Druga domača naloga pri algoritmih

Lucija Fekonja

7. april 2024

Naloga 1: Ujemanje vzorcev

KMP predponska funkcija za vzorec `p = ababbabbabbababababbabb` je

$$\pi(p) = 00120120120123434567 8.$$

Spodaj podajam tabelo za prehodno funkcijo δ za končni avtomat, ki poišče vse pojavitve vzorca `AUGUGG`.

q	S	q'	S'	Δ
start	A	A1	#	$\rightarrow$
start	U	start	O	$\rightarrow$
start	G	start	O	$\rightarrow$
start	C	start	O	$\rightarrow$
start	␣	end	␣	-
delete	#	start	O	$\rightarrow$
delete	A	delete	A	$\leftarrow$
delete	U	delete	U	$\leftarrow$
delete	G	delete	G	$\leftarrow$
delete	C	delete	C	$\leftarrow$
A1	U	U1	U	$\rightarrow$
A1	A	brisi	A	$\leftarrow$
A1	G	brisi	G	$\leftarrow$
A1	C	brisi	C	$\leftarrow$
A1	␣	brisi	␣	$\leftarrow$
U1	G	G1	G	$\rightarrow$
U1	A	brisi	A	$\leftarrow$
U1	U	brisi	U	$\leftarrow$

q	S	q'	S'	Δ
U1	C	brisi	C	$\leftarrow$
U1	␣	brisi	␣	$\leftarrow$
G1	U	U2	U	$\rightarrow$
G1	A	brisi	A	$\leftarrow$
G1	G	brisi	G	$\leftarrow$
G1	C	brisi	C	$\leftarrow$
G1	␣	brisi	␣	$\leftarrow$
U2	G	G2	G	$\rightarrow$
U2	A	brisi	A	$\leftarrow$
U2	U	brisi	U	$\leftarrow$
U2	C	brisi	C	$\leftarrow$
U2	␣	brisi	␣	$\leftarrow$
G2	G	start	G	$\rightarrow$
G2	A	brisi	A	$\leftarrow$
G2	U	brisi	U	$\leftarrow$
G2	C	brisi	C	$\leftarrow$
G2	␣	brisi	␣	$\leftarrow$

Stroj začne v stanju `start`. Sprehodi se od leve proti desni in si shrani začetke iskanega podniza z `#`. Vse ostale znake v začetnem stanju označi z `O`. Če v nadaljevanju ugotovi, da opazovan podniz ni iskan podniz, se vrne levo do `#`, jo spremeni v `O` in začne znova pri naslednjem znaku. Če so vsi znaki ravno znaki iskanega niza, si jih shrani na trak in ponovno začne iskanje. Proces se konča, ko pride do konca vhodnega niza.

Iskanje aspargina in metionina oziroma podnizov `AAUAUG` in `AACAUG` poteka na podoben način. Število stanj ostaja enako, saj lahko iz stanja `A2`, ko najde drugi `A`, preide v stanje `UC`, kjer preverja ali je opazovan znak `U` ali `C`. Če je katerikoli drugi, gre v stanje `delete`. Spodaj podajam še prehodno funkcijo za iskanje nizov `AAUAUG` in `AACAUG`.

q	S	q'	S'	Δ
start	A	A1	#	$\rightarrow$
start	U	start	O	$\rightarrow$
start	G	start	O	$\rightarrow$
start	C	start	O	$\rightarrow$
start	_	end	_	-
delete	#	start	O	$\rightarrow$
delete	A	delete	A	$\leftarrow$
delete	U	delete	U	$\leftarrow$
delete	G	delete	G	$\leftarrow$
delete	C	delete	C	$\leftarrow$
A1	A	A2	A	$\rightarrow$
A1	U	brisi	U	$\leftarrow$
A1	G	brisi	G	$\leftarrow$
A1	C	brisi	C	$\leftarrow$
A1	_	brisi	_	$\leftarrow$
A2	U	CU	U	$\rightarrow$
A2	C	CU	C	$\leftarrow$
A2	A	brisi	A	$\leftarrow$

q	S	q'	S'	Δ
U1	G	brisi	G	$\leftarrow$
U1	_	brisi	_	$\leftarrow$
CU	A	A3	A	$\rightarrow$
CU	U	brisi	U	$\leftarrow$
CU	G	brisi	G	$\leftarrow$
CU	C	brisi	C	$\leftarrow$
CU	_	brisi	_	$\leftarrow$
A3	U	U1	U	$\rightarrow$
A3	A	brisi	A	$\leftarrow$
A3	G	brisi	G	$\leftarrow$
A3	C	brisi	C	$\leftarrow$
A3	_	brisi	_	$\leftarrow$
U1	G	start	G	$\rightarrow$
U1	A	brisi	A	$\leftarrow$
U1	U	brisi	U	$\leftarrow$
U1	C	brisi	C	$\leftarrow$
U1	_	brisi	_	$\leftarrow$

Končni avtomat, ki ustreza zgornji tabeli in program za iskanje vseh indeksov pojavitev `AAUAUG` in `AACAUG`, je v datoteki `FSM.ipynb`.

Naloga 2: IP posredovanje in vEB drevesa

Časovna zahtevnost operacije `vEB-Tree-Successor`

Algoritem za iskanje naslednika elementa x v van Emde Boas drevesu T je sledeč

function vEB-Tree-Successor (T, x)
  if x < T.min **then**
    return T.min
  if x ≥ T.min **then**
    return M
  i = floor(x /$\sqrt{\text{M}}$)
  l = x mod $\sqrt{\text{M}}$
  if l < T.children[i].max **then**
    return ($\sqrt{\text{M}}$ i) + vEB-Tree-Successor (T.children[i], l)
  j = vEB-Tree-Successor (T.aux, i)
  return ($\sqrt{\text{M}}$ i) + T.children[j].min

Tukaj je T.min minimalni element v drevesu T, T.max je maksimalni element in T.children[i] je i-ti otrok drevesa T. T.aux shranjuje katera poddrevesa niso prazna, torej T.aux vsebuje vrednost j, če T.children[j] ni prazno.

Poiščimo najprej rekurzivno zvezo za $T(m)$, kjer je $M = 2^m$ velikost univerzalne množice. Vemo, da osnovne računske operacije, kot so seštevanje, primerjanje, korenjenje in zaokroževanje, delujejo v konstantnem času $O(1)$. Tudi T.min, T.max in T.aux vračajo v konstantnem času, saj so atributi vEB drevesa.

Rekurzija se v algoritmu pojavi na dveh mestih - pri klicu vEB-Tree-Successor (T.children[i], l) in pri vEB-Tree-Successor (T.aux, i) - vendar se nikoli ne izvedeta oba klica v isti izvedbi algoritma. Prvi klic se izvede, ko je l < T.children[i].max, drugi pa če l ≥ T.children[i].max. Prav tako za vEB drevo velikosti m velja, da so otroci korena velikosti $m/2$. Torej če se algoritem izvaja na drevesu velikosti m, se zgornja klica izvajata na drevesih velikosti $m/2$.

Rekurzivna zveza za $T(m)$ se glasi

$$T(m) = T(\frac{m}{2}) + c,$$

kjer je c konstanta.

Izrek Master pravi, da če velja zveza $T(n) = aT(\frac{n}{b}) + f(n)$, lahko s primerjanjem $f(n)$ ugotovimo časovno zahtevnost algoritma. V našem primeru sta $a = 1$ in $b = 2$. Računamo

Primer 1: $f(m) = O(1) = O(m^{\log_b a - \epsilon}) = O(m^{\log_2 1 - \epsilon}) = O(m^{-\epsilon})$

Primer 2: $f(m) = O(1) = \Theta(m^{\log_b a}) = \Theta(m^{\log_2 1}) = \Theta(m^0) = \Theta(1)$

Primer 3: $f(m) = O(1) = \Omega(m^{\log_b a + \epsilon}) = \Omega(m^{\log_2 1 = \epsilon}) = \Omega(m^{\epsilon})$

Opazimo, da velja drugi primer. Iz predavanj in vaj vemo, da je časovna zahtevnost v tem primeru $\Theta(m^{\log_b a} \cdot \log m)$, kar se poračuna v $\Theta(\log m)$. Spomnimo se zveze $M = 2^m$. Izrazimo lahko torej $m = \log M$. Iz tega sledi, da je časovna zahtevnost algoritma vEB-tree-successor enaka $O(\log \log M)$.

Štetje bitov s pomočjo povzetkovne tabele z Lulea algoritmom

Naivni pristop pri šteju bitov s pomočjo povzetkovne tabele z Luela algoritom potrebuje tri pomnilniške reference: eno za branje iz povzetkovne tabele, eno za branje iz bitmap-a in eno za branje imena elementa v vozlišču. Bitmap in povzetkovno tabelo lahko združimo v novo podatkovno strukturo, ki bo informacije o obeh hranila na istem oziroma vsaj na bližnjih mestih. Tako bi lahko z enim klicem dostopali tako do bitmapa, kot tudi do povzetkovne tabele.

Uporabimo lahko na primer seznam terk, katerih prva komponenta hrani kos povzetkovne tabele, druga pa kos bitmap-a.

Lahko bi tudi prepletli povzetkovno tabelo in bitmap tako, da za vsakim kosom bitmapa dodamo vsoto iz povzetkovne tabele.

Definiramo lahko tudi svojo podatkovno strukturo, ki ima dve spremenljivki: kos podatkovne tabele in kos bitmap-a.

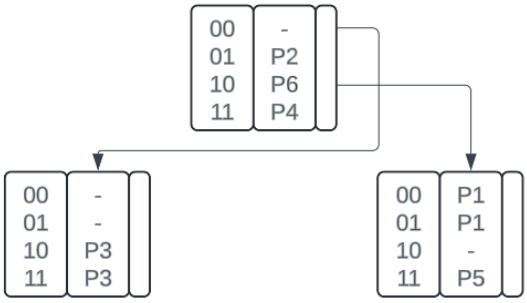
V vseh treh primerih se reducira število klicev iz dveh na enega.

Fixed stride številsko drevo

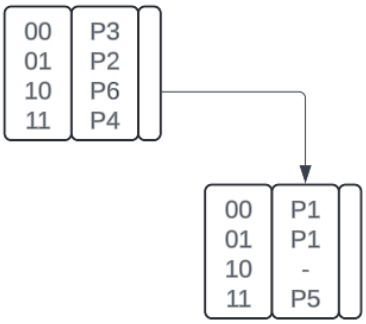
Dana imamo omrežja `100*`, `01*`, `001*`, `11*`, `1011*` in `1*`. Priredimo naslove tako, da so njihove dolžine deljive z 2:

- `P1 = 100*  →  10.00* (P1), 10.01* (P1)`
- `P2 = 01*  →  01* (P2)`
- `P3 = 001*  →  00.10* (P3), 00.11* (P3)`
- `P4 = 11*  →  11* (P4)`
- `P5 = 1011*  →  10.11* (P5)`
- `P6 = 1*  →  10* (P6), 11*`

Prvotno dobimo drevo na sliki 1. Ker je `P3` edini naslov, ki se začne z `00`, ga lahko prestavimo v prvo tabelo. Tako dobimo drevo na sliki 2



Slika 1: Fixed stride številsko drevo s stride velikostjo 2.



Slika 2: Popravljeno fixed stride številsko drevo s stride velikostjo 2.

Naloga 3: Kompaktne podatkovne strukture

BP in LOUDS oblika ordinalnega drevesa

BP oblika: ((()(()))(((()())) oziroma `111011000111010000`, kjer `1` predstavlja oklepaj in `0` predstavlja zaklepaj.
LOUDS oblika: `10110110100101100000` oziroma v tabelarični obliki

	a	b	c	d	e	f	g	h	i
10	110	110	10	0	10	110	0	0	0

BP in LOUDS oblika kardinalnega drevesa

V kardinalnem drevesu je pomembno ali je otrok levi ali desni, zato liste označujemo z (()()), vozlišča, ki imajo le levega otroka z (T()), vozlišča, ki imajo le desnega otroka z (()T), kjer je T BP oblika poddrevesa, vozlišča, ki imajo tako levega kot tudi desnega otroka pa predstavimo z (T_1 T_2), kjer je T_1 levo poddrevo, T_2 pa desno.
BP oblika: ((((())(((())()))(()(((())(()()))))) oziroma
`11110100111010010001101111010011010000`0.
V LOUDS obliki kardinalnega (binarnega) drevesa je vsako vozlišče predstavljeno s tremi biti. Prvi bit je `1`, če je lelo poddrevo neprazno, sicer je `0`, drugi bit je `1`, če je desno poddrevo neprazno, sicer je `0`, tretji bit je vedno `0` in označuje konec vozlišča.
LOUDS oblika: `1011011001000010011000000000`

	a	b	c	d	e	f	g	h	i
10	110	110	010	000	100	110	000	000	000

rmM-drevo

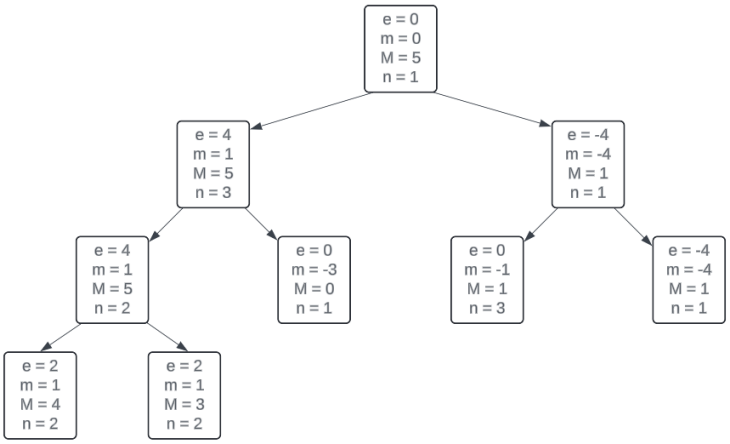
BP obliko kardinalnega drevesa razdelimo na bloke dolžine $b = 8$. Najprej izračunamo razliko viškov na začetku in koncu bloka, minimalni višek, maksimalni višek in število maksimalnih viškov vsakega bloka od s do e po naslednjih formulah:

$$\begin{aligned}
\text{višek (B, i)} &= 2\text{Rank}_1\text{(B, i)} - \text{i} \\
\text{e} &= \text{višek (B, e)} - \text{višek (B, s - 1)} \\
\text{m} &= \min\{\text{višek (B, p)} - \text{višek (B, s - 1)} | s \leq p \leq e\} \\
\text{M} &= \max\{\text{višek (B, p)} - \text{višek (B, s - 1)} | s \leq p \leq e\}
\end{aligned}$$

S tem dobimo liste rmM-drevesa. Vsebino vozlišč, ki niso listi izračunamo po naslednjih formulah:

$$\begin{aligned}
\text{e} &= \text{e}_L + \text{e}_D \\
\text{m} &= \min\{\text{m}_L, \text{e}_L + \text{m}_D\} \\
\text{M} &= \max\{\text{M}_L, \text{e}_L + \text{M}_D\} \\
\text{n} &= \begin{cases} \text{n}_L & | \quad \text{m}_L < \text{e}_L + \text{m}_D \\ \text{n}_D & | \quad \text{m}_L > \text{e}_L + \text{m}_D \\ \text{n}_L + \text{n}_D & | \quad \text{m}_L = \text{e}_L + \text{m}_D \end{cases},
\end{aligned}$$

kjer je e_L višek levega poddrevesa in e_D višek desnega. Ostale oznake so analogne. Tako dobimo naslednje rmM-drevo.



Slika 3: rmM-drevo z velikostjo blokov $b = 8$.

Funkcija `Izberi (T, i)`

V tej nalogi zapišemo kodo za iskanje i-tega elementa v razširjenem drevesu T. Če poznamo velikost levega poddrevesa vsakega vozlišča, je koda naslednja:

definition Izberi (T, i)
  if i = | L | + 1 **then**
    return root(T)
  if i < | L |+ 1 **then**
    return Izberi (L, i)
  if i > | L |+ 1 **then**
    return Izberi (R, i −| L |− 1)

Velikost levega poddrevesa primerjamo z danim i. V kolikor je $i = |L| + 1$ je i-to vozlišče kar koren drevesa T. Če je $i < |L| + 1$ iščemo i-to vozlišče levega poddrevesa, sicer pa $i - |L| - 1$-vo vozlišče desnega poddrevesa.