Proiectarea Algoritmilor 2009-2010

Laborator 1

Divide et Impera

Cuprins

1. Obiective laborator

- Intelegerea conceptului teoretic
- Implementarea algoritmilor bazati pe D&I

2. Importanță – aplicații practice

Paradigma divide et impera sta la baza construirii de algoritmi eficienti pentru diverse probleme:

- Sortari (ex: merge sort [1], quicksort [2])
- Inmultirea numerelor mari (ex: Karatsuba [3])
- Analiza sintactica (ex: parsere top-down [4])
- Calcularea transformatei Fourier discreta (ex: FFT [5])

Un alt domeniu de utilizare a tehnicii divide et impera este programarea paralela pe mai multe procesoare, subproblemele fiind executate pe masini diferite.

3. Descrierea problemei şi a rezolvărilor

Expresia provine din latina unde exista si ca proverb: „Divide et impera”. In romana este mai cunoscut ca: „dezbina şi stapaneste”. Termenul este preluat din domeniul militar/politic unde reprezinta o strategie de castigare si mentinere a puterii prin divizarea unei populatii in entitati mai mici. [6]

3.1. Prezentare generală a problemei

O descriere a tehnicii D&I: “Divide and Conquer algorithms break the problem into several subproblems that are similar to the original problem but smaller in size, solve the subproblems recursively, and then combine these solutions to create a solution to the original problem.” [7]

Deci un algoritm D&I **imparte** problema in mai multe subprobleme similare cu problema initiala si de dimensiuni mai mici, **rezolva subproblemele** recursiv si apoi **combina** solutiile obtinute pentru a obtine solutia problemei initiale.

Sunt trei pasi pentru aplicarea algoritmului D&I:

- **Divide:** imparte problema in una sau mai multe *probleme similare* de *dimensiuni mai mici*.
- **Impera (stapaneste):** rezolva subprobleme recursiv; daca dimensiunea subproblemelor este mica se rezolva iterativ.
- **Combina:** combina solutiile subproblemelor pentru a obtine solutia problemei initiale.

Complexitatea algoritmilor D&I se calculeaza dupa formula:

$T(n) = D(n) + S(n) + C(n),$

unde D(n), S(n) si C(n) reprezinta complexitatile celor 3 pasi descrisi mai sus: divide, stapaneste respectiv combina.

3.2. Sortarea prin interclasare

Sortarea prin interclasare [1] este un algoritm de sortare de vectori ce foloseste paradigma D&I:

- **Divide:** imparte vectorul initial in doi subvectori de dimensiune n/2.
- **Stapaneste:** sorteaza cei doi subvectori recursiv folosind sortarea prin interclasare; recursivitatea se opreste cand dimensiunea unui subvector este 1 (deja sortat).
- **Combina:** Interclaseaza cei doi subvectori sortati pentru a obtine vectorul initial sortat.

Pseudocod:

```
MergeSort(v, p, q)              // v – vector, p – lim inf, q – lim sup
    if (p == q) return;         // conditia de oprire
    m = (p + q) / 2;            // etapa divide
    MergeSort(v, p, m);         // etapa stapaneste
    MergeSort(v, m+1, q);
    Merge(v, p, q);             // etapa combina
```

```
Merge(v, p, q)                          // interclasare subvectori
        m = (p + q) / 2;
        i = p;
        j = m + 1;
        k = 1;

        while (i <= m && j <= q)
                if (v[i] <= v[j]) u[k++] = v[i++];
                else u[k++] = v[j++];

        while (i <= m)
                u[k++] = v[i++];

        while (j <= q)
                u[k++] = v[j++];

        copy(v[p..q], u[1..k-1]);
```

Complexitatea algoritmului este data de formula: *T(n) = D(n) + S(n) + C(n)*, unde D(n)=O(1), S(n) = 2*T(n/2) si C(n) = O(n), rezulta *T(n) = 2 * T(n/2) + O(n)*. Folosind teorema Master [8] gasim complexitatea algoritmului: **T(n) = O(n * lg n)**.

3.3. Cautarea binara

Se da v[1..n] un **vector sortat crescator** ce contine valori reale distincte si o valoare x. Sa se gaseasca la ce pozitie apare x in vectorul dat.

Pentru rezolvarea acestei probleme folosim un algoritm D&I:

- **Divide:** impartim vectorul in doi subvectori de dimensiune n/2.
- **Stapaneste:** aplicam algoritmul de cautare binara pe subvectorul care contine valoarea cautata.
- **Combina:** solutia subproblemei este de fapt solutia problemei initiale, motiv pentru care nu mai este nevoie de etapa de combinare.

Pseudocod:

```
BinarySearch(v, p, q, x)
        if (p > q) return;          // conditia de oprire (x nu se afla in v)
        m = (p + q) / 2;                    // etapa divide
        // etapa stapaneste
        if (v[m] == x) return m
        if (v[m] > x) return BinarySearch(v, p, m-1, x);
        if (v[m] < x) return BinarySearch(v, m+1, q, x);
```

Complexitatea algoritmului este data de relatia *T(n) = T(n/2) + O(1)*, ceea ce implica **T(n) = O(lg n)**.

3.4. Turnurile din Hanoi

Se considera 3 tije A, B, C si n discuri de dimensiuni distincte (1, 2.. n ordinea crescatoare a dimensiunilor) situate initial toate pe tija A in ordinea 1,2..n (de la varf catre baza). Singura operatie care se poate efectua este de a selecta un disc ce se afla in varful unei tije si plasarea lui in varful altei tije astfel incat sa fie asezat deasupra unui disc de dimensiune mai mare decat a sa. Sa se gaseasca un algoritm prin care se muta toate discurile pe tija B (problema turnurilor din Hanoi).

Pentru rezolvarea problemei folosim urmatoarea strategie [9]:

- mutam primele n-1 discuri de pe tija A pe tija C folosindu-ne de tija B.
- mutam discul n pe tija B.
- mutam apoi cele n-1 discuri de pe tija C pe tija B folosindu-ne de tija A.

Pseudocod [10]:

```
Hanoi(n, A, B, C)      // muta n discuri de pe tija A pe tija B fol tija C
    if (n >= 1)
        Hanoi(n-1, A, C, B);
        Muta_disc(A, B);
        Hanoi(n-1, C, B, A);
```

Complexitatea: $T(n) = 2*T(n-1) + O(1)$, recurenta ce conduce la $T(n) = O(2^n)$.

4. Concluzii

Divide et impera este o tehnica folosita pentru a reliza algoritmi eficienti pentru diverse probleme. In cadrul acestei tehnici se disting trei etape: *divide*, *stapaneste* si *combina*.

Mai multe exemple de algoritmi care folosesc tehnica divide et impera puteti gasi aici [11].

5. Referinte

[1] - MergeSort - http://www.sorting-algorithms.com/merge-sort

[2] – QuickSort - http://www.sorting-algorithms.com/quick-sort

[3] – Karatsuba - http://en.wikipedia.org/wiki/Karatsuba_algorithm

[4] – Top down parser - http://en.wikipedia.org/wiki/Top-down_parser

[5] – FFT - http://en.wikipedia.org/wiki/Fast_Fourier_transform

[6] – Divide et impera - http://en.wikipedia.org/wiki/Divide_and_rule

[7] – T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms

[8] – Teorema Master - http://people.csail.mit.edu/thies/6.046-web/master.pdf

[9] – Hanoi Applet - http://www.mathcs.org/java/programs/Hanoi/index.html

[10] - Cristian A. Giumale, Introducere in Analiza Algoritmilor (cap. 2.5.1)

[11] - http://www.cs.berkeley.edu/~vazirani/algorithms/chap2.pdf

Proiectarea Algoritmilor 2009-2010

Laborator 1

Divide et Impera

6. Problema 1

a) Sa se scire un algoritm care calculeaza produsul a doua matrice folosind metoda divide et impera. Se va folosi urmatoarea abordare: o matrice NxN se poate scrie ca o matrice de dimensiune 2x2 ce contine ca elemente submatrici de dimensiuni N/2 x N/2.

b) Algoritmul de mai sus are complexitatea $O(n^3)$, la fel ca algoritmul normal (cel cu 3 for-uri). Pentru a reduce complexitatea putem folosi algorimul Strassen: inmultirea a doua matrice 2x2 se face folosind doar 7 inmultiri recursive.

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$P_1 = a \cdot (f - h) \qquad r = P_5 + P_4 - P_2 + P_6$$
$$P_2 = (a + b) \cdot h \qquad s = P_1 + P_2$$
$$P_3 = (c + d) \cdot e \qquad t = P_3 + P_4$$
$$P_4 = d \cdot (g - e) \qquad u = P_5 + P_1 - P_3 - P_7$$
$$P_5 = (a + d) \cdot (e + h)$$
$$P_6 = (b - d) \cdot (g + h)$$
$$P_7 = (a - c) \cdot (e + f)$$

6. Problema 2

Fie S o multime (elemente diferite). Scrieti un algoritm folosind D&I care sa gaseasca al k-lea cel mai mic element din multime. Hint: folositi functia de partitionare de la qsort.