← i + 1
        SfDacă
    SfCâttimp
    Câttimp i ≤ n execută
        a[i] ← 0
        i ← i + 1
    SfCâttimp
SfSubalgoritm
```

D.
```
Subalgoritm mutaZero(a, n):
    i ← 1
    f ← n
    Câttimp i < f execută
        Câttimp (i < f) ȘI (a[i] ≠ 0) execută
            i ← i + 1
        SfCâttimp
        Câttimp (i < f) ȘI (a[f] = 0) execută
            f ← f - 1
        SfCâttimp
        Dacă i < f atunci
            tmp ← a[i]
            a[i] ← a[f]
            a[f] ← tmp
        SfDacă
    SfCâttimp
SfSubalgoritm
```

17. Fie șirul ***X***=1,2,2,3,3,3,4,4,4,4,5,5,5,5,5,6,6,6,6,6,6,7…, în care fiecare număr ***n*** apare de ***n*** ori pe poziții consecutive. Considerând că primul element din șir este pe poziția 1, pe ce poziții din șir apare valoarea 21?

A. Pe pozițiile din intervalul [210,230]
B. Pe pozițiile din intervalul [211,231]
C. Pe pozițiile din intervalul [212,232]
D. Pe pozițiile din intervalul [209,229]

18. Se consideră subalgoritmul `f(a, b)`, unde ***a*** si ***b*** sunt numere întregi (-10000 ≤ **a**, **b** ≤ 10000).
```
Subalgoritm f(a, b):
    Scrie "FMI"
    Dacă (a = 0) SAU (b = 0) atunci
        returnează 1
    SfDacă
    Dacă a > b atunci
        returnează f(a - b * b, a * (a - b) - b * (a - b))
    SfDacă
    Dacă a ≤ b atunci
        returnează f(b - a * a, a * (a - b) - b * (a - b))
    SfDacă
SfSubalgoritm
```

Precizați de câte ori se scrie textul *FMI* la executarea secvenței de cod:
```
f(f(3, 2), f(2, 3))
```

A. De 8 ori
B. De 6 ori
C. De 3 ori
D. De o infinitate de ori

19. Se consideră subalgoritmul recursiv `ceFace(n, i)`, unde ***n*** este un număr natural (2 ≤ ***n*** ≤ 1000).

```
Subalgoritm ceFace(n, i):
    Dacă i * i > n atunci
        returnează 0
    SfDacă
    Dacă i * i = n atunci
        returnează i
    SfDacă
    Dacă n MOD i = 0 atunci
        returnează i + n DIV i + ceFace(n, i + 1)
    altfel
        returnează ceFace(n, i + 1)
    SfDacă
SfSubalgoritm
```

Precizați care dintre următoarele afirmații sunt adevărate pentru apelul `ceFace(n, 2)`:

A. Subalgoritmul calculează și returnează dublul sumei tuturor divizorilor proprii ai numărului `n`.
B. Subalgoritmul calculează și returnează suma divizorilor proprii ai numărului `n`.
C. Subalgoritmul calculează și returnează suma divizorilor proprii și improprii ai numărului `n`.
D. Subalgoritmul verifică dacă `n` este pătrat perfect. În caz afirmativ, returnează rădăcina lui pătrată. Altfel, returnează 0.

20. Se consideră subalgoritmul `ceFace(T, n, e)`, care primește ca și parametru un șir **T** cu **n** numere naturale ordonate crescător (T[1], T[2], …, T[n]) și numerele naturale **n** și **e** (1 ≤ **n, e** ≤ 10000).

```
Subalgoritm ceFace(T, n, e):
    Dacă e MOD 2 = 0 atunci
        a ← 1
        b ← n
        Câttimp a ≤ b execută
            m ← (a + b) DIV 2
            Dacă e < T[m] atunci
                b ← m - 1
            altfel
                Dacă e > T[m] atunci
                    a ← m + 1
                altfel
                    returnează adevărat
                SfDacă
            SfDacă
        SfCâttimp
        returnează fals
    altfel
        c ← 1
        Câttimp c ≤ n execută
            Dacă e = T[c] atunci
                returnează adevărat
            SfDacă
            c ← c + 1
        SfCâttimp
        returnează fals
    SfDacă
SfSubalgoritm
```

Precizați care dintre următoarele afirmații sunt adevărate:

A. Subalgoritmul nu verifică dacă numărul **e** se află pe o poziție pară în șirul **T**.
B. Subalgoritmul verifică dacă numărul **e** se află în șirul **T**, iar dacă numărul **e** este impar, algoritmul de căutare folosit este căutarea binară.
C. Subalgoritmul verifică dacă numărul **e** se află în șirul **T**, iar dacă numărul **e** este par, algoritmul de căutare folosit este căutarea binară.
D. Subalgoritmul verifică dacă numărul **e** se află în șirul **T** doar dacă numărul **e** este impar.

21. Se dorește afișarea triunghiurilor echilaterale folosind doar caracterele * (asterisc) și . (punct). Exemplul de mai jos ilustrează un triunghi având latura de n=5 asteriscuri. Pentru acesta a fost necesară utilizarea a 12 asteriscuri și 23 de puncte.

```
....*
...*.*
..*...*
.*.....*
*.*.*.*.*
```

Care din afirmațiile de mai jos sunt adevărate?

A. Pentru n=2, este nevoie de exact 3 asteriscuri și 4 puncte.
B. Pentru n=7, este nevoie de exact 18 asteriscuri și 52 puncte.
C. Pentru n=7, este nevoie de exact 18 asteriscuri și 48 puncte.
D. Pentru n=15, este nevoie de exact 42 asteriscuri și 288 puncte.

22. Spunem că un șir având **n** caractere este antipalindrom dacă toate perechile de caractere egal depărtate de extremități sunt distincte două câte două (cu excepția celui din mijloc dacă **n** este impar). De exemplu, *asdfg* și *xlxe* sunt antipalindroame, dar *asdsg* nu este.

Fie subalgoritmul `antipalindrom(s, stânga, dreapta)` care primește ca și parametru un șir **s** cu **n** ($1 \leq$ **n** ≤ 10000) caractere (s[1], s[2], …, s[n]), și numerele naturale `stânga` și `dreapta`.

Care din următoarele implementări vor returna *adevărat* pentru apelul `antipalindrom(s, 1, n)` dacă și numai dacă șirul **s** este antipalindrom?

A.
```
Subalgoritm antipalindrom(s, stânga, dreapta):
    Dacă stânga = dreapta atunci
        returnează adevărat
    altfel
        prim ← s[stânga]
        ultim ← s[dreapta]
        Dacă prim = ultim atunci
            returnează fals
        altfel
            returnează antipalindrom(s, stânga + 1, dreapta - 1)
        SfDacă
    SfDacă
SfSubalgoritm
```

B.

```
Subalgoritm antipalindrom(s, stânga, dreapta):
    Dacă stânga ≥ dreapta atunci
        returnează adevărat
    SfDacă
    prim ← s[stânga]
    ultim ← s[dreapta]
    Dacă prim = ultim atunci
        returnează fals
    altfel
        returnează antipalindrom(s, stânga + 1, dreapta - 1)
    SfDacă
SfSubalgoritm
```

C.

```
Subalgoritm antipalindrom(s, stânga, dreapta):
    Dacă stânga > dreapta atunci
        returnează adevărat
    altfel
        prim ← s[stânga]
        ultim ← s[dreapta]
        Dacă prim ≠ ultim atunci
            returnează fals
        altfel
            returnează antipalindrom(s, stânga + 1, dreapta - 1)
        SfDacă
    SfDacă
SfSubalgoritm
```

D.

```
Subalgoritm antipalindrom(s, stânga, dreapta):
    Dacă stânga > dreapta atunci
        returnează adevărat
    SfDacă
    prim ← s[stânga]
    ultim ← s[dreapta]
    Dacă prim ≠ ultim atunci
        returnează adevărat
    SfDacă
    returnează antipalindrom(s, stânga + 1, dreapta - 1)
SfSubalgoritm
```

23. Fie subalgoritmul `ordo(n, a)` care primește ca și parametru un număr natural **n** (1 ≤ **n** ≤ 10000) și un șir **a** având 2n elemente numere naturale (a[1], a[2], …, a[2n]).
Considerând că numărul de elemente pare ale șirului **a** este egal cu numărul de elemente impare, care din următorii subalgoritmi rearanjează elementele șirului **a** astfel încât elementele impare să aibă indici impari, iar elementele pare să aibă indici pari?

A.

```
Subalgoritm ordo(n, a):
  Pentru i ← 1, 2 * n - 1 execută
    Dacă a[i] MOD 2 ≠ i MOD 2 atunci
      Pentru j ← i + 1, 2 * n execută
        Dacă a[j] MOD 2 ≠ j MOD 2 atunci
          a[i] ← a[i] + a[j]
          a[j] ← a[i] - a[j]
          a[i] ← a[i] - a[j]
        SfDacă
      SfPentru
    SfDacă
  SfPentru
SfSubalgoritm
```

B.
```
Subalgoritm ordo(n, a):
  Pentru i ← 1, 2 * n - 1 execută
    Dacă a[i] MOD 2 ≠ i MOD 2 atunci
      Pentru j ← i + 1, 2 * n execută
        Dacă (a[i] MOD 2 ≠ i MOD 2) ȘI (a[j] MOD 2 ≠ j MOD 2) atunci
          a[i] ← a[i] + a[j]
          a[j] ← a[i] - a[j]
          a[i] ← a[j] - a[i]
        SfDacă
      SfPentru
    SfDacă
  SfPentru
SfSubalgoritm
```

C.
```
Subalgoritm ordo(n, a):
  Pentru i ← 1, 2 * n - 1 execută
    Dacă a[i] MOD 2 ≠ i MOD 2 atunci
      Pentru j ← i + 1, 2 * n execută
        Dacă (a[i] MOD 2 ≠ i MOD 2) ȘI
             (a[j] MOD 2 ≠ j MOD 2) ȘI
             (a[i] MOD 2 ≠ a[j] MOD 2) atunci
          a[i] ← a[i] + a[j]
          a[j] ← a[i] - a[j]
          a[i] ← a[i] - a[j]
        SfDacă
      SfPentru
    SfDacă
  SfPentru
SfSubalgoritm
```

D.
```
Subalgoritm ordo(n, a):
  Pentru i ← 1, 2 * n - 1 execută
    Pentru j ← i + 1, 2 * n execută
      Dacă (a[j] MOD 2 = 0) ȘI
           ((a[j] MOD 2 ≠ 0) SAU (a[j] MOD 2 ≠ 0)) ȘI
           (a[j] MOD 2 = 0) atunci
        a[i] ← a[i] + a[j]
        a[j] ← a[i] - a[j]
        a[i] ← a[i] - a[j]
      SfDacă
    SfPentru
  SfPentru
SfSubalgoritm
```

24. Dorim să partiționăm un șir de **n** ($1 \leq n \leq 1000$) valori în **k** ($1 \leq k \leq n$) subsecvențe adiacente de lungimi egale (fiecare element al șirului aparține exact unei subsecvențe). Dacă **n** nu este divizibil cu **k**, acceptăm ca diferența de lungime între oricare două subsecvențe să fie cel mult 1.
Se dau mai jos patru variante de a genera indicii primelor elemente ale tuturor subsecvențelor **j** ($1 \leq j \leq k$). Numerotând elementele șirului de la 1, care dintre aceste variante satisfac cerința de mai sus?

A. `((j * n - 1) DIV k) - 1`
B. `((j - 1) * n ) DIV k + 1`
C. `(j - 1) * (n DIV k)`
D. `((j - 1) * n + k) DIV k`

25. Fie $b_n b_{n-1} \ldots b_0$ reprezentarea binară a numărului natural B, $2021 \leq B \leq 2021^{2021}$. Care dintre următoarele afirmații sunt adevărate?

A. Dacă valoarea expresiei b_0-b_1+b_2-b_3+.....+$(-1)^n$ *b_n este zero, atunci B este divizibil cu 3
B. Dacă valoarea expresiei b_0-b_1+b_2-b_3+....+$(-1)^n$ *b_n este divizibilă cu 3, atunci B este divizibil cu 3
C. B este divizibil cu 3 dacă suma cifrelor binare este divizibilă cu 3, dar nu cu 9
D. Dacă B este divizibil cu 3, atunci valoarea expresiei b_0-b_1+b_2-b_3+....+$(-1)^n$ *b_n. este divizibilă cu 3

26. Considerăm subalgoritmul `prefix(n)`, care dat fiind numărul natural **n** ($9 < n < 10^{20}-1$), caută cel mai lung prefix al său care se regăsește și în interiorul numărului (exceptând prima și ultima cifră a sa). Subalgoritmul returnează lungimea acestui prefix.
Exemplu:

pentru ***n*** = 12133121, prefixul este 12 și are lungimea 2.
pentru ***n*** = 34534536, prefixul este 3453 și are lungimea 4.
pentru ***n*** = 1223, un astfel de prefix nu există (considerăm că are lungime 0).

Știind că indexarea șirurilor începe de la 1, care din următoarele variante reprezintă implementări corecte ale subalgoritmului `prefix(n)`?

A.
```
Subalgoritm prefix(n):
  nr ← n
  c ← 0
  p ← 1
  CâtTimp nr > 0 execută
    c ← c + 1
    nr ← nr DIV 10
    p ← p * 10
  SfCâtTimp

  f1 ← 100
  f2 ← p DIV 100
  k ← 1
  ok ← 0
  CâtTimp ok = 0 execută
    n1 ← n DIV f1
    f3 ← 10
    Pentru i ← 1 ,k execută
      n2 ← (n DIV f3) MOD f2
      Dacă n1 = n2 atunci
        ok ← 1
        returnează c - k - 1
      SfDacă
      f3 ← f3 * 10
    SfPentru
    f1 ← f1 * 10
    f2 ← f2 DIV 10
    k ← k + 1
  SfCâtTimp
  returnează -1
SfSubalgoritm
```

B.

```
Subalgoritm prefix(n):
  c ← [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
  nr ← n
  p ← 0
  CâtTimp nr > 0 execută
    c[p + 1] ← nr MOD 10
    nr ← nr DIV 10
    p ← p + 1
  SfCâtTimp
  Pentru i ← 1, p - 2 execută
    Pentru j ← p – 1, i + 1, -1 execută
      ok ← 1
      Pentru k ← 0, i - 1 execută
        Dacă c[p - 1 - k] ≠ c[j - k] atunci
          ok ← 0
        SfDacă
      SfPentru
      Dacă ok = 1 atunci
        returnează i
      SfDacă
    SfPentru
  SfPentru
  returnează -1
SfSubalgoritm
```

C.

```
Subalgoritm prefix(n):
  c ← [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
  nr ← n
  p ← 0
  CâtTimp nr > 0 execută
    c[p + 1] ← nr MOD 10
    nr ← nr DIV 10
    p ← p + 1
  SfCâtTimp
  Pentru i ← p - 2, 1, -1 execută
    Pentru j ← p – 1, i + 1, -1 execută
      ok ← 1
      Pentru k ← 0, i - 1 execută
        Dacă c[p - k] ≠ c[j - k] atunci
          ok ← 0
        SfDacă
      SfPentru
      Dacă ok = 1 atunci
        returnează j
      SfDacă
    SfPentru
  SfPentru
  returnează 0
SfSubalgoritm
```

D.

```
Subalgoritm prefix(n):
  nr ← n
  c ← 0
  p ← 1
  CâtTimp nr > 0 execută
    c ← c + 1
    nr ← nr DIV 10
    p ← p * 10
  SfCâtTimp
  f1 ← p DIV 10
  f2 ← 10
  k  ← c - 2
  ok ← -1
  Pentru t ← 1, c - 2 execută
    n1 ← n DIV f1
    f3 ← 10
    Pentru i ← 1, k execută
      n2 ← (n DIV f3) MOD f2
      Dacă n1 = n2 atunci
        Dacă ok < c - k - 1 atunci
          ok  ←  c - k - 1
        SfDacă
      SfDacă
      f3 ← f3 * 10
    SfPentru
    f1 ← f1 DIV 10
    f2 ← f2 * 10
    k ← k - 1
  SfPentru
  Dacă ok < 0 atunci:
    returnează 0
  SfDacă
  returnează ok
SfSubalgoritm
```

27. Se consideră tabelul de mai jos, având 16 celule (4 linii notate cu 1, 2, 3, 4, și 4 coloane notate cu A, B, C, D). Unele celule conțin valori constante (de ex. celula B3), altele, care încep cu “=” conțin rezultatul unei expresii aritmetice cu 2 termeni. Fiecare termen este fie o valoare constantă, fie, dacă termenul începe cu simbolul $, o referință către valoarea dintr-o altă celulă. De exemplu, în celula A4 avem rezultatul operației aritmetice de scădere din valoarea constantă 5 a valorii din celula A2. Pentru o anumită celulă **i**, notăm cu X(i) numărul minim de celule (inclusiv cele care conțin valori constante) ale căror valori trebuie cunoscute înainte de a calcula valoarea din celula **i**. Similar, notăm cu Y(i) numărul maxim de celule (inclusiv cele care conțin valori constante, dar excluzând celula **i**) ale căror valori pot fi calculate fără a cunoaște valoarea din celula **i**. Care dintre următoarele afirmații sunt adevărate despre valorile lui X(A2) și Y(A2)?

	A	B	C	D
1	= $B4 – $C1	=$B3 + $D3	3	= $A4 * $C3
2	= $B1 + $B2	= $D3 + 11	= $D3 + $D2	2
3	= $B1 – $D3	11	= $D4 * $D4	= $D2 + 2
4	= 5 - $A2	= $C1 * $C1	= $A3 / 2	=15 / 3

A. X(A2) = 2 și Y(A2) = 1
B. X(A2) = 5 și Y(A2) = 13
C. X(A2) = 6 și Y(A2) = 4
D. X(A2) = X(C4) și Y(A2) = Y(B2) + 1

28. Subalgoritmul `simplifică(nr, num)` obține fracția ireductibilă **aux1 / aux2** cu proprietatea că **aux1 / aux2 = nr / num** (**aux1**, **aux2**, **nr**, **num** numere naturale, **num, aux2** ≠ 0).

```
Subalgoritm  simplifică(nr, num):
  d ← funcție(nr, num)
  aux1 ← nr DIV d
  aux2 ← num DIV d
SfSubalgoritm
```

Care dintre variantele următoare ale subalgoritmului `funcție(a, b)` sunt corecte?

A.
```
Subalgoritm funcție(a, b):
 d ← 1
 Câttimp adevărat execută
   Dacă (a MOD d = 0) ŞI (b MOD d = 0) atunci
     returnează d
   SfDacă
   d ← d + 1
 SfCâttimp
SfSubalgoritm
```

B.
```
Subalgoritm funcție(a, b):
 Câttimp b ≠ 0 execută
   c ← a MOD b
   a ← b
   b ← c
 SfCâttimp
 returnează a
SfSubalgoritm
```

C.
```
Subalgoritm funcție(a, b):
 Câttimp a ≠ b execută
   Dacă a > b atunci
     a ← a - b
   altfel
     b ← b - a
   SfDacă
 SfCâttimp
 returnează a
SfSubalgoritm
```

D.
```
Subalgoritm funcție(a, b):
  d ← a
  Câttimp ((a MOD d ≠ 0) SAU (b MOD d ≠ 0)) execută
    d ← d - 1
  SfCâttimp
  returnează d
SfSubalgoritm
```

29. Se consideră următorul subalgoritm recursiv `fibonacci(n)`, unde **n** este un număr natural (1 ≤ n ≤ 100). Să se determine de câte ori se afișează mesajul "`Aici`" în cazul unui apel `fibonacci(n)` (considerând împreună apelul inițial și toate apelurile recursive generate).

```
Subalgoritm fibonacci(n):
  Dacă n ≤ 1 atunci
    returnează 1
  altfel
    Scrie "Aici"
    returnează fibonacci(n - 1) + fibonacci(n - 2)
  SfDacă
SfSubalgoritm
```

A. De `fibonacci(n)` ori.
B. De `fibonacci(n-1)` ori.
C. De `fibonacci(n)-1` ori.
D. De `fibonacci(n) - fibonacci(n-1)` ori.

30. Se considera expresia: $E(x) = a_0 + a_1*x + a_2*x^2 + a_3*x^3 + a_5*x^5$, unde a_0, a_1, a_2, a_3, a_5 și x sunt numere reale nenule. Numărul minim de înmulțiri necesare pentru a calcula valoarea expresiei ***E(x)*** este:

A. 4
B. 5
C. 7
D. 11