6. Aufgabenblatt zur Vorlesung

Entwurf und Analyse von Algorithmen

(Günter Rote, Astrid Kaffanke, Christian Knauer)

Abgabe: 30. November 2000

Die erste Klausur findet am 7. Dezember statt!

1. **Kürzeste Wege mit gewichteten Knoten** (3 Punkte).

   Gegeben ist ein gerichteter Graph $G = (V, E)$ mit gewichteten Kanten *und* mit gewichteten Knoten (damit können z.B. Umsteigezeiten modelliert werden), d.h. es sind Zahlen a_{ij} für $ij \in E$ und g_i für $i \in V$ gegeben, das Gewicht eines Weges $w = i_0, i_1, \ldots, i_k$ ist gegeben durch

   $$\sum_{l=1}^{k} a_{i_{l-1},i_l} + \sum_{l=0}^{k} g_l.$$

   Wie kann man hier (wenn es keinen negativen Kreis gibt) mit Hilfe eines schon bekannten Algorithmus kürzeste Wege von einem Startknoten zu allen anderen Knoten finden?

2. **Dijkstra und negative Kantengewichte** (4 Punkte).

   Gib ein Beispiel eines Graphen mit negativen Kantengewichten (aber natürlich ohne negativen Kreis) an, bei dem der Algorithmus von Dijkstra schiefgeht (d.h. es wird nicht für alle Knoten die Länge eines kürzesten Weges ausgegeben).

   Analysiere, woran das liegt (an welcher Stelle im Korrektheitsbeweis braucht man, dass die Kantengewichte positiv sind?).

3. **Dijkstra für ganzzahlige Kantengewichte – Teil 2** (4 Punkte).

   Verbessere Deinen Algorithmus aus Aufgabe 4 vom fünften Aufgabenblatt so, dass er eine Laufzeit von $O((V + E) \log W)$ hat.

4. **Implementierung von Graphalgorithmen**[1] (10 Punkte). **Praktische Aufgabe!** Abgabe bis zum 7.12.2000

   Implementiere Dijkstras Algorithmus. Dein Programm sollte es dem Benutzer ermöglichen, interaktiv nach einem kürzesten Pfad (sowie dessen Länge) zu einzelnen Knoten zu fragen. Teste deine Implementierung mit zufällig erzeugten gerichteten, gewichteten Graphen auf 100 Knoten (Kantenwahrscheinlichkeit $p = 1/2$, *ganzzahlige* Kantengewichte gleichverteilt aus dem Intervall $[1, 100]$).

5. **Analyse dynamischer Listen** (5 Punkte).

   In dieser Aufgabe betrachten wir eine Implementierung von Listen mit automatischer Speicherverwaltung, die es gestattet mit Listen beliebiger Länge zu arbeiten. Die Datenstruktur verfügt über die folgenden Operationen:

[1] Stell Dir vor, dass hier der übliche Hinweis zu Programmieraufgaben steht.

(a) *Append*$(\mathcal{L}, x)$: Fügt das Element x am Ende der Liste $\mathcal{L}$ ein; dadurch verlängert sich die Länge der Liste um Eins.

(b) *Remove*$(\mathcal{L})$ $(\mathcal{L} \neq \emptyset)$: Entfernt das Element am Ende der Liste; dadurch verkürzt sich die Länge der Liste um Eins.

(c) *Access*$(\mathcal{L}, i)$ $(1 \leq i \leq |\mathcal{L}|)$: Liefert das i-te Element der Liste zurück.

Intern werden die Elemente von $\mathcal{L}$ in einem Array der Länge $m_{\mathcal{L}} \geq |\mathcal{L}|$ gespeichert. Verlängert man eine Liste $\mathcal{L}$ mit $m_{\mathcal{L}} = |\mathcal{L}|$ um ein Element durch ein *Append*$(\mathcal{L}, x)$, so wird ein neues Array der Länge $2m_{\mathcal{L}}$ angelegt und $\mathcal{L}$ in dieses Array kopiert.

Die Kosten für eine *Append* Operation sind in diesem Fall $O(|\mathcal{L}|)$, und $O(1)$ sonst.

Die Kosten für die Operationen *Access* und *Remove* sind immer $O(1)$.

Was sind die maximalen Kosten einer Folge von N *Access*, *Append* und *Remove* Operationen, wenn man mit einer leeren Liste beginnt?