Universität Karlsruhe (TH)
Institut für Theoretische Informatik

Prof. Dr. Peter Sanders, Dr. Rob van Stee
G. Veit Batz, Johannes Singler

2. Übungsblatt zur Algorithmentechnik WS 2007/08

http://algo2.iti.uni-karlsruhe.de/algotech.php
{sanders|vanstee|batz|singler}@ira.uka.de

Aufgabe 1 (*Perfektes Hashing, Schwierigkeit* $1 + 1 + 2$)

a) Wie wahrscheinlich ist es, dass es zu mindestens einer Hashkollision kommt, wenn 50 Elemente in eine Hashtabelle mit 1000 Einträgen eingefügt werden, und die Hashfunktion zufällig ist?

b) Wie groß ist die erwartete Anzahl an Kollisionen für n Schlüssel in einer Tabelle von m Einträgen und einer 1-universellen Hash-Familie H_m?

c) Zeigen Sie, dass in diesem Fall für eine Tabelle der Größe $n(n-1)$ die Wahrscheinlichkeit $> 1/2$ ist, dass das Einfügen kollisionsfrei bleibt.

Aufgabe 2 (*Zugreservierungen, Schwierigkeit* 2)

Für einen bestimmten Zug liegen n Reservierungswünsche als Paar (Startbahnhof, Zielbahnhof) vor. Entwerfen Sie einen Algorithmus, der in $O(n \log n)$ entscheidet, ob alle Wünsche erfüllt werden können, wobei der Zug m Sitzplätze hat. Für jede Reservierung soll auch die Sitznummer (von 1 bis m) ausgegeben werden. Dabei ist die Reihenfolge der Bahnhöfe bekannt, nach dieser kann in konstanter Zeit verglichen werden.
Geben Sie Pseudo-Code an.

Aufgabe 3 (*Multiples Auswählen, Schwierigkeit* $4 + 2$)

Betrachten Sie den Algorithmus QuickSelect aus dem Manuskript von Mehlhorn und Sanders (Abschnitt 5.5, Pseudo-Code in Abbildung 5.9). Nehmen sich ruhig etwas Zeit, dieses Verfahren — das im Mittel $\mathcal{O}(n)$ Zeit benötigt — gut zu verstehen.

a) Entwerfen Sie ein verallgemeinerte Version von QuickSelect, nämlich QuickMultiSelect. Als Eingabe soll dieser neue Algorithmus eine Folge von Elementen $s = \langle e_1, \dots e_n \rangle$ sowie eine Menge von natürlichen Zahlen $K = \{k_1, \dots, k_\ell\}$ akzeptieren, wobei $k_1, \dots, k_\ell \leq n$ gelten muss. Als Ergebnis soll nun eine Menge $\{(e'_1, k_1), \dots (e'_\ell, k_\ell)\}$ mit $e'_1, \dots, e'_\ell \in s$ geliefert werden, so dass e'_i das k_i-größte Element von s ist. **Hinweis:** Orientieren Sie sich zunächst eher an QuickSort als an QuickSelect und übertragen Sie dann die Konzepte von QuickSelect auf QuickSort.

b) Gehen Sie nun davon aus, dass die folgenden (zugegebenermaßen stark vereinfachenden) Annahmen gelten:

- Die Länge n von s und die Kardinalität ℓ von K sind jeweils Zweierpotenzen.
- Das „Pivotelement“ p liegt immer in der Mitte von s.
- In jedem Rekursionsschritt wird K „halbiert“.

Wenn Sie den Algorithmus ausreichend gut gestaltet haben, können Sie nun nachweisen, dass er unter den obigen Vorraussetzungen eine Laufzeit von $\mathcal{O}(n \log \ell)$ hat. Führen Sie diesen Nachweis.

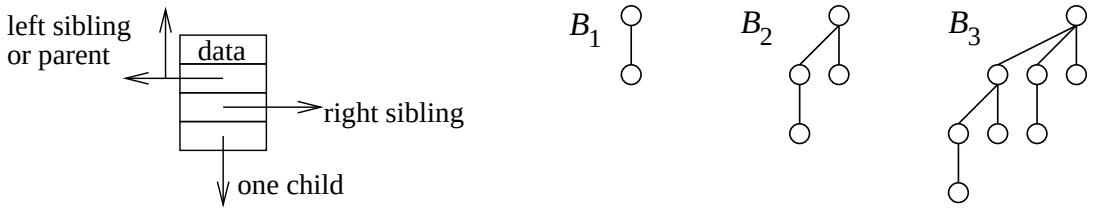

Abbildung 1: Ein Heap-Item für eine Drei-Zeiger-Implementierung eines Pairing Heaps (links) sowie die Bäume B_1, B_2 und B_3.

Aufgabe 4 (*Priority Queues: Pairing Heap, Schwierigkeit* $1 + 1 + 1 + 1$)

Betrachten Sie eine Implementierung von *Pairing Heaps*, die — wie in der Vorlesung vorgestellt — mit Hilfe von drei Pointern pro Heap-Item realisiert wurde. Abbildung 1 zeigt die schematische Darstellung eines einzelnen Heap-Items für diese Implementierung. Man darf aber nicht vergessen, dass auch die Menge der Wurzelknoten („Root-Set“) irgendwie dargestellt werden muss. Wie in der Vorlesung angegeben, soll dies in Form einer doppelt verketteten Liste geschehen.

a) Wie stellt die Datenstruktur einen Baum der Form B_3 (siehe Abbildung 1) im Detail dar? Zeichnen Sie diesen Fall. Null-Zeiger sollen durch leere Kästchen symbolisiert werden. Das Root-Set soll uns in dieser Teilaufgabe aber noch nicht interessieren, ignorieren Sie es also.

b) Wie kann das Root-Set als doppelt verkettete Liste im Detail realisiert werden? Verwenden Sie nur die Heap-Items und keine zusätzlichen Datentypen. Wie stellt Ihre Realisierung einen Pairing Heap, der zwei Bäume der Form B_1 bzw. B_2 enthält, im Detail dar? Zeichnen Sie, diesmal *mit* Darstellung des Root-Set.

c) Geben Sie den Pseudo-Code für die Operationen $cut(h : Handle)$ und $decreaseKey(h : Handle, k : Key)$ an. Spendieren Sie dazu den Heap-Items aber eine Markierung, um deren Mitgliedschaft im Root-Set anzuzeigen.

d) Angenommen, Sie statten die Heap-Items *nicht* mit einer Markierung für die Wurzeleigenschaft aus. Wie wirkt sich das auf den Pseudo-Code von *decreaseKey* bzw. *cut* aus?

Aufgabe 5 (*Priority Queues: Fobonacci-Heap, Schwierigkeit* $2 + 1$)

Gegeben sei ein leerer Fibonacci-Heap M. Nun werde folgende Operationsfolge ausgeführt ($M.insert(n)$ liefere dabei einen Zeiger auf ein neu eingefügtes Element mit Schlüssel $n \in \mathbb{N}$ zurück):

$$M.insert(3); M.insert(8); M.insert(2);$$
$$x := M.insert(5); M.insert(10); M.deleteMin();$$
$$M.insert(6); M.insert(7); M.decreaseKey(x, 2); M.deleteMin()$$

a) Welche Zustände durchläuft der Fibonacci-Heap im Verlauf der Ausführung? Zeichnen Sie abstrakt (d. h. die einzelnen Zeiger müssen nicht gezeichnet werden).

b) Nehmen Sie nun an, dass der Fibonacci-Heap mit Hilfe von vier Pointern pro Heap-Item realisiert wird (vgl. Buch von Mehlhorn und Sanders, Abbildung 6.8). Zeichnen Sie den Zustand nach Ausführung der letzten Operation detailiert.