Grundzüge von Algorithmen und Datenstrukturen, WS 15/16: Lösungshinweise zum 4. Übungsblatt

Markus Bläser

Aufgabe 4.1 (Lösungshinweise)

a) Wir durchlaufen a und kopieren zuerst alle Elemente mit dem Wert 0 in ein neues Array b. Dann durchlaufen wir a ein zweites Mal und kopieren alle Elemente mit dem Wert 1 nach b. Dabei bleibt die relative Reihenfolge unter Elementen mit dem gleichen Key erhalten.

b) Der Algorithmus steht schon in der Aufgabe: In einer For-Schleife zählen wir $i = 0, \ldots, b-1$ hoch und sortieren das Array a wie in a) mit dem Schlüsseln $\mathrm{Bit}(a[j], i)$, $1 \leq j \leq n$.

Für die Korrektheit zeigen wir per Induktion in i: Nach dem $i$ten Durchlauf ist das Array a korrekt sortiert, wenn man nur die unteren $i+1$ Bits betrachtet. Daraus folgt die Korrektheit für $i = b-1$.

Für $i = 0$ folgt dies aus Teil a).

Für den Induktionsschritt $i-1 \to i$ können wir annehmen, dass a korrekt sortiert ist, wenn man die unteren i Bits betrachtet. Nach dem nächsten Schleifendurchlauf stehen alle Elemente, deren $(i+1)$-tes Bit 1 ist rechts von denen mit $(i+1)$-ten Bit 0. Da das Verfahren aus a) stabil ist, bleibt die relative Reihenfolge in diesen beiden Gruppen erhalten und die Aussage folgt.

Die Laufzeit ist offentsichtlich $O(bn)$. (Anmerkung: Wenn alle Schlüssel verschieden sind, so ist $b \geq \log n$ und man hat "nichts gewonnen".)

c) Das geht erst einmal schief: Betrachte 10, 01.

Und dies zu reparieren, muss man die beiden Gruppen separat sortieren. Also erst mit a) in die zwei Gruppen gemäß dem höchsten Bit teilen und die beiden Gruppen separat rekursiv sortieren.

Aufgabe 4.2 (Lösungshinweise)

a) Wir bauen einen vollständigen binären Baum. An die Blätter schreiben wir die einzelnen Elemente. Wir vergleichen dann jeweils zwei Knoten mit gleichem Vater und schreiben das größere Element in den gemeinsamen Vater (K.o.-System). Das zweitgrößte Element muss mit dem größten verglichen worden sein, daher reicht es, das Maximum der Elemente zu finden, die mit dem größten Element verglichen wurden. Dies sind $m = \log(n-1)$ viele.

b) S heißt abwärts abgeschlossen, wenn gilt: Aus $x \in S$ und $x > y$ folgt stets $y \in S$. $\sum 2^{\#\text{Maxima in } S}$, wobei über alle S summiert wird, die genau $n-1$ Elemente enthalten und abwärts abgeschlossen sind, ist eine Potentialfunktion: Sei v ein beliebiger Konten und S eine Menge mit $n-1$ Elementen, die abwärts abgeschlossen ist. Seien x und y die Elemente, die an v verglichen werden. Sind beide Elemente in S oder keines, so ist S weiterhin abwärts abgeschlossen. Ist eines der Elemente in S, so ist S in einem der beiden möglichen Ausgängen abgeschlossen. Es folgt damit, dass obige Summe eine Potentialfunktion ist.

   Das Potential der Wurzel ist $n2^{n-1}$, das Potential jedes Blattes 2. Damit folgt die Schranke aus den Resultaten der Vorlesung.

c) Wir ersetzen das größte Element durch $-\infty$. Dann wandern wir den Pfad entlang zur Wurzel und führen dort lokal die Vergleiche neu aus. Auf diese Art und Weise erhalten wir einen Baum, dessen neues Maximum das zweitgrößte Element ist. Dies können wir wiederholen und für jeweils $\log(n-1)$ Vergleiche das nächstgrößte Element extrahieren.

d) Dann ist $t = n$ und damit die Laufzeit $O(n \log n)$.

Aufgabe 4.3 (Lösungshinweise)

a) Wir sortieren das Array. Dann gehen wir einmal linear durch das Array und schauen, ob zwei benachbarte Elemente gleich sind.

b) Ein Knoten in dem Comparison-Tree v hat nun drei Kinder $v_<$, $v_=$ und $v_>$, die den drei möglichen Ausgängen des Vergleiches entsprechen.

Die Eingaben $X_1, \ldots, X_n$ sind nicht mehr durch eine unbekannte Permutation geordnet (d.h. durch eine totale irreflexive Ordnungsrelation) sondern durch eine totale, transitive Relation. Dieser gruppiert Eingaben zusammen (Elemente, die gleich sind) und sortiert diese Gruppen mittels einer irreflexiven Ordnungsrelation.

Ebenso wird an den Knoten des Baumes keine irreflexive Ordnung mehr berechnet, sondern eine transitive Relation.

Die Behauptung ist nun, dass jeder Comparison-Tree, der feststellt ob die Eingabe ein Element doppelt enthält, $n!$ Blätter hat. Dazu betrachten wir alle Eingaben, bei denen alle Elemente verschieden sind. Bei diesen müssen benachbarte Elemente miteinander verglichen worden sein, sonst könnte man nicht wissen, ob diese verschieden sind. Damit hat man genug Information, um diese auch sortieren zu können. Damit folgt das Ergebnis aus dem Resultat aus der Vorlesung.