Recursivitate. Dividae et Impera.

1. Obiectivele lucrării

În această lucrare se va studia *recursivitatea* şi metoda *dividae et impera*.

2. Breviar teoretic

O **funcție recursivă** este o funcție care se apelează pe ea însăşi. La fiecare apel al funcției, parametrii funcției şi variabilele locale ei (cele care nu sunt statice) sunt alocate pe stivă (la fiecare apel în zone diferite de memorie). La revenirea din apelul funcției recursive, are loc descărcarea stivei.

Funcțiile recursive au avantajul că permit o exprimare uşoară, clară, a relațiilor recurente. Dezavantajul lor este că determină mărirea timpului de execuție a programului, față de varianta nerecursivă (datorită timpului consumat cu încărcarea/descărcarea stivei).

Metoda **dividae et impera** se bazează pe descompunerea problemei inițiale în două sau mai multe subprobleme mult mai uşor de rezolvat, iar soluția pentru problema inițială se obține combinând soluțiile subproblemelor în care a fost descompusă. Subproblemele în care a fost descompusă problema inițială, pot fi descompuse şi ele la rândul lor, în alte subprobleme. Descompunerea poate continua până se ajunge la subprobleme care admit o rezolvare imediată. Soluția problemei se obține prin combinarea soluțiilor problemelor parțiale.

Soluțiile obținute prin această metodă sunt mai rapide deoarece dimensiunea problemei scade în progresie geometrică.

Metoda dividae et impera se implementează de obicei recursiv, deoarece soluția recursivă este mai clară decât cea nerecursivă.

Probleme tipice care se rezolvă prin această metodă, sunt:

- ✓ sortarea prin interclasare a unui vector (merge-sort),
- ✓ căutarea binară,
- ✓ problema turnurilor din Hanoi,
- ✓ aflare maxim/minim vector,
- ✓ etc.

3. Probleme rezolvate

Se vor edita şi apoi executa programele descrise în continuare.

1. Să se realizeze algoritmul de descompunerea unui număr ca produs de factori primi:
Ex: **nr = 24**, se va afişa: **2 2 2 3**

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
void afisareFactor(int nr, int factor);
void main()
{
 int nr;
 clrscr();
 printf("nr=");
 scanf("%d",&nr);
 afisareFactor(nr,2);
 getch();
}
void afisareFactor(int nr, int factor)
{
 //limita:
 if(nr==1)return;
 if(nr%factor==0){
  printf("\n%d",factor);
  //reduc:
  nr=nr/factor;
  afisareFactor(nr,factor);
 }
 else afisareFactor(nr,factor+1);
}
```

2. Să se scrie o funcție recursivă ce calculează şi returnează maximul dintr-un vector.

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
#define N 5 //numarul de elemente din vector
int maxim(int A[], int n);

void main()
{
 int A[N]={1, 5, 20, 1, 3};
```

```
 int max;
 clrscr();
 max=maxim(A,N);
 printf("maxim=%d",max);
}

int maxim(int A[], int n)
{
 /*Algoritm:
daca vectorul are o singura componenta: returneaza
A[0] altfel:
      1)calculeaza max1: maximul din vectorul redus
(fara ultima componenta)
      2)returneaza maximul dintre max1 si ultima
componenta
  */
  int max1;
  if(n==1)return A[0];
  else{
        max1=maxim(A,n-1);
        if(max1>A[n-1])return max1;
        else return A[n-1];
  }
}
```

3. Să se sorteze în ordine crescătoare, folosind **metoda selecției maximului**, un vector de N componente, numere întregi.

Astfel, se va calcula maximul dintre cele N numere. Acesta se va muta de pe poziția lui, pe ultima poziție (poziția N-1), comutând cele două numere între ele. Apoi, se calculează maximul dintre primele N-1 numere, şi acesta se comută cu numărul de pe penultima poziție (poziția N-2), ş.a.m.d.

Exemplu:

N=5

A={17, 5, 0, 9, 4}

Pasul 1: maxim1=17. După comutare: A={4, 5, 0, 9, 17}

Pasul 2: maxim2=9. După comutare: A={4, 5, 0, 9, 17}

Pasul 3: maxim3=5. După comutare: A={4, 0, 5, 9, 17}

Pasul 4: maxim4=4. După comutare: A={0, 4, 5, 9, 17}

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
```

```
#define N 5 //numarul de elemente din vector
void ordonare(int A[], int dim);//prototipul
//functiei recursive.
//functia getMax() afla maximul si pozitia lui,
//intr-un vector ce are dim componente:
void getMax(int A[], int dim, int *pMax, int *
pIndex);

void main()
{
 int A[N]={17, 5, 0, 9, 4};
 int i;
 clrscr();
 ordonare(A,N);
 //Afisare vector ordonat:
 for(i=0;i<N;i++)
      printf("\n%d",A[i]);
}

void ordonare(int A[], int dim)
{
 int max;
 int index;
 /* Algoritm:
 daca dim=0 atunci este ordonat
 altfel:
1)comuta maximul (dintre cele dim elemente),cu
     elementul de pe ultima pozitie
2)ordoneaza ( apel recursiv) primele dim-1 elemente
 */
 if(dim==0)return;//este ordonat
 //afla maximul si indexul lui:
 getMax(A,dim,&max,&index);
 //comutarea maximului gasit (pe pozitia index), cu
elemental de pe ultima pozitie (pozitia dim-1):
 A[index]=A[dim-1];
 A[dim-1]=max;
 //se repeta procedeul pentru primele dim-1
//componente ale lui A:
 ordonare(A,dim-1);
}

void getMax(int A[], int dim, int *pMax, int *
pIndex)
```

```
{
 int max;
 int pozMax;//pozitia (indexul) maximului
 int i;
 max=A[0];
 pozMax=0;
 for(i=1;i<dim;i++)
        if(A[i]>max){
             max=A[i];
             pozMax=i;}
 *pMax=max;
 *pIndex=pozMax;
}
```

4. Să se realizeze algoritmul pentru căutarea maximului dintr-un vector folosind metoda dividae et impera.

Sursa programului:

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>

#define N 5

int max(int a[], int st, int dr)
{
      if(st==dr)
            return a[st];
      //descompunem problema in doua subprobleme
      int m = (st+dr)/2;
      //rezolvam fiecare subproblema
      int max1 = max(a, st, m);
      int max2 = max(a, m+1, dr);
      //combinam solutiile
      if(max1>max2)
            return max1;
      return max2;
}

void main()
{
      //presupunem ca vectorul este deja sortat
      int a[N]={1,3,4,7,12};//vectorul sortat
crescator
```

```
	//cautare maxim in vector
	cout<<"Maximul din vector este:" << max(a, 0,
N-1);

	getch();
}
```

5. Să se realizeze algoritmul pentru căutarea binară a unui număr oarecare x, într-un vector sortat crescător, folosind metoda dividae et impera.

Observație:

Pentru **căutarea liniară** sunt, în medie, **N/2 cazuri (comparații)**, ordinul de mărime **O(N)**, iar pentru **căutarea binară** ordinul de mărime este $\log_2 N$.

De exemplu, pentru 1000 de numere:

→ căutarea liniară necesită, în medie, 500 comparații;

→ căutarea binară necesită, în medie, 10 comparații.

Sursa programului:

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>

#define N 5

int cautareBinara(int a[], int st, int dr, int x)
{
	if(dr<st)
		return 0;
	if(st==dr)
		if(a[st]==x)
			return 1;
		else
			return 0;
	int m = (st + dr)/2;
	if(a[m]==x)
		return 1;
	if(a[m]<x)
		return cautareBinara(a, m+1, dr, x);
	return cautareBinara(a, st, m-1, x);
}

void main()
```

```
{
	//presupunem ca vectorul este deja sortat
	int a[N]={1,3,4,7,12};//vectorul sortat
//crescator
	int x; //numarul cautat
	cout<<"x=";
	cin>>x;
	int rez = cautareBinara(a, 0, N-1, x);
	if(rez==0)
		cout<<"Numarul cautat nu exista in
vector!";
	else
		cout<<"Numarul cautat exista in
vector!";
	getch();
}
```

6. Să se determine cmmdc-ul a N numere utilizând tehnica dividae et impera.

Algoritm:

- ✓ Memorăm numerele într-un vector;
- ✓ Împărțim vectorul în două jumătăți;
- ✓ Calculăm cmmdc-ul din fiecare jumătate;
- ✓ Combinăm soluțiile.

Sursa programului:

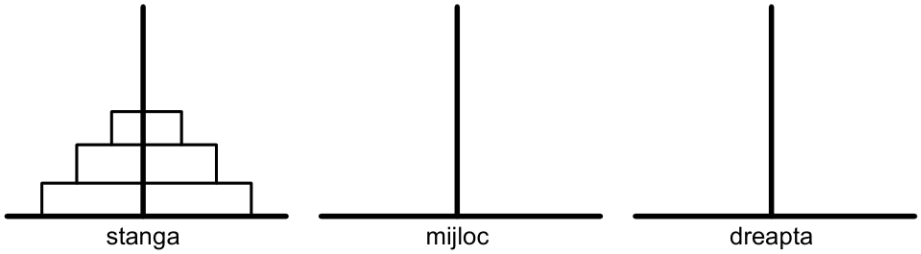
```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>

#define N 5

int cmmdc_2nr(int a, int b)
{
	while(a!=b)
	{
		if(a==b)
			return a;
		else if(a>b)
			a=a-b;
		else
			b=b-a;
	}
}
```

```
int cmmdc(int a[], int st, int dr)
{
      if(st==dr)
            return a[st];
      //descompunem problema in doua subprobleme
      int m = (st+dr)/2;
      //rezolvam fiecare subproblema
      int nr1 = cmmdc(a, st, m);
      int nr2 = cmmdc(a, m+1, dr);
      //combinam solutiile
      return cmmdc_2nr(nr1, nr2);
}

void main()
{
      int a[N]={1,3,4,7,12};//vectorul ce contine
//cele N numere
      int rez_cmmdc = cmmdc(a, 0, N-1);

      //afisare rezultat
      cout<<"Cmmdc-ul numerelor din vector este:" <<
rez_cmmdc;
      getch();
}
```

4. Probleme propuse

1. *Să se implementeze o funcţie recursivă care să calculeze termenul de rang n din şirul lui Fibonacci.*
   Şirul Fibonacci se defineşte astfel:
   $a_0 = 1$
   $a_1 = 1$
   pentru $n > 1 \rightarrow a_n = a_{n-1} + a_{n-2}$
2. *Să se scrie algoritmul pentru calculul sumei elementelor unui vector folosind metoda dividae et impera.*
3. *Să se realizeze algoritmul pentru căutarea minimului dintr-un vector folosind metoda dividae et impera.*
4. *Să se realizeze un program C++ care rezolvă problema turnurilor din Hanoi. Se dau trei tije (stânga, mijloc, dreapta) şi n discuri de diferite dimensiuni, aranjate pe tija stânga în*

ordine descrescătoare a dimensiunilor lor, formând un turn ca în figura de mai jos.



Turnurile din Hanoi

Cele n discuri de pe tija din stânga trebuie mutate pe tija din dreapta astfel încât acestea să fie ordonate ca la început. Mutările se fac cu următoarele restricţii:

- *La fiecare mişcare se poate muta doar un disc.*
- *Un disc cu diametrul mai mare nu poate fi mutat peste unul cu diametrul mai mic.*
- *Tija din mijloc poate fi utilizată ca tijă intermediară.*