Tema 1 – Structuri de Date

Student: Nimară Dan Gabriel
Grupa 141

1 1,5 puncte

Demonstrati ca orice algoritm care construieste un arbore binar de cautare cu n numere ruleaza in timp $\Omega(n \log n)$.

Teorema 9.1 Orice arbore de decizie care sortează n elemente are înălţimea $\Omega(n \lg n)$.

Demonstraţie. Să considerăm un arbore de decizie având înălţimea h şi care sortează n elemente. Întrucât există $n!$ permutări ale celor n elemente, fiecare permutare reprezentând o ordine distinctă de sortare, arborele trebuie să aibă cel puţin $n!$ frunze. Deoarece un arbore binar având înălţimea h nu are mai mult de 2^h frunze, avem

$$n! \leq 2^h,$$

iar, prin logaritmare, rezultă

$$h \geq \lg(n!),$$

deoarece funcţia lg este monoton crescătoare. Din aproximarea lui Stirling (2.11), avem

$$n! > \left(\frac{n}{e}\right)^n,$$

unde $e = 2.71828...$ este baza logaritmilor naturali; astfel

$$h \geq \lg\left(\frac{n}{e}\right)^n = n \lg n - n \lg e = \Omega(n \lg n).$$

Presupunem prin reducere la absurd existența unui algoritm de construire al unui arbore binar de căutare bazat pe comparații ce ar rula în timp $o(n \log n)$. În urma unei parcurgeri în inordine a arborelui construit(ce ar rula în Θ(n), prezentat la curs) obținem șirul de n numere inițiale sortat.

În acest caz, ar însemna că se sortează un șir de n numere în complexitate $o(n \log n)$. Contradicție cu demonstrația din cursul din data de 23.04.2020(+teorema de mai sus) conform căreia orice algoritm de sortare pe bază de comparații ar rula în timp $\Omega(n \log n)$. Așadar, orice algoritm de construire al unui arbore binar de căutare cu n elemente bazat pe comparații rulează în timp $\Omega(n \log n)$.

2 1,5 puncte

Demonstrati ca daca $f(n) = \Theta(g(n))$ si $g(n) = \Theta(h(n))$ atunci $f(n) = \Theta(h(n))$.

Utilizând informațiile din Cursul 1 și Seminarul 1 știm că:
f(n) = Θ(g(n)) ⇔ ∃ c_1, c_2, n_0 > 0 ∀ n ≥ n_0 : c_1f(n)≤ g(n) ≤ c_2f(n) (1)
g(n) = Θ(h(n)) ⇔ ∃ c_3, c_4, n_1 > 0 ∀ n' ≥ n_1 : c_3g(n)≤ h(n') ≤ c_4g(n) (2)
Dacă înmulțim relația c_1f(n)≤ g(n) cu c_3 → c_1c_3f(n) ≤ c_3g(n)
Și relația c_2f(n)≤ g(n) cu c_4 → c_2c_4f(n) ≤ c_4g(n)
Notăm $c_1c_3=c_5$ și $c_2c_4=c_6$ și înlocuim în a doua relație vom avea pentru: c_5f(n'')≤ h(n'') ≤ c_6f(n') ∃ c_1, c_2, n_0 > 0 ∀ n'' ≥ n_0 → f(n) = Θ(h(n)).

3 1,5 puncte

Demonstrati ca $\log n = o(\sqrt{n})$.

Pentru a demonstra egalitatea de mai sus, voi utiliza definiția cu limită a notației-o(definiții complete Introducere în Algoritmi de Thomas H. Cormen(ediția în lb. română, pg. 25-26)).

$f(n) = o(g(n))$	Small O; Small Oh	f is dominated by g asymptotically	$\forall k > 0\ \exists n_0\ \forall n > n_0\ \lvert f(n)\rvert < k \cdot g(n)$	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = \lim_{n\to\infty} \frac{\frac{1}{n \ln 2}}{\frac{1}{2\sqrt{n}}} = \lim_{n\to\infty} \frac{2}{\ln 2} \times \frac{1}{\sqrt{n}} = \frac{2}{\ln 2} \lim_{n\to\infty} \frac{1}{\sqrt{n}} = \frac{2}{\ln 2} \times \frac{1}{\infty} = 0$$

Rezultă că log n = o($\sqrt{n}$).

4 1,5 puncte

Se da un sir cu n numere de la 1 la n, cu exceptia unui numar care apare de 2 ori. Determinati numarul care apare de doua ori. Pentru un algoritm de complexitate $O(n^2)$ se acorda 0,5 puncte. Pentru un algoritm de complexitate $O(n \log n)$ veti primi 1 punct, iar pentru un algoritm de complexitate $O(n)$ care foloseste doar $O(1)$ spatiu suplimentar (adica fara vector de frecvente) veti primi 1,5 puncte.

Exemplul 1: 2 1 3 3 4
Elementul duplicat este: 3

Exemplul 2: 4 1 5 5 2 3
Elementul duplicat este: 5

```cpp
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
    int i, n, s=0, x;
    cin>>n;
    for(i=0;i<n;i++){
        cin>>x;
        s+=x;
    }
    int dupl=s-(n*(n-1))/2;
    cout<<"Elementul duplicat este: "<<dupl;
    return 0;
}
///Rezolvare:
///Citesc sirul de n numere liniar cu un for.
///In acelasi timp, adaug elementul citit la suma
///La final, pentru a determina elementul duplicat
///Scad din suma numarul total ce ar fi trebuit sa rezulte
///daca nu se repeta niciun numar si afisez rezultatul.
///Complexitate timp: O(n) cu O(1) spatiu suplimentar.
```

Rezolvare în cazul în care un număr se poate repeta de mai multe ori:

```cpp
#include <iostream>
#include <cmath>
using namespace std;
void duplicat(int v[], int n)
{
    for(int i=0;i<n;i++)
    {
        int val=abs(v[i]);
        if(v[val-1]>=0){
            v[val-1]=-v[val-1];
        }
        else cout<<val<<" ";
    }
}
int main()
{
    int v[] = {2, 4, 5, 1, 3, 2}, n=6;
    duplicat(v,n);
    return 0;
}
///Functia duplicat:
///Parcurg liniar sirul de n numere cu un for. Elementului
///curent ii retin valoarea in modul, in caz ca a devenit
///anterior un nr negativ. Verific daca pe pozitia
///elementului(daca vectorul ar fi sortat) este un numar
///pozitiv. In acest caz, transform nr. de pe acea pozitie
///in nr negativ. Daca dau a doua oara de numar in sir,
/// la pozitia respectiva este deja un nr negativ, adica
/// l-am mai intalnit, deci este cel cautat si il afisez.
///Complexitate timp: O(n) cu O(1) spatiu suplimentar.
```

5 1,5 puncte

Fie $X[1 :: n]$ si $Y[1 :: n]$ doi vectori, fiecare continand n numere *sortate*. Prezentati un algoritm care sa gaseasca mediana celor 2n elemente. Mediana unei multimi de n elemente este elementul de pe pozitia $[n/2]$ in sirul sortat. De exemplu, mediana multimii $3, 1, 7, 6, 4, 9$ este 4.

In functie de timpul de rulare al algoritmului veti primi urmatoarele punctaje: $O(n \log n)$ - (0,25 puncte); $O(n)$ - (0,5 puncte); $O(\log^2 n)$ - (1 punct); $O(\log n)$ - (1,5 puncte).

Algoritm:

Calculăm medianele medx și medy ale celor 2 vectori.

Dacă medx==medy, am găsit mediana, o returnăm.

Altfel, dacă medx>medy

Căutăm mediana în primele elemente din vectorul X până la medx și în ultimele elemente din vectorul Y începând cu medy

dacă medy>medx

Căutăm mediana în primele elemente din vectorul Y până la medy și în ultimele elemente din vectorul X începând cu medx

Repetăm până când lungimile subvectorilor devin egale cu 2.

Atunci când lungimile subvectorilor sunt egale cu 2(avem în total 4 elemente), returnăm al doilea cel mai mic element dintre cele 4 elemente, indiferent de cum sunt împărțite acestea între cei doi vectori.

Complexitate timp: O(log n)

Deoarece căutăm în câte două jumătăți din vectorii inițiali, avem următoare formula de recursie: T(n) = T(n/2) + 1.

Inducție:

Pasul 1 este verificat deoarece pentru n=1 timpul de rulare al algoritmului este constant.

Vrem să arătăm că T(n)$\le c \log n$

Presupunem că T(n/2) $\le c \log\frac{n}{2}$ → T(n) $\le c \log\frac{n}{2} + 1 \le c \log n - c \log 2 + 1 \le c \log n - c + 1$ → Deci, T(n)$\le c \log n$ pentru $n \ge 1$

6 1,5 puncte

Sa presupunem urmatoarele. Ati castigat la loterie si v-ati cumparat o vila pe care doriti sa o mobilati. Deoarece Ferrari-ul dvs. are capacitate limitata, doriti sa faceti cat mai putine drumuri de la magazin la vila. Mai exact, Ferrari-ul are capacitate n, iar dumneavoastra aveti de cumparat k bunuri de dimensiune $x_1, x_2, \ldots, x_k$.

Fie urmatorul algoritm greedy. Parcurgem bunurile in ordinea $1, 2, \ldots k$ si incercam sa le punem in masina. In momentul in care un bun nu mai incape in masina, efecutam un transport si continuam algoritmul.

1. Demonstarti ca algoritmul prezentat mai sus nu este optim. (0.5 puncte)
2. Fie OPT, numarul de drumuri in solutia optima. Demonstrati ca algoritmul greedy prezentat mai sus efectueaza cel mult 2OPT drumuri. (1 punct).

1. Pentru o capacitate de 7 și 6 bunuri de dimensiuni 2, 3, 6, 2, 7 și 1 algoritmul prezentat ar face mai întâi drumul cu primele două bunuri(al treilea nu ar încăpea, 2+3<7, 2+3+6>7), un al doilea drum cu al treilea bun (al 4-lea nu ar încăpea, 6+2>7), un al treilea drum cu bunul de dimensiune 2 (al 5-lea bun nu ar încăpea, 2+7>7), al 4-lea drum pentru bunul 5(7+1>7) și un al 5-lea drum pentru bunul 6(de capacitate 1).

Soluția optimă ar fi să se facă primul drum pentru bunurile de dimensiuni 2, 3 și 2, al doilea pentru bunurile de dimensiuni 6 și 1 și ultimul drum pentru bunul de dimensiune 7.

Dacă algoritmul ar fi fost optim, s-ar fi făcut cele 3 drumuri. În cazul algoritmului prezentat se fac 5 drumuri, deci nu este optim, deoarece există o soluție mai bună decât cea prezentată.

2. Referințe: https://en.wikipedia.org/wiki/Bin_packing_problem#Next-Fit_(NF)

Voi utiliza notațiile:

- sum[i] = suma dimensiunilor tuturor bunurilor transportate la drumul i;
- d = ca numărul drumurilor efectuate;
- cap = capacitatea mașinii;

în soluția prezentată.

Pentru două drumuri vecine știm că sum[i-1]+sum[i]>cap, atunci:

sum[1]+sum[2]>cap
sum[3]+sum[4]>cap
……………………
sum[d-1]+sum[d]>cap

Însumând, vom avea o sumă totală(știm că cel puțin d-1 drumuri își acoperă cu siguranță peste jumătate din capacitate – referințe):

$$\sum_{i=1}^{n} sum[i] > \frac{cap \times (d-1)}{2}$$
$$cap \times OPT \geq \sum_{i=1}^{n} sum[i]$$
$$cap \times OPT \geq \frac{cap \times (d-1)}{2}$$
$$d - 1 < 2 \times OPT$$
$$2 \times OPT \geq d$$