Tema 2

Mihail Feraru
Grupa 142 - Structuri de Date
UNIVERSIATEA DIN BUCURESTI

May 12, 2020

Problema 1

Fie C un alfabet si T arborele binar corespunzator codului sau optim pentru a coda un set de date oarecare. Pentru orice $c \in C$ consideram $c.freq$ numarul de aparitii al lui c in setul de date ce trebuie codat, iar $d_T(c)$ lungimea codarii lui c. Prin cod optim ne referim la un cod cu urmatoarele doua proprietati:

> **Proprietatea 1.** Oricum am alege doua reprezentari a doua elemente, niciuna din reprezentari nu este prefix pentru cealalta.
>
> **Proprietatea 2.** Costul codului definit astfel: $B(T) = \sum_{c \in C} c.freq \cdot d_T(c)$ este minim.

Presupunem prin absurd ca exista un arbore binar T' care nu este plin si corespunde unui cod optim pentru C. Cum T' nu este plin, atunci exista un nod x care are un singur fiu, y care este asociat unui element din C. Fara a incalca **proprietatea 1**, putem interschimba x si y, obtinem astfel un arbore notat T''. Cum $d_{T''}(y) < d_{T'}(y) \implies B(T'') < B(T')$, dar T' corespunde unui cod optim, contradictie.

In concluzie, orice arbore arbore asociat unui cod optim trebuie sa fie plin. □

Problema 2

Algoritmul de sortare quicksort ruleaza in cazul cel mai defavorabil in timp $O(n^2)$ din cauza alegerii nebalansate a pivotului. Recurenta generala este $T(n) = T(n-p-1)+T(p-1)+\Theta(n)$, unde p reprezinta pozitia pivotului in sirul sortat, iar $\Theta(n)$ este timpul pentru efectuarea partitionarii. Observam ca daca pivotul ar coincide cu mediana sirului, recurenta se reduce la $T(n) = 2T([n/2]) + \Theta(n)$, care, aplicand teorema Master, ruleaza in timp $O(n \log n)$ in cel mai nefavorabil caz.
Pentru a atinge complexitatea dorita este necesara determinarea medianei unui sir nesortat in $O(n)$. Acest lucru se poate realiza prin aplicarea algoritmului descris in *Introduction to Algorithms, capitolul 9, sectiunea 2* implementat in metoda **RANDOMIZED-SELECT**. Pe scurt, este o adaptare a algoritmului quicksort care selecteaza a n-a valoare daca sirul s-ar sorta, fara ca acesta sa fie sortat. In cele din urma, obtinem un quicksort care ruleaza in $O(n \log n)$, insa selectarea pivotului in modul descris anterior aduce o constanta semnificativa in ecuatie, algoritmul devenind impractic intr-o gama larga de cazuri.

Problema 3

Fie x un nod dintr-un arbore binar de cautare. Definim $x.left$ si $x.right$ cei doi fii ai sai, $x.val$ valoarea sa, iar $NEXT(x)$ si $PREV(x)$ succesorul si predecesorul sau in parcurgerea in-order.

Propozitia 3. Predecesorul lui x nu are fiu drept.
Demonstratie. Presupunem prin absurd ca exista un nod y astfel incat $PREV(x).right = y$. Atunci, $y.val \geq PREV(x).val$ si $y.val \leq x.val$, deci $PREV(x) = y$, contradictie.

Propozitia 4. Succesorul lui x nu are fiu stang.
Demonstratie. Presupunem prin absurd ca exista un nod y astfel incat $NEXT(x).left = y$. Atunci, $y.val \leq NEXT(x).val$ si $y.val \geq x.val$, deci $NEXT(x) = y$, contradictie.

Consecinta directa a **propozitiilor 3 si 4** este ca predecesorul lui x nu poate avea fiu drept, iar succesorul nu poate avea fiu stang. $\square$

Problema 4

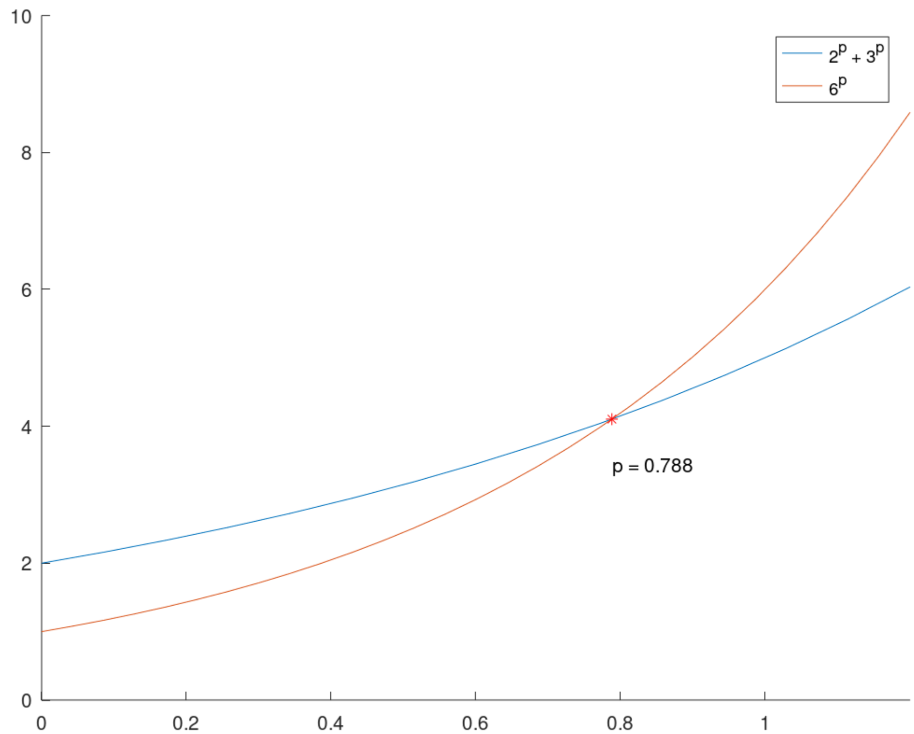
Vom folosi metoda Akra-Bazzi pentru rezolvarea recurentei. Forma generala este descrisa astfel:

$$T(x) = g(x) + \sum_{i=1}^{k} a_i T(b_i x + h_i(x))$$

Pentru recurenta $T(n) = T(n/2) + T(n/3) + 1$ avem:

$$g(n) = 1 \in O(n^c), c \in \mathbb{R}, \quad a_i = 1, \quad h_i(n) = 0 \ \ \forall i \in \mathbb{N}, \quad b_1 = \frac{1}{2}, \ b_2 = \frac{1}{3}, \quad k = 2$$

Urmatorul pas al metodei este rezolvarea ecuatiei $\sum_{i=1}^{k} a_i b_i^p = 1 \iff (\frac{1}{2})^p + (\frac{1}{3})^p = 1 \iff 2^p + 3^p = 6^p$. Vom utiliza reprezentarea grafica pentu a obtine o aproximare cat mai buna a lui p.

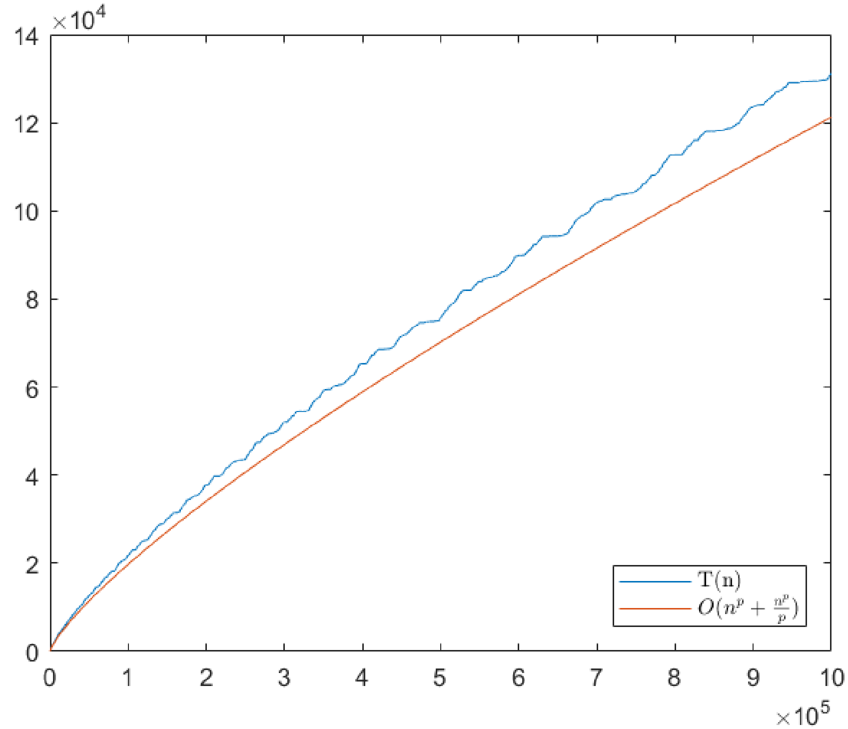


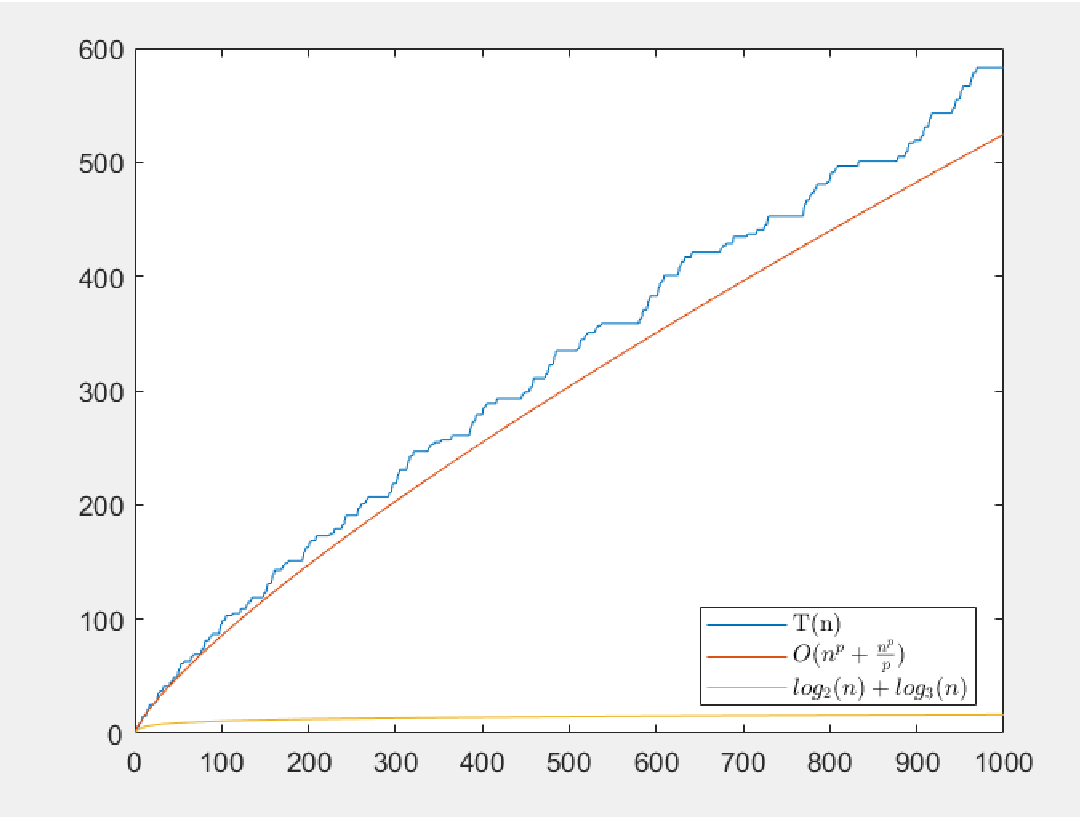
$$I = \int_1^n \frac{g(n)}{u^{p+1}} du = \int_1^n \frac{1}{u^{p+1}} du = \int_1^n u^{-(p+1)} du = \frac{u^{-p}}{-p}\Big|_1^n = \frac{1}{-pu^p}\Big|_1^n \ \ n \underset{=}{\to} \infty \ \ \frac{1}{p}$$

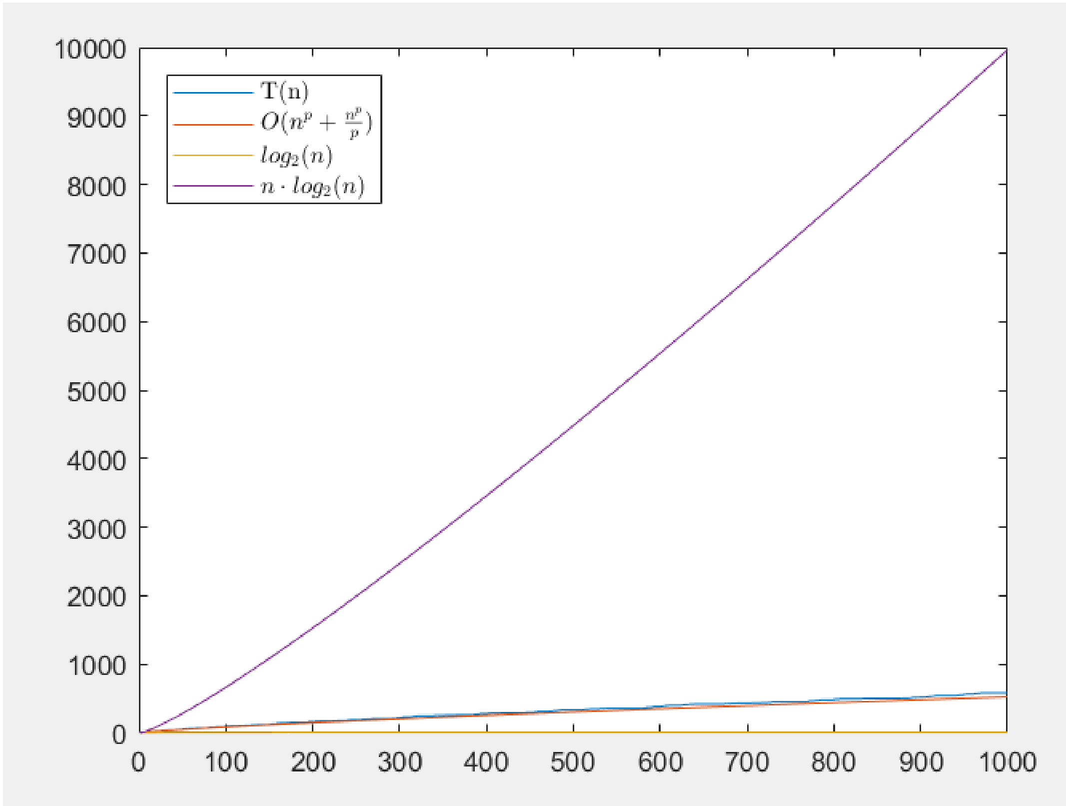
Conform metodei utilizate, $T(n)$ se incadreaza in clasa asimptotica:

$$T(n) \in \Theta(n^p(1+I)) \iff T(n) \in \Theta(n^p + \frac{n^p}{p}) \implies T(n) \in \Theta(n^p) = \Theta(n^{0.78})$$

Compararea complexitatii lui T(n) cu diferite clase






10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
100
200
300
400
500
600
700
800
900
1000
T(n)
$O(n^p + \frac{n^p}{p})$
$log_2(n)$
$n \cdot log_2(n)$