Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Prof. Dr. Ernst W. Mayr/Dr. Sven Kosub
Stefan Eckhardt

Wintersemester 2004/05
Übungsblatt 7
6. Dezember 2004

Effiziente Algorithmen und Datenstrukturen I

Abgabetermin: 20.12.2004 **vor** *der Vorlesung*

Aufgabe 1

Sei D ein geordnetes Wörterbuch auf n Schlüsseln, das als AVL-Baum implementiert ist. Die Funktion countAllRange(D, k_1, k_2) zählt die Anzahl der Schlüssel k in D, für die $k_1 \leq k \leq k_2$ gilt. Zeigen Sie, dass man die Funktion CountAllRange(D, k_1, k_2) implementieren kann, so dass die Laufzeit Ihrer Implementierung $O(\log n)$ ist. (Hinweis: Erweitern Sie die Datenstruktur geeignet.)

Aufgabe 2

Es seien T und U zwei $(2, 4)$-Bäume, die n bzw. m Schlüssel enthalten. Alle Schlüssel in T sind kleiner als die Schlüssel in U. Die Funktion Join(T, U) fügt zwei derartige $(2, 4)$-Bäume zu einem $(2, 4)$-Baum zusammen, der alle $n + m$ Schlüssel enthält. Zeigen Sie, wie Sie Join(T, U) implementieren, so dass die Laufzeit Ihrer Implementierung $O(\log n + \log m)$ ist. (Hinweis: Die alten Bäume T und U können zerstört werden.)

Aufgabe 3

Ein Misch-Heap ist eine Datenstruktur, die die Operationen Insert(k, x), Delete(k), UnionWith(L) und MinKey() unterstützt, wobei UnionWith(L) den Heap H und den Heap L zu einem neuen Heap zusammenfügt. Beschreiben Sie, wie Sie die obigen Operationen implementieren würden, so dass Ihre Implementierung für jede Operation eine Laufzeit von $O(\log n)$ hat, wobei n die Anzahl der Elemente im Heap ist. Nehmen Sie hierzu an, dass alle Schlüssel verschieden sind.

Aufgabe 4

Wir betrachten unsortierte Listen mit den Operationen Insert(x_i) („Füge x_i ein“), Find(x_i) („Finde x_i“) und Delete(x_i) („Lösche x_i“). Um auf ein Element zuzugreifen (Insert, Find), wird die Liste immer vom Anfang an durchsucht. Die Kosten sind also proportional zur Position des Elementes in der Liste.
Zeigen Sie, dass für eine gegebene Menge von Elementen $(x_1, \ldots, x_n)$ und eine feste Zugriffsfolge Find(x_{i_1}), Find(x_{i_1}), $\ldots$, Find(x_{i_m}), die Liste, welche die Elemente in der Häufigkeit ihrer Zugriffe anordnet, optimal ist. (D.h. es gibt keine Listenordnung mit einer besseren Gesamtlaufzeit).