Grundzüge von Algorithmen und Datenstrukturen, WS 15/16: Lösungshinweise zum 13. Übungsblatt

Christian Hoffmann, Fabian Bendun

Aufgabe 13.1 (a) Sei $j - i + 1 = n$ die Größe des zu sortierenden Bereichs des Arrays. Wir zeigen durch Induktion über n, dass Crazy-Sort $a[i..j]$ sortiert. **(1 Punkt)**

Als Induktionsanfang haben wir bei $n = 0$, also $i = j$, die Beobachtung, dass nichts zu tun ist und der Algorithmus auch nichts tut. Ähnliches gilt für $n = 1$, also $j = i + 1$: entweder ist schon $a[i] \leq a[i + 1]$ oder die ersten beiden Anweisungen stellen die richtige Reihenfolge dieser beiden Array-Einträge her. **(1 Punkt)**

Sei nun $n \geq 2$ und die Aussage gezeigt für $i - j + 1 < n$ (Induktionsvoraussetzung). Wir zeigen sie für $i - j + 1 = n$ (Induktionsbehauptung).

Nach Induktionsvoraussetzung *sortieren* die drei rekursiven Aufrufe den jeweiligen Teilbereich des Arrays. **(1 Punkt)**

Wir betrachten die drei Bereiche $A = a[i..i + k - 1]$, $B = [i + k..i + 2k - 1]$ und $C = [i + 2k..j]$. Als As bezeichnen wir solche Elemente von $a[i..j]$, die im Bereich A sein müssen, wenn $a[i..j]$ sortiert ist. Analog sind Bs und Cs definiert. Als ABs bezeichnen wir solche Elemente von $a[i..j]$, die im Bereich A oder im Bereich B stehen müssen, wenn $a[i..j]$ sortiert ist. **(1 Punkt)**

Wir zeigen nun: Nach dem ersten rekursiven Aufruf können im A-Bereich keine Cs stehen. Beweis: Es gibt insgesamt $\frac{2}{3}n$ ABs. Davon werden im ersten rekursiven Aufruf höchstens $\frac{1}{3}n$ nicht betrachtet. Also stehen nach dem ersten rekursiven Aufruf mindestens $\frac{1}{3}n$ Elemente links jedes Cs und füllen somit den kompletten A-Bereich. Also kann dort kein C-Element stehen. **(1 Punkt)**

Analog zeigt man: Nach dem ersten rekursiven Aufruf können im B-Bereich keine As stehen. **(1 Punkt)**

Damit wird im zweiten rekursiven Aufruf ein Bereich sortiert, der alle As enthält. Diese werden folglich in den A-Bereich einsortiert und füllen ihn komplett und nach Induktionsvoraussetzung in der richtigen Reihenfolge aus. **(1 Punkt)**

Daher stehen in dem Bereich, der im dritten rekursiven Aufruf sortiert wird, nur Bs und Cs, die dann nach Induktionsvoraussetzung an die richtigen Plätze befördert werden.

Die Bepunktung ist eine Orientierungshilfe, für was Punkte vergeben werden können. Insgesamt gibt es aber höchstens 5 Punkte. Für volle Punktzahl muss insgesamt schlüssig und lückenlos argumentiert werden, die Details dürfen von dieser Musterlösung abweichen.

(b) $T(n)$ sei die maximale Laufzeit von Crazy-Sort(a, i, j), wenn $j - i + 1 \leq n$. Wenn $n = 0$ oder $n = 1$, wird nur eine feste Anzahl von Schritten ausgeführt. Also ist $T(0), T(1) \in O(1)$. **(1 Punkt)**

Die Größe des dem ersten rekursiven Aufruf übergebenen Teilbereichs ist $(j - k) - i + 1 = n - \lfloor \frac{n}{3} \rfloor = \lceil \frac{2}{3} n \rceil$. Ähnlich zeigt man auch, dass die Größen der den beiden anderen rekursiven Aufrufen übergebenen Array-Teilbereiche $\leq \lceil \frac{2}{3} n \rceil$ sind und dass die beiden zu tauschenden Teilbereiche $\leq \lceil \frac{1}{3} n \rceil$ Elemente umfassen. **(1 Punkt)** Also gilt

$$T(n) = 3T\Big(\Big\lceil \frac{2}{3} n \Big\rceil\Big) + O(n).$$

(1 Punkt)

Wir wenden nun das Master-Theorem an. Wir haben $b = \frac{3}{2}$ und $a = 3$. **(1 Punkt)** Es ist $\log_b a = 2.70951\ldots$ und $O(n) \subseteq O(n^{\log_b a - 1})$. Also gilt $T(n) \in O(n^{\log_b a}) \subseteq O(n^{2.71})$. **(1 Punkt)**

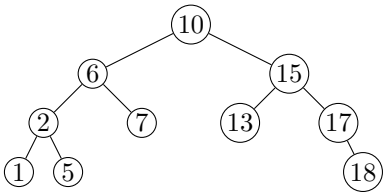

Abbildung 1: Original

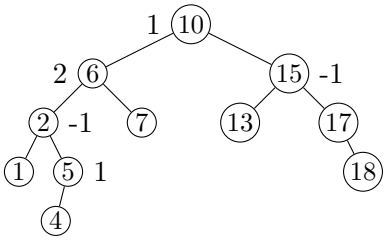

Abbildung 2: After insert 4

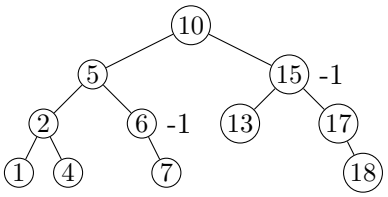

Abbildung 3: After double rotation

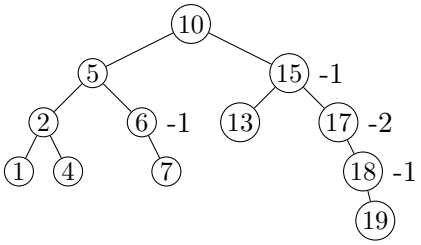

Abbildung 4: Nach einfügen 19

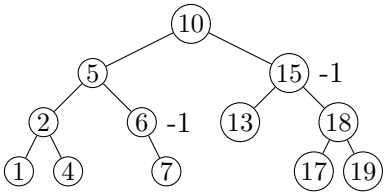

Abbildung 5: Nach rotation

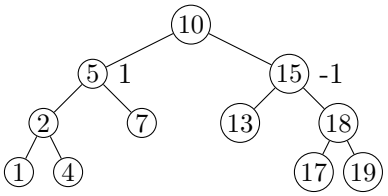

Abbildung 6: Nach löschen von 6

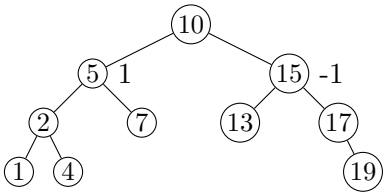

Abbildung 7: Nach löschen von 18

Aufgabe 13.2

Aufgabe 13.3 a) Entfernt man von einem Binomial-Baum mit 2^i Knoten die Wurzel, so zerfällt dieser in i Binomial-Bäume mit jeweils $1, 2, 4, \ldots, 2^{i-1}$ Elementen. **(1 Punkt)** Ist T heap-geordnet, so ist sind auch diese Bäume heap-geordnet. **(1 Punkt)** Falls $m = 2^i$, so ist nichts zu tun. Sonst sei $m = \sum_{j=1}^{i-1} b_j 2^j$ die Binärdarstellung von m. Die eine Menge besteht aus den Bäumen mit $b_i = 1$, die andere aus den übrigen plus die entfernte Wurzel. **(1 Punkt)**. Nur der Baum mit einem Knoten kommt zweimal vor. **(1 Punkt)**.

b) Der Algorithmus ergibt sich aus der Aufteilung aus a). Dies sind fast schon Binomialheaps. **(1 Punkt)** Falls in einer Menge zweimal der Baum mit einem Knoten vorkommt, so kann dies mit BH-union bereinigt werden **(1 Punkt)**. Für die Aufteilung werden $i = \log n$ viele Bäume je einmal betrachtet. BH-Union braucht ebenfalls $O(\log n)$ Zeit **(1 Punkt)**.

c) Wir gehen die Bäume im Heap der Reihe nach durch und summieren die Anzahl der Knoten auf, bis wir mehr als r haben. Sei m die erreichte Anzahl und 2^i die Anzahl der Knoten im letzten Baum T. T teilen wir in zwei Heaps B_1 und B_2 mit $s := r - (m - 2^i)$ bzw. $2^i - s$ Knoten auf. Die Bäume vor T vereinigen wir mit B_1, die danach mit B_2. **(3 Punkte)** Insgesamt sind $\log n$ Bäume im Heap. Das Aufteilen von T braucht $O(\log n)$ Zeit nach b), das Vereinigen ebenfalls. **(1 Punkt)**

Aufgabe 13.4 a) Ein optimales Schedule von $I_1, \ldots, I_j$ verwendet entweder I_j oder nicht. **(1 Punkt)** Dann ist $\mathrm{Opt}(j) = \mathrm{Opt}(j-1)$. **(1 Punkt)** Oder es verwendet I_j. Dann ist dieses Schedule ohne I_j ein optimales Schedule von $I_1, \ldots, I_{p(j)}$. **(1 Punkt)** Also gilt $\mathrm{Opt}(j) = \mathrm{Opt}(p(j)) + w_j$. **(1 Punkt)** Insgesamt gilt

$$\mathrm{Opt}(j) = \max\{\mathrm{Opt}(j-1), \mathrm{Opt}(p(j)) + w_j\}.$$

b) Eine For-Schleife. **(2 Punkte)** Korrektheit unmittelbar aus b) **(1 Punkt)**, Laufzeit ist $O(n)$ **(1 Punkt)**

c) Setze ein Flag $a[j] = 1$, falls $\mathrm{Opt}(j) = \mathrm{Opt}(p(j)) + w_j$. **(1 Punkt)** Gehe nun durch a rückwärts. Falls $a[j] = 0$, gehe zu $a[j-1]$. **(1 Punkt)** Falls $a[j] = 1$, gebe j aus und mache mit $a[p(j)]$ weiter. **(1 Punkt)**

Aufgabe 13.5 (a) Sei T ein minimaler Spannbaum von G. Wir nehmen an, dass alle Kanten minimalen Gewichts, die V_1 und V_2 verbinden, nicht zu T gehören und konstruieren einen Spannbaum T' mit kleinerem Gewicht als T, was ein Widerspruch ist. Sei dazu $e = \{x, y\}$ eine Kante minimalen Gewichts unter denen, die V_1 und V_2

verbinden, ohne Einschränkung sei $x \in V_1$ und $y \in V_2$. G' sei G eingeschränkt auf die Kanten aus T. Da G' zusammenhängend ist, gibt es dort einen Weg von x nach y. Mindestens eine Kante, sagen wir $f = \{a, b\}$, dieses Weges verbindet V_1 mit V_2. Wir behaupten, dass $T' = (T \setminus \{f\}) \cup \{e\}$ ebenfalls ein Spannbaum von G ist. Dann ist dies der Widerspruch: $w(T') = w(T) - w(f) + w(e) < w(T)$ (da $w(f) > w(e)$, weil e minimales Gewicht unter den V_1 und V_2 verbindenden Kanten hat, aber keine solche minimale Kante in T ist, aber $f \in T$).

Zu zeigen bleibt also, dass T' ein Spannbaum von G ist. Dazu zeigen wir, dass a und b in T' verbunden sind. Damit kann dann jeder Weg in T, der f benutzt, durch einen Weg in T' ersetzt werden.

Sei $xv_1 \dots v_\ell abu_1 \dots u_r y$ der Weg von x nach y in T. Dann ist $v_\ell \dots v_1 xyu_r \dots u_1 b$ ein Weg von a nach b in T'.

(3 Punkte)

(b) Wir betrachten den Kreis mit vier Knoten und vier Kanten e_1, e_2, e_3, e_4, die alle Gewicht 1 haben. Dann ist $T = \{e_1, e_2, e_3\}$ ein minimaler Spannbaum. Wenn wir das Gewicht von e_1 auf 2 ändern, ist T nicht mehr minimal, weil $\{e_2, e_3, e_4\}$ kleineres Gewicht hat. **(1 Punkt)**

(c) Hier funktioniert jedes Beispiel, in dem e eine Schnittkante ist, also eine Kante, so dass G ohne e nicht mehr zusammenhängend ist. Dann muss jeder Spannbaum von G, e enthalten. Beispiel: Graph, der nur aus einer Kante e und ihren Endknoten besteht. **(1 Punkt)**

(d) Seien V_1 und V_2 die zwei Komponenten, in die G zerfällt, wenn wir auf Kanten von T einschränken und e entfernen. Nach Aufgabe (a) bleibt T bei Gewichtsänderung von e ein minimaler Spannbaum dann und nur dann, wenn e unter den von V_1 nach V_2 führenden Kanten eine Kante minimalen Gewichts bleibt. („$\Rightarrow$“: Wenn T minimaler Spannbaum bleibt, muss e, da es die einzige V_1-V_2-Kante in T ist, nach (a) minimales Gewicht unter den V_1-V_2-Kanten haben. – „$\Leftarrow$“: Sei T' ein Spannbaum. Wenn er e enthält, gilt sowohl vor als auch nach der Gewichtsänderung $w(T') \geq w(T)$. Wenn er e nicht enthält, enthält er eine andere V_1-V_2-Kante f. Es gilt $w(T' \setminus \{f\}) \geq w(T \setminus \{e\})$, weil T minimaler Spannbaum vor der Gewichtsänderung war. Also gilt nach der Gewichtsänderung, da dann immer noch $w(f) \geq w(e)$: $w(T') = w(T' \setminus \{f\}) + w(f) \geq w(T \setminus \{e\}) + w(e)$.) **(2 Punkte)** Wenn es keine Kante außer e gibt, die V_1 mit V_2 verbindet, bleibt T bei jeder Gewichtsänderung von e ein minimaler Spannbaum. **(1 Punkt)** Ansonsten gilt: T ist kein minimaler Spannbaum mehr genau dann, wenn das Gewicht von e um mehr als

$$\min_{\substack{f \in E,\ f \neq e \\ f \text{ verbindet } V_1 \text{ und } V_2}} w(f) - w(e)$$

erhöht wird. **(1 Punkt)** Diesen Wert bestimmt der folgende Algorithmus.

(3 Punkte für den Algorithmus)

1: Bestimme mit Breitensuche die Komponenten V_1, V_2, in die G zerfällt, wenn die Kantenmenge auf die Kanten von T ohne e eingeschränkt wird
2: $\Delta := \infty$
3: **for** $f \in E$ **do**
4: **if** $f \neq e \wedge f$ verbindet V_1 und $V_2 \wedge w(f) - w(e) < \Delta$ **then**
5: $\Delta := w(f) - w(e)$
6: **if** $\Delta = \infty$ **then return** „T bleibt minimaler Spannbaum bei jeder Gewichtsänderung von e“
7: **elsereturn** „T nicht mehr minimaler Spannbaum, sobald $w'(e) > w(e) + \Delta$“