Prof. Dr. Knut Reinert
Jakob Schulz
René Rahn

Institut für Informatik
AG Algorithmische Bioinformatik

Algorithmen und Datenstrukturen in der Bioinformatik
Weihnachtsaufgabe WS 14

Abgabe Montag, 19.01., 15:00 Uhr per SVN

Readmapping Gegeben ist ein Genom $\mathcal{G}$ und einer Menge von Reads $\mathcal{R}$. Für jeden Read $R \in \mathcal{R}$ soll die Position und die Anzahl der Fehler mit der R in $\mathcal{G}$ gefunden wurde ausgegeben werden. Die Aufgabe wird mit **doppelter Punktzahl** bewertet und die 3 schnellsten Programme [1] (und dennoch akzeptabler Genauigkeit bei den Ergebnissen) wird mit einem kleinen Preis belohnt :)

Programm-Spezifikation Schreiben Sie ein Programm mit dem Namen `aufgabe5.cpp`, das die folgenden Schritte implementiert:

- Einlesen des Genoms.
- Einlesen der Reads.
- Aufbau eines qGram-Indexes über das Genom.
- Mappen der Reads gegen das Genom mit Hilfe des Indexes.
- Ausgeben der Matches.
- Das Alphabet ist $\Sigma = \{A, C, G, T\}$.

qGram-Index Schreiben Sie eine Klasse `qGramIndex` mit den folgenden Membervariablen:

- `unsigned q` - Die qGram-Größe.
- `unsigned alphabetSize` - Die Alphabetgröße.
- `std::vector<unsigned> qGramDir` - Die qGram-Tabelle, die die Startposition des qGrams im Suffix Array (SA) speichert.
- `std::vector<unsigned> sufTab` - Das Suffix Array sortiert nach den ersten q Positionen.

Implementieren Sie zusätzlich folgende Interfaces zum Arbeiten mit dem Index und zum Verifizieren eines möglichen Treffers:

[1] gemessen wird auf einem 8-Kerne Uni-Rechner

- `unsigned getHash(qGramIndex & index, std::string::iterator it)` - Berechnet den Hashwert für das aktuelle qGram gegeben durch einen Iterator auf den Anfang des zu berechnenden qGrams und gibt ihn zurück.
- `unsigned getNextHash(qGramIndex & index, std::string::iterator it, unsigned previousHash)` - Berechnet den darauf folgenden Hashwert und gibt ihn zurück.
- `void buildQGramIndex(qGramIndex & index, std::string & genome)` - Baut den qGram Index.
- `bool lookupQGramIndex(qGramIndex & index, std::string::iterator, std::pair<unsigned, unsigned> & range)` - Gibt das Intervall über `sufTab` wieder, in dem das gegebene qGram vorkommt. Das Intervall wird in `range` gespeichert. Sollte das qGram nicht gefunden werden, bleibt `range` leer und die Funktion gibt `false` zurück. Ansonsten `true`.
- `bool verifyMatch(std::string::iterator textBegin, std::string::iterator patternBegin, std::string::iterator patternEnd, unsigned mismatches)` - Gibt `true` zurück sollte der Treffer verifiziert werden, ansonsten `false`. Nur die Hammingdistanz wird verglichen. Sie finden unter `https://svn.imp.fu-berlin.de/aldabi/WS14/material/material_a5/aufgabe5.cpp` eine Vorlage mit den geforderten Interfaces.

Eingabe-Spezifikation Die folgenden Argumente werden dem Program beim Aufruf mit übergeben.

1. `GENOME` - Die Eingabedatei mit dem Genom in einer einzelnen Zeile.
2. `READS` - Die Eingabeatei mit den Reads. Ein Read pro Zeile.
3. `k` - Die maximal zugelassenen Fehler.
4. `RESULT` - Die Ausgabedatei mit den Ergebnissen. Die Endung des Dateinamens muss *.mapout* entsprechen.

Ausgabe-Spezifikation

1. Jeder verifizierter Treffer wird in eine separaten Zeile geschrieben.
2. Die Zeilen bestehen aus dem Format: `READ POS ERROR`, where
   - `READ` - Die Readsequenz die gefunden wurde,
   - `POS` - Die **End**position des matchenden Reads im Genom,
   - `ERROR` - Die Anzahl der Fehler mit die der Read matched

   ist.

Beispiel

- Datei mit dem Genom:
  AAGATACATTTTTAAAAAAAAAACAATTT

- Datei mit den Reads:
  GATACA
  CATTTT
- Ausgabedatei:
  GATACA 7 0
  CATTTT 11 0
  CATTTT 28 1

Praktikumshinweise Beachten Sie die Hinweise unter `https://www.mi.fu-berlin.de/w/ABI/AlDaBiWS14Praktikum`.