BWT a FM-index

©kuko 20.4.2018

1 Úvod

- sufixový strom (ST) – aj keď si dáme pozor a snažíme sa použiť čo najmenej pamäte, vyžaduje asi 10–20B/znak
- sufixové pole (SA) potrebuje 1 int/znak; ak máme text do 4miliárd znakov, môžeme použiť 32bitový int, čo je 4B/znak (ak nepoužijeme LCP pole – ďalších 12B/znak) + samotný text
- vezmime si napríklad ľudský genóm, čo je reťazec asi 3miliardy znakov nad abecedou `A`, `C`, `G`, `T`
- samotný string teda zaberie asi 3GB (ak použijeme 1B/znak), alebo 750MB, ak použijeme zhustenú reprezentáciu a 2bity/znak
- sufixový strom bude zaberať 30–60GB a sufixové pole asi 12GB (+samotný reťazec 0.75GB)
- a to je len pamäť výslednej štruktúry, kde nepočítame pamäť použitú dočasne počas konštrukcie
- pri spracovaní väčších vstupov nás teda bude limitovať veľkosť RAM
- v tejto prednáške si ukážeme, ako dosiahnuť pamäťovo ešte úspornejšie riešenie
- dokonca si ukážeme, že vstupný text môžeme skomprimovať tak, že pritom stále umožníme rýchle vyhľadávanie
- výsledná štruktúra, FM-index, je založená na Burrows-Wheelerovej transformácii a rotáciách reťazca T

2 Burrows-Wheelerova transformácia

- uvažujme všetky rotácie stringu T a zotrieďme ich lexikograficky
- t.j. uvažujme BW $n \times n$ maticu M
- T^{bwt} je reťazec pozostávajúci z posledného písmena z každého riadku – posledný stĺpec matice
- triediť všetky rotácie je podobné ako triediť všetky sufixy, preto neprekvapí, že medzi T^{bwt} a sufixovým poľom existuje jednoduchý vzťah:
  - $T^{\text{bwt}}[i] = T[SA[i] - 1]$ (kde $T[-1] = \$$)
- z toho vyplýva, že T^{bwt} vieme vypočítať v lineárnom čase (existujú aj priame konštrukcie bez SA)

2.1 Použitie v kompresii

- príklad: ak v anglickom texte objavíme substring „`ATTLE` “, aké bolo predchádzajúce písmeno?
  - s najväššou pravdepodobnousťou `B` (zo slov battle, embattle) alebo `C` (cattle – dobytok)
  - ale je zopár ďalších možností: `R` (prattle – tárať), `T` (tattle – klebetiť)
- ak zotriedime rotácie T, všetky rotácie začínajúce sa na „`ATTLE` “ budú po sebe, takže v poslednom stĺpci bude v tejto oblasti veľa `B`čok a `C`čok a zopár `R`, `T`; písmená s rovnakým kontextom sa pomocou BWT dostanú k sebe
- vo všeobecnosti bude T^{bwt} obsahovať úseky s opakovanými písmenami a všeobecnejšie dlhšie úseky, kde sa vyskytuje len zopár písmen – takýto reťazec sa oveľa ľahšie komprimuje
- napríklad algoritmus `bzip2` sa skladá z viacerých krokov:
  - 1. BWT – získame opakované písmená a dlhé úseky s malým počtom písmen
  - 2. MTF (pri kódovaní si udržiavame zoznam písmen; i-te písmeno nahradíme číslom i a zároveň ho presunieme na začiatok zoznamu – move-to-front) – úsek s rovnakými písmenami sa tak premení na úsek samých núl; úsek s malým počtom písmen sa premení na úsek s malými číslami
  - 3. RLE (run-length encoding) – k písmen c po sebe nahradíme zakódovaním (k, c)
  - 4. nakoniec použijeme Huffmanov kód na zakódovanie jednotlivých symbolov

2.2 Reverzná transformácia $T^{\text{bwt}} \to T$

- predpokladajme na chvíľu, že všetky znaky v T sú rôzne – triedenie rotácií je potom jednoduché, lebo stačí porovnať prvé písmeno
- ak zotriedime znaky $L = T^{\text{bwt}}$, dostaneme prvý stĺpec F
- uvedomme si, že ak máme prvý aj posledný stĺpec, vieme ľahko zrekonštruovať T, totiž L_i je písmeno, ktoré sa v T nachádza pred F_i
- stačí teda začať od konca s \$, nájsť riadok, kde $F_i = \$$; tak zistíme posledný znak $c = L_i$; opäť nájdeme riadok, kde $F_j = c$ a odčítame predchádzajúci znak $c = L_j$ – takto pokračujeme až na začiatok reťazca
- ostáva domyslieť: ak sa v T znak c opakuje, ktorá pozícia v L prislúcha ku ktorej pozícií v F?
- nie je ťažké nahliadnuť, že i-ty výskyt c v L zodpovedá i-temu výskytu c v F:
  - ak cx je pred cy, potom $x < y$ a tým pádom xc je pred yc
  - inými slovami, všetky riadky začínajúce sa na c sú utriedené podľa zvyšku reťazca – a v tom istom poradí sú riadky, ktoré majú c na konci
- na efektívnu implementáciu reverznej transformácie potrebujeme pre L_i rýchlo nájsť zodpovedajúci riadok F_j
- toto nazývame LF-zobrazenie a dá sa jednoducho vypočítať už počas triedenia L na F – avšak my budeme navyše chcieť LF-zobrazenie reprezentovať s malou pamäťou
- na to stačí pre každý znak c a pre ľubovoľné i vedieť zistiť počet c-čok v $L[0..i]$ (toto je klasická úloha $\text{rank}_c(L, i)$) a kde sa začína úsek c-čok v F

3 FM-index

3.1 Vyhľadávanie

- v predošlej sekcií sme popísali použitie BWT v kompresii a inverznú transformáciu; ako však v T (resp. T^{bwt}) vyhľadávať?
- keďže BWT úzko súvisí s SA, mohli by sme skúsiť binárne vyhľadávanie rovnako, ako v SA; avšak vo výslednom FM-indexe si nebudeme pamätať priamo T a zistenie j-teho znaku v i-tom riadku bude pomalšie ako v SA – tým pádom bude celé binárne vyhľadávanie pomalšie
- existuje však lepší spôsob, pomocou LF-zobrazenia
- vyhľadávanie $P = p_0 \dots p_{m-1}$ pôjde odzadu, od posledného znaku; postupne budeme vyhľadávať $P_{i\dots} = p_i p_{i+1} \dots p_{m-1}$ pre $i = m-1, \dots, 0$
- presnejšie: keďže riadky pomyselnej matice M sú zotriedené lexikograficky, riadky začínajúce sa slovom w tvoria jeden súvislý úsek; nech teda $[s_i, e_i)$ je interval riadkov začínajúcich sa sufixom $P_{i\dots}$
- ak poznáme $[s_{i+1}, e_{i+1})$, ako zistíme $[s_i, e_i)$?
- poznáme úsek riadkov, ktoré začínajú na $P_{i+1\dots}$; niektoré z týchto riadkov končia na p_i – tieto riadky zodpovedajú výskytom $P_{i+1\dots}$, pred ktorými sa nachádza p_i
- hoci riadky, ktoré začínajú na $P_{i+1\dots}$ a končia na p_i nemusia tvoriť jeden súvislý úsek, ich rotácie o 1 vľavo sú riadky začínajúce sa na $P_{i\dots} = p_i P_{i+1\dots}$ – tie sa nachádzajú v M v rovnakom poradí a tvoria súvislý úsek riadkov – ak sa pozrieme na všetky riadky začínajúce sa na p_i, budú v rovnakom poradí ako všetky riadky končiace sa na p_i
- vieme, že $[s_i, e_i)$ bude podinterval riadkov, ktoré začínajú na p_i ($[F[p_i], F[p_i + 1])$); riadky $[F[p_i], s_i)$ sú riadky, ktoré začínajú na $p_i x$, kde $x < P_{i+1\dots}$, riadky $[s_i, e_i)$ začínajú na $P_{i\dots}$ (tento úsek hľadáme), a riadky $[e_i, F[p_i + 1])$ sú riadky, ktoré začínajú na $p_i y$, kde $y > P_{i+1\dots}$
- stačí teda zistiť $\text{rank}_{p_i}(L, s_{i+1})$ – koľkokrát sa znak p_i vyskytuje pred s_{i+1}, t.j. počet výskytov $p_i x$, kde $x < P_{i+1\dots}$, a $\text{rank}_{p_i}(L, e_{i+1})$ – počet výskytov $p_i x$ pre $x \leq P_{i+1\dots}$ ($|x| = m - i - 1$)
- $[s_{m-1}, e_{m-1}) \leftarrow [F[p_{m-1}], F[p_{m-1} + 1])$
- $[s_i, e_i) \leftarrow [F[p_i] + \text{rank}_{p_i}(L, s_{i+1}), F[p_i] + \text{rank}_{p_i}(L, e_{i+1}))$
- na konci dostaneme interval $[s_0, e_0)$ – riadky začínajúce na P

- ostáva domyslieť „detaily“:
- ako pre daný riadok M zistiť pozíciu v T
- presne túto informáciu sme si pamätali v sufixovom poli – $SA[k]$ = pozícia k-teho najmenšieho sufixu/rotácie v T – my si však nechceme pamätať celé sufixové pole (chceme menšiu pamäť)
- riešenie: budeme si pamätať podmnožinu SA – napríklad iba hodnoty $SA[k]$ deliteľné s
- ak chceme zistiť hodnotu $SA[k]$, ktorú nemáme uloženú, budeme sa pomocou LF-zobrazenia posúvať na predchádzajúce rotácie, až kým po $< s$ krokoch nenarazíme na hodnotu SA, ktorú sme si uložili
- ako reprezentujeme $L = T^{\text{bwt}}$ tak, aby sme vedeli rýchlo zistiť $\text{rank}_c(L, i)$?
- toto je téma na celú ďalšiu prednášku – jednoduchý spôsob je predpočítať a uložiť si hodnotu $\text{rank}_c(L, i)$ pre každý znak c, ale iba pre každú b-tu pozíciu (trade-off medzi časom a pamäťou pre malé abecedy a dostatočne veľké b to nebude veľa pamäte)
- z T^{bwt} vieme vypočítať pôvodné T v $O(n)$; vedeli by sme však zistiť znak T_i, resp. podreťazec $T_{i\ldots j}$ aj bez toho, aby sme si pamätali T? a bez toho, aby sme robili celú inverznú BWT?
- áno: potrebujeme len vedieť rýchlo zistiť pre pozíciu v T zodpovedajúci riadok v M – resp. riadok v SA
- riešenie: hodnoty (podmnožiny) SA si uložíme tak, aby sme vedeli rýchlo odpovedať aj $SA[i] =?$ aj $SA[?] = i$
- keď chceme extrahovať $T_{i\ldots j}$, „zaokrúhlime“ najskôr j nahor na najbližšiu hodnotu deliteľnú s, zistíme príslušný riadok M a pomocou LF-zobrazenia postupne dekódujeme T až po i-tu pozíciu

- výsledná štruktúra sa bude skladať z:
  - F (prvý stĺpec M) – $|\Sigma|$ intov
  - $L = T^{\text{bwt}}$ (posledný stĺpec M) – n znakov (zatiaľ – ale dá sa skomprimovať)
  - štruktúra pre $\text{rank}_c(L, i)$ – $n|\Sigma|/b$ intov (zatiaľ – ukážeme si lepšie riešenia)
  - podmnožina SA – $2n/s$ intov
- koľko pamäti to zaberie? pre náš príklad s DNA, kde $|\Sigma| = 4$, $s = 64$ a $b = 128$: $|F|$=16B, $|L|$=750MB, $|SA|$=12GB×2/64=375MB, $|\,\text{rank}\,|$=12GB×4/128=375MB – spolu len 1.5GB, čiže 2× veľkosť pôvodného reťazca
- všimnite si, že na rozdiel od ST a SA si nepotrebujeme pamätať pôvodný reťazec T (vieme ho rekonštruovať z $L = T^{\text{bwt}}$)
- z 30–60GB sufixového stromu, cez 12GB sufixové pole sme sa teda dostali na 1.5GB FM-index (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 DŠ, ktorá zaberá 30–50% pamäte pôvodného reťazca a navyše v nej vieme rýchlo vyhľadávať
- s kompresiou a ďalšími vylepšeniami vieme dosiahnuť DŠ, ktorá zaberá 30–50% pamäte pôvodného reťazca a navyše v nej vieme rýchlo vyhľadávať