Stefan Kurtz **Universität Hamburg**
Mathias von Behren **Zentrum für Bioinformatik**

Programmierung in der Bioinformatik
Wintersemester 2015
Übungen zur Vorlesung: Ausgabe am 04.01.2016

Punktevergabe:

- Aufgabe 11.1: 3 Punkte
- Aufgabe 11.2: 3 Punkte
- Aufgabe 11.3: 4 Punkte

Aufgabe 11.1 Eine typische Aufgabe in der Programmierung besteht in der Auswertung von Kommandozeilenoptionen.

Dafür gibt es ein Ruby-Standardmodul, das Sie dabei unterstützt. Das Modul heißt `OptionParser` und wird in der Ruby-Dokumentation im Internet beispielhaft erklärt. `http://ruby-doc.org/stdlib-2.0.0/libdoc/optparse/rdoc/OptionParser.html`

Ihre Aufgabe ist es nun, einen Parser für die Kommandozeilenoptionen eines Programms zum Parsen von Genbank-Dateien zu schreiben. Es sollen die folgende Optionen implementiert werden:

`--selecttop <entry>[,<entry> ... ]`
: spezifiziert mindestens einen Genbank-Toplevel-Bezeichner, also z.B. `LOCUS`, `DEFINITION`, oder `ORIGIN`. Mehrere Bezeichner werden durch Kommata getrennt. [1]

`--echo` spezifiziert, dass der echo-Modus verwendet werden soll. (Es wird erst in einer Aufgabe auf einem späteren Übungsblatt klar, was das bedeutet.)

`--search <regexp>` spezifiziert, dass der search-Modus mit dem angegebenen regulären Ausdruck verwendet werden soll. (Es wird erst in einer Aufgabe auf einem späteren Blatt klar, was das bedeutet.)

`--help` Gibt eine kurze Anleitung zur Benutzung des Skripts aus.

Außerdem soll folgendes für die Optionen gelten:

- Ihr Programm soll wie folgt aufgerufen werden:

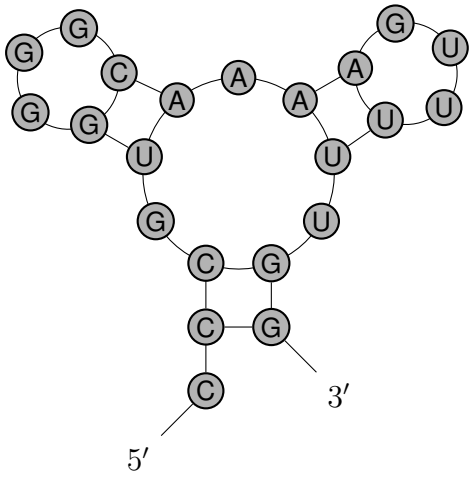
  ```
  Usage: #{$0} [options] inputfile
  ```

  (Dies können Sie im OptionParser mit dem accessor `.banner = "usagemessage"` setzen)
- `inputfile` ist ein notwendiger Parameter und soll der Name einer existierenden Datei sein.
- Keine der Optionen darf doppelt vorkommen.
- `--echo` und `--search` dürfen nicht zusammen benutzt werden.
- Bei der Benutzung von `--help` bricht das Skript nach der Ausgabe ab.
- Nach `--selecttop` dürfen nur Genbank-Toplevel-Bezeichner folgen.

[1] Für einen Hinweis zur Auswertung suchen Sie nach „List of arguments“, in der `OptionParser` Dokumentation.

Aufgabe 11.2 RNAs sind aus Ribonukleinsäuren aufgebaute einzelsträngige Moleküle. Ähnlich wie DNA sind sie aus 4 Basen aufgebaut, nämlich `A`, `C`, `G` und `U`. Die Funktion von vielen Klassen von RNAs ist durch ihre Tertiärstruktur, d.h. ihre Faltung im dreidimensionalen Raum, bestimmt. Da die Tertiärstruktur aber schwer computergestützt vorhergesagt werden kann, vereinfacht man in der Bioinformatik die Struktur von RNA zu einer zweidimensionalen Darstellung, welches in erster Linie die intermolekularen Basenpaare repräsentiert.

Dies könnte z.B. so aussehen:



Man kann eine solche RNA-Sekundärstruktur durch eine Sequenz von Klammern und Punkten, der sogenannten *dot-bracket*-Notation (auch Vienna-Notation) darstellen. Basen, die ein Watson-Crick Basenpaar bilden, sind durch zusammengehörige Klammern gekennzeichnet, ungepaarte Basen durch einen Punkt. Zusammen mit der RNA-Sequenz kann man daraus jederzeit die RNA-Sekundärstruktur rekonstruieren. Die zu obiger Abbildung gehörende Vienna-Notation sieht z.B. so aus:

```
CCCGUGGGGCAAAAGUUUUUGG
.((.((...)).((...)).))
```

Ihre Aufgabe ist es, eine Ruby-Methode `validateDotBracket` zu schreiben, die eine RNA-Sekundärstruktur in dot-bracket-Notation validiert. D.h. sie überprüft, ob der übergebene Punkt/Klammer-String nach den folgenden Regeln gebildet werden kann:

- `.` ist eine gültige RNA-Sekundärstruktur
- `()` ist eine gültige RNA-Sekundärstruktur
- falls r eine gültige RNA-Sekundärstruktur ist, dann ist es auch `(`r`)`
- falls r und s gültige RNA-Sekundärstrukturen sind, dann ist auch rs eine gültige RNA-Sekundärstruktur.

Wenn ein Fehler gefunden wird, soll die Methode einen Ausnahmfehler mit `raise` produzieren. Ist die RNA-Sekundärstruktur valide, soll die Methode alle Basenpaare in folgender Form ausgeben:

```
1-20
9-13
...
```

und `true` zurückgeben.

Verwenden Sie dazu den Stack, den Sie bereits entwickelt haben. Alternativ können Sie die STiNE zur Verfügung gestellte Klasse verwenden.

Benutzen Sie Ihre Methode in einem Skript, dass mehrere RNA-Sekundärstrukturen aus einer Datei einliest und validiert.

Das Dateiformat für diese Aufgabe ist wie folgt: Eine RNA-Sekundärstruktur wird am Anfang einer Zeile ohne Leerzeichen angegeben. Nach einem Leerzeichen folgt eines der beiden Schlüsselworte `valid` oder `invalid`. Damit können Sie in Ihrem Skript testen, ob ihre Validierungsmethode korrekt arbeitet.

In STiNE finden Sie die Datei `bracketTest.txt` mit mehreren Einträgen im obigen Format.

Aufgabe 11.3 Viele Methoden in der Bioinformatik basieren auf Matrizen. Es ist sinnvoll, die in einem Programm verwendeten Martizen als Instanzen einer Klasse zu implementieren. Entwickeln Sie eine Klasse `Matrix2D` für zweidimensionale Matrizen, welche genau die folgenden Methoden bereitstellt:

- `new(m,n)` initialisiert ein neues Objekt (Instanz) der Matrix2D Klasse mit m Zeilen und n Spalten. Alle Werte der Matrix sollen mit **`nil`** initialisiert werden.
- `set_value(i,j,value)` setzt den Wert des Matrixelements in Zeile i und Spalte j auf *value*.
- `get_value(i,j)` liefert den Wert des Matrixelements in Zeile i und Spalte j.
- `get_number_of_rows()` liefert die Anzahl der Zeilen der Matrix.
- `get_number_of_columns()` liefert die Anzahl der Spalten der Matrix.
- `pretty_print()` gibt die Matrix auf der Standardausgabe aus.

Die Indizierung der Zeilen und Spalten der Matrix beginnt jeweils mit 1. Implementieren Sie die Matrix2D-Klasse so, dass die ursprüngliche Größe der Matrix nicht mehr verändert werden kann. Behandeln Sie außerdem fehlerhafte Anfragen, bei denen die übergebenen Zeilen- und Spaltenindizes die Matrixgrenzen überschreiten, mit `raise(ArgumentError, msg).msg` soll dabei eine sinnvolle Fehlermeldung sein. Die Ausgabe der Matrix soll als kommaseparierte Tabelle erfolgen (csv).

Die Lösungen zu diesen Aufgaben werden am 18.01.2016 besprochen.