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

Grundlagen der Sequenzanalyse
Wintersemester 2022/2023
Übungen zur Vorlesung: Ausgabe am 15.11.2022

Aufgabe 4.1 (6 Punkte) Seien u und v zwei beliebige Sequenzen der Länge m bzw. n. Für alle $i, 0 \leq i \leq m$ und $j, 0 \leq j \leq n$ bezeichne $Aligns(i,j)$ die Anzahl der Alignments von $u[1 \ldots i]$ und $v[1 \ldots j]$. In der Vorlesung wurde die folgende Rekurrenz für $Aligns$ entwickelt:

$$Aligns(i,j) = \begin{cases} 1 & \text{falls } i = 0 \text{ oder } j = 0 \\ Aligns(i-1,j)+ & \\ Aligns(i,j-1)+ & \\ Aligns(i-1,j-1) & \text{sonst} \end{cases}$$

Teil 1

Schreiben Sie eine Python-Funktion `aligns_rec`, die nach obiger Rekurrenz den Wert $Aligns(m,n)$ für zwei natürliche Zahlen m und n berechnet. Die Funktion soll rekursiv sein, d.h. sich selbst aufrufen.

Hinweis für alle Studierenden, die noch nicht viel Erfahrung in Python haben: In Python kann man eine Funktion `f` mit zwei Parametern `a` und `b` durch den Funktionskopf **`def`** `f(a,b):` definieren. Diesem folgt ein Block mit Anweisungen, die eingerückt sind. In diesen Anweisungen kann man auf die Parameter `a` und `b` zugreifen. Eine Funktion kann mit einer **`return`**-Anweisung Funktionswerte zurückliefern. Details finden Sie in den Folien zur Vorlesung PfN1 (Datei `python-slides.pdf`) in Abschnitt 15 ab frame 199. Alle Studierenden, die nicht an PfN1 teilnehmen, können bei Bedarf unter

`https://attachment.rrz.uni-hamburg.de/4a2a6fa6/python-slides.pdf` auf diese Datei zugreifen.

In den Materialien finden Sie eine Datei `aligns_template.py`. Bitte benennen Sie diese in `aligns.py` um und fügen hier Ihre rekursive Funktion ein. Die Datei enthält ein Hauptprogramm, das Werte für m und n von der Kommandozeile liest, `aligns_rec` aufruft und das Ergebnis ausgibt.

Teil 2

Schreiben Sie nun eine Python-Funktion `aligns`, die $Aligns(m,n)$ nach der obigen Rekurrenz mit Hilfe einer $(m+1) \times (n+1)$-Matrix $Alignstab$ berechnet und mit einer **`return`**-Anweisung den Wert $Alignstab[m,n]$ zurückliefert. Dabei gilt die Gleichung

$$Alignstab[i,j] = Aligns(i,j)$$

für alle i, j, $0 \leq i \leq m$, $0 \leq j \leq n$.

Beispiel: Hier sehen Sie die Matrix $Alignstab$ für $m = 4$ und $n = 5$. Daher ist $Aligns(4, 5) = 681$.

1	1	1	1	1	1
1	3	5	7	9	11
1	5	13	25	41	61
1	7	25	63	129	231
1	9	41	129	321	681

Bitte beachten Sie, dass das Füllen einer Matrix nach einer Rekurrenz *nicht* zwangsweise bedeutet, dass die Matrixwerte durch rekursive Funktionsaufrufe gefüllt wird. Es ist bei dieser Teilaufgabe eine *nicht*-rekursive Lösung gefordert. In Python3 kann man eine $(m + 1) \times (n + 1)$ Matrix `tab`, in der alle Werte auf `None` gesetzt werden (d.h. undefiniert sind), durch die folgende Anweisung berechnen:

```
tab = [[None] * (n+1) for i in range(m+1)]
```

Der Ausdruck `tab[i][j]` bezeichnet dann den Eintrag in Zeile i und Spalte j in `tab`, wobei ab 0 indiziert wird.

Teil 3

Durch `make test` verifizieren Sie, das Ihr Programm für alle m, $0 \leq m \leq 9$ und alle n, $0 \leq n \leq 9$ die richtigen Werte berechnet.

Teil 4

In der Vorlesung wurde gezeigt, wie man die Anzahl der Alignments von zwei Sequenzen der Länge n abschätzen kann. Ermitteln Sie die Qualität dieser Abschätzung, in dem Sie mit einem Python-Programm `aligns_estim.py` für alle n, $1 \leq n \leq 100$ den Wert von $Aligns(n, n)$, den Schätzwert $estim(n)$ für n und den relativen Fehler

$$\left| 1 - \frac{estim(n)}{Aligns(n, n)} \right|$$

berechnen und ausgeben. Damit Sie die Funktion `aligns` nicht kopieren müssen, verwenden Sie in `aligns_estim.py` die folgende **`import`**-Anweisung:

```
from aligns import aligns
```

Zudem müssen Sie das Modul `math` importieren, da Sie für den Schätzwert die Quadratwurzel berechnen müssen. Geben Sie die Werte zeilenweise aus. Für die Zählwerte verwenden Sie wissenschaftliche Notation mit zwei Nachkommastellen. Geben Sie den relativen Fehler ganzzahlig aus. Die ersten fünf Zeilen der Ausgabe könnten folgendermaßen aussehen:

```
n Aligns(n,n) estim(n) rel. error
1 3.00e+00 1.41e+01 3
2 1.30e+01 1.16e+02 7
3 6.30e+01 8.28e+02 12
4 3.21e+02 5.57e+03 16
```

Speichern Sie Ihre Werte in einer Tab-separierten Datei `aligns_estimate.tsv`. Ermitteln Sie, ob die Werte von $Aligns(n, n)$ über- oder unterschätzt werden. Betrachten Sie die Abhängigkeit des relativen Fehlers von n. Hierfür wäre es sinnvoll, die relativen Fehler in Abhängigkeit von n zu plotten. Das müssen Sie nicht unbedingt in Python implementieren, sondern können auch ein Software-Werkzeug Ihrer Wahl verwenden. Wie kann man die Abhängigkeit mathematisch einfach beschreiben?

Punktevergabe:

- Jeweils 2 Punkte für die korrekte rekursive und iterative Funktion.
- 1 Punkte für die Erstellung der Tabelle aus Teil 4
- 1 Punkte für eine nachvollziehbare Beschreibung der Werte aus Teil 4.

Aufgabe 4.2 (3 Punkte)

Überlegen Sie sich eine speichereffiziente Datenstruktur zur Repräsentation vieler lokaler Alignments der gleichen Sequenzen. Ihre Datenstruktur für ein einzelnes Alignment kann z.B. auf Strings oder Listen von Zeichen oder ganzen Zahlen basieren. Gehen Sie davon aus, dass viele Substrings in den Sequenzen mehrfach aligniert werden und dass die Sequenzen bereits im Speicher vorliegen. D.h. es ist nicht notwendig, die Substrings für jedes Alignment nochmals zu speichern. Beschreiben Sie die Datenstruktur informell. Begründen Sie Ihre Entscheidung bezüglich der Datenstruktur und diskutieren Sie ihre Vor- und Nachteile. Wie wird mit Ihrer Repräsentation ein Alignment im Allgemeinen gespeichert? Mit welchem Speicherplatzaufwand geschieht dies? Gibt es Redundanzen in der Darstellung?

Wie sieht in Ihrer Datenstruktur eine konkrete Repräsentation der folgenden Alignments aus:

```
acgtaga--tataga-gat          acgtaga----tatagag-at
|||  |||   ||  |||  | |      |||  | |     ||  |||||  |
acgaagaggta-agagggt          acg-a-agaggta-agagggt
```

Aufgabe 4.3 (4 Punkte) In der Vorlesung wurde der Smith-Waterman Algorithmus zur Berechnung optimaler lokaler Alignments von zwei Sequenzen u und v vorgestellt. Der Algorithmus beruht auf der Berechnung einer $(m + 1) \times (n + 1)$-Matrix loc_σ nach der folgenden Rekurrenz:

$$loc_\sigma(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \max \left\{ \begin{array}{ll} 0 & \\ loc_\sigma(i-1,j) & + \sigma(u[i] \to \varepsilon) \\ loc_\sigma(i,j-1) & + \sigma(\varepsilon \to v[j]) \\ loc_\sigma(i-1,j-1) & + \sigma(u[i] \to v[j]) \end{array} \right\} & \text{otherwise} \end{cases}$$

Dabei ist $m = |u|$, $n = |v|$ und σ eine Score-Funktion. Zur Berechnung eines optimalen lokalen Alignments bestimmt man einen maximierenden Knoten bzgl. loc_σ, d.h. ein (i, j), so dass

$loc_\sigma(i,j)$ maximal ist. Dann bestimmt man ausgehend von (i,j) rückwärts einen Pfad entlang maximierender Kanten, bis man das erste Mal den Wert 0 in loc_σ erreicht. Dieses Verfahren benötigt $O(mn)$ Speicher, da für das Traceback in jedem Matrixeintrag eine konstante Anzahl von Werten gespeichert werden muss.

Ein alternatives Verfahren berechnet für jeden Knoten im Editgraphen $G(u,v)$ seinen Ursprung. Dieser ist wie folgt definiert:

Sei (i,j) ein Knoten in $G(u,v)$. Der Knoten (i',j') aus $G(u,v)$ ist ein *Ursprung von* (i,j) *in* loc_σ, falls es einen maximierenden Pfad von (i',j') nach (i,j) gibt, so dass (i',j') der einzige Knoten auf diesem Pfad ist, für den $loc_\sigma(i',j') = 0$ gilt.

Nach dieser Definition gilt für den Knoten (i,j) mit $loc_\sigma(i,j) = 0$, dass (i,j) der einzige Ursprung von (i,j) ist. Der am weitesten entfernte Ursprung von (i,j) ist ein Ursprung (i',j') von (i,j), so dass $i' + j' \leq i'' + j''$ für jeden Ursprung (i'',j'') von (i,j) gilt.

Entwickeln Sie eine Rekurrenz für eine $(m+1) \times (n+1)$-Matrix MO, so dass für alle (i,j), mit $0 \leq i \leq m$, $0 \leq j \leq n$, $MO(i,j)$ ein am weitesten entfernter Ursprung von (i,j) ist. Bei Ihrer Rekurrenz müssen Sie auf loc_σ Bezug nehmen.

Beispiel: Hier sehen Sie eine Matrix für die beiden Sequenzen $u =$ `PQRAFADCSTVQ`, $v =$ `FYAFDACSLL` und die Scorefunktion, die auch im Beispiel aus der Vorlesung verwendet wurde. In der dargestellten Matrix werden die Werte aus loc_σ und MO kombiniert, d.h. für alle (i,j) steht in Zeile i und Spalte j das Triplett $(loc_\sigma(i,j), i', j')$, wobei (i',j') ein am weitesten entfernter Ursprung von (i,j) ist.

	0	1	2	3	4	5	6	7	8	9	10
0	(0, 0, 0)	(0, 0, 1)	(0, 0, 2)	(0, 0, 3)	(0, 0, 4)	(0, 0, 5)	(0, 0, 6)	(0, 0, 7)	(0, 0, 8)	(0, 0, 9)	(0, 0, 10)
1	(0, 1, 0)	(0, 1, 1)	(0, 1, 2)	(0, 1, 3)	(0, 1, 4)	(0, 1, 5)	(0, 1, 6)	(0, 1, 7)	(0, 1, 8)	(0, 1, 9)	(0, 1, 10)
2	(0, 2, 0)	(0, 2, 1)	(0, 2, 2)	(0, 2, 3)	(0, 2, 4)	(0, 2, 5)	(0, 2, 6)	(0, 2, 7)	(0, 2, 8)	(0, 2, 9)	(0, 2, 10)
3	(0, 3, 0)	(0, 3, 1)	(0, 3, 2)	(0, 3, 3)	(0, 3, 4)	(0, 3, 5)	(0, 3, 6)	(0, 3, 7)	(0, 3, 8)	(0, 3, 9)	(0, 3, 10)
4	(0, 4, 0)	(0, 4, 1)	(0, 4, 2)	(2, 3, 2)	(1, 3, 2)	(0, 4, 5)	(2, 3, 5)	(1, 3, 5)	(0, 4, 8)	(0, 4, 9)	(0, 4, 10)
5	(0, 5, 0)	(2, 4, 0)	(1, 4, 0)	(1, 3, 2)	(4, 3, 2)	(3, 3, 2)	(2, 3, 2)	(1, 3, 2)	(0, 5, 8)	(0, 5, 9)	(0, 5, 10)
6	(0, 6, 0)	(1, 4, 0)	(0, 6, 2)	(3, 4, 0)	(3, 3, 2)	(2, 3, 2)	(5, 3, 2)	(4, 3, 2)	(3, 3, 2)	(2, 3, 2)	(1, 3, 2)
7	(0, 7, 0)	(0, 7, 1)	(0, 7, 2)	(2, 4, 0)	(2, 3, 2)	(5, 3, 2)	(4, 3, 2)	(3, 3, 2)	(2, 3, 2)	(1, 3, 2)	(0, 7, 10)
8	(0, 8, 0)	(0, 8, 1)	(0, 8, 2)	(1, 4, 0)	(1, 3, 2)	(4, 3, 2)	(3, 3, 2)	(6, 3, 2)	(5, 3, 2)	(4, 3, 2)	(3, 3, 2)
9	(0, 9, 0)	(0, 9, 1)	(0, 9, 2)	(0, 9, 3)	(0, 9, 4)	(3, 3, 2)	(2, 3, 2)	(5, 3, 2)	(**8**, 3, 2)	(7, 3, 2)	(6, 3, 2)
10	(0, 10, 0)	(0, 10, 1)	(0, 10, 2)	(0, 10, 3)	(0, 10, 4)	(2, 3, 2)	(1, 3, 2)	(4, 3, 2)	(7, 3, 2)	(6, 3, 2)	(5, 3, 2)
11	(0, 11, 0)	(0, 11, 1)	(0, 11, 2)	(0, 11, 3)	(0, 11, 4)	(1, 3, 2)	(0, 11, 6)	(3, 3, 2)	(6, 3, 2)	(5, 3, 2)	(4, 3, 2)
12	(0, 12, 0)	(0, 12, 1)	(0, 12, 2)	(0, 12, 3)	(0, 12, 4)	(0, 12, 5)	(0, 12, 6)	(2, 3, 2)	(5, 3, 2)	(4, 3, 2)	(3, 3, 2)

Der maximale Scorewert $8 = loc_\sigma(9,8)$ ist hervorgehoben und $(3,2)$ ist ein am weitesten entfernter Ursprung von $(9,8)$. Damit sind $u' = u[3+1 \ldots 9] =$ `AFADCS` und $v = v[2+1 \ldots 8] =$ `AFDACS` zwei Substrings von u bzw. v, deren optimales globales Alignment den Score 8 hat. D.h. dieses globale Alignment von u' und v' ist ein optimales lokales Alignment von u und v. □

Bitte die Lösungen zu diesen Aufgaben bis zum 20.11.2022 um 22:00 Uhr an gsa@zbh.uni-hamburg.de schicken.