UNIUNEA EUROPEANĂ

GUVERNUL ROMÂNIEI

Instrumente Structurale
2007-2013

Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Proiectarea Algoritmilor

2. Scheme de algoritmi – Divide & Impera

Cuprins

- Scheme de algoritmi
- Divide et impera
- Exemplificare folosind Merge sort
- Alte exemple de algoritmi divide et impera
- Greedy
- Exemplificare folosind arbori Huffman
- Demonstrația corectitudinii algoritmului Huffman

Bibliografie

- Giumale – Introducere in Analiza Algoritmilor cap 4.4
- Cormen – Introducere în Algoritmi cap. 17
- http://www.cs.ucsb.edu/~suri/cs235/ClosestPair.pdf
- http://www.cs.umass.edu/~barring/cs611/lecture/4.pdf
- http://thor.info.uaic.ro/~dlucanu/cursuri/tpaa/resurse/Curs6.pps
- http://www.math.fau.edu/locke/Greedy.htm
- http://en.wikipedia.org/wiki/Greedoid

Scheme de algoritmi

- Prin scheme de algoritmi înțelegem tipare comune pe care le putem aplica in rezolvarea unor probleme similare.
- O gamă largă de probleme se poate rezolva folosind un număr relativ mic de scheme.
- => Cunoașterea schemelor determină o rezolvare mai rapidă si mai eficientă a problemelor.

Divide et impera (I)

- Ideea (divide si cucerește) este atribuită lui Filip al II-lea, regele Macedoniei (382-336 i.e.n.), tatăl lui Alexandru cel Mare si se referă la politica acestuia față de statele grecești.

- In CS – Divide et impera se referă la o clasă de algoritmi care au ca principale caracteristici faptul că împart problema in subprobleme similare cu problema inițială dar mai mici ca dimensiune, rezolvă problemele recursiv si apoi combină soluțiile pentru a crea o soluție pentru problema originală.

Divide et impera (II)

- Schema Divide et impera constă in 3 pași la fiecare nivel al recurenței:
  - Divide problema dată intr-un număr de subprobleme;
  - Impera (cucerește) – subproblemele sunt rezolvate recursiv. Dacă subproblemele sunt suficient de mici ca date de intrare se rezolvă direct (ieșirea din recurență);
  - Combină – soluțiile subproblemelor sunt combinate pentru a obține soluția problemei inițiale.

Divide et impera – Avantaje si Dezavantaje

- Avantaje:
  - Produce algoritmi eficienți.
  - Descompunerea problemei in subprobleme facilitează paralelizarea algoritmului in vederea execuției sale pe mai multe procesoare.
- Dezavantaje:
  - Se adaugă un overhead datorat recursivității (reținerea pe stivă a apelurilor funcțiilor).

Merge Sort (I)

- Algoritmul Merge Sort este un exemplu clasic de rezolvare cu D&I.
- Divide: Divide cele n elemente ce trebuie sortate in 2 secvențe de lungime n/2.
- Impera: Sortează secvențele recursiv folosind *merge sort.*
- Combină: Secvențele sortate sunt asamblate pentru a obține vectorul sortat.
- Recurența se oprește când secvența ce trebuie sortată are lungimea 1 (un vector cu un singur element este întotdeauna sortat ☺) .
- Operația cheie este asamblarea soluțiilor parțiale.

Merge Sort (II)

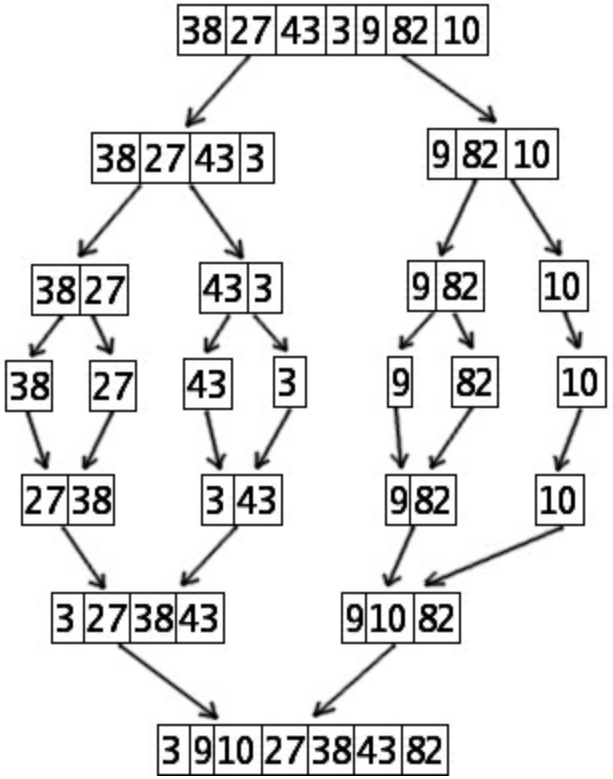
- Algoritm [Cormen]
  - MERGE-SORT(A, p, r)
  - 1 **Dacă** $p < r$
  - 2 **Atunci** $q \leftarrow [(p + r)/2]$ // divide
  - 3 MERGE-SORT(A, p, q) //impera
  - 4 MERGE-SORT(A, $q + 1$, r)
  - 5 MERGE(A, p, q, r) // combină
    // (interclasare)

Merge Sort (III) – Algoritmul de interclasare

- Algoritm [Cormen]
  - MERGE(A, p, q, r) // p si r sunt capetele intervalului, q este "mijlocul"
  - 1 $n_1 \leftarrow q - p + 1$ // numărul de elemente din partea stânga
  - 2 $n_2 \leftarrow r - q$ // numărul de elemente din partea dreapta
  - 3 **creează** vectorii $S[1 \rightarrow n_1 + 1]$ si $D[1 \rightarrow n_2 + 1]$
  - 4 **Pentru** $i = 1 \rightarrow n_1$
  - 5 $S[i] \leftarrow A[p + i - 1]$ // se copiază partea stânga in L
  - 6 **Pentru** $j = 1 \rightarrow n_2$
  - 7 $D[j] \leftarrow A[q + j]$ // si partea dreapta in R
  - 8 $S[n_1 + 1] \leftarrow \infty$
  - 9 $D[n_2 + 1] \leftarrow \infty$
  - 10 $i \leftarrow 1$
  - 11 $j \leftarrow 1$
  - 12 **Pentru** $k = p \rightarrow r$ // se copiază înapoi in vectorul de
  - 13 **Dacă** $S[i] \leq D[j]$ // sortat elementul mai mic din cei
  - 14 **Atunci** $A[k] \leftarrow S[i]$ // doi vectori sortați deja
  - 15 $i \leftarrow i + 1$
  - 16 **Altfel** $A[k] \leftarrow D[j]$
  - 17 $j \leftarrow j + 1$

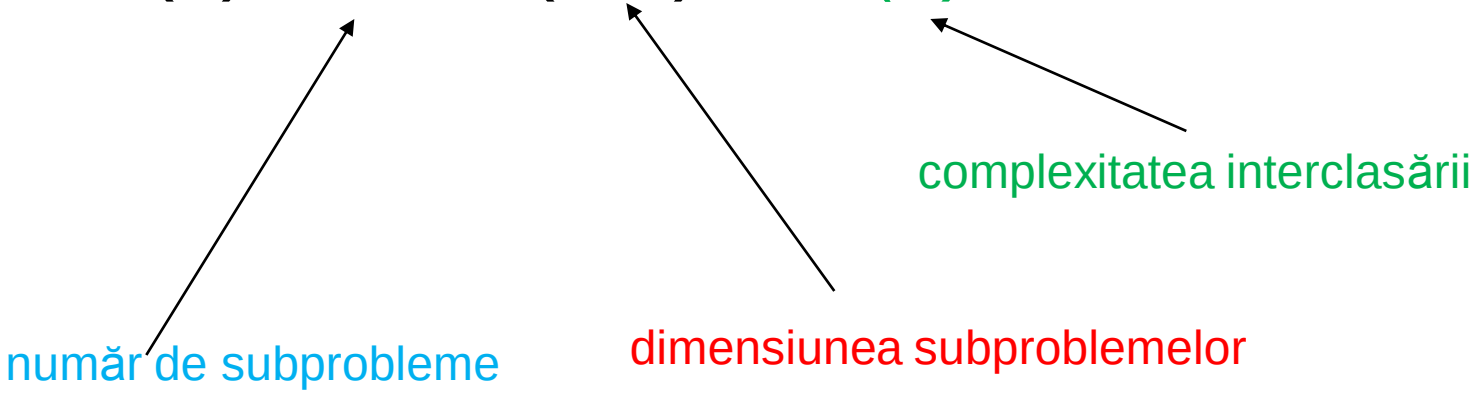
Exemplu funcționare Merge Sort

- Exemplu funcționare [Wikipedia]:



Merge Sort - Complexitate

- $T(n) = 2 * T(n/2) + \Theta(n)$



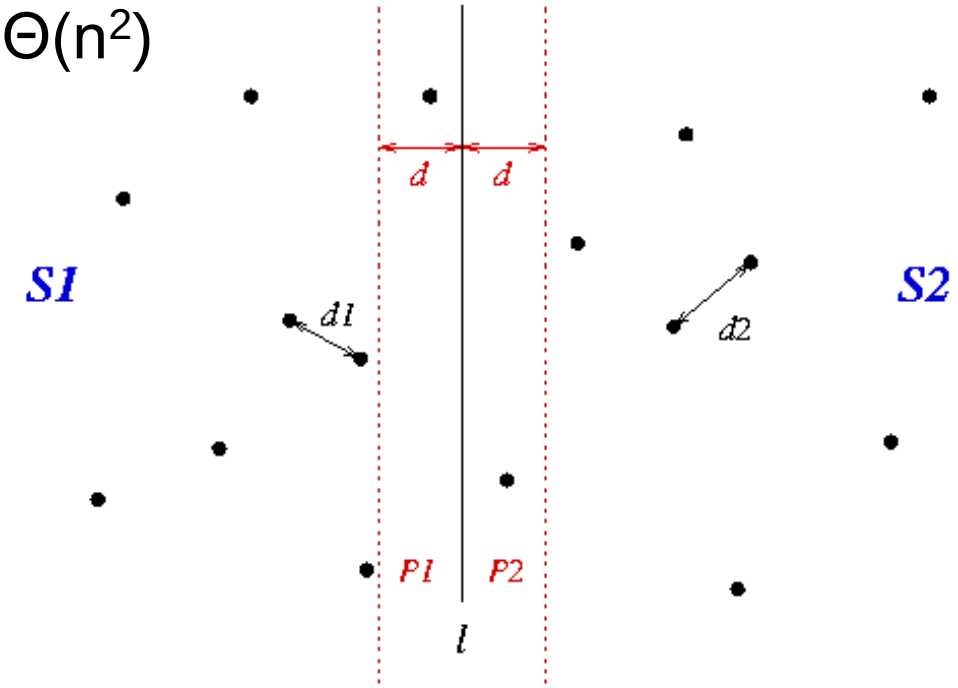
=> (din T. Master) $T(n) = \Theta(n * \log n)$

Divide et impera – alte exemple (I)

- Calculul puterii unui număr: x^n
  - Algoritm "clasic":
    - **Pentru** i = 1 → n rez = rez * x; **întoarce** rez
  - Complexitate: $\Theta(n)$
  - Algoritm divide et impera:
    - **Dacă** n este par
      - **Atunci Întoarce** $x^{n/2} * x^{n/2}$
      - **Altfel** (n este impar) **Întoarce** $x * x^{(n-1)/2} * x^{(n-1)/2}$
  - Complexitate: $T(n) = T(n/2) + \Theta(1) = \Theta(\log n)$

Divide et impera – alte exemple (II)

- Calculul celei mai scurte distante intre 2 puncte din plan (http://www.cs.ucsb.edu/~suri/cs235/ClosestPair.pdf)
  - algoritmul naiv – $\Theta(n^2)$



Divide et impera – alte exemple (III)

- Sortează punctele in ordinea crescătoare a coordonatei x ($\Theta(n \log n)$);
- Împărțim setul de puncte in 2 seturi de dimensiune egală si calculăm recursiv distanța minimă in fiecare set (l = linia ce împarte cele 2 seturi, d = distanța minimă calculată in cele 2 seturi);
- Elimină punctele care sunt plasate la distanța de l > d;
- Sortează punctele rămase după coordonata y;
- Calculează distantele de la fiecare punct rămas la cei 5 vecini (nu pot fi mai mulți);
- Dacă găsește o distantă < d, atunci actualizează d.

Divide et impera – temă de gândire

- Se dă o mulțime M de numere întregi si un număr x. Se cere sa se determine daca există $a,b \in M$ a.i. $a + b = x$
- Algoritmul propus trebuie sa aibă complexitatea $\theta(n \log n)$
- Temele de la curs sunt facultative!☺