Prof. Dr. Leif Kobbelt
Stefan Dollase, Ira Fesefeldt, Alexandra Heuschling, Gregor Kobsik

Lösung - Übung 5

Aufgabe 5 (Quicksort): **12 Punkte**

Sortieren Sie das folgende Array mithilfe von Quicksort. Geben Sie dazu das Array nach jeder Partition-Operation an und markieren Sie das jeweils verwendete Pivot-Element.

7	3	6	4	5	1	8	9	2

Lösung

7	3	6	4	5	1	8	9	2
1	2	6	4	5	7	8	9	3
1	2	3	4	5	7	8	9	6
1	2	3	4	5	6	8	9	7
1	2	3	4	5	6	8	9	7
1	2	3	4	5	6	7	9	8
1	2	3	4	5	6	7	8	9

Aufgabe 6 (Mergesort): 8 Punkte

Sortieren Sie das folgende Array mithilfe von Mergesort. Geben Sie dazu das Array nach jeder Merge-Operation an.

7	3	6	4	5	1	8	9	2

Lösung

7	3	6	4	5	1	8	9	2
3	7	6	4	5	1	8	9	2
3	6	7	4	5	1	8	9	2
3	6	7	4	5	1	8	9	2
3	4	5	6	7	1	8	9	2
3	4	5	6	7	1	8	9	2
3	4	5	6	7	1	8	2	9
3	4	5	6	7	1	2	8	9
1	2	3	4	5	6	7	8	9

Aufgabe 7 (Heapsort): 11 Punkte

Sortieren Sie das folgende Array mithilfe von Heapsort. Geben Sie dazu das Array nach jeder Swap-Operation an.

7	3	6	4	5	1	8	9	2

Lösung

7	3	6	4	5	1	8	9	2
7	3	6	9	5	1	8	4	2
7	3	8	9	5	1	6	4	2
7	9	8	3	5	1	6	4	2
7	9	8	4	5	1	6	3	2
9	7	8	4	5	1	6	3	2
2	7	8	4	5	1	6	3	9
8	7	2	4	5	1	6	3	9
8	7	6	4	5	1	2	3	9
3	7	6	4	5	1	2	8	9
7	3	6	4	5	1	2	8	9
7	5	6	4	3	1	2	8	9
2	5	6	4	3	1	7	8	9
6	5	2	4	3	1	7	8	9
1	5	2	4	3	6	7	8	9
5	1	2	4	3	6	7	8	9
5	4	2	1	3	6	7	8	9
3	4	2	1	5	6	7	8	9
4	3	2	1	5	6	7	8	9
1	3	2	4	5	6	7	8	9
3	1	2	4	5	6	7	8	9
2	1	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9

Aufgabe 8 (d-Heaps): 3 + 2 + 2 + 2 = 9 Punkte

Ein Baum mit Grad d ist ein solcher Baum, bei dem jeder Knoten v maximal d viele Kinder hat. Ein d-Heap ist ein Baum mit Grad d in dem die erweiterte Heap-Bedingung gilt, d.h. für alle Knoten v und alle deren Teilbäume $t_1 \dots t_n$ gilt $\max(t_i) \leq v$.

a) Wir wollen zuerst eine Einbettung eines d-Heaps in das Array A finden. Wir wollen genauso wie bei regulären Heaps, dass wir links-vollständige Bäume mit Grad d in ein Array einbetten können und zwischen Einträgen im Array keine leeren Einträge sind.

i. Geben Sie nun eine Abbildung an, die für einen Elternknoten $A[i]$ die Position aller ihrer Nachfolger $A[j] \dots A[k]$ in einem vollständigen Baum mit Grad d angibt.

ii. In welchem Eintrag müssen Sie die Wurzel speichern?

iii. Begründen Sie auch die Korrektheit ihrer Abbildung!

b) Was ist die Höhe eines d-Heaps mit n vielen Knoten? Begründen Sie ihre Antwort!

c) *i.* Erklären Sie, wie Sie die Methode sink aus der Vorlesung so erweitern können, dass diese auch auf d-Heaps arbeitet.

ii. Geben Sie außerdem eine Laufzeitanalyse für den Worstcase in O-Notation ihrer erweiterten Methode sink in Abhängigkeit von dem Grad des Baumes d und der Anzahl der Knoten im Baum n an.

d) *i.* Erklären Sie schließlich noch, wie Sie die Methode heap_sort anpassen müssen, damit diese mit d-Heaps arbeiten können.

ii. Geben Sie auch für diese Methode eine Laufzeitanalyse für den Worstcase in O-Notation in Abhängigkeit von dem Grad des Baumes d und der Anzahl der Knoten im Baum n an.

Lösung

a) Wir benutzen hier abweichend von der Vorlesung ein Array das mit 0 indiziert ist, damit die Wurzel im Eintrag 0 gespeichert wird. Dann sind für einen Knoten $A[i]$ die Nachfolger gegeben als $A[d \cdot i + 1] \cdot A[d \cdot i + d]$. Für die Wurzel $i = 0$ ist dies klar, denn deren Nachfolger in den Einträgen $A[d \cdot 0 + 1] = A[1]$ bis $A[d \cdot 0 + d] = A[d]$ gespeichert werden. Für ein beliebigen Knoten $A[i]$ wissen wir, dass der letzte Nachfolger von $A[i-1]$ an der Stelle $A[d \cdot (i-1) + d] = A[d \cdot i]$ gespeichert wird und der erste Nachfolger von $A[i]$ an der Stelle $A[d \cdot i + 1]$ gespeichert wird. Damit gibt es zwischen den Nachfolgern von $A[i-1]$ und $A[i]$ keine Lücken, womit auch der ganze Baum lückenlos in das Array eingebettet wird.

b) Die Anzahl der Knoten in einem vollständigen Baum mit Grad d und der Tiefe k sind genau d^k. Auf Tiefe $k = 0$ haben wir $d^0 = 1$ - nur die Wurzel. Wir nehmen nun an für ein beliebiges, aber festes k ist die Anzahl der Knoten in einem vollständigen Baum mit Grad d und Tiefe k genau d^k, dann ist in einem vollständigen Baum mit Grad d und Tiefe $k+1$ auch $d^k \cdot d = d^{k+1}$.

Damit ist aber auch die Anzahl der Knoten in einem Baum mit Höhe h höchstens $\sum_{i=0}^{h} d^i = \frac{d^{h+1}-1}{d-1}$ (das lässt sich per vollständiger Induktion zeigen). Also ist die maximale Höhe mit n Knoten auch $\lceil \log_d(n \cdot d - n + 1) - 1 \rceil$, wobei $\lceil x \rceil$ den Wert von x aufrundet.

c) Statt nur $A[2 \cdot i]$ und $A[2 \cdot i + 1]$ zu vergleichen, müssen wir stets das Maximum aller $A[d \cdot i + 1]$ bis $A[d \cdot i + d]$ suchen und diesen dann mit $A[i]$ vergleichen. Damit haben wir für jede Iteration eine Laufzeit in $O(d)$.

Weiterhin müssen wir die Schleifenbedingung ändern, da wir in den inneren Knoten sind, solange $0 \leq i < \lceil (n-1)/d \rceil$. Für 0 bis $d+1$ Einträge lässt sich überprüfen, dass $\lceil (n-1)/d \rceil$ die Anzahl der inneren Knoten angibt. Da wir 0 indiziert sind, fordern wir hier dass der Index kleiner ist als die Anzahl. Für Einträge größer hilft uns die Beobachtung, dass das Hinzufügen von d Einträgen die Anzahl an inneren Knoten um 1 erhöht, womit die Anzahl an inneren Knoten von $n+d$ Einträgen genau $\lceil (n-1)/d \rceil + 1 = \lceil (n+d-1)/d \rceil$ ist.

Wir haben maximal so viele Iterationen wie der Baum Höhe hat, also maximal $\lceil \log_d(n \cdot d - n + 1) - 1 \rceil =$

$O(\frac{\log(n \cdot d)}{\log(d)}) = O(\frac{\log(n)+\log(d)}{\log(d)}) = O(\frac{\log(n)}{\log(d)} + 1) = O(\frac{\log(n)}{\log(d)})$ viele Iterationen.

Zusammen ist die Laufzeit also in $O(d \cdot \frac{\log(n)}{\log(d)})$.

d) Für die erste Schleife müssen wir lediglich den Start- und Endwert anpassen und diesen auf $\lceil (n-1)/d \rceil - 1$ und -1 respektive anpassen. Die zweite Schleife müssen wir ebenfalls auf 0 Indizierung anpassen.

Die Laufzeit von der ersten Schleife ist die Anzahl an Iterationen (die in $O(\frac{n}{d})$ ist, wie oben erklärt) und die Laufzeit von sink, die nach der vorherigen Teilaufgabe in $O(d \cdot \frac{\log(n)}{\log(d)})$ ist. Zusammen also in $O(\frac{n \log(n)}{\log(d)})$.

Die Laufzeit von heap_sort ist dann die Laufzeit von der ersten Schleife plus die Anzahl an Iterationen (das sind $O(n)$ viele) mal die Kosten von sink und von Swap. Swap hat nur eine Laufzeit von $O(1)$, sink eine Laufzeit von $O(d \cdot \frac{\log(n)}{\log(d)})$. Zusammen ergibt das eine Laufzeit in $O(\frac{n \log(n)}{\log(d)} + n \cdot d \cdot \frac{\log(n)}{\log(d)}) = O(\frac{n \cdot d \cdot \log(n)}{\log(d)})$.