ASE Übung 5 - Needleman / Wunsch Algorithmus

Einführung

Das Ziel dieser Übung ist die Implementation einer einfachen Version des *Needleman-Wunsch* Algorithmus, welcher für gegebene Parameter optimale Sequenzalignments zweier Sequenzen berechnet.

Der Algorithmus besteht aus 2 Teilen. Im ersten Teil wird eine Score-Matrix berechnet, welche für Teil-Alignments optimale Scores liefert. Im zweiten Teil wird aus den berechneten Daten das optimale Alignment rekonstruiert. Der erste Teil soll nun implementiert werden während der zweite bereits vorgegeben ist.

Es folgt eine Erklärung der Berechnung der Matrizen, zusätzliche Python-Syntax und Anweisungen zu Implementation und Testen des Algorithmus mit Hilfe von bereitgestellten Python-Skripten.

Score-Matrix

Die Grundidee des Algorithmus ist es für alle möglichen Paare von Präfixen der Sequenzen $(a[0..i], b[0..j])$ die optimalen Alignment-Scores zu berechnen. Für die Berechnung des Scores eines bestimmten Paars von Präfixen, werden die bereits berechneten Scores kürzerer Präfixe verwendet.

Es bietet sich an alle berechneten Scores in einer Matrix zu verwalten, deren Dimensionen $m + 1, n + 1$ betragen, wenn m und n die Längen der beiden Sequenzen sind. Von der zweiten Zeile an ist jede Zeile einem Element der ersten Sequenz zugeordnet. Das gleiche gilt für die Spalten bzw. die Elemente der zweiten Sequenz. Jedes Element der Matrix (i, j) soll dann den optimalen Alignment Score der Sequenzen bis zu den entsprechenden Positionen in den Sequenzen $(i - 1, j - 1)$ enthalten.

Die erste Zeile und Spalte der Matrix enthalten die Scores, die sich ergeben, wenn das Alignment bis zur jeweiligen Position nur aus Insertionen bzw. Deletionen besteht. Der Score für jede andere Position (i, j) ergibt sich aus dem Maximum dreier Summen:

- Dem Score an Position $(i - 1, j)$ und einer Deletion,
- dem Score an Position $(i, j - 1)$ und einer Insertion,
- oder dem Score an Position $(i-1, j-1)$ und dem Score der Substitution der Elemente der Sequenzen $a[i - 1]$ und $b[j - 1]$.

Gegeben seien die Parameter Match = 1, Mismatch = -1 und Gap = -2. Für zwei Beispielsequenzen (`ACGAT`, `CGCT`) geht der Algorithmus entsprechend folgendermaßen vor:

Zunächst werden die erste Zeile und Spalte der Matrix mit den kumulativen Gap-Scores gefüllt, da diese unabhängig von anderen Elementen berechnet werden können:

	-	C	G	C	T
-	**0**	**-2**	**-4**	**-6**	**-8**
A	**-2**				
C	**-4**				
G	**-6**				
A	**-8**				
T	**-10**				

Anschließend wird beginnend bei Position $(1, 1)$ zeilenweise für jede freie Position der Score als Maximum der oben genannten Summen berechnet und eingetragen:

	-	C	G	C	T
-	0	-2	-4	-6	-8
A	-2	**-1**			
C	-4				
G	-6				
A	-8				
T	-10				

	-	C	G	C	T
-	0	-2	-4	-6	-8
A	-2	-1	**-3**		
C	-4				
G	-6				
A	-8				
T	-10				

	-	C	G	C	T
-	0	-2	-4	-6	-8
A	-2	-1	-3	**-5**	
C	-4				
G	-6				
A	-8				
T	-10				

	-	C	G	C	T
-	0	-2	-4	-6	-8
A	-2	-1	-3	-5	**-7**
C	-4				
G	-6				
A	-8				
T	-10				

	-	C	G	C	T
-	0	-2	-4	-6	-8
A	-2	-1	-3	-5	-7
C	-4	**-1**			
G	-6				
A	-8				
T	-10				

	-	C	G	C	T
-	0	-2	-4	-6	-8
A	-2	-1	-3	-5	-7
C	-4	-1	**-2**		
G	-6				
A	-8				
T	-10				

Der optimale Score für das Alignment beider Sequenzen findet sich auf Position m, n der vollständig ausgefüllten Matrix:

	-	C	G	C	T
-	0	-2	-4	-6	-8
A	-2	-1	-3	-5	-7
C	-4	-1	-2	-2	-4
G	-6	-3	0	-2	-3
A	-8	-5	-2	-1	-3
T	-10	-7	-4	-3	**0**

Traceback-Matrix

Zur Rekonstruktion des Alignments ist es notwendig für jede Position zu wissen, ob diese durch einen Insertion-, Deletion- oder Substitution-Schritt erreicht wurde. Es bietet sich an, diese Information direkt bei der Erstellung der Score-Matrix zu speichern. Hierzu wird eine weitere Matrix (die *Traceback*-Matrix) benötigt, deren Elemente einen von drei Werten halten können, z.B. `'i'` für Insertion, `'d'` für Deletion oder `'s'` für Substitution.

Im Verlauf des Algorithmus können die erste Spalte und Zeile zunächst entsprechend mit Deletion bzw. Insertion Werten befüllt werden. Beim Berechnen der restlichen Scores kann abhängig davon, welche Summe den maximalen Wert ergeben hat, eine Deletion, Insertion oder Substitution eingetragen werden.

Das Verfahren zur Rekonstruktion des Alignments verläuft dann folgendermaßen: Für die beiden alinierten Sequenzen werden zunächst zwei leere Zeichenketten erzeugt. Beginnend bei der letzten Position der Traceback-Matrix ($i = m, j = n$ wird für das aktuelle Element geschaut, um welchen Eintrag es sich handelt.

Ist es eine Substitution, so werden den Zeichenketten die Elemente der jeweiligen Sequenzen an den aktuell betrachteten Positionen $a[i-1]$ und $b[i-1]$ (um 1 versetzt, da die Matrix eine Zeile und Spalte mehr hat als die Längen der Sequenzen) vorangestellt und anschließend die diagonale Nachbarposition $(i-1, j-1)$ betrachtet.

Ist es eine Deletion, so wird das aktuelle Element aus der ersten Sequenz der entsprechenden Zeichenkette vorangestellt, während der anderen Sequenz ein Lückensymbol (`'-'`) vorangestellt wird. Anschließend wird die Nachbarposition in der darüber liegenden Zeile betrachtet.

Für die Insertion wird das aktuelle Element der zweiten Sequenz seiner entsprechenden Zeichenkette vorangestellt, während die andere eine Lücke bekommt, und anschließend die Nachbarposition in der links liegenden Spalte betrachtet.

Das Verfahren wird durchgeführt, bis die Position $(i = 0, j = 0)$ erreicht wurde. Die erstellten Zeichenketten bilden dann das Alignment der beiden Sequenzen.

In folgender Traceback-Matrix ist der Pfad des Alignments der Sequenzen `ACGAT` und `GCGT` markiert:

```
  |  -  G  C  G  T
--+--------------
- |  -  i  i  i  i
A |  d  s  s  s  s
C |  d  s  s  i  i
G |  d  s  d  s  i
A |  d  d  s  d  s
T |  d  d  s  d  s
```

```
ACGAT
GCG-T
```

Python Syntax

Listen

Für die Implementation wird eine Matrix Datenstruktur benötigt, welche Zahlenwerte halten kann. Hierzu eignet sich die Liste. Eine Python Liste kann eine Anzahl beliebiger Werte halten, wobei die Reihenfolge relevant ist. Eine Liste kann leer oder bereits Werte beinhaltend erzeugt werden:

```
>>> an_empty_list = []
>>> a_list_of_integers = [4, 3, 2, 1, 0]
```

Auf Elemente der Liste lässt sich mit den Indexklammern zugreifen. Ebenso ist es möglich Werte in der Liste zu überschreiben:

```
>>> a_list_of_integers[1]
3
>>> a_list_of_integers[4] = 10
>>> a_list_of_integers
[4, 3, 2, 1, 10]
```

Da Listen beliebige Werte halten können, ist das Schachteln von Listen möglich. Einfach geschachtelte Listen lassen sich dann z.B. als Matrizen interpretieren:

```
>>> two_by_two_identity_matrix = [[1, 0], [0, 1]]
>>> two_by_two_identity_matrix[0][0]
1
>>> two_by_two_identity_matrix[0][1]
0
```

For-Schleife

Wie in der Beschreibung des Algorithmus angedeutet, ist es notwendig über eine Reihe von Zahlen zu iterieren. Dies ist mit der bereits bekannten `while`-Schleife möglich, wenn vorher eine Indexvariable und ein entsprechendes Abbruchkriterium definiert werden und die Indexvariable in jeder Iteration aktualisiert wird.

Oft ist es jedoch bequemer die `for`-Schleife zu nutzen, welche über Datenstrukturen, welche aus mehreren Elementen bestehen, iteriert und jeweils ein Element in jeder Iteration an eine lokale Variable bindet. Die Folgende `for`-Schleife gibt in jeder Iteration ein Zeichen der definierten Zeichenkette aus:

```
>>> some_string = "abc"
>>> for c in some_string:
...     print(c)
...
a
b
c
```

`for`-Schleifen lassen sich gut mit der `range()` Funktion kombinieren, welche für Zwei Ganzzahlparameter den Zahlenbereich zwischen den Zahlen (exklusiv der letzten) zurück gibt. So ist es möglich über einen bestimmten Zahlenbereich nur mit lokalen Variablen zu iterieren. Die folgende Schleife gibt alle Zahlen von 10 bis einschließlich 12 nach einander aus:

```
>>> for i in range(10, 13):
...     print(i)
...
10
11
12
```

Max Funktion

Der Algorithmus muss an einer Stelle das Maximum von drei Werten ermitteln. In Python steht hierfür die `max()` Funktion bereit, welche für beliebig viele Zahlenparameter den maximalen Wert zurück gibt:

```
>>> max(0, 12, 5, 7)
12
```

Implementation

Für die Implementation des Algorithmus wird ein Python-Skript bereitgestellt (`needleman_wunsch.py`). Dieses verwendet vor-implementierte Hilfsfunktionen aus einer anderen Datei (`util.py`). Weiterhin stehen mehrere Testsequenzen (`sequences.fasta`) und erwartete Ergebnisse (`results.txt`) bereit. Kopieren Sie sich den Ordner mit diesen Dateien in Ihr Home-Verzeichnis mittels der Kommandozeile und wechseln Sie hinein:

```
cp -r /home/olzhabaev/Public/ase_uebung_05 .
cd ase_uebung_05
```

Das Skript beginnt mit dem Import von Hilfsfunktionen und dem Lesen der Testsequenzen. Für ein Test-Alignment werden zwei Sequenzen ausgewählt und deren Längen ermittelt. Weiterhin werden initial mit 0 befüllte Score- und Traceback-Matrizen erstellt und Variablen für die Match-, Mismatch- und Gap-Scores angelegt.

Ab Zeile 55 beginnen die Schleifen, welche die Score- und Traceback-Matrixwerte berechnen, zunächst für die erste Spalte und Zeile, und anschließend für die restlichen Positionen. Der Rahmen der Schleifen ist vorgegeben, die konkreten Berechnungen und Zuweisungen fehlen jedoch.

Ab Zeile 120 beginnt die Alignment-Rekonstruktion mit Hilfe von zwei initial leeren Zeichenketten und einer `while`-Schleife, welche einem Pfad durch die Traceback-Matrix folgt und dabei die Alignment-Zeichenketten erweitert.

Abschließend wird die Score-Matrix mittels einer speziellen Funktion und das Alignment mittels einfachen `print` Anweisungen in die Konsole ausgegeben.

Vervollständigen Sie Schleifen des Algorithmus zur Berechnung der Score- und Traceback-Matrizen. Folgen Sie dabei den (in den Kommentaren markierten und detaillierter beschriebenen) Schritten 1-6:

1. In der ersten Scheife (Z. 55) die erste Spalte der Score- und Traceback-Matrizen korrekt befüllen.
2. In der zweiten Scheife (Z. 66) die erste Zeile der Score- und Traceback-Matrizen korrekt befüllen.
3. In der geschachtelten Schleife (Z. 81) zunächst die Inertion- und Deletion Scores berechnen und speichern.
4. Danach den Substitution-Score in Abhängigkeit vom beobachteten Match / Mismatch berechnen und speichern.
5. Den größten der drei Scores ermitteln und in die Score-Matrix schreiben.
6. Ermitteln, durch welche Operation der größte Score zustande kommt und diese in die Traceback-Matrix eintragen.

Testen

Testen Sie Ihre Implementation, indem Sie das fertiggestellte Skript ausführen und die Ergebnisse evaluieren. Die Datei `results.txt` enthält für alle Paare unterschiedlicher Sequenzen die optimalen Scores und Alignments, die zu diesen führen. Prüfen Sie ebenfalls Alignment von Sequenzen mit sich sebst (wobei offenslichtlich ein Score von 10 und lückenlose Alignments entstehen sollen). Die Wahl der verwendeten Testsequenzen findet im Skript in den Zeilen 23 und 24 statt.

Fragen

1. Wie wächst die Laufzeit und der Speicherbedarf der vorgestellten Version des Needleman-Wunsch Algorithmus in Abhängigkeit von den Längen der Sequenzen?
2. In der Vorlesung wurde eine kompliziertere Version des Algorithmus vorgestellt, welche zwischen den Kosten für das Erzeugen und dem Erweitern von Lücken unterscheidet. Wie verhält sich die Laufzeit und der Speicherbedarf dort?
3. Überlegen Sie, wie das Verfahren erweitert werden könnte, um ein optimales Alignment zwischen beliebig vielen Sequenzen zu berechnen. Wie würde die Laufzeit und der Speicherbedarf dabei aussehen?