Institut für Informatik I
Prof. Dr. Heiko Röglin
Tobias Brunsch

universitätbonn

Algorithmen und Berechnungskomplexität I
Wintersemester 2010/11

Musterlösung Übungsblatt 7

Aufgabe 7.4

Ansatz 1

Zunächst diskutieren wir eine elegante Lösung zur Realisierung von `Init` und `PrintSmallerValues`, die von einigen Studenten vorgeschlagen wurde. Leider ist nicht klar, wie man mit diesem Ansatz die `PrintSmallest`-Funktion umsetzen soll. Wir würden uns über jede Idee zur Behebung dieses Problems freuen.

In der `Init`-Funktion bringen wir das Feld A auf *min-Heap*-Gestalt. In der Vorlesung wurde (für *max-Heaps*) gezeigt, dass dies in Zeit $O(n)$ möglich ist, wenn n die Länge von A ist. Ein Aufruf `PrintSmallerValues`(z) lässt sich durch einen Aufruf `PrintSmallerValues`$(z, 1)$ realisieren. Die Funktion `PrintSmallerValues`(z, i) gibt dabei alle Einträge des Heaps unter der Wurzel i aus, die kleiner oder gleich z sind. Außerdem nehmen wir hier an, dass die Einträge von A beginnend mit 1 indiziert sind. Die allgemeine `PrintSmallerValues`-Funktion kann wie folgt implementiert werden.

```
PrintSmallerValues(z, i)
{
    if i > n then return
    if A[i] > z then return
    Gib A[i] aus
    PrintSmallerValues(z, 2i)
    PrintSmallerValues(z, 2i + 1)
}
```

Von den betrachteten Einträge $A[i]$ werden genau die ausgegeben, die kleiner oder gleich z sind. Es werden nur Einträge von A nicht betrachtet, die sich in einem Teilbaum des *min-Heaps* befinden, dessen Wurzel größer als z ist. (In einem formalen Beweis müsste man das noch beweisen.) Aufgrund der *min-Heap*-Eigenschaft sind diese auch größer als z, müssen also nicht betrachtet werden. Daher werden genau die Einträge $A[i]$ ausgegeben, die kleiner oder gleich z sind.

Für die Laufzeitanalyse betrachten wir nur den Fall, dass mindestens ein Eintrag ausgegeben wird. Der andere Fall ist trivial. Statt eine Rekursionsgleichung aufzustellen und zu lösen zählen wir alle Aufrufe der Gestalt `PrintSmallerValues`(z, i). Da das Durchführen jedes Aufruf (ohne den rekursiven Aufwand, den wir bei den daraus resultierenden Aufrufen bereits berücksichtigen) $O(1)$ Zeit benötigt, ist die Gesamtlaufzeit gleich $\Theta(m)$, wenn m die Anzahl der Aufrufe ist. Sei m_1 die Anzahl der erfolgreichen Aufrufe. Dies seien die Aufrufe, bei denen $A[i]$ ausgegeben wird. Außerdem sei m_2 die Anzahl der restlichen, also erfolglosen Aufrufe. Da der Algorithmus korrekt ist, gilt $m_1 = k$. Jeder erfolglose Aufruf wurde von einem erfolgreichen Aufruf aus aufgerufen. Da in jedem Aufruf maximal zwei rekursive Aufrufe erfolgen, gilt daher $m_2 \leq 2m_1$ und damit $m = m_1 + m_2 \leq 3m_1 = 3k = \Theta(k)$ wie gefordert.

Zum Abschluss dieses Ansatzes betrachten wir noch das Problem bei der Implementierung der `PrintSmallest`-Funktion. Eine Möglichkeit wäre, die `PrintSmallerValues`-Funktion zu verwenden. Dazu kann man den k-t kleinsten Eintrag z von A bestimmen und dann `PrintSmallerValues` mit diesem Wert aufrufen. Bei doppelten Einträgen muss man etwas vorsichtig sein, aber das wesentliche Problem ist die Bestimmung von z. Dies ist laut Hinweis in Zeit $O(n)$, aber für $k = o(n)$ nicht in Zeit $O(k)$ möglich. Man könnte auch einen ähnlichen Ansatz wie `PrintSmallerValues` wählen, die Rekursion als Job-Abarbeitung verstehen und durch eine Job-Verarbeitung mit Hilfe einer *Priority-Queue* ersetzen. Dieser Ansatz benötigt jedoch Zeit $\Theta(k \log k)$ statt $\Theta(k)$.

Ansatz 2

Wie schon im Hinweis beschrieben, ist das Sortieren des Feldes zu aufwändig. Wir werden daher das Feld nur (unvollständig) vorsortieren. Da wir für kleine Zahlen k weniger Zeit haben, die k kleinsten Einträge des Feldes zu lokalisieren, müssen kleine Einträge besonders gut vorsortiert sein. Sei l die eindeutig bestimmte natürliche Zahl mit $2^l \leq n < 2^{l+1}$. Wir setzen $p_i = 2^i$, $i = 0, \ldots, l$ sowie $p_{l+1} = n$ und $p_{l+2} = n + 1$ und wollen das Feld A so umsortieren, dass für alle $i = 0, \ldots, l$ gilt:

(1) Die Einträge $A[1], \ldots, A[p_i - 1]$ sind nicht größer als $A[p_i]$.

(2) Die Einträge $A[p_i + 1], \ldots, A[n]$ sind nicht kleiner als $A[p_i]$.

Das kann erreicht werden, indem in der `Init`-Funktion zunächst der größte Feldeintrag an Position $p_{l+1} = n$ vertauscht wird und anschließend für $i = l, \ldots, 0$ folgende zwei Schritte durchgeführt werden.

1. Bestimme den p_i-t kleinsten Eintrag von $A[1], \ldots, A[p_{i+1} - 1]$.
2. Partitioniere das Feld $A[1], \ldots, A[p_{i+1} - 1]$ mit diesem Pivotelement. Sollte das Pivotelement mehrmals in dem Teilfeld vorkommen, stelle sicher, dass nach der Partitionierung alle diese Einträge in einem zusammenhängenden Block stehen.

Für den Fall, dass Zahlen mehrfach in dem Feld vorkommen, sollten wir noch spezifizieren, was der k-t kleinste Eintrag des Feldes ist. Das ist diejenige Zahl, die an Position k des Feldes stehen würde, wenn wir das Feld sortieren würden. Laut Hinweis ist die Laufzeit des obigen Verfahrens $\sum_{i=0}^{l} \Theta(p_{i+1}) = \sum_{i=0}^{l} \Theta(2^{i+1}) = \Theta(2^{l+2}) = \Theta(n)$. Die Vorsortierung des Feldes durch die `Init`-Funktion können wir uns nun wie folgt zu Nutze machen.

`PrintSmallerValues`(z):

1. Bestimme die natürliche Zahl i mit $A[p_i] \leq z < A[p_{i+1}]$.
2. Gib die Einträge $A[1], \ldots, A[p_i]$ aus.
3. Falls $p_i < n$: Gib von den Einträgen $A[p_i + 1], \ldots, A[p_{i+1}]$ die Einträge aus, die kleiner oder gleich z sind.

Die Funktion `PrintSmallerValues` gibt von allen betrachteten Einträgen genau die aus, die kleiner oder gleich z sind. Es bleibt zu zeigen, dass die nichtbetrachteten Einträge größer als z sind. Die nichtbetrachteten Einträge sind die Einträge rechts von $A[p_{i+1}]$. In `Init` wurde sichergestellt, dass diese größer oder gleich $A[p_{i+1}]$ sind, und $A[p_{i+1}]$ ist per Konstruktion größer als z.

Zur Bestimmung von i sollten wir keine binärer Suche verwenden, da das eine Laufzeit $\Theta(\log n)$ implizieren würde, was für $k = o(\log n)$ gleich $\omega(k)$ ist. Stattdessen zählen wir i von 0 hoch, bis $i = l + 2$ oder $A[p_i] > z$, und dekrementieren es anschließend um 1. Dies ist in $\Theta(i + 1)$ möglich, was wegen $i = \Theta(\log p_i)$ und $p_i \leq k$ gleich $\Theta(\log k + 1) = O(k)$ ist. Da wir in Schritt 2 und 3 jeden Eintrag $A[1], \ldots, A[p_{i+1}]$ genau einmal betrachten und $p_{i+1} \leq 2p_i \leq 2k$ gilt, beträgt die Gesamtlaufzeit $O(k)$.

`PrintSmallest`(k):

1. Bestimme die natürliche Zahl i mit $p_i \leq k < p_{i+1}$.
2. Gib die Einträge $A[1], \ldots, A[p_i]$ aus.
3. Falls $k' := k - p_i = 0$, dann sind wir fertig.
4. Bestimme den k'-t kleinsten Eintrag z von $A[p_i + 1], \ldots, A[p_{i+1}]$.
5. Gib von den Einträgen $A[p_i + 1], \ldots, A[p_{i+1}]$ die Einträge aus, die kleiner als z sind. Dies seien k'' viele.
6. Gib von den Einträgen $A[p_i + 1], \ldots, A[p_{i+1}]$ die ersten $k' - k''$ Einträge aus, die gleich z sind.

Aufgrund der durch `Init` hergestellten Eigenschaften des Feldes und der Wahl von p_i gehören die Einträge $A[1], \ldots, A[p_i]$ zu den k kleinsten Feldeinträgen; umgekehrt sind die k kleinsten Feldeinträge unter den Einträgen $A[1], \ldots, A[p_{i+1}]$ zu finden. Die k kleinsten Einträge sind damit die Einträge $A[1], \ldots, A[p_i]$ und die k' kleinsten Einträge von $A[p_i + 1], \ldots, A[p_{i+1}]$. Der Aufwand, der in den Schritten 5 und 6 getrieben wird, ist nur nötig, weil Zahlen in dem Feld mehrfach vorkommen könnten. (In einem formalen Beweis wäre noch zu zeigen, dass tatsächlich die k' kleinsten Einträge von $A[p_i + 1], \ldots, A[p_{i+1}]$ ausgegeben werden.)

Das Bestimmen von i ist, wie oben, in Zeit $O(k)$ möglich. Da nur die Einträge $A[1], \ldots, A[p_{i+1}]$ betrachtet werden (ggf. zweimal), das Bestimmen des k'-t kleinsten Eintrages von $A[p_i + 1], \ldots, A[p_{i+1}]$ laut Hinweis in Zeit $O(p_{i+1})$ möglich ist, und wegen $p_i \leq k$ und $p_{i+1} \leq 2p_i$ ist die Gesamtlaufzeit gleich $O(p_{i+1}) = O(p_i) = O(k)$.