Rot-Schwarz-Baum

Vortrag

Hochschule Mittweida

13.01.2014

Tom Kretzschmar & Dominic Richter

Medieninformatik 2012

0. Inhalt

1. Binärbaum

- gewurzelter Baum
- jeder Knoten hat höchstens zwei Kindknoten
- Meist wird verlangt, dass sich die Kindknoten eindeutig in ein linkes und rechtes Kind einteilen lassen.
- Ein anschauliches Beispiel ist die Ahnentafel, bei der allerdings die Elternteile den Kindknoten zuzuordnen sind.
- Ein Binärbaum ist entweder leer, oder er besteht aus einer Wurzel mit einem linken und rechten Teilbaum, die wiederum Binärbäume sind.
- Ist ein Teilbaum leer, bezeichnet man den entsprechenden Kindknoten als fehlend.

2. Rot-Schwarz-Baum

- Rot-Schwarz-Bäume wurden zuerst 1972 von Rudolf Bayer beschrieben (Linke Person)
- Nannte sie symmetric binary B-trees
- Der heutige Name geht auf Leo J. Guibas (mittlere Person) und Robert Sedgewick zurück (rechte Person)
- die 1978 die rot-schwarze Farbkonvention einführten.
- Rot-Schwarz-Bäume sind immer ausbalanciert
- Blätter sind NIL-Marker -> speichern keine Schlüssel
- Und Anwendung finden diese in:
    - In-Memory Index-Strukturen
    - Maps, assoziative Arrays (bei C++ z.B. STL map, bei Java TreeMap

3. Eigenschaften

- Jeder Knoten hat das Attribut Farbe -> Rot oder Schwarz
- Eigenschaften garantieren, dass ein Rot-Schwarz-Baum immer balanciert ist
- wodurch die Höhe mit n Werten nie größer als 2 $*\log_2(n+2)-2$ wird
- Somit können die wichtigsten Operationen in Suchbäumen – suchen, einfügen und löschen – garantiert in O(log n) ausgeführt werden.
- Was Vorteil der schnellen Suche bzw Zugriffszeit bringt (gegenüber normalen Binärbäumen)

3.1 Eigenschaft 1

Root-Element und alle letzten Elemente (NIL) sind schwarz.

3.2 Eigenschaft 2

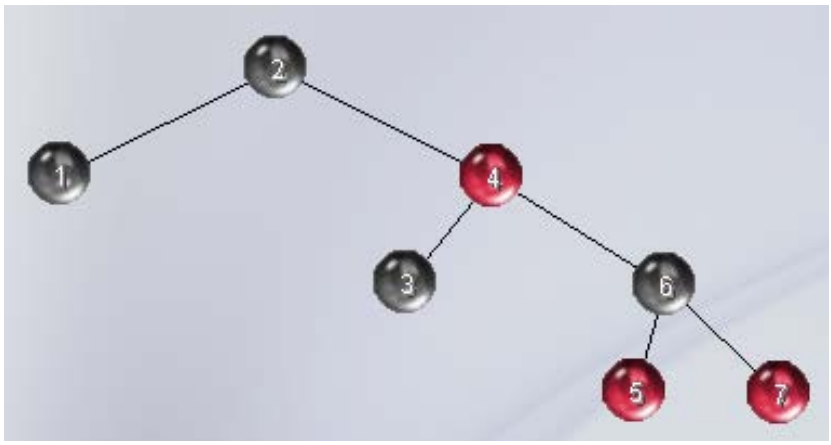
Wenn ein Knoten rot ist, dann sind beide Kinder schwarz

3.3 Eigenschaft 3

Der Pfad von einem gegebenen Knoten zu einem Blattknoten enthält immer die gleiche Anzahl schwarzer Knoten.

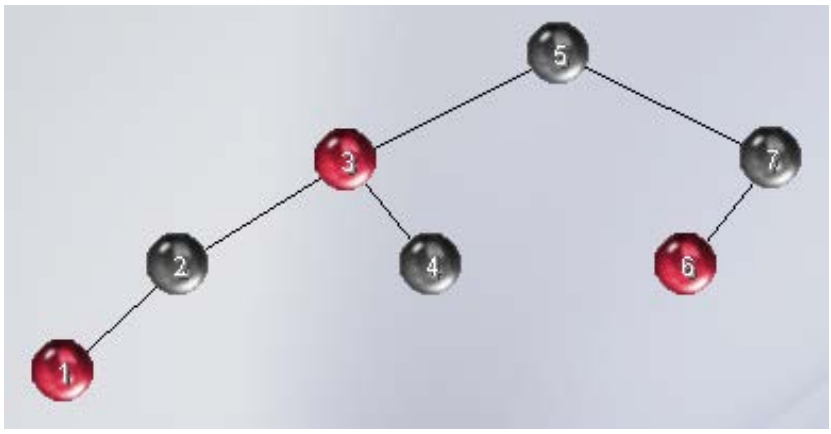
4. Programm

- Tom zeigt die Test.java
- Dominic malt jeweils die Bäume an die Tafel



Beispiel 1

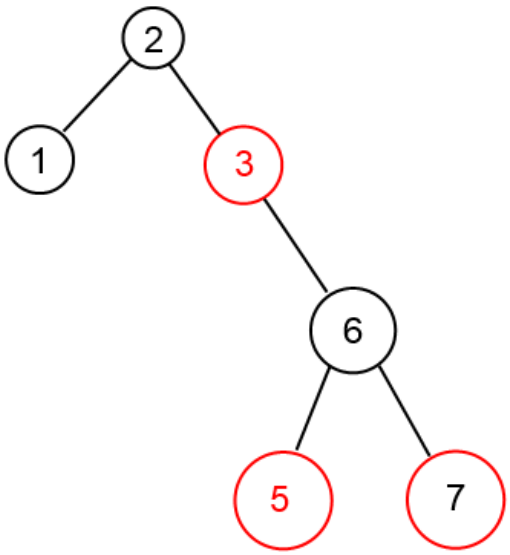
1-2-3-4-5-6-7



Beispiel 2

7-4-5-3-2-1-6

Beispiel 3 -> Beispiel 1 mit gelöschter 4



5. Knoten

- Node.java bildet Grundlage für die Datenstruktur und ist für das Speichern der Objekte zuständig
- Zugriff erfolgt über den Key
- Zur besseren objektorientierter Programmierung sind get() und set() Methoden im Klassenkörper

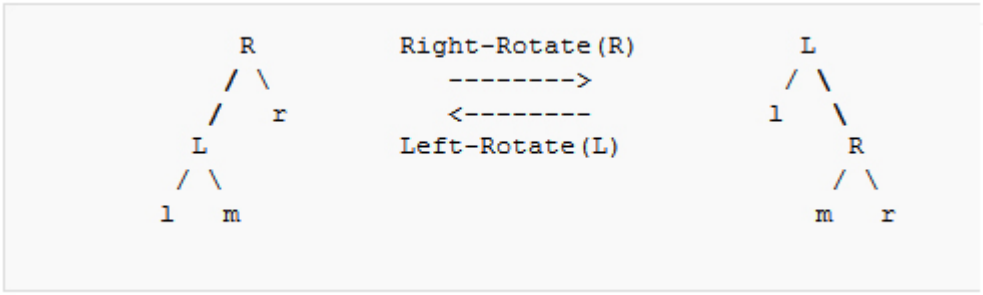
6. Hilfsmethoden

Erklärung zu Right-Rotate(R)

- „R“ wird zum rechten Teilbaum von „L“.
- Der ursprünglich rechte Teilbaum „m“ von „L“ wird zum linken Teilbaum von „R“.

Erklärung zu Left-Rotate(L)

- „L“ wird zum linken Teilbaum von „R“.
- Der ursprünglich linke Teilbaum „m“ von „R“ wird zum rechten Teilbaum von „L“.



In beiden Fällen ändert sich die Aufhängung des neuen Baums von oben her, wogegen die Aufhängung der Teilbäume „l“ und „r“ gleich bleibt.

7. Einfügen

- Das Einfügen in den Rot-Schwarz-Baum funktioniert wie das Einfügen in einen binären Suchbaum.
- Neue Knoten werden rot gefärbt
- Nach dem Einfügen: 2 Forderung (Elternknoten rot so sind beide Kinder schwarz) kann verletzt sein
- Deswegen muss der Baum durch bis zu 5 Fälle repariert werden.

7.1 Balancieren nach Einfügen

Betrachte wir die 5 Einfüge-Fälle genauer:

7.1.1 Einfüge-Fall 1

Der eingefügte Knoten ist der erste Knoten im Baum und damit das Wurzelelement.

-> eingefügter Knoten wird schwarz eingefärbt

7.1.2 Einfüge-Fall 2

Der Vater des eingefügten Knotens ist schwarz.

- dritte Forderung (Pro Pfad – gleiche Anzahl Schwarzer Knoten) könnte verletzt sein
- da der neue Knoten selbst wieder zwei schwarze NIL-Knoten mitbringt und somit die Schwarztiefe auf einem der Pfade um eins erhöht wird.
- der eingefügte Knoten selbst ist aber rot
- hat beim Einfügen einen schwarzen NIL-Knoten verdrängt

- Schwarztiefe bleibt auf allen Pfaden erhalten und es ist nichts zu tun.

7.1.3 Einfüge-Fall 3

Sowohl der Onkel als auch der Vater des eingefügten Knotens sind rot.

- beide Knoten einfach schwarz färben
- den Großvater rot färben (dritte Forderung wiederhergestellt)
- Problem um zwei Ebenen nach oben verschoben
- durch den nun rot gefärbten Großvater könnte die zweite Forderung verletzt sein
- Großvater muss nun betrachtet werden
- wird solange rekursiv fortgesetzt, bis alle Forderungen erfüllt sind

7.1.4 Einfüge-Fall 4

Der eingefügte Knoten hat einen schwarzen oder keinen Onkel und ist das rechte Kind seines roten Vaters.

- Linksrotation um den Vater ausführen
- Vertauscht die Rolle des einzufügenden Knotens und seines Vaters
- Den ehemaligen Vaterknoten mit dem fünften Fall bearbeiten
- Durch die Rotation wurde ein Pfad so verändert, dass er nun durch einen zusätzlichen Knoten führt
- während ein anderer Pfad so verändert wurde, dass er nun einen Knoten weniger hat.
- in beiden Fällen handelt es sich um rote Knoten, ändert an Schwarztiefe nichts -> dritte Forderung erfüllt

7.1.5 Einfüge-Fall 5

Der eingefügte Knoten hat einen schwarzen oder keinen Onkel und ist das linke Kind seines roten Vaters.

- Rechtsrotation um den Großvater
- der ursprüngliche Vater ist nun der Vater von sowohl dem neu einzufügenden Knoten als auch dem ehemaligen Großvater
- da der Vater rot war, muss nach der zweiten Forderung (Kein roter Knoten hat ein rotes Kind) der Großvater schwarz sein
- Vertauschung der Farben des ehemaligen Großvaters bzw. Vaters -> zweite Forderung wieder erfüllt
- dritte Forderung bleibt erfüllt, da alle Pfade, die durch einen dieser drei Knoten laufen, vorher durch den Großvater liefen und nun alle durch den ehemaligen Vater laufen, der inzwischen der einzige schwarze der drei Knoten ist

19 Balancieren Ende

Übersicht Ende

7.2 Einfügen Beispiel

→ Anzeichen – siehe Einfügebeispiel

8. Löschen

Roten Knoten löschen:

- so kann man diesen durch sein Kind ersetzen
- da Vater des gelöschten Knotens ebenfalls schwarz sein muss wird die zweite Forderung somit nicht mehr verletzt

- alle Pfade, die ursprünglich durch den gelöschten roten Knoten verliefen, nun durch einen roten Knoten weniger verlaufen, ändert sich an der Schwarztiefe ebenfalls nichts, womit die dritte Forderung erfüllt bleibt.

Schwarzen Knoten löschen:

- der zu löschende Knoten ist schwarz, aber ein rotes Kind
- in diesem Fall wird einfach der schwarze Knoten gelöscht
- dadurch könnte sowohl die zweite als auch die dritte Forderung verletzt werden
- das Kind wird schwarz gefärbt
- Somit treffen garantiert keine zwei roten Knoten aufeinander und alle Pfade, die durch den gelöschten schwarzen Knoten verliefen, verlaufen nun durch sein schwarzes Kind, wodurch beide Forderungen erfüllt bleiben.

8.1 Balancieren nach Löschen

Betrachten wir uns die 6 Löschen-Falle

8.1.1 Fall 1

Der Konfliktknoten ist die neue Wurzel.

- Eigenschaften bleiben erhalten
- Da schwarzer Knoten von jedem Pfad entfernt wurde

8.1.2 Fall 2

Der Bruder des Konfliktknotens ist rot

- die Farben des Vaters und des Bruders des Konfliktknotens invertieren

- anschließend eine Linksrotation um den Vaterknoten
- Bruder des Konfliktknotens wird dessen Großvater
- Alle Pfade haben weiterhin dieselbe Anzahl an schwarzen Knoten
- der Konfliktknoten hat nun einen schwarzen Bruder und einen roten Vater -> weswegen man nun zu Fall 4, 5, oder 6 weitergehen kann.

8.1.3 Fall 3

Der Vater des Konfliktknotens, sein Bruder und dessen Kinder sind schwarz.

- Bruder wird rot eingefärbt
- wodurch alle Pfade die durch diesen Bruder führen einen schwarzen Knoten weniger haben
- die ursprüngliche Ungleichheit ist wieder ausgeglichen
- alle Pfade, die durch den Vater laufen haben nun einen schwarzen Knoten weniger als jene Pfade die nicht durch den Vater laufen
- die dritte Forderung wird verletzt
- Versuch den Vaterknoten zu reparieren, indem man versucht, einen der sechs Fälle – angefangen bei Fall 1 – anzuwenden

8.1.4 Fall 4

Der Vater des Konfliktknotens ist rot, sein Bruder und dessen Kinder sind schwarz

- die Farben des Vaters und des Bruders tauschen
- die Anzahl der schwarzen Knoten auf den Pfaden bleibt unverändert

- fügt aber einen schwarzen Knoten auf allen Pfaden, die durch den Konfliktknoten führen, einen schwarzen Knoten hinzu
- gleicht somit den gelöschten schwarzen Knoten auf diesen Pfaden aus.

8.1.5 Fall 5

Das linke Kind des Bruders ist rot, das rechte Kind und der bruder des Konfliktknotens sind schwarz

- eine Rechtsrotation um den Bruder ausführen
- das linke Kind (SL) des Bruders wird dessen neuer Vater und damit der Bruder des Konfliktknotens
- vertauscht man die Farben des Bruders und seines neuen Vaters
- alle Pfade immer noch die gleiche Anzahl an schwarzen Knoten
- aber Konfliktknoten hat einen schwarzen Bruder dessen rechtes Kind rot ist

8.1.6 Fall 6

Der Bruder des Konfliktknotens ist schwarz und das rechte/linke Kind des Bruders ist rot.

- Linksrotation um den Vater des Konfliktknotens ausführen, so dass S Großvater von N wird
- die Farben von S und P zu tauschen und SR schwarz zu färben
- N hat immer noch dieselbe Farbe, wodurch die zweite Forderung erfüllt bleibt
- Aber N hat nun einen weiteren schwarzen Vorfahren:

- Falls P vor der Transformation noch nicht schwarz war, so ist er nach der Transformation schwarz
- falls P schon schwarz war, so hat der Konfliktknoten nun S als schwarzen Großvater, weswegen die Pfade, welche durch den Konfliktknoten laufen, nun einen zusätzlichen schwarzen Knoten passieren.

- In beiden Fällen ändert sich die Anzahl der schwarzen Knoten auf den Pfaden nicht: die zweite Forderung ist wiederhergestellt.

29 Balancieren Ende

Übersicht – das war‘s mit den Fällen – kommen wir zu einem Beispiel.

8.2 Löschen Beispiel

-> anzeichnen – siehe Löschenbeispiel

9. Unterschied zu AVL Bäume

- AVL- und Rot-Schwarz-Bäume sind beides selbstbalancierende binäre Suchbäume (mathematisch sehr ähnlich).
- Ausbalancierungen sind unterschiedlich, laufen aber in konstanter Zeit.
- Unterschiedliche maximale Höhen bei n-Knoten.
- Der AVL-Baum ist stärker bzw. starrer balanciert als Rot-Schwarz-Bäume, welches zu langsameren Einfüge- und Löschoperationen aber schnellerem Suchen führt
- Die Worst-Case-Höhe des AVL-Baums kleiner als die des Rot-Schwarz-Baums, und zwar um den Faktor $c/2 \approx 0{,}720$.
- Jeder AVL-Baum kann ein Rot-Schwarz-Baum sein, aber nicht jeder Rot-Schwarz-Baum ist ein AVL-Baum.

- Der Speicherplatzbedarf ist praktisch identisch: 1 Bit zum Aufzeichnen der Farbe resp. der Balance.

10. Ende und Quellen

Vielen Dank – Unsere Präsentation ist unter http://mi12.de/rbtree downloadbar.