FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Komprese souborů FASTQ
Student:	Bc. Jakub Novák
Vedoucí:	Ing. Radomír Polách
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce zimního semestru 2021/22

Pokyny pro vypracování

Nastudujte formát FASTQ [1,2] pro ukládání DNA sekvencí, které jsou výsledkem sekvenování.
Navrněte, analyzujte a implementujte kompresní algoritmus pro kompresi souborů FASTQ (SET/PET) s podporou streamování dat bez použití reference.
Implementaci proveďte jako samostatný datový formát, který bude součástí knihovny SCT dodané vedoucím práce.
Implementaci testujte na datech dodaných vedoucím práce a také srovnejte s [3, 4].

Seznam odborné literatury

[1] Illumina. FASTQ files explained. https://support.illumina.com/bulletins/2016/04/fastq-files-explained.html, 2019. [Online; accessed 21-February-2020]
[2] Wikipedia. FASTQ format. http://en.wikipedia.org/w/index.php?title=FASTQ%20format&oldid=931102164, 2020. [Online; accessed 21-February-2020]
[3] Bonfield JK, Mahoney MV (2013) Compression of FASTQ and SAM Format Sequencing Data. PLoS ONE 8(3): e59190. https://doi.org/10.1371/journal.pone.0059190[4] El Allali, A., Arshad, M. MZPAQ: a FASTQ data compression tool. Source Code Biol Med 14, 3 (2019). https://doi.org/10.1186/s13029-019-0073-5

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 28. února 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Komprese souborů FASTQ

Bc. Jakub Novák

Katedra Teoretické Informatiky
Vedoucí práce: Ing. Radomír Polách

30. července 2020

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Radomíru Poláchovi za námět k této diplomové práci a jeho podnětné připomínky. Dále děkuji Karolíně za její neocenitelnou podporu především během psaní této diplomové práce. Také děkuji své rodině rodině, která mě podporovala po dobu celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 30. července 2020 . . . . . .

Odkaz na tuto práci

Novák, Jakub. *Komprese souborů FASTQ*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato práce se zabývá analýzou struktury FASTQ souborů, výběrem vhodného kompresního algoritmu a jeho implementací. Velká část práce se zabývá popisem algoritmu mixování kontextů a jeho implementací. Dále v práci popisuji výsledky provedených testování, porovnávám je s existujícími řešeními a na jejich základě navrhuji další možná vylepšení. Výsledný kompresní algoritmus dosahuje lepšího kompresního poměru než základní komprimovací balíček GZip, který se v praxi pro ukládání FASTQ souborů běžně používá. Praktická část je provedena v programovacím jazyce Java a stala se součástí kompresní knihovny *Small Compression Toolkit*.

Klíčová slova Komprese, komprese FASTQ souborů, mixování kontextů, sekvenování DNA

Abstract

This thesis concerns with the analysis of a FASTQ file structure, selection of a suitable compression algorithm and its implementation. A large part of this thesis describes a context mixing algorithm and how to implement it. Furthermore, the results of performed tests are described, compared with existing solutions and possible enhancements are proposed. Implemented compression algorithm achieves better compression ratios than standard compression tool GZip, that is currently being used for storing of FASTQ files. The practical segment of this thesis is realized in the Java programming language a became part of a compression library called *Small Compression Toolkit*.

Keywords Compression, FASTQ file compression, context mixing, DNA sequencing

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Motivace

V posledních desítkách let došlo k rychlému rozvoji mnoha technologií využívaných k poznání o biologii člověka, ale i dalších živých organismů. Jednou z takových technologií zásadní pro vědecká odvětví zabývající se genetikou je sekvenování DNA, které je základním předpokladem nejen pro vědecký popis, ale také pro diagnostiku a v posledních letech i pro některé léčebné postupy.

Sekvenování DNA je proces určování pořadí nukleových bází v úseku DNA. V poslední době vzniklo v odvětví sekvenování mnoho moderních technologií, které se často nazývájí souhrným názvem sekvenování nové generace (Next-generation sequencing nebo NGS). Díky těmto technologiím se sekvenování DNA i RNA stalo rychlejším a levnějším. V posledních letech (počínaje přibližně rokem 2008) došlo ke snížení ceny sekvenování jednoho genomu na téměř jednu desetitisícinu [1]. To mělo za následek rozvoj studování genomiky a molekulární biologie. Spolu s tímto rozvojem vznikla poptávka po kompresi genomických dat, kterých tou dobou začalo vznikat ohromné množství. Jejich efektivní uložení tedy významně přispívá ke snížení nákladů ve dříve zmíněných odvětvích.

Zprocesovaná genomická data jsou v současné době standardně ukládána do souborů typu FASTQ [2]. Jedná se o textový formát zaobalující sekvence DNA ve formátu FASTA [3] a kvalitativní skóry, které sekvenovací přístroje přiřazují jednotlivým přečteným nukleovým bazím. Tyto soubory mohou dosahovat i několika desítek gigabytů. Přesto, že mají soubory FASTQ velmi specifickou a dobře popsanou strukturu, je stále běžnou praxí k jejich kompresi používat základní komprimovací balíčky jako je například GNU ZIP formát [4].

Ve své práci jsem se proto pokusil co možná nejlépe využít nejen této specifické struktury souborů FASTQ, ale také některých vlastností skladby samotné DNA.

Hlavní cíle a požadavky

Zadání této diplomové práce zahrnuje nastudování formátu FASTQ, který je v současné době standardem pro ukládání genomických dat, a na základě získaných poznatků a analýzy existujících řešení pro tyto data navrhnout a implementovat vhodný kompresní algoritmus. Při volbě navrhování dat ve své práci vycházím nejen ze specifikace formátu FASTQ, ale také z vlastností DNA sekvencí, konkrétně stavbou a pořadím aminokyselin, ze kterých se DNA skládá.

Dalším požadavkem na práci je provedení implementace všech potřebných algoritmů a struktur pro kompresi FASTQ do knihovny SCT *(Small Compression Toolkit)* [5] a rozšířit tak tuto knihovnu o možnost komprimování genomických dat. Jedná se o open-source knihovnu napsanou v programovacím jazyce Java, která se momentálně nachází v repositáři dostupném na fakultním gitlab serveru [6]. Za nefunční požadavek na tuto práci lze proto také považovat zachování programovacích stylů a struktur použitých napříč touto knihovnou.

V neposlední řadě je výsledný algoritmus nutno řádně otestovat. K tomuto účelu jsem od vedoucího práce dostal vzdálený přístup na virtuální počítač s velkým množstvím testovacích souborů. Dále jsem také obdržel odkazy na vědecké články zabývající se stejnou problematikou, se kterými jsem mohl své výsledky porovnat.

Související práce

Tato práce navazuje v první řadě na bakalářské i diplomové práce mnohých studentů FIT ČVUT, kteří se podobně jako já podíleli na vzniku knihovny SCT. Velkou část práce na ní zanechal především Jiří Bičan a to v rámci své diplomové práce *Implementace vylepšení kompresní metody ACB v jazyce Java* [7]. Při své práci totiž položil základy této knihovny a vytvořil mechanismy, které zjednodušují přidávání nových kompresních algoritmů a zároveň tento postup sjednocují.

Tematikou komprese genomických dat ve formátu FASTQ se zabývali například Achraf El Allali a Mariam Arshad, kteří své poznatky publikovali v časopise Source Code for Biology and Medicine. Ve své publikaci s názvem *MZPAQ: a FASTQ data compression tool* [8] analyzovali řadu kompresních algoritmů a na základě kompresního poměru, rychlosti komprese a množství použité paměti sestavili vlastní algoritmus, který nazvali MZPAQ. Z jejich výsledků jsem vycházel při volbě vhodných a algoritmů a postupů.

Stejným tématem se zabývali také James K. Bonfield, Matthew V. Mahoney a Michael Gormley v publikaci s názvem *Compression of FASTQ and SAM Format Sequencing Data* [9]. V tomto článku porovnávájí algoritmy přihlášené

do soutěže *SequenceSqueeze*, jejíž cílem bylo najít vhodný způsob komprese FASTQ souborů.

Kapitola 1

Definice pojmů a zkratek

V této kapitole jsou vysvětleny důležité pojmy a zkratky z oblasti datové komprese a sekvenování DNA, které se napříč touto prací vyskytují. V případě, že je možné nějaký výraz vyložit více způsoby, je v rámci této práce použit ve smyslu popsaném v této části.

Definice 1.0.1. Komprese
Komprese je proces konvertování vstupního datového proudu na proud výstupní (zkomprimovaný), s cílem vstupní proud zmenšit.

Definice 1.0.2. Dekomprese
Dekomprese je proces inverzní ke kompresi. Cílem dekomprese je rekonstrukce komprimovaných dat do původní podoby.

Definice 1.0.3. Kompresní poměr (CR)
Kompresní poměr je podíl mezi velikostí zkomprimovaných dat a dat původních. Pokud během komprese dojde ke zmenšení původních dat, je kompresní poměr reálné číslo mezi 0.0 a 1.0. Pokud kompresní poměr přesáhne hranici 1.0, mluvíme o negativní kompresi. Jeho hodnotu můžeme vyjádřit následující rovnicí:

$$\mathrm{CR} = \frac{\text{velikost po kompresi}}{\text{původní velikost}}. \tag{1.1}$$

Definice 1.0.4. Adaptivní metody
Adaptivní kompresní metody vytvářejí model dat potřebný při kompresi v závislosti na komprimovaných datech. Tento model není třeba ukládat společně se zakomprimovanými daty. Při dekompresi si dekodér vytváří stejný model dat jako kodér při kompresi [10].

Definice 1.0.5. Slovníkové kompresní metody
Jedná se o kompresní algoritmy, které fungují na principu ukládání frází do slovníku. Pokud se fráze vyskytne v datovém proudu opakovaně, stačí ji reprezenovat pouze odkazem na část slovníku, kde se nachází.

Definice 1.0.6. Sekvenování DNA
Sekvenování DNA je zjišťování posloupnosti nukleových bází (A, C, G, T), ze kterých se skládá molekula DNA.

Definice 1.0.7. Model
Modelem je v této diplomové práci myšlen algoritmus, který spravuje datové struktury obsahující statistické informace o symbolech nebo bitech a provadí nad nimy výpočty pro určení pravděpodobnosti následujících symbolů nebo bitů.

Definice 1.0.8. Frekvenční tabulka
Frekvenční tabulkou je v této diplomové práci myšlena datová struktura, která ukládá frekvence symbolů v různých kontextech. Nějaká forma frekvenční tabulky je použita ve všech modelech v této práci.

Kapitola 2

Specifikace FASTQ

2.1 Proces vzniku FASTQ souboru

V závislosti na sekvenovací platformě lze pomocí chemické syntézy (technologie zvaná sequencing by synthesis, zkráceně SBS) sekvenovat miliony až miliardy klastrů na jedné flowcell. Zjednodušeně se jedná o skleněná sklíčka vybavená malými kanálky pro proudění tekutiny. Kromě toho, že tato technologie umožňuje zpracovávat velké množství klastrů naráz, došlo s jejím rozvojem i ke zpřesnění výstupu sekvenovacích strojů. Během zpracování klastrů jsou pro jednotlivé klastry vytvářeny a ukládány soubory BCL (podle slovního spojení base call) obsahující výstup softwarového analyzátoru [11].

Po skončení tohoto procesu je potřeba převést soubory BCL na sekvenovací data pomocí FASTQ konverze. Před jejím provedením dochází také k filtraci, která vyřazuje nevhodné a poškozené vzorky. Informace o úspěšně zpracovaných vzorcích jsou následně zapsány do FASTQ souboru [11].

2.2 Dominantní sekvenovací technologie

Pro sekvenování se používají především dva druhy technologií. Takzvané Short-read single-end tag technologie (SET) a Long-read paired-end tag technologie (PET).

Technologie SET jsou jednodušší a umožňují sekvenovat kratší úseky DNA (od 25 do přibližně 300 bází) ve velkém množství a krátkém čase [12]. Paired-end sekvenování je modernější přístup využívaný next-generation sequencing (NGS) systémy, ve kterém se zpracovávaný vzorek sekvenuje oběma směry a následně se obě čtení zarovnají k sobě, čím vzniknou mnohem kvalitnější data. Tento přístup kromě vyšší kvality čtení také zpřesňuje zarovnávání různých čtení k sobě a umožňuje detekci běžných změn v uspořádání DNA, jako jsou vložení (insertions), vymazání (deletions) a inverze (inversions), což není s použitím SET technologie možné [13]. Zároveň je díky metodě zvané *de novo* možné dosáhnout až dvojnásobného počtu sekvenovaných bazí [14].

2.3 Popis formátu

FASTQ soubory obsahují informace o sekvenovaných genomických datech v textovém formátu. Mezi tyto informace patří vždy identifikátor zpracované sekvence, sekvence samotná, oddělovač a skór kvality čtení pro každou nukleovou bázi. Každá tato informace je zapsaná na jeden řádek. Jednotlivá čtení je možné skládat jednoduše za sebe a vytvořit tak posloupnost téměř libovolného množství genomických dat.

Každé jednotlivé čtení je v souboru zapsáno následujícím způsobem:

- **Identifikátor sekvence**
  - Obvykle obsahuje informaci o které čtení se jedná a z kterého klastru vzorek pochází.
  - Konkrétní struktura je závislá na platformě a softwaru použitém při FASTQ konverzi.
  - Vždy začíná symbolem @, dále může obsahovat znaky A–Z, a–z, 0–9, _, ., :, – a mezera.
- **DNA Sekvence**
  - Obsahuje přečtenou sekvenci DNA tvořenou nukleovými bázemi.
  - Skládá se ze symbolů A, C, G, T a N, označujícícch nukleové báze Adenin, Cytosin, Guanin a Thymin. Symbol N se v řetezci objeví v tom případě, kdy sekvenovací software nebyl schopný rozeznat příslušnou bázi. K tomuto jevu dochází zejména na začátku a konci čtení sekvenovaného vzorku, kde je zpracování dat na okraji flowcell výrazně náročnější a tím pádem náchylnější k chybám.
- **Oddělovač**
  - Typicky se jedná pouze o symbol +. Ve vyjímečných případech následuje za ním následuje identifikátor skórů kvality, který je ale vždy shodný s identifikátorem korespondující sekvence. Proto se téměř vždy vynechává.
- **Skóry kvality**
  - Skóry sekvenovací kvality – také nazývaný Phred nebo Q skór – měří pravděpodobnost chybného určení nukleonové báze sekvenovacím přístrojem. Takový skór je přiřazen každé jednotlivé vyhodnocené bázi, tedy délka tohoto řetězce vždy odpovídá délce řetězce sekvenovaných bází.
  - Pro pravděpobonost chyby P dostaneme skór kvality Q jako:

$$Q = -10 \log_{10}(P)$$

– Hodnota skóru Q je ve FASTQ souboru zakódována pomocí ASCII kódování a to tak, že se vybere symbol ASCII odpovídající hodnotě Q+33. Toto kódování zaručuje, že jsou všechny Q skóry zakódované pomocí 1 bytu.
– Q může nabývat hodnot od 0 do 42. V ASCII kódování tedy může být reprezentován symboly [! – K].

```
@HISEQ_HU01:89:H7YRLADXX:1:1101:1943:2146 1:N:0:ATCACG
GTCTCCCACCTGTAATCCTGGCACTTTGGGAGGCCAAGGCGGGCAGATTGCCTGAGATCAGGAGTTCAAGACCAGTCTGGCCAACATGGTGAAACCCCATC
+
?8=DADDDDF;DAEFEAF<CGFEFFB<FFFA??F=FFI1BGFFDCCCABA>A3>(9A:5:5>@8+9:@A>@@<384:@:34<BB<@9@@4>@:AABBB7?9
```

Obrázek 2.1: Ukázka sekvenované DNA ve FASTQ formátu

Na obrázku 2.1 je možné vidět záznam jedné DNA sekvence zapsaný ve formátu FASTQ. Toto sekvenovaní bylo provedeno sekvenovacím strojem z řady HiSeq od společnosti Illumina, která je v současnosti dominantní platformou v oblasti NGS technologií [15].

Kapitola 3

Volba kompresních algoritmů

V této kapitole se zaměřím na volbu kompresních algoritmů a na její zdůvodnění. Velká část těchto rozhodnutí vzešla z osobních konzultací s vedoucím práce a souvisí také s dříve zmíněným výzkumem Achrafa El Allaliho a Mariam Arshad, zejména z jejich článku *MZPAQ: a FASTQ data compression tool* [8]. Ti ve své práci došli k závěru, že k dosažení lepšího kompresního poměru napomáhá předzpracovat data takovým způsobem, aby se identifikátory sekvencí, sekvence samotné i skóry kvality komprimovaly samostatně, specializovanými algoritmy.

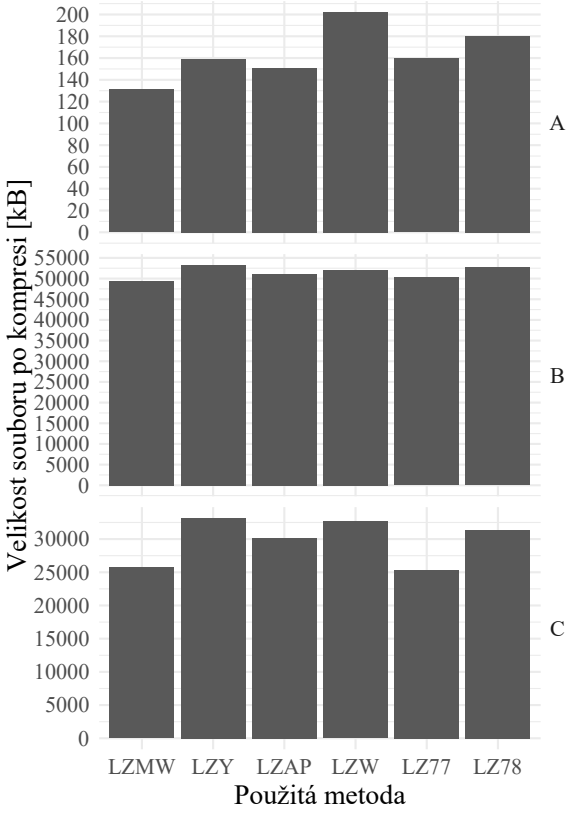
Na tomto poznatku jsem předzpracování dat založil a tedy je mnou navržená metoda dělí na tři výše zmínené streamy. Teoreticky možný čtvrtý stream, obsahující symbol '+' a eventuální komentář shodný s identifikátorem, je možné při kompresi ignorovat, jelikož jeho obsah lze při dekompresi znovu vygenerovat [16].

3.1 Identifikátory

Pro první stream, obsahující identifikátory sekvencí, jsem se rozhodl použít některou z metod z rodiny algoritmů LZ. Jedná se o slovníkové metody (def. 1.0.5) jejichž základy položili Abraham Lempel a Jakob Ziv. V současné době existuje již mnoho variant a vylepšení k původním LZ77 a LZ78 metodám (například LZW, LZMA, LZMW a další), které jsou využívány například v grafickém formátu GIF (LZW) nebo 7-Zip (LZMA).

K tomuto rozhodnutí mě dovedlo především pozorování, že v rámci souboru, který obsahuje sérii čtení pořízenou na jednom sekvenovacím stroji, se identifikátory v jednotlivých čtení mění pouze minimálním způsobem. Většina identifikátoru (obsahující informace o čtecím přístoji, klastru, flowcell, nebo použité technologii) zůstává shodná pro stovky po sobě jdoucích záznamů. Tato úvaha se ukázala být správnou podle výsledcích výše zmíněné studie, kde porovnávali například algortimy gzip (LZ77 + Huffmanovo kódování) nebo LZMA, které spadají do výše zmíněné rodiny algoritmů.

Podobným experimentem jsem se rozhodl vybrat kompresi identifikátorů i já a využil k tomu LZ metody, které už knihovna SCT obsahovala. Vzal jsem proto 3 různé soubory FASTQ, data rozdělil na stream identifikátorů, sekvencí a skórů kvality a na streamu identifikátorů jsem tyto algoritmy otestoval.

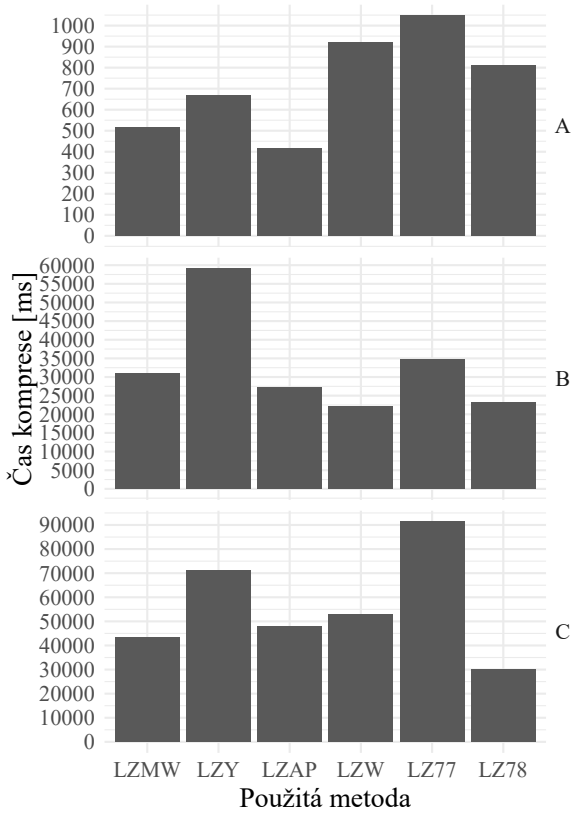


Obrázek 3.1: Metody LZ - komprese

Tabulka 3.1: Velikosti identifikátorů v testovaných souborech

Soubor	Velikost souboru	Velikost identifikátorů
A	6 539 KB	1 389 KB
B	1 442 165 KB	305 161 KB
C	110 032 210 KB	337 331 KB

Na grafech 3.1 a 3.2 můžeme vidět porovnání LZ metod z knihovny SCT na identifikátorech sekvencí třech různých FASTQ souborů. Tedy v rámci předzprocesování jsem identifikátory oddělil a měření prováděl pouze na této části dat. Soubor A je z malého testovacího datasetu sloužícího k testování FASTQ a BAM souborů od Hartwig Medical Foundation [17]. Soubor B je ob-



Obrázek 3.2: Metody LZ - rychlost

sahuje informace ze čtení genomu mořské řasy (*Zostera marina*), které provedl přístroj Illumina HiSeq 2000. Soubor pochází z archivu European Nucleotide Archive [18]. Poslední a největší testovaný soubor C je z dat poskytnutých vedoucím práce. Další informace o těchto souborech v původním stavu před kompresí lze vyčíst z tabulky 3.1. U souboru C si můžeme všimnout, že originální velikost identifikátoru je nepoměrně nižší v porovnání s původní velikostí souborů. Je to tím, že pro urychlení testování jsem ze souboru C použil pouze 5 milionů záznamů (tedy 5 milionů různých identifikátorů). Toto urychlení nemá vliv na výsledek testování použitých metod, jelikož identifikátory udržují v rámci celého souboru stejnou strukturu a také z důvodu, že soubor musí být tak jako tak zpracováván po blocích, jelikož není možné ho celý načíst do paměti.

Z této analázy 3.1 a 3.2 jsem došel k závěru, že nejlepší komprese dosahovala metoda LZMW, kterou do knihovny implementoval v rámci své bakalářské práce *Implementace kompresních metod LZY, LZMW a LZAP v jazyce Java* Ján Bobot [19]. Na souborech A, B, C dosáhla kompresního poměru (def. 1.0.3) postupně přibližně 0.0943, 0.1620 a 0.0763, čímž překonala ostatní

použité metody. Z hlediska rychlosti patřila také mezi nejlepší. Je však důležité upozornit, že výsledky algoritmů mohou být ovlivněny kvalitou implementace, která nemusí být v rámci celé knihovny a všech algoritmů shodná. Je tedy možné, že by se při řádné implementaci v oficiální knihovně výsledky těchto metod mohly lišit.

3.2 Sekvence a Q skóry

Sekvence a skóry kvality tvoří převážnou část souboru FASTQ. Se zlepšujícími se technologiemi, které jsou každým rokem schopné v rámci jednoho čtení zpracovat delší a delší sekvence, můžeme předpokládat že se poměr objemu sekvencí (a kvality skórů) oproti identifikátorům bude stále zvětšovat. Problém komprese FASTQ souboru je tedy do velké míry závislý na tom, jak dobré komprese jsme schopni dosáhnout pro tyto dva datové streamy.

Achraf El Allali a Mariam Arshad se touto problematikou ve svém článku [8] zabývali také. Ve své analýze ukázali jako vhodné kandidáty pro kompresi algoritmy využívající principu míchání/mixování kontextů (context mixing) a v diskuzi poukázali na to, že tento druh kompresních algoritmů se pro oblast genomických dat jeví velmi slibně.

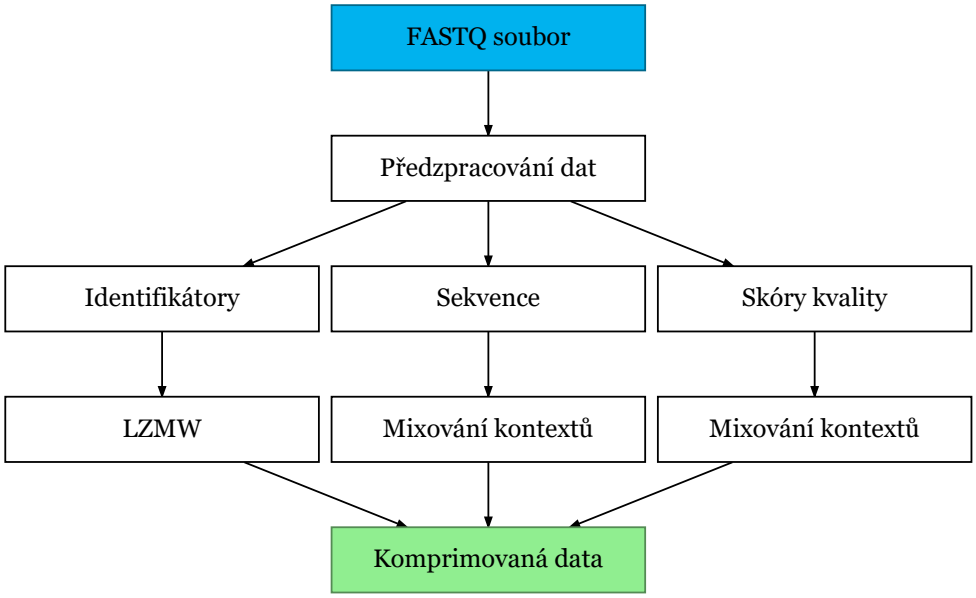
Mixování kontextů je způsob komprimování dat, při kterém následující symbol predikujeme zkombinováním výstupů dvou nebo více statistických modelů. Tento přístup může být obzvlášť výhodný za situace, kdy předem známe data, které budeme komprimovat, a můžeme díky tomu navrhnout statistické modely, které jsou schopné je co nejlépe popsat.

Knihovna SCT doposud žádnou metodu na principu mixování kontextů neobsahovala a proto jsme se s vedoucím práce rozhodli, že by bylo vhodné knihovnu o takovou metodu rozšířit. Zbytek teoretické části této práce a také její realizace se proto bude týkat navrhování modelů, kombinování jejich výsledků a dalších témat z oblasti komprese mixováním kontextů.

3.3 Finální návrh komprese

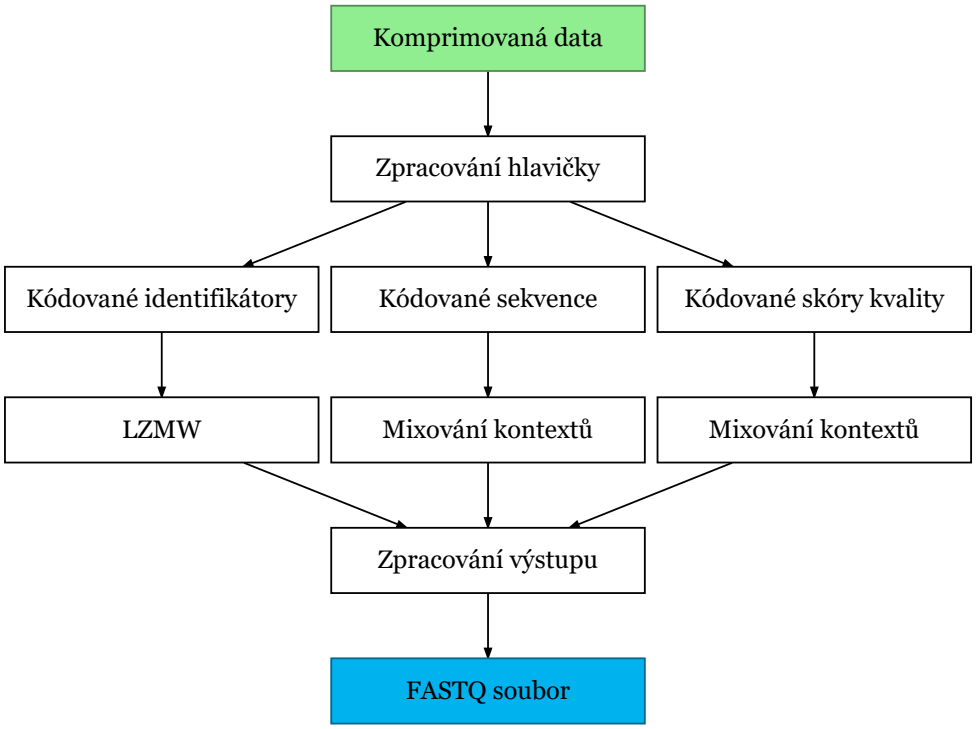
Na základě analýzy FASTQ souborů, existujícího výzkumu a vlastního porovnání dat jsem tedy dospěl k návrhu komprese, který je zobrazen na diagramu 3.3. FASTQ soubor vstupující do komprese je předzpracován tak, že se rozdělí na tři datové proudy – Identifikátory čtení, sekvence DNA a skóry kvality. Čtvrtý proud, který typicky obsahuje pouze symbol '+', se ignoruje, jelikož je možné ho při dekompresi zcela zrekonstruovat bez nutnosti jeho kódování (například použitím přepínače, který specifikuje, jakým způsobem oddělovač vygenerovat). Datový proud obsahující identifikátory je zkomprimován pomocí metody LZMW, zbylé dva proudy jsou zkomprimovány metodou mixování kontextů (každý jinou instancí). Jak je z diagramu 3.3 patrné, jsou všechny proudy navzájem nezávislé a je proto možné zpracovávat

je nezávisle na sobě (paralelně). Po dokončení komprese se výstupní data jednotlivých proudu zapíší za sebe na výstup.



Obrázek 3.3: Návrh komprese FASTQ souborů

Dekomprese, zobrazená na diagramu 3.4, funguje velmi podobným způsobem. Z komprimovaných dat se načtou postupně zakódované identifikátory, sekvence a skóry kvality. Takto rozdělené se pošlou odpovídajícím dekompresním algoritmům, které je opět dekódují. Na závěr je potřeba z dekódovaných proudů opět zkompletovat záznamy o jednotlivých čteních a vygenerovat oddělovač mezi sekvence a skóry kvality. Dekódované a zkompletované záznamy se zapíší na výstup.



Obrázek 3.4: Návrh dekomprese FASTQ souborů

Kapitola 4

Mixování kontextů

Míchání nebo mixování kontextů (anglicky context-mixing) je – spíše než konkrétní metoda – přístup k datové kompresi, při kterém se kvalitní komprese snažíme dosáhnout tím, že následující symbol nebo bit predikujeme zkombinováním výstupů dvou nebo více statistických modelů (def. 1.0.7). Často se totiž ukazuje, že predikce vzniklá kombinací výstupu několika typicky slabších modelů může být kvalitnější, než predikce jednoho silného modelu. Podobnou myšlenku můžeme pozorovat také v oblasti strojového učení u takzvaných ensemble learning metod, kde je výstup algoritmu získán jako kombinace výstupu více nezávislých modelů.

Pod výrazem kontext si můžeme představit téměř libovolnou vlastnost již zpracovaných dat, kterou lze využít pro predikci následujících bitů. Typ kontextu je vhodné volit v závislosti na typu dat, které se chystáme komprimovat. Pro text lze jako kontext použít například N předchozích symbolů (takzvané n-order modely), pro obrázky ale můžeme jako kontext použít třeba 2D matici okolních pixelů, a tak dále. Kvalita kompresního poměru je tedy přímo závislá na kvalitě kontextových modelů, které v algoritmu použijeme.

Postup algoritmu během komprese lze shrnout přibližně takto. Ze vstupu se načte symbol, který chceme zkomprimovat, a ten se předá statistickým modelům. Všechny modely nejprve na základě aktuálního kontextu připraví svoji predikci a teprve poté aktualizují svůj kontext novým symbolem (V opačném případě by nebylo možné zpětné dekódování). Například v situaci, kdy modely predikují následující bit, může být výstup každého modelu reálné číslo P_0 reprezentující pravděpodobnost, že následující bit bude roven 0. V dalším kroku je potřeba výstupy všech modelů vzít a zkombinovat do jedné celkové predikce.

Tím se dostáváme k základní otázce, jakým způsobem lze tyto predikce kombinovat. Předpokládejme, že máme k dispozici predikce dvou modelů, $p_1 = P(y = 0|A)$ a $p_2 = P(y = 0|B)$. Jedná se o pravděpodobnosti, že predikovaný bit y má hodnotu 0 při kontextech A a B. Dále předpokládejme, že oba tyto kontexty se již vyskytly v dostatečném množství tak, aby oba modely

byly schopné vytvářet spolehlivé predikce, ale zatím nikdy se tyto kontexty nevyskytly společně. Jak tedy můžeme určit $p = P(y = 0|A, B)$? Podle teorie pravděpodobnosti na tuto otázku neexistuje odpověď. Snadno lze vytvořit situace, kdy oba modely predikují jeden bit, ale skutečnost je jiná. V reálné situaci však dává smysl pravděpodobnosti získané od jednotlivých modelů nějakým způsobem zprůměrovat [20].

Při kombinování predikcí do výsledné pravděpodobnosti je také důležité brát v potaz jistotu predikce jednotlivých modelů a také jejich dosavadní přesnost. Pokud si je jeden model svou predikcí jistý na 99 % a druhý pouze na 60 %, chceme dát větší význam predikci prvního z nich. Pokud ale predikce prvního modelu odpovídaly realitě pouze ve třetině případů, zatímco druhý model predikoval správně ve většině situací, měly by jsme tyto jevy do výsledné predikce také nějak zohlednit. Jako možné řešení této situace lze použít například metody lineárního (linear mixing) nebo logistického (logistic mixing) míchání. Obě zmíněné metody přiřazují a v průběhu komprese/dekomprese upravují váhy všech modelů a tím pomáhají získat co nejlepší kombinovanou predikci.

Kvalita kombinované predikce přímo ovlivňuje kvalitu komprese a proto je důležité, aby byla co nejlepší. Je totiž společně s komprimovaným bitem předána jako vstup algoritmu aritmetického kódování. Zjednodušenně řečeno, aritmetické kodóvaní – podrobněji popsané v sekci 4.4 – používá tyto informace k zužování pracovního intervalu. Čím větší pravděpodobnost dává kombinovaná predikce komprimovanému bitu, tím méně je potřeba pracovní interval a tím méně často je potom nutné zapsat na data výstup. Svým způsobem tak můžeme říct, že kompresní poměr metody mixování kontextů je přímo úměrný jakési nejistotě nebo nepřesnosti použitých modelů a jejich kombinované predikce.

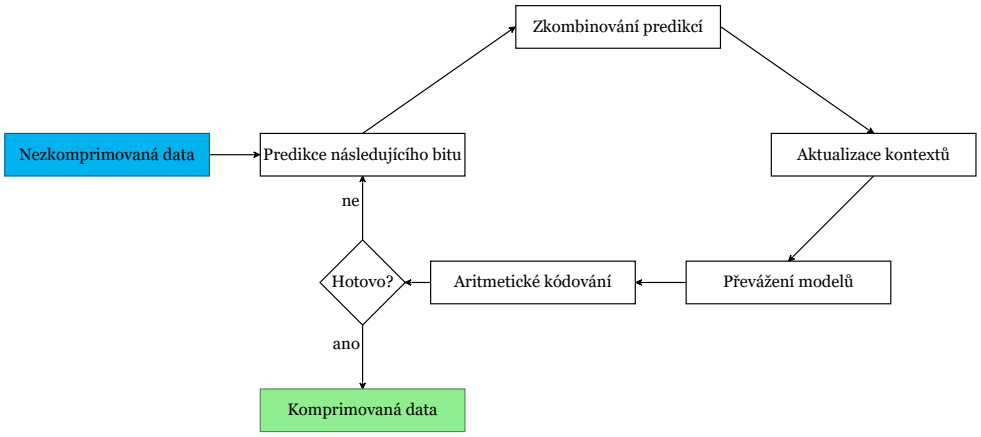
Na obrázku 4.1 je pro lepší představu možné vidět diagram popisující průběh komprese touto metodou.

4.1 Existující implementace

Průkopníkem v oblasti míchání kontextů je zcela jistě Matthew V. Mahoney a jeho algoritmus PAQ, jehož první verzi PAQ1 autor představil ve článku *The PAQ1 Data Compression Program* [21] v roce 2002. Tento algoritmus poté sám několikrát vylepšil. K rychlému vylepšování algoritmu přispělo i to, že byl od počátku vydán licencí GNU General Public License.

S několika verzemi tohoto algoritmu Mahoney získal různá ocenění. V roce 2004 vyhrál Calgary Challenge s algoritmem PAQ6 a v roce 2006 Hutter prize s algoritmem PAQ8HP5.

Původní verze algoritmu pracovaly s jednoduchými modely s pevně stanovenou váhou. V novějších verzích potom přibyly složitější modely, došlo ke zlepšení kombinování predikcí nebo třeba aktualizování váh modelů po-



Obrázek 4.1: Diagram komprese metodou mixování kontextů

mocí adaptivního algoritmu. Tyto vylepšení uvedl v roce 2005 ve článku [22]. Od verze PAQ7 se pro kombinování výsledků modelů začaly póužívat neuronové sítě a došlo také k výraznému zrychlení komprese i dekomprese. Algoritmy PAQ cílí především na kvalitní kompresi, proto což si však nese daň na paměťové a hlavně časové složitosti, kterou způsobuje především množství a složitost použitých modelů.

4.2 Statistické modely

Statistické modely (def. 1.0.7) jsou základní stavební jednotkou metody mixování kontextů. Slouží k odhadování rozdělení pravděpodobnosti zpracovávaného vstupu pro kompresní algoritmus a to tak, že na základě kontextu vytvářejí predikce na následující bit nebo znak.

Na základě použitého kontextu můžeme modely rozdělit do dvou hlavních kategorií. Modely, jejichž kontext je statický, a modely adaptivní, které ho upravují v závislosti na vstupních datech.

U statických modelů je rozdělení pravděpodobnosti neměnné v celém průběhu komprese či dekomprese. Toho lze dosáhnout tak, že se datové struktury pro model vytvoří v prvním průchodu vstupním souborem, ještě předtím, než se začne s kompresí. Tento model je poté však nutné uložit společně s komprimovanými daty, protože informace z něj jsou potřeba i pro dekompresor. Dále také existuje typ statických modelů, které jsou zcela nezávislé na vstupních datech. Příkladem může být model, který používá pravděpodobnostní rozdělení písmen v nějakém jazyce. Představme si situaci, kdy komprimujeme anglický text a máme statický model, který používá kontext délky (nebo řádu) 1. Dále předpokládejme, že poslední zpracovaný symbol na vstupu bylo písmeno *e*. Potom se model podívá do své datové struktury na pravděpodobnosti všech symbolů následujících po písmeni *e* a vytvoří odpovídající pravděpodobnostní

rozdělení. Při každém nálezu písmene e vytvoří tento model pro následující znak vždy stejnou predikci. Hlavní výhodou tohoto druhého typu statických modelů je, že je není nutné ukládat společně s komprimovanými daty.

Naopak u adaptivních modelů se datová struktura modelu vyvíjí dynamicky při zpracovávání vstupního souboru. Model se tedy na datech učí a přizpůsobuje své chování na základě přečtených znaků. Vzhledem k tomu, že následující symbol při kompresi model predikuje předtím, než si s ním aktualizuje kontext, je umožněno, aby dekompresor replikoval toto chování a mohl při dekompresi pracovat vždy s totožným kontextem, jako měl ve stejné situaci během komprese. Tento přístup má oproti statickým modelům mnoho výhod, protože stačí pouze jeden průchod souborem a zároveň můžeme používat model, který se učil na datech která komprimujeme, bez toho, aby jsme ho následně museli uložit společně z komprimovanými daty.

4.2.1 Modely n-tého řádu

První typ statistického modelu, který jsem implementoval patří do kategorie modelů s kontextem n-tého řádu. Jak název naznačuje, jedná se o modely kde kontext odpovídá posledním n zpracovaným bitům (může se jednat také o n posledních symbolů, ale v rámci této práce se zaměřím na modely pracující s bity). Při vytváření predikce se model podívá do frekvenční tabulky (def. 1.0.8) a pro aktuální kontext v ní nalezne frekvence nul a jedniček. Na základě této informace model jednoduše určí s jakou pravděpodobností bude následující bit nula nebo jedna. Po vytvoření predikce model inkrementuje patřičnou frekvenci v tabulce podle reálné hodnoty bitu, kterou model predikoval.

Ukažme si práci tohoto algoritmu na jednoduchém příkladu 4.2.1. Mějme model druhého řádu a na vstupu posloupnost bitů 11110110. Zpracování vstupu modelem by potom vypadalo následovně:

Příklad 4.2.1. (Ukázka fungování modelu druhého řádu)

kontext = ε	input = **1**1110110	predikce = (0.5, 0.5)
kontext = 1	input = **1**110110	predikce = (0.5, 0.5)
kontext = 11	input = **1**10110	predikce = (0.5, 0.5)
kontext = 11	input = **1**0110	predikce = (0, 1)
kontext = 11	input = **0**110	predikce = (0, 1)
kontext = 10	input = **1**10	predikce = (0.5, 0.5)

kontext = 01	input = **1**0	predikce = (0.5, 0.5)
kontext = 11	input = **0**	predikce = ($0.\overline{33}$, $0.\overline{66}$)

V prvním sloupci příkladu 4.2.1 je znázorněn kontext, se kterým model v daném kroku pracuje. Ve druhém sloupci zatím nezpracovaná část vstupu, zvýrazněný bit na vstupu se model snaží predikovat. V posledním sloupci je predikce jako dvojice pravděpodobností (p_0, p_1), kde p_0 je pravděpodobnost, že následující bit bude 0 a p_1 pravděpodobnost bitu 1. Součet těchto pravděpodobností se musí rovnat jedné.

4.2.2 Model s kontextem n nukleových bází

Jako další typ modelu, který jsem sám navrhl, je model predikující následující bit dle kontextu n posledních přečtených nukleových bází. Modely n-tého řádu používali jako kontext posledních n bitů a po každém přečteném bitu je tento kontext aktualizován. Na rozdíl od nich tento model používá jako kontext n posledních ukončených bytů, tedy kontext je aktualizován každých 8 bitů. Ve své datové struktuře tedy udržuje frekvence n-tice nukleových bází a frekvence bází, které je následovali. Pokud je potřeba predikovat například třetí bit báze na vstupu, podívá se model do své tabulky, nalezne frekvence bází které se historicky vyskytly v daném kontextu a predikci vytvoří podle třetího bitu těchto bází. Důvodem, proč jsem tento model vytvořil je ten, že nukleové báze jsou hlavní složkou nukleotidů, které tvoří kodóny. Každý kodón reprezentuje jednu z aminokyselin a je tvořen právě třemi nukleotidy neboli tripletem [23]. Některé kodóny se v DNA objevují častěji než jiné (například stop kodóny ukončující genetickou informaci) a proto jsem se rozhodl pokusit se tuto vlastnost DNA využít pro kompresi. Tomu, aby mohl mít model s kontextem posledních n (ideálně dvou nebo tří) bází ještě lepší predikce, brání fakt, že sekvenovaná data mohou začínat uprostřed tripletu a také mohou obsahovat chyby čtení sekvenovacího přístroje.

Jak z popisu FASTQ formátu víme, sekvence se skládají ze symbolů A, C, G, T (pro nukleové báze) a N (v případě chybného čtení). Pro pochopení fungování algoritmu je nutné znát binární zápis alespoň některých symbolů. Znak A má v ASCII kódu hodnotu 65 neboli 01000001_2, znak T má hodnotu 84, což odpovídá zápisu 01010100_2. Nyní předpokládejme, že se nacházíme v kontextu CG, kde se báze A vyskytla dvakrát, báze T jednou a že jsou to jediné dvě báze, co se v daném kontextu vyskytly. Dále předpokládejme, že na vstupu je aktuálně báze T, tedy binární řetězec 01010100. Potom následujících 8 bitů bude predikováno tímto způsobem:

Příklad 4.2.2. (Ukázka fungování bázového modelu druhého řádu)

kontext = CG	input = **0**1010100	predikce = (1, 0)
kontext = CG	input = **1**1010100	predikce = (0, 1)
kontext = CG	input = **0**10100	predikce = (1, 0)
kontext = CG	input = **1**0100	predikce = ($0.\overline{66}$, $0.\overline{33}$)
kontext = CG	input = **0**100	predikce = (1, 0)
kontext = CG	input = **1**00	predikce = (0, 1)
kontext = CG	input = **0**0	predikce = (1, 0)
kontext = CG	input = **0**	predikce = (1, 0)
kontext = GT	input = ...	predikce = ...

Na předešlém příkladě 4.2.2 bych rád upozornil na několik důležitých věcí. První 3 bity pro báze A i T jsou stejné a proto model dává v prvních třech krocích zcela jistou predikci. Ve čtvrtém kroku, kde se bity obou bází liší, se uplatní informace o frekvencích (báze A se v daném kontextu vyskytla dvakrát, zatímco báze T pouze jednou). Proto je nulový bit predikován s pravděpodobností $\frac{2}{3}$ a 1 s pravděpodobností $\frac{1}{3}$. Po přečtení čtvrtého bitu, který se rovná jedné, je jisté, že aktuálně zpracovávaná báze nemůže být A, jelikož ta má čtvrtý bit nulový. Proto je báze A z následujích predikcí vynechána a model dále k predikci používá pouze bity báze T, což vede k ke zlepšení predikcí. Přečtením posledního symbolu báze T dojde ke změně používáného kontextu z CG na GT a metoda pokračuje stejným způsobem s novým symbolem na vstupu.

Tento model se nemusí nutně používat pouze při predikování nukleových bází. Lze stejně dobře využít i pro obecné predikce následujícího symbolu ve vstupním textu. S konkrétně nastaveným parametrem n by však měl nukleové báze predikovat obzlášť spolehlivě.

4.3 Principy kombinování predikcí modelů

4.3.1 Lineární mixování

Lineární mixování (Linear evidence mixing) je použito například v algoritmu PAQ6 [22], kde jsou pravděpodnosti nul a jedniček udávány frekvencí výskytů v kontextech jednotlivých modelů. Pravděpodonosti jsou pak kombinovány pomocí váženého součtu těchto frekvencí a to tak, že je počet výskytů bitu v každém modelu před sečtením vynásobem jeho váhou. Váhy modelů jsou

inicializovány tak, aby se jejich součet rovnal jedné. Následně jsou v každém kroku upraveny tak, aby algoritmus favorizoval přesnější modely.

Kombinované pravděpodobnosti p_0 a p_1 jsou kalkulovány následujícím způsobem:

$$S_0 = \epsilon + \sum_i w_i n_{0i}$$
$$S_1 = \epsilon + \sum_i w_i n_{1i}$$
$$S = S_0 + S_1$$
$$p_0 = \frac{S_0}{S}$$
$$p_1 = \frac{S_1}{S}$$

kde w_i je nezáporná váha i-tého modelu a n_{0i} s n_{1i} vyjadřují frekvence nul a jedniček tohoto modelu v aktuálním kontextu. Epsilon ve vzorci je malá, pozitivní konstanta, zabraňující možnému dělení nulou za situace, kdy by se S mělo jinak rovnat nule.

Aktualizace váh v každém kromu je navržena tak, aby optimalizovala cenu kódování (minimalizace velikosti komprimovaných dat) přes vektor vah všech modelů. Po zakódovaní bitu y (0 nebo 1) je váha i-tého modelu aktualizována podle následujícího vzorce:

$$w_i = \max\left(0, w_i + \frac{(y - p_i)(S n_{1i} - S_1 n_i)}{S_0 S_1}\right)$$

Ke zlepšení komprese se také používá upravené počítání frekvencní, které klade větší důraz na novější data. Pokud se na vstupu objeví nějaký bit a model v rámci své aktualizace inkrementuje frekvenci výskytů v daném kontextu, vydělí model počet frekvencí opačného bitu dvěma a přičte jedna. To však udělá pouze za situace, že se opačný bit v tomto kontextu vyskytl již alespoň třikrát. Pokud je například stav frekvencí (n_0, n_1) v aktuálním kontextu roven (10,0), po přečtení jedné se frekvence upraví na (6,1). Toto omezení na počty frekvencí zároveň slouží k lepšímu balancování vah všech modelů.

4.3.2 Logistické mixování

Logistické mixování (logistic mixing) bylo představeno v algoritmu PAQ7, kde nahradilo mixování lineární, protože dosahovalo lepších výsledků. Nepracuje s frekvencemi symbolů, ale pouze s pravděpodobnostmi, což rozšiřuje algoritmus o možnost používání modelů, které s frekvencemi bitů nepracují.

Mějme od každého modelu jeho predikci p_i a jeho váhu w_i. Potom kombinovanou pravděpodobnost získáme následujícím způsobem:

$$p = squash\left(\sum_i w_i stretch(p_i)\right)$$

kde:

$$stretch(\mathrm{p}) = \ln\left(\frac{\mathrm{p}}{1-\mathrm{p}}\right)$$
$$squash(x) = \frac{1}{1+e^{-x}}$$

Můžeme si povšimnout, že squash a stretch jsou inverzní operace, tedy že $squash(x) = stretch^{-1}(x)$. Podobně jako u lineárního mixování lze vzorec pro aktualizování váh modelů získat pomocí parciální derivace funkce ceny kódování s ohledem na váhy modelů. Po zakódovaní bitu y (0 nebo 1) je váha i-tého modelu aktualizována podle následujícího vzorce:

$$\mathrm{w}_i = \mathrm{w}_i + \lambda(\mathrm{y}-\mathrm{p})stretch(\mathrm{p}_i)$$

kde λ je míra učení (learning rate), která se typicky pohybuje kolem hodnoty 0.01, a (y-p) je chyba predikce. Na rozdíl od lineární mixování se může stát, že váhy modelů klesnou pod nulu.

4.4 Aritmetické kódování

Aritmetické kódování je metoda bezeztátové datové komprese, která pro množinu symbolů a pravděpodobnosti jejich výskytů dosahuje téměř optimální komprese. Princip této metody vymyslel Peter Elias někdy před rokem 1963 a jeho algortimus by mohl dokonce dosahovat optimálního kompresního poměru. Pro jeho dosažení by však bylo potřeba počítat s nekonečnou přesností a tím pádem používat nekonečně velký buffer, což je ze své podstaty nepraktické. Existují však modifikace tohoto algoritmu, které jsou schopné tento problém obejít. V současné době se tento algoritmus velmi hojně používá v mnoha kompresních algoritmech [24].

Základní myšlenka algorimu spočívá v tom, že se každému symbolu na základě pravděpodobnosti jeho výskytu přidělí odpovídající poměrná část intervalu <0,1). Na začátku algoritmus začne s celým intervalem <0,1), s každým symbolem na vstupu tento interval omezuje z obou stran podle toho, jaká část intervalu je danému symbolu přiřazena. Takto upravené meze intervalu se pak stanou novým základem pro další symbol a to tak, že se tento podinterval podle pravděpodobnosti výskytů také přerozdělí na poměrné části stejným způsobem jako původní interval na začátku. Kódovaný vstupní řetězec se reprezentuje libovolným realným číslem ležícím v intervalu, který jsme získali pro přečtení všech vstupních symbolů.

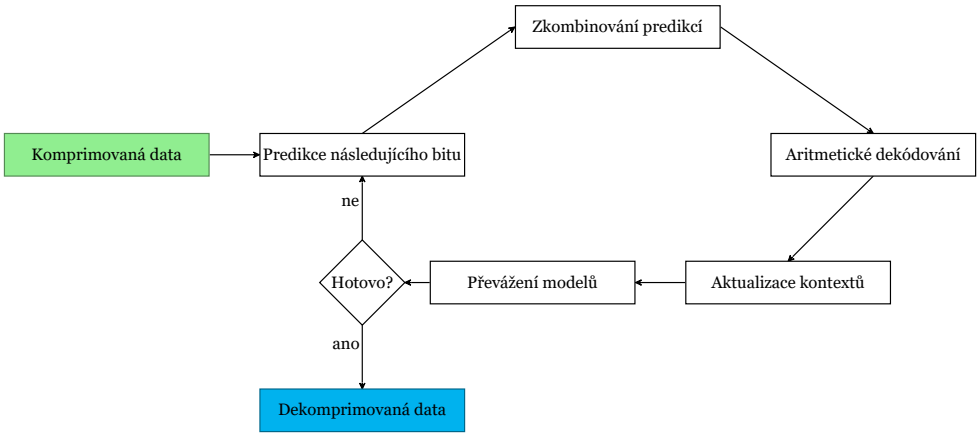
Aby se obešel problém s nekonečnou přesností, je možné interval <0,1) nahradit intervalem celých čísel. V praxi bývají meze intervalů inicializovány například 1 a maximální hodnotou neznaménkového integeru nebo neznaménkového typu long. Každému symbolu je pak na základě pravděpodobnosti jeho výskytu přiřazená část tohoto celočíselného intervalu, podobným způsobem

jako se pracovalo s intervalem <0,1). Rovnají-li se nejvýznamější bity v binárním zápise těchto intervalů, jsou tyto bity zapsány na výstup, protože už nemůže dojít k jejich změně pouhým zúžením pracovních intervalů. Následně jsou obě čísla reprezentující meze binárním shiftem posunuta doleva o odebraný počet bitů a doplněna nulovými bity pro dolní mez a jedničkovými pro mez horní. Například pro dolní mez Low = **001**00101 a horní mez High = **001**11001 dojde k oříznutí společného prefixu **001**, který se následně zapíše na výstup. Zbylé části 00101 a 11001 se doplní zprava na Low = 00101**000** a High = 11001**111**.

Při implementaci tohoto algoritmu je nutné dát si pozor na jev známý jako problém podtečení. K němu může dojít, pokud se k sobě meze intervalů těsně přiblíží, ale jejich nejsignifikantnější bity jsou rozdílné (Např.: High = $10000..._2$ a Low = $011111..10_2$). V takové situaci by se čísla sice stále více přibližovala, ale nikdy bychom nebyli schopni zapsat nejvíce signifikantní bity na výstup. V takové situaci je bity obou čísel potřeba posouvat doleva od druhé pozice dále dokud si nezačnou být rovny. Když k tomu dojde, zapíšeme na výstup bity, které jsme zahodili. Aby šla zpráva zpětně dekódovat, je potřeba přidat na konec zprávy speciální ukončovací symbol nebo společně se zprávou zakódovat její délku.

4.5 Dekomprese

Dekomprese metody mixování kontextů je velmi podobné jako komprese. K úspěšné dekompresi je potřeba dodržet několik jednoduchých pravidel. V první řadě je samozřejmě nutné použít totožné statistické modely a stejný způsob pro kombinování jejich predikcí. Podobně jako komprese začíná i dekomprese predikcí dalšího bitu a zkombinováním predikcí na základě aktuálních vah modelů. Tato predikce se předá aritmetickému dekodéru. Dekódovaný bit je následně použit pro aktualizaci kontextů všech modelů a zároveň přepočítání jejich vah, jelikož jsme díky dekódovanému bitu získat informaci o dosavadní přesnosti všech modelů. Poté jsou všechny modely sesynchronizovány se stavem, v jakém se nacházeli při kompresi, a mohou tak na následující bit podat totožnou predikci. Takto algoritmus pokračuje ve smyčce tak dlouho, dokud nejsou dekódována všechna data (dekódoval se ukončující symbol nebo počet symbolů odpovídající délce původní zprávy). Pro lepší představu je diagram dekomprese uveden níže na obrázku 4.2.



Obrázek 4.2: Diagram dekomprese metodou mixování kontextů

Kapitola 5

Implementace

V této kapitole se věnuji technologiím, které byly při vývoji této práce použity, a také více přiblížuji implementaci použitých algoritmů a v nich využitých datových struktur. Dále se v této kapitole vyskytují ukázky kódu, je zde popsán vstup a výstup algoritmu, paralelizace a v neposlední řadě adresářová struktura zdrojových kódů.

5.1 Použité technologie

Algoritmy implementované v rámci této diplomové práce se staly součástí knihovny SCT [5], která je vyvíjena v programovacím jazyce Java, aktuálně ve verzi 8. K programování jsem použil vývojové prostředí IntelliJ IDEA 2020 Community Edition, které nabízí dobrou podporu pro vývoj Java aplikací. K sestavování projektu a správě závislostí na externích knihovnách byl použit nástroj Maven od společnosti Apache.

Jako verzovací systém používá knihovna SCT službu GIT, momentálně se nachází v repozitáři na školním serveru gitlab [6]. Vývoj jsem prováděl ve vlastní vývojové větvi s názvem FASTQ, aby se zabránilo případným kolizím s dalšími developery.

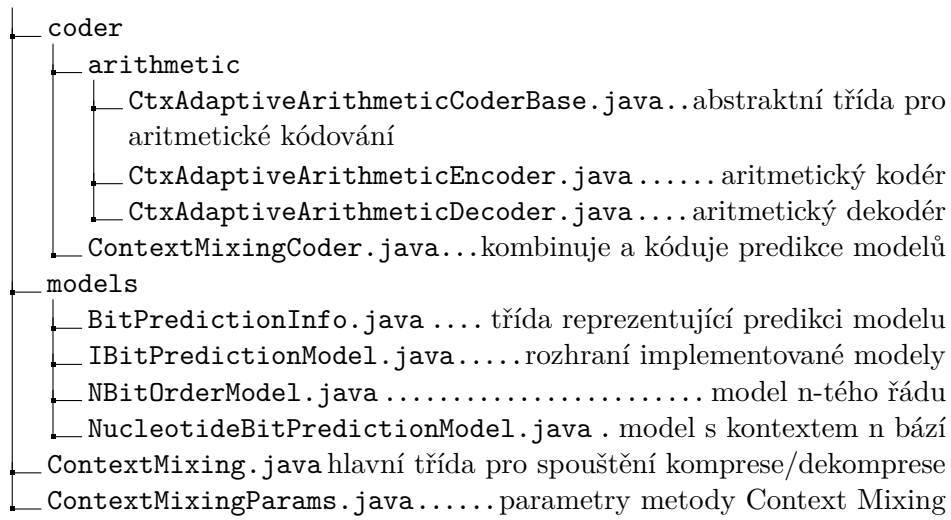
5.2 Rozdělení implementace

V rámci knihovny jsem svou implementaci rozdělil na dvě hlavní části. První část se nachází v adresáři (Java package) s názvem contextMixing a obsahuje implementaci algoritmu míchání kontextů a také použitých modelů. Druhá část (package s názvem fastq) potom obsahuje implementaci datových struktur pro pracování s FASTQ soubory a taky kodér a dekodér s možností komprimovat datové proudy z FASTQ souborů pomocích různých algoritmů.

5.2.1 Implementace mixování kontextů

Implementace mixování kontextů proběhla v adresáři contextMixing a skládá se z implementace aritmetického kodéru, modelů a kodéru, který kombinuje jejich predikce.

V první řadě popíšu zmíněnou adresářovou strukturu pro package contextMixing. Ta vypadá následovně:



Obrázek 5.1: Adresářová struktura metody Context Mixing

Mezi nejdůležitější a nejzajímavější části implementace této metody patří především modely (`NBitOrderModel` a `NucleotideBitPredictionModel`) vysvětlené v sekcích 4.2.1 a 4.2.2. Oba modely implementují rozhraní `IBitPredictionModel`, které jim předepisuje následující metody:

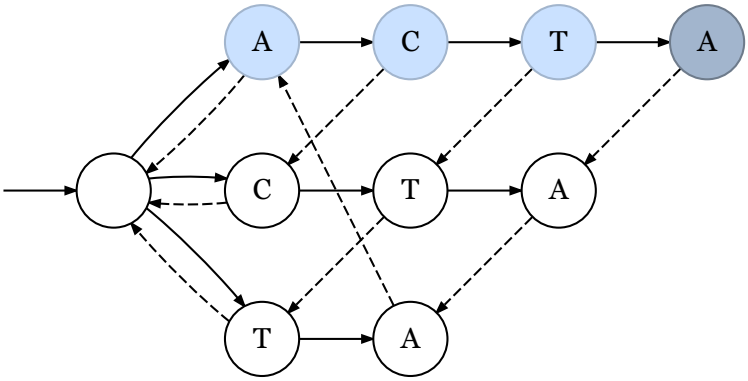
```
public interface IBitPredictionModel {
        BitPredictionInfo makePrediction();
        void update(int bit);
        void setWeight(double weight);
        double getWeight();
}
```

Tyto metody pak volá třída `ContextMixingCoder`, která implementuje lineární i logistické míchání kontextů, během kterého je potřeba získávat predikce modelů (metoda `makePrediction()`) a aktualizovat kontext a váhu všech modelů (metoda `update(int bit)` a `setWeight(double weight)`). Metoda `getWeight()` je pak důležitá při výpočtu kombinované predikce všech použitých modelů.

V modelu `NBitOrderModel` je k získávání predikce použitá frekvenční tabulka, která obsahuje počty výskytů různých posloupností bitů. Vytvoření

predikce je tedy jednoduché získání informace o množství vyskytnutých nulových a jednočkových bitů v aktuálním kontextu délky n. Aktualizování kontextu metodou `update(int bit)` inkrementuje frekvenci bitu předaného jako argument funkce v této tabulce a kontextu.

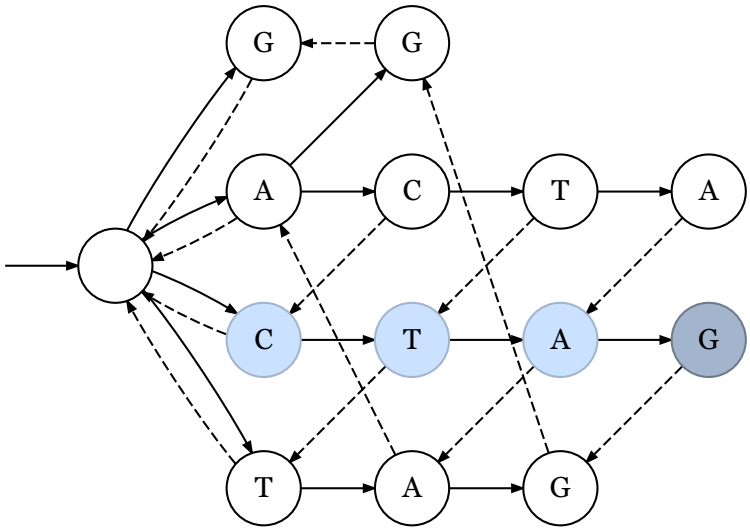
Třída `NucleotideBitPredictionModel` narozdíl od `NBitOrderModel` používá jako podpůrnou datovou strukturu *Suffix Trie*, kde jeden uzel stromu reprezentuje jeden symbol a cesta od kořene do tohoto uzlu má význam kontextu, ve kterém se symbol nachází. Zároveň má v sobě každý uzel uložený počet výskutů symbolu v daném kontextu. Jak je vysvětleno v sekci 4.2.2, model pro vytvoření predikce potřebuje znát frekvence všech symbolů, které se vyskytly v aktuálním kontextu. Tuto informaci lze díky stromové struktuře získat jednoduše tak, že se podíváme na potomky uzlu reprezentující daný kontext. Důvod, proč jsem vybral právě suffixový strom, je však možnost „rotovat“ kontext během metody update pomocí takzvaných *suffix linků*. Jedná se o hrany do předchozí úrovně ve stromové struktuře, které vedou do uzlu reprezentujícího největší možný suffix (kromě sebe samotného) k uzlu, ze kterého vychází. Například z uzlu reprezentujícího posloupnost symbolů ACTA vede suffix link do uzlu CTA, z něj do TA, dále do A a nakonec to kořenového uzlu. Představme si, že máme model který jako kontext používá 4 symboly (báze), akutální kontext je ACTA a právě se zpracoval symbol G. Potom nový kontext získáme tak, že z uzlu reprezentujícího kontext ACTA přejdeme po suffix linku do uzlu CTA a odtamtud pokračujeme již normální hranou do uzlu G. Tím se dostaneme do uzlu CTAG, což odpovídá novému kontextu a to v konstantním čase, který je nezávislý na hloubce stromu.



Obrázek 5.2: Suffix Trie po vložení symbolů ACTA

Na obrázku 5.2 je zobrazen Suffix Trie hloubky 4, do kterého byly postupně vloženy symboly označující báze ACTA. Pohyb po směru plných hran reprezentuje zvětšení kontextu o symbol v cílovém uzlu této hrany. Naopak přerušované hrany reprezentují suffix linky vysvětlené v předchozím odstavci. Uzel A označený tmavším odstínem reprezentuje uzel, se kterým se pracovalo

posledně a světle obarvené uzly zvýrazňují kontext, ve kterém se tento uzel nachází.



Obrázek 5.3: Suffix Trie po přidání symbolu G

Na obrázku 5.3 je pak zobrazen stejný Suffix Trie po přidání nového symbolu G, jak je popsáno na předchozím příkladu. Jak je z obrázku patrné, pro přidání tohoto nového uzlu stačilo přejít po suffixové hraně předchozího symbolu A do uzlu, kam tato hrana směřovala a přidat nového potomka – uzel G. Tento nový uzel a nový kontext jsou na obrázku opět barevně zvýrazněny a jak lze vidět, odpovídají řetězci CTAG, stejně jako na použitém příkladu. Zároveň se tomutu uzlu rovnou vytvoří jeho suffix link a rekurzivně se do stromové struktury přidají všechny potřebné uzly. V nejhorším případě, jako je tento, kdy se přidává symbol, který se ve stromě ještě nevyskytoval, je potřeba vytvořit uzly a jejich suffix linky od nově přidaného uzlu G až ke kořeni stromu. V tomto případě se do stromu kromě uzlu představujícího řetězec CTAG musely přidat také řetězce TAG, AG a G.

Nyní přiblížím implementaci aritmetického kódování, popsaného v sekci 4.4, které převádí kombinované predikce použitých modelů na výstupní komprimované data. Jedná se o mírně upravenou verzi adaptivního aritmetického kódování, kde se poměrné části intervalu pro symboly (v tomto případě bity) upravují na základě frekvence jejich výskytů během komprese nebo dekomprese. V tomto případě se o rozdělení intervalů na části stará třída `ContextMixingCoder`, která tuto informaci algoritmu aritmetického kódování dodává společně s bitem ke komprimaci. Nové rozdělení intervalu vždy odpovídá aktuální kombinované predikci všech modelu pro kódovaný bit. Vzhledem k tomu, že kódujeme bity, interval $<low, high>$ se vždy rozdělí na pouze dvě části. Část $<low, mid)$ je poměrná část pro nulový bit, část $<mid, high>$ pro jedničkový. Hodnotu

oddělovače *mid* můžeme z kombinované predikce a mezí intervalu spočítat následujícím způsobem:

$$mid = low + p_0(high - low + 1)$$

kde p_0 je kombinovaná pravděpodobnost použitých modelů, že kódovaný bit y je 0. Nové meze intervalu pak získáme takto:

$$<low, high> := \begin{cases} <low, mid), & \text{if } y = 0 \\ <mid, high>, & \text{if } y = 1 \end{cases}$$

Například pro interval (1,100), predikci p_0 = 75 % a p_1 = 25 % a kódovaný bit 0, vypočteme hodnotu *mid* rovnou 76 a nové meze intervalu upravíme na (1, 75). Pokud by jsme naopak kódovali bit 1, nové meze intervalu by se nastavily na (76,100). Jak lze pozorovat, lepší predikce slouží k tomu, aby se meze intervalů přibližovaly pomaleji díky čemuž stačí zapisovat na výstup méně často a algoritmus dosahuje lepší komprese.

5.2.2 Implementace FASTQ kódování

Implementace FASTQ kódování proběhla v adresáři fastq a skládá se z implementace FASTQ kodéru a dekodéru, datového modelu, klienta a dalších podpůrných tříd.

Adresářová struktura zmíněné části vypadá následovně:

- `coder`
  - `FASTQEncoder.java` ..... třída pro komprimování FASTQ dat
  - `FASTQDecoder.java` ..... třída pro dekomprimování FASTQ dat
- `data`
  - `FASTQData.java` ..... datový model pro FASTQ soubory
- `FASTQ.java` ..... hlavní třída pro spouštění komprese/dekomprese
- `FASTQClient.java` . implementace konzolového uživatelského rozhraní
- `FASTQProvider.java` ..... třída poskytující parametry a kompresní algoritmy
- `FASTQTest.java` ..... třída pro testování algoritmu

Obrázek 5.4: Adresářová struktura FASTQ kódování

Hlavní třídou zaobalující proces komprese i dekomprese je třída s názvem `FASTQ`, která pro své vytvoření pomocí konstruktoru vyžaduje instanci třídy `FASTQProvider`. Třída `FASTQProvider` obsahuje parametry, kterými si kompresní algoritmy řídí například velikost zpracovávaných datových bloků nebo

povolení paralelizace, a zároveň kompresní algoritmy inicializuje a dodává je na potřebná místa. Třída `FASTQ` potom inicializuje `FASTQEncoder` nebo `FASTQDecoder` podle toho, zda se pouští komprese či dekomprese.

V rámci třídy `FASTQEncoder` se pak spouští kompresní algoritmy na jednotlivé datové proudy (identifikátory, sekvence, kvality skóry) a jejich výsledky se – společně s metadaty potřebnými pro dekódování – poskládají na výstup.

Běh kompresních metod se v knihovně řídí třídou `ChainBuilder`, která slouží k řetězení operací podobně jako tomu je v Java Stream API. Vytvoří se posloupnost funkcí, které po provedení svého výpočtu předají svůj výstup další funkci v řetězci. Spuštění metody LZMW s využitím řetězení vypadá následovně:

```
List<byte[]> encodedIDs = new ArrayList<>();
ChainBuilder.create(this::wrapBytes)
        .chain(provider.getLzmw()::compress)
        .chain(provider.getTriplet2Byte())
        .end(encodedIDs::addAll)
        .accept(fastqData.getIDs());
```

Vstupní data pro řetězení se zadávají metodě `accept`. V tomto případě se tedy jedná o identifikátory sekvencí. Ty se předají první metodě v řetězení `wrapBytes`, která je zabalí do třídy `ByteBuffer` a předá metodě `compress` algoritmu LZMW. Kompresní algoritmus LZWM přemění vstupní data na takzvané triplety, které jsou pak předány aritmetickému kódování (metodou `getTriplet2Byte`) ke kompresi. Výsledek tohoto kódování se uloží do proměnné `encodedIDs`.

Podobným způsobem se spouští i komprese pomocí mixování kontextů. Na následující ukázce lze vidět spuštění komprese touto metodou na datový proud sekvencí a uložení jejího výstupu do proměnné `encodedSequences`.

```
List<byte[]> encodedSequences = new ArrayList<>();
ChainBuilder.create(this::wrapBytes)
        .chain(provider.getContextMixing()::compress)
        .end(byteBuffer -> encodedSequences.add(byteBuffer.array()))
        .accept(fastqData.getSequences());
```

Třída `FASTQDecoder` naopak pracuje se zakódovanými daty a ty předává kompresním metodám k dekompresi totožným způsobem, jako se tomu dělo u komprese. Po získání výstupu všech dekodérů je potřeba dekomprimovaná data zkompletovat zpět do původní podoby, tedy do formátu odpovídajícímu FASTQ. K tomu slouží funkce `createOutput`, která z dekódovaných dat vytvoří posloupnost záznamů ve formě: identifikátor, sekvence, oddělovač, kvality skór, a tak dále až do konce dat. Oddělovač je jednoduše vygenerován jako řetězec "+\n", případně lze pomocí přepínače `-g` vygenerovat za symbol + také identifikátor dané sekvence.

5.3 Vstup a výstup

O vstup a výstup se stará třída `FileIO`. Tato třída již byla součástí knihovny a obsahuje metody k načítání souborů do paměti a jejich ukládání na disk. Do této třídy jsem vytvořil metodu `parseFASTQ`, která slouží k načítání souboru typu FASTQ a zároveň k rozdělení vstupních dat na datové proudy identifikátorů, sekvencí a skórů kvality. V rámci mixování kontextů jsou poslední dva proudy obaleny třídou `BitInputStream`, která umožňuje čtení datového streamu po jednotlivých bitech. Podobným způsobem třída `BitOutputStream` složí k vytvoření výstupu algoritmem aritmetického kódování s možností zapisovat po jednotlivých bitech. Výsledek komprese je pak uložen do souboru metodou `saveParsed` ze třídy `FileIO`, která bere jako parametry ByteBuffer a jméno výstupního souboru.

5.3.1 Segmentování vstupu

Vzhledem k tomu, že FASTQ soubory nabývají velikostí i ve stovkách GB, není možné načtení jejich celého obsahu do paměti za běhu algoritmu. Proto je vstup segmentován do bloků a zpracováván postupně. Velikost těchto bloků je nastavitelná přepínačem `-n <number>`, kterým lze specifikovat množství záznamů v jednom zpracovávaném bloku. Tímto způsobem lze tedy do velké míry upravovat paměťovou náročnost celého algoritmu.

5.3.2 Metadata

Pro možnost dekomprimování dat se společně s výstupem algoritmů ukládá malé množství metadat, které obsahuje velikosti jednotlivých komprimovaných částí. Každý segment je uložen na výstup následujícím způsobem:

Velikost komprimovaných identifikátorů → komprimované identifikátory → velikost komprimovaných sekvencí → původní velikost sekvencí → komprimované sekvence → velikost komprimovaných QS → původní velikost QS → komprimované QS.

Takto se za sebe skládají jednotlivé segmenty tak dlouho, dokud není zakódovaný celý vstupní FASTQ soubor. Při dekódování se pak vždy přečte velikost komprimované sekce, a odpovídající část bufferu se předá patřičnému dekompresnímu algoritmu.

5.4 Paralelizace

Zrychlení komprese FASTQ souborů jsem se pokusil dosáhnout pomocí paralelizace. Jak lze vidět z obrázků 3.3 a 3.4, komprese a dekomprese jednotlivých

datových proudů je na sobě nezávislá a tedy lze provádět zároveň. Pokud je paralelizace zapnutá, spustí se komprese identifikátorů, sekvencí a skóry kvality v jednotlivých vláknech. Po doběhnutí vláken se výstup zapíše do souboru a to ve zmíněném pořadí. Během dekomprese se musí před zapisováním do souboru počkat na dokončení práce všech vláken, jelikož je potřeba výsledky před zapsáním do souboru zkompletovat do správného pořadí. Algoritmy pracují paralelně, pokud je metoda spuštěna s přepínačem `-p`. V opačném případě se pro kompresi i dekompresi dodržuje pořadí identifikátory → sekvence → skóry kvality.

5.5 Klient

Pro testování komprese a dekomprese slouží třída `FASTQClient`, která implementuje rozhraní pro použití přes příkazovou řádku. Momentálně je ve vývojové větvi FASTQ pro jednodušší testování tento klient nastavený jako hlavní třída projektu (main) a lze proto spustit jednoduše pomocí:

```
java -jar <jar file>
```

Druhá možnost pro spuštění klienta je pomocí buildovací utility Maven příkazem:

```
mvn exec:java -Dexec.mainClass="cz.cvut.fit.fastq.FASTQClient"
-Dexec.args="${ARGS}"
```

kde proměnná `ARGS` obsahuje seznam argumentů. V případě nevalidního spuštění vypíše klient do konzole následující nápovědu:

```
usage: java -jar <jar name> <input> <output> [options]
       or
       mvn exec:java -Dexec.mainClass="cz.cvut.fit.fastq.FASTQClient"
       -Dexec.args=[options]
input - input file or directory
output - output file
-de,--decompress           decompress input (default is to compress)
-h,--help                  print this help
-lin,--linear              context mixing uses linear mixing for
                           combining predictions (default is logistic
                           mixing)
-log,--log-level <level>   sets logging level of the application
-m,--measure               measured program process data printed to
                           standard output(default: be silent)
-n <N>                     number of entries compressed in one block
                           (default is 1_000_000)
```

```
-p,--parallel              compress identifiers, sequences and quality
                           scores in parallel
-g                         generate sequence identifiers as comment
-o                         optimize compression (uses more models)
```

Pro správné spuštění je nutné zadat validní vstupní soubor (input) a jméno pro výstupní soubor (output). Zbytek paramentrů je volitelný, respektive má defaultní hodnoty, specifikované ve vypsané nápovědě.

Kapitola 6

Výsledky

Implementovaný algoritmus pro kompresi FASTQ souborů jsem testoval z hlediska kompresního poměru, času komprese i dekomprese, a také spotřebované paměti. Ve svém testováním porovnávám kompresní poměr nejen na celých souborech, ale také v rámci jednotlivých částí souboru, tedy opět identifikátorů, sekvencí a skórů kvalitu. Závěrem této kapitoly svůj algoritmus také porovnávám s výsledky ve výzkumech *MZPAQ: a FASTQ data compression tool* [8], ve které autoři porovnávali existující metody a navrhli svůj vlastní kompresní algoritmus, a dále s publikací *Compression of FASTQ and SAM Format Sequencing Data* [9] porovnávající algoritmy přihlášené do kompresní soutěže *SequenceSqueeze*.

Všechna měření byla provedena na jednom zařízení s následujícími parametry:

- Procesor – Intel(R) CoreTM i7-7700HQ CPU (2800GHz),
- RAM – 8 GB,
- Operační systém – Windows 10 (64 bitový).

6.1 Testovací soubory

Většinu testování jsem provedl na třech FASTQ souborech, kde jsem zjišťoval chování algoritmu s různě navolenými parametry. Především jsem se zaměřil na vliv typu kontextového mixování a počtu modelů na kompresní poměr, ale také časovou a paměťovou náročnost. Jako první testovací soubor jsem použil sekvenování genomu mořské řasy *Zostera marina*, které provedl přístroj Illumina HiSeq 2000, a je dostupný v archivu European Nucleotide Archive [18] pod označením SRR2980545. Další dva soubory jsem dostal od vedoucího práce v adresářích označených exander2 a fey2. Podle informací v indetifikátorech pochází data ze sekvenovacího přístroje NovaSeq společnosti Illumina. V této kapitole dále označovat jmény jejich adresářů. Pro lepší porování

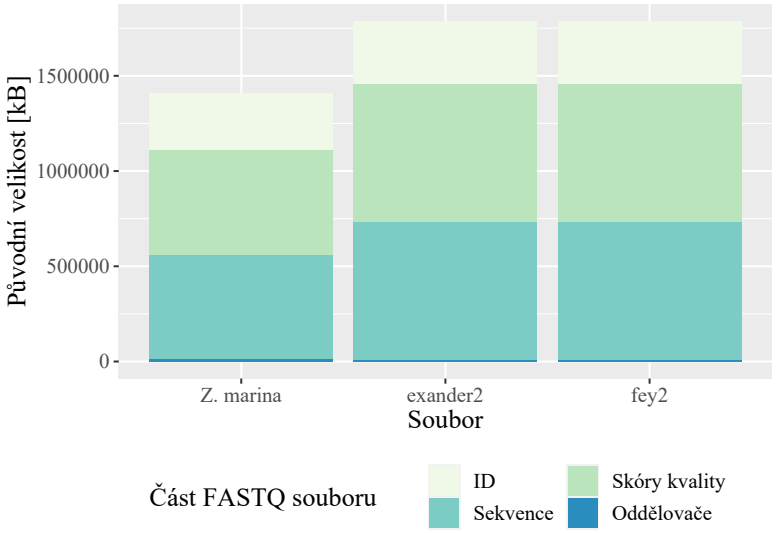
s publikací *Compression of FASTQ and SAM Format Sequencing Data* [9] se mi pak podařilo dohledat jeden z FASTQ souborů použitých k testování jejími autory, a to konkrétně soubor obsahující sekvenování lidského genomu SRR007215_1. Tato data byla vytvořena sekvenovací technologií SOLiD. Informace o velikostech testovacích souborů jsou k nalezení v tabulce 6.1.

Tabulka 6.1: Informace o použitých testovacích souborech

Soubor	**Velikost souboru**	**Použitá velikost**	**Počet záznamů**
Z. marina	1,408,364 KB	1,408,364 KB	5,519,434
exander2	107,453,331 KB	1,786,465 KB	5,000,000
fey2	134,057,063 KB	1,787,157 KB	5,000,000
SRR007215_1	673,164 KB	673,164 KB	4,771,141

Jak je naznačeno v tabulce 6.1, soubory exander2 a fey2 jsem kvůli jejich značné velikosti omezil na 5 milionů záznamů. Vzhledem k tomu, že soubory jsou vždy vytvářeny jedním sekvenovacím přístrojem, lze o nich říci, že dodržují stejnou strukturu v rámci celého souboru. Tím pádem můžeme do značné míry předpokládat, že algoritmus by si svůj kompresní poměr a ostatní vlastnosti zachoval i při použití na zbytek souboru.

Na grafu 6.1 je pak možné si prohlédnout skladbu těchto souborů, co se týče jednotlivých datových proudů. Především si můžeme všimnout, že převážnou část souboru tvoří sekvence a skóry kvality. Naopak téměř nepatrné jsou na grafu oddělovače, které se v každém záznamu skládají pouze ze 2 B.



Obrázek 6.1: Skladba testovacích souborů

6.2 Nastavení modelů

Pro sjednocení testování jsem zvolil dvě verze nastavení modelů. Jejich počet a složitost ovlivňuje nejel kvalitu komprese, ale především časové a paměťové nároky celého algoritmu. Slabší verze používá celkem 4 modely, z toho 2 jsou modely n-tého řádu s parametrem n = 8 a 16. Zbylé dva jsou modely s kontextem n nukleových bází s parametrem n = 2 a 3.

Silnější verze algoritmu pak používá celkem 8 modelů. Čtyři z nich jsou n-tého řádu s parametrem n = 4, 8, 12 a 16. Dále silnější verze používá 4 nukleonové modely s parametrem n = 1, 2, 3 a 4.

Tyto hodnoty parametru byly zvoĺeny na základě předpokladu, že nukleové báze se v DNA vyskytují v určitém pořadí a tvoří tak kodóny, které reprezentují aminokyseliny. Díky tomu, že se některé kodóny objevují v DNA častěji než jiné, lze předpokládat, že i na základě využití krátkého kontextu můžeme získat dobré predikce. S nastavením délky kontextů pro modely by se však dalo dlouze experimentovat.

6.3 Lineární a logistické mixování

V první řadě jsem vlastnosti svého algoritmu otestoval s ohledem na kompresní poměr a časovou a paměťovou složitost pro obě implementované možnosti kombinování predikcí modelů (sekce 4.3). Pro sjednocení podmínek jsem použil slabší verzi algoritmu. Ten byl tedy spuštěn celkem se 4 modely.

V tabulce 6.2 jsou uvedeny informace o velikost před a po kompresi a z nich vypočtený kompresní poměr (defn. 1.0.3). Na naměřených datech je vidět, že komprese s použitím logistického mixování dosahovala vždy lepších výsledků. Řádově se rozdíly mezi metodami ve velikosti komprimovaných souborů pohybovaly v desetinách až jednotkách procent. Tento výsledek je v souladu s poznatky o obou algortimech, jelikož metoda lineárního mixování predikcí byla používaná pouze ve starších verzích kompresních algoritmů PAQ. Později byla nahrazena právě logistickou metodou, protože dosahovala lepšího kompresního poměru.

Tabulka 6.2: Komprese testovacích souborů

Soubor	**Metoda**	**Původní v.**	**Nová v.**	**CR**
Z. marina	linearní	1,408,364 KB	335,926 KB	0.23852
Z. marina	logistická	1,408,364 KB	325,304 KB	0.23098
exander2	linearní	1,786,465 KB	235,411 KB	0.13178
exander2	logistická	1,786,465 KB	233,043 KB	0.13045
fey2	linearní	1,787,157 KB	243,996 KB	0.13653
fey2	logistická	1,787,157 KB	241,695 KB	0.13524

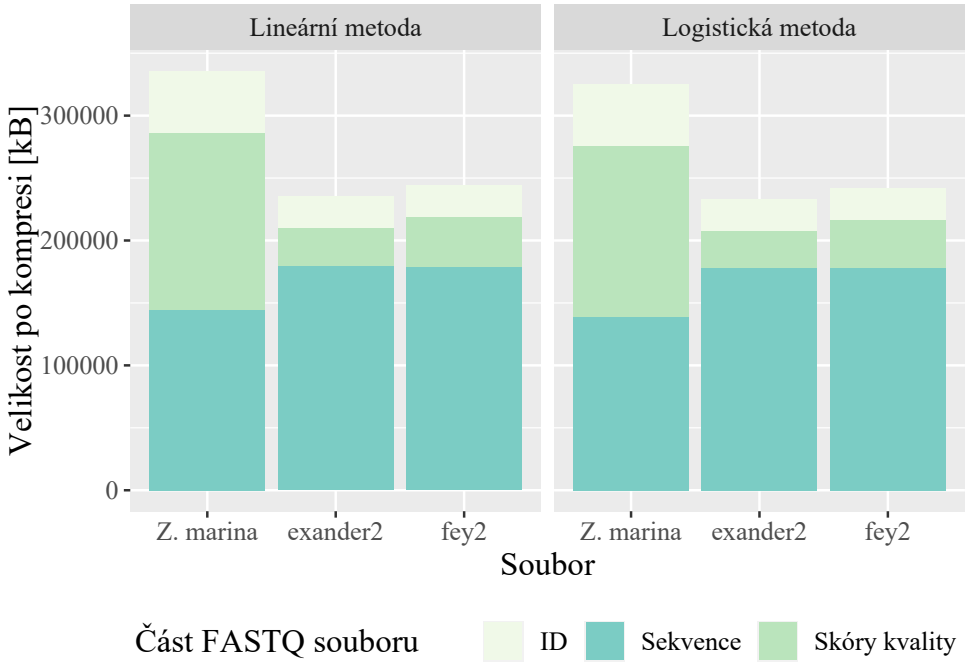
Pro porovnání uvádím výsledky standardního kompresního algoritmu GZip, který se pro komprimaci FASTQ souborů běžně používá. Ten u souboru Z. marina dosahuje kompresního poměru 0.295, na souborech exander2 a fey2 pak postupně 0.17 a 0.177.

V tabulce 6.3 jsou k dispozici informace o kompresi jednotlivých datových proudů. Na kompresi identifikátorů neměla volba metody kontextového mixování samozřejmě vliv, jelikož na ně byl aplikován algoritmus LZMW, kterého se toto nastavení netýká. Pro proudy sekvencí a kvality skórů se metoda logistického mixování ukázala jako lepší. Z této tabulky lze také vypozorovat, že hlavním důvodem pro značně rozdílný kompresní poměr souboru *Zostera marina* v porovnání se soubory *exander2* a *fey2* byly skóry kvality. Zatímco u *Zostera marina* byly tyto skóry zkomprimovány na přibližně 25 % původní velikosti, u souborů *exander2* a *fey2* se jednalo zhruba 4 respektive 5 %.

Tabulka 6.3: Komprese jednotlivých datových proudů

Soubor	**Data**	**Metoda**	**Původní v.**	**Nová v.**	**CR**
Z. marina	identifikátory	linearní	298,009 KB	50,114 KB	0.16816
Z. marina	sekvence	linearní	549,787 KB	144,356 KB	0.26257
Z. marina	QS	linearní	549,787 KB	141,455 KB	0.25729
Z. marina	identifikátory	logistická	298,009 KB	50,114 KB	0.16816
Z. marina	sekvence	logistická	549,787 KB	138,726 KB	0.25233
Z. marina	QS	logistická	549,787 KB	136,463 KB	0.24821
exander2	identifikátory	linearní	329,426 KB	25,808 KB	0.07834
exander2	sekvence	linearní	723,637 KB	179,337 KB	0.24783
exander2	QS	linearní	723,637 KB	30,265 KB	0.04182
exander2	identifikátory	logistická	329,426 KB	25,808 KB	0.07834
exander2	sekvence	logistická	723,637 KB	178,019 KB	0.24601
exander2	QS	logistická	723,637 KB	29,215 KB	0.04037
fey2	identifikátory	linearní	329,467 KB	25,306 KB	0.07681
fey2	sekvence	linearní	723,962 KB	179,377 KB	0.24777
fey2	QS	linearní	723,962 KB	39,312 KB	0.05430
fey2	identifikátory	logistická	329,467 KB	25,306 KB	0.07681
fey2	sekvence	logistická	723,962 KB	178,107 KB	0.24602
fey2	QS	logistická	723,962 KB	38,282 KB	0.05288

Tato naměřená data jsou také v přehlednější formě uvedena na obrázku 6.2. Na grafu 6.3 jsou pak k prohlédnutí časy komprese a dekomprese obou testovaných metod mixování kontexů. Z grafu lze vypozorovat, že doba komprese testovaných souborů je velmi podobná, což odpovídá tomu, že tyto soubory jsou podobné velikosti a byly testovány se stejnými parametry modelů. Časy komprese a dekomprese se také liší pouze minimálně, jelikož algoritmus při dekompresi funguje víceméně stejně.

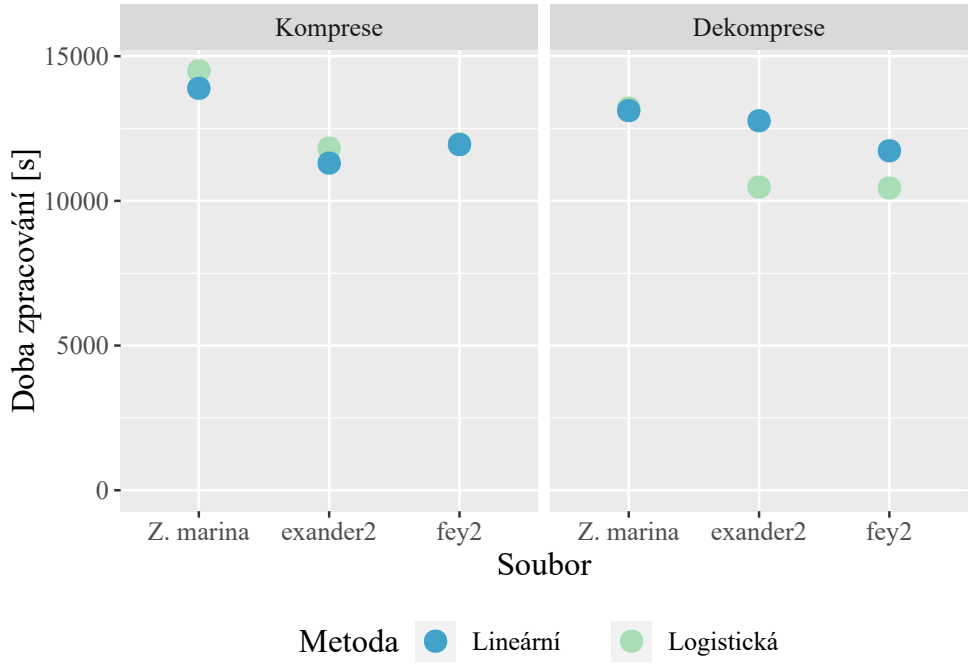


Obrázek 6.2: Výsledky lineárního a logistického mixování

6.4 Počet modelů a velikost bloků

V další fázi testování jsem se zaměřil na rozdíl mezi slabší a silnější verzí algoritmu (slabší verze používá 4 modely, silnější verze celkem 8). Testování jsem omezil na soubor *fey2* a metodu logistického mixování predikcí, abych zamezil vlivům vedlejších faktorů na výsledky měření. Kromě vlivu komprese jsem se v této fázi zaměřil především na časovou a paměťovou náročnost algoritmu, jelikož oba testované parametry s těmito vlastnostmi úzce souvisí. Velikostí bloku je myšlen počet záznamů, kde každý záznam se skládá z jednoho identifikátoru, sekvence, atd. Tento parametr má vliv na objem dat nahraný v paměti za běhu metody a také ovlivňuje množství metadat zapsaných do výsledného souboru. Výsledky testování jsou uvedené v tabulce 6.4.

Z výsledků v tabulce 6.4 můžeme v první řadě vyčíst, že přidání dalších 4 modelů sice zlepšilo kvalitu komprese, ale pouze nepatrným způsobem. Naopak výrazné rozdíly při různém nastavení parametrů lze pozorovat u doby běhu a množství využité paměti. Použití dvojnásobného počtu modelů mělo pouze malý vliv na spotřebu operační paměti, ale významně se podílelo na době běhu kompresního algoritmu. Doba běhu silnější verze komprese byla v porovnání se slabší verzí přibližně 1,7x větší. Ke zlepšení komprese však přispěla pouze v řádu desetin procenta. Upráva velikosti zpracovávaného bloku paměti neměla na výslednou kompresi a časovou náročnost žádný pozorovatelný vliv



Obrázek 6.3: Časová složitost komprese s různým kombinováním predikcí

Tabulka 6.4: Vliv počtu modelů a velikosti bloků

Soubor	Varianta	Velikost bloku	Nová v.	Pamět	Čas
fey2	slabší	1,000,000	241,695 KB	2,674 MB	10 445s
fey2	slabší	500,000	241,692 KB	2,069 MB	11 815s
fey2	slabší	250,000	241,715 KB	1,274 MB	11 969s
fey2	silnější	1,000,000	241,377 KB	2,678 MB	18 550s
fey2	silnější	500,000	241,381 KB	2,138 MB	20 316s
fey2	silnější	250,000	241,379 KB	1,312 MB	19 758s

(použití velmi malých bloků by však mohlo zvýšit dobu běhu kvůli zbytečně časté a krátké práci se soubory). Naopak podle předpokladu lze úpravou tohoto parametru jednodušše snižovat a zvyšovat pamětovou složitost. Větší paměťové bloky pak mohou mít pozitivní efekt na kompresi, jelikož datové struktury použitých algoritmů budou obsahovat větší množství dat, tento efekt však v měření nebyl pozorován.

6.5 Porovnání výsledků

Naměřené výsledky jsem porovnal s článkem autorů Jamese K. Bonfielda, Matthew V. Mahoneyho a Michaela Gormleyho *Compression of FASTQ and SAM Format Sequencing Data* [9]. Aby bylo výsledky možné porovnat, podařilo se mi dohledat jeden ze souborů použitých v této studii, konkrétně soubor SRR007215_1 obsahující data ze sekvenování lidského genomu. Naměřená data o kompresi tohoto souboru jsou uvedené v tabulce 6.5.

Tabulka 6.5: Komprese souboru SRR007215_1

Data	**Původní v.**	**Nová v.**	**CR**
identifikátory	412,242 KB	35,295 KB	0.08561
sekvence	125,802 KB	31,283 KB	0.24867
QS	125,802 KB	67,566 KB	0.53708
celý soubor	**673,164 KB**	**134,145 KB**	**0.19927**

Z tabulky můžeme kromě údajů o kompresi vyčíst také informaci, že identifikátory tvořili předvážnou část testovaného souboru. Je to z důvodu, že k jeho vytvoření byla použita starší sekvenovací technologie a sekvence v jednotlivých čteních jsou výrazně kratší než u dříve testovaných souborů. Informace o skórech kvality se v souboru stále a nepravidelně mění, a proto nedopadla jejich komprese nejlépe.

Ve výše zmíněné studii si se souborem nejlépe poradila metoda *fqzcomp*, která dosáhla kompresního poměru 0.1419. Nejhorší naměřené komprese pak dosáhla metoda GZip s kompresním poměrem 0.2524. Mnoho testovaných algoritmů si se souborem neporadilo vůbec. Důvodem byla proměnlivá délka sekvencí a skórů kvality napříč záznamy. S tímto problémem se mnou implementovaný algoritmus zvládne vypořádat. Vyjma standardních kompresních algoritmů (GZip a BZip2) a algoritmů nepodporujících proměnlivé délky sekvencí, se kompresní poměry soutěžních metod pohybovaly v intervalu od 0.1419 do 0.1605.

Kapitola 7

Diskuze

Ačkoliv implementovaný algoritmus dosahuje kompresního poměru lepšího než běžně používané standardní kompresní metody, nemůže se těmto metodám rovnat v časové náročnosti.

Jako hlavní problémy délky běhu metody jsem identifikoval především nedostatečnou paralelizaci, programovací jazyk Java a případné nedostatky s optimalizací v rámci samotné implementace (např: možné zbytečné kopírování bufferů).

Rychlé kompresní metody vyžadují kvalitní a optimalizované implementace, ideálně v jazycích jako C nebo C++. Programovací jazyk Java, ve kterém se knihovna SCT implementuje, vyžaduje běh pomocí virtuálního prostředí Java Virtual Machine (JVM) a tím implicitně ztrácí na rychlosti a možnostech některých optimalizací. Tento programovací jazyk však nebyl vybrán z důvodu rychlosti, ale proto, že přes jeho velkou popularitu v něm není implementováno mnoho kompresních algoritmů, což byl jeden z důvodů ke vzniku této knihovny.

Další možností pro zrychlení je zlepšení paralelizace. Ve své implementaci jsem paralelizaci použil pouze pro kompresi nezávislých datových proudů (identifikátorů, sekvencí, skórů kvality) v rámci jednoho bloku dat načtených do paměti. Za zvážení by zcela jistě stálo například paralelní zpracování více bloků dat najednou za cenu vyšší paměťové náročnosti. S dostatkem operační paměti by se tak dalo dosáhnout až několikanásobného zrychlení komprese i dekomprese. Se zvýšením množství bloků by se však mohla snížit kvalita komprese, kvůli redukci dlouhodobých kontextů.

Druhou možností by pak byla případná paralelizace samotných použitých kompresních algoritmů, především mixování kontextů (algoritmus LZMW pracuje výrazně rychleji). Paralelizace tohoto algoritmu je v praxi sice zcela jistě jmožná, nicméně sám Matthew Mahoney (autor algoritmů PAQ) několikrát poznamenal, že primárním cílem jeho algoritmů je dosažení maximální komprese bez ohledu na časové a paměťové požadavky, a proto je většina implementací jednovláknových. Paralelizace by však docílit šlo například spuštěním

každého modelu v jiném vlákně. Kombinování predikcí by se však stalo výrazně komplikovanější a došlo by také ke zvýšení paměťové náročnosti algoritmu.

Závěr

Během zpracování diplomové práce jsem se seznámil s oblastí sekvenovaní DNA a produkovanými daty ve formátu FASTQ, které je potřeba efektivně ukládat. Především jsem se v práci zaměřil na implementaci algoritmu mixování kontextů. Jedná se přístup, který se ukazuje být vhodnou volbou pro kvalitní kompresi genomických dat. Na základě analýzy a vlastního testování jsem navrhnul a implementoval kompresní algoritmus pro kompresi FASTQ souborů využívající metodu LZMW pro kompresi identifikátorů a metodu mixování kontextů pro kompresi a dekompresi DNA sekvencí a skórů kvality. Výsledkem práce je metoda pro kompresi FASTQ souborů implementovaná do knihovny *Small Compression Toolkit* a spustitelná pomocí konzolového rozhraní. Metodu je implementovaná ve slabší a silnější verzi a pomocí nastavení parametrů je možné měnit například její paměťovou náročnost. Testování algoritmu jsem provedl na datech dodaných vedoucím práce, ale také na datech vlastních nebo použitých v rámci soutěže *SequenceSqueeze*. Na testovaných datech implementovaná metoda dosahovala komprese mezi 13 % a 24 %, čímž dosáhla lepších výsledků než standardní kompresní algoritmy jako GZip, který se v praxi pro ukládání FASTQ dat běžně používá. Hlavní nevýhodou této metody je její časová náročnost, jejíž možná řešení jsem rozebral v diskuzi. Vzhledem k uspokojivým výsledkům testování mé implementace věřím, že by na tuto práci mohli navázat další studenti a časem tak vytvořit nástroj dosahující kvalitní komprese s nižšími nároky na čas.

Literatura

[1] *The Cost of Sequencing a Human Genome*. [online]. National Human Genome Research Institute. 30. říj. 2019. URL: https://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost (cit. 04. 07. 2020).

[2] *Sequence File Formats*. [online]. Illumina. URL: https://emea.illumina.com/informatics/sequencing-data-analysis/sequence-file-formats.html (cit. 10. 07. 2020).

[3] *FASTA format*. [online]. The Yang Zhang Lab: University of Michigan. URL: https://zhanglab.ccmb.med.umich.edu/FASTA/ (cit. 11. 07. 2020).

[4] *FASTQ Files*. [online]. BaseSpace Sequence Hub. URL: https://help.basespace.illumina.com/articles/descriptive/fastq-files/ (cit. 09. 07. 2020).

[5] *Small Compression Toolkit*. [online]. URL: https://gitlab.fit.cvut.cz/polacrad/sct (cit. 10. 07. 2020).

[6] *GitLab FIT*. [online]. URL: https://gitlab.fit.cvut.cz/ (cit. 09. 07. 2020).

[7] Jiří Bican. "Implementace vylepšení kompresní metody ACB v jazyce Java". Magisterská práce. České vysoké učení technické v Praze, 2017.

[8] Achraf El Allali a Mariam Arshad. "MZPAQ. a FASTQ data compression tool". In: *Source Code for Biology and Medicine* 14.1 (2019). ISSN: 1751-0473. DOI: 10.1186/s13029-019-0073-5. URL: https://scfbm.biomedcentral.com/articles/10.1186/s13029-019-0073-5 (cit. 12. 07. 2020).

[9] James K. Bonfield, Matthew V. Mahoney a Michael Gormley. "Compression of FASTQ and SAM Format Sequencing Data". In: *PLoS ONE* (2013-3-22). ISSN: 1932-6203. DOI: 10.1371/journal.pone.0059190. URL: https://dx.plos.org/10.1371/journal.pone.0059190 (cit. 19. 07. 2020).

[10] *Stringology, lexikon pojmů*. [online]. URL: http://www.stringology.org/DataCompression/lexikon.html (cit. 18. 07. 2020).

[11] *FASTQ files explained*. [online]. Illumina. URL: https://emea.support.illumina.com/bulletins/2016/04/fastq-files-explained.html (cit. 07. 07. 2020).

[12] Günter Kahl. *The dictionary of genomics, transcriptomics and proteomics*. Fifth, greatly enlarged edition. Weinheim, [2015]. ISBN: 978-352-7328-529. DOI: 10.1002/9783527678679.

[13] *Paired-End vs. Single-Read Sequencing Technology*. [online]. URL: https://emea.illumina.com/science/technology/next-generation-sequencing/plan-experiments/paired-end-vs-single-read.html (cit. 12. 07. 2020).

[14] *Sequencing Read Length*. [online]. Illumina. URL: https://www.illumina.com/science/technology/next-generation-sequencing/plan-experiments/read-length.html (cit. 12. 07. 2020).

[15] *Sequencing Platforms Archives*. [online]. AllSeq. URL: https://allseq.com/kb-category/sequencing-platforms/ (cit. 12. 07. 2020).

[16] Aníbal Guerra, Jaime Lotero a Sebastián Isaza. "Performance comparison of sequential and parallel compression applications for DNA raw data". In: *The Journal of Supercomputing* 72.12 (2016), s. 4696–4717. ISSN: 0920-8542. DOI: 10.1007/s11227-016-1753-4. URL: http://link.springer.com/10.1007/s11227-016-1753-4 (cit. 14. 07. 2020).

[17] *Hartwig Medical Foundation - GitHub*. [online]. URL: https://github.com/hartwigmedical (cit. 12. 07. 2020).

[18] *Genotyp Zostera marina*. [online]. European Nucleotide Archive. URL: https://www.ebi.ac.uk/ena/data/view/SRR2980545 (cit. 14. 07. 2020).

[19] Ján Bobot. "Implementace kompresních metod LZY, LZMW a LZAP v jazyce Java". Bakalářská práce. České vysoké učení technické v Praze, 2019.

[20] Matt Mahoney. *Context Mixing*. [online]. URL: http://mattmahoney.net/dc/dce.html#Section_43 (cit. 15. 07. 2020).

[21] Matt Mahoney. *The PAQ1 Data Compression Program*. [online]. [2002]. URL: http://www.mattmahoney.net/dc/paq1.pdf (cit. 15. 07. 2020).

[22] Matt Mahoney. *Adaptive Weighing of Context Models for Lossless Data Compression*. [online]. [2005]. URL: https://pdfs.semanticscholar.org/f758/d8e6dee5348c2b5bf56f33994ad96e88a591.pdf (cit. 11. 07. 2020).

[23] Eberhard Passarge. *Color atlas of genetics*. Fifth edition, revised and updated. Stuttgart: Thieme, [2018]. ISBN: 978-313-2414-402.

[24] *Aritmetické kódování*. [online]. URL: http://www.stringology.org/DataCompression/ak-np/index_cs.html (cit. 16. 07. 2020).

Příloha A

Seznam použitých zkratek

FASTQ Formát souborů s genomickými daty

NGS New generation sequencing

CR Kompresní poměr (Compression ratio)

DNA Deoxyribonukleová kyselina

API Rozhraní pro programovací aplikace

QS Skóry kvality

PAQ Rodina algoritmů bezeztrátové komprese na bázi mixování kontextů

SET Short-read single-end tag

PET Long-read paired-end tag

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy
  - `thesis` ..... diplomová práce ve formátu LaTeX
- `text`
  - `DP_novak_jakub_2020.pdf` ..... text práce ve formátu PDF