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

Grundlagen der Sequenzanalyse
Wintersemester 2022/2023
Übungen zur Vorlesung: Ausgabe am 29.11.2022

Aufgabe 6.1 (4 Punkte) In Aufgabe 5.1 wurde ein Python-Programm zur Berechnung der Editdistanz-Matrix E_δ und der Edit-Distanz implementiert. Dabei war δ die Einheitskostenfunktion. Entwickeln Sie aus der Funktion `fillDPtable` aus Aufgabe 5.1 eine Funktion `fillDPtable_minedges`, so dass jeder Eintrag der berechneten Matrix nicht nur den entsprechenden Distanzwert, sondern auch eine Repräsentation der eingehenden minimierenden Kante(n) speichert. In der Vorlesung wurde gezeigt, wie man diese Information in 3 Bits speichern kann. Sie können jedoch auch eine Liste von drei boolschen Werten (`True` oder `False`) verwenden. Jeder Wert in der Matrix ist damit ein Paar von zwei Werten, dem entsprechenden Distanzwert und den drei Bits. In einer späteren Aufgabe werden wir die zusätzliche Information für die Rekonstruktion von Alignments verwenden. Anstatt Ihre eigene Lösung zu erweitern, können Sie auch eine Erweiterung der Musterlösung, siehe Materialien, vornehmen.

Ergänzen Sie Ihre Implementierung um eine Funktion `show_matrix` zur Ausgabe der Matrix im Textformat. D.h. für jeden Matrix-Eintrag wird der Distanzwert und die Werte der 3 Bits ausgegeben (siehe auch das entsprechende Beispiel aus der Vorlesung). Dabei ist die Reihenfolge DIR, d.h. das erste Bit (bzw. der boolsche Wert) bezieht sich auf die eingehende Löschkante, das zweite auf die eingehende Einfügekante und das dritte Bit auf die eingehende Ersetzungskante.

Benutzen Sie die Funktion `fillDPtable_minedges` und `show_matrix` schließlich in einem Programm `editgraph.py`, das zwei Sequenzen als Kommandozeilenparameter erhält und deren Edit-Distanz sowie die Matrix im Textformat ausgibt. Hier ist die Ausgabe für die beiden Sequenzen acg und agc aus der Vorlesung. Die Werte in jeder Zeile sind durch Tabulatoren separiert.

```
acg agc 2
0/000 1/010 2/010 3/010
1/100 0/001 1/010 2/010
2/100 1/100 1/001 1/001
3/100 2/100 1/001 2/111
```

Die Materialien enthalten ein Testskript sowie die erwarteten Ergebnisse für drei Paare von kurzen Sequenzen. Durch den Aufruf von `make test` verifizieren Sie die Korrektheit Ihrer Implementierung bzgl. der Testdaten.

Punkteverteilung:

- 3 Punkte: funktionierende Implementierung von `fillDPtable_minedges`.
- 1 Punkt: funktionierende Implementierung von `show_matrix`.

Aufgabe 6.2 (8 Punkte) Seien u und v zwei Sequenzen der Länge m bzw. n. Eine gemeinsame Subsequenz von u und v ist eine Folge $(i_1, j_1), \ldots, (i_r, j_r)$, so dass gilt:

1. $1 \leq i_1 < \ldots < i_r \leq m$ und

2. $1 \leq j_1 < \ldots < j_r \leq n$ und
3. $u[i_1] = v[j_1], \ldots, u[i_r] = v[j_r]$.

Bitte beachten Sie, dass jede *gemeinsame* Subsequenz von u und v auch eine Subsequenz von u und v (wie in der Vorlesung definiert) ist. Beim Begriff gemeinsame Subsequenz ist zusätzlich gefordert, dass die Zeichen der beiden Sequenzen an den Positionen der Subsequenz identisch sind, siehe Bedingung 3.

Beispiel: Sei $u = acatcagac$ und $v = aactacgc$. Die Länge der längsten gemeinsamen Subsequenz von u und v ist 6. Z.B. ist $(1,1), (2,3), (3,5), (5,6), (7,7), (9,8)$ eine längste gemeinsame Subsequenz von u und v. Sie repräsentiert den String $acacgc$.

Entwickeln Sie einen Algorithmus zur Berechnung der Länge der längsten gemeinsame Subsequenz von zwei Sequenzen u und v der Länge m bzw. n. Verfolgen Sie den gleichen Ansatz wie bei der Entwicklung des Algorithmus zur Berechnung der Edit Distanz. D.h. überlegen Sie sich zunächst wie ein Graph zur Lösung des genannten Problems aussieht. Wir nennen diesen Graphen LCS-Graph. Beantworten Sie dazu die folgenden Fragen:

1. Aus welchen Knoten und Kanten besteht der LCS-Graph?
2. Wie werden die Kanten des LCS-Graphen bewertet?
3. Aus welchen Pfaden im LCS-Graph ergibt sich die Länge einer längsten gemeinsamen Subsequenz.

Leiten Sie aus den Antworten eine Matrix LCS ab. Spezifizieren Sie die Anzahl der Einträge der Matrix und die Eigenschaft eines einzelnen Eintrags in der Matrix (bitte formal definieren). Geben Sie eine Rekurrenz zur Berechnung der Matrixeinträge an, analog zu den Beispielen aus der Vorlesung.

Nachdem Sie als Teil der Lösung in einer Datei `lcslength_method.tex` diese Fragen beantwortet haben, implementieren Sie in einer Datei `lcslength.py` eine Funktion `lcslength`. Diese hat genau zwei Argumente, nämlich zwei Strings und liefert die Länge der längsten gemeinsamen Subsequenz dieser Strings.

Die Materialien enthalten ein Hauptprogramm, das Ihre Funktion aufruft und die Ergebnisse mit Referenzergebnissen vergleicht. Durch den Aufruf von `make test_small` und `make test_large` verifizieren Sie die Korrektheit Ihres Algorithmus für zwei verschiedene Testdatensätze. Durch `make test` werden beide Tests durchgeführt.

Optimierung:

Vereinfachen Sie Ihre Rekurrenz so weit wie möglich. Sie könnten z.B. Kanten im LCS-Graph identifizieren, die für die Berechnung der optimalen Lösung nicht notwendig sind.

Punkteverteilung:

- Antworten auf die Fragen 1.-3.: jeweils 0.5 Punkte
- Spezifikation der Matrix inklusive Rekurrenz: 2.5 Punkte
- Implementierung der Funktion `lcslength`: 1 Punkt

- bestandene Tests (jeweils 0.5 Punkte)
- Optimierung (2 Punkte)

Bitte die Lösungen zu diesen Aufgaben bis zum 04.12.2022 um 22:00 Uhr an gsa@zbh.uni-hamburg.de schicken.