Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Efektívna reprezentácia množiny krátkych reťazcov

Bakalárska práca

2016
Jaroslav Petrucha

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Efektívna reprezentácia množiny krátkych reťazcov

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	Mgr. Tomáš Vinař, PhD.

Bratislava, 2016
Jaroslav Petrucha

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Jaroslav Petrucha
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: Efektívna reprezentácia množiny krátkych reťazcov
Efficient Representation of Short String Collections

Cieľ: Pri sekvenovaní DNA vzniká množstvo krátkych reťazcov, ktoré je potrebné reprezentovať v rámci efektívnych dátových štruktúr. Jednou z možných reprezentácií je nadslovo, ktoré obsahuje všetky podreťazce dĺžky k týchto reťazcov. Úlohou je navrhnúť praktické algoritmy na vytvorenie takéhoto nadslova a vyhodnotiť úspešnosť ich aplikácie na sekvenovacie dáta druhej generácie.

Vedúci: Mgr. Tomáš Vinař, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 28.10.2015

Dátum schválenia: 28.10.2015 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie:

Veľká vďaka patrí školiteľovi, Tomášovi Vinařovi, za všetky dobré rady, podporu a občasné dokopávanie. Taktiež sa chcem poďakovať Mišofovi, ktorý mi poskytoval rýchle informácie a odkazy na známe aj trošku menej známe algoritmy na riešenie rôznych problémov. Veľká vďaka patrí aj Šandyne, ktorá mi poskytla pomoc pri ovládaní Inkscapu a tvorbe obrázkov. V neposlednom rade ďakujem rezidentom T2 za podporu a pocit, že v tom nie som sám.

Abstrakt

Pri hľadaní sekvencie DNA nejakého organizmu začíname vo väčšine prípadov s krátkymi podreťazcami tejto sekvencie, ktoré nazývame čítania. V tejto práci predstavujeme novú štruktúru na indexáciu týchto čítaní s nízkymi pamäťovými nárokmi za pomoci špeciálneho nadslova. Táto štruktúra vie odpovedať na otázky, koľké, resp. ktoré čítania obsahujú ako podreťazec daný reťazec dĺžky k. Hodnota k je pre inštanciu štruktúry nemenná a musí byť daná vopred.

Kľúčové slová: čítania, nadslovo, index čítaní, DeBruijnov graf

Abstract

When discovering the DNA sequence for an organism, we often begin with a collection of short substrings of the sequence called reads. In this thesis, we present a new structure for read indexation with low memory footprint using a special kind of superstring. This structure can answer queries about the number or identifiers of reads containing a given string of length k as substring. The value of k is constant for an instance of the structure and must be given beforehand.

Keywords: reads, superstring, read indexing, DeBruijn graph

Obsah

Zoznam obrázkov

Úvod

S vývojom rozmanitých biologických technológií sa začalo pri skúmaní organizmov objavovať nevídané množstvo dát. Kvôli ich enormnému množstvu, špecifickým vlastnostiam a nárokom na spracovanie vznikol odbor bioinformatika.

Podstatná skupina dát sa zaoberá DNA organizmov. V tejto práci sa budeme venovať čítaniam, krátkym podreťazcom DNA, ktoré vieme presne zisťovať počas hľadania celej DNA nejakého organizmu. Aby sme vedeli rýchlo zisťovať mnohé zaujímavé skutočnosti z týchto reťazcov, je nutné mať štruktúru, ktorá dokáže rýchlo odpovedať na rôzne otázky o nich. Pre mnohé účely stačí, aby sme vedeli rýchlo odpovedať na otázky o nejakých kratších podreťazcoch týchto čítaní.

Našim prístupom k tomuto problému bude vytvorenie vhodného nadslova čítaní, ktoré bude obsahovať všetky ich podreťazce dĺžky k (pre vopred dané k). Pomocou niekoľkých podporných štruktúr potom budeme vedieť odpovedať na otázky o podreťazcoch dlhých presne k.

V prvej kapitole sa oboznámime so základnými biologickými znalosťami a pojmami, z ktorých vychádza táto práca. Podrobnejšie si popíšeme motiváciu a ciele práce, spomenieme si predošlé riešenia k tomuto problému a zadefinujeme zopár pojmov, ktoré budeme neskôr používať. V druhej kapitole sa budeme venovať hľadaniu spomenutého nadslova, spomenieme si zopár teoretických výsledkov o tomto probléme a popíšeme, ako sa dá toto hľadanie implementovať. V tretej kapitole sa pozrieme, ako a s pomocou akých štruktúr budeme vedieť vďaka nájdenému nadslovu odpovedať na rôzne otázky. V štvrtej kapitole sa pozrieme, aké sú nároky na pamäť a čas našej implementácie a porovnáme si ju s inou prácou v oblasti.

Súčasťou práce je aj ukážková implementácia v jazyku C++.

Kapitola 1

Základné pojmy, motivácia a súvisiace práce

V tejto kapitole čiatateľa oboznámime so základnými biologickými a informatickými pojmami, s ktorými sa stretneme v tejto práci, s cieľom práce a s motiváciou pre riešenie tohto problému. Taktiež si spomenieme niekoľko prác súvisiacich s problematikou.

1.1 Biologické pojmy a motivácia

Genetická informácia všetkých známych bunkových organizmov je uložená v deoxyribonukleovej kyseline, skrátene nazývanej *DNA*. Ide o zložitú molekulu pozostávajúcu z dvoch chemicky naviazaných polymérových vlákien stočených do špirálovej štruktúry. Základné stavebné prvky, z ktorých sa tieto reťazce skladajú, sa nazývajú nukleotidy. Každý nukleotid pozostáva z fosfátového zvyšku, deoxyribózy a dusíkatej bázy. Nukleotidy sa od seba líšia v dusíkatej báze, ktorá môže byť štyroch typov: adenín, guanín, tymín, cytozín.

Vlákna jednej molekuly DNA sú medzi sebou prepojené práve cez dusíkaté bázy. S tým súvisí taktiež tzv. komplementárnosť týchto vlákien. Keď v jednom vlákne máme adenínový nukleotid, spojený je s tymínovým nukleotidom v druhom vlákne a naopak. Podobne sa guanínový nukleotid spája s cytozínovým.

Celá genetická informácia v jednej molekule DNA je tak kódovaná v podstate iba usporiadaním nukleotidov, pre krátkosť sa toto usporiadanie zapisuje reťazcom pozostávajúcim zo znakov `A`, `G`, `T` a `C`, pričom každý znak zodpovedá príslušnej dusíkatej báze. Kvôli chemickej štruktúre vlákna DNA záleží aj na tom, v ktorom smere tento

reťazec zapisujeme, čiže napr. reťazec `GGA` je zásadne odlišný od reťazca `AGG`. Smery komplementárnych vlákien sú opačné.

Bežný organizmus obsahuje viacero molekúl DNA, ktoré sú poskladané do štruktúr nazývaných chromozómy. Genetickú informáciu, ktorou jedinec disponuje, nazývame súhrnne menom *genóm*.

1.1.1 Sekvenovanie

Sekvenovanie je proces, ktorým získavame genóm príslušného organizmu. Väčšina sekvenovacích postupov má spoločné črty. Pomocou zložitých chemických a fyzikálnych procesov sa najprv zistí mnoho krátkych reťazcov, ktoré sa v reťazci DNA nachádzajú na vopred neznámych miestach. Tieto krátke reťazce sa nazývajú *čítania*, z anglického *read*.

Z týchto čítaní sa potom pomocou rôznych algoritmov zisťuje, aký bol celý reťazec DNA. Tento proces, nazývaný *skladanie sekvencií*, je pomerne náročný, keďže čítania sa nachádzajú na neznámych miestach v DNA a navyše, väčšina metód na ich zistenie má malú šancu pomýliť sa pri určovaní niektorého nukleotidu. Preto sa pri každej metóde sekvenovania generuje toľko čítaní, aby v priaznivom prípade mnohonásobne pokryli celý reťazec DNA. Očakávaný počet, koľkokrát sme pokryli jeden konkrétny nukleotid týmito čítaniami, sa nazýva *pokrytie*, z anglického *coverage*.

1.1.2 Indexácia čítaní

Pokiaľ máme k dispozícii DNA reťazec nejakého jedinca, môže nás zaujímať, koľko a prípadne ktoré čítania sa prekrývajú s nejakou časťou. Pomocou takýchto dotazov vieme zisťovať mnohé zaujímavé informácie. Jednou možnosťou je napríklad hľadať rozdiely medzi jedincami jedného druhu, inou je zisťovať, ktoré časti DNA majú akú funkciu, prípadne ktoré majú vôbec nejakú. Pre prípady, kedy máme k dispozícii referenčnú sekvenciu živočíšneho druhu, existujú efektívne štruktúry, ktoré vedia na dané dotazy odpovedať.

V mnohých prípadoch však referenčnú sekvenciu k dispozícii nemáme, napriek tomu by sme radi vedeli odpovedať na podobné otázky. Jedno možné využitie štruktúry, ktorá vie odpovedať na takýto typ dotazov, je aj v niektorých algoritmoch na skladanie sekvencií, napríklad GAML [1].

V práci Philippe, Salsona a kol. [13] sa spomína sedem bežných typov otázok, ktoré môžeme mať na nejakú sadu čítaní:

1. Ktoré čítania obsahujú daný podreťazec f?
2. Koľko čítaní obsahuje daný podreťazec f?
3. Na ktorých pozíciach v čítaniach sa nachádza daný podreťazec f?
4. Koľkokrát sa dohromady vyskytuje daný podreťazec f v čítaniach?
5. V ktorých čítaniach sa vyskytuje podreťazec f iba raz?
6. V koľkých čítaniach sa vyskytuje podreťazec f iba raz?
7. Na ktorých pozíciach v čítaniach sa nachádza podreťazec f, pričom nás zaujímajú iba tie čítania, ktoré ho obsahujú raz?

1.2 Súvisiace práce

Prvou významnou prácou pri indexácii čítaní sú tzv. Gk-Arrays [13], ktoré vyhľadávajú podreťazce s pomocou upravených sufixových polí nad zreťazením všetkých čítaní. Myšlienku Gk-arrays posunuli ďalej Välimäki a Rivals, keď prišli s komprimovanou verziou [16].

S myšlienkovo odlišným riešením prišli Boža a Jursa. Namiesto zreťazenia čítaní použili v CR-indexe [2] nadslovo všetkých čítaní, v ktorom nájdu pozície všetkých čítaní a pomocou toho odpovedajú na rôzne dotazy. Keďže čítania sú v zamýšľanom použití podreťazcami celého genómu s malou šancou chýb, očakávaná dĺžka ich nadslova je zhruba veľkosť genómu.

1.3 Ciele práce

Jedným z problémov CR-indexu boli chyby v čítaniach. Aj s pomerne malou chybovosťou v dátach sa pri veľkom pokrytí objaví veľa čítaní s aspoň jednou chybou, ktoré s vysokou pravdepodobnosťou nezapadnú do nadslova a budú napísané na konci nadslova s malými prekryvmi s inými čítaniami.

Na druhú stranu, existujú nástroje, ktorými sa dá väčšina chýb v čítaniach zdetekovať a odstrániť. V CR-indexe pomocou takýchto nástrojov väčšinu chýb opravia

a nadslovo hľadajú k opraveným čítaniam. Keďže si chcú zachovať korektnosť, chyby ukladajú tak, že pre každú pozíciu každého čítania vedia povedať, či sa oproti časti nadslova, ku ktorej je čítanie zarovnané, líši.

Všetky čítania a pozície, ktoré nájdu, musia skontrolovať, či obsahujú chyby na správnych miestach. Takéto riešenie je dobré, pokiaľ chceme odpovedať na otázky prvého, tretieho, piateho a siedmeho typu, keďže pri nich musíme aj tak nejakým spôsobom prejsť cez všetky výsledky. Pokiaľ nás však zaujímajú počty, prípadne iná agregovaná informácia, v takomto riešení musíme aj tak overiť všetky potenciálne čítania, ktoré získame.

V tejto práci sa pozrieme na to, aké otázky sa dajú efektívne zodpovedať, pokiaľ oslabíme požiadavky na nadslovo, ktoré budeme pre čítania hľadať. Po tomto nadslove budeme požadovať iba to, aby obsahovalo každú k-ticu znakov, ktorá sa vyskytovala v niektorom čítaní, pre nejaké vopred zadané k. Aj keď zamýšľané použitie takéhoto prístupu sa orientuje na efektívne uchovávanie čítaní, problémy a algoritmy budeme formulovať všeobecnejšie, aby boli použiteľné aj v prípade, že nás zaujímajú podobné informácie o iných dátach s podobnými vlastnosťami, aké majú čítania.

1.4 Definície

Vrámci celej práce budeme považovať nulu za prirodzené číslo a polia budeme indexovať od nuly. Pod podslovom slova s budeme rozumieť súvislú podpostupnosť jeho písmen.

Jednou z centrálnych definícií bude pojem k-nadslova.

Definícia 1. *Nech $k > 1$ je celé číslo. Nech Σ je nejaká abeceda a nech S je množina slov dlhých aspoň k nad abecedou Σ. Potom k-nadslovom množiny S nazveme také slovo nad abecedou Σ, pre ktoré platí, že každé podslovo dĺžky k nejakého slova z S je zároveň podslovom k-nadslova.*

Nad poľom pravdivostných hodnôt si zadefinujeme operáciu $rank(i)$ ako počet hodnôt *Pravda* na pozíciách od nuly po i vrátane.

1.4.1 Grafové definície

V tejto práci budeme taktiež používať orientované grafy. Pod pojmom orientovaný graf G budeme rozumieť dvojicu (V, E), kde V je množina vrcholov a E je množina

usporiadaných dvojíc (v_1, v_2), kde v_1 aj v_2 sú vrcholmi grafu. Budeme hovoriť, že hrana e vedie z vrchola v_1 do vrchola v_2, pokiaľ $e = (v_1, v_2)$. Za slučku označíme každú hranu idúcu z vrchola do neho samého, čiže každú hranu typu (v, v).

Pre graf G budeme používať $V(G)$ na označenie jeho množiny vrcholov a $E(G)$ na označenie jeho množiny hrán. Pod množinou výstupných hrán vrchola v budeme rozumieť množinu $out(v) = \{(v, v_2)|v_2 \in V(G)\}$ a podobne, pod množinou vstupných hrán budeme rozumieť množinu $in(v) = \{(v_1, v)|v_1 \in V(G)\}$. Pre vrchol v určíme jeho výstupný stupeň ako $d_O(v) = |out(v)|$ a jeho vstupný stupeň ako $d_I(v) = |in(v)|$.

Za podgraf grafu G označíme ľubovoľný graf G' taký, že $V(G') \subseteq V(G)$ a zároveň $E(G') \subseteq E(G)$. Za faktorový podgraf grafu G označíme taký podgraf G', ktorý obsahuje všetky vrcholy G, čiže $V(G) = V(G')$.

V prípade, že budeme hovoriť o viacerých grafoch a z kontextu nie je jasné, na ktorý z nich sa niektorý z pojmov vzťahuje, tento graf zadefinujeme v hornom indexe, napríklad $d_O^{G_2}(v)$ bude označovať výstupný stupeň vrchola v v grafe G_2.

Pod pojmom *sled grafu* G budeme rozumieť ľubovoľnú striedavú postupnosť vrcholov a hrán $v_1e_1v_2e_2 \dots v_{n-1}e_{n-1}v_n$, ktorá začína aj končí nejakým vrcholom grafu G a v ktorej každá hrana e_i vedie z vrchola v_i do vrchola v_{i+1}. Sled označíme ako prázdny, pokiaľ je jeho postupnosť prázdna. Dĺžku sledu definujeme ako počet vrcholov v slede.

V orientovanom grafe je možnosť definovať rôzne typy súvislosti, v tejto práci sa stretneme iba so súvislosťou v neorientovanom zmysle. V ďalšej časti si formálne popíšeme tento typ súvislosti.

Za orientovaný doplnok grafu G označíme taký graf G', ktorý obsahuje každú hranu z grafu G obojsmerne, formálne zapísané:

1. $V(G) = V(G')$,
2. $E(G) \subseteq E(G')$,
3. $\forall v_1, v_2 \in V(G') : (v_1, v_2) \in E(G') \Rightarrow (v_2, v_1) \in E(G')$.

Na ľubovoľnom grafe G si zadefinujeme reláciu medzi vrcholmi nasledovne: vrchol v_1 je spojený s vrcholom v_2 práve vtedy, keď existuje sled v orientovanom doplnku grafu G, ktorý začína vo vrchole v_1 a končí vo vrchole v_2. Ľahko vidíme, že táto relácia je reláciou ekvivalencie. Za komponenty (súvislosti) grafu G označíme triedy ekvivalencie tejto relácie. Nakoniec za súvislý graf označíme taký, ktorý má jeden komponent.

Kapitola 2

Hľadanie nadslova

Problém, ktorý riešime v tejto práci sa dá rozdeliť na dve základné úrovne. Prvou z nich je hľadanie k-nadslova vstupných slov, druhou je indexácia čítaní pomocou k-nadslova.

V tejto kapitole sa pozrieme na prvú z týchto úrovní, pozrieme sa na zložitosť problému a na spôsob, akým vieme k-nadslovo hľadať efektívne, pokiaľ sú splnené pre naše použitie rozumné predpoklady.

Mohlo by sa zdať, že hľadanie k-nadslova je slabšou verziou problému hľadania spoločného nadslova, keďže ľubovoľné nadslovo nejakej množiny reťazcov bude zároveň k-nadslovom tejto množiny reťazcov a ľahko vidno, že naopak to neplatí.

Na druhú stranu, každé slovo dlhé $n > k$ vieme rozvinúť na $n - k + 1$ slov dlhých presne k tak, že hľadané nadslovo ostane presne to isté. V ďalšej časti si zhrnieme a popíšeme zopár výsledkov o zložitosti problému najkratšieho k-nadslova.

2.1 Zložitosť problému

Definícia 2. *Pod pojmom* rozhodovací problém k-nadslova *budeme rozumieť nasledovné: Pre danú množinu slov S, v ktorej každé slovo má dĺžku presne k, a dané l zistiť, či existuje k-nadslovo množiny S dlhé nanajvýš l.*

Najprv sa pozrieme na problém pre neobmedzenú abecedu. Keďže tento problém je vo svojej podstate rovnaký ako rozhodovací problém nadslova s fixnou dĺžkou reťazcov, môžeme využiť mnohé výsledky pre obyčajný problém nadslova: pre $k \leq 2$ je problém polynomiálne riešiteľný, pre $k \geq 3$ je $\mathcal{NP}$-úplný pomocou redukcie z problému

hamiltonovskej kružnice v orientovanom grafe [9]. Konštrukciu pre $k = 3$ si popíšeme.

Prvým krokom redukcie v článku Gallanta, Meiera a Storera je skonštruovať orientovaný graf G' s vopred určeným počiatočným aj koncovým vrcholom, v ktorom každý vrchol V okrem koncového spĺňa $d_O(V) > 1$. Pre daný orientovaný graf G skonštruujú nový graf rozdelením niektorého vrchola na počiatočný vrchol s a koncový vrchol t tak, že počiatočný si zachová všetky vychádzajúce a koncový všetky vchádzajúce hrany. Pre dodržanie veľkosti stupňa pridajú ešte tri vrcholy t', a, b, pridajú hrany z každého vrchola do t' a navyše hrany $(t, a), (t, b), (a, b), (b, a), (a, t')$. Tento nový graf G' obsahuje hamiltonovskú cestu práve vtedy, keď graf G obsahuje hamiltonovskú kružnicu.

Druhým krokom redukcie je skonštruovať z upraveného grafu množinu slov. Abecedu týchto slov definujú ako $\Sigma = V \cup B \cup S$, kde V sú vrcholy grafu označené číslami od 1 po n (1 je počiatočný a n koncový vrchol), $B = \{\bar{v} | v \in V\}$ je množina označených vrcholov a $S = \{\#, \$, \%\}$.

Ďalej skonštruujú množinu slov prislúchajúcich hranám vrchola v ako $A_v = \{\bar{v} w_i \bar{v} | w_i \in R_v\} \cup \{w_i \bar{v} w_{i\oplus 1} | w_i \in R_v\}$, kde R_v je postupnosť hrán vychádzajúcich z v a $\oplus$ je sčítanie modulo $d_O(v)$. Nakoniec skonštruujú množiny $C = \{\bar{v}\#v | v \in V\}$ a $T = \{\%\#\bar{1}, \bar{n}\#\$\}$. Nakoniec v článku dokážu, že graf G' obsahuje hamiltonovskú cestu práve vtedy, keď existuje nadslovo množiny slov $C \cup T \cup (\bigcup_{v \in V} A_v)$.

Dôležitou vlastnosťou tejto konštrukcie je, že stačí, aby každý reťazec mal dĺžku 3. Ďalšou dobrou vlastnosťou je, že konštrukcia používa $(2n + 10)$-znakovú abecedu.

2.1.1 Obmedzená veľkosť abecedy

S obmedzenou veľkosťou abecedy Σ je náročné hovoriť o zložitosti problému, keďže slov dĺžky k nad obmedzene veľkou abecedou je konečne veľa. Za dobrý odhad zložitosti problému však považujeme také číslo n, že pomocou rozhodovacieho problému k-nadslova vieme rozhodovať aj ľubovoľnú inštanciu problému hamiltonovskej kružnice s nanajvýš n vrcholmi.

Pri probléme s obmedzenou abecedou si teda vieme vystačiť s vhodným zakódovaním ľubovoľnej abecedy Σ_V do obmedzene veľkej abecedy Σ_M. Jediná vec, ktorú potrebujeme zaručiť je, že v k-nadslove sa nebudú kódy jednotlivých znakov prekrývať. Inštanciu problému s neobmedzenou abecedou potom vieme prekódovať do inštancie s obmedzenou abecedou a z každého k-nadslovu nájdeného v inštancii s obmedzenou abecedou vieme zrekonštruovať k-nadslovo pre pôvodnú inštanciu.

Pre jednoduchosť budeme uvažovať iba k deliteľné tromi. Pre ostatné k sa potom dá použiť podobná konštrukcia z inštancie problému orietovanej hamiltonovskej cesty, kde je graf ešte navyše bipartitný. Všetky kódovania budeme konštruovať tak, aby každému znaku z pôvodnej abecedy zodpovedalo rovnako dlhé slovo z obmedzenej abecedy.

Ako prvý si rozoberieme prípad, kde $|\Sigma_M| = n \geq 3$. Bez ujmy na všeobecnosti, nech $\Sigma = \{0, 1, \ldots, n-1, \$\}$. Znaky v abecede Σ_V si očíslujeme číslami od 0 po $|\Sigma_V| - 1$. Ako kód i-teho znaku použijeme reťazec $x\$$, kde x je číslo i zapísané v $(n-1)$-tkovej sústave, doplnené nulami tak, aby malo dĺžku presne $\frac{k}{3} - 1$. Keďže sa každý znak končí zarážkou nepoužitou na inom mieste v kóde, kódy jednotlivých znakov sa nemôžu prekrývať.

Pri uvedenom kódovaní má každý kód pôvodného znaku dĺžku presne $\frac{k}{3}$. Keďže konštrukcia inštancie problému nadslova k inštancii problému orientovanej hamiltonovskej kružnice používa iba slová dĺžky 3, všetky slová, ktoré vygenerujeme v našom kódovaní, budú mať dĺžku k. Keďže zakódovať vieme nanajvýš $(n-1)^{\frac{k}{3}-1}$ rôznych vrcholov, pre dané k vieme problém orientovanej hamiltonovskej cesty s nanajvýš $\frac{(n-1)^{\frac{k}{3}-1}-10}{2}$ vrcholmi zredukovať na problém hľadania najkratšieho k-nadslova reťazcov nad n-znakovou abecedou.

2.2 Abstrakcia

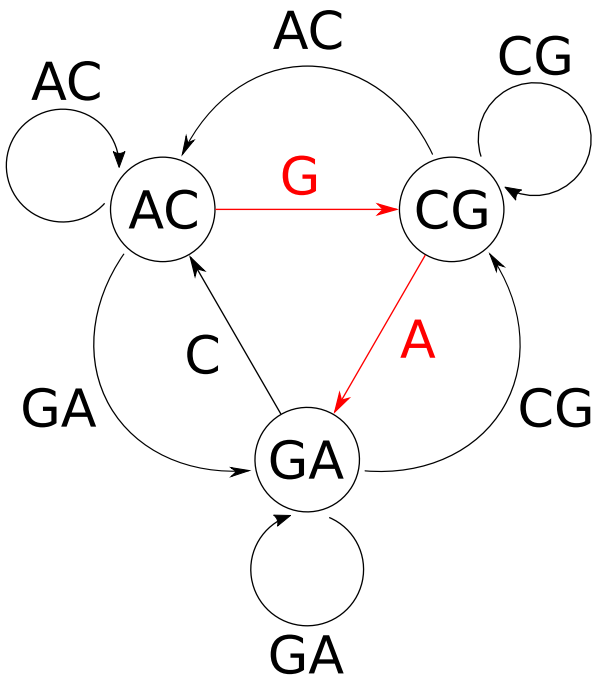
Po zdôvodnení a dokázaní zložitosti riešeného problému sa pozrieme na spôsob, akým ho budeme riešiť. Ako prvé si objasníme abstrakciu problému, na ktorej riešenie postavíme.

Sadu vstupných slov si budeme reprezentovať ako mierne upravený ohodnotený de Bruijnov [3] graf stupňa $k-1$ nad abecedou Σ. Každej hrane symbolicky priradíme reťazec, ktorého dĺžka bude vždy zároveň hodnotou hrany. Naše riešenie, čiže nejaké spoločné k-nadslovo, budeme reprezentovať ako sled v tomto grafe.

Vrcholmi grafu budú všetky $(k-1)$-tice znakov, ktoré sa vyskytujú v aspoň jednom slove. Hrany v grafe pôjdu z každého vrchola do každého, vrátane slučiek. Tieto hrany si rozdelíme na dve skupiny: *nutné* a *nepovinné*. Za *nutnú* hranu označíme každú takú hranu $e = ((x_1 x_2 \ldots x_{k-1}), (y_1 y_2 \ldots y_{k-1}))$, pre ktorú platí $x_{i+1} = y_i$ pre každé i v rozsahu od 1 po $k-2$ a zároveň k-tica znakov $x_1 y_1 y_2 \ldots y_{k-1}$ sa vyskytuje v aspoň jednom slove. Ostatné hrany budú *nepovinné*.

Než si popíšeme, aké reťazce priradíme hranám, budeme potrebovať poriadne zadefinovať ešte jeden pojem.

Definícia 3. *Nech s_1 a s_2 sú reťazce znakov nad abecedou Σ a nech z je najdlhší reťazec*



Obr. 2.1: Príklad 2-grafu množiny $\{\texttt{ACG}, \texttt{CGA}\}$. *Nutné* hrany sú vyznačené červenou farbou, čierne hrany sú *nepovinné*.

taký, že

$$\exists x, y \in \Sigma^* : s_1 = xz \wedge s_2 = zy \wedge |x| \geq 1$$

Potom prekryvom reťazca s_1 s reťazcom s_2 označíme reťazec z a dokončením s_1 v s_2 označíme taký reťazec y, ktorý spĺňa $s_2 = zy$.

Každej hrane z vrchola S_1 do vrchola S_2 priradíme dokončenie S_1 v S_2. Pripomíname, že hodnotou tejto hrany bude dĺžka priradeného reťazca. Takto skonštruovaný graf k množine vstupných slov S budeme ďalej označovať ako ***k*-graf množiny *S***. Konkrétny príklad môžeme vidieť v obr. 2.1.

Kvôli časti o implementácii si zadefinujeme ešte pojem *nutného podgrafu*. Pod pojmom *nutný podgraf* k-grafu množiny S budeme rozumieť faktorový podgraf k-grafu množiny S obsahujúci iba nutné hrany.

Môžeme si všimnúť, že reťazec priradený každej nutnej hrane bude mať vždy jeden znak. Ďalším pozorovaním, ktoré sa nám bude hodiť najmä pri implementácii je, že reťazec priradený každej hrane vieme vypočítať iba z označení jej koncových vrcholov. Ďalej, každý vrchol v nutnom podgrafe k-grafu množiny S má vstupný aj výstupný vrchol zhora ohraničený veľkosťou abecedy. Posledné dôležité pozorovanie je, že každá hrana má kladnú cenu.

Slovo prislúchajúce sledu v k-grafe množiny S získame ako zreťazenie zr, kde z je

označenie prvého vrchola sledu a r sú pospájané reťazce priradené hranám v takom poradí, v akom sa tieto hrany nachádzajú v slede.

Formálnejšie zapísané, nech $slovo(V)$ označuje reťazec dĺžky $k-1$, označenie vrchola V a $slovo(e)$ označuje reťazec priradený hrane e. Potom $slovo(w)$, slovo prislúchajúce sledu $w = V_1e_1V_2e_2 \dots e_{n-1}V_n$ skonštruujeme nasledovne:

$$slovo(w) = slovo(V) \cdot slovo(e1) \cdot slovo(e2) \cdots slovo(e_{n-1}).$$

Pre prázdny sled w definujeme $slovo(w) = \varepsilon$, čiže prázdne slovo.

Z tejto konštrukcie vidíme, že dĺžku slova priradeného neprázdnemu sledu vieme vypočítať ako $k - 1 + \sum_{i=1}^{n-1} cena(e_i)$

Sled v k-grafe množiny S budeme volať *korektný*, ak obsahuje všetky nutné hrany. Teraz si dokážeme, že slovo prislúchajúce korektnému sledu v k-grafe množiny S je k-nadslovom množiny S. K tomu budeme potrebovať ešte jednu pomocnú lemu.

Lema 4. *Nech $V_1e_1V_2e_2 \dots e_{n-1}V_n$ je neprázdny sled v k-grafe nejakej množiny S. Potom sa reťazec r prislúchajúci tomuto sledu končí $(k-1)$-ticou znakov zhodnou s označením vrchola V_n.*

Keďže túto lemu je možné dokázať čisto technickou matematickou indukciou, dôkaz prenechávame čitateľovi.

Veta 5. *Nech W je korektný sled v k-grafe množiny S. Potom $slovo(W)$ je k-nadslovom množiny slov S.*

Dôkaz. Vezmime si ľubovoľné slovo w také, že $|w| = k$ a w je podslovom nejakého slova z S. Nech $x_1, x_2, \cdots x_k$ sú znaky tohto slova v poradí. Potom sa v grafe vyskytujú vrcholy $V_1 = (x_1, x_2, \cdots, x_{k-1})$ a $V_2 = (x_2, x_3, \cdots x_k)$ také, že z V_1 ide hrana e do V_2 a e je nutná. Z toho vyplýva, že $W = U_1V_1eV_2U_2$, kde U_1, U_2 sú nejaké časti sledu. Z predošlej lemy potom vyplýva, že existuje $q \in \Sigma^*$ také, že $slovo(U_1V_1) = q \cdot x_1 \cdots x_{k-1}$, a teda

$$slovo(W) = slovo(U_1V_1eV_2U_2) = q{\cdot}x_1 \cdots x_{k-1}{\cdot}slovo(e){\cdot}slovo(V_2U_2) = q{\cdot}w{\cdot}slovo(V_2U_2).$$

□

Pre úplnosť uvedieme ešte jednu dobrú vlastnosť k-grafu množiny S, ktorá neskôr uľahčí konštrukciu k-nadslova.

Lema 6. *Nech U a V sú vrcholmi k-grafu množiny S. Potom najlacnejší sled začínajúci v U a končiaci vo V obsahujúci aspoň jednu hranu, má rovnakú cenu, ako hrana e z vrchola U do V.*

Dôkaz. Sporom. Nech $w := Ue_1U_1e_2 \cdots e_nV$ je kratšia cesta. Z lemy 4 vieme, že aj $slovo(UeV)$ aj $slovo(w)$ sa končia označením vrchola V, zároveň, keďže w má menšiu cenu ako e, platí $|slovo(w)| < |slovo(UeV)|$. To je ale v spore s tým, že prekryv U s V je najdlhší možný, $slovo(w)$ nám dáva návod, ako zostrojiť kratší. □

2.2.1 Vlastnosti abstrakcie

Zatiaľ čo každému korektnému sledu v abstrakcii prislúcha práve jedno k-nadslovo, nie každé k-nadslovo vieme reprezentovať ako sled v našom grafe.

Vezmime si ako vstupné slová `ACG` a `CGA` a k rovné trom. Graf prislúchajúci takémuto vstupu je znázornený na obrázku 2.1. V grafe máme tri vrcholy zodpovedajúce dvojiciam `(AC)`, `(CG)` a `(GA)`. Hrany aj s označením, ktoré budeme používať ďalej:

- *nutná* hrana e_1 z vrchola `(AC)` do `(CG)` s priradeným reťazcom `G`,
- *nutná* hrana e_2 z vrchola `(CG)` do `(GA)` s priradeným reťazcom `A`,
- *nepovinná* hrana e_3 z vrchola `(AC)` do `(GA)` s reťazcom `GA`,
- *nepovinná* hrana e_4 z vrchola `(CG)` do `(AC)` s reťazcom `AC`,
- *nepovinná* hrana e_5 z vrchola `(GA)` do `(AC)` s reťazcom `C`,
- *nepovinná* hrana e_6 z vrchola `(GA)` do `(CG)` s reťazcom `CG`.
- *nepovinná* hrana e_7 z vrchola `(AC)` do `(AC)` s reťazcom `AC`,
- *nepovinná* hrana e_8 z vrchola `(CG)` do `(CG)` s reťazcom `CG`,
- *nepovinná* hrana e_9 z vrchola `(GA)` do `(GA)` s reťazcom `GA`.

Celkom ľahko vidíme, že najkratšie možné k-nadslovo musí mať aspoň štyri znaky. Vhodné k-nadslovo takejto dĺžky naozaj existuje, napríklad `ACGA`. V našom grafe ho vieme reprezentovať ako sled:

$$\texttt{(AC)}e_1\texttt{(CG)}e_2\texttt{(GA)}$$

Ak sa ale pozrieme napríklad na slovo TTACGTTTCGATT, neexistuje žiaden sled, ktorému prislúcha toto k-nadslovo, keďže žiaden vrchol ani hrana neobsahujú znak T. Ľahko ale vidíme, že toto slovo vieme skrátiť vynechaním znakov T, ktoré sa nemôžu objaviť v žiadnom k-nadslove konštruovanom k sledu, na stále vyhovujúce nadslovo ACGCGA. Toto slovo vieme reprezentovať ako sled:

$$\texttt{(AC)}e_1\texttt{(CG)}e_8\texttt{(CG)}e_2\texttt{(GA)}$$

Zdá sa teda, že ak nejakú časť k-nadslova nevieme reprezentovať ako časť sledu v k-grafe, môžeme ju nejakým spôsobom vynechať. Túto vlastnosť si ďalej zhrnieme. Predtým si zadefinujeme konštrukciu sledu k ľubovoľnému k-nadslovu.

Definícia 7. *Nech S je množina slov nad abecedou Σ, $s = x_1x_2\cdots x_n$ je nejakým slovom nad abecedou Σ, nech G je k-grafom množiny S, nech $Z = \{i \in \{1\ldots n-k+2\} | (x_ix_{i+1}\cdots x_{i+k-2}) \in V(G)\}$, čiže Z je množina takých indexov v s, že $(k-1)$-tica písmen začínajúca na tejto pozícii označuje vrchol v grafe G a nech v_i je i-ty najmenší prvok množiny Z. Potom pre neprázdnu množinu Z určíme k-sled slova s pri množine S ako*

$$sled_{k,S}(s) = V_1e_1V_2e_2\cdots e_{p-1}V_p,$$

kde $V_i = (x_{v_i}x_{v_i+1}\cdots x_{v_i+k-2})$ a e_i je hrana medzi V_i a V_{i+1}. Pre prázdnu množinu Z určíme $sled_{k,S}(s)$ ako prázdny sled.

Na túto konštrukciu sa vieme pozerať aj tak, že zo slova s vytiahne tie časti – vrcholy, ktoré vieme reprezentovať v našom grafe G a pospája ich hranami. To, že táto konštrukcia zachytáva všetky pre nás dôležité časti slova a zároveň ho nenafúkne, si ďalej dokážeme.

Veta 8. *Nech $s = x_1x_2\cdots x_n$ je k-nadslovom množiny S. Potom aj $slovo(sled_{k,S}(s))$ je k-nadslovom množiny S.*

Dôkaz. Nech $q = a_1a_2\cdots a_k$ je ľubovoľná k-tica písmen vyskytujúca sa v nejakom slove množiny S. Keďže s je k-nadslovom množiny S, platí nasledovné:

$$\exists i : a_1 = x_i \wedge a_2 = x_{i+1} \wedge \ldots \wedge a_k = x_{i+k-1}$$

Zároveň z konštrukcie k-grafu množiny S vyplýva, že obsahuje vrcholy $U_1 := (x_ix_{i+1}\cdots x_{i+k-2})$ a $U_2 := x_{i+1}x_{i+2}\cdots x_{i+k-1}$. Tým pádom množina Z z konštrukcie sledu obsahuje aj x_i, aj x_{i+1}, čiže $sled_{k,S}(s)$ obsahuje podsled U_1eU_2 a teda $slovo(sled_{k,S}(s))$ obsahuje q. □

Veta 9. *Nech S je množina slov nad abecedou Σ, n je prirodzené a $k \geq 2$ je celé číslo, nech $s = a_1a_2 \cdots a_n$ je nejakým slovom nad Σ. Potom $|slovo(sled_{k,S}(s))| \leq |s|$.*

Dôkaz. Nech G je k-graf množiny S. Dokážeme matematickou indukciou na n, dĺžku slova s.

$1^0 : n < k-1$. Tento prípad pokrýva napríklad prázdne slová s, čiže je vhodný ako báza indukcie.

V s sa nenachádza žiadna $(k-1)$-tica písmen, čiže množina Z z konštrukcie k-sledu bude prázdna, teda aj $sled_{k,S}(s)$ bude prázdny, čiže $slovo(sled_{k,S}(s)) = \varepsilon, |\varepsilon| = 0 \leq n$.

2^0 : Položíme $s' := a_1a_2 \cdots a_{n-1}$, čiže s skrátený o posledný znak. V indukčnom kroku nás zaujímajú iba slová s dĺžky aspoň $k-1$. Rozoberieme tri základné možnosti:

- $(a_{n-k+2}a_{n-k+3} \cdots a_n) \notin V(G)$. Množina Z z konštrukcie sledu bude rovnaká pre s aj s', čiže $slovo(sled_{k,S}(s)) = slovo(sled_{k,S}(s'))$ a teda $|slovo(sled_{k,S}(s))| = |slovo(sled_{k,S}(s'))| \leq |s'| < |s|$, kde prvá nerovnosť vyplýva z indukčného predpokladu.

- $(a_{n-k+2}a_{n-k+3} \cdots a_n) \in V(G)$ a $sled_{k,S}(s')$ je prázdny. Potom množina Z z konštrukcie sledu k s bude obsahovať jeden prvok, čiže $sled_{k,S}(s)$ obsahuje iba jeden vrchol. Preto $|slovo(sled_{k,S}(s))| = k - 1 \leq |s|$.

- $V_0 := (a_{n-k+2}a_{n-k+3} \cdots a_n) \in V(G)$ a $sled_{k,S}(s')$ je neprázdny. Položíme $j := max\{i \in \{1 \ldots n-k+1\} | (x_ix_{i+1} \cdots x_{i+k-2}) \in V(G)\}$, čiže začiatok poslednej $(k-1)$-tice zodpovedajúcej nejakému vrchola v G a $s'' := a_1a_2 \cdots a_{j+k-2}$, čiže s' skrátené po koniec poslednej $(k-1)$-tice zodpovedajúcej nejakému vrchola v G. Slová s' a s'' majú rovnakú množinu Z z konštrukcie sledu, čiže majú priradený rovnaký sled a teda $slovo(sled_{k,S}(s')) = slovo(sled_{k,S}(s''))$.

  Položme $sled_{k,S}(s'') = V_1e_1 \cdots V_p$. Sled k slovu s obsahuje práve jeden vrchol navyše, čiže $sled_{k,S}(s) = V_1e_1 \cdots V_pe_pV_0$, kde e_p je hrana z V_p do V_0. Z konštrukcie slova k sledu vieme nasledovné:

$$slovo(sled_{k,S}(s)) = slovo(V_1)slovo(e_1)slovo(e_2) \cdots slovo(e_p) =$$
$$= slovo(sled_{k,S}(s'')) \cdot slovo(e_p).$$

  Podľa lemy 4 sa $slovo(sled_{k,S}(s''))$ končí označením vrchola V_q, čiže rovnako, ako s'', podobne sa $slovo(sled_{k,S}(s))$ končí tou istou $(k-1)$-ticou ako s, z indukčného predpokladu vyplýva, že $|slovo(sled_{k,S}(s''))| \leq |s''|$. Ak by teda platilo $|slovo(sled_{k,S}(s))| > |s|$, bolo by dokončenie s v s'' kratšie, než dokončenie V_0

vo V_q, čo je v spore s tým, že prekryv V_0 s V_q je definovaný ako najdlhší možný, podobne ako v dôkaze lemy 6. Aj v tomto prípade teda platí indukčný krok. □

Dôsledok 10. *Nejaké najkratšie k-nadslovo množiny S zodpovedá nejakému najlacnejšiemu* korektnému *sledu v k-grafe množiny S.*

2.3 Algoritmus na hľadanie najkratšieho k-nadslova

V predošlej časti sme si dokázali, že pre nájdenie najkratšieho k-nadslova množiny slov S nám stačí nájsť najlacnejší korektný sled v k-grafe množiny S, čiže sled, ktorý obsahuje všetky *nutné* hrany grafu. Ide o pomerne známy **problém vidieckeho poštára** z grafovej teórie, ktorý je vo všeobecnom prípade $\mathcal{NP}$-ťažký [11].

Ukazuje sa však, že pokiaľ podgraf k-grafu množiny S obsahujúci iba *nutné* hrany je súvislý, tento problém je polynomiálne riešiteľný a riešenie je takmer zhodné s riešením **problému čínskeho poštára** na orientovaných grafoch [6]. Keďže zamýšľané použitie k-nadslova je na reprezentáciu sekvenčných čítaní, ktoré sú krátkymi podreťazcami celého genómu s vysokou mierou pokrytia, je pomerne pravdepodobné, že nutné hrany k-grafu týchto čítaní budú tvoriť stále súvislý graf, prípadne že budú obsahovať iba málo komponentov. Najprv sa teda pozrieme, ako hľadať k-nadslovo v súvislom k-grafe.

2.3.1 Súvislý graf

Problém čínskeho poštára [4] sa zaoberá hľadaním najlacnejšieho sledu v grafe takého, že každou hranou prejde aspoň raz.

Základnou myšlienkou riešenia problému čínskeho poštára je pridať hrany s čo najmenším súčtom do grafu tak, aby obsahoval Eulerovský ťah [4]. Tento prístup funguje preto, že každý sled priamo zodpovedá Eulerovskému ťahu pre nejaké doplnenie hrán do grafu.

Súvislý orientovaný graf obsahuje Eulerovskú kružnicu práve vtedy, keď vstupný stupeň d_I je rovný výstupnému stupňu d_O v každom vrchole, Eulerovský ťah existuje aj v prípade, keď sa v najviac dvoch vrcholoch vstupný a výstupný stupeň líšia o jedna. Ak teda k-graf množiny S má túto vlastnosť, nájdenie korektného sled je už len záležitosťou nájdenia Eulerovského ťahu.

Pokiaľ nutný podgraf k-graf množiny S túto vlastnosť nemá, potrebujeme čo najlacnejšie pridať hrany z vrcholov s $d_O < d_I$ do vrcholov s $d_I < d_O$, pričom využívať môžeme aj nepovinné hrany. Z lemy 6 vyplýva, že najlacnejšia cesta medzi dvomi vrcholmi je hrana medzi nimi, čiže zaujímať nás budú iba hrany medzi vrcholmi. Z toho vyplýva aj konštrukcia nasledovného grafu, ktorého schematické znázornenie uvádzame v obr. 2.2.

Definícia 11. *Nech G je k-graf množiny S a G' je jeho nutný podgraf. Tokovým grafom grafu G označíme graf H, ktorého množinu vrcholov skonštruujeme nasledovne: $V(G') = \{s, t\} \cup V_h \cup V_p$, kde V_h (hladné) je množina $\{v \in V(G) | d_O^{G'}(v) > delta_I^{G'}(v)\}$ a V_p (presýtené) je množina $\{v \in V(G) | d_I^{G'}(v) > delta_O^{G'}(v)\}$. Vrcholy množiny V_h budeme volať hladné, vrcholy množiny V_p presýtené. Hrany grafu H budú troch typov:*

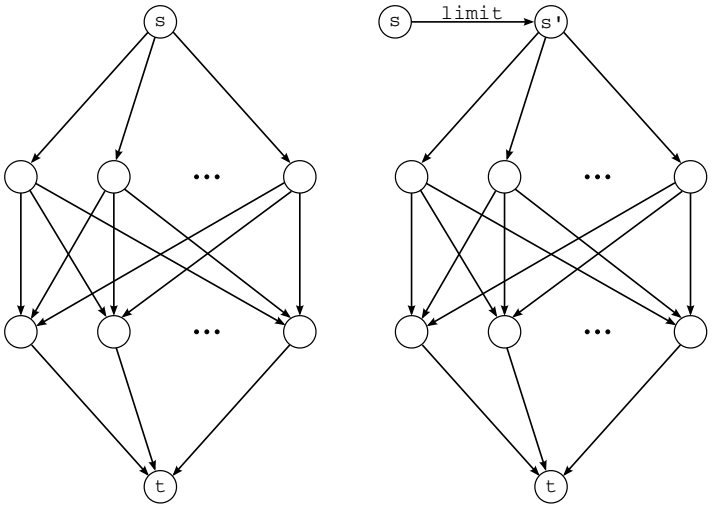
1. *Tie hrany z grafu G, ktoré idú z presýteného vrchola do hladného s cenou rovnakou ako v G a kapacitou ∞,*
2. *Hrany od s do každého vrchola vo V_p s cenou 0. Kapacitu takéhoto typu hrany do vrchola U určíme ako $d_I^{G'}(U) - d_O^{G'}(U)$,*
3. *Hrany od každého vrchola z V_h do t s cenou 0. Kapacitu takéhoto typu hrany z vrchola U určíme ako $d_O^{G'}(U) - d_I^{G'}(U)$.*

Žiadne iné hrany v grafe H nebudú.

Nájdenie najlacnejšieho maximálneho toku v tokovom grafe k-grafu bude zodpovedať najlacnejšiemu možnému pridaniu hrán do nutného podgrafu k-grafu tak, aby v ňom existovala Eulerovská kružnica. Jednoducho pre každý nasýtený vrchol V a hladný vrchol U pridáme x hrán z V do U, kde x je veľkosť toku tečúca z vrchola V do U.

Keďže náš korektný sled nemusí začínať aj končiť v tom istom vrchole, potrebujeme nájsť v skutočnosti najlacnejší tok, ktorý je o 1 menší než maximálny a pridať hrany podľa neho. Takýto tok môžeme nájsť napríklad pomocou algoritmu Edmondsa a Karpa [5] v čase $O(n^3)$, kde n je počet vrcholov v tokovom grafe nášho k-grafu, prípadne pomocou akéhokoľvek potenciálne rýchlejšieho algoritmu na dve simulácie, kde v prvej zistíme hodnotu maximálneho toku a v druhej vhodným pridaním jedného vrchola vynútime obmedzenie najväčšieho toku na požadovanú hodnotu (obr. 2.2).

Ako posledné nám teda stačí nájsť Eulerovský ťah v takto upravenom grafe a zrekonštruovať z neho nadslovo. Týmto sme si popísali polynomiálny algoritmus na hľadanie k-nadslova, pokiaľ je nutný podgraf k-grafu súvislý. Pre rekapituláciu si celý algoritmus uvedieme v dôkaze nasledujúcej vety.



Obr. 2.2: Ukážka tokových grafov. Naľavo je znázornený základný tokový graf, napravo je znázornený tokový graf s obmedzením maximálneho toku.

Veta 12. *Nech S je množina slov dlhých aspoň k nad abecedou Σ. Potom ak je nutný podgraf k-grafu množiny S súvislý, existuje polynomiálny algoritmus na hľadanie k-nadslova.*

Dôkaz. Uvedieme si algoritmus, ktorý k-nadslovo nájde. Ku každému kroku uvedieme aj časovú zložitosť od počtu slov v množine S, pre jednoduchosť za predpokladu, že každý reťazec má rozumne malú fixnú dĺžku.

1. Zostav nutný podgraf k-grafu zo vstupných slov a prečísluj vrcholy do intervalu $[0, pocet)$. $O(nk)$
2. Vyrieš problém najlacnejšieho maximálneho toku na tokovom grafe. $O(n^3)$
3. Pridaj hrany do grafu podľa výsledného toku. $O(n^2)$
4. Nájdi Eulerovský ťah. $O(n)$
5. Transformuj Eulerovský ťah na k-nadslovo. $O(nk)$ □

Poslednou vecou, ktorou sa budeme zaoberať v súvislom prípade, je časová zložitosť. Všetky časti algoritmu sa vykonávaju v čase lineárne závislom od veľkosti vstupu, okrem druhého a tretieho bodu. V týchto bodoch je pomerne nepravdepodobné, že existuje riešenie rýchlejšie, než kvadratické. To je spôsobené tým, že v najhoršom prípade môžu takmer všetky vrcholy grafu mať d_I rôzne od d_O a algoritmus, ktorý nájde najlepšie priradenie, sa potrebuje pozrieť na každú hranu.

Keďže hľadanie k-nadslova chceme využiť do dátovej štruktúry schopnej reprezentovať milióny až stovky miliónov genetických čítaní, chceli by sme nájsť nejaký spôsob, ako v lineárnom čase nájsť aspoň nejaké rozumné pospájanie týchto vrcholov. V našom navrhovanom riešení jednoducho vyskúšame pevne stanovený počet náhodných priradení a vyberieme z nich to najlacnejšie.

2.3.2 Nesúvislý graf

Už na začiatku sme si dokázali, že hľadanie k-nadslova je v istom zmysle ťažký problém. Keďže sme si popísali a zdôvodnili polynomiálny algoritmus, ktorý vie nájsť k-nadslovo pre súvislý nutný podgraf k-grafu vstupnej množiny, je pomerne pravdepodobné, že v našej špeciálnej triede grafov neexistuje polynomiálne riešenie problému vidieckeho poštára.

Naše riešenie v nesúvislom prípade teda bude vyzerať úplne rovnako, v predošlom riešení pozmeníme len posledný krok, hľadanie Eulerovského ťahu. Keďže taký v našom grafe ani po pridaní hrán nemusí existovať, jednoducho nájdeme Eulerovský ťah v každom komponente a výsledné slová nadpojíme na seba. Na toto sa môžeme pozerať aj tak, že si medzi niektoré vrcholy v rôznych komponentoch pridáme hranu.

Ako sme spomínali vyššie, z povahy dát, pre ktoré hľadáme k-nadslovo, je celkom pravdepodobné, že náš nutný podgraf bude súvislý, prípadne že bude mať iba málo komponentov. Zároveň, v našom riešení pridáme nanajvýš $c - 1$ hrán navyše oproti optimálnemu riešeniu. Ďalej vieme, že sme do grafu pridávali hrany najlacnejším spôsobom tak, aby v každom vrchole platilo $d_I = d_I$. Keďže najväčšia možná hodnota hrany je $k - 1$, znamená to, že riešenie, ktoré takto nájdeme, bude najviac o $(k - 1)(c - 1)$ drahšie ako optimálne, čiže takto nájdené k-nadslovo bude nanajvýš o toľko dlhšie od najkratšieho možného.

Kapitola 3

Indexácia k-nadslova

V predošlej kapitole sme si popísali, ako vieme nájsť k-nadslovo k nejakej množine reťazcov. V tejto kapitole si popíšeme rôzne typy dotazov, ktoré vieme implementovať s pomocou nájdeného k-nadslova.

Spomenuté typy dotazov sú určené primárne na zisťovanie rôznych vlastností sady čítaní, preto množinu vstupných reťazcov budeme nazývať *čítania*. Pripomíname, že všetky dotazy budú zisťovať nejakú informáciu o podreťazcoch, ktoré sú dlhé práve k.

3.1 Prítomnosť podreťazca

Úplne najzákladnejšou otázkou, na ktorú chceme vedieť odpovedať, je prítomnosť nejakého podreťazca dĺžky presne k vo vstupných čítaniach. Ako sa neskôr ukáže, vedieť odpovedať na takýto dotaz nám pomôže vybudovať dodatočné podporné štruktúry, pomocou ktorých budeme vedieť odpovedať aj na zložitejšie dotazy.

Ako prvé je dôležité všimnúť si, že prítomnosť nejakého reťazca v k-nadslove nám nezaručuje, že sa vyskytoval aj v nejakom z pôvodných čítaní. Vezmime si napríklad $k = 4$ a dve čítania, `ACGA` a `ACGT`. Ich k-nadslovom môže byť napríklad reťazec `ACGACGT`, ktorý síce obsahuje aj podreťazec `CGAC`, no ten sa nevyskytuje ani v jednom z pôvodných čítaní.

Pre tento účel mierne modifikujeme algoritmus na hľadanie k-nadslova tak, aby nám vedel pre každý znak výsledného k-nadslova povedať, či sa ním končí nejaká k-tica, ktorá sa vyskytuje v niektorom pôvodnom čítaní. To docielime jednoducho tak, že pre každú hranu, ktorú budeme mať v našom grafe, si zapamätáme, či reprezentuje

nutnú hranu, inými slovami povedané, či sa k-tica, ktorú reprezentuje, vyskytuje v niektorom zo vstupných čítaní.

Pri prepisovaní sledu na konkrétny reťazec si budeme zároveň vytvárať pole pravdivostných hodnôt, do ktorého pre každý znak, ktorý pripisujeme na koniec generovaného k-nadslova, uložíme hodnotu *Pravda*, ak práve pridávaný znak prichádza z nutnej hrany. Prvých $k-1$ znakov, preklad prvého vrchola, dostane hodnotu *Pravda*, keďže každý vrchol predstavuje $k-1$-ticu znakov, ktorá sa nachádzala v niektorom vstupnom slove. Toto pole budeme nazývať *poľom výskytov*.

Nakoniec potrebujeme pre každý dotazovaný podreťazec určiť, kde sa nachádza v k-nadslove. Pre tento účel využijeme štruktúru FM-index [8], konkrétne jeho implementáciu z knižnice SDSL [10]. Pomocou FM-indexu vieme zistiť, kde sa začínajú všetky výskyty nejakého podreťazca v k-nadslove v čase priamo úmernom súčtu dĺžky vyhľadávaného podreťazca a počtu výskytov. Napriek tomu zaberá pri dĺžke uloženého slova m zhruba $m/2$ bajtov.

Na dotazovaný reťazec r teda nájdeme všetky jeho začiatky v k-nadslove. Ak v poli výskytov nájdeme aspoň pre jeden koniec výskytu hodnotu *Pravda*, odpovieme tiež *Pravda*, v opačnom prípade odpovieme *Nepravda*.

Aby sme čo najviac zmenšili veľkosť poľa výskytov, uložíme si ho skomprimované podľa techniky z článku Ramana, Ramana a Raa [14]. Takto zostrojené pole dĺžky m s n nastavenými bitmi má veľkosť $\beta(n,m) + o(n) + O(loglogn)$ bitov, kde $\beta(n,m)$ je dolné teoretické obmedzenie na počet bitov potrebných na reprezentáciu n-prvkovej podmnožiny z m-prvkovej množiny, zároveň však vie zisťovať hodnotu na danej pozícii v konštantnom čase. Používame implementáciu z SDSL.

3.2 Počet výskytov podreťazca

Podobne ako zisťovanie prítomnosti nejakého podreťazca môžeme vyriešiť aj podporu pre dotazy, koľkokrát sa daný podreťazec nachádza v zadaných čítaniach. Jediný rozdiel bude v tom, že si pre každú nutnú hranu budeme pamätať, koľkokrát sa vyskytovala jej zodpovedajúca k-tica písmen v čítaniach; pre nepovinné hrany si budeme pamätať nulu.

Podobne ako pri dotazoch na prítomnosť si pre každý znak v k-nadslove budeme pamätať jednu hodnotu – koľko k-tic zo vstupu sa končí týmto znakom. Toto pole budeme ďalej nazývať poľom počtu výskytov.

Spracovávanie dotazu potom vyzerá podobne, ako pri zisťovaní prítomnosti. Nájdeme si všetky výskyty dotazovaného reťazca v FM-indexe. Spomedzi týchto výskytov nás bude zaujímať ten, u ktorého poslednému znaku zodpovedá nenulové číslo v poli počtu výskytov a toto číslo vrátime ako odpoveď.

Keďže každú k-ticu pri konštruovaní reprezentujeme ako jednu nutnú hranu, práve jeden výskyt zodpovedá tejto hrane a je nenulový, ostatné musia byť nulové. Dôležité je však poznamenať, že takéto riešenie dotazu nemusí dávať správnu odpoveď pre dotazované reťazce kratšie ako k. Pre daný podreťazec r kratší než k môžeme mať viacero výskytov, ktoré zodpovedajú nejakej nutnej hrane. Ak však sčítame počty výskytov týchto hrán, stále nemusíme započítať všetky výskyty, ktoré treba.

Základným problémom sú výskyty na začiatkoch čítaní. Vezmime si napríklad nejaké čítanie c. Všetky výskyty nejakého podreťazca kratšieho než k, ktoré sa končia na k-tom, alebo neskoršom znaku čítania c, započítame správne, keďže pre ne existuje práve jedna k-tica, ktorá sa nimi končí a v nej tieto kratšie výskyty započítame. Pre výskyty, ktoré sa končia na skoršom mieste v čítaní c, neexistuje žiadna hrana, v ktorej by sme ich mohli započítať. Podobný problém nastáva aj pri zisťovaní prítomnosti podreťazca kratšieho, než k.

Aby toto pole počtov výskytov nezaberalo oveľa viac pamäte, než samotný FM-index, uložíme si ho skomprimované pomocou prefixového kódovania [7], implementovaného s náhodným prístupom v konštantnom čase v knižnici v SDSL [10]. Prístup k nejakému prvku je vďaka vzorkovaniu poľa v konštantnom čase, aj keď pomalšie, než do bežného poľa. Experimenty ukázali, že takto skomprimované pole zaberá zhruba toľko miesta, čo FM-index.

3.3 Čítania s podreťazcom

Posledný typ dotazu, na ktorý sa pozrieme, bude pre daný podreťazec zistiť, v ktorých čítaniach sa nachádza. Ako uvidíme neskôr, podobne budeme vedieť zistiť aj to, kde sa v daných čítaniach nachádza.

Základnou myšlienkou nášho riešenia je, podobne ako v CR-indexe [2], uložiť si začiatky jednotlivých čítaní. Keďže sa rôzne k-tice z čítania môžu nachádzať na rôznych miestach, pre každé čítanie môžeme mať týchto počiatočných pozícií viacero. Keď si nejaké konkrétne čítanie priložíme na nejakú pozíciu v k-nadslove, nie všetky jeho znaky sa budú zhodovať s nadslovom.

Informáciu o tom, ktoré pozície čítania sa zhodujú s nadslovom, si budeme ukladať do jedného poľa pravdivostných hodnôt, nazvime ho pole správnych pozícií. Pre jednoduchosť budeme predpokladať, že každé čítanie má rovnakú dĺžku q. Pre každý začiatok nejakého prekryvu si vyhradíme q hodnôt, pričom každá z nich označuje, či sa znak na zodpovedajúcom mieste zhoduje s príslušným čítaním.

Zistiť začiatky a pole správnych pozícií je pomerne priamočiare. Znova prejdeme cez všetky čítania. Pre každú k-ticu si zistíme, kde sa nachádza v k-nadslove pomocou FM-indexu. To nám povie, kde by sa muselo začínať čítanie tak, aby sa prekrývalo s k-nadslovom v tejto k-tici. Nakoniec zlúčime všetky rovnaké začiatky vrámci jedného čítania a doplníme ich na koniec poľa začiatkov.

Keďže jedna k-tica sa môže v k-nadslove nachádzať viackrát, spomedzi všetkých jej výskytov vyberieme ten, ktorý má na konci v poli výskytov hodnotu *Pravda*.

Každej k-tici z každého čítania teraz zodpovedá práve jeden začiatok v k-nadslove, viacerým k-ticiam môže zodpovedať ten istý. V poli správnych pozícií pre nejaký začiatok z teda označíme za správne pozície iba tie, ktoré zodpovedajú nejakej k-tici, pre ktorú sme začiatok z vytvorili.

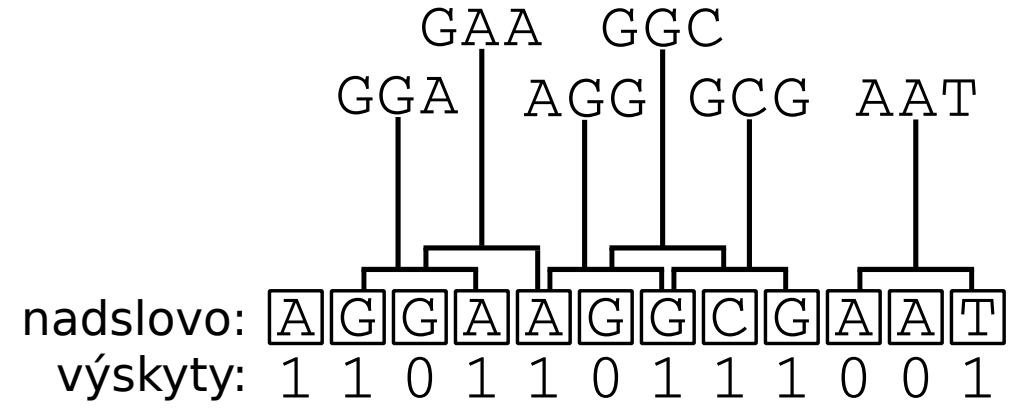
Nakoniec namiesto toho, aby sme si ukladali pre každú pozíciu z čítania samostatnú pravdivostnú hodnotu, využijeme vlastnosť, že pre každú začiatočnú pozíciu budú správne pozície v poli správnych pozícií s vysokou pravdepodobnosťou pri sebe a že každá správna pozícia je v súvislom bloku s aspoň $k-1$ ďalšími. Pre každý súvislý blok správnych hodnôt si na jeho začiatok a na políčko za jeho koncom uložíme jednu hodnotu *Pravda*. Pokiaľ máme obmedzenie, že každé čítanie má dĺžku najviac q, pre každý začiatok potrebujeme v poli správnych pozícií $q+1$ hodnôt. Konkrétny príklad je uvedený v obrázku 3.1.

Ak chceme pre i-ty začiatok z a pre nejaký úsek pozícií od x po $x+c$ v tomto začiatku zistiť, či je tento úsek správny, stačí nám overiť si dve vlastnosti:

1. $rank(i*(q+1)+x) \equiv 1 \pmod 2$
2. $rank(i*(q+1)+x) = rank(i*(q+1)+x+c)$

Prvou z týchto požiadaviek zaručíme, že pozícia x je naozaj správna, druhou zaručíme, že medzi x a $x+c$ sú tiež všetky pozície správne, keďže v nich nenastal žiaden koniec správneho úseku.

Očakávame, že takáto reprezentácia poľa správnych pozícií bude obsahovať iba málo hodnôt *Pravda*, na jej uloženie teda môžeme využiť vhodnú štruktúru pre riedke polia,



začiatok	zarovnanie	správne pozície
4	A G G C G A G G A A G G C G A A T	100001
1	G G A A T A G G A A G G C G A A T	100010
7	G G A A T A G G A A G G C G A A T	001001

Obr. 3.1: Ukážkový príklad začiatkov a zodpovedajúcich úsekov v poli správnych pozícií pre vstupné slová `ACCGC` a `GGAAT` s k-nadslovom `AGGAAGGCGAAT` a poľom výskytov `110110111001`, kde `1` reprezentuje hodnotu *Pravda*.

sdarray [12], opäť použijeme implementáciu z SDSL. Na reprezentáciu poľa dĺžky n s m hodnotami potrebujeme $m \cdot (2 + \lceil log\frac{n}{m} \rceil)$ bitov, počítanie *rank* nám zaberie nanajvýš $O(log\frac{n}{m}) + O(\frac{log^4 m}{logn})$ operácii, aj keď v praxi je táto operácia rýchlejšia.

Aby sme vedeli rýchlo vyhľadávať, zavedieme si ešte jedno pole, ktoré bude obsahovať smerníky na začiatky, usporiadané podľa pozície začiatku. Pomocou tohto poľa budeme vedieť rýchlo zistiť, ktoré začiatky za prekrývajú s nejakým úsekom pozícií v k-nadslove.

Aby sme vedeli povedať, ktorému čítaniu zodpovedá nejaký začiatok, zavedieme si nové pole pravdivostných hodnôt. Toto pole bude rovnako dlhé ako pole začiatkov a hodnotu *Pravda* bude mať pri prvom začiatku každého čítania. Dôležité je uvedomiť si, že začiatky vytvárame postupne pre prvé čítanie, potom pre druhé, atď. Pre i-ty začiatok zistíme, ktorému čítaniu zodpovedá, pomocou $rank(i)$ v tomto novom poli.

3.3.1 Riešenie dotazu

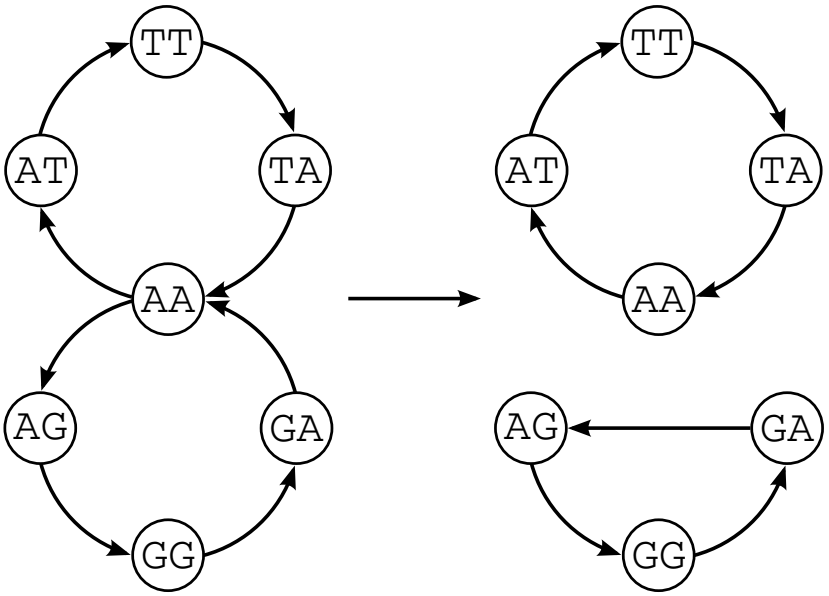
Pre hľadaný reťazec r si najprv nájdeme, kde sú jeho výskyty v k-nadslove pomocou FM-indexu. Spomedzi všetkých týchto výskytov nás budú zaujímať iba tie, ktorých koniec má v poli výskytov hodnotu *Pravda*.

Pre každý takýto výskyt binárne vyhľadáme najmenší začiatok, ktorý sa kryje s celým výskytom a overíme si v poli správnych pozícií, či nejde o falošný prekryv. Ak je prekryv naozaj pravý, pridáme čítanie, ktorému zodpovedal daný začiatok, do výsledku. Podobne ako pri počítaní výskytov budú nastávať problémy pre r kratšie než k.

Časová zložitosť tohto dotazu závisí od vyhľadávania v FM-indexe, nájdenia prvého začiatku a následného kontrolovania potenciálnych začiatkov. Keďže očakávame, že v FM-indexe bude iba málo výskytov dotazovaného reťazca a prvý potenciálny začiatok binárne vyhľadávame, väčšina času vykonávania tohto dotazu bude plynúť z kontrolovania potenciálnych začiatkov. Časová zložitosť kontrolovania potenciálnych začiatkov je lineárne závislá od ich počtu.

3.3.2 Redukcia počtu začiatkov

Veľkosť všetkých týchto pomocných štruktúr a čas vykonávania dotazu sú úzko späté s počtom začiatkov, ktoré si ukladáme. V tejto časti si popíšeme, ako budeme ešte počas vytvárania k-nadslova zmenšovať počet týchto začiatkov.



Obr. 3.2: Ukážkový príklad rozpadu grafu na komponenty pri spojení hrán.

Vo výslednom korektnom slede v k-grafe vstupnej množiny slov budeme preferovať, aby niektoré hrany boli použité priamo po sebe. Pokiaľ máme napríklad pri $k = 3$ vstupné slovo `AGGC`, budeme preferovať, aby po hrane z vrchola `(AG)` do `(GG)` nasledovala hrana z `GG` do `GC`. Tým dosiahneme, že vo výslednom k-nadslove sa bude nachádzať podreťazec `AGGC`, čiže k-tice `AGG` a `GGC` budú mať ten istý začiatok.

Keďže samotné rýchle hľadanie Eulerovského ťahu v grafe sa ťažko prispôsobuje preferovaniu nejakej následnosti hrán, zavedieme si jednoduchú operáciu spájania hrán na grafe, ktorú budeme vykonávať ešte pred hľadaním ťahu. Pokiaľ máme hrany $e_1 = (V_1, V_2)$ a $e_2 = (V_2, V_3)$ a preferujeme, aby e_2 nasledovala po e_1, môžeme obe hrany z grafu odstrániť a pridať namiesto nich *spojenú* hranu $e_3 = (V_1, V_3)$. Spojiť môžeme samozrejme aj viacero hrán dohromady. Pre každú spojenú hranu si budeme pamätať postupnost pôvodných hrán, z ktorých sa skladá.

Jednou nevýhodou takejto operácie je, že môže zvýšiť počet komponentov v grafe. Ako jednoduchý príklad môže slúžiť graf pozostávajúci z dvoch cyklov, ktoré zdieľajú jeden vrchol V. Ak v jednom z týchto cyklov prepojíme hrany prechádzajúce cez V, pôvodne súvislý graf sa rozpadne na dva komponenty. Tento príklad sme znázornili na obr. 3.2.

Spájať hrany môžeme viacerými spôsobmi, niektoré z nich môžu zaručiť, že sa nezvýši počet komponentov, niektoré môžu aj napriek zvýšeniu počtu komponentov pospájať viacero hrán dohromady. V našej implementácii sme sa rozhodli pre druhú z týchto možností, keďže predpokladáme, že nám takto vznikne pomerne málo nových komponentov a za každý rozpad sa nám k-nadslovo predĺži najviac o $k - 1$ znakov.

Základný algoritmus nadpájania hrán, ktorý používame, bude nasledovný:

1. Prejdi cez všetky vstupné slová a pre každú dvojicu hrán e_1 a e_2, ktoré môžu ísť po sebe zisti, v koľkých vstupných slovách naozaj idú po sebe.
   Túto hodnotu označíme ako skóre spojenia hrán e_1 a e_2.

2. V každom vrchole grafu nájdi párovanie hrán s najväčším súčtom skóre. Z párenia odstráň spojenia hrán s nulovým skóre.

3. Spoj všetky postupnosti hrán, ktoré vznikli v predošlom kroku.

Pri predbežných testoch tento postup naozaj počet komponentov grafu k-grafu vstupných slov zvýšil iba minimálne. Tým pádom sa nijako výrazne nezmenila dĺžka k-nadslova. Počet začiatkov však klesol zhruba na polovicu.

Kapitola 4

Výsledky a porovnanie

V tejto kapitole sa pozrieme na výkonnosť nášho riešenia v porovnaní s CR-indexom. Testovanie prebehlo na počítači s procesorom Intel Xeon CPU E5-2670 taktovaným na $2.60GHz$.

Pre účely testovania sme používali dve sady vstupných dát. Prvou sadou sú čítania získané nástrojom MiSeq z baktérie E.coli, kmeňa MG1655 získané od spoločnosti Illumina. Dĺžka genómu tejto baktérie je 4.7 miliónov bázových párov, čítania majú viac ako 180× pokrytie a chybovosť 0.75%. Druhú sadu tvoria čítania zo štrnásteho ľudského chromozómu, získané z projektu GAGE [15]. Celková dĺžka reťazca je 107 miliónov bázových párov, čítania majú 42× pokrytie a chybovosť 1.5%.

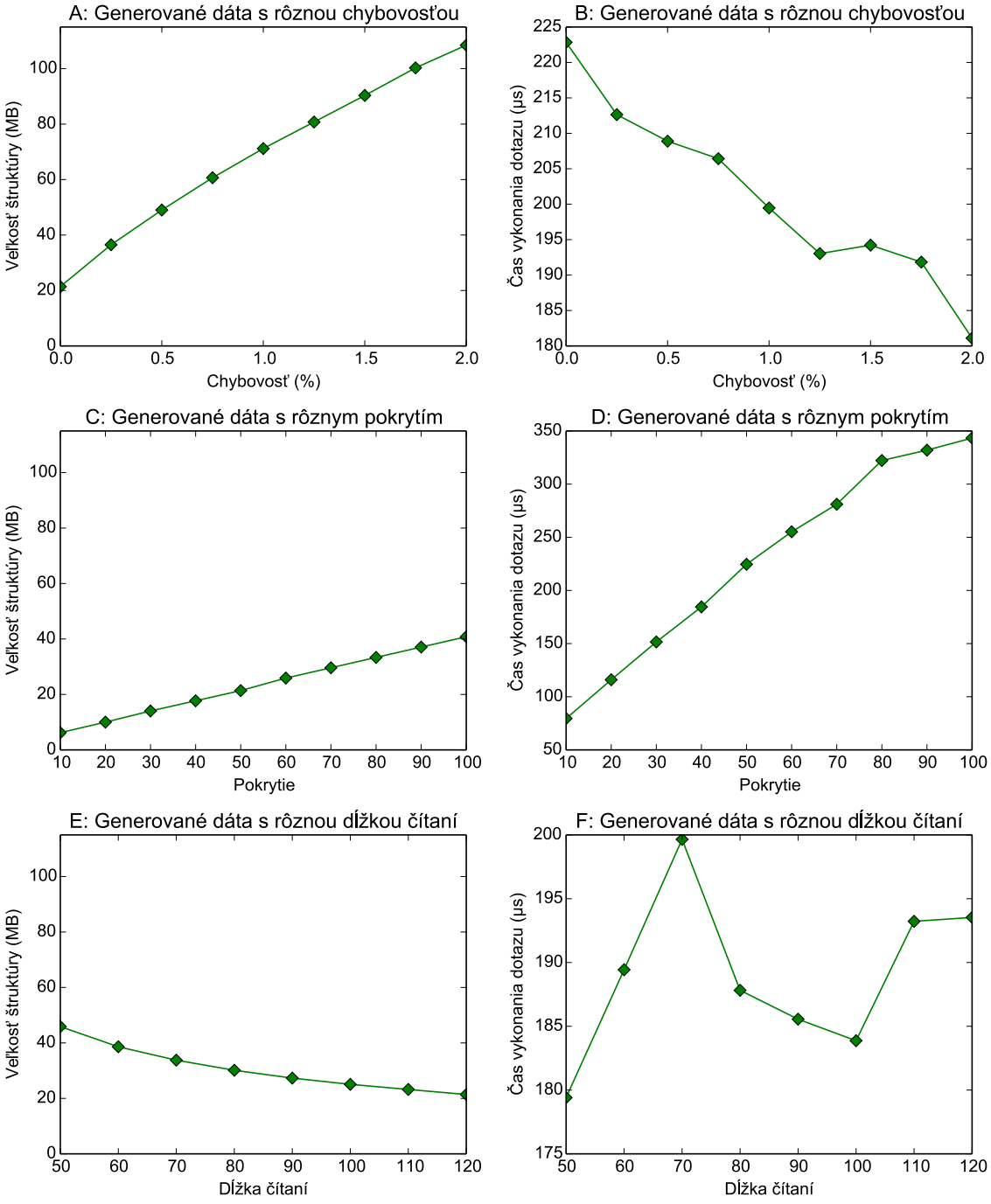
Okrem týchto dvoch sád čítaní sme použili aj simulované čítania náhodne generované z osekvenovaného genómu baktérie E.coli s rôznou dĺžkou aj chybovosťou.

Testovali sme rýchlosť hľadania čítaní s daným podreťazcom a veľkosť štruktúr, ktoré potrebujeme pre odpovedanie na takéto dotazy za rôznych podmienok. Pokiaľ nie je povedané inak, hodnota k je 20.

Podreťazce pre dotazy sme generovali tak, aby 95% podreťazcov pochádzalo z náhodných čítaní a zvyšných 5% boli úplne náhodné reťazce.

4.1 Generované dáta

Obrázok 4.1 prezentuje výsledky pre simulované dáta. Ako prvé sme generovali čítania dĺžky 120 s rôznou šancou na chybné určenie bázy (panely A a B). Čítania mali 50×



Obr. 4.1: Panely A a B: vplyv chybovosti dát na rýchlosť vykonávania dotazu a množstvo pamäte potrebnej na uloženie všetkých pomocných štruktúr. Panely C a D: vplyv pokrytia. Panely E a F: vplyv dĺžky čítaní. Pri druhej a tretej dvojici panelov boli dáta generované bez chýb.

pokrytie. Ako druhé sme generovali čítania s rôznym pokrytím bez chýb (panely C a D). Na záver sme generovali čítania s 50× pokrytím, líšiace sa v dĺžke čítaní (panely E a F).

Prvý experiment (panely A a B) ukazuje, že vyššia chybovosť spôsobuje zásadné zväčšenie celej štruktúry, ale čas na vykonanie dotazu klesá. Za nárast pamäťovej náročnosti môžu dve skutočnosti. Prvou je predĺženie k-nadslova, ktoré musí obsahovať všetky k-tice s chybou. Druhou je zväčšený počet začiatkov, ktorý je spôsobený tým, že k-tice s chybou v čítaní sú oddelené od zvyšku bez chyby. Za mierne zrýchlenie dotazov pri väčšej chybovosti je pravdepodobne zodpovedné predĺženie k-nadslova, kvôli ktorému sa zníži priemerný počet začiatkov na pozíciu v k-nadslove.

Na paneloch C a D vidíme, že vyššie pokrytie navyšuje aj pamäťové, aj časové nároky. Pri každom pokrytí bola dĺžka k-nadslova takmer rovnaká, navyšoval sa iba počet začiatkov. Z tohto dôvodu stúpal priemerný počet začiatkov na znak v k-nadslove, čím si vysvetľujeme vyšší čas potrebný na vyriešenie dotazu.

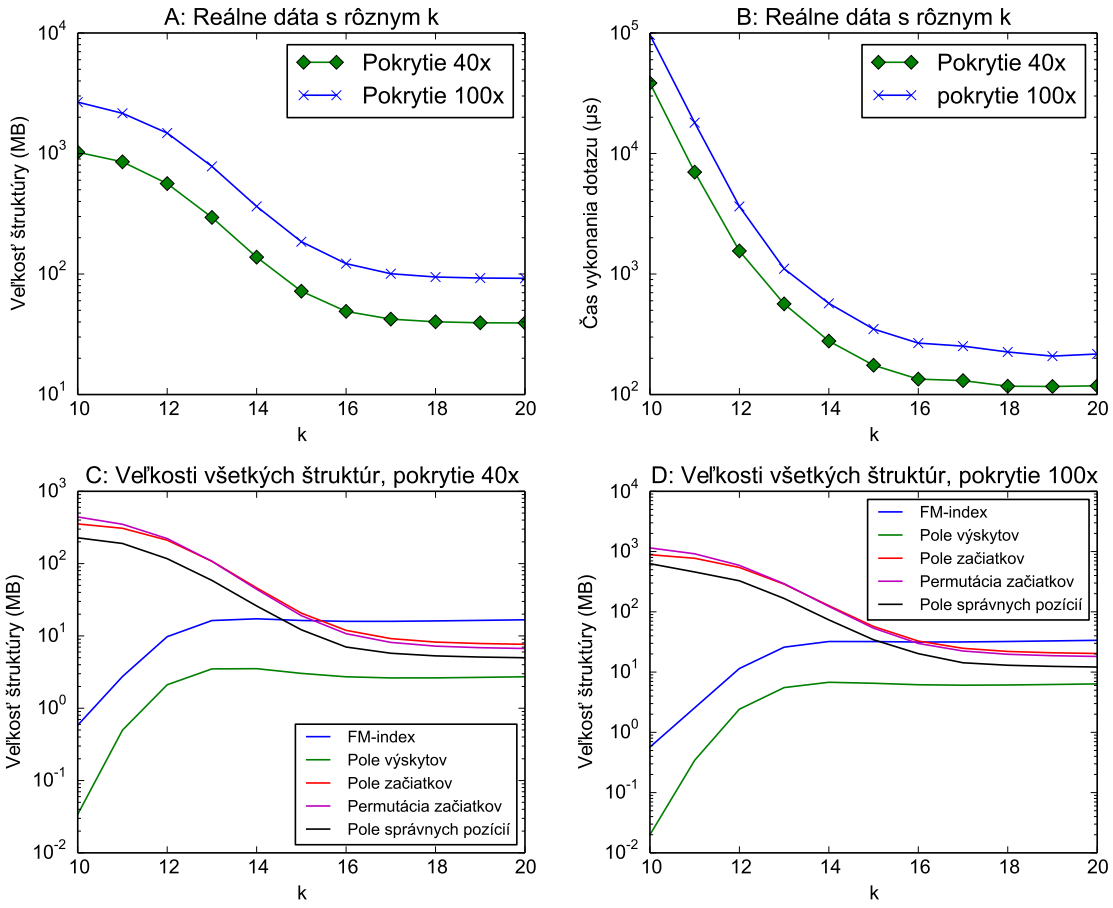
V poslednej dvojici panelov E a F vidíme, že dlhšie čítania sa našim postupom uložia na menej miesta. Toto je spôsobené tým, že sa zmenší počet začiatkov. Zmeny v čase potrebnom na riešenie dotazu sú malé a vysvetľujeme si ich ako náhodný šum.

4.2 Rôzne k

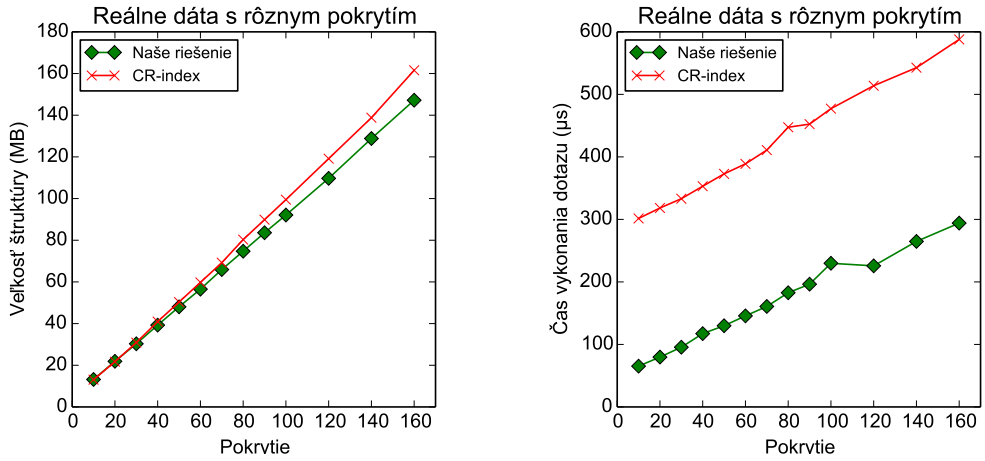
Na druhej sade testov si ukážeme, aký vplyv má hodnota k na veľkosť a rýchlosť celej štruktúry (obr. 4.2). Tentokrát sme použili podmnožinu čítaní pre baktériu *E.coli* so 40× a 100× pokrytím.

Pre malé hodnoty k máme vysokú pravdepodobnosť, že sa rôzne časti genómu budú zhodovať v k-ticiach. Zatiaľ čo dĺžka k-nadslova je rekordne malá, jednotlivé čítania sú roztrúsené po tomto k-nadslove, čo má za efekt veľmi veľký počet začiatkov.

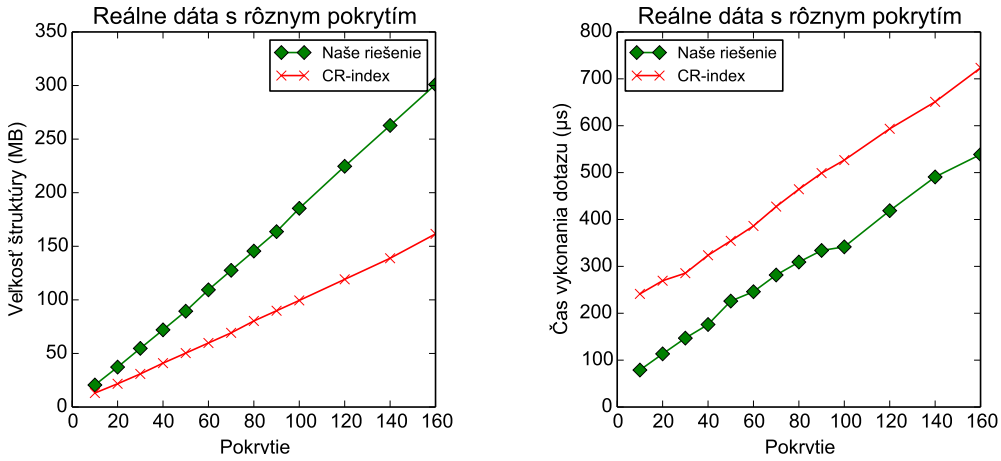
Pre rastúce k spočiatku rastie dĺžka nadslova, až sa pri $k \geq 15$ ustáli na takmer rovnakej hodnote. Zároveň môžeme pozorovať, že pre $k \geq 16$ sa takmer vôbec nemení ani počet začiatkov. Tento fakt si vysvetľujeme tak, že pre takéto k začína k-nadslovo pomerne dobre reprezentovať celý genóm *E.coli*. Rovnako ako s rastúcim k klesá počet začiatkov, klesá aj čas spracovávania dotazu, čo len potvrdzuje, že táto rýchlosť závisí od počtu skontrolovaných potenciálnych začiatkov.



Obr. 4.2: Výsledky testovania na čítaniach baktérie E.coli pre rôzne k. Panely A a B: vplyv hodnoty k na pamäťové nároky štruktúry a čas spracovávania dotazu. Panel C: veľkosti jednotlivých polí a štruktúr, ktoré používame, v závislosti od k pri 40× pokrytí. Panel D: veľkosti štruktúr pri 100× pokrytí. Kvôli veľkému rozsahu nameraných hodnôt sú veľkosti aj čas udávané v logaritmickej škále.



Obr. 4.3: Porovnanie pamäťovej a časovej náročnosti CR-indexu a nášho riešenia na čítaniach baktérie E.coli pri rôznom pokrytí.



Obr. 4.4: Porovnanie pamäťovej a časovej náročnosti CR-indexu a nášho riešenia na čítaniach baktérie E.coli pre $k = 15$ pri rôznom pokrytí.

4.3 Porovnanie s CR-indexom

Ako posledné si ukážeme porovnanie nášho riešenia s CR-indexom na obr. 4.3. Z čítaní z baktérie *E.coli* sme náhodne povyberali podmnožiny s daným pokrytím. Pri každom pokrytí používalo naše riešenie o trochu menej pamäte, napriek tomu bolo o viac než $200\mu s$ rýchlejšie. Na obr. 4.4 zas môžeme pozorovať, že pre menšie hodnoty k je síce naše riešenie rýchlejšie, ale vyžaduje zhruba dvakrát viac pamäte, než CR-index. Pre menšie k môžeme predpokladať, že CR-index bude ešte o trochu rýchlejší, naše riešenie bude ešte pomalšie kvôli väčšiemu množstvu začiatkov.

Podobne sme otestovali oba programy na čítaniach z ľudského chromozómu 14. CR-index na týchto čítaniach potreboval $1.2GB$ pamäte, naša štruktúra potrebovala až vyše $2.1GB$. Z časového hľadiska to vyšlo naopak, CR-index potreboval okolo $21ms$ na spracovanie dotazu, naša štruktúra to zvládla v priemere za $9ms$.

4.4 Zhrnutie

So zvyšujúcou sa hodnotou k veľkosť štruktúry klesá a jej rýchlosť stúpa. Očakávame však, že pre dostatočne veľké k začne byť dĺžka nadslova dominantným faktorom vo veľkosti celej štruktúry a teda celková pamäťová zložitosť začne stúpať. Jedným z možných smerov ďalšieho vývoja je podpora pre dotazy kratšie než k, vďaka ktorej sa možno budú dať využiť pamäťové a časové výhody väčšieho k aj pre kratšie dotazy.

Ďalšie pozorovanie je, že so zvyšujúcou sa chybovosťou narastá veľkosť štruktúry.

S týmto by sme sa mohli vysporiadať podobne, ako CR-index. Očakávame však, že použitím riešenia z CR-indexu stratíme náskok v rýchlosti.

Ďalšie pozorovanie je, že zatiaľ čo s malým k je dĺžka k-nadslova pomerne malá, na reprezentáciu čítaní potrebujeme veľmi veľa pamäte. Pre väčšie k sa pomery obrátia, k-nadslovo bude väčšie, zatiaľ čo začiatky budeme vedieť reprezentovať v oveľa menšej pamäti. Z tohto vyplýva, že naše riešenie je efektívne hlavne v špeciálnych prípadoch, keď je krátke k-nadslovo a počet začiatkov je malý.

Pri porovnaní s CR-indexom bolo naše riešenie pre stredne veľké k rýchlejšie, túto skutočnosť pripisujeme spôsobu, akým sa CR-index vysporadúva s chybami. Zatiaľ čo CR-index pre každý dotaz na podreťazec r hľadá veľké množstvo reťazcov, ktoré je možné získať úpravou jedného znaku, naše riešenie hľadá iba jeden.

Pre malé k je naše riešenie pomerne neefektívne, pokiaľ nás zaujímajú čítania obsahujúce dotazovaný podreťazec. Napriek tomu môže byť použiteľné aj pre takéto k, pokiaľ bude podporovať iba dotazy na počty výskytov.

Záver

V tejto práci sme sa zaoberali indexáciou potenciálne veľkej sady čítaní tak, aby bolo možné vyhodnocovať dotazy na podreťazce dĺžky presne k. Základným prístupom k problému bolo nájdenie k-nadslova k čítaniam a vytvorenie pomocných polí a štruktúr, aby sme vedeli odpovedať na dotazy o pôvodných čítaniach.

Popísali sme si spôsob, ako nájsť zaručene najkratšie k-nadslovo v polynomiálnom čase, pokiaľ je nutný podgraf k-grafu čítaní súvislý. Ďalej sme si popísali spôsob, ako vieme hľadať k-nadslovo aj v nesúvislom k-grafe, pričom toto k-nadslovo bude dlhšie od najkratšieho možného o najviac $(k-1)$-násobok počtu komponentov. Taktiež sme si popísali, ako vieme s využitím náhody nájsť nejaké o málo dlhšie nadslovo v čase lineárne závislom od veľkosti k-grafu.

Pri indexácii sme si popísali tri základné typy dotazov, na ktoré vieme s využitím k-nadslova odpovedať. Podobne ako spomenuté dotazy vieme riešiť aj zvyšné dotazy spomenuté v článku N. Philippe a kol. [13], spomenuté v prvej kapitole tejto práce:

1. Ktoré čítania obsahujú daný podreťazec f?
   Túto otázku sme riešili priamo v práci v sekcii 3.3.

2. Koľko čítaní obsahuje daný podreťazec f?
   Túto otázku vieme riešiť podobne, ako počet výskytov podreťazca v čítaniach, stačí odlišne vypočítať pole počtov výskytov.

3. Na ktorých pozíciach v čítaniach sa nachádza daný podreťazec f?
   Táto otázka sa dá zodpovedať pomocou rovnakých štruktúr, ako prvá. Pri počítaní si nebudeme ukladať iba, ktoré čítanie sme práve našli, uložíme si aj pozíciu v ňom.

4. Koľkokrát sa dohromady vyskytuje daný podreťazec f v čítaniach?
   Túto otázku sme riešili piamo v práci v sekcii 3.2.

Zvyšné dotazy sú len modifikáciou prvých štyroch, keď nás zaujímajú iba čítania,

ktoré daný podreťazec obsahujú iba raz. Takéto dotazy vieme spracovať s pomocou malých prispôsobení navrhovaných dátových štruktúr.

Testovanie našej štruktúry ukázalo, že naša štruktúra je pre väčšie k efektívnejšia než CR-index [2], zatiaľ čo pre menšie k je od neho horšia. Na dátach pochádzajúcich zo sekvenovania s väčšou chybovosťou naša štruktúra síce zaberá o niečo viac pamäte, ale dotazy vie spracovávať rýchlejšie.

V tejto práci sme neriešili chyby v čítaniach, keďže vieme odhadnúť, že každá chyba v dátach môže k-nadslovo predĺžiť najviac o $2 * k - 1$. Keďže je však naša štruktúra efektívnejšia pre väčšie k, riešenie chýb môže patriť k nevyhnutným krokom k použiteľnosti našej štruktúry pre relatívne veľké hodnoty k.

Na druhú stranu, s riešením chýb podobným ako v CR-indexe prichádzajú aj skryté úskalia. Jedným z nich je nutnosť pri každom dotaze na podreťazec r spracovať viacero reťazcov, ktoré mohli vzniknúť ako oprava nejakého čítania, ktoré pôvodne r obsahovalo. Druhým z nich je odpovedanie na otázky o počte výskytov. Keďže niektoré k-tice sú uložené v pozmenenej podobe, je nutné aj pri počítaní overovať jednotlivé čítania.

Malým problémom, ktorý sme nepreskúmali je spôsob, akým sa dá vysporiadať sa s dotazmi na podreťazce kratšie ako k. Keďže jediný problém spôsobujú začiatky čítaní, jedno možné riešenie by mohlo využívať nejako vhodne reprezentované pozície, na ktorých sa nachádza prvá k-tica z nejakého čítania.

Ďalším problémom do budúcnosti je otázka, či neexistuje ešte iný spôsob, ako sformulovanť nadslovo, s pomocou ktorého by sa dali implementovať ešte efektívnejšie štruktúry. Jedna možná formulácia je nadslovo so skóre, v ktorom si môžeme dovoliť rozdeliť čítanie na dve časti za cenu nejakého záporného skóre, do ktorého sa premietne nutnosť nejakým spôsobom uložiť informáciu o tomto rozdelení.

Literatúra

[1] Vladimír Boža, Broňa Brejová, and Tomáš Vinař. GAML: genome assembly by maximum likelihood. *Algorithms for Molecular Biology*, 10(1):1–10, 2015.

[2] Vladimír Boža, Jakub Jursa, Broňa Brejová, and Tomáš Vinař. Fishing in read collections: Memory efficient indexing for sequence assembly. In *String Processing and Information Retrieval*, pages 188–198. Springer, 2015.

[3] NG de Bruijn. A combinatorial problem. *Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen. Series A*, 49(7):758, 1946.

[4] Jack Edmonds and Ellis L Johnson. Matching, euler tours and the chinese postman. *Mathematical programming*, 5(1):88–124, 1973.

[5] Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

[6] Horst A Eiselt, Michel Gendreau, and Gilbert Laporte. Arc routing problems, part ii: The rural postman problem. *Operations research*, 43(3):399–414, 1995.

[7] Peter Elias. Universal codeword sets and representations of the integers. *Information Theory, IEEE Transactions on*, 21(2):194–203, 1975.

[8] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE, 2000.

[9] John Gallant, David Maier, and James Astorer. On finding minimal length superstrings. *Journal of Computer and System Sciences*, 20(1):50–58, 1980.

[10] Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pages 326–337, 2014.

[11] Jan Karel Lenstra and AHG Kan. On general routing problems. *Networks*, 6(3):273–280, 1976.

[12] Daisuke Okanohara and Kunihiko Sadakane. Practical entropy-compressed rank/select dictionary. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, pages 60–70. Society for Industrial and Applied Mathematics, 2007.

[13] Nicolas Philippe, Mikael Salson, Thierry Lecroq, Martine Leonard, Therese Commes, and Eric Rivals. Querying large read collections in main memory: a versatile data structure. *BMC bioinformatics*, 12(1):1, 2011.

[14] Rajeev Raman, Venkatesh Raman, and S Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 233–242. Society for Industrial and Applied Mathematics, 2002.

[15] Steven L Salzberg, Adam M Phillippy, Aleksey Zimin, Daniela Puiu, Tanja Magoc, Sergey Koren, Todd J Treangen, Michael C Schatz, Arthur L Delcher, Michael Roberts, et al. GAGE: A critical evaluation of genome assemblies and assembly algorithms. *Genome research*, 22(3):557–567, 2012.

[16] Niko Välimäki and Eric Rivals. Scalable and versatile k-mer indexing for high-throughput sequencing data. In *Bioinformatics Research and Applications*, pages 237–248. Springer, 2013.

Príloha A

Táto práca obsahuje priložené CD so zdrojovým kódom našej implementácie. Ten istý zdrojový kód je prístupný aj na adrese `https://github.com/Sameth/SRindex`.