Instituția Publică Liceul Teoretic “Spiru Haret”

PROIECT

LA INFORMATICĂ

TEMA: ” Tehnici de elaborare a algoritmilor.

Metoda *desparte* și *stăpânește*

(DIVIDE ET IMPERA)“

Efectuat de: Rîșneanu Ilinca

Profesor: Guțu Maria

Chișinău, 2019

Cuprins

Informații generale

Metoda de programare DIVIDE ET IMPERA constă în împărțirea problemei inițiale de dimensiuni [n] în două sau mai multe probleme de dimensiuni reduse. În general, se execută împărțirea în două subprobleme de dimensiuni aproximativ egale și anume [n/2]. Împărțirea în subprobleme are loc până când dimensiunea acestora devine suficient de mică pentru a fi rezolvate în mod direct (cazul de bază).După rezolvarea celor două subprobleme se execută faza de combinare a rezultatelor în vederea rezolvării întregii probleme.

Metoda DIVIDE ET IMPERA se poate aplica în rezolvarea unei probleme care îndeplinește următoarele condiții :

- se poate descompune în două sau mai multe subprobleme ;
- aceste subprobleme sunt independente una față de alta (o subproblemă nu se rezolvă pe baza alteia și nu se folosesc rezultatele celeilalte);
- aceste subprobleme sunt similare cu problema inițială;
- la rândul lor subproblemele se pot descompune (dacă este necesar) în alte subprobleme mai simple;
- aceste subprobleme simple se pot soluționa imediat prin algoritmul simplificat.

Deoarece puține probleme îndeplinesc condițiile de mai sus, aplicarea metodei este destul de rară.

Etapele rezolvării

Dupa cum sugereaza si numele « desparte si stapaneste », etapele rezolvarii unei probleme (numita problema initiala) in DIVIDE ET IMPERA sunt :

- descompunerea problemei initiale in subprobleme independente similare problemei de baza ,de dimensiuni mai mici;
- descompunerea treptata a subproblemelor in alte subprobleme din ce in ce mai simple ,pana cand se pot rezolva imediata prin algoritmul simplificat;
- rezolvarea subproblemelor simple;
- combinarea solutiilor gasite pentru construirea solutiilor subproblemelor de dimensiuni din ce in ce mai mari;

- combinarea ultimelor solutii determina obtinerea solutiei problemei initiale.

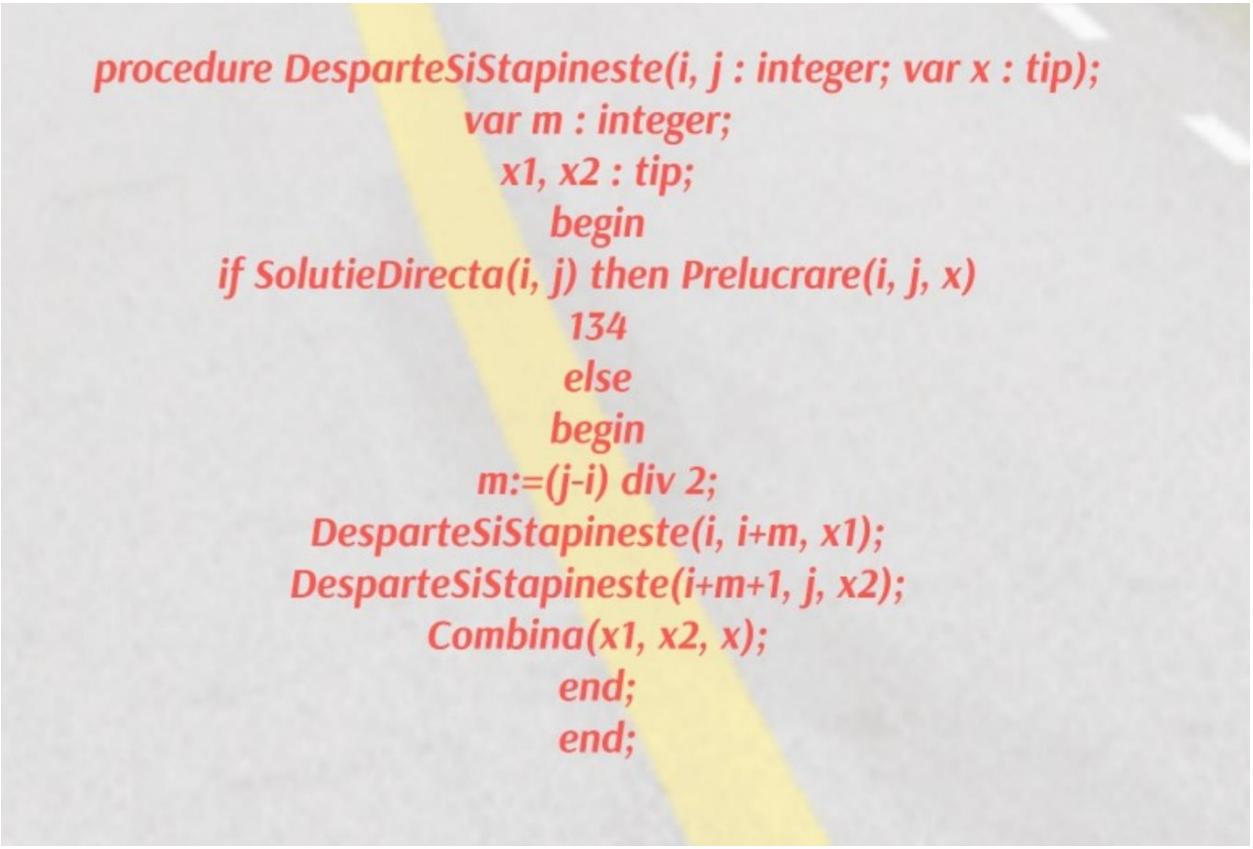
Metoda DIVIDE ET IMPERA admite o **implementare recursiva**, deoarece subproblemele sunt similare problemei initiale, dar de dimensiuni mai mici. Principiul fundamental al recursivitatii este autoapelarea unui subprogram cand acesta este activ, ceea ce se intampla la un nivel, se intampla la orice nivel, avand grija sa asiguram conditia de terminare ale apelurilor repetate. Asemanator se intampla si in cazul metodei DIVITE ET IMPERA, 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 DIVIDE ET IMPERA se produce combinarea subproblemelor (rezolvate deja) prin secventele de revenire din apelurile recursive.

Schema generală

Schema generală a unui algoritm bazat pe metoda desparte și stăpânește poate fi redată cu ajutorul unei procedure 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)
134
else
begin
m:=(j-i) div 2;
DesparteSiStapineste(i, i+m, x1);
DesparteSiStapineste(i+m+1, j, x2);
Combina(x1, x2, x);
end;
end;
```

Probleme rezolvate. Aplicații

1. Maximul dintr-un vector

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

Funcția căutată va genera valoarea maximă dintre numerele reținute în vector pe o poziție dintre i și j (inițial, i=1, j=n). Pentru aceasta, se procedează astfel:

- dacă i=j, valoarea maximă 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.

```
#include <iostream>
using namespace std;
int v[10],n;

int max(int i, int j)
{
  int a, b, m;
  if (i==j) return v[i];
  else
  {
    m = (i+j)/2;
    a = max(i, m);
    b = max(m+1, j);
    if  (a>b) return a;
    else return b;
  }
}

int main( )
{
  cout<<"n=";cin>>n;
  for (int i=1; i<=n; i++)
  {
    cout<<"v[“<<i<<”]=";
cin>>v[i];
  }
  cout<<"max="<<max(1,n);
  return 0;
}
```

2. 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.

3. Căutarea binară într-un şir

Sa se verifice daca o valoare data x exista intr-un sir de numere intregi ordonate crescator. Se va folosi un algoritm de cautare bazat pe metoda Divide Et Impera.

Se descompune problema in doua subprobleme de acelasi tip. Se imparte vectorul in doi subvectori: primul subvector va contine elementele pana la pozitia de mijloc, iar al doilea va fi alcatuit din elementele de dupa pozitia din mijloc. Verificam daca valoarea cautata se gaseste chiar pe pozitia din mijloc si in caz afirmativ cautarea se opreste. Daca valoarea cautata este mai mica decat elementul situat pe pozitia din mijloc, cautarea trebuie continuata in subvectorul stang, iar daca este mai mare, cautarea continua in subvectorul drept.

4. Poziția k

Se citeste de la tastatura un sir de n elemente numere intregi.Sa se gaseasca elementul aflat pe o pozitie data k in sirul ordonat crescator, fara a efectua ordonarea.

5. Partiționarea unui şir

Se considera un sir de n numere intregi, ordonat crescator si un numar intreg x. Sa se partitioneze sirul dat in doua subssiruri, astfel incat toate elementele primului sir sa fie mai mici decat x, iar toate elementele celui de-al doilea sir sa fie mai mari decat x.

Problema 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 3

```
program cautare;
type vector=array[1..20] of integer;
var v:vector;n,x,i:integer;
function caut(p,q:integer):integer;
var mij:integer;
begin
if q<p then caut:=-1
else
begin
mij:=(p+q) div 2;
if v[mij]=x then caut:=mij
else if x<v[mij] then caut:=(p,mij-1)
else caut:=caut(mij+1,q);
end;
end;
begin
write('n=');
readln(n);
write('v[1]=');
readln(v[1]);
for i:=2 to n do
repeat
write('v[',i,']=');
readln(v[i]);
until v[i]>v[i-1];
write('x=');
readln(x);
writeln(caut(1,n));
end.
```

Problema 4

```
program pozitia_k;
const nmax=20;
type indice=1..nmax;
var a:array[indice] of integer;
n,k:indice;
procedure citire;
var i:indice;
begin
write('n=');
readln(n);
for i:=1 to n do begin write('a[',i,']=');
readln(a[i]);
end;
end;
procedure afisare;
var i:indice;
begin
writeln('vectorul este');
for i:=1 to n do write(a[i]:4);
writeln;
end;
function divide(p,q:indice):indice;
var st,dr:indice;x:integer;
begin
st:=p;
dr:=q;
x:=a[p];
while st<dr do
begin
while (st<dr) and (a[dr]>=x) do dec(dr);a[st]:=a[dr];
while (st<dr) and (a[dr]<=x) do inc(st);a[dr]:=a[st];
end;
a[st]:=x;
divide:=st;
end;
function qselect(st,dr,k:indice):integer;
var p:indice;
begin
p:=divide(st,dr);
if k=p-st+1 then qselect:=qselect(st,p-1,k)
else qselect:=qselect(p+1,dr,k-(p-st+1));
end;
begin
citire;
write('k=')
;readln(k);
writeln('pozitia ',k,' in vectorul sortat este ',qselect(1,n,k));
afisare;
end.
```

Problema 5

```
program partitionare;
type vector=array[1..20] of integer;
var v:vector;n,x,i,ref:integer;
function part(p,q):integer;
var mij:integer;
begin
if q<p then part:=p
else begin mij:=(p+q) div 2;
if x=v[mij] then part:=mij else
if x<v[mij] then part:=part(p,mij-1)
else part:=part(mij+1,q);
end;
end;
begin
write('n=');
readln(n);
write('v[1]=');
readln(v[1]);
for i:=2 to n do
repeat
write('v[',i,']=');
readln(v[i]);
until v[i]>=v[i-1];
write('x=');
readln(x);
ref:=part(1,n);
writeln('primul vector');
for i:=1 to ref-1 do write (v[i]:5);
writeln;
writeln('al doilea vector');
for i:=ref to n do write(v[i]:5);
readln;
end.
```

Concluzii

Algoritmii de tip Divide et Impera au o bună comportare în timp, dacă se îndeplinesc următoarele condiții:

- dimensiunile subprogramelor (în care se împarte problema inițială) sunt aproximativ egale("principiul balansării");

- lipsesc fazele de combinare a soluțiilor subproblemelor (cautare binară).

Evident, nu toate problemele pot fi rezolvate prin utilizarea acestei tehnici. Se poate afirma că numărul celor rezolvabile prin *"divide et impera"* este relativ mic, tocmai datorită cerinței ca problema să admită o descompunere repetată.

Dat fiind că problemele se împart în subprobleme în mod recursiv, de obicei împartirea se realizează pâna când șirul obținut este de lungime , caz în care rezolvarea subproblemei este foarte ușoară, chiar banală.

Bibliografie

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

- ✓ https://prezi.com/_hzd2fwjkmrm/metoda-desparte-si-stapineste/

- ✓ https://sites.google.com/site/metodadivideetimpera01/prezentare-generala?fbclid=IwAR3MH921pSjjwUM1tyEu2T3roRI7hyLlTDKKIX6rjoT6uQqkPG_f_aMZtBo

- ✓ https://ro.wikipedia.org/wiki/Divide_et_impera_(informatic%C4%83)