BABEŞ-BOLYAI UNIVERSITY
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Mate-Info UBB Competition and Admissions Exam - Model 2021

Written Test in Computer Science

1. The `generate(n)` subalgorithm processes natural number ***n*** (0 < ***n*** < 100).

```
Subalgorithm generate(n):
   nr ← 0
   For i ← 1, 1801 do
      used_i ← false
   EndFor
   While not used_n do
      sum ← 0, used_n ← true
      While (n ≠ 0) do
         digit ← n MOD 10, n ← n DIV 10
         sum ← sum + digit * digit * digit
      EndWhile
      n ← sum, nr ← nr + 1
   EndWhile
   return nr
EndSubalgorithm
```

State the effect of this subalgorithm.

A. repeatedly calculates the sum of the cubes of ***n***'s digits until the sum is equal to ***n***, after which it returns the number of repetitions
B. calculates the sum of the cubes of ***n***'s digits and returns this sum
C. calculates the sum of the cubes of ***n***'s digits, replaces ***n*** with the obtained sum and returns it
D. calculates how many times number ***n*** was replaced with the sum of the cubes of ***n***'s digits until a previously calculated value or the number itself is obtained; this number is then returned

2 Let us consider array ***s*** with ***k*** boolean elements and subalgorithm `evaluation(s, k, i)`, where ***k*** and ***i*** are natural numbers (0 ≤ ***i*** ≤ ***k*** ≤ 100).

```
Subalgorithm evaluation(s, k, i)
   If i ≤ k then
      If s_i then
         return s_i
      else
         return (s_i or evaluation(s, k, i + 1))
      EndIf
   else
      return false
   EndIf
EndSubalgorithm
```

State how many times subalgorithm `evaluation(s, k, i)` is self-called given the following instruction sequence:

```
s ← (false, false, false, false, false, false, true, false, false, false)
k ← 10
i ← 3
evaluation(s, k, i)
```

A. 3 times
B. the same number of times as in the following instruction sequence

```
s ← (false, false, false, false, false, false, false, true)
k ← 8
i ← 4
evaluation(s, k, i)
```

C. 6 times
D. infinite number of times

3. Consider the `expression(n)` subalgorithm, with ***n*** a natural number ($1 \leq \boldsymbol{n} \leq 10000$).

```
Subalgorithm expression(n):
   If n > 0 then
      If n MOD 2 = 0 then
         return -n * (n + 1) + expression(n - 1)
      else
         return n * (n + 1) + expression(n - 1)
      EndIf
   else
      return 0
   EndIf
EndSubalgorithm
```

Specify the mathematical form for expression *E(n)* as calculated by the following subalgorithm:

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. An integer data type represented using ***x*** bits (***x*** a strictly positive natural number) can hold values from the following range:

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. Consider subalgorithm `f(a, b)`:

```
Subalgorithm f(a, b):
       If a > 1 then
              return b * f(a - 1, b)
       else
              return b * f(a + 1, b)
       EndIf
EndSubalgorithm
```

How many times will subalgorithm `f` be self-called by executing the following instruction sequence:

```
a ← 4
b ← 3
c ← f(a, b)
```

A. 4 times
B. 3 times
C. infinite number of times
D. never

6. Let us consider the following logical expression: **(NOT** ***Y*** **OR** ***Z*****) OR (*****X*** **AND** ***Y*****)**. Choose values for ***X***, ***Y***, ***Z*** such that the result of evaluating the expression is *true*:

A. ***X*** ← *false*; ***Y*** ← *false*; ***Z*** ← *false*;
B. ***X*** ← *false*; ***Y*** ← *true*; ***Z*** ← *false*;
C. ***X*** ← *true*; ***Y*** ← false; ***Z*** ← *true*;
D. ***X*** ← *false*; ***Y*** ← *true*; ***Z*** ← *true*;

7. Specify which of the following expressions is true if and only if the natural number ***n*** is divisible by 3 and has the last digit 4 or 6:

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

8. Consider the following subalgorithm:

```
Subalgorithm f(a):
    If a != 0 then
        return a + f(a - 1)
    else
        return 0
    EndIf
EndSubalgorithm
```

Which of the following statements are false?

A. if a is negative, the subalgorithm returns 0
B. the value calculated by f is a * (a + 1) / 4
C. the subalgorithm computes the sum of the natural numbers smaller or equal to a
D. calling f(-5) results in an infinite cycle.

9. Consider the following subalgorithm:

```
Subalgorithm SA9(a):
    If a < 50 then
        If a MOD 3 = 0 then
            return SA9(2 * a - 3)
        else
            return SA9(2 * a - 1)
        EndIf
    else
        return a
    EndIf
EndSubalgorithm
```

For which values of input parameter ***a*** will the subalgorithm return 61?

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

10. Consider subalgorithm `process(v, k)`, where ***v*** is an array of ***k*** natural numbers ($1 \leq k \leq 1\ 000$).

```
Subalgorithm process(v, k)
   i ← 1, n ← 0
   While i ≤ k and vi ≠ 0 do
      y ← vi, c ← 0
      While y > 0 do
         If y MOD 10 > c then
            c ← y MOD 10
         EndIf
         y ← y DIV 10
      EndWhile
      n ← n * 10 + c
      i ← i + 1
   EndWhile
   return n
EndSubalgorithm
```

State for which values of ***v*** and ***k*** the subalgorithm returns 928.

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

11. Let us consider the following logical expression: `(X OR Z) AND (NOT X OR Y)`. Choose values for ***X***, ***Y***, ***Z*** such that the result of evaluating the expression is `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. Consider the following program:

C Version

```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 result = prelVector(v, &n);
  printf("%d;%d", n, result);
  return 0;
}
```

C++ Version

```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  result = prelVector(v, n);
   cout << n <<";" <<  result;
   return 0;
}
```

Pascal Version

```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,  result :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;
    result := prelVector(v,n);
   write(n, ';',  result);
end.
```

Which of the following pairs is displayed after the execution of the program.

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

13. Consider the following algorithm in pseudocode:

```
read a
For i=1, a-1 do
       For j=i+2, a do
              If i+j>a-1 then
                     write a, ‘ ‘, i, ‘ ‘, j
                     start new line
              EndIf
       EndFor
EndFor
```

How many pairs of numbers will be displayed after the execution of the algorithm for a=8?

A. 13
B. 15
C. 20
D. No answer is correct

14. Which of the following subalgorithms returns the largest multiple of number ***a***, multiple that is smaller or equal with natural number ***b*** (0 < ***a*** < 10 000, 0 < ***b*** < 10 000, ***a*** < ***b***)?

A.
```
Subalgorithm f(a, b):
  c ← b
  While c MOD a = 0 execute
    c ← c - 1
  EndWhile
  return c
EndSubalgorithm
```

B.
```
Subalgorithm f(a, b):
  If a < b then
    return f(2 * a, b)
  else
    If a = b then
      return a
    else
      return b
    EndIf
  EndIf
EndSubalgorithm
```

C.
```
Subalgorithm f(a, b):
  return (b DIV a) * a
EndSubalgorithm
```

D.
```
Subalgorithm f(a, b):
  If b MOD a = 0 then
    return b
  EndIf
  return f(a, b - 1)
```

15. Consider all arrays having length ***l*** ϵ {1, 2, 3} built using letters from the set {*a*, *b*, *c*, *d*, *e*}. How many of them are sorted strictly decreasing and have an odd number of vowels? (*a* and *e* are vowels)

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

16. Consider subalgorithm `belongs(x, a, n)`, which checks whether natural number ***x*** belongs to set ***a*** having ***n*** elements; ***a*** is an array of ***n*** elements that represents a natural numbers set ($1 \le \boldsymbol{n} \le 200$, $1 \le \boldsymbol{x} \le 1000$). Let us consider subalgorithms `reunion(a, n, b, m, c, p) and compute(a, n, b, m, c, p)`, described below, where ***a***, ***b*** and ***c*** are arrays that represent natural number sets having ***n***, ***m*** and ***p*** elements respectively ($1 \le \boldsymbol{n} \le 200$, $1 \le \boldsymbol{m} \le 200$, $1 \le \boldsymbol{p} \le 400$). Input parameters are ***a***, ***n***, ***b***, ***m*** and ***p***, and output parameters are ***c*** and ***p***.

```
1.    Subalgorithm reunion(a, n, b, m, c, p):
2.      If n = 0 then
3.        For i ← 1, m do
4.          p ← p + 1
5.          c_p ← b_i
6.        EndFor
7.      else
8.        If not belongs(a_n, b, m) then
9.          p ← p + 1
10.         c_p ← a_n
11.       EndIf
12.       reunion(a, n - 1, b, m, c, p)
13.     EndIf
14.   EndSubalgorithm
```

```
1. Subalgorithm compute(a, n, b, m, c,
   p):
2.   p ← 0
3.   reunion(a, n, b, m, c, p)
4. EndSubalgorithm
```

Which of the following statements are always true:

A. when set ***a*** contains a single element, calling subalgorithm `compute(a, n, b, m, c, p)` results in an infinite loop
B. when set ***a*** contains 4 elements, calling subalgorithm `compute(a, n, b, m, c, p)` results in executing the instruction on line 12 a number of 4 times
C. when set ***a*** contains 5 elements, calling subalgorithm `compute(a, n, b, m, c, p)` results in executing the instruction on line 2 of the `reunion` subalgorithm a number of 5 times
D. when sets ***a*** and ***b*** have the same elements, after the execution of subalgorithm `compute(a, n, b, m, c, p)` set ***c*** will have the same number of elements as set ***a***

17. Consider the `compute(n)` subalgorithm, with $\boldsymbol{n}$ a natural number ($1 \le \boldsymbol{n} \le 10000$).

```
Subalgorithm compute(n):
  x ← 0, z ← 1
  While z ≤ n do
    x ← x + 1
    z ← z + 2 * x
    z ← z + 1
  EndWhile
  return x
EndSubalgorithm
```

Which of the following statements are **false**?

A. If $\boldsymbol{n} < 8$, then `compute(n)` returns 3.
B. If $\boldsymbol{n} \ge 85$ and $\boldsymbol{n} < 100$, then `compute(n)` returns 9.
C. The subalgorithm computes and returns the number of strictly positive and strictly smaller than $\boldsymbol{n}$ perfect squares.
D. The subalgorithm computes and returns the integer part of $\boldsymbol{n}$'s root.

18. Consider square matrix ***mat*** of size $\boldsymbol{n} \times \boldsymbol{n}$ ($\boldsymbol{n}$ – odd natural number, $3 \le \boldsymbol{n} \le 100$) and subalgorithm `placeB(mat, n, i, j)` which places character 'b' on certain positions of matrix ***mat***. Parameters $\boldsymbol{i}$ and $\boldsymbol{j}$ are natural numbers ($1 \le \boldsymbol{i} \le \boldsymbol{n}$, $1 \le \boldsymbol{j} \le \boldsymbol{n}$).

```
Subalgorithm placeB(mat, n, i, j):
   If i ≤ n DIV 2 then
      If j ≤ n - i then
         mat[i][j] ← 'b'
         placeB(mat, n, i, j + 1)
      else
         placeB(mat, n, i + 1, i + 2)
      EndIf
   EndIf
EndSubalgorithm
```

How many times will subalgorithm `placeB(mat, n, i, j)` is self-called for the following instruction sequence:

```
n ← 7, i ← 2, j ← 4
placeB(mat, n, i, j)
```

A. 5 times
B. The same number of times as for the next instruction sequence
```
n ← 9, i ← 3, j ← 5
placeB(mat, n, i, j)
```
C. 10 times
D. infinite number of times

19. Consider the `compute(a, b)` subalgorithm, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are natural numbers, $1 \le \boldsymbol{a} \le 1000$, $1 \le \boldsymbol{b} \le 1000$.

```
Subalgorithm compute(a, b):
   If a ≠ 0 then
      return compute(a DIV 2, b + b) + b * (a MOD 2)
   EndIf
   return 0
EndSubalgorithm
```

Which of the following statements are **false**?

A. if ***a*** and ***b*** are equal, the subalgorithm return ***a*** value
B. if ***a*** = 1000 and ***b*** = 2, the subalgorithm calls itself 10 times
C. the subalgorithm computes and returns ***a*** / 2 + 2 * ***b***
D. the instruction on line 5 is executed only once

20. Consider the `primeFactors(n, d, k, x)` subalgorithm that determines the ***k*** prime factors for natural number ***n***, starting the search from the value ***d***. Input parameters are natural numbers ***n***, ***d*** și ***k***, and the output parameter is array ***x*** consisting of the ***k*** prime factors (1 ≤ ***n*** ≤ 10000, 2 ≤ ***d*** ≤ 10000, 0 ≤ ***k*** ≤ 10000).

```
Subalgorithm primeFactors(n, d, k, x):
   If n MOD d = 0 then
      k ← k + 1
      x[k] ← d
   EndIf
   While n MOD d = 0 do
      n ← n DIV d
   EndWhile
   If n > 1 then
      primeFactors(n, d + 1, k, x)
   EndIf
EndSubalgorithm
```

How many times will the `primeFactors(n, d, k, x)` subalgorithm is self-called by executing the following instruction sequence:

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

A. 3 times
B. 5 times
C. 9 times
D. the same number of times as in the following sequence:

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

21. Consider the natural numbers ***m*** and ***n*** (0 ≤ ***m*** ≤ 10, 0 ≤ ***n*** ≤ 10) and the subagorithm `Ack(m, n)` which computes the value of Ackermann function for ***m*** and ***n*** values.

```
Subalgorithm Ack(m, n)
  If m = 0 then
       return n + 1
  else
    If m > 0 and n = 0 then
      return Ack(m - 1, 1)
    else
      return Ack(m - 1, Ack(m, n - 1))
    EndIf
  EndIf
EndSubalgorithm
```

How many times the subalgorithm `Ack(m, n)` is self-called executing the following sequence:

```
m ← 1, n ← 2
Ack(m, n)
```

A. 7 times
B. 5 times
C. 10 times
D. the same number of times as in the following sequence:

```
m ← 1, n ← 3
Ack(m, n)
```

22. Let us define the operation *truncation* of a natural number with $\boldsymbol{k}$ digits $\overline{c_1 c_2 \ldots c_k}$ so that: $truncation(\overline{c_1 c_2 \ldots c_k}) = \begin{cases} 0, & \text{if } k < 2; \\ \overline{c_1 c_2}, & \text{else} \end{cases}$.

State which of the following subalgorithms computes the sum of the truncations of the elements of an array $\boldsymbol{x}$ with $\boldsymbol{n}$ natural numbers smaller than 1 000 000 ($\boldsymbol{n}$ – număr natural, $1 \le \boldsymbol{n} \le 1\,000$)? For example, if $\boldsymbol{n} = 4$ and $\boldsymbol{x} = (213, 7, 78347, 22)$, then the sum of truncations is 21 + 0 + 78 + 22 = 121.

A.

```
Subalgorithm sumTruncation(n, x)
s ←0
While n > 0 do
    If x[n] > 9 then
      While x[n] > 99 do
         x[n] ← x[n] DIV 10
      EndWhile
      s ← s + x[n]
    EndIf
    n ← n - 1
 EndWhile
 return s
EndSubalgorithm
```

B.

```
Subalgorithm sumTruncation (n, x)
    s ← n
    While n > 0 do
       If x[n] > 9 then
       While x[n] > 99 do
                x[n] ← x[n] DIV 10
       EndWhile
       s ← s + x[n]
       EndIf
       n ← n - 1
    EndWhile
    return s
EndSubalgorithm
```

C.

```
Subalgorithm sumTruncation (n, x)
    s ← 0
    While n > 0 do
       If x[n] > 9 then
         While x[n] > 99 do
              x[n] ← x[n] DIV 10
              s ← s + x[n]
         EndWhile
```

```
      EndIf
      n ← n - 1
    EndWhile
    return s
EndSubalgorithm
```

D.

```
Subalgorithm sumTruncation (n, x)
    s ← 0
    While x[n] > 99 do
       x[n] ← x[n] DIV 10
    EndWhile
       s ← s + x[n]
    return s
EndSubalgorithm
```

23. Let us consider the array ***s*** with natural number elements, where $s_i = \begin{cases} x, & \text{if } i = 1 \\ x+1, & \text{if } i = 2 \\ s_{(i-1)}@s_{(i-2)} & \text{if } i > 2 \end{cases}$, (***i*** = 1, 2, ...). The operator @ concatenates the digits of the left-hand operand with the digits of the right-hand operand, in this order (the digits correspond to the base-10 representation of the number), and ***x*** is a natural number ($1 \leq x \leq 99$). For example, if ***x*** = 3, the array ***s*** contains the following values 3, 4, 43, 434, 43443, ... State the number of digits of the element of ***s*** that precedes the element with ***k*** digits ($1 \leq k \leq 30$).

A. if ***x*** = 15 and ***k*** = 6, the number of digits of the element of ***s*** that precedes the element with ***k*** digits is 5.
B. dacă ***x*** = 2 și ***k*** = 8, the number of digits of the element of ***s*** that precedes the element with ***k*** digits is 5.
C. dacă ***x*** = 14 și ***k*** = 26, the number of digits of the element of ***s*** that precedes the element with ***k*** digits is 16.
D. dacă ***x*** = 5 și ***k*** = 13, the number of digits of the element of ***s*** that precedes the element with ***k*** digits is 10.

24. Let us consider the array ***x*** with ***n*** natural number elements ($3 \leq n \leq 10\,000$) and a natural number ***k*** ($1 \leq k < n$). The subalgorithm `permCirc(n, k, x)` should generate a circular permutation of the array ***x*** with ***k*** positions to left (for example, the array (4, 5, 2, 1, 3) is a circular permutation with 2 positions to left of the array (1, 3, 4, 5, 2)). Unfortunately `permCirc(n, k, x)` subalgorithm is not correct, because it does not compute the correct result for certain values of ***n*** and ***k***.

```
Subalgorithm permCirc(n, k, x)
   c ← k
   For j = 1, c do
      permTo ← j
      nr ← x[permTo]
      For i = 1, n / c – 1 do
         permFrom ← permTo + k
         If permFrom > n then
            permFrom ← permFrom - n
         EndIf
         x[permTo] ← x[permFrom]
         permTo ← permFrom
      EndFor
      x[permTo] ← nr
   EndFor
EndSubalgorithm
```

Chose the values of ***n***, ***k*** and ***x*** for which the subalgorithm `permCirc(n, k, x)` generates a circular permutation of the array ***x*** with ***k*** positions to left:

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. A non-zero natural number ***x*** is *lucky* if its square can be written as a sum of ***x*** consecutive natural numbers. For example, 7 is a lucky number because $7^2 = 4 + 5 + 6 + 7 + 8 + 9 + 10$.

Which of the following subalgorithms check whether natural number ***x*** ($2 \leq x \leq 1000$) is *lucky*? Each subalgorithm has ***x*** as input, and non-zero natural number ***start*** and boolean variable ***isLucky*** as outputs. If ***x*** is lucky, then ***isLucky*** = *true* and the value of ***start*** is the first number of the sum (for example, if ***x*** = 7, then ***start*** = 4); if ***x*** is not lucky, then ***isLucky*** = *false* and ***start*** = -1.

A.
```
Subalgorithm lucky(x, start, isLucky):
  xsquare ← x * x
  isLucky ← false
  start ← -1, k ← 1, s ← 0
  While k ≤ xsquare - x and not isLucky do
     For i ← k, k + x - 1 do
        s ← s + I
     EndFor
     If s = xsquare then
        isLucky ← true
        start ← k
     EndIf
  EndWhile
EndSubalgorithm
```

B.
```
Subalgoritm lucky(x, start, isLucky):
  xsquare ← x * x
  isLucky ← false
  start ← -1, k ← 1
  While k ≤ xsquare - x and not isLucky do
     s ← 0
     For i ← k, k + x - 1 do
        s ← s + i
     EndFor
     If s = xsquare then
        isLucky ← true
        start ← k
     EndIf
     k ← k + 1
  EndWhile
EndSubalgorithm
```

C.

```
Subalgorithm lucky(x, start, isLucky):
  If x MOD 2 = 0 then
     isLucky ← false
     start ← -1
  else
     isLucky ← true
     start ← (x + 1) DIV 2
  EndIf
EndSubalgorithm
```

D.

```
Subalgorithm lucky(x, start, isLucky):
  If x MOD 2 = 0 then
     isLucky ← false
     start ← -1
  else
     isLucky ← true
     start ← x DIV 2
  EndIf
EndSubalgorithm
```

26. Consider the `alg(x, b)` subalgorithm, with input parameters natural numbers ***x*** and ***b*** ($1 \leq x \leq 1000$, $1 < b \leq 10$).

```
Subalgorithm alg(x, b):
   s ← 0
   While x > 0 do
      s ← s + x MOD b
      x ← x DIV b
   EndWhile
   return s MOD (b - 1) = 0
EndSubalgorithm
```

What is the effect of the subalgorithm above.

A. checks whether the sum of ***x***'s digits in base ***b – 1*** is divisible with ***b – 1***.
B. checks whether natural number ***x*** is divisible with ***b – 1.***
C. checks whether the sum of ***x***'s digits in base ***b*** is divisible with ***b – 1***.
D. checks whether the sum of ***x***'s digits in base ***b*** is divisible with ***b – 1.***

27. Consider the series (1, 2, 3, 2, 5, 2, 3, 7, 2, 4, 3, 2, 5, 11, ...), built as follows: starting from the natural numbers' series, replace each non-prime number with its proper divisors, with each divisor *d* considered once per number. Which of the following subalgorithms determines the ***n***-th element of this series (***n*** – natural number, $1 \leq n \leq 1000$)?

A.

```
Subalgorithm identification(n):
    a ← 1, b ← 1, c ← 1
    While c < n do
        a ← a + 1, b ← a, c ← c + 1, d ← 2
        f ← false
        While c ≤ n and d ≤ a DIV 2 do
            If a MOD d = 0 then
```

```
                c ← c + 1, b ← d, f ← true
            EndIf
            d ← d + 1
        EndWhile
        If f then
            c ← c - 1
        EndIf
    EndWhile
    return b
EndSubalgorithm
```

B.

```
Subalgorithm identification(n):
    a ← 1, b ← 1, c ← 1
    While c < n do
        c ← c + 1, d ← 2
        While c ≤ n and d ≤ a DIV 2 do
            If a MOD d = 0 then
                c ← c + 1, b ← d
            EndIf
            d ← d + 1
         EndWhile
         a ← a + 1, b ← a
    EndWhile
    return b
EndSubalgorithm
```

C.

```
Subalgorithm identification(n):
    a ← 1, b ← 1, c ← 1
    While c < n do
        a ← a + 1, d ← 2
        While c < n and d ≤ a do
            If a MOD d = 0 then
                c ← c + 1, b ← d
            EndIf
            d ← d + 1
        EndWhile
    EndWhile
    return b
EndSubalgorithm
```

D.

```
Subalgorithm identification(n):
    a ← 1, b ← 1, c ← 1
    While c < n do
        b ← a, a ← a + 1, c ← c + 1, d ← 2
        While c ≤ n and d ≤ a DIV 2 do
            If a MOD d = 0 then
                c ← c + 1, b ← d
            EndIf
            d ← d + 1
        EndWhile
    EndWhile
    return b
EndSubalgorithm
```

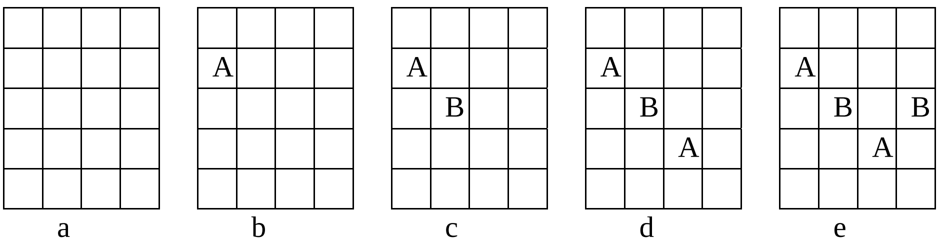
28. A rectangle with sides ***m*** and ***n*** (***m***, ***n*** – natural numbers, 0 < ***m*** < 101, 0 < ***n*** < 101) is split in squares of side **1**. Consider the `rectangle(m, n)` subalgorithm:

```
Subalgorithm rectangle(m, n)
   d ← m
   c ← n
   While d ≠ c do
      If d > c then
         d ← d – c
      else
         c ← c – d
      EndIf
   EndWhile
   return m + n – d
EndSubalgorithm
```

State the effect of this subalgorithm.

A. The subalgorithm computes and returns the number of squares of side 1 crossed by a diagonal of the rectangle.
B. The subalgorithm computes in ***d*** the greatest common divisor of rectangle sides and returns the difference between the sum of recatangle sides and ***d***.
C. If ***m*** = 8 and ***n*** = 12, the subalgorithm returns 16.
D. If ***m*** = 6 and ***n*** = 11, the subalgorithm returns 15.

29. Let us consider a rectangular board divided into ***n*** × ***m*** cells (***n*** is the number of rows, ***m*** is the number of columns, ***n, m*** are natural numbers, 2 ≤ ***n*** ≤ 100, 2 ≤ ***m*** ≤ 100). Two players, A and B perform alternate moves in the following way: in each turn a player colors one single cell which is not colored yet and is diagonally neighboring the cell colored in the previous turn by the other player. The player who cannot color a cell, loses the game. Player A performs the first move, coloring any cell on the board.



Example of a game board: a) initially (n = 5 and m = 4), b) after the first move (player A's move), c) after the second move (player B's move), d) after the third move (player A's move), e) after the fourth move (player B's move)

Determine the condition under which player A has a secure winning strategy (meaning that (s)he will win the game, no matter what player B does) and what could be the first move made by player A to win the game.

A. condition: the number m is odd
first move of player A: a cell on the first line of the board (line 1) and a column with odd index.
B. condition: the number n is odd
first move of player A: a cell on a line with an even index and on the leftmost column of the board (column 1)

C. condition: both n and m are even
first move of player A: the cell on the top left corner of the board (on line 1 and column 1)

D. condition: at least one of the numbers n and m is odd
first move of player A: the cell on the top left corner of the board (on line 1 and column 1)

30. A matrix has 8 rows, contains only 1's and 0's and has the following three properties:
a) there is a single element with value 1 on the first row,
b) row $\boldsymbol{j}$ contains twice as many non-zero elements as row $\boldsymbol{j} - 1$, for all $\boldsymbol{j} \in \{2, 3, \ldots, 8\}$,
c) on the last row there is a single element with value 0.

What is the total number of 0 elements in the matrix?

A. 777
B. 769
C. 528
D. there is no such matrix

Answers:

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