Johannes Blömer
u.v.a.m.

Paderborn, den 15.Juni 2004

Musterlösungen zur Vorlesung

Datenstrukturen und Algorithmen

SS 2004

Blatt 7

AUFGABE 1:
In diesem Fall ist der Median von $2n$ Zahlen gesucht, also eine der Eingabezahlen, die größer ist als $n-1$ andere Eingabezahlen. Oder äquivalent, so dass $n-1$ andere Zahlen kleiner sind.
Hier ist eine informelle Beschreibung des Algorithmus, den wir SORTED-MEDIAN nennen.

SORTED-MEDIAN(U,T)

1.Schritt Falls die Arrays U und T jeweils nur ein Element enthalten, gebe das Minimum von $U[1], T[1]$ aus.

2.Schritt Falls $U[\lfloor (n+1)/2 \rfloor] \leq T[\lfloor (n+1)/2 \rfloor]$, benutze SORTED-MEDIAN rekursiv um den gemeinsamen Median der Zahlen in $U[\lceil (n+1)/2 \rceil .. n]$ und $T[1..\lfloor (n+1)/2 \rfloor]$ zu finden.

3.Schritt Falls $T[\lfloor (n+1)/2 \rfloor] \leq U[\lfloor (n+1)/2 \rfloor]$, benutze SORTED-MEDIAN rekursiv um den gemeinsamen Median der Zahlen in $T[\lceil (n+1)/2 \rceil .. n]$ und $U[1..\lfloor (n+1)/2 \rfloor]$ zu finden.

Um die Korrektheit dieses Algorithmus zu zeigen, müssen wir zeigen

1. Die Ausgabe im 1.Schritt ist korrekt.
2. Die beiden Intervalle im 2. bzw. 3.Schritt haben jeweils gleiche Größe.
3. Im 2. und 3. Schritt ist der gemeinsame Median der jeweiligen Teilarrays der gemeinsame Median der ursprünglichen Arrays.

2. wird hierbei benötigt, um zu zeigen, dass die rekursiven Aufrufe im 2. und 3.Schritt überhaupt Sinn haben.

zu 1. Bei 2 Zahlen ist der Median gerade das Minimum der beiden Zahlen. Daher ist die Ausgabe im 1.Schritt korrekt.

zu 2. Sowohl im 2. als auch im 3.Schritt hat eines der Teilarrays des rekursiven Aufrufs $\lfloor (n+1)/2 \rfloor$ Elemente und das andere Teilarray besitzt $n - \lceil (n+1)/2 \rceil + 1$ Elemente. Man überprüft nun leicht, das für jede natürliche Zahl m gilt

$$\lfloor m/2 \rfloor + \lceil m/2 \rceil = m.$$

Daraus folgt sofort

$$\lfloor (n+1)/2 \rfloor = n - \lceil (n+1)/2 \rceil + 1.$$

zu 3. Wir betrachten nur die Situation im 2.Schritt. Im 3.Schritt ist alles symmetrisch, mit den Rollen von U und T vertauscht.

Zunächst einige Notationen. Wir setzen

$$\begin{array}{lcl} U_0 & := & U[1..\lceil (n+1)/2 \rceil - 1] \\ U_1 & := & U[\lceil (n+1)/2 \rceil..n] \\ T_0 & := & T[1..\lfloor (n+1)/2 \rfloor] \\ T_1 & := & T[\lfloor (n+1)/2 \rfloor + 1..n] \end{array}$$

Seien u_0, u_1, t_0, t_1 die Anzahl der Zahlen in U_0, U_1, T_0, T_1. Es gilt

$$\begin{array}{lcl} u_0 & = & \lceil (n+1)/2 \rceil - 1 \\ u_1 & = & n - \lceil (n+1)/2 \rceil + 1 = \lfloor (n+1)/2 \rfloor \\ t_0 & = & \lfloor (n+1)/2 \rfloor \\ t_1 & = & n - \lfloor (n+1)/2 \rfloor. \end{array}$$

Weiter sei m ein Median der Zahlen in den Teilarrays U_1, T_0. Die Zahl m ist also größer als $\lfloor (n+1)/2 \rfloor - 1$ andere Zahlen in U_1, T_0. Es ist zu zeigen, dass m größer ist als $n-1$ andere Zahlen in U, T zusammen. Hierzu unterscheiden wir zwei Fälle: m liegt im Teilarray U_1 und m liegt im Teilarray T_0.

m in U_1: Nun ist m größer als alle Zahlen in U_0. Ausserdem muss jede Zahl in T_1 größer als m sein, denn gibt es eine Zahl in T_1, die kleiner als m ist, müssen auch alle Zahlen in T_0 kleiner als m sein. Dieses kann aber nicht sein, da es in T_0 und U_1 zusammen nur $\lfloor (n+1)/2 \rfloor - 1$ Zahlen gibt, die kleiner m sind, und T_0 alleine bereits $\lfloor (n+1)/2 \rfloor$ Zahlen enthält. Damit ist die Anzahl der Zahlen in U, T, die kleiner als m sind, gegeben durch

$$u_0 + \lfloor (n+1)/2 \rfloor - 1 = \lceil (n+1)/2 \rceil - 1 + \lfloor (n+1)/2 \rfloor - 1 = n + 1 - 2 = n - 1.$$

Dann also ist m auch der gesuchte Median der Zahlen in U, T.

m in T_0: Dann sind sicherlich alle Zahlen in T_1 größer als m. Wir zeigen nun, dass alle Zahlen in U_0 kleiner als m sind. Gibt es ein Element in U_1, das kleiner als m ist, so gilt dieses auch für alle Zahlen in U_0. Wir müssen also nur noch den Fall betrachten, dass alle Zahlen in U_0 größer als m sind. Da aber m der Median der Zahlen in U_1, T_0 ist und T_0 genau $\lfloor (n+1)/2 \rfloor$ Zahlen enthält, gilt dann $m = T[\lfloor (n+1)/2 \rfloor]$. Nun gilt aber

$$m = T[\lfloor (n+1)/2 \rfloor] \leq U[\lfloor (n+1)/2 \rfloor] \leq U[\lceil (n+1)/2 \rceil - 1],$$

denn $\lceil (n+1)/2 \rceil - 1 \geq \lfloor (n+1)/2 \rfloor$. Dann aber sind alle Zahlen in U_0 wie behauptet kleiner als m. Da nach Definition von m in den Teilarrays U_1, T_0 die Anzahl der Zahlen, die kleiner als m sind, gegeben ist durch $\lfloor (n+1)/2 \rfloor - 1$, ist die Anzahl der Zahlen in U, T, die kleiner als m sind, gegeben durch

$$u_0 + \lfloor (n+1)/2 \rfloor - 1 = n - 1.$$

Damit ist auch in diesem Fall m der Median der Zahlen in U, T.

Nun zur Laufzeit von Algorithmus SORTED-MEDIAN. Mit jedem rekursivem Aufruf wird die Problemgröße halbiert. Ausserdem wird pro rekursiven Aufruf zusätzlich konstante Zeit benötigt. Wir erhalten daher für die Laufzeit $T(n)$ des Algorithmus die Rekursionsgleichung

$$T(n) = T(n/2) + c,$$

wobei c eine Konstante ist. Wie wir schon häufiger gesehen haben, gilt dann $T(n) = \mathcal{O}(\log(n))$.

AUFGABE 2:
In der Vorlesung haben wir gesehen, dass der (untere) Median von n Zahlen in Zeit $\mathcal{O}(n)$ berechnet werden kann. Mit MEDIAN(A) bezeichnen wir einen Algorithmus, der bei Eingabe eines Arrays A, den Median der Zahlen in A in Zeit linear in der Länge von A berechnet. Genauer nehmen wir an, dass MEDIAN(A) den Index des Medians der Zahlen in A zurückgibt.
Nun benutzen wir die folgende Partitionsfunktion

MEDIAN-PARTITION(A, p, r)

```
1 i ← MEDIAN(A[p..r])
2 A[r] ↔ A[i]
3 return PARTITION(A, p, r)
```

Hierbei ist PARTITION die Partitionsfunktion, die wir bei Quicksort kennengelernt haben.
Wir erhalten dann die folgende Variante von Quicksort.

MEDIAN-QUICKSORT(A, p, r)

```
1 if p < r
2   then q ← MEDIAN-PARTITION(A, p, r)
3        MEDIAN-PARTITION(A, p, q − 1)
4        MEDIAN-PARTITION(A, q + 1, r)
```

Die Korrektheit dieses Algorithmus folgt wie bei Quicksort selber. Nun zur Laufzeit. Der Algorithmus
MEDIAN-PARTITION(A, p, r) wird nach Definition des Medians und nach Definition des Algorithmus PARTITION das Array $A[p..r]$ in ein Teilarray $A[p..q-1]$ mit $\lfloor (n+1)/2 \rfloor - 1$ Elementen und in ein Teilarray $A[q+1..r]$ mit $n - \lfloor (n+1)/2 \rfloor$ Elementen aufteilen. Hierbei ist $n = r - p + 1$ die Länge von $A[p..r]$. Nun gilt

$$\lfloor (n+1)/2 \rfloor - 1 \leq n/2 \text{ und } n - \lfloor (n+1)/2 \rfloor \leq n/2.$$

Damit ist die Laufzeit $T(n)$ von MEDIAN-QUICKSORT bei Eingabe eine Arrays mit n Elementen gegeben durch

$$T(n) \leq 2T(n/2) + cn,$$

wobei c eine Konstante ist und die Laufzeit von MEDIAN-PARTITION durch cn beschränkt ist. Diese Rekursion haben wir z.B. bei der Analyse von MERGE-SORT bereits betrachtet und gesehen, dass Sie auf $T(n) = \mathcal{O}(n \log(n))$ führt.

AUFGABE 3:
Den ersten Stack S_1 organisieren wir wie in der Vorlesung beschrieben in einem Array S. Den zweiten Stack S_2 jedoch starten wir an der letzten Arrayposition $S[n]$. Beim Pushen eines Elementes wird dieses dann *vor* die bereits im Stack befindlichen Elemente positioniert. Hier sind die Algorithmen für die wesentlichen Stackoperationen des so organisierten Stack.

STACK-EMPTY'(S_2)

```
1 if top[S2] = n + 1
2   then return TRUE
3   else return FALSE
```

PUSH'(S_2, x)

```
1 top[S_2] ← top[S_2] − 1
2 S[top[S_2]] ← x
```

POP'(S_2)

```
1 if STACK-EMPTY
2    then error "underflow"
3    else top[S_2] ← top[S_2] + 1
4         return S[top[S_2] − 1]
```

Sind nun nie mehr als n Elemente in den beiden Stacks S_1, S_2 gemeinsam gespeichert, so wird keine Arrayposition von beiden Stacks beansprucht. Beide Stacks können dann korrekt arbeiten.

AUFGABE 4:
Zu einem Überlauf bei ENQUEUE kommt es, wenn n Elemente in einer Queue realisiert in einem n-elementigen Array gespeichert werden sollen. Dass das einzufügenden Element das n-te Element der Queue wäre, erkennt man daran, dass $tail[Q]$ die Position unmittelbar *vor* $head[Q]$ ist, also $tail[Q] = head[Q] - 1$ modulo n. Benutzten wir den Algorithmus ENQUEUE aus der Vorlesung, erhalten wir dann folgenden Algorithmus ENQUEUE', der auch Überläufe abfängt.

ENQUEUE'(Q, x)

```
1 if tail[Q] = head[Q] − 1  mod n
2    then error "Überlauf"
3    else ENQUEUE(Q, x)
```