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

universitätbonn

Algorithmen und
Berechnungskomplexität I
Wintersemester 2010/11

Musterlösung Probeklausur

Aufgabe 1

(a) Gemäß der unteren Schranke aus der Vorlesung haben vergleichsbasierte Sortierverfahren eine Laufzeit von $\Omega(n \log n)$. Prof. G. Witzt muss sich also irren, denn $n \log \log n = o(n \log n)$. Das heißt, es gibt keine Funktion f mit $f = O(n \log \log n)$ und $f = \Omega(n \log n)$.

(b) $f(n) = \ln(n)$

(c) Die Sequenzen (2) und (3) können nicht bei der Suche nach dem Schlüssel 220 entstehen. (Bei beiden ist der Grund, dass wir bei der Suche irgendwann Schlüssel 127 sehen und daher im rechten Teilbaum nach 220 weitersuchen müssen. Im rechten Teilbaum kann aber nicht Schlüssel 108 stehen.)

(d) Die starken Zusammenhangskomponenten des Graphen sind $\{1, 3, 5, 10\}$, $\{2, 6, 7, 9\}$ und $\{4, 8\}$.

(e) Ist der Graph zusammenhängend (das ist der interessante Fall), dann ist die worst-case Laufzeit des Algorithmus von Kruskal gleich $\Theta(m \log m)$. Diese Zeit ergibt sich unter anderem durch das Sortieren der m Kanten nach ihrem Gewicht. Ohne einen Sortieralgorithmus mit worst-case Laufzeit $o(m \log m)$ kann man die worst-case Laufzeit des Algorithmus von Kruskal nicht dadurch verbessern, dass man eine effizientere *Union-Find*-Datenstruktur verwendet.

(Ist der Graph nicht zusammenhängend, dann ist die worst-case Laufzeit des Algorithmus von Kruskal gleich $O(m+n)$. In diesem Fall wird die *Union-Find*-Datenstruktur gar nicht benötigt und hat keinen Einfluss auf die Laufzeit.)

Aufgabe 2

(a) Wird `Swap`(i) aufgerufen und es kommt zu einer Vertauschung, dann reduziert sich die Anzahl der Inversionen um genau 1, da genau die Inversion $(i, i+1)$ behoben wird. Alle anderen Inversionen bleiben unverändert und es entstehen auch keine neuen. Da ein Feld A genau dann sortiert ist, wenn es keine Inversionen besitzt, müssen genau $\chi(A)$ Inversionen aufgehoben, also genau $\chi(A)$ Vertauschungen durchgeführt werden.

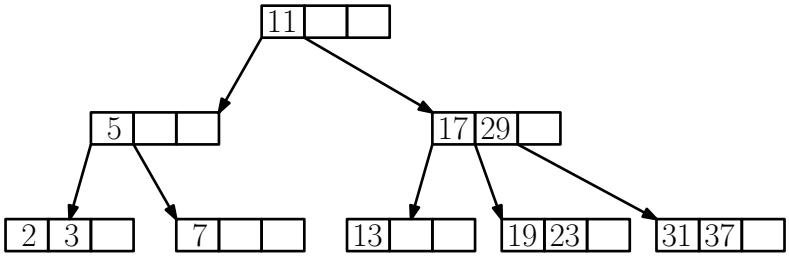
Die Zahl der Inversionen eines absteigend sortierten Feldes ist $\binom{n}{2}$. Ein Sortierverfahren, das nur mit Hilfe der `Swap`-Funktion Änderungen im Feld durchführt, muss in diesem Fall also $\Theta(n^2)$ Vertauschungen durchführen. Da nicht bekannt ist, wie das Verfahren genau implementiert ist, können wir keine obere Schranke für die Laufzeit angeben. Aufgrund der Anzahl der Vertauschungen ist aber $\Omega(n^2)$ eine untere Schranke für die worst-case Laufzeit.

(b) Da wir wissen, dass alle Einträge von A Elemente aus $\{1, \dots, n\}$ sind, können wir ein weiteres Feld B der Größe n anlegen, in dem wir zählen, wie oft eine Zahl $i \in \{1, \dots, n\}$ in A vorkommt (diese Anzahl sei $B[i]$). Das Zählen der Vorkommen der Zahlen i ist insgesamt in Zeit $O(n)$ möglich. Nun gehen wir noch einmal durch das Feld und geben $B[i]$-mal die Zahl i aus, $i = 1, \dots, n$.

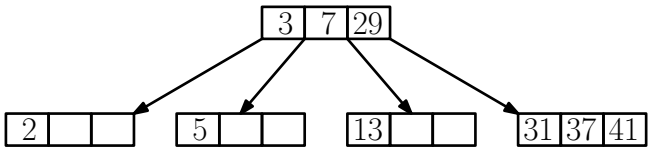
Aufgabe 3

(a) Die Wurzel eines B-Baumes der Ordnung $t \geq 2$ und der Höhe $h \geq 1$ besitzt mindestens 2 Söhne. Alle anderen Knoten besitzen mindestens t Söhne. Die Mindestanzahl an Knoten eines solchen B-Baumes ist daher $1 + 2\sum_{i=0}^{h-1} t^i$.

(b) Durch das Einfügen von Knoten 19 läuft der Knoten mit den Schlüsseln $13, 17, 23$ über. Dieser wird (vor dem Einfügen von 19) gesplittet und die 17 muss in den Vaterknoten mit den Schlüsseln $5, 11, 29$ eingefügt werden. Dieser läuft dadurch auch über und muss ebenfalls gesplittet werden, wodurch die neue Wurzel 11 als einzigen Schlüssel erhält.

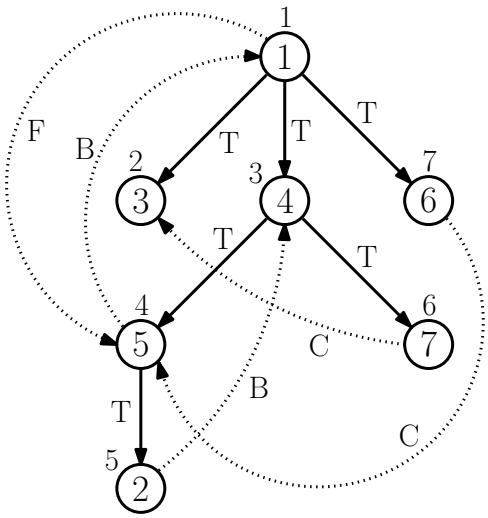


(c) Um Schlüssel 11 zu löschen, ersetzen wir ihn durch seinen symmetrischen Vorgänger 7 und löschen den symmetrischen Vorgänger im Blattknoten. Dadurch gibt es einen Unterlauf in diesem Blattknoten. Da der linke Nachbar des Blattknotens nicht minimal viele Schlüssel besitzt, rotieren wir von diesem Nachbarn in den Blattknoten mit dem Unterlauf, d.h. die 5 wird in den Blattknoten rotiert und die 3 aus dem Nachbarknoten wird an die alte Stelle der 5 rotiert.

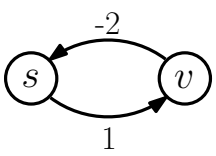


Aufgabe 4

(a) Bei der Tiefensuche ergibt sich folgender Tiefensuchbaum.



(b) Das einfachste Gegenbeispiel verwendet negative Kreise. Liegt s auf einem negativen Kreis, dann ist per Definition die Länge eines kürzesten Weges von s zu jedem Knoten auf diesem Kreis gleich $-\infty$. Dijkstra weist jedoch all diesen Knoten einen endlichen Wert zu. In dem unten abgebildeten Graphen berechnet der Algorithmus von Dijkstra als Distanzen $d[s] = -1$ und $d[v] = 1$.



Der Algorithmus von Dijkstra arbeitet aber auch dann nicht immer korrekt, wenn es keine negativen Kreise gibt. In folgendem Graphen berechnet der Algorithmus von Dijkstra die Distanzen $d[s] = 0$, $d[v] = 0$, $d[u] = 2$ und $d[w] = 2$. Die Länge eines kürzesten Weges von s nach w ist allerdings 1.