Programmiertechnik II

Bäume

© 2009 Martin v. Löwis

Symboltabellen

- Suche nach Werten (items), die unter einem Schlüssel (key) gefunden werden können
  - Bankkonten: Schlüssel ist Kontonummer
  - Flugreservierung: Schlüssel ist Flugnummer, Reservierungsnummer, ...
- Symboltabelle: Abstrakter Datentyp
  - Einfügen von neuen Werten unter einem Schlüssel
    - Optional Ändern des Wertes, welcher bereits unter diesem Schlüssel abgelegt ist
  - Auffinden/Ermitteln des Wertes, welches unter einem Schlüssel gespeichert ist
  - Optional: Löschen eines Schlüssels
  - Optional: Ermittlung aller Schlüssel-Wert-Paare
  - Anderer Name: Dictionary, Mapping
  - Charakteristika: Einfügen kommt weit häufiger vor als bei gedrucktem Wörterbuch

© 2009 Martin v. Löwis

Symboltabellen als Felder/Listen

- Strategie 1: Unsortiertes Feld von Schlüssel-Wert-Paaren
  - Suche in linearer Zeit (lineare Suche)
  - Einfügen evtl. in konstanter Zeit
    - aber: Vergrößerung des Felds
- Strategie 2: Sortiertes Feld
  - Einfügen in linearer Zeit
  - Binäre Suche: logarithmische Zeit
- Strategie 3: Indiziertes Feld
  - Spezialfall von Zahlen als Schlüsseln
  - Schlüssel ist Index
  - Einfügen/Suchen in konstanter Zeit
  - Speicherbedarf linear mit Wertebereich

© 2009 Martin v. Löwis

Binäre Suchbäume

- Ziel: besser-als-linear für Einfügen und Suchen
- Binärer Suchbaum:
  - Binärer Baum: Knoten mit zwei Kindern
    - Kindknoten sind eventuell nicht vorhanden (*null*)
  - Jeder Knoten enthält Schlüssel-Wert-Paar
  - Schlüssel im linken Teilbaum alle kleiner, Schlüssel im rechten Teilbaum alle größer
    - doppelte Schlüssel?
- Suche: beginnend bei Wurzel, rekursiv
- Komplexität der Suche gleich Tiefe des Baums
  - best case: lg N (vollständiger Binärbaum)
  - average case (Gleichverteilung der Schlüssel): 2 ln N (≈ 1.39 lg N)
  - worst case: N (entarteter Baum: Liste)

© 2009 Martin v. Löwis

Einfügen in Binäre Suchbäume

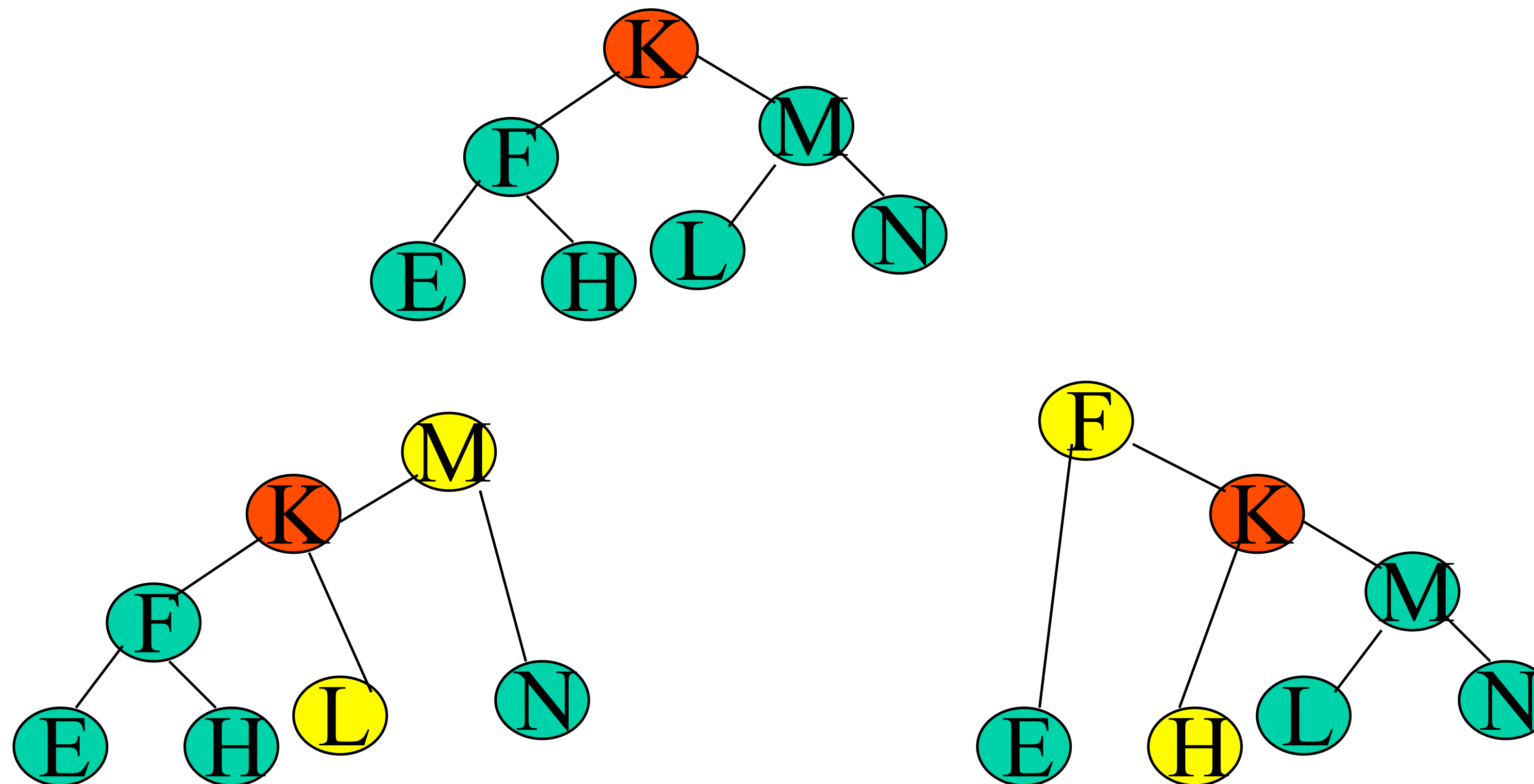
- naive Lösung: neue Knoten werden immer als Blätter (ohne Kindknoten) eingefügt
  - Suche beginnend bei Wurzel
  - Absteigen zu Kindknoten entsprechend Ordnung, bis Kindzeiger null
  - Problem: Baum kann entarten
- balanzierte Bäume: Nach Einfügen wird Baum wieder ausgeglichen

© 2009 Martin v. Löwis

Rotation in Bäumen

- Linksrotation: Wurzel wird linkes Kind

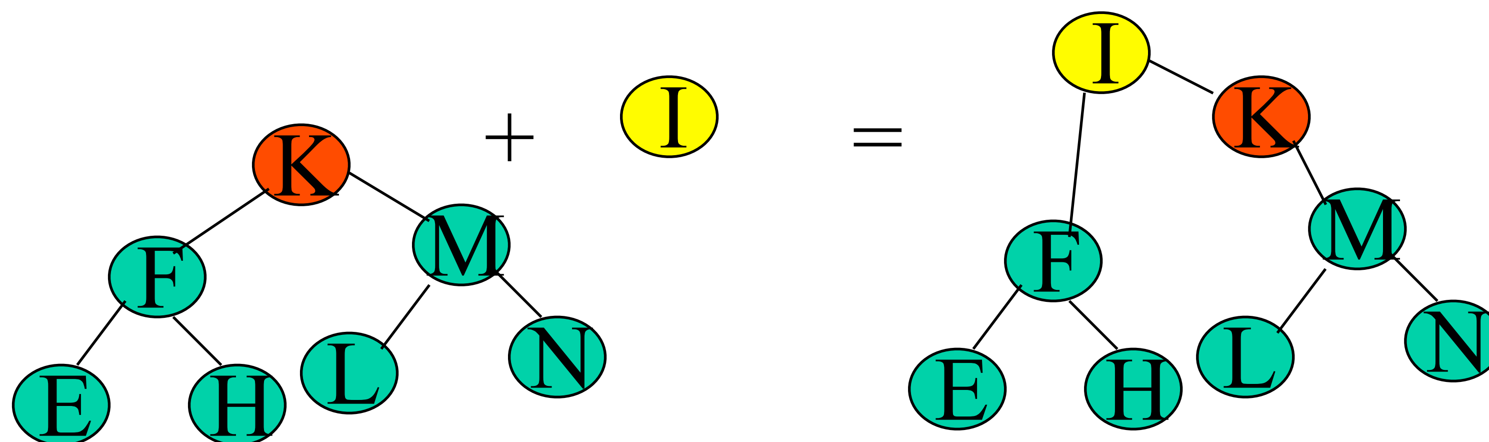
- Rechtsrotation: Wurzel wird rechtes Kind



© 2009 Martin v. Löwis

Einfügen in Wurzel

- naives Einfügen: neuer Knoten als Blatt
- Einfügen in Wurzel: neuer Knoten wird Wurzel
  – Suche nach frisch eingefügten Schlüsseln geht schneller
- Problem: Binäre Ordnung wird potentiell verletzt
- Lösung: Rotieren des Baums nach Einfügen in Teilbaum
  – Rechtsrotation nach Einfügen in linken Teilbaum
  – Linksrotation nach Einfügen in rechten Teilbaum



© 2009 Martin v. Löwis

Einfügen in Wurzel (2)

```
static Node insert(Node h, ITEM x)
{
 if (h==null) return new Node(x);
 if (less(x.key(), h.item.key())) {
   h.l = insert(h.l, x); h = rotR(h);
 } else {
   h.r = insert(h.r, x); h = rotL(h);
 }
 return h;
}
```

© 2009 Martin v. Löwis

Balanzierte Bäume

- Ziel: entartete Bäume sollen vermieden werden
- Lösung 1: probabilistischer Algorithmus
  – Schlüssel werden in randomisierter Reihenfolge eingefügt
- Lösung 2: amortisierende Algorithmen
  – einzelne Einfügeoperationen eventuell teuer durch Neubalanzierung
  – Mittelwert für viele Operationen soll “günstig” sein
- Lösung 3: optimale Algorithmen
  – Performancegarantie für jede Operation
  – Buchhaltung über Struktur des Baums erforderlich

© 2009 Martin v. Löwis

Splay Trees

- Erfunden von Sleator und Tarjan 1985
- Umordnung des Baums sowohl beim Einfügen als auch beim Suchen
  - gesuchter Schlüssel wird stets Wurzel des Baums
  - to splay: "spreizen, teilen"
- Umsortieren durch Rotieren
  - Knoten N, Elternknoten P, Großelternknoten G

  2. N links von P, links von G: doppelte Rechtsrotation
  3. N rechts von P, rechts von G: doppelte Linksrotation
  4. N links von P, rechts von G: rechts-dann-links
  5. N rechts von P, links von G: links-dann-rechts

– Amortisierte Komplexität: M Operationen (Einfügen oder Suchen) auf einem Baum mit N Knoten benötigen O((N+M) log (N+M))
  - Worst-case für eine einzelne Operation: O(N+M)

© 2009 Martin v. Löwis

2-3-4-Bäume

- Drei Arten von Knoten
  - 2-Knoten: Ein Schlüssel, zwei Kindknoten
  - 3-Knoten: Zwei Schlüssel, drei Kindknoten
  - 4-Knoten: Drei Schlüssel, vier Kindknoten
- Balanzierter 2-3-4-Baum: Alle Blätter haben den gleichen Abstand von der Wurzel
- Einfügen in balanzierten Baum erhält Balanz
  - Einfügen neuer Schlüssel immer in Blättern
  - Einfügen in 2-Blatt erzeugt 3-Blatt
  - Einfügen in 3-Blatt erzeugt 4-Blatt
  - Zerlegen von 4-Knoten auf dem Weg von der Wurzel
    - 2-Knoten mit 4-Knoten als Kind wird zu 3-Knoten mit 2 weiteren 2-Kindern
    - 3-Knoten mit 4-Knoten als Kind wird zu 4-Knoten mit 2 weitern 4-Kindern
    - Ist die Wurzel ein 4-Knoten, wird sie in 2-Knoten zerlegt – Tiefe des Baums steigt

© 2009 Martin v. Löwis

2-3-4-Bäume: Analyse

- Baum mit N Knoten enthält weniger als 3N Schlüssel
- Suchen in Baum mit N Knoten: höchstens lg N + 1 Knoten werden inspiziert
- Einfügen in Baum mit N Knoten: höchstens lg N + 1 Teilungen
  - average case bisher nicht analysiert
  - Vermutung: im Mittel weniger als eine Teilung pro Einfügeoperation

© 2009 Martin v. Löwis

Rot-Schwarz-Bäume

- erfunden von Bayer 1972
- Binärer Suchbaum
- Knoten haben “Farbe”: rot oder schwarz
  - oft repräsentiert in einem Bit
  - Darstellung der Farbe entweder in Knoten oder in Verweis auf den Knoten
- Wurzel ist immer schwarz
- Jeder Pfad im Baum hat gleich viele schwarze Knoten
- Jeder rote Knoten hat nur schwarze Kinder
  - Gesamttiefe des Baums höchstens 2x Zahl der schwarzen Knoten
- Interpretation des 2-3-4-Baums als Rot-Schwarz-Baum
  - Knoten mit schwarzen Kindern: 2-Knoten
  - Knoten mit einem roten Kind: 3-Knoten
  - Knoten mit zwei roten Kindern: 4-Knoten

© 2009 Martin v. Löwis

AVL-Bäume

- Г. М. Адельсон-Вельский, Е. М. Ландис. Один алгоритм организации информации
  - Доклады АН СССР. 1962. Т. 146, № 2. С. 263–266.
  - Adelson-Velsky, Landis
- Forderung: Höhe des linken Teilbaums unterscheidet sich von Höhe des rechten Teilbaums höchstens um 1
  - Speicherung der Höhendifferenz im Elternknoten (-1/0/+1)
- Neubalancierung nach Verletzung der AVL-Bedingung:
  - Höhendifferenz ist +/-2 nach Einfügen oder Löschen
  - betrachte wieder drei Knoten N, P, G, so, dass N tieferes Kind (+1) von P und P tieferes Kind von G
  - N links von P links von G: Rechtsrotation von G
  - N rechts von P links von G: Linksrotation von P, dann Rechtsrotation von G
  - andere 2 Fälle symmetrisch

© 2009 Martin v. Löwis

AVL-Bäume (2)

- Einfügen
  - füge in Blatt ein
  - danach: aktualisiere Höhenangaben, rebalanziere
- Löschen
  - eines Blatts: entferne Blatt
  - eines inneren Knotens
    - finde wahlweise kleinsten folgenden Schlüssel (kleinsten Knoten im rechten Kind) oder größten vorhergenden Schlüssel; gefundener Knoten hat höchstens einen Kindknoten
    - ersetze zu löschenden Knoten mit gefundenem
  - danach: aktualisiere Höhenangaben, rebalanziere

© 2009 Martin v. Löwis

- Aktualisierung von Höhenangaben:
  - jede Operation (Einfügen, Löschen) meldet Höhenänderung
    - Höhenunterschied 0 kann sofort "nach oben" gemeldet werden
  - addiere zu aktuellem Höhenunterschied
  - +/-2: rebalanziere
  - Einfügen: Höhenunterschied +/-1 bedeutet insgesamt Höhenzunahme
  - Löschen: Höhenunterschied 0 bedeutet Höhenverlust
- Komplexität
  - worst-case für Tiefe des Baums: ca 1.44 log N
  - Suche: O(log N)
  - Einfügen: maximal 1 Doppelrotation nötig; O(log N)
  - Löschen: im worst case 1 Rebalanzierung pro Knoten auf Pfad; O(log N)

© 2009 Martin v. Löwis