Algoritmi
Študijsko leto 2021/2022

2. domača naloga

Poročilo domače naloge

Ivo Pajer
Vpisna št. 63180218

Ljubljana, 3. april 2022

Kazalo

1 Problem 1 - Ujemanje vzorcev

1.1 A - KMP preponska funkcija

Za podan vzorec s moramo izračunati KMP preponsko funkcijo π.

$$s = abababbabbababababbabb$$

KMP preponska funkcija je definirana kot seznam števil dolžine n, kjer je $\pi[k]$ dolžina najdaljše predpone podniza $s[0...k]$. Rezultat funkcije je torej:

$$\pi = [0, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 3, 4, 3, 4, 5, 6, 7, 8]$$

1.2 B - končni avtomat

Prva naloga pri B delu je narediti končni avtomat, ki bo sprejemal pojavitve vzorca **AUGUGG**. Avtomat bomo predstavili kot:

- množica stanj $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_f\}$
- začetno stanje - q_0
- vhodna abeceda $\sum = \{A, U, C, G\}$
- množica končnih stanj $F = \{q_f\}$
- prehoda funkcija - δ

Tabela funkcije δ:

Stanje \Vhod	**A**	**U**	**C**	**G**
q0	q1	q0	q0	q0
q1	q1	q2	q0	q0
q2	q1	q0	q0	q3
q3	q1	q4	q0	q0
q4	q1	q0	q0	q5
q5	q1	q0	q0	qf
qf	q1	q0	q0	q0

Druga naloga v B delu, pa je, da naredimo končni avtomat ki bo sprejemal nize **AAUAUG** in **AACAUG**. Avtomat bomo predstavili kot:

- množica stanj $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_f\}$
- začetno stanje - q_0
- vhodna abeceda $\sum = \{A, U, C, G\}$
- množica končnih stanj $F = \{q_f\}$
- prehoda funkcija - δ

Tabela funkcije δ:

Stanje \Vhod	**A**	**U**	**C**	**G**
q0	q1	q0	q0	q0
q1	q2	q0	q0	q0
q2	q2	q3	q3	q0
q3	q4	q0	q0	q0
q4	q1	q5	q0	q0
q5	q1	q0	q0	qf
qf	q1	q0	q0	q0

2 Problem 2 - IP posredovanje in vEB drevesa

2.1 A - Master teorem

Naša naloga je, da z uporabo teorema Master izračunamo časovno zahtevnost operacije *vEB-Tree-Successor* vEB drevesa.
Teorem Master pravi:
$T(n) = aT(\frac{n}{b}) + \mathcal{O}(n^c)$
kjer $a \geq 1, b \geq 2, d \geq 0$, potem

$$T(n) = \begin{cases} \Theta(n^d), \text{ če } a < b^d \\ \Theta(n^d log n), \text{ če } a = b^d \\ \Theta(n^{log_b a}), \text{ če } a > b^d \end{cases}$$

- a - nam pove, koliko podproblemov problem razdelimo
- b - nam pove, koliko manjši je naš pod-problem
- d - nam pove, zahtevnost operacije deljenja

Predpostavili smo, da je univerzum dolžine M, ta pa je oblike 2^k. Koren eVB drevesa ima seznam $\sqrt{M}$ otrok. ker je delitev konstantne zahtevnosti, lahko sklepamo, da je d enak 0. Vsak podproblem je torej reda $\sqrt{M} = \mathcal{O}(M^{\frac{1}{2}})$, in ker je M oblike 2^k, lahko sklepamo, da je podproblem reda $\mathcal{O}(2^{\frac{k}{2}})$, ker pomeni, da je vrednost b enaka 2. Vrednost a je 1, ker problem razdelimo na 1 podproblem. Enačba master theorema se zato glasi:

$$T(k) = T(\frac{k}{2}) + \mathcal{O}(1)$$

Iz a,b,c, lahko sklepamo, da imamo drugi scenarij, saj $1 = 2^0$. Iz tega nato sledi, da

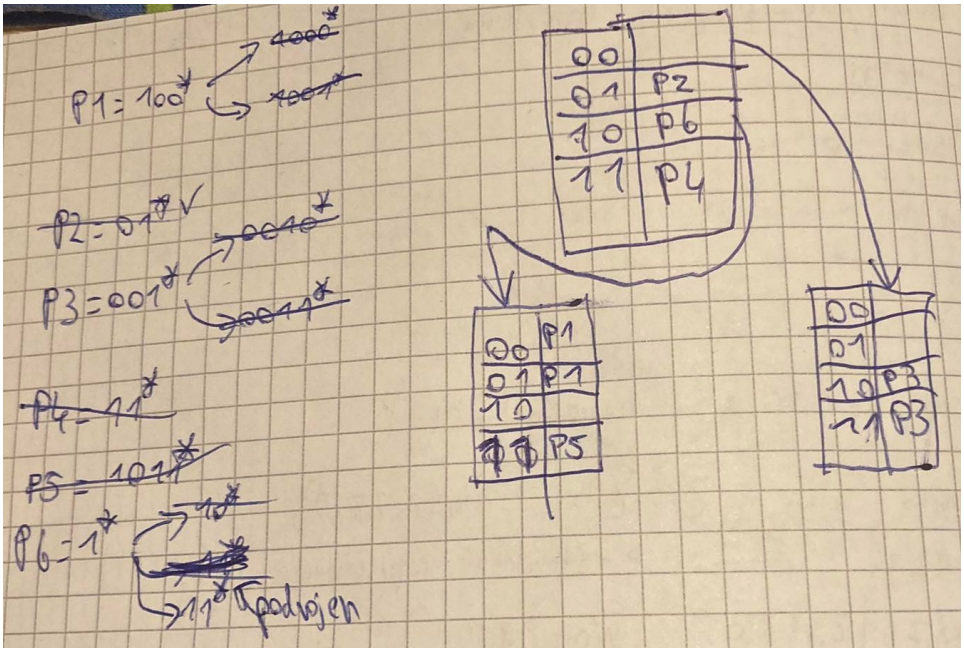
$$T(k) = \mathcal{O}(k^0 log k) = \mathcal{O}(loglog M)$$

2.2 B - Luela algoritem

Pri naivnemu pristopu štetju bitov z Luela algoritmom potrebujemo 3 pomn. reference. Naša naloga je, da razložimo, kako prva dva dostopa lahko združimo v enega.
Luela algoritem kot že omenjeno zahteva 3 pomn. reference in sicer na: povzetkovno tabelo, bitno polje in polje vozlišča. Bitno polje razdelimo na enako velike kose, kar nam omogoči, da ti tabeli združimo v eno tabelo s tem, da podatke na istem nivoju konkatiniramo med seboj. Rezultat je ena tabela, ki vsebuje podatke obeh tabel skupaj, kar zniža pomnilniške dostope iz 3 na 2.

2.3 C - fixed stride številsko drevo

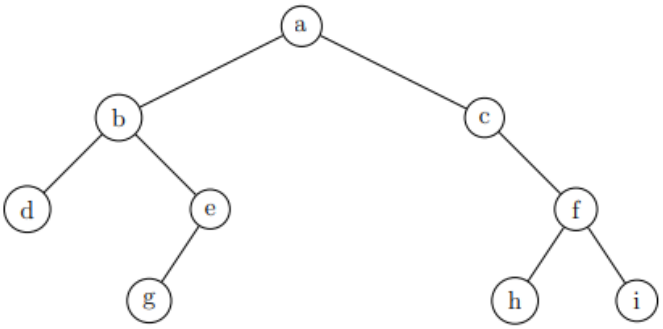
Naša naloga je sestaviti fixed stride številsko drevo s stride velikostjo 2 za omrežja 100*, 01*, 001*, 11*, 1011* in 1*.



Slika 1: Fixed stride številsko drevo

3 Problem 3 - Kompaktne podatkovne strukture

Podano imamo drevo:



Slika 2: Podano drevo

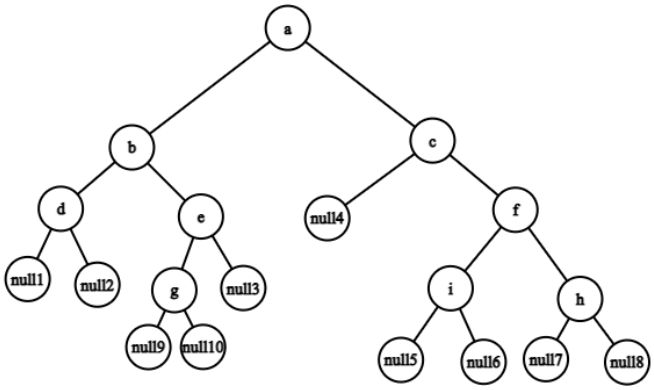
3.1 A - Ordinalno BP in LODUS

Ordinalna drevesa so takšna, da imajo vozlišča razvrščena po vrsti (angl. ordered), v našem primeru je drevo urejeno po abecedi.

BP: ((()(()))(((()))))
LOUDS: 1011011010010110000

3.2 B - Kardinalno BP in LODUS

Kardinalna drevesa imajo konstantno število otrok. Ti so ali prazni ali pa imajo svoje otroke.
BP: ((((()))(((()))()))(()((()))(()())))
LOUDS: 10110110110110110011000110011010110000000



Slika 3: Kardinalno drevo

3.3 C - rmM-drevo za b=8

3.4 D - Funkcija Izberi(T,i)

Naloga je napisati algoritem **Izberi(T, i)**, ki bo našel i-ti element v razširjenem iskalnem dvojiškem drevesu. Naše drevo bo T. i bo predstavljal indeks, ki ga iščemo (prvi element ima indeks 0). Drevo ima v vsakem vozlišču podatek, kolikšna je moč levega pod-drevesa (koliko vrednosti je manjših od vrednosti v vozlišču), ki je označen z |L|. Funkcija vrne T.value oz. vrednost vozlišča, če je indeks enak |L|. Če je indeks manjši od |L|, potem vemo, da je iskana vrednost v levem poddrevesu. V nasprotnem primeru pa iščemo naprej v desnem poddrevesu, s tem, da indeksu odštejemo |L| + 1.

Algorithm 1 Izberi

```
1: function IZBERI(T, i)
2:     if i == |L| then
3:         return T.value                          ▷ Vrednost i-tega elementa
4:     else
5:         if i < |L| then
6:             return Izberi(T.left, i)
7:         else
8:             return Izberi(T.right, i − |L| − 1)
```