S. Kurtz, T. Schwabe,
G. Gonnella, D. Willrodt

Universität Hamburg
Zentrum für Bioinformatik

Programmierung für Naturwissenschaften
Sommersemester 2015
Übungen zur Vorlesung: Ausgabe am 02.07.2015

Punkteverteilung: Aufgabe 10.1: 4 Punkte, Aufgabe 10.2: 6 Punkte

Abgabe bis zum 08.07.2015, 10:00 Uhr.

Aufgabe 12.1 Blast Parsing

BLAST (*Basic Local Alignment Search Tool*) ist eines der meistgenutzten Programme, um Aminosäure- oder Nucleotidsequenzen mit Sequenzen in einer Datenbank zu vergleichen.

Ein typischer Output einer Suche mit BLAST enthält folgende Informationen:

- einen Block mit der zeilenweisen Auflistung aller Sequenzen in der Datenbank, die ein signifikantes Alignment mit der Such-Sequenz haben. Die Zeilen enthalten:
  - den Namen der Sequenz
  - den Score des Alignments
  - den dazu gehörenden E-Value
- nach diesem Block folgen für jede Sequenz weitere Informationen, z.B. eine Darstellung des Alignments.

In STiNE findet sich die Datei (`blastout.txt`), welche das Ergebnis einer BLAST-Suche mit der Sequenz des Enzyms *3-deoxy-7-phosphoheptulonate synthase* enthält. Laden Sie diese Datei herunter. Ihre Aufgabe ist es nun einen Parser zu schreiben, der diese Datei einließt und einen Report ausgibt. Verwenden Sie dazu Python und erstellen Sie zwei Varianten für den Parser, einmal mit und einmal ohne die Verwendung regulärer Ausdrücken.

Der Report soll zeilenweise für jede Sequenz im Output die folgenden Punkte enthalten:

1. den Identifier (Substring nach `'>'` bis zum ersten Leerzeichen)
2. den BitScore des Treffers
3. den raw Score des Treffers (in Klammern hinter dem BitScore)
4. den E-Value
5. die Anzahl der identischen Aminosäuren (identities)
6. die Anzahl der positiv bewerteten Aminosäuren (positives)
7. die Anzahl der Lücken (gaps)

Dabei sollen die Felder eines Eintrages durch Tabulatoren getrennt werden. Als Beispiel die erste Zeile, die Ihr Programm ausgeben soll:

```
>gb|EGJ41676.1|␣714␉1844␉0.0␉343␉343␉0
```

(␉ = Tabulator)

Aufgabe 12.2 Multiprozessualer Sequenzvergleich

Wie Sie bereits wissen sollten, gibt es in der biologischen Sequenzanalyse viele „alle-gegen-alle“-Probleme, bei der für eine gegebene Menge M von Sequenzen alle Paare $s, s' \in M$ verglichen werden, um jeweils einen Distanzwert zu ermitteln. Der Vergleich zweier Sequenzen ist meist sehr aufwändig. So werden typischerweise Verfahren zum Sequenzvergleich angewendet, die für zwei Sequenzen s und s' eine Laufzeit von $O(|s| \cdot |s'|)$ haben. Dabei ist $|s|$ die Länge der Sequenz s. Diese Verfahren rechnen also viel auf relativ wenigen Daten und liefern wenige Ausgaben. Daher kann man diese „alle-gegen-alle“-Probleme gut mit Hilfe von Shared-Memory Multithreading Ansätzen lösen.

In dieser Aufgabe sollen Sie ein Python-Programm `all-against-all.py` schreiben, das eine Datei im vereinfachten multiplen-Fasta Format einliest und für alle darin enthaltenen Sequenzpaare die Edit-Distanz berechnet. Für die Berechnung der Edit-Distanz zweier Frequenzen soll die C-Funktion `eval_unit_edist` über die das Modul `ctypes` benutzt werden. Eine Implementierung der Funktion finden Sie in den Materialien.

Dateien im vereinfachten multiplen Fasta-Format bestehen aus Kopfzeilen, die mit dem Zeichen > beginnen, gefolgt von genau einer Sequenz-Zeile, in der die Sequenzinformation steht. In den Materialien finden Sie die Beispiel-Datei: `sw175.fna`.

Das Ziel ist es, k Paare von Sequenzen mit minimalen Distanzwerten zu ermitteln, wobei k ein durch den Benutzer definierter Wert ist, der als Option an das Programm übergeben werden soll. Außerdem soll die Möglichkeit bestehen, dass die Auswertung multiprozessual erfolgt. Daher soll auch eine Opton t genutzt werden, um die Anzahl der Prozesse anzugeben. Schließlich wird auch der Name der Inputdatei an das Skript übergeben. Der Aufruf erfolgt also:

```
all-against-all.py -t <t> -k <k> <inputdatei>
```

Um die Parallelisierung zu ermöglichen, soll der eigentliche Frequenzvergleich in einer Funktion erfolgen, die jeweils für einen gegeben Satz von Frequenzpaaren die k besten Paare bestimmt und zurück gibt. Falls der Aufruf mehrfach (mit unterschiedlichen Sets von Paaren) erfolgt, soll abschließend im Hauptprogramm die finallen k besten Paare ermittelt werden. So kann vermieden werden, dass die Unterprozesse auf einer gemeinsamen Datenstruktur arbeiten müssen.

Versuchen Sie, eine Aufteilung der Paare auf t Sets zu erzielen, so dass jedes Set in etwa die gleich Laufzeit benötigt. Schätzen Sie die Laufzeit jedes Paars als proportional zum Produkt der Längen der beiden Sequenzen ab.

Vermeiden Sie, Datenstrukturen zu verwenden, deren Speicherbedaruf mit $O(N^2)$ skalieren; also insbesondere eine Liste aller Laufzeiten oder eine Liste aller Frequenzpaare.

Ein Referenzoutput nach Analyse von `sw175.fna` für die 50 besten Frequenzpaare finden Sie in der Datei `result.txt`.

Die Lösungen zu diesen Aufgaben werden am 09.07.2015 besprochen.