Stefan Kurtz **Universität Hamburg**
Emanuel Ehmki **Zentrum für Bioinformatik**

Programmierung in der Bioinformatik
Wintersemester 2019/2020
Übungen zur Vorlesung: Ausgabe am 20.11.2019

Ab dieser Woche können Sie jeweils Mittwochs über *QuickFeedback* anonymisiert Rückmeldung zur Vorlesung und Übung geben. Sie könnten z.B. Hinweise geben zu Abschnitten aus der Vorlesung, die noch nicht verstanden wurden und wiederholt werden sollten. Auch Hinweise zu Übungsaufgaben, die unklar formuliert oder zu schwierig waren, wären hilfreich und könnten helfen, die Veranstaltung zu verbessern. Hier ist der Link und der QRcode zur *QuickFeedback* Web-Seite für dieses Modul:

`https://feedback.informatik.uni-hamburg.de/PfN1/wise2019-2020`

Aufgabe 5.1 (2 Punkte) Das Programm `fixerrors.py` enthält viele Fehler. Ihre Aufgabe ist es, diese zu finden und zu korrigieren. Sie können natürlich den Python-Interpreter verwenden, um Hinweise auf die Fehlerstellen zu erhalten.

Nach der Korrektur aller Fehler müssen Sie testen, ob das Programm die richtige Ausgabe liefert. Das wird durch den Aufruf von `make test` im Verzeichnis der Materialien zu dieser Übungsaufgabe bewerkstelligt.

Aufgabe 5.2 (5 Punkte) In dieser Aufgabe geht es darum, aus einer Textdatei ganze Zahlen und Fließkommazahlen zu extrahieren, hierfür jeweils eine Verteilung zu bestimmen und diese dann auszugeben. Der Inhalt der Textdatei wurde von Blast, einem Programm zum Vergleich biologischer Sequenzen, ausgegeben und wurde für die Aufgabe vereinfacht. Uns interessieren hier drei Schlüsselwerte aus den Ergebnissen des Sequenzvergleichs, nämlich der Bitscore, der Erwartungswert (Expect) und der Anteil der identischen Zeichen (Identities). Was diese genau bedeuten ist in dieser Aufgabe nicht wichtig. Wir beschränken uns in dieser Aufgabe auf Zeilen der Blast-Ausgabe, die wie folgt aussehen:

```
Score = 132 bits (332), Expect = 5e-32
Identities = 108/341 (32%)
Score = 45.8 bits (107), Expect = 6e-05
Identities = 26/62 (42%)
```

Aus den ersten beiden Zeilen sollen die Werte 132 (der Bitscore), $5e-32$ (der Erwartungswert) und 32 (der relative Anteil der identischen Zeichen beim Sequenzvergleich) extrahiert werden. Aus den letzten beiden Zeilen sind das die Werte 45.8, $6e-05$ und 42. In den Materialien zu dieser Aufgabe finden Sie eine Datei `blaststat.txt` mit insgesamt 50 solcher Paare von Zeilen. Es sollen nun drei Verteilungen berechnet werden, jeweils eine für die drei genannten Werte und bzgl. aller Zeilen der Eingabe-Datei.

- Es soll die Verteilung der Werte der Bitscores ausgegeben werden. Beachten Sie, dass Bitscores auch als rationale Zahlen angegeben werden können (siehe Zeile 3 oben). Diese sollen gerundet und dann in eine ganze Zahl konvertiert werden. Hierfür verwenden Sie die Methode `round`. So liefert `round(48.6)` den Wert `49`.
- Es soll die Verteilung der 10er-Logarithmen der Erwartungswerte ausgegeben werden. Zur Berechnung der log-Werte verenden Sie die Funktion `log10` aus dem Math-Modul. Dazu muss die Zeile `from math import log10` am Anfang Ihres Python-Skriptes eingefügt werden. Den Fall, dass der Erwartungswert 0 ist, müssen Sie gesondert behandeln, denn `log10(0)` ist undefiniert.
- Es soll die Verteilung der Prozent-Identities-Werte ausgegeben werden.

In der Eingabedatei finden Sie nur Zeilen der beiden obigen Formen. Die einzelnen Werte und Strings in den Zeilen sind jeweils durch genau ein Leerzeichen getrennt. Verwenden Sie zwei verschiedene reguläre Ausdrücke mit *Captures* (ausgedrückt durch Paare von runden Klammern), um die genannten Werte zu extrahieren. Die regulären Ausdrücke sollen spezifisch für die beiden Zeilenformen sein, d.h. auch die Teile der Zeilen berücksichtigen, die Werte enthalten, die hier nicht von Interesse sind.

Falls der reguläre Ausdruck matcht, sind die extrahierten Werte, die durch die Methode `group` geliefert werden Strings. Diese müssen Sie für die Weiterverarbeitung mit den Methoden `int` und/oder `float` konvertieren.

Um die Verteilungen zu speichern, verwenden Sie dictionaries. Die Verteilungen sollen tabulatorsepariert und numerisch sortiert nach den Schlüsselwerten ausgegeben werden. Dazu können Sie Programmzeilen der Form

```
for k in sorted(dist):
    print("{}\t{}".format(k,dist[k]))
```

verwenden. Dabei ist `dist` das dictionary, das die Verteilung speichert. Vor der Ausgabe jeder Verteilung gibt es zwei Kopfzeilen (jeweils beginnend mit dem Zeichen #), die die Verteilung beschreiben und die Spaltenbezeichner einführen. Die genaue Form der Kopfzeilen finden Sie in der Datei `blaststat_output.txt`, die die erwartete Ausgabe enthält.

Benennen Sie die Datei `blaststat_template.py` aus den Materialien zu dieser Übungsaufgabe um in `blaststat.py`. In dieser Datei erfolgt Ihre Implementierung. Im Material finden Sie die Eingabedatei und eine Datei mit dem erwarteten Ergebnis. Durch `make test` verifizieren Sie die Korrektheit Ihres Programms.

Aufgabe 5.3 (3 Punkte) Eine Primzahl ist eine ganze Zahl $p \geq 2$, die sich nicht ganzzahlig durch eine kleinere Zahl $q \geq 2$ teilen lässt. In dieser Aufgabe soll der Algorithmus „Sieb des Eratosthenes“ zur Generierung einer Liste von Primzahlen $\leq n$ implementiert werden. n ist dabei ein vom Benutzter definierter Wert. Der Algorithmus funktioniert wie folgt:

Man notiert zunächst alle Zahlen i, $2 \leq i \leq n$. Z.B. ergibt sich für $n = 20$ die folgende Zahlenfolge

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Im nächsten Schritt werden alle echten Vielfachen von 2 markiert (hier durch das Symbol $\times$ dargestellt.

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
		×		×		×		×		×		×		×		×		×

Nun markiert man zusätzlich alle echten Vielfachen der kleinsten unmarkierten Zahl (in diesem Fall 3):

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
		×		×		×	×	×		×		×	×	×		×		×

In jedem Schritt werden also jeweils die echten Vielfachen der kleinsten noch nicht markierten Zahl zusätzlich markiert. Das wird solange wiederholt bis die kleinste unmarkierte Zahl größer als $\sqrt{n}$ ist. Die noch nicht markierten Zahlen sind dann die gesuchten Primzahlen.

In der Datei `eratosthenes25_animation.pdf` finden Sie eine Visualisierung des Algorithmus für $n = 25$, in der die 25 Zahlen in einem Quadrat angeordnet werden. Das Programm zur Darstellung der PDF-Datei muss allerdings mit „Overlays“ umgehen können. Das gilt z.B. für den Adobe Acrobat Reader ™. Falls kein geeigneter PDF-Viewer zur Verfügung steht, schauen Sie sich die statische Darstellung in `eratosthenes25_steps.pdf` an.

Die Markierung läßt sich am besten mit einer Liste von boolschen Werten (d.h. jeder Wert der Liste ist entweder `True` oder `False`) repräsentieren. Sei `marked` diese Liste. Der i-te Wert in der Liste ist genau dann `True`, wenn die Zahl i markiert ist. Am Anfang müssen alle Werte ab Index 2 auf `False` gesetzt werden, da noch keine Zahl markiert ist. Die Werte an Index 0 und 1 in der Liste können Sie auf `None` setzen, da sie nicht benötigt werden. Die einzelnen Phasen des Algorithmus bestehen darin, zu testen, ob eine Zahl `i` markiert ist (dann gilt `marked[i]`) bzw. nicht markierte Zahlen zu markieren, d.h. `marked[i] = True` zu setzen, entsprechend der obigen Beschreibung des Algorithmus.

Vor der Implementierung benennen Sie die Datei `eratosthenes_template.py` aus den Materialien um in `eratosthenes.py`. Fügen Sie Ihren Python-Code in diese Datei an der markierten Stelle ein.

Es ist bereits Programmcode vorhanden zum Einlesen einer vom Benutzter übergebenen ganzen Zahl n. Es wird die Anzahl der Primzahlen $\leq n$ sowie die Liste der 10 größten Primzahlen $\leq n$ ausgegeben. Im Material zu dieser Übungsaufgabe finden Sie eine Datei mit der erwarteten Ausgabe für $n = 1\,000$ sowie ein Makefile. Durch Aufruf von `make test` verifizieren Sie die Korrektheit Ihrer Implementierung.

Bitte die Lösungen zu diesen Aufgaben bis zum 25.11.2019 um 18:00 Uhr an pfn1@zbh.uni-hamburg.de schicken. Die Besprechung der Lösungen erfolgt am 27.11.2019.