Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Prof. Dr. Ernst W. Mayr
Matthias Baumgart

Wintersemester 2006/2007
Übungsblatt 7
1. Dezember 2006

Effiziente Algorithmen und Datenstrukturen I

Abgabetermin: 08.12.2006 **vor** *der Vorlesung*

Aufgabe 1 (10 Punkte)

In einem Algorithmus soll ein großes Array verwendet werden. Der Algorithmus verlangt, dass zu Beginn alle Arraypositionen initialisiert werden (zum Beispiel mit Null). Einerseits dauert uns das Initialisieren des Arrays aber zu lang, andererseits kann in einem nicht-initialisierten Speicher natürlich beliebiger Unsinn stehen. Beschreiben Sie eine Möglichkeit, wie man ohne die aufwendige Initialisierung auskommen kann. Genauer gesagt soll beim lesenden Zugriff auf eine Array-Position das Folgende passieren:

- Falls der Algorithmus vorher einen Eintrag in diese Position geschrieben hat, wird dieser Eintrag zurückgegeben.
- Falls in diese Position noch nicht geschrieben wurde (diese also „Bitmüll“ enthält), wird Null zurückgegeben.

Wie kann man ein solches Array so implementieren, dass ein (lesender oder schreibender) Zugriff auf ein Arrayelement Zeit $O(1)$ kostet und dass wir beim lesenden Zugriff immer die richtige Antwort zurückgeliefert bekommen? Das Zurücksetzen des Arrays (jeder lesende Zugriff liefert wieder Null) soll ebenfalls in Zeit $O(1)$ möglich sein.

Hinweis: Benutzen Sie eine Extradatenstruktur, in der steht, welche Positionen des Arrays schon *echt* beschrieben wurden. (Dort werden sozusagen *Zeugen* der Echtheit verwaltet.) Benutzen Sie außerdem eine Extrainformation in jeder Arrayposition, die gegebenenfalls auf den Zeugen verweist.

Aufgabe 2 (10 Punkte)

Schlagen Sie eine Erweiterung der 2-Level-Buckets PQ vor, so dass auch Werte effizient eingefügt werden können, die unter dem minimalen Wert liegen. Führen Sie eine amortisierte Analyse Ihrer Lösung mit einer geeignet gewählten Potentialfunktion durch.

Aufgabe 3 (10 Punkte)

Verallgemeinern Sie die Idee der 2-Level-Buckets auf k-Level-Buckets. Die amortisierte Laufzeit von Insert und DecreaseKey soll wieder $O(1)$ sein, die von ExtractMin nur noch $O(\sqrt[k]{C})$, wobei C wieder die obere Schranke für die Differenz zwischen dem größten und dem kleinsten Schlüssel sein soll.

Hinweis: Sie können zur Vereinfachung annehmen, dass nur Schlüssel aus der Menge $\{0, \ldots, C-1\}$ in den k-Level-Bucket eingefügt werden

Aufgabe 4 (10 Punkte)

Stellen Sie die van-Emde-Boas Priority Queue für die Menge

$$S = \{1, 2, 3, 4, 6, 7, 18, 21, 28, 31\} \subset \{0, 1, \ldots, 31\}$$

mit allen auftretenden k-Strukturen graphisch dar. Führen Sie auf dieser Priority Queue eine ExtractMin-Operation und danach die Operation Insert(5) aus, und stellen Sie die wesentlichen Schritte und das jeweilige Ergebnis dieser Operationen wiederum graphisch dar.