Übungen zum Sequenzanalyse-Praktikum

Universität Bielefeld, WS 2020/21
Dr. Roland Wittler · M. Sc. Tizian Schulz
http://gi.cebitec.uni-bielefeld.de/teaching/2020winter/sequaprak
praktikum-seqan@CeBiTec.Uni-Bielefeld.DE

Übungsblatt 2 vom 10./11.11.2020
Abgabe bis Sonntag bzw. Montag, 24:00 Uhr.

Aufgabe 1 (Kompression mit `bzip2`)

Erstelle vier Textdateien je der Größe 100.000 Bytes, die folgendermaßen aufgebaut sein sollen:

1. Zufallszeichen aus einem 4-Buchstabenalphabet,
2. DNA-Sequenz,
3. Protein-Sequenzen,
4. natürlichsprachlicher Text.

Erstelle diese Dateien folgendermaßen:

zu 1. Implementiere eine entsprechende Methode (z.B. in Java oder Python).

zu 2.–4. Im Ornder `/vol/seqan/Praktikum/Kompression` sind entsprechende Dateien hinterlget, die als Grundlage dienen sollen. Kopiere sie in dein Benutzerverzeichnis und verarbeite sie dort mit den folgenden Hilfestellungen weiter. Verwende *Pipes* (|) um die einzelnen Verarbeitungssschritte direkt zu verzahnen. Um einen Eindruck vom Aufbau der Dateien zu erlangen, ohne sie in einem Editor öffnen zu müssen, verwende Komandozeilen-Tools wie `head`, `less`, oder `more`.

zu 2.
- Wie können mit dem Tool `grep` Header-Zeilen entfernt werden?
- Wie können mit dem Tool `tr` Zeilenumbrüche und Ns entfernt werden?
- Wie können mit einer Kombination der Tools `head` und `tail` die Zeichen 1.000.000–1.100.000 extrahiert werden? (Um Randeffekte zu vermeiden, verwenden wir nicht die ersten oder letzten 100.000 Zeichen.)

zu 3. Hier müssen selbstverständlich die Ns nicht entfernt werden, und es könnnen die ersten 100.000 Zeichen extrahiert werden.

zu 4.
- Wie kann mit dem Tool `cut` die ersten Spalten entfernt werden?
- Verwende `tr` um Zeilenumbrüche durch Leerzeichen zu ersetzen, und `head` und die ersten 100.000 Zeichen zu extrahieren.

Kontrolliere, ob alle vier Dateien vor der Komprimierung die gleiche Größe haben. Komprimiere diese Dateien nun mit `bzip2` und vergleiche dann deine Beobachtungen in einer Tabelle und versuche sie zu erklären. Betrachte dabei sowohl die Krompressionsfaktoren, die sich allein aus der Alphapetgröße ergeben, als auch weitere für die Kompression relevante Eigenschaften der Texte.

— Bitte wenden —

Aufgabe 2 (Bowtie 2)

Im Folgenden verwenden wir den Read-Aligner Bowtie 2. Das Tool ist auf dem CeBiTec-System unter `/vol/biotools/lib/bowtie2-2.4.1` installiert – hier findest du auch ein Manual und einen Ordner `example`. Wie alle rechenintensive Befehle, sollte auch Bowtie nur auf dem Compute Cluster, also von einem `qxterm` aus aufgerufen werden.

1. Verschaffe dir im Manual einen Überblick über das Tool. Du musst nicht alles im Detail lesen.
2. Im Ordner `example` findest du einige Beispieldateien. Da du in diesem Ordner keine Schreibrechte hast, solltest du ihn dir in dein Home-Verzeichnis kopieren. Führe die nachfolgenden Schritte mit der Referenzsequenz `lambda_virus.fa` aus.
   (a) Erstelle den Index für die Referenz mit dem Programm `bowtie2-build`.
   (b) Aligniere die Reads in der Datei `reads_1.fq` als ungepaarte Reads zur Referenzsequenz mit dem Programm `bowtie2` und speichere die Ergebnisse im SAM-Format ab (du musst hier nicht wissen, wie genau dieses Format aufgebaut ist). Schaue dir das Ergebnis an. Bowtie gibt zusätzlich eine kleine Statistik auf der Konsole aus. Wie viele Reads konnten erfolgreich aligniert werden?
   (c) Nun wollen wir Variationen in den Reads im Vergleich zur Referenzsequenz finden (SNPs, kurze Insertionen und Deletionen). Dazu verwenden wir `samtools` und `bcftools`. Beide Tools sind im CeBiTec installiert und sollten ebenfalls nur auf dem Compute Cluster aufgerufen werden.
   Nutze `samtools view` und `samtools sort`, um das Ergebnis deines Alignments in ein sortiertes BAM-file zu konvertieren (BAM ist die komprimierte Version von SAM).
   Um die Varianten zu finden, führe `samtools mpileup -uf <referenz>.fa <sortiertes bam> | bcftools call -vc -Ob > <ausgabename>.raw.bcf` aus.
   Du kannst dir das Ergebnis mit `bcftools view` anschauen. Wie viele Varianten enthält deine Ausgabe?
   (d) Wie verhält sich die Laufzeit für das Mapping in Relation zur Länge der Referenz?
   - Hierzu erzeuge zunächst eine Fasta-Datei, die nur etwa die Hälfte der Referenz enthält. Nutze `grep` um zu zählen, wie viele Header- und Sequenz-Zeilen die Referenzdatei enthält, und `head` um eine Datei zu erzeugen, die halb so viele Sequenz-Zeilen enthält. Erstelle einen Index für diese verkürze Referenz. Wie viele Reads können hier erfolgreich aligniert werden?
   - Bestimme die Laufzeiten für das Mapping auf die vollständige und auf die verkürzte Referenz. Um den gesuchten Effekt für diese vergleichsweise kleinen Eingabedaten hervorzuheben, beschränken wir das Mapping mit dem Parameter `--score-min 'C,0'` auf exakte Matches. Um die Laufzeit über 100 Läufe zu mitteln, bette den Mapping-Aufruf in eine Schleife ein:
     `for i in {1..100}; do ...; done`
     Miss die Gesamtlaufzeit durch Voranstellen des Kommandos `/usr/bin/time` und protokolliere die *User time*.
   - Vergleiche und diskutiere die beiden Laufzeiten. Welches Verhältnis könnte man bei den Längen der Referenzsequenzen und Anzahl Matches erwarten? Wird diese Erwartung erfüllt? Erläutere.

Beschreibe in deinem Protokoll die einzelnen Programmaufrufe und was sie bewirken. Beantworte alle Fragen in eigenen Worten anstatt Ergebnisdateien oder Konsolenausgaben ins Protokoll zu übernehmen.