RWTH Aachen **SoSe 2007**
Lehrgebiet Theoretische Informatik
Kneis—Mölle—Richter—Rossmanith 09.10.2007

Klausur zur Vorlesung Datenstrukturen und Algorithmen

Aufgabe 1 (5+5 Punkte)

a) Beweisen oder widerlegen Sie: Der Mergesort-Algorithmus aus der Vorlesung arbeitet *in-place*, benötigt also neben dem Eingabe-Array nur eine feste Menge zusätzlicher Variablen und Array-Elemente.

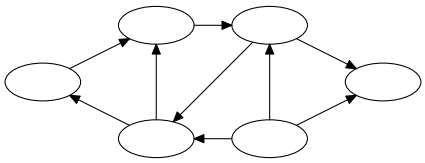
b) Beweisen oder widerlegen Sie: Es gibt ein Sortierverfahren, bei dem Schlüssel allein durch Vertauschungen benachbarter Array-Elemente bewegt werden und das auf n-elementigen Arrays im Durchschnitt nur Zeit $O(n \log n)$ benötigt. Wir wollen hierbei annehmen, daß alle Array-Elemente verschieden sind und jede Permutation mit gleicher Wahrscheinlichkeit vorkommt.

Lösungsvorschlag:
Der Mergesort-Algorithmus aus der Vorlesung (Folie 238) benötigt ein zweites n-elementiges Array, um die rekursiv sortierten Teile des ersten Arrays in der richtigen Reihenfolge zusammenzuführen. Er arbeitet also nicht *in-place*.

Für den zweiten Aufgabenteil wollen wir davon ausgehen, daß das n-elementige Eingabe-Array a aufsteigend sortiert werden soll. Wir definieren eine *Inversion* als ein Paar (i, j) mit $1 \leq i < j \leq n$ und $a[i] > a[j]$, also als ein Paar von Indices zweier genau in falscher Reihenfolge befindlicher Elemente.
Es ist offensichtlich, daß eine Vertauschung benachbarter Array-Elemente höchstens eine Inversion aufheben kann. Allerdings gibt es $n(n-1)/2$ viele Paare (i, j) mit $1 \leq i < j \leq n$, und bei Gleichverteilung beträgt die Wahrscheinlichkeit, daß ein solches Paar eine Inversion darstellt, genau $\frac{1}{2}$. Mit der Linearität des Erwartungswerts folgt nun, daß das Eingabe-Array im Durchschnitt $n(n-1)/4$ Inversionen hat. Offensichtlich liegt die Laufzeit somit nicht in $O(n \log n)$.
Siehe hierzu auch Folie 228.

Aufgabe 2 (10 Punkte)

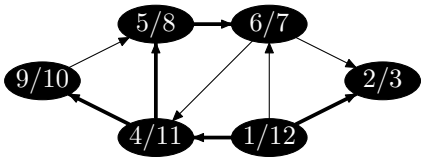
Gegeben sei der hier abgebildete Graph G.



(a) Führen Sie auf G eine Tiefensuche durch. Ermitteln Sie hierbei die Anfangs- und Endzeiten sowie die Kantentypen. (b) Wie kann man mit Tiefensuche herausfinden, ob ein gerichteter Graph kreisfrei ist? (c) Geben Sie die starken Zusammenhangskomponenten von G an (ohne Begründung)!

Lösungsvorschlag:

Die Anfangs- und Endzeiten hängen natürlich von der Reihenfolge ab, in der wir bei der Tiefensuche die Anfangsknoten und die Nachfolger wählen. Ein mögliches Resultat wäre folgendes:



Im folgenden identifizieren wir die Knoten anhand ihrer *discovery time*. Für den obigen DFS-Baum ergibt sich:

- Die Kanten $(1, 2)$, $(2, 3)$, $(3, 4)$ und $(3, 6)$ sind Baumkanten.
- Es gibt keine Vorwärtskanten.
- Die Kanten $(4, 1)$ und $(4, 2)$ sind Rückwärtskanten.
- Alle von 11 ausgehenden Kanten sind Querkanten.

Für den zweiten Aufgabenteil erinnern wir uns an einen Satz aus der Vorlesung, der da besagt: „Gegeben sei ein gerichteter Graph G. Dann können wir in linearer Zeit feststellen, ob G azyklisch ist (keine Kreise enthält).“ Im Beweis dieses Satzes haben wir gesehen, daß ein gerichteter Graph genau dann azyklisch ist, wenn die Tiefensuche keine Rückwärtskanten liefert.

Der dritte Aufgabenteil läßt sich durch Hinsehen lösen. Die Knoten 1 bis 4 bilden eine starke Zusammenhangskomponente. Die Knoten 6 und 11 hingegen hängen nicht stark zusammen, bilden also zwei jeweils einelementige starke Zusammenhangskomponenten.

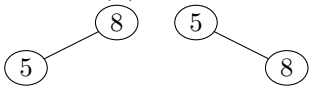
Aufgabe 3 (10 Punkte)

Fügen Sie die Elemente $8, 5, 12, 7, 10$ in dieser Reihenfolge in einen anfangs leeren Splay-Baum ein und notieren Sie dabei jeden Zwischenschritt. Was geschieht, wenn wir abschließend nach der 11 suchen?

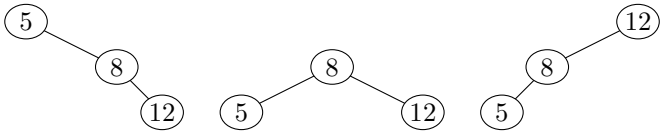
Lösungsvorschlag:

`insert`(8):

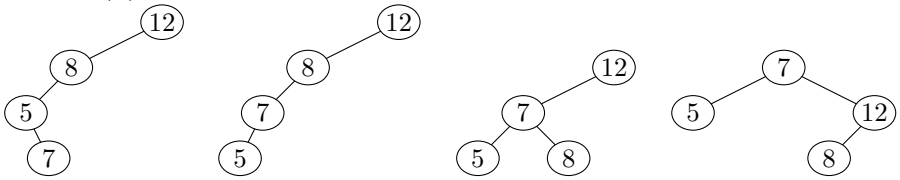
`insert`(5):



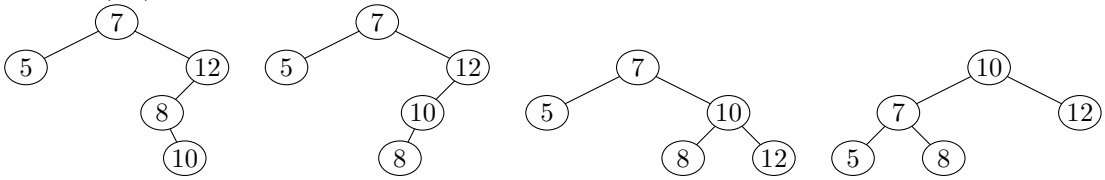
`insert`(12):



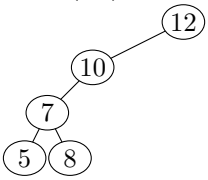
`insert`(7):



`insert`(10):



`find`(11):



Die Suche nach 11 ist zwar erfolglos, bedingt aber dennoch eine Splay-Operation auf 12 (ein Zag).

Aufgabe 4 (10 Punkte)

a) Beweisen Sie das folgende Theorem aus der Vorlesung:
Ein AVL-Baum der Höhe h besitzt zwischen F_h und $2^h - 1$ viele Knoten.

b) Beweisen Sie mit (a) das folgende Theorem aus der Vorlesung:
Die Höhe eines AVL-Baums mit n Knoten ist $\Theta(\log n)$.

Zur Erinnerung: F_h sind die Fibonacci-Zahlen mit $F_0 = 0$, $F_1 = 1$ und $F_h = F_{h-1} + F_{h-2}$ für $h \geq 2$. Es gilt $F_h = \Theta(\phi^h)$ mit $\phi = (1 + \sqrt{5})/2$.

Lösungsvorschlag:
Wir bedienen uns der Beweise aus der Vorlesung. Zuerst Teil (a):
Ein vollständiger Binärbaum der Höhe h hat offensichtlich genau $2^h - 1$ viele interne Knoten – und ist ein AVL-Baum.
Für den Beweis der unteren Schranke verwenden wir Induktion über h. Der Anfang ist leicht: Ein Baum der Höhe null hat keinen internen Knoten, und es gilt $F_0 = 0$. Ein Baum der Höhe eins hat genau einen internen Knoten, und es gilt $F_1 = 1$.
Falls $h > 1$, dann hat der linke oder rechte Unterbaum Höhe $h-1$ und damit mindestens F_{h-1} viele interne Knoten. Der andere Unterbaum hat mindestens Höhe $h-2$ und damit mindestens F_{h-2} viele interne Knoten.
Insgesamt macht das mindestens $F_{h-1} + F_{h-2} = F_h$ viele Knoten.

Die Aussage aus (b) ist jetzt relativ einfach zu beweisen:
Wir wissen bereits, daß $F_h \leq n \leq 2^h$ gilt. Wenn wir den Logarithmus bilden, ergibt sich

$$\log F_h \leq \log n \leq h$$

und aus der zweiten Ungleichung sofort $h = \Omega(\log n)$. Mit $F_h = \Theta(\phi^h)$ gilt aber

$$\log F_h = \log \Theta(\phi^h) = h \cdot \log \phi + O(1)$$

und deshalb auch

$$\log n \geq \log F_h = h \cdot \log \phi + O(1) = \Theta(h).$$

Letzteres impliziert natürlich

$$h = O(\log n).$$