Tudorache Alexandru-Theodor, grupa 142

Tema 1 – Structuri de Date

1.

Orice algoritm care construieste un arbore binar de cautare se bazeaza pe comparatii intre elementele ce trebuie adaugate in arbore.

Pentru construirea arboreului trebuie parcurse cele n elemente, asadar complexitatea algoritmului este $\geq \Omega(n)$. Pentru adaugarea fiecarui element in arbore, el trebuie comparat(in best-case, adica atunci cand arborele binar este arbore binar complet, exceptand ultimul rand, la fiecare pas) cu minim ‘h’ elemente, unde ‘h’ este inaltimea curenta a arborelui. In best-case(adica atunci cand arborele binar va fi complet, exceptie facand ultimul rand), ‘h’-ul va fi log n.

Asadar, am demonstrat ca pentru construirea unui arbore binar de cautare este nevoie de un numar minim de n(log n) comparatii, deci complexitatea algoritmului va fi, in cel mai bun caz, $\Omega(n \log n)$.

2.

$f(n) = \Theta(g(n)) \Rightarrow \exists\ c_1, c_2, n_0 > 0$ a.i. $\forall\ n > n_0 \quad c_1 * f(n) \leqq g(n) \leqq c_2 * f(n)$

$g(n) = \Theta(h(n)) \Rightarrow \exists\ c_3, c_4, n_1 > 0$ a.i. $\forall\ n > n_1 \quad c_3 * g(n) \leqq h(n) \leqq c_4 * g(n)$

$c_1 * f(n) \leq g(n), \forall\ n > n_0 \Rightarrow c_1 * c_3 * f(n) \leq c_3 * g(n), \forall\ n > n_0$. Dar $c_3 * g(n) \leq h(n), \forall\ n > n_1$, de unde $c_1 * c_3 * f(n) \leq h(n), \forall\ n > \max(n_0,n_1)$.

$g(n) \leqq c_2 * f(n), \forall\ n > n_0 \Rightarrow c_4 * g(n) \leqq c_2 * c_4 * f(n), \forall\ n > n_0$. Dar $h(n) \leqq c_4 * g(n), \forall\ n > n_1$, de unde $h(n) \leq c_2 * c_4 * f(n)\ \forall\ n > \max(n_0,n_1)$.

Deci $\exists\ c_5, c_6, n_2 > 0$ a.i. $\forall\ n > n_2 \quad c_5 * f(n) \leqq h(n) \leqq c_6 * f(n)$,

unde $c_5 = c_1 * c_3$, $c_6 = c_2 * c_4$, iar $n_2 = \max(n_0,n_1)$.

In concluzie, $f(n) = \Theta(h(n))$.

3.

log n = o(√ n) daca

$$\lim_{n\to\infty}\left(\frac{\log_2(n)}{\sqrt{n}}\right)=0$$

La limita ce trebuie calculata observam cazul de nedeterminare ∞ / ∞. Aplicand regula lui L'Hospital avem:

$$\lim_{n\to\infty}\left(\frac{\log_2(n)}{\sqrt{n}}\right)=\lim_{n\to\infty}\left(\frac{\frac{1}{n\ln(2)}}{\frac{1}{2\sqrt{n}}}\right)=\lim_{n\to\infty}\left(\frac{2}{\ln(2)\sqrt{n}}\right)$$

Cum $\lim_{n\to\infty}\left(\sqrt{n}\right)=\infty$, limita noastra va fi

$$=\frac{2}{\ln(2)}\cdot\frac{1}{\infty}=0$$

Asadar, am demonstrat ca $\lim_{n\to\infty}\left(\frac{\log_2(n)}{\sqrt{n}}\right)=0$, deci log n = o(√ n).

4.

Algoritm pseudocod:

Citeste n
S < – 0
 Pentru i < – 1, n + 1
    Citeste x
    S < – S+ x
    (Sfarsit Pentru)
Afiseaza S – (n*(n + 1) / 2)

Algoritmul se bazeaza pe faptul ca suma tuturor numerelor din sirul dat este 1 + 2 + 3 + … + n + numarul care apare de 2 ori. Astfel putem obtine numarul care apare de 2 ori scazand din suma tuturor numerelor din sir suma 1 + 2 + 3 + … + n = n*(n + 1) / 2. Algoritmul ruleaza in timp liniar deoarece sirul este parcurs o singura data, la citire, pentru calcularea sumei numerelor.

5.

Algoritmul ce urmeaza sa fie prezentat se bazeaza pe principiul Divide et Impera si este unul recursiv. Mai jos sunt prezentate cazul de baza si cazul general din algoritmul recursiv.

- Caz de baza: Doi vectori de lungime 2.

Daca avem X={x1 ,x2} si Y={y1, y2}, atunci dupa interclasarea (teoretica, deoarece nu interclasam vectorii la propriu) vectorilor pe prima pozitie vom avea min(x1, y1), iar pe ultima pozitie vom avea max(x2, y2). Astfel pe pozitiile 2 si 3 raman max(x1, y1) si min(x2, y2), dar nu stim in ce ordine. Mediana unui vector de lungime 4 este elementul de pe pozitia 4/2 = 2. Deci mediana celor 2 vectori concatenati este min(max(x1, y1), min(x2, y2)).

- Cazul general: Doi vectori de lungime > 2

Vom nota m1 = mediana vectorului X, m2 = mediana vectorului Y si Z = vectorul ce ar rezulta daca am interclasa vectorii X si Y.

  - Cazul 1: Daca m1 = m2

  In acest caz, in Z, pe primele n – 1 pozitii am avea toate elementele din X mai mici decat m1 si toate elementele din Y mai mici decat m2(nu ne intereseaza ordinea), iar pe ultimele n – 1 pozitii am avea toate elementele din X mai mari decat m1 si toate elementele din Y mai mari decat m2(nu ne intereseaza ordinea), iar pe pozitiile din mijloc am avea pe m1 si m2, care vor fi egale si vor fi mediana pe care o cautam.

  - Cazul 2: Daca m1 < m2

  In acest caz, in Z, m1 ar fi inaintea lui m2. Inainte de m1 ar fi cel putin [n / 2] –1 elemente (elementele mai mici decat m1 din X) si cel mult n – 1 elemente (in cazul in care toatele elementele din Y mai mici decat m2 sunt mai mici decat m1), iar dupa m2 ar fi cel putin [n / 2] –1 elemente (elementele mai mari decat m2 din Y) si cel mult n – 1 elemente (in cazul in care toate elementele din X mai mari decat m1 sunt mai mari decat m2). Deoarece pe noi ne intereseaza mediana lui Z, putem da la o parte toate elementele din X mai mici decat m1 si elementele din Y mai mari decat m2 si sa apelam recursiv algoritmul de cautare al medianei pentru vectorii obtinuti dupa inlaturarea elementelor specificate mai sus.

- Cazul 3: Daca m1 > m2

In acest caz, in Z, m2 ar fi inaintea lui m1. Inainte de m2 ar fi cel putin [n / 2] –1 elemente (elementele mai mici decat m2 din Y) si cel mult n – 1 elemente (in cazul in care toate elementele din X mai mici decat m1 sunt mai mici decat m2), iar dupa m2 ar fi cel putin [n / 2] –1 elemente(elementele mai mari decat m1 din X) si cel mult n – 1 elemente (in cazul in care toate elementele din Y mai mari decat m2 sunt mai mari decat m1). Deoarece pe noi ne intereseaza mediana lui Z, putem da la o parte toate elementele din Y mai mici decat m2 si elementele din X mai mari decat m1 si sa apelam recursiv algoritmul de cautare al medianei pentru vectorii obtinuti dupa inlaturarea elementelor specificate mai sus.

Algoritmul prezentat are timp de rulare T(n) = T(n/2) + 1.
Acest timp de rulare este = O(log n), dupa cum a fost demonstrat la curs.

6.

1.

Algoritmul greedy prezentat nu este optim deoarece acesta presupune ca atunci cand nu s-ar mai putea adauga un bun ' i ' in Ferrari deoarece acesta nu incape, sa existe posibilitatea sa ramana in Ferrari un spatiu gol ‘s’ in care ar fi putut incapea un bun ‘j’ cu (x_j <= s), unde j > i si j <= k. Din cauza acestui fapt se pot efectua mai multe drumuri decat este necasr.

Un exemplu de caz in care algoritmul greedy nu ar fi optim este:

n = 5
k = 4
X = [4, 3, 1, 2]

In acest caz, algoritmul greedy prezentat ar face 3 drumuri astfel: 1. bunul 1(dimensiune totala 4); 2. bunul 2 si bunul 3(dimensiune totala 4); 3. bunul 4(dimensiune totala 2).
Un algoritm optim ar face doua drumuri: 1. bunul 1 si bunul 3(dimensiune totala 5); 2. bunul 2 si bunul 4(dimensiune totala 5).

Asadar, algoritmul greedy prezentat nu este optim.