Datenstrukturen (DS)

Sommersemester 2015

Dipl-Inf. Bert Besser
Prof. Dr. Georg Schnitger
Hannes Seiwert, M.Sc.

GOETHE
UNIVERSITÄT
FRANKFURT AM MAIN

Institut für Informatik
AG Theoretische Informatik

Übung 4

Ausgabe: 02.06.2015
Abgabe: 16.06.2015

Aufgabe 4.1. *Flugverkehr* (4+4 Punkte)

(Vorlesung am 02.06.2015, Abschnitt 4.4.4 im Skript)

Die Fluggesellschaft Alexand-Air speichert das Angebot ihrer Flugverbindungen als gerichteten Graphen $G = (V, E)$, wobei jeder Knoten einen Flughafen und jede gerichtete Kante (a, b) eine Direktverbindung von a nach b modelliert. Wir interessieren uns hier im Folgenden nicht für Flug*dauer* oder Abflugzeiten, sondern nur für die Anzahl der Zwischenstopps einer Verbindung.

a) Horst plant seinen Urlaub. Er möchte von Georgetown nach Johannesburg fliegen und dabei unbedingt einen Zwischenstopp in St. Petersburg einlegen, um dort seine Cousine zu besuchen. Er möchte die Verbindung so wählen, dass möglichst wenige Zwischenstopps erforderlich sind.

   Entwerfen Sie einen Algorithmus, der für einen gegebenen Startflughafen a und Zielflughafen b eine Flugverbindung mit möglichst wenigen Zwischenstopps ermittelt, unter der Einschränkung, dass ein gewünschter Zwischenstopp c angeflogen werden *muss*. Die Laufzeit soll durch $\mathcal{O}(|V|+|E|)$ beschränkt sein.

b) Während ihrer Rundreise durch Japan möchte Hannah von der Insel Kyūshū zu den Ryūkyū-Inseln fliegen. In beiden Regionen gibt es mehrere Flughäfen, zwischen denen jeweils unterschiedliche Verbindungen bestehen.

   Sei $S = \{s_1 \ldots, s_k\} \subseteq V$ die Menge möglicher Startflughäfen und $Z = \{z_1, \ldots, z_j\} \subseteq V$ die Menge möglicher Zielflughäfen. Gesucht ist ein Flug von *irgendeinem* Startpunkt $s \in S$ zu *irgendeinem* Zielpunkt $z \in Z$ mit möglichst wenigen Zwischenstopps. Entwerfen Sie einen Algorithmus, der eine solche Verbindung berechnet und höchstens Laufzeit $\mathcal{O}(|V|+|E|)$ benötigt.

   Hinweis: Es bietet sich an, den Eingabegraphen G zu modifizieren.

Hinweis: Alle Algorithmen aus der Vorlesung oder bisherigen Übungsaufgaben dürfen ohne weitere Begründung verwendet werden. In Aufgabe 4.1 darf bei *strukturierter* Darstellung eine Beschreibung durch Pseudocode entfallen.

Aufgabe 4.2. *Hands on Heaps* (3+3+3 Punkte)

(Vorlesung am 09.06.2015, Abschnitt 4.5 im Skript)

Gegeben sei ein Max-Heap $H = [-|37\,|\,27\,|\,9\,|\,17\,|\,7\,|\,8\,|\,5\,|\,7\,|\,5\,|\,3\,|\,1]$ als Array, wobei $H[0] =$ "$-$" dafür steht, dass die "nullte" Zelle ungenutzt bleibt. Die folgenden Teilaufgaben bauen nicht aufeinander auf: Die geforderte Folge von Operationen ist jeweils auf dem ursprünglichen Heap H auszuführen. Stellen Sie den Heap nach jeder Operation als Baum und als Array dar.

a) Führen Sie nacheinander die Operationen `insert(19)` und `delete_max()` auf H aus.

b) Führen Sie nacheinander die Operationen `delete_max()` und `insert(19)` auf H aus.

c) Führen Sie nacheinander die Operationen `change_priority(4,42)` und `change_priority(2,6)` auf H aus.
   Erinnerung: `change_priority(i,x)` bedeutet, dass die Priorität $H[i]$ zu x geändert wird.

Aufgabe 4.3. *Mächtigere Max-Heaps* (4+4 Punkte)

(Vorlesung am 09.06.2015, Abschnitt 4.5 im Skript)

Wir wollen Heaps zu einer mächtigeren Datenstruktur erweitern und zwei neue Funktionen `lookup` und `max` implementieren.

a) Wir verwalten Objekte aus der Menge $\{0, \ldots, n\}$ in einem Heap H, wobei ein Objekt nicht mehrfach in H enthalten sein darf. Die Priorität von Objekt i ist p_i. Dafür speichern wir in einer Zelle von H ein Objekt i mitsamt seiner Priorität als Paar (i, p_i). Die Heap-Ordnung wird durch die Prioritäten bestimmt: Das Paar (i, p_i) ist größer als das Paar (j, p_j) genau dann, wenn $p_i > p_j$ gilt.

   Wir möchten eine beliebige Folge von Operationen (`insert`, `delete_max`, `change_priority`, `remove`, etc.) auf H ausführen. Erinnern Sie sich, dass diese Operationen auf `repair_up` und `repair_down` basieren. Nehmen Sie an, dass H anfänglich leer ist.

   Implementieren Sie eine Funktion `lookup(i)`, welche die aktuelle Position des Paares (i, p_i) im Array H in konstanter Zeit ausgibt: Beachten Sie, dass sich diese Position während der Ausführung der Operationen ändern kann. Ist (i, p_i) nicht in H enthalten, soll -1 zurückgegeben werden.

   Hinweis: Modifizieren Sie die Funktionen `repair_up` und `repair_down` in geeigneter Weise ohne deren asymptotische Laufzeit zu verändern; aber auch `insert` und `delete_max` müssen angepasst werden. Sie können annehmen, dass (zusätzlich zu H) ein mit -1'en initialisiertes Array $A[0 \ldots n]$ zur Verfügung steht.

b) Gegeben sei ein Max-Heap H mit n Elementen. Die Funktion `max(p)` soll alle *p-großen* Prioritäten im Heap, d.h. solche Prioritäten x mit $x \geq p$, ausgeben. Entwerfen Sie einen Algorithmus für `max(p)`, dessen Laufzeit linear in der Anzahl p-großer Prioritäten, also $\mathcal{O}(|\{x \in H : x \geq p\}|)$ ist.

   Hinweis: Es ist nicht zielführend, H von vorne nach hinten zu durchlaufen. Das würde zu lange dauern!

Aufgabe 4.4. *k-faches Verschmelzen* (7 Punkte)

(Vorlesung am 09.06.2015, Abschnitt 4.5 im Skript)

Gegeben seien k Listen $L_1, \ldots, L_k$ der Länge n, die jeweils aufsteigend sortiert sind. Wir wollen diese Listen zu einer einzigen Liste L verschmelzen, sodass L alle nk Elemente in aufsteigender Reihenfolge enthält.
Entwerfen Sie einen Algorithmus, der dies in Laufzeit $\mathcal{O}(nk \cdot \log(k))$ bewerkstelligt. Verwenden Sie dazu einen Min-Heap H, in dem anfänglich von jeder Liste L_i jeweils das erste (d.h. kleinste) Element gespeichert ist.

Kommentar: In diesem Ansatz wird jedes Element einer Liste L_i genau einmal aus L_i gelesen und genau einmal in die Ergebnisliste geschrieben. Wenn die Listen zu groß für den Hauptspeicher sind, können auf diese Weise Zugriffe auf einen langsamen Externspeicher gespart werden.