Lehrstuhl Algorithmen
und Komplexität
Prof. Dr. Fabian Kuhn
Sebastian Daum, Oleksii Saukh

Algorithmen und Datenstrukturen
(Informatik II) SS 2014

http://ac.informatik.uni-freiburg.de/teaching

UNI FREIBURG

Übungsblatt 10

Abgabe: Dienstag, 22. Juli, 16:00 Uhr

Laden Sie Ihre Lösungen für theoretische *Übungen im Format "loesung-xx-y.pdf" hoch, wobei "xx" die Nummer des aktuellen Übungsblattes ist (bei Bedarf mit führender Null) und "y" die Aufgabennummer, d.h., erzeugen Sie* pro (theoretische) Aufgabe *eine* .pdf*-Datei. Scans, Bilder von Scans etc. bitte zuerst ins pdf-Format umwandeln. Bei Programmieraufgaben müssen Sie – wann immer es relevant/durchführbar ist – Unit Tests sowie einen Style-check durchführen. Laden Sie außerdem eine Datei "erfahrungen.txt" hoch, in welcher Sie Ihre Erfahrungen und Meinungen zu dem Übungsblatt teilen.*

Aufgabe 1 (5 Punkte)

Betrachten Sie die folgende Modifikation des Dijkstra-Algorithmus. Statt einem Startknoten s wird eine Menge S bereitgestellt. Statt der Notation $\delta(s, v)$ aus der Vorlesung verwenden wir $\delta(S, v)$. Für alle Knoten $s \in S$ setzen wir $\delta(S, s) := 0$ und fügen s in den Heap H mit value 0 ein (in beliebiger Reihenfolge). Außerdem wird die Abfrage "if $u \neq s$ then ..." durch "if $u \notin s$ then ..." ersetzt.

Wenn Sie diesen Algorithmus laufen lassen, was genau erhalten Sie als Ergebnis? Begründen Sie kurz Ihre Lösung.

Aufgabe 2 (15 Punkte)

In der von Prof. Bast gehaltenen Vorlesung geht es um die Editierdistanz von zwei Strings. In dieser Aufgabe sollen Sie einen Algorithmus implementieren, der zu einem gegebenen langen String T und einem *pattern* p einen *zusammenhängenden* Substring t in T findet, welcher die minimalste Editierdistanz zu p aufweist. Zum Beispiel, für T ="akfspehdnskfppsankeksle" und p ="kepak" hat der Substring "ak" eine Editierdistanz von 3, "kfp", "kfpp" und "kfpps" auch, "kfppsank" lässt sich in 4 Schritten zu "kepak" umwandeln, die kleinste Editierdistanz von 2 wird allerdings von "kek" erreicht, indem man "pa" einfügt.

Die Lösung erfolgt über dynamische Programmierung. Bezeichne T_j den Substring von T, welcher bei $T[1]$ startet und bei $T[j]$ endet, und analog sei p_i der Substring von p, welcher bei $p[1]$ startet und bei $p[i]$ endet. Bauen Sie ein zweidimensionales Array A, so dass Feld $A[i, j]$ die Lösung zu einem *Unterproblem* enthält. Dieses Unterproblem ist wie folgt definiert. Gefunden werden soll die kleinste Editierdistanz $\eta_{i,j}$ von p_i zu einem Substring t_j von T_j, welcher auf $T[j]$ *endet*.

Wieder am Beispiel: p_3 wäre "kep", T_{12} wäre "akfspehdnskf". Gesucht ist also ein Substring in T_{12} (und seine Editierdistanz), welcher auf "...skf" endet und minimale Editierdistanz zu p_3 aufweist. Dieser minimale Substring wäre im Beispiel "kf" – es werden zwei Schritte benötigt, um auf "kep" zu kommen.

Der Trick ist es, $A[i, j]$ effizient zu berechnen. Das geht so.

$$A[i, j] = \min\{A[i-1, j] + 1, A[i, j-1] + 1, A[i-1, j-1] + c(i, j)\},$$

wobei $c(i, j) = 0$, falls $T[j] = p[i]$ und andernfalls gilt $c(i, j) = 1$. Jede dieser 3 Reduktionen auf einen bekannten Wert entspricht einer der Operationen *Insert*, *Delete*, *Replace*, wobei letztere Kosten von 0 haben, falls ein Zeichen unverändert bleibt.

In der letzten Zeile des zweidimensionalen Arrays haben wir all die Lösungen für die Unterprobleme, in welchen ein auf $T[i]$ endender Teilstring zum pattern p umgewandelt wird. Um die eigentliche Aufgabe zu lösen, müssen

wir nur das Minimum in dieser Zeile suchen, denn der optimale Substring t endet garantiert auf irgendein i (außer $p = \epsilon$, der leere String). Was noch fehlt ist eine Initialisierung der Tabelle A in der ersten Spalte und der ersten Zeile, um alle $A[i, j]$ berechnen zu können. Das geht am Einfachsten, indem man das Array mit einer 0-ten Zeile und Spalte initialisiert, mit $A[0, j] = 0$ für jedes j und $A[i, 0] = i$ für jedes i.

Implementieren Sie diese dynamische Programmieraufgabe. Im SVN-Verzeichnis *public* finden Sie eine Datei *text.txt* und eine Datei *patterns.txt*. Ihr Algorithmus soll den Inhalt der ersten Datei als T interpretieren. Die zweite Datei enthält eine Menge von *patterns* p. Ihr Algorithmus soll eine Datei *result.txt* erstellen und für jedes *pattern*, die Editierdistanz η, den Substring t für welchen η erreicht wird, sowie die Position von $t[0]$ im Text T in die Datei $result.txt$ schreiben.