Proiectarea Algoritmilor 2011-2012

Laborator 1

Divide et impera

Cuprins

1 Obiective laborator

- Înțelegerea conceptului teoretic din spatele descompunerii
- Rezolvarea de probleme abordabile folosind conceptul de Divide et Impera

2 Importanţă – aplicaţii practice

Paradigma Divide et Impera stă la baza construirii de algoritmi eficienți pentru diverse probleme:

- Sortări (ex: merge sort [1], quicksort [2])
- Înmulțirea numerelor mari (ex: Karatsuba [3])
- Analiza sintactică (ex: parsere top-down [4])
- Calcularea transformatei Fourier discretă (ex: FFT [5])

Un alt domeniu de utilizare a tehnicii divide et impera este programarea paralelă pe mai multe procesoare, sub-problemele fiind executate pe mașini diferite.

3 Prezentarea generală a problemei

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

Deci un algoritm D&I **împarte** problema în mai multe subprobleme similare cu problema inițială şi de dimensiuni mai mici, **rezolva sub-problemele** recursiv şi apoi **combina** soluțiile obţinute pentru a obține soluția problemei inițiale.

Sunt trei pași pentru aplicarea algoritmului D&I:

- **Divide:** împarte problema în una sau mai multe *probleme similare* de *dimensiuni mai mici*.
- **Impera (stăpâneşte):** rezolva subprobleme recursiv; dacă dimensiunea sub-problemelor este mica se rezolva iterativ.
- **Combină:** combină soluțiile sub-problemelor pentru a obține soluția problemei inițiale.

Complexitatea algoritmilor D&I se calculează după formula:

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

unde D(n), S(n) şi C(n) reprezintă complexitățile celor 3 pași descriși mai sus: divide, stăpânește respectiv combină.

4 Probleme clasice

4.1 Sortarea prin interclasare

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

- **Divide:** împarte vectorul inițial în doi sub-vectori de dimensiune n/2.
- **Stăpânește:** sortează cei doi sub-vectori recursiv folosind sortarea prin interclasare; recursivitatea se oprește când dimensiunea unui sub-vector este 1 (deja sortat).
- **Combina:** Interclasează cei doi sub-vectori sortați pentru a obține vectorul inițial sortat.

Pseudocod:

```
MergeSort(v, p, q)          // v – vector, p – limită inferioară, q –
                            limită superioară
      if (p == q) return;   // condiția de oprire
      m = (p + q) / 2;      // etapa divide
      MergeSort(v, p, m);   // etapa stăpânește
      MergeSort(v, m+1, q);
      Merge(v, p, q);       // etapa combină
```

```
Merge(v, p, q)                    // interclasare sub-vectori
      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 dată de formula: *T(n) = D(n) + S(n) + C(n)*, unde D(n)=O(1), S(n) = 2*T(n/2) și C(n) = O(n), rezulta *T(n) = 2 * T(n/2) + O(n)*.

Folosind teorema Master [8] găsim complexitatea algoritmului: **T(n) = O(n * lg n)**.

4.2 Căutarea binară

Se dă un **vector sortat crescător** (v[1..n]) ce conține valori reale distincte și o valoare x. Sa se găsească la ce poziție apare x în vectorul dat.

Pentru rezolvarea acestei probleme folosim un algoritm D&I:

- **Divide:** împărțim vectorul în doi sub-vectori de dimensiune n/2.
- **Stăpânește:** aplicăm algoritmul de căutare binară pe sub-vectorul care conține valoarea căutată.
- **Combină:** soluția sub-problemei devine soluția problemei inițiale, motiv pentru care nu mai este nevoie de etapa de combinare.

Pseudocod:

```
BinarySearch(v, p, q, x)
      if (p > q) return;          // condiția de oprire (x nu se află în v)
      m = (p + q) / 2;            // etapa divide
      // etapa stăpânește
      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 relația *T(n) = T(n/2) + O(1)*, ceea ce implica: **T(n) = O(lg n)**.

4.3 Turnurile din Hanoi

Se considera 3 tije A, B, C şi n discuri de dimensiuni distincte (1, 2.. n ordinea crescătoare a dimensiunilor) situate inițial toate pe tija A în ordinea 1,2..n (de la vârf către baza). Singura operație care se poate efectua este de a selecta un disc ce se află în vârful unei tije şi plasarea lui în vârful altei tije astfel încât să fie aşezat deasupra unui disc de dimensiune mai mare decât a sa. Sa se găsească un algoritm prin care se mută toate discurile pe tija B (problema turnurilor din Hanoi).

Pentru rezolvarea problemei folosim următoarea 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) // mută 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 **$T(n) = O(2^n)$**.

5 Concluzii

Divide et impera este o tehnică folosită pentru a realiza algoritmi eficienți pentru diverse probleme. În cadrul acestei tehnici se disting trei etape: *divide*, *stăpânește* și *combină*.

Mai multe exemple de algoritmi care folosesc tehnica divide et impera puteți găsi la [11].

6 Referințe

[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] Chapter 2, Divide-and-conquer algorithms, Berkeley University

http://www.cs.berkeley.edu/~vazirani/algorithms/chap2.pdf