Technische Universität München
Institut für Informatik
Lehrstuhl für Effiziente Algorithmen
Prof. Dr. Ernst W. Mayr

WS 2002/03
Übungsblatt 8
6. Dezember 2002

Effiziente Algorithmen und Datenstrukturen I

Letzter Abgabetermin: 16. Dezember 2002 (vor der Übung)

Aufgabe 1

Für ein beliebiges $k \in \mathbb{N}$ gelte $N = 2^k \log k$ und $S \subseteq [0 \,..\, N-1]$. Ferner sei $S_i = \{x \in S \mid i \log k \leq x < (i+1) \log k\}$ für $0 \leq i < 2^k$. Die Menge S ist wie folgt dargestellt.

Die Menge $\{i \mid S_i \neq \emptyset\}$ ist in einer van-Emde-Boas Priority-Queue und jede nichtleere Menge S_i in einer linearen Liste gespeichert.

a) Zeigen Sie, dass diese Datenstruktur auf linearem Platz $\mathcal{O}(N)$ gespeichert und in linearer Zeit $\mathcal{O}(N)$ aufgebaut werden kann.

b) Ferner ist zu beweisen, dass die drei Operationen Insert, Delete und Find_Min in Zeit (worst-case) $\mathcal{O}(\log \log N)$ durchgeführt werden können.

Aufgabe 2

In der Vorlesung wurde die Laufzeit $\mathcal{O}(k \log^* n)$ für eine Union-Find-Struktur mit *weighted union* und *Pfadkompression* bewiesen. Hierbei wurde für jeden Knoten der folgende Wert definiert:

$$\mathit{rank}(x) = \text{Höhe des Unterbaums mit Wurzel } x \text{ in der Datenstruktur } \textit{ohne} \text{ Pfadkompression.}$$

a) Zeigen Sie, dass für alle Knoten in der Datenstruktur mit Pfadkompression gilt, dass die Werte von *rank* auf allen Pfaden von den Blättern zur Wurzel strikt steigen.

b) Sei $\mathit{rank}_p(x)$ der Wert von *rank* des aktuellen Vaters vom Knoten x. Zeigen Sie, dass $\mathit{rank}_p(x)$ bei einem Wechsel des Vaterknotens von x nicht kleiner wird.

Aufgabe 3

Eine Union-Find-Struktur sei durch In-Trees definiert. Die Union-Operation ist jedoch nicht mittels *weighted union* realisiert, sondern es ist erlaubt, die Wurzel eines beliebigen der beiden Bäume als Kind der Wurzel des anderen Baumes einzuhängen. Die Find-Operation soll mit Pfadkompression durchgeführt werden.

Zeigen Sie, dass mit dieser Variante der Union-Operation die Laufzeit $\mathcal{O}(k \log^* n)$ für eine Folge von Operationen nicht erreicht werden kann (k sei die Anzahl aller Operationen, mit genau n Make-New-Set Aufrufen).

Hinweis: Konstruieren Sie zuerst einen Binomialbaum $B_{\frac{n}{2}}$ und führen Sie dann eine geeignete Folge von Make-New-Set, Union und Find Operationen durch, sodass insgesamt eine Laufzeit $\Omega(k \log n)$ erzielt wird.

Aufgabe 4

Das *off-line minimum* Problem arbeitet über einer variablen Menge T von Elementen aus dem Universum $\{1, 2, \ldots, N\}$ und den Operationen INSERT und EXTRACT-MIN. Als Eingabe bekommt das Problem eine Sequenz S von n INSERT und m EXTRACT-MIN-Operationen, wobei jeder Schlüssel aus $\{1, 2, \ldots, n\}$ höchstens einmal eingefügt wird. Die Aufgabe besteht darin, zu bestimmen, welcher Schlüssel bei jedem EXTRACT-MIN Aufruf zurückgegeben wird. Präzise formuliert soll ein Array $\mathit{extracted}[1..m]$ zurückgeliefert werden, wobei in $\mathit{extracted}[i]$ der Schlüssel steht, der beim i-ten Aufruf von EXTRACT-MIN zurückgegeben wird. Das Problem ist als *off-line*-Problem definiert, das bedeutet, dass die komplette Sequenz S zu Beginn bekannt ist.

Zur Lösung des Problems wird folgende Vorgehensweise vorgeschlagen. In der Sequenz S wird jedes INSERT durch die Zahl, die eingefügt wird, und jedes EXTRACT-MIN durch den Buchstaben E ersetzt, also z.B.

$$4, 8, E, 3, E, 9, 2, 6, E, E, E, 1, 7, E, 5.$$

Für den Algorithmus wird diese Folge wie folgt in Teilsequenzen aufgeteilt:

$$I_1, E, I_2, E, I_3, \ldots, I_m, E, I_{m+1}$$

wobei jedes E wiederum einen einzelnen EXTRACT-MIN Aufruf repräsentiert und jedes I_j für eine (möglicherweise leere) Folge von INSERTs steht. Zu Beginn wird nun für jedes I_j eine Menge K_j definiert, die alle Schlüssel enthält, die durch I_j eingefügt werden. Falls I_j leer ist, ist K_j die leere Menge. Nun wird folgender Algorithmus aufgerufen:

OFF-LINE-MINIMUM(m, n)
for i **from** 1 **to** n
  do determine j such that $i \in K_j$
    if $j \neq m + 1$
      then $\mathit{extracted}[j] = i$
        l = smallest value greater than j for which set K_j still exists
        $K_l = K_l \cup K_j$
        destroy K_j
    fi
  od
return *extracted*

a) Zeigen Sie, dass das Array *extracted*, das vom vorgeschlagenen Algorithmus zurückgegeben wird, der Aufgabenstellung entspricht.

b) Beschreiben Sie, wie man eine Union-Find-Datenstruktur verwenden kann, um den Algorithmus OFF-LINE-MINIMUM effizient zu implementieren. Bestimmen Sie die *worst-case* Laufzeit Ihrer Implementierung.