**PROIECT IDIVIDUAL**

**LA INFORMATICĂ**

**TEMA: Metoda desparte și stăpînește**

**A REALIZAT:Stoianov Victor, clasa a XI-a ”C”**

**A VERIFICAT: Maria Guțu**

**IPLT,,Spiru Haret”**

**Informaţie generală:**

Metoda desparte şi stăpîneşte (în latină divide et impera) este o metodă generală de elaborare a algoritmilor, care presupune:

1) împărţirea repetată a unei probleme de dimensiuni mari în două sau mai multe subprobleme de acelaşi tip, dar de dimensiuni mai mici;

2) rezolvarea subproblemelor în mod direct, dacă dimensiunea lor permite aceasta, sau împărţirea lor în alte subprobleme de dimensiuni şi mai mici;

3) combinarea soluţiilor subproblemelor rezolvate pentru a obţine soluţia problemei iniţiale.

În limbaj matematic, admitem că la un anumit pas al algoritmului se dă o mulţime ordonată

**A = (ai , ai+1, ..., aj** ) şi că trebuie efectuată o prelucrare oarecare asupra elementelor sale. Pentru a împărţi problema curentă în două subprobleme de aproximativ aceleaşi dimensiuni, stabilim **m = (j – i) div 2** şi împărţim mulţimea A în două submulţimi care vor fi prelucrate separat: **A1 = (ai , ai+1, ..., ai+m); A2 = (ai+m+1, ai+m+2, ..., aj ).**

În continuare, mulţimile A1 şi A2 se împart din nou în cîte două submulţimi, respectiv A1-1, A1-2 şi A2-1, A2-2. Acest proces va continua pînă cînd soluţiile ce corespund submulţimilor curente vor putea fi calculate în mod direct. Pentru exemplificare, în fi gura 5.6 este prezentat modul de împărţire a mulţimii **A=(a1, a2, ..., a7)** în cazul divizării problemelor curente în cîte două subprobleme de acelaşi tip.

Schema generală a unui algoritm bazat pe metoda desparte şi stăpîneşte poate fi redată cu ajutorul unei proceduri recursive:

**procedure** DesparteSiStapineste(i, j : integer; **var** x : tip);

**var** m : integer;

x1, x2 : tip;

**begin**

**if** SolutieDirecta(i, j) **then** Prelucrare(i, j, x)

**else**

**begin**

m:=(j-i) **div** 2;

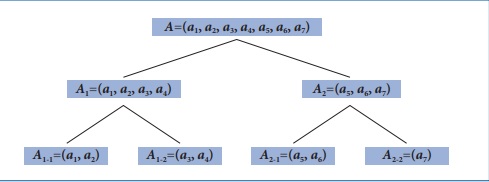
DesparteSiStapineste(i, i+m, x1);

DesparteSiStapineste(i+m+1, j, x2);

Combina(x1, x2, x);

**end;**

**end**;



**Descompunerea mulțimii A în submulțimi**

Metoda **desparte și stăpîneșe** admite o implementare recursiva ,deorece subproblemele sunt similare   problemei  initiale, dar    de dimensiuni mai mici .

Principiul fundamental al recursivitatii este autoapelarea unui subprogram cand acesta este activ;ceea ce se intampla la un nivel ,se intampla la orice nivel ,avand grija sa asiguram conditia de terminare ale apelurilor repetate .Asemanator se intampla si in cazul metodei **desparte și stăpîneșe**; la un anumit nivel sunt doua posibilitati :

s-a ajuns la o (sub)problema simpla ce admite o rezolvare imediata caz in care se rezolva (sub)problema si se revine din apel (la subproblema anterioara,de dimensiuni mai mari);

s-a ajuns la o  (sub)problema care nu admite o rezolvare imediata ,caz in care o descompunem in doua sau mai multe subprobleme si pentru fiecare din ele se continua apelurile recursive(ale procedurii sau functiei).

In etapa finala a metodei **desparte și stăpîneșe**  se produce combinarea subproblemelor (rezolvate deja) prin secventele de revenire din apelurile recursive.

Etapele metodei   **desparte și stăpîneșe** (prezentate anterior)se pot reprezenta prin urmatorul subprogram general (procedura sau functie )recursiv exprimat in limbaj natural:

**Exemple de programe unde se aplică această metodă**

**Problema nr1: PROBLEMA TURNURILOR DIN HANOI**

**Prezentarea algoritmului rezolvarii**

Fie trei tije verticale notate A,B,C .Pe tija A se gasesc asezate  n discuri de diametre diferite ,in ordinea crescatoare a diametrelor,privind de sus in jos . Initial ,tijele B si C  sunt goale Sa se afiseze toate mutarile prin care discurile de pe tija A se muta pe tija B , in aceeasi ordine  ,folosind ca tija de manevra C  si resspectand urmatoarele reguli:

la fiecare pas se muta un singur disc; un disc se poate aseza numai peste un disc cu diametrul mai mare .

Rezolvarea acestei probleme se bazeaza pe urmatoarele considerente logice :

-daca n=1 ,atunci mutarea este immediata AàB(mut discul de pe A pe B);

-daca n=2,atunci sirul mutarilor este :  AàC,AàB,CàB;

-daca n>2 procedam astfel :

-mut (n-1)discuri AàC;

-mut un disc AàB ;

-mut cele (n-1)discuri CàB.

Observam ca problema initiala se descompune in trei subprobleme mai simple ,similare problemei initiale: mut (n-1)discuri AàC ,mut ultimul disc pe B ,mut cele (n-1)discuri C-->B.Dimensiunile acestor subprobleme sunt : n-1,1,n-1.Aceste subprobleme sunt independente ,deoarece tijele initial (pe      care sunt dispuse discurile ),tijele finale si tijele intermediare sunt diferite.Notam H(n,A,B,C)=sirul mutarilor a n discuri de pe A pe B, folosind C.

program turnurile \_hanoi;

var n:byte;

 procedure hanoi(n:byte;a,b,c:char);

 begin

 if n=1 then writeln(a,'à',b)

 else begin

  hanoi(n-1,a,c,b);

    writeln(a,'à',b);

  hanoi(n-1,c,b,a);

   end;

   end;

   begin

   write('nr discuri pe tija A =');readln(n);

   writeln('mutarile sunt urmatoarele :');

   hanoi(n,'A','B','C');

   readln;readln end.

## Problema nr2.  Maxim intr-un vector

Se citeste un vector cu n componente, numere naturale. Se cere sa se tipareasca valoarea maxima.

>      daca i=j, valoarea maxima va fi v[i];

>      contrar vom imparti vectorul in doi **vectori:**primul vector va contine componentele de la i la (i+j) div 2, al doilea vector va contine componentele de la (I+J) div 2 +1 la j; rezolvam problemele (aflam maximul pentru fiecare din ele) iar solutia problemei va fi data de valoarea maxima dintre rezultatele celor doua subprobleme.

program maxim;

var v:array[1..10] of integer;

n,i:integer;

function max(i,j:integer):integer;

var a,b:integer;

begin

if i=j then max:=v[i]

else begin

a:=max(i, (i+j) div 2);

b:=max((i+j) div 2+1,j);

if a>b then max:=a

else max:=b;

end;

end;

begin

write(’n=’);

readln(n);

for i:=1 to n do read(v[i]);

writeln(maximul este ’,max(1,n));

end.

## Problema nr3:Cel mai mare divizor comun

Fie n valori numere naturale a1,a2,a3,..,an. Determinati cel mai mare divizor comun al lor prin metoda Divide Et Impera. Se imparte sirul a1,a2,a3,..,an in doua subsiruri a1,a2,a3,..,am,respectiv am+1,am+2,.,an,unde m reprezinta pozitia de mijloc,m=(1+n) div 2.

Cmmdc(a1,a2,..,an)= Cmmdc(a1,a2,..,am), Cmmdc(am+1,am+2,.,an))

program cmmdc\_sir;

const nmax=20;

type indice=1..nmax;

var a:array[indice] of word;

n:indice;

procedure citire;

var i:indice;

begin

readln(n);

for i:=1 to n do read(a[i]);

end;

function euclid(x,y:word):word;

var r:word;

begin

while y<>0 do

begin

r:=x mod y;

x:=y;

y:=r;

end;

euclid:=x;

end;

function cmmdc(p,q:indice):word;

var m:indice;

begin

if q-p<=1 then cmmdc:=euclid(a[p],a[q])

else

begin

m:=(p+q) div 2;

cmmdc:=euclid(cmmdc(p,m),cmmdc(m+1,q));

end;

end;

begin

citire;

writeln(’cmmdc=’,cmmdc(1,n));

readln;

end.

## Problema nr4: Codare Sir

Se considera un sir cu n elemente, initial toate egale cu n. Se imparte sirul pe jumatate, elementele din jumatatea stanga incrementandu-se, iar cele din jumatatea dreapta decrementandu-se cu o unitate. Daca exista element 'nepereche' exact la mijloc acesta ramane neschimbat. Celor doua jumatati li se aplica acelasi 'tratament' si jumatatilor jumatatilor la fel etc. pana cand se obtin secvente de cate un element.

program codare;

var a:array[1..100] of integer;

n,i:integer;

procedure codare(p,q):integer;

var m,i:integer;

begin

if q-p=2 then begin a[q]:=a[q]-1;

a[p]:=a[p]+1;

end

else if q-p=1 then begin a[q]:=a[q]-1;

a[p]:=a[p]+1;

end

else if (q-p) mod 2=0 then

begin m:=(p+q) div 2;

for i:=p to m-1do a[i]:=a[i]+1;

for i:=m+1 to q do a[i]:=a[i]-1;

codare(p,m-1);

codare(m+1,q);

end

else

begin m:=(p+q) div 2;

for i:=p to m do a[i]:=a[i]+1;

for i:=m+1 to q do a[i]:=a[i]-1;

codare(p,m);

codare (m+1,q);

end;

end;

begin

readln(n);

for i:=1 to n do a[i]:=n;

codare(1,n);

for i:=1 to n do write(a[i]:4);

writeln;

end.

## Problema nr5: Sortare rapida (quicksort)

Un tablou V se completeaza cu n elemente numere reale .Sa se ordoneze crescator folosind metoda de sortare rapida .

 Solutia problemei se bazeaza pe urmatoarele etape implementate in programul principa

se apeleaza procedura “quick” cu limita inferioara li=1 si limita superioara ls=n;

functia ”poz”  realizeaza mutarea elementului v[i] exact pe                               pozitia ce o va ocupa acesta in vectorul final ordonat; functia ”poz” intoarce (in k ) pozitia ocupata de acest element;

in acest fel ,vectorul V se imparte in doua parti : li …k-1 si k+1…ls;

pentru fiecare din aceste parti se reapeleaza procedura“quick”,cu limitele modificate corespunzator ;

in acest fel primul element din fiecare parte va fi pozitionat exact pe pozitia finala ce o va ocupa in vectorul final ordonat (functia “poz”);

fiecare din cele doua parti va fi astfel impartita in alte doua parti procesul continua pana cand limitele partilor ajung sa se suprapuna, ceea ce indica ca toate elementele vectorului au fost mutate exact pe pozitiile ce le vor ocupa in vectorul final; deci vectorul este ordonat ;

in acest moment se produc intoarcerile  din apelurile recursive si programul isi termina executia .

program quicksort;

 type  vector= array [1..50] of real;

 var v:vector; i,n,k:integer;

function poz(li,ls:integer):integer;

var i,j,modi,modj,m:integer;

man:real;

begin

i:=li; j:=ls;

modi:=0; modj:=-1;

while i<j do

begin

if  v[i]>v[j] then

begin

man:=v[i];

v[i]:=v[j];

|  |
| --- |
|  |

v[j]:=man;

m:=modi ;

modi:=-modj;

modj:=-m;

end;

i:=i+modi;

  j:=j+modj;

end;

poz:=i;

end;

procedure   quick(li,ls:integer);

begin

if  li<ls then begin

 k::=poz(li,ls);

quick(li,k-1);

quick(k+1,ls);

end;

end;

begin

write(‘cate elemente are vectorul ?=’);readln(n);

for i:=1 to n do

begin

write(‘tastati elementul ‘,i,’=’);

readln(v[i]);

end;

quick(1,n);

writeln(‘vectorul ordonat este :’);

for i:=1 to n do writeln(v[i]);

readln;

end.

**OBSERVATIE**

daca elementul se afla in stanga ,atunci se compara cu elementele din dreapta lui si se sar (j:=j-1)elementele mai mari decat el ;

 daca elementul se afla in dreapta ,atunci se compara cu elemente din stanga lui si se sar (i:=i+1)elementele mai mici decat el.

**Concluzii:**Metoda desparte și stăpânește reprezintă o metodă efecientă a utilizarii limbajului de programare.Această metodă oferă posibilitatea structurării rezolvării unei probleme mai simplu. Metoda desparte și stăpanește admite o implementare recursiva,deoarece subproblemele sunt similare problemei  initiale, dar  de dimensiuni mai mici. Principiul fundamental al recursivitatii este autoapelarea unui subprogram cand acesta este activ, ceea ce se intampla la un nivel, se intampla la orice nivel, avand grija sa asiguram conditia de terminare ale apelurilor repetate.

**Date bibliografice:**

**-**Manual clasa a XI a

- <http://www.creeaza.com/referate/informatica/Metoda-de-programare-DIVIDE-ET449.php>

- <http://www.scritub.com/stiinta/informatica/METODA-DIVIDE-ET-IMPERA25186243.php>

- <http://prohorenco.blogspot.com/2017/05/tehnici-de-programare-desparte-si.html>