UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Vizualizácia algoritmov pre Wheelerove grafy

Bakalárska práca

2022

Adam Struharňanský

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Vizualizácia algoritmov pre Wheelerove grafy

Bakalárska práca

Študijný program: Informatika
Študijný odbor: Informatika
Školiace pracovisko: Katedra informatiky
Vedúci práce: Mgr. Adrián Goga

Bratislava, 2022

Adam Struharňanský

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Adam Struharňanský
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Vizualizácia algoritmov pre Wheelerove grafy
Visualization of algorithms for Wheeler graphs

Anotácia: Wheelerove grafy sú zovšeobecnením Burrows-Wheelerovej transformácie, ktorá sa ako súčasť úsporných dátových štruktúr používa predovšetkým na vyhľadávanie v texte, zarovnávanie sekvencií v bioinformatike a podobne. Doposiaľ bolo v oblasti Wheelerových grafov prebádaných niekoľko aspektov umožňujúcich efektívne vyhľadávanie, úspornú reprezentáciu atď. Podľa dostupných informácií však neexistuje softvér, ktorý by tieto metódy vizualizoval na úrovni použiteľnej pre pedagogické alebo výskumné účely. Tvorba takého softvéru je predmetom tejto práce.

Cieľ: Cieľom práce je návrh a implementácia vizualizácie niekoľkých vybraných algoritmov na Wheelerových grafoch.

Kľúčové slová: Wheelerove grafy, vizualizácia, dátové štruktúry

Vedúci: Mgr. Adrián Goga
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 26.10.2021

Dátum schválenia: 04.11.2021 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestné vyhlásenie

Čestne vyhlasujem, že som bakalársku prácu vypracoval samostatne za použitia uvedenej literatúry.

Poďakovanie

Chcel by som sa poďakovať svojmu školiteľovi za to, že mi ukázal nový svet reťazcov a za pomoc pri jeho objavovaní.

Tiež by som sa chcel poďakovať svojej rodine za to, že strpeli moju nervozitu, a tiež za ich oporu počas náročného obdobia opravovania chýb v kóde.

A nakoniec by som sa chcel poďakovať Martinovi Bednárovi za jeho pomoc pri finalizovaní programu.

ABSTRAKT

V práci sa zaoberáme dátovou štruktúrou sufixové pole, Burrows-Wheelerovou transformáciou a tiež Wheelerovými grafmi a aj niektorými ich aplikáciami. V ďalších častiach práce sa zameriame na špecifikáciu a implementáciu programu, určeného na pedagogické a vedecké účely, ktorý bude dané štruktúry vizualizovať. Ako prílohu prikladáme CD so zdrojovými súbormi a spustiteľnou aplikáciou.

KĽÚČOVÉ SLOVÁ
Wheelerove grafy, vizualizácia, dátové štruktúry

ABSTRACT

In this work we deal with the suffix array data structure, the Burrows-Wheeler transformation and also the Wheeler graphs and some of their applications. In the following sections of the work, we will focus on the specification and implementation of a program designed for pedagogical and scientific purposes that will visualize the structures. In an appendix we provide a CD with source codes and an executable application.

KEYWORDS
Wheeler graphs, visualization, data structures

OBSAH

ÚVOD

Dôvodom, pre ktorý sme sa rozhodli venovať v tejto práci Wheelerovým grafom, Burrows-Wheelerovej transformácií a sufixovým poliam je to, že táto téma je veľmi zaujímavá, a tiež aj kvôli tomu, že doteraz podľa našich zistení neexistuje softvér, ktorý by tieto štruktúry vizualizoval.

V prvej kapitole tejto práce sa budeme zaoberať Burrows-Wheelerovu transformáciou, ktorá bola pôvodne vymyslená pre použitie v kompresii, ale neskôr boli ukázané aj jej ďalšie nemenej zaujímavé možnosti využitia. Spomenieme tiež, že Burrows-Wheelerova transformácia sa dá vytvoriť v lineárnom čase s použitím sufixových polí. A nakoniec sa budeme venovať Wheelerovým grafom, ktoré sú zovšeobecnením Burrows-Wheelerovej transformácie, a ich aplikáciám.

V druhej kapitole uvedieme princípy vizualizácie, podľa ktorých sme naprogramovali program, ktorý vizualizuje dátové štruktúry ukázané v prvej časti a tiež niektoré ich aplikácie.

V ďalšej kapitole opíšeme implementáciu vytvoreného programu, predstavíme jeho hlavné časti a funkcie.

Nakoniec prezentujeme popis daného softvéru, jeho vzhľad a ovládacie prvky pre užívateľa.

1. TEÓRIA

V tejto kapitole zadefinujeme teoretické pojmy a objekty, ktorých algoritmy neskôr budeme vizualizovať. Okrem ich definícií sú v tejto kapitole uvedené aj niektoré ich vlastnosti a prepojenia medzi sebou aj medzi inými známymi objektmi, ktoré nebudeme definovať. Tiež zahrnieme do tejto časti aj aplikácie, na ktoré sa dané objekty zvyčajne používajú.

1.1. POJMY A DEFINÍCIE

V tejto sekcii sú vymedzené základné terminologické označenia a definície.

- Reťazec S je konečná postupnosť znakov z usporiadanej abecedy Σ. Číslom n budeme označovať dĺžku reťazca S, n := |S|. V tomto texte budeme indexovať reťazce od 0 (a teda posledná pozícia v reťazci je n – 1)
- Znakom $ budeme označovať znak, ktorý je menší, ako hociktorý znak v abecede Σ
- 0-ukončený reťazec – je taký reťazec, v ktorom sa znak $ vyskytuje práve raz, a to na konci daného reťazca
- S[i] – i-ty znak v reťazci S (budeme používať rovnaké označenie aj pre i-ty objekt v poli)
- S[i, j] – podreťazec reťazca S, so začiatkom na i-tej, a koncom na j-tej pozícií.
- $S <_{lex} T$ – znamená, že S je lexikograficky menšie ako T, teda buď je S kratšie ako T a zároveň je S prefixom T, alebo existuje nejaké prirodzené číslo i, ktoré je menšie ako dĺžka kratšieho z reťazcov S a T, pre ktoré platí S[0, i – 1] = T[0, i – 1], a zároveň S[i] < T[i]
- S^R označuje reverz reťazca, teda S^R := S[n – 1] S[n – 2] … S[1] S[0]

Definícia (Úsporná dátová štruktúra) Nech b je teoreticky optimálny (minimálny) počet bitov, potrebný na reprezentáciu štruktúry H. Hovoríme, že dátová štruktúra je úsporná, ak uloženie štruktúry H zaberie b + o(b) bitov priestoru. Takúto dátovú štruktúru budeme tiež nazývať aj ako úsporná reprezentácia H.

Definícia (C_S) Operácia $C_S(c)$ vráti počet výskytov znakov v reťazci S, ktoré sú lexikograficky menšie ako c, teda:

$$C_S(c) := |\{ k \in \{0, \ldots, n - 1\} \mid S[k] < c\}|$$

Definícia ($rank_S$) Operácia $rank_S(c, i)$ pre daný reťazec S, znak c, a pozíciu i v reťazci vráti počet výskytov znaku c, v podreťazci S[0, i - 1]:

$$rank_S(c, i) := |\{ k \in \{0, \ldots, i - 1\} \mid S[k] = c\}|$$

Definícia ($select_S$) Operácia $select_S(c, i)$ pre daný reťazec S, znak c, a prirodzené číslo i, ktoré je menšie alebo rovné ako $rank_S(c, n - 1)$, vráti pozíciu i-teho výskytu znaku c, v reťazci S:

$$select_S(c, i) := \max \{ k \in \{0, \ldots, i\} \mid rank_S(c, k) = i\}$$

Operáciu C budeme tiež používať ako pole, ktoré je indexované znakmi príslušnej abecedy, teda $C_S(c) = C_S[c]$. Toto pole si vieme vypočítať dopredu, kde najskôr vypočítame výskyt každého zo znakov abecedy, a potom pre každý znak uložíme do poľa súčet hodnôt výskytov lexikograficky menších znakov (riadky 1 až 8 v Algoritme 1). Dotaz operácie $C_S(c)$ bude teda zhodný s vrátením hodnoty $C_S[c]$.

Jacobson ukázal [Jac89], že pre reťazec S, ktorého abeceda má len 2 znaky, je možné odpovedať na $rank_S$ dotazy v konštantnom čase, za použitia úspornej dátovej štruktúry. Analogický výsledok pre $select_S$ dotazy, teda pre reťazec S nad dvojprvkovou abecedou, použitím úspornej dátovej štruktúry a v konštantnom čase ukázal Clark [Cla96].

Odpovedanie na dotazy $rank_S$ a $select_S$ pre reťazec S nad ľubovoľnou abecedou, je možné pridaním pomocnej dátovej štruktúry – Wavelet stromu [Nav14]. V tejto dátovej štruktúre sú rekurzívne volané $rank_S$, alebo $select_S$ dotazy pre reťazce nad dvojprvkovými abecedami, ktoré sú v čase O(1). Hĺbka tejto rekurzie rastie logaritmicky vzhľadom na veľkosť abecedy, a teda zodpovedanie $rank_S$ alebo $select_S$ dotazu pre reťazec S nad všeobecnou abecedou Σ je v čase $O(\log_2|\Sigma|)$. Tiež je možné použitím Wavelet stromu pre reťazec S získať hodnotu znaku na pozícií i, teda S[i] (tzv. $access_S$ dotaz), v čase $O(\log_2|\Sigma|)$, bez samotného uloženia reťazca.

Vyššie uvedené funkcie C_S, $rank_S$, a $select_S$, budeme v pseudokódoch označovať ako funkcie volané na reťazcoch, teda postupne prepísané ich budeme označovať S.C(c), S.rank(c, i) a S.select(c, i).

1.2.SUFIXOVÉ POLE

Dátovú štruktúru sufixové pole zadefinovali Manber a Myers [MM93], ktorí avizovali ich hlavnú výhodu oproti sufixovým stromom, priestorovú úspornosť. Zároveň však sufixové polia umožňujú riešiť úlohu nájdenia všetkých výskytov reťazca R v reťazci S v konkurencieschopnom čase so sufixovými stromami. Neskôr bolo dokonca ukázané, že každý algoritmus, ktorý používa sufixový strom ako dátovú štruktúru, môže byť nahradený algoritmom, ktorý ako dátovú štruktúru používa sufixové pole s pomocnými štruktúrami, pričom tento algoritmus bude riešiť rovnaký problém v asymptoticky rovnakej časovej zložitosti [AKO04]. Zároveň sufixové pole s danými pomocnými štruktúrami bude zaberať menej priestoru ako sufixový strom.

Sufixové pole reťazca S je možné zostrojiť v lineárnom čase od dĺžky reťazca S [KS03]. Základom sufixového poľa sú lexikograficky zoradené sufixy reťazca S. Keďže každý sufix v reťazci S je jednoznačne určený jeho začiatočnou pozíciou v reťazci S, tak je postačujúce uložiť v sufixovom poli len tieto pozície, nie celé sufixy. Sufixové pole pre reťazec S budeme označovať SA. Na obrázku 1.1. je príklad sufixového poľa SA, pre reťazec S = abrakadabra$.

Definícia (Sufixové pole) Nech S je reťazec dĺžky n. Potom sufixové pole SA reťazca S je permutácia prirodzených čísel z množiny {0, … , n-1}, tak, že platí:

$$\forall\, i, j \in \{0, \ldots, n-1\};\ i < j \Rightarrow S[SA[i],\ n-1] <_{lex} S[SA[j], n-1]$$

SA[i]	S[SA[i], n-1]
11	$
10	a$
7	abra$
0	abrakadabra$
5	adabra$
3	akadabra$
8	bra$
1	brakadabra$
6	dabra$
4	kadabra$
9	ra$
2	rakadabra$

Obrázok 1.1: Sufixové pole SA pre reťazec S = abrakadabra$, spolu so sufixami, prislúchajúcimi k jednotlivým hodnotám.

1.3. BURROWS-WHEELEROVA TRANSFORMÁCIA

Burrows-Wheelerova transformácia (BWT) je vratná transformácia textu, ktorá bola pôvodne vytvorená pre kompresiu dát [BW94]. Táto transformácia samotná nekomprimuje vstupný reťazec, jej výsledkom je reťazec, ktorý je iba permutáciou znakov pôvodného reťazca.

Základom BWT daného reťazca S je vytvoriť všetky rotácie (cyklické posuny) reťazca S, následne ich lexikograficky usporiadať, a nakoniec postupne, od lexikograficky najmenšej po lexikograficky najväčšiu, vybrať z každej rotácie posledný znak. V pôvodnej definícií bolo výstupom transformácie okrem permutácie znakov pôvodného reťazca aj číslo I, nazývané tiež BWT index, označujúce pozíciu prvej takej rotácie v poradí usporiadaných rotácií, ktorá je zhodná s reťazcom S. Ako je ukázané nižšie, BWT index je potrebný pri inverzii BWT, kde pri nie 0-ukončených reťazcoch je možné iba z reťazca L získať len niektorú rotáciu reťazca S, bez možnosti zistiť pôvodný reťazec S. Pri 0-ukončených reťazcoch tento problém nevzniká. Je jednoduché zmeniť reťazec na 0-ukončený (pridaním znaku na koniec, ktorý je lexikograficky menší ako všetky predchádzajúce), a preto sme sa rozhodli použiť definíciu iba pre 0-ukončené reťazce, ktorej výstupom je iba permutácia znakov, bez BWT indexu.

Definícia (Burrows-Wheelerova transformácia) Nech S je 0-ukončený reťazec dĺžky n. Nech M je matica veľkosti n × n, ktorej prvky sú znaky, a ktorej riadky sú lexikograficky usporiadané rotácie reťazca S. Posledný stĺpec matice M nazývame Burrows-Wheelerova transformácia. Posledný stĺpec matice M budeme tiež označovať L, prvý stĺpec F.

Maticu M v definícií vyššie budeme taktiež označovať ako Burrows-Wheelerova matica (BWT matica). Na obrázku 1.2 je BWT matica pre reťazec abrakadabra$.

Na BWT je založený napr. kompresný algoritmus bzip [Sew96]. Jeho hlavné časti sú (poradie ako pri komprimovaní) BWT, move-to-front transformácia (MTFT), run-length encoding, a Huffmanov kód.

Najdôležitejšou vlastnosťou BWT pre komprimačné algoritmy je zhlukovanie rovnakých znakov. Je to spôsobené tým, že v obyčajnom texte, podobným kontextom zvykne

F											L
$	a	b	r	a	k	a	d	a	b	r	a
a	$	a	b	r	a	k	a	d	a	b	r
a	b	r	a	$	a	b	r	a	k	a	d
a	b	r	a	k	a	d	a	b	r	a	$
a	d	a	b	r	a	$	a	b	r	a	k
a	k	a	d	a	b	r	a	$	a	b	r
b	r	a	$	a	b	r	a	k	a	d	a
b	r	a	k	a	d	a	b	r	a	$	a
d	a	b	r	a	$	a	b	r	a	k	a
k	a	d	a	b	r	a	$	a	b	r	a
r	a	$	a	b	r	a	k	a	d	a	b
r	a	k	a	d	a	b	r	a	$	a	b

Obrázok 1.2: Burrows-Wheelerova matica pre reťazec = abrakadabra$. Reťazec L = ard$kraaaabb je zároveň BWT. Je možné si v ňom všimnúť zhlukovanie niektorých písmen.

predchádzať malý počet rozličných znakov. Napríklad v slovenskom texte, kontextu esto s veľkou pravdepodobnosťou prechádzajú znaky c, g, m a v (**c**esto, **g**esto, **m**esto, d**v**esto). Ak by sme mali text s viacerými výskytmi týchto štyroch slov, tak po lexikografickom zoradení všetkých rotácií, budú tie rotácie, ktoré začínajú kontextom esto vedľa seba, a teda v BWT vznikne úsek zložený len z písmen c, g, m a v. Ak nejakému kontextu predchádza iba jeden znak, napríklad kontextu eťazec predchádza znak r, tak v prislúchajúcom úseku v BWT bude úsek zložený len zo znakov r. Čím je vstupný reťazec viac repetitívny, tým viac rovnakých kontextov má, a tým sú väčšie zhluky písmen v BWT.

Algoritmus bzip po použití BWT vykoná MTFT. Pri vykonávaní tejto transformácie si algoritmus udržuje zoznam zložený zo znakov abecedy Z. Pri spracovaní i-teho písmena v reťazci T, zistí aký je jeho index v zozname Z, a tento index zapíše do výstupu. Potom presunie daný znak v zozname Z na začiatok, takže bude mať index 0. Táto transformácia zmení úsek zložený z rovnakých znakov na úsek samých núl okrem prvého prvku úseku, a úsek s malým počtom znakov na úsek s malými číslami. Pri spracovaní BWT teda zmení lokálne zhluky rovnakých písmen na reťazec zložený z intervalov, ktoré obsahujú malé čísla, okrem začiatkov daných intervalov.

Za týmto úkonom algoritmus bzip použije run-length encoding, v ktorom sa zakóduje každý úsek zložený z rovnakých znakov c, dĺžky n, na dvojicu (n, c).
V ďalšom kroku sa použije Huffmanov kód, ktorý kóduje jednotlivé symboly, na základe ich výskytu – vo všeobecnosti, čím väčší výskyt, tým kratší kód znaku.
Aj keď má bzip pomerne malý kompresný pomer, tak z toho, ako je popísané nižšie, je nutné pri inverzii BWT použiť pseudonáhodné skoky v pamäti, pričom architektúra moderných pamätí, ktoré používajú vyrovnávacie pamäte, nie je na to vhodná, z čoho vyplývajú dlhé prístupové časy, a teda trochu pomalá dekompresia.

1.3.1. VÝPOČET BURROWS-WHEELEROVEJ TRANSFORMÁCIE V LINEÁRNOM ČASE

Pri výpočte BWT nie je nutné vytvoriť celú BWT maticu, kde by prípadný algoritmus používajúci tento spôsob na výpočet mal minimálne kvadratickú časovú aj priestorovú zložitosť. Jedno lineárne riešenie predstavené už autormi BWT je na základe podobnosti medzi sufixovými poľami a BWT – sufixové polia lexikograficky zoraďujú sufixy, a BWT lexikograficky zoraďuje rotácie. Pre 0-ukončený reťazec S sú tieto usporiadania zhodné, i-ty reťazec v sufixovom poli (S[SA[i], n – 1]) je neprázdnym prefixom i-teho riadku BWT matice. Na obrázku 1.3 je ukážka takéhoto usporiadania pre reťazec S = abrakadabra$. Toto usporiadanie je rovnaké preto, že pozícia reťazca v BWT matici nezávisí od znakov, ktoré nasledujú po znaku $. Je z nej zjavné, že F[i] = S[SA[i]]. V riadkoch BWT matice sú rotácie, a teda hodnoty L[i] a F[i] sú susedné znaky v reťazci S, alebo je to posledný a prvý znak reťazca S. Potom platí nasledujúca rovnosť:

$$\forall i \in \{0, \dots, n-1\};\ L[i] = \begin{cases} S[SA[i]-1] & \text{,ak } SA[i] \neq 0 \\ \$ & \text{,inak} \end{cases}$$

	SA	
$ a b r a k a d a b r a	11	$
a $ a b r a k a d a b r	10	a $
a b r a $ a b r a k a d	7	a b r a $
a b r a k a d a b r a $	0	a b r a k a d a b r a $
a d a b r a $ a b r a k	5	a d a b r a $
a k a d a b r a $ a b r	3	a k a d a b r a $
b r a $ a b r a k a d a	8	b r a $
b r a k a d a b r a $ a	1	b r a k a d a b r a $
d a b r a $ a b r a k a	6	d a b r a $
k a d a b r a $ a b r a	4	k a d a b r a $
r a $ a b r a k a d a b	9	r a $
r a k a d a b r a $ a b	2	r a k a d a b r a $

Obrázok 1.3: Vzťah medzi BWT a sufixovými poľom pre reťazec S = abrakadabra$.

Lineárny algoritmus pre získanie BWT teda najskôr vytvorí sufixové pole, a potom v cykle pomocou danej rovnosti zistí pre každé i hodnotu L[i].

1.3.2. INVERZNÁ BURROWS-WHEELEROVA TRANSFORMÁCIA

Môžeme si všimnúť, že oba reťazce F a L, tak ako sú definované v BWT, sú permutácie znakov vstupného reťazca S. Tiež sme v predchádzajúcej časti spomenuli, že pre každé i, je L[i] znak nachádzajúci sa v reťazci S hneď pred znakom F[i] (prípadne je to posledný a prvý znak). Ak vieme, že znak L[x] reprezentuje tú istú pozíciu v reťazci S ako znak F[y], potom vieme, že v reťazci S sú hneď za sebou znaky L[x], L[y]. Ak by sme vedeli nájsť pre každý znak L[x] číslo y také, že F[y] reprezentujúci tú istú pozíciu v reťazci S, potom by sme reťazec S vedeli zrekonštruovať. Dané pole hodnôt, ktoré toto spĺňa, nazývame LF-mapovanie. Budeme ho označovať LF. Na obrázku 1.4 je zobrazené LF-mapovanie, spoločne s reťazcami L a F pre reťazec S = abrakadabra$.

i	LF[i]	F[i]	L[i]	S[i]
0	1	$0	a0	a
1	10	a0	r0	b
2	8	a1	d0	r
3	0	a2	$0	a
4	9	a3	k0	k
5	11	a4	r1	a
6	2	b0	a1	d
7	3	b1	a2	a
8	4	d0	a3	b
9	5	k0	a4	r
10	6	r0	b0	a
11	7	r1	b1	$

Obrázok 1.4: LF-mapovanie, spoločne s reťazcami F a L, pre reťazec S = abrakadabra$. Znaky v reťazcoch F a L majú indexy rovné ich doterajšiemu výskytu v danom reťazci. Môžeme si všimnúť, LF-mapovanie zobrazuje to, že i-ty výskyt znaku v F a i-ty výskyt znaku v L zodpovedajú tomu istému znaku. Reťazec F je vykreslený sivou farbou, pretože v algoritme nie je uložený v pamäti.

Definícia (LF-mapovanie) Nech S je reťazec dĺžky n, a nech L je posledným stĺpcom BWT matice pre reťazec S. LF-mapovanie je permutácia čísel {0, …, n – 1}, taká, že pre ňu platí: $LF[i] = C_L[L[i]] + rank_L(L[i], i)$.

```
1 C = [σ]
2 for i := 0 to n - 1 do
3    C[L[i]] := C[L[i]] + 1
4 sum := 0
5 for i := 0 to sigma - 1 do
6    tmp := C[i]
7    C[i] := sum
8    sum := sum + tmp
9 LF = [n]
10 for i := 0 to n - 1 do
11    LF[i] := C[L[i]]
12    C[L[i]] := C[L[i]] + 1
13 S = [n]
14 S[n - 1] := $
15 j  := 0
16 for i := n - 2 to 0 do
17    S[i] := L[j]
18    j := LF[j]
```

Algoritmus 1: Výpočet inverznej BWT. Riadky 1 až 8 počítajú pole C, riadky 9 až 12 počítajú LF-mapovanie a riadky 13 až 18 počítajú inverznú BWT pomocou LF-mapovania. S menšími zmenami prevzatý z [Bai20].

Rovnosť v definícií LF-mapovania platí kvôli tomu, že i-ty výskyt znaku c v F a i-ty výskyt znaku c v L odpovedajú tomu istému znaku, na tej istej pozícií v S. To platí kvôli tomu, že riadky, ktoré začínajú znakom c, sú zoradené iba podľa pravých kontextov prvých znakov (riadkov bez prvého znaku). Podobne, riadky, ktoré končia znakom c, sú zoradené podľa ľavých kontextov posledného znaku (riadkov bez posledného znaku). Ľavé a pravé kontexty pre daný znak c sú tie isté množiny, keďže riadky BWT matice sú rotácie. Obe tieto množiny sú lexikograficky usporiadané, keďže podľa nich sú relatívne usporiadané príslušné riadky, a teda dané množiny majú aj rovnaké poradie. A teda i-ty výskyt znaku c v reťazci L odpovedá i-temu lexikograficky najmenšiemu kontextu, ktorý odpovedá i-temu výskytu znaku c, v reťazci F.

Algoritmus 1, ktorý sme použili na vizualizáciu inverznej BWT, najskôr vypočíta hodnoty LF[i] pre každé i, nastaví posledný znak reťazca S na $, a potom v cykle postupne zapĺňa od konca reťazec S, pričom nasledujúcu hodnotu vie s pomocou LF-mapovania, a skutočnosti, že F[i] a L[i] znaky sa nachádzajú v rotácií reťazca S hneď vedľa seba. V skutočnosti nie je nutné mať uložené LF-mapovanie, je možné priamo počítať hodnoty pre daný prvok tak, ako je zadané v definícií. Ak by nebol reťazec 0-ukončený, potom by reťazec S nekončil znakom $, a algoritmus by nevedel zistiť, ktorým znakom S končí, a vrátil by iba niektorú rotáciu reťazca S. V tomto prípade je nutné

použiť BWT index I, ktorý hovorí, že na pozícií L[I] sa nachádza posledný znak reťazca S.

1.3.3. BWT PRE VYHĽADÁVANIE VÝSKYTOV VZORU V REŤAZCI

FM index, definovaný Ferraginim a Manzinim [FM00], je založený na BWT, a obsahuje ďalšie pomocné štruktúry, s ktorými dokáže odpovedať na dotazy typu nájsť výskyty vzoru V v reťazci S, v čase $O(m \log_2\Sigma + occ)$, kde occ je počet výskytov vzoru V v reťazci. Odpovedanie na daný dotaz využíva to, že riadky BWT matice sú lexikograficky zoradené. Z toho vyplýva, že riadky BWT matice, pre ktoré je reťazec R prefixom, tvoria súvislý interval. Ak vieme, že práve pre riadky BWT matice na intervale $[t_0, b_0]$ je reťazec R prefixom, tak pomocou toho, že riadky BWT matice sú rotácie, vieme zistiť, či sa reťazec cR, pre ľubovoľné c, nachádza v pôvodnom reťazci S – stačí zistiť, či sa nachádza znak c v podreťazci $L[t_0, b_0]$. Toto vieme zistiť dvoma $rank_L$ dotazmi, ktorých rozdiel nám tiež povie, koľko sa v reťazci S nachádza výskytov vzoru cR. Zároveň nám tieto $rank_L$ dotazy povedia, koľký výskyt v reťazci L má prvý znak c z daného intervalu, a ktorý výskyt v reťazci L má posledný znak. Z tohto vieme pomocou princípov LF-mapovania vytvoriť nový interval $[t_1, b_1]$, pre ktorý platí, že práve riadky BWT matice na ňom majú prefix cR. Týmto spôsobom vieme predĺžiť hľadaný vzor. Na obrázku 1.5 je zobrazený príklad pre BWT maticu reťazca S = abrakadabra$, pri hľadaní vzoru V = abra, s vyznačenými sufixami vzoru danej BWT matici.

Keďže sa daný vzor hľadá odzadu, nazýva sa tento proces spätné vyhľadávanie, a jeho jeden krok je *spätný krok*. Na začiatku sa nastaví interval tak, aby zaberal všetky riadky BWT matice, keďže prázdny sufix ľubovoľného vzoru sa nachádza všade.

Ak sme už našli konečný interval $[t_m, b_m]$, pre ktorý platí, že práve pre riadky BWT matice na ňom majú hľadaný vzor, je potrebné toto pretransformovať na skutočné miesta v reťazci S, na ktorom sa daný vzor nachádza. Pre 0-ukončené reťazce je toto uložené v sufixovom poli SA pre reťazec S, presnejšie, i-ty riadok BWT matice začína v reťazci S na pozícií SA[i].

Keďže sa problém vyhľadávania vzoru v reťazci dá riešiť s použitím iba sufixového poľa (spolu s inými pomocnými štruktúrami), v čase závislom iba od dĺžky vzoru, a jedným z cieľom FM indexu bolo znížiť pamäťové nároky, tak v pôvodnej definícií nebolo uložené celé sufixové pole. Namiesto toho sa uložila iba jeho podmnožina, a to hodnoty, ktoré boli deliteľné číslom p. Potom sa buď našla hodnota v sufixovom poli, alebo sa

	F											L
0	$0	a	b	r	a	k	a	d	a	b	r	a0
1	a0	$	a	b	r	a	k	a	d	a	b	r0
2	a1	b	r	a	$	a	b	r	a	k	a	d0
3	a2	b	r	a	k	a	d	a	b	r	a	$0
4	a3	d	a	b	r	a	$	a	b	r	a	k0
5	a4	k	a	d	a	b	r	a	$	a	b	r1
6	b0	r	a	$	a	b	r	a	k	a	d	a1
7	b1	r	a	k	a	d	a	b	r	a	$	a2
8	d0	a	b	r	a	$	a	b	r	a	k	a3
9	k0	a	d	a	b	r	a	$	a	b	r	a4
10	r0	a	$	a	b	r	a	k	a	d	a	b0
11	r1	a	k	a	d	a	b	r	a	$	a	b1

Obrázok 1.5: BWT matica pre reťazec S = abrakadabra$, v strede výpočtu hľadania vzoru V = abra.. Ku znakom v F a L sú pridané indexy, určujúce ich doterajší výskyt v reťazci. Vidíme sufixy dĺžky 3 vzoru V, ako sú zvýraznené na 6-tom a 7-mom riadku matice. V ďalšom kroku budú kontrolované (a neskôr pridané) znaky a_1 a a_2, ktoré sú na L[6] a L[7].

budeme pomocou LF-mapovania presúvať po riadkoch BWT matice, až kým nenájdeme hodnotu v sufixovom poli. Výsledné miesto v reťazci je potom súčet tejto hodnoty v sufixovom poli a počet použití LF-mapovania. Okrem takto upraveného sufixového poľa, obsahoval pôvodný FM index ďalšiu pomocnú štruktúru, ktorá umožňovala efektívny dotaz rank_L, ktorú sme nevizualizovali.

V našej implementácií, Algoritmus 2, sme použili polootvorený interval a ako vstup sme mali aj celé sufixové pole, a je teda skrátený o hľadanie výsledných pozícií pomocou LF-mapovania.

```
1 top := 0
2 bottom := n
3 for i := m - 1 to 0 do
4    c := P[i]
5    top := L.C(c) + L.rank(c, top)
6    bottom := L.C(c) + L.rank(c, bottom)
7    if top >= bottom
8       return
9 return [SA[top], SA[bottom-1]]
```

Algoritmus 2: Hľadanie všetkých výskytov vzoru v reťazci. Premenné top a bottom ohraničujú polootvorený interval, v ktorom sa nachádza doteraz nájdený sufix vzoru. Ak je hodnota premennej top väčšia rovná ako hodnota premennej bottom, potom je daný reťazec prázdny a môžeme sa vrátiť, keďže neexistuje žiaden výskyt.

1.4. WHEELEROVE GRAFY

Po zavedení BWT ako kompresného nástroja, bolo vytvorených viacero variantov BWT a tiež štruktúr a techník, ktoré používali BWT, alebo s ňou boli nejak prepojené. Tieto štruktúry zahrňovali stále zložitejšie objekty, od stromov, cez grafy, až po zarovnania. Gagie a kol. [GMS17], sa na základe ich spoločných vlastností, pokúsili pre tieto štruktúry vytvoriť zjednocujúci rámec a navrhli koncept Wheelerových grafov. Aj keď, ako ukázali, sa ním nedá reprezentovať každá známa štruktúra súvisiaca s BWT, dajú sa ním reprezentovať napríklad prefixové stromy[1], de Bruijnove grafy, Wavelet stromy, alebo samotné BWT.

V tejto práci nebudeme používať pôvodné definície, ale budeme používať trochu modifikované definície Baierom [Bai20]. Pri každej napíšeme, v čom sa líšia od pôvodnej, a prečo bola daná modifikácia vykonaná.

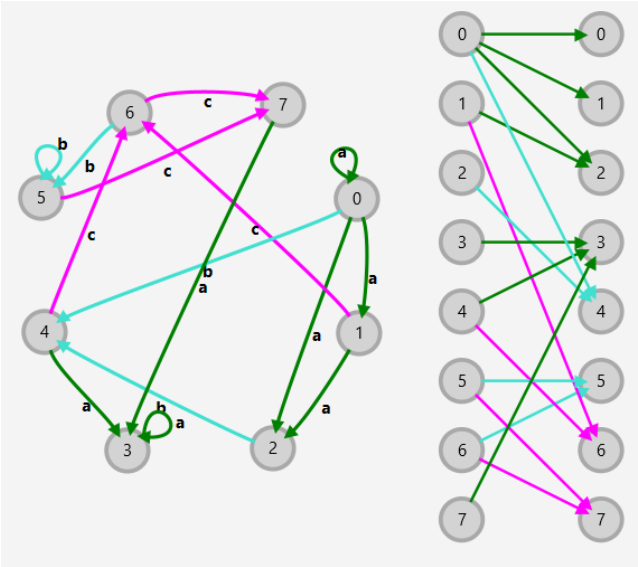
Definícia 1.4.1. (Wheelerov graf) (podľa [Bai20]) Nech G je orientovaný multigraf, ktorého každá vrchol má aspoň jednu vstupnú, a jednu výstupnú hranu. Nech $\lambda: E \rightarrow \Sigma$ je funkcia, ktorá označí každú hranu znakom z abecedy Σ. G je Wheelerov graf práve vtedy, keď existuje usporiadanie vrcholov také, že pre každú dvojicu hrán $e = (u, v)$ a $e' = (u', v')$ platia nasledujúce podmienky monotónnosti:

(i) $\lambda(e) < \lambda(e') \Longrightarrow v \leq v'$

(ii) $\lambda(e) = \lambda(e') \wedge u < u' \Longrightarrow v \leq v'$

[1] Tiež známe ako dátová štruktúra trie

V tejto definícií boli vykonané tri zmeny oproti pôvodnej. Je použitý multigraf, namiesto grafu, čoho užitočnosť je ukázaná neskôr v podsekcii o tunelovaní. Pre každý vrchol bola pridaná požiadavka mať aspoň jednu vstupnú a výstupnú hranu, okrem iného kvôli charakterizácii LF-mapovania, ako jedného celistvého cyklu v BWT, z čoho vyplýva jednoduchá reprezentácia BWT pomocou Wheelerových grafov. A nakoniec bola uvoľnená podmienka (i), kde je povolená aj rovnosť v a v′. To znamená, že vrchol môže mať vstupné hrany s rozdielnymi označeniami.



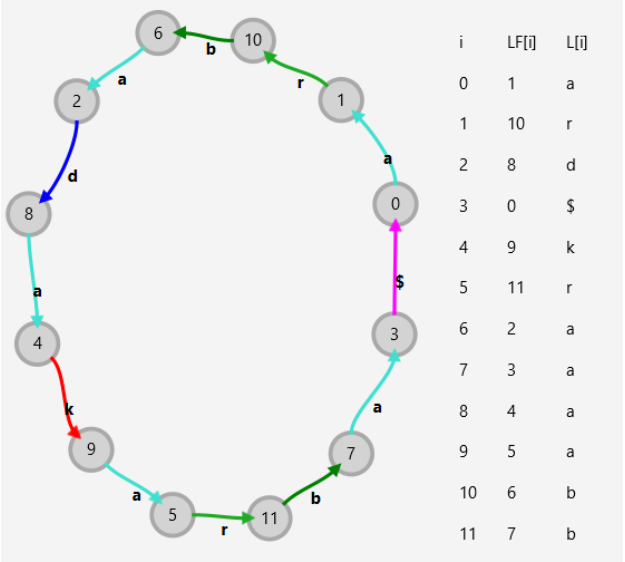
Obrázok 1.6: Wheelerov graf, podobný grafu na obrázku v [GMS17] – oproti tomu sú tu pridané hrany tak, aby mal každý vrchol aspoň jednu vstupnú, aj výstupnú hranu. Vľavo je zobrazený pomocou uzlového zobrazenia, vpravo pomocou metódy zobrazenia v [Lan21]

Príklad Wheelerovho grafu je na obrázku 1.6. Na danom obrázku vpravo sme použili metódu zobrazenia grafu z edukačného videa o Wheelerových grafoch [Lan21]. V tejto metóde má každý vrchol dve repliky, z jednej hrany iba vystupujú (vľavo), do druhej iba vstupujú (vpravo). Dané repliky sa zoradia do dvoch radov, podľa usporiadania vrcholov. Pri tejto metóde zobrazenia, sa dajú jednoducho zobraziť podmienky monotónnosti. Podmienka (i) je zobrazená tak, že v rade zložených iba zo vstupných replík, sú najskôr vrcholy so vstupnými hranami s najmenším znakom, potom s druhým najmenším znakom, atď. V pôvodnej definícií boli tieto intervaly vrcholov disjunktné, v tejto definícií môžu mať krajné vrcholy daných intervalov vstupné hrany označené viacerými znakmi. Podmienka (ii) je zobrazená v tom, že žiadne hrany, ktoré majú rovnaké označenie, sa nepretínajú.

1.4.1. WHEELEROVE GRAFY AKO REPREZENTÁCIA BURROWS-WHEELEROVEJ TRANSFORMÁCIE

Ako bolo spomenuté vyššie, Wheelerove grafy môžu reprezentovať BWT. Na reprezentovanie BWT L, s dĺžkou n a LF-mapovaním LF, vytvoríme n vrcholov, ktoré očíslujeme od 0 do n – 1. Potom pridáme každému vrcholu s označením i výstupnú hranu s označením L[i], ktorá je zároveň vstupnou hranou pre vrchol s označením LF[i]. Potom je graf s danými vrcholmi a hranami Wheelerov graf [Bai20]. Na obrázku 1.7 je zobrazený Wheelerov graf pre BWT L = ard$kraaaabb (pôvodný reťazec S = abrakadabra$).

Získanie pôvodného reťazca je možné pochôdzkou v grafe, so začiatkom v hrane s označením $, keďže prechod na ďalšiu hranu odpovedá použitiu LF-mapovania v BWT, ktorú daný graf reprezentuje.



i	LF[i]	L[i]
0	1	a
1	10	r
2	8	d
3	0	$
4	9	k
5	11	r
6	2	a
7	3	a
8	4	a
9	5	a
10	6	b
11	7	b

Obrázok 1.7: Wheelerov graf G reprezentujúci BWT L = ard$kraaaabb. Riadky predstavujú zoznam susedov grafu G

1.4.2. ÚSPORNÁ REPREZENTÁCIA WHEELEROVÝCH GRAFOV

Jednou z vlastností Wheelerových grafov pri predstavení ako konceptu pre reprezentovanie štruktúr nad reťazcami, bola schopnosť ich kompaktne uložiť, kde bola zároveň definovaná aj ich úsporná reprezentácia.

Definícia (Úsporná reprezentácia Wheelerových grafov) (podľa [GMS17]) Nech G je Wheelerov graf s n vrcholmi a m hranami. Nech $x_0 < x_1 < ... < x_{n-1}$ označuje usporiadanie vrcholov grafu G. Pre každé i = 0, ..., n – 1 označme l_i ako počet výstupných, a k_i ako počet vstupných hrán vrcholu x_i. Nech in a out sú reťazce nad binárnou abecedou {0, 1} s dĺžkou n + m, pre ktoré platí:

$$in = 0^{k_0}10^{k_1}1\ldots0^{k_{n-1}}, \; out = 0^{l_0}10^{l_1}1\ldots0^{l_{n-1}}$$

Nech L_i označuje multimnožinu označení výstupných hrán z vrcholu x_i, a nech L je reťazec, pre ktorý platí $L = L_0L_1...L_{n-1}$ (teda má dĺžku m). Potom reťazce L, in, a out, spolu tvoria úspornú reprezentáciu Wheelerovho grafu G.

Táto úsporná reprezentácia bola definovaná pre Wheelerove grafy s pôvodnou definíciou, a teda pre grafy, ktorých niektoré vrcholy nemuseli mať vstupné alebo výstupné hrany. Keďže používame Baierovú definíciu Wheelerových grafov, kde každý vrchol musí mať aspoň jednu vstupnú a výstupnú hranu, potom by použitím vyššej definície úspornej reprezentácie vyplývalo, že pred každým znakom 1 v reťazcoch in a out je aspoň jeden výskyt znaku 0. Túto 0 môžeme zanedbať, a dostaneme novú, kompaktnejšiu reprezentáciu Wheelerových grafov.

Definícia (Baierova úsporná reprezentácia Wheelerových grafov) (podľa [Bai20]) Použime všetky označenia a priradenia tak, ako v definícií vyššie, okrem reťazcov in a out. Tieto majú dĺžku m + 1, a platí pre ne:

$$in = 10^{k_0-1}10^{k_1-1}1\ldots0^{k_{n-1}-1}1, \; out = 10^{l_0-1}10^{l_1-1}1\ldots0^{l_{n-1}-1}1$$

Potom reťazce L, in, a out spolutvoria úspornú reprezentáciu Wheelerovho grafu G.

i	L[i]	out[i]	in[i]	F[i]
0	b	1	1	a
1	a	0	1	a
2	a	0	1	a
3	a	0	0	a
4	c	1	1	a
5	a	0	0	a
6	b	1	0	a
7	a	1	1	b
8	c	1	0	b
9	a	0	1	b
10	c	1	0	b
11	b	0	1	c
12	c	1	0	c
13	b	0	1	c
14	a	1	0	c
		1	1	

Obrázok 1.8: Úsporná reprezentácia Wheelerovho grafu z obrázku 1.6. Okrem definovaných reťazcov L, out a in, je tu priradený aj reťazec F, v ktorom sú označenia vchádzajúcich hrán

V tomto texte budeme používať Baierovu definíciu úspornej reprezentácie Wheelerových grafov (teda podľa [Bai20]). V tejto definícií sú na koncoch reťazcov in a out pridané ukončovacie znaky 1. Tieto zjednodušujú zistenie toho, či má posledný vrchol násobné hrany, prípadne zistenie označení daných hrán.

Na získanie reťazca L stačí postupne prejsť všetkými vrcholmi grafu od najmenšieho po najväčší, a pre každý vrchol pridať na koniec reťazca L znaky výstupných hrán. Podobne sa dajú vytvoriť reťazce in a out, kde sa prechodom vrcholom x_i pridá na koniec reťazca in 10^{k_0-1} a koniec reťazca out 10^{l_0-1}, len s tým rozdielom, že na koniec sa pridajú ešte ukončovacie 1-tky.

V reťazci L sú uložené označenia hrán, reťazec out reprezentuje výstupné hrany, a reťazec in vstupné. Tiež si dodefinujeme reťazec F, v ktorom sú uložené označenia vstupných hrán. Tento síce nie je potrebný pre reprezentáciu grafu, ale je vhodný pre vizualizáciu.

Pre i-ty vrchol platí, že znaky podreťazca L[$select_{out}(1, i)$, $select_{out}(1, i + 1) – 1$] sú práve znaky všetkých označení jeho vychádzajúcich hrán. Pre i-tu hranu v reťazci out platí, že jej označenie je L[i], a vrchol, z ktorého vychádza je $rank_{out}(1, i + 1) – 1$. Funguje tu tiež LF mapovanie, takže i-ty výskyt znaku v L a i-ty výskyt znaku v F predstavuje tú istú hranu. Pre i-tu hranu v in platí analogicky, že jej označenie je F[i] a vrchol, do ktorého

vchádza je $rank_{in}(1, i + 1) - 1$. Tiež vieme zistiť pozíciu i-teho vrcholu v reťazci out, jednoduchým dotazom $select_{out}(1, i)$. Pomocou týchto vecí je možné sa navigovať po úspornej reprezentácií Wheelerovho grafu.
Príklad úspornej reprezentácie Wheelerovho grafu možno nájsť na obrázku 1.8, kde je úsporná reprezentácia Wheelerovho grafu z obrázku 1.6. Pre Wheelerove grafy, ktoré reprezentujú BWT je táto reprezentácia zložená z BWT ako reťazca L, a reťazce in a out sú iba samé jednotky.

1.4.3. TUNELOVANIE WHEELEROVÝCH GRAFOV

V sekcii o BWT sme uviedli, že najdôležitejšou vlastnosťou BWT pre kompresné algoritmy je zhlukovanie rovnakých znakov dokopy. Ukázali sme, že je to spôsobené tým, že podobným kontextom často predchádza rovnaký znak. BWT využíva len predchádzanie jedného znaku. Často sa však stáva, že podobným kontextom predchádzajú rovnaké reťazce. Túto skutočnosť si všimol a ako prvý využil Baier [Bai18] na zmenšenie veľkosti BWT, pri zachovaní jej bezstratovej vratnosti. Zadefinoval prefixové intervaly[2], čo sú reťazce, ktoré sa opakujú, a predchádzajú podobné kontexty.

Definícia (Prefixový interval) (podľa [Bai20])Nech G je Wheelerov graf, s označovacou funkciou hrán λ. Prefixový interval je súbor $h \geq 2$ ciest, $p_0 = (v_{0,0}, v_{0,1}, \dots, v_{0, w-1})$, ..., $p_{h-1} = (v_{h-1,0}, v_{h-1,1}, \dots, v_{h-1, w-1})$, s rovnakou dĺžkou $w \geq 2$, pre ktoré platia nasledujúce podmienky:

1. Každý vrchol $v_{i, j}$ má práve jednu vstupnú a jednu výstupné hranu
2. Dané cesty sú „paralelné“, teda $v_{i+1, j}$ je priamym nasledovníkom $v_{i, j}$, v zmysle usporiadanie vrcholov Wheelerovho grafu
3. Každá cesta je rovnako „označená“, teda $\lambda(v_{i, j}, v_{i, j+1}) = \lambda(v_{i', j}, v_{j'+1})$

pre každé korektné i, i´, j, w.
Definujeme dĺžku prefixového intervalu ako w – 1, jeho výšku ako h.

Baier v svojej práci [Bai20] uvádza aj algoritmus, ktorý pre Wheelerov graf reprezentujúci BWT vypočíta všetky prefixové intervaly, ktoré sa nedajú predĺžiť vľavo, a zároveň sa nedá pridať k žiadnemu z nich ďalšia cesta. Tento algoritmus používame v programe, ale nevizualizujeme jeho vykonávanie. Použijeme iba jeho výsledok, a to

[2] Pôvodne nazývané bloky

tak, že vyberieme z výsledných prefixových intervalov taký, ktorý má práve dve cesty, ktoré sú vrcholovo disjunknté a má čo najväčšiu dĺžku. Tieto cesty potom zafarbíme dvoma rozličnými farbami. Príklad takéhoto prefixového intervalu je možné nájsť v ľavej časti obrázku 1.9, kde sú jeho cesty zafarbené modrou, a červenou farbou.

Definícia (Tunelovanie) (podľa [Bai20]) Nech G = (V, E) je Wheelerov graf, s funkciou λ označujúcou hrany, a nech p_0, až p_{h-1} sú vrcholovo disjunktné cesty prefixového intervalu grafu G, so šírkou w, a dĺžkou w.
Proces tunelovania prefixového intervalu je definovaný ako spájanie vrcholov z každého stĺpca prefixového intervalu a tiež aj hrán, medzi dvoma susednými stĺpcami. Definujme F, ako množinu vrcholov ciest p_2 až p_n. Potom stunelovaný Wheelerov graf $G' = (V', E')$ z funkciou λ' pre označovanie hrán definujeme ako

$$V' = V \setminus F$$

$$E' = E \setminus \big((F \times V) \cup (V \times F)\big)$$

$$\cup \big\{\big(u, v_{1,1}\big) \mid u \in V, \big(u, v_{1,j}\big) \in E, \text{ pre niektoré } j,\ 1 \le j \le h\big\}$$

$$\cup \{(u_w, v) \mid v \in V, \big(v_{w,j}, v\big) \in E, \text{ pre niektoré } j,\ 1 \le j \le h\}$$

$$\lambda' = \begin{cases} \lambda\big(u, v_{1,j}\big) & \text{ak } v = v_{1,1} \text{ a } \big(u, v_{1,j}\big) \in E \\ \lambda\big(u_{w,j}, v\big) & \text{ak } u = v_{w,1} \text{ a } \big(v_{w,j}, v\big) \in E \\ \lambda(u, v) & \text{inak} \end{cases}$$

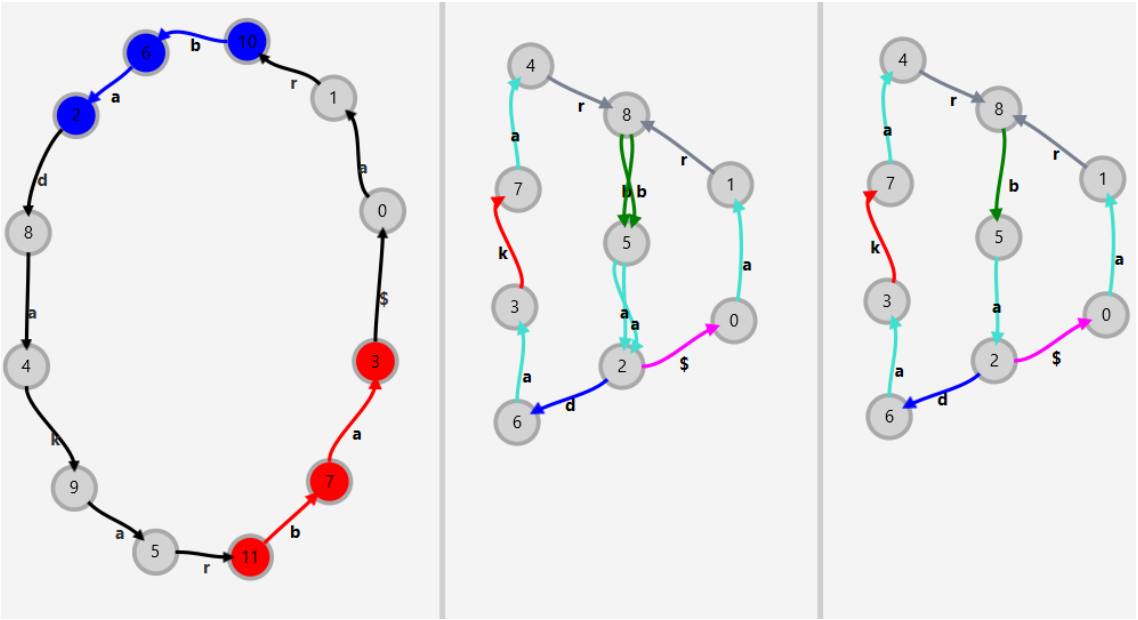
Samotný algoritmus na tunelovanie prefixového intervalu sa skladá z dvoch krokov.
V prvom kroku sa postupne spájajú vrcholy v jednotlivých stĺpcoch prefixového intervalu. Pre spojenie dvoch vrcholov s hodnotami i a i + 1 algoritmus zmení prvky polí in a out na pozícii i + 1 z 1 na 0. Zmena prvku poľa out na pozícii i + 1 z 1 na 0 znamená, že každá hrana, ktorá vychádzala z vrcholu s hodnotou k, k > i, bude vychádzať z vrcholu s hodnotou k – 1. Analogicky pre pole in, kde sa podobne menia vstupné pozície hrán. Ak sa takto zmenia obe polia in a out, potom sa hrany incidentné s vrcholom i + 1 stanú incidentné s vrcholom i. Týmto posunom sa zároveň odstránia hrany z najväčšieho vrcholu s nejakými hranami, keďže tieto hrany budú po posune incidentné s vrcholom s hodnotou o jedna menšou. Pre spojenie n vrcholov algoritmus zmení prvky polí in a out na pozíciách [i + 1, i + n – 1] z 1 na 0. Po vykonaní tohto úkonu pre všetky stĺpce sa vytvorí graf, ktorý je podobný výslednému grafu, len medzi vrcholmi budúceho tunelu

vznikli násobné hrany, pričom násobné hrany medzi dvoma vrcholmi majú rovnaké označenie.

V ďalšom kroku sa tieto hrany spoja do jednej, a vznikne stunelovaný Wheelerov graf. Pri tomto spomenieme, že pri tunelovaní susedných prefixových intervalov vznikne násobná hrana, a preto je zmena v definícií Wheeleroveho grafu na mutltigraf dôležitá.

Pri vizualizácií prvej časti tunelovania neposúvame hrany pri každej zmene polí in alebo out, ale až keď sa zmenia obe tieto hodnoty, a teda je výsledkom spojenie niektorých vrcholov. Druhú časť algoritmu nevizualizujeme krok po kroku, ale zobrazíme až výsledný graf so zlúčenými hranami.

Na obrázku 1.9 vľavo je zobrazený graf pred vykonaním prvej časti algoritmu (s vyznačenými cestami prefixového intervalu, ktoré sa budú tunelovať). V strednej časti daného obrázka je zobrazený, ako ho vizualizujeme po dokončení prvej časti. Nie je to však jeho skutočná podoba vzhľadom na úspornú reprezentáciu, s ktorou algoritmus pracuje. V tej si totiž ukladá pozície začiatku a konca tunela a medzi skutočným a vykresleným grafom je malý rozdiel. V pravej časti obrázka je ukážka po druhej časti algoritmu a teda je na nej stunelovaný Wheelerov graf.

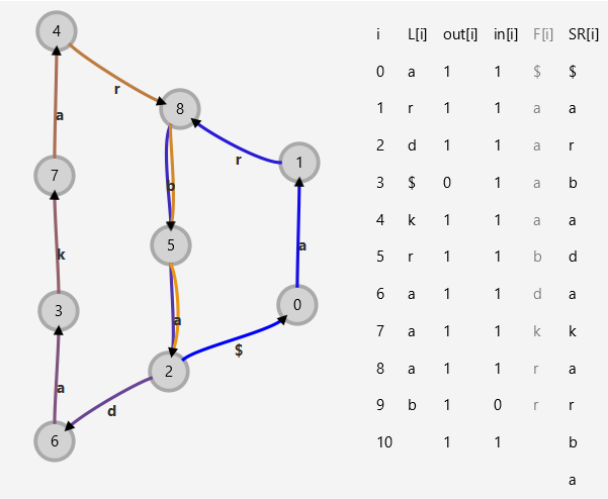


Obrázok 1.9: Na obrázku vľavo sú vyznačené cesty prefixového intervalu. V strednej časti sú vrcholy týchto ciest po pároch zlúčené a všetky vrcholy grafu sú prečíslované. V pravej časti sú zlúčené násobné hrany medzi už zlúčenými vrcholmi.

Ak graf G, ktorý reprezentuje BWT pre 0-ukončený reťazec S, stunelujeme množinou prefixových intervalov, z ktorej sú všetky dvojice vrcholovo disjunktné, tak z výsledného

stunelovaného grafu sa dá získať pôvodný reťazec S [Bai20]. Rovnako ako pre pôvodný graf G, sa to dá vykonaním pochôdzky so začiatkom v hrane s označením $. Je to možné kvôli tomu, že tunelovanie nemení štruktúru ani relatívne usporiadanie hrán, okrem spojenia paralelných ciest v prefixovom intervale. Jediná zmena sa nachádza pri vstupe a výstupe z tunela. Pri vstupe do tunela problém nevzniká, keďže cesty sa spájajú a je dané jednoznačné pokračovanie pochôdzky. Problém nastáva ale pri výstupe z tunela, kde je potrebné sa rozhodnúť, ktorou hranou chceme pokračovať. Platí však, že hrane, ktorá je i-ta v relatívnom poradí hrán, ktoré vstupujú do tunela, prislúcha i-ta výstupná hrana v relatívnom poradí z daného tunela. To vyplýva z toho, že pred tunelovaním boli dané cesty paralelné a teda vstupu i-tou hranou do prefixového intervalu prislúchala i-ta cesta, a zároveň i-ta výstupná hrana z daného prefixového intervalu. Keďže tunelovanie nemení relatívne usporiadanie hrán, stačí si pri vstupe do tunela zapamätať relatívnu pozíciu vstupnej hrany a pri východe použiť danú hodnotu. Označenia hrán takto vytvoreného sledu vytvárajú reverz pôvodného reťazca S (vykonať reverz reťazca je triviálne). Časová zložitosť nájdenia ďalšej hrany je $O(\log_2|\Sigma|)$, keďže je potrebné okrem iného vykonať rank_L dotaz. Zvyšné časti majú konštantnú časovú zložitosť. Celková časová zložitosť nájdenia pôvodného reťazca je teda $O(n \log_2|\Sigma|)$.

Na obrázku 1.10 je príklad takto opísanej pochôdzky v grafe.



Obrázok 1.10: Graf z pravej časti obrázku 1.9 po vykonaní pochôdzky. Začiatok sledu je modrej farby, postupne prechádzajúcej do oranžovej ku jeho koncu. Pri východe z tunelu (8,5,2) sa pokračuje hranou (2,6), keďže je v relatívnom poradí prvá, rovnako ako pri vstupe bola v prvá relatívnom poradí hrana (1,8).

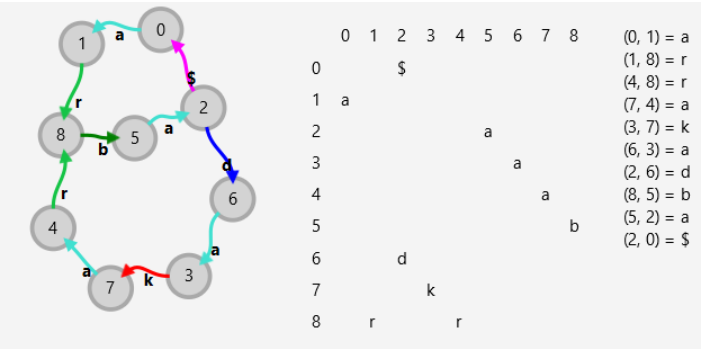
2. VIZUALIZÁCIA GRAFOVÝCH ALGORITMOV

2.1. VIZUALIZÁCIA GRAFOV

V tejto časti si ukážeme, aká metóda zobrazenia grafov je pre naše účely vhodná a potom pre ňu navrhneme, aké podmienky a možnosti pre užívateľa budeme požadovať.

2.1.1. METÓDY ZOBRAZENIA GRAFOV

Bežné metódy na vizualizáciu grafov sú pomocou zoznamu susedov, matice susedností, a pomocou uzlov (ktoré môžeme zobraziť ako kruhy s prípadnou hodnotou vrcholu napísanou v strede daného kruhu) ako vrcholy a úsečiek alebo šípok ako hrany, túto možnosť nazveme uzlové zobrazenie. Na obrázku 2.1 je zobrazenie jedného grafu pomocou týchto troch metód. Prejdime si tieto metódy a pozrime sa na ich využiteľnosť pre naše potreby.



Obrázok 2.1: Zľava doprava: uzlové zobrazenie, matica susedností, a zoznam susedov pre ten istý graf.

Zoznam susedov sme vylúčili, keďže neponúka veľké možnosti pre vizualizáciu. Tiež predstavuje veľké obmedzenia pri prípadnom rozšírení aplikácie, kde je zložité vytvárať vrcholy bez unikátnych identifikačných prvkov.

V istom zmysle je uzlové zobrazenie podobné zoznamu susedov, len vzdialenosť na obrazovke môže byť medzi dvoma susednými vrcholmi ľubovoľná a tiež incidencia hrán je viditeľná automaticky. Keďže uzlové zobrazenie nám dáva väčšiu mieru voľnosti v zobrazení, budeme preferovať toto zobrazenie.

Keďže sme zoznam susedov vylúčili, porovnajme maticu susedností a uzlové zobrazenie. Pre niektoré potreby sú vhodné obidve metódy, napríklad zobrazenie ešte neidentifikovateľného vrcholu, prípadne zvýraznenie hrán, alebo vrcholov a presunutie

vrcholov blízko seba. Pri matici susedností je trochu zložitejšie zobraziť neoznačenú hranu tak, aby to bolo odlíšiteľné od označenia hrany, prípadne jej zvýraznenia. Avšak pri niektorých našich dôležitých potrebách je možné nájsť rozdiely medzi týmito metódami zobrazenia. Jednou z hlavných úloh Wheelerových grafov je reprezentácia reťazcov, ktorým zodpovedá cesta v grafe, a teda dobrá čitateľnosť ciest je jednou z hlavných potrieb. Pri porovnávaní čitateľnosti grafov [GFC04] bolo pokusmi ukázané, že nájsť cestu medzi dvoma vrcholmi trvalo testovaným dlhšie pri maticiach susedností ako pri uzlovom zobrazení. Síce žiaden algoritmus, ktorý chceme vizualizovať, nehľadá cestu medzi dvoma vrcholmi, ale chceme vizualizovať algoritmy, ktoré hľadajú cesty na základe označenia hrán. Môžeme teda predpokladať, že pre užívateľov bude jednoduchšie sledovať tieto cesty na uzlovom zobrazení.

2.1.2. PODMIENKY PRE UZLOVÉ ZOBRAZENIE

Vyššie sme si vybrali iba uzlové zobrazenie, pozrime sa teraz na to, čo by bolo vhodné povoliť užívateľovi a z toho vyplývajúce požiadavky pre túto metódu zobrazenia.

Keďže je výskyt toho istého podreťazca vo Wheelerových grafoch možný aj viackrát, bolo by vhodné pre užívateľa mať možnosť hýbať vrcholmi, aby dokázal zvýrazniť niektoré, preňho zaujímavé výskyty. Tieto výskyty sa môžu aj prekrývať, teda niektoré hrany a vrcholy by bolo žiadúce vedieť označiť viacerými možnosťami.

Tiež je vhodné pridať funkcionalitu, s ktorou by sa sprehľadnil daný graf automaticky, to znamená, aby každá možná cesta bola čo najlepšie čitateľná.

Bolo by tiež žiadúce usporiadať vrcholy na jednu líniu podľa ich hodnôt a zmeniť vykreslenie hrán tak, aby boli viditeľné všetky ich označenia. Žiaľ táto funkcionalita nebola implementovaná, kvôli časovej tiesni.

2.2. VIZUALIZÁCIA ALGORITMOV

Na to, aby sme mohli vizualizovať algoritmy, je potrebné ukázať, čím sú vo všeobecnosti definované, ako ich definícia určuje možnosti ich vizualizácie a tiež ako podmieňuje niektoré jej vlastnosti. V Knuthovej definícií [Knu08] má algoritmus týchto 5 vlastností: konečnosť, determinovanosť, vstup, výstup a efektivitu. V nasledujúcej časti opíšeme každú z týchto vlastností a pokúsime sa určiť vlastnosti, ktoré budeme požadovať od programu, ktorý bude vizualizovať algoritmy tejto práce.

Determinovanosť – *každý krok je jednoznačne a presne definovaný*. Pre vizualizáciu je dôležité, aby sme v každom čase vedeli zvýrazniť časť, s ktorou práve pracuje, pričom je potrebné odlíšiť viaceré zvýraznené časti. Ak by pracoval na viacerých miestach súčasne, chceme všetky časti od seba odlíšiť. Pre lepšie pochopenie vykonávajúceho sa kroku je vhodné ho animovať. Takisto chceme vypísať užívateľovi, aké hodnoty majú jednotlivé premenné, prípadne, aký je celý obraz pamäti dôležitej pre algoritmus. Tiež v prípade, že je pri algoritme zobrazený aj pseudokód, je vhodné zvýrazniť riadok, v ktorý algoritmus práve vykonáva. Rovnako je vhodný slovný opis vykonávaného kroku.

Efektívnosť a konečnosť– *algoritmus má byť efektívny a musí skončiť po vykonaní konečného počtu krokov*. Keďže má byť tento softvér využitý aj na edukačné účely, je vhodné obmedziť dĺžku vykonávania na vhodne krátku dobu, keďže čas strávený sústredením sa na určitú dobu nemôže byť ľubovoľne dlhý. Vieme to urobiť viacerými metódami: ohraničením vstupu, prípadne zobrazovaním iba častí, v ktorých sa deje niečo, čo sa dá vizualizovať a zvyšné vypočítať bez zaťažovania užívateľa, prípadne mu zobraziť iba výsledok danej operácie. Tiež sa dá tento problém vyriešiť pri niektorých algoritmoch pomocou pridania možnosti skákať na určité miesta v kóde, odkiaľ by softvér korektne vizualizoval ďalšie kroku algoritmu.

Vstup – *algoritmus pracuje s nejakým počtom vstupov*. Chceme vizualizovať viaceré algoritmy, kde vstupy môžu byť napríklad orientované grafy s označenými hranami a ohodnotenými vrcholmi, alebo reťazce znakov. Užívateľ by mal mať možnosť pracovať s ľubovoľnými, správne formátovanými vstupmi, ohraničenými jedine ich veľkosťou. Mal by mať taktiež možnosť ľubovoľný vstup vytvoriť. Toto vytvára rôzne zložitosti pre vstupy. Kým vloženie ľubovoľného reťazca je celkom jednoduché, vložiť ľubovoľný graf je zložitejšie, pretože je potrebné umožniť vytvoriť ohodnotenie hrán aj očíslovanie vrcholov. Malo by to však byť stále jednoduché, aby mohol užívateľ vyskúšať čo najväčšie spektrum vstupov. V konečnom dôsledku by mal byť program pre každý vstup schopný vizualizovať správne algoritmy.

Výstup – *algoritmus má aspoň jeden výstup*. Chceme, aby boli výstupy ľahko čitateľné a zároveň, aby bolo užívateľovi čo najzrozumiteľnejšie, ako sa algoritmus k danému výstupu dostal.

Tiež budeme od každého výstupu každého algoritmu požadovať, aby jeho výsledkom bola záporná odpoveď (napríklad pri zisťovaní či je daný graf Wheelerov graf), alebo aby sa dal ľahko zmeniť na vstup pre nejaký (nie nutne iný) algoritmus (napríklad pri ukončení hľadania reťazca chceme, aby mohol užívateľ na danej štruktúre znovu hľadať nejaký reťazec). Toto odstráni nutnosť vytvárania toho istého vstupu, ak by mal záujem užívateľ tento vstup skúmať hlbšie.

2.3. VIZUALIZÁCIA ZVYŠNÝCH ČASTÍ

Zo zvyšných častí, ktoré vizualizujeme, je to hlavne BWT matica a pseudokódy algoritmov.

Pre matice chceme mať možnosť vpisovať indexy, pre zaznačenie výskytu znakov, zvýrazňovať jednotlivé prvky a aj možnosť mať šípky medzi jednotlivými prvkami kvôli LF-mapovaniu.

Pri vizualizovaní pseudokódu sa snažíme použiť čo najvšeobecnejší kód, založený na jazykoch podobných jazyku C. Problém vznikol pri inicializovaní poľa, kde často býva namiesto „skutočného“ kódu popis, aké pole, s akými hodnotami, veľkosťou a typom premenných sa vytvára. Keďže má byť daný softvér viac-jazyčný, pokúsili sme sa toto zjednodušiť a inicializácia poľa P, dĺžky n je v pseudokóde: P = [n].

3. IMPLEMENTÁCIA

V tejto kapitole je popis implementácie. Nižšie sú opísané hlavné časti programu a najdôležitejšie funkcie na prácu s nimi.

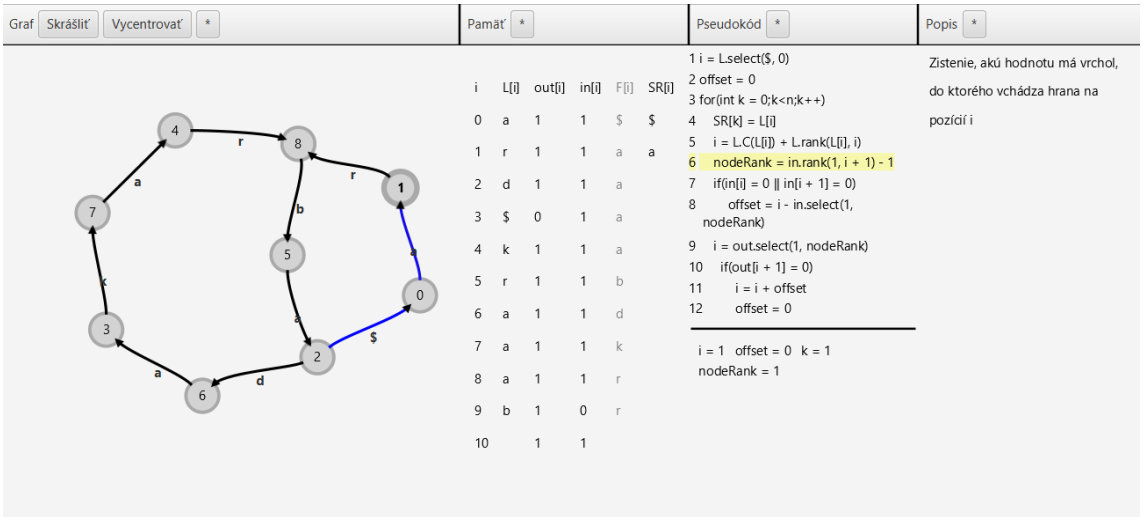
Celý program je napísaný v jazyku Java (verzia 17), s použitím JavaFX. Používali sme vývojové prostredie IntelliJ Idea [JB01].

3.1. DISPLAY

Trieda Display predstavuje jedno z okien, na ktorom sa vizualizuje algoritmus. Z tejto triedy dedia konkrétne typy okien predstavené nižšie.

Oknu vytvorenému triedou Display sa dá meniť veľkosť aj počas behu programu, keďže každý jej potomok prepisuje funkciu *resize()*. Tiež je možné ich minimalizovať v prípade, ak sa užívateľ rozhodne, že dané okno nepotrebuje, respektíve maximalizovať v opačnom prípade.

Z grafického hľadiska je okno rozdelené na dve časti, v spodnej je plocha pre vizualizáciu, v hornej časti je lišta z názvom daného okna, prípadne tlačidlá, ktoré pridal konkrétny potomok triedy Display a na koniec tlačidlo na minimalizáciu/maximalizáciu daného okna. Na obrázku 3.1 je príklad štyroch rozličných okien uprostred vizualizácie algoritmu.



Obrázok 3.1: Ukážka 4 okien vytvorených potomkami triedy Display. Zľava: GraphDisplay, MatrixDisplay, CodeDisplay, TextDisplay

3.1.1. CODEDISPLAY

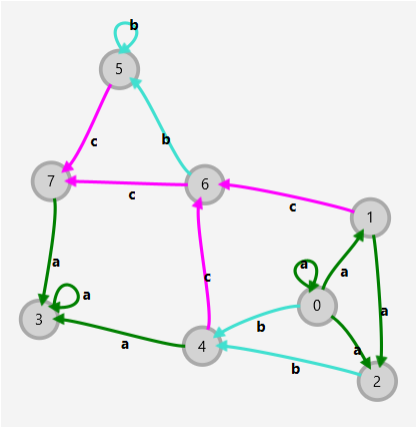
CodeDisplay je potomok triedy Display, určený na zobrazenie kódu. Umožňuje pridávať riadky kódu ako reťazce, ktoré potom automaticky očísluje. Tiež umožňuje pridávať a odoberať premenné, ktorým sa dajú meniť hodnoty. Dané premenné sa dajú zvýrazňovať, napríklad pri ich zmene. Podobne sa dajú zvýrazniť riadky kódu, čo je použité pri zvýraznení riadku, ktorý sa práve vykonáva. Jeho pracovné okno je rozdelené na dve časti. Vo vrchnej časti sú riadky kódu, v spodnej sú premenné. Príklad CodeDisplay-a je v tretej časti obrázku 3.1.

3.1.2. GRAPHDISPLAY

Znázornenie uzlového zobrazenia grafov umožňuje trieda GraphDisplay. Táto trieda umožňuje pridávať hrany a vrcholy. Je možné zvýrazňovať pridané vrcholy a nastavovať im súradnice na pracovnej ploche. Vrcholom aj hranám je možné nastavovať farbu a tiež meniť ich ohodnotenia. Graficky sú vrcholy tvorené kruhom s textom jeho ohodnotenia v strede. Hrany sú tvorené Bézierovou krivkou tretieho stupňa. Príslušný koniec danej krivky nastavuje vrchol tak, aby viacero jeho incidentných hrán nemalo spoločný dotyk s daným vrcholom. Zvyšné kontrolné body sú nastavené tak, aby bol vstup do vrcholu pod pravým uhlom. Hranám je tiež možné priradiť viacero „ciest“, čo znamená, že sa vykreslia viaceré krivky namiesto jednej. Táto funkcia je použitá pri získavaní pôvodného reťazca zo stunelovaného Wheelerovho grafu, kde v tejto verzii programu dáme každej hrane tunelu dve takéto „cesty“. Pomocou tejto funkcie sa dá teoreticky znázorniť aj viacnásobná hrana (my sme násobné hrany zobrazovali ako dve rozličné hrany).

GraphDisplay má tiež funkciu *beautify()*, ktorá pomocou silovo riadeného kreslenia prekreslí daný graf. V tomto type kreslenia sa umiestňujú vrcholy na ploche priradením určitých síl hranám a vrcholom. Vrcholy sú považované za energeticky kladne nabité častice, pre ktoré platí Coulombov zákon. Hrany sú považované za pružiny, na ktoré je aplikovaný Hookov zákon. Tiež sme použili slabú lineárnu gravitačnú silu a malý náhodný pohyb. Na obrázku 3.2 je zobrazený graf z obrázku 1.6, na ktorou bola aplikovaná daná funkcia.

V prvej časti obrázku 3.1 je zobrazený príklad GraphDisplay-a.



Obrázok 3.2: Graf z obrázku 1.6, na ktorom bola aplikovaná funkcia *beautify()*

3.1.3. MATRIXDISPLAY

Trieda MatrixDisplay umožňuje zobrazovanie matíc. V našom programe ju používame aj na zobrazenie polí, kde v prvom riadku vypíšeme ich názvy a vo zvyšných hodnoty prvkov daných polí. Jednotlivý prvok matice je možné zvýrazniť, meniť mu farbu textu alebo farbu pozadia, zmeniť jeho text, alebo mu pridať pravý dolný index.

Tiež je možné pridať šípku medzi dvoma prvkami tabuľky (využité pri vizualizácií LF-mapovania). Tejto šípke je možné nastaviť farbu a tiež ju zvýrazniť. Naprogramovali sme mať možnosť šípku iba medzi rovnakými alebo susednými stĺpcami/riadkami, keďže pre naše potreby stačí dokonca iba šípka medzi susednými stĺpcami. Tiež preto, že sme nenašli algoritmus, ktorý by potreboval šípky medzi ľubovoľnými prvkami matice.

Príklad MatrixDisplayu je na obrázku 3.1 v druhej časti, kde je využitý pre zobrazenie viacerých polí.

3.1.4. TEXTDISPLAY

Trieda pre znázornenie textu. V tejto verzii je možné pridať text, ktorému však nie je možné zmeniť štýl. Je však možné vyskladať text z viacerých častí a zadať pre každú časť, či sa má prekladať, alebo nie. Používame ju na popis algoritmu. Na obrázku 3.1 je v štvrtej časti.

Podobný TextDisplay-u je SelectorDisplay, ktorý obsahuje nadpis, text a tlačidlo, ktorý používame na výber ďalšieho algoritmu.

3.2. ANIMÁCIE

Pre potreby animovať jednotlivé kroky vizualizovaného algoritmu sme naprogramovali triedu Animation. Dajú sa do nej pridávať objekty, ktoré potom môže viacerými spôsobmi animovať. Typy objektov (Interface-y), ktoré sa dajú animovať sú:

- ColorAnimatable: Pre zmenu farby daného objektu. Implementujú ho vrcholy a hrany v grafe, šípka v matici, a texty, pomocou čoho zvýrazňujeme napr. prvky v maticiach a riadky kódu.
- AppearAnimatable: podobné ako ColorAnimatable, ale pred začiatkom najskôr zviditeľní objekt
- MoveAnimatable a RelativeMoveAnimatable: pre pohyb objektu na pracovnej ploche Displaya. Implementuje ich trieda vrchol v grafu

Pri vkladaní objektu do triedy Animation sa spolu s ním vložia argumenty pre začiatok a koniec, buď začiatočná a konečná farba, alebo začiatočná a konečná pozícia (v prípade vloženia iba jednej sa ako druhá zoberie aktuálna hodnota objektu). Pri spustení animácie funkciou *startAnimation()* sa pre každý vložený objekt vypočíta zmena, aká sa má vykonať za jednotlivý krok a potom sa pomocou triedy java.util.Timer vykoná približne každých 16 milisekúnd.

Animáciu je možné vykonať oboma smermi pomocou funkcie *setForward(boolean)*.

Podobná trieda k Animation je trieda StepManager, v ktorej sa vykonávajú kroky algoritmu, ktoré nie sú zatiaľ animované – napr. zmeny hodnôt premenných. Máme v pláne v budúcnosti tieto triedy zlúčiť do jednej, aby sa všetky zmeny Display-ov vykonávali v jednej triede.

3.3. WINDOWMANAGER

Trieda WindowManager je jednou z hlavných tried programu. Používame pri nej návrhové vzory Factory a Observer. Vytvára potomkov triedy Display a zároveň si ukladá ich referencie. Pri zmene veľkosti okna, alebo pri minimalizácií/maximalizácií niektorého z Display-ov prepočíta pre každý Display jeho novú veľkosť a tú mu nastaví. Tiež sa pomocou tejto triedy pridávajú ovládacie prvky programu: tlačidlá, vstupné polia a podobne.

3.4. ALGORITHM

Každý vizualizovaný algoritmus má vlastnú triedu. Vizualizovali sme definície sufixového poľa, BWT, Wheelerových grafov, a potom inverznú BWT, vytvorenie BWT pomocou sufixového poľa, vyhľadávanie vzoru pomocou BWT, vytvorenie Wheelerovho grafu reprezentujúceho BWT, tunelovanie takéhoto grafu a nakoniec získanie reprezentovaného reťazca zo stunelovaného grafu.

Okrem toho sú tu ďalšie dve triedy, ktoré sme zahrnuli do algoritmov, aj keď nič nevizualizujú. To sú dve triedy pre výber algoritmov pomocou viacerých SelectorDisplay-ov.

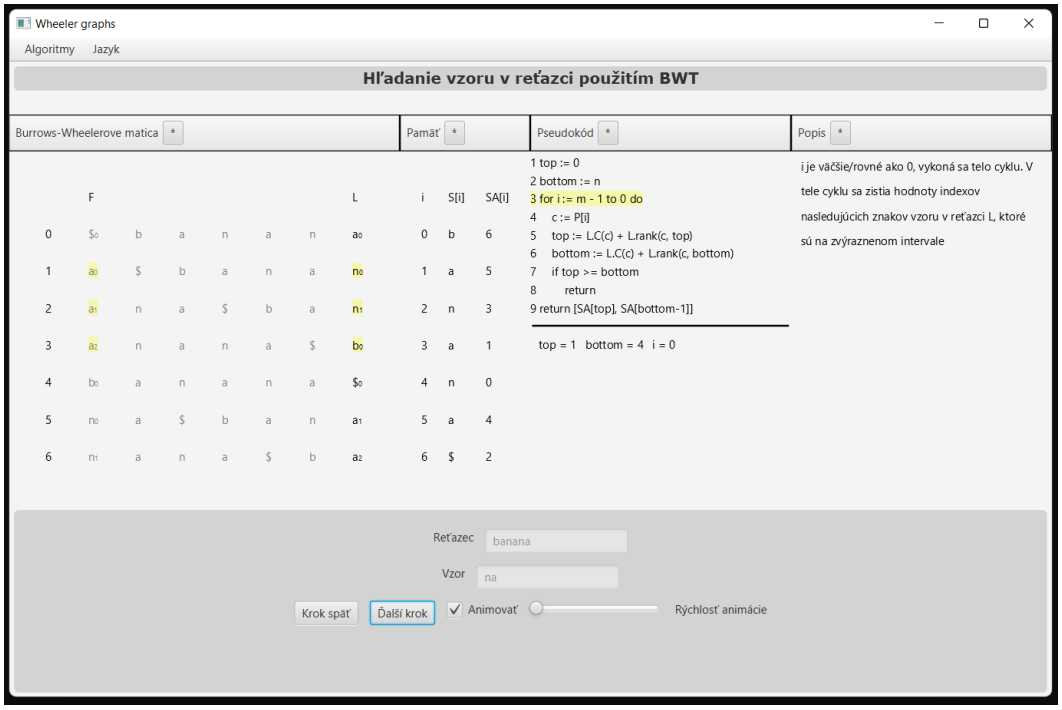
Každý konštruktor každého algoritmu obsahuje referenciu na triedu AlgorithmManager, ktorá funguje podobne ako Factory, s tým rozdielom, že keď algoritmus po svojom skončení zavolá funkciu *changeAlgorithm()*, tak AlgorithmManager odstráni predošlý algoritmus a potom vytvorí nový.

3.5. ZVYŠNÉ SÚČASTI IMPLEMENTÁCIE

Zo zvyšných vlastností aplikácie je asi najdôležitejšou viacjazyčnosť. Pre toto sme pretransformovali a trochu vylepšili súčasť inej aplikácie [Ko+07], v ktorej už bola viacjazyčnosť implementovaná.

4. POPIS OVLÁDANIA

Obrazovka softvéru sa skladá z menu s výberom dátových štruktúr a s menu pre zmenu jazyka. Pod tým je názov práve vykonávaného algoritmu. Pod ním je rad Display-ov, na ktorých sa vizualizuje algoritmus. Úplne na spodku okna je obdĺžnik s ovládacími prvkami. Takmer každý algoritmus má jeden riadok s ovládacími prvkami rovnaký a to s tlačidlami Krok späť a Ďalší krok, so zaškrtávacím políčkom hovoriacim či animovať a s posuvnou lištou na nastavenie rýchlosti animácie. Na obrázku 4.1 sa nachádza ukážka okna aplikácie.



Obrázok 4.1: Výzor okna aplikácie pri vizualizácií algoritmu

ZÁVER

V tejto práci sme najprv popísali teoretické pojmy a objekty, menovite sufixové polia, Burrows-Wheelerovu transformáciu, Wheelerove grafy, a tiež sme popísali ich niektoré bežné aplikácie. V druhej kapitole sme si uviedli teoretické základy a princípy vizualizácie, podľa ktorých sme naprogramovali aplikáciu, ktorá vizualizuje vybrané algoritmy a dátové štruktúry definované v prvej časti. Nakoniec sme opísali implementáciu našej aplikácie, jej hlavné časti a ich najdôležitejšie funkcie.

Počas budúcej práce na projekte by sme chceli opraviť chyby, ktoré aplikácia obsahuje, rozšíriť možnosti terajších algoritmov, napríklad pridať opakované tunelovanie toho istého grafu, a tiež pridať ďalšie algoritmy, ktoré pracujú s danými štruktúrami. Chceli by sme tiež pridať možnosti na vykresľovanie niektorých súčasti, napríklad grafov, kde by sme chceli pridať možnosť pre viac hranovo orientovaný prístup.

ZOZNAM POUŽITEJ LITERATÚRY

[AKO04] ABOUELHODA, Mohamed Ibrahim; KURTZ, Stefan; OHLEBUSCH, Enno (2004) "Replacing suffix trees with enhanced suffix arrays", Journal of Discrete Algorithms 2, strany 53-86.

[Bai18] BAIER, Uwe (2018) "On undetected redundancy in the Burrows-Wheeler Transform", Annual Symposium on Combinatorial Pattern Matching, 3:3 – 3:15

[Bai20] BAIER, Uwe (2020) "BWT Tunneling", Dizertačná práca, Universität Ulm

[BW94] BURROWS, Michael; WHEELER, David J. (1994) "A block sorting lossless data compression algorithm", Technical Report 124, Digital Equipment Corporation

[Cla96] CLARK, David R. (1996) "Compact pat trees", Dizertačná práca, University of Waterloo

[FM00] FARRAGINA, Paolo; MANZINI, Giovanni (2000) "Opportunistic data structures with applications", Proceedings 41st Annual Symposium on Foundations of Computer Science, strany 390-398

[GFC04] GNONIEM, Mohammad; FEKETE, Jean-Daniel; CASTAGLIOLA, Philippe (2004) "A Comparison of Readability of Graphs Using Node-Link and Matrix-Based Representations", IEEE Symposium on Information Visualization, strany 17-24

[GMS17] GAGIE, Travis; MANZINI, Giovanni; SIRÉN, Jouni (2017). "Wheeler graphs: A framework for BWT-based data structures", Theoretical Computer Science, Volume 698, Strany 67-78

[Jac89] JACOBSON, Guy J. (1989) "Space-efficient static trees and graphs", 30th Annual Symposium on Foundation of Computer Science, strany 549-554

[Knu08] KNUTH, Donald E. (2008). "Umění programování 1. díl, Základní algoritmy", Computer Press, a.s, ISBN 978-80-251-2025-5. Kapitola 1.1 Algoritmy

[KS03] KÄRKKÄINEN, Juha, SANDERS, Pater (2003) “Simple Linear Work Suffix Array Construction.”, Proceedings of the 30th International Conference on Automata, Languages and Programming, strany 943-955

[MM90] MANBER, Udi; MYERS, Gene (1990) “Suffix Arrays: A New Method for Online String Searches”, First Annual ACM-SIAM Symposium on Discrete Algorithms, strany 319-327

[Nav14] NAVARRO, Gonzalo (2014) “Wavelet trees for all”, Journal of Discrete Algorithms 25, strany 2-20

ONLINE ZDROJE

[JB01] JetBrains, “The Capable & Ergonomic Java IDE by JetBrains”
https://www.jetbrains.com/idea/
Navštívené: Máj 2022

[Ko+07] KOVÁČ, Jakub; a kolektív (2007) “Gnarley Trees”
https://people.ksp.sk/~kuko/gnarley-trees/Intro.html
Navštívené: Máj 2022

[Lan21] LANGMEAD, Benjamin T. (2021) “Wheeler graphs, part 3: Definition”
https://www.youtube.com/watch?v=C17g1j2SiI8&t=2003s
Navštívené: Máj 2022

[Sew96] SEWARD, Julian (1996), “bzip2 file compressor”,
http://bzip.org
Navštívené: Máj 2022.