Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Tomáš Šurín

Kompresia biologických sekvencií

Kabinet software a výuky informatiky

Vedoucí diplomové práce: RNDr. František Mráz, CSc.

Studijní program: Informatika

Studijní obor: Teoretická informatika

Praha 2012

Ďakujem RNDr. Františkovi Mrázovi, CSc. za odborné vedenie mojej práce, za čas, ktorý mi behom vypracovania tejto práce venoval, a za cenné rady, ktoré prispeli k skvalitneniu tejto práce. Taktiež ďakujem ľuďom, ktorí mi pomohli s korektúrami.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 13.7.2012 Tomáš Šurín

Název práce: Kompresia biologických sekvencií

Autor: Tomáš Šurín

Katedra: Kabinet software a výuky informatiky

Vedoucí diplomové práce: RNDr. František Mráz, CSc.

Abstrakt: Objem dát získavaných sekvenovacími technológiami novej generácie rastie podstatne rýchlejšie ako kapacity úložných médií. Výstupy sekvenátorov okrem samotných prečítaných krátkych úsekov DNA obsahujú ďalšie informácie napr. o spoľahlivosti/kvalite čítania každého symbolu DNA. Takéto dáta je treba ďalej uchovávať aj po zostavení sekvencie kompletného genómu. Štandardným formátom pre ukladanie dát v tejto oblasti je formát SAM (Sequence Alignment/Mapping Format) a jeho binárna komprimovaná verzia BAM, ktorá umožňuje náhodný prístup k svojmu obsahu. V tejto práci popíšeme konštrukciu lepšej kompresnej schémy pre bezstratovú kompresiu súborov vo formáte SAM/BAM. Táto kompresná schéma dosahuje podstatne menšiu veľkosť komprimovaného súboru ako pri formáte BAM. Navyše však zostáva zachovaná možnosť náhodného prístupu k dátam v komprimovanom súbore. Implementácia tejto kompresnej schémy je platformovo nezávislá a umožňuje jednoduchú konfiguráciu použitých kompresných metód. Navrhovali sme ju tak, aby bola v budúcnosti možná jej jednoduchá rozšíriteľnosť – vďaka tomu bude možné reagovať na zmeny v sekvenovacích platformách, ako aj na zmeny vo formáte SAM.

Klíčová slova: bioinformatika, kompresia, DNA, formát SAM

Title: Compression of biological sequences

Author: Tomáš Šurín

Department: Department of Software and Computer Science Education

Supervisor: RNDr. František Mráz, CSc.

Abstract: Volumes of data obtained from the next generation sequencing platforms is growing faster than the available capacity of storage media. Sequencers mainly produce short reads of DNA. However, output of the sequencing machines also contains other information, for example information about read reliability-/quality. This data must be archived even after successful complete genome assembly. Standard file format used for this type of data is format SAM (Sequence Alignment/Mapping Format) and its binary compressed version BAM. In this thesis we describe the construction of a better lossless compression scheme for compression of files in the SAM/BAM format. This compression scheme provides better compression ratios than the BAM format. In addition, random access to data in the compressed file is retained. Implementation of this compression scheme is platform independent and allows simple configuration of the compression process. Implementation also offers easy extensibility. Thanks to this, we will be able to respond to changes in current sequencing platforms as well as to changes in the SAM format.

Keywords: bioinformatics, compression, DNA, SAM format

Obsah

1. Úvod

Dokončenie projektu sekvenovania kompletného ľudského genómu v roku 2003 bolo počiatkom novej éry výskumu v biológii. Tento výskum priniesol nástroje, ktoré umožňujú skúmať obsah a funkcie genetického kódu veľkého množstva organizmov. Možnosť priameho prístupu k sekvencii DNA umožní lepšie pochopenie fundamentálnych procesov prebiehajúcich v živých bunkách. Predpokladá sa taktiež veľký dopad na medicínu – znalosť genómu pacienta umožní lepšie cielenú liečbu a spoľahlivejšiu a rýchlejšiu diagnostiku geneticky podmienených chorôb.

Projekt sekvenovania kompletného ľudského genómu mal rozpočet 3 miliardy dolárov. Nástup sekvenovacích technológií novej generácie umožnil radikálne zlacnenie sekvenovania DNA. V súčasnosti je možné vytvoriť sekvenciu kompletného ľudského genómu s cenou na hranici tisícok dolárov [39]. Preto nie je prekvapivé, že objem bioinformatických dát rastie podstatne rýchlejšie ako kapacity úložných médií [47]. Dostupné technológie sa navyše rýchlo vyvíjajú. V blízkej budúcnosti je očakávané dosiahnutie ceny 1000 a menej dolárov za osekvenovanie kompletného ľudského genómu. Môžeme teda predpokladať, že objem sekvenovaných dát bude v budúcnosti rásť ešte rýchlejšie ako v súčasnosti.

Získané sekvencie sú po vytvorení analyzované. Typickým príkladom sú komparatívne štúdie s cieľom nájsť genetické variácie medzi jedincami rovnakého druhu. Pre potreby analýzy je často potrebné prenášať biologické sekvencie po sieti. Pri prenose po sieti nás zaujíma najmä čas prenosu. Kvôli jeho zníženiu je vhodné prenášané dáta komprimovať.

Po samotnej analýze je veľmi často potrebné získané dáta dlhodobo archivovať. Pre potreby archivácie biologických sekvencií je možné využiť viacero prístupov. Najjednoduchšie je zvýšenie kapacity úložných médií. Ako sme však spomínali, objem dát v bioinformatike rastie rýchlejšie ako kapacity týchto médií. Preto je tento prístup bez jeho kombinácie s iným prístupom dlhodobo neudržateľný.

Ďalšou možnosťou je odstránenie nepotrebných alebo menej potrebných dát. Napríklad je možné po analýze sekvencie DNA túto sekvenciu odstrániť. Toto je možné iba v prípade, že je ľahko dostupná biologická vzorka, z ktorej bola vytvorená táto sekvencia. Problém nastane, ak je problematické získanie vzorky – napr. pri výskume nádorových alebo iných vzácnejších ochorení. Ďalej je možné ukladať iba zaujímavé časti sekvencií (napr. oblasti, ktoré kódujú známe gény) alebo využiť stratovú kompresiu.

Jednoduchý prístup k biologickým dátam prakticky z ľubovoľného miesta je možné získať po ich uložení do centralizovaného úložiska. Príkladom takýchto úložísk sú NCBI[1] alebo EBI[2]. Pri spracovávaní nám potom stačí prenášať sieťou iba časť dát, ktorá je pre nás v danej chvíli relevantná. Tento prístup nemusí byť vhodný pre citlivé dáta, menšie alebo komerčné projekty.

Prístupom k archivácii biologických dát, ktorým sa zaoberáme v tejto práci je využitie kompresie dát. Nevýhodou tohto prístupu je zníženie transparentnosti

[1] <http://www.ncbi.nlm.nih.gov/>
[2] <http://www.ebi.ac.uk/>

pri manipulácii s dátami a zvýšenie výpočtovej náročnosti práce s týmito dátami. Vhodnou kombináciou uvedených metód môžeme dosiahnuť vhodný kompromis medzi množstvom ukladaných dát a finančnou náročnosťou ich archivácie.

Existuje pomerne veľké množstvo prác, ktoré sa zaoberajú kompresiou samotných sekvencií DNA. Avšak dáta, ktoré je potrebné uchovávať obsahujú aj ďalšie informácie. Štandardom pre ukladanie dát získaných zo sekvenovacích technológií novej generácie je formát SAM (Sequence Alignment/Mapping Format) a jeho binárna komprimovaná verzia BAM. Súbor vo formáte SAM/BAM obsahuje okrem osekvenovaných báz aj informácie o kvalitách týchto báz, informácie o zarovnaní prečítaných segmentov k referenčnej sekvencii a ďalšie informácie. Súbory vo formáte BAM sú rozdelené na bloky, ktoré sú indexované. Toto umožňuje náhodný prístup k dátam v týchto súboroch. To je nutné, pretože súbory BAM dosahujú gigabytové veľkosti a bez indexu umožňujú len sekvenčný prístup.

Cieľom tejto práce je vytvorenie vhodnej kompresnej schémy pre kompresiu súborov vo formátoch SAM a BAM. Navyše je naším cieľom zachovať rýchly náhodný prístup k dátam vo výslednom komprimovanom súbore. Našou víziou je, že riešenie vytvorené v tejto práci bude dostupné jednotlivcom, ktorí budú schopní vykonávať pomerne zložité štúdie na štandardnom osobnom počítači s dostupným dátovým úložiskom. Budeme sa preto snažiť o vytvorenie rozšíriteľného a platformovo nezávislého riešenia, ktoré jednoducho zapadne do dostupného ekosystému programov používaných pre prácu s formátmi SAM a BAM.

1.1 Štruktúra práce

Implementáciou kompresnej schémy, ktorá je výsledkom tejto práce je aplikácia SamZip.

Práca začína uvedením do problematiky z pohľadu bioinformatiky v kapitole 2. V kapitole 3 zase uvedieme základné pojmy z teórie kompresie dát a princíp používaných kompresných metód. V kapitole 4 uvedieme existujúce prístupy ku kompresii sekvencií DNA, v kapitole 5 popíšeme formát súborov SAM/BAM a v kapitole 6 uvedieme popis existujúcich prác, ktoré sa venujú kompresii SAM súborov. V kapitole 7 uvedieme detaily implementácie aplikácie SamZip. Nosnou časťou práce je kapitola 8, v ktorej popíšeme proces výberu vhodnej kompresnej schémy. Ďalej v kapitole 9 uvedieme výsledky našej metódy na väčších reálnych dátach spolu s porovnaním s dostupnou implementáciou. Nakoniec v kapitole 10 zhodnotíme dosiahnuté výsledky a reálny prínos tejto práce.

Práca je doplnená o viaceré dodatky. V dodatku A popíšeme obsah priloženého média. Dodatok B predstavuje užívateľskú príručku aplikácie SamZip pre užívateľov, ktorí nepotrebujú meniť parametre použitej kompresnej schémy. V dodatku C tieto informácie doplníme o informácie pre pokročilejších užívateľov, ktorí chcú zmeniť parametre našej kompresnej schémy a prispôsobiť si ju tak svojim potrebám. Nakoniec v dodatku D uvedieme návod pre užívateľov, ktorí potrebujú rozšíriť možnosti programu SamZip implementovaním nových modulov.

2. Úvod do bioinformatiky

V tejto kapitole popíšeme základné koncepty z oblasti bioinformatiky potrebné pre pochopenie zvyšku práce. V sekcii 2.1 popíšeme základy molekulárnej biológie. V sekcii 2.2 sa budeme venovať metódam sekvenovania DNA. Dáta získané pri sekvenovaní sa uchovávajú v rôznych formátoch. Najvýznamnejšie z nich popíšeme v sekcii 2.3. Nakoniec v sekcii 2.4 popíšeme niektoré projekty sekvenovania kompletných genómov.

2.1 Úvod do molekulárnej biológie

V tejto sekcii popíšeme základy molekulárnej biológie. Čerpali sme z kníh, ktoré ponúkajú úvod do oblasti bioinformatiky [32], [27] a z učebnice biológie [29], ktorá je voľne dostupná online.

Základnou funkčnou jednotkou každého živého systému je bunka. Bunky sú riadené zložitým systémom chemických reakcií, ktorých základom je syntéza proteínov. Každá bunka obsahuje návod pre syntézu proteínov a pre svoju replikáciu – genetickú informáciu. Skoro všetky bunky v organizme obsahujú rovnakú genetickú informáciu. Nositeľom genetickej informácie sú nukleové kyseliny – DNA (deoxyribonukleová kyselina) a RNA (ribonukleová kyselina). Úlohou DNA je uchovávať genetickú informáciu. Základnou úlohou RNA je prenášať genetickú informáciu z DNA k miestu jej využitia.

V časti 2.1.1 popíšeme štruktúru molekúl DNA a RNA. V časti 2.1.2 potom uvedieme ako tieto molekuly vytvárajú genetickú informáciu organizmov. Nakoniec v časti 2.1.3 uvádzame ako sa podieľajú DNA/RNA molekuly na syntéze proteínov.

2.1.1 Štruktúra DNA/RNA

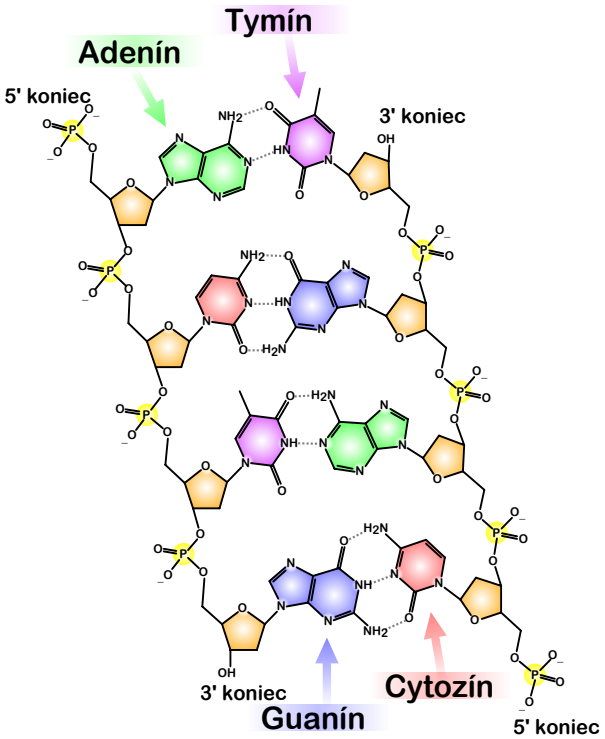
Molekuly DNA a RNA sú polyméry. Sú tvorené jednotkami nazývanými *nukleotidy*. Nukleotidy pozostávajú z kostry, ktorá je tvorená zo zvyšku kyseliny fosforečnej (PO_4^{3-}) a cukru ribózy (RNA) alebo deoxyribózy (DNA). Na túto kostru sú naviazané *dusíkaté bázy* (v ďalšom texte označované len ako bázy). Práve bázy tvoria kód pre uchovávanie a prenášanie genetickej informácie – *genetický kód*. V DNA sa vyskytujú 4 typy báz: adenín (A), guanín (G), tymín (T), cytozín (C). V RNA je tymín nahradený uracylom (U).

Molekula RNA je tvorená jedným vláknom. Molekula DNA je tvorená dvomi vláknami. Na protiľahlých pozíciach v týchto vláknach sa vždy viažu rovnaké dvojice báz (*komplementárne bázy*): adenín – tymín a cytozín – guanín. Tento jav vyplýva z chemickej štruktúry nukleotidov[1]. V každom z vlákien jednej molekuly DNA je uložená rovnaká informácia – zo znalosti jedného vlákna je možné

[1] Adenín a tymín sú vzájomne spojené dvomi vodíkovými mostíkmi, cytozín a guanín tromi

odvodiť druhé vlákno jednoduchým nahradením originálnych báz ich komplementárnymi bázami. Jednotlivé vlákna môžu mať spojené konce – takáto molekulu DNA/RNA sa nazýva *kruhová*. Ak vlákna nemajú spojené konce, tak sa taká molekula DNA/RNA nazýva *lineárna*. Molekula RNA sa môže viazať na molekulu DNA, čo je využívané pri syntéze proteínov (skopírovanie oblasti DNA). Na obrázku 2.1 uvádzame schému primárnej[2] štruktúry molekuly DNA.

Na lineárnej molekule DNA/RNA rozlišujeme 2 konce - 5'[3] a 3'[4]. Konce vlákien molekuly DNA sú k sebe orientované opačne (tj. ak jedno vlákno je orientované ako 5' → 3', druhé je orientované ako 3' → 5'). Existuje konvencia, že bázy molekúl DNA/RNA čítame od 5' konca k 3' koncu. Táto orientácia je používaná aj pri ukladaní osekvenovaných sekvencií v digitálnej podobe.



Obr. 2.1: Primárna štruktúra molekuly DNA (Zdroj: *upravená schéma od Madeleine Price Ball, PhD.*)

2.1.2 Genetická informácia

Kompletnú sadu genetickej informácie organizmu nazývame *genóm*. Veľkosť genómu udávame ako súhrnný počet bázových párov (*bp*) DNA v bunke. Genóm jednoduchších organizmov tvorí z veľkej časti kruhová DNA voľne plávajúca v bunke alebo RNA v prípade vírusov. U zložitejších organizmov je situácia podstatne

[2]Primárna štruktúra nezohladňuje priestorové usporiadanie molekuly DNA, ktorá je v bunke väčšinou vo forme dvojzávitnice

[3]Ukončený zvyškom kyseliny fosforečnej

[4]Ukončený hydroxidovou (OH) skupinou z ribózy resp. deoxyribózy

komplikovanejšia. Ich genóm tvoria *chromozómy*, ktoré obsahujú zvinuté molekuly DNA. Chromozómy sa nachádzajú v jadre bunky. Počet chromozómov závisí od živočíšneho druhu. Napríklad človek má 23 párov chromozómov.

Poznámka 2.1. Genóm sa výhradne nenachádza len vo vyššie uvedených štruktúrach. Napríklad kúsky DNA/RNA sa môžu nachádzať v cytoplazme bunky alebo v mitochondriách (tzv. mitochondriálna DNA).

Základnou jednotkou genetickej informácie je *gén*. Jedná sa o sekvenciu báz DNA, ktorá kóduje špecifický proteín. U zložitejších organizmov zvyčajne gén pozostáva z viacerých kódujúcich častí (*exóny*) oddelených od seba nekódujúcimi časťami (*intróny*). Dôležitou súčasťou génu je regulačná oblasť nachádzajúca sa pred prvým exónom génu. Vďaka tejto oblasti je bunka schopná zistiť, kde má v DNA sekvencii začať so syntézou proteínov.

Poznámka 2.2. Ukazuje sa, že gén nekóduje práve 1 proteín, ako bolo uvedené vyššie, ale môže kódovať aj viacero proteínov alebo nekódovať žiadny. Pre naše potreby však postačuje vyššie uvedená predstava.

U človeka dosahuje genóm veľkosť 3 miliardy *bp*. Avšak len malá časť z tohto rozsahu kóduje gény. Tých obsahuje ľudský genóm asi $20000 - 25000$. Genómy jedincov určitého druhu sú si navzájom veľmi podobné. Napr. genóm dvoch ľudí sa líši len v asi 3 miliónoch *bp* (asi 0.1%).

2.1.3 Syntéza proteínov

Proteíny sú základnou stavebnou jednotkou bunky. Majú však aj ďalšie funkcie. Vo forme enzýmov slúžia napríklad na reguláciu chemických reakcií bunky.

Proteíny sú polyméry. Ich základnou stavebnou jednotkou sú *aminokyseliny*. Je známych 20 aminokyselín, ktoré sa podieľajú na ich syntéze. Postupnosť aminokyselín v proteíne je kódovaná genetickým kódom. Aby bolo možné zakódovať 20 známych aminokyselín kódom s abecedou $\{A, C, G, T\}$, tak sú potrebné kódové slová o dĺžke 3^5. Tieto kódové slová nazývame *kodóny*. Je vidieť, že tento kód je degenerovaný, tj. viac kodónov kóduje jednu aminokyselinu.

Pri syntéze proteínov zohrávajú dôležitú úlohu nasledujúce typy molekúl RNA:

mRNA (mediátorová RNA) Obsahuje postupnosť báz, ktoré kódujú proteín. Používa sa pre prenos genetickej informácie z DNA do proteínov.

tRNA (prenosová RNA) Je molekula RNA, slúžiaca pre prenos aminokyselín na miesto tvorby proteínov. Každá molekula tRNA prenáša špecifickú aminokyselinu a obsahuje komplementárny kodón ku kodónu, ktorý kóduje túto aminokyselinu. Molekuly tRNA sa nachádzajú voľne v cytoplazme bunky.

Zjednodušený proces syntézy proteínov je znázornený na obrázku 2.2. Pozostáva z nasledujúcich fází:

[5] Kódové slová o dĺžke 2 dokážu kódovať len $2^4 = 16$ kombinácií

1. **Transkripcia** Vytvorenie RNA molekuly z oblasti DNA, ktorá obsahuje gén kódujúci požadovaný proteín. Výsledkom tohto kroku je RNA, ktorá obsahuje aj intróny aj exóny.

2. **Splicovanie** Odstránenie intrónov z RNA získanej v predchádzajúcom kroku. Výsledkom splicovania je molekula mediátorovej RNA.

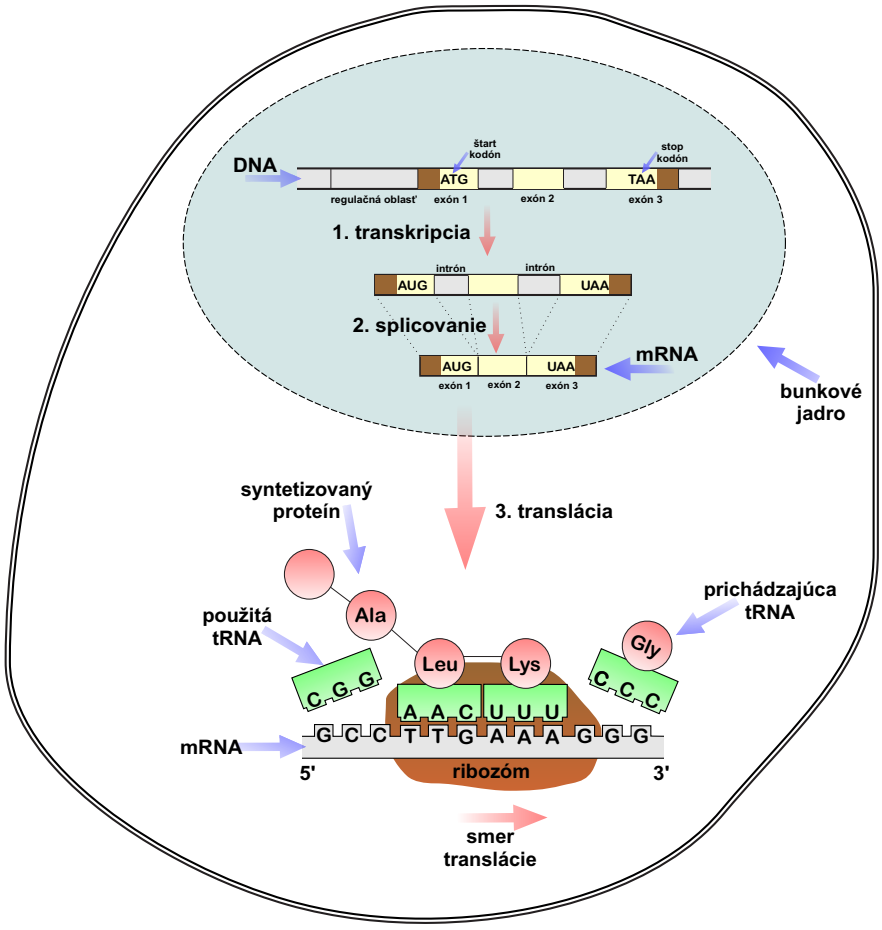
3. **Translácia** Samotná syntéza proteínov prebieha na ribozómoch v cytoplazme bunky v smere 5' → 3'. Každý kodón je spracovaný nasledovne:

   (a) Naviazanie vhodnej tRNA molekuly, ktorá obsahuje antikodón ku kodónu na aktuálnej pozícii v mRNA.

   (b) Aminokyselina asociovaná s touto tRNA je pripojená k reťazcu aminokyselín práve vytváraného proteínu.

   (c) Posunutie aktuálnej pozície v mRNA.

   Ak sa v mRNA na aktuálnej pozícii nachádza jeden z 3 špeciálnych kodónov, tzv. stop kodónov, syntéza proteínu končí.



Obr. 2.2: Schematické znázornenie procesu syntézy proteínov

2.2 Sekvenovanie DNA

Možnosť získať sekvencie DNA má veľký význam v biologických výskumoch. Príkladom môžu byť evolučné štúdie vývoja organizmov alebo hľadanie génov. U človeka umožňuje získanie genómu veľkého množstva ľudí výskum genetických porúch a zistenie dopadu konkrétnych zmien v genóme na zdravie. V tejto sekcii popíšeme proces sekvenovania DNA a metódy, ktoré sa pre tento účel používajú.

Najskôr v časti 2.2.1 uvedieme definície pojmov, ktoré budeme používať v ďalšom texte. V časti 2.2.2 popíšeme princíp sekvenovania a informácie o existujúcich metódach. V častiach 2.2.3 a 2.2.4 uvedieme informácie o zarovnávaní a zostavovaní osekvenovaných sekvencií. Pri sekvenovaní a zostavovaní DNA sekvencií sa vyskytujú chyby, preto je nutné sledovať kvalitu čítania každej bázy. V časti 2.2.5 uvedieme, ako je táto kvalita reprezentovaná. Nakoniec v časti 2.2.6 popíšeme aké typy dát sú produkované pri sekvenovaní.

2.2.1 Použitá terminológia

Sekvenovanie DNA je proces získania postupnosti báz DNA molekuly. Výsledkom sekvenovania je *sekvencia* DNA. Prístroj, ktorý umožňuje sekvenovanie budeme nazývať *sekvenátor*.

Knižnica (library) je označenie pre vhodne pripravenú vzorku DNA, ktorú chceme osekvenovať. Najčastejšie sa jedná o množstvo malých fragmentov DNA (*šablón*) pripravených vhodným technologickým postupom.

Segment je spojitá podsekvencia šablóny, ktorú prečítal sekvenátor. Každá šablóna môže pozostávať z viacerých segmentov (napr. pri párovom čítaní, viď sekcia 2.2.4). Šablóna nemusí byť pri sekvenovaní prečítaná celá.

Pokrytie (coverage) je priemerný počet segmentov pokrývajúcich danú pozíciu v sekvencii. Vysoké pokrytie je potrebné pre minimalizovanie množstva chýb pri zostavovaní sekvencie. Obyčajne sa používa značenie typu: 2X pre dvojnásobné pokrytie – inými slovami, každá báza je priemerne pokrytá dvomi prečítanými segmentmi.

2.2.2 Metódy sekvenovania DNA

Molekula DNA má príliš malú veľkosť na to, aby sme boli schopní prečítať poradie jej báz priamo (tj. použitím optického alebo elektrónového mikroskopu). Preto je základom najrôznejších sekvenovacích metód šikovné využitie známych biologických a chemických procesov.

Príprava knižnice vzoriek

Aby sme boli schopní osekvenovať genóm zvoleného organizmu potrebujeme dostatočné množstvo vzoriek odobratých tomuto organizmu. Tieto nespracované vzorky však nie je možné priamo osekvenovať. Najskôr ich je potrebné vhodne

predspracovať. Pre výsledok predspracovania sa obvykle používa označenie knižnica vzoriek.

Väčšina používaných sekvenovacích metód využíva pri príprave knižnice vzoriek podobné kroky:

1. **Shotgun sekvenovanie**. DNA molekuly sú až na niekoľko výnimiek[6] príliš dlhé. Preto je potrebné DNA rozbiť na menšie, prelínajúce sa fragmenty. Tento proces sa nazýva shotgun sekvenovanie. Názov je odvodený od podobnosti tohto prístupu k pomerne náhodnému rozptylu nábojov pri výstrele z brokovnice. Výsledkom shotgun sekvenovania sú krátke fragmenty DNA, ktoré budeme nazývať *šablóny* (templates).

2. **Klonovanie šablón**. Aby bolo technologicky možné detekovať výsledky reakcií prebiehajúcich pri sekvenovaní, je potrebné, aby prebiehali v dostatočne veľkom množstve. Preto potrebujeme mať možnosť vytvoriť kópie šablón, ktoré chceme sekvenovať.

   Príkladom metódy pre klonovanie šablón je PCR (polymerase chain reaction). Je založená na znalosti krátkych sekvencií DNA, tzv. primérov, ktoré sú špeciálne vytvorené tak, aby sa naviazali na začiatok a koniec požadovanej oblasti DNA[7]. Pri PCR sú vlákna molekuly DNA navzájom oddelené pôsobením zvýšenej teploty. Po pridaní dostatočného množstva nukleotidov, spomínaných primérov a DNA polymerázy[8] je vytvorená dvojvláknová DNA z pôvodnej jednovláknovej. Opakovaním tohto procesu docielime exponenciálne množenie šablón.

Klasické metódy sekvenovania

Metóda používaná pri sekvenovaní prvého ľudského genómu bolo Sangerovo sekvenovanie [44].

Sangerovo sekvenovanie pozostáva z nasledujúcich krokov:

1. Denaturácia dvojvláknovej DNA na jednotlivé vlákna zahriatím. Výsledkom je jednovláknová DNA.

2. K denaturovanej DNA je pridaná DNA polymeráza, vhodný primer, klasické nukleotidy a špeciálne upravené nukleotidy. Upravené nukleotidy sú fluorescenčne označené tak, aby každý typ bázy emitoval svetlo s inou vlnovou dĺžkou, a aby kopírovanie DNA nemohlo po ich začlenení do sekvencie pokračovať. Vhodným pomerom týchto upravených nukleotidov a ich klasických variant je docielený vznik fragmentov všetkých dĺžok. Platí, že každý fragment je predčasne ukončenou komplementárnou sekvenciou k sekvenovanej DNA. Na konci každého fragmentu sa nachádza upravená báza.

3. Vzniknuté fragmenty sú vložené do gélu, v ktorom sa elektroforézou zoradia podľa dĺžky. Tento proces je schopný rozlíšiť fragmenty, ktorých dĺžky sa

[6] Napríklad u jednoduchých baktérií

[7] Obsahujú postupnosť komplementárnych báz k postupnostiam v týchto oblastiach

[8] Enzým, ktorý umožňuje kopírovanie DNA

líšia len jednou bázou. Neskôr bol tento krok nahradený kapilárnou elektroforézou (rozdelenie fragmentov podľa dĺžky pomocou kapiláry), ktorá urýchlila sekvenovanie.

4. Vhodným zariadením (napr. laserom) je prečítaná sekvencia báz.

Obmedzením tejto technológie je maximálna dĺžka segmentu 800 bp. Jeden beh Sangeroveho sekvenovania trvá približne 3 hodiny, pričom sekvenátor zväčša umožňuje paralelne osekvenovať desiatky segmentov za jeden beh [50].

Metódy sekvenovania novej generácie

Od roku 2005 sa začali vo väčšej miere používať modernejšie metódy sekvenovania. Priniesli vyššiu rýchlosť (vysoká paralelizácia sekvenovania), avšak za cenu produkovania kratších segmentov. Vývoj v oblasti algoritmov pre zostavovanie sekvencií však umožnil nenákladné zostavovanie sekvencií aj z takýchto krátkych segmentov. Tieto vlastnosti platforiem novej generácie umožnili lacnejšie sekvenovanie a následné rozšírenie sekvenátorov aj do menších laboratórií.

Podrobnejšie informácie o sekvenovacích metódach novej generácie (niekedy označované aj ako druhá generácia) je možné nájsť v prehľadových štúdiách [36], [41] alebo [50].

Budúcnosť sekvenovania

V súčasnosti prebieha vývoj a nasadzovanie nových sekvenovacích technológií. Mali by podporovať čítanie sekvencie už pri výskyte jednej molekuly DNA (Single molecule sequencing – SMS) a tiež by mali produkovať dlhšie segmenty. Tieto metódy sú občas označované ako metódy tretej generácie.

Cieľom súčasného vývoja je zlacnenie sekvenovania celého ľudského genómu na hranicu 1000 dolárov. Príkladom môže byť súťaž Archon X Prize [1], ktorej víťaz by mal byť schopný osekvenovať 100 kompletných ľudských genómov pod 30 dní za cenu menšiu ako 1000 dolárov za jeden genóm.

Pre predstavu množstva dát, ktoré produkujú moderné sekvenovacie technológie, uvádzame v tabuľke 2.1 parametre vybraných sekvenovacích platforiem.

2.2.3 Zarovnávanie sekvencií

Základným problémom bioinformatiky je hľadanie podobností v osekvenovaných dátach. Dôvodom výskytu týchto podobností môžu byť napr. mutácie alebo spoločný evolučný predok. *Zarovnanie* je spôsob zápisu sekvencií, v ktorom sú pod sebou v riadkoch zapísané jednotlivé sekvencie doplnené o medzery (postupnosti prázdnych báz) tak, aby boli zapísané riadky rovnako dlhé a v stĺpcoch sa čo najčastejšie nachádzali rovnaké bázy.

Zarovnávanie sekvencií (sequence alignment) je proces, ktorého cieľom je vytvoriť zarovnanie sekvencií. Kvalitu zarovnania určujeme podľa zvoleného skórovania. Zarovnanie 2 sekvencií nazveme *párové* a zarovnanie viacerých sekvencií *viacnásobné*.

Platforma	Dĺžka prečítaného segmentu	Čas behu	Produkcia dát za beh
Platformy druhej generácie			
Roche 454™	až 1000 bp	23 hodín	700 Mbp
Solexa Illumina™	2×100 bp	1.5 – 11 dní	až 600 Gbp
ABI SOLiD™	2×50 bp	14 dní	50 Gbp
Multiplex Polony	28 bp	4 dni	10 – 35 Gbp
Platformy tretej generácie			
SMRT™	2000 bp	1 hodina	100 Mbp
Heliscope™	55 bp	8 dní	21 – 35 Gbp
IonTorrent™	>400 bp	<2 hodiny	>1 Gbp

Tab. 2.1: Parametre vybraných sekvenovacích platforiem novej generácie (Zdroj: [36] a [48])

Príklad 2.1. Uvažujme skórovanie, ktoré hodnotí zhodu báz ako +1, nezhodu penalizuje hodnotou −3 a medzeru hodnotí ako −2. Príklad párového zarovnania 2 sekvencií:

```
AGATTTGGC--TCTGTGAGCTTTTTCAGA
AGAC--GGCGGTCTG--AGGTTT-TCCGA
```

Skóre tohto zarovnania podľa vyššie uvedeného skórovania je $19 \times (+1) + 3 \times (-3) + 7 \times (-2) = -4$.

Bližšie informácie o algoritmoch pre zarovnávanie sekvencií, ktoré sa používajú pri sekvenovaní novej generácie, je možné nájsť napríklad v prehľadovej štúdii [35].

2.2.4 Zostavenie sekvencií

Výstup shotgun sekvenovania je tvorený veľkým množstvom krátkych segmentov. Proces poskladania týchto segmentov do súvislej sekvencie budeme nazývať zostavenie (assembly).

V praxi sú používané dve stratégie zostavovania sekvencií:

- **Zostavenie de novo.** Tento prístup je vhodný aj v prípade sekvenovania úplne nového organizmu.
- **Zarovnanie k známej referenčnej sekvencii**, kedy ku každému prečítanému segmentu nájdeme najlepšie zarovnanie k známej referenčnej sekvencii. Výhodou je výrazne nižšia cena a výpočtová náročnosť než v prípade de novo zostavenia. Nevýhodou tohto prístupu je, že potrebujeme mať k dispozícii dostatočne kvalitnú sekvenciu rovnakého alebo podobného organizmu.

Zostavovanie nám sťažujú viaceré aspekty:

- krátke prečítané segmenty,

- veľká dĺžka zostavovanej sekvencie,
- veľa opakujúcich sa segmentov,
- chyby v prečítaných segmentoch,
- výskyt segmentov z nesúvisiaceho úseku DNA.

Pri sekvenovaní novej generácie je veľmi často používané tzv. párové čítanie s cieľom zjednodušiť zostavovanie. Výsledkom párového čítania šablóny je dvojica segmentov prečítaných z oboch koncov šablóny. Kedže poznáme očakávanú veľkosť šablóny, je možné túto informáciu využiť pri zostavovaní výslednej sekvencie.

2.2.5 Kvalita sekvenovaných dát

Z dát získaných pri sekvenovaní nie je možné s úplnou istotou určiť požadovanú bázu. Pre stanovenie spoľahlivosti, s akou bola určená báza pri sekvenovaní sa používa systém kvalít. Najčastejšie je používané skórovanie PHRED [18, 19]. Je definované ako:

$$q = -10 \log_{10} P(\text{báza je chybná}), \tag{2.1}$$

kde $P(\text{báza je chybná})$ je pravdepodobnosť toho, že báza je určená chybne.

Príklad 2.2. Ak je udávaná kvalita $q = 10$, tak je báza určená správne na 90%. Pri kvalite $q = 20$ na 99% a pri kvalite $q = 60$ na 99.9999%. Za dostatočne vysokú kvalitu bázy sú považované hodnoty $q \geq 20$.

2.2.6 Produkované dáta

V tejto sekcii popíšeme hlavné typy dát, ktoré sú produkované pri sekvenovaní a následnom zostavení sekvencií.

RAW dáta

Sekvenátory veľmi často produkujú veľké objemy obrazových dát vo formáte TIFF alebo podobnom. Príkladom ďalších RAW dát produkovaných niektorými sekvenátormi sú dáta, ktoré sa dajú reprezentovať ako signál – napr. koncentrácia špecifických iónov v roztoku.

Z RAW dát sú po sekvenovaní určené bázy a ich kvality pomocou algoritmov pre určovanie báz (base calling). Výsledkom sú prečítané segmenty s kvalitami báz. Uchovávanie RAW dát väčšinou nie je po tomto kroku potrebné, a preto môžu byť zmazané.

Prečítané segmenty

Segmenty prečítané sekvenátorom sa bežne ukladajú aj so svojimi kvalitami. Najbežnejším formátom používaným pre ukladanie tohto typu dát je formát FASTQ, ktorý podrobnejšie popíšeme v sekcii 2.3.2.

Tieto dáta predstavujú vstup pre algoritmy zarovnávania a zostavovania sekvencií. Pomerne často je však potrebné ich uchovávanie aj po zostavení. Dôvodom je, že sa môžu objaviť lepšie algoritmy pre zostavenie a zarovnávanie sekvencií, prípadne nastane potreba zarovnať prečítané segmenty k inej referenčnej sekvencii.

Zarovnané sekvencie

Pre uloženie zarovnaných sekvencií sú v dnešnej dobe najčastejšie používané formáty SAM a BAM, ktorých kompresiou sa zaoberáme v tejto práci. Zbežne ich popíšeme v sekcii 2.3.3. V kapitole 5 potom uvedieme podrobnejší popis týchto formátov.

Tieto dáta sú obyčajne používané pri analýzach – napr. na určovanie genetických variácií. Je z nich taktiež možné vytvoriť zostavenie vo FASTA formáte.

Zostavené sekvencie

Obsahujú najpravdepodobnejšie bázy, ktoré sa vyskytujú na danej pozícii v sekvencii. Obyčajne sú ukladané bez dát o kvalite. Najbežnejším formátom pre ich uloženie je formát FASTA, ktorý popíšeme v sekcii 2.3.1. V kapitole 4 potom popíšeme existujúce kompresné metódy pre tieto dáta.

2.3 Bežné formáty používané v bioinformatike

V tejto sekcii popíšeme najbežnejšie formáty, ktoré sa používajú pre ukladanie dát získaných pri sekvenovaní. Najskôr v časti 2.3.1 popíšeme formát FASTA. Ďalej v časti 2.3.2 popíšeme formát FASTQ. Nakoniec v časti 2.3.3 uvedieme základné informácie o formátoch SAM a BAM.

2.3.1 FASTA

Formát FASTA je najpoužívanejším formátom pre ukladanie DNA sekvencií. Jedná sa o jednoduchý textový formát. Jeden FASTA súbor môže obsahovať viacero sekvencií.

Každá sekvencia začína riadkom začínajúcim znakom '`>`', ktorý obsahuje hlavičku sekvencie. Hlavička sekvencie nemá pevný formát. Zvyčajne obsahuje polia znakov oddelené znakom '`|`'.

Na ďalších riadkoch nasleduje samotná sekvencia. Sekvencia môže obsahovať symboly uvedené v tabuľke 2.2. Zápis 'X/Y' v druhom stĺpci tejto tabuľky znamená, že na danom mieste sa môžu vyskytovať bázy X alebo Y. V praxi sa používajú takmer výhradne symboly `A`, `C`, `G`, `T` a `N`.

Príklad 2.3. Príklad FASTA súboru. (Zdroj: NCBI)

```
>gi|6092754|emb|A79849.1| Sequence 6 from Patent WO9637624
TAAAAAAACCTATAAATAATGCCCTTGTACAAATTGTTAAACGTTTTGTGGTTGGTCGCCGT
```

```
TTCTAACGCGATTCCCGGGGACTTTCCAGGAAATGACAACAGCACAGCAACGCTGTGCCTGG
GACATCATGCAGTGCCAAACGGAACGCTAGTGAAAACAATCACGAATGATCAAATTGAAGTG
ACTAATGCTACTGAGCTGGTTCAGAGTTCCTCAACAGGTAGAATATGCGACAGTCCTCACCG
AATCCTTGATGGAAAAAACTGCACACTGATAGATGCTCTATTGGGAGACCCTCATTGTGATG
```

Symbol	Význam	Symbol	Význam
A	adenín	M	A/C
C	cytozín	W	A/T
T	tymín	R	G/A
G	guanín	B	G/T/C
N	ľubovoľná báza	D	G/A/T
K	G/T	H	A/C/T
S	G/C	V	G/C/A
Y	T/C	-	medzera

Tab. 2.2: Zoznam symbolov, ktoré sa môžu nachádzať v DNA sekvencii uloženej vo FASTA súbore

2.3.2 FASTQ

FASTQ [16] je formát, ktorý vznikol rozšírením formátu FASTA o kvality báz. Kvality sú kódované tak, aby ich bolo možné reprezentovať tlačiteľnými ASCII znakmi. Táto reprezentácia nie je štandardizovaná. V súčasnosti sa najčastejšie používa reprezentácia, v ktorej kódy použitých ASCII znakov predstavujú hodnotu kvality PHRED (viď rovnica 2.1), ku ktorej bolo pripočítané číslo 33.

Súbor vo formáte FASTQ môže obsahovať ľubovoľné množstvo sekvencií. FASTQ súbory sú bežne používané pre ukladanie segmentov prečítaných sekvenátorom pred ich zostavením. Obyčajne preto jeden FASTQ súbor obsahuje veľké množstvo sekvencií (každá predstavuje jeden segment). Hlavička každej sekvencie začína znakom '`@`', za ktorým nasleduje identifikátor sekvencie. Na ďalších riadkoch sú uvedené sekvencie báz a kvalít, ktoré sú oddelené znakom '`+`' nachádzajúcom sa na samostatnom riadku.

Príklad 2.4. Príklad jedného prečítaného segmentu z FASTQ súboru. (Zdroj: „1000 genomes project“)

```
@ERR013133.55 IL20_4460:7:1:1160:15961/1
AATCCATGTTTTACATTCAGTGGGTTGTTTTCAACNNNTATNGGTTATGNGTGTNNNNNNNNN
+
.4;26<41<;7D7=;<;@>7FEB8=;3>8:0)5(5!!!424!5685+44!4=44!!!!!!!!
```

Na druhom riadku sa nachádzajú prečítané bázy. Na poslednom riadku zase hodnoty kvalít zakódované ako znaky ASCII. Kódy znakov týchto kvalít sú nasledovné:

```
46 52 59 50 54 60 52 49 60 59 55 68 55 61 59 60 59 64 62 55 70 69
66 56 61 59 51 62 56 58 48 41 53 4 0 53 33 33 33 52 50 52 33 53 54
56 53 43 52 52 33 52 61 52 52 33 33 33 33 33 33 33 33
```

To zodpovedá výsledným hodnotám kvalít:

```
13 19 26 17 21 27 19 16 27 26 22 35 22 28 26 27 26 31 29 22 37 36
33 23 28 26 18 29 23 25 15 8 20 7 20 0 0 0 19 17 19 0 20 21 23 20
10 19 19 0 19 28 19 19 0 0 0 0 0 0 0 0
```

2.3.3 SAM/BAM

Sekvenovacie technológie novej generácie často využívajú spôsob zostavovania sekvencií ich zarovnaním k referenčnej sekvencii. Avšak výstup rôznych sekvenátorov a nástrojov pre zarovnávanie bol v minulosti v rôznych formátoch.

Autori formátu SAM [33], ktorý bol zverejnený v roku 2009, sa snažili o vytvorenie univerzálneho formátu pre ukladanie takto zostavených sekvencií. Formát SAM sa o chvíľu na to stal štandardom pre ukladanie prečítaných segmentov po ich zarovnaní k referenčnej sekvencii. SAM súbor obsahuje okrem informácií o zarovnaniach segmentov aj ďalšie metainformácie, napr. kvality báz alebo informácie o párových čítaniach. Autori sa snažili formát navrhnúť tak, aby bol čo najviac nezávislý na použitej sekvenovacej a zostavovacej technológii. Umožňuje reprezentovať viaceré sekvencie a rozdelovať dáta do logických celkov (napr. sekvencia alebo jedinec). Formát BAM navyše podporuje indexáciu pre rýchly náhodný prístup.

Po technickej stránke je SAM súbor textovým súborom so stĺpcami oddelenými tabulátormi. Súbor BAM je zase jeho binárnou podobou doplnenou o blokovú gzip kompresiu. Súbor BAM dosahuje približne 25% veľkosť odpovedajúceho SAM súboru. Podrobnejšie informácie a popis formátov SAM a BAM sú uvedené v kapitole 5.

2.4 Vybrané projekty sekvenovania DNA

V tejto sekcii popíšeme niektoré z najdôležitejších projektov, ktoré sa zaoberajú alebo zaoberali sekvenovaním kompletných sekvencií DNA. Väčšina uvedených projektov má verejne dostupné dáta.

2.4.1 Human genome project

Projekt „Human genome project“ [4] bol dokončený v roku 2003 po trinástich rokoch práce. Jeho cieľom bolo vytvoriť dostatočne kvalitnú sekvenciu kompletného ľudského genómu[9]. Pri sekvenovaní bolo používané Sangerovo sekvenovanie. Rozpočet projektu bol 3 miliardy dolárov.

[9] V skutočnosti neobsahovala asi 8% ľudského genómu

Paralelne s týmto projektom prebiehal projekt sekvenovania ľudského genómu súkromnej spoločnosti Celera Corporation [2]. Oba projekty boli publikované v približne rovnakom čase.

Výsledkom týchto projektov a ich nasledovníkov bolo vytvorenie dostatočne kvalitnej referenčnej sekvencie ľudského genómu a vytvorenie právneho systému, ktorý znemožňuje diskrimináciu ľudí na základe ich genetickej informácie [3] (prijaté napríklad v USA).

2.4.2 1000 genomes project

Projekt „1000 genomes project“ [11] má za cieľ vytvoriť katalóg genetických variácií ľudského genómu. Toto umožní lepšie pochopenie závislosti fenotypu (vonkajší prejav génov) od genotypu (genetická informácia) u ľudí.

K sekvenovaniu sú používané platformy novej generácie. Plánom projektu je osekvenovať približne 2500 rôznych ľudí z rozdielnych populácií (geografických regiónov) so štvornásobným pokrytím. Autori udávajú, že toto pokrytie umožní nájsť 95% genetických variant, ktoré sa v populácií nachádzajú s frekvenciou vyššou ako 1%.

Viac informácií ako aj verejne prístupné dáta je možné získať na stránkach projektu: `<http://www.1000genomes.org/>`.

2.4.3 Mouse genomes project

Dokončený projekt „Mouse genomes project“ [28] mal za cieľ vytvoriť sekvencie kompletných genómov 17 najdôležitejších kmeňov laboratórnych myší. Myš je jeden z najčastejšie používaných modelových organizmov v biologickom výskume[10]. Tieto dáta sú preto veľmi dôležité pri výskume ľudských chorôb alebo vývoji nových liekov. Dáta projektu sú dostupné na stránkach projektu: `<http://www.sanger.ac.uk/resources/mouse/genomes/>`.

2.4.4 Personal genome project

Dlhodobým cieľom projektu „Personal genome Project“ [6] je vytvorenie kompletného genómu 100 000 ľudí. Získané dáta majú byť verejne dostupné aj so zdravotnými záznamami zúčastnených dobrovoľníkov. Projekt má umožniť výskum v oblasti personalizovanej medicíny, ktorá by umožnila zdravotnú starostlivosť šitú na mieru konkrétneho pacienta. Dôležitou súčasťou projektu bude skúmanie sociálnych rizík (napr. diskriminácia), ktoré hrozia účastníkom projektu.

Viaceré spoločnosti[11] v súčasnosti ponúkajú sekvenovanie kompletného genómu ako pomerne lacnú službu (tisícky dolárov).

[10] Myš a človek majú spoločných asi 80% génov

[11] Napr. Illumina (`<http://www.illumina.com>`) alebo Complete Genomics (`<http://www.completegenomics.com/>`)

2.4.5 UK10K

Cieľom projektu „UK10K“ [9] je výskum pomerne vzácnych genetických variácií v ľudskom genóme a ich dopadu na vznik niektorých chorôb – napr. autizmus, schizofrénia, obezita. Projekt sa má taktiež zaoberať výskumom detí, ktorých zdravie a vývoj boli dlhodobo skúmané od ich narodenia v rokoch 1991 a 1992 (Avon Longitudinal Study of Parents and Children). Tento cieľ chcú dosiahnuť na vzorke 10 000 kompletných ľudských genómov.

Vyprodukované dáta majú obsahovať informácie aj o genotype aj o fenotype skúmaných osôb. Mali by preto poskytnúť lepšiu sadu dát ako „1000 genomes project“.

2.4.6 Genome 10K

Cieľom projektu „Genome 10K“ [23] je vytvorenie 10 000 kompletných sekvencií genómov rôznych stavovcov. Tieto dáta budú vhodné napr. ako podklady pre evolučné štúdie. Dáta sú dostupné na stránkach projektu: `<http://www.genome10k.org/>`.

2.4.7 1001 Genomes Project

Cieľom projektu „1001 Genomes Project“ [7] je vytvorenie kompletných genómov modelovej rastliny Arabidopsis thaliana[12]. Tento projekt je zaujímavý tým, že osekvenované rastliny vyrástli zo semien, ktoré sú bežne dostupné. Tieto dáta môžu byť preto použité pre podrobné štúdie závislosti fenotypu a genotypu.

2.4.8 Výskum nádorových ochorení

Inštitúcia „International Cancer Genome Consortium“ [8] zastrešuje množstvo projektov venujúcich sa výskumu nádorových ochorení. V čase písania tejto práce uvádzali, že ich dátový portál obsahuje 3561 genómov rakoviny.

V súčasnosti prebiehajú dva veľké projekty výskumu nádorových ochorení – „Cancer Genome Project“[13] v Európe (Veľká Británia) a „The Cancer Genome Atlas“[14] vo Spojených Štátoch.

Tieto projekty sú pomerne špecifické. Vzorky s ktorými pracujú sú vzácne a vytvorené dáta sú veľmi citlivé. Ostatné sekvenovacie projekty, ktoré sme uviedli, sú anonymné alebo sa ich zúčastňujú dobrovoľníci. V prípade výskumu rakoviny však veľmi často platí, že si darcovia vzoriek alebo ich pozostalí neprajú zverejnenie týchto veľmi osobných informácií. Preto zvyčajne dáta z týchto projektov nie sú dostupné pre širokú verejnosť. Ďalšou zvláštnosťou je, že aby bolo možné

[12] Po slovensky Arábkovka Thalova. Zaujímavosťou je, že to bola prvá rastlina s kompletne osekvenovaným genómom.

[13] `<http://www.sanger.ac.uk/genetics/CGP/>`

[14] `<http://cancergenome.nih.gov/>`

skúmať mechanizmus a dôvody výskytu danej formy rakoviny, pri sekvenovaní jedného pacienta sa musia vytvoriť dve kompletné sekvencie genómu – jedna pre zdravú bunku a ďalšia pre bunky nádoru.

2.4.9 Sequence read archive

„Sequence read archive“ (SRA) [30] síce nie je sekvenovacím projektom, ale predstavuje dôležitý zdroj dát vytvorených platformami novej generácie. Projekty financované z verejných zdrojov alebo prispievajúce do verejných časopisov sú povinné uverejniť dáta v tomto archíve. Veľká časť dát v SRA je verejne dostupná.

SRA vzniklo kolaboráciou troch veľkých inštitúcií, ktoré sa stretli v projekte „International Nucleotide Sequence Database Collaboration“ (INSDC):

- European Bioinformatics Institute (EBI) vo Veľkej Británii. SRA dostupné z adresy `<http://www.ebi.ac.uk/ena/>`
- National Center for Biotechnology Information (NCBI) v Spojených Štátoch. SRA dostupné z adresy `<http://www.ncbi.nlm.nih.gov/sra>`
- DNA Data Bank of Japan (DDBJ) v Japonsku. SRA dostupné z adresy `<http://trace.ddbj.nig.ac.jp/dra>`

Obsah SRA nemusí byť vo všetkých jeho inštanciách rovnaký. Zapojené inštitúcie si však pravidelne vymieňajú nové príspevky. Dáta v SRA sú poskytované v rôznych formátoch, ktoré závisia od inštitútu, ktorý užívateľ využije pre prístup k SRA. U EBI a DDBJ sa dáta nachádzajú v pomerne širokom množstve formátov. Väčšina dát, ktoré obsahujú prečítané segmenty je však poskytovaných aj v podobe FASTQ súborov. Odporúčanými formátmi určenými pre prispievanie do archívu sú formáty BAM a FASTQ.

NCBI ponúka dáta vo svojom internom formáte SRA. Práve tento formát je taktiež používaný pre výmenu dát medzi jednotlivými inštitúciami. Viac informácií o formáte SRA uvedieme v sekcii 6.3. Odporúčané formáty určené na prispievanie do archívu sú taktiež BAM a FASTQ.

Autori v práci [30] udávajú, že v roku 2011 obsahovalo SRA nad 100 Tbp voľne dostupných dát. Platforma Illumina tvorila 84%, SOLiD 12% a Roche/454 2% uložených dát. Najväčším projektom, ktorý SRA v tom čase obsahovala bol „1000 genomes project“, ktorý tvoril skoro tretinu celého archívu. Z osekvenovaných organizmov mal najväčšie zastúpenie človek, ktorého genómy tvorili až 61% dát.

3. Kompresia dát

Hlavnou témou tejto práce je kompresia biologických dát. Preto v tejto kapitole uvedieme základy teórie kompresie dát a podrobnejšie popíšeme kompresné metódy, ktoré budeme využívať v ďalších častiach práce. Najskôr v sekcii 3.1 zadefinujeme používané pojmy. V sekcii 3.2 uvedieme niektoré z jednoduchších kompresných algoritmov. V sekcii 3.3 uvedieme metódy používané pre kódovanie prirodzených čísel. V sekcii 3.4 uvedieme metódy pre kódovanie prirodzených čísel, ktoré využívajú štatistické rozdelenie výskytu symbolov na vstupe. V sekcii 3.5 popíšeme slovníkové kompresné metódy. Nakoniec v sekcii 3.6 popíšeme metódu BZip2.

3.1 Základné pojmy

V tejto sekcii uvádzame základné pojmy používané v ďalšom texte. Ďalšie z používaných termínov, ktoré tu neuvádzame, je možné nájsť napr. v [43] alebo [38]. V úvode popíšeme, čo to je kompresia dát. Ďalej v časti 3.1.1 zadefinujeme pojmy týkajúce sa reťazcov. Nakoniec v časti 3.1.2 ponúkneme úvod do teórie informácie a kódovania.

Definícia 3.1. *Kompresia dát* je spôsob kódovania dát, ktorého cieľom je zmenšenie ich veľkosti. Aby bola prakticky využiteľná, je nutné, aby bolo možné pôvodné dáta spätne zrekonštruovať. Metódu používanú pre kompresiu dát budeme nazývať *kompresná metóda*.

Definícia 3.2. *Vstupno/výstupné prúdy* ponúkajú abstrakciu vstupu a výstupu. Príkladom môže byť napr. súbor na úložnom zariadení, lokalita v sieti alebo oblasť pamäte.

Definícia 3.3. *Kompresia* je proces transformácie nespracovaných dát zo vstupu na komprimované dáta. Algoritmus popisujúci tento proces nazývame *kompresný algoritmus* alebo *enkodér*.

Definícia 3.4. *Dekompresia* je inverzný proces ku kompresii. Algoritmus popisujúci tento proces nazývame *dekompresný algoritmus* alebo *dekodér*.

Definícia 3.5. *Kodek* je označenie pre pár kompresného a dekompresného algoritmu.

Účinnosť kompresnej metódy budeme vyjadrovať kompresným pomerom.

Definícia 3.6. *Kompresný pomer* je číslo

$$K_p = \frac{\text{veľkosť komprimovaných dát}}{\text{veľkosť originálnych dát}}. \tag{3.1}$$

Kompresný pomer zvyčajne udávame v percentách.

Základné delenie kompresných metód je na *bezstratové* a *stratové*. Bezstratové kompresné metódy sú schopné po dekompresii kompletne zrekonštruovať pôvodné dáta. Stratové metódy toho nie sú schopné, pretože s cieľom zvýšenia účinnosti kompresie vynechávajú menej potrebné informácie. V práci sa zaoberáme bezstratovými kompresnými metódami. Pri výbere kompresných metód však budeme krátko diskutovať aj nad možnosťou využitia stratovej kompresie.

Ďalším typickým delením kompresných metód je delenie na *blokové* a *prúdové* metódy. Prúdové kompresné metódy spracovávajú vstup po jednotlivých bitoch/bajtoch. Na druhej strane blokové kompresné metódy spracovávajú vstup po väčších blokoch.

Poznámka 3.1. Pri biologických sekvenciách býva často udávaná úroveň kompresie ako počet bitov potrebných pre uloženie jednej bázy. Pri súboroch, ktoré obsahujú aj iné dáta ako samostatné sekvencie DNA (napr. SAM a BAM) je možné vypočítať počet bitov potrebných pre uloženie jednej bázy ako

$$\text{bitov/báza} = \frac{\text{veľkosť súboru}}{\text{počet uložených báz}}. \tag{3.2}$$

3.1.1 Reťazce

Definícia 3.7. *Symbolom* rozumieme najmenšiu jednotku prenosu informácie. Môže sa jednať o bity, znaky, čísla, ale aj o slová prirodzeného jazyka.

Definícia 3.8. *Abeceda* Σ je konečná množina symbolov, na ktorej je definované usporiadanie. Jej veľkosť označujeme ako $|\Sigma|$. *Vstupná abeceda* je množina všetkých symbolov vyskytujúcich sa na vstupe. *Kódovacia abeceda* je množina všetkých symbolov používaných pre reprezentáciu komprimovaných dát.

Definícia 3.9. *Reťazec* (slovo, správa) T nad abecedou Σ je konečná postupnosť symbolov z abecedy Σ. Prázdny reťazec označujeme symbolom ε. Dĺžku reťazca značíme ako $|T|$. Množinu všetkých reťazcov nad abecedou Σ označujeme ako Σ^*. Množinu všetkých neprázdnych reťazcov nad abecedou Σ označujeme ako Σ^+. $T[i]$ značí i-ty symbol reťazca T.

Definícia 3.10. *Zreťazenie* reťazcov $T_1 = T_1[1] \dots T_1[n_1]$ a $T_2 = T_2[1] \dots T_2[n_2]$ definujeme ako

$$T_1T_2 = T_1[1] \dots T_1[n_1]T_2[1] \dots T_2[n_2]. \tag{3.3}$$

Definícia 3.11. Majme reťazec T a pozície i a j také, že $i < j$. Potom *podreťazec* reťazca T začínajúci na pozícii i a končiaci na pozícii j definujeme ako $T[i, j] = T[i]T[i + 1] \dots T[j]$.

Definícia 3.12. *Prefix* reťazca $T = T[1] \dots T[n]$ je každý jeho podreťazec $T[1, k]$ pre $1 \leq k \leq n$.

Definícia 3.13. *Suffix* reťazca $T = T[1] \dots T[n]$ je každý jeho podreťazec $T[k, n]$ pre $1 \leq k \leq n$.

3.1.2 Teória informácie a kódovania

V tejto sekcii uvedieme základy teórie informácie a kódovania. Viac informácií o tejto problematike je možné nájsť v [26].

Definícia 3.14. ([26]) Nech je daná vstupná abeceda Σ a kódovacia abeceda Σ_C. *Kódovanie* je funkcia $\Phi : \Sigma^* \to \Sigma_C^*$. Kódovanie je *jednoznačne dekódovateľné* práve vtedy, keď

$$\forall u, v \in \Sigma^*, \Phi(u) = \Phi(v) \Rightarrow u = v. \tag{3.4}$$

V ďalšom texte budeme používať binárnu kódovaciu abecedu, tj. $\Sigma_C = \{0, 1\}$.

Jednoducho vidíme, že aby bolo kódovanie prakticky použiteľné musí byť jednoznačne dekódovateľné.

Pri kompresii sa typicky postupuje nasledovne:

1. vstup v rozdelíme na slová $v = w_1 w_2 \dots w_n, w_i \in \Sigma^*$,
2. určíme $\Phi(w_1), \Phi(w_2) \dots, \Phi(w_n)$,
3. položíme $\Phi(v) = \Phi(w_1)\Phi(w_2) \dots \Phi(w_n)$.

Definícia 3.15. *Kód* je funkcia $\phi : \Sigma \to \Sigma_C^*$. Každý kód *generuje kódovanie* $\Phi : \Sigma^* \to \Sigma_C^*$ definované ako

$$\Phi(s_1 s_2 \dots s_n) = \phi(s_1)\phi(s_2) \dots \phi(s_n), \text{ kde } s_1, s_2 \dots s_n \in \Sigma \tag{3.5}$$

Kód je jednoznačne dekódovateľný, ak je jednoznačne dekódovateľné ním generované kódovanie.

Definícia 3.16. Prvky oboru hodnôt kódovania sa nazývajú *kódové slová*. Ak je dĺžka kódových slov konštantná pre všetky kódové slová, nazývame kód definovaný týmto kódovaním *kód s pevnou dĺžkou kódového slova*. Ak toto neplatí, nazývame takýto kód *kód s premenlivou dĺžkou kódového slova*.

Pre kódovanie symbolov, ktorých výskyt sa neriadi uniformným rozdelením, je vhodné používať kód s premenlivou dĺžkou kódového slova. Pri dekódovaní musí byť dekodér schopný identifikovať, kde končia jednotlivé kódové slová v komprimovanom prúde. V praxi je taktiež výhodné mať možnosť dekódovania jediným lineárnym priechodom komprimovaným prúdom. Túto požiadavku splňujú prefixové kódy.

Definícia 3.17. Kód nazveme *prefixovým*, ak žiadne jeho kodové slovo nie je predponou iného kódového slova.

Každý prefixový kód je možné reprezentovať stromom, ktorý nazývame *prefixový strom*[1]. Preto sa jedná o jednoznačne dekódovateľný kód. Prefixový strom je využívaný napríklad v Huffmanovom kódovaní (viď sekcia 3.4.1).

[1]Špeciálne pre binárnu abecedu sa jedná o binárny strom

Definícia 3.18. Je daný kód ϕ so vstupnou abecedou $\Sigma = \{x_1, x_2, \ldots x_n\}$. *Priemernú dĺžku kódového slova* $\bar{l}$ kódu ϕ pre vstupnú abecedu Σ definujeme ako

$$\bar{l} = \sum_{x \in \Sigma} P(x)|\phi(x)|, \tag{3.6}$$

kde $P(x)$ je pravdepodobnosť výskytu symbolu x vo vstupe.

Definícia 3.19. Je daná vstupná abeceda $\Sigma = \{x_1, x_2, \ldots x_n\}$. Kódovanie $\Phi_o : \Sigma^* \to \Sigma_C^*$ nazveme *optimálnym*, ak pre každé ďalšie kódovanie Φ platí,

$$\forall s \in \Sigma^*: |\Phi(s)| \geq |\Phi_o(s)|. \tag{3.7}$$

Definícia 3.20. *Entrópia* $H(X)$ diskrétnej náhodnej veličiny X s oborom hodnôt $\mathcal{H}$ a s rozdelením pravdepodobnosti $P(X = x) = p(x), x \in \mathcal{H}$, je definovaná vzťahom

$$H(X) = -\sum_{x \in \mathcal{H}} p(x) \log_2 p(x). \tag{3.8}$$

Nahradením teoretickej pravdepodobnosti výskytu symbolov ich relatívnou početnosťou získame hodnotu empirickej entrópie. Vzťah entrópie ku kompresii vyjadruje hodnota $nH(P)$, ktorá udáva minimálny počet bitov potrebných na kódovanie vstupu o dĺžke n so vstupnou abecedou s pravdepodobnostným rozdelením výskytu symbolov P. Teda pre zakódovanie takéhoto vstupu treba priemerne $H(P)$ bitov na symbol. Výpočet empirickej entrópie vstupu nám umožňuje získať predstavu o jeho zložitosti.

3.2 Jednoduché metódy

V tejto sekcii popíšeme niektoré veľmi jednoduché metódy kompresie dát. Tieto metódy sú štandardnou súčasťou viacerých kompresných metód.

3.2.1 Kódovanie behov

Definícia 3.21. *Behom* nazveme postupnosť po sebe idúcich zhodných symbolov v reťazci.

Kódovanie behov (Run-Length Encoding – RLE) je kompresná metóda založená na kódovaní behov vo vstupných dátach. V priebehu kompresie nahradí RLE enkodér všetky behy vo vstupe dvojicou `(symbol, dĺžka behu)`. Behové kódovanie je vhodné pre kódovanie vstupu obsahujúceho veľa dlhých behov.

V práci používame dve implementácie tejto metódy:

- `RLEU` vytvára na výstupe 2 prúdy. Prvý prúd je používaný pre ukladanie hodnôt znakov behu a druhý pre dĺžky behov. Hodnoty v týchto prúdoch je možné zakódovať vhodným kódovaním čísel.

- `RLE` pracuje priamo nad výstupným prúdom (tj. nepoužíva kódovanie čísel). Jeho abeceda je obmedzená na hodnoty $(0, 255)$. Ak narazí na beh tak zapíše na výstup tri bajty: symbol `0xff` (255), dĺžku behu a symbol behu. Aby komprimované dáta neboli väčšie ako originálne, tak kóduje len behy dlhšie ako 3 symboly – dĺžka týchto behov je kódovaná ako dĺžka behu mínus 4. Behy dlhšie ako 258 znakov[2] sú rozdelené na úseky po maximálne 258 znakoch. Symbol `0xff` (255) je kódovaný špeciálne a to postupnosťou dvoch symbolov `0xff`.

Príklad 3.1. Na vstupe máme postupnosť symbolov `accccabaac`. Výsledkom algoritmu `RLEU` bude postupnosť dvojíc (symbol, dĺžka behu): `(a,1), (c,4), (a,1), (b,1), (a,2), (c,1)`.

Výsledkom algoritmu `RLE` bude postupnosť: `a@0cabaac`. Kde `@` zastupuje znak `0xff`.

3.2.2 Delta kódovanie

Delta kódovanie je kompresná metóda, ktorá kóduje rozdiely medzi hodnotami susedných symbolov. Je založená na predpoklade, že je jednoduchšie kódovať menšie hodnoty rozdielov než pôvodné dáta. Táto metóda je vhodná pre kódovanie vstupu obsahujúceho dáta, v ktorých majú po sebe idúce symboly veľmi blízke hodnoty.

Príklad 3.2. Na vstupe máme postupnosť symbolov `50, 53, 54, 54, 51, 55`. Delta kódovanie zakóduje tieto symboly ako postupnosť `50, 3, 1, 0, -3, 4`.

Okrem tradičného delta kódovania používame v práci ďalšie dve na ňom založené metódy. Jediným rozdielom je, že v nich nekódujeme rozdiely dvoch po sebe idúcich hodnôt ale rozdiely voči aritmetickému priemeru a k mediánu hodnôt na vstupe.

3.2.3 Kódovanie s presunom na začiatok

Základnou myšlienkou metódy kódovania presunom na začiatok (Move To Front – MTF) je kódovanie naposledy použitých symbolov malými číslami. Kvôli tomu si metóda udržiava zoznam symbolov abecedy v poradí, v akom boli naposledy použité. Konkrétny symbol je potom zakódovaný ako počet symbolov, ktoré ho predchádzajú v tomto zozname. Po zakódovaní je tento symbol presunutý na začiatok zoznamu.

Metóda je vhodná na kódovanie vstupov pri ktorých môžeme predpokladať, že rovnaké symboly sa nachádzajú blízko pri sebe.

Príklad 3.3. Na vstupe máme postupnosť symbolov `abcdccabc`. Výsledkom move to front kódovania bude postupnosť hodnôt `0,1,2,3,1,0,3,3,2`.

[2]Ich dĺžka je kódovaná ako znak `0xfe` (254)

3.3 Kódovanie prirodzených čísel

Pri konštrukcii kompresných metód je často nutné využiť vhodné kódovanie prirodzených čísel. Pri voľbe vhodného spôsobu kódovania čísel by sme mali prihliadať k pravdepodobnostnému rozdeleniu jednotlivých hodnôt na vstupe. Napríklad hodnoty na výstupe delta kódovania popísaného v sekcii 3.2.2 je vhodné zakódovať metódou, ktorá menším číslam prideľuje kratšie kódy. Výber vhodnej metódy teda závisí od rozsahu hodnôt a ich očakávaného pravdepodobnostného rozdelenia.

V tejto sekcii popíšeme metódy, ktoré nevyužívajú charakter vstupných dát pre zlepšenie kompresie.

Definícia 3.22. Je daná vstupná abeceda $\Sigma = \{x_1, x_2, \dots x_n\}$, kódovacia abeceda $\Sigma_C = \{c_1, c_2, \dots c_m\}$ a kód $\phi : \Sigma \to \Sigma_C^*$ s kódovými slovami o dĺžkach $l_1 \leq l_2 \dots l_n$. Nech p_i je pravdepodobnosť výskytu symbolu x_i v ľubovoľnom vstupnom slove. Kód ϕ nazveme *univerzálny*, ak pre každé rozdelenie pravdepodobnosti vstupnej abecedy $P = (p_1, \dots p_n)$ také, že $p_1 \geq p_2 \geq \dots p_n$ platí:

$$\frac{\sum_{i=1}^{n} p_i l_i}{max(1, H(P))} \leq K, \tag{3.9}$$

kde $H(P)$ je entrópia náhodnej veličiny s rozdelením P a K je konštanta.

Inými slovami pre kódovanie ľubovoľnej vstupnej abecedy univerzálnym kódom platí, že priemerná dĺžka kódového slova tohto kódu je až na konštantu optimálna pre rozdelenie pravdepodobnosti danej vstupnej abecedy.

3.3.1 Unárny kód

Unárny kód (α) reprezentuje kladné číslo $i > 0$ ako $0^{i-1}1$. Je zrejmé, že jeho pamäťová zložitosť je i bitov. Unárne kódovanie tvorí základ viacerých ďalších kódovaní.

Unárne kódovanie je optimálne pre pravdepodobnostné rozdelenie

$$p(i) = \frac{1}{2^i}, \forall i \in \mathbb{N}. \tag{3.10}$$

3.3.2 Binárny kód

Binárny kód (β) kóduje číslo v binárnej sústave. Pre zakódovanie čísla i potrebuje $\lfloor \log_2 i + 1 \rfloor$ bitov. Pretože nie je binárny kód prefixový, je jeho použitie obmedzené len na kódy s pevnou dĺžkou kódového slova.

Binárny kód s pevnou dĺžkou

Aby bolo možné binárny kód dekomprimovať, je potrebné používať pevnú dĺžku kódového slova. Takýto kód sa nazýva binárny kód s pevnou dĺžkou. Pre zakódovanie čísla i je potrebné použiť binárny kód s dĺžkou aspoň $\lceil \log_2 i \rceil$ bitov. Z popisu

metódy je vidieť, že nie je vhodná pre kódovanie číselných množín, ktoré nie sú obmedzené.

Binárny kód s pevnou dĺžkou je optimálne kódovanie pre pravdepodobnostné rozdelenie

$$p(i) = \frac{1}{|\Sigma|}, \forall i \in \Sigma. \tag{3.11}$$

Minimálny (redukovaný) binárny kód

Ak rozsah vstupnej abecedy nie je mocninou dvojky, tak je vhodné namiesto binárneho kódu s pevnou dĺžkou použiť minimálny binárny kód.

Nech $|\Sigma| = n$ a $k = \lceil \log_2 i \rceil$ potom číslo i zakódujeme minimálnym binárnym kódom (β_n) nasledovne:

- ak $i < 2^k - n$ použijeme $(k-1)$ bitový binárny kód pre i,
- inak použijeme k bitový binárny kód pre $i + 2^k - n$.

3.3.3 Eliasove kódy

Eliasove kódy [17] sú skupinou univerzálnych kódov. Ich konštrukcia je iteratívna. Číslo i kódujú ako $X(|\beta(i)|)\beta'(i)$, kde X je eliasov kód z predchádzajúcej iterácie (v prvej iterácii unárny kód), a $\beta'(i)$ je binárny kód čísla i bez prvého bitu.

Gama kód

Gama kód (γ) kóduje číslo i ako

$$\gamma(i) = \alpha(|\beta(i)|)\beta'(i), \tag{3.12}$$

má pamäťovú zložitosť

$$|\gamma(i)| = 2\lfloor \log_2 i \rfloor + 1. \tag{3.13}$$

Delta kód

Delta kód (δ) kóduje číslo i ako

$$\delta(i) = \gamma(|\beta(i)|)\beta'(i), \tag{3.14}$$

má pamäťovú zložitosť

$$|\delta(i)| = 1 + \lfloor \log_2 i \rfloor + 2\lfloor \log_2 (1 + \lfloor \log_2 i \rfloor) \rfloor. \tag{3.15}$$

Delta kód je lepší pre vstup, ktorý obsahuje veľké hodnoty. Gama kód je lepší, ak vstup obsahuje veľa malých čísel.

3.3.4 Fibonacciho kód

Fibonacciho kód (Fib) [21] je univerzálny kód. Využíva, že každé kladné prirodzené číslo je možné reprezentovať súčtom rôznych Fibonacciho čísel, pričom žiadne

Číslo	α	β	γ	δ	Fib
1	1	1	1	1	11
2	01	10	010	0100	011
3	001	11	011	0101	0011
4	0001	100	00100	01100	1011
5	00001	101	00101	01101	00011
6	000001	110	00110	01110	10011
7	0000001	111	00111	01111	01011
8	00000001	1000	0001000	00100000	000011
9	...	1001	0001001	00100001	100011
10	...	1010	0001010	00100010	010011
11	...	1011	0001011	00100011	001011
12	...	1100	0001100	00100100	101011

Tab. 3.1: Príklady unárneho kódu, binárneho kódu, Eliasovho gama kódu, Eliasovho delta kódu a Fibonacciho kódu

2 po sebe idúce Fibonacciho čísla nie sú použité v tomto súčte. Tento súčet je možné reprezentovať kódovým slovom ako v ľubovoľnej inej pozičnej sústave (1 je na mieste Fibonacciho čísla použitého v súčte a 0 na ostatných miestach). Pridaním bitu 1 na koniec takéhoto kódového slova vytvoríme kód, ktorý má tú zaujímavú vlastnosť, že je schopný sa zotaviť z chyby v komprimovanom prúde. Dôvodom je práve spomínaná vlastnosť Fibonacciho reprezentácie čísel a to, že každé kódové slovo tohto kódu je ukončené dvomi po sebe idúcimi jednotkami.

Vo Fibonacciho kóde je každé číslo i kódované kódovým slovom $C = c_0 c_1 \dots c_k$, pre ktoré platí:

$$i = \sum_{j=0}^{k-1} c_j F(j+2) \text{ a } c_k = c_{k-1} = 1, \tag{3.16}$$

kde $F(j)$ je j-té Fibonacciho číslo.

Pre dĺžku kódového slova Fibonacciho kódu platí:

$$|Fib(i)| = 1 + \lfloor \log_\Phi \sqrt{5} i \rfloor, \text{ kde } \Phi = \frac{1+\sqrt{5}}{2}. \tag{3.17}$$

Na obrázku 3.1 uvádzame porovnanie dĺžok kódových slov univerzálnych kódov s binárnym kódom v závislosti na hodnote kódovaného čísla. V tabuľke 3.1 potom uvádzame príklady týchto kódov.

3.3.5 Golombov kód

Golombov kód (G_m) s parametrom $m \in \mathbb{N}^+$ kóduje číslo i ako:

$$G_m(i) = \alpha \left(\left\lfloor \frac{i-1}{m} \right\rfloor + 1 \right) \beta_m[(i-1) \bmod m], \tag{3.18}$$

Číslo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
m=2	100	101	110	111	0100	0101	0110	0111
m=5	100	101	110	1110	1111	0100	0101	0110

Tab. 3.2: Príklad Golombovho kódu

Číslo	**Binárny kód**	**VByte kód**
34	100010	**1**0100010
62	111110	**1**0111110
150	10010110	**0**0010110 **1**0000001
1250	10011100010	**0**1100010 **1**0001001

Tab. 3.3: Príklad VByte kódu

kde β_m je minimálny binárny kód.

Golombov kód nie je univerzálny. Je optimálny [43] pre geometrické rozdelenie

$$p_i = (1-p)^{i-1}p, \tag{3.19}$$

s hodnotou parametra

$$m = \left\lceil -\frac{\log_2(1+p)}{\log_2 p} \right\rceil. \tag{3.20}$$

Špeciálnym prípadom Golombovho kódu je unárny kód pre $m = 1$, a Riceov kód pre m, ktoré je mocninou dvojky.

V tabuľke 3.2 uvádzame príklad Golombovho kódu.

3.3.6 VByte kód

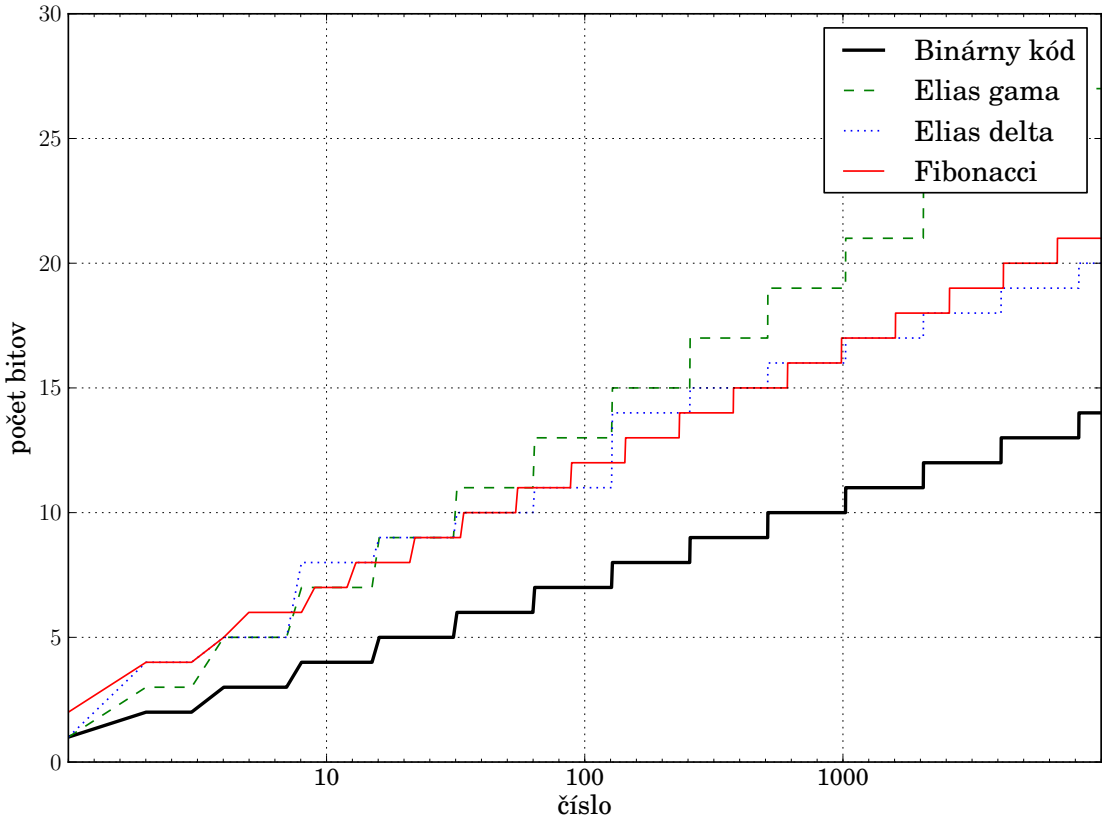
VByte kód [51] využíva variabilný počet celých bajtov pre uloženie prirodzených čísel v ich binárnej podobe. Najvyšší bit každého bajtu je využitý ako indikátor konca kódového slova.

V tabuľke 3.3 uvádzame príklady VByte kódu. Tučným písmom je znázornený bit, ktorý indikuje koniec kódového slova.

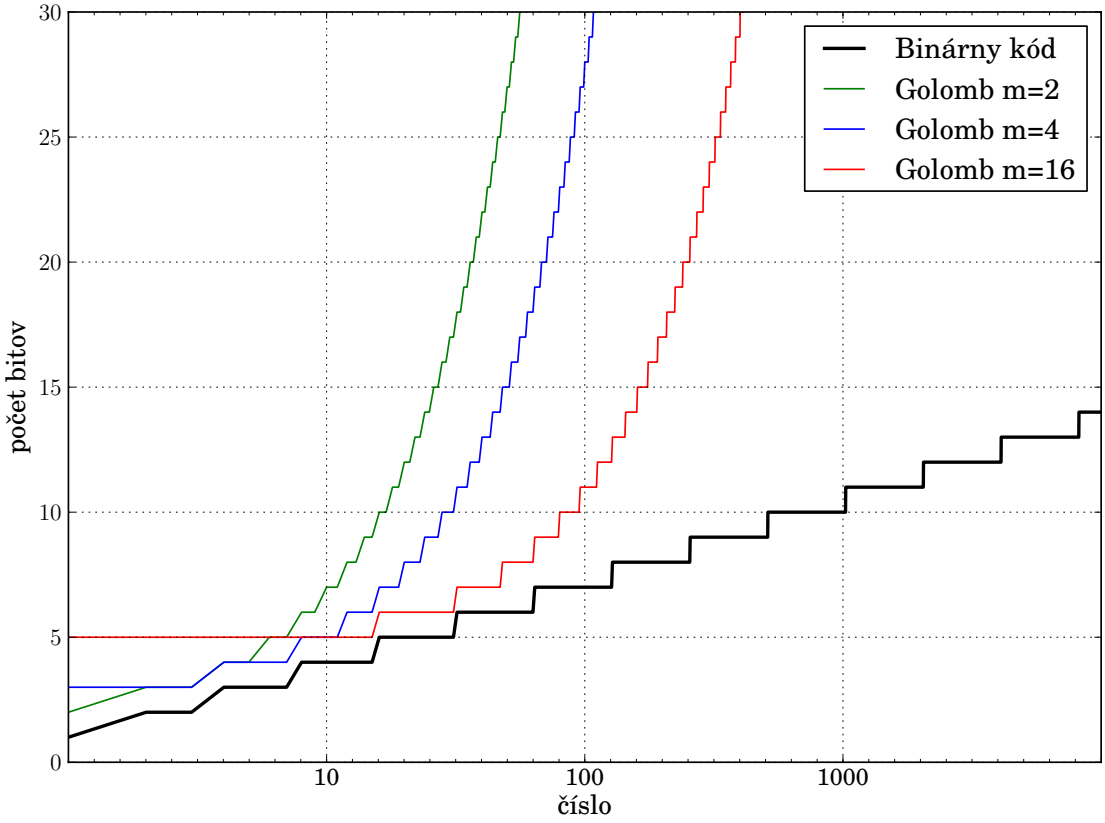
Výstup VByte kódovania tvoria celé bajty, je preto veľmi vhodný na následnú kompresiu napríklad slovníkovými kompresnými metódami.

3.3.7 Záporné čísla a nula

Popisované kódy umožňujú kódovať len kladné čísla. Ak je potrebné kódovať aj záporné čísla alebo nulu, je nutné zobraziť tieto hodnoty na prirodzené čísla (prostým zobrazením).



Obr. 3.1: Porovnanie veľkosti binárneho, Eliasovho gama, Eliasovho delta a Fibonacciho kódu



Obr. 3.2: Porovnanie veľkosti binárneho kódu a Golombovho kódu pre rôzne hodnoty parametra m

Príkladom môže byť alterujúce zobrazenie, kedy je postupnosť čísel $0, 1, -1, 2, -2, 3, -3 \ldots$ kódovaná ako čísla $1, 2, 3, 4, 5, 6, 7 \ldots$.

V tejto práci používame nasledujúcu metódu pre rozšírenie kódovaní prirodzených čísel na celé čísla:

- Od kódovaného čísla odčítame jednotku, aby bolo možné kódovať nulu a
- ku každému kódovému slovu pridáme bit, ktorý indikuje znamienko kódovaného čísla.

3.4 Štatistické metódy kódovania čísel

Kódovania čísel, ktorými sme sa zaoberali v sekcii 3.3 sú optimálne pre konkrétne rozdelenia výskytov symbolov na vstupe. V prípade neznámeho vstupu alebo vstupu, v ktorom nie je možné pravdepodobnosť výskytu symbolov na vstupe charakterizovať žiadnym jednoduchým rozdelením, je vhodnejšie využiť štatistické metódy kódovania čísel.

Štatistické metódy kódovania čísel využívajú, že niektoré hodnoty sa na vstupe vyskytujú častejšie než iné. Najpoužívanejšou metódou v tejto oblasti je Huffmanovo kódovanie, ktoré popíšeme v časti 3.4.1. V časti 3.4.2 potom popíšeme Aritmetické kódovanie.

3.4.1 Huffmanov kódovanie

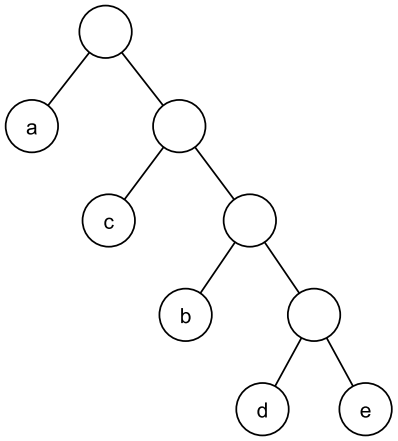
Huffmanovo kódovanie umožňuje vytvoriť optimálny prefixový kód pre vstupné dáta. Algoritmus konštrukcie Huffmanovho stromu potrebuje poznať relatívne početnosti jednotlivých symbolov na vstupe. Jeho výstupom je prefixový[3] strom, ktorý je používaný ku kompresii a dekompresii. Listy tohto stromu predstavujú jednotlivé znaky vstupnej abecedy. Kód konkrétneho znaku je jednoznačne určený cestou od vrchola k listu s týmto znakom.

Príklad 3.4 (Konštrukcia Huffmanovho stromu)**.** Máme nasledujúcu tabuľku početností symbolov na vstupe:

znak	a	b	c	d	e
početnosť	30	16	25	5	10

Tejto tabuľke odpovedá Huffmanov strom:

[3]Binárny pre binárnu kódovaciu abecedu



Výsledný kód vyzerá nasledovne:

znak	a	b	c	d	e
kód	0	110	10	1110	1111

Nevýhodou Huffmanovho kódovania je, že pre konštrukciu Huffmanovho stromu je potrebná znalosť relatívnych početností symbolov na vstupe. Kedže je tento strom potrebné konštruovať aj pri dekompresii, početnosti symbolov musia byť uložené spolu s komprimovaným prúdom, čo oproti klasickým kódovaniam čísel môže v niektorých prípadoch znamenať nárast veľkosti komprimovaných dát. Tento problém je najviac pálčivý v prípade veľkej vstupnej abecedy.

Problém veľkej vstupnej abecedy rieši Kanonický Huffmanov kód [37]. Táto metóda namiesto početností znakov ukladá do komprimovaného prúdu početnosti dĺžok kódových slov Huffmanovho kódu. Výhodou je úspora miesta pri veľkých abecedách. Pri kompresii/dekompresii si navyše nepotrebuje udržovať Huffmanov strom, ale iba relatívne jednoduchú tabuľku, čo prináša pozitívny dopad na rýchlosť tejto metódy.

Existuje pomerne veľké množstvo ďalších variant Huffmanovho kódovania. Príkladom je adaptívne kódovanie, ktoré funguje aj na vstupoch, ktoré nie sú celé známe pri kompresii. Adaptívne Huffmanovo kódovanie vytvára Huffmanov strom až počas samotnej kompresie/dekompresie z dovtedy známeho vstupu.

3.4.2 Aritmetické kódovanie

Aritmetické kódovanie [49] je možné považovať za zovšeobecnenie Huffmanovho kódovania. Namiesto kódovania každého symbolu však celej správe na vstupe priraďuje reálne číslo z intervalu $[0, 1]$.

Podobne ako Huffmanovo kódovanie potrebuje pre svoju činnosť poznať relatívne početnosti symbolov na vstupe, prípadne použiť adaptívnu variantu, ktorá túto početnosť odhadne. Kompresia zjednodušene funguje tak, že je interval $[0, 1]$ rozdelený na podintervaly, z ktorých každý zodpovedá jednému symbolu zo vstupnej abecedy. Dĺžky týchto intervalov sú v rovnakom pomere v akom sú početnosti

symbolov na vstupe. Algoritmus najskôr zvolí interval, ktorý zodpovedá prvému znaku správy. Tento interval je následne rozdelený na podintervaly, podobne ako interval $[0, 1]$ na začiatku algoritmu. Proces výberu intervalu sa potom opakuje pre ďalšie symboly na vstupe. Z výsledného intervalu je zvolené číslo, ktoré je najlepšie kódovateľné. Toto číslo sa po prípadnom zaokrúhlení uloží.

Aritmetické kódovanie môže dosahovať lepšie výsledky ako Huffmanovo kódovanie. Dôvodom je, že Huffmanovo kódovanie používa len celočíselné dĺžky kódových slov. Huffmanovo kódovanie je však v praxi častejšie používané. Dôvodom je jeho jednoduchosť a rýchlosť. Navyše časti algoritmu aritmetického kodovania boli istý čas viazané patentmi, čo odradilo viacerých záujemcov o jeho využívanie.

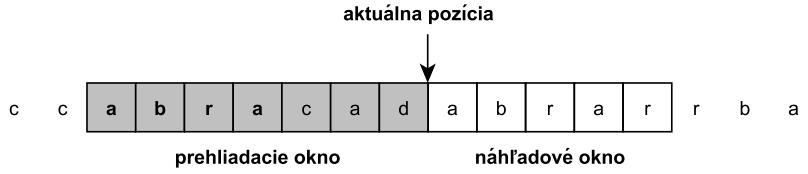
3.5 Slovníkové metódy

Základnou myšlienkou slovníkových metód je nahradenie časti textu odkazom na časť už spracovaného textu. Algoritmus si pre tento účel udržuje slovník.

V súčastnosti najpoužívanejšou slovníkovou metódou je LZ77 (Lempel-Ziv [52]). Metóda používa posuvné okno pevnej veľkosti, ktoré je rozdelené na prehliadacie a náhľadové okno. Každý krok metódy LZ77 vyhľadá čo najdlhšie slovo S, ktoré začína v prehliadacom okne a je zhodné s nejakou predponou slova v náhľadovej časti. Toto slovo je potom zakódované trojicou `(offset, dĺžka, znak)`, kde `offset` je vzdialenosť začiatku slova S od začiatku náhľadového okna, `dĺžka` je veľkosť slova S, a `znak` je znak, ktorý nasleduje za slovom S. Posuvné okno je potom posunuté o `dĺžka`+1 znakov do prava.

Hľadanie vhodných fráz v prehliadacom okne je časovo náročné, preto je kompresia metódou LZ77 pomalá. Na druhej strane je však dekompresia veľmi rýchla.

Príklad 3.5. Majme nasledujúcu situáciu:



Metóda LZ77 zakóduje slovo začínajúce na aktuálnej pozícii ako (7, 4, r) a posunie celé okno o 5 znakov doprava.

3.5.1 GZip

Metóda gzip je založená na variante metódy LZ77 (konkrétne DEFLATE), ktorá je doplnená o Huffmanovo kódovanie. Predstavuje štandardnú metódu používanú pre kompresiu všetkých typov dát. Používa sa napríklad v kompresnom formáte zip, pri kompresii v HTTP protokole a v grafickom formáte PNG.

Používa prehliadacie okno s veľkosťou 2^{15} B a náhľadové okno s veľkosťou 256 B.

3.5.2 LZMA

LZMA [40] (Lempel-Ziv Markov chain algorithm) je sofistikovaná implementácia metódy LZ77. Je jednou z najúčinnejších univerzálnych metód dneška, ktoré majú praktickú časovú náročnosť. Je používaná v kompresnom formáte 7zip.

Môže používať okno s veľkosťou až 1 GB. Pre kódovanie čísel využíva variantu Aritmetického kódovania.

3.6 BZip2

Metóda bzip2 [46] je bloková metóda – vstup rozdeľuje na bloky s veľkosťami 100 až 900 kB. Základom metódy je Burrows-Wheelerova transformácia [12], ktorá má tú vlastnosť, že vytvorí zo vstupu sekvenciu, ktorá obsahuje rovnaké znaky blízko pri sebe. Táto sekvencia je následne spracovaná kódovaním s presunom na začiatok a Huffmanovým kódovaním.

Metóda BZip2 dosahuje lepšie kompresné pomery ako metóda GZip. Zároveň je však pomalšia. Metóda LZMA je pomalšia pri kompresii, ale dosahuje lepší kompresný pomer a rýchlejšu dekompresiu ako BZip2.

4. Kompresia sekvencií DNA

V tejto kapitole popíšeme existujúce prístupy ku kompresii DNA sekvencií. Tejto oblasti sa venuje veľa úsilia, preto sa obmedzíme len na niektoré vybrané prístupy. Všetky popisované metódy sa zaoberajú iba kompresiou DNA sekvencií bez ďalších doplňujúcich informácií (napr. kvality). Pre našu prácu sú však dôležité myšlienky, ktoré tieto metódy priniesli. Kapitola je rozdelená na 2 časti. V časti 4.1 popíšeme štandardné metódy, ktoré pri kompresii využívajú len informácie z práve komprimovanej sekvencie. V časti 4.2 popíšeme metódy, ktoré ukladajú len rozdiely oproti vhodnej referenčnej sekvencii.

4.1 Tradičné metódy

Pretože DNA sekvencie obsahujú len štyri symboly, je možné využiť kódovanie, ktoré využíva pre kódovanie každej bázy 2 bity. Týmto spôsobom je možné dosiahnuť kompresný pomer 25% vzhľadom k sekvencii vo FASTA formáte. V tabuľke 2.2 sme však uviedli, že sekvencia vo FASTA formáte môže obsahovať aj ďalšie symboly (až 16 z ktorých sa v praxi používa 5). Preto nie je možné tento prístup aplikovať na väčšinu reálnych dát.

Vieme, že DNA sekvencie obsahujú veľa opakujúcich sa častí. Preto je vhodné pre ich kompresiu využiť slovníkové metódy. Najjednoduchšie je použiť univerzálnu metódu, z ktorých sú vhodné napr. gzip, zip alebo 7zip. Nevýhodou týchto metód je, že nevyužívajú znalosť charakteru vstupných dát a komprimujú DNA sekvencie rovnako ako ľubovoľný iný text. Preto bolo špeciálne pre tento účel vytvorené pomerne veľké množstvo iných metód.

Najjednoduchšie z týchto metód sú metódy, ktoré sú založené na klasických slovníkových metódach. V sekvenciách však hľadajú nielen opakujúce sa podreťazce, ale napríklad aj zrkadlovo otočené časti sekvencií alebo komplementárne sekvencie. Príkladom takejto metódy je BioCompress [25]. Tento prístup je možné rozšíriť o používanie nepresných zhôd, čo využíva napríklad GenCompress [14]. Metóda ukladá nepresné zhody ako postupnosť operácií, ktoré sú potrebné pre vytvorenie požadovaného reťazca z frázy v slovníku. Používa nasledujúce operácie:

- *bodová mutácia* (zámena bázy na jednej pozícii),
- *vloženie* časti sekvencie a
- *odstránenie* časti sekvencie.

Modernejším prístupom je metóda XM [13], ktorá hľadá opakovania v sekvenciách a zároveň využíva štatistický model pre odhad pravdepodobnosti ďalšieho symbolu. Táto pravdepodobnosť je využitá pri kódovaní tohto symbolu aritmetickým kódovaním. Autori udávajú priemerný kompresný pomer tejto metódy ako 1.6940 bitov/báza (ekvivalent kompresného pomeru vzhľadom k formátu FASTA je približne 21%). Štúdia [24] udáva, že v čase jej vydania (2009) bola táto metóda najúspešnejšou metódou v oblasti kompresie DNA sekvencií.

Veľa z tradičných metód pre kompresiu biologických sekvencií nie je veľmi rozšírených. Hlavný dôvod ich neúspechu vidíme v tom, že ich implementácie nie sú pripravené pre produkčné nasadenie. Autori týchto metód sa väčšinou zameriavajú na dosiahnutie čo najlepšieho kompresného pomeru. To však často spôsobuje nízku rýchlosť a vysokú spotrebu pamäte. Navyše popisované metódy neumožňujú náhodný prístup ku komprimovaným dátam, čo je pri dlhých sekvenciách (stovky milióny báz) veľmi žiadúce.

4.2 Využitie referenčnej sekvencie

Sekvencie rovnakých alebo podobných organizmov sú si veľmi podobné. Napríklad genómy dvoch rôznych ľudí sú si podobné na 99.9%. Využitím týchto podobnosti je možné dosiahnuť lepšiu úroveň kompresie ako tradičnými metódami.

Najčastejšie je využívaný prístup, kedy je jedna sekvencia vyhlásená za referenčnú a u ostatných sekvencií sa ukladajú len rozdiely oproti tejto referenčnej sekvencii. Táto myšlienka sa objavila napríklad v práci [15]. Autori dosiahli kompresiu genómu Jamesa Watsona z približne 3GB na úroveň 4MB. Využívali k tomu referenčnú sekvenciu a tzv. SNP mapu, ktorá obsahuje zoznam známych mutácií. Obe tieto štruktúry dosahovali veľkosti 4.2GB. Referenčnú sekvenciu využíva aj slovníková metóda RLZ [31], ktorá umožňuje náhodný prístup ku komprimovaným dátam. Slovník tejto metódy tvorí práve referenčná sekvencia.

Najväčšou nevýhodou využitia referenčnej sekvencie je, že pri dekompresii musíme mať k dispozícii vhodnú referenčnú sekvenciu. Ak však ukladáme veľké množstvo sekvencií rovnakého druhu, výhody použitia referenčnej sekvencie jednoznačne prevýšia jej nevýhody.

Pre dosiahnutie čo najlepšej kompresie je taktiež dôležitá správna voľba referenčnej sekvencie. Súbory SAM/BAM obsahujú segmenty, ktoré sú zarovnané ku konkrétnej referenčnej sekvencii. Teoreticky je síce možné, že by použitie inej referenčnej sekvencie zlepšilo kompresiu. Týmto smerom sa však uberať nebudeme, pretože to je náročný problém a predpokladáme, že jeho vyriešenie neprinesie zásadné zlepšenie účinnosti kompresie.

5. Formát súborov SAM/BAM

V tejto kapitole popíšeme štruktúru formátov SAM a BAM [33]. Kompletnú špecifikáciu je možné nájsť v [10]. Najskôr v sekcii 5.1 uvedieme celkový popis formátu SAM súboru a uvedieme príklad takéhoto súboru. Ďalej v sekcii 5.2 popíšeme formát hlavičky SAM súboru. V sekcii 5.3 uvedieme popis povinných a v sekcii 5.4 nepovinných polí SAM súboru. V sekcii 5.5 uvedieme popis formátu BAM.

5.1 Úvod

SAM súbor je textový súbor so stĺpcami oddelenými tabulátormi. Pozostáva z nepovinnej hlavičky a hlavnej sekcie, ktorá obsahuje záznamy (SAM záznamy) s informáciami o prečítaných segmentoch. Každý riadok hlavnej sekcie reprezentuje jeden záznam, ktorý obsahuje informácie o jednom segmente. Každý záznam obsahuje 11 povinných polí a môže obsahovať ľubovoľné množstvo ďalších nepovinných polí.

Príklad 5.1. Predpokladajme, že máme nasledujúcu referenčnú sekvenciu a segmenty r001/1, r001/2, r002, r003 a r004. Segmenty r001/1 a r001/1 sú výsledkom párového čítania a r004 obsahuje preskočenú časť sekvencie[1]. Znaky +/− určujú, že segment je orientovaný ako 5' → 3' resp. 3' → 5'.

```
pozícia:    12345678901234  5678901234567890123456789012345
ref:        AGCATGTTAGATAA**GATAGCTGTGCTAGTAGGCAGTCAGCGCCAT

+r001/1           TTAGATAAAGGATA*CTG
+r002            aaaAGATAA*GGATA
+r004                       ATAGCT..............TCAGC
-r003                              ttagctTAGGC
-r001/2                                          CAGCGCCAT
```

Tejto situácii odpovedá nasledujúci SAM súbor:

```
@HD VN:1.3 SO:coordinate
@SQ SN:ref LN:45
r001 163 ref  7 30 8M2I4M1D3M = 37 39 TTAGATAAAGGATACTG *
r002   0 ref  9 30 3S6M1P1I4M * 0 0 AAAAGATAAGGATA      *
r004   0 ref 16 30    6M14N5M * 0 0 ATAGCTTCAGC         *
r003  16 ref 29 30       6H5M * 0 0 TAGGC               * NM:i:0
r001  83 ref 37 30         9M = 7 -39 CAGCGCCAT         *
```

Príklad je upravenou verziou príkladu zo špecifikácie formátu SAM [10].

[1] Označené bodkami. Takýto segment môže vzniknúť napr. pri chybe pri čítaní sekvencie.

5.2 Hlavička

Na začiatku SAM súboru sa môže nachádzať nepovinná hlavička. Riadky hlavičky začínajú znakom `@`, ktorý je nasledovaný dvojznakovým kódom typu záznamu aktuálneho riadku. Každý riadok potom môže obsahovať položky tvaru `TAG:HODNOTA`, kde `TAG` je dvojznakový kód typu položky a `HODNOTA` je hodnota položky. Jednotlivé položky na riadku sú oddelené tabulátormi. Najdôležitejšie položky hlavičky sú:

- `@HD` Prvý riadok hlavičky. Obsahuje identifikátor verzie formátu a spôsob usporiadania záznamov v súbore. Pre nás bude dôležité usporiadanie `coordinate`, ktoré hovorí, že súbor je usporiadaný primárne podľa referenčnej sekvencie a sekundárne podľa pozície v nej.
- `@SQ` Zoznam referenčných sekvencií. Obsahuje popis referenčnej sekvencie a jej identifikátor. Tieto identifikátory sú používané v povinných poliach `RNAME` a `RNEXT`. Poradie `@SQ` záznamov definuje poradie referenčných sekvencií pri usporiadaní `coordinate`. Referenčné sekvencie typicky predstavujú chromozómy.

Ďalej môže hlavička obsahovať napríklad záznamy o sekvenovacích knižniciach alebo záznamy o programoch, ktoré menili obsah súboru.

5.3 Povinné polia

Každý SAM záznam obsahuje 11 povinných polí – ich hodnota musí byť uvedená. Môžu však nadobúdať hodnotu '`0`' (pri číselných hodnotách) alebo '`*`' (pri reťazcoch), ak ich hodnota nie je známa. V tabuľke 5.1 uvádzame prehľad povinných polí.

`QNAME`

Pole `QNAME` je identifikátorom šablóny z ktorej pochádza segment. Obsahuje reťazec s dĺžkou maximálne 255 znakov. Pole `QNAME` SAM súborov z rôznych zdrojov môže obsahovať dáta v rôznych formátoch.

`FLAG`

Pole `FLAG` obsahuje pole bitových príznakov so šírkou 16 bitov. Význam jednotlivých bitov je uvedený v tabuľke 5.2.

Pole	Typ	Popis
QNAME	reťazec	identifikátor šablóny
FLAG	číslo	pole príznakov
RNAME	reťazec	identifikátor referenčnej sekvencie
POS	číslo	pozícia zarovnania
MAPQ	číslo	kvalita zarovnania
CIGAR	reťazec	CIGAR reťazec
RNEXT	reťazec	identifikátor referenčnej sekvencie ďalšieho segmentu v šablóne
PNEXT	číslo	pozícia ďalšieho segmentu v šablóne
TLEN	číslo	dĺžka šablóny, ktorá bola pozorovaná v dátach
SEQ	reťazec	postupnosť báz segmentu
QUAL	reťazec	postupnosť kvalít báz segmentu

Tab. 5.1: Prehľad povinných polí SAM súboru

RNAME

Pole RNAME obsahuje identifikátor referenčnej sekvencie ku ktorej je zarovnaný segment. Tento identifikátor sa musí nachádzať v zozname referenčných sekvencií v hlavičke SAM súboru.

POS

Pole POS obsahuje pozíciu zarovnania k referenčnej sekvencii s identifikátorom RNAME. Udáva pozíciu prvej zhodnej bázy segmentu s referenčnou sekvenciou v zarovnaní. Prvá báza referenčnej sekvencie má offset 1. Položka POS môže nadobúdať hodnoty z intervalu $[0, 2^{29} - 1]$.

MAPQ

Kvalita zarovnania segmentu k referenčnej sekvencii. Obsahuje pravdepodobnosť, že pozícia zarovnania je chybná. Hodnota je uvedená v PHRED skórovaní (viď sekcia 2.2.5). Môže nadobúdať hodnoty z intervalu $[0, 2^8 - 1]$.

CIGAR

Obsahuje CIGAR reťazec. CIGAR reťazec popisuje zarovnanie segmentu k referenčnej sekvencii. Obsahuje postupnosť dvojíc (dĺžka operácie, operácia). Povolené operácie sú uvedené v tabuľke 5.3.

Súčet dĺžok operácií 'M/I/S/=/X' sa musí zhodovať s dĺžkou sekvencie v poli SEQ.

Poznámka 5.1. Operácia 'M' je definovaná trochu nešťastne. Názov „zhoda“ (match) evokuje zhodu bázy v zarovnaní s referenčnou sekvenciou. Táto operácia

Bit	Popis
0x1	segment je súčasťou šablóny, ktorá obsahuje viac segmentov
0x2	každý segment šablóny je zarovnaný k referenčnej sekvencii
0x4	segment nie je zarovnaný k referenčnej sekvencii
0x8	ďalší segment v šablóne nie je zarovnaný k referenčnej sekvencii
0x10	`SEQ` je uvedená v komplementárnom tvare
0x20	`SEQ` ďalšieho segmentu v šablóne je uvedená v komplementárnom tvare
0x40	segment je prvým segmentov v šablóne
0x80	segment je posledným segmentom v šablóne
0x100	záznam je sekundárnym zarovnaním daného segmentu
0x200	segment neprešiel kontrolou kvality
0x400	segment je duplikátom

Tab. 5.2: Popis bitových príznakov používaných v poli `FLAG`

Operácia	Popis
M (match)	v zarovnaní nie je medzera
I (insert)	vloženie do referenčnej sekvencie
D (delete)	vymazanie z referenčnej sekvencie
N (noncoding)	preskočenie časti referenčnej sekvencie
S (soft clip)	časť sekvencie je v `SEQ` ale nie v zarovnaní
H (hard clip)	časť sekvencie nie je ani v `SEQ` ani v zarovnaní
P (padding)	výplň – používané vo viacnásobnom zarovnaní sekvencií
=	zhoda sekvencie s referenčnou sekvenciou
X	nezhoda sekvencie s referenčnou sekvenciou

Tab. 5.3: Popis operácií, ktoré sa môžu vyskytovať v CIGAR reťazci

však v skutočnosti znamená, že zarovnanie neobsahuje medzeru, tj. na danom mieste môže nastať aj zhoda aj nezhoda s referenčnou sekvenciou.

Operácia '`P`' je používaná vo viacnásobných zarovnaniach sekvencií a pre naše účely ju môžeme ignorovať.

Príklad 5.2. Máme nasledujúce zarovnanie:

```
referenčná sekvencia:  AGGAACTG*CCAGGTGACACACTCCCACCATGGACCTC
zarovnaný segment:         ACTAACCAGGTG**ACA
```

Pole `POS` bude obsahovať hodnotu 5 a CIGAR reťazec bude `4M1I7M2D3M`.

RNEXT

Polia RNEXT, PNEXT a TLEN slúžia pre uloženie informácií o ďalšom segmente v šablóne, pričom posledný segment obsahuje informácie o prvom segmente. Používajú sa napríklad pre uloženie informácií o párových čítaniach.

Pole RNEXT obsahuje identifikátor referenčnej sekvencie ku ktorej je zarovnaný ďalší segment zo šablóny z ktorej pochádza segment. Ak obsahuje znak '=' tak je referenčná sekvencia zhodná s RNAME. Inak pre túto položku platia rovnaké podmienky ako pre položku RNAME.

PNEXT

Pozícia zarovnania ďalšieho segmentu k referenčnej sekvencii RNEXT. Platí pre ňu to isté ako pre položku POS.

TLEN

Pozorovaná dĺžka šablóny. Udáva počet báz od najľavejšej zarovnanej bázy k najpravejšej zarovnanej báze zo všetkých segmentov šablóny. Segment, ktorý sa nachádza najviac vľavo v šablóne má zápornú hodnotu, ten najviac vpravo má kladnú hodnotu. Znamienko ostatných segmentov šablóny nie je definované. Pole môže nadobúdať hodnoty z intervalu $[-2^{29}+1, 2^{29}-1]$.

SEQ

Obsahuje reťazec báz segmentu.

QUAL

Obsahuje reťazec s PHRED kvalitami (viď sekcia 2.2.5) báz v SEQ. Kvality sú kódované rovnako ako sme to popisovali pri formáte FASTQ v sekcii 2.3.2. Ak je táto položka uvedená, musí mať rovnakú dĺžku ako SEQ.

5.4 Nepovinné polia

Po povinných poliach môže SAM záznam obsahovať ľubovoľný počet nepovinných polí. Tieto slúžia napríklad pre uloženie údajov špecifických pre použitú sekvenovaciu platformu.

Nepovinné polia sú vo formáte TAG:TYP:HODNOTA, kde TAG je reťazec dĺžky 2, ktorý udáva druh nepovinného poľa. Každý TAG sa môže v zázname vyskytovať len jeden krát. TYP definuje dátový typ poľa – znak (A), celé číslo (i), reťazec (Z) alebo pole hodnôt (B). Špecifikácia formátu uvádza viacero štandardných

nepovinných polí. Používatelia SAM súborov si navyše môžu definovať vlastné nepovinné polia.

V tabuľke 5.4 sú uvedené niektoré dôležité nepovinné polia.

TAG	Typ	Popis
BQ	Z	reťazec ofsetov kvalít k BAQ kvalitám (viď sekcia 8.7.2)
CS	Z	sekvencia segmentu v farebnom priestore používanom v sekvenátore SOLiD (viď sekcia 8.7.1)
CQ	Z	reťazec kvalít sekvencie vo farebnom priestore (viď sekcia 8.7.2)
MD	Z	reťazec, ktorý kóduje bázy, ktoré sa nezhodujú s referenčnou sekvenciou
NM	i	editačná vzdialenosť segmentu k referenčnej sekvencii
OQ	Z	reťazec originálnych kvalít báz (pred spracovaním) (viď sekcia 8.7.2)
RG	Z	z akej skupiny segmentov pochádza segment (ich zoznam je uložený v `@RG` tagu hlavičky)

Tab. 5.4: Dôležité nepovinné položky definované v špecifikácií SAM formátu

5.5 Formát BAM

Súbory vo formáte BAM obsahujú rovnaké dáta ako súbor SAM ale v binárnej podobe. Navyše sú ešte skomprimované upraveným programom gzip. Úpravou štandardnej kompresie gzip je rozdelenie súboru na bloky s veľkosťou 64 kB. Súbor BAM dosahuje asi 25% veľkosti odpovedajúceho SAM súboru.

Voliteľne je možné vytvoriť index BAM súboru. Pred indexáciou musí byť BAM súbor utriedený triedením `coordinate`. Tento index spolu s rozdelením súboru na pomerne malé bloky umožňuje rýchly náhodný prístup k dátam.

5.6 Predpoklady pre vstupné súbory

Predpokladáme, že vstupné súbory sú už pred použitím utriedené triedením `coordinate` a v hlavičke obsahujú slovník použitých referenčných sekvencií[2].

Vstupný súbor je možné usporiadať pomocou aplikácie samtools (viď dodatok B.1.2).

[2]Toto je vyžadované aj pri vytváraní BAM súboru

6. Podobné práce

V tejto kapitole popíšeme dostupné implementácie a práce, ktoré sa venujú problematike kompresie súborov vo formáte SAM.

6.1 Improving Transmission Efficiency of Large Sequence Alignment/Map (SAM) Files

Cieľom autorov [42] bolo navrhnutie kompresnej schémy pre kompresiu SAM súborov, ktorá by znížila čas prenosu týchto súborov po sieti – požadovali, čo najmenšiu veľkosť súboru a dostatočnú rýchlosť kompresie/dekompresie.

Jednotlivé SAM polia komprimujú samostatne (každé vhodnou metódou). Výsledné dáta následne komprimujú dostupným kompresným nástrojom (WinRAR). Výsledný komprimovaný súbor neumožňuje náhodný prístup k dátam.

Na testovacích dátach dosahovala táto metóda kompresný pomer 55% – 60% oproti veľkosti súborov vo formáte BAM. Zdroj súborov použitých pri testovaní nie je uvedený. Autori uvádzajú, že pri prenose SAM/BAM súborov po sieti s rýchlosťou 1.5 Mbit/s je možné pri využití tejto metódy usporiť asi 25% času oproti použitiu formátu BAM (čas prenosu zahŕňa čas kompresie, samotný čas prenosu po sieti a čas dekompresie).

6.2 Efficient storage of high throughput DNA sequencing data using reference-based compression

Cieľom autorov [22] bolo navhnúť metódu pre kompresiu biologických dát (konkrétne pre súbory vo formáte BAM a FASTQ) s využitím referenčnej sekvencie – ukladajú len zmeny oproti tejto referenčnej sekvencii.

Kvalitatívne dáta komprimujú stratovo – ukladajú len kvality báz nezhodujúcich sa s referenčnou sekvenciou a voliteľnú časť ostatných báz[1].

Popisovaná metóda umožňuje, pri uložení kvalít pre 2% báz, asi 20-násobné zmenšenie veľkosti vstupných súborov oproti veľkosti týchto súborov vo formáte BAM. Tento výsledok bol dosiahnutý na vstupnom súbore z „1000 genomes project“ a na privátnom BAM súbore.

Na tejto práci je založená implementácia a kompresný formát CRAM[2]. Predprodukčná verzia aplikácie CRAM je dostupná na stránkach projektu. Formát umožňuje náhodný prístup ku komprimovaným dátam.

[1] Výber vhodných kvalít je skôr biologickou než technickou otázkou – napr. môže závisieť od pokrytia alebo zložitosti a dôležitosti daného regiónu DNA

[2] `<http://www.ebi.ac.uk/ena/about/cram_toolkit>`

6.3 SRA

Formát SRA je interným formátom inštitúcie NCBI[3] pre ukladanie dát v SRA archíve (viď sekcia 2.4.9). Formát SRA umožňuje uložiť okrem samotných biologických sekvencií aj veľké množstvo ďalších informácií (napr. výsledky analýz alebo zarovnania).

Dáta sú vo formáte SRA uložené ako objekty podobné tabuľkám v relačných databázach. Jednotlivé typy dát sú komprimované vhodnými metódami. Formát navyše podporuje indexáciu pre umožnenie náhodného prístupu.

Rozšírením formátu SRA je cSRA, v ktorom je pri kompresii využívaná referenčná sekvencia podobne ako v nástroji CRAM. Autori formátu cSRA uvažujú aj nad stratovou kompresiou kvalít.

Na stránkach projektu sú k dispozícii nástroje, ktoré umožňujú konverziu súborov v SRA formáte na ďalšie najbežnejšie používané formáty (FASTQ, BAM). Nedostali sme sa však k nástroju, ktorý by umožňoval vytvorenie SRA alebo cSRA súborov. Všetky informácie sme čerpali z dokumentov dostupných na stránkach projektu `<http://www.ncbi.nlm.nih.gov/Traces/sra/>`.

[3]National Center for Biotechnology Information (NCBI)

7. Implementácia

V tejto kapitole uvedieme technické detaily implementácie aplikácie SamZip a komprimovaného formátu SamZip, ktorý vytvára a číta táto aplikácia. Najskôr v sekcii 7.1 uvedieme základné informácie o aplikácii SamZip. V sekcii 7.2 popíšeme aplikáciu ako celok. V ďalších sekciách sa potom budeme zameriavať na jej jednotlivé súčasti. V sekcii 7.3 uvedieme formát komprimovaných SamZip súborov. V sekcii 7.4 sa budeme zaoberať implementáciou kompresných metód. V sekcii 7.5 uvedieme informácie o konfiguračnom súbore, ktorý riadi kompresiu v aplikácii. V sekcii 7.6 uvedieme informácie o implementácii SAM procesorov. Nakoniec v sekcii 7.7 popíšeme index, ktorý je používaný pre implementáciu náhodného prístupu ku komprimovaným dátam v SamZip súbore.

7.1 Základný popis

Základnou úlohou aplikácie SamZip je kompresia a dekompresia SAM/BAM súborov. Z používateľského hľadiska sa jedná o konzolovú aplikáciu.

Pri návrhu aplikácie sme na ňu kládli nasledujúce požiadavky:

- viacplatformovosť,
- jednoduché testovanie použitých kompresných metód,
- jednoduchá rozšíriteľnosť a
- možnosť náhodného prístupu k dátam.

Dôležite pre nás bolo aj to, aby bola práca s nami vytvorenými komprimovanými súbormi čo najviac užívateľsky prívetivá. Preto sme navrhli jej programové rozhranie (API) a rozhranie príkazovej riadky tak, aby mohla jednoducho nahradiť štandardné nástroje pre čítanie SAM/BAM súborov.

Užívateľskú príručku pre aplikáciu SamZip sme rozdelili do 2 častí. V dodatku B uvedieme dokumentáciu pre nenáročného užívateľa, ktorý sa nechce zaoberať nastavovaním parametrov kompresie. V dodatku C potom popíšeme, ako je možné ovplyvniť priebeh kompresie upravením kompresného profilu. Taktiež v ňom popíšeme kompresné metódy a SAM procesory, ktoré štandardne dodávame s našou implementáciou.

7.1.1 Platforma aplikácie

Aplikácia bola vyvíjaná pre platformu Java SE v jazyku Java.

Platforma Java bola zvolená z dôvodu svojej viacplatformovosti. Jedná sa síce o interpretovaný kód, avšak podľa [5] ponúka v priemere asi len o 30% nižšiu rýchlosť ako v prípade natívneho kódu.

7.1.2 Systémové požiadavky

Pre beh aplikácie je potrebné mať nainštalované behové prostredie Javy (Java Runtime Environment – JRE) minimálne vo verzii 1.7 a minimálne 128 MB pamäte RAM (záleží od veľkosti komprimovaného bloku a počtu vlákien použitých pri kompresii). Doporučujeme však minimálne 1 GB pamäte RAM pre kompresiu s využitím 4 vlákien a 2 GB pri využití 8 vlákien.

Aplikácia bola testovaná s oficiálnou implementáciou behového prostredia Javy od Oracle[1] vo verzii 1.7.0 na operačných systémoch Windows 7 a Ubuntu 11.0, a implementáciou OpenJDK (IcedTea 2.1[2]) na operačnom systéme Gentoo s Linux kernelom vo verzii 3.0.29. Všetky tieto behové prostredia boli 32 bitové.

7.2 Celkový pohľad

Aplikácia SamZip má tri základné funkcie:

- kompresia vstupného SAM alebo BAM súboru,
- dekompresia celého komprimovaného súboru a
- dekompresia časti komprimovaného súboru.

Aby bolo možné zachovať rýchly náhodný prístup k jednotlivým záznamom v komprimovanom súbore, je vstup rozdelený na bloky obsahujúce určitý počet záznamov (napr. 1000 – 5000). Pri náhodnom prístupe potom stačí dekomprimovať len bloky, ktoré obsahujú požadované dáta. Pozíciu týchto blokov v komprimovanom súbore je možné získať z indexu, ktorý je vytváraný pri kompresii.

Priebeh kompresie je riadený užívateľsky nastaviteľným konfiguračným súborom, ktorý budeme v ďalšom texte nazývať *kompresný profil*. Kompresný profil obsahuje definície používaných kompresných metód a definície procesných elementov, ktoré majú na starosti kompresiu a dekompresiu bloku SAM záznamov. Tieto elementy budeme v ďalšom texte nazývať *SAM procesory*. Postupnosť SAM procesorov v poradí, v akom sú uvedené v kompresnom profile tvorí *kompresnú pipeline*. Pri kompresii a dekompresii sú postupne aplikované SAM procesory v tejto pipeline.

Vďaka použitiu kompresného profilu je aplikácia jednoducho konfigurovateľná a jednoducho rozšíriteľná o nové kompresné metódy a SAM procesory.

Pre urýchlenie kompresie môže užívateľ špecifikovať počet vlákien, ktoré budú použité pri kompresii. Implicitne využíva aplikácia toľko vlákien, koľkými procesormi/jadrami disponuje stroj na ktorom beží. V každom vlákne je možné komprimovať jeden blok. Po kompresii určitého počtu blokov sú tieto bloky v správnom poradí zapísané do komprimovaného súboru. Nevýhodou tohto prístupu je, že si v pamäti musíme uchovávať dáta viacerých komprimovaných blokov a viacerých

[1] `<http://www.oracle.com/technetwork/java/javase/downloads/index.html>`
[2] `<http://icedtea.classpath.org/>`

kompresných štruktúr používaných pri kompresii (napr. slovníky slovníkových metód). Pri väčšom počte používaných vlákien sa preto zväčšujú nároky na veľkosť dostupnej pamäte RAM.

Poznámka 7.1. Pojem *pipeline* používame pre postupnosť elementov (v našom prípade SAM procesory), ktoré spracovávajú dáta. Každý z týchto elementov spracuje určitú časť vstupu (napr. kompresia časti SAM súboru alebo ľubovoľný výpočet).

7.2.1 Použité knižnice

Formáty SAM/BAM sú pomerne zložité, preto je pre prácu s nimi vhodné využiť niektorú z dostupných knižníc. Tieto knižnice existujú pre väčšinu hlavných programovacích jazykov – C++, Java, Python, Perl a iné. V aplikácii SamZip využívame pre prácu so SAM/BAM súbormi knižnicu Picard[3]. Táto knižnica poskytuje abstrakciu vstupného SAM/BAM súboru, preto sa nemusíme starať o to, či spracovávame textový SAM alebo binárny BAM súbor. V ďalšom texte budeme preto, ak to nebude vyslovene spomenuté inak, pod názvom SAM formát (súbor) myslieť oba tieto formáty. Knižnica Picard je oficiálna knižnica pre prácu so SAM súbormi pre Javu.

Ďalej využívame knižnicu Fastutil[4] s rýchlejšími implementáciami štandardných kontajnerov (zoznam, asociatívne pole, hašovacie tabuľky ...) a knižnicu JCommander[5] pre spracovanie parametrov z príkazovej riadky.

7.2.2 Organizácia kódu

Prehľad najdôležitejších balíčkov aplikácie SamZip je znázornený na obrázku 7.1. Šipky medzi jednotlivými balíčkami znázorňujú ich vzájomné závislosti.

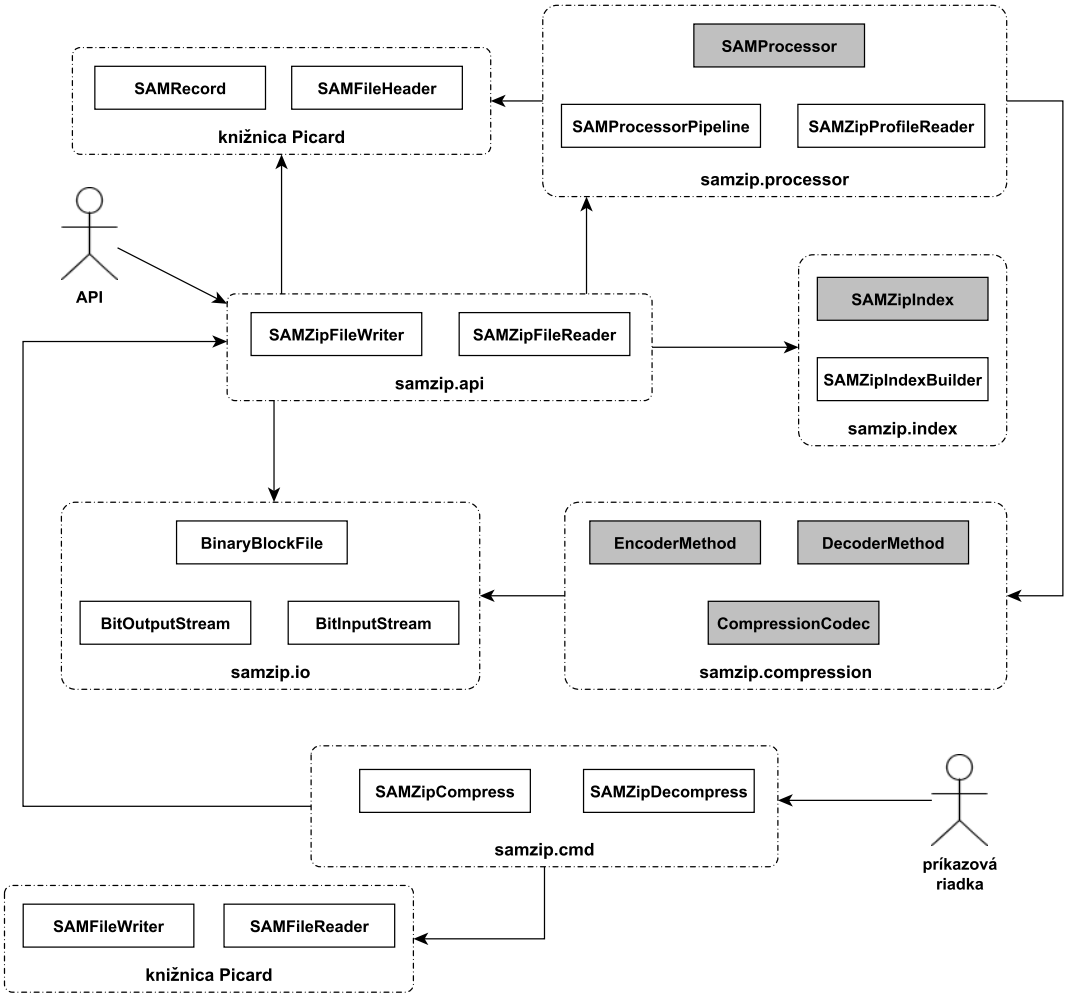
Ako už bolo spomínané, pre prácu so SAM súbormi používame knižnicu Picard. Pri kompresii čítame vstupné SAM súbory pomocou triedy `SAMFileReader`. Pre zápis SAM súborov pri dekompresii potom využívame triedu `SAMFileWriter`. Prístup k údajom v hlavičke súboru umožňuje trieda `SAMFileHeader`. Prístup k údajom v jednotlivých SAM záznamoch zase umožňuje trieda `SAMRecord`. Knižnica Picard ďalej ponúka niekoľko ďalších užitočných pomocných tried (napr. pre spracovanie CIGAR reťazcov), ktoré sme využívali pri implementácii SAM procesorov.

Funkčnosť aplikácie integrujú nasledujúce balíčky:

- `samzip.cmd` Tento balíček obsahuje triedy, ktoré predstavujú implementáciu konzolového rozhrania aplikácie SamZip. Tieto triedy ponúkajú vysokoúrovňové rozhranie pre kompresiu/dekompresiu SAM/BAM súborov.

- `samzip.api` Pre vytvorenie komprimovaného súboru slúži trieda `SAMZipFile`

[3] `<http://picard.sourceforge.net/index.shtml>`
[4] `<http://fastutil.dsi.unimi.it>`
[5] `<http://jcommander.org>`



Obr. 7.1: Celkový pohľad na aplikáciu SamZip. Aktéri (postavičky) označujú očakávané miesta vstupu k rozhraniu aplikácie

`Writer`, ktorá umožňuje zápis SAM záznamov (`SAMRecord`) do komprimovaného SamZip súboru. Pre dekompresiu SamZip súboru slúži trieda `SAMZipFileReader`, ktorá ponúka dekompresiu záznamov – sekvenčne alebo pomocou indexového dotazu. Obe tieto triedy sú kompatibilné s triedami `SAMFileWriter` a `SAMFileReader` z knižnice Picard. Je preto pomerne jednoduché zakomponovať SamZip kompresiu do existujúcich programov, ktoré využívajú knižnicu Picard.

Balíček `samzip.compression` zapúzdruje implementáciu kompresných metód. Potomkovia abstraktných tried `EncoderMethod` a `DecoderMethod` reprezentujú enkodéry a dekodéry. Tieto triedy sú potomkami štandardných prúdov v Jave (`InputStream` a `OutputStream`). Inštancie potomkov triedy `EncoderMethod` môžu pre svoj výstup využívať výstupný prúd alebo ľubovoľné množstvo ďalších inštancií potomkov tejto triedy. U potomkov triedy `DecoderMethod` je situácia analogická. Nastavenie parametrov kompresie a vytvorenie enkodéra a dekodéra majú na starosti potomkovia abstraktnej triedy `CompressionCodec`, ktoré reprezentujú kompresné kodeky.

Balíček `samzip.io` obsahuje triedy pre prácu so vstupom a výstupom. Dôležité pre implementáciu kompresných metód sú bitové prúdy (`BitInputStream` a `BitOutputStream`). Na prácu s binárnym súborom s blokmi, ktorého formát popisujeme v sekcii 7.3 slúži trieda `BinaryBlockFile`.

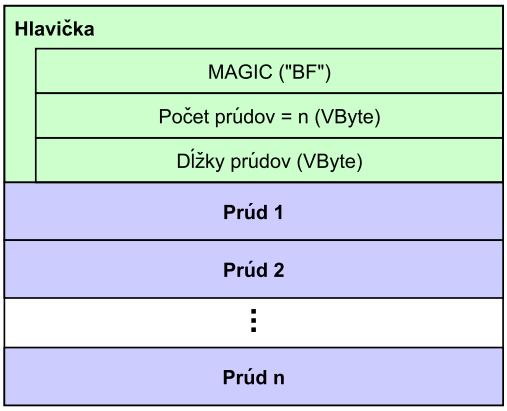
Balíček `samzip.processor` obsahuje abstraktnú triedu `SAMProcessor`, ktorá popisuje rozhranie SAM procesorov a ponúka metódy pre uľahčenie ich vývoja. Ďalej obsahuje implementáciu kompresnej pipeline `SAMProcessorPipeline`. Pre spracovanie kompresného profilu slúži trieda `SAMZipProfileReader`.

Balíček `samzip.index` obsahuje triedu `SAMZipIndexBuilder`, ktorá slúži pre vytvorenie indexu komprimovaného súboru. Ďalej obsahuje rozhranie `SAMZipIndex`, ktoré ponúka API pre prístup k tomuto indexu. S aplikáciou dodávame implementáciu indexu, ktorý popisujeme v sekcii 7.7

Pre rozšírenie možností aplikácie SamZip je možné vytvoriť novú kompresnú metódu alebo nový SAM procesor. Podrobnejšie informácie o možnostiach rozšírenia aplikácie SamZip uvádzame v dodatku D.

7.3 Formát komprimovaného súboru

Komprimovaný súbor vo formáte SamZip pozostáva z postupnosti blokov s formátom znázorneným na obrázku 7.2. Každý blok pozostáva z hlavičky a z ľubovoľného množstva komprimovaných prúdov. Hlavička pozostáva z magickej sekvencie obsahujúcej dva bajty so znakmi `'B'` a `'F'`. Ďalej nasleduje počet prúdov a dĺžky jednotlivých prúdov v bajtoch. Pre uloženie týchto hodnôt je používané VByte kódovanie čísel (viď sekcia 3.3.6).



Obr. 7.2: Formát bloku komprimovaného súboru

Prvý blok komprimovaného súboru obsahuje hlavičku SAM súboru a kompresný profil. Obe tieto položky sú komprimované metódou LZMA (viď sekcia 3.5.2). Ďalej už nasledujú bloky s komprimovanými dátami. Prvý prúd týchto blokov obsahuje hlavičku s počtom záznamov, ktoré obsahuje daný blok. Aj táto hodnota je kódovaná, podobne ako hodnoty v hlavičke bloku, VByte kódovaním. Za

posledným blokom nasledujú dáta indexu. Na konci komprimovaného súboru je uložená 8 bajtová hodnota ofsetu začiatku indexu v komprimovanom súbore.

7.4 Implementácia kompresie

Pri návrhu implementácie kompresie sme vyžadovali, aby splňovala:

- Poskytne možnosť jednoduchej implementácie rôznych typov kompresných metód (napr. blokové, prúdové, kódovania čísel),
- bude mať možnosť zreťazenia jednotlivých metód a
- umožní jednoduché nastavenie parametrov kompresie.

Zvolili sme implementáciu kompresných metód ako potomkov štandardných prúdov dostupných v Jave[6]. Výhodou tohto riešenia je, že môžeme takéto implementácie kompresných metód využiť v širokom kontexte, všade tam kde sa používajú štandardné prúdy. Riešenie je jednoducho použiteľné pri implementácii prúdovej kompresie. Pre implementáciu blokovej kompresie stačí, aby sme používali vyrovnávaciu pamäť, a dáta komprimovali blokovou kompresiou až keď ich budeme mať k dispozícii dostatočné množstvo[7]. Pri čítaní stačí dekomprimovať blok dát do vyrovnávacej pamäte a použiť prúdové rozhranie pre ich návrat k používateľovi.

Reprezentácia kompresných metód ako potomkov štandardných prúdov však priniesla problém, pretože štandardné prúdy v Jave umožňujú zapisovať a čítať dáta len po bajtoch. Základná metóda pre zápis do výstupného prúdu (`void write(int data)`) reprezentuje svoj parameter ako neznamienkovú bajtovú hodnotu. Metóda pre čítanie z vstupného prúdu (`int read()`) zase vracia neznamienkovú bajtovú hodnotu alebo číslo menšie ako 0, ak sme na konci vstupu. Niektoré kompresné metódy však potrebujú zapisovať a čítať dáta vo väčšom rozsahu (napríklad kódovania čísel umožňujú kódovať aj veľké číselné hodnoty). Tento problém sme vyriešili možnosťou nastavenia rozsahu vstupných dát pre každú kompresnú metódu, tzv. *dátových módov*. Zoznam dostupných dátových módov je uvedený v tabuľke 7.1. Druhý stĺpec tabuľky obsahuje hodnotu, ktorá označuje koniec prúdu (EOS). Tretí stĺpec obsahuje rozsah hodnôt konkrétneho dátového módu.

Komprimované dáta môžu kompresné metódy ukladať buď do štandardného prúdu, alebo využiť zreťazenie kompresných metód. Pod pojmom zreťazenie budeme rozumieť pripojenie výstupu kompresnej metódy k vstupu inej metódy. Ide teda o podobný prístup ako pri spájaní programov pomocou rúry[8]. Zreťazenie kompresných metód je povinné vtedy, keď je výstup kompresnej metódy potrebné spracovať ďalšou metódou. Konkrétna kompresná metóda môže využívať viacero takýchto zreťazení, ktoré sú pomenované, aby bola umožnená ich jednoduchá konfigurácia. Príkladom je kódovanie behov (RLE), ktoré vyžaduje jednu kompresnú

[6] V implementácii sme používali prúdy s výstupom do pamäte a do súboru

[7] Štandardná knižnica Javy má podobne implementovanú gzip kompresiu

[8] Funkcia príkazovej riadky, ktorá umožňuje spojiť vstup programu s výstupom iného programu

Mód	EOS	Rozsah
BYTE	-2^{31}	$[-128, 127]$
UBYTE	-1	$[0, 255]$
SHORT	-2^{31}	$[-32768, 32767]$
USHORT	-1	$[0, 65535]$
INT	-2^{31}	$[-2^{31}+1, 2^{31}-1]$
UINT	-1	$[0, 2^{31}-1]$

Tab. 7.1: Dostupné dátové módy

metódu pre kódovanie symbolov behov a ďalšiu pre kódovanie dĺžok behov. Podporované dátové módy v tomto prípade závisia len na implementácii zreťazených metód.

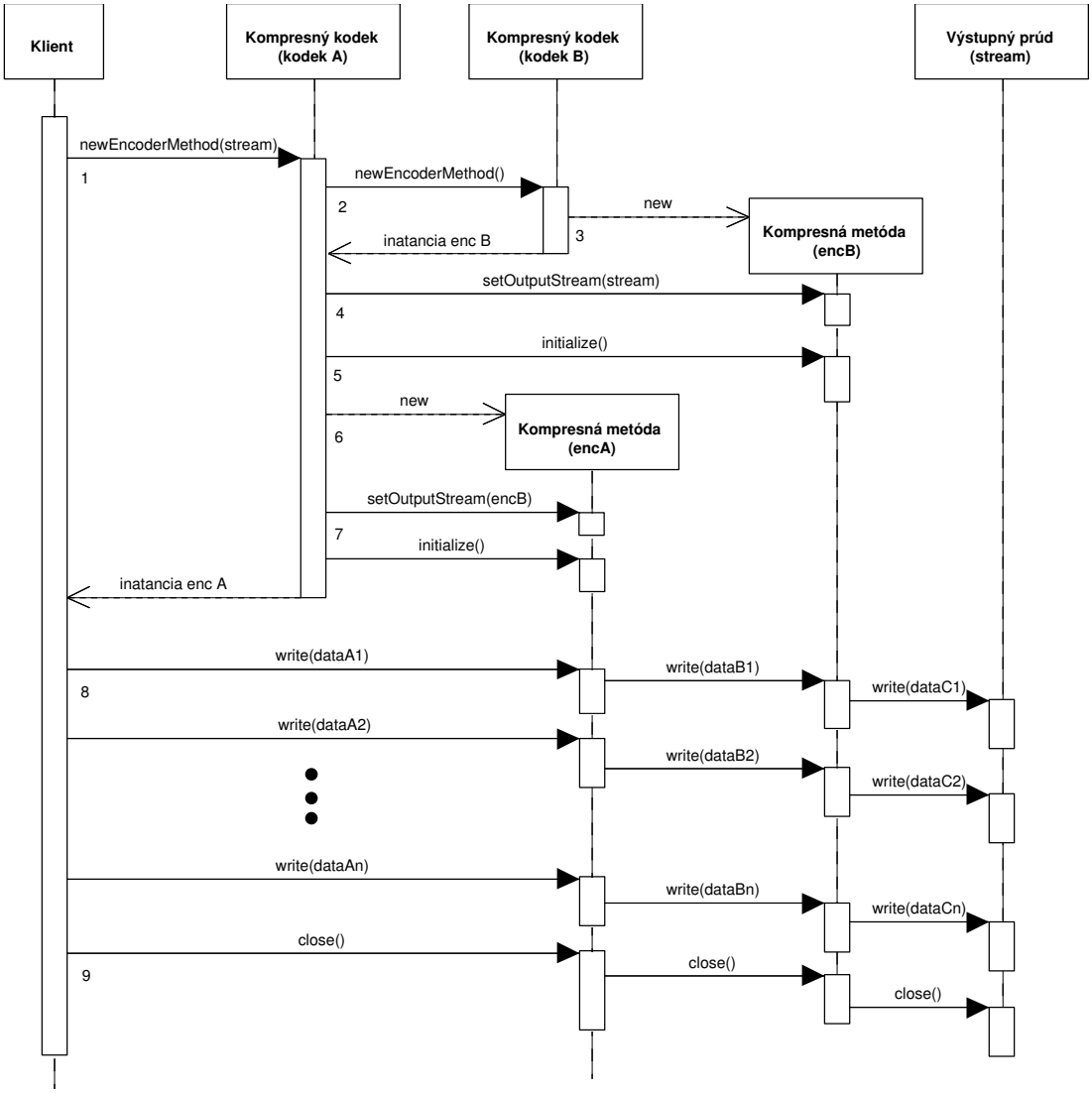
Ľubovoľné dve kompresné metódy je možné zreťaziť, ak použijeme jednu z nich ako výstupný prúd druhej. Výhodou tohto spôsobu zreťazenia oproti povinnému zreťazeniu je, že je univerzálne a nezávislé od implementácie kompresnej metódy. Nevýhodou však je, že výstupná metóda má obmedzený dátový mód na `UBYTE`. Týmto spôsobom je možné vytvoriť ľubovoľne dlhý reťazec kompresných metód.

Aby bolo možné jednoduché nastavenie parametrov kompresie využívame abstrakciu kompresných kodekov. Kompresný kodek umožňuje nastaviť parametre kompresie a vytvoriť inštancie enkodéra a dekodéra s požadovanou konfiguráciou. Zreťazením kompresných kodekov je umožnené jednoduché vytvorenie zreťazených enkodérov a dekodérov.

Pre predstavu uvádzame diagram 7.3, ktorý znázorňuje spôsob práce s kompresnou metódou. Predpokladáme, že klient má k dispozícii inštanciu kompresného kodeku A, ktorý je zreťazený s kodekom B. Chce použiť kompresnú metódu kodeku A na zápis dát do výstupného prúdu `stream`. Klient požiada kodek A o vytvorenie kompresnej metódy s výstupným prúdom `stream` (1). Kompresný kodek A požiada kodek B o vytvorenie kompresnej metódy (2). Ten vytvorí kompresnú metódu (3) a vráti jej inštanciu `encB`. Kodek A nastaví prúd `stream` ako výstupný prúd metóde `encB` (4) a zinicializuje ju (5). Ďalej vytvorí inštanciu kompresnej metódy `encA` (6). Potom nastaví jej výstupný prúd na `encB` (7) a zinicializuje ju. Po návrate inštancie kompresnej metódy `encA` môže klient začať zapisovať (8). Prácu s kompresnou metódou je potrebné zakončiť jej uzavretím (9), kedy sa kompresná metóda postará o uzavretie metód, ktoré ma zreťazené a o uzavretie výstupného prúdu. Analogicky prebieha práca s dekompresnými metódami.

7.5 Kompresný profil

Jednou z podmienok, ktorú sme si kládli pri návrhu aplikácie SamZip, bolo aby umožňoval jednoduché definovanie spôsobu kompresie bez zmeny samotnej aplikácie. Dôvodom bolo zjednodušenie testovania kompresných metód v tejto práci a jednoduchá rozšíriteľnosť v prípade budúceho vývoja. Postup kompresie preto



Obr. 7.3: Príklad použitia kompresnej metódy

nie je natvrdo implementovaný v aplikácii, ale je určený nastaveniami v konfiguračnom súbore – kompresnom profile. Užívateľ môže pri kompresii špecifikovať profil, ktorý najlepše vyhovuje jeho požiadavkám na rýchlosť a úroveň kompresie pre jeho špecifické dáta.

Z technického hľadiska sa jedná o štandardný XML súbor. Formát XML bol zvolený, pretože je štrukturovaný, jednoducho vytvoriteľný a ľahko čitateľný človekom. Aby bola umožnená dekompresia, je kompresný profil pri kompresii uložený v komprimovanom súbore. Kompresný profil obsahuje dve sekcie:

- **Definície kompresných metód.** Táto sekcia môže obsahovať ľubovoľné množstvo definícii kompresných kodekov. Každá definícia kompresného kodeku umožňuje nastavenie parametrov daného kodeku.
- **Definície SAM procesorov.** V tejto sekcii je uvedená postupnosť definícií SAM procesorov, ktoré tvoria kompresnú pipeline. Každá definícia SAM procesora môže obsahovať nastavenie jeho parametrov a používaných kodekov.

Podrobný popis formátu kompresného profilu, ako aj popis kompresných kodekov a SAM procesorov, ktoré sú dodávané s našou implementáciou je uvedený v dodatku C.

7.6 SAM procesor

SAM procesory ponúkajú abstrakciu operácii nad SAM súbormi. Umožňujú implementáciu aj úplne nových prístupov ku kompresii/spracovaniu SAM súborov.

Základným účelom SAM procesorov je kompresia bloku záznamov. Každý SAM procesor môže spracovávať ľubovoľné polia SAM súboru. Pri implementácii SAM procesora je potrebné určiť, ktoré polia spracováva (komprimuje)[9], a ktoré polia už musia byť spracované inými procesormi pred jeho spustením (závislosti).

Pre predstavu uvádzame diagram 7.4, ktorý znázorňuje typickú činnosť SAM procesora pri kompresii. SAM procesory sú uložené v kompresnej pipeline. Táto pri kompresii postupne (sekvenčne) spustí každý SAM procesor, ktorý obsahuje. Každému z nich umožní realizovať sa pri kompresii poľa SAM záznamov (pole objektov `SAMRecord`) do bloku `fileBlock` v komprimovanom súbore (1).

SAM procesor má prístup k slovníku kompresných kodekov, ktorý bol vytvorený pri konfigurácii. Najskôr z tohto slovníku získa kompresný kodek, ktorý potrebuje ku kompresii (2). Potom vytvorí inštanciu kompresnej metódy tohto kodeku (3). Kodek sa sám postará o vytvorenie nového prúdu v bloku `fileBlock` (4). SAM procesor potom inštanciu vytvorenej kompresnej metódy využije ku kompresii dát, ktoré spracováva (5). Nakoniec SAM procesor zakončí kompresiu a skončí (6). Kompresná pipeline potom spustí ďalší SAM procesor v poradí.

SAM procesor má k dispozícii referenciu k poľu SAM záznamov, ktoré má spracovávať. Zmeny týchto záznamov sú potom viditeľné aj pre ďalšie procesory v pipeline. SAM procesor preto môže dáta spracovávať aj iným spôsobom než len kompresiou – napr. je možné naimplementovať SAM procesor, ktorý odstráni alebo zmení určité časti dát[10], ale nepostará sa o kompresiu týchto dát[11].

Referenciu k poľu SAM záznamov dostáva SAM procesor aj pri dekompresii. V tomto prípade SAM procesor zapisuje dekomprimované dáta do tohto poľa. Po dokončení všetkých SAM procesorov v pipeline toto pole obsahuje dekomprimované dáta.

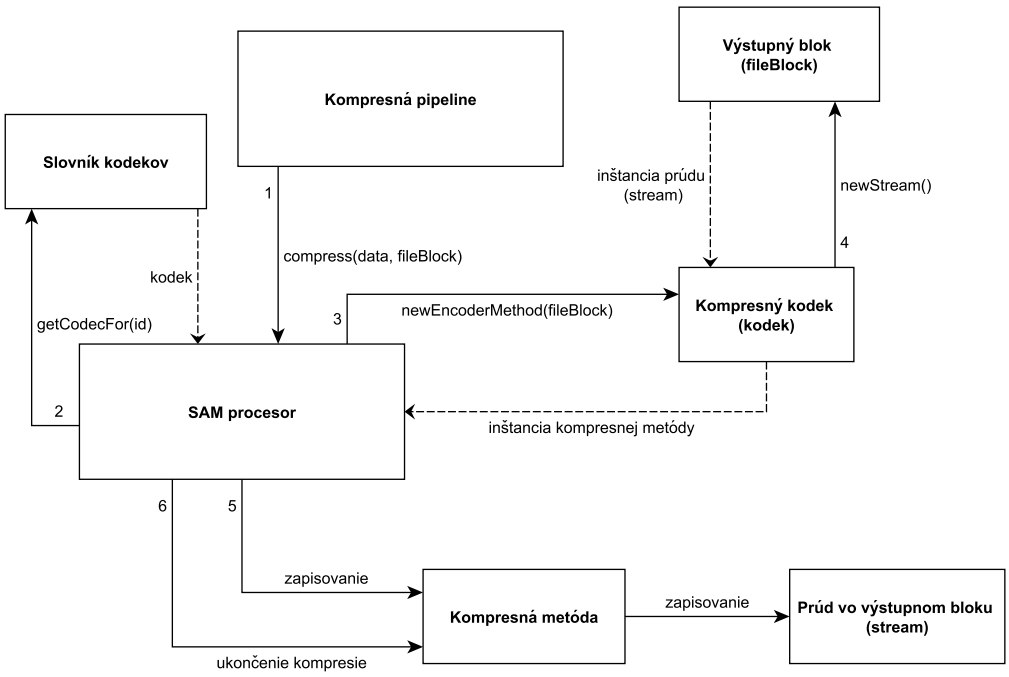
7.7 Index

Aplikácia SamZip umožňuje okrem dekompresie celého súboru aj dekompresiu zvolenej časti dát. K nájdeniu bloku, v ktorom sa nachádzajú požadované dáta

[9] Umožňuje to kontrolu konfigurácie v kompresnom profile, aby nebolo konkrétne pole uložené v komprimovanom súbore viac krát

[10] Napr. uloží len rozdiely oproti referenčnej sekvencii v poliach `CIGAR` a `SEQ`

[11] Túto úlohu môže prevziať ďalší SAM procesor v pipeline



Obr. 7.4: Príklad kompresie s využitím SAM procesora

využíva index, ktorý je vytvorený pri kompresii a uložený na konci komprimovaného SamZip súboru. Pri konštrukcii indexu sme využívali predpoklad usporiadania SAM záznamov v súbore – primárne podľa referenčnej sekvencie a sekundárne podľa pozície zarovnania k tejto sekvencii.

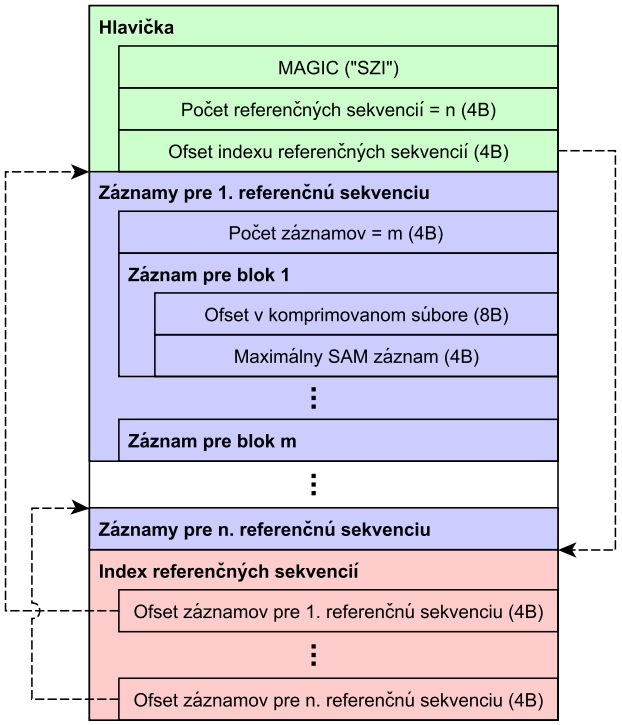
Formát indexu je znázornený na obrázku 7.5. Index začína hlavičkou (zelená časť v obrázku). Na začiatku hlavičky je uvedená magická sekvencia `"SZI"`. Potom nasleduje počet referenčných sekvencií v súbore. Na konci hlavičky je uvedený ofset indexu referenčných sekvencií.

Ďalej nasleduje hlavná časť indexu (modrá časť). Pre každú referenčnú sekvenciu[12] obsahuje postupnosť položiek, ktoré reprezentujú bloky obsahujúce SAM záznamy zarovnané k tejto referenčnej sekvencii. Pred prvou položkou indexu konkrétnej referenčnej sekvencie sa nachádza počet položiek pre túto referenčnú sekvenciu. Každá položka indexu obsahuje informácie o jednom bloku v komprimovanom súbore – ofset začiatku bloku v komprimovanom súbore a maximálna pozícia konca zarovnania[13] SAM záznamov v tomto a vo všetkých predchádzajúcich blokoch.

Na konci indexu sa nachádza index referenčných sekvencií (červená časť), ktorý pre každú referenčnú sekvenciu obsahuje ofset začiatku indexových položiek pre túto referenčnú sekvenciu v hlavnej časti indexu.

Pre spracovanie dotazu na SAM záznamy, ktoré pokrývajú referenčnú sekvenciu `ref` na pozícii `pos`, stačí v položkách v hlavnej časti indexu pre referenčnú sekvenciu `ref` vyhľadať prvú, ktorej hodnota maximálnej pozície konca zarovnania je väčšia alebo rovná ako `pos`. Teraz stačí dekomprimovať blok, ktorý začína na

[12]Ich zoznam je uvedený v slovníku referenčných sekvencií v hlavičke súboru
[13]Hodnota v poli `POS` plus dĺžka zarovnania vypočítaná z obsahu CIGAR reťazca



Obr. 7.5: Formát indexu komprimovaného súboru

ofsete uvedenom v tejto položke, a nájsť prvý SAM záznam pre ktorý platí, že prekrýva pozíciu `pos`. Od tejto chvíle sa už jedná o sekvenčnú dekompresiu – postupne dekomprimujeme ďalšie bloky v komprimovanom súbore a vraciame tie záznamy, ktoré vyhovujú dotazu. Pre tento konkrétny dotaz je potrebné ukončiť dekompresiu, keď narazíme na SAM záznam, ktorého pozícia začiatku zarovnania (obsah poľa `POS`) je väčšia ako `pos`. Zmena podmienky ukončenia dekompresie umožňuje implementáciu ďalších dotazov – napr. získanie SAM záznamov, ktoré prekrývajú referenčnú sekvenciu v určitom intervale (dekompresiu ukončíme, ak neplatí podmienka pokrytia konca intervalu).

Poznámka 7.2. Všetky číselné hodnoty v indexe sú celé čísla bez znamienka s poradím bajtov big endian.

Všetky ofsety okrem ofsetu bloku v komprimovanom súbore sú v indexe uložené ako ofsety od začiatku indexu (vrátane hlavičky indexu).

8. Výber kompresných metód

V tejto kapitole popíšeme spôsob výberu vhodných kompresných metód pre blokovú kompresiu SAM súborov v aplikácii SamZip. Na začiatku v sekcii 8.1 popíšeme, akým spôsobom prebiehalo testovanie a výber vhodných metód. V sekcii 8.2 sa budeme zaoberať voľbou vhodnej veľkosti komprimovaného bloku.

V ďalších častiach kapitoly sa už budeme zaoberať samotným výberom kompresných metód. Najskôr v sekciách 8.3 a 8.4 popíšeme voľbu vhodných kompresných metód pre polia, ktoré tvoria hlavnú časť SAM súboru – `CIGAR`, `SEQ` a `QUAL`. V sekcii 8.5 sa budeme zaoberať výberom kompresných metód pre polia, ktoré nesú informáciu o párových čítaniach – `RNEXT`, `PNEXT` a `TLEN`. Potom v sekciách 8.6 a 8.7 popíšeme výber vhodných metód pre ostatné povinné resp. nepovinné polia. Nakoniec v sekcii 8.8 zhodnotíme kombinácie metód, ktoré boli najvhodnejšie pre jednotlivé polia, a v sekcii 8.9 budeme diskutovať o možnosti stratovej kompresie kvalít.

8.1 Metodika testovania

Návrh dobrej kompresnej schémy si vyžaduje navrhnutie vhodných postupov pre testovanie kompresných metód ako aj postupov na vyhodnotenie týchto testov. Naším cieľom bolo navrhnúť testy tak, aby boli reprodukovateľné na systémoch totožných s naším testovacím strojom. Navyše sme požadovali, aby bolo v budúcnosti možné tieto testy opakovať a zvoliť vhodnejšie metódy ako tie, ktoré dopadli najlepšie v našich testoch. Týmto spôsobom je pomerne jednoduché prispôsobiť kompresnú schému použitú v aplikácii SamZip konkrétnej situácii – je možné reagovať na zmeny v architektúre nových počítačových a sekvenovacích systémov.

V sekcii 8.1.1 popíšeme postup testovania vhodných kompresných metód. V sekcii 8.1.2 popíšeme spôsob prezentácie výsledkov našich experimentov. V sekcii 8.1.3 uvedieme popis vstupných dát, na ktorých sme testovali zvolené metódy. V sekcii 8.1.4 uvedieme konfiguráciu počítača, na ktorom sme testovali.

8.1.1 Postup testovania

Väčšina polí SAM súboru obsahuje dáta nezávislé na ostatných poliach. Dáta v jednom stĺpci sú si naproti tomu veľmi podobné. Preto najskôr vyberieme najvhodnejšie metódy pre jednotlivé polia SAM súboru a až potom sa budeme zaoberať ich kombináciami. Aby sme boli schopní zvoliť vhodnú metódu pre rôznorodé dáta, všetky kompresné metódy budeme testovať na viacerých dátových súboroch. Popis týchto súborov uvádzame v sekcii 8.1.3.

Pre každé pole zvolíme a otestujeme metódy, ktoré pokladáme za vhodné pre daný typ vstupných dát. Budeme sledovať účinnosť kompresie a časovú náročnosť testovaných metód. Pri finálnom výbere najvhodnejších metód budeme klásť

vyššiu váhu rýchlosti dekompresie ako rýchlosti kompresie. Ak to bude vhodné, zvolíme pre každé pole viacero vhodných metód – metódu s najlepšou účinnosťou kompresie a metódu, ktorá ponúka rozumný kompromis medzi svojou účinnosťou a rýchlosťou.

8.1.2 Prezentácia výsledkov

Poznámka 8.1. Kvôli použitej kompresii nie sme schopní zistiť veľkosť jednotlivých polí v súbore, ktorý je vo formáte BAM. Preto budeme udávať ich veľkosť v textovej podobe vo formáte SAM.

Tabuľka 8.1 je príkladom tabuľky, ktorú budeme používať pre znázornenie účinnosti kompresie testovaných metód. Riadky tejto tabuľky predstavujú jednotlivé kompresné metódy. Stĺpec „kompresný pomer“ udáva pomer veľkosti komprimovaných dát k veľkosti vstupných dát pre jednotlivé testované súbory. Ak to nebude vyslovene spomenuté inak, tieto veľkosti predstavujú veľkosti dát v textovej podobe v SAM súbore. Aby sme lepšie zvýraznili úroveň kompresie, riadky tabuľky sú zoradené podľa priemeru kompresných pomerov pre jednotlivé testované súbory. Najlepší kompresný pomer pre jednotlivé súbory je zvýraznený tučným písmom a šedou farbou pozadia. Navyše ak budeme testovať väčšie množstvo metód, pre prehľadnosť zvýrazníme tri najlepšie kompresné pomery pre daný súbor tučným písmom.

Jednotlivé polia neprispievajú rovnakou mierou k veľkosti výsledného súboru. Chceme, aby bol dostatočne viditeľný nielen dopad kompresie na veľkosti samostatných polí, ale aby bol viditeľný aj jej dopad na veľkosť výsledného komprimovaného súboru. Preto v poslednom stĺpci tejto tabuľky uvádzame priemerný pomer veľkosti komprimovaných dát k priemernej veľkosti testovacích súborov vo formáte BAM.

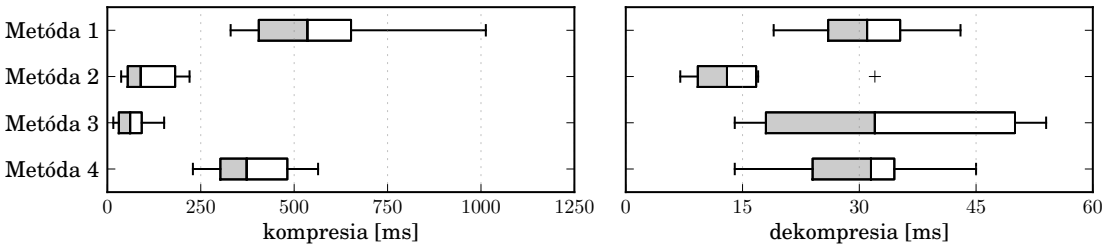
Kompresná metóda	**Kompresný pomer**						**BAM** %
	HG00109	**NA20589**	**KB1454**	**Manuel**	**Mouse**	**Ecoli**	
Metóda 1	**50.96%**	**47.34%**	**46.16%**	**48.49%**	**42.76%**	**37.87%**	12.03%
Metóda 2	**51.08%**	**47.40%**	49.50%	**48.67%**	**43.33%**	**38.12%**	12.10%
Metóda 3	**51.11%**	48.06%	**48.06%**	**49.16%**	**43.10%**	**38.48%**	12.18%
Metóda 4	51.81%	**47.35%**	**49.35%**	49.59%	43.34%	38.78%	12.27%

Tab. 8.1: Príklad tabuľky, ktorú budeme používať pre znázornenie účinnosti kompresných metód

Obrázok 8.1 je príkladom diagramu, ktorý budeme používať pre znázornenie časovej náročnosti testovaných metód. Jedná sa o tzv. boxplot diagram, ktorý znázorňuje hodnoty časov kompresie resp. dekompresie jednotlivými metódami na testovacích súboroch. Boxplot diagram obsahuje pre každú testovanú metódu obdĺžnik, ktorého ľavá hrana tvorí dolný kvartil a pravá hrana horný kvartil týchto hodnôt. Vo vnútri obdĺžnika je čiarou znázornený medián. Z obdĺžnika sú vedené čiary (tzv. fúzy), ktoré dosahujú po minimum z maximálnej hodnoty a 1.5 násobku kvartilového rozpätia[1]. V prípade, že sa niektorá hodnota nachádza mimo

[1]Rozdiel hodnôt horného a dolného kvartilu

týchto fúzov, je táto hodnota v diagrame uvedená ako samostatný bod. Vždy budeme uvádzať dvojicu týchto diagramov – jeden pre časy kompresie a ďalší pre časy dekompresie.



Obr. 8.1: Príklad boxplot diagramov, ktoré budeme používať pre znázornenie časovej náročnosti kompresných metód

V texte, tabuľkách a v diagramoch budeme používať značenie testovaných kompresných metód tak, ako je uvedené v tabuľke 8.2.

8.1.3 Vstupné dáta

Pre testy kompresie sme zvolili použitie vstupných súborov, ktoré sme vytvorili z nasledujúcich dátových súborov:

- `HG00109` Dvadsiaty chromozóm človeka s nízkym pokrytím. K sekvenovaniu bol použitý sekvenátor SOLiD. Dáta pochádzajú z projektu „1000 Genomes Project“.
- `NA20589` Dvadsiaty chromozóm človeka s nízkym pokrytím. K sekvenovaniu bol použitý sekvenátor ILLUMINA. Dáta pochádzajú z projektu „1000 genomes project“.
- `KB1454` Dvadsiaty chromozóm človeka z Afriky. K sekvenovaniu bol použitý sekvenátor Roche 454. Neobsahuje párové čítania a žiadne nepovinné polia. Dáta pochádzajú zo štúdie [45].
- `Manuel` Exónová oblasť genómu Manuela Corpasa. K sekvenovaniu bol použitý sekvenátor ILLUMINA. Zdroj dát: `<http://manuelcorpas.com/tag/bam/>`.
- `Mouse` Kompletný genóm myši s vysokým pokrytím (kmeň „WSB/EiJ“). K sekvenovaniu bol použitý sekvenátor ILLUMINA. Dáta pochádzajú z projektu „Mouse Genomes Project“.
- `Ecoli` Kompletný genóm baktérie Escherichia coli s vysokým pokrytím. K sekvenovaniu bol použitý sekvenátor ILLUMINA. Neobsahuje párové čítania. Zdroj dát: `<http://ncbi.nih.gov>`.

Každý testovací súbor obsahuje 200 000 záznamov (riadkov SAM súboru). Tieto záznamy pozostávajú z náhodne vybraných neprekrývajúcich sa blokov, z ktorých každý obsahuje 5000 po sebe idúcich záznamov. Použitie 5000 po sebe idúcich záznamov nie je náhodné. Táto voľba bude diskutovaná v sekcii 8.2.

Identifikátor	Popis
RLEU+X	Behové kódovanie – implementácia RLEU s výstupom kódovaným metódou X (3.2.1)
RLEU+X+Y	Behové kódovanie – implementácia RLEU, ktorá kóduje symboly metódou X a dĺžky behov metódou Y (3.2.1)
RLE+X	Behové kódovanie – implementácia RLE, ktorej výstup je komprimovaný metódou X (3.2.1)
Delta+X	Delta kódovanie s výstupom kódovaným metódou X (3.2.2)
Mean+X	Delta kódovanie vzhľadom k hodnote aritmetického priemeru vstupných hodnôt s výstupom kódovaným metódou X (3.2.2)
Median+X	Delta kódovanie vzhľadom k hodnote mediánu vstupných hodnôt s výstupom kódovaným metódou X (3.2.2)
MTF+X	Kódovanie s presunom na začiatok s výstupom kódovaným metódou X (3.2.3)
EliasGamma	Eliasov gama kód (3.3.3)
EliasDelta	Eliasov delta kód (3.3.3)
Fibonacci	Fibonacciho kód (3.3.4)
Golomb	Golombov kód s adaptívnym parametrom určeným priemerom vstupných hodnôt (3.3.5)
Rice	Riceov kód s adaptívnym parametrom určeným priemerom vstupných hodnôt (3.3.5)
VByteGZip	VByte kód, ktorého výstup je komprimovaný metódou GZip (3.3.6)
VByteLZMA	VByte kód, ktorého výstup je komprimovaný metódou LZMA (3.3.6)
Arithmetic	Aritmetické kódovanie (adaptívna varianta) (3.4.2)
Huffman	Huffmanovo kódovanie (implementácia Kanonického Huffmanovho kódu) (3.4.1)
GZip	Slovníková metóda GZip (3.5.1)
LZMA	Slovníková metóda LZMA (3.5.2)
BZip2	Metóda BZip2 (3.6)

Tab. 8.2: Identifikátory kompresných metód, ktoré používame v tejto kapitole. Znakmi X a Y sú označené miesta v identifikátoroch, za ktoré môže byť dosadený identifikátor inej kompresnej metódy. V zátvorke uvádzame sekciu, v ktorej sa nachádza popis konkrétnej metódy

V tabuľke 8.3 je uvedená veľkosť testovacích súborov vo formátoch SAM a BAM. Na obrázku 8.2 je znázornený podiel jednotlivých polí na veľkosti testovacích súborov vo formáte SAM. V tabuľke 8.4 uvádzame entrópiu dát v jednotlivých poliach týchto súborov. Polia `SEQ` a `QUAL` tvoria veľkú časť testovacích súborov. Pole `SEQ` má pomerne nízku entrópiu, preto by jeho kompresia nemala predstavovať veľký problém. Problematické je však pole `QUAL`, ktoré ma skoro najvyššiu entrópiu spomedzi všetkých polí. Pri troch z testovacích súborov sa taktiež nachádzajú niektoré z nepovinných polí `CS`, `CQ`, `OQ` a `BQ`, ktorých veľkosť je rovnaká ako veľkosť polí `SEQ` a `QUAL`. Pole `BQ` má nízku entrópiu, preto bude jednoducho komprimovateľné. Vzhľadom na podobné hodnoty entrópie bude kompresia polí `CQ` a `OQ` predstavovať podobne zložitý problém ako u poľa `QUAL`.

Súbor	**Veľkosť SAM**	**Veľkosť BAM**	**|BAM| / |SAM|**	**bity/báza**
HG00109	60 026 091	22 592 495	37.64%	18.07
NA20589	61 060 712	15 215 372	24.92%	11.01
KB1454	156 230 274	44 862 834	28.72%	5.00
Manuel	91 330 895	26 705 623	29.24%	11.87
Mouse	51 902 429	14 518 863	27.97%	8.85
Ecoli	69 720 585	14 519 680	20.83%	5.81

Tab. 8.3: Veľkosť testovacích súborov vo formátoch SAM a BAM v bajtoch. Stĺpec BAM/SAM obsahuje pomer veľkosti súboru vo formáte BAM k veľkosti súboru vo formáte SAM. Stĺpec bity/báza udáva priemerný počet bitov použitých pre uloženie jednej bázy v BAM súbore

8.1.4 Platforma

Testy kompresných metód boli vykonávané na počítači s procesorom `Intel Core i7 920@2.67Ghz`. Jedná sa o 4 jadrový procesor, ktorý umožňuje vykonávať 8 nezávislých vlákien.

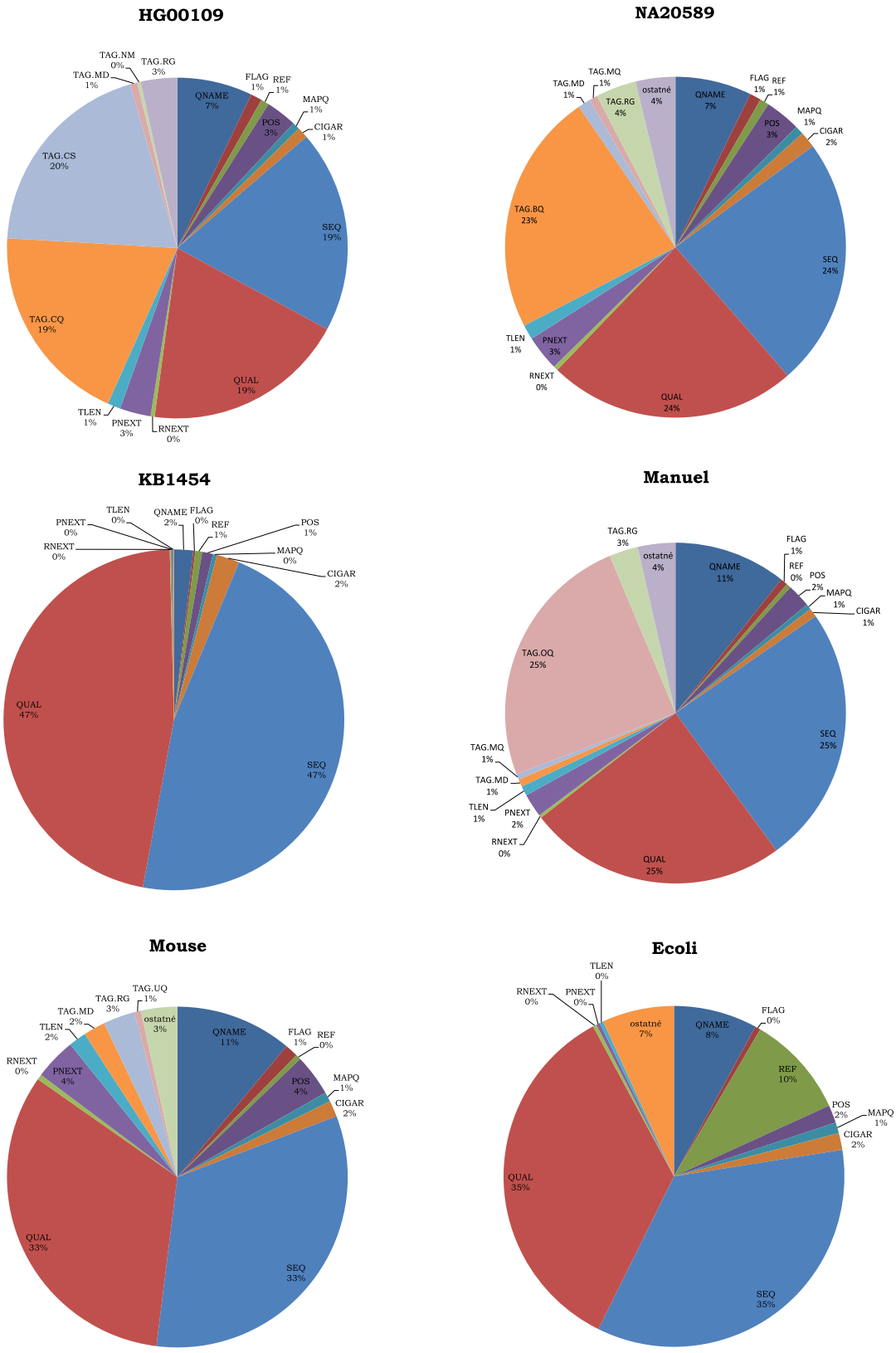
Na počítači bežal 32 bitový operačný systém Gentoo Linux s kernelom vo verzii 3.0.29. Použili sme JVM OpenJDK vo verzii 1.7.0_03 (IcedTea 2.1).

Testy sme spúšťali nasledujúcim volaním:

```
java -Xms256m -Xmx1024m -server -jar samzip.jar test -p={kompresný
    profil} --in={vstupný BAM súbor} --out={výstupný komprimovaný
    súbor} --dec={dekomprimovaný súbor} -t=4 -b=5000 --stringency=
    LENIENT
```

Využívali sme 4 vlákna a bloky s veľkosťou 5000 záznamov. Behové prostredie Javy malo k dispozícii maximálne 1024 MB pamäte RAM.

Na porovnateľne výkonnom počítači by sa mali testy kompresných metód správať podobne ako na tomto testovacom stroji. Kedže kompresia pomerne dosť zaťažuje pevný disk predpokladáme, že rýchlosť pevného disku bude prispievať nezanedba-



Obr. 8.2: Podiel SAM polí na veľkosti testovacích súborov. Označenie tvaru `TAG.XX` predstavuje nepovinné pole `XX`

Pole	Entrópia [bit]					
	HG00109	**NA20589**	**KB1454**	**Manuel**	**Mouse**	**Ecoli**
QNAME	3.47	3.60	4.90	4.13	3.64	3.63
FLAG	2.53	2.79	1.58	2.79	2.76	1.58
REF	1.00	1.00	2.32	2.48	0.00	3.88
POS	3.30	3.30	3.31	3.31	3.30	3.30
MAPQ	2.85	2.02	0.92	1.63	1.58	1.14
CIGAR	1.93	2.84	3.34	1.79	2.50	2.20
SEQ	2.08	1.99	2.00	2.00	1.99	2.01
QUAL	4.40	4.38	4.46	4.00	4.15	3.47
RNEXT	0.00	0.00	0.00	0.00	0.00	0.00
PNEXT	3.30	3.30	0.00	3.31	3.30	0.00
TLEN	3.21	3.32	0.00	3.18	3.34	0.00
TAG:BQ	*	0.19	*	*	*	*
TAG:CQ	4.18	*	*	*	*	*
TAG:CS	2.11	*	*	*	*	*
TAG:OQ	*	*	*	3.65	*	*

Tab. 8.4: Empiricky získané hodnoty entrópie jednotlivých SAM polí pre testovacie súbory. Hviezdička je uvedená pri poliach, ktoré sa nevyskytujú v konkrétnom dátovom súbore

teľným spôsobom k celkovému času kompresie a dekompresie. Rýchlosť pevného disku by však nemala ovplyvňovať pomer medzi rýchlosťami jednotlivých metód. Veľkosť pamäte RAM nad hranicou, ktorá je využívaná behovým prostredím Javy nemá vplyv na rýchlosť ani úroveň kompresie.

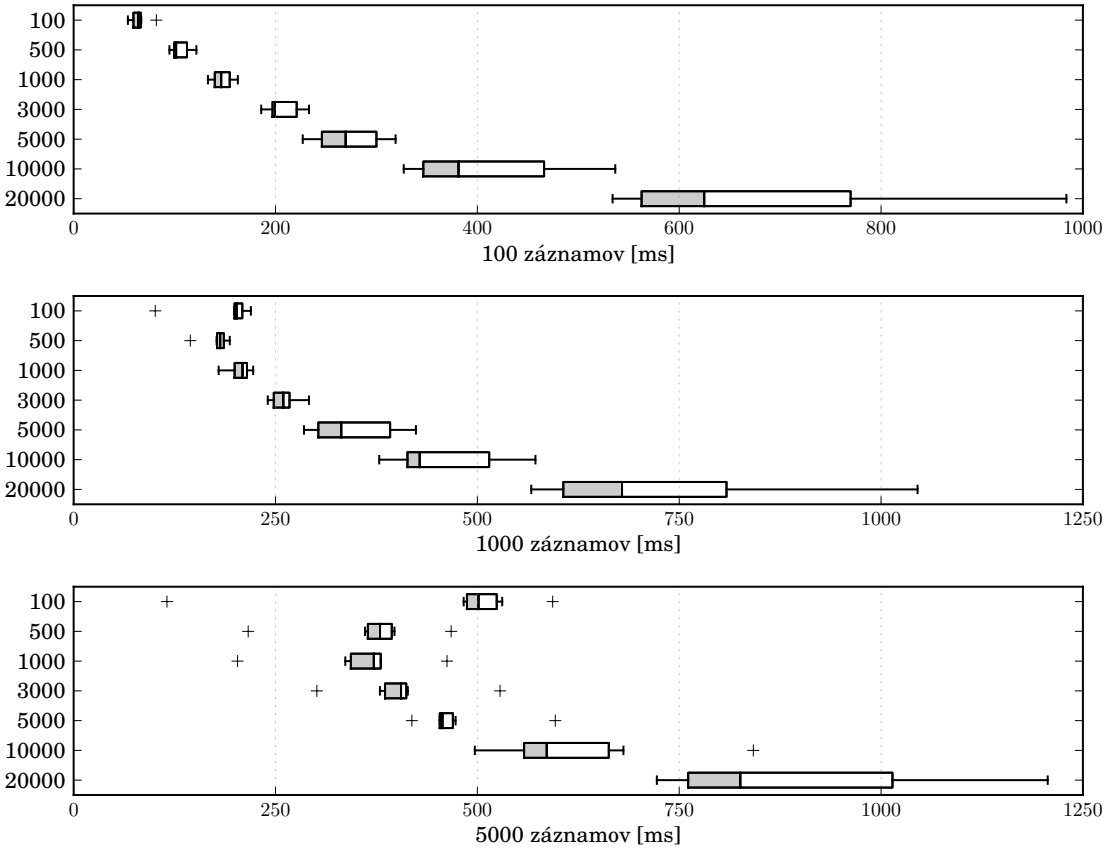
8.2 Veľkosť bloku

SAM súbory sú zvyčajne veľmi veľké – často dosahujú až desiatky GB. Preto je dôležité umožniť náhodný prístup aj k dátam v komprimovanom súbore SAM. Formát BAM bol navrhnutý tak, aby umožňoval náhodný prístup k svojmu obsahu. Kompresná metóda používaná v súboroch BAM pred kompresiou rozdeľuje vstupné dáta do blokov o maximálnej veľkosti 64kB, ktoré sú komprimované nezávisle na sebe. Formát SamZip nie je limitovaný pevnou veľkosťou bloku ako to je vo formáte BAM. Bloky v SamZip súbore nemajú pevnú veľkosť, obsahujú však pevný počet záznamov.

Užívateľ môže pri kompresii špecifikovať požadovaný počet záznamov, ktoré sa uložia v jednom bloku. Pretože testovanie rôznych nastavení tohto parametra spolu s rôznymi kompresnými metódami by bolo časovo náročné, rozhodli sme sa v našich experimentoch pracovať s pevne nastavenou hodnotou veľkosti bloku. Aby sme mali predstavu o dopade veľkosti bloku na účinnosť kompresie, vyskú-

šali sme skomprimovať vstupné súbory metódou LZMA s rôznymi nastaveniami veľkosti bloku. Metódu LZMA sme zvolili, pretože sa jedná o slovníkovú metódu, u ktorej predpokladáme, že je najviac závislá na veľkosti bloku vstupných dát. V tabuľke 8.5 sú uvedené účinnosti kompresie pre rôzne hodnoty počtu záznamov v bloku. Ako sa dalo predpokladať, pri použití väčšieho bloku sme dosiahli lepší kompresný pomer.

V diagrame 8.3 uvádzame rýchlosť dekompresie pri náhodnom prístupe ku komprimovanému súboru v závislosti na veľkosti bloku.



Obr. 8.3: Boxplot diagram času dekompresie 100, 1000 a 5000 po sebe idúcich záznamov z náhodného miesta v komprimovanom súbore. Hodnoty sme získali 100 opakovaniami pre každý testovací súbor. Vertikálna osa diagramu reprezentuje počet záznamov v bloku

Za predpokladu, že kompresná metóda používaná vo formáte BAM dosahuje kompresný pomer 30% oproti veľkosti SAM súboru, tak veľkosť jedného bloku vo formáte BAM dosahuje priemerne 20kB. Podľa priemerných veľkostí blokov uvedených v tabuľke 8.5 preto predpokladáme, že jeden blok BAM súboru priemerne obsahuje 100 až 500 záznamov[2]. Táto hodnota je podľa nášho názoru zbytočne malá. Budeme preto používať bloky obsahujúce 5000 záznamov, ktoré podľa nás predstavujú vhodný kompromis medzi veľkosťou bloku, účinnosťou kompresie a časom potrebným pre náhodný prístup ku komprimovaným dátam.

[2]Pri testovacích súboroch bola priemerná veľkosť bloku 11kB pri 100 uložených záznamoch a 56kB pri 500 uložených záznamoch

Počet záznamov v bloku	Kompresný pomer						Priemerná veľkosť bloku [kB]
	HG00109	NA20589	KB1454	Manuel	Mouse	Ecoli	
20000	69.04%	62.08%	69.51%	66.50%	63.86%	68.03%	2253
10000	69.43%	62.64%	69.73%	66.75%	64.29%	68.50%	1126
5000	69.93%	63.41%	70.09%	67.06%	64.93%	69.16%	563
3000	70.47%	64.19%	70.56%	67.44%	65.69%	69.80%	336
1000	72.03%	66.47%	71.96%	68.51%	68.04%	71.70%	113
500	73.78%	68.70%	73.21%	69.84%	70.66%	73.59%	56
100	81.96%	79.33%	78.29%	77.15%	83.25%	82.93%	11

Tab. 8.5: Účinnosť kompresie metódou LZMA v závislosti na počte záznamov v jednom bloku. V poslednom stĺpci uvádzame priemernú veľkosť blokov v textovej podobe, ktoré obsahujú daný počet záznamov. Kompresný pomer je udávaný vzhľadom k veľkosti vstupného súboru vo formáte BAM

8.3 Polia `CIGAR` a `SEQ`

Hlavnou časťou SAM súboru je pole `SEQ`. S týmto poľom úzko súvisí pole `CIGAR`. SAM súbor obsahuje záznamy, ktoré sú zarovnané k referenčnej sekvencii. Je preto možné pri kompresii ukladať, namiesto celého obsahu poľa `SEQ`, len rozdiely oproti tejto referenčnej sekvencii. Pre účely uloženia informácií o týchto rozdieloch poupravíme originálny CIGAR reťazec. Pri dekompresii zrekonštruujeme pôvodný obsah polí `CIGAR` a `POS` pomocou referenčnej sekvencie použitej pri kompresii.

Nevýhodou použitia referenčnej sekvencie pri kompresii je, že musíme mať k dispozícii rovnakú referenčnú sekvenciu aj pri dekompresii. Preto sme zvolili aj ďalší prístup, kedy si pri kompresii vytvoríme vlastnú referenčnú sekvenciu z dostupných dát. Túto sekvenciu je možné vytvoriť aj pri dekompresii – všetky dáta potrebné pre dekompresiu sú preto obsiahnuté v komprimovanom súbore. Tento prístup funguje v princípe rovnako ako bežná slovníková metóda, využíva však aj informácie o pozícii zarovnaní. Preto predpokladáme, že dosiahne lepšie výsledky ako štandardné slovníkové metódy.

Pri výbere vhodnej metódy sa budeme najskôr zaoberať samostatným výberom vhodných kompresných metód pre každé z polí `CIGAR` a `SEQ`. Potom budeme testovať kombinácie týchto metód pri komprimovaní polí v nespracovanej podobe, s využitím referenčnej sekvencie alebo s využitím nami vytvorenej referenčnej sekvencie.

V tabuľke 8.6 uvádzame niektoré charakteristiky našich dátových súborov, ktoré súvisia s poliami `SEQ` a `CIGAR`.

8.3.1 Pole `CIGAR`

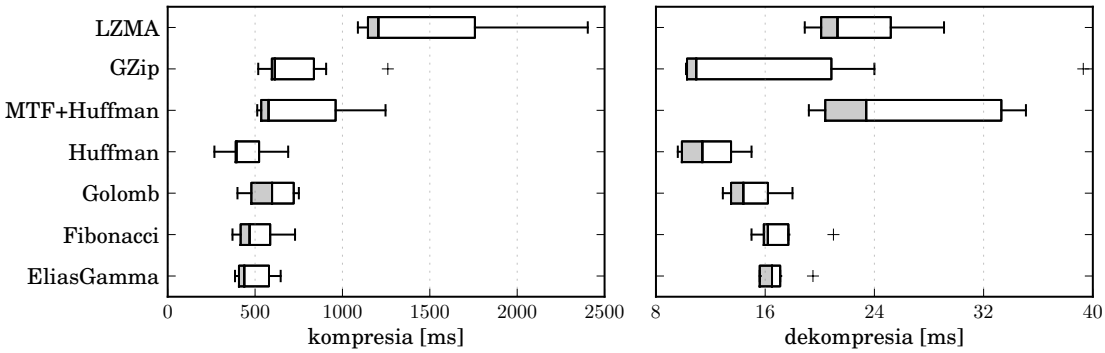
Pre kompresiu poľa `CIGAR` sme využili dva rozdielne prístupy – samostatná kompresia CIGAR operácií a dĺžok týchto operácií dvomi rôznymi metódami a kompresia CIGAR reťazca v podobe poľa znakov.

Súbor	Počet báz	Dĺžka segmentu	Pokrytie	Nezhody s referenciou			
				≠	I	D	S
HG00109	10 000 000	50.00	5.20	2.59%	0.01%	0.02%	0.00%
NA20589	11 056 003	55.28	3.11	2.59%	0.01%	0.01%	6.82%
KB1454	71 817 068	359.09	9.55	2.58%	0.42%	0.32%	0.30%
Manuel	18 000 000	90.00	10.11	0.28%	0.01%	0.01%	0.15%
Mouse	13 117 939	65.59	19.68	1.46%	0.03%	0.04%	0.00%
Ecoli	20 000 000	100.00	303.25	1.25%	0.33%	0.09%	0.00%

Tab. 8.6: Celkový počet osekvenovaných báz v testovacích súboroch, priemerný počet báz v každom osekvenovanom segmente (jednom zázname SAM súboru) a priemerné pokrytie pre jednotlivé dátové súbory. Nakoniec uvádzame, aká časť zo sekvencií v jednotlivých súboroch sa nezhoduje s referenčnou sekvenciou. Stĺpec ≠ sú rozdielne bázy. Ostatné stĺpce odpovedajú CIGAR operáciám – vloženie, vymazanie, soft clip

Kompresná metóda	Kompresný pomer						BAM %
	HG00109	NA20589	KB1454	Manuel	Mouse	Ecoli	
LZMA	**5.02%**	**5.29%**	**6.31%**	**1.52%**	**1.72%**	**2.95%**	0.23%
GZip	**5.37%**	**5.87%**	**7.25%**	**1.63%**	**1.86%**	**3.23%**	0.26%
MTF+Huffman	**4.54%**	**5.76%**	**7.75%**	**2.32%**	**2.21%**	**3.26%**	0.28%
Huffman	12.62%	12.31%	11.12%	12.65%	12.58%	10.12%	0.56%
Golomb	42.80%	37.08%	24.22%	41.41%	41.28%	31.13%	1.51%
Fibonacci	51.31%	44.00%	29.67%	49.63%	49.48%	37.25%	1.83%
EliasGamma	59.86%	50.37%	33.70%	57.85%	57.71%	43.30%	2.10%

Tab. 8.7: Účinnosť kompresie testovaných kompresných metód pre CIGAR operácie. Kompresný pomer vyjadruje pomer veľkosti komprimovaných dát k pôvodnej veľkosti poľa `CIGAR`



Obr. 8.4: Časová náročnosť testovaných metód pre CIGAR operácie

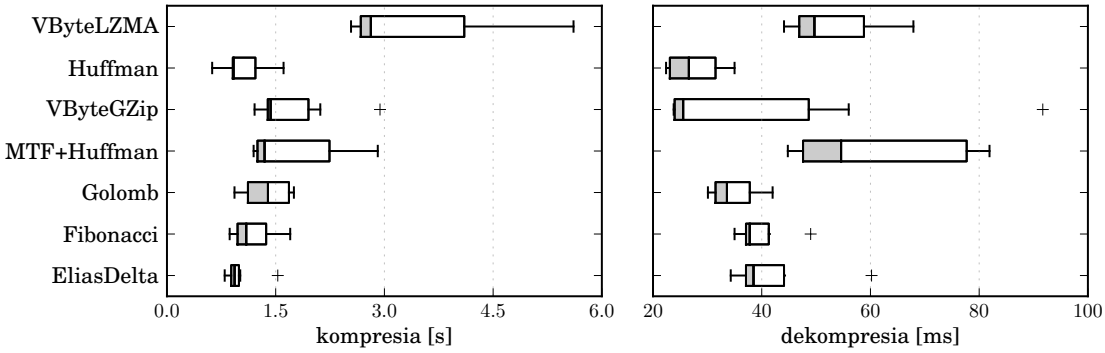
Operácie CIGAR operácie sú reprezentované jedným z 9 možných znakov uvedených v tabuľke v sekcii 5.3. V tabuľke 8.7 a v diagrame 8.4 sú uvedené výsledky testovaných metód. Najlepší kompresný pomer dosahuje metóda LZMA. Len nepatrne horšie výsledky ponúka rýchlejšia metóda GZip.

Dĺžky Dĺžky CIGAR operácií nadobúdajú pomerne malé hodnoty (1 – 200), ktoré sa často opakujú. V tabuľke 8.8 a v diagrame 8.5 sú uvedené výsledky testovaných metód.

V tabuľke 8.9 a v diagrame 8.6 sú už uvedené výsledky kombinácií najvhodnejších metód pre kompresiu operácií a dĺžok v reťazci CIGAR. V identifikátoroch testovaných metód sú lomítkom oddelené metódy pre kódovanie operácií a pre kódovanie dĺžok. Ak je uvedená len jedna metóda, jedná sa o kompresiu CIGAR reťazca ako poľa znakov. Najlepšie výsledky podáva metóda, ktorá používa pre kódovanie operácií metódu LZMA a pre kódovanie dĺžok metódu VByte+LZMA. Ak požadujeme vyššiu rýchlosť, tak by sme mohli uvažovať aj nad metódami v ďalších riadkoch a o metóde GZip, ktorá ponúka dostatočný kompresný pomer a vyššiu rýchlosť.

Kompresná metóda	**Kompresný pomer**						**BAM %**
	HG00109	**NA20589**	**KB1454**	**Manuel**	**Mouse**	**Ecoli**	
VByteLZMA	**1.18%**	**12.50%**	**27.07%**	**1.78%**	**7.14%**	**6.52%**	0.80%
Huffman	4.57%	**13.21%**	**27.83%**	5.39%	7.89%	8.39%	0.87%
VByteGZip	**1.35%**	**12.97%**	**30.16%**	**2.08%**	**7.48%**	**7.09%**	0.89%
MTF+Huffman	**2.77%**	18.76%	32.84%	**4.61%**	9.29%	10.30%	1.03%
Golomb	27.51%	29.32%	34.79%	33.39%	31.15%	27.68%	1.57%
Fibonacci	35.12%	36.47%	30.91%	45.38%	40.89%	33.52%	1.67%
EliasDelta	39.03%	39.87%	36.06%	45.51%	43.19%	34.24%	1.85%
EliasGamma	42.86%	44.02%	35.99%	53.62%	49.23%	39.12%	1.98%

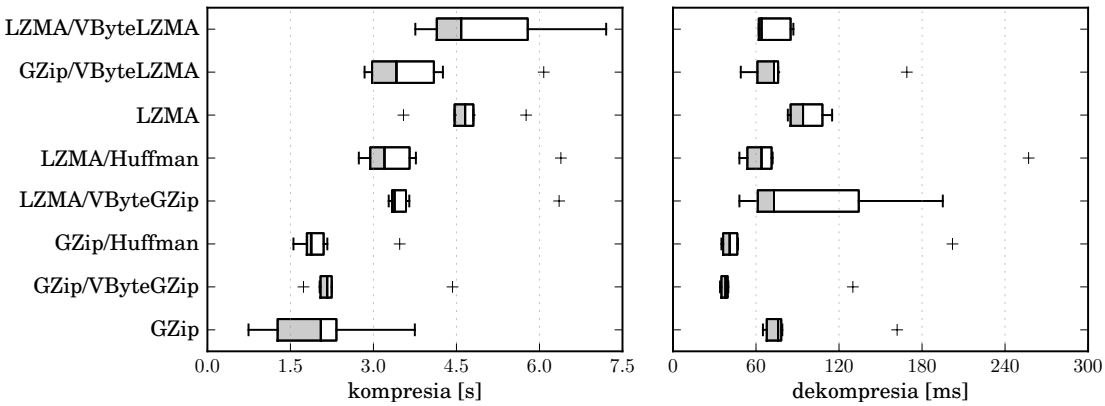
Tab. 8.8: Účinnosť kompresie testovaných metód pre dĺžky CIGAR operácií. Kompresný pomer vyjadruje pomer veľkosti komprimovaných dát k pôvodnej veľkosti poľa `CIGAR`



Obr. 8.5: Časová náročnosť testovaných metód pre dĺžky CIGAR operácií

Kompresná metóda	Kompresný pomer						BAM %
	HG00109	NA20589	KB1454	Manuel	Mouse	Ecoli	
LZMA/VByteLZMA	**6.20%**	**17.79%**	**33.38%**	**3.30%**	**8.86%**	**9.47%**	1.04%
GZip/VByteLZMA	**6.55%**	18.37%	**34.32%**	**3.40%**	**9.00%**	**9.75%**	1.07%
LZMA	6.79%	**16.59%**	34.63%	3.46%	10.25%	**9.53%**	1.07%
LZMA/Huffman	9.58%	18.50%	**34.14%**	6.91%	9.61%	11.34%	1.10%
LZMA/VByteGZip	**6.37%**	**18.26%**	36.47%	3.61%	**9.20%**	10.04%	1.12%
GZip/Huffman	9.94%	19.08%	35.08%	7.02%	9.75%	11.62%	1.13%
GZip/VByteGZip	6.72%	18.84%	37.41%	3.71%	9.34%	10.32%	1.15%
GZip	7.70%	18.73%	37.96%	**3.35%**	11.91%	10.02%	1.18%

Tab. 8.9: Účinnosť kompresie testovaných metód pre pole `CIGAR`



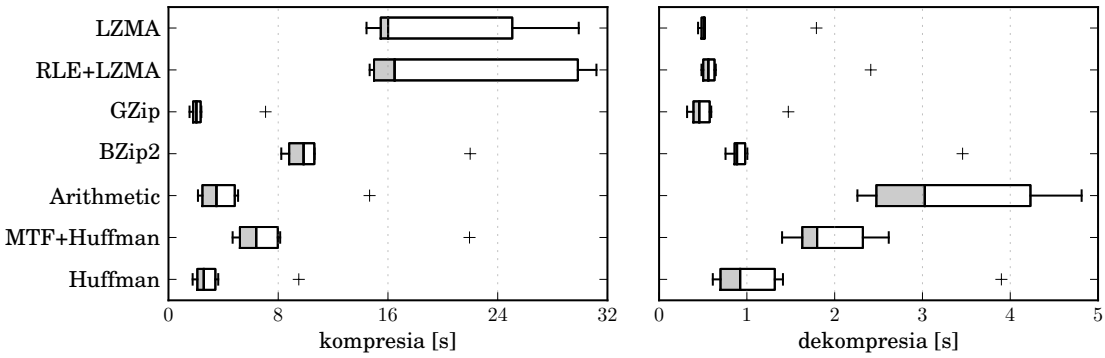
Obr. 8.6: Časová náročnosť testovaných metód pre pole `CIGAR`

8.3.2 Pole `SEQ`

Pole `SEQ` obsahuje postupnosť báz získaných pri sekvenovaní. Môže obsahovať znaky `A`, `C`, `G`, `T` alebo `N`. Výsledky testovaných metód uvádzame v tabuľke 8.10 a v diagrame 8.7. Najlepší kompresný pomer dosahuje metóda LZMA. Ak však požadujeme vyššiu rýchlosť, je vhodnejšie zvoliť metódu GZip.

Kompresná metóda	Kompresný pomer						BAM %
	HG00109	NA20589	KB1454	Manuel	Mouse	Ecoli	
LZMA	**15.73%**	**14.91%**	**5.88%**	**5.13%**	**6.75%**	**2.95%**	7.11%
RLE+LZMA	**16.26%**	**15.53%**	**6.02%**	**5.39%**	**7.18%**	**3.37%**	7.41%
GZip	**18.54%**	**17.29%**	**6.79%**	**6.78%**	**8.69%**	**4.43%**	8.59%
BZip2	21.36%	22.03%	10.50%	9.17%	11.82%	5.63%	11.88%
Arithmetic	27.85%	26.63%	25.26%	26.04%	26.36%	26.11%	26.90%
MTF+Huffman	30.19%	28.75%	26.78%	28.06%	28.24%	28.39%	28.80%
Huffman	30.69%	30.60%	29.77%	30.30%	29.99%	30.65%	31.32%

Tab. 8.10: Účinnosť kompresie testovaných metód pre pole `SEQ`



Obr. 8.7: Časová náročnosť testovaných metód pre pole `SEQ`

8.3.3 Kombinácia metód pre kompresiu polí `CIGAR` a `SEQ`

Kombinácie metód, ktoré poskytovali dobré výsledky pre polia `CIGAR` a `SEQ` sme otestovali pri kompresii týchto polí v nespracovanej podobe, s využitím dostupnej referenčnej sekvencie a s využitím nami vytvorenej referenčnej sekvencie. Výsledky týchto testov sú uvedené v tabuľke 8.11 a v diagrame 8.8. Identifikátory pred lomítkom udávajú spôsob kompresie:

- REF – využitie referenčnej sekvencie,
- SELF – využitie nami vytvorenej referenčnej sekvencie a
- RAW – kompresia dát v nespracovanej podobe.

Metódy pre kompresiu polí `SEQ` a `CIGAR` sú uvedené za lomítkom, oddelené znakom '+' (v poradí `SEQ`, `CIGAR`).

Najhoršie výsledky sme dosiahli pri kompresii polí v nespracovanej podobe. Trochu lepšiu úroveň kompresie pre väčšinu testovacích súborov sme dosiahli pri použití nami vytvorenej referenčnej sekvencie. Tento prístup sa navyše pozitívne podpísal na čase kompresie pri využití metódy LZMA. Ako sa dalo predpokladať, najlepšie výsledky sme dosiahli pri využití referenčnej sekvencie. Využitie referenčnej sekvencie umožnilo dosiahnutie $2-4$ krát lepšieho kompresného pomeru oproti kompresii polí v nespracovanej podobe. Veľký rozdiel dosiahnutých kompresných pomerov pri využití referenčnej sekvencie a pri využití vytvorenej referenčnej sekvencie sme očakávali z nasledujúcich dôvodov:

- Pri vytvorenej referenčnej sekvencii musíme uložiť prvý výskyt bázy na danej pozícii. Pri reálnej referenčnej sekvencii tieto dáta ukladať nemusíme.
- Našu referenčnú sekvenciu môžeme vytvárať len z už spracovaných dát v aktuálnom bloku.
- Niektoré z testovacích súborov majú pomerne malé pokrytie a obsahujú len málo sa prekrývajúce segmenty. Toto nevadí pri využití reálnej referenčnej sekvencie. Pri vytvorenej referenčnej sekvencii je však potrebné ukladať všetky dáta okrem tých, ktoré sa zhodujú v týchto prekrývajúcich sa segmentoch.

Pre kompresiu polí `CIGAR` a `SEQ` je podľa výsledkov našich testov najvhodnejšie kódovať len rozdiely oproti známej referenčnej sekvencii s využitím kompresie metódou LZMA pre obe polia. Ak požadujeme rýchlejšiu kompresiu, tak je vhodné pre obe polia zvoliť metódu GZip. V prípade, že nemáme k dispozícii referenčnú sekvenciu je vhodné vytvoriť si vlastnú referenčnú sekvenciu pri kompresii. V tomto prípade je taktiež najvhodnejšie využiť pre kódovanie výstupu metódy LZMA (vyššia úroveň kompresie) alebo GZip (vyššia rýchlosť) pre obe polia.

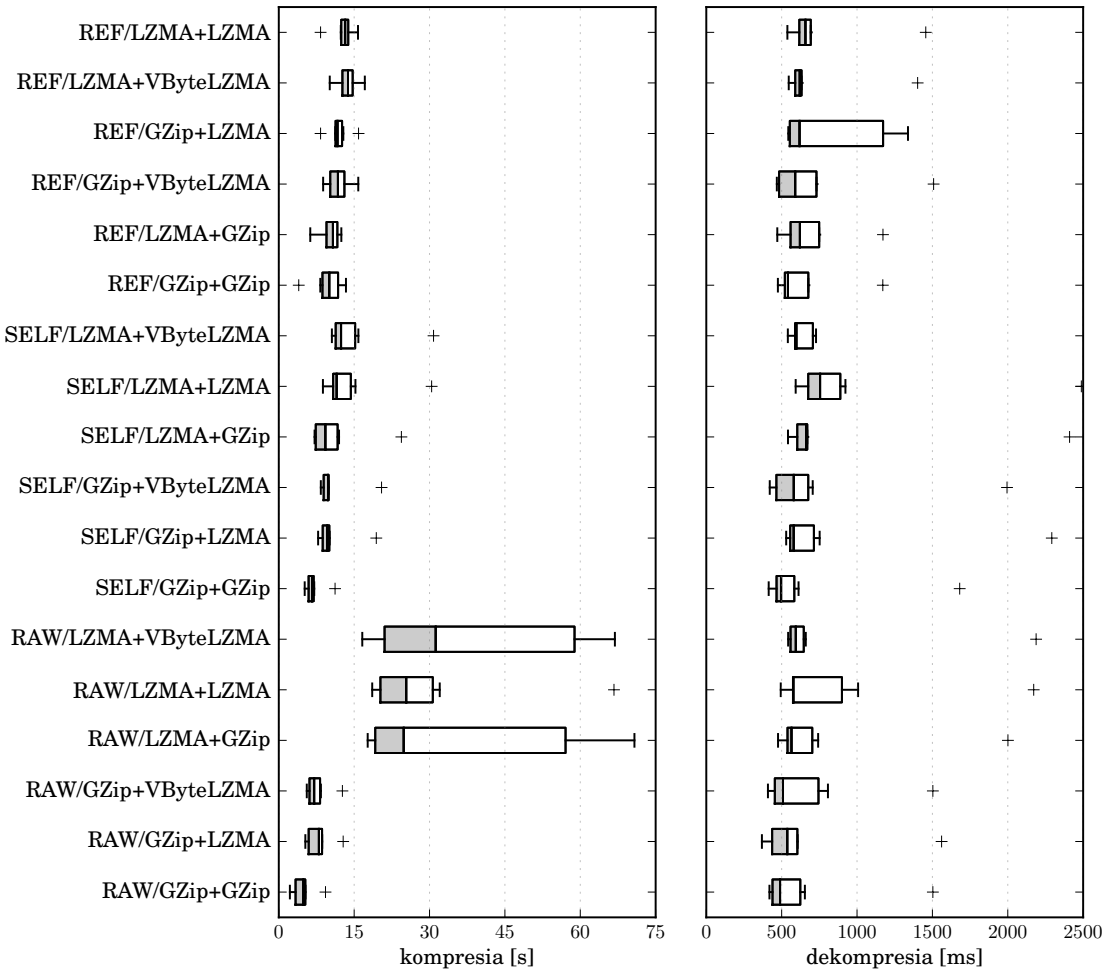
Kompresná metóda	**Kompresný pomer**						**BAM**
	HG00109	**NA20589**	**KB1454**	**Manuel**	**Mouse**	**Ecoli**	**%**
REF/LZMA+LZMA	**7.02%**	**4.16%**	2.11%	**0.92%**	**3.28%**	**2.41%**	2.85%
REF/LZMA+VByteLZMA	7.12%	4.33%	**2.06%**	0.98%	3.38%	2.51%	2.88%
REF/GZip+LZMA	7.33%	4.30%	2.13%	0.92%	3.32%	2.43%	2.90%
REF/GZip+VByteLZMA	7.43%	4.48%	2.08%	0.99%	3.43%	2.53%	2.93%
REF/LZMA+GZip	7.28%	4.37%	2.30%	0.99%	3.55%	2.66%	3.06%
REF/GZip+GZip	7.59%	4.52%	2.31%	1.00%	3.60%	2.67%	3.11%
SELF/LZMA+VByteLZMA	13.21%	**12.49%**	**5.02%**	4.28%	5.88%	3.91%	6.54%
SELF/LZMA+LZMA	**13.17%**	12.50%	5.15%	**4.21%**	**5.81%**	**3.79%**	6.58%
SELF/LZMA+GZip	13.55%	12.80%	5.38%	4.37%	6.16%	4.17%	6.87%
SELF/GZip+VByteLZMA	13.88%	13.34%	5.33%	4.48%	6.03%	3.95%	6.88%
SELF/GZip+LZMA	13.83%	13.35%	5.46%	4.41%	5.96%	3.83%	6.92%
SELF/GZip+GZip	14.22%	13.65%	5.69%	4.58%	6.31%	4.21%	7.21%
RAW/LZMA+VByteLZMA	**15.24%**	15.09%	**7.10%**	**5.07%**	**6.84%**	**3.24%**	8.15%
RAW/LZMA+LZMA	15.27%	**15.02%**	7.16%	5.08%	6.91%	3.24%	8.19%
RAW/LZMA+GZip	15.32%	15.15%	7.31%	5.08%	6.98%	3.26%	8.29%
RAW/GZip+VByteLZMA	17.91%	17.32%	7.97%	6.66%	8.70%	4.65%	9.63%
RAW/GZip+LZMA	17.94%	17.24%	8.03%	6.67%	8.76%	4.66%	9.66%
RAW/GZip+GZip	17.99%	17.38%	8.17%	6.66%	8.83%	4.68%	9.77%

Tab. 8.11: Účinnosť kompresie testovaných metód pre polia `SEQ` a `CIGAR`

8.4 Pole `QUAL`

Pole `QUAL` obsahuje postupnosť kvalít osekvenovaných báz. Dĺžka tohto poľa je rovnaká ako dĺžka poľa `SEQ`. Štandardne sú hodnoty kvalít uvedené v skórovaní PHRED (viď sekcia 2.2.5) s rozsahom hodnôt $[0, 93]$. Jednotlivé kvality sú v SAM súbore uložené ako tlačiteľné ASCII znaky s rovnakým spôsobom kódovania ako pri formáte FASTQ (viď sekcia 2.3.2).

Ako sme už spomínali v úvode kapitoly, pole `QUAL` prispieva z veľkej časti k veľkosti nami testovaných súborov a obsahuje veľmi ťažko komprimovateľné dáta. Z údajov v tabuľke 8.12 vidíme, že v testovacích súboroch neobsahuje pole `QUAL` hodnoty z celého rozsahu $[0, 93]$. Tieto hodnoty navyše nie sú rovnomerne rozložené. Nie je preto vhodné využívať pre ich kompresiu kódovania čísel, ktoré nevedia využiť rozdelenie týchto hodnôt. Z kódovaní čísel preto zaradíme k testovaniu len entropické kódy – Huffmanovo a Aritmetické kódovanie. Z tabuľky je taktiež vidieť, že súbory obsahujú pomerne dlhé behy hodnôt. Preto k testovaným metódam zaradíme aj behové kódovanie.



Obr. 8.8: Časová náročnosť testovaných metód pre polia `SEQ` a `CIGAR`

V tabuľke 8.13 a v diagrame 8.9 sú uvedené výsledky testovaných metód. Najlepšie výsledky dosahovali metódy LZMA a RLE+LZMA. Ak by sme požadovali väčšiu rýchlosť, je vhodné zvoliť metódu GZip. Metóda RLE+BZip2 nie je podľa nášho názoru vhodná, pretože má pomalú dekompresiu. Metóda GZip bola spomedzi testovaných metód najrýchlejšia. Preto je jej voľba vhodná v prípade, že chceme minimalizovať čas kompresie a zachovať pomerne dobrý kompresný pomer.

Ďalšie zlepšenie úrovne kompresie kvalít je veľmi problematické. V sekcii 8.9 uvedieme, akým spôsobom je možné zlepšiť kompresný pomer pri stratovej kompresii kvalít.

8.5 Polia `RNEXT`, `PNEXT` a `TLEN`

Polia `RNEXT`, `PNEXT` a `TLEN` obsahujú informácie o ďalšom segmente v šablóne. V našich testovacích súboroch sa konkrétne jedná o informácie o ďalšom segmente v páre pri párovom čítaní. Ak vstupný súbor neobsahuje párové čítania, tak nie je potrebné ukladať obsah týchto polí (obsahujú hodnoty '`*`' alebo '`0`'). V testovacích súboroch neobsahujú párové čítania súbory `KB1454` a `Ecoli`.

Súbor	Hodnoty			Behy hodnôt			
	Rozsah	Priemer	SD	Rozsah	Medián	Priemer	SD
HG00109	[0, 63]	39.59	8.38	[1, 47]	22.5	1.14	0.43
NA20589	[0, 50]	30.89	10.28	[1, 432]	101.5	1.30	2.77
KB1454	[0, 72]	30.32	8.90	[1, 207]	85.5	2.50	3.85
Manuel	[1, 43]	34.77	4.74	[1, 25]	11	1.14	0.42
Mouse	[0, 40]	27.77	6.66	[1, 81]	34.5	1.48	1.58
Ecoli	[2, 41]	32.52	10.85	[1, 195]	82.5	1.64	3.07

Tab. 8.12: Štatistiky hodnôt a dĺžok behov týchto hodnôt v poli `QUAL`

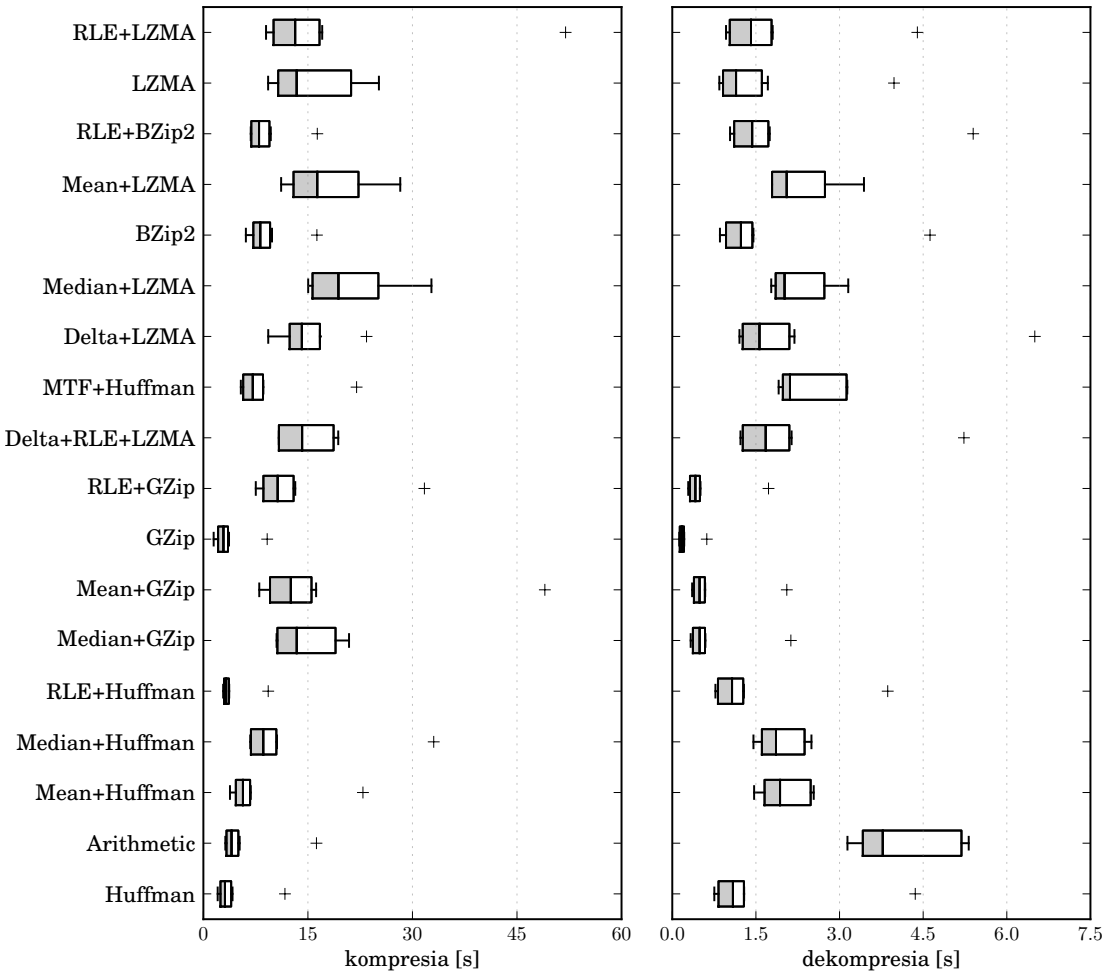
Kompresná metóda	Kompresný pomer						BAM %
	HG00109	NA20589	KB1454	Manuel	Mouse	Ecoli	
RLE+LZMA	**50.69%**	**47.08%**	**34.36%**	**47.43%**	**43.85%**	**38.12%**	41.081%
LZMA	**50.57%**	**47.06%**	34.59%	**47.24%**	**43.34%**	**37.87%**	41.083%
RLE+BZip2	51.82%	47.97%	**33.74%**	50.04%	44.62%	38.94%	41.445%
Mean+LZMA	**50.72%**	**47.77%**	35.10%	**47.88%**	**43.66%**	**38.48%**	41.617%
BZip2	51.84%	48.09%	**34.03%**	50.08%	44.68%	39.05%	41.631%
Median+LZMA	51.40%	47.79%	35.15%	48.35%	43.92%	38.78%	41.819%
Delta+LZMA	54.12%	49.57%	34.69%	50.93%	46.04%	42.21%	42.954%
MTF+Huffman	57.70%	50.07%	34.78%	52.47%	45.55%	42.74%	43.531%
Delta+RLE+LZMA	56.38%	51.12%	35.93%	53.14%	47.44%	43.11%	44.432%
RLE+GZip	55.21%	51.12%	37.63%	53.04%	47.89%	41.74%	45.060%
GZip	55.10%	51.28%	38.84%	52.94%	48.03%	42.64%	45.827%
Mean+GZip	55.13%	51.30%	38.85%	52.96%	48.06%	42.66%	45.845%
Median+GZip	55.12%	51.29%	38.85%	52.96%	48.06%	42.66%	45.846%
RLE+Huffman	55.56%	51.18%	45.12%	50.72%	49.29%	40.92%	48.694%
Median+Huffman	55.26%	53.74%	51.59%	50.27%	50.24%	42.07%	52.430%
Mean+Huffman	55.26%	53.74%	51.59%	50.27%	50.24%	42.07%	52.430%
Arithmetic	55.16%	54.65%	55.73%	50.07%	51.18%	43.22%	54.874%
Huffman	55.49%	55.00%	56.31%	50.39%	51.66%	44.15%	55.448%

Tab. 8.13: Účinnosť kompresie testovaných metód pre pole `QUAL`

8.5.1 Pole `RNEXT`

Pole `RNEXT` obsahuje identifikátor referenčnej sekvencie ďalšieho segmentu v šablóne alebo znak '=', ak je jeho referenčná sekvencia rovnaká ako referenčná sekvencia uvedená v poli `RNAME`. V testovacích súboroch toto pole väčšinou obsahuje práve znak '='. Pri kompresii si preto udržiavame bitové pole, ktoré pre každý záznam obsahuje príznak toho, či pole `RNEXT` obsahuje znak '='. Ak pole obsahuje inú hodnotu (identifikátor referenčnej sekvencie), tak sa táto hodnota uloží do samostatného prúdu.

V tabuľke 8.14 a v diagrame 8.10 uvádzame výsledky testov kompresných metód pre vyššie spomínané pole bitových príznakov. Najlepší priemerný kompresný pomer a dobrú časovú náročnosť podávala metóda RLE+Huffman. Túto metódu budeme používať aj v ďalších experimentoch pre kompresiu polí bitových príznakov.



Obr. 8.9: Časová náročnosť testovaných metód pre pole `QUAL`

Výsledky metód, ktoré sme testovali pre uloženie hodnôt poľa `RNEXT`, ktoré nie sú rovné '=', sú uvedené v tabuľke 8.15 a v diagrame 8.11. V hodnotách je zahrnuté aj pole bitových príznakov, ktoré bolo skomprimované metódou RLE+Huffman. Vidíme, že testované metódy podávali veľmi podobné výsledky. Najvhodnejšia je metóda Rice, ktorá dosahuje dobrý kompresný pomer aj rýchlosť.

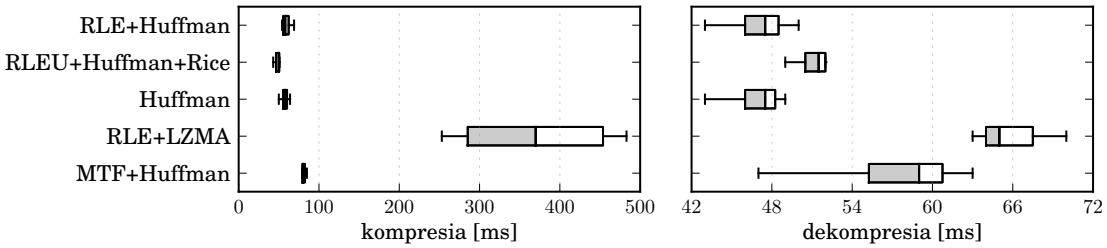
8.5.2 Polia `PNEXT` a `TLEN`

Pole `PNEXT` obsahuje pozíciu ďalšieho segmentu v šablóne. Aby sme zlepšili účinnosť kompresie, ukladáme len rozdiel hodnoty tohto poľa a hodnoty poľa `POS`. Pole `TLEN` obsahuje relatívne malé čísla so znamienkom.

V tabuľkách 8.16, 8.17 a diagramoch 8.12, 8.13 uvádzame výsledky testovaných metód pre polia `PNEXT` a `TLEN`. Závery sú pre obe polia rovnaké. Najlepšie výsledky podávajú metódy VByteLZMA a VByteGZip. Vhodnejšou metódou však je VByteGZip z dôvodu menšej časovej náročnosti.

Kompresná metóda	Kompresný pomer				BAM %
	HG00109	NA20589	Manuel	Mouse	
RLE+Huffman	**39.68%**	**26.63%**	**4.33%**	**15.51%**	0.027%
RLEU+Huffman+Rice	**45.99%**	**27.16%**	**3.92%**	**15.95%**	0.029%
Huffman	**35.74%**	**24.28%**	14.32%	19.82%	0.030%
RLE+LZMA	49.82%	32.22%	**5.23%**	**17.44%**	0.033%
MTF+Huffman	88.02%	49.80%	7.04%	30.87%	0.056%

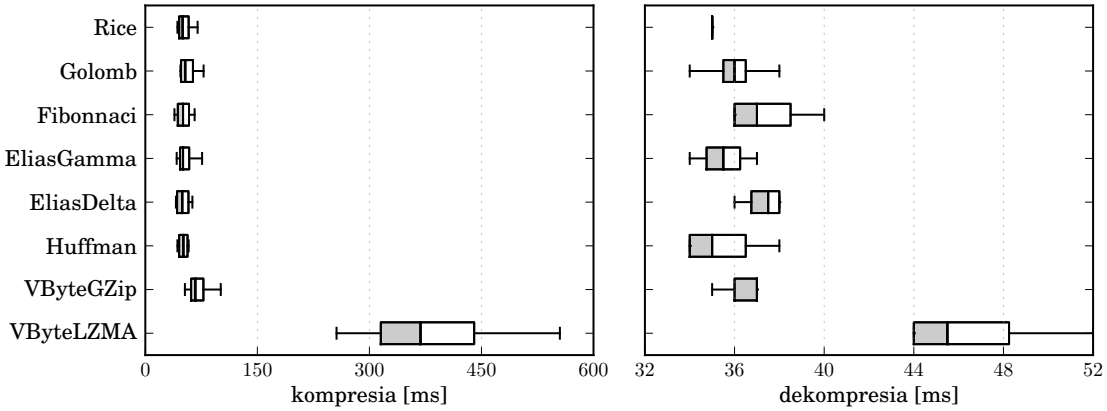
Tab. 8.14: Účinnosť kompresie testovaných metód pre pole bitových príznakov



Obr. 8.10: Časová náročnosť testovaných metód pre kompresiu poľa bitových príznakov

Kompresná metóda	Kompresný pomer				BAM %
	HG00109	NA20589	Manuel	Mouse	
Rice	**44.50%**	**31.75%**	4.94%	**19.41%**	0.039%
Golomb	**44.77%**	**32.14%**	5.42%	**19.91%**	0.040%
Fibonnaci	**46.15%**	**31.79%**	**4.78%**	**20.38%**	0.040%
EliasGamma	47.25%	32.40%	**4.84%**	21.16%	0.041%
EliasDelta	48.75%	32.96%	**4.89%**	21.55%	0.042%
Huffman	47.44%	34.68%	5.79%	21.67%	0.042%
VByteGZip	48.56%	36.06%	8.00%	23.44%	0.045%
VByteLZMA	49.68%	36.33%	7.18%	22.84%	0.045%

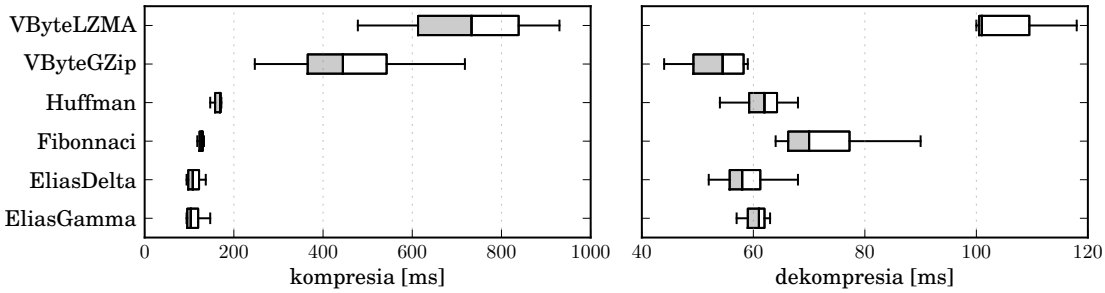
Tab. 8.15: Účinnosť kompresie testovaných metód pre pole `RNEXT`



Obr. 8.11: Časová náročnosť testovaných metód pre pole `RNEXT`

Kompresná metóda	Kompresný pomer				BAM %
	HG00109	NA20589	Manuel	Mouse	
VByteLZMA	**57.26%**	**67.74%**	**75.17%**	**65.88%**	1.38%
VByteGZip	**63.37%**	**77.70%**	**83.76%**	**76.35%**	1.56%
Huffman	**89.03%**	**81.37%**	**86.14%**	**82.36%**	1.80%
Fibonnaci	112.57%	86.16%	91.82%	87.04%	2.03%
EliasDelta	108.67%	93.82%	97.82%	92.06%	2.09%
EliasGamma	146.52%	107.31%	111.51%	107.51%	2.56%

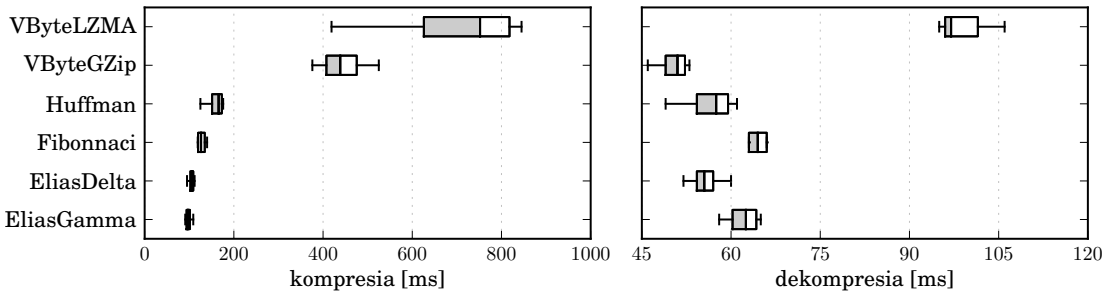
Tab. 8.16: Účinnosť kompresie testovaných metód pre pole `PNEXT`



Obr. 8.12: Časová náročnosť testovaných metód pre pole `PNEXT`

Kompresná metóda	Kompresný pomer				BAM %
	HG00109	NA20589	Manuel	Mouse	
VByteLZMA	**59.59%**	**63.87%**	**57.85%**	**64.32%**	1.16%
VByteGZip	**68.93%**	**75.38%**	**68.51%**	**75.67%**	1.37%
Huffman	**74.44%**	**77.00%**	**62.60%**	**80.07%**	1.39%
Fibonnaci	92.49%	82.43%	78.63%	83.58%	1.59%
EliasDelta	92.59%	91.37%	87.84%	93.34%	1.73%
EliasGamma	111.94%	100.22%	96.37%	103.36%	1.94%

Tab. 8.17: Účinnosť kompresie testovaných metód pre pole `TLEN`



Obr. 8.13: Časová náročnosť testovaných metód pre pole `TLEN`

8.6 Ďalšie povinné polia

V tejto sekcii popíšeme výber kompresných metód pre ostatné povinné polia SAM súboru – `QNAME` (8.6.1), `FLAG` (8.6.2), `RNAME` (8.6.3), `POS` (8.6.4) a `MAPQ` (8.6.5).

8.6.1 Pole `QNAME`

Pole `QNAME` obsahuje reťazce, v ktorých sa často opakujú rovnaké podreťazce. Preto bude asi najvhodnejšie použiť k jeho kompresii slovníkovú metódu.

Príklad 8.1. Obsah poľa `QNAME` sa u jednotlivých dátových súborov líši. Príklad obsahu tohto poľa pre testovacie súbory (každý riadok predstavuje obsah poľa v jednom zázname):

HG00109 a NA20589:

```
ERR013029.13666541
ERR013029.7592908
ERR003707.4597433
ERR003818.1749052
ERR015494.378041
ERR003881.1300999
ERR003707.838033
ERR003881.1300999
```

KB1454:

```
FQ6A6YF02INKCM
FTD8QAW01CP9BO
FWX4VV301C5XW7
FQ6A6YF02FHT2U
FRHR2BV01DEE47
F2JW3UG01BLHAX
FSVRH9T02FND8Y
FTD8VL302FZWPK
```

Mouse:

```
IL37_2813:5:50:161:18
IL37_2813:5:50:161:18
IL7_2847:7:68:240:534
IL7_2847:7:68:240:534
IL8_2737:7:8:358:1328
IL7_2847:8:23:1426:1806
IL7_2847:3:84:799:1732
IL37_2813:3:16:141:1664
```

Ecoli:

```
EAS20_8_6_44_371_993/1
EAS20_8_6_45_68_1692/1
EAS20_8_6_45_641_1280/1
EAS20_8_6_46_198_1670/1
EAS20_8_6_47_11_1746/1
EAS20_8_6_47_1184_887/1
EAS20_8_6_48_724_1520/1
EAS20_8_6_48_1500_1789/1
```

Manuel:

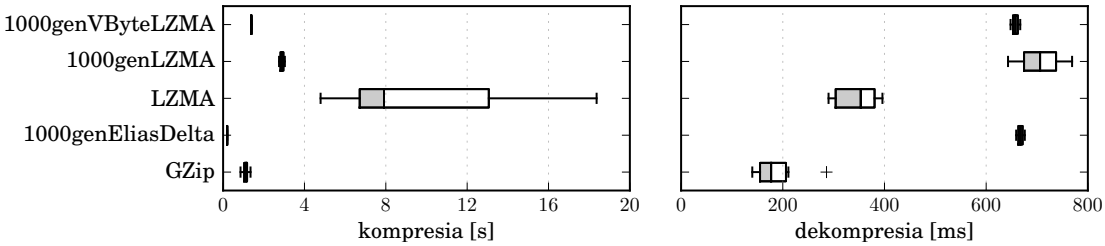
```
FCB021RACXX:4:1102:20878:75397#CAGATCAT
FCB021RACXX:4:1102:20878:75397#CAGATCAT
FCB021RACXX:4:1203:17908:115699#CAGATCAT
FCD044UACXX:4:1101:7427:145757#CAGATCAT
FCB021RACXX:4:1301:2439:27642#CAGATCAT
FCB021RACXX:4:1106:16412:5384#CAGATCAT
FCB021RACXX:4:1202:16305:73742#CAGATCAT
FCD044UACXX:4:2107:16301:168706#CAGATCAT
```

V dátových súboroch pochádzajúcich z 1000 genomes project obsahuje pole `QNAME` identifikátor skupiny segmentov, ktorý sa zhoduje s obsahom nepovinného poľa `RG` a pomerne veľké číslo. Identifikátory skupiny segmentov sú uvedené v hlavičke SAM súboru. Testované súbory obsahujú nepovinné pole `RG`. Preto nie je potrebné túto časť ukladať do komprimovaného súboru. Metódy, ktoré využívajú tento

prístup označujeme identifikátorom `1000genX`, kde X je identifikátor kompresnej metódy použitej na kódovanie čísla, ktoré zostane po odstránení časti zhodnej s poľom `RG`. Formátom ostatných dátových súborov sme sa nezaoberali. Tento spôsob kompresie sme testovali len kvôli tomu, aby sa mohol čitateľ rozhodnúť, či je preňho výhodné vytvárať špeciálnu kompresnú metódu pre ním používané dáta, alebo využiť univerzálnu metódu.

Kompresná metóda	**Kompresný pomer**						**BAM**
	HG00109	**NA20589**	**KB1454**	**Manuel**	**Mouse**	**Ecoli**	**%**
1000genVByteLZMA	**14.72%**	**13.84%**	*	*	*	*	2.68%
1000genLZMA	16.29%	14.80%	*	*	*	*	2.93%
LZMA	17.03%	19.09%	**34.56%**	**12.18%**	**20.78%**	**22.99%**	3.73%
1000genEliasDelta	23.56%	22.07%	*	*	*	*	4.29%
GZip	21.52%	23.22%	40.32%	15.12%	24.11%	26.34%	4.45%

Tab. 8.18: Účinnosť kompresie testovaných metód pre pole `QNAME`



Obr. 8.14: Časová náročnosť testovaných metód pre pole `QNAME`

Z tabuľky 8.18 vidíme, že u súborov z 1000 genomes project sme boli schopní pomocou vyššie popísanej metódy dosiahnuť lepší kompresný pomer oproti štandardným slovníkovým metódam. Ak však chceme zvoliť kompresnú metódu, ktorá bude použiteľná pre všetky vstupné súbory, tak je lepšie zvoliť štandardné slovníkové metódy LZMA a GZip.

8.6.2 Pole `FLAG`

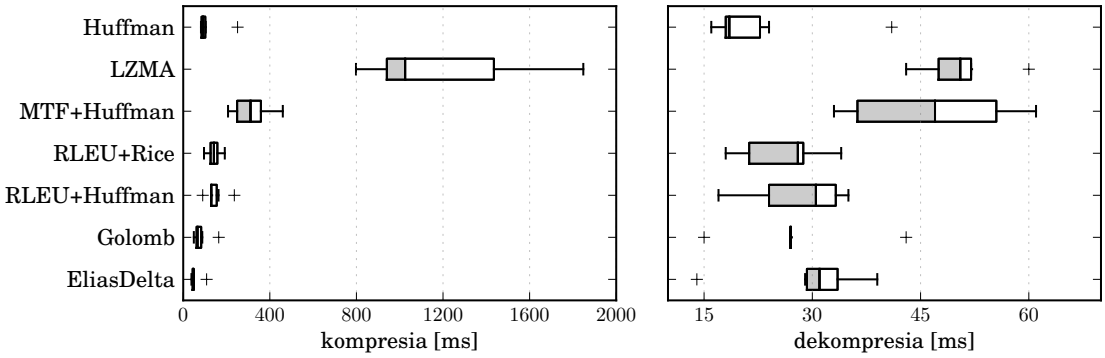
Pole `FLAG` obsahuje niekoľko často sa opakujúcich hodnôt z oboru hodnôt $[0, 2047]$. Preto môže byť vhodné využiť entropické kódovanie čísel.

V tabuľke 8.19 a v diagrame 8.15 sú uvedené výsledky testovaných metód. Najlepší kompresný pomer dosiahlo Huffmanovo kódovanie, ktoré dokázalo využiť vhodné rozdelenie vstupných hodnôt. Taktiež je vidieť, že Golombove alebo Eliasove delta kódy, ktoré nevyužívajú znalosť rozdelenia výskytu hodnôt, nedosahujú v porovnaní s ostatnými testovanými metódami dobré výsledky.

Najvhodnejšou testovanou metódou pre kompresiu poľa `FLAG`, ktorá je vhodná pre ľubovoľný dátový súbor, je Huffmanovo kódovanie. Táto metóda dosahovala najlepšie kompresné pomery pri väčšine testovaných súborov. Navyše bola aj dostatočne rýchla v porovnaní s ostatnými testovanými metódami. V prípade súborov `KB1454` a `Ecoli` podávala lepšie výsledky metóda MTF+Huffman. Dôvod vidíme v tom, že v týchto súboroch obsahuje pole `FLAG` len 2 hodnoty, ktoré sa často opakujú.

Kompresná metóda	Kompresný pomer						BAM %
	HG00109	NA20589	KB1454	Manuel	Mouse	Ecoli	
Huffman	**18.16%**	**13.53%**	**12.38%**	**13.06%**	**13.09%**	**12.52%**	0.27%
LZMA	**18.39%**	**16.53%**	**11.77%**	**15.86%**	**15.17%**	**12.50%**	0.30%
MTF+Huffman	**23.31%**	**18.79%**	**9.10%**	**17.84%**	**17.30%**	**9.90%**	0.33%
RLEU+Rice	24.51%	19.65%	13.31%	18.65%	18.64%	16.07%	0.37%
RLEU+Huffman	26.59%	22.84%	16.61%	22.01%	22.03%	19.76%	0.43%
Golomb	40.97%	43.13%	39.84%	43.16%	42.35%	39.50%	0.81%
EliasDelta	57.12%	61.74%	41.86%	61.93%	62.17%	41.73%	1.09%

Tab. 8.19: Účinnosť kompresie testovaných metód pre pole `FLAG`



Obr. 8.15: Časová náročnosť testovaných metód pre pole `FLAG`

8.6.3 Pole `RNAME`

Pole `RNAME` obsahuje identifikátor referenčnej sekvencie ku ktorej je zarovnaný aktuálny záznam. Hlavička SAM súboru obsahuje zoznam použitých referenčných sekvencií[3]. Preto budeme pri kompresii ukladať len index referenčnej sekvencie v tomto zozname. Rovnakým spôsobom je pole `RNAME` reprezentované aj vo formáte BAM.

Pole `RNAME` obsahuje dlhé behy rovnakých hodnôt. Preto je vhodné pre jeho kompresiu využiť behové kódovanie.

Kompresná metóda	Kompresný pomer						BAM %
	HG00109	NA20589	KB1454	Manuel	Mouse	Ecoli	
RLEU+Fibonaccii	**0.040%**	**0.040%**	**0.016%**	**0.048%**	**0.060%**	**0.002%**	0.001%
RLEU+EliasDelta	**0.040%**	**0.040%**	**0.016%**	**0.050%**	**0.080%**	**0.003%**	0.001%
RLEU+EliasGamma	**0.050%**	**0.050%**	**0.020%**	**0.059%**	**0.080%**	**0.003%**	0.001%
RLEU+Golomb	0.090%	0.090%	0.036%	0.113%	0.180%	0.006%	0.002%
RLEU+Huffman	0.090%	0.090%	0.036%	0.113%	0.180%	0.006%	0.002%
LZMA	0.730%	0.730%	0.292%	0.913%	1.460%	0.052%	0.013%

Tab. 8.20: Účinnosť kompresie testovaných metód pre pole `RNAME`

V tabuľke 8.20 sú uvedené výsledky testovaných metód. Časovú náročnosť týchto metód neuvádzame, pretože všetky testované metódy boli v tomto smere vyrov-

[3]V nami testovaných súboroch sa jedná o chromozómy

nané. Je vidieť, že dosiahnuté kompresné pomery týchto metód sú takisto veľmi vyrovnané. Najlepší kompresný pomer dosiahla metóda RLEU+Fibonaccii.

8.6.4 Pole `POS`

Predpokladáme, že vstupné súbory sú usporiadané podľa referenčnej sekvencie a pozície zarovnania k tejto sekvencii. Preto je možné využiť pre kódovanie poľa `POS`, ktoré obsahuje tieto pozície, delta kódovanie.

Výsledky testovaných metód sú v tabuľke 8.21 a v diagrame 8.16. Vidíme, že u súborov s vysokým pokrytím `Mouse` a `Ecoli` sme boli schopní dosiahnuť významne lepšie kompresné pomery ako pri ostatných testovacích súboroch.

Najlepší priemerný kompresný pomer sme dosiahli pri použití metódy VByteLZMA pre kódovanie rozdielov hodnôt v delta kódovaní. Časová náročnosť tejto metódy je však vďaka použitiu metódy LZMA neprimerane vysoká pre pole, ktoré zásadne neprispieva k veľkosti SAM súboru. Preto je vhodnejšie použiť niektorú z rýchlejších metód. Dobré výsledky sme dosiahli aj s využitím Huffmanovho kódovania, ktoré dosahovalo výborné časy kompresie a dekompresie. Preto sme zvolili túto metódu pre kódovanie delta hodnôt poľa `POS`.

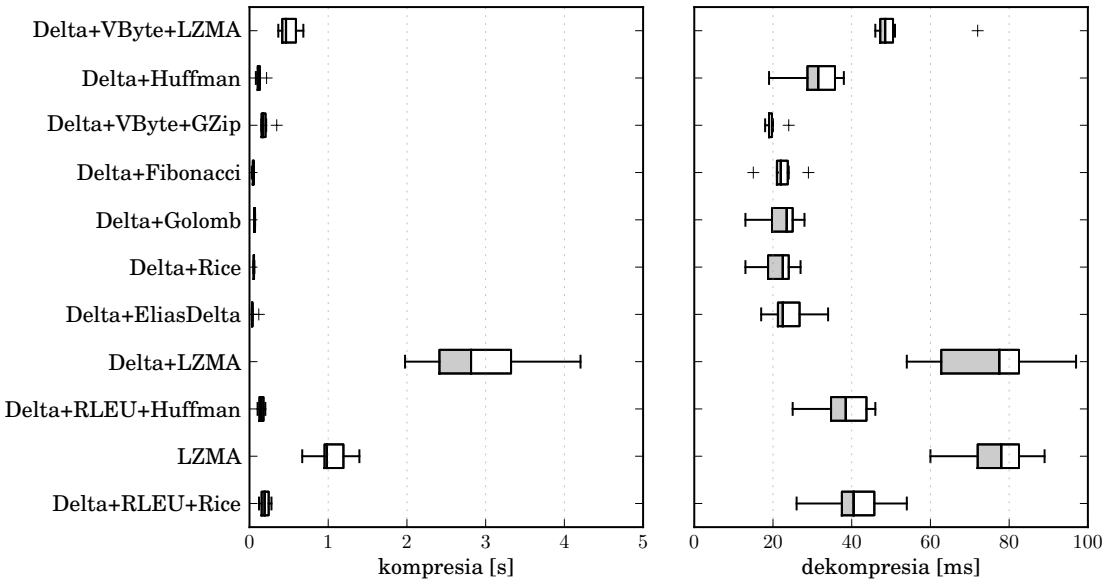
Kompresná metóda	**Kompresný pomer**						**BAM %**
	HG00109	**NA20589**	**KB1454**	**Manuel**	**Mouse**	**Ecoli**	
Delta+VByte+LZMA	**7.40%**	**9.13%**	**10.77%**	**6.19%**	**5.27%**	**3.50%**	0.46%
Delta+Huffman	**7.45%**	9.17%	11.34%	7.04%	**5.23%**	**3.50%**	0.48%
Delta+VByte+GZip	8.26%	9.30%	11.20%	6.92%	6.03%	3.63%	0.50%
Delta+Fibonacci	**7.50%**	9.87%	11.20%	**6.47%**	6.11%	6.04%	0.51%
Delta+Golomb	7.79%	**8.99%**	**11.13%**	11.05%	**5.22%**	**3.49%**	0.53%
Delta+Rice	7.85%	**8.99%**	**11.14%**	11.29%	5.36%	5.25%	0.55%
Delta+EliasDelta	8.17%	11.36%	12.35%	**6.90%**	6.61%	5.01%	0.55%
Delta+LZMA	9.00%	10.86%	12.62%	7.22%	6.45%	3.88%	0.55%
Delta+RLEU+Huffman	9.73%	11.73%	13.94%	8.33%	6.53%	5.42%	0.61%
LZMA	9.81%	12.01%	13.21%	9.06%	7.89%	6.25%	0.63%
Delta+RLEU+Rice	10.12%	12.07%	13.99%	9.01%	7.25%	5.71%	0.64%

Tab. 8.21: Účinnosť kompresie testovaných metód pre pole `POS`

8.6.5 Pole `MAPQ`

Pole `MAPQ` obsahuje hodnoty z oboru hodnôt $[0, 255]$. Zo štatistík v tabuľke 8.22 vidíme, že pri niektorých súboroch obsahuje dlhšie behy hodnôt. Preto môže byť vhodné pre jeho kompresiu využiť behové kódovanie.

Výsledky testovaných metód uvádzame v tabuľke 8.23 a v diagrame 8.17. Najlepší priemerný kompresný pomer dosahovala metóda LZMA. Má však vysokú časovú náročnosť, preto je vhodnejšie využiť metódu RLE+Huffman, ktorá dosahovala druhý najlepší kompresný pomer s výbornými časmi kompresie a dekompresie.



Obr. 8.16: Časová náročnosť testovaných metód pre pole `POS`

Súbor	Hodnoty			Behy hodnôt			
	Rozsah	Priemer	SD	Rozsah	Medián	Priemer	SD
HG00109	[0, 60]	29.37	16.02	[1, 29]	7.5	1.17	0.56
NA20589	[0, 70]	51.49	16.95	[1, 127]	49.5	3.11	6.45
KB1454	[255, 255]	255.00	0.00	[200000, 200000]	200000	200000.00	0.00
Manuel	[0, 70]	52.37	17.85	[1, 1399]	101.5	7.52	23.09
Mouse	[0, 99]	74.78	38.93	[1, 12322]	68	3.51	60.40
Ecoli	[0, 255]	250.00	29.49	[1, 13106]	500	38.80	427.27

Tab. 8.22: Štatistiky hodnôt a dĺžok behov týchto hodnôt v poli `MAPQ`

8.7 Nepovinné polia

Nepovinné polia môžu obsahovať rôznorodé dáta. Najskôr vyskúšame komprimovať dáta nepovinných polí ako celok, tak ako sú uvedené v textovej podobe v súbore SAM. Potom sa zameriame na niektoré vybrané polia, ktoré zaberajú pomerne veľkú časť vstupných súborov. Súbor `KB1454` neobsahuje nepovinné polia, preto sa ním nebudeme v tejto sekcii zaoberať.

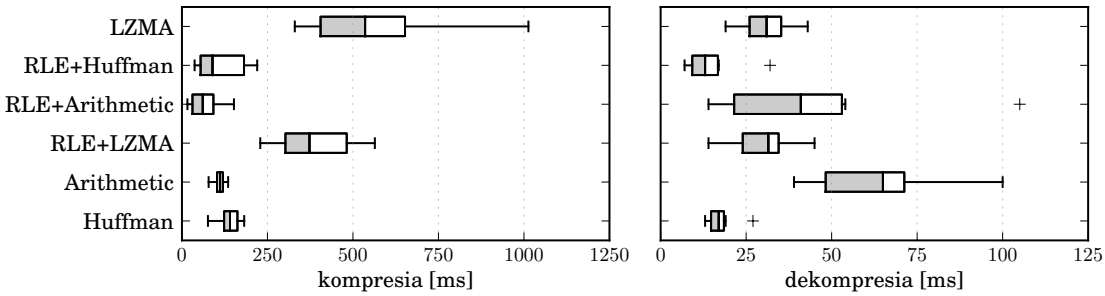
V tabuľke 8.24 a v diagrame 8.18 uvádzame výsledky zvolených metód pre kompresiu textovej podoby nepovinných polí. Najlepší kompresný pomer dosahovala metóda BZip2, čo nás po predošlých skúsenostiach s touto metódou prekvapilo.

8.7.1 Farebný priestor

K sekvenovaniu súboru `HG00109` bol použitý sekvenátor ABI SOLiD. Špecifikom tohto sekvenátora je, že produkuje prečítané segmenty v tzv. farebnom priestore (color space). Na zakódovanie dvoch po sebe idúcich báz používa 4 farby (čísla 0,1,2 a 3). Prevodovú tabuľku pre tento kód uvádzame v tabuľke 8.25. Autori tohto sekvenátora uvádzajú, že toto kódovanie umožňuje zlepšenie presnosti

Kompresná metóda	Kompresný pomer						BAM %
	HG00109	NA20589	KB1454	Manuel	Mouse	Ecoli	
LZMA	**30.94%**	**12.70%**	0.27%	**5.95%**	**14.65%**	**1.31%**	0.18%
RLE+Huffman	**30.91%**	13.02%	**0.15%**	**6.52%**	**14.91%**	1.60%	0.18%
RLE+Arithmetic	31.67%	13.20%	**0.24%**	6.63%	**14.71%**	**1.35%**	0.18%
RLE+LZMA	31.51%	13.34%	**0.15%**	**6.36%**	15.58%	**1.31%**	0.19%
Arithmetic	31.38%	**12.45%**	1.24%	7.44%	15.11%	2.94%	0.20%
Huffman	**30.54%**	**12.77%**	4.21%	9.38%	16.33%	5.71%	0.23%

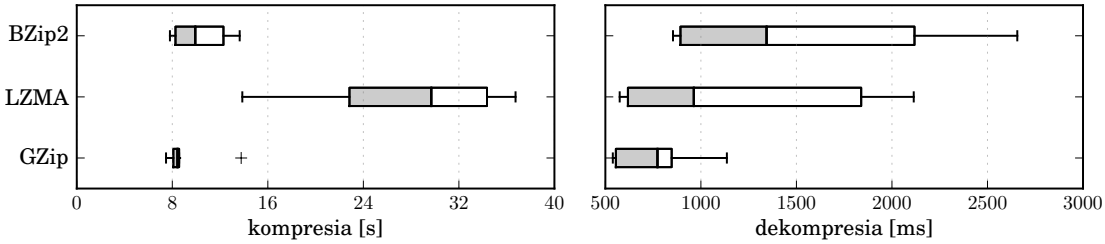
Tab. 8.23: Účinnosť kompresie testovaných metód pre pole `MAPQ`



Obr. 8.17: Časová náročnosť testovaných metód pre pole `MAPQ`

Kompresná metóda	Kompresný pomer					BAM %
	HG00109	NA20589	Manuel	Mouse	Ecoli	
BZip2	34.94%	**5.34%**	**27.77%**	**21.86%**	**13.45%**	11.96%
LZMA	**33.61%**	7.24%	28.68%	27.76%	15.97%	12.33%
GZip	39.12%	10.10%	34.50%	38.85%	21.75%	14.99%

Tab. 8.24: Účinnosť kompresie testovaných metód pre nepovinné polia



Obr. 8.18: Časová náročnosť testovaných metód pre nepovinné polia

sekvenovania [36].

Prvá báza \ Druhá báza	A	C	G	T
A	0	1	2	3
C	1	0	3	2
G	2	3	0	1
T	3	2	1	0

Tab. 8.25: Kód pre dekódovanie sekvencií vo farebnom priestore. K dekódovaniu sekvencie je navyše potrebná aspoň jedna báza zo sekvencie (primér) – ukladá sa spolu so sekvenciou vo farebnom priestore

Príklad 8.2. Majme nasledujúcu sekvenciu DNA:

`ACGTTTACGAT`,

k nej odpovedajúca sekvencia vo farebnom priestore je:

`A1310031323`,

kde prvým znakom je primér, ktorý určuje prvú bázu sekvencie.

Pole `CS` obsahuje sekvencie vo farebnom priestore. Pretože už máme uložené osekvenované bázy v poli `SEQ`, nie je potrebné uchovávať pri kompresii celý obsah tohto poľa. Stačí uložiť len primér a farby, ktoré odpovedajú neznámym bázam `N` v poli `SEQ`. Pri dekompresii je potom možné obsah poľa `CS` spätne zrekonštruovať zo znalosti sekvencie báz, priméru a sekvencie farieb v častiach poľa `SEQ`, ktoré obsahujú neznáme bázy `N`.

V tabuľke 8.26 uvádzame výsledky testovaných metód pri kompresii poľa `CS`. Vidíme, že pri dopočítavaní obsahu tohto poľa pri dekompresii je možné dosiahnuť nezanedbateľnú úsporu veľkosti ukladaných dát.

8.7.2 Kvalitatívne dáta

Nepovinné polia, ktoré zaberajú najväčšiu časť nami testovaných súborov, v prevažnej väčšine prípadov obsahujú kvalitatívne dáta, ktoré vznikli pri analýzach. Výpočet týchto kvalít je pomerne náročný na zdroje (výpočtová kapacita), preto sú uložené spolu s ostatnými dátami v SAM súbore. Polia, ktoré tu popíšeme majú rovnakú dĺžku ako pole `SEQ` alebo `QUAL` a používajú podobné kódovania hodnôt kvalít ako pole `QUAL`.

Pre testovanie kompresie týchto nepovinných polí sme zvolili metódy, ktoré dosahovali najlepšie výsledky pri kompresii poľa `QUAL`. Navyše sme k testovaniu zaradili metódu GZip, kvôli jej rýchlosti, a Huffmanovo kódovanie pre porovnanie s entropickým kódovaním. V tabuľke 8.27 uvádzame výsledky týchto metód

Kompresná metóda	% CS	% BAM	Čas kompresie [ms]	Čas dekompresie [ms]
Huffman	**1.28%**	0.57%	804	385
RLE+BZip2	1.34%	0.59%	1802	420
LZMA	1.42%	0.63%	1869	421
RLE+LZMA	1.45%	0.64%	2063	411
GZip	1.50%	0.66%	1192	361
RAW/LZMA	**17.99%**	7.95%	13770	670
RAW/RLE+LZMA	18.86%	8.34%	13940	645
RAW/GZip	20.98%	9.28%	2904	364
RAW/RLE+BZip2	23.78%	10.52%	8103	845
RAW/Huffman	31.64%	13.99%	1806	661

Tab. 8.26: Výsledky testovaných metód pre kompresiu nepovinného poľa `CS`. Prvá polovica tabuľky obsahuje výsledky pre metódu, ktorá dopočítava obsah poľa `CS` pri dekompresii. Druhá časť tabuľky obsahuje výsledky pre testované metódy pri kompresii pôvodného obsahu poľa `CS`

pre polia `CQ`, ktoré obsahoval súbor `HG00109`; `BQ`, ktoré obsahoval súbor `NA20589`; a `OQ`, ktoré obsahoval súbor `Manuel`.

Pole `BQ` obsahuje rozdiel hodnôt kvalít v poli `QUAL` a kvalít BAQ (Base Alignment Quality). Autor práce [34] tvrdí, že využívanie BAQ kvalít pri hľadaní nových genetických variácií zmenšuje počet chybne určených variácií. Toto pole obsahuje relatívne malé hodnoty a veľmi dlhé behy rovnakých hodnôt. Z tabuľky 8.27 vidíme, že testované metódy dosiahli výborné kompresné pomery pri jeho kompresii.

Špecifikácia SAM formátu udáva, že pole `OQ` obsahuje originálne kvality báz pred rekalibráciou. Podrobnejšie informácie o význame tohto poľa sa nám nepodarilo nájsť. Testované metódy dosiahli trochu lepšie výsledky ako pri kompresii poľa `QUAL`.

Pole `CQ` obsahuje kvality báz vo farebnom priestore. Testované metódy dosahovali podobný kompresný pomer ako pri kompresii poľa `QUAL`.

Kompresná metóda	Kompresný pomer			Čas kompresie / dekompresie [ms]		
	CQ	BQ	OQ	CQ	BQ	OQ
LZMA	**50.58%**	2.47%	34.05%	10300 / 933	13973 / 274	23749 / 1197
RLE+BZip2	51.46%	**2.06%**	**33.96%**	5910 / 1073	1604 / 178	8987 / 1485
RLE+LZMA	50.60%	2.16%	34.52%	8756 / 1034	1507 / 172	16048 / 1343
GZip	55.01%	2.89%	38.87%	2227 / 193	694 / 225	4109 / 244
Huffman	52.68%	13.45%	45.87%	2448 / 795	1567 / 387	4021 / 1244

Tab. 8.27: Výsledky testovaných metód pre nepovinné atribúty, ktoré obsahujú kvalitatívne dáta

8.7.3 Ostatné polia

Niektoré nepovinné polia je možné pomerne rýchlo vypočítať z ostatných dát. Príkladom takýchto polí sú polia `MD` a `NM`, ktoré je možné vypočítať ak máme k dispozícii referenčnú sekvenciu.

SAM súbory môžu obsahovať veľké množstvo štandardne definovaných nepovinných polí ako aj ľubovoľné množstvo nových polí. Nie je v našich silách, aby sme sa zaoberali všetkými týmito poliami. Aktuálne preto všetky ostatné nepovinné polia komprimujeme v ich textovej podobe bez využitia informácii o ich formáte. Program SamZip je však možné jednoducho rozšíriť aj o podporu týchto dát.

8.8 Kombinácie metód

Pre testy kombinácií najvhodnejších metód pre jednotlivé SAM polia sme vytvorili 4 kompresné profily:

- `Best`, ktorý je tvorený metódami, ktoré dosahovali najlepšie kompresné pomery spomedzi nami testovaných metód. Pre kompresiu polí `SEQ` a `CIGAR` používa referenčnú sekvenciu vytvorenú pri kompresii/dekompresii.
- `Best+Ref`, ktorý je tvorený rovnakými metódami ako profil `Best`, ale využíva reálnu referenčnú sekvenciu. V tomto profile nie je ukladaný obsah nepovinných polí `MD` a `NM` – tie sú vypočítané pri dekompresii.
- `Optimal`, ktorý je tvorený metódami, ktoré ponúkajú dobrý kompromis medzi účinnosťou kompresie a ich časovou náročnosťou. Pre kompresiu polí `SEQ` a `CIGAR` používa referenčnú sekvenciu vytvorenú pri kompresii/dekompresii.
- `Optimal+Ref`, ktorý je tvorený rovnakými metódami ako profil `Optimal`, ale využíva reálnu referenčnú sekvenciu. V tomto profile nie je ukladaný obsah nepovinných polí `MD` a `NM` – tie sú vypočítané pri dekompresii.

Popis použitých metód v profiloch `Best` a `Optimal` uvádzame v tabuľke 8.28.

V tabuľke 8.29 a v diagrame 8.19 uvádzame dosiahnutú úroveň kompresie na testovacích súboroch.

Na obrázkoch 8.20 a 8.21 je znázornené, ktoré SAM polia najviac prispievajú k veľkosti komprimovaného súboru pri použití referenčnej sekvencie resp. bez jej použitia. Vidíme, že najväčšiu časť zaberajú kvality a nepovinné polia.

V tabuľke 8.30 uvádzame rýchlosť kompresie a dekompresie týchto kompresných profilov. Vidíme, že rýchlosť kompresie narastá, ak je pri kompresii využívané väčšie množstvo vlákien. Nárast rýchlosti kompresie sa však s rastúcim počtom vlákien spomaľuje. Pravdepodobne začína byť obmedzením prístup na disk – čítanie vstupného BAM súboru a zapisovanie skomprimovaných blokov na disk je možné vykonávať len v jednom vlákne. V prípade dekompresie môže byť prekvapivé, že pri dekompresii priamo do formátu BAM sa znížila rýchlosť dekompresie

až na polovicu. Dôvodom je, že dekomprimované dáta sú pri ukladaní do súboru BAM komprimované metódou gzip.

Pre veľké množstvo úkonov/analýz, ktoré sú bežne vykonávané na súboroch vo formáte SAM je potrebná referenčná sekvencia – pre užívateľa nepredstavuje problém využiť túto referenčnú sekvenciu aj pri kompresii. Nechávame na zvážení užívateľa, či je preňho výhodné používať referenčnú sekvenciu v iných situáciach (tj. ak ju nepotrebuje pre analýzy), alebo sa uspokojí s trochu horším kompresným pomerom, keď ju nebude využívať.

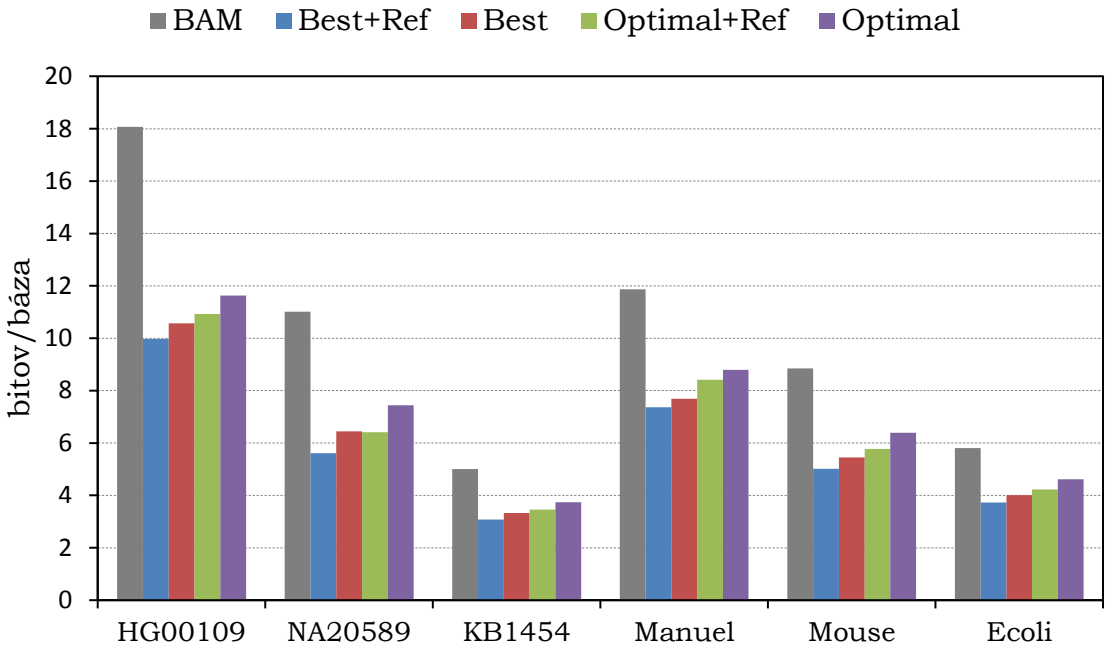
Pole	**Profil** `Best`	**Profil** `Optimal`
`QNAME`	LZMA	GZip
`FLAG`	Huffman	Huffman
`RNAME`	RLEU+Fibonacci	RLEU+Fibonacci
`POS`	Delta+Huffman	Delta+Huffman
`MAPQ`	RLE+Huffman	RLE+Huffman
`CIGAR`	LZMA	GZip
`SEQ`	LZMA	GZip
`QUAL`	LZMA	GZip
`RNEXT`	Rice/RLE+Huffman[1]	Rice/RLE+Huffman[1]
`PNEXT`	VByteLZMA	VByteGZip
`TLEN`	VByteLZMA	VByteGZip
`TAG:BQ`	RLE+BZip2	GZip
`TAG:CQ`	LZMA	GZip
`TAG:CS`	výpočet + Huffman	výpočet + Huffman
`TAG:OQ`	RLE+BZip2	GZip
ostatné	BZip2	GZip

[1] Použité pre kompresiu poľa bitových príznakov, ktoré indikujú zhodu s `RNAME`

Tab. 8.28: Metódy používané v profiloch `Best` a `Optimal`

Profil	**Kompresný pomer**					
	HG00109	**NA20589**	**KB1454**	**Manuel**	**Mouse**	**Ecoli**
Best+Ref	55.21%	50.92%	61.56%	62.03%	56.64%	64.22%
Best	58.52%	58.58%	66.65%	64.76%	61.56%	69.05%
Optimal+Ref	60.49%	58.27%	69.06%	70.86%	65.21%	72.75%
Optimal	64.39%	67.55%	74.72%	74.13%	72.18%	79.52%

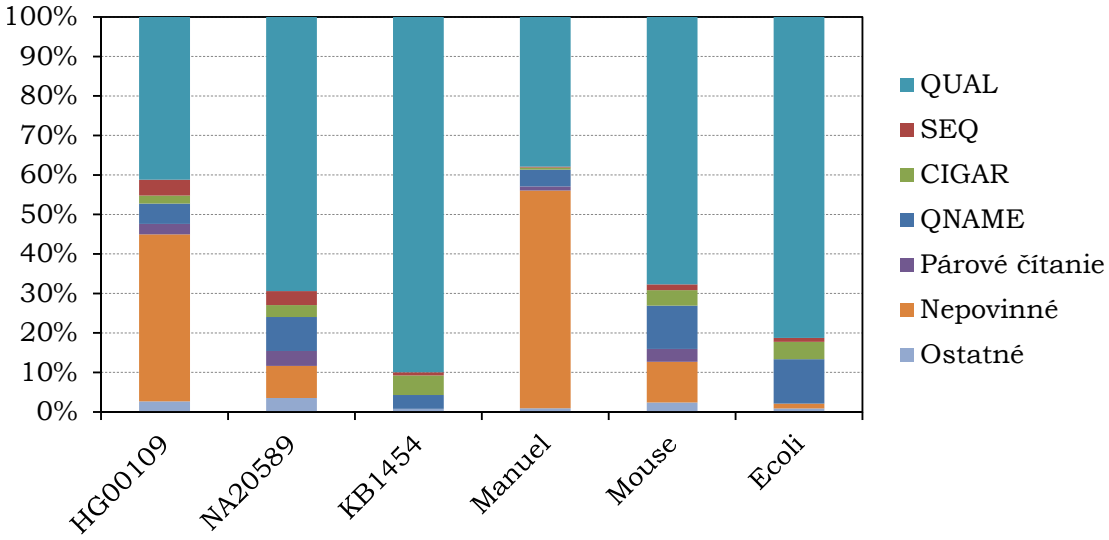
Tab. 8.29: Účinnosť kompresie pri použití zvolených kompresných profilov. Kompresný pomer je udávaný vzhľadom k veľkosti vstupného súboru vo formáte BAM



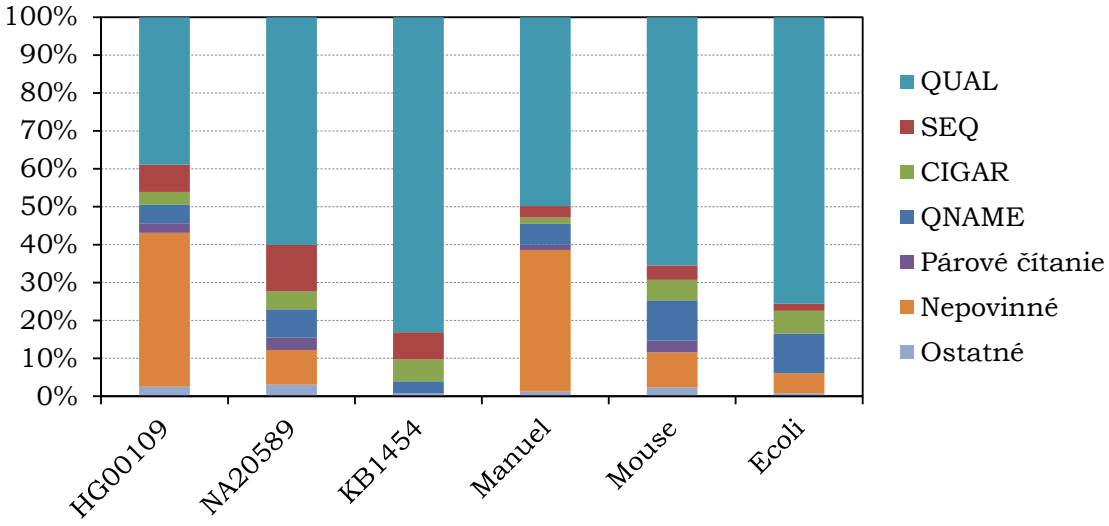
Obr. 8.19: Účinnosť kompresie pri použití zvolených kompresných profilov udávaná ako počet bitov potrebných pre uloženie jednej bázy v komprimovanom súbore

Profil	**Kompresia**				**Dekompresia**	
	1	**2**	**4**	**8**	**BAM**	**SAM**
Best+Ref	723	1214	1759	1883	2545	4289
Best	646	1147	1802	1862	2373	3747
Optimal+Ref	2171	3549	4453	4322	3265	7543
Optimal	1808	3201	4750	5361	3476	7473

Tab. 8.30: Priemerná rýchlosť kompresie a dekompresie testovaných kompresných profilov v kB/s (z veľkosti vstupného BAM súboru). Stĺpec kompresia je rozdelený na stĺpce podľa počtu vlákien, ktoré boli použité pri kompresii. Dekompresia využívala len 1 vlákno. Stĺpec dekompresia je rozdelený na stĺpce BAM a SAM, ktoré udávajú rýchlosť dekompresie v kB/s pri ukladaní dekomprimovaných dát do súboru vo formáte BAM resp. SAM



Obr. 8.20: Pomer veľkosti SAM polí v komprimovaných súboroch pri použití referenčnej sekvencie



Obr. 8.21: Pomer veľkosti SAM polí v komprimovaných súboroch bez použitia referenčnej sekvencie

8.9 Stratová kompresia kvalít

Najjednoduchším spôsobom, ktorým je možné dosiahnuť zvýšenie účinnosti kompresie kvalít je ich kvantizácia. Pri kvantizácii sú nahradené pôvodné hodnoty hodnotami z užšieho oboru hodnôt (obsahuje menej rôznych hodnôt). Dáta s užším oborom hodnôt sú zvyčajne jednoduchšie komprimovateľné ako dáta so širokým oborom hodnôt.

Parameter kvantizácie	**Kompresný pomer**					
	HG00109	**NA20589**	**KB1454**	**Manuel**	**Mouse**	**Ecoli**
10	42.94%	35.14%	30.25%	40.92%	36.36%	29.91%
8	44.24%	37.55%	34.08%	42.70%	37.43%	33.06%
5	47.58%	41.46%	41.22%	46.70%	41.30%	39.52%
2	53.86%	50.48%	56.78%	56.63%	51.67%	55.81%

Tab. 8.31: Úroveň kompresie pri stratovej kompresii kvalít. Kompresný pomer je udávaný vzhľadom k veľkosti vstupného súboru vo formáte BAM

V tabuľke 8.31 uvádzame úroveň kompresie, ktorú sme dosiahli pri použití profilu `Best+Ref` doplneného o kvantizáciu hodnôt kvalít. Zvolili sme veľmi jednoduchú metódu kvantizácie, v ktorej je každá vstupná hodnota zaokrúhlená k najbližšiemu celému násobku kvantizačného parametra. Vidíme, že pri zaokrúhlení hodnôt kvalít na celé desiatky sme dosiahli zlepšenie kompresného pomeru oproti bezstratovej kompresii až na polovicu.

Iným prístupom k stratovej kompresii kvalít je využitie znalosti referenčnej sekvencie. Pri analýzach je často potrebné poznať kvality len pre bázy, ktoré sa nezhodujú s referenčnou sekvenciou. V takom prípade nie je potrebné ukladať kvality pre bázy, ktoré sú zhodné s referenčnou sekvenciou. Tento prístup využívajú napríklad v práci [22], ktorú sme podrobnejšie popísali v sekcii 6.2.

9. Súhrnné testy

V tejto kapitole uvedieme výsledky testov aplikácie SamZip pri kompresii veľkých súborov. Uvedieme taktiež porovnanie s aplikáciou CRAM, ktorá má podobné zameranie ako aplikácia SamZip. Najskôr v sekcii 9.1 popíšeme spôsob testovania. Potom v sekcii 9.2 uvedieme a zhodnotíme výsledky týchto testov.

9.1 Metodika testovania

Testy v tejto kapitole pozostávajú z kompresie a dekompresie väčších BAM súborov. Hlavnou úlohou týchto testov je overenie funkčnosti aplikácie SamZip na väčších súboroch. V testoch budeme sledovať účinnosť kompresie a časovú náročnosť kompresie a dekompresie.

Pri testovaní otestujeme všetky kompresné profily, ktoré sme popísali v sekcii 8.8.

Pre porovnanie uvedieme výsledky kompresie a dekompresie testovacích súborov s aplikáciou CRAM (viď sekcia 6.2). V testoch budeme využívať len bezstratovú kompresiu. CRAM neumožňuje kompresiu bez referenčnej sekvencie.

9.1.1 Vstupné dáta

Testy sme vykonali na desiatich súboroch vo formáte BAM. Šesť z nich tvoria súbory, ktoré sme použili pre vytvorenie testovacích súborov v kapitole 8. Ich popis sme uviedli v sekcii 8.1.3. Kompresiou týchto súborov chceme overiť, že sme pri výbere kompresných metód používali dostatočne reprezentatívnu vzorku týchto súborov – tj. testované kompresné profily sa budú správať podobne pri kompresii celých súborov ako sa správali pri kompresii vzorky týchto súborov.

Ďalej sme zvolili nasledujúce súbory:

- `NA12878` Exónová oblasť genómu človeka. K sekvenovaniu bol použitý sekvenátor ILLUMINA. Dáta pochádzajú z projektu „1000 Genomes Project“.
- `NA12878/N` Pochádza z rovnakého sekvenovania ako súbor NA12878, obsahuje však segmenty, ktoré nebolo možné zarovnať k referenčnej sekvencii pri zarovnávaní. Pri tomto súbore preto nebudeme testovať profily, ktoré využívajú referenčnú sekvenciu.
- `NA21144` Kompletný genóm človeka s nízkym pokrytím. K sekvenovaniu bol použitý sekvenátor SOLiD. Dáta pochádzajú z projektu „1000 genomes project“.
- `NA21144/N` Pochádza z rovnakého sekvenovania ako súbor `NA21144`. Obsahuje však segmenty, ktoré nebolo možné zarovnať k referenčnej sekvencii pri zarovnávaní. Pri tomto súbore preto nebudeme testovať profily, ktoré využívajú referenčnú sekvenciu.

V tabuľke 9.1 uvádzame veľkosti testovacích súborov vo formátoch SAM a BAM.

Súbor	**Veľkosť SAM**	**Veľkosť BAM**	**|BAM| / |SAM|**	**bity/báza**
HG00109	1831 MB	695 MB	37.95%	18.18071
NA20589	1035 MB	259 MB	25.02%	11.00536
KB1454	1202 MB	344 MB	28.62%	4.968263
Manuel	13.6 GB	4 GB	29.64%	11.93853
Mouse	187.1 GB	54.3 GB	29.00%	9.080957
Ecoli	4.6 GB	968 MB	20.61%	5.711903
NA12878	73.5 GB	16.1 GB	21.89%	8.562303
NA12878/N	2500 MB	416 MB	16.65%	3.734062
NA21144	25.6 GB	10.3 GB	40.30%	20.26514
NA21144/N	27.4 GB	11.9 GB	43.22%	19.70855

Tab. 9.1: Veľkosť testovacích súborov vo formátoch SAM a BAM. Stĺpec BAM/SAM obsahuje pomer veľkosti súboru vo formáte BAM k veľkosti súboru vo formáte SAM. Stĺpec bity/báza udáva priemerný počet bitov použitých pre uloženie jednej bázy v BAM súbore

9.1.2 Platforma

Všetky testy sme vykonávali na rovnakej platforme ako pri výbere kompresných metód. Túto platformu sme podrobnejšie popísali v sekcii 8.1.4.

Testy aplikácie SamZip sme spúšťali nasledujúcim volaním:

```
java -Xms256m -Xmx2048m -server -jar samzip.jar test -p={kompresný
    profil} --in={vstupný BAM súbor} --out={výstupný SamZip súbor} --
    dec={dekomprimovaný súbor} --bam -t=8 -b=5000 --stringency=
    LENIENT
```

Pri kompresii aplikáciou SamZip sme využívali 8 vlákien a bloky s veľkosťou 5000 záznamov. Dekomprimovali sme priamo do formátu BAM. Behové prostredie Javy malo k dispozícii maximálne 2048 MB pamäte RAM.

V testoch aplikácie CRAM sme využívali verziu `0.85-b44` tejto aplikácie. Testy kompresie aplikáciou CRAM sme spúšťali nasledujúcim volaním:

```
java -Xms256m -Xmx3072m -server -jar cramtools.jar cram --input-bam-
    file {vstupný BAM súbor} --reference-fasta-file {referenčná
    sekvencia} --output-cram-file {výstupný (komprimovaný) súbor} --
    capture-insertion-quality-scores --capture-substitution-quality-
    scores --capture-unmapped-quality-scores --capture-masked-quality
    -scores --capture-all-tags --include-unmapped-reads --capture-
    piled-quality-scores
```

Počet vlákien, ktoré aplikácia CRAM využíva ku kompresii závisí na počte procesorov dostupných v systéme. V našom prípade využívala 8 vlákien. Aplikácia CRAM vyžadovala väčšie množstvo pamäti ako aplikácia SamZip[1].

Testy dekompresie CRAM súborov sme spúšťali nasledujúcim volaním:

```
java -Xms256m -Xmx3072m -server -jar cramtools.jar bam --input-cram-
    file {komprimovaný súbor} --reference-fasta-file {referenčná
    sekvencia} --output-bam-file {dekomprimovaný súbor}
```

9.2 Výsledky

V diagrame 9.1 uvádzame úroveň kompresie, ktorú sme dosiahli v našich testoch. Pri súbore `Mouse` mala aplikácia CRAM problémy s kompresiou, preto nebudeme výsledok tohto testu uvádzať.

V tabuľke 9.2 uvádzame rýchlosť kompresie a dekompresie. Celkový čas kompresie a dekompresie uvádzame v diagramoch 9.2 a 9.3.

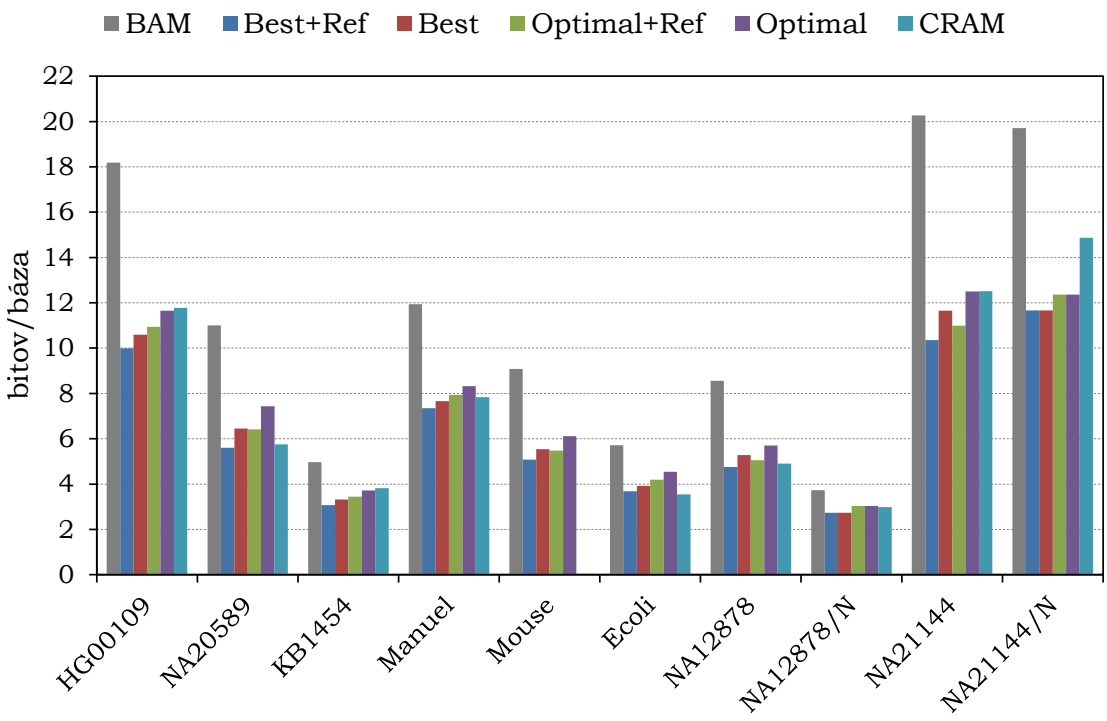
Dosiahnuté výsledky pre prvých šesť súborov sa zhodujú s výsledkami, ktoré sme dosiahli na vzorkách týchto súborov (viď sekcia 8.8). Môžeme preto prehlásiť, že vzorky týchto súborov, ktoré sme používali pri výbere kompresných metód, boli dostatočne reprezentatívne.

Kompresný profil `Best+Ref` dosahuje vo všetkých súboroch okrem `Ecoli` lepšiu úroveň kompresie ako CRAM. Kompresia týmto profilom však vyžadovala 2 – 3 krát viac času ako CRAM. Na druhú stranu je čas dekompresie porovnateľný alebo dokonca lepší ako u CRAM.

U ostatných kompresných profilov sme pri niektorých súboroch dosiahli lepšiu úroveň kompresie ako CRAM. Toto je zaujímavé hlavne z toho hľadiska, že rýchlosť kompresie a dekompresie s týmito profilmi bola vyššia ako u CRAM – v niektorých prípadoch až dvojnásobne.

Profily `Optimal+Ref` a `Optimal` obyčajne dosahovali rýchlejšiu dekompresiu ako ostatné profily. Výnimkou je súbor `Mouse`, pri ktorom bola dekompresia pri použití týchto profilov zásadne pomalšia ako pri profiloch `Best+Ref` a `Best`. Vysvetlenie tejto anomálie nepoznáme, vyskytla sa však aj po opakovanom testovaní.

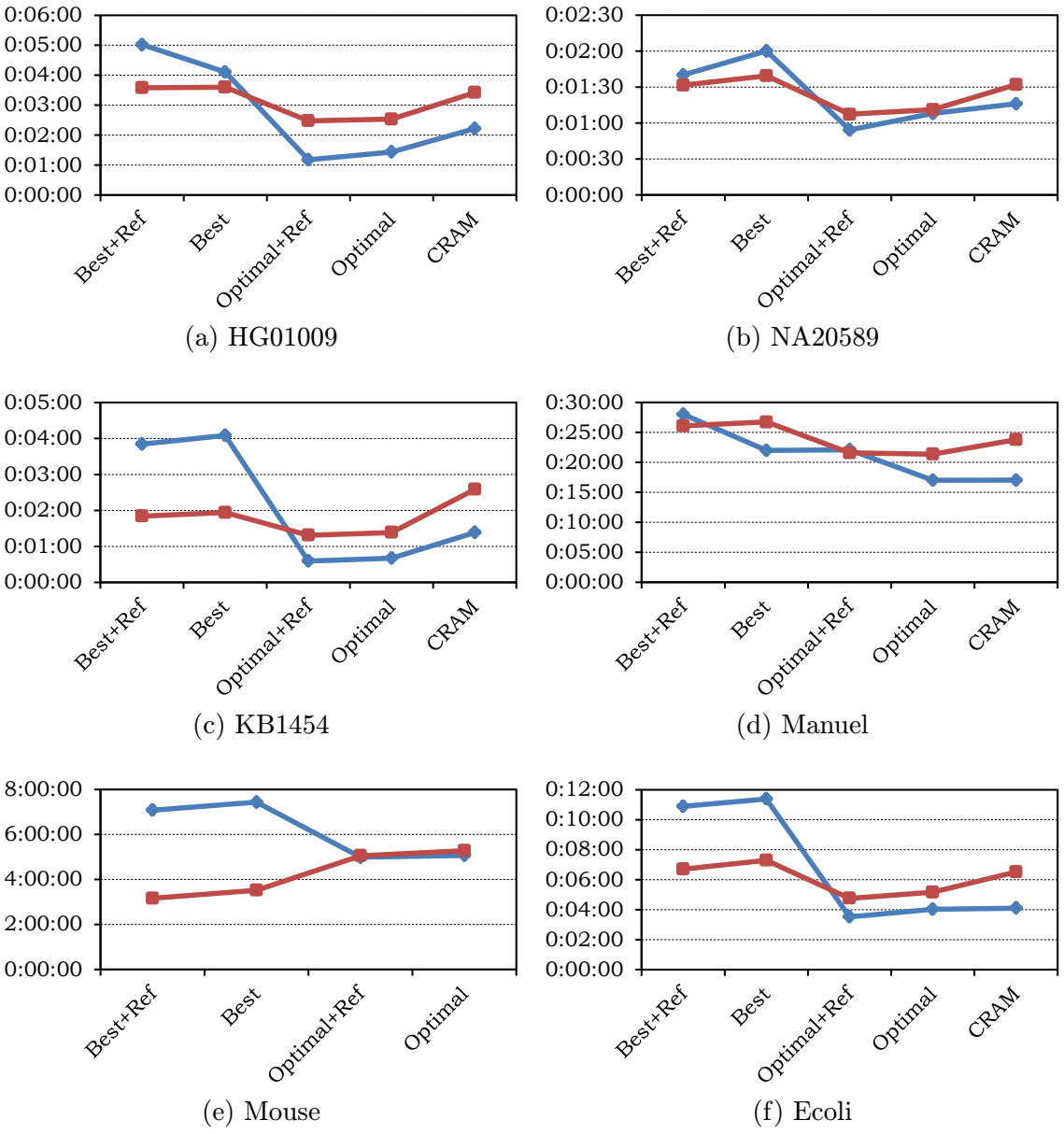
[1] Padala na nedostatok pamäte pri nastavení `-Xmx2048m`



Obr. 9.1: Účinnosť kompresie pre testované kompresné profily a aplikáciu CRAM. Účinnosť kompresie udávame ako počet bitov potrebných pre uloženie jednej bázy v komprimovanom súbore. Pre porovnanie udávame aj údaje pre súbory vo formáte BAM

Súbor	**Kompresia**					**Dekompresia**				
	Best+Ref	**Best**	**Optimal+Ref**	**Optimal**	**CRAM**	**Best+Ref**	**Best**	**Optimal+Ref**	**Optimal**	**CRAM**
HG00109	2365	2891	10094	8267	5351	3315	3297	4792	4684	3471
NA20589	2648	2208	4896	3898	3488	2896	2668	3942	3729	2881
KB1454	1529	1439	9892	8715	4244	3194	3026	4485	4237	2273
Manuel	2507	3193	3180	4126	4122	2692	2627	3254	3287	2954
Mouse	2232	2125	3165	3118	*	4998	4492	3127	2993	*
Ecoli	1516	1450	4696	4105	4029	2465	2264	3479	3198	2535
NA12878	2381	2415	2732	2993	2954	2500	2535	2780	2917	2466
NA21144	3591	3672	5995	6511	4810	3224	3009	3791	3800	3172
NA12878/N	*	839	*	1842	2842	*	2181	*	2807	1837
NA21144/N	*	3043	*	7689	4652	*	3299	*	4080	3097

Tab. 9.2: Rýchlosť kompresie a dekompresie testovaných kompresných profilov a aplikácie CRAM v kB/s (z veľkosti vstupného BAM súboru). Hviezdičkou sú označené experimenty, ktoré sme nevykonali

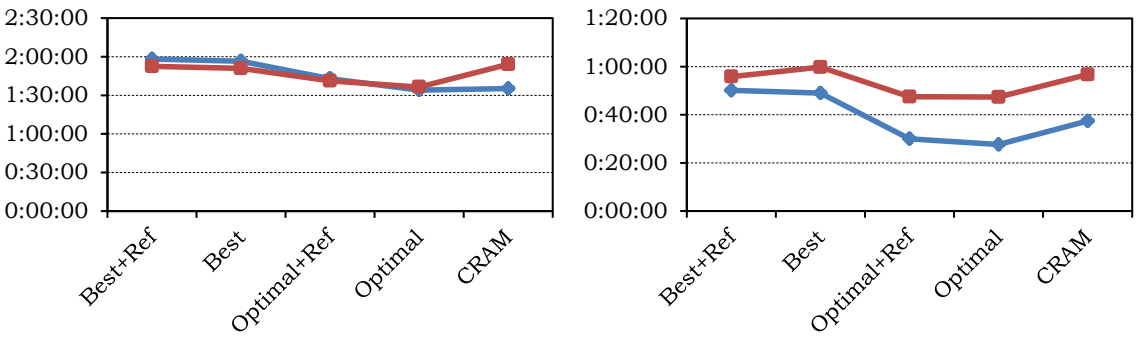


(a) HG01009 (b) NA20589

(c) KB1454 (d) Manuel

(e) Mouse (f) Ecoli

Obr. 9.2: Časová náročnosť pre testované kompresné profily a aplikáciu CRAM. Modrá čiara je čas kompresie, červená čiara je čas dekompresie. Čas na vertikálnej osi je vo formáte *hodiny:minúty:sekundy*



(a) NA12878 (b) NA21144

(c) NA12878/N (d) NA21144/N

Obr. 9.3: Časová náročnosť pre testované kompresné profily a aplikáciu CRAM. Modrá čiara je čas kompresie, červená čiara je čas dekompresie. Čas na vertikálnej osi je vo formáte *hodiny:minúty:sekundy*

10. Záver

Kompresná schéma, ktorú sme vytvorili v tejto práci umožňuje zmenšenie veľkosti binárnych komprimovaných súborov BAM až o 30 – 50%. Najväčší problém pri kompresii predstavovala kompresia hodnôt kvalít, ktoré aj po kompresii tvoria veľkú časť komprimovaného súboru. Využitím stratovej kompresie týchto hodnôt je možné ďalšie zlepšenie úrovne kompresie. Stratovou kompresiou kvalít sme sa však hlbšie nezaoberali, pretože jej implementácia si vyžaduje náročné experimenty pre určenie dopadu straty informácie na presnosti analýz.

Implementácia vytvorenej kompresnej schémy, aplikácia a formát SamZip, je spoľahlivá a dostatočne rýchla. Pri kompresii vytvára index komprimovaného súboru, ktorý umožňuje náhodný prístup do celého súboru prostredníctvom jednoduchých dotazov.

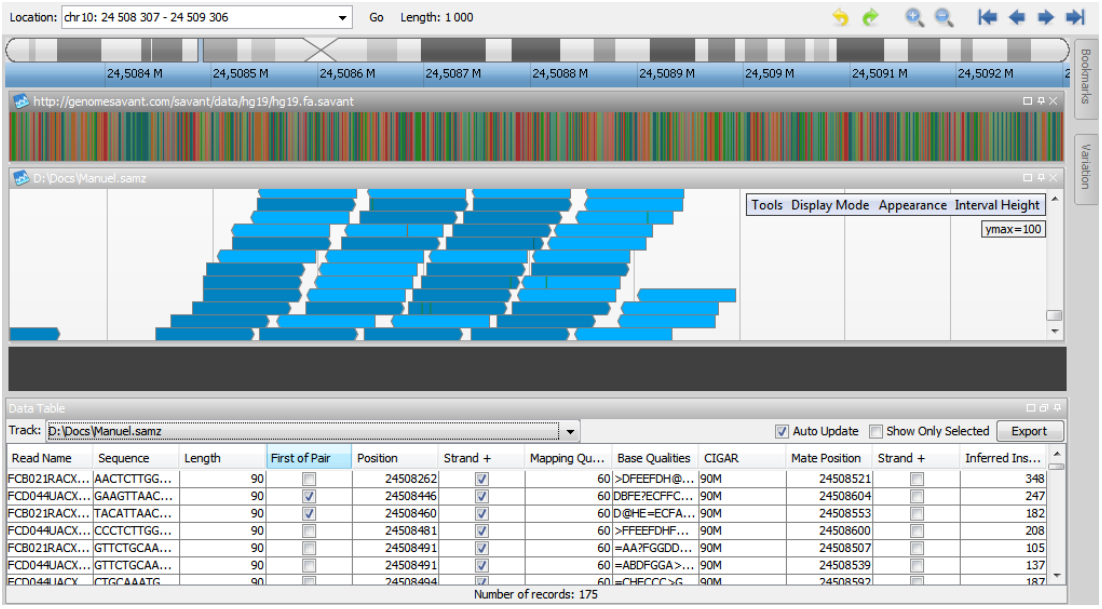
Pre implementáciu aplikácie SamZip sme použili jazyk Java, čo umožnilo jej nezávislosť na použitom operačnom systéme. Pri návrhu aplikačného rozhrania (API) aplikácie sme sa snažili o to, aby bolo jednoduché a čo najviac podobné rozhraniu bežne dostupných knižníc pre prácu so súbormi vo formáte SAM/BAM. Kompresnú schému použitú v aplikácii je možné konfigurovať pomocou konfiguračných XML súborov. Aplikáciu je taktiež možné jednoducho rozšíriť o podporu nových kompresných metód alebo nových postupov pre kompresiu SAM súborov (SAM procesorov).

Našu implementáciu sme porovnali s dostupnou implementáciou aplikácie CRAM, ktorá taktiež umožňuje kompresiu súborov vo formáte SAM. Pri väčšine testovacích súborov dosiahla naša implementácia lepšiu úroveň kompresie ako aplikácia CRAM. Ak sme dosiahli horšiu úroveň kompresie, tak bola porovnateľná (nebola zásadne horšia) s CRAM. Čo sa týka časovej náročnosti, naša kompresná schéma vyžadovala 2 – 3 krát viac času pri kompresii, a porovnateľne alebo menej času pri dekompresii.

Aj keď existuje veľké množstvo automatických procedúr pre analyzovanie bioinformatických dát, pre koncové analýzy je často potrebné mať možnosť vizualizácie týchto dát. Aby sme demonštrovali príklad použitia aplikácie SamZip, implementovali sme zásuvný modul do aplikácie Savant [20][1], ktorá umožňuje takúto vizualizáciu. Tento zásuvný modul umožňuje zobrazenie obsahu SamZip súborov v aplikácii Savant. Na obrázku 10.1 uvádzame snímok obrazovky aplikácie Savant, ktorá využíva tento zásuvný modul.

Takéto riešenie je možné nasadiť na bežne dostupných počítačoch, ktoré nemusia disponovať obzvlášť výkonným hardvérom. Vďaka použitiu kompresie prostredníctvom aplikácie SamZip je navyše možné uchovávať na dostupných úložných zariadeniach väčšie množstvo dát ako v prípade uchovania týchto súborov vo formáte BAM.

[1] `<http://genomesavant.com>`



Obr. 10.1: Snímok obrazovky z aplikácie Savant. Horná časť obrazovky umožňuje prístup ku konkrétnej pozícii v referenčnej sekvencii. Pod ňou sa nachádza stopa s referenčnou sekvenciou. Ďalej sú už uvedené samotné SAM záznamy z komprimovaného SamZip súboru, ktoré sprostredkúva náš zásuvný modul. Savant umožňuje zobrazenie ľubovoľného množstva ďalších stôp – napr. známe genetické variácie alebo gény. V dolnej časti okna sa nachádza tabuľka s obsahom aktuálne zobrazených SAM záznamov

10.1 Budúcnosť aplikácie SamZip

Aplikácia SamZip v súčasnej podobe poskytuje aplikačné rozhranie a rozhranie príkazového riadku pre kompresiu a dekompresiu súborov vo formáte SAM/BAM. Taktiež umožňuje rozšírenie o nové kompresné metódy a nové SAM procesory. Pri ďalšom vývoji by bolo vhodné pridať nasledujúcu funkcionalitu:

- **Umožnenie prístupu po sieti**. Niektoré knižnice pre prácu so súbormi vo formáte SAM umožňujú prístup k častiam týchto súborov po sieti.
- **Filtrovanie výstupu pri dekompresii**. Pri analýzach je často potrebné mať k dispozícii len SAM záznamy, ktoré spĺňujú určité kritériu. Preto by bolo vhodné aby aplikácia SamZip umožnila nastavenie filtrov, ktoré by pri dekompresii testovali zvolené kritérium a vrátili len záznamy, ktoré mu odpovedajú.
- **Nástroje pre analýzy**. Aplikácie, ktoré sú štandardne používané pre prácu so SAM súbormi ponúkajú okrem zobrazovania SAM záznamov a konverzie medzi formátmi SAM a BAM taktiež možnosti pre analýzu týchto súborov (napr. automatické hľadanie genetických variácií). Preto by bolo vhodné implementovať mechanizmus, ktorý umožní jednoduchú rozšíriteľnosť aplikácie SamZip aj o podporu týchto analýz.

Zoznam použitej literatúry

[1] Archon Genomics. *Archon Genomics X Prize* [online]. c2012 [cit. 2012-07-09]. Dostupné z: <http://genomics.xprize.org>.

[2] Celera Corporation. *Celera: Personalizing Disease Management* [online]. c2012 [cit. 2012-07-09]. Dostupné z: <https://www.celera.com>.

[3] Human Genome Project. *Genetics Privacy and Legislation* [online]. September 2008 [cit. 2012-07-09] Dostupné z: <http://www.ornl.gov/sci/techresources/Human_Genome/elsi/legislat.shtml>.

[4] Human Genome Project. *Human Genome Project Information* [online]. 2011 [cit 2012-07-09]. Dostupné z: <http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml>.

[5] *The Computer Language Benchmarks Game* [online]. c2012 [cit 2012-07-09]. Dostupné z: <http://shootout.alioth.debian.org>.

[6] Personal Genome Project. *Personal Genome Project - Homepage* [online]. c2012 [cit 2012-07-09]. Dostupné z: <http://www.personalgenomes.org>.

[7] 1001 Genomes. *A Catalog of Arabidopsis thaliana Genetic Variation* [online]. c2010 [cit 2012-07-09]. Dostupné z: <http://www.1001genomes.org>.

[8] *International Cancer Genome Consortium* [online]. Naposledy aktualizované 2012-03-16 [cit 2012-07-09]. Dostupné z: <http://www.icgc.org>.

[9] *UK10K* [online]. Naposledy aktualizované 2011-05-28 [cit 2012-07-09]. Dostupné z: <http://www.uk10k.org>.

[10] *The SAM Format Specification* [online]. Verzia 1.4-r985. September 2011 [cit 2012-07-09]. Dostupné z: <http://samtools.sourceforge.net/SAM1.pdf>.

[11] 1000 Genomes Project Consortium. A map of human genome variation from population-scale sequencing. *Nature*. 2010, vol. 467, no. 7319, s. 1061–1073.

[12] BURROWS, Michael; WHEELER, David. *A Block-Sorting Lossless Data Compression Algorithm.* Digital SRC Research Report, 1994.

[13] CAO, Minh Duc, et al. *A Simple Statistical Algorithm for Biological Sequence Compression.* 2007.

[14] CHEN, Xin; KWONG, Sam; LI, Ming. A compression algorithm for DNA sequences and its applications in genome comparison. *Proceedings of the fourth annual international conference on Computational molecular biology*, 2000.

[15] CHRISTLEY, Scott, et al. Human genomes as email attachments. *Bioinformatics*. 2009, vol. 25, no. 2, s. 274–275.

[16] COCK, Peter, et al. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic acids research*. 2010, vol. 38, no. 6, s. 1767–1771.

[17] ELIAS, Peter. *Universal Codeword Sets and Representations of the Integers*. MIT, 1975.

[18] EWING, Brent; GREEN, Phil, et al. Base-Calling of Automated Sequencer Traces UsingPhred. I. Accuracy Assessment. *Genome Research*. 1998, vol. 8, no. 3, s. 175–185.

[19] EWING, Brent; GREEN, Phil, et al. Base-Calling of Automated Sequencer Traces UsingPhred. II. Error Probabilities. *Genome Research*. 1998, vol. 8, no. 3, s. 186–194.

[20] FIUME, Marc, et al. Savant: genome browser for high-throughput sequencing data. *Bioinformatics*. 2010, vol. 26, no. 16, s. 1938–1944.

[21] FRAENKEL, Aviezri S.; KLEIN, Shmuel T. Robust Universal Complete Codes for Transmission and Compression. *Discrete Applied Mathematics*. 1996, vol. 64, s. 31–55.

[22] FRITZ, Markus, et al. Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome research*. 2011, vol. 21, no. 5, s. 734–740.

[23] Genome 10K Community of Scientists. Genome 10K: a proposal to obtain whole-genome sequence for 10,000 vertebrate species. *The Journal of heredity*. 2009, vol. 100, no. 6, s. 659–674.

[24] GIANCARLO, R.; SCATURRO, D.; UTRO, F. Textual data compression in computational biology: a synopsis. *Bioinformatics*. 2009, vol. 25, no. 13, s. 1575–1586.

[25] GRUMBACH, Stéphane; TAHI, Fariza. A New Challenge for Compression Algorithms: Genetic Sequences. *Information Processing & Management*. 1994, vol. 30.

[26] HANKERSON, Darrel; HARRIS, Greg A.; JOHNSON, Peter D. *Introduction to information theory and data compression*. 2nd ed. Chapman & Hall/CRC, 2003. 362pp.

[27] JONES, Neil; PEVZNER, Pavel. *An Introduction To Bioinformatics Algorithms*. The MIT Press, 2004. 448pp. ISBN 9780262101066.

[28] KEANE, Thomas, et al. Mouse genomic variation and its effect on phenotypes and gene regulation. *Nature*. 2011, vol. 477, no. 7364, s. 289–294.

[29] KIMBALL, John. *Kimball's Biology Pages* [online]. 1999, naposledy aktualizované 2012-06-22 [cit. 2012-07-09]. Dostupné z: <http://biology-pages.info>.

[30] KODAMA, Yuichi, et al. The Sequence Read Archive: explosive growth of sequencing data. *Nucleic acids research*. 2012, Vol. 40, Database issue.

[31] KURUPPU, Shanika, et al. Relative Lempel-Ziv Compression of Genomes for Large-Scale Storage and Retrieval. *Proceedings of the 17th international conference on String processing and information retrieval*. 2010, s. 201–206.

[32] LESK, Arthur. *Introduction to Bioinformatics*. 2nd ed. Oxford University Press, 2002. 308pp. ISBN 9780199251964.

[33] LI, Heng; Durbin, Richard, et al. The sequence alignment/map format and SAMtools. *Bioinformatics*. 2009, vol. 25, no. 16, s. 2078–2079.

[34] LI, Heng. Improving SNP discovery by base alignment quality. *Bioinformatics*. 2011, vol. 27, no. 8, s. 1157–1158.

[35] LI, Heng; HOMER, Nils. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics*. 2010, vol. 11, no. 5, s. 473–483.

[36] METZKER, Michael L. Sequencing technologies - the next generation. *Nature Reviews Genetics*. 2010, vol. 11, s. 31–46.

[37] MOFFAT, Alistair; TURPIN, Andrew. On the implementation of minimum redundancy prefix codes. *IEEE Transactions on Communications*. 1997, vol. 45, no. 10, s. 1200–1207.

[38] NELSON, Mark; GAILLY, Jean-Loup. *The Data Compression Book*. 2nd ed. New York : M&T Books, 1996. 541pp. ISBN 1-55851-434-1.

[39] NIEDRINGHAUS, Thomas, et al. Landscape of next-generation sequencing technologies. *Anal. Chem.*. 2011, vol. 83, no. 12, s. 4327–4341.

[40] PAVLOV, Igor. *LZMA SDK* [online]. c2012 [cit. 2012-07-09]. Dostupné z: `<http://www.7-zip.org/sdk.html>`.

[41] PETTERSSON, Erik; LUNDEBERG, Joakim; AHMADIAN, Afshin. Generations of sequencing technologies. *Genomics*. 2009, vol. 93, no. 2, s. 105–111.

[42] SAKIB, Muhammad, et al. Improving Transmission Efficiency of Large Sequence Alignment/Map (SAM) Files. *PloS one*. 2011, vol. 6, no. 12, e28251.

[43] SALOMON, David; MOTTA, Giovanni. *Handbook of Data Compression*. 5th ed. Springer, 2010. 1361pp. ISBN 978-1-84882-902-2.

[44] SANGER, F.; NICKLEN, S.; COULSON, A. DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*. 1977, vol. 74, no. 12, s. 5463.

[45] SCHUSTER, Stephan C., et al. Complete Khoisan and Bantu genomes from southern Africa. *Nature*. 2010, vol. 463, no. 7283, s. 943–947.

[46] SEWARD, Julian. *bzip2* [online]. c2012 [cit. 2012-07-09]. Dostupné z: `<http://www.bzip.org/>`.

[47] STEIN, Lincoln D., et al. The case for cloud computing in genome informatics. *Genome Biology*. 2010, vol. 11, no. 5, s. 207.

[48] THUDI, Mahendar, et al. Current state-of-art of sequencing technologies for plant genomics research. *Briefings in Functional Genomics*. 2012, vol. 11, no. 1, s. 3–11.

[49] VITTER, J. et al. Arithmetic Coding for Data Compression. *Information Processing and Management*. 1994, s. 749–763.

[50] VOELKERDING, Karl; DAMES, Shale; DURTSCHI, Jacob. Next-generation sequencing: from basic research to diagnostics. *Clinical chemistry*. 2009, vol. 55, no. 4, s. 641–658.

[51] WILLIAMS, Hugh E.; ZOBEL, Justin. Compressing Integers for Fast File Access. *The Computer Journal*. 1999, vol. 42, no. 3, s. 193–201.

[52] ZIV, Jacob; LEMPEL, Abraham. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*. 1977, vol. 23, no. 3, s. 337–343.

A. Popis priloženého média

/

- `dip.pdf` Elektronická verzia tohto textu

/bin/

Binárna distribúcia programov SamZip, samtools a Savant. Tento adresár obsahuje nasledujúce podadresáre:

- `samzip` Distribúciu aplikácie SamZip
- `samtools` Spustiteľné súbory aplikácie samtools pre Windows a pre Unix
- `savant` Distribúcia aplikácie Savant

/src/

Zdrojové kódy. Tento adresár obsahuje nasledujúce podadresáre:

- `samzip` Zdrojové kódy aplikácie SamZip
- `savant_samzip_plugin` Zdrojové kódy zásuvného modulu pre aplikáciu Savant, ktorý umožňuje zobrazovanie záznamov v komprimovanom SamZip súbore

/data/

Príklad súboru vo formátoch SAM, BAM a SamZip.

Zdrojom dát v tomto súbore je `<http://manuelcorpas.com/tag/bam/>`.

/test/

Skripty, ktoré sme používali pri testovaní kompresných metód v kapitole 8.

B. Užívateľská príručka – základná práca

Poznámka B.1. Povinné argumenty v popisoch príkazov budeme uvádzať do zložených zátvoriek. Nepovinné argumenty budeme uvádzať do hranatých zátvoriek.

B.1 Aplikácia samtools

Štandardne je pre prácu so súbormi vo formátoch SAM a BAM používaná aplikácia samtools[1]. V tejto sekcii popíšeme základné funkcie tejto aplikácie, ktorých znalosť je vhodná pre prácu s aplikáciou SamZip.

Aplikáciu samtools spustíme z príkazového riadku príkazom

```
samtools {príkaz} {argumenty príkazu}
```

B.1.1 Príkaz `view`

Príkaz `view` umožňuje konverziu súborov vo formátoch SAM a BAM:

```
samtools view [prepínače] {vstupný súbor}
```

Najdôležitejšie prepínače príkazu `view` sú:

- `-h` Zapíše na výstup aj hlavičku vstupného súboru (implicitne zapisuje len SAM záznamy).
- `-S` Indikuje, že vstupný súbor je vo formáte SAM.
- `-b` Na výstup zapíše dáta vo formáte BAM.

Poznámka B.2. Príkaz `view` zapisuje dáta na štandardný výstup. Ak potrebujeme výstupné dáta uložiť do súboru, je potrebné presmerovať výstup programu samtools do tohto súboru.

Zobrazenie obsahu BAM súboru

```
samtools view {vstupný BAM súbor}
```

[1] <http://samtools.sourceforge.net/>

Konverzia BAM → SAM

```
samtools view -h {vstupný BAM súbor} > {výstupný SAM súbor}
```

Konverzia SAM → BAM

```
samtools view -Sb {vstupný SAM súbor} > {výstupný BAM súbor}
```

Poznámka B.3. Vstupný SAM súbor musí obsahovať slovník referenčných sekvencií.

B.1.2 Príkaz `sort`

Aplikácia SamZip vyžaduje, aby boli vstupné súbory usporiadané v usporiadaní `coordinate`. To je možné dosiahnuť príkazom

```
samtools sort {vstupný bam súbor} {výstupný (utriedený) bam súbor}
```

B.1.3 Príkaz `faidx`

Pri používaní kompresie voči referenčnej sekvencii je potrebné k referenčnej sekvencii vo formáte FASTA vytvoriť index. Index FASTA súboru je možné vytvoriť príkazom

```
samtools faidx {FASTA súbor}
```

Tento príkaz vytvorí pre vstupný súbor `in.fa` indexový súbor `in.fa.fai`.

B.2 Popis príkazovej riadky aplikácie SamZip

Aplikáciu SamZip je možné spustiť z príkazového riadku príkazom

```
java [argumenty JVM] -jar samzip.jar {príkaz} [argumenty príkazu]
```

alebo príkazom

```
java [argumenty JVM] -cp "samzip.jar:lib/*" samzip.cmd.Main {príkaz}
     [argumenty príkazu]
```

Poznámka B.4. Pozor! Na systémoch rodiny Windows je potrebné používať v argumente `-cp` namiesto oddeľovača `:`, oddeľovač `;`.

Tieto varianty sú navzájom ekvivalentné. V ďalšom texte budeme kvôli úspore miesta a prehľadnosti uvádzať zápis, v ktorom je volanie Javy nahradené reťazcom `samzip`:

```
samzip {príkaz} [argumenty príkazu]
```

Príkaz môže byť jeden z nasledujúcich:

- `compress` Kompresia vstupného SAM/BAM súboru
- `decompress` Dekompresia komprimovaného súboru
- `test` Test kompresie a dekompresie

Všetky príkazy majú spoločné nasledujúce argumenty:

- `--stringency=hodnota` Nastaví úroveň validácie pri čítaní a zapisovaní SAM/BAM súborov. Dovolené hodnoty sú: `STRICT` (po narazení na chybu skončí výnimkou), `LENIENT` (po narazení na chybu vypíše túto chybu na výstup) a `SILENT` (nevypisuje chyby). Implicitná hodnota je `STRICT`.
- `-D{kľúč}={hodnota}` Nastavenie parametra SAM procesora. Pár `(kľúč, hodnota)` je predaný každému použitému SAM procesoru. Interpretácia tohto argumentu záleží na implementácii konkrétneho SAM procesoru. Napríklad parameter `-Dref=hg19.fa` nastaví referenčnú sekvenciu `hg19.fa` pre SAM procesor, ktorý komprimuje rozdiely oproti referenčnej sekvencii.

Odporúčané argumenty behového prostredia Javy

```
java -Xms256m -Xmx2048m -server
```

Argumenty `-Xms256m` a `-Xmx2048m` určujú minimálnu a maximálnu hodnotu pamäte dostupnú pre behové prostredie Javy na 256 MB resp. 2048 MB. Spotreba pamäte pri kompresii záleží na veľkosti komprimovaného bloku, použitých kompresných metódach a hlavne na počte vlákien použitých pri kompresii. V prípade, že pri behu programu nastane výnimka `OutOfMemoryError`, je potrebné tieto hodnoty zväčšiť.

Argument `-server` nastaví JIT[2] kompilátor Javy na serverový mód. Toto prináša trochu lepší výkon ako pri použití implicitného JIT kompilátora[3].

B.2.1 Príkaz `compress`

Príkaz `compress` umožňuje kompresiu vstupných súborov vo formátoch SAM alebo BAM. Tieto súbory musia byť pred kompresiou usporiadané (viď sekcia

[2] Just In Time compilation = kompilácia Java kódu do natívneho kódu za behu programu

[3] Serverový mód je optimalizovaný pre aplikácie od ktorých je požadovaný maximálny výkon, a ktoré zvyčajne bežia dlhšiu dobu

B.1.2) a v ich hlavičke sa musí nachádzať slovník použitých referenčných sekvencií.

Príkaz `compress` má nasledujúce povinné argumenty:

- `--in={súbor}` Vstupný SAM/BAM súbor
- `--out={súbor}` Výstupný (komprimovaný) SamZip súbor
- `-p={súbor}` Súbor s kompresným profilom

Ďalej je možné uviesť nepovinné argumenty:

- `-b={číslo}` Nastavenie počtu záznamov, ktoré budú uložené v jednom komprimovanom bloku (Implicitná hodnota: 1000).
- `-t={číslo}` Nastavenie počtu vlákien, ktoré budú používané pri kompresii. Jedno ďalšie vlákno bude použité na čítanie vstupu a zápis komprimovaných dát na pevný disk (Implicitná hodnota: počet procesorov v systéme).
- `-s` Pri kompresii sa budú počítať veľkosti jednotlivých komprimovaných prúdov a časová náročnosť použitých SAM procesorov.

Vytvorenie SamZip súboru

```
samzip compress --in={vstupný SAM/BAM súbor} --out={výstupný SamZip
    súbor} -p={profil}
```

B.2.2 Príkaz `decompress`

Na dekompresiu súborov vo formáte SamZip slúži príkaz `decompress`.

Príkaz má jediný povinný atribút `--in={súbor}` pre nastavenie vstupného súboru vo formáte SamZip. Bez uvedenia ďalších argumentov, príkaz `decompress` dekomprimuje SAM záznamy na štandardný výstup bez hlavičky SAM súboru (rovnaké správanie ako pri zobrazovaní BAM súboru v samtools).

Ďalej je možné uviesť nepovinné argumenty:

- `--out={súbor}` Nastavenie výstupného súboru pre dekomprimované dáta. Dekomprimované dáta sú implicitne zapisované na štandardný výstup.
- `-q={dotaz}` Dekompresia záznamov, ktoré splňujú podmienku uvedenú v dotaze.
- `-b` Výstupné dáta budú vo formáte BAM. Ak je uvedený tento argument, musí byť uvedený aj argument `--out` pre nastavenie výstupného súboru. Implicitne sú výstupné dáta vo formáte SAM.
- `-h` Na výstup zapíše aj hlavičku SAM súboru. Implicitne zapisuje len SAM záznamy.

- `-H` Na výstup zapíše len hlavičku SAM súboru. V prípade uvedenia tohto argumentu nie sú efektívne prepínače `-q`, `-b`, `-h` a `-P`.
- `-P` Na výstup zapíše kompresný profil, ktorý bol použitý pri kompresii vstupného súboru. V prípade uvedenia tohto argumentu nie sú efektívne prepínače `-q`, `-b`, `-h` a `-H`.

Dotazy v argumente `-q` môžu byť v jednom z nasledujúcich formátov:

- `ref:štart:koniec` Zapíše na výstup SAM záznamy, ktoré pokrývajú referenčnú sekvenciu s identifikátorom `ref` na pozíciach z intervalu `[štart, koniec]`. Príklad: `chr20:60030:60100`.
- `ref:štart+počet` Dekomprimuje maximálne `počet` záznamov, ktoré začínajú prvým záznamom, ktorý pokrýva referenčnú sekvenciu `ref` na pozícii `štart`. Príklad: `chr20:60030+100`.

Zobrazenie obsahu SamZip súboru

```
samzip decompress --in={vstup}
```

Konverzia SamZip → SAM

```
samzip decompress -h --in={vstupný SamZip súbor} --out={výstupný SAM
    súbor}
```

Konverzia SamZip → BAM

```
samzip decompress -b --in={vstupný SamZip súbor} --out={výstupný BAM
    súbor}
```

Dekompresia časti súboru

```
samzip decompress --in={vstupný SamZip súbor} -q={dotaz}
```

B.2.3 Príkaz `test`

Príkaz `test` slúži na otestovanie kompresie a dekompresie. Test spočíva v kompresii vstupného súboru a následnej dekompresii vytvoreného komprimovaného súboru. Implicitne počíta veľkosti jednotlivých komprimovaných prúdov a časovú náročnosť použitých SAM procesorov (rovnako ako keby bol uvedený argument `-s` pri príkaze `compress`). Voliteľne je možné pôvodný a dekomprimovaný súbor porovnať (argument `-d`).

Príkaz `test` má nasledujúce povinné argumenty:

- `--in={súbor}` Vstupný súbor
- `--out={súbor}` Výstupný (komprimovaný) súbor
- `-p={súbor}` Súbor s kompresným profilom

Ďalej je možné uviesť nepovinné argumenty:

- `--dec={súbor}` Výstupný súbor pre dekomprimované dáta. Implicitne sú dekomprimované dáta vymazané.
- `-b={číslo}` Nastavenie veľkosti bloku. Má rovnaký význam ako pri príkaze `compress`.
- `--bam` Výstupné dáta budú vo formáte BAM. Má rovnaký význam ako argument `-b` príkazu `decompress`.
- `-t={číslo}` Nastavenie počtu vlákien, ktoré budú používané pri kompresii. Má rovnaký význam ako pri príkaze `compress`.
- `-d` Nakoniec dekompresie porovná pôvodný a dekomprimovaný súbor.

B.3 Štandardné profily

S aplikáciou SamZip dodávame kompresné profily `Best` a `Optimal`, ktoré sme popísali v sekcii 8.8. Tieto profily je možné uviesť v argumente `-p` príkazov `compress` a `test` ako `@best` resp. `@optimal`.

Kompresia profilom `best` bez použitia referenčnej sekvencie

```
samzip compress --in={vstupný SAM/BAM súbor} --out={výstupný SamZip
    súbor} -p=@best
```

Analogicky je možné využiť kompresný profil `optimal`.

Kompresia profilom `best` s použitím referenčnej sekvencie

```
samzip compress --in={vstupný SAM/BAM súbor} --out={výstupný SamZip
    súbor} -p=@best -Dref={referenčná sekvencia}
```

Analogicky je možné využiť kompresný profil `optimal`.

Poznámka B.5. V tomto prípade nemyslíme pod pojmom referenčná sekvencia sekvencie, ku ktorým sú zarovnané segmenty vo vstupnom SAM súbore. V tomto kontexte budeme vždy pod pojmom referenčná sekvencia myslieť fyzický súbor vo FASTA formáte, ktorý obsahuje všetky tieto sekvencie (tj. všetky referenčné sekvencie, ku ktorým je zarovnaný SAM súbor).

V adresári s referenčnou sekvenciou `názov.fa` sa musí náchádzať aj index tejto sekvencie s názvom `názov.fa.fai` (viď sekcia B.1.3).

Poznámka B.6. Pri testoch sme používali nasledujúce referenčné sekvencie:

- Pre súbory z projektu „1000 genomes project“ a pre súbor `Manuel` sme používali referenčnú sekvenciu `NCBI37/hg19`[4].
- Pre súbor `KB1454` sme používali referenčnú sekvenciu `NCBI36/hg18`[5].
- Pre súbor `Mouse` sme používali referenčnú sekvenciu `NCBIM37`[6].
- Pre súbor `Ecoli` sme používali referenčnú sekvenciu `MG1655`[7].

[4] <ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/>

[5] <http://hgdownload.cse.ucsc.edu/goldenPath/hg18/bigZips/>

[6] <ftp://ftp-mouse.sanger.ac.uk/ref/>

[7] <ftp://ftp.ncbi.nih.gov/genomes/Bacteria/Escherichia_coli_K_12_substr_
_MG1655_uid57779/>

C. Užívateľská príručka – pokročilý užívateľ

V prípade, že užívateľovi nevyhovujú kompresné profily, ktoré sú štandardne dodávané s aplikáciou SamZip môže si vytvoriť vlastný kompresný profil.

Kompresný profil je XML súbor, ktorý riadi celý proces kompresie. V tejto sekcii popíšeme jeho formát a uvedieme popis kompresných metód a SAM procesorov, ktoré sú súčasťou aplikácie SamZip.

Kompresný profil obsahuje dve sekcie:

- **Definície kompresných metód.** Táto sekcia môže obsahovať ľubovoľné množstvo definícii kompresných kodekov. Každá definícia kompresného kodeku umožňuje nastavenie parametrov daného kodeku.
- **Definície SAM procesorov.** V tejto sekcii je uvedená postupnosť definícií SAM procesorov, ktoré tvoria kompresnú pipeline. Každá definícia SAM procesora môže obsahovať nastavenie jeho parametrov a používaných kodekov.

C.1 Definície kompresných metód

Sekcia definícii kompresných metód je obsahom elementu `methods`. Môže obsahovať ľubovoľné množstvo definícii kompresných kodekov, z ktorých každá je obsahom elementu `codec`. Element `codec` má nasledujúce atribúty:

- `class` Názov triedy kompresného kodeku. Ak názov začína znakom `@`, tak sa trieda s daným názvom hľadá v štandardne dodávaných triedach. Nové triedy je možné špecifikovať uvedením celého názvu triedy (tj. aj s balíčkom, v ktorom sa nachádza).
- `id` (nepovinný atribút) Obsahuje identifikátor kodeku. Každý kodek, ktorý chceme referencovať v ďalších častiach kompresného profilu musí mať nastavený identifikátor.
- `shared` (nepovinný atribút) Nastavenie kodeku ako zdieľaného. Nezdieľaný kodek vytvára vždy nové inštancie enkodéra a dekodéra, ale zdieľaný vytvorí len jednu. Zdieľaný kodek umožňuje uložiť výstup viacerých SAM procesorov do jedného zdieľaného prúdu. Atribút `shared` môže obsahovať hodnotu `true` alebo `false`. Implicitne obsahuje hodnotu `false`.

Definíciu kodeku je možné referencovať použitím elementu `<codec ref="id kodeku"/>`.

Telo definície kompresného kodeku môže obsahovať parametre kodeku a zreťazené kompresné kodeky.

Parametre kodeku

Parametre kompresného kodeku je možné nastaviť v elemente `param` s atribútmi `key`, ktorý reprezentuje kľúč parametra a `value`, ktorý reprezentuje hodnotu parametra.

Všetky kodeky umožňujú nastavenie parametra `mode` pre nastavenie dátového módu. Implicitne je dátový mód odvodený automaticky podľa toho, aké dáta je potrebné ukladať. Po uvedení parametra `mode` v definícii kodeku je hodnota tohto parametra použitá namiesto implicitného dátového módu. Parameter `mode` môže nadobúdať hodnoty `BYTE`, `UBYTE`, `SHORT`, `USHORT`, `INT` a `UINT`. Rozsah hodnôt jednotlivých dátových módov je uvedený v tabuľke 7.1.

Zreťazené kodeky

Zreťazené kodeky (viď sekcia 7.4) je môžné nastaviť uvedením definície nového kompresného kodeku alebo referencovaním už definovaného kodeku. Ak môže mať kodek zreťazených viacero metód, je možné v atribúte `for` elementu `codec` uviesť názov pomenovaného zreťazenia. Ak tento atribút nie je uvedený, tak je kodek zreťazený ako implicitný (tj. ak má definovaný kodek povinné len jedno zreťazenie, je zreťazený kodek nastavený pre toto zreťazenie; ak neobsahuje žiadne povinné zreťazenie, je zreťazený ako výstupný prúd).

C.2 Definície SAM procesorov

Sekcia definícii SAM procesorov je obsahom elementu `processing`. Obsahuje postupnosť definícií SAM procesorov. Poradie, v akom sú uvedené v kompresnom profile určuje poradie, v akom budú jednotlivé SAM procesory používané pri kompresii a dekompresii.

Definícia SAM procesora je obsahom elementu `processor`. Rovnako ako pri kompresnom kodeku je potrebné uviesť atribút `class`, ktorý obsahuje názov triedy SAM procesora.

Telo elementu `processor` môže obsahovať ľubovoľné množstvo parametrov, ktoré je možné špecifikovať rovnako ako u definície kompresného kodeku (element `param` s atribútmi `key` a `value`). Ďalej je možné nastaviť ľubovoľné množstvo kompresných kodekov, rovnako ako pri nastavovaní zreťazených kodekov v tele definície kompresného kodeku (element `codec`).

C.3 Príklad kompresného profilu

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE samzip SYSTEM "samzipProfile.dtd">
3 <samzip>
```

```
 4 <methods>
 5    <codec id="LZMA" class="@LZMACodec"/>
 6    <codec id="VByte" class="@VByteCodec"/>
 7    <codec id="EliasDelta" class="@EliasDeltaCodec"/>
 8    <codec id="Huffman" class="@HuffmanCodec"/>
 9    <codec id="RLE" class="@RunLengthEncodingCodec">
10       <param key="algorithm" value="universal"/>
11       <codec for="value" class="@HuffmanCodec"/>
12       <codec for="length" ref="EliasDelta"/>
13    </codec>
14    <codec id="DeltaHuffman" class="@DeltaEncodingCodec">
15       <codec class="@HuffmanCodec"/>
16    </codec>
17    <codec id="VByteLZMA" class="@VByteCodec">
18       <codec ref="LZMA"/>
19    </codec>
20    <codec id="sharedFlag" class="@HuffmanCodec" shared="true"/>
21 </methods>
22 <processing>
23  <processor class="@FieldProcessor">
24      <param key="field" value="FLAG"/>
25      <codec ref="Huffman"/>
26  </processor>
27  <processor class="@FieldProcessor">
28      <param key="field" value="POS"/>
29      <codec ref="DeltaHuffman"/>
30  </processor>
31  <processor class="@FieldProcessor">
32      <param key="field" value="REF"/>
33      <codec ref="RLE"/>
34  </processor>
35  <processor class="@NextSegmentProcessor">
36      <codec for="ref" ref="EliasDelta"/>
37      <codec for="ref.flag" ref="sharedFlag"/>
38      <codec for="pos" ref="VByteLZMA"/>
39      <codec for="tempLen" ref="Huffman"/>
40  </processor>
41  <processor class="@TagProcessor">
42      <param key="tag" value="BQ"/>
43      <codec for="flag" ref="sharedFlag"/>
44      <codec ref="RLE"/>
45  </processor>
46 </processing>
47 </samzip>
```

Riadok 1 obsahuje deklaráciu XML dokumentu. Ďalší riadok obsahuje definíciu používaného DTD. Na riadku 3 nasleduje koreňový element XML dokumentu.

Riadky 4 až 21 tvoria sekciu definícií kompresných metód. Na riadku 5 je definovaný kodek metódy LZMA, ktorý nemá zreťazený žiadny ďalší kodek. Riadky 9 až 13 obsahujú definíciu kompresného kodeku behového kódovania. Tento kodek

má definovaný jeden parameter na riadku 10 a dva zreťazené kompresné kodeky – na riadku 11 definíciu nového kodeku Huffmanovho kódovania pre výstup hodnôt behov, a na riadku 12 referencovanie kodeku Eliasovho delta kódovania, ktorý bol definovaný na riadku 7, pre výstup dĺžok behov. Na 20 riadku je uvedený príklad definície zdieľaného kodeku.

Riadky 22 až 46 tvoria sekciu definícií SAM procesorov. Na riadku 23 až 26 je uvedený príklad procesora, ktorý má na starosti spracovanie povinného poľa `FLAG`. Využije k tomu Huffmanovo kódovanie. Riadky 35 až 40 sú definíciou procesora, ktorý umožňuje nastavenie viacerých kompresných metód. Procesor definovaný na riadkoch 41 až 45 má za úlohu spracovanie nepovinného poľa `BQ`. Ku kompresii poľa príznakov výskytu tohto poľa (`for="flag"`) využije zdieľaný kompresný kodek definovaný na riadku 20. Kodek referencovaný na riadku 44 nemá špecifikovaný atribút `for`, preto bude použitý ako implicitný výstup tohto SAM procesora.

Definované SAM procesory budú spracovávať vstup postupne. Začnú procesorom, ktorý spracováva pole `FLAG`. Skončia procesorom, ktorý spracováva nepovinný atribút `BQ`.

C.4 Štandardné kodeky

V tejto sekcii uvedieme kompresné kodeky, ktoré sú štandardnou súčasťou aplikácie SamZip. Pripomíname, že pre ich použitie je potrebné v parametri `class` tagu `codec` uviesť znak `@` pred názvom triedy konkrétneho kodeku.

Ak v popise kodeku neuvádzame popis parametrov, potom daný kodek neponúka žiadne konfigurovateľné parametre. Všetky parametre kompresných kodekov sú nepovinné – majú vždy definovanú implicitnú hodnotu.

Ak v popise kodeku uvádzame popis zreťazených kodekov, potom je povinné špecifikovať tieto zreťazenia v definícii kodeku. Ak popis povinných zreťazení neuvádzame, potom daný kodek nevyžaduje žiadne povinné zreťazenia. Pre každý kodek je však možné špecifikovať implicitné zreťazenie, ktoré zreťazí kodek ako výstupný prúd.

`ArithmeticCodec`

Implementácia adaptívneho aritmetického kódovania (viď sekcia 3.4.2). Zdrojom implementácie je `<http://www.colloquial.com/ArithmeticCoding/>`.

Podporuje len dátový mód `UBYTE`.

`BZip2Codec`

Implementácia kompresnej metódy bzip2 (viď sekcia 3.6). Zdrojom implementácie je `<http://code.google.com/p/jbzip2/>`.

Podporuje len dátový mód `UBYTE`.

DeltaCodec

Implementácia delta kódovania (viď sekcia 3.2.2).

Parametre

- `unsigned` Ak nadobúda hodnotu `true`, tak metóda predpokladá, že vstupné dáta tvoria neklesajúcu postupnosť. Implicitná hodnota je `false`.

Zreťazené kodeky

- `default` Kodek, ktorý bude použitý pre kódovanie výstupu delta kódovania.

EliasDeltaCodec

Implementácia Eliasovho delta kódovania (viď sekcia 3.3.3).

EliasGammaCodec

Implementácia Eliasovho gamma kódovania (viď sekcia 3.3.3).

ExpectedValueCodec

Implementácia delta kódovania, ktoré kóduje rozdiely k hodnote aritmetického priemeru alebo k hodnote mediánu (viď sekcia 3.2.2).

Parametre

- `statistics` Nastavenie hodnoty, vzhľadom ku ktorej sú počítané rozdiely v delta kódovaní. Môže nadobúdať hodnotu `ARITHMETIC_MEAN` pre použitie aritmetického priemeru alebo `MEDIAN` pre použitie mediánu hodnôt na vstupe. Implicitná hodnota je `ARITMETIC_MEAN`.

Zreťazené kodeky

- `default` Kodek, ktorý bude použitý pre kódovanie výstupu delta kódovania.

FibonacciCodec

Implementácia Fibonacciho kódovania (viď sekcia 3.3.4).

GolombCodec

Implementácia Golombovho kódovania (viď sekcia 3.3.5).

Parametre

- `adaptive` Ak nadobúda hodnotu `true`, parameter m Golombovho kódu bude vypočítaný pri kompresii ako aritmetický priemer vstupných hodnôt. Implicitná hodnota je `true`.
- `m` Nastavenie hodnoty parametra Golombovho kódu. Hodnota tohto parametra je použitá len v prípade, keď hodnota parametra `adaptive` je nastavená na `false`. Implicitná hodnota je 1024.

`GZipCodec`

Implementácia metódy gzip (viď sekcia 3.5.1). Zdrojom implementácie je štandardná implementácia metódy dostupná v Jave.

Podporuje len dátový mód `UBYTE`.

`HuffmanCodec`

Implementácia kanonického Huffmanovho kódovania (viď sekcia 3.4.1).

`LZMACodec`

Implementácia metódy LZMA (viď sekcia 3.5.2). Zdrojom implementácie je `<http://www.7-zip.org/>`.

Parametre

- `dictionarySize` Nastavenie veľkosti slovníku, ktorý bude používaný pri kompresii. Veľkosť je udávaná v bajtoch. Implicitná hodnota je 1 048 576 (1 MB).

Podporuje len dátový mód `UBYTE`.

`MoveToFrontCodec`

Implementácia kódovania s presunom na začiatok (viď sekcia 3.2.3).

Zreťazené kodeky

- `default` Kodek, ktorý bude použitý pre kódovanie výstupu kódovania s presunom na začiatok.

`NullCodec`

Kodek, ktorý neukladá žiadne dáta do komprimovaného súboru. Použitím tohto kodeku je možné dosiahnuť odstránenie dát, ktoré si užívateľ nepraje ukladať.

RawCodec

Kodek, ktorý ukladá dáta bez kompresie. Tento kodek je vhodný napr. pre situácie kedy potrebujeme komprimovať dáta kodekom, ktorý podporuje len dátový mód `UBYTE`.

Príklad C.1. Chceme komprimovať celočíselné dáta so znamienkom pomocou metódy LZMA. V tomto prípade je najvhodnejším dátovým módom mód `INT`. Metóda LZMA však podporuje len mód `UBYTE`. Je preto možné použiť kodek `RawCodec` s nastaveným dátovým módom `INT` a výstupom kódovaným metódou LZMA:

```
<codec class="@RawCodec">
  <param key="mode" value="INT"/>
  <codec class="@LZMACodec"/>
</codec>
```

RiceCodec

Implementácia Riceovho kódovania (viď sekcia 3.3.5).

Parametre

- `adaptive` Ak nadobúda hodnotu `true`, parameter m Riceovho kódu bude vypočítaný pri kompresii ako aritmetický priemer vstupných hodnôt. Implicitná hodnota je `true`.
- `m` Nastavenie hodnoty parametra Riceovho kódu. Hodnota tohto parametra je použitá len v prípade, keď hodnota parametra `adaptive` je `false`. Implicitná hodnota je 8.

RunLengthEncodingCodec

Implementácia behového kódovania (viď sekcia 3.2.1). Poskytuje 2 rôzne algoritmy behového kódovania – RLE a RLEU.

Parametre

- `algorithm` Určuje ktorý algoritmus behového kódovania bude použitý. Môže nadobúdať hodnoty `stream` pre RLE alebo `universal` pre RLEU. Implicitná hodnota je `universal`.

Zreťazené kodeky pre algoritmus `universal`

- `value` Kodek pre kódovanie hodnôt behov.
- `length` Kodek pre kódovanie dĺžok behov.

- `both` Kodek pre kódovanie hodnôt aj dĺžok behov. Pre hodnoty aj dĺžky behov bude použitý rovnaký prúd.

Pre algoritmus RLEU je povinná špecifikácia kodekov buď pre pomenované zreťazenia `value` a `length` alebo pre zreťazenie `both`.

`VByteCodec`

Implementácia VByte kódovania (viď sekcia 3.3.6).

C.5 Štandardné SAM procesory

V tejto sekcii uvedieme SAM procesory, ktoré sú štandardnou súčasťou aplikácie SamZip. Pripomíname, že pre ich použitie je potrebné v parametri `class` tagu `processor` uviesť znak `@` pred názvom triedy konkrétneho SAM procesoru.

Ak v popise SAM procesora neuvádzame popis parametrov, potom daný procesor neponúka žiadne konfigurovateľné parametre. Pri popise každého povinného parametra uvedieme, že je povinný. Pri popise nepovinných parametrov vždy uvedieme implicitnú hodnotu týchto parametrov.

Ak to nebude vyslovene spomenuté inak, všetky kompresné kodeky, ktoré uvedieme v popise SAM procesorov sú povinné.

`FieldProcessor`

Základný SAM procesor, ktorý slúži pre spracovanie povinných polí SAM súboru. Pole, ktoré má tento procesor spracovávať, je potrebné uviesť v parametri `field`.

Parametre

- `field` Povinné pole SAM súboru, ktoré bude spracovávať tento procesor. Môže nadobúdať hodnoty `QNAME`, `FLAG`, `RNAME`, `POS`, `MAPQ`, `CIGAR`, `RNEXT`, `PNEXT`, `TLEN`, `SEQ` alebo `QUAL`. Tento parameter je povinný.

- `arraycode` Nastavenie spôsobu kompresie reťazcov pri reťazcových poliach (`QNAME`, `CIGAR`, `SEQ` a `QUAL`). Môže nadobúdať hodnoty:

  - `delimiter/delim` Obsah jednotlivých reťazcov bude oddelený znakom `delim`. Kód znaku `delim` je uvedený v hexadecimálnej sústave. Napríklad pri nastavení hodnoty `delimiter/0x0A` bude použitý na oddeľovanie jednotlivých komprimovaných reťazcov znak konca riadka.

  - `length` Dĺžky jednotlivých reťazcov budú uložené do samostatného prúdu. Pri použití tejto hodnoty je potrebné uviesť kodek `length` pre kompresiu hodnôt týchto dĺžok.

– `sequence` Dĺžka jednotlivých reťazcov nebude ukladaná. SAM procesor bude pri dekompresii predpokladať, že konkrétny reťazec má rovnakú dĺžku ako pole `SEQ`. Pole `SEQ` musí byť spracované v predchádzajúcich procesoroch v kompresnej pipeline.

Implicitnou hodnotou je `delimiter/0x0A`.

Kompresné kodeky

- `implicitný` Kodek, ktorý bude použitý pre kompresiu hodnôt spracovávaného poľa.
- `length` Kodek, ktorý bude použitý pre kompresiu hodnôt dĺžok reťazcov pri nastavení hodnoty parametra `arraycode` na `length`. Implicitne používa dátový mód `UINT`.

Implicitný kodek implicitne používa nasledujúce dátové módy pre jednotlivé polia:

Pole	Dátový mód
QNAME	UBYTE
FLAG	USHORT
RNAME	UINT
POS	UINT
MAPQ	UBYTE
CIGAR	UBYTE
SEQ	UBYTE
RNEXT	UINT
PNEXT	UINT
TLEN	INT
QUAL	UBYTE

TagProcessor

Základný SAM procesor, ktorý slúži pre spracovanie nepovinných polí SAM súboru. Dvojznakový tag nepovinného poľa, ktoré má tento procesor spracovávať, je potrebné uviesť v parametri `tag`.

Parametre

- `tag` Dvojznakový tag nepovinného poľa, ktoré bude spracovávať tento procesor. Tento parameter je povinný.
- `arraycode` Má rovnaký význam ako pri SAM procesore `FieldProcessor`.

Kompresné kodeky

- `implicitný` Kodek, ktorý bude použitý pre kompresiu hodnôt spracováva-

ného poľa. Ak je spracovávané pole celočíselného typu, implicitne používa dátový mód `INT`, inak používa dátový mód `UBYTE`.

- `length` Kodek, ktorý bude použitý pre kompresiu hodnôt dĺžok reťazcov pri nastavení hodnoty parametra `arraycode` na `length`. Implicitne používa dátový mód `UINT`.
- `flag` Kodek, ktorý bude použitý pre kompresiu poľa príznakov, ktoré udávajú či konkrétny záznam obsahuje spracovávané nepovinné pole. Tento kodek nie je potrebné uviesť v prípade, že spracovávané pole je reťazec alebo pole hodnôt a SAM procesor má nastavený parameter `arraycode` na `delimiter` alebo `length`. Implicitne používa dátový mód `UBYTE`.

`RawTagProcessor`

SAM procesor, ktorý slúži pre uloženie nepovinných polí SAM súboru v textovej podobe.

Parametre

- `excluded` Zoznam tagov nepovinných polí, ktoré nebudú spracované (uložené). Jednotlivé tagy je potrebné oddeliť medzerou. Automaticky nie sú spracované polia, ktoré už boli spracované v niektorom z predchádzajúcich SAM procesorov v kompresnej pipeline.

Kompresné kodeky

- `implicitný` Kodek, ktorý bude použitý pre kompresiu dát nepovinných polí. Implicitne používa dátový mód `UBYTE`.

`ReferenceSequenceProcessor`

SAM procesor, ktorý umožňuje kompresiu polí `CIGAR` a `SEQ` s využitím reálnej referenčnej sekvencie alebo vytvorenej referenčnej sekvencie. Tento SAM procesor sme využívali pri výbere kompresných metód v sekcii 8.3.

Tento SAM procesor vyžaduje, aby boli polia `RNAME` a `POS` spracované v predchádzajúcich procesoroch v kompresnej pipeline.

Parametre

- `ref` Cesta k súboru s referenčnou sekvenciou vo formáte FASTA. Táto cesta je relatívna k vstupnému súboru pri kompresii a ku komprimovanému súboru pri dekompresii. Ak nie je tento parameter uvedený, pri kompresii/dekompresii je vytvorená vlastná referenčná sekvencia z dostupných vstupných dát.

  Tento parameter je možné nastaviť aj z príkazovej riadky argumentom `-Dref={cesta k referenčnej sekvencii}`. V tomto prípade je však cesta k referenčnej sekvencii interpretovaná buď ako absolútna, ak je uvedená

absolútna cesta, alebo ako relatívna k aktuálnemu pracovnému adresáru (typicky adresár z ktorého bola spustená aplikácia SamZip).

- `calculateNM` Ak nadobúda hodnotu `true` a je používaná reálna referenčná sekvencia, pri dekompresii sa vypočíta obsah nepovinného poľa `NM`. Implicitná hodnota je `true`.
- `calculateMD` Ak nadobúda hodnotu `true` a je používaná reálna referenčná sekvencia, pri dekompresii sa vypočíta obsah nepovinného poľa `MD`. Implicitná hodnota je `true`.

Kompresné kodeky

- `seq` Kodek, ktorý bude použitý pre kompresiu dát poľa `SEQ`. Komprimované dáta obsahujú bázy, ktoré sa nezhodujú s referenčnou sekvenciou. Implicitne používa dátový mód `UBYTE`.
- `cigar` Kodek, ktorý bude použitý pre kompresiu dát poľa `CIGAR` v textovej podobe. Toto pole je poupravené tak, aby obsahovalo informácie o zhodách a nezhodách s referenčnou sekvenciou. Implicitne používa dátový mód `UBYTE`.
- `cigar.op` a `cigar.len` Kodeky, ktoré budú použité pre kompresiu operácii (`cigar.op`) a dĺžok (`cigar.len`) poľa `CIGAR`. Je povinné uviesť buď kodek `cigar`, alebo oba kodeky `cigar.op` a `cigar.len`. Kodek `cigar.op` implicitne používa dátový mód `UBYTE`, kodek `cigar.len` implicitne používa dátový mód `UINT`.

`NextSegmentProcessor`

SAM procesor, ktorý slúži pre uloženie informácií o párových čítaniach (polia `RNEXT`, `PNEXT` a `TLEN`). Tento SAM procesor sme využívali pri výbere kompresných metód v sekcii 8.5.

Tento SAM procesor vyžaduje, aby boli polia `FLAG`, `RNAME` a `POS` spracované v predchádzajúcich procesoroch v kompresnej pipeline.

Kompresné kodeky

- `ref` Kodek, ktorý bude použitý pre kompresiu hodnôt v poli `RNEXT`, ktoré sa nezhodujú s hodnotou poľa `RNAME`. Implicitne používa dátový mód `UINT`.
- `ref.flag` Kodek, ktorý bude použitý pre kompresiu poľa bitových príznakov, ktoré indikujú zhodu polí `RNEXT` a `RNAME`. Implicitne používa dátový mód `UBYTE`.
- `pos` Kodek, ktorý bude použitý pre kompresiu rozdielov hodnôt polí `POS` a `PNEXT`. Implicitne používa dátový mód `INT`.
- `tempLen` Kodek, ktorý bude použitý pre kompresiu poľa `TLEN`. Implicitne používa dátový mód `INT`.

QName1000GenomeProcessor

SAM procesor, ktorý slúži pre spracovanie poľa QNAME v SAM súboroch z projektu „1000 genomes project“. Funguje tak, že z obsahu poľa QNAME odstráni časť, ktorá sa zhoduje s nepovinným poľom RG. Preto vyžaduje, aby bolo nepovinné pole RG spracované pred týmto SAM procesorom. Tento SAM procesor sme využívali pri výbere kompresných metód v sekcii 8.6.1.

Kompresné kodeky

- implicitný Kodek, ktorý bude použitý pre kompresiu číselných hodnôt, ktoré zostanú v poli QNAME po odstránení časti zhodujúcej sa s nepovinným poľom RG. Implicitne používa dátový mód UINT.

ColorSequenceProcessor

SAM procesor, ktorý slúži pre uloženie sekvencie vo farebnom priestore pri sekvenártore SOLiD (pole CS). Využívali sme ho pri výbere kompresných metód v sekcii 8.7.1.

Pri kompresii ukladá len primér a farby odpovedajúce neznámym bázam (N) v poli SEQ. Pri dekompresii je pôvodný obsah poľa CS dopočítaný z týchto dát. Vyžaduje, aby boli polia FLAG a SEQ spracované v predchádzajúcich procesoroch v kompresnej pipeline.

Kompresné kodeky

- implicitný Kodek, ktorý bude použitý pre kompresiu priméru a farieb, ktoré odpovedajú bázam N v poli SEQ. Implicitne používa dátový mód UBYTE.

ReadGroupProcessor

SAM procesor, ktorý slúži pre spracovanie nepovinného poľa RG, ktoré obsahuje identifikátor skupiny segmentov. Zoznam týchto identifikátorov je uvedený v hlavičke SAM súboru, preto je možné ukladať len index identifikátora v tomto zozname v hlavičke.

Kompresné kodeky

- implicitný Kodek, ktorý bude použitý pre kompresiu hodnôt. Implicitne používa dátový mód UINT.

ClippedBasesRemover

SAM procesor, ktorý odstráni bázy z poľa SEQ, ktoré nie sú zarovnané s referenčnou sekvenciou (odpovedá CIGAR operácii S – soft clip). Odstráni tiež k nim odpovedajúce kvality z polí QUAL, BQ, CQ, OQ a k nim odpovedajúce čas-

ti zo sekvencie vo farebnom priestore v poli `CS`. Tento SAM procesor musí byť v kompresnej pipeline uvedený pred procesormi, ktoré spracovávajú tieto polia.

`QualityQuantizationProcessor`

SAM procesor, ktorý umožňuje kvantizáciu hodnôt kvalít v poli `QUAL`. V kompresnej pipeline musí byť uvedený pred procesorom, ktorý spracováva pole `QUAL`.

Parametre

- `level` Kvantizačný parameter. Každá hodnota kvality bude zaokrúhlená k najbližšiemu násobku tohto parametra. Implicitná hodnota je 8.

D. Rozširovanie programu SamZip

Pre rozšírenie možností aplikácie SamZip je možné vytvoriť novú kompresnú metódu alebo nový SAM procesor. V tejto sekcii popíšeme spôsob ich implementácie. Táto sekcia je určená pokročilým užívateľom, ktorý majú znalosti programovania v jazyku Java.

D.1 Kompresná metóda

Pri implementácii novej kompresnej metódy je potrebné vytvoriť implementácie abstraktných tried `EncoderMethod`, `DecoderMethod` a `CompressionCodec`. Trieda `EncoderMethod` predstavuje enkodér, trieda `DecoderMethod` zase dekodér. Trieda `CompressionCodec` predstavuje kompresný kodek, ktorý umožňuje jednoduché vytvorenie inštancií enkodéra a dekodéra s požadovanou konfiguráciou.

D.1.1 `EncoderMethod`

Abstraktná trieda `EncoderMethod` predstavuje enkodér. Je potomkom štandardného výstupného prúdu v Jave (`OutputStream`). Popis rozhrania tejto triedy je uvedený v tabuľke D.1.

Trieda `EncoderMethod` funguje podobne ako štandardný výstupný prúd. Jedinou výnimkou je, že pred jej použitím je potrebná inicializácia. Typicky prebieha práca s touto triedou nasledovne:

```
  ...
  OutputStream outputStream = ...;
  EncoderMethod enc = new KonkretnaImplementacia(...);
// nastavenie výstupného prúdu
  enc.setOutputStream(outputStream);
// nastavenie dátového módu - v tomto prípade UBYTE
  enc.setDataMode(DataMode.UBYTE);
// inicializácia
  enc.initialize();
// teraz už je možné zapisovať dáta
  enc.write(0);
  enc.write(0);
  enc.write(2);
  ...
// nakoniec je potrebné enkodér uzavrieť
  enc.close();
```

D.1.2 DecoderMethod

Abstraktná trieda `DecoderMethod` predstavuje dekodér. Je potomkom štandardného vstupného prúdu v Jave (`InputStream`). Popis rozhrania tejto triedy je uvedený v tabuľke D.2.

Trieda `DecoderMethod` funguje podobne ako štandardný vstupný prúd. Jedinou výnimkou je, že pred jej použitím je potrebná inicializácia. Typicky prebieha práca s touto triedou nasledovne:

```
  ...
  InputStream inputStream = ...;
  DecoderMethod dec = new KonkretnaImplementacia(...);
// nastavenie vstupného prúdu, ktorý obsahuje komprimované dáta
  dec.setInputStream(inputStream);
// nastavenie dátového módu - v tomto prípade UBYTE
  dec.setDataMode(DataMode.UBYTE);
// inicializácia
  dec.initialize();
// teraz už je možné čítať dáta (tu čítame až do konca prúdu)
  int res;
  while ((res = dec.read()) != dec.EOS()) {
    ...
  }
// nakoniec je vhodné dekodér uzavrieť
  dec.close();
```

D.1.3 CompressionCodec

Aby bolo možné implementovanú kompresnú metódu použiť v kompresnom profile programu SamZip, je potrebné vytvoriť implementáciu abstraktnej triedy `CompressionCodec`, ktorá predstavuje kompresný kodek. Kompresný kodek umožňuje nastavenie parametrov kompresie a vytvorenie enkodéra a dekodéra s požadovanou konfiguráciou. Popis rozhrania tejto triedy je uvedený v tabuľke D.3.

Typicky prebieha práca s touto triedou nasledovne:

```
  ...
  OutputStream outputStream = ...;
  CompressionCodec codec = new KonkretnaImplementacia(...);
  CompressionCodec codec2 = new KonkretnaImplementacia2(...);
// nastavenie parametra kompresného kodeku
  codec.setPreference("parameter", "hodnota");
// zreťazenie kodeku codec2 ako implicitného pre kodek codec
  codec.addCodec("default", codec2);
// nastavenie dátového módu - v tomto prípade UBYTE
  codec.setDataMode(DataMode.UBYTE);
// vytvorenie novej inštancie enkodéra
  EncoderMethod enc = codec.newEncoderMethod(outputStream);
// teraz už je možné komprimovať dáta pomocou získanej inštancie
```

```
// enkodéra
  enc.write(0);
  enc.write(2);
  ...
// nakoniec je potrebné enkodér uzavrieť
  enc.close();
```

Aby bolo možné automatické vytvorenie objektov kompresných kodekov, je potrebné aby ich implementácie obsahovali konštruktor bez parametrov.

D.2 SAM procesor

Pri implementácii nového SAM procesora je potrebné vytvoriť implementáciu abstraktnej triedy `SAMProcessor`. Popis rozhrania tejto triedy je uvedený v tabuľke D.4.

Príklad volania SAM procesorov v kompresnej pipeline pri kompresii bloku SAM záznamov:

```
// blok SAM záznamov zo vstupného súboru
  SAMRecord[] records = ...;
// vytvorenie nového bloku v komprimovanom súbore
  OutputBlock outputBlock = BinaryBlockFile.newBlock();
// zápis počtu záznamov v bloku do hlavičky (prvý prúd v bloku)
  OutputStream headerStream = block.newStream("header");
  VByteCode.encodeUnsigned(data.length, headerStream);
// spustenie kompresie v jednotlivých procesoroch v pipeline
  for(SAMProcessor proc : processors) {
    proc.compressBlock(records, outputBlock);
  }
// reinicializácia stavu procesorov v pipeline
  for(SAMProcessor proc : processors) {
    proc.reset();
  }
```

Aby bolo možné automatické vytvorenie objektov SAM procesorov, je potrebné aby ich implementácie obsahovali konštruktor bez parametrov.

D.3 Využitie v aplikácii SamZip

Aby bolo možné využiť vytvorené kompresné kodeky a SAM procesory v aplikácii SamZip, je potrebné aby sa ich triedy nachádzali v „classpath“ virtuálneho stroja Javy.

Najjednoduchšie je vytvorenie jar balíčka, ktorý obsahuje nové triedy. Aplikáciu SamZip je potom možné spustiť nasledujúcim spôsobom:

```
java [argumenty JVM] -cp "samzip.jar:lib/*:plugin.jar" samzip.cmd.
    Main {príkaz} [argumenty príkazu]
```

kde `plugin.jar` je jar balíček s novými triedami. Potom je už možné využiť novo vytvorený kodek alebo SAM procesor v kompresnom profile.

Príklad D.1. Napríklad sme vytvorili nový kompresný kodek `NovyKodek`, ktorý sme umiestnili v balíčku `samzip.plugin`. V kompresnom profile je možné tento kodek špecifikovať uvedením nasledujúcej definície:

```
<codec class="samzip.plugin.NovyKodek"/>
```

Poznámka D.1. Pozor! Na systémoch rodiny Windows je potrebné používať v parameteri `-cp` namiesto oddeľovača `:`, oddeľovač `;`.

Metóda/Atribút	**Popis**
Abstraktné metódy	
`public void write(int data)`	Zakóduje hodnotu `data`. Pre reprezentáciu vstupných dát využíva aktuálne nastavený dátový mód.
Verejné (public) metódy	
`void write(byte[] data)`	Zakóduje hodnoty v poli `data`.
`void write(byte[] data, int off, int len)`	Zakóduje `len` hodnôt z poľa `data`, ktoré v tomto poli začínajú na pozícii `off`.
`void setDataMode(DataMode mode)`	Nastavenie používaného dátového módu.
`void setOutputStream (OutputStream stream)`	Nastavenie výstupného prúdu, ktorý bude použitý pre uloženie komprimovaných dát.
`void initialize()`	Inicializácia enkodéra – napr. inicializácia kompresných štruktúr. Túto metódu je potrebné zavolať pred vlastnou kompresiou.
`void close()`	Ukončí kompresiu a zapíše ešte nezapísané dáta do výstupného prúdu.
Chránené (protected) atribúty	
`DataMode dataMode`	Používaný dátový mód.
`boolean initialized`	Indikuje, či bola metóda inicializovaná volaním `initialize()`.
`OutputStream outputStream`	Výstupný prúd do ktorého sú ukladané komprimované dáta.

Tab. D.1: Popis rozhrania abstraktnej triedy `EncoderMethod`

Metóda/Atribút	Popis
Abstraktné metódy	
`public int read()`	Dekóduje jednu hodnotu zo vstupu. V prípade, že na vstupe už nie je viac dát, vráti znak konca prúdu. Hodnota znaku konca prúdu závisí na nastavenom dátovom móde, je však vždy záporná.
Verejné (public) metódy	
`int read(byte[] data)`	Dekóduje maximálne `data.length` bajtov do poľa `data`. Vráti počet dekódovaných bajtov.
`int read(byte[] data, int off, int len)`	Dekóduje maximálne `len` bajtov do poľa `data` od pozície `off`. Vráti počet dekódovaných bajtov.
`void setDataMode(DataMode mode)`	Nastavenie používaného dátového módu.
`void setInputStream (InputStream stream)`	Nastavenie vstupného prúdu, ktorý obsahuje komprimované dáta určené pre dekompresiu.
`void initialize()`	Inicializácia dekodéra – napr. inicializácia kompresných štruktúr. Túto metódu je potrebné zavolať pred vlastnou dekompresiou.
`void close()`	Ukončenie dekompresie.
`int EOS()`	Vráti hodnotu znaku konca prúdu pre používaný dátový mód.
Chránené (protected) atribúty	
`DataMode dataMode`	Používaný dátový mód.
`boolean initialized`	Indikuje, či bola metóda inicializovaná volaním `initialize()`.
`InputStream inputStream`	Vstupný prúd používaný pri dekompresii.

Tab. D.2: Popis rozhrania abstraktnej triedy `DecoderMethod`

Metóda/Atribút	Popis
Abstraktné metódy	
`protected EncoderMethod createRawEncoderMethod (OutputStream outputStream)`	Vytvorí novú inštanciu enkodéra, ktorý bude využívať výstupný prúd `outputStream`. Vrátená inštancia by nemala byť inicializovaná.
`protected DecoderMethod createRawDecoderMethod (InputStream inputStream)`	Vytvorí novú inštanciu dekodéra, ktorý bude využívať vstupný prúd `inputStream`. Vrátená inštancia by nemala byť inicializovaná.
Verejné (public) metódy	
`void addCodec(String pipeID, CompressionCodec codec)`	Nastavenie kodeku `codec` pre pomenované zreťazenie `pipeID`.
`void setPreference(String key, String value)`	Nastavenie hodnoty parametra `key` na `value`.
`void setDataMode(DataMode mode)`	Nastavenie používaného dátového módu.
`EncoderMethod newEncoderMethod()`	Vytvorí nový neinicializovaný enkodér.
`EncoderMethod newEncoderMethod (OutputStream outputStream)`	Vytvorí nový inicializovaný dekodér, ktorý bude využívať výstupný prúd `outputStream`.
`DecoderMethod newDecoderMethod()`	Vytvorí nový neinicializovaný dekodér.
`DecoderMethod newDecoderMethod (InputStream inputStream)`	Vytvorí nový inicializovaný dekodér, ktorý bude využívať vstupný prúd `inputStream`.
Chránené (protected) metódy	
`void preferenceChanged (String whatKey)`	Táto metóda je zavolaná vždy pri zmene parametra `whatKey`. K jednotlivým parametrom je možné pristupovať pomocou metód objektu `preferences`.

Tab. D.3: Popis rozhrania abstraktnej triedy `CompressionCodec`

Metóda/Atribút	Popis
Abstraktné metódy	
`public void compressBlock (SAMRecord[] rec, BinaryBlockFile.OutputBlock outputBlock)`	Kompresia alebo spracovanie bloku záznamov `rec`. Pre výstup je možné použiť blok v komprimovanom súbore `outputBlock`.
`public void decompressBlock (SAMRecord[] rec, BinaryBlockFile.InputBlock inputBlock)`	Dekompresia bloku záznamov. Vstupné dáta je možné čerpať z bloku v komprimovanom súbore `inputBlock`. Výsledné SAM záznamy je potrebné zapísať do bloku záznamov `rec`.
Verejné (public) metódy	
`void addCodec(String pipeID, CompressionCodec codec)`	Nastavenie kodeku `codec` pre vstup/výstup s identifikátorom `pipeID`.
`CompressionCodec getCodecFor(String id)`	Vráti kompresný kodek, ktorý je nastavený pre výstup s identifikátorom `pipeID`, alebo `null` ak nie je pre daný identifikátor nastavený žiadny kodek.
`void setPreference(String key, String value)`	Nastavenie hodnoty parametra `key` na `value`.
`void initialize()`	Inicializácia SAM procesora. Táto metóda je volaná po nastavení všetkých parametrov a kompresných kodekov pri čítaní kompresného profilu.
`void reset()`	Reinicializácia SAM procesora. Táto metóda je volaná vždy po ukončení všetkých procesorov v kompresnej pipeline. Jej hlavnou úlohou je uzavretie zdieľaných výstupných prúdov.
`SAMField[] getDependencies()`	Vráti SAM polia, ktoré musia byť spracované v predchádzajúcich procesoroch v kompresnej pipeline.
`SAMField[] getProcessedFields()`	Vráti SAM polia, ktoré sú ukladané týmto procesorom.
`void addPrecedingFields (SAMField[] fields)`	Nastavenie polí, ktoré sú spracované v kompresnej pipeline pred aktuálnym procesorom.
`boolean isValid()`	Vráti `true`, ak sú nastavenia SAM procesora v poriadku.
Chránené (protected) metódy	
`void preferenceChanged (String whatKey)`	Táto metóda je zavolaná vždy pri zmene parametra `whatKey`. K jednotlivým parametrom je možné pristupovať pomocou metód objektu `preferences`.

Tab. D.4: Popis rozhrania abstraktnej triedy `SAMProcessor`