LUCRARE DE LABORATOR NR. 3

Tema: Metoda divide et impera

Scopul lucrării:

1. Studierea metodei divide et impera.
2. Analiza şi implementarea algoritmilor bazaţi pe metoda divide et impera.

Note de curs:

1. Tehnica divide et impera

Divide et impera este o tehnica de elaborare a algoritmilor care constă în:

1. Descompunerea cazului ce trebuie rezolvat într-un număr de subcazuri mai mici ale aceleiaşi probleme.
2. Rezolvarea succesivă şi independentă a fiecăruia din aceste subcazuri.
3. Combinarea subsoluţiilor astfel obţinute pentru a găsi soluţia cazului iniţial.

Algoritmul formal al metodei divide et impera:

function *divimp*(*x*)
{returnează o soluţie pentru cazul *x*}
1: **if** *x* este suficient de mic
2: **then return** *adhoc*(*x*)
3: {descompune *x* în subcazurile $x_1, x_2, \ldots, x_k$}
4: **for** $i \leftarrow 1$ **to** *k* **do** $y_i \leftarrow divimp(x_i)$
5: {recompune $y_1, y_2, \ldots, y_k$ în scopul obţinerii soluţiei *y* pentru *x*}
6: **return** *y*

unde *adhoc* este subalgoritmul de bază folosit pentru rezolvarea micilor subcazuri ale problemei în cauză.

Un algoritm divide et impera trebuie să evite descompunerea recursivă a subcazurilor "suficient de mici", deoarece, pentru acestea, este mai eficientă aplicarea directă a subalgoritmului de bază.

Observăm că metoda divide et impera este prin definiţie recursivă. Uneori este posibil să eliminăm recursivitatea printr-un ciclu iterativ. Implementată pe o maşina convenţionala, versiunea iterativă *poate fi* ceva mai rapidă (in limitele unei constante multiplicative). Un alt avantaj al versiunii iterative ar fi faptul că economiseşte spaţiul de memorie. Versiunea recursiva foloseşte o stivă necesară memorării apelurilor recursive. Pentru un caz de mărime n, numărul apelurilor recursive este de multe ori în $\Omega(\log n)$, uneori chiar în $\Omega(n)$.

2. Algoritmi de sortare „divide et impera"

2.1. Mergesort (sortarea prin interclasare)

Fie $T[1 .. n]$ un tablou pe care dorim sa-l sortam crescător. Prin tehnica divide et impera putem proceda astfel: separăm tabloul T în două părţi de mărimi cât mai apropiate, sortăm aceste părţi prin apeluri recursive, apoi interclasăm soluţiile pentru fiecare parte, fiind atenţi să păstrăm ordonarea crescătoare a elementelor. Obţinem următorul algoritm pentru rezolvarea unei subprobleme:

procedure *mergesort* (*T*, *p*, *r*)
1: **if** $p < r$
2: **then** $q \leftarrow \lfloor (p+r)/2 \rfloor$
3: *mergesort* (*T*, *p*, *q*)
4: *mergesort*(*T*, *q*+1, *r*)
5: *merge* (*T*, *p*, *q*, *r*)

Acum putem sorta întreg tabloul T apelând procedura *mergesort* (*T*, 1, *n*)

Etapa cea mai importantă a algoritmului este interclasarea *merge* (*T, p, q, r*). Scopul acestei prelucrări este construirea unui tablou ordonat pornind de la două tablouri ordonate. Ideea prelucrării constă în a parcurge în paralel cele două tablouri şi a compara elementele curente. În tabloul final se transferă elementul mai mic dintre cele două elemente curente, iar contorul utilizat pentru parcurgerea tabloului din care s-a transferat un element este incrementat. Procesul continuă până când unul din tablouri este transferat în întregime. Elementele celuilalt tablou sunt transferate direct în tabloul final.

Algoritmul *mergesort* ilustrează perfect principiul divide et impera:

- divide problema în subprobleme;
- stăpâneşte subproblemele prin rezolvare;
- combină soluţiile subproblemelor.

In algoritmul *mergesort*, suma mărimilor subcazurilor este egală cu mărimea cazului iniţial. Această proprietate nu este în mod necesar valabilă pentru algoritmii divide et impera. Este esenţial ca subcazurile să fie de mărimi cât mai apropiate (sau, altfel spus, subcazurile să fie cât mai *echilibrate*).

2.2. Quicksort (sortarea rapida)

Algoritmul de sortare *quicksort*, inventat de Hoare în 1962, se bazează de asemenea pe principiul divide et impera. Spre deosebire de *mergesort*, partea nerecursivă a algoritmului este dedicata construirii subcazurilor şi nu combinării soluţiilor lor.

Ca prim pas, algoritmul alege un element *pivot* din tabloul care trebuie sortat. Tabloul este apoi partiţionat în două subtablouri, alcătuite de-o parte şi de alta a acestui pivot în următorul mod: elementele mai mari decât pivotul sunt mutate în dreapta pivotului, iar celelalte elemente sunt mutate în stânga pivotului. Acest mod de partiţionare este numit *pivotare*. În continuare, cele două subtablouri sunt sortate în mod independent prin apeluri recursive ale algoritmului. Rezultatul este tabloul complet sortat; nu mai este

necesară nici o interclasare. Pentru a echilibra mărimea celor două subtablouri care se obţin la fiecare partiţionare, ar fi ideal să alegem ca pivot elementul median. Intuitiv, *mediana* unui tablou T este elementul m din T, astfel încât numărul elementelor din T mai mici decât m este egal cu numărul celor mai mari decât m. Din păcate, găsirea medianei necesita mai mult timp decât merită. De aceea, putem pur şi simplu să folosim ca pivot primul element al tabloului. Iată cum arată acest algoritm:

```
procedure quicksort (T, p, r )
1: if p < r
2:   then   q ← Partition (T, p, r )
3:          quicksort (T, p, q)
4:          quicksort t(T, q+1, r)
```

Mai rămâne să concepem un algoritm de partiţionare (pivotare) cu timp liniar, care să parcurgă tabloul T o singură dată. Putem folosi următoarea procedură

```
procedure Partition (T, p, r )
1: x ← T[p]
2: i ← p –1;
3: j ← r + 1
4: while TRUE do
5:          repeat j ← j – 1 until T[j] ≤ x
6:          repeat i ← i + 1 until T[i] > x
7:          if i < j
8:                  then interschimbă T[i] ↔ T[j]
9:                  else return j
```

Intuitiv, ne dăm seama că algoritmul *quicksort* este ineficient, dacă se întâmpla în mod sistematic ca subcazurile să fie puternic neechilibrate. Operaţia de pivotare necesită un timp în $\Theta(n)$.

Dacă elementele lui *T* sunt distincte, cazul cel mai nefavorabil este atunci când iniţial tabloul este ordonat crescător sau descrescător, fiecare partiţionare fiind total neechilibrată.

SARCINA DE BAZĂ:

1. Studiaţi noţiunile teoretice despre metoda divide et impera.
2. Implementaţi algoritmii Mergesort şi Quicksort.
3. Efectuaţi analiza empirică a algoritmilor Mergesort şi Quicksort.
4. Implementați +1 algoritm de sortare (algoritmul sa fie unic per elev în subgrupa)
5. Faceţi o concluzie asupra lucrării efectuate.

Întrebări de control:

1. De ce această metodă se numeşte divide et impera ?
2. Explicaţi noţiunea de algoritm recursiv.
3. Descrieţi etapele necesare pentru rezolvarea unei probleme prin metoda divide et impera.
4. Ce tip de funcţie descrie timpul de execuţie al unui algoritm de tipul divide et impera?
5. Care sunt avantajele şi dezavantajele algoritmilor divide et impera?

Atenție: Pentru punctul **4** al **Sarcinii de bază**, dacă în grupa se vor depista 2+ algoritmi gemeni, atunci se va penaliza cu -1 punct la nota, pentru fiecare geaman.