Algoritmi care lucrează cu tablouri unidimensionale

Problema 1

Enunț

Se dă un șir de caractere care conține o propoziție formată din cuvinte separate prin spații. Scrieți un algoritm care inversează **ordinea cuvintelor** din propoziție, fără a inversa literele din interiorul fiecărui cuvânt, și stochează rezultatul într-un nou șir de caractere.

Exemplu

A. Date de intrare: "ana are mere"
B. Date de ieșire: "mere are ana"

Cod C++

```
#include <iostream>
#include <string.h>
using namespace std;

void ceFace(char sentence[], int len, char result[]) {

    int end = len - 1;
    int pos = 0;

    while (end >= 0) {
        while (end >= 0 && sentence[end] == ' ')
            end--;
        int we = end;
        while (end >= 0 && sentence[end] != ' ')
            end--;
        int ws = end + 1;

        for (int i = ws; i <= we; i++)
            result[pos++] = sentence[i];

        if (end >= 0)
            result[pos++] = ' ';
    }
    result[pos] = '\0';
}



int main() {
    char sentence[200];
    char result[200];

    cout << "Enter a sentence: ";
    cin.getline(sentence, 200);
```

```
    ceFace(sentence, strlen(sentence), result);

    cout << result << endl;
    return 0;
}
```

Rezolvare

1. Sărim peste spații de la sfârșit:
    - Pornim de la ultimul caracter al șirului.
    - Dacă întâlnim spații, le ignorăm și mergem mai în stânga.
2. Identificăm un cuvânt:
    - Odată ce găsim o literă, mergem în continuare spre stânga până când găsim un spațiu sau ajungem la începutul șirului. Astfel găsim începutul (***ws*** în implementare) și sfârșitul (***we*** în implementare) cuvântului curent.
3. Copiem cuvântul în rezultat:
    - Copiem caracterele de la ***ws*** la ***we*** în noul șir, de la stânga la dreapta. Nu vrem să schimbăm ordinea literelor dintr-un cuvânt.
    - Adăugăm un spațiu după cuvânt (dacă nu este ultimul).
4. **Repetăm**:
    - Continuăm același proces pentru restul șirului, până când am parcurs toate cuvintele.

Ilustrare

W	E	L	C	O	M	E		S	P	R	I	N	G	_	_
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

we = 13

ws = 8

result = spring; pos = 6

W	E	L	C	O	M	E		S	P	R	I	N	G	_	_
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

we = 6

ws = 0

result = spring welcome; pos = 13

W	E	L	C	O	M	E		S	P	R	I	N	G	_	_
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Cum am putea să o întâlnim pe subiectul de admitere

Se dă algoritmul ***ceFace***, unde ***sentence*** este un şir de caractere de dimensiune ***len***, 0 < ***len*** < 200.

```
Algorithm ceFace(sentence, len):
    end ← len - 1
    pos ← 0
    While end ≥ 0 execute
        While end ≥ 0 AND sentence[end] = ' ' execute
            end ← end - 1
        EndWhile
        we ← end
        While end ≥ 0 AND sentence[end] ≠ ' ' execute
            end ← end - 1
        EndWhile
        ws ← end + 1
        For i ← ws to we execute
            result[pos] ← sentence[i]
            pos ← pos + 1
        EndFor
        If end ≥ 0 then
            result[pos] ← ' '
            pos ← pos + 1
        EndIf
    EndWhile
EndAlgorithm
```

Selectaţi răspunsurile care sunt adevărate după apelul algoritmului ***ceFace(sentence, len)***.

A. Dimensiunea şirului ***sentence*** va fi întotdeauna egală cu dimensiunea şirului ***result***.
B. Există cel puţin un caz în care şirul ***result*** va fi identic cu şirul ***sentence***.
C. Şirul ***result*** va fi identic cu şirul ***sentence*** dacă şi numai dacă şirul de caractere ***sentence*** este palindrom.
D. Algoritmul elimină doar spaţiile duplicate de la finalul şirului ***sentence***.

Problema 2

Enunț. Secvența *count-and-say* este definită recursiv astfel:

- countAndSay(1) = "1"
- countAndSay(n) reprezintă algoritmul Run Length Encoding (RLE) aplicat pe secvența countAndSay(n-1).

Scrieți un algoritm care determină al n-lea termen din secvența *count-and-say* pentru 1<= ***n*** < *20.*

RLE este o metodă de compresie în care fiecare grup de caractere identice consecutive este înlocuit cu numărul aparițiilor urmat de caracterul în sine. De exemplu, "3322251" devine "23321511" deoarece "33" → "23" (doi de 3), "222" → "32" (trei de 2), "5" → "15" (un 5) și "1" → "11" (un 1).

Run-length encoding este cel mai eficient pe date care conțin multe secvențe repetitive, de exemplu imagini grafice simple precum iconițe, desene tehnice și jocuri simple. Pentru fișierele care nu conțin multe secvențe repetitive, aplicarea RLE ar putea crește dimensiunea fișierului.

Aplicând regula de la countAndSay recursiv pentru n=5, se obține următoarea secvență:

```
countAndSay(1) = "1"
countAndSay(2) = "11"      (un 1)
countAndSay(3) = "21"      (doi de 1)
countAndSay(4) = "1211"    (un 2, un 1)
countAndSay(5) = "111221"  (un 1, un 2, doi de 1)
```

Rezolvare

Funcția primește ***n*** și un buffer ***result*** în care va fi scris răspunsul. Cazul de bază este simplu, când n este 1, răspunsul este "1".

Pentru orice n > 1, funcția calculează mai întâi ***countAndSay(n-1)*** recursiv într-un buffer local ***prev***, după care aplică RLE pe acesta. Partea de RLE funcționează prin parcurgerea lui ***prev*** cu indexul j - pentru fiecare poziție, se reține caracterul curent ***ch*** și se numără de câte ori se repetă consecutiv. Odată ce secvența se termină, se scrie ***count*** urmat de ***ch*** în result, apoi se trece la următorul grup.

Cod C++

```
#include <string.h>
#include <iostream>
using namespace std;

void countAndSay(int n, char result[1024]) {
    if (n == 1) {
        result[0] = '1';
        result[1] = '\0';
        return;
    }

    char prev[5000];
    countAndSay(n - 1, prev);

    int len = strlen(prev);
    int pos = 0, j = 0;

    while (j < len) {
        char ch = prev[j];
        int count = 0;

        while (j < len && prev[j] == ch) {
            j++;
            count++;
        }

        result[pos++] = '0' + (count % 10);
        result[pos++] = ch;
    }

    result[pos] = '\0';
}

int main() {
    char result[5000];
    countAndSay(5, result);
    cout << result << endl;
    return 0;
}
```

<u>Cum am putea să o întâlnim pe subiectul de admitere</u>

Fie algoritmul ***ceFace*(*n, result*),** unde ***n*** este un număr pozitiv ($0 < n < 20$), iar ***result*** este un şir de caractere cu o capacitate de 1024 de elemente inţializat cu 0. Funcţia ***copiază*(*dest, src*)**, copiază conţinutul şirului ***src*** în şirul de caractere ***dest***.

```
Algorithm countAndSay(n, result):
    prev[0] ← '1'
    size ← 1

    For i ← 1 to n-1 execute   // Linia *
        copiaza(result, prev)  // Linia **
        pos ← 0
        j ← 0

        While j < size execute
            ch ← prev[j]
            count ← 0

            While j < size AND prev[j] = ch execute
                j ← j + 1
                count ← count + 1
            EndWhile

            result[pos] ← '0' + count
            pos ← pos + 1
            result[pos] ← ch
            pos ← pos + 1
        EndWhile

        size ← pos
    EndFor

    Return size
EndAlgorithm
```

Selectaţi răspunsurile adevărate după apelul algoritmului ***countAndSay*:**

A. Dacă n = 3, șirul result va conține elementele 2 și 1 (în această ordine).
B. Dacă n = 1, bucla For nu se execută niciodată.
C. Dacă șirul ***prev*** (la linia marcată cu `// Linia **`) conține doar caractere identice consecutive, atunci după execuţia buclei While, variabila ***size*** va fi exact 2.
D. Există o valoare a lui ***n*** pentru care result de la pasul ***i*** este identic cu result de la pasul ***i-1*** pentru bucla ***for*** marcată cu `// Linia *`.

Problema 3

Enunț

Se dă un șir de numere naturale care reprezintă înălțimile unor obstacole așezate unul lângă altul. Scrieți un algoritm care calculează cantitatea totală de apă ce poate fi reținută între obstacole după o ploaie.

Exemple

Date de intrare: {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1}

Date de ieșire: 6

Date de intrare: {0, 1, 0, 2, 1, 0, 3}

Date de ieșire: 4

Rezolvare

Apa se va strânge în "văi" între obstacolele înalte. Întrebarea este: **câtă apă se strânge deasupra fiecărei poziţii?**

Apa poate sta pe o poziţie ***i*** doar dacă există obstacole mai înalte atât în stânga cât și în dreapta ei. Nivelul apei deasupra unei poziţii este limitat de cel mai scund dintre cele două ziduri (stânga și dreapta):

```
apa deasupra poziției i = min(cea mai înaltă poziție din stânga, cea mai înaltă
poziție dreapta) – înălțimea poziției i
```

De exemplu, pentru poziţia 1:

Înălţimea maximă din stânga: 2
Înălţimea maximă din dreapta: 3
Înălţimea poziţiei curente (i=1): 1
Cantitatea apei care poate fi captată: min(3, 2) – 1 = 1

Deci, pentru fiecare poziţie ***i,*** avem nevoie de:

- ***leftMax*** cel mai înalt obstacol văzut de la stânga până la ***i***
- ***rightMax*** cel mai înalt obstacol văzut de la dreapta până la ***i***

Le calculăm **o singură dată** și le stocăm, ca să nu repetăm munca.

Tablou de intrare

{0, 1, 0, 2, 1, 0, 3}

0	1	0	2	1	0	3

Calculul lui ***leftMax***: mergem de la stânga la dreapta şi calculăm pentru fiecare element cel mai înalt vecin din stânga lui:

leftMax[i] = max(arr[i], leftMax[i-1])

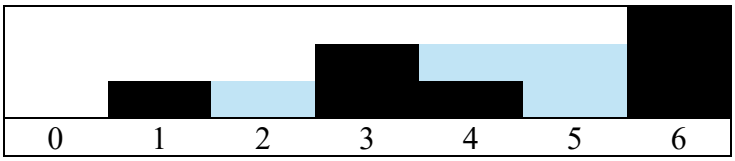
Tabloul de intrare						
0	1	0	2	1	0	3
leftMax						
0	1	1	2	2	2	3

Procedăm similar pentru calcului lui ***rightMax*****:** pornim de la dreapta la stânga şi aplicăm formula:

rightMax[i] = max(arr[i], rightMax [i+1])

Tabloul de intrare						
0	1	0	2	1	0	3
rightMax						
3	3	3	3	3	3	3

Date de intrare: {0, 1, 0, 2, 1, 0, 3}



Obstacol	**Înălțime**	**leftMax**	**rightMax**	**min(leftMax, rightMax)**	**Apă**
0	0	0	3	0	**0**
1	1	1	3	1	**0**
2	0	1	3	1	**1**
3	2	2	3	2	**0**
4	1	2	3	2	**1**
5	0	2	3	2	**2**
6	3	3	3	3	**0**

Cod C++

```
#include <iostream>
#include <cmath>
using namespace std;

#define MAX_ARR_SZ 100

int waterAmount(int arr[], int n) {
    if (n == 0) return 0;

    int leftMax[MAX_ARR_SZ], rightMax[MAX_ARR_SZ];

    leftMax[0] = arr[0];
    for (int i = 1; i < n; i++)
        leftMax[i] = max(leftMax[i - 1], arr[i]);

    rightMax[n - 1] = arr[n - 1];
    for (int i = n - 2; i >= 0; i--)
        rightMax[i] = max(rightMax[i + 1], arr[i]);
```

```
    int total = 0;
    for (int i = 0; i < n; i++)
        total += min(leftMax[i], rightMax[i]) - arr[i];

    return total;
}

int main() {
    int arr[] = { 0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Apa retinuta: " << waterAmount(arr, n) << endl;
    return 0;
}
```

Cum am putea să o întâlnim pe subiectul de admitere

Se dă algoritmul ***ceFace(arr, n)***, unde ***arr*** este un vector de numere pozitive de dimensiune ***n***, 0 < ***n*** < 200. Funcţia ***zeros***(n) returnează un tablou unidimensional cu n elemente, toate iniţializate cu 0.

```
Algorithm ceFace(arr, n):
    leftMax ← zeros(n)
    rightMax ← zeros(n)
    leftMax[0] ← arr[0]
    For i ← 1 to n - 1 execute
        If arr[i] > leftMax[i - 1] then
            leftMax[i] ← arr[i]
        Else
            leftMax[i] ← leftMax[i - 1]
        EndIf
    EndFor

    rightMax[n - 1] ← arr[n - 1]
    For i ← n - 2 to 0 execute
        If arr[i] > rightMax[i + 1] then
            rightMax[i] ← arr[i]
        Else
            rightMax[i] ← rightMax[i + 1]
        EndIf
    EndFor

    total ← 0
    For i ← 0 to n - 1 execute
        total ← total + min(leftMax[i], rightMax[i]) - arr[i]
    EndFor

    Return total
EndAlgorithm
```

Care dintre următoarele afirmaţii sunt adevărate?

A. În urma apelului algoritmului ***ceFace*** vectorii ***leftMax*** şi ***rightMax*** vor conţine elemente în ordine crescătoare.
B. În urma apelului ceFace([0, 3, 4, 0, 0, 2, 1, 0], 8) se va afişa valoarea 4.
C. Dacă tabloul de intrare are dimensiunea ***n*** conţine doar două valori de 1 şi restul sunt zero, valorea maximă care poate fi afişată este ***n-2***.
D. Dacă tabloul ***arr*** este sortat în ordine crescătoare, valoarea returnată de algoritm este întotdeauna 0.
E. Dacă tabloul ***arr*** are exact un singur element nenul, iar restul sunt 0, valoarea returnată depinde exclusiv de poziția acelui element în tablou.
F. Există un tablou ***arr*** pentru care ***leftMax*** și ***rightMax*** sunt identice element cu element.