Tema 1 - Structuri de date

Borza Maria - Cristina

April 9, 2020

1

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

Solutie:

Fie $f(n)$ timpul necesar pentru construirea unui arbore binar cu n noduri.

Presupunem prin absurd ca $f(n) \notin \Omega(nlogn)$

Consideram urmatorul algoritm: pentru un sir de numere dat, de lungime n, construim un arbore binar de cautare in care inseram numerele, dupa care facem parcurgerea inordine a arborelui. La final, vom obtine sirul sortat. Acest algoritm are complexitate $f(n) + g(n)$, unde $g(n)$ este timpul necesar pentru parcurgearea unui arbore binar de cautare cu n noduri.

Vom demonstra prin inductie ca $g(n) \in \Theta(n)$

Deoarece algoritmul de parcurgere inordine viziteaza fiecare nod, e clar ca $g(n) \in \Omega(n)$

Vrem sa demonstram acum ca $g(n) \in O(n)$

Stim ca $g(n) = g(k) + g(n - k - 1) + c'$, unde k = numarul de noduri din subarborele stang.

Vom presupune ca $g(x) = cx, \forall x < n$. Vrem sa aratam ca $g(n) \leq cn$

$g(n) = g(k)+g(n-k-1)+c' \Rightarrow g(n) \leq ck+c(n-k-1)+c' \Rightarrow g(n) \leq ck+cn-ck-c+c' \Rightarrow g(n) \leq cn - c + c' \Rightarrow g(n) \in O(n)$

Cum $g(n) \in O(n)$ si $g(n) \in \Omega(n) \Rightarrow g(n) \in \Theta(n)$

$g(n) \in \Theta(n) \Rightarrow \exists c1', c2', n0' > 0$ astfel incat $\forall n \geq n0'$, $c1'n \leq g(n) \leq c2'n$

Sa presupunem prin absurd ca $f(n) + g(n) \in \Omega(nlogn) \Rightarrow \exists c'', n0'' > 0$ astfel incat $\forall n \geq n0''$, $c''nlogn \leq f(n) + g(n)$.

Fie n0 = $max(n0', n0'')$. Atunci $\exists c'' > 0$ astfel incat $c''nlogn \leq f(n) + g(n)$ si $\exists c2' > 0$ astfel incat $g(n) \leq c2'n$, $\forall n \geq n0$, $\Rightarrow f(n) + g(n) \leq c2'n + f(n) \Rightarrow c''nlogn \leq f(n) + g(n) \leq c2'n + f(n) \Rightarrow c''nlogn \leq c2'n + f(n) \Rightarrow c''nlogn - c2'n \leq f(n) \Rightarrow f(n) \in \Omega(nlogn) - contradictie \Rightarrow f(n) + g(n) \notin \Omega(nlogn) \Rightarrow$ Exista o sortare prin comparare cu timpul de rulare $\notin \Omega(nlogn)$ (1).

Vom demonstra acum ca orice sortare prin comparare necesita $\Omega(nlgn)$ comparatii in cazul nefavorabil.

Se stie ca numarul de comparatii pe care il necesita un algortim de sortare prin comparare este egal cu inaltimea arborelui de decizie. Fie h aceasta inaltime si l numarul de frunze. Pentru ca sunt $n!$ permutari posibile si fiecare trebuie sa fie ca frunza, avem ca $l \leq n!$. Dar, intr-un arbore binar cu inaltimea h, nu putem avea mai mult de 2^h frunze, asa ca $l \leq 2^h \Rightarrow n! \leq l \leq 2^h \Rightarrow lg(n!) \leq h$

Pe de alta parte, $lg(n!) = lg(1)+lg(2)+...+lg(n) \Rightarrow lg(n!) \geq lg(\frac{n}{2})+lg(\frac{n}{2}+1)+...+lg(n) \geq lg(\frac{n}{2}) + lg(\frac{n}{2}) + ... + lg(\frac{n}{2}) \Rightarrow lg(n!) \geq \frac{n}{2}lg(\frac{n}{2}) = \frac{n}{4}lg(n) \Rightarrow \exists c = \frac{1}{4}$ si $n0 = 1$ astfel incat $lg(n!) \geq cnlgn, \forall n \geq n0 \Rightarrow lg(n!) \in \Omega(nlgn)$.

Asadar, $h \in \Omega(nlgn) \Rightarrow$ orice sortare prin comparare necesita $\Omega(nlgn)$ comparatii in cel mai rau caz. (2)

Din (1) si (2) obtinem o contradictie $\Rightarrow$ prespunerea facuta e falsa $\Rightarrow f(n) \in \Omega(nlogn)$.

2

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

Solutie:
$f(n) \in \Theta(g(n)) \leftrightarrow \exists c1', c2', n0' > 0$ astfel incat $\forall n \geq n0'$, $c1'g(n) \leq f(n) \leq c2'g(n)$
$g(n) \in \Theta(h(n)) \leftrightarrow \exists c1'', c2'', n0'' > 0$ astfel incat $\forall n \geq n0''$, $c1''h(n) \leq g(n) \leq c2''h(n)$
$c1''h(n) \leq g(n) \Rightarrow c1'c1''h(n) \leq c1'g(n) \leq f(n) \Rightarrow c1'c1''h(n) \leq f(n)$
$g(n) \leq c2''h(n) \Rightarrow f(n) \leq c2'g(n) \leq c2'c2''h(n) \Rightarrow f(n) \leq c2'c2''h(n)$
Alegem $c1 = c1'c1''$, $c2 = c2'c2''$ si $n0 = \max(n0', n0'')$

$c1h(n) \leq f(n) \leq c2h(n), \forall n \geq n0 \Rightarrow f(n) \in \Theta(h(n))$

3

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

Solutie:
$f(n) = o(g(n)) \leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
$logn = o(\sqrt{n}) \leftrightarrow \lim_{n\to\infty} \frac{logn}{\sqrt{n}} = 0$
$\lim_{n\to\infty} \frac{logn}{\sqrt{n}} = \lim_{n\to\infty} \frac{\frac{1}{n}}{\frac{1}{2\sqrt{n}}} = \lim_{n\to\infty} \frac{2\sqrt{n}}{n} = \lim_{n\to\infty} \frac{2}{\sqrt{n}} = 0 \Rightarrow logn = o(\sqrt{n})$

4

Se da un sir cu n + 1 numere de la 1 la n, cu exceptia unui numar care apare de 2 ori. Determinati numarul care apare de doua ori.

Solutie 1:

1. Parcurgem sirul si, folosind un vector de frecventa de lungime n, numaram de cate ori apare fiecare element in sir (se creste cu 1 numarul aparitiilor in vectorul de frecventa).
2. Parcurgem vectorul de frecventa folosit pentru a determina numarul aparitiilor fiecarui element in sirul initial. Ne oprim atunci cand am intalnit un element care apare de doua ori.

Solutie 2:

Fie x_i, i = $\overline{1, n+1}$ numerele date, si a numarul care apare de doua ori.

1. Parcurgem numerele si calculam suma lor. Fie S = $\sum_{i=1}^{n+1} x_i$.
2. Stim ca S = $(\sum_{i=1}^{n} i) + a$. Dar $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \Rightarrow S = \frac{n(n+1)}{2} + a \Rightarrow a = S - \frac{n(n+1)}{2}$.
3. Asadar, numarul cautat va fi suma tuturor numerelor minus $\frac{n(n+1)}{2}$.

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.

Solutie:
Vom folosi un algoritm Divide et Impera.
La fiecare pas al recursiei:

1. Calculam in $O(1)$ medianele celor doi vectori, fie ele m1 si m2 (vor fi elementele de pe pozitia $[\frac{n}{2}]$ din fiecare vector).

   - Daca m1 = m2: Atunci mediana elementelor din cei doi vectori este chiar m1 = m2.
   - Daca m1 < m2: Atunci cu siguranta mediana celor 2 vectori nu se afla in elementele de pe pozitia 1 pana la pozitia $[\frac{n}{2}]$ din primul vector, si nici in elementele de pe pozitia $[\frac{n}{2}]$ pana la sfarsitul celui de-al doilea vector, asadar putem "elimina" aceste elemente. Se apeleaza recursiv functia pentru subsecventa de la $[\frac{n}{2}]$ la n din X si pentru subsecventa de la 1 la $[\frac{n}{2}]$ din Y.
   - Daca m1 > m2: Atunci cu siguranta mediana celor 2 vectori nu se afla in elementele de pe pozitia $[\frac{n}{2}]$ pana la sfarsitul primului vector, si nici in elementele de pe pozitia 1 pana la pozitia $[\frac{n}{2}]$ in cel de-al doilea vector, asadar putem "elimina" aceste elemente. Se apeleaza recursiv functia pentru subsecventa de la 1 la $[\frac{n}{2}]$ din X si pentru subsecventa de la $[\frac{n}{2}]$ la n din Y.

2. Daca am ajuns sa avem 2 vectori de lungime 1, atunci mediana lor este minimul cele 2 elemente.

Complexitate timp:

T(n) = T(n / 2) + c
Folosind teorema master (a = 1, b = 2, $f(n) = c$):
$c \in \Theta(n^{\log_2 1}) = \Theta(1) \Rightarrow$ T(n) $\in \Theta(n^{\log_2 1} lgn) = \Theta(lgn)$

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 $x_1, x_2, ..., x_k$.

Fie urmatorul algoritm greedy. Parcurgem bunurile in ordinea 1, 2, . . . 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.
2. Fie OPT, numarul de drumuri in solutia optima. Demonstrati ca algoritmul greedy prezentat mai sus efectueaza cel mult 2OPT drumuri.

Solutie:

1. Fie urmatorul exemplu: k = 4, n = 3, x = 2, 2, 1, 1
   In acest caz, algoritmul greedy prezentat mai sus va proceda in felul urmator: va pune in masina primul obiect, de dimensiune 2, va incerca sa il puna si pe al doilea, dar nu va avea loc, asa ca va efectua un transport doar cu primul obiect. In al doilea transport va pune obiectele 2 si 3, de greutate 2 si, respectiv, 1, iar cand va incerca sa puna si obiectul 4, va constata ca nu mai are loc, asa ca va efectua un transport doar cu obiectele 2 si 3. La ultimul drum va transporta obiectul 4, de greutate 1, si se va opri, deoarece nu mai sunt obiecte de transportat. Astfel, algoritmul va genera o solutia cu 3 transporturi. Exista insa posibilitatea de a trasporta obiectele 1 si 3 la un drum, iar obiectele 2 si 4 la un al doilea drum, obtinand astfel o solutie cu doar 2 transporturi. Asadar, algoritmul greedy prezentat mai sus nu este optim.
2. Fie G numarul de drumuri din solutia generata de algoritmul prezentat mai sus.
   Sa consideram varianta problemei in care putem fractiona obiecte (Vom parcurge obiectele in ordine si la fiecare pas punem obiectul in masina. Daca la un moment dat un obiect nu mai incape, il vom "imparti" - adica vom pune in transportul curent o parte din el pana cand se umple masina, iar ceea ce ramane vom pune la drumul

urmator). Atunci numarul de drumuri necesare este $[\frac{S+(n-1)}{n}]$, unde S = $\sum_{i=1}^{k} x_i$. Este clar ca $[\frac{S+(n-1)}{n}] \leq$ OPT.

Fie $x_1, x_2, ..., x_{k_1}$ obiectele transportate in solutia generata de algorimtul greedy la primul drum, $x_{k_1+1}, ..., x_{k_2}$ obiectele transportate la al doilea drum, ..., $x_{k_G-1+1}, ..., x_k$ obiectele transportate la ultimul drum, iar $d_i = n - \sum_{j=k_{i-1}+1}^{k_i} x_j$ (d_i reprezinta practic spatiul ramas liber in masina la al i-lea drum)

Sa consideram acum urmatorul input: n - ul ramane la fel, dar acum avem obiectele cu urmatoarele greutati: $x_1, x_2, .., x_{k_1}, d_1, x_{k_1+1}, ..., x_{k_2}, d_2, ..., x_k$. Acum suma greutatilor obiectelor va fi $S' = \sum_{i=1}^{k} x_i + \sum_{i=1}^{G-1} d_i$. Dar $d_i \leq x_{k_i+1}$ (daca ar fi fost altfel, atunci algoritmul greedy ar fi pus si obiectul x_{k_i+1} in acelasi drum cu obiectele $x_{k_i-1+1}, .., x_{k_i}$) $\Rightarrow \sum_{i=1}^{G-1} d_i \leq \sum_{i=1}^{G-1} x_{k_i+1} \leq \sum_{i=1}^{k} x_i \leq S \Rightarrow S' \leq \sum_{i=1}^{k} x_i + S \leq 2S$. Asadar, varianta in care obiectele pot fi fractionate va genera o solutie cu cel mult $[\frac{2S+(n-1)}{n}]$. Din modul in care a fost construit acest input, se poate observa ca numarul de drumuri pentru aceste date de intrare, in varianta in care pot fractiona obiectele, este egal cu G $\Rightarrow$ G $\leq [\frac{2S+(n-1)}{n}]$. Cum $[\frac{S+(n-1)}{n}] \leq$ OPT $\Rightarrow [\frac{2S+(n-1)}{n}] \leq$ 2OPT, dar si $G \leq [\frac{2S+(n-1)}{n}] \Rightarrow$ OPT $\leq$ G $\leq$ 2OPT.