Treaps

Theorem

Einfügen, Suchen und Löschen in einen Treap mit n Elementen benötigt $O(\log n)$ Schritte im Mittel, falls die Prioritäten aus einem ausreichend großen Bereich zufällig gewählt werden.

- Treaps sind in der Praxis sehr schnell.
- Standardimplementation für assoziative Arrays in LEDA.
- Einfach zu analysieren.
- Einfach zu implementieren.
- Sehr hübsch.

Splay-Bäume

- Splay-Bäume sind eine selbstorganisierende Datenstruktur.
- Basiert auf binären Suchbäumen.
- Restrukturiert durch Rotationen.
- Keine Zusatzinformation in Knoten.
- Nur amortisiert effizient.
- Einfach zu implementieren.
- Viele angenehme Eigenschaften (z.B. „selbstlernend“)
- Nur eine komplizierte Operation: splay

Die Splay-Operation

Gegeben ist ein binärer Suchbaum und ein Knoten x.

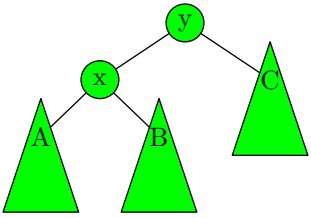
Algorithmus

procedure splay(node x) :
  while x $\neq$ root **do**
    splaystep(x)
  od

Wir führen Splay-Schritte auf x aus, bis es zur Wurzel wird.

Ein Splay-Schritt ist ein zig, zag, zig-zig, zig-zag, zag-zig oder zag-zag.

Zig und Zag

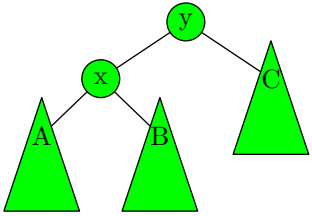


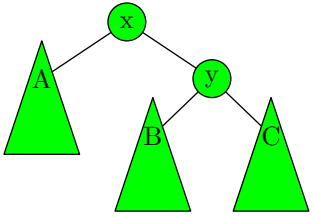
Ein zig auf x ist eine Rechtsrotation des Vaters von x.

Sie wird nur ausgeführt, wenn x das linke Kind der Wurzel ist.

Ein zag ist eine Linksrotation des Vaters, wenn x das rechte Kind der Wurzel ist.

Zig und Zag



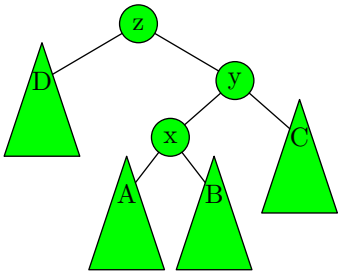


Ein zig auf x ist eine Rechtsrotation des Vaters von x.

Sie wird nur ausgeführt, wenn x das linke Kind der Wurzel ist.

Ein zag ist eine Linksrotation des Vaters, wenn x das rechte Kind der Wurzel ist.

Zig-zag und Zag-zig

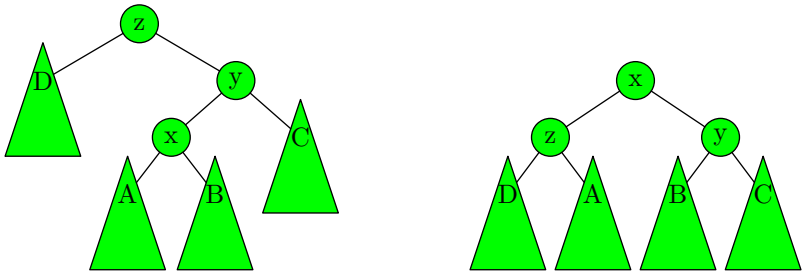


Ein Zig-zag auf x ist eine Rechtsrotation auf y gefolgt von einer Linksrotation auf z.

Dabei muß y das rechte Kind von z und x das linke Kind von y sein.

Zag-zig ist symmetrisch hierzu.

Zig-zag und Zag-zig

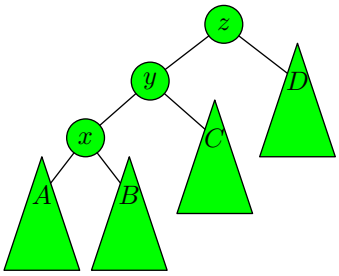


Ein Zig-zag auf x ist eine Rechtsrotation auf y gefolgt von einer Linksrotation auf z.

Dabei muß y das rechte Kind von z und x das linke Kind von y sein.

Zag-zig ist symmetrisch hierzu.

Zig-zig und Zag-zag



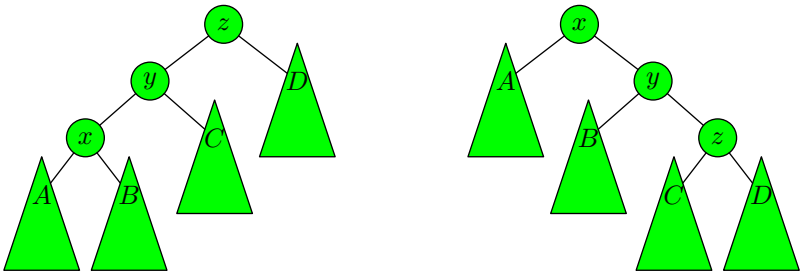
Ein Zig-zig auf x ist eine Rechtsrotation auf z gefolgt von einer Rechtsrotation auf y.

Dabei muß y das linke Kind von z und x das linke Kind von y sein.

Diese Operation sieht unerwartet aus!

Zag-zag ist wieder symmetrisch hierzu.

Zig-zig und Zag-zag



Ein Zig-zig auf x ist eine Rechtsrotation auf z gefolgt von einer Rechtsrotation auf y.

Dabei muß y das linke Kind von z und x das linke Kind von y sein.

Diese Operation sieht unerwartet aus!

Zag-zag ist wieder symmetrisch hierzu.

Amortisierte Analyse

Jeder Knoten x habe ein Gewicht $g(x) \in \mathbf{N}$.

Definition

- Der Kontostand eines Splay-Baums T ist $\sum_{v \in T} r(v)$.
- $r(v) = \left\lfloor \log(\bar{g}(v)) \right\rfloor$.
- $\bar{g}(v) = \sum_{u \in T(v)} g(u)$.
- $T(v)$ ist der Unterbaum mit Wurzel v

Amortisierte Analyse

Definition

Gegeben sei ein Splay-Schritt, der einen Splay-Baum T in einen Splay-Baum T' verwandelt.
Die amortisierten Kosten dieses Schrittes betragen

$$\sum_{v \in T'} r(v) - \sum_{v \in T} r(v) + 1.$$

Ein Schritt sind die tatsächlichen Kosten.

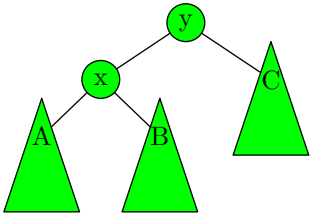
Der Rest ist eine Einzahlung oder Abhebung vom Konto.

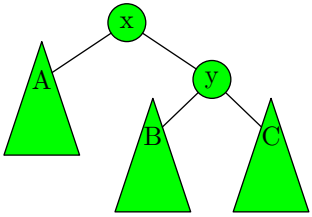
Lemma

Die amortisierten Kosten eines

- *zig sind* $\leq 1 + 3(r(y) - r(x))$,
- *zig-zag sind* $\leq 3(r(z) - r(x))$,
- *zig-zig sind* $\leq 3(r(z) - r(x))$.

x, y, z sind die Knoten in den entsprechenden Zeichnungen.
Auf x wird die Operation ausgeführt.
z ist Großvater, y ist Vater von x.

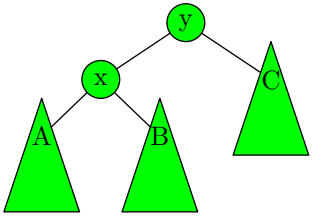


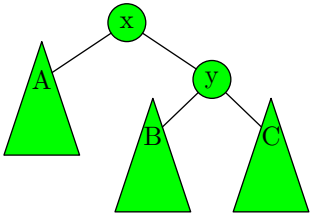


- $r'(x) = r(y)$
- $r'(y) \leq r(y)$
- $r(y) \geq r(x)$

Die amortisierten Kosten eines zig sind

$$1 + r'(x) - r(x) + r'(y) - r(y) \leq 1 + r'(y) - r(x)$$
$$\leq 1 + r(y) - r(x) \leq 1 + 3(r(y) - r(x))$$

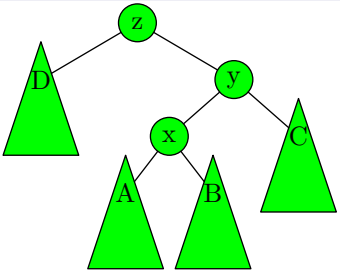


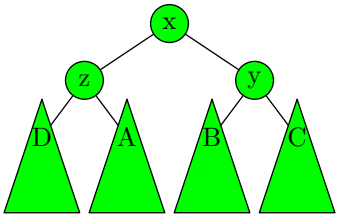


- $r'(x) = r(y)$
- $r'(y) \leq r(y)$
- $r(y) \geq r(x)$

Die amortisierten Kosten eines zig sind

$$1 + r'(x) - r(x) + r'(y) - r(y) \leq 1 + r'(y) - r(x)$$
$$\leq 1 + r(y) - r(x) \leq 1 + 3(r(y) - r(x))$$



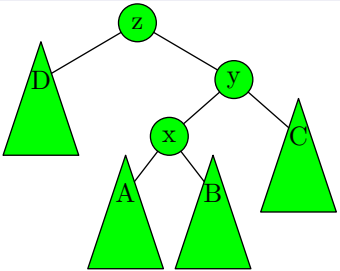


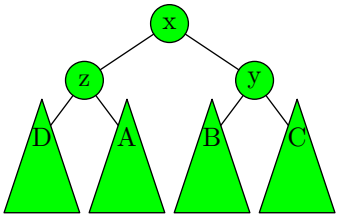
Wir nehmen erst einmal $r(z) > r(x)$ an.

- $r'(x) = r(z)$
- $r'(y) \leq r'(x) = r(z)$
- $r'(z) \leq r'(x) = r(z)$

- $r(y) \geq r(x)$
- $1 \leq r(z) - r(x)$

$$1 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z) \leq 1 + r'(y) + r'(z) - r(x) - r(y)$$
$$\leq r(z) - r(x) + r(z) + r(z) - r(x) - r(x) = 3(r(z) - r(x))$$



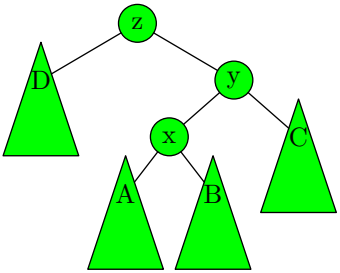


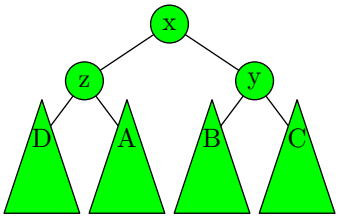
Wir nehmen erst einmal $r(z) > r(x)$ an.

- $r'(x) = r(z)$
- $r'(y) \leq r'(x) = r(z)$
- $r'(z) \leq r'(x) = r(z)$

- $r(y) \geq r(x)$
- $1 \leq r(z) - r(x)$

$$1 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z) \leq 1 + r'(y) + r'(z) - r(x) - r(y)$$
$$\leq r(z) - r(x) + r(z) + r(z) - r(x) - r(x) = 3(r(z) - r(x)).$$



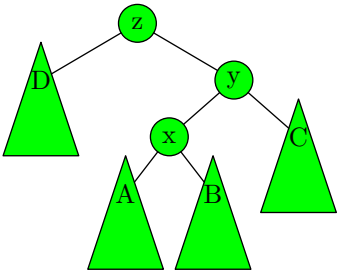


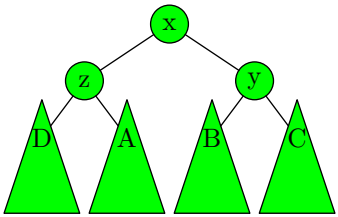
Jetzt nehmen wir $r(z) = r(x)$ an.

- $r'(x) = r(z)$
- $r'(y) < r(x)$ oder $r'(z) < r(x)$ (sonst $r'(x) > r(z)$)

$$1 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$$
$$\leq 1 + r'(y) + r'(z) - r(x) - r(y) \leq 0 = 3(r(z) - r(x))$$

(Zig-zig ähnlich)





Jetzt nehmen wir $r(z) = r(x)$ an.

- $r'(x) = r(z)$
- $r'(y) < r(x)$ oder $r'(z) < r(x)$ (sonst $r'(x) > r(z)$)

$$1 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$$
$$\leq 1 + r'(y) + r'(z) - r(x) - r(y) \leq 0 = 3(r(z) - r(x))$$

(Zig-zig ähnlich)

Splay-Bäume – Analyse

Theorem

Die amortisierten Kosten einer Splay-Operation auf einem Knoten x sind $O\big(\log(\bar{g}(w)/\bar{g}(x))\big)$, wenn w die Wurzel des Baums ist.

Beweis.

Die amortisierten Kosten sind höchstens

$$1 + 3(r(v_t) - r(v_{t-1})) + 3(r(v_{t-1}) - r(v_{t-2})) + \\ + \ldots + 3(r(v_2) - r(v_1)) = 1 + 3r(v_t) - 3r(v_1),$$

wobei $v_t = w$ und $v_1 = x$.

$$1 + 3r(w) - 3r(x) = O(\log(\bar{g}(w)) - \log(\bar{g}(x))) = O(\log(\bar{g}(w)/\bar{g}(x)))$$

□

Splay-Bäume – Analyse

Theorem

Die amortisierten Kosten einer Splay-Operation auf einem Knoten x sind $O\big(\log(\bar{g}(w)/\bar{g}(x))\big)$, wenn w die Wurzel des Baums ist.

Beweis.

Die amortisierten Kosten sind höchstens

$$1 + 3(r(v_t) - r(v_{t-1})) + 3(r(v_{t-1}) - r(v_{t-2})) + \\ + \ldots + 3(r(v_2) - r(v_1)) = 1 + 3r(v_t) - 3r(v_1),$$

wobei $v_t = w$ und $v_1 = x$.

$$1 + 3r(w) - 3r(x) = O(\log(\bar{g}(w)) - \log(\bar{g}(x))) = O(\log(\bar{g}(w)/\bar{g}(x)))$$

□

Splay-Bäume – Analyse

Theorem

Die amortisierten Kosten einer Splay-Operation auf einem Knoten x sind $O\big(\log(\bar{g}(w)/\bar{g}(x))\big)$, wenn w die Wurzel des Baums ist.

Beweis.

Die amortisierten Kosten sind höchstens

$$1 + 3(r(v_t) - r(v_{t-1})) + 3(r(v_{t-1}) - r(v_{t-2})) + \\ + \ldots + 3(r(v_2) - r(v_1)) = 1 + 3r(v_t) - 3r(v_1),$$

wobei $v_t = w$ und $v_1 = x$.

$$1 + 3r(w) - 3r(x) = O(\log(\bar{g}(w)) - \log(\bar{g}(x))) = O(\log(\bar{g}(w)/\bar{g}(x)))$$

□

Splay-Bäume – Suchen

Wir suchen folgendermaßen nach Schlüssel k:

1. Normale Suche im Suchbaum
2. Endet in Knoten x mit Schlüssel k oder in x^+ oder x^-
3. Wende Splay auf den gefundenen Knoten an
4. Siehe nach, ob k in Wurzel

Amortisierte Laufzeit:

$$O\left(\log\left(\frac{(\bar{g}(w))}{\bar{g}(x)}\right)\right) \text{ erfolgreiche Suche}$$

$$O\left(\log\left(\frac{\log(\bar{g}(w))}{\min\{\bar{g}(x^+), \bar{g}(x^-)\}}\right)\right) \text{ erfolglose Suche}$$

Splay-Bäume – Suchen

Wir suchen folgendermaßen nach Schlüssel k:

1. Normale Suche im Suchbaum
2. Endet in Knoten x mit Schlüssel k oder in x^+ oder x^-
3. Wende Splay auf den gefundenen Knoten an
4. Siehe nach, ob k in Wurzel

Amortisierte Laufzeit:

$$O\left(\log\left(\frac{(\bar{g}(w))}{\bar{g}(x)}\right)\right) \text{ erfolgreiche Suche}$$

$$O\left(\log\left(\frac{\log(\bar{g}(w))}{\min\{\bar{g}(x^+), \bar{g}(x^-)\}}\right)\right) \text{ erfolglose Suche}$$

Splay-Bäume – Einfügen

Wir fügen einen Schlüssel k mit Gewicht a ein, der noch nicht vorhanden ist:

- Normale Suche im Suchbaum
- Einfügen eines neuen Knotens als Blatt
- Splay-Operation auf diesen Knoten

Amortisierte Laufzeit:

Das Konto wird zunächst erhöht.
x sei der neu eingefügte Knoten.
Die Splay-Operation benötigt $O(\log(\bar{g}(\bar{w}))/\bar{g}(x))$.

Splay-Bäume – Einfügen

Wir fügen einen Schlüssel k mit Gewicht a ein, der noch nicht vorhanden ist:

- Normale Suche im Suchbaum
- Einfügen eines neuen Knotens als Blatt
- Splay-Operation auf diesen Knoten

Amortisierte Laufzeit:

Das Konto wird zunächst erhöht.
x sei der neu eingefügte Knoten.
Die Splay-Operation benötigt $O(\log(\bar{g}(\bar{w}))/\bar{g}(x))$.

Splay-Bäume – Löschen

Wir löschen einen Schlüssel k:

1. Suche nach dem Schlüssel k
2. Siehe nach k in der Wurzel ist
3. Splay-Operation auf dem größten Knoten im linken Unterbaum
4. Klassisches Löschen von k

Amortisierte Laufzeit:
Zuerst wie Suche:

$$O\left(\log\left(\frac{(\bar{g}(w))}{\bar{g}(x)}\right)\right) \text{ erfolgreiche Suche}$$

$$O\left(\log\left(\frac{\log(\bar{g}(w))}{\min\{\bar{g}(x^+), \bar{g}(x^-)\}}\right)\right) \text{ erfolglose Suche}$$

Dann sinkt der Kontostand, was wir aber nicht ausnutzen.

Splay-Bäume – Löschen

Wir löschen einen Schlüssel k:

1. Suche nach dem Schlüssel k
2. Siehe nach k in der Wurzel ist
3. Splay-Operation auf dem größten Knoten im linken Unterbaum
4. Klassisches Löschen von k

Amortisierte Laufzeit:
Zuerst wie Suche:

$$O\left(\log\left(\frac{(\bar{g}(w))}{\bar{g}(x)}\right)\right) \text{ erfolgreiche Suche}$$

$$O\left(\log\left(\frac{\log(\bar{g}(w))}{\min\{\bar{g}(x^+), \bar{g}(x^-)\}}\right)\right) \text{ erfolglose Suche}$$

Dann sinkt der Kontostand, was wir aber nicht ausnutzen.

Splay-Bäume als assoziatives Array

Theorem

In einem anfänglich leeren Splay-Baum können n Operationen (Suchen, Einfügen, Löschen) in $O(n \log n)$ Schritten ausgeführt werden.

Beweis.

Wir setzen $g(x) = 1$.

Dann ist $\bar{g}(T)$ die Anzahl der Knoten im Baum.
Also ist $\bar{g}(T) \leq n$.
Die amortisierten Kosten einer Operation sind $O(\log(\bar{g}(T)) = O(\log n)$.
(Beim Einfügen kommt zur Splay-Operation noch die Erhöhung des Kontostands um $O(\log n)$ hinzu.) □

Splay-Bäume als assoziatives Array

Theorem

In einem anfänglich leeren Splay-Baum können n Operationen (Suchen, Einfügen, Löschen) in $O(n \log n)$ Schritten ausgeführt werden.

Beweis.

Wir setzen $g(x) = 1$.

Dann ist $\bar{g}(T)$ die Anzahl der Knoten im Baum.
Also ist $\bar{g}(T) \leq n$.
Die amortisierten Kosten einer Operation sind $O(\log(\bar{g}(T)) = O(\log n)$.
(Beim Einfügen kommt zur Splay-Operation noch die Erhöhung des Kontostands um $O(\log n)$ hinzu.) □

Splay-Bäume als assoziatives Array

Theorem

In einem anfänglich leeren Splay-Baum können n Operationen (Suchen, Einfügen, Löschen) in $O(n \log n)$ Schritten ausgeführt werden.

Beweis.

Wir setzen $g(x) = 1$.

Dann ist $\bar{g}(T)$ die Anzahl der Knoten im Baum.
Also ist $\bar{g}(T) \leq n$.
Die amortisierten Kosten einer Operation sind $O(\log(\bar{g}(T)) = O(\log n)$.
(Beim Einfügen kommt zur Splay-Operation noch die Erhöhung des Kontostands um $O(\log n)$ hinzu.) □

Amortisierte Analyse – Wiederholung

Eine Datenstruktur werde durch n Operationen verändert:

$$D_0 \xrightarrow{t_1} D_1 \xrightarrow{t_2} D_2 \xrightarrow{t_3} \cdots \xrightarrow{t_n} D_n$$

Die Zeit dafür ist $\sum_{k=1}^{n} t_i$.

Klassische Analyse:

- Jede Operation benötigt höchstens $f(n)$ Schritte
- Die Gesamtzeit ist $O(f(n)n)$.

Problematisch, wenn t_i sehr schwankend.

Amortisierte Analyse – Wiederholung

Eine Datenstruktur werde durch n Operationen verändert:

$$D_0 \xrightarrow{t_1} D_1 \xrightarrow{t_2} D_2 \xrightarrow{t_3} \cdots \xrightarrow{t_n} D_n$$

Die Zeit dafür ist $\sum_{k=1}^{n} t_i$.

Klassische Analyse:

- Jede Operation benötigt höchstens $f(n)$ Schritte
- Die Gesamtzeit ist $O(f(n)n)$.

Problematisch, wenn t_i sehr schwankend.

Amortisierte Analyse – Wiederholung

Eine Datenstruktur werde durch n Operationen verändert:

$$D_0 \xrightarrow{t_1} D_1 \xrightarrow{t_2} D_2 \xrightarrow{t_3} \cdots \xrightarrow{t_n} D_n$$

Die Zeit dafür ist $\sum_{k=1}^{n} t_i$.

Klassische Analyse:

- Jede Operation benötigt höchstens $f(n)$ Schritte
- Die Gesamtzeit ist $O(f(n)n)$.

Problematisch, wenn t_i sehr schwankend.

Amortisierte Analyse – Wiederholung

$$D_0 \xrightarrow{t_1} D_1 \xrightarrow{t_2} D_2 \xrightarrow{t_3} \cdots \xrightarrow{t_n} D_n$$

Wir definieren uns eine Potentialfunktion Φ mit:

1. $\Phi(D_0) = 0$
2. $\Phi(D_i) \geq 0$
3. $a_i := t_i + \Phi(D_i) - \Phi(D_{i-1})$ für $i > 0$
4. a_i nicht sehr schwankend

Amortisierte Analyse:

- Zeige, daß $a_i \leq g(n)$
- Gesamtzeit höchstens $O(ng(n))$

$$ng(n) \geq \sum_{k=1}^{n} a_i = \sum_{k=1}^{n} t_i + \Phi(D_n) + \Phi(D_0) \geq \sum_{k=1}^{n} t_i$$

Amortisierte Analyse – Wiederholung

$$D_0 \xrightarrow{t_1} D_1 \xrightarrow{t_2} D_2 \xrightarrow{t_3} \cdots \xrightarrow{t_n} D_n$$

Wir definieren uns eine Potentialfunktion Φ mit:

1. $\Phi(D_0) = 0$
2. $\Phi(D_i) \geq 0$
3. $a_i := t_i + \Phi(D_i) - \Phi(D_{i-1})$ für $i > 0$
4. a_i nicht sehr schwankend

Amortisierte Analyse:

- Zeige, daß $a_i \leq g(n)$
- Gesamtzeit höchstens $O(ng(n))$

$$ng(n) \geq \sum_{k=1}^{n} a_i = \sum_{k=1}^{n} t_i + \Phi(D_n) + \Phi(D_0) \geq \sum_{k=1}^{n} t_i$$

Splay-Bäume – Beispiel

Die Schlüssel von 1 bis 40 werden zufällig eingefügt und gelöscht.