Dynamische Mengen

Eine dynamische Menge ist eine Datenstruktur, die eine Menge von Objekten verwaltet.

Jedes Objekt x trägt einen eindeutigen Schlüssel $key[x]$.

Die Datenstruktur soll mindestens die folgenden Operationen anbieten:

- $\textsc{Search}(S, k)$ testet, ob ein Element x mit $key[x] = k$ in S vorhanden ist, und liefert es ggf. zurück.
- $\textsc{Insert}(S, x)$ fügt Element x in S ein.
- $\textsc{Delete}(S, x)$ entfernt Element x aus S.

Die folgenden Operationen werden manchmal noch angeboten, falls es auf den Schlüsseln eine totale Ordnung gibt.

- $\textsc{Minimum}(S)$ und $\textsc{Maximum}(S)$ liefern das Element mit minimalem bzw. maximalem Schlüssel zurück.
- $\textsc{Successor}(S, x)$ liefert dasjenige Element y in S zurück, dessen Schlüssel minimal unter denjenigen mit $key[y] > key[x]$ ist. Dual dazu ist $\textsc{Predecessor}(S, x)$.

Realisierungen durch Bäume

Dynamische Mengen können durch binäre Suchbäume realisiert werden.

Notation für binäre Bäume:

- Wurzel des Baumes T gespeichert in $root[T]$.
- Jeder Knoten x hat Zeiger auf die Söhne $left[x]$ und $right[x]$, sowie auf den Vater $p[x]$ (ggf. $\textsc{nil}$).

Die Operationen

- $\textsc{Tree-Search}(T, k)$
- $\textsc{Tree-Insert}(T, x)$
- $\textsc{Tree-Delete}(T, x)$
- $\textsc{Tree-Minimum}(T)$ und $\textsc{Tree-Maximum}(T)$
- $\textsc{Tree-Successor}(T, x)$ und $\textsc{Tree-Predecessor}(T, x)$

haben sämtlich eine Laufzeit von $O(h)$, wobei h die Tiefe des Baumes T ist.

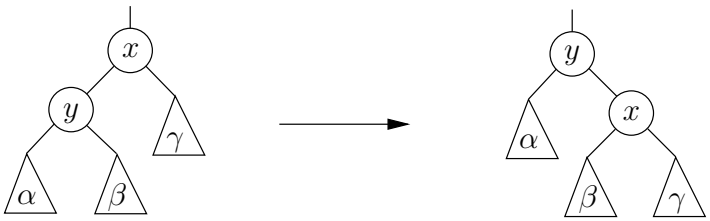
Balancierte binäre Suchbäume

Damit die Operationen auf binären Suchbäumen effizient sind, sollten diese balanciert sein, d.h. die Höhe eines Baumes mit n Knoten ist $O(\log n)$.

Bekanntes Beispiel balancierter Suchbäume: AVL-Bäume

Balancieren erfolgt durch Rotationsoperationen:

Operation ROTATE-RIGHT(T, x)



Dual dazu ($\leftarrow$) wirkt ROTATE-LEFT(T, y).

Rot-Schwarz-Bäume

Rot-Schwarz-Bäume: binäre Suchbäume mit zusätzlicher Struktur, die dafür sorgt, dass diese balanciert sind, also die Höhe höchstens $O(\log n)$ ist.

Jeder Knoten wird als innerer Knoten aufgefasst, Blätter sind zusätzliche, leere Knoten (Wächter).

Knoten haben ein zusätzliches Feld $color[x]$, mit den Werten RED und BLACK. Wir sprechen von roten und schwarzen Knoten.

Rot-Schwarz-Eigenschaft:

1. Die Wurzel und alle Blätter sind schwarz.
2. Beide Söhne eines roten Knotens sind schwarz.
3. Für jeden Knoten x gilt:
   jeder Pfad von x zu einem Blatt enthält gleich viele schwarze Knoten.

Eigenschaften von Rot-Schwarz-Bäumen

Wir zeigen, dass Rot-Schwarz-Bäume balanciert sind:

Satz:

Die Höhe eines Rot-Schwarz-Baums mit n inneren Knoten ist höchstens $2\log_2(n+1)$.

Daraus folgt:

Die Operationen Tree-Search, Tree-Minimum, Tree-Maximum, Tree-Successor und Tree-Predecessor benötigen für Rot-Schwarz-Bäume $O(\log n)$ Operationen.

Auch Tree-Insert und Tree-Delete laufen auf Rot-Schwarz-Bäumen in Zeit $O(\log n)$, erhalten aber nicht die Rot-Schwarz-Eigenschaft.

$\rightsquigarrow$ Für diese benötigen wir spezielle Einfüge- und Löschoperationen.

Einfügen in einen Rot-Schwarz-Baum

Neuer Knoten x wird mit Tree-Insert eingefügt und rot gefärbt.

Problem: Vater $p[x]$ kann auch rot sein $\rightsquigarrow$ Eigenschaft 2 verletzt.

Diese Inkonsistenz wird in einer Schleife durch lokale Änderungen im Baum nach oben geschoben:

- Fall 1: $p[x] = \mathit{left}[p[p[x]]]$:
  - Betrachte x' Onkel $y = \mathit{right}[p[p[x]]]$.
  - Fall 1.1: y ist rot. Dann färbe $p[x]$ und y schwarz, und $p[p[x]]$ rot. Aber: $p[p[p[x]]]$ kann rot sein $\rightsquigarrow$ Inkonsistenz weiter oben.
  - Fall 1.2: y ist schwarz, und $x = \mathit{left}[p[x]]$. Dann wird $p[x]$ schwarz und $p[p[x]]$ rot gefärbt, und anschliessend um $p[p[x]]$ nach rechts rotiert. $\rightsquigarrow$ keine Inkonsistenz mehr.
  - Fall 1.3: y ist schwarz, und $x = \mathit{right}[p[x]]$. Wird durch eine Linksrotation um $p[x]$ auf Fall 1.2 zurückgeführt.
- Fall 2: $p[x] = \mathit{right}[p[p[x]]]$ ist symmetrisch.

Einfügen in einen Rot-Schwarz-Baum

Am Ende wird die Wurzel $root[T]$ schwarz gefärbt.

Bemerke:

- In den Fällen 1.2 und 1.3 bricht die Schleife sofort ab.
- Schleife wird nur im Fall 1.1 wiederholt, dann ist die Tiefe von x kleiner geworden.
- Rotationen werden nur in 1.2 und 1.3 durchgeführt.

$\rightsquigarrow$ Laufzeit ist $O(\log n)$, und es werden höchstens 2 Rotationen durchgeführt.

Löschen aus einem Rot-Schwarz-Baum

Zu löschender Knoten z wird mit Tree-Delete(T, z) gelöscht.

Erinnerung: Tree-Delete(T, z) entfernt einen Knoten y, der höchstens einen (inneren) Sohn hat.

- Hat z höchstens einen Sohn, so ist $y = z$.
- Andernfalls $y \leftarrow$ Tree-Successor(T, z).

Problem: Wenn der entfernte Knoten y schwarz war, ist Eigenschaft 3 verletzt.

Intuition: der (einzige) Sohn x von y erbt dessen schwarze Farbe, und ist jetzt "doppelt schwarz".

Das zusätzliche Schwarz wird in einer Schleife durch lokale Änderungen im Baum nach oben geschoben.

Löschen aus einem Rot-Schwarz-Baum

- Ist x rot, so wird es schwarz gefärbt, und die Schleife bricht ab.
  Andernfalls unterscheide zwei Fälle:
- Fall 1: $x = \mathit{left}[p[x]]$.
  - Betrachte x' Bruder $w = \mathit{right}[p[x]] \neq \text{NIL}$, und unterscheide 2 Fälle:
  - Fall 1.1: w ist rot. Dann muss $p[x]$ schwarz sein, also färbe $p[x]$ rot, w schwarz und rotiere links um $p[x]$
    $\rightsquigarrow$ reduziert auf Fall 1.2.
  - Fall 1.2: w ist schwarz. Es gibt drei weitere Fälle:
    * Fall 1.2.1: Beide Kinder von w sind schwarz. Also können wir w rot färben, und das zusätzliche Schwarz von x zu $p[x]$ versetzen
      $\rightsquigarrow$ nächste Iteration.
    * Fall 1.2.2: $\mathit{left}[w]$ ist rot, $\mathit{right}[w]$ ist schwarz. Dann färbe w rot, $\mathit{left}[w]$ schwarz und rotiere rechts um w
      $\rightsquigarrow$ reduziert auf Fall 1.2.3.
    * Fall 1.2.3: $\mathit{right}[w]$ ist rot. Dann vertausche die Farben von w und $p[x]$, färbe $\mathit{right}[w]$ schwarz, und rotiere links um $p[x]$
      $\rightsquigarrow$ Zusätzliches Schwarz ist verbraucht.

Löschen aus einem Rot-Schwarz-Baum

- Fall 2: $x = \mathit{right}[p[x]]$ ist symmetrisch.

Auch beim Löschen wird am Ende die Wurzel $\mathit{root}[T]$ schwarz gefärbt.

Bemerke:

- In den Fällen 1.2.2 und 1.2.3 bricht die Schleife sofort ab.
- Schleife wird nur im Fall 1.2.1 wiederholt, dann ist die Tiefe von x kleiner geworden.
- Im Fall 1.1 wird die Tiefe von x größer, aber die Schleife bricht danach im Fall 1.2 sofort ab.

$\rightsquigarrow$ Laufzeit ist $O(\log n)$, es werden höchstens 3 Rotationen durchgeführt.

Zusammengefasst: die Operationen Search, Minimum, Maximum, Successor, Predecessor, Insert und Delete können für Rot-Schwarz-Bäume mit Laufzeit $O(\log n)$ realisiert werden.