Grundzüge von Algorithmen und Datenstrukturen, WS 15/16: Lösungshinweise zum 5. Übungsblatt

Christian Hoffmann, Dr. Bodo Manthey

Aufgabe 5.1 a) Für die 3er-Gruppen ergibt sich eine Rekursionsgleichung $T(n) \leq T(\frac{n}{3}) + T(\frac{2n}{3}) + O(n)$. Da $\frac{1}{3} + \frac{2}{3} = 1$, funktioniert das Argument aus der Vorlesung nicht mehr und die lineare Laufzeit ist nicht mehr sicher. (Dies reicht zur Bearbeitung der Aufgabe.)

Als obere Schranke kann man $O(n \log n)$ zeigen. (Rekursionstiefe ist $O(\log n)$ und auf jeder Ebene ist die Arbeit jeweils $O(n)$.) Die tatsächliche Laufzeit ist unbekannt, eine anders lautende Behauptung im CLRS ist falsch.

b) Aufgrund der Verwirrung hier noch einmal der intendierte Algorithmus:

(i) Teile das Array in $n/3$ 3-Gruppen und bestimme die Mediane dieser Gruppen in jeweils konstanter Zeit.

(ii) Gruppiere die Mediane aus (i) wieder in $n/9$ 3-Gruppen und bestimme die Mediane dieser Gruppen in jeweils konstanter Zeit.

(iii) Bestimme rekursiv den Median a der Mediane aus Punkt (ii).

(iv) Rufe Partition mit a als Pivot auf und mache jetzt weiter wie im Algorithmus im Skript.

Laufzeit: Es gibt zwei rekursive Aufrufe. Der erste im dritten Punkt erfolgt auf einem Array der Größe $n/9$. Für den zweiten Aufruf muss man die Grösse der beiden Partitionen abschätzen. In Schritt (ii) sind $n/18$ der Median größer als a. Oberhalb dieser Mediane liegen nochmals $n/18$ Elemente. Also insgesamt $n/9$ Elemente. (Diese Abschätzungen sind nicht ganz exakt, die 3-Gruppe mit a muss eigentlich gesondert betrachtet werden. Asymptotisch macht dies jedoch keinen Unterschied.) Da diese $n/9$ Elemente alle Mediane von 3-Gruppen aus Schritt 1 sind, gibt es weitere $n/9$ Elemente, die größer sind, nämlich jeweils das Element aus den 3-Gruppen in (i), das größer als der Median ist. Also ist die Partition mit den Elementen kleiner a höchstens $n - 2n/9 = 7n/9$ groß. Auf die gleiche Art und Weise folgert man, dass es $2n/9$ Elemente geben muss, die kleiner als a sind und damit die andere Partition auch höchstens $7n/9$ groß ist. Damit ergibt sich für die Laufzeit

$$T(n) \leq T(n/9) + T(7n/9) + O(n).$$

Da $1/9 + 7/9 < 1$ ist, lässt sich die Analyse aus der Vorlesung anwenden und man erhält lineare Laufzeit.

Aufgabe 5.2 Dies ist eine Vorarbeit zur amortisierten Analyse!!!

Bei einer naiven Analyse ergibt sich das Problem, dass jede Multipop-Operation im Worst-Case $n + s$ kostet. Naiv käme man auf Kosten $n(n + s)$. Allerdings ist nach dem Entfernen von z.B. $n + s$ Elementen der Stack leer, es müssen erst wieder Elemente gepushed werden, bevor man neue Elemente entfernen kann.

Genauer gilt: Jedes Element, dass von Stack entfernt wird, muss entweder vorab schon auf dem Stack gewesen sein oder mittels einer Push-Operation hinzugefügt worden sein. Da insgesamt nur n Operationen ausgeführt werden, können insgesamt maximal $n + s$ Elemente jemals auf dem Stack gewesen sein. Also sind die Kosten *aller* Multipop-Operationen insgesamt $n + s$. Dazu kommen noch die Kosten für die Push-Operationen, die insgesamt maximal n sein können.

Aufgabe 5.3 Realisierung mit doppelt-verketterer Liste plus einen Pointer auf das letzte Element. Das erste Element der Liste ist der Repräsentant. Ein Element, das in keiner Liste ist, muss irgendwie markiert werden. (Array mit Flags, der Pointer auf das nächste Element zeigt "irgendwo" hin.)

Make-set: Erschaffen einer neuen Liste. Zeit $O(1)$. Es muss nicht abgetestet werden, ob x schon in einem Set ist.

Union: Beide Listen werden verkettet. Da es einen Pointer auf das letzte Element gibt, geht dies in Zeit $O(1)$.

Find: Man geht die Liste von x aus durch bis man das erste Element findet, dies gibt man zurück. Zeit is linear in der Länge der Liste, also $O(n)$.

Man kann Find beschleunigen, indem man einen Pointer von jedem Element auf den Repräsentanten hat. Dann geht Find in $O(1)$, aber bei Union müssen alle diese Pointer von einer der beiden Listen neu gesetzt werden. Dann hat Find aber Zeit $O(n)$.