Metoda Divide et Impera

Slide 1:

Descrierea metodei.

Metoda “Divide et impera” (desparte şi stăpâneşte) este o metodă generală de elaborare a algoritmilor. Ea constă în împărţirea repetată a unei probleme de dimensiune mare în două sau mai multe subprobleme de acelaşi tip, urmată de combinarea soluţiilor subproblemelor rezolvate pentru a obţine soluţia problemei propuse.

Pentru clarificare să presupunem că se dă un vector A şi că trebuie efectuată o prelucrare oarecare asupra elementelor sale. Mai mult, presupunem că pentru orice *p, q* numere naturale cu există m astfel încât şi prelucrarea secvenţei se poate face prelucrând secvenţele şi şi apoi combinând rezultatele pentru a obţine prelucrarea dorită a întregii secvenţe . În continuare vom face o primă descriere a metodei “Divide et impera” folosind un algoritm recursiv ce transcrie cele de mai sus. Procedura se va numi DIV_IMP(*p, q, r*) unde:

– *p* – indică începutul secvenţei ce trebuie prelucrată;

– *q* – indică sfârşitul secvenţei ce trebuie prelucrată;

– *r* – indică rezultatul prelucrării.

Procedura se va apela prin ***call*** DIV_IMP(*1, n; r*).

procedure DIV_IMP*(p, q; r)*

if *p – q* < EPS

then

call PREL(*p, q; r*)

else

call DIV(*p, q; m*)

call DIV_IMP(*p, m;*)

call DIV_IMP(*m+1, q;*)

call COMB(,; *r*)

end if

return

Slide 2:

Cautare binara:

Se citesc *n* numere întregi sortate crescător. De asemenea se citeşte un număr întreg nr. Se cere să se decidă dacă nr. se găseşte în şirul celor *n* numere citite.

Căutarea se efectuează între numerele reţinute de componentele de indice între valorile reţinute de două variabile *li* şi *ls* (iniţial *li* = 1 şi *ls* = n).

Fiind date *li* şi *ls* procedăm astfel:

- se calculează indicele componentei din mijloc, în cazul în care n este impar, sau a uneia din cele două plasate în mijloc, în cazul în care *n* este par (*k = (li+ls) div 2*);
- apar trei posibilităţi:
  - valoarea reţinută de componenta de indice calculat este egală cu nr (caz în care căutarea se termină cu succes);
  - valoarea reţinută de componenta de indice calculat este mai mică decât nr (caz în care numărul va fi căutat între componentele de indice *li = k+1* şi *ls*);
  - valoarea reţinută de componenta de indice calculat este mai mare decât nr (caz în care numărul va fi căutat între componentele de indice *li* şi *ls = k -1*).

Căutarea se termină când numărul a fost identificat sau când *li > ls* (căutare cu succes).

Slide 3:

Sortarea prin interclasare

Algoritmul de **sortare prin intercasare** constituie un exemplu reprezentativ pentru folosirea metodei „*divide et impera*" în programare. Astfel, dacă avem de sortat un vector, atunci îl împărţim în două, sortăm – la fel – cele două părţi ale vectorului, apoi le interclasăm. Dacă şi vectorii rezultaţi după împărţire sunt destul de mari (mai mult decât un singur element), atunci procedăm la împărţirea şi a acestui vectori şi tot aşa.

Astfel, vom scrie o procedură SortInterclas(început, sfarsit: Integer); care va sorta vectorul *A* între poziţia *început* şi poziţia *sfarsit.* Procedura va determina poziţia din *mijloc* şi se va autoapela pentru *început* şi *mijloc*, apoi pentru *mijloc*+1 şi *sfârşit,* după care vor interclasa cele două părţi ale vectorului.

Slide 5:

Turnurile din Hanoi

Se spune că în Vietnamul antic, în Hanoi, erau trei turnuri, pe unul din ele fiind puse, în ordinea descrescătoare a diametrelor lor, mai multe (opt) discuri de aur. Din motive obiective, nişte călugări, care le aveau îngrijă, trebuiau să aşeze discurile pe cel de-al doilea turn, în aceeaşi ordine. Ei puteau să folosească, eventual, turnul al treilea, deoarece, altfel discurile nu ar fi avut stabilitate. Discurile puteau fi mutate unul câte unul. De asemenea, se renunţa din start la ideea de a aşeza un disc mai mare peste unul mai mic, deoarece exista riscul ca să se strice discurile, din cauza diferenţei mari de masă.

Deşi, aparent simplă, după câteva încercări, cu un prototip în faţă, se va constata că problema nu e banală. Însă este posibilă o rezolvare optimă a ei (cu numai mutări, deci 255, pentru n=8), folosind tehnica recursivă „divide et impera".

Problema este de a muta n discuri de la turnul 1 la turnul 2. Pentru a o rezolva, să vedem cum se mută, în general, *m* discuri de la un turn *p* la un turn *q*. Se mută primele *m*+1 discuri de pe *r*, *r* fiind turnul auxiliar, apoi singurul disc rămas pe *p*(discul cel mare) se mută de pe *p* pe *q,* după care cele *m*-1 discuri sunt mutate de pe *r* pe *q*. Fireşte, mutarea celor *m*-1 discuri este corectă, deoarece existenţa discurilor de diametre mai mari, la bazele celor trei turnuri nu afectează cu nimic mutările discurilor mai mici.

În cazul limită *m*=1, avem doar o mutare a discului din vârful turnului *p* spre *q.*

Programul de mai jos soluţionează (cu animaţie) problema descrisă, pentru *n*=8. Procedura de bază este *Han*, iar celelalte proceduri sunt pentru mişcarea discului curent. Există şi două proceduri cu structură inedită. Ele sunt scrise în limbaj de asamblare. Folosind întreruperea 10*h*, realizează ascunderea, respectiv reafişarea cursorului text.

Slide 9:

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.

Slide 10:

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))