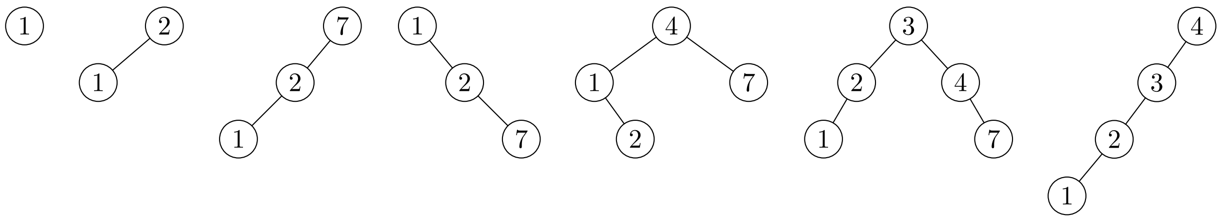
RWTH Aachen **SoSe 2007**
Lehrgebiet Theoretische Informatik Blatt 5
Kneis—Mölle—Richter—Rossmanith—Thiemann 7.5.2007

Übungen zur Vorlesung Datenstrukturen und Algorithmen

T10

Was passiert, wenn in einen anfangs leeren Splay-Baum die Elemente 1, 2, 7, 1, 4, 3 eingefügt werden und anschließend die 7 wieder gelöscht wird?

Lösungsvorschlag:
Die aus den einzelnen Operationen resultierenden Splay-Bäume sehen wie folgt aus:



T11

Eine Datenstruktur für ein assoziatives Array, das besonders schnelles Einfügen erlauben soll, sehe so aus: Es gibt einen AVL-Baum *und* eine verkettete Liste. Soll ein Element in diese Datenstruktur eingefügt werden, wird es einfach in konstanter Zeit an den Anfang der Liste gehängt. Soll hingegen ein Element gesucht oder gelöscht werden, so werden zunächst alle Elemente aus der Liste in den AVL-Baum eingefügt und gleichzeitig aus der Liste entfernt. Danach wird die Such- oder Löschoperation im AVL-Baum vorgenommen.

Geben Sie eine möglichst gute obere Laufzeitschranke für das gemischte Ausführen n beliebiger Operationen bei anfangs leerer Datenstruktur an! Verwenden Sie hierzu sowohl die Methode der amortisierten Analyse als auch ein einfacheres Argument.

Lösungsvorschlag:
Man nehme die Potentialfunktion $c \cdot l \log n$, wobei l die Länge der Liste, n die Gesamtzahl der Elemente in der Datenstruktur und $c > 0$ eine geeignete Konstante ist, die wir später festlegen.
Die amortisierten Kosten des Einfügens sind höchstens

$$O(1) + c(l+1)\log(n+1) - cl\log n = O(\log n),$$

wobei $O(1)$ die tatsächlichen Kosten und der Rest die Veränderung der Potentialfunktion darstellen.
Für das Suchen ergibt sich $O(l \log n) + O(\log n) - cl \log n \leq 0$, wenn wir c entsprechend groß wählen. Dabei sind $O(l \log n)$ die Kosten für das Einfügen der Listenelemente in den AVL-Baum und $O(\log n)$ die Kosten für das Suchen im AVL-Baum. Die Potentialfunktion

sinkt auf 0, da die Liste anschließend nichts mehr enthält. Ganz ähnlich verhält es sich beim Löschen.

Die amortisierten Kosten sind also für jede Operation nur $O(\log n)$.

Ein anderes Argument ist folgendes: Jedes Einfügen ist in Wirklichkeit ein Einfügen in den AVL-Baum, findet aber zu einem anderen Zeitpunkt statt. Das macht für die Summe der Laufzeiten natürlich keinen Unterschied.

T12

Es bezeichnen $n\downarrow$, $n?$ und $n\uparrow$ das Einfügen, Suchen bzw. Löschen des Elements n in eine(r) Datenstruktur. Wir definieren die Operationenfolge F_n rekursiv als $F_0 = \varepsilon$, $F_1 = 1\downarrow\ 1\uparrow$ und

$$F_n = n\downarrow\ F_{n-1}\ n?\ F_{n-1}\ n\uparrow.$$

Welche der Ihnen bekannten Datenstrukturen ist auf derartigen Operationenfolgen besonders effizient? Wie verhalten sich hier binäre Suchbäume und AVL-Bäume?

Lösungsvorschlag:

Die Anzahl f_n der Operationen in F_n beträgt für $n \geq 1$ genau $f_n = 2^n + 3(2^{n-1} - 1)$. Dies kann leicht durch Induktion bewiesen werden:

$$f_n = 2f_n + 3 = 2(2^{n-1} + 3(2^{n-2} - 1)) + 3 = 2^n + 3(2^{n-1} - 1).$$

Untypischerweise ist bei der Operationenfolge F_n der Stack besonders effizient. Da immer nur das Element gesucht oder gelöscht wird, das ganz oben liegt, benötigt jede einzelne Operation nur $O(1)$ Zeit. Der Gesamtaufwand ist daher in $O(2^n)$ und somit linear in der Anzahl der Operationen.

Ein binärer Suchbaum entartet direkt am Anfang, nämlich beim initialen Einfügen der Elemente $n, n-1, \ldots, 1$. Obwohl diese Kette durch die fortwährende Manipulation immer wieder verkürzt und verlängert wird, bleibt der Baum zu jedem Zeitpunkt eine Liste. Man sieht nun leicht, daß die Hälfte der Einfügeoperationen das Element 1 betreffen und somit Aufwand n verursachen: Das Element i wird grundsätzlich mindestens doppelt so oft wie das Element $i+1$ eingefügt. Das gleiche gilt für die Löschoperationen. Andererseits machen die Einfüge- und Löschoperationen mehr als die Hälfte aller Operationen in F_n aus. Damit haben wir eine untere Schranke von $f_n/2 \cdot n/2 = \Omega(2^n \cdot n)$ für den Gesamtaufwand.

Ein AVL-Baum hingegen hat garantiert logarithmische Tiefe. Da die 1 als kleinstes Element in einem Suchbaum kein linkes Kind besitzt, andererseits aber die AVL-Eigenschaft erfüllen muß, ist sie maximal zwei Schritte von einem Blatt entfernt. Weil die 1 aufgrund der Operationenreihenfolge nur dann vorkommt, wenn sich auch alle Elemente von 2 bis n im Baum befinden, und weil die Blätter in einem AVL-Baum mit n Elementen die Tiefe $\Omega(\log n)$ haben, gilt dies somit auch für die 1. Deshalb benötigen mindestens die Hälfte aller Operationen in F_n Zeit $\Omega(\log n)$. Der Gesamtaufwand ist daher mindestens $f_n/2 \cdot \Omega(\log n) = \Omega(2^n \cdot \log n)$.

H10 (10 Punkte)

Geben Sie eine Datenstruktur an, die folgende Eigenschaften erfüllt, und beweisen Sie dies. Wie gewöhnlich sollen die Operationen Einfügen, Löschen und Suchen unterstützt werden. Dabei soll das Löschen lediglich konstante Zeit erfordern, während die Gesamtzeit für n beliebige Operationen durch $O(n \log n)$ beschränkt sein muß.

Lösungsvorschlag:

Analog zu Aufgabe T11 verwenden wir eine Liste und einen AVL-Baum. Einfügungen und Suchen werden dabei stets im AVL-Baum durchgeführt. Zu löschende Elemente werden jedoch zunächst nur vorne in die Liste eingefügt; ihre tatsächliche Löschung im AVL-Baum findet erst statt, wenn wieder eine Einfüge- oder Suchoperation auftritt.

H11 (10 Punkte)

Ein Mensch ohne Zufallsgenerator schlägt als Datenstruktur einen modifizierten Treap vor, bei dem die Prioritäten nicht zufällig gewählt werden, sondern mit der Zeit sinkende Werte sind (z.B. $-t$ bei der t-ten Operation). Zeigen Sie, daß sich diese Datenstruktur vergleichsweise schlecht verhalten kann.

Lösungsvorschlag:

Wenn wir die Zahlen $1, 2, \ldots, n$ in dieser Reihenfolge einfügen, ist der resultierende Treap eine nach unten links verlaufende Liste der Länge n und somit entartet.

Noch schöner wäre es, hier eine Reihenfolge zu finden, bei der im t-ten Schritt sogar t Rotationen erfolgen (mit Beweis sollte das Bonuspunkte geben). Dann läge schon der Zeitaufwand für die Konstruktion bei $\Omega(n^2)$. Für die oben genannte Reihenfolge gilt dies jedoch nicht, da pro Schritt nur eine Rotation auftritt.