Splaybaum

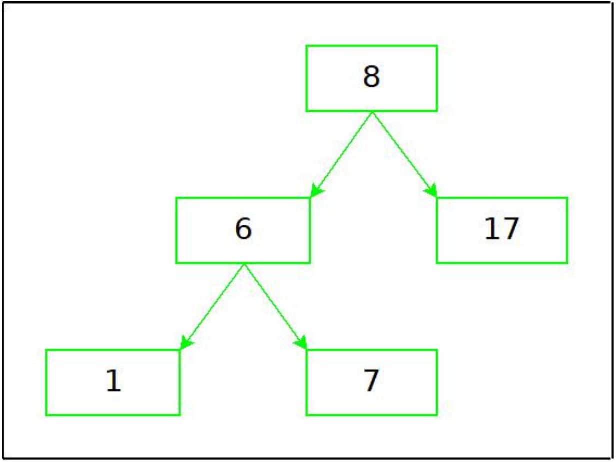
Andreas Windorfer

27. Oktober 2020

Übersicht

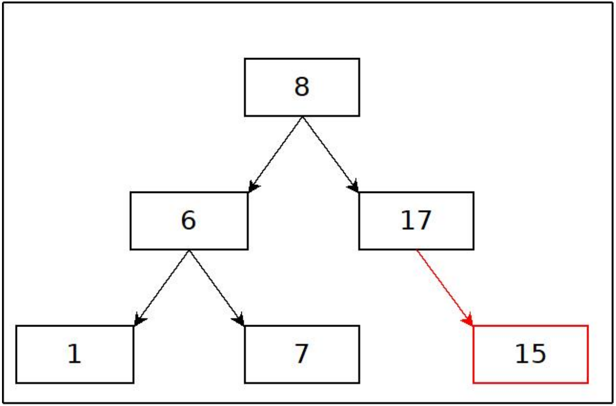
Binärer Suchbaum

- links kleinere Schlüssel
- rechts größere Schlüssel



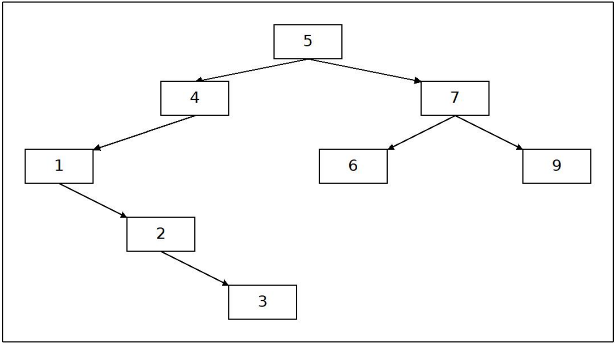
Binärer Suchbaum

- links kleinere Schlüssel
- rechts größere Schlüssel



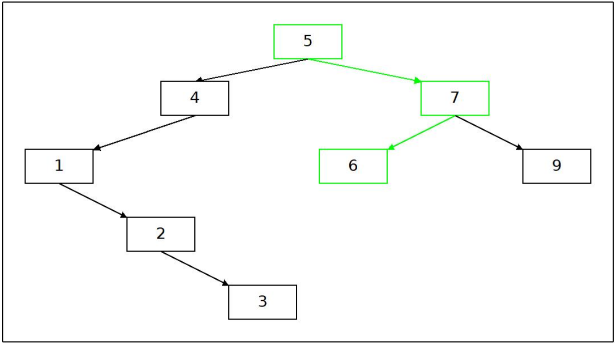
Binärer Suchbaum

Suche nach Schlüssel 6



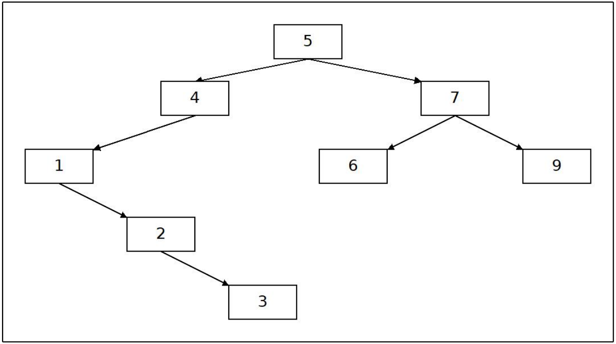
Binärer Suchbaum

Suche nach Schlüssel 6



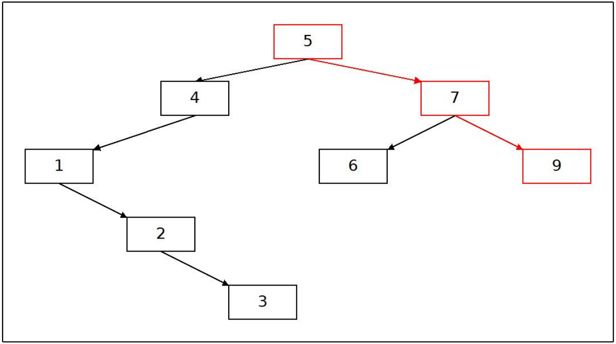
Binärer Suchbaum

Einfügen von Schlüssel 8



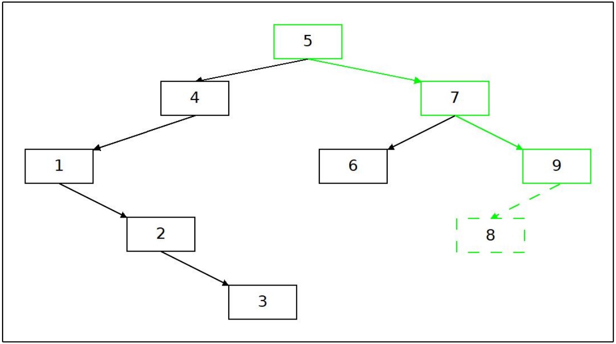
Binärer Suchbaum

Einfügen von Schlüssel 8



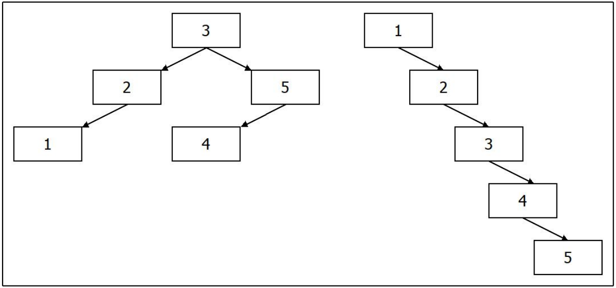
Binärer Suchbaum

Einfügen von Schlüssel 8



Binärer Suchbaum

Einfügereihenfolgen



Aufbau des Splaybaum

Besonderheiten

Aufbau des Splaybaum

Besonderheiten

- selbstorganisierend / dynamisch

Aufbau des Splaybaum

Besonderheiten

- selbstorganisierend / dynamisch
- häufig benutzte Elemente oben

Aufbau des Splaybaum

Besonderheiten

- selbstorganisierend / dynamisch
- häufig benutzte Elemente oben
- i.d.R. kürzere Pfadlängen

Übersicht

Operationen

- splay
- suchen
- aufteilen
- vereinigen
- einfügen
- löschen

tree splay(key x, tree s)

- normale Suche
- Rotationen
  - zig / zag
  - zig-zag / zag-zig
  - zig-zig /zag-zag

tree splay(key x, tree s)

- normale Suche
- Rotationen
  - zig / zag
  - zig-zag / zag-zig
  - zig-zig /zag-zag
- x neue Wurzel

tree splay(key x, tree s)

- normale Suche
- Rotationen
  - zig / zag
  - zig-zag / zag-zig
  - zig-zig /zag-zag
- x neue Wurzel
- links-rechts Anordnung

tree splay(key x, tree s)

- normale Suche
- Rotationen
  - zig / zag
  - zig-zag / zag-zig
  - zig-zig /zag-zag
- x neue Wurzel
- links-rechts Anordnung
- halbierte Pfadlängen

zig Rotation

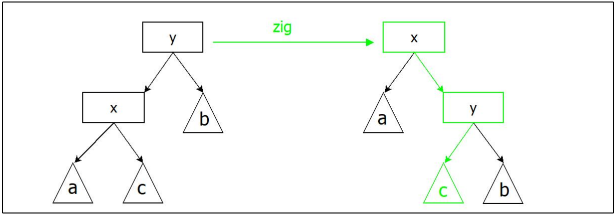
Anwendung

- x liegt direkt unter Wurzel
- x ist linker Nachfolger

zig Rotation

Anwendung

- x liegt direkt unter Wurzel
- x ist linker Nachfolger



zag Rotation

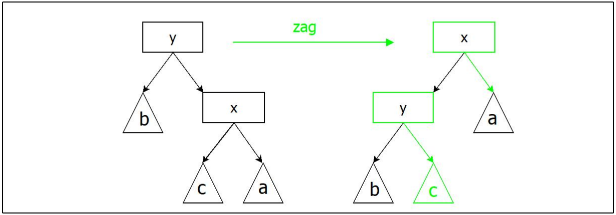
Anwendung

- x liegt direkt unter Wurzel
- x ist rechter Nachfolger
- symmetrisch zu zig

zag Rotation

Anwendung

- x liegt direkt unter Wurzel
- x ist rechter Nachfolger
- symmetrisch zu zig



zig-zag Rotation

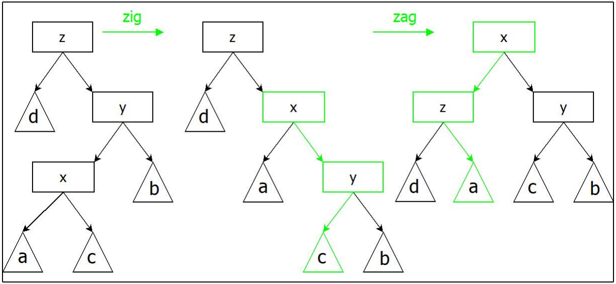
Anwendung

- x ist linker Nachfolger von y
- y ist rechter Nachfolger

zig-zag Rotation

Anwendung

- x ist linker Nachfolger von y
- y ist rechter Nachfolger



zag-zig Rotation

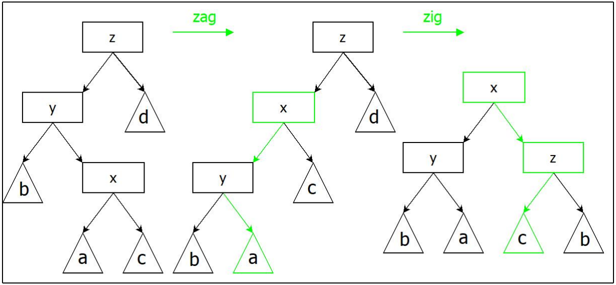
Anwendung

- x ist rechter Nachfolger von y
- y ist linker Nachfolger
- symmetrisch zu zig-zag

zag-zig Rotation

Anwendung

- x ist rechter Nachfolger von y
- y ist linker Nachfolger
- symmetrisch zu zig-zag



zig-zig Rotation

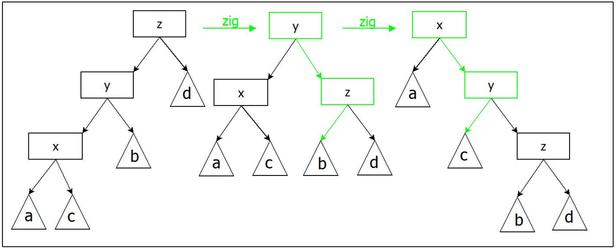
Anwendung

- x ist linker Nachfolger von y
- y ist linker Nachfolger

zig-zig Rotation

Anwendung

- x ist linker Nachfolger von y
- y ist linker Nachfolger



zag-zag Rotation

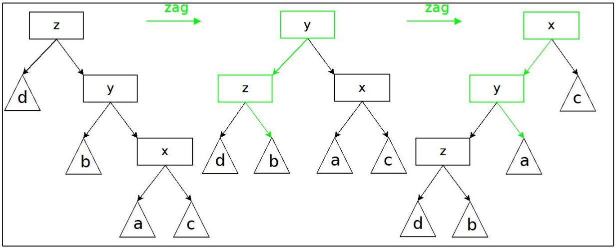
Anwendung

- x ist rechter Nachfolger von y
- y ist rechter Nachfolger
- symmetrisch zu zig-zig

zag-zag Rotation

Anwendung

- x ist rechter Nachfolger von y
- y ist rechter Nachfolger
- symmetrisch zu zig-zig

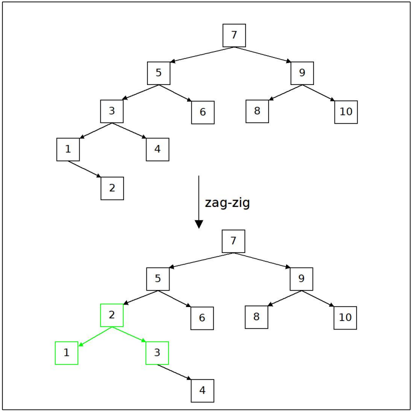


splay gesamt

- tree splay(key 2, tree s)

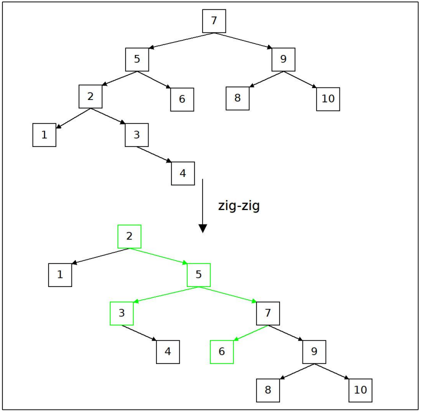
splay gesamt

- tree splay(key 2, tree s)



splay gesamt

- tree splay(key 2, tree s)



splay

- halbierte Pfadlängen

splay

- halbierte Pfadlängen

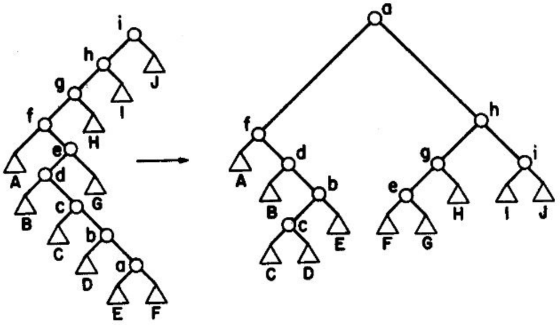


FIG. 4. Splaying at node *a*.

Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. J. ACM, 32(3):652–686, July 1985.

Operationen

- splay
- suchen
- aufteilen
- vereinigen
- einfügen
- löschen

tree suchen(key x, tree s)

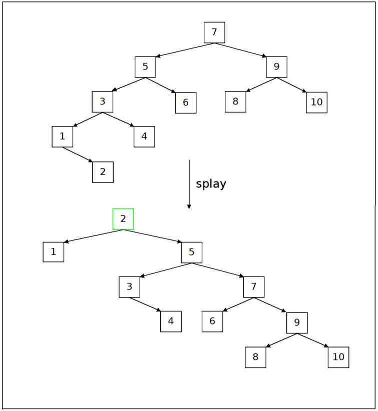
suchen (2, s)

1. splay (2 , s)

tree suchen(key x, tree s)

suchen (2, s)

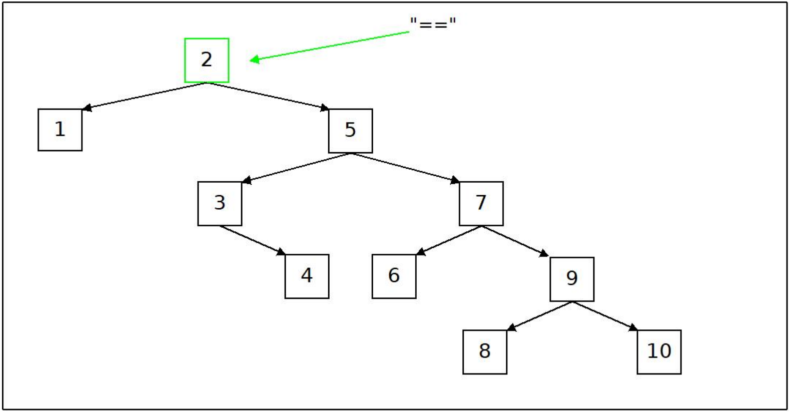
1. splay (2 , s)



tree suchen(key x, tree s)

suchen (2, s)

1. splay (2 , s)
2. Test der Wurzel



Operationen

- splay
- suchen
- aufteilen
- vereinigen
- einfügen
- löschen

tree, tree aufteilen(key x, tree s)

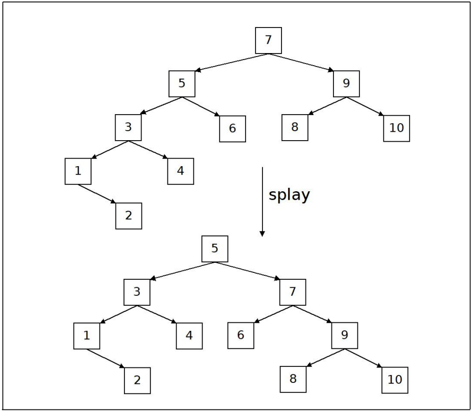
aufteilen (5, s)

1. splay (5 , s)

tree, tree aufteilen(key x, tree s)

aufteilen (5, s)

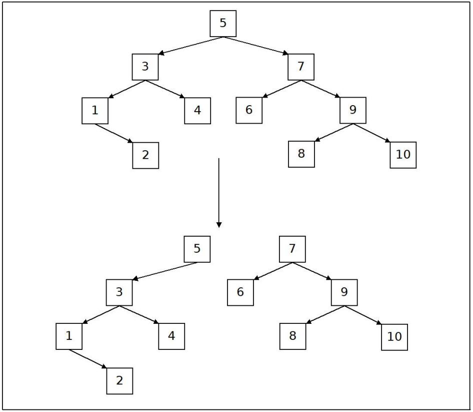
1. splay (5 , s)



tree, tree aufteilen(key x, tree s)

aufteilen (5, s)

1. splay (5 , s)
2. rechts abtrennen



Operationen

- splay
- suchen
- aufteilen
- vereinigen
- einfügen
- löschen

tree vereinigen(tree t1 , tree t2)

vereinigen (t1, t2)

Bedingung

$maxKey(t1) < minKey(t2)$

1. splay (maxValue , t1)

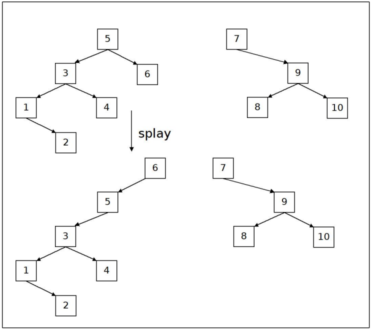
tree vereinigen(tree t1 , tree t2)

vereinigen (t1, t2)

Bedingung

$maxKey(t1) < minKey(t2)$

1. splay (maxValue , t1)



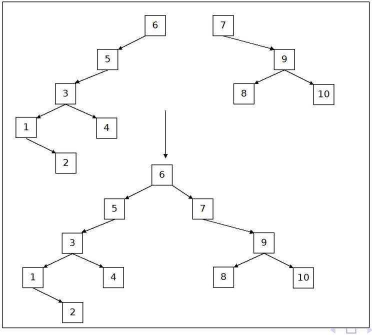
tree vereinigen(tree t1 , tree t2)

vereinigen (t1, t2)

Bedingung

$maxKey(t1) < minkey(t2)$

1. splay (maxValue , t1)
2. t2 rechts anhängen



Operationen

- splay
- suchen
- aufteilen
- vereinigen
- einfügen
- löschen

tree einfügen(key x, tree s)

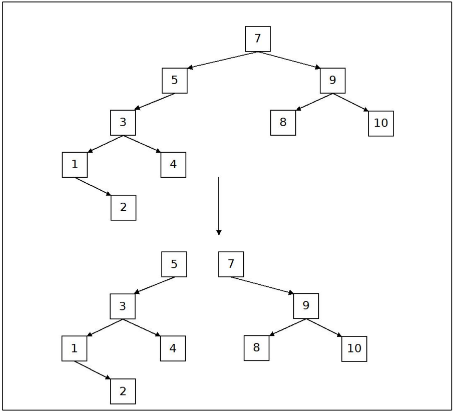
einfügen (6, s)

1. aufteilen (6 , s)

tree einfügen(key x, tree s)

einfügen (6, s)

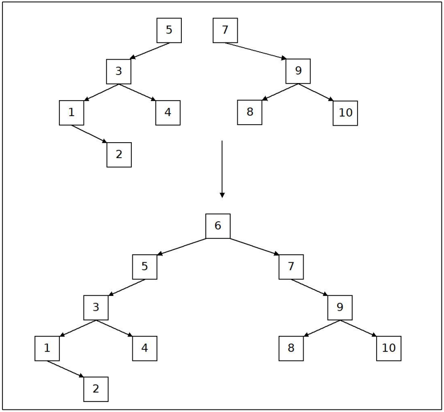
1. aufteilen (6 , s)



tree einfügen(key x, tree s)

einfügen (6, s)

1. aufteilen (6 , s)
2. t1, t2 über x zusammenfügen



Operationen

- splay
- suchen
- aufteilen
- vereinigen
- einfügen
- löschen

tree löschen (key x, tree s)

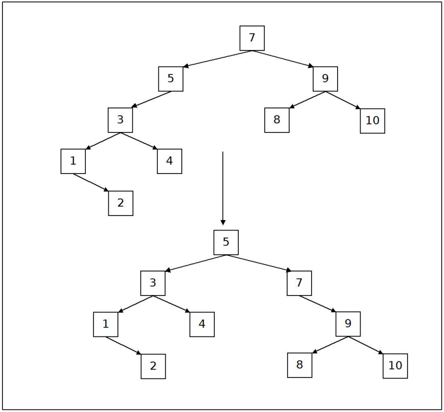
löschen (5, s)

1. suchen (5, s)

tree löschen (key x, tree s)

löschen (5, s)

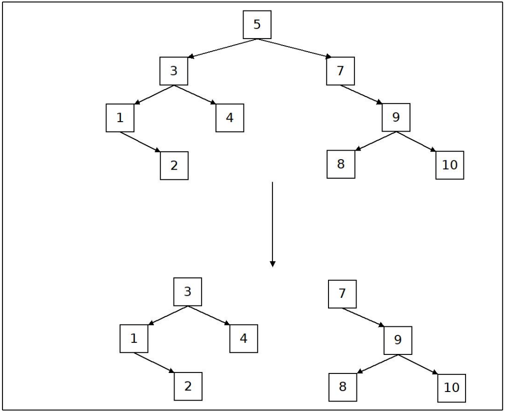
1. suchen (5, s)



tree löschen (key x, tree s)

löschen (5, s)

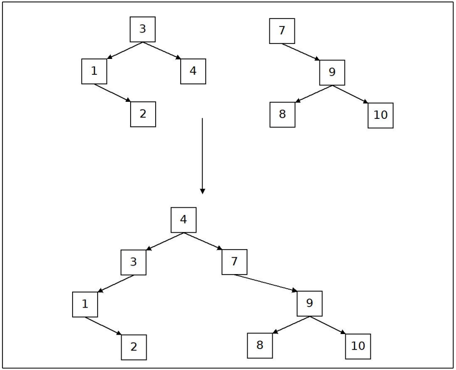
1. suchen (5, s)
2. Wurzel entfernen



tree löschen (key x, tree s)

löschen (5, s)

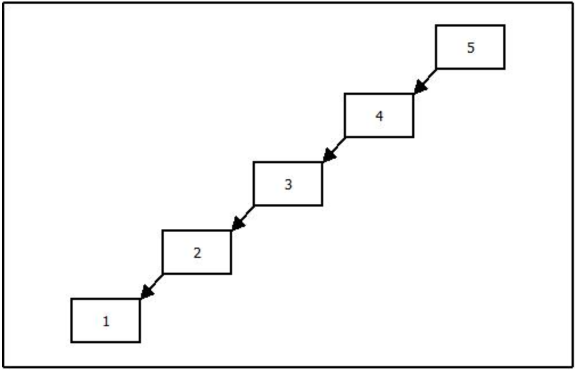
1. suchen (5, s)
2. Wurzel entfernen
3. vereinigen (t1, t2)



Übersicht

Liste

Einfügereihenfolge: 1, 2, 3, 4, 5

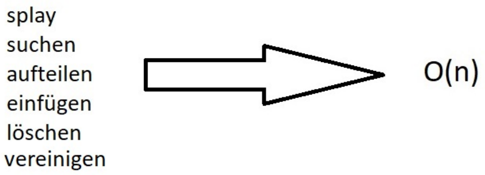


Laufzeit Einzeloperationen

- splay: n Ebenen

Laufzeit Einzeloperationen

- splay: n Ebenen



amortisierte Laufzeit

- Operationsfolgen im schlechtesten Fall
  - Bankkontomethode
  - Potentialfunktionmethode

amortisierte Laufzeit

- Operationsfolgen im schlechtesten Fall
  - Bankkontomethode
  - Potentialfunktionmethode
- Bei Splaybaum Potentialfunktionmethode

Bankkontomethode

- günstige Operationen subventionieren teure
- amortisierte Kosten sind obere Schranke

Bankkontomethode

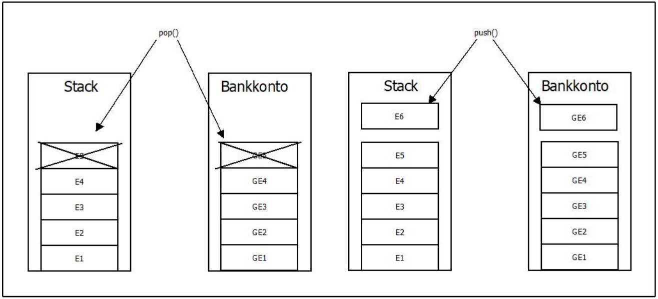
- günstige Operationen subventionieren teure
- amortisierte Kosten sind obere Schranke
- Beispiel Stack mit Operationen
    - pop() mit Kosten $y = O(1)$
    - popAll() mit Kosten $ny = O(n)$
    - push() mit Kosten $x = O(1)$

Bankkontomethode

- Guthaben G
- tatsächliche Kosten c_i
- amortisierte Kosten $a_i = c_i + G_i - G_{i-1}$
- $\sum\limits_{i=1}^{n} (a_i - c_i) \geq 0$

Bankkontomethode

- push() zahlt y ein
- pop() entnimmt y



Potentialfunktionmethode

- baut auf Bankkontomethode auf
- Φ_i bestimmt Guthaben
- amortisierte Kosten $a_i = c_i + \Phi_i - \Phi_{i-1}$
- amortisierte Gesamtosten $(\sum_{i=1}^{n} c_i) + \Phi_n + \Phi_0$

Potentialfunktionmethode

- baut auf Bankkontomethode auf
- Φ_i bestimmt Guthaben
- amortisierte Kosten $a_i = c_i + \Phi_i - \Phi_{i-1}$
- amortisierte Gesamtosten $(\sum_{i=1}^{n} c_i) + \Phi_n + \Phi_0$
- Beim Stack-Beispiel $\Phi_i = y * \text{Anzahl Elemente}$

amortisierte splay-Kosten

- Gewichtsfunktion $iw(i) \geq 1$ für Schlüssel i

amortisierte splay-Kosten

- Gewichtsfunktion $iw(i) \geq 1$ für Schlüssel i
- Gesamtgewichtsfunktion $tw(n) = \sum_{i \in t_n} iw(i)$ für Knoten n

amortisierte splay-Kosten

- Gewichtsfunktion $iw(i) \geq 1$ für Schlüssel i
- Gesamtgewichtsfunktion $tw(n) = \sum_{i \in t_n} iw(i)$ für Knoten n
- Rang $r(n) = \lfloor \log_2(tw(n)) \rfloor$

amortisierte splay-Kosten

- Gewichtsfunktion $iw(i) \geq 1$ für Schlüssel i
- Gesamtgewichtsfunktion $tw(n) = \sum\limits_{i \in t_n} iw(i)$ für Knoten n
- Rang $r(n) = \lfloor \log_2(tw(n)) \rfloor$
- Potentialfunktion $\phi = \sum\limits_{n \in T} r(n)$

amortisierte splay-Kosten

Zugriffslemma:

Eine splay Operation mit Knoten x in einem Baum mit Wurzel v hat maximal amortisierte Kosten a von
$3(r(v) - r(x)) + 1 = O(log((tw(v)/tw(x))$

amortisierte splay-Kosten

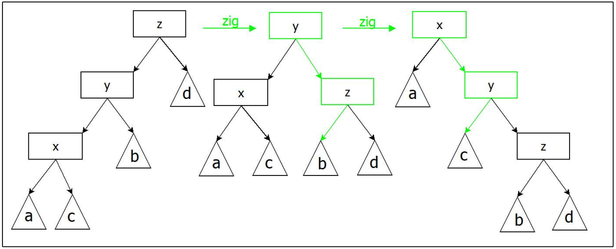
Zugriffslemma:

Eine splay Operation mit Knoten x in einem Baum mit Wurzel v hat maximal amortisierte Kosten a von
$3(r(v) - r(x)) + 1 = O(log((tw(v)/tw(x))$

Kosten der Rotationen:

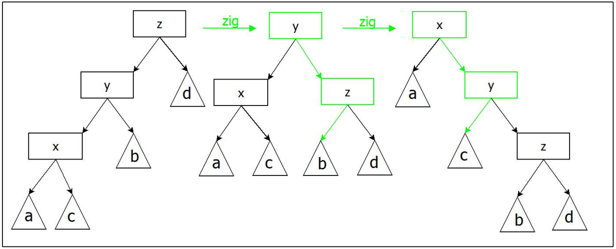
zig bzw zag :	$3(r('x) - r(x)) + 1$
zigZag bzw zagZig :	$3(r'(x) - r(x))$
zigZig bzw zagZag :	$3(r'(x) - r(x))$

Beispiel zigZig



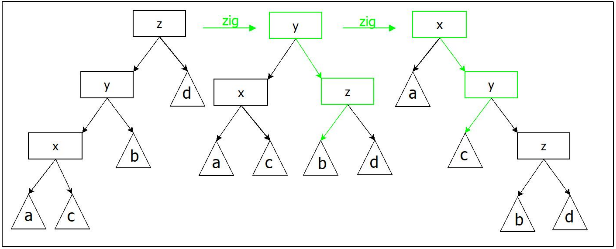
$$\phi' - \phi = r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$$

Beispiel zigZig



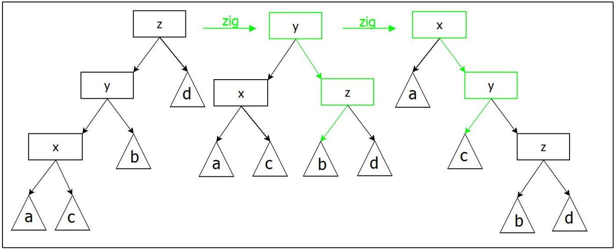
$$\phi' - \phi = r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$$
$$= \quad r'(y) + r'(z) - r(x) - r(y) \leq 2(r'(x) - r(x))$$

Beispiel zigZig



Fall: $r'(x) > r(x)$
$3(r'(x) - r(x)) - 2(r'(x) - r(x)) > 0$

Beispiel zigZig



Fall: $r'(x) = r(x)$

$\Rightarrow r'(x) = r(y) = r(z) = r(x)$

$\Rightarrow r(z') < r(z)$ und $r'(y) \leq r(y)$

$\Rightarrow \phi' < \phi$

aufaddierte Rotationen

$$3(r'(x) - r(x)) + 3(r''(x) - r'(x)) + 3(r'''(x) - r''(x)) + 1$$

aufaddierte Rotationen

$$
\begin{aligned}
& 3(r'(x) - r(x)) + 3(r''(x) - r'(x)) + 3(r'''(x) - r''(x)) + 1 \\
= \quad & 3r'''(x) + 3r''(x) - 3r''(x) + 3r'(x) - 3r'(x) - 3r(x) + 1
\end{aligned}
$$

aufaddierte Rotationen

$$
\begin{aligned}
& 3(r'(x) - r(x)) + 3(r''(x) - r'(x)) + 3(r'''(x) - r''(x)) + 1 \\
= \quad & 3r'''(x) + 3r''(x) - 3r''(x) + 3r'(x) - 3r'(x) - 3r(x) + 1 \\
= \quad & 3r'''(x) - 3r(x) + 1
\end{aligned}
$$

aufaddierte Rotationen

$$
\begin{aligned}
& 3(r'(x) - r(x)) + 3(r''(x) - r'(x)) + 3(r'''(x) - r''(x)) + 1 \\
= \quad & 3r'''(x) + 3r''(x) - 3r''(x) + 3r'(x) - 3r'(x) - 3r(x) + 1 \\
= \quad & 3r'''(x) - 3r(x) + 1 \\
= \quad & 3(r(v) - 3r(x)) + 1
\end{aligned}
$$

Übersicht

Aufbau des Splaybaum

Operationen

Laufzeitverhalten

Besondere Eigenschaften

statische Optimalität

- Elemente E_1, E_2, $E_3..E_n$
- Zugriffshäufigkeit $p(E_1)$, $p(E_2)$, $p(E_3)...p(E_n)$

statische Optimalität

- Elemente E_1, E_2, $E_3..E_n$
- Zugriffshäufigkeit $p(E_1)$, $p(E_2)$, $p(E_3)...p(E_n)$

$\longrightarrow$ statisch optimaler Suchbaum konstruierbar

statische Optimalität

Satz zur statischen Optimalität:

Es sei $q(i)$ die Anzahl der Zugriffe auf den Schlüssel i. n die Anzahl der Knoten und m die Anzahl der Zugriffe insgesamt. Gilt für jeden Schlüssel i, $q(i) \geq 1$. Dann gilt für die Kosten k des Gesamtzugriffes $k = O(m + \sum_{i=1}^{n} q(i) log(\frac{m}{q(i)}))$.

Beweis:

Nachvollziehbar in

Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. J. ACM, 32(3):652–686, July 1985.).

dynamische Optimalität

Sei A ein binärer Suchbaumalgorithmus mit den Eigenschaften:

1. Start ab Wurzel
2. komplette Pfade
3. Knotenzugriff in konstanter Zeit
4. Rotation in konstanter Zeit
5. beliebig viele Rotationen

dynamische Optimalität

Sei A ein binärer Suchbaumalgorithmus mit den Eigenschaften:

1. Start ab Wurzel
2. komplette Pfade
3. Knotenzugriff in konstanter Zeit
4. Rotation in konstanter Zeit
5. beliebig viele Rotationen

Vermutung: Splaybaum ist dynamisch optimal.