Johannes Blömer
u.v.a.m.

Paderborn, den 24. Mai

Musterlösungen zur Vorlesung

Datenstrukturen und Algorithmen

SS 2004

Blatt 4

AUFGABE 1:
Der Algorithmus HEAP-DELETE(A, i) erhöht zunächst mit Hilfe des Algorithmus HEAP-INCREASE-KEY den Schlüssel des Knoten i auf ∞. Damit wird dann der Schlüssel ∞ in der Wurzel des Heaps stehen. Mit dem Algorithmus HEAP-EXTRACT-MAX entfernen wir dann diese Wurzel aus dem Heap. Wir verzichten allerdings in HEAP-EXTRACT-MAX auf die Ausgabe des Maximums. Hier ist der Algorithmus in Pseudocode.

HEAP-DELETE(A, i)

```
1 if heap-size[A] < i
2    then error "Heap zu klein"
3 HEAP-INCREASE-KEY(A, i, ∞)
4 HEAP-EXTRACT-MAX(A)
```

Die Korrektheit des Algorithmus folgt aus der Korrektheit von HEAP-INCREASE-KEY und HEAP-EXTRACT-MAX. Da die Laufzeit von HEAP-INCREASE-KEY und HEAP-EXTRACT-MAX jeweils $\mathcal{O}(\log(n)$ ist, ist auch die Laufzeit von DELETE-HEAP $\mathcal{O}(\log(n)$.

AUFGABE 2:
Hier ist zunächst die Strategie die unser Algorithmus HEAP-MERGE verfolgen wird. Der Algorithmus sucht immer unter den noch nicht einsortierten Elementen das größte Element heraus, um es in die nächste Position des Ausgabearrays B zu kopieren. Um unter den noch nicht einsortierten Elementen das Maximum zu finden, genügt es, für jedes Array A_i das größte noch nicht einsortierte Element zu betrachten und unter diesen k Zahlen das Maximum zu finden. Diese wiederholte Maximumsuche führt der Algorithmus HEAP-MERGE mit Hilfe eines k-elementigen Heaps durch. Dieser Heap enthält zu jedem Zeitpunkt aus jedem Eingabearray A_i das größte bislang noch nicht einsortierte Element. Nutzt man weiterhin aus, dass die Arrays A_i absteigend sortiert sind und wird die absteigend sortierte Gesamtfolge im Array B gespeichert, erhält man den folgenden Algorithmus

HEAP-MERGE

1. Schritt: Füge an jedes der Eingabearray ein Feld mit Schlüssel $-\infty$ an.

2. Schritt: Mit Hilfe des Algorithmus BUILD-MAX-HEAP errichte einen max-Heap H auf den jeweils ersten Elementen $A_1[1], \ldots, A_k[1]$ der Eingabearrays $A_1, \ldots, A_k$.

3. Schritt: Für $i \leftarrow 1$ bis n wiederhole folgende Schritte:

a) Setze $B[i] \leftarrow H[1]$ und entferne $H[1]$ aus dem Heap H mit HEAP-EXTRACT-MAX.

b) War das zuletzt aus H entfernte Element das j-te Element aus A_l, so füge mit Hilfe von MAX-HEAP-INSERT ein Element mit Schlüssel $A_l[j+1]$ in den Heap H ein.

Der 1. Schritt hat nur technische Gründe. Mit Hilfe der Einträge ∞ sparen wir uns Abfragen, ob Arrays bereits leer sind. In Schritt 3b) muss der Algorithmus wissen, welche Elemente aus dem Arrays $A_1, \ldots, A_k$ sich aktuell im Heap befinden. Dieses kann erreicht werden, indem man die Positionen der sich im Heap befindlichen Elemente in einem k-elementigen Array verwaltet. Weiter muss man von jedem Heap-Element wissen, aus welchem der Eingabearrays es stammt. Dieses kann erreicht werden, indem für jeden Knoten des Heaps zusätzlich gespeichert wird, aus welchem Array die in diesem Knoten aktuell gespeicherte Zahl stammt. Um die Korrektheit des Algorithmus HEAP-MERGE zu zeigen, müssen wir zeigen, dass in der Schleife im 3. Schritt jeweils das größte noch nicht einsortierte Element gefunden und nach B kopiert wird. Dieses ergibt sich aber aus der Organisation von H. weiter ist zu beachten, dass die max-Heap-Eigenschaft von H durch das Einfügen eines neuen Elements mit Hilfe von MAX-HEAP-INSERT immer erhalten bleibt.
Nun zur Laufzeit. Der erste Schritt benötigt Zeit $\mathcal{O}(k)$. Nach der Ergebnissen der Vorlesung über den Algorithmus BUILD-MAX-HEAP benötigt der 2. Schritt ebenfalls Zeit $\mathcal{O}(k)$. Die Schleife im 3. Schritt wird n-mal durchlaufen. In jedem Durchlauf der Schleife wird einmal HEAP-EXTRACT-MAX und einmal MAX-HEAP-INSERT jeweils auf k-elementigen Heaps aufgerufen. Damit erfordert jeder Schleifendurchlauf Zeit $\mathcal{O}(\log(k))$. Die Schleife insgesamt erfordert Zeit $\mathcal{O}(n\log(k))$. Damit ist die Gesamtlaufzeit $\mathcal{O}(n\log(k))$.

AUFGABE 3:
Wir zeigen die Korrektheit mittels des folgenden Invariante:

Invariante: Vor Durchlauf der Schleife in den Zeilen 2 und 3 mit Index i ist das Teilarray $A[1..i-1]$ ein max-Heap auf den ersten $i-1$ Eingabezahlen.

Jetzt zeigen wir Initialisierung, Erhaltung und Terminierung.

Initialisierung: Der erste Schleifendurchlauf erfolgt mit $i = 2$. Also müssen wir zeigen, dass vor dem Durchlauf mit $i = 2$ das Teilarray $A[1..1] = A[1]$ ein max-Heap auf der ersten Eingabezahl ist. Aber jedes Array bestehend aus einem Element ist ein max-Heap.

Erhaltung: Sei die Invariante vor dem Durchlauf mit Index i erfüllt, d.h., das Teilarray $A[1..i-1]$ ist ein max-Heap auf den ersten $i-1$ Eingabezahlen. Dann aber wird durch den Algorithmus MAX-HEAP-INSERT bei Aufruf mit Parametern $A, A[i]$ diesen Heap um den Eintrag $A[i]$ erweitern. Die Erhaltung folgt also aus der Korrektheit von MAX-HEAP-INSERT.

Terminierung: Die Invariante besagt, dass vor Durchlauf der Schleife mit $i = length[A] + 1$ das Array $A[1..length[A]$ ein max-Heap auf den ersten $length[A]$ vielen Eingabezahlen ist. Damit ist aber auf allen Eingabezahlen ein max-Heap aufgebaut und der Algorithmus BUILD-MAX-HEAP' ist korrekt.

Nun zur Laufzeit des Algorithmus BUILD-MAX-HEAP'. Diese ergibt sich aus der Laufzeit von Algorithmus MAX-HEAP-INSERT. Wir wissen, dass die Laufzeit dieses Algorithmus bei Aufruf mit einem Heap der Größe n von der Form $\mathcal{O}(\log(n))$ ist. Abgesehen von Zeile 1, die

nur konstante Zeit benötigt, ruft Algorithmus BUILD-MAX-HEAP' den Algorithmus MAX-HEAP-INSERT n-mal mit Heaps der Größe höchstens n auf, hierbei ist $n = length[A]$. Damit ist die Laufzeit von BUILD-MAX-HEAP' $\mathcal{O}(n \log(n))$.