Abteilung für Genominformatik **Universität Hamburg**
Stefan Kurtz **Zentrum für Bioinformatik**

Genominformatik
Sommersemester 2014
Übungen zur Vorlesung: Ausgabe am 14.04.2014

Punkteverteilung: Aufgabe 2.1: 6 Punkte, Aufgabe 2.2: 3 Punkte, Aufgabe 2.2: 4 Punkte.

Abgabe bis zum 1.5.2014, 23:59 Uhr.

Aufgabe 2.1 Implementieren Sie den in der Vorlesung vorgestellten output-sensitiven Algorithmus zur Berechnung der Edit-Distanz von zwei Sequenzen u und v für die Einheitskostenfunktion. Dieser Algorithmus benutzt eine Matrix von *front*-Werten $front(h, d)$ für $d \geq 0$ und $-d \leq h \leq d$. Gehen Sie bei Ihrer Implementierung wie folgt vor:

1. Überlegen Sie sich, wie man die $front$-Werte der d-ten Generation, d.h. die Werte $front(h, d)$ für $-d \leq h \leq d$ speichern kann. Beachten Sie, dass h negativ sein kann, aber in einem Array keine nicht-negativen Indizes möglich sind.
2. Implementieren Sie eine Funktion $lcplen$, die für zwei beliebige Strings die Länge ihres längsten gemeinsamen Präfixes berechnet.
3. Implementieren Sie eine Funktion `outsense_next_front`, die aus der $(d-1)$-ten Generation die d-te Generation von $front$-Werten berechnet. Hier ist der Kopf der Funktion in C-Syntax:

   ```
   void outsense_next_front(unsigned long *destfront,
                            const unsigned long *sourcefront,
                            unsigned long d,
                            const unsigned char *useq,
                            unsigned long ulen,
                            const unsigned char *vseq,
                            unsigned long vlen)
   ```

   Dabei sind `useq` und `vseq` Zeiger auf die beiden Sequenzen u und v der Längen `ulen` und `vlen`. `sourcefront` ist ein Zeiger auf die $(d-1)$-te Generation von $front$-Werten und `destfront` ist ein Zeiger auf die d-te Generation von $front$-Werten. Es gilt dabei `d>0`.
4. Implementieren Sie eine Funktion `outsense_edist`, die den Speicherplatz für die jeweiligen Generationen von *front*-Werten bereitstellt, die $front(0, 0)$ berechnet, und die durch Aufrufe von `outsense_next_front` nacheinander die Generationen von *front*-Werten bestimmt. Sobald das Abbruchkriterium $front(vlen - ulen, d) = ulen$ gilt, ist der aktuelle d-Wert die Edit-Distanz, die von der Funktion als `return`-Wert geliefert wird.

   Hier ist der Kopf der Funktion in C-Syntax:

```
unsigned long outsense_edist(const unsigned char *useq,
                             unsigned long ulen,
                             const unsigned char *vseq,
                             unsigned long vlen)
```

5. Falls Sie Ihr Programm in C implementiert haben, finden Sie im Material zur Übung (siehe STiNE) einen Testrahmen, der es erlaubt, Ihr Programm einfach zu testen. Sie müssen lediglich in einer Datei `outsenseedist.c` die Funktion `outsense_edist` implementieren, `make` aufrufen und dann `outsenseedist.x` q aufrufen, wobei q eine positive ganze Zahl ≤ 9 ist. Es werden dann alle Paare von Sequenzen der Längen $\leq q$ über dem Alphabet $\{a, b\}$ generiert, und für jedes Paar wird getestet, ob die von `outsense_edist` berechnete Edit-Distanz korrekt ist.
6. Falls Sie Ihr Programm nicht in C implementiert haben, finden Sie in der Datei `testcases.tsv` 1900 Testfälle, die Sie zum Testen verwenden sollen. Jede Zeile enthält Tabulator-separiert zwei kurze Sequenzen und die entsprechende Edit-Distanz für die Einheitskostenfunktion.

Aufgabe 2.2 Sei $S(n)$ die Anzahl der potentiell möglichen RNA-Sekundärstrukturen einer Sequenz der Länge $n \geq 1$. Die minimale Länge eines Hairpin-Loops soll 1 sein. Entwickeln Sie eine Rekurrenz für $S(n)$. Implementieren Sie diese Rekurrenz durch eine rekursive Funktion (DP-Verfahren ist nicht notwendig) und bestimmen Sie die Anzahl der Strukturen der Länge n, für $0 \leq n \leq 16$.

Aufgabe 2.3 Ein Punkt-Klammer-String (auch Vienna-Notation genannt) ist eine einfache Repräsentation der Sekundärstruktur einer RNA-Sequenz. Die Vienna-Notation besteht aus den drei Zeichen `.`, `(` und `)`. Zu jeder geöffneten Klammer `(` gibt es eine dazugehörige geschlossene Klammer `)`. Jedes Klammerpaar steht für ein Paar von gepaarten Positionen. Das Zeichen `.` repräsentiert eine ungepaarte Basenposition. Als Beispiel repräsentiert der Vienna-Notation-String

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

die RNA-Sekundärstruktur

$$\{(2, 22), (3, 21), (5, 11), (6, 10), (13, 19), (14, 18)\}$$

für eine Sequenz der Länge 22. Implementieren Sie nun zwei Programme `vienna2pairlist` und `pairlist2vienna`. Das erste Programm liefert für eine Vienna-Notation die Länge der repräsentierten Sequenz und die geordnete Folge der Basenpaar-Positionen. Das zweite Programm liefert für eine Sequenzlänge und eine geordnete Folge von Basenpaar-Positionen die entsprechende Vienna-Notation. Testen Sie Ihre Implementierung, in dem Sie für alle Vienna-Notationen aus der Datei `Vienna-examples.txt` (in STiNE verfügbar) die entsprechende Folge von Basenpaar-Positionen und die Länge berechnen, um daraus dann die Vienna-Notation zu bestimmen. Diese muss mit dem Original übereinstimmen.

Die Lösungen zu diesen Aufgaben werden am 05.05.2014 besprochen.