Übungen zum Sequenzanalyse-Praktikum

Universität Bielefeld, SoSe 2025
M. Sc. Leonard Bohnenkämper · M. Sc. Luca Parmigiani · Dr. Roland Wittler
https://gi.cebitec.uni-bielefeld.de/teaching/2025summer/sequaprak
praktikum-seqan@CeBiTec.Uni-Bielefeld.DE

Übungsblatt 4 vom 06.05.2025
Abgabe bis Sonntag, 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 Ordner `/prj/seqan/Praktikum/Kompression` sind entsprechende Dateien hinterlegt, die als Grundlage dienen sollen. Kopiere sie in dein Benutzerverzeichnis und verarbeite sie dort mit den folgenden Hilfestellungen weiter. Verwende *Pipes* (|) um die einzelnen Verarbeitungsschritte direkt zu verzahnen. Um einen Eindruck vom Aufbau der Dateien zu erlangen, ohne sie in einem Editor öffnen zu müssen, verwende Kommandozeilen-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 *N*s 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 *N*s nicht entfernt werden, und es können 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` um 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 Kompressionsfaktoren, die sich allein aus der Alphabetgröße 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 rechenintensiven Befehle, sollte auch Bowtie nur auf dem Compute Cluster, also von einem `slxterm` 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 `bcftools mpileup <sortiertes bam> -f <referenz>.fa | bcftools call -vc > <ausgabename>.raw.bcf` aus.
   Wie viele Varianten enthält deine Ausgabe?
   (d) Wie verhält sich die Laufzeit für das Mapping in Relation zur Anzahl Treffer? Hierzu erzeugen wir zwei verschiedene Varianten der Referenzsequenz.
   - Eine Fasta-Datei, die die Headerzeile und dann drei Kopien der Referenzsequenz enthält.
   - Eine Fasta-Datei, die die Headerzeile, zwei Kopien der Referenzsequenz und eine Variante der Referenzsequenz enthält, in der alle 'A's durch 'G's ersetzt werden. Im modifizierten dritten Teil sind somit Treffer ausgeschlossen.

   Um den gesuchten Effekt für diese vergleichsweise kleinen Eingabedaten hervorzuheben, beschränken wir ab jetzt das Mapping mit den Parametern `--score-min 'C,0' --no-unal -k 3` auf (bis zu drei) exakte Matches. Bestimme für beide Versionen der Referenz jeweils die Laufzeit für das Mapping und die Anahl Treffer.

   Achtung, da die Reads nun mehrfach gemappt werden können, kann die Anzahl Treffer nicht einfach aus der Bowtie2-Ausgabe abgelesen werden. Sie kann aber aus der SAM-Datei gewonnen werden.

   Um die Laufzeit über 100 Läufe zu mitteln, bette den Mapping-Aufruf in eine Schleife ein:

   ```
   for i in {1..100}; do ...; done
   ```

   (Tipp: Füge bei bowtie2 Parameter `--quiet` hinzu, um die wiederholte Ausgabe zu unterdrücken.) Miss die Gesamtlaufzeit durch Voranstellen des Kommandos `time` und protokolliere die *User time*.

   Vergleiche und diskutiere die Anzahl Matches und Laufzeiten für die beiden Fälle. Welches Verhältnis der Laufzeiten könnte man bei den Anzahlen 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.