Tema Curs Structuri de Date

Horjea Cosmin-Marian
Anul 1 - Grupa 143
Universitatea din Bucuresti

April 24, 2020

Question 1

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

Presupunem ca putem avea un timp de executie mai bun decat $\Omega(n \log n)$ Putem pargurge un arbore binar de cautare in inordine (fiu stang, radacina , fiu drept) intr-un timp de $O(n)$. Parcurgand astfel, trecem prin elementele arborelui in ordine crescatoare a valorilor, deci avem valorile sortate.

Dar in cel mai defavorabil caz, timpul pentru un algoritm de sortate bazat pe comparatii este $\Omega(n \log n)$, deci daca am putea construi un arbore binar de cautare intr-un timp mai bun decat $\Omega(n \log n)$ am avea si o sortare mai rapida bazata pe comparatii folosind parcuregerea in inordine, ceea ce este *imposibil* $\square$

Question 2

Demonstrati ca daca $f(n) = \Theta(g(n))$ si $g(n) = \Theta(h(n))$ atunci $f(n) = \Theta(h(n))$.
Din definitie: Prin definitie avem :
$f(n) = \Theta(g(n)) \iff \exists\, c_1, c_2, n_0 > 0$ astfel incat $\forall n \geq n_0$ avem $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$,
iar $g(n) = \Theta(h(n)) \iff \exists\, c_3, c_4, n_0 > 0$ astfel incat $\forall n \geq n_0, c_3 \cdot h(n) \leq g(n) \leq c_4 \cdot h(n)$
Avem :

$$f(n) \leq c_2 \cdot g(n) \Rightarrow f(n) \leq c_2 \cdot c_4 \cdot h(n)$$

$$c_1 \cdot g(n) \leq f(n) \Rightarrow c_3 \cdot c_1 \cdot h(n) \leq f(n)$$

Deci:

$$c_3 \cdot c_1 \cdot h(n) \leq f(n) \leq c_2 \cdot c_4 \cdot h(n)$$

Ceea ce inseamna ca exista doua constante astfel incat $f(n) = \Theta(h(n))\square$

Question 3

Demonstrati ca $\log n = o(\sqrt{n})$
In cazul genereal, cand spunem $f(n) = o(g(n))$ functia $f(n)$ devine neglijabila relativ la $g(n)$ cand $n \to \infty$, adica

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

in cazul nostru $f(n) = \log n$ si $g(n) = \sqrt{n}$, deci vom avea :

$$\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} \overset{\text{L'Hospital}}{=\!=} \lim_{n\to\infty} \frac{2\sqrt{n}}{n} = \lim_{n\to\infty} \frac{2\sqrt{n}}{\sqrt{n}\cdot\sqrt{n}} = \lim_{n\to\infty} \frac{2}{\sqrt{n}} = 0$$

Deci putem afirma ca $\log n = o(\sqrt{n})$ $\square$

Question 4

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
Ideea de rezolvare este aceea ca trebuie sa marcam cumva numerele prin care deja am trecut, cum avem numere in intervalul $1 \dots n$ putem sa le schimbam semnul si sa le facem negative accesand elementul care are pozitia egala cu valoarea elementului selectat la o anumita iteratie prin vector , iar daca incercam sa schimbam un numar negativ in unul pozitiv, inseamna ca pozitia pe care vrem sa o accesam este numarul duplicat.

```
numere[1 :: n]
for i = 1, i ≤ n do
    if numere[| numere[i] |] < 0 then
        return | numere[i] |
    end if

    numere[| numere[i] |] = numere[| numere[i] |] * (−1)
end for
```

Parcurgem o singura data vectorul, accesand doar pozitii valide (deoarece valorile sunt in intervalul $1 \dots n$), si ne oprim doar cand accesam o valoare negativa, ceea ce inseamna ca avem cel mult n operatii, deci o complexitate de $\mathcal{O}(n)$ a timpului de executie si fara spatiu suplimentar. $\square$

Qustion 5

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

Putem afla mediana fiecaruia din vectori in timp constant, deoarece sunt sortati. Dupa ce avem medianele le verificam valorile: daca sunt egale, inseamna ca numerul de valori mai mici si mai mare decat acea valoare este egal in reuniunea vectorilor, deci acea valoare este mediana.

Daca mediana unui vector este mai mare decat cealalta, atunci apelam recursiv functia pentru prima jumatate a vectorului cu mediana mai mare si a doua jumatate a celuilalt, deoarece in vectorul interclasat mediana mai mare se afla in dreapta vectorului si cea mai mica in stanga, deci alegem doar partile care ne aduc mai aproape de mijloc, unde se afla mediana cautata.

In cazul in care avem vectori de lungime 2 interclasam si returnam mediana vectorului rezultat.

function mediana2VectoriSortati(X,Y,n):

```
if n ≤ 2 then
  return (X ∪ Y)[n/2]
end if
med1 ← X[n/2]
med2 ← Y[n/2]
if med1 = med2 then
  return med1
else
  if med1 > med2 then
    return mediana2VectoriSortati(X[1 :: n/2], Y[n/2 :: n], n/2)
  else
    return mediana2VectoriSortati(X[n/2 :: n], Y[1 :: n/2], n/2)
  end if
end if
```

Avem in total 2n elemente si la fiecare apel injumatatim numarul, deoarece eliminam jumatate din fiecare vector, iar *"taierea"* vectorilor o consideram operatie constanta.

Astfel, formula de recurenta este:

$$T(n) = T(n/2) + 1$$

In concluzie, timpul de rulare este $\mathcal{O}(n)\square$

Question 6

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 $x_1, x_2 \dots x_k$. Fie urmatorul algoritm greedy. Parcurgem bunurile in ordinea $1, 2, \dots k$ si incercam sa le punem in masina. In momentul in care un bun nu mai incape in masina, efecutam un transport si continuam algoritmul.

Question 6.1

Demonstarti ca algoritmul prezentat mai sus nu este optim.
Sa consideram urmatoarele date de intratre : $n = 10, k = 4$ iar dimesniunile obiectelor sunt $[6, 7, 3, 4]$.
Conform algoritmului prezentat am transporta obiectele de dimensiune $6, 7$ in doua transporturi separate si pe cele de $3, 4$ in unul singur, astfel efectuand *trei* trasnporturi, insa, putem lua obiectul de dimensiune 6 cu cel de dimensiune 4, efectuam un transport, iar dupa aceea pe cele de 3 si 7, in total efectuand doua transporturi.
In concluzie, algoritmul prezentat nu este optim □

Question 6.2

Fie OPT, numarul de drumuri in solutia optima. Demonstrati ca algoritmul greedy prezentat mai sus efectueaza cel mult 2OPT drumuri.
Consideram un algoritm optim, un algoritm care *umple* cat de mult posibil capacitatea masinii inainte sa efectueze un transport. Fie un numar $x \leq n$ suma dimensiunilor obiectelor pe care le transporta masina efectuand un trasport folosind algoritmul eficient.
Atunci cand folosim algoritmul propus, conditia ca sa efectuam un transport este ca obiectul pe care vrem sa-l transportam nu mai incape in masina, adica $transport_i + ob_j > n$, unde $transport$ este cantitatea deja incarcata in masina iar ob dimenisiunea obiectului pe care vrem sa-l transportam. Deci in oricare doua tranporturi avem obiecte de dimensiuni mai mari strict decat n.
Presupunem ca am avea nevoie de 3 transporturi pentru a echivala cantitatea x tranportata pe un drum de algoritmul eficient. Asta ar insemna ca $transport_i + transport_j < x$(doua transporturi care duc obiecte mai mici decat dimensiunea x), dar am stabilit ca x este mereu mai mic sau egal decat n(capacitatea masinii), deci asta ar insemna ca obiectele din $transport_i$ si $transport_j$ trebuie puse intr-un singur transport folosind algoritmul propus. Deci ajungem la contradictie cu instructiunile algoritmului.
Prin urmare $\forall$ transport din algoritmul eficient poate fi reprodus in *cel mult* doua transporturi folosind algoritmul propus, avand nevoie astfel de *cel mult* $2OPT$ transporturi□

Alternativ, daca tot am castigat loteria, o solutie mai eficienta ar fi fost sa inchiriem un camion care sa ne transporte bunurile, sau macar o dubita, iar Ferrari-ul il putem pastra pentru pista de curse pe care o vom cumpara :)