Prof. Andreas Goerdt
Frank Schädlich

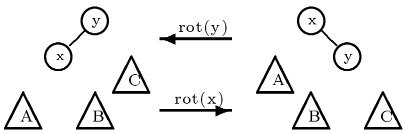
TU Chemnitz
Sommersemester 2003

Theoretische Informatik III
5. Übung - Lösungsvorschläge

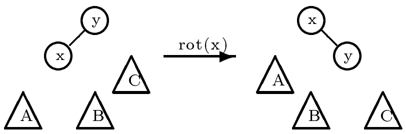
1. Aufgabe

Grundlegende Operationen:

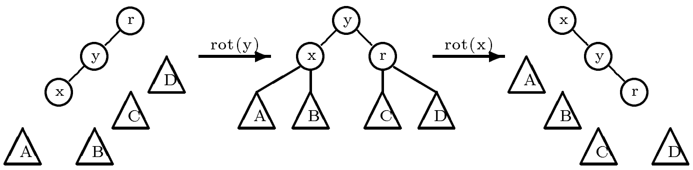
Rotation um x bzw. y



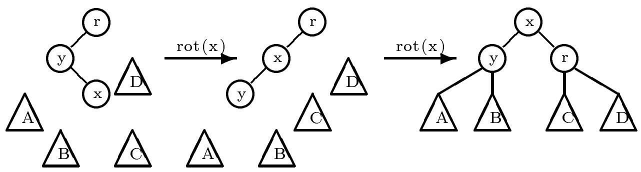
`Splay`(x)
Knoten x ist *Sohn der Wurzel*. `Splay` entspricht einfacher Rotation.



Zig-Zig-Fall: Knoten x ist linker (rechter) Sohn eines linken (rechten) Sohnes.



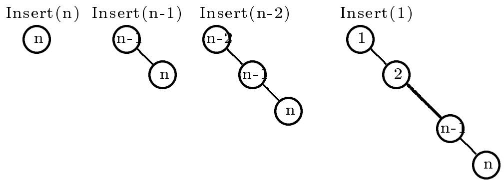
Zig-Zag-Fall: Knoten x ist linker (rechter) Sohn eines rechten (linken) Sohnes.



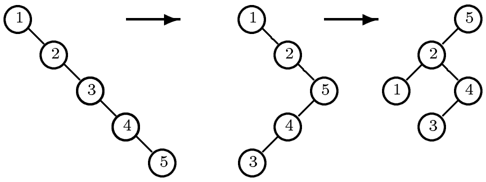
Andere Richtung analog. `Splay` wird solange rekursiv wiederholt, bis x Wurzel des

Baumes ist.

Da immer ein minimales Element eingefügt wird, entsteht durch `Insert`$(n), \ldots,$ `Insert`(1) eine Liste. Dies liegt daran, daß die Suche nach dem einzufügendem Element bereits bei der Wurzel abbricht, da alle Elemente im Baum größer sind und somit der linke Teilbaum leer ist, wenn das vorhergehende Element auch auf diese Weise eingefügt wurde. Jedes `Insert` hat einen Aufwand $\mathrm{O}(1)$.



Ein `Find`(n) führt wegen `Splay` zu einer Kompaktierung des Baumes. `Find`(n) hat hier den Aufwand $\mathrm{O}(n)$. Da jedes der n `Inserts` ein Aufwand von $\mathrm{O}(1)$ hat, ergibt sich ein Gesamtaufwand von $\mathrm{O}(n)$ für die Folge `Insert`$(n), \ldots,$ `Insert`(1), `Find`(n) von $\mathrm{O}(n)$, was offensichtlich in $\mathrm{O}(n \log n)$ ist. Bemerkenswert ist, daß (bei diesem Beispiel) trotz des teuren `Find` nur amortisierte Kosten von $\mathrm{O}(1)$ anfallen. Illustration von `Find`(n) am Beispiel $n = 5$.



Beim weiteren Ausführen von `Finds` in der Reihenfolge der dazugehörigen `Inserts` entsteht wieder die ursprünglich Liste. Dies wird dadurch verursacht, daß bei `Find` immer das maximale Element eines Teilbaums der Wurzel gesucht wird. Erreicht das gesuchte Element nun das Niveau eines Wurzelsohnes kann es selbst nur einen Sohn haben, einen linken. Da das Element selbst der linke Sohn der Wurzel sein muß (Wurzel ist größer), wird am Ende nur die Wurzel nach rechts rotiert, ohne das ein rechter Teilbaum des gesuchten Elementes als linker Sohn an die alte Wurzel gehängt werden könnte. Daher können die nach rechts rotierten Wurzeln nur rechte Söhne haben. Eine Liste entsteht, weil wegen der `Finds` jedes der Elemente einmal Wurzel sein muß.

2. Aufgabe

In der Vorlesung wurden für Splay-Bäume das individuelle Gewicht $\mathrm{I}_S(x)$ eines Knotens und der Rang $\mathrm{Rg}_S(x)$ eingführt. Sei $N(x)$ die Menge der Nachfolgerknoten von x im Baum, dann gilt:

$$\mathrm{I}_S(x) > 0$$

$$\mathrm{Rg}_S(x) = \log_2 \left(\sum_{x', x' \in N(x)} \mathrm{I}_S(x') \right)$$

Weiterhin wurde festgestellt, daß ein $\mathrm{Splay}_S(x)$ den amortisierten Aufwand

$$a_{\mathrm{Splay}_S(x)} = 1 + 3(\mathrm{Rg}_S(l) - \mathrm{Rg}_S(x))$$

hat, wobei l die Wurzel von S ist. Wählt man $\mathrm{I}_S = 1$ gilt $\mathrm{Rg}_S(l) \leq \log_2 n$ und $\mathrm{Rg}_S(x) \geq \log_2 1$ (n ist die Anzahl der Elemente im Baum). Also ist

$$a_{\mathrm{Splay}_{S(x)}} \leq 1 + 3(\log_2 n - \log_2 1)$$

und damit in $\mathrm{O}(\log n)$. Als Potential $\Phi(S)$ eines (Splay-) Baumes S war die Summe der Ränge aller Knoten definiert. Betrachtet man m Splay-Operationen, beginnend auf dem (nicht unbedingt leeren) Baum S_0, so ergeben sich die Gesamtkosten

$$\begin{aligned} K &= \sum_{i=1}^{m} a_{Splay} + \underbrace{\Phi S_0}_{\leq n \cdot \log n} - \underbrace{\Phi S_m}_{\geq 0} \\ &\leq m \cdot \log n + n \cdot \log n \\ &= O((n+m)\log n) \quad . \end{aligned}$$

Für den Beweis reicht es jetzt, alle Operation auf eine konstante Anzahl an Splay-Operationen zurückzuführen.

```
Find(S, x)
Splay(S, x)
Test, ob Wurzel x enthaelt

Delete(S, x)
Splay(S, x)
IF x an Wurzel von S THEN
    T := linker Teilbaum der Wurzel von S
    Splay(T, $\infty$)
    haenge rechten Teilbaum von S
    rechts an die Wurzel von T
    S := T

Insert(S, x)
Find(S, x)
y := Wurzel von S
IF x > y THEN
    T := rechter Teilbaum von S
    U := linker Teilbaum von S
    konstruiere    x
                  / \
                 y   T
                /
               U
ELSE
    T := rechter Teilbaum von S
    U := linker Teilbaum von S
    konstruiere    x
                  / \
```

```
U   y
     \
      T
```

Find: Zeile 1 in O($\log n$), Zeile 2 in $O(1)$
Find hat Kosten in O($\log n$)

Delete: Zeilen 1 und 4 in O($\log n$), alle anderen in O(1) (nur einmalige Vergleiche und Zuweisungen)
Delete hat Kosten in O($\log n$)

Insert: Zeile 1 in O($\log n$), alle anderen in O(1) (nur einmalige Vergleiche und Zuweisungen, Konstruktion durch einfaches Umhängen von Zeigern)
Insert hat Kosten in O($\log n$)

Ingesamt hat eine Folge von m Inserts, Deletes und Finds also einen Aufwand in O($m \log n$) bzw. O($(m+n)\log n$), da vorher n Elemente mit Insert eingefügt wurden.

3. Aufgabe

(a) $T_{W_S}(A) = 1$, $T_{W_S}(B) = 8$, $T_{W_S}(C) = 1$, $T_{W_S}(D) = 1$

(b) Insert(1), Insert(2), $\cdots$, Insert(n), Find(1), Find(2), $\cdots$, Find(n) $\Rightarrow$ Kosten $\Theta(n^2)$