Prof. Dr. Benjamin Doerr, Dr. Reto Spöhel
Übungsleitung: Alexander Kobel

Wintersemester 2011/12

Übung zu Grundzüge von Algorithmen und Datenstrukturen

http://www.mpi-inf.mpg.de/departments/d1/teaching/ws11/grads/

Übungsblatt 13 Abgabe: keine

Der auf diesem Blatt vermittelte Stoff ist genau so klausurrelevant wie der Stoff auf den bisherigen 12 Übungsblättern, und Sie sollten die Aufgaben so sorgfältig wie immer bearbeiten. Sie brauchen Ihre Lösungen aber nicht abzugeben; das Blatt ist nicht relevant für die Klausurzulassung. Die Punkteangaben dienen nur als ungefähre Angabe, wie schwierig/aufwendig die einzelnen Aufgaben sind. Das Blatt ist mit Absicht weniger umfangreich als die bisherigen Aufgabenblätter.

Die Lösung der Aufgaben wird in der Vorlesung vom 2. Februar besprochen.

Aufgabe 1 (*Schriftliche Übung, 5 Punkte*)

Wir betrachten die untenstehende Pseudocode-Implementierung des in der Vorlesung diskutierten Dijkstra-Algorithmus. Wenn wir im Folgenden von Distanzen, kürzesten Pfaden etc. sprechen, beziehen wir uns immer auf den gegebenen Inputgraphen $G = (V, E)$ mit den gegebenen Kantengewichten $w : E \to \mathbb{N}$. Die *Länge eines Pfades* ist gegeben durch die Summe der auf dem Pfad auftretenden Kantengewichte (die wir hier also als Längenangaben interpretieren).

Für $u, v \in V$ bezeichnen wir mit $\delta(u, v)$ die Länge eines kürzesten Pfades von u nach v. Wir definieren $\delta(u, v) := \infty$, falls es keinen Pfad von u nach v gibt. Für $u, v \in V$ und $W \subseteq V$ bezeichnen wir mit $\delta_W(u, v)$ die Länge eines kürzesten Pfades von u nach v, der außer u und v nur Knoten aus W verwendet. Wir definieren $\delta_W(u, v) := \infty$, falls es keinen solchen Pfad gibt.

Algorithm 1 DIJKSTRA

Input: Graph $G = (V, E)$, weight function $w : E \to \mathbb{N}$, node $s \in V$

Output: For all $v \in V$ we have $d[v] = \delta(s, v)$. The edges $\{p[v], v\}$, $v \in V$, form a shortest paths tree from s.

1: **for all** $v \in V$ **do**
2:     $d[v] := \infty$
3:     $p[v] := null$
4: $W := \emptyset$
5: $d[s] := 0$
6: **while** $d_{\min} := \min \big\{ d[v] \,\big|\, v \in V \setminus W \big\} < \infty$ **do**
7:     Find a node $u \in V \setminus W$ with $d[u] = d_{\min}$
8:     $W := W \cup \{u\}$ /* $d[u] = \delta(s, u)$ */
9:     **for all** $v \in V \setminus W$ adjacent to u **do**
10:         **if** $d[v] > d[u] + w(\{u, v\})$ **then**
11:             $d[v] := d[u] + w(\{u, v\})$ /* $d[v] = \delta_W(s, v)$ */
12:             $p[v] := u$

---

a) Zeigen Sie, dass nach jeder Iteration der **while**-Schleife die folgenden Invarianten gelten:

   (i) Für alle $u \in W$ gilt $d[u] = \delta(s, u)$.

   (ii) Für alle $v \in V \setminus W$ gilt $d[v] = \delta_W(s, v)$.

   Wie in der Vorlesung diskutiert sind die Ungleichungen $d[u] \geq \delta(s, u)$ und $d[v] \geq \delta_W(s, v)$ mehr oder weniger offensichtlich (warum?); konzentrieren Sie sich also auf den Beweis, dass $d[u] \leq \delta(s, u)$ und $d[v] \leq \delta_W(s, v)$.

   Hinweis: Beweisen Sie beide Aussagen gemeinsam per Induktion. Nehmen Sie im Induktionsschritt widerspruchsweise an, dass für den neu zu W hinzugefügten Knoten u die Ungleichung $d[u] > \delta(s, u)$ gilt, es also einen s-u-Pfad kürzer als $d[u]$ gibt. Betrachten Sie den ersten Knoten dieses Pfades, der nicht in W liegt (warum gibt es diesen?), und finden Sie einen Widerspruch zur Wahl von u in Zeile 7.

b) Folgern Sie aus a) die Korrektheit von DIJKSTRA.

Aufgabe 2 (*mündliche Übung, 0 Punkte*)

Lassen Sie sich nochmals in Ruhe durch den Kopf gehen, wie der in der Vorlesung diskutierte und unten in Pseudocode gegebene Floyd-Warshall-Algorithmus funktioniert. Wir verwenden wieder die Notationen aus Aufgabe 1.

Algorithm 2 Floyd-Warshall

Input: Graph $G = (V, E)$ on vertex set $V = \{1, \dots, n\}$, weight function $w : E \to \mathbb{N}$.

Output: For all $i, j \in V$ we have $F_n[i, j] = \delta(i, j)$, and a shortest path from i to j can be obtained recursively by appending the edge $\{P_n[i, j], j\}$ to a shortest path from i to $P_n[i, j]$.

for $i = 1, \dots, n$ **do**
  for $j = 1, \dots, n$ **do**
    $F_0[i,j] := \begin{cases} w(\{i,j\}) & \text{if } \{i,j\} \in E \\ 0 & \text{if } i = j \\ \infty & \text{else} \end{cases}$ /* $F_0[i,j] = \delta_\emptyset(i,j)$ */
    $P_0[i,j] := \begin{cases} i & \text{if } \{i,j\} \in E \\ \mathit{null} & \text{else} \end{cases}$
for $k = 1, \dots, n$ **do**
  for $i = 1, \dots, n$ **do**
    for $j = 1, \dots, n$ **do**
      $F_k[i,j] := \min\left\{F_{k-1}[i,j], F_{k-1}[i,k] + F_{k-1}[k,j]\right\}$ /* $F_k[i,j] = \delta_{\{1,\dots,k\}}(i,j)$ */
      $P_k[i,j] := \begin{cases} P_{k-1}[i,j] & \text{if } F_k[i,j] = F_{k-1}[i,j] \\ P_{k-1}[k,j] & \text{else} \end{cases}$

Aufgabe 3 (*Schriftliche Übung, 5 Punkte*)

Sei A eine nichtleere Menge. Es bezeichne $x = x_1 x_2 \dots x_n$ mit $x_i \in A$ eine *Folge* der Länge n über A. Jede Folge, die aus x durch Auslassen von gewissen (eventuell keinem) x_i gewonnen werden kann, heisst *Teilfolge* von x. Beispielsweise sind *abc* und *bbde* Teilfolgen von *aabbbbcde*, *adf* und *bac* jedoch nicht.

Beim Problem der Längsten Gemeinsamen Teilfolge sind zwei Folgen $x = x_1 \dots x_n$ und $y = y_1 \dots y_m$ gegeben. Gesucht ist eine *längste* Folge z, die sowohl von x als auch von y Teilfolge ist.

a) Geben Sie einen Algorithmus an, der die Länge einer längsten gemeinsamen Teilfolge von x und y in Zeit $O(nm)$ bestimmt.

   Hinweis. Jede Folge $x^{(i)} = x_1 x_2 \dots x_i$ mit $i = 0, \dots, n$ heisst *Präfix* der Folge $x = x_1 x_2 \dots x_n$. Dabei steht der Fall $i = 0$ für das leere Präfix. Verwenden Sie das Prinzip der dynamischen Programmierung, indem Sie rekursiv für alle $1 \leq i \leq n$ und $1 \leq j \leq m$ die Länge einer längsten gemeinsamen Teilfolge von $x^{(i)}$ und $y^{(j)}$ berechnen. [3P.]

b) Überlegen Sie sich, wie man den Algorithmus modifizieren muss, damit er nicht nur die Länge einer längsten gemeinsamen Teilfolge, sondern tatsächlich auch eine gemeinsame Teilfolge z dieser Länge ausgibt. [2P.]