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

Grundlagen der Sequenzanalyse
Wintersemester 2022/2023
Übungen zur Vorlesung: Ausgabe am 17.01.2023

Aufgabe 11.1 (2 Punkte)

Aus Zeitgründen können wir in der Vorlesung das Thema „Searching position specific scoring matrices“ nicht besprechen. Dieser Abschnitt wird daher im Peer-Teaching Format behandelt.

Für diese Übungsaufgabe muss sich aus jeder Kleingruppe ein Student bzw. eine Studentin anhand der Vorlesungsfolien (siehe `Vorlesung/pssm_slides.pdf`) auf dieses Thema vorbereiten. Sie sollten untereinander frühzeitig klären, wer diese Vorbereitung in dieser Woche übernimmt.

Das in der Vorbereitung erworbene Wissen soll an die anderen Mitglieder der jeweiligen Kleingruppe dadurch weitergegeben werden, dass man zusammen den Inhalt der Folien zum genannten Thema bespricht. Das kann entweder in den ersten 20-25 Minuten der Übung erfolgen oder zu einem anderen Zeitpunkt rechtzeitig vor Abgabe des Übungsblattes.

Der oder die Studierende, die sich vorbereitet hat, kann ggf. bei Unklarheiten Fragen, zumindest zu den Grundbegriffen, beantworten oder Erläuterungen mündlich ergänzen. Falls Sie den in den Folien dargestellten Programmcode ausprobieren möchten, können Sie auf die Materialien zurückgreifen.

Nach Bearbeitung dieser Aufgabe dokumentiert jede Kleingruppe in der Textdatei `bearbeitung.txt` in höchstens 15 Zeilen mit maximal 80 Zeichen pro Zeile ihr Vorgehen bei der Erarbeitung des Themas und ggf. noch bestehende Verständnisfragen oder Hinweise zu Unklarheiten in den Folien. Willkommen sind natürlich auch Bemerkungen zur Lehrform selbst. Es soll nicht der Inhalt der Folien wiedergegeben werden. Selbstverständlich müssen Sie in der Datei auch die üblichen Metadaten angeben.

Aufgabe 11.2 (7 Punkte) In der Vorlesung wurde der Begriff der k-Umgebung $Env_k(w)$ für eine Sequenz w der Länge q und eine gegebene Scorefunktion σ definiert. σ liefert dabei jeweils den Score für eine Ersetzungsoperation. In der Vorlesung wurde ein Algorithmus skizziert, der die Umgebung für ein gegebenes q-Wort w berechnet. Dieser Algorithmus basiert auf einer Tiefensuche in einem Trie mit Knoten, deren maximale Tiefe q ist. Ein solcher Trie muss jedoch nicht explizit im Speicher konstruiert werden. Er ist damit virtuell. Im genannten Algorithmus reicht ein Stack für die noch zu erfüllenden Aufgaben (d.h. die noch zu besuchenden Knoten im virtuellen Trie) für die Konstruktion der Umgebung für w. Jedes Element des Stacks hat dabei drei Werte:

- Die Tiefe q' eines Knotens im virtuellen Trie. Es gilt $0 \leq q' \leq q$.
- Der String u der Länge q', der durch diesen Knoten repräsentiert wird.
- Der Score $s = \sum_{i=1}^{q'} \sigma(w[i], u[i])$.

Der Algorithmus erzeugt aus den 3-Tupeln (q', u, s) mit $q' < q$ solche mit der Tiefe $q' + 1$, indem nacheinander jeweils ein Zeichen des Alphabets an u angehängt und der neue entsprechende Score zu s hinzugefügt wird. Falls ein 3-Tupel der Tiefe q mit einem Score $\geq k$ entsteht, wird dieses als Element der Umgebung weiter prozessiert (z.B. ausgegeben oder gespeichert).

Falls im Algorithmus ein 3-Tupel (q', u, s) mit $q' < q$ entsteht, wird überprüft, ob u zu einem Element der Umgebung verlängert werden kann. Dazu verwendet man die Technik des Lookahead-Scoring. Dazu muss man für den verbleibenden Suffix $w[q'+1 \ldots q]$ (Indizierung ab 0) von w ermitteln, was der maximale Score $mss(q')$ ist, den dieser Suffix mit einem String der Länge $q-q'$ über dem gegebenen Alphabet erreichen kann. Da dieser maximale Score für alle Suffixe von w nur von w und σ abhängt, kann er einmal vorberechnet und gespeichert werden.

Fall 1: Wenn $s + mss(q') < k$ (oder äquivalent $s < k - mss(q')$) ist, muss man das 3-Tupel (q', u, s) nicht weiter betrachten.

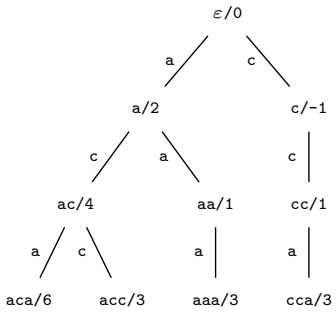
Fall 2: Wenn $s + mss(q') \geq k$ (oder äquivalent $s \geq k - mss(q')$) ist, muss man das 3-Tupel (q', u, s) auf den Stack legen.

Der Algorithmus stoppt, sobald der Stack leer ist.

Beispiel (siehe Beispiel im Kapitel über Blast aus der Vorlesung): Sei $\mathcal{A} = \{a, c\}$, $q = 3$, $k = 3$ und σ die Einheitsscore-Funktion mit $\sigma(a,a) = \sigma(c,c) = 2$ und $\sigma(a,c) = \sigma(c,a) = -1$. Sei $w = aca$. Mit dem Suffix ca von aca (d.h. für $q' = 1$) erreicht man maximal den Score 4, mit dem Suffix a (d.h. für $q' = 2$) erreicht man maximal der Score 2 und mit dem leeren Suffix (d.h. für $q' = 3$) erreicht man maximal den Score 0.

Durch Subtraktion dieser 3 Werte von k erhält man die Liste $[3-4, 3-2, 3-0] = [-1, 1, 3]$ der Zwischenschwellwerte. Hierbei ergeben sich die Zwischenschwellwerte aus dem Ausdruck $k - mss(q')$ für alle echten Suffixe von w.

Für $w = aca$ sieht der virtuelle Trie nun so aus:



Es ergeben sich damit die folgenden 3-Tupel, die zu den Knoten des Tries korrespondieren:

$(0, \varepsilon, 0)$, $(1, a, 2)$, $(2, ac, 4)$, $(3, aca, 6)$, $(3, acc, 3)$, $(2, aa, 1)$, $(3, aaa, 3)$, $(1, c, -1)$, $(2, cc, 1)$, $(3, cca, 3)$.

Diese stehen natürlich nicht alle gleichzeitig auf dem Stack, sondern werden nacheinander aus $(0, \varepsilon, 0)$ erzeugt.

Offensichtlich fehlen im obigen Trie die Knoten für die Strings aac, ccc, ca, caa und cac mit den Scores 0, -1, -2, 0, und -3. Diese sind nicht in der k-Umgebung enthalten und lassen sich nicht zu einem Element der Umgebung erweitern. So erreicht ca den Score -2, der aber kleiner ist als der obige Zwischenschwellwert 1. Damit wird $(2, ca, -2)$ nicht auf den Stack gelegt und die 3-Tupel $(3, caa, 0)$ und $(3, cac, -3)$ werden nicht generiert.

Implementieren Sie nun in Python in einer Datei `kenv_enum.py` den hier beschriebenen Algorithmus durch die zwei folgenden Funktionen:

- `thresholds_get(scorematrix,k,w)` berechnet die Liste der Zwischenschwellwerte und liefert sie durch eine **`return`**-Anweisung zurück. 2 Punkte

- `enum_environment(scorematrix,k,w)` ist ein Generator, der die Elemente u der `k`-Umgebung von `w` aufzählt und zusammen mit dem entsprechenden Score durch eine `yield`-Anweisung an den aufrufenden Kontext liefert. 4 Punkte

Für die Parameter der beiden Funktionen gilt:

- `scorematrix` ist eine Instanz der Klasse `Scorematrix` aus dem Modul `scorematrix.py`. Diese Instanz wird in den Unittests erzeugt. `scorematrix.characters_get()` liefert die Liste der Zeichen des Alphabets $\mathcal{A}$. `scorematrix.getscore(c,b)` liefert den Score der Ersetzung des Zeichen `c` durch das Zeichen `b`.
- `k` ist die Scoreschranke.
- `w` das q-Wort.

Im Material zu dieser Übung finden Sie Unittests. Durch `make test` werden diese Tests durchgeführt. Für bestandene Tests wird ein Punkt vergeben.

Aufgabe 11.3 (5 Punkte) In dieser Aufgabe geht es um die Implementierung einer Variante des Smith-Waterman Algorithmus zur Berechnung des lokal optimalen Alignment-Scores inklusive der Koordinaten von Substrings, die diesen Score erreichen.

Benennen Sie die Datei `swalign_template.py` in `swalign.py` um und implementieren Sie die folgenden zwei Funktionen:

- `swcoords_single_entry(indelscore,repscore,curr,we,nw,no)` : berechnet in der Matrix L_σ einen neuen Eintrag $L_\sigma(i,j)$ für $i,j > 0$. Die Werte des Eintrags werden in `curr` gespeichert, so dass die Funktion keinen Rückgabewert hat. `we`, `nw` und `no` sind die Einträge in Matrix L_σ im Westen (d.h. $L_\sigma(i,j-1)$), im Nordwesten (d.h. $L_\sigma(i-1,j-1)$), und im Norden (d.h. $L_\sigma(i-1,j)$). `indelscore` ist der negative Score für eine Löschung bzw. eine Einfügung eines einzelnen Zeichens. In diesem Kontext werden also lineare Gapkosten verwendet. `repscore` ist der Score für die entsprechende Replacement-Operation des i-ten und j-ten Zeichens der alignierten Sequenzen. Beachten Sie, dass `curr`, `we`, `nw` und `no` Instanzen der Klasse `LMatrixEntry` sind (siehe unten). Auf die entsprechenden Member-Variablen können Sie zugreifen.
- `better(self,entry)` ist eine Methode der Klasse `BestCoordinate` (siehe unten). Es vergleicht einer Instanz `entry` der Klasse `LMatrixEntry` mit der entsprechenden `BestCoordinate`-Instanz `self` und liefert genau dann `True` zurück, wenn `entry` besser ist als `self`. Was das genau heißt, ist in `swalign.py` definiert.

Eine Instanz der Klasse `BestCoordinate` speichert die folgenden Werte:

1. den Score des bisher besten lokalen Alignments,
2. die Zeile i und Spalte j der L_σ-Matrix, aus der der bisher beste Eintrag stammt,
3. die Länge des alignierten Substrings in der Sequenz u und
4. die Länge des alignierten Substrings in der Sequenz v

Eine Deklaration der Klasse `BestCoordinate` finden Sie (nach der Umbenennung) in der Datei `swalign.py`.

Diese enthält auch eine Deklaration einer Klasse `LMatrixEntry`. Die Instanz dieser Klasse für den Eintrag in Zeile i und Spalte j der Matrix L_σ enthält drei Werte:

Abbildung 1: Matrix mit Werten des Typs `LMatrixEntry` für die Sequenzen `useq=PQRAFADCSTVQ` und `vseq=FYAFDACSLL` und die Scorematrix aus der Vorlesung. Der erste Wert jedes Tripletts ist der Score-Wert, die anderen beiden Werte sind die Werte von `aligned_u` und `aligned_v`. Der maximale Score-Wert 8 in Zeile 9 und Spalte 8 ist fett gedruckt. Die beiden Werte `aligned_u=6` und `aligned_v=6` geben also jeweils die Länge der alignierten Suffixe von `useq[:9+1]=PQRAFADCS` bzw. `useq[:8+1]=FYAFDACS` an. Diese Länge ist jeweils 6, d.h. ein gobales Alignment der Sequenzen `AFADCS` und `AFDACS` hat den Score 8. Der entsprechende Pfad in der L_σ-Matrix beginnt damit bei (i', j') mit $i' = 9 - 6 = 3$ und $j' = 8 - 6 = 2$, also bei dem Eintrag, dessen Score-Wert 0 fett gedruckt ist.

$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$
$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$
$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$
$(0,0,0)$	$(0,0,0)$	$(\mathbf{0},0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$
$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(2,1,1)$	$(1,1,2)$	$(0,0,0)$	$(2,1,1)$	$(1,1,2)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$
$(0,0,0)$	$(2,1,1)$	$(1,1,2)$	$(1,2,1)$	$(4,2,2)$	$(3,2,3)$	$(2,2,4)$	$(1,2,5)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$
$(0,0,0)$	$(1,2,1)$	$(0,0,0)$	$(3,2,3)$	$(3,3,2)$	$(2,3,3)$	$(5,3,4)$	$(4,3,5)$	$(3,3,6)$	$(2,3,7)$	$(1,3,8)$
$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(2,3,3)$	$(2,4,2)$	$(5,4,3)$	$(4,4,4)$	$(3,4,5)$	$(2,4,6)$	$(1,4,7)$	$(0,0,0)$
$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(1,4,3)$	$(1,5,2)$	$(4,5,3)$	$(3,5,4)$	$(6,5,5)$	$(5,5,6)$	$(4,5,7)$	$(3,5,8)$
$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(3,6,3)$	$(2,6,4)$	$(5,6,5)$	$(\mathbf{8},6,6)$	$(7,6,7)$	$(6,6,8)$
$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(2,7,3)$	$(1,7,4)$	$(4,7,5)$	$(7,7,6)$	$(6,7,7)$	$(5,7,8)$
$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(1,8,3)$	$(0,0,0)$	$(3,8,5)$	$(6,8,6)$	$(5,8,7)$	$(4,8,8)$
$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(0,0,0)$	$(2,9,5)$	$(5,9,6)$	$(4,9,7)$	$(3,9,8)$

- der Score $L_\sigma(i, j)$ wird in der Instanzvariable `score` gespeichert.
- die beiden Längen `aligned_u` und `aligned_v` von alignierten Suffixen von `useq[:i+1]` bzw. `vseq[:j+1]`, deren optimales globales Alignment den Score $L_\sigma(i, j)$ hat. Der Score-Wert ergibt sich aus der Rekurrenz für den Smith-Waterman Algorithmus. Die Längenwerte ergeben sich aus den maximierenden Kanten im impliziten Edit-Graphen für lokale Alignments. Falls ein Score-Wert 0 ist, dann sind auch die beiden Längenwerte 0.

In Abbildung 1 finden Sie ein Beispiel für eine entsprechende Matrix.

In `swalign.py` finden Sie bereits eine Funktion `swcoords`, die die beiden oben genannten Funktionen aufruft. Diese Funktion darf nicht verändert werden, aber Sie müssen sich diese Funktion genauer anschauen. Dokumentieren Sie daher die einzelnen Schritte in der Funktion an den Stellen, an denen in der Funktion bereits ein Kommentar eingefügt ist.

Hinweis: Der Vorteil der Technik, die die Längen der alignierten Suffixe zusammen mit dem Score protokolliert, besteht darin, dass die Matrix in linearem Speicherplatz berechnet werden kann. Sie hatten bereits in Aufgabe 4.3 eine Rekurrenz für die Berechnung einer Matrix entwickelt, die zu

jedem Eintrag in L_σ den am weitesten entfernten Ursprung berechnet. Dieser wird durch die Werte der Member-Variablen der Klasse `LMatrixEntry` repräsentiert.

Damit die Lösungen, die Ihr Programm berechnet, eindeutig werden, maximieren Sie bei gleichen Score-Werten die Summe der Längen der alignierten Suffixe. Bei gleichen Score-Werten und Längensummen werden Ersetzungen vor Löschungen und Löschungen vor Einfügungen präferiert.

Im Material zu dieser Übung finden Sie die Dateien mit Sequenzen und Score-Matrizen sowie ein Hauptprogramm `swalign_mn.py`, das Ihre Implementierung der Funktion `swcoords` aufruft. Durch `make test` verifizieren Sie, dass die Ergebnisse für einige Testbeispiele korrekt sind.

Punkteverteilung:

- 2 Punkte für `swcoords_single_entry`
- 1 Punkt für `better`
- 1 Punkt für die Dokumentation
- 1 Punkt für bestandene Tests

Bitte die Lösungen zu diesen Aufgaben bis zum 22.01.2023 um 22:00 Uhr an gsa@zbh.uni-hamburg.de schicken.