# Tablouri unidimensionale

## Problema 1

Să se determine mulţimea cifrelor unui număr natural n > 0, dat.

- Exemplu: n=1723237 ⇒ Cifre = {1,2,3,7}

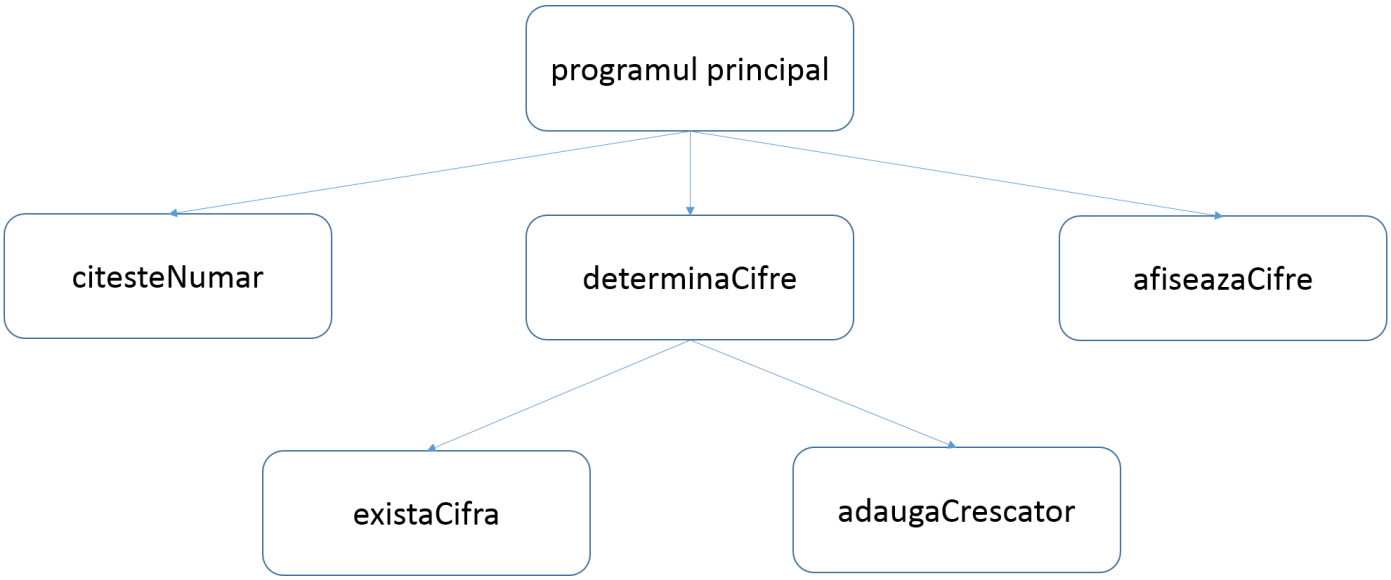
Se cere să se utilizeze subprograme care să comunice între ele şi cu programul principal prin parametri. Fiecare subprogram trebuie specificat.

Idee:

- folosim un vector Cifre în care punem pe rând cifrele lui n
- o cifră se adaugă în mulţimea Cifre dacă nu există deja în mulțime
- mulţimea Cifre conţine cifrele în ordine crescătoare; cifrele se inserează în așa fel încât în orice moment să fie sortate crescător
- vectorul Cifre va începe indexarea de la 1
- rezolvare iterativă & rezolvare recursivă

## Analiză

### Identificarea subalgoritmilor




## Cod sursă C++

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

/*     Citeste un numar natural n > 0.
Date: -
Rezultate: n - numarul natural nenul citit */
void citesteNumar(long &n){                              // citire n, transmis prin referinta
   do{
      cout<<"introduceti un numar natural > 0: ";
      cin>>n;
         if(n<=0)
        cout<<"Numarul citit trebuie sa fie natural si strict pozitiv. ";
   }while (n<=0);
}
```

```
/*     Cauta o valoare intre 0 si 9 într-un vector de valori intregi cuprinse intre 0 si 9.
Date: val - valoarea care se cauta, val e un numar intreg cuprins intre 0 si 9
      V - vectorul de valori intregi intre 0 si 9
      lungV - numar natural; reprezinta lungimea lui V
Rezultate: returnează true daca valoarea val apare in V, false altfel */
bool existaCifra(int val, int lungV, int V[]){
  int i = 1;
  while (i <= lungV && val != V[i])      // parcurgere V pana cand se gaseste val in V
    i++;
  if (i > lungV)
    return false;                        // sau val nu apare in V (i > lungV in acest caz)
  return true;                           // cazul in care val exista in V
}

/*     Adauga o valoare intreaga cuprinsa intre 0 si 9 intr-un vector, mentinand o ordine
crescatoare a valorilor din vector.
Date: val - valoarea care se adauga, val e un numar intreg cuprins intre 0 si 9
      V - vectorul de valori intregi intre 0 si 9
      lungV - numar natural; reprezinta lungimea lui V
Rezultate: lungV - incrementata cu o unitate
           V - vectorul actualizat, la valorile sale initiale s-a adaugat valoarea val */
void adaugaCrescator(int val, int &lungV, int V[]){
  int i = lungV;                         // V se parcurge de la sfarsit spre inceput
  while (i>0 && val<V[i]){               // daca val<V[i] se muta V[i] cu o pozitie
    V[i + 1] = V[i];                     // la dreapta
    i--;
  }
  V[i + 1] = val;                        // se adauga val pe pozitia corespunzatoare
  lungV++;                               // creste dimensiunea lui V
}

/*     Determina mulțimea cifrelor numarului n în vectorul V, de lungime lungV.
Date: n - un numar natural nenul
Rezultate: V - mulțimea cifrelor numarului n
           lungV - numar natural; reprezinta lungimea lui V */
void determinaCifre(long n, int &lungV, int V[]){
  while (n > 0){
      int uCifra = n % 10;                       //izolare ultima cifra, (n modulo 10)
      if (!existaCifra(uCifra, lungV, V))        //daca nu exista uCifra in V, trebuie adaugata
          adaugaCrescator(uCifra, lungV, V);
         n = n / 10;                             // eliminarea ultimei cifre a lui n
     }
}

/*     Afiseaza multimea cifrelor lui n
Date: n - un numar natural nenul
     Cifre - vectorul care retine multimea cifrelor, contine numere naturale intre 0 si 9, diferite
     intre ele
     lungCifre - un numar natural, reprezinta lungimea vectorului Cifre
Rezultate: - se afiseaza elementele din vectorul Cifre */
void afiseazaCifre(long n, int lungCifre, int Cifre[]){
  cout << n << " are multimea de cifre:";
  for (int i = 1; i <= lungCifre; i++)
    cout << Cifre[i] << " ";
}

int main(){
  long n;                                    // numarul n din enunt
  int Cifre[10];                             // multimea de cifre
  int lungCifre = 0;                         // cardinalul multimii Cifre, e vida initial
  citesteNumar(n);
  determinaCifre(n, lungCifre, Cifre);       // apel determinare multime Cifre
  afiseazaCifre(n, lungCifre, Cifre);        // apel afisare multime Cifre
  cout << endl << "Program terminat" << endl;
  return 0;
}
```

## Cod sursă C++ - variantă cu subalgoritmi recursivi

```
#include <iostream>
#include <iomanip>
using namespace std;
void citesteNumarRec(long &n){
  cout<<"introduceti un numar natural > 0: ";
  cin >> n;
  if (n <= 0)
  {
    cout<<"Numarul citit trebuie sa fie natural si strict pozitiv. ";
    citesteNumarRec(n);
  }
}

bool existaCifraRec(int val, int lungV, int V[]){  // cauta valoarea val in vectorul V;
  if (lungV > 0 && val != V[lungV])
    return existaCifraRec(val, lungV - 1, V);      // daca val != V[lungV], trecem la urmatorul
                                                   // element (recursiv)
  if (lungV == 0) return false;                    // val nu exista in V, cand lungV e 0
  /*else*/  return true;                           // daca val = V[lungV], returneaza true,
                                                   // intrucat val exista in V
}

void adaugaCrescatorRec(int val, int lungV, int V[]){  // adauga val in multimea V, astfel incat
                                                       // ordinea elementelor din multime sa fie
                                                       //crescatoare
  if (lungV>0 && val<V[lungV]){                        // daca val<V[lungV], se muta V[lungV] cu o
                                                       // pozitie la dreapta
     V[lungV + 1] = V[lungV];
     adaugaCrescatorRec(val, lungV - 1, V);            // V se parcurge de la sfarsit spre inceput
  }
  else V[lungV + 1] = val;                             // se adauga val pe pozitia corespunzatoare
}

void determinaCifreRec(long n, int &lungV, int V[]){   // determinare multime a cifrelor V
    if (n>0) {
       int uCifra = n % 10;                      // izolare ultima cifra (uCifra), (n modulo 10)
       if (!existaCifraRec(uCifra, lungV, V)){  // daca nu exista uCifra in V, trebuie adaugata;
             adaugaCrescatorRec(uCifra, lungV, V);
             lungV++;
          }                                      // creste lungimea lui V;
       determinaCifreRec(n / 10, lungV, V);      // apel recursiv cu stergerea ultimei cifre
                                                 // a lui n
    }
}

void afiseazaCifreRec(long  n, int lungCifre, int Cifre[]){   // afisare multime cifre recursiv
  if (lungCifre>0)  {
    afiseazaCifreRec(n, lungCifre - 1, Cifre);
    cout << " " << Cifre[lungCifre];
  }
  else
    cout << n << " are multimea de cifre:";
}

int main(){
  long n;                                    // numarul n din enunt;
  int Cifre[10];                             // multimea de cifre;
  int lungCifre = 0;                         // cardinalul multimii Cifre, e vida initial;
  citesteNumarRec(n);
  determinaCifreRec(n, lungCifre, Cifre);    // apel determinare multime Cifre
  afiseazaCifreRec(n, lungCifre, Cifre);     // apel afisare multime Cifre;
  cout << endl << "Program terminat" << endl;
  return 0;
}
```

## Exemple

| Date de intrare | Rezultate |
|---|---|
| 1723237 | 1, 2, 3, 7 |
| 9834758 | 3, 4, 5, 7, 8, 9 |
| 3333 | 3 |

# Problema 2

Produs maxim

Se consideră un șir X cu n ($3 \leq n \leq 10\ 000$) elemente numere întregi mai mari decât -30 000 și mai mici decât 30 000.

Scrieți un subalgoritm care determină trei elemente din șirul X al căror produs este maxim. Parametrii de intrare ai subalgoritmului sunt n și X, iar cei de ieșire vor fi a, b și c, reprezentând trei elemente din șirul X, având proprietatea cerută. Dacă problema are mai multe soluții, determinați una singură.

Se cer trei subalgoritmi cu complexitățile: $O(n^3)$, $O(n^2)$, $O(n)$.

- Exemplu: dacă n = 10 și X = (3, -5, 0, 5, 2, -1, 0, 1, 6, 8), cele trei numere sunt: a = 5, b = 6, c = 8.

Idei:

1. "forța brută"
   - parcurgem vectorul cu trei cicluri for imbricate, se generează toate tripletele ($x_i$, $x_j$, $x_k$) distincte ca indici
   - verificăm toate produsele posibile
   - complexitate: $T(n) = O(n^3)$
2. sortare
   - se sortează șirul
   - soluția conține sau (cele mai mici două numere și maximul) sau (cele mai mari trei numere); soluția va conține întotdeauna maximul șirului
   - complexitate: T(n) = complexitatea algoritmului de sortare
   - utilizăm sortarea prin selecție $\Rightarrow T(n) = O(n^2)$
3. se determină primele două minime și primele trei maxime utilizând un subalgoritm similar cu sortarea prin selecție, dar de complexitate liniară
   - soluția conține sau (cele mai mici două numere și maximul) sau (cele mai mari trei numere)
   - complexitate: T(n) = O(n)

## Cod sursă C++ pentru subalgoritm

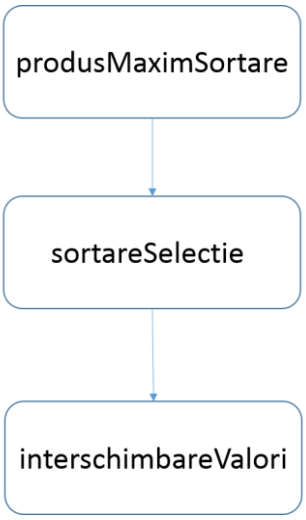
**1. Versiunea 1 ($T(n) = O(n^3)$)**

```
//Descriere: determină trei numere din șirul X, astfel încât produsul lor să fie maxim
//Date: X - vector, X = (xi|i = 1..n, xi ϵ Z, xi ϵ [-30000, 30000])
n ϵ N - lungimea vectorului X, 3 ≤ n ≤ 10 000
//Rezultate: a, b, c ϵ Z - trei numere întregi din vectorul X, astfel încât produsul lor să fie
maxim
void produsMaxim(int n, int X[], int& a, int& b, int& c){            //forta bruta cu O(n^3)
	int  i, j, k;
	a = X[1];
	b = X[2];
	c = X[3];
	for (i = 1; i <= n - 2; i++){
		for (j = i + 1; j <= n - 1; j++){
			for (k = j + 1; k <= n; k++){
				if (((long(X[i]))*X[j] * X[k]) > ((long(a))*b*c))	{
					a = X[i];
					b = X[j];
					c = X[k];
				}
			}
		}
	}
}
```

**2. Versiunea 2 (T(n) = O($n^2$))**

## Analiză

### Identificarea subalgoritmilor



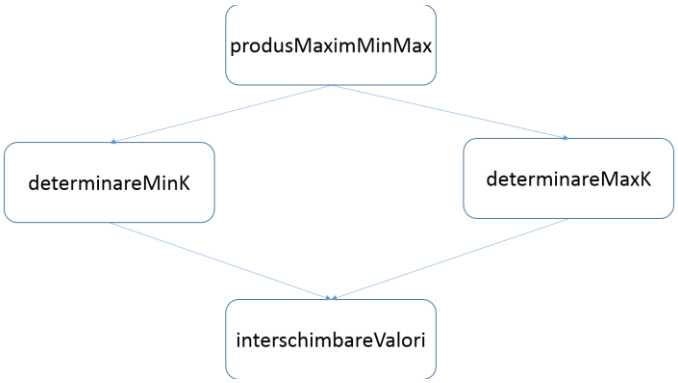

```
/*Descriere: determină trei numere din șirul X, astfel încât produsul lor să fie maxim
Date:  X - vector, X = (xi|i = 1..n, xi ∈ Z, xi ∈ [-30000, 30000])
       n ∈ N - lungimea vectorului X, 3 ≤ n ≤ 10 000
Rezultate: a, b, c ∈ Z - trei numere întregi din vectorul X, astfel încât produsul lor să fie
            maxim*/
void produsMaximSortare(int n, int X[], int& a, int& b, int& c){
       sortareSelectie(n, X);
       c = X[n];
       if (((long)X[1]) * X[2] * X[n] > ((long)X[n - 2]) * X[n - 1] * X[n]){
              a = X[1];
              b = X[2];
       }
       else{
              a = X[n - 2];
              b = X[n - 1];
       }
}

/*Descriere: se sortează crescător șirul X prin metoda selecției
Date:  X - vector, x = (xi|i = 1..n, xi ∈ Z, xi ∈ [-30000, 30000])
       n ∈ N - lungimea vectorului X, 3 ≤ n ≤ 10 000
Rezultate: vectorul X sortat crescător*/
void sortareSelectie(int n, int X[]){
       int pozMin, aux;
       for (int i = 1; i < n; i++){
              pozMin = i;
              for (int j = i + 1; j <= n; j++){
                     if (X[j] < X[pozMin])
                            pozMin = j;
              }
              if (pozMin != i)
                     interschimbareValori(X[i], X[pozMin])
       }
}

/*Descriere: interschimbă valorile a și b
Date: a, b ∈ Z
Rezultate: a, b actualizate; a conține vechea valoare a lui b, iar b vechea valoarea a lui a*/
void  interschimbareValori(int& a, int& b){
       int aux;
       aux = a;
```

```
	a = b;
	b = aux;
}
```

### 3. Versiunea 3 (T(n) = O(n))

## Analiză

### Identificarea subalgoritmilor




```
/* Descriere: determină trei numere din șirul X, astfel încât produsul lor să fie maxim
Date:  X - vector, X = (xi|i = 1..n, xi ϵ Z, xi ϵ [-30000, 30000])
	n ϵ N - lungimea vectorului X, 3 ≤ n ≤ 10 000
Rezultate: a, b, c ϵ Z - 3 numere întregi din vectorul X, astfel încât produsul lor să fie maxim*/
void produsMaximMinMax(int n, int X[], int& a, int& b, int& c){
	determinaMinK(n, X, 2);			//primele 2 minime pe primele pozitii;
	determinaMaxK(n, X, 3);			//primele 3 maxime, pe ultimele pozitii
	c = X[n];
	if (((long)X[1]) * X[2] * X[n] > ((long)X[n - 2]) * X[n - 1] * X[n]) {
		a = X[1];
		b = X[2];
	}
	else	{
		a = X[n - 2];
		b = X[n - 1];
	}
}

/* Descriere: determină primele k minime din șirul X folosind metoda sortării prin selecție și le
plasează pe primele k poziții în șir
Date:  X - vector, X = (xi|i = 1..n, xi ϵ Z, xi ϵ [-30000, 30000])
	n ϵ N - lungimea vectorului X
	k ϵ N - numărul de minime care se dorește a fi determinat
Rezultate: vectorul X actualizat: primele k poziții sunt ocupate, în ordine, de primele k minime din
vector
Obs. Pentru k = 2: T(n) = O(n)*/
void determinareMinK(int n, int X[], int k){	//determinare primelor k minime pe primele k pozitii
	for (int i = 1; i <= k; i++){		//pentru k=2, T(n)=O(n)
		int poz = i;
		for (int j = i + 1; j <= n; j++)
		if (X[j]<X[poz])
			poz = j;
		interschimba(X[i], X[poz]);
	}
}

/* Descriere: determină primele k maxime din șirul X folosind metoda sortării prin selecție și le
plasează pe ultimele k poziții în șir
Date:  X - vector, X = (xi|i = 1..n, xi ϵ Z, xi ϵ [-30000, 30000])
```

```
	n ∈ N - lungimea vectorului
	k ∈ N - numărul de maxime care se dorește a se determina
Rezultate: vectorul X actualizat: ultimele k poziții sunt ocupate, în ordine crescătoare, de primele
k maxime din vector
Obs. Pentru k = 3, T(n) = O(n)*/
void determinareMaxK(int n, int X[], int k){//determinare primelor k maxime pe ultimele k pozitii
	for (int i = n; i >= n - k + 1; i--){//pentru k=3 avem T(n)=O(n)
		int poz = i;
		for (int j = i - 1; j >= 1; j--){
			if (X[j]>X[poz]) poz = j;
			interschimba(X[i], X[poz]);
		}
	}
}

//void  interschimbareValori(int& x, int& y) - vezi solutia 2
```

## Exemple

| Date de intrare | | Rezultate |
|---|---|---|
| n | X | cele trei numere |
| 7 | -3, -100, -101, -2, -1, -45, -22 | -3, -2, -1 |
| 5 | -100, -99, -1, 1, 2 | -100, -99, 2 |
| 10 | 8, -7, 9, -4, 10, 6, -3, -1, 6, -2 | 8, 9, 10 |

# Problema 3

Monitorizare date Facebook

Veți implementa o aplicație care gestionează date agregate referitoare la activitatea de pe Facebook a utilizatorilor din România. Între altele, aplicația va permite:

1. citirea unui număr natural *K* între 1 și 100, a unui an de început *i* și a unui an de sfârșit *s* între 2004 și 2030, astfel încât $s > i + 2$;

2. citirea numărului de conturi și a numărului de pagini existente în rețea la începutul fiecărui an în România, din anul *i* până în anul *s*; numărul de conturi, respectiv cel de pagini dintr-un an, este un număr natural nenul;

3. determinarea și afișarea celui mai lung interval de timp în care o mărire a ratei de creștere anuale a numărului de conturi într-un an este urmată de o descreștere a ratei respective în anul următor și reciproc, o descreștere a ratei de creștere anuale a numărului de conturi într-un an este urmată de o creștere a ratei în anul următor; pentru intervalul găsit, se afișează și rata de creștere din fiecare an; în cazul în care în doi ani consecutivi rata de creștere stagnează, se consideră că proprietatea nu este îndeplinită;

- rata de creștere se calculează începând cu al doilea an în care se monitorizează numărul anual de conturi și este reprezentată de procentul cu care a crescut sau scăzut numărul de conturi față de anul precedent; de exemplu, dacă în 2008 sunt 50.000 de conturi, iar în 2009 sunt 300.000 de conturi, rata de creștere la începutul anului 2009 este de 500%, la începutul anului 2009 fiind cu 500% (cu 250.000, adică 250.000/50.000 * 100) mai multe conturi decât cele existente la începutul anului precedent; dacă în 2020 sunt 10.000.000 de conturi și în 2021 sunt 9.000.000 de conturi, rata de creștere la începutul anului 2021 este negativă și are valoarea -10%; rata de creștere este un număr real;
- cel mai lung interval de timp căutat are cel puțin trei ani consecutivi (o creștere într-un an față de anul precedent este urmată de o descreștere în anul următor sau invers) pentru rata de creștere (deci de patru ani pentru numărul de conturi);
- în cazul în care există mai multe soluții (mai multe intervale de timp care au aceeași lungime), se afișează una dintre ele;

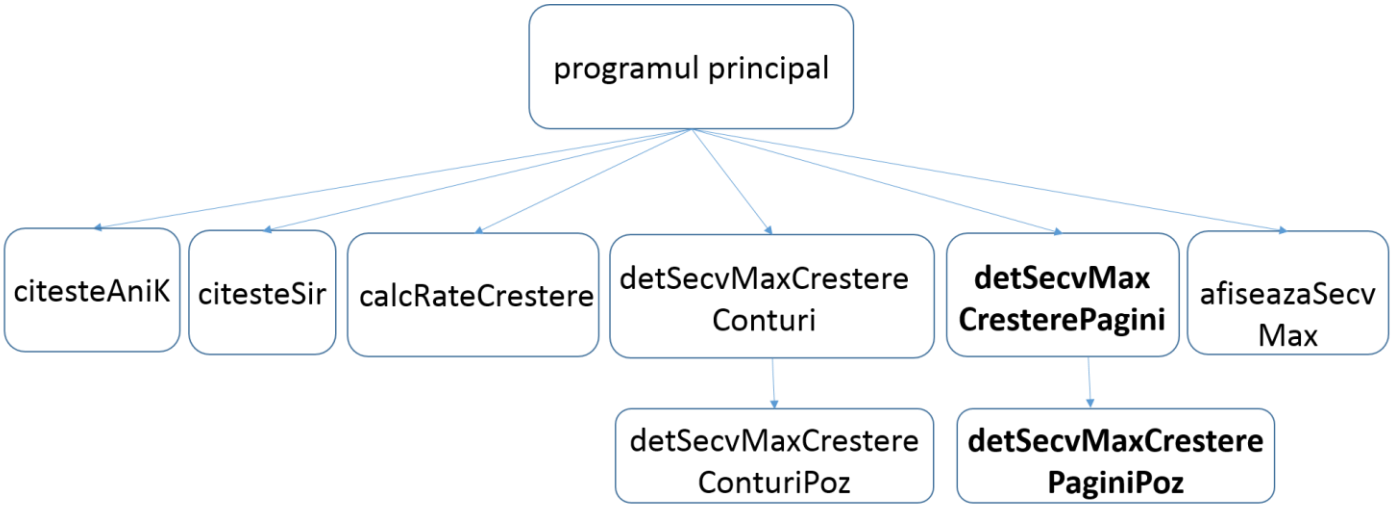
4. determinarea celei mai lungi perioade în care numărul de pagini create crește, de la un an la altul, cu cel puțin K% - **temă**.

Exemplul 1

- număr de conturi din 2008 până în 2017: 50.000, 300.000, 500.000, 2.000.000, 3.000.000, 5.000.000, 6.000.000, 8.000.000, 8.800.000, 9.000.000
- rata de creștere a numărului de conturi calculată la începutul fiecărui an relativ la anul anterior, din 2009 până în 2017: 500%, 66.67%, 300%, 50%, 66.67%, 20%, 33.33%, 10%, 2.27%
- cel mai lung interval de timp în care rata de creștere cunoaște scăderi și creșteri alternative, de la un an la altul, este 2009 - 2016, iar valorile ratei de creștere pentru intervalul respectiv sunt: 500%, 66.67%, 300%, 50%, 66.67%, 20%, 33.33%, 10%.

Este necesară utilizarea subprogramelor care comunică între ele și cu programul principal, precum și specificarea acestora.

## Analiză
### Identificarea subalgoritmilor




## Cod sursă C++

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


/* Descriere: Citește anul de la care începe analiza, anul în care se încheie analiza și procentul
K. Perioada analizată trebuie să cuprindă cel puțin patru ani consecutivi.
Date: -
Rezultate: anInceput, anSfarsit, K
	anInceput - anul de la care începe analiza, anInceput >= 2004 AND anInceput <=
	2030
	anSfarsit - anul în care se sfârșește analiza, anSfarsit >= 2004 AND
	anSfarsit <= 2030; anSfarsit > anInceput + 2
	K - procent utilizat în problemă, K > = 1 AND K <= 100 */
void citesteAniK(int& anInceput, int& anSfarsit, int& K){
	do	{
		cout << "Introduceti anul de inceput: ";
		cin >> anInceput;
	} while (anInceput < 2004 || anInceput > 2030);
	do{
		cout << "Introduceti anul de sfarsit: ";
		cin >> anSfarsit;
	} while (anSfarsit < 2004 || anSfarsit > 2030 || anSfarsit <= anInceput + 2);
	do{
		cout << "Introduceti valoarea procentului K: ";
		cin >> K;
	} while (K <= 0 || K > 100);
}

/* Descriere:  Citește un șir de numere naturale nenule, câte unul pentru fiecare an din perioada
determinată de anInceput și anSfarsit; calculează lungimea șirului
Date: anInceput - primul an pentru care se citește un număr natural nenul
	anSfarsit - ultimul an pentru care se citește un număr natural nenul
	tipDateSir - semnificația unui număr natural din șir: număr de pagini de Facebook
	sau număr de conturi de Facebook
Rezultate: sir - șir de numere naturale nenule, câte unul pentru fiecare an aflat în
```

```
	intervalul determinat de anInceput și anSfarsit
	lungSir - lungimea șirului de numere citite, lungSir >= 4 */
void citesteSir(int sir[], int anInceput, int anSfarsit, int& lungSir, char tipDateSir[20]){
	lungSir = anSfarsit - anInceput + 1; // se citeste un sir de numere naturale nenule
	for (int i = 1; i <= lungSir; i++)
	{
		do
		{
			cout << "Numarul de " << tipDateSir << " de Facebook in anul " << anInceput + i
- 1 << ": ";
			cin >> sir[i];
		} while (sir[i] <= 0);
	}
}


/* Descriere:  Pentru un șir de numere naturale nenule, calculează cu ce procent este mai mare sau
mai mic un număr din șir față de predecesorul sau, începând cu al doilea număr din șir.
Date: sir - un șir de numere naturale nenule
	lungSir -  lungimea șirului de numere pentru care se calculează procentele de
	creștere, lungSir >= 4
Rezultate: sirCrestere - șir de numere reale în care sirCrestere[i] reprezintă
	procentul cu care sir[i+1] este mai mare sau mai mic decât sir[i], pentru i
	între 1 și lungSir - 1
	lungSirCrestere - lungimea șirului de rate de creștere
	lungSirCrestere = lungSir - 1, lungSirCrestere >= 3 */
void calcRateCrestere(int sir[], int lungSir, double sirCrestere[], int& lungSirCrestere){
	int i;
	for (i = 1; i < lungSir; i++)
		sirCrestere[i] = (sir[i + 1] - sir[i]) * 100.0 / sir[i];
	lungSirCrestere = lungSir - 1;
}

/* Descriere: Calculează lungimea secvenței de rate de creștere care cresc și scad alternativ, de la
un an la altul, relativ la o poziție dată.
Date: crestereConturi - șir de numere reale, reprezentând fiecare câte o rată de creștere
	anuală a numărului de conturi
	lungCrestereC - lungimea șirului de rate de creștere anuale ale numărului de conturi
	lungCrestereC >= 3
	poz - reprezintă poziția din șir începând de la care se verifică proprietatea
	cerută: dacă este o scădere față de rata de creștere anterioară și este urmată de
	o creștere sau invers, dacă reprezintă o mărire față de rata de creștere anterioară și este
	urmată de o scădere; poz >= 2
Rezultate: lungSecv - lungimea secvenței de rate care scad și cresc alternativ relativ
	la poziția dată; dacă ultima rată de creștere care îndeplinește proprietatea
	cerută se află la poziția poz + i, atunci lungSecv va avea valoarea
	(poz + i) - poz + 3 */
void detSecvMaxCrestereConturiPoz(double crestereConturi[], int lungCrestereC, int poz, int&
lungSecv){
	lungSecv = 2;
	int i = poz;
	while (i < lungCrestereC && (crestereConturi[i] - crestereConturi[i - 1]) *
(crestereConturi[i + 1] - crestereConturi[i]) < 0)
		i++;
	lungSecv = i - poz + 2;
}


/* Descriere:  Calculează cea mai lungă secvență de rate de creștere anuale ale numărului de
conturi, care cresc și scad alternativ, de la un an la altul.
Date: crestereConturi - șir de numere reale, reprezentând fiecare câte o rată de creștere
	anuală a numărului de conturi
	lungCrestereC - numar natural, reprezinta lungimea șirului de rate de creștere
	anuale ale numărului de conturi, lungCrestereC >= 3
Rezultate: poz - pentru cea mai lungă secvență de rate de creștere care cresc și scad
	alternativ, de la un an la altul, poz reprezintă poziția din șir a primei rate
```

```
	de creștere care respectă proprietatea cerută: reprezintă o scădere față de
	rata de creștere anterioară și este urmată de o creștere sau invers, reprezintă
	o mărire față de rata de creștere anterioară și este urmată de o scădere
	lungSecvMax - lungimea celei mai lungi secvențe de rate de creștere care cresc
	și scad alternativ din șirul dat; reprezintă numărul de rate de creștere din
	secvență */
void detSecvMaxCrestereConturi(double crestereConturi[], int lungCrestereC, int& lungSecvMax, int&
poz){
	int i = 2;
	while (i <= lungCrestereC - 1){
		int lungSecv = 2;
		detSecvMaxCrestereConturiPoz(crestereConturi, lungCrestereC, i, lungSecv);
		if (lungSecv > lungSecvMax){
			lungSecvMax = lungSecv;
			poz = i;
			i += lungSecv - 2;
		}
		else
			i++;
	}
}

/* Descriere: Afișează cel mai lung interval de timp în care ratele de creștere cresc și scad
alternativ, de la un an la altul, precum și ratele de creștere respective.
Date: crestereConturi - șir de numere reale, reprezentând fiecare câte o rată de creștere
	anuală a numărului de conturi
	lungCrestereC - lungimea șirului de rate de creștere anuale ale numărului de conturi
	lungCrestereC >= 3
	inceputInterval - poziția începând de la care trebuie afișate ratele de creștere
	din cea mai lungă secvență din șir; rata de creștere aflată pe poziția
	inceputInterval + 1 este prima valoare care respectă proprietatea cerută: reprezintă
	o scădere față de rata de creștere de la poziția inceputInterval și este urmată de
	o creștere sau invers, reprezintă o mărire față de rata de creștere  de la poziția
	inceputInterval și este urmată de o scădere (pe pozitia inceputInterval+2)
	lungSecv - lungimea secvenței de rate de creștere care trebuie afișată
	lungSecv >= 3
	anInceput - anul începând cu care se afișează ratele de creștere
Rezultate: se afișează cel mai lung interval de timp în care ratele de creștere cresc și
	scad alternativ, de la un an la altul, precum și ratele de creștere respective */
void afiseazaSecvMax(double crestereConturi[], int lungCrestereC, int inceputInterval, int
lungimeSecventa, int anInceput){
	if (inceputInterval == -1){
		cout << "Nu exista niciun interval de timp cu proprietatea dorita" << endl;
		return;
	}
	int anIncInterval = anInceput + inceputInterval;
	int anSfInterval = anIncInterval + lungimeSecventa - 1;
	cout << "Cel mai lung interval in care ratele de crestere au crescut si scazut alternativ
este: " << anIncInterval << " - " << anSfInterval << endl;
	int i = inceputInterval;
	cout << "Valorile ratei de crestere in intervalul mentionat sunt: " << endl;
	while (i <= inceputInterval + lungimeSecventa - 1){
		cout << crestereConturi[i] << "  ";
		i++;
	}
}

int main(){
	int conturi[100];
	double crestereConturi[100];
	int lungC, anInceput, anSfarsit, k, lungCrestereC, inceputInterval, lungSecv, poz;

	lungC = 0;
	anInceput = 0, anSfarsit = 0, k = 0; //citire param, conturi
	citesteAniK(anInceput, anSfarsit, k);
	citesteSir(conturi, anInceput, anSfarsit, lungC, "conturi");
```

```
        //calcul rate crestere conturi
        lungCrestereC = 0;
        calcRateCrestere(conturi, lungC, crestereConturi, lungCrestereC);
        // identificam cea mai lunga secventa in care ratele
        // de crestere cresc si scad alternativ, de la un an la altul
        lungSecv = 2;
        poz = 0;
        detSecvMaxCrestereConturi(crestereConturi, lungCrestereC, lungSecv, poz);
        inceputInterval = poz - 1;
        afiseazaSecvMax(crestereConturi, lungCrestereC, inceputInterval, lungSecv, anInceput);
}
```

## Exemple

| Date de intrare | | | Rezultate |
|---|---|---|---|
| An inceput | An sfarsit | Sir conturi Facebook | |
| 2020 | 2023 | 6.000.000 4.000.000 2.000.000 1.000.000 | Nu exista niciun interval de timp cu proprietatea dorita. |
| 2008 | 2017 | 50.000, 300.000, 500.000, 2.000.000, 3.000.000, 5.000.000, 6.000.000, 8.000.000, 8.800.000, 9.000.000 | Cel mai lung interval in care ratele de crestere au crescut si scazut alternativ, de la un an la altul, este: 2009 - 2016.<br>Cel mai lung interval in care ratele de crestere au crescut si scazut alternativ, de la un an la altul, este: 500%, 66.67%, 300%, 50%, 66.67%, 20%, 33.33%, 10%. |
| 2021 | 2030 | 10.000, 40.000, 120.000, 480.000, 2.400.000, 2.640.000, 3.168.000, 3.484.800, 4.181.760, 5.436.288 | Cel mai lung interval in care ratele de crestere au crescut si scazut alternativ, de la un an la altul, este: 2024 - 2029.<br>Valorile ratei de crestere in intervalul mentionat sunt: 300%, 400%, 10%, 20%, 10%, 20%. |
| 2005 | 2010 | 55.000, 110.000, 132.000, 171.600, 188.760, 226512 | Cel mai lung interval in care ratele de crestere au crescut si scazut alternativ, de la un an la altul, este: 2006 - 2010.<br>Valorile ratei de crestere in intervalul mentionat sunt: 100%, 20%, 30%, 10%, 20%. |
| 2007 | 2013 | 1.000.000, 1.100.000, 1.320.000, 1.452.000, 1.597.200, 1.916.640, 2.108.304 | Cel mai lung interval in care ratele de crestere au crescut si scazut alternativ, de la un an la altul, este: 2008 - 2010.<br>Valorile ratei de crestere in intervalul mentionat sunt: 10%, 20%, 10%. |