FM-index

Túto časť sme začali so sufixovými stromami a ukázali si, ako vieme daný text predspracovať tak, že veľa iných problémov sa výrazne zjednoduší. Okrem iného vieme vyhľadávať vzorky v čase úmernom dĺžke hľadaného reťazca. Jediná nevýhoda sufixových stromov je, že žerú strašne veľa pamäte.

To nás priviedlo k hľadaniu úspornejších štruktúr. Sufixové polia predstavovali veľký praktický pokrok, pretože zaberajú podstatne menej miesta. My sa však nechceme uspokojiť len s tým, že štruktúra je 3- až 4-krát menšia. V duchu nášho hesla „Dá sa to *ešte* lepšie?“ pátrame ďalej.

Pokračujme v našom príklade s ľudským genómom. Ľudská DNA je reťazec dĺžky približne 3 miliardy znakov nad abecedou `A`, `C`, `G`, `T`. Samotný genóm teda zaberie asi 3 GB (ak použijeme 1 bajt na znak), alebo približne 750 MB, ak využijeme zhustenú reprezentáciu s 2 bitmi na znak.

Sufixový strom, aj keby sme sa snažili byť maximálne úsporní, potrebuje približne 10–20 bajtov na znak. Pre ľudský genóm by teda zaberal 30–60 GB pamäte. Sufixové pole je na tom podstatne lepšie: potrebujeme jedno číslo na každý znak. Ak máme text do 4 miliárd znakov, vystačíme si s 32-bitovým `int`om, čo sú 4 bajty na znak. Celé pole teda zaberie asi 12 GB (plus samotný reťazec 0.75 GB). Ak by sme pridali aj LCP pole, je to ďalších 8 bajtov (2 `int`y) na znak.

A to stále hovoríme len o výslednej štruktúre, nepočítajúc dočasnú pamäť počas konštrukcie. Pri spracovaní sa tak ľahko dostaneme na hranicu veľkosti RAM. Akonáhle sa RAM zaplní, operačný systém začne *page-ovať* – odkladať časti pamäte na disk. No a čítanie či zápis na disk je rádovo pomalšie než prístup do hlavnej pamäte.

V tejto kapitole si ukážeme, ako dosiahnuť pamäťovo *ešte úspornejšie* riešenie. A dokonca niečo až neuveriteľné: vstupný text môžeme *skomprimovať* a pritom stále umožniť rýchle vyhľadávanie!

Tieto dva ciele pôsobia na prvé počutie protichodne. Ak ste si niekedy otvorili skomprimovaný súbor, videli ste, že text je úplne nečitateľný. Ak poznáte niektoré kompresné algoritmy, viete, že využívajú opakovania v texte a nahrádzajú ich rôznymi „skratkami“ či odkazmi na iné časti. To však komplikuje vyhľadávanie. Navyše sa často používajú kódovania, kde rôzne znaky zaberajú rôzne veľa bitov, takže sa posúvajú pozície a nie je jednoduché určiť, kde presne začína i-ty znak.

Pointou kompresie je totiž stlačiť vstup a vyjadriť tú istú informáciu čo najúspornejšie, zatiaľ čo pri indexovaní potrebujeme uložiť dáta štruktúrovane a pridať pomocné informácie, ktoré uľahčia vyhľadávanie. Aj na druhý pohľad to teda vyzerá ako nekompatibilné ciele.

Napriek tomu si ukážeme, že sa to dá. Predstavíme si štruktúru, ktorá už vo svojej najjednoduchšej podobe dokáže uchovať celý ľudský genóm na približne 1.5 GB pamäte. V ďalších častiach si potom ukážeme, ako jej jednotlivé komponenty reprezentovať ešte efektívnejšie.

Dátová štruktúra	**Pamäť, ktorú zaberie ľudský genóm**
pôvodný reťazec	3mld písmen, 0.75 GB (ale nevieme vyhľadávať)
sufixový strom	30–60 GB [kap. **??**]
sufixové pole	12 GB [kap. **??**]
FM-index	1.5 GB [v tejto kapitole]
. . . s kompresiou	$<$ 0.75 GB [kap. **??**]

Výsledná štruktúra, *FM-index*, je založená na tzv. *Burrows–Wheelerovej transformácii* (BWT). Ide o zaujímavú operáciu, ktorá sa využíva v rôznych kompresných algoritmoch, napríklad v známom `bzip2`. Transformácia sama osebe ešte text nekomprimuje, ale „premieša“ písmená tak, že výsledok sa dá oveľa ľahšie komprimovať. Dôležité je, že ide o *vratnú* operáciu – z transformovaného textu vieme pôvodný text jednoznačne zrekonštruovať (čo sa pri dekompresii celkom hodí).

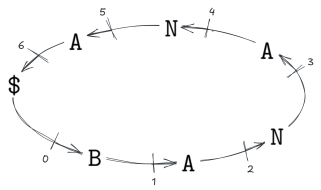
Na nasledujúcich stranách si postupne vysvetlíme:

1. čo je Burrows-Wheelerova transformácia,
2. ako sa využíva v kompresii a prečo je taká užitočná,
3. a ako dokážeme z transformovaného textu obnoviť pôvodný.

A až potom príde to najzaujímavejšie: ukážeme si, ako v takto transformovanom texte vieme vyhľadávať a ako celú dátovú štruktúru reprezentovať tak, aby bola nielen pamäťovo úsporná, ale aj rýchla.

Burrows-Wheelerova transformácia

Predstavme si, že písmená textu navlečieme ako korálky na retiazku; *i-ta rotácia* textu T je reťazec, ktorý vznikne tak, že začneme čítať od i-teho znaku, prejdeme cez ukončovací symbol `$` a pokračujeme dookola, až kým sa nevrátime na začiatok. Formálne: i-ta rotácia je jednoducho reťazec $T_{i\ldots n-1}T_{0\ldots i-1}$.



Ako príklad vezmime reťazec `BANANA$` (tak ako v predchádzajúcich kapitolách, aj tu pridávame na koniec jedinečný ukončovací znak). Všetky jeho rotácie sú:

0	B	A	N	A	N	A	$
1	A	N	A	N	A	$	B
2	N	A	N	A	$	B	A
3	A	N	A	$	B	A	N
4	N	A	$	B	A	N	A
5	A	$	B	A	N	A	N
6	$	B	A	N	A	N	A

Zotrieďme teraz všetky riadky lexikograficky (teda podľa abecedy):

6	$	B	A	N	A	N	A
5	A	$	B	A	N	A	N
3	A	N	A	$	B	A	N
1	A	N	A	N	A	$	B
0	B	A	N	A	N	A	$
4	N	A	$	B	A	N	A
2	N	A	N	A	$	B	A

Takto usporiadaná $n \times n$ matica všetkých rotácií sa nazýva *Burrows–Wheelerova matica (BWM)*.

Burrows-Wheelerovu transformáciu textu T (budeme ju označovať T^{bwt}) definujeme ako *posledný stĺpec tejto matice*. Konkrétne, pre $T = \texttt{BANANA\$}$ platí, že

$$T^{\mathrm{bwt}} = \texttt{ANNB\$AA}.$$

Samozrejme, v praxi nechceme zostavovať celú maticu – tá má kvadraticky veľa prvkov, čo by bolo pamäťovo neúnosné a výpočtovo pomalé. Otázka teda znie: ako vieme vypočítať T^{bwt} efektívne? Skúste porozmýšľať sami, so znalosťami s predchádzajúcich kapitol by to nemal byť problém.

Všimnite si, že BWT je vždy len *permutáciou* znakov pôvodného textu. Prečo?

Stačí sa pozrieť na pôvodnú maticu *pred zotriedením*. Posledný stĺpec bol `$BANANA`, čo je zjavne permutácia pôvodného reťazca. Matica je dokonca symetrická: i-ty stĺpec je rovnaký ako i-ty riadok, čo je i-ta rotácia textu. Keď riadky zotriedime, v každom stĺpci sa písmenká iba poprehadzujú a zloženie dvoch permutácií (rotácia plus triedenie) je opäť permutácia.

Použitie v kompresii

Prečo je BWT taká cool? Ukážme si to na *trochu väčšom* príklade. A keď hovorím trochu väčšom, myslím naozaj veľkom: vezmime všetky diela Williama Shakespeara. Od Sonetov, cez Antónia a Kleopatru, Hamleta, Kráľa Leara, Macbetha, Rómea a Júliu, Veselé panie z Windsoru, až po báseň Venuša a Adonis. Pre jednoduchosť text znormalizujeme: odstránime interpunkciu, necháme iba malé písmená `a-z` a medzery. Dostaneme tak necelých 5 MB textu, takmer milión slov (z toho približne 27 500 rôznych).

Predstavme si teraz tento 5-miliónový reťazec a všetky jeho rotácie, zotriedené podľa abecedy. Napríklad:

> Miliónty riadok (404 643-ta rotácia) začína takto:
> `adam so had you need i scarce can speak to thank you`
> `for myself` ... [ďalších zopár miliónov znakov] ... a končí
> `orlando i thank you most for him`

(z komédie *Ako sa vám páči*).

Slávna Hamletova replika na písmeno „T“ je zhruba 4.5-miliónta v abecednom poradí:

> Riadok 4 537 084 (931 406-ta rotácia):
> `to be or not to be ..... enter hamlet hamlet`

Prvý riadok začína `$` a pôvodným textom. Druhý riadok je 2 846 296-ta rotácia a začína sa preklepom: `␣a␣a␣farm␣house`...

Posledný riadok sa začína písmenami `zzy`, ktoré pochádzajú zo stredu slova „dizzy“:

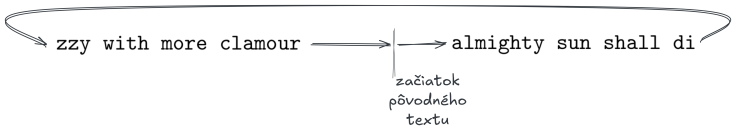
> Riadok 4 980 782 (4 340 678-ma rotácia):
> `zzy with more clamour neptunes ear ..... shall di`

z tragédie *Troilos a Kressida*.

A teraz tri hádanky:

1. Predstavte si riadky, ktoré začínajú písmenami `attle`. Aký bude posledný znak v takom riadku?
2. Aké budú posledné znaky v riadkoch, ktoré začínajú na `fe␣`?
3. Aké budú posledné znaky v riadkoch, ktoré začínajú na `a␣`?

Treba si uvedomiť, že pri rotáciách reťazca je na konci text, ktorý sa v pôvodnom tvare nachádza *tesne pred* textom na začiatku:



Takže aké písmeno môže byť pred `attle`?

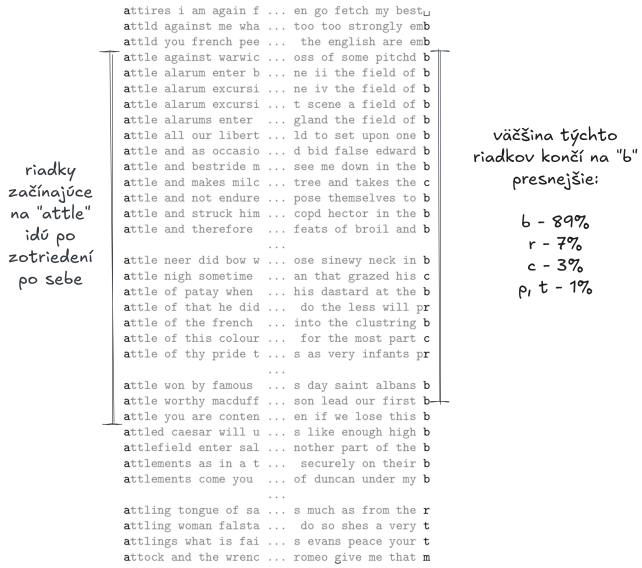
S najväčšou pravdepodobnosťou

- `b` (zo slov „battle“, „embattle“ – boj, zoradiť do boja), alebo
- `c` (cattle – dobytok).

Ak ste Shakespeare, alebo máte bohatšiu slovnú zásobu, sú aj ďalšie možnosti:

- `r` (zo slov „prattle“ – tárať, „rattle“ – hrkot/rinčanie, „berattle“ – urážať),
- `t` („tattle“ – klebetiť)
- `p` zo slova „pattle“? V skutočnosti má v hre *Henry V* kapitán Fluellen waleský prízvuk, čo Shakespeare zachytil foneticky vo vete „fought a most prave pattle here in France“.

Ak spočítame výskyty, reťazec `attle` sa v Shakespearovi nachádza 170-krát. Z toho 151-krát je predtým `b`, 12-krát `r` a 5-krát `c`; po jednom výskyte majú `p` a `t`.



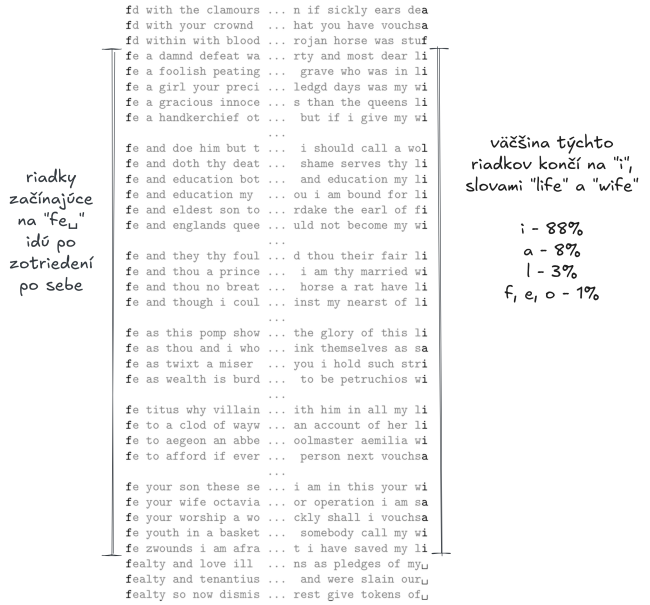
Obr. 1: Malý výsek Burrows-Wheelerovej matice zo Shakespearovského textu. V riadkoch začínajúcich na `attle` sa v poslednom stĺpci nachádza najmä `b`, zriedka `r`, `c`, `p`, `t` a nič iné.

A aké písmeno sa môže nachádzať pred `fe␣`?
Ako prvé nám asi napadne

- `i` zo slov „life“, „wife“, či „knife“... a Shakespeare k tomu ešte dodá „strife“, „Fife“ (miesto v Škótsku), „greife“ („grief“)
- `a` zo slov „safe“/„unsafe“, a Shakespeare pridá aj „vouchsafe“ – uráčiť, „chafe“ – trieť/dráždiť

V Shakespearovskej angličtine so starým pravopisom pribudne ešte veľa ďalších slov ako „deafe“, „selfe“, „halfe“, „greefe“, „roofe“, „proofe“, „staffe“, „theefe“, „wiffe“, „wolfe“, atď.

Napriek tomu, z 1842 výskytov reťazca `fe␣` je 1615-krát predchádzajúce `i` a 152-krát `a`. Iba 51-krát `l`, 10-krát `f`, 6-krát `e` a 4-krát `o`. Ďalšie možnosti sa nevyskytujú.



Obr. 2: Ďalší výsek Burrows-Wheelerovej matice Shakespearovského textu, riadky začínajúce na `fe␣`.

A posledná hádanka: aký znak bude predchádzať `a␣`?

Toto je možno trochu chyták. Najčastejšie viacpísmenové slovo končiace na „a“ je „sea“ – more (286 výskytov). Ďalej sú to citoslovcia ako „ha“ (233) alebo „yea“ (200), a, samozrejme, aj množstvo ženských mien (počet výskytov závisí najmä od mien hlavných postáv v jednotlivých hrách – nie nutne od frekvencie mien v bežnej populácii): Cleopatra (264 výskytov), Emilia (233), Desdemona (226), Helena (194), Portia (175), Isabella (159), Olivia (147), Julia (145), Viola (142), a mnoho ďalších až po Angelica, Julietta, či Polyxena (po jednom výskyte).

Jednoznačne najčastejšie slovo, ktoré končí na `a`, je však samotný neurčitý člen „a“. To znamená, že posledným znakom v riadku začínajúcom na `a␣` je *medzera*. V skutočnosti je to až 16 089 riadkov z 22 151 (72%).

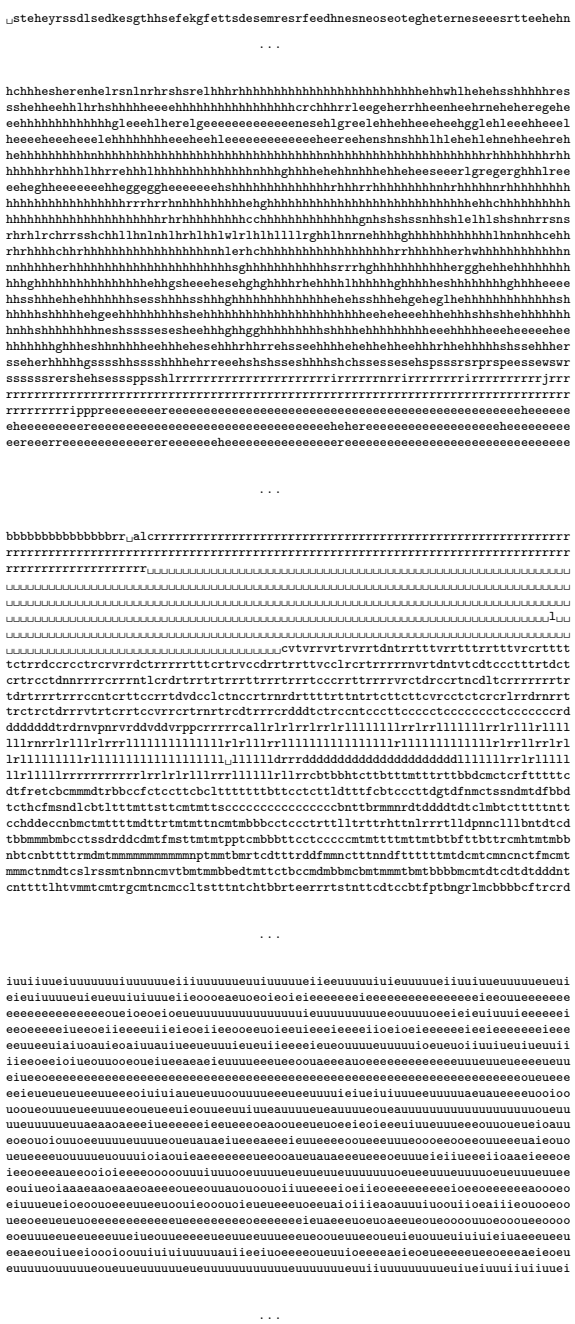
Tým, že riadky zotriedime, zabezpečíme, že rotácie začínajúce rovnakými písmenami budú *vedľa seba*.

A teraz pointa. Ako bude vyzerať BWT Shakespearovho textu? Označme ho SH^{bwt}.

Burrows-Wheelerova transformácia je definovaná ako posledný stĺpec Burrows-Wheelerovej matice, teda posledné písmeno z každého riadku (z každej rotácie),

ktoré sme utriedili podľa abecedy. Tým, že riadky zotriedime, dosiahneme, že rotácie začínajúce rovnakými písmenami, sa ocitnú *vedľa seba.*

SH^{bwt} je takmer 5-miliónový reťazec, ktorý je permutáciou pôvodného textu. Pre lepšiu predstavu, tu je zopár úryvkov z BTW Shakespearovho textu:



nnazzanuuuuuuuuueauuuuuuuuuiuuuuuauuiauauiuauiiaiiuuaiuauaaiiii

Začiatok SH^{bwt}, ␣steheyrssdlsedk…, síce vyzerá náhodne – prvé riadky totiž začínajú medzerou, takže posledný znak v riadku je posledné písmeno nejakého slova. Keď sa však dostaneme k riadkom začínajúcim na a␣, drvivá väčšina bude končiť medzerou, a v SH^{bwt} tak vznikne dlhý úsek (22 151 znakov), z ktorých 72% sú medzery. Keď sa dostaneme k riadkom začínajúcim na attle, v poslednom stĺpci budú samé b-čka (občas r, c, p, t a žiadne iné písmená). Riadky začínajúce na fe␣ zas vytvoria dlhý blok, v ktorom budú najmä i-čka, sem-tam a alebo l a úplne zriedkavo iné tri písmená, ale nič viac.

Riadky, ktoré začínajú slovom „Romeo“, budú končiť *vždy* medzerou, pretože žiadne iné slovo neobsahuje podreťazec romeo. V SH^{bwt} to vytvorí súvislý úsek 317 po sebe idúcich medzier.

Vo všeobecnosti bude SH^{bwt} obsahovať úseky s opakovanými písmenami a ešte všeobecnejšie, dlhšie úseky, v ktorých sa vyskytuje len veľmi málo rôznych znakov. Tento jav sa objavuje nielen pri Shakespearovi, ale aj pri iných textoch: v prirodzených i umelých jazykoch, pri biologických sekvenciách plných opakovaní a podobne.

Ide o to, že znak na danej pozícii sa často dá ľahko uhádnuť z kontextu, teda z toho, čo nasleduje za ním. Keď rotácie zotriedime, podobné kontexty sa dostanú vedľa seba. Výsledkom sú dlhé bloky, kde sa buď opakuje ten istý znak, alebo sa strieda veľmi malá abeceda.

Takýto text sa komprimuje oveľa ľahšie než pôvodný reťazec. Ako na to?

Najjednoduchší nápad je použiť tzv. run-length encoding (RLE). Myšlienka je priamočiara: ak sa v texte n-krát po sebe opakuje ten istý znak c (povedzme pre $n \geq 3$), tak reťazec

$$\underbrace{ccc\ldots ccc}_{n} \quad \text{zakódujeme ako dvojicu} \quad (n, c).$$

Aby sme pri dekompresii vedeli správne interpretovať bajty, musíme si niekam poznačiť, či daný bajt reprezentuje znak alebo dĺžku behu. Jednoduchý spôsob je využiť najvyšší (prvý) bit každého bajtu: ak je nastavený na 0, zvyšných 7 bitov kóduje znak, ak je nastavený na 1, zvyšných 7 bitov udáva počet opakovaní.

Už pri takomto jednoduchom postupe vieme Shakespearov text skomprimovať na takmer polovičnú veľkosť. Tento výsledok je fascinujúci najmä preto, že sme ešte ani nepoužili sofistikovanejšie techniky kompresie – iba sme využili veľmi „výhodné“ usporiadanie znakov, ktoré vyprodukovala Burrows-Wheelerova transformácia.

Ďalšia možnosť, ako využiť štruktúru, ktorú BWT odhalí, je použiť kódovanie *s premenlivou dĺžkou kódov*. Čím častejšie sa znak vyskytuje, tým kratší kód by mal mať, a naopak – zriedkavé znaky môžu mať dlhšie kódy. Týmto spôsobom sa dosiahne, že priemerný počet bitov na znak bude menší, než keby sme každému znaku priradili kód rovnakej dĺžky. Príklady takéhoto prístupu sú Huffmanovo kódovanie, aritmetické kódovanie, alebo modernejšie ANS (asymetrické numerické systémy).

Ak sa pozrieme na normalizovaný text všetkých Shakespearových diel, najčastejšie znaky sú: medzera ␣ (19.3%), `e` (9.6%), `t` (7.1%), `o` (6.6%), `a` (6.2%), `i` (5.0%). Tieto by sme teda mali kódovať čo najkratšími kódmi. Naopak, znaky ako `z`, `q`, `j`, či `x`, ktoré sa vyskytujú veľmi zriedka, môžu dostať dlhšie kódy.

Tento klasický prístup však vieme v kombinácii s BWT ešte vylepšiť. Po prvé, v transformovanom texte T^{bwt} jednotlivé úseky zodpovedajú konkrétnym *kontextom*, v ktorých sa znaky vyskytujú. Čím lepšie poznáme tento kontext, tým presnejšie dokážeme odhadnúť pravdepodobnosť jednotlivých znakov – podobne ako keď sme hádali, ktoré písmeno sa objaví pred daným podreťazcom. Po druhé, rôzne úseky transformovaného textu T^{bwt} zodpovedajú *odlišným kontextom*, a teda aj rozdelenie frekvencií znakov sa v nich môže zásadne líšiť.

Napríklad v riadkoch, ktoré začínajú medzerou, posledný znak určite nebude medzera. A hoci písmená `o`, `a`, `i` patria medzi najčastejšie v angličtine, *nepatria* medzi tie, ktoré sa najčastejšie objavujú na konci slova. Na konci slov sa omnoho častejšie vyskytujú písmená `e`, `s`, `t`, `d`, `r`.

Ako túto informáciu využiť?

Jedna možnosť je rozdeliť text na bloky a každý z nich zakódovať samostatným Huffmanovým kódom. Ak majú jednotlivé bloky rôzne frekvencie písmen, môžeme im priradiť aj odlišné kódy, čím dosiahneme lepšiu kompresiu.

Ďalšou možnosťou je použiť *adaptívne* Huffmanovo kódovanie (pozri Vitter 1987). Hlavná myšlienka je jednoduchá: pri postupnom spracúvaní textu priebežne počítame frekvencie znakov (buď od úplného začiatku, alebo len v rámci okna posledných w znakov). Podľa týchto frekvencií, ktoré sa neustále menia, si dynamicky udržiavame Huffmanov kód, ktorý sa prispôsobuje aktuálnym frekvenciám znakov.

Algoritmus `bzip2` (Seward 1996) používa ešte iný trik: na T^{bwt} aplikuje tzv. Move-to-front (MTF) transformáciu. Myšlienka je jednoduchá: počas kódovania si udržiavame zoznam všetkých znakov a i-ty znak kódujeme číslom i. Zároveň vždy keď zakódujeme nejaký znak, presunieme ho na začiatok zoznamu („move to front“).

Napríklad ak kódujeme reťazec

`D A D D D D C D C D D B D D D`

a začíname so zoznamom ␣ `A B C D E ..`, postup vyzerá takto:

- písmeno `D` je štvrté (počítame od 0), takže ho zakódujeme ako `4` a presunieme na začiatok zoznamu: `D` ␣ `A B C E ..`,
- následne `A` zakódujeme ako `2` a presunieme na začiatok: `A D` ␣ `B C E ..`,
- ďalšie `D` zakódujeme ako `1` a `D` sa dosane opäť na začiatok zoznamu: `D A` ␣ `B C E ..`,
- nasledujúce tri `D`-čka zakódujeme ako `0`, lebo `D` je na začiatku,
- atď.

Výsledná zakódovaná postupnosť je:

4 2 1 0 0 0 4 1 1 1 0 4 1 0 0.

Dôležité vlastnosti MTF transformácie sú tieto: Ak sa v texte n-krát po sebe opakuje to isté písmeno, po aplikovaní MTF sa zakóduje podľa aktuálnej pozície a následne $n-1$ núl. Podstatné je, že v rôznych častiach textu sa môžu opakovať rôzne písmená, no po MTF budú tieto behy vždy vyzerať ako postupnosť núl.

Všeobecnejšie: ak nejaký úsek obsahuje iba r rôznych znakov, ich prvé výskyty zakódujeme možno väčšími číslami, no potom sa hneď presunú na začiatok zoznamu. Zvyšok úseku potom kódujeme len číslami $0, 1, 2, \ldots, r-1$. Takže ak sa v rôznych častiach textu vyskytujú malé podmnožiny znakov, po MTF sa tam budú objavovať takmer výlučne malé čísla $0, 1, 2, \ldots$.

MTF je teda elegantný spôsob, ako všetky dlhé úseky s rôznymi malými abecedami namapovať na dlhé úseky tvorené malými číslami $0, 1, 2, \ldots$. Takto spracovaný text sa následne výborne hodí na Huffmanovo kódovanie.

V skutočnosti `bzip2` pozostáva zo štyroch krokov:

1. Burrows-Wheelerova transformácia usporiada text tak, že vzniknú dlhé úseky s malou abecedou,
2. Move-to-front transformácia tieto úseky ďalej prevedie na postupnosti prevažne veľmi malých čísel, najčastejšie núl,
3. Run-length encoding skomprimuje jednopísmenové behy do tvaru (počet, znak), a nakoniec
4. Huffmanovo kódovanie zakóduje výslednú postupnosť pomocou prefixového kódu, pričom častejšie symboly dostanú kratšie kódy.

Inverzná transformácia

Začnime jednoduchým príkladom: vezmime slovo „bedač“, kde sú všetky písmená rôzne. Ak vytvoríme všetky rotácie reťazca `BEDAC$` a zotriedime ich lexikograficky, dostaneme maticu, ktorej posledný stĺpec je $T^{\mathrm{bwt}} =$ `CD$AEB`.

5	$	B	E	D	A	**C**
3	A	C	$	B	E	**D**
0	B	E	D	A	C	**$**
4	C	$	B	E	D	**A**
2	D	A	C	$	B	**E**
1	E	D	A	C	$	**B**

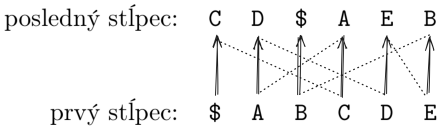
Otázka znie: ako sa vieme od `CD$AEB` dostať naspäť k pôvodnému textu? Nápoveda: predstavme si Burrows-Wheelerovu maticu. T^{bwt} je jej posledný stĺpec. Okrem neho existuje ešte jeden špeciálny stĺpec, ktorý dokážeme z T^{bwt} veľmi ľahko zrekonštruovať. Ktorý?

Ak hovoríte, že ten špeciálny stĺpec je prvý,máte pravdu. Ako sme spomínali vyššie, každý stĺpec Burrows-Wheelerovej matice je permutáciou znakov pôvodného textu. Špeciálne prvý stĺpec vznikne tak, že sa znaky utriedia podľa abecedy, pretože celé riadky triedime lexikograficky.

To znamená, že poznáme

posledný stĺpec:	C	D	$	A	E	B
prvý stĺpec:	$	A	B	C	D	E

Čo ďalej? Uvedomme si, že znak v poslednom stĺpci je vždy ten, ktorý sa v príslušnom riadku nachádza *hneď pred* znakom v prvom stĺpci.



Ak pôjdeme odzadu, vieme, že posledný znak pôvodného textu je `$`. Podľa tabuľky sa pred ním nachádza `C`. Pred `C`-čkom je zase `A`, a tak ďalej. Takto postupujeme, až kým neprídeme späť na začiatok textu – pred `B` je znovu `$`, čo nám signalizuje, že máme končiť.

Jediné, čo si potrebujeme rozmyslieť, je: ako pre daný znak v poslednom stĺpci nájdeme jeho pozíciu v prvom stĺpci. (Mimochodom, proces by sa dal robiť aj opačne a dekódovať pôvodný reťazec od začiatku do konca, ale mapovanie prvého stĺpca na posledný by sa robilo trochu komplikovanejšie.)

A čo v prípade opakujúcich sa znakov?

Vezmime si úvodný príklad, reťazec `BANANA$`:

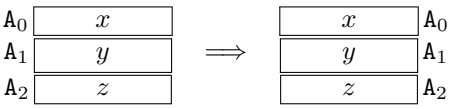
$_0$	B	A	N	A	N	A$_0$
A$_0$	$	B	A	N	A	N$_0$
A$_1$	N	A	$	B	A	N$_1$
A$_2$	N	A	N	A	$	B$_0$
B$_0$	A	N	A	N	A	$_0$
N$_0$	A	$	B	A	N	A$_1$
N$_1$	A	N	A	$	B	A$_2$

Tvrdím, že ak si jednotlivé písmená označíme poradovými číslami, aby sme ich vedeli od seba odlíšiť, tak všetky výskyty rovnakého písmena v prvom stĺpci *budú zoradené v rovnakom poradí* ako tie isté písmená v poslednom stĺpci. Napríklad všetky `A`-čka v prvom stĺpci budú zoradené v tom istom poradí ako `A`-čka v poslednom stĺpci.

Prečo je to tak?

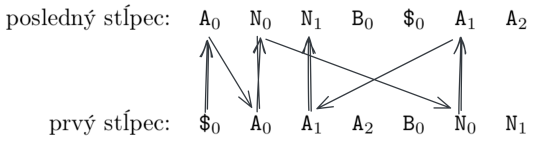
Zamerajme sa iba na riadky Burrows–Wheelerovej matice, ktoré začínajú písmenom `A`. Keď tieto riadky zotriedime, dostanú sa vedľa seba. O ich vzájomnom poradí však rozhoduje zvyšok riadku, teda text, ktorý nasleduje *po* úvodnom `A`-čku. Ak máme riadky Ax, Ay, Az, ich poradie bude dané poradím reťazcov x, y, z.

Čo sa stane s riadkami, ktoré končia na A? Tie presne zodpovedajú predchádzajúcej skupine riadkov, len posunutej o jednu pozíciu doľava: z Ax, Ay, Az sa stanú xA, yA, zA. Tieto riadky síce v utriedenej matici nemusia stáť vedľa seba, no ich vzájomné poradie je opäť určené poradím x, y, z.



Výsledkom je, že poradie všetkých A-čiek v prvom stĺpci presne zodpovedá poradiu všetkých A-čiek v poslednom stĺpci. A rovnako to platí aj pre všetky ostatné písmená.

Vďaka tomu presne vieme, ktorý znak v poslednom stĺpci zodpovedá ktorému znaku v prvom stĺpci, a môžeme použiť ten istý postup ako predtým: postupne sa vracať po znakoch smerom späť a zrekonštruovať pôvodný text.



FM-index

Keď už vieme, ako funguje Burrows-Wheelerova transformácia a jej použitie pri kompresii, poďme sa pozrieť na to, ako sa dá v transformovanom texte T^{bwt} vyhľadávať.

Vyhľadávanie

Ukážme si postup na konkrétnom príklade: Máme text

$$T = \texttt{ATAGACCGCCATTACATAGATGAGTATAGAGACT\$},$$

spočítame si jeho Burrows-Wheelerovu transformáciu

$$T^{\text{bwt}} = \texttt{TTGGTGTTG\$TCGCACGACAAAATACACTAAAGAA}.$$

Celá Burrows-Wheelerova matica je na obr. 3 – pripomeňme, že ju tu uvádzame len na ilustráciu – v praxi ju nikdy celú nezostrome. Zaujíma nás z nej len prvý stĺpec F a posledný stĺpec $L = T^{\text{bwt}}$, zvyšné znaky matice nemáme k dispozícii, preto sú znázornené šedou.

V skutočnosti aj prvý stĺpec

$$F = \texttt{\$AAAAAAAAAAAAAACCCCCCCGGGGGGGTTTTTTTT}$$

nemáme uložený ako n znakov – je to zbytočne veľa pamäte. Namiesto toho si stačí pamätať počty znakov: 1 ukončovací znak, 13 A-čok, 6 C-čok, 7 G-čok a 8

T-čok. Ako uvidíme ochvíľu, ešte lepšie uložiť si tabuľku začiatkov:

znaky	$	A	C	G	T	koniec
začínajú od riadku	0	1	14	20	27	35

Na obr. 3 sme si tiež očíslovali výskyty jednotlivých znaky, podobne ako pri inverznej BWT, aby sme vedeli, ktorý znak v poslednom stĺpci zodpovedá ktorému znaku v prvom stĺpci. Napríklad `A 3` znamená, že ide o tretie `A`-čko v danom stĺpci (číslujeme od 0), resp., že pred týmto `A` sú v texte ešte ďalšie tri.

Predstavme si, že chceme vyhľadať všetky výskyty reťazca `TAG`. Postup je tiež znázornený na obr. 3. Budeme postupovať odzadu a hľadať čoraz dlhšie sufixy hľadaného reťazca. Najskôr nájdeme všetky riadky, ktoré začínajú posledným písmenom `G`, potom nájdeme riadky, ktoré začínajú na `AG`, až nakoniec nájdeme všetky riadky, ktoré začínajú na `TAG`.

Jeden krok vyhľadávacieho algoritmu. Predstavme si, že už sme našli interval riadkov, ktoré začínajú na podreťazec P (vyšrafovaná oblasť v strede). To znamená, že riadky nad tým začínajú reťazcom menším ako P a riadky pod tým začínajú reťazcom väčším ako P. Predpokladajme ďalej, že chceme nájsť riadky, ktoré začínajú na `c`p, kde `c` je jeden znak.

Pozrime sa na riadky, ktoré *končia na* `c`. Riadky končiace nultým a prvým `c` sú *nad* naším intervalom. To znamená, že v rotáciach za $\mathtt{c}_0$ a $\mathtt{c}_1$ nasleduje reťazec menší ako p. Podobne riadky končiace piatym a šiestym `c` sú *pod* naším intervalom, takže za $\mathtt{c}_5$ a $\mathtt{c}_6$ nasleduje reťazec väčší ako p.

Riadky končiace druhým, tretím a štvrtým `c` sa nachádzajú v našom intervale, tzn. začínajú na p. To znamená, že v rotáciách za $\mathtt{c}_2$, $\mathtt{c}_3$ a $\mathtt{c}_4$ nasleduje p a interval od druhého po štvrté `c` je náš hľadaný interval.

Ranky

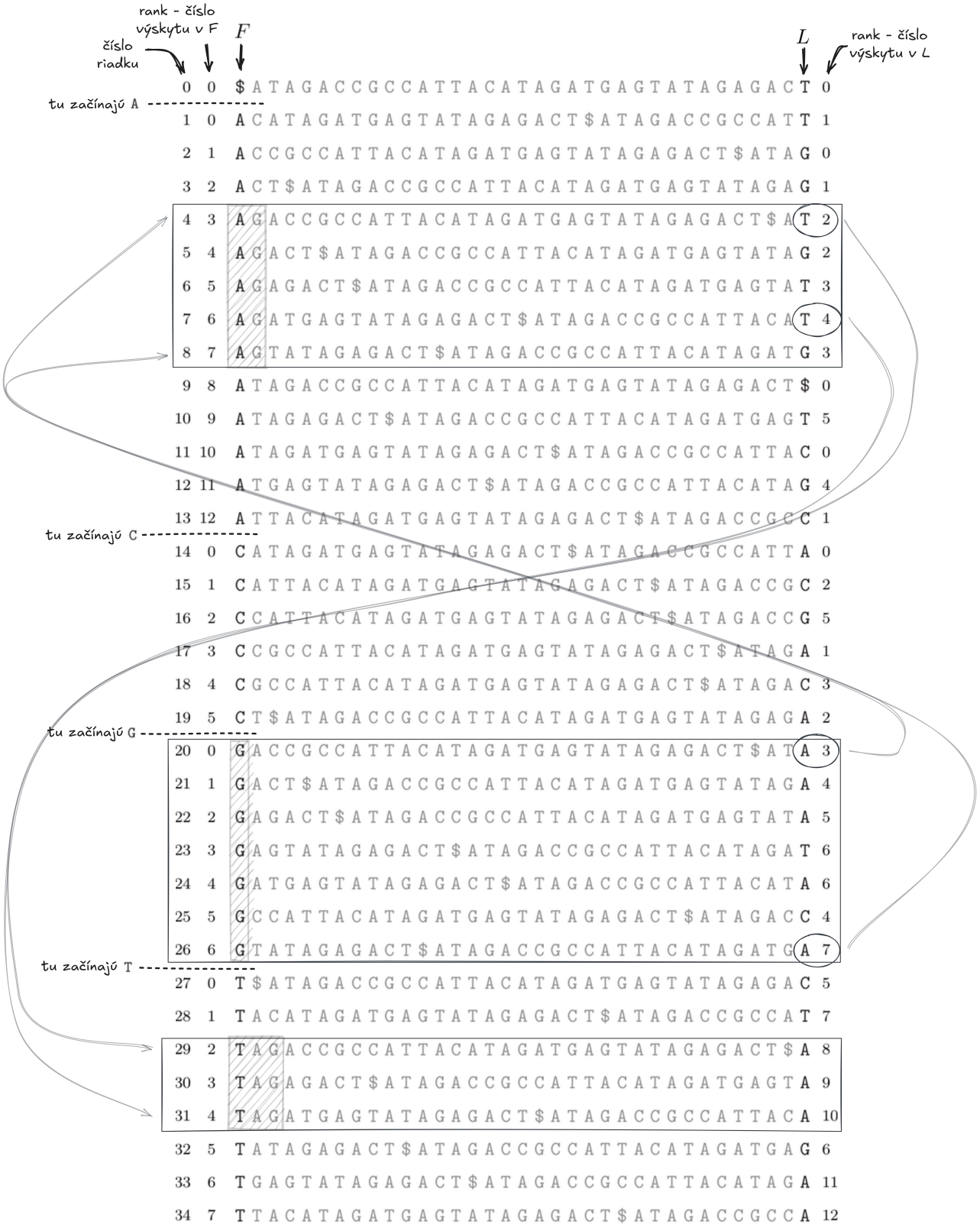
Pozície v texte

Výsledná štruktúra

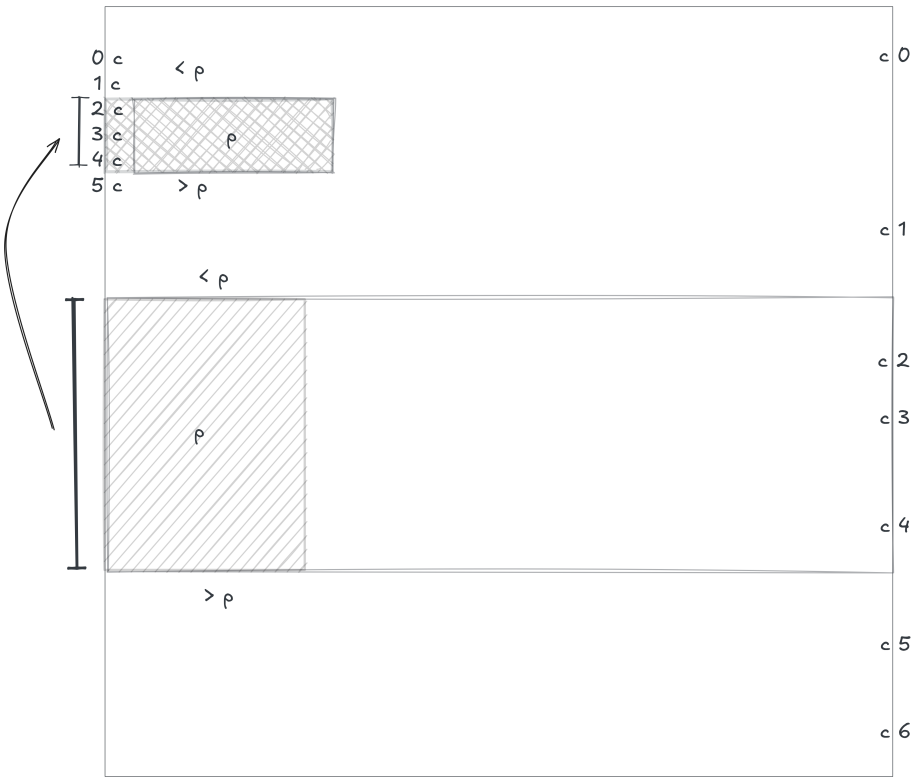
Na prípravu jedného FM-indexu budeme potrebovať: vstupný text T. Spočítame sufixové pole $SA(T)$. Zo sufixového poľa odvodíme posledný stĺpec Burrows-Wheelerovej matice L, ktorý si uložíme. Zo samotného sufixového poľa si necháme iba malú podmnožinu a zvyšok zahodíme. Spočítame frekvencie jednotlivých znakov a pole čiastočných súčtov, čím dostaneme tabuľku F pre prvý stĺpec. Následne vybudujeme štruktúru pre $\mathrm{rank}_{\mathtt{c}}(L, i)$, vďaka ktorej vieme rýchlo povedať pre ľubovoľný znak `c` a pozíciu i, koľko `c`-čiek sa nachádza v L pred i.

Výsledný FM-index sa skladá zo štyroch častí:

- $L = T^{\mathrm{bwt}}$, posledného stĺpca Burrows-Wheelerovej matice,
- F, prvého stĺpca,



Obr. 3: BWT textu $T =$ `ATAGACCGCCATTACATAGATGAGTATAGAGACT$`.
Prvý stĺpec: $F =$ `$AAAAAAAAAAAAACCCCCCGGGGGGGTTTTTTTT`,
Posledný stĺpec: $L =$ `TTGGTGTTG$TCGCACGACAAAATACACTAAAGAA`.



Obr. 4: Jeden krok vyhľadávacieho algoritmu. Predstavme si, že už sme našli interval riadkov, ktoré začínajú na podreťazec P (vyšrafovaná oblasť v strede) a chceme nájsť riadky, ktoré začínajú na cp, kde c je jeden znak. Potrebujeme zistiť, *ktoré* c v poslednom stĺpci sa nachádzajú v našom intervale. V tomto príklade sú to $\mathtt{c}_2$ až $\mathtt{c}_4$. Preto interval od druhého po štvrté c je hľadaný interval, ktorý začína na cp.

- štruktúry pre $\text{rank}_{\mathtt{c}}(L, i)$,
- podmnožiny sufixového poľa $SA(T)$.

Koľko miesta to zaberie?

- $L = T^{\text{bwt}}$ – n znakov, rovnako ako vstupný text (zatiaľ – avšak už sme videli, že L je reťazec, ktorý sa dá dobre komprimovať)
- F – jedno číslo pre každý znak abecedy; abeceda je väčšinou malá (napr. menej ako 256 znakov), takže toto je väčšinou úplne zanedbateľná hodnota,
- $\text{rank}_{\mathtt{c}}(L, i)$ – zatiaľ sme si ukázali riešenie s pamäťou $n|\Sigma|/b$, kde b je konštanta, v nasledujúcej kapitole si ukážeme lepšie riešenia,
- podmnožina sufixového poľa – $2n/s$ čísel, kde s je nejaká konštanta.

Ukážme si to na príklade ľudskej DNA z úvodu. Máme teda 4-písmenovú abecedu a zvoľme $s = 64$ a $b = 128$. Potom

- L zaberie 750 MB (ako pôvodný reťazec),
- F sú 4 čísla (zanedbateľných 16 bajtov).

3 miliardy 4-bajtových čísiel zaberie 12 GB, tzn.

- štruktúra pre rank zaberie $12\,\text{GB} \times 4/128 = 375\,\text{MB}$,
- 1/64 sufixového poľa zaberie $12\,\text{GB} \times 2/64 = 375\,\text{MB}$

Podtrženo a sečteno: spolu len 1.5 GB, čiže dvakrát veľkosť pôvodného reťazca. Všimnite si, že na rozdiel od sufixových stromov a polí si nepotrebujeme pamätať pôvodný reťazec T (vieme ho rekonštruovať z $L = T^{\text{bwt}}$).

Z 30–60 GB sufixového stromu, cez 12 GB sufixové pole sme sa teda dostali na 1.5 GB FM-index. Dosiahli sme 20–40× menej pamäte a to sme ešte nekomprimovali L a použili sme len veľmi jednoduché riešenie pre rank. S kompresiou a ďalšími vylepšeniami vieme dosiahnuť dátovú štruktúru, ktorá zaberá len 30–50% pamäte pôvodného reťazca a navyše v nej vieme rýchlo vyhľadávať. Ako na to, si postupne ukážeme v ďalších kapitolách.

Referencie

Seward, Julian (1996). *bzip2 and libbzip2*. URL: `https://sourceware.org/bzip2/manual/manual.html`.

Vitter, Jeffrey Scott (1987). „Design and analysis of dynamic Huffman codes“. In: *Journal of the ACM (JACM)* 34.4, s. 825–845.