Übungsblatt 11

Abgabe am 21.01.2019

Aufgabe 1: Dynamisches Programmieren I - Sequenzalignment (5+2 Punkte)

Um die Funktion eines neu entdeckten Gens zu verstehen, wird eine Datenbank bekannter Gene nach "ähnlichen" Genen durchsucht. Die Ähnlichkeit zweier Gene wird daran gemessen, wie gut man die Gene abgleichen kann. Dazu sehen wir ein Gen als einen langen String, der aus den Buchstaben aus $\Sigma = \{A, C, G, T\}$ besteht. Betrachten Sie zwei Gene (Strings) $x = ATGCC$ und $y = TACGCA$. Ein Abgleich von x und y geschieht hier durch spaltenweises vergleichen, beispielsweise

$$\begin{array}{cccccccc} - & A & T & - & G & C & C \\ T & A & - & C & G & C & A \end{array}$$

Hier symbolisiert "-" eine Lücke, das heißt die Strings können mit "-" unterbrochen werden und müssen nicht eins zu eins gegenüber gestellt werden. Die Buchstaben jedes Strings müssen in der richtigen Reihenfolge stehen und jede Spalte muss einen Buchstaben von mindestens einem der beiden Strings enthalten. Der Score eines Abgleichs ist gegeben durch eine Score-Matrix δ der Größe $(|\Sigma| + 1) \times (|\Sigma| + 1)$, wobei die extra Spalte und die extra Zeile aufgrund der Lücke eingeführt wurden. Wenn zum Beispiel der Buchstabe A mit C verglichen wird, finden wir den Score $\delta(A, C)$ als Eintrag in der Matrix δ durch die zu A zugehörige Zeile und die zu C zugehörige Spalte. Insgesamt hat der obige Abgleich zum Beispiel den Score:

$$\delta(-, T) + \delta(A, A) + \delta(T, -) + \delta(-, C) + \delta(G, G) + \delta(C, C) + \delta(C, A).$$

Die Score-Matrix ist beliebig und könnte "replace"-Operationen anders gewichten als "insert"-Operationen, oder gar eine höhere Belohnung für gewisse Buchstaben geben.

a) Implementieren Sie einen Ansatz des Dynamischen Programmierens, der als Input zwei Strings $x[1, ..., n]$ und $y[1, ..., m]$ und eine Scoring Matrix δ annimmt, und den höchsten Score für einen Abgleich ausgibt. Die Laufzeit soll $O(nm)$ sein. Im Moodle-Portal finden Sie dazu die Datei `seqAlign.java`. Laden Sie Ihren Code im Moodle hoch. Wieder sind weitere Imports nicht gestattet (und auch nicht notwendig).

b) In der Java-Datei finden Sie einen String, der für eine unbekannte DNA steht. Außerdem stehen Ihnen 4 Vergleichs-DNAs zur Verfügung. Können Sie die Vergleichs-DNA finden, die die geringste Mutation zur unbekannten DNA aufweist? Welche ist es?

Aufgabe 2: Dynamisches Programmieren II (4 Punkte)

Ein Alphabet Σ ist eine endliche Menge von Buchstaben. Sei Σ^+ die Menge aller möglicher Strings positiver Länge, die aus Buchstaben aus Σ bestehen. Gegeben sei ein endlicher String $\mathbf{s} = s_1 \dots s_n \in \Sigma^+$ der Länge n mit $s_i \in \Sigma$ für alle i. Es wird vermutet, dass $\mathbf{s}$ einen Text darstellt, aus dem alle Leerzeichen und Interpunktionen entfernt wurden. Zum Beispiel könnte die Eingabe so aussehen: `diesistsinnvoll`. Außerdem liegt uns ein Wörterbuch W vor, dass alle gültigen Wörter enthält, das heißt, W ist eine Teilmenge von Σ^+.
Nun haben Sie Zugang zu einer Wörterbuch-Funktion $dict : \Sigma^+ \to \{0, 1\}$, welche für eine beliebige Eingabe w bestimmt, ob w ein gültiges *einzelnes* Wort aus W ist. Falls $dict(w) = 1$, so ist das Wort w ein gültiges Wort aus dem Wörterbuch W, andernfalls nicht.
Nutzen Sie dynamische Programmierung, um effizient zu ermitteln, ob der Eingabestring $\mathbf{s}$ in eine Folge gültiger Wörter zerlegt werden kann. Die Laufzeit darf $\mathcal{O}(n^2)$ betragen, wobei jeder Aufruf von *dict* Zeit $\mathcal{O}(1)$ benötigt.

Aufgabe 3: Dynamische Programmierung III (4 + 3 + 2 Punkte)

Gegeben sei eine endliche Folge positiver Zahlen $a = (a_1, \ldots, a_n)$. Das Gewicht einer Teilfolge sei die Summer ihrer Einträge. Wir suchen das maximale Gewicht einer Teilfolge a' mit nicht-benachbarten Einträgen, das heißt a' kann a_i oder a_{i+1} enthalten, aber nicht beide Elemente gleichzeitig.

a) Geben Sie den Pseudo-Code eines Algorithmus an, der dieses maximale Gewicht in Laufzeit $\mathcal{O}(n)$ berechnet, und begründen Sie seine Korrektheit.

b) Adaptieren Sie Ihren Algorithmus aus (a), sodass er zusätzlich eine entsprechende Teilfolge mit maximalem Gewicht ausgibt. Begründen Sie die Korrektheit Ihrer Adaption.

c) Funktionieren Ihre Algorithmen aus (a) bzw. (b) auch dann noch, wenn die Einträge in a nun nicht mehr notwendigerweise positiv sind? Begründen Sie.