Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
Y. Maus, O. Saukh

15 Juli, 2016

Algorithmen und Datenstrukturen
Sommersemester 2016
Übungsblatt 11

Abgabe bis 12:00, Freitag, 22 Juli, 2016

Achtung: Bitte fügen Sie Ihrer Lösung eine (kurze) Datei *erfahrungen.txt* mit Ihren Erfahrungen mit dem jeweiligen Aufgabenblatt hinzu. Das Forum können (und sollen) Sie nicht nur für Fragen zur Übung, sondern auch für Fragen zur Vorlesung benutzen.
Achtung: Dieses Blatt ist ein Bonusblatt, d.h., die in diesem Blatt erreichten Punkte zählen zum Erreichen der 50% der Übungspunkte, jedoch erhöhen die Punkte dieses Blattes nicht die zu erreichende Punktzahl.

1 Aufgabe 1 (10 Punkte)

Der Sattel eines Packesels hat zwei Taschen, eine links und eine rechts des Esels. Der Esel soll nun möglichst gleichmäßig mit $N \geq 1$ Gegenständen beladen werden, das heißt, der Gewichtsunterschied zwischen den beiden Taschen soll minimal werden. Wir nehmen an, dass der i-te Gegenstand ($1 \leq i \leq N$) genau $g_i \in \mathbb{N}_+$ Kilogramm wiegt. Zur Lösung des Problems soll ein Algorithmus entwickelt werden, der auf Dynamischem Programmieren beruht. Es sei $G = \sum_{i=1^N} g_i$ das Gesamtgewicht aller Gegenstände. Weiter sei x_{ij} eine binäre Variable mit $x_{ij} = 1$, falls sich die ersten i Gegenstände so auf die Taschen verteilen lassen, dass die Gewichtsdifferenz zwischen linker und rechter Tasche $= j$ ist, und $x_{ij} = 0$ sonst ($1 \leq i \leq N; -G \leq j \leq G$). Beispiel: Ist $x_{ij} = 1$ und wird der $(i+1)$-te Gegenstand in die linke Tasche gelegt, so folgt $x_{i+1,j+g_{i+1}} = 1$.

a) (1 Punkt) Geben Sie ein Beispiel für N und $g_1, \ldots, g_N$ an, bei dem sich die Gegenstände nicht so auf die Taschen verteilen lassen, dass beide Taschen gleich schwer sind.

b) (2 Punkte) Geben Sie die Werte für x_{1j} an.

c) (3 Punkte) Geben Sie eine Rekursionsformel für x_{ij} an.

d) (4 Punkte) Geben Sie in Pseudocode einen Algorithmus an, der die minimale absolute Gewichtsdifferenz ausgibt. (Achtung: Es ist nicht nach einer optimalen Verteilung der Gegenstände selbst gefragt!)

e) (1 Punkt) Was ist die Laufzeit von dem Algorithmus?

Aufgabe 2: (10 Punkte)

In der Vorlesung haben wir die Probleme *Editierdistanz* und *Approximate String Matching* betrachtet. 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 (d.h. sie sollen einen Approximate String Matching Algorithmus implementieren). 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.
Im Folgenden wiederholen wir weite Teile der Vorlesung: 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.