Model Examen – Seria 14
Rezolvari

1 Exprimarea în notația Θ

Cerinta: Exprimati functiile urmatoare in notatie Θ:

a. $\sqrt{n} + \log n + n$. Gasim termenul dominant:

- $\underset{n\to\infty}{lim} \frac{\sqrt{n}}{n} = 0 \to \sqrt{n} \in o(n)$.
- $\underset{n\to\infty}{lim} \frac{\log n}{n} = 0 \to \log n \in o(n)$.

$\to \sqrt{n} + \log n + n \in \Theta(n)$.

b. $(\log n^{100})^3 + n$. Gasim termenul dominant:

- $\underset{n\to\infty}{lim} \frac{(\log n^{100})^3}{n} = \underset{n\to\infty}{lim} \frac{(100 \log n)^3}{n} = 10^6 * \underset{n\to\infty}{lim} \frac{\log^3 n}{n} = 0 \to (\log n^{100})^3 \in o(n)$.

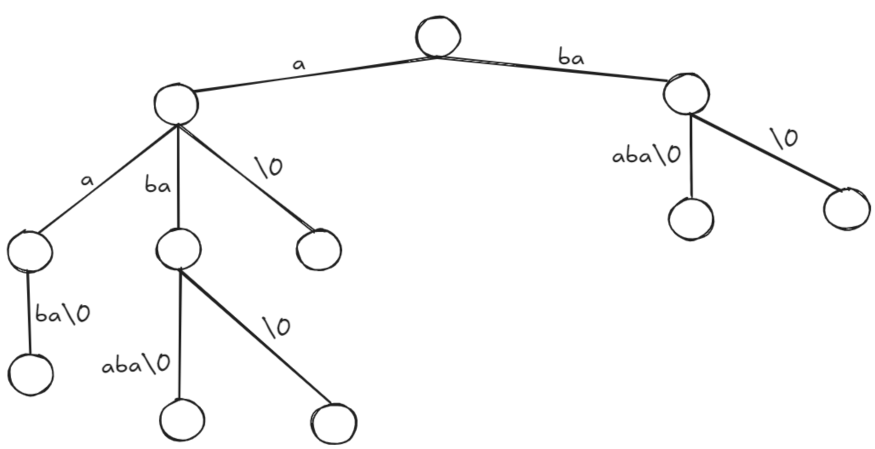
$\to (\log n^{100})^3 + n \in \Theta(n)$.

c. $\log(n^n) = n \log n \in \Theta(n \log n)$.

2 Suffix Trees si Suffix Arrays

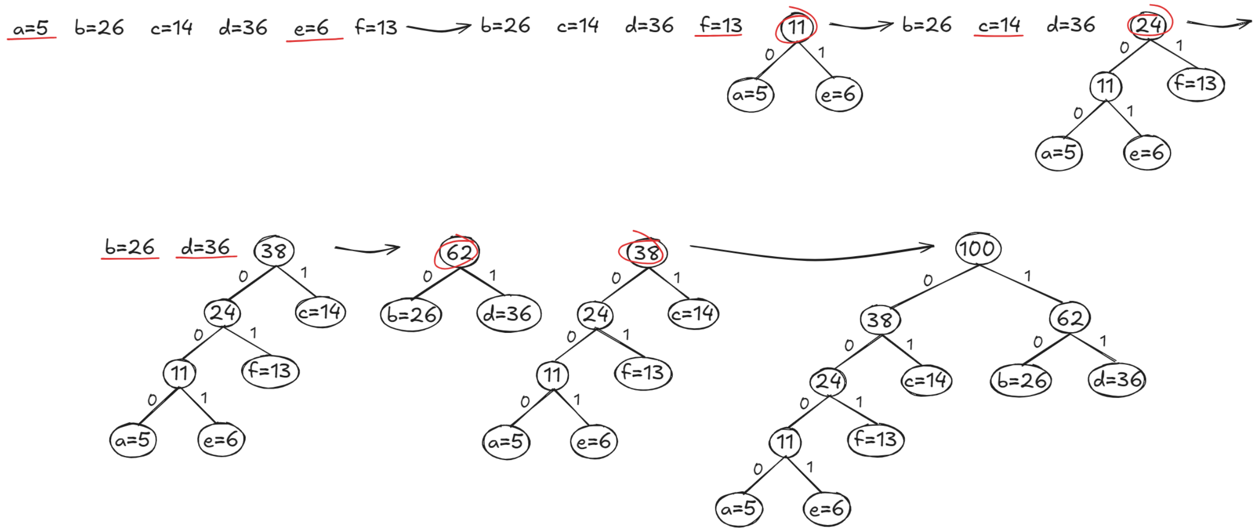
Cerinta: Construiti *suffix tree* si *suffix array* pentru sirul *abaaba*.

Sufixele pentru sirul *abaaba* sunt urmatoarele: $0 : abaaba$, $1 : baaba$, $2 : aaba$, $3 : aba$, $4 : ba$, $5 : a$. Sortandu-le in ordine lexicografica, obtinem suffix array-ul $5 : a$, $2 : aaba$, $3 : aba$, $0 : abaaba$, $4 : ba$, $1 : baaba$. De asemenea, am atasat suffix tree-ul desenat.



3 Arbori Huffman

Cerinta: Sa se deseneze arborele Huffman pentru literele urmatoare ce au frecventele a = 5, b = 26, c = 14, d = 36, e = 6, f = 13. Scrieti si codul optim (binar) pentru fiecare litera.

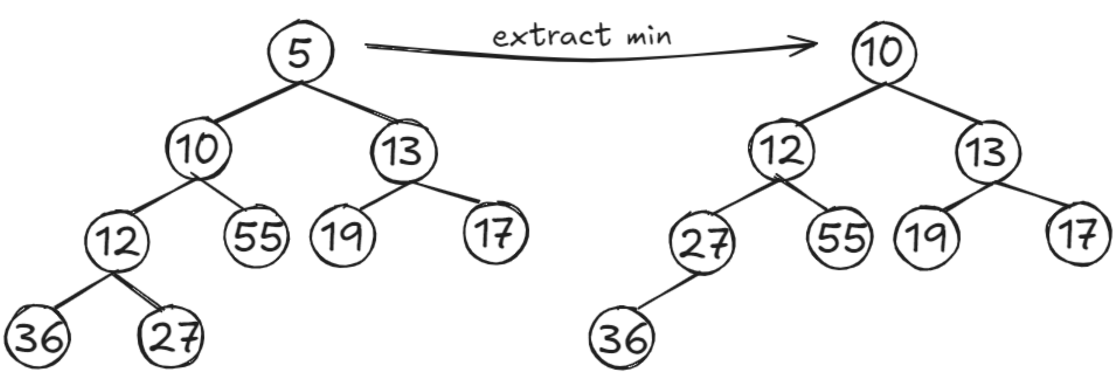


Codurile pentru litere sunt urmatoarele:

- a = 0000
- b = 10
- c = 01
- d = 11
- e = 0001
- f = 001

4 Heaps

Cerinta: Sa se construiasca un min-heap obtinut prin insertia pe rand a urmatoarelor chei, iar apoi sa se extraga radacina din arborele rezultat: 10, 5, 19, 36, 55, 13, 17, 12, 27.



5 Complexitati

Cerinta: Demonstrati ca daca $f(n) \in O(g(n))$ si $g(n) \in O(h(n))$, atunci $h(n) \in \Omega(f(n))$.

Daca $f(n) \in O(g(n))$, atunci stim ca de la un input n_0 incolo, vom avea $f(n) \le c * g(n)$, pentru orice input $n \ge n_0$ si o constanta $c > 0$. In mod similar, $g(n) \le c * h(n)$. Astfel, obtinem ca pentru orice input de la un n_0 incolo, vom avea $h(n) \ge c * f(n)$, unde $c > 0$. Acea inegalitate este echivalenta cu $h(n) \in \Omega(f(n))$ (h este marginit inferior de f).

6 Recurente

Cerinta: Rezolvati recurenta $T(n) = 4T(\frac{n}{3})+n$. Demonstrati prin inductie ca rezultatul este corect (arborele de recurenta si teorema master nu sunt considerate demonstratii).

In primul rand, ne putem folosi (nu pentru demonstratie) de teorema master pentru a avea un reper de unde sa pornim: $a = 4, b = 3 \rightarrow \log_b a = \log_3 4$. Incadram: $1 < \log_3 4 < 2$.
Stim ca $n \in O(n^1) \rightarrow 1 < \log_3 4 \rightarrow$ conform teoremei master, avem ca $T(n) \in O(n^{\log_3 4})$. Acum trebuie sa demonstram acest lucru.
Reformulam ceea ce vrem sa demonstram: $T(n) \in O(n^{\log_3 4}) \leftrightarrow T(n) \le c * n^{\log_3 4}$, unde $c > 0$. Consideram cazul de baza al recursivitatii ca fiind $n = 1$ si verificam: $T(0) \le c * 1^{\log_3 4}$; consideram $c \ge T(0)$ ca fiind adevarat, ca sa avem cazul de baza corect.
Presupunem ca este adevarat faptul ca: $\forall k < n \rightarrow T(k) \le c * k^{\log_3 4}$. Cum $\frac{n}{3} < n$, obtinem ca $T(\frac{n}{3}) \le c * (\frac{n}{3})^{\log_3 4}$ este adevarat. Astfel, il inlocuim pe $T(\frac{n}{3})$ in relatia originala si obtinem $T(n) = 4T(\frac{n}{3}) + n \le 4 * c * (\frac{n}{3})^{\log_3 4} + n = \frac{4}{3^{\log_3 4}} * c * n^{\log_3 4} + n = c * n^{\log_3 4} + n$. Stim ca $n \in o(n^{\log_3 4})$, pentru ca $\lim\limits_{n \to \infty} \frac{n}{n^{\log_3 4}} = 0$, deci n este un termen neglijabil pe care il inlocuim cu $n^{\log_3 4}$, iar inegalitatea tot ramane adevarata: $T(n) \le c * n^{\log_3 4} + n^{\log_3 4} = n^{\log_3 4}(c + 1) \rightarrow T(n) \in O(n^{\log_3 4})$.
Putem sa mergem mai departe si sa demonstram ca $T(n) \in \Omega(n^{\log_3 4}) \leftrightarrow T(n) \ge c * n^{\log_3 4}$, unde $c > 0$. Consideram cazul de baza ca fiind adevarat: $T(0) \ge c$.
Presupunem ca este adevarat faptul ca: $\forall k < n \rightarrow T(k) \ge c * k^{\log_3 4}$. Cum $\frac{n}{3} < n$, este adevarat ca $T(\frac{n}{3}) \ge c * (\frac{n}{3})^{\log_3 4}$. Ne folosim de asta in relatia initiala: $T(n) = 4T(\frac{n}{3}) + n \ge 4 * c * (\frac{n}{3})^{\log_3 4} + n = \frac{4}{3^{\log_3 4}} * c * n^{\log_3 4} + n = c * n^{\log_3 4} + n \ge c * n^{\log_3 4} \leftrightarrow T(n) \in \Omega(n^{\log_3 4})$.
Deoarece $T(n) \in \Omega(n^{\log_3 4})$ si $T(n) \in O(n^{\log_3 4})$, obtinem $T(n) \in \Theta(n^{\log_3 4})$.

7 Cod - Primul exercitiu

Cerinta: Se da un sir S cu n caractere. Sa se gaseasca cel mai lung subsir care apare de cel putin doua ori in S. In functie de timpul de rulare al algoritmului, se primesc urmatoarele punctaje: $O(n^3)$ - 0.5 puncte, $O(n^2)$ - 0.75 puncte, $O(n \log n)$ - 1 punct, $O(n)$ - 1.5 puncte.

Daca am avea un suffix tree pentru sirul din input, am putea proceda in felul urmator: la inceput, consideram ca cel mai lung sir este ""; in functie de nodul in care ne aflam, exista urmatoarele cazuri:

- Daca nodul curent are doi sau mai multi copii, asta inseamna ca drumul (concatenarea muchiilor) pana in nodul respectiv este un prefix comun pentru cel putin doua sufixe $\rightarrow$ verificam daca subsirul/prefixul obtinut are o lungime mai mare decat cel determinat pana in momentul respectiv. In caz afirmativ, am gasit o solutie partiala mai buna.
- Daca nodul curent este o frunza, nu facem nimic.
- Acest proces se repeta pentru toate nodurile din arbore (de exemplu cu DFS).

Cum putem construi un suffix tree? Un mod simplu este, mai intai, sa consideram ca avem o singura frunza care contine primul sufix, adica $s[1 \ldots n]$. Trecem la urmatorul sufix: $s[2 \ldots n]$, iar aici exista doua cazuri:

- Cele doua sufixe **nu** au un prefix comun $\rightarrow$ trebuie sa adaugam o noua muchie din radacina catre o noua frunza, cu noul sufix.
- Cele doua sufixe au un prefix comun $\rightarrow$ muchia ia valoarea prefixului, iar frunza devine nod intern cu doua frunze: in stanga este vechiul sufix (fara prefixul comun), iar in dreapta este noul sufix (fara prefixul comun).

Acest proces se repeta si pentru restul sufixelor pe care le adaugam. Totusi, are complexitate $O(n^2)$, unde $n = len(s)$. In schimb, putem folosi algoritmul *Ukkonen* pentru a construi un suffix tree in $O(n)$, iar apoi aplicam parcurgerea discutata.

8 Cod - Al doilea exercitiu

Cerinta: Se da un sir $A[1 \dots n]$ de numere naturale pozitive (strict mai mari decat 0). Numim o secventa $A[i \dots j]$, $1 \leq i \leq n$, speciala daca cifra de control a sumei secventei este 9. Sa se determine numarul de secvente speciale in A.

Notam cu $c(x)$ cifra de control a lui x si $sum(x)$ suma cifrelor lui x. Atunci, $c(x)$ se defineste astfel:

- daca $x \leq 9$, atunci $c(x) = x$.
- daca $x > 9$, atunci $c(x) = c(sum(x))$.

Exemplu: $n = 7$ si $A = [1, 7, 6, 1, 11, 5, 9]$. Numarul de secvente speciale este 2. Cele doua secvente speciale sunt $A[3 \dots 5]$ (suma secventei este 18, iar $c(18) = 9$), $A[7 \dots 7]$ (suma secventei este 9, iar $c(9) = 9$).

In functie de timpul de rulare al algoritmului, se primesc urmatoarele punctaje: $O(n^3)$ - 0.5 puncte, $O(n^2)$ - 1 punct, $O(n)$ - 1.5 puncte.

Ca sa determinam eficient daca un numar are sau nu cifra de control egala cu 9, trebuie sa avem in vedere:

- Un numar este divizibil cu 9 daca suma cifrelor sale este egala cu 9.
- Cand calculam suma cifrelor unui numar natural cu cel putin doua cifre, aceasta va fi mereu mai mica decat numarul in sine. Astfel, aplicand recursiv acest proces, mereu vom ajunge la un numar cu o singura cifra.

Combinand logica de la cele doua puncte de mai sus, asta inseamna ca un numar divizibil cu 9 are cifra de control egala cu 9. In alte cuvinte, daca avem un numar **x**, atunci $x \texttt{ mod } 9 = 0 \leftrightarrow sum\_digits(x) \texttt{ mod } 9 = 0 \leftrightarrow sum\_digits(sum\_digits(x)) \texttt{ mod } 9 = 0 \leftrightarrow \dots$, iar apelul recursiv pentru suma cifrelor se va opri cand se va ajunge la un numar cu o singura cifra (singurele numere de o cifra divizibile cu 9 sunt 0 sau 9).

Astfel, am modificat putin sarcina problemei: trebuie sa determinam numarul de subsecvente (elemente consecutive) care au suma elementelor divizibila cu 9, adica $sum(A[i \dots j]) \ \% \ 9 = 0$.

Totusi, putem sa mai rescriem formula: stim ca $sum(A[i \dots j]) = A[i] + \dots + A[j] = (A[0] + \dots + A[i] + \dots + A[j]) - (A[0] + \dots + A[i-1]) = prefix\_sum[j] - prefix\_sum[i-1]$, care trebuie sa fie divizibil cu 9.

Sa presupunem ca avem $(a - b) \ \% \ 9 = 0 \rightarrow a - b = 9k,\ k \in Z \rightarrow a = b + 9k \rightarrow a \ \% \ 9 = (b + 9k) \ \% \ 9 = b \ \% \ 9$. Astfel, ultima relatie poate fi reformulata: $(prefix\_sum[j] - prefix\_sum[i-1]) \ \% \ 9 = 0 \leftrightarrow prefix\_sum[j] \ \% \ 9 = prefix\_sum[i-1] \ \% \ 9$. Un mic detaliu: daca $i = 0$, atunci spunem ca $prefix\_sum[i-1] = 0$.

In concluzie, pentru $prefix\_sum[j]$ vrem sa numaram pentru cate valori anterioare $0 \leq i \leq j - 1$ avem $prefix\_sum[j] \ \% \ 9 = prefix\_sum[i] \ \% \ 9$. Destul de simplu: folosim un vector de frecventa.

Pasii algoritmului au devenit:

1. Consideram variabila $prefix\_sum = A[0]$ si vectorul de frecvente $freq[i] = 0,\ \forall i \in [0, n-1]$ si $freq[prefix\_sum \ \% \ 9] = 1$.

2. Iteram prin elementele vectorului, incepand de la al doilea element (indexul 1). Calculam $prefix\_sum = prefix\_sum + A[j]$. Intr-o variabila $count$ vom stoca numarul de perechi pe care le putem forma intre indecsi din trecut si indexul curent: $count = count + freq[prefix\_sum \ \% \ 9]$, si apoi $freq[prefix\_sum \ \% \ 9]++$.

3. Afisam $count$.