Algorithmen und Datenstrukturen (für ESE) WS 2010 / 2011

Vorlesung 7, Montag, 6. Dezember 2010
(Binäre Suchbäume)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

- Organisatorisches
  - Morgen gibt es wieder eine Übungsgruppe (Ü5: Hashing)
  - Ihre Erfahrungen mit dem 6. Übungsblatt
  - Anwesenheit in der Vorlesung
- Binäre Suchbäume (binary search trees)
  - Was ist das?
  - Wofür braucht man das?
  - Übungsaufgabe: schreiben Sie eine Klasse BinarySearchTree

Ihre Erfahrungen mit dem 6. Ü-Blatt

UNI FREIBURG

- Zusammenfassung von Ihrem Feedback
  - Das Blatt fanden die meisten gut und einfacher als das letzte
  - Prioritätswarteschlange wurde gut erklärt
  - Die meiste Zeit ging wie immer für Fehlersuche drauf
  - Der Teufel steckt im Detail
  - Den meisten macht Programmieren (immer mehr) Spaß

Sortierte Folgen

- Problem
  - Wir wollen wieder (key, value) Paare / Elemente verwalten
  - Wir haben wieder eine Ordnung **<** auf den Keys
  - Diesmal wollen wir folgende Operationen unterstützen
    - insert(key, value): füge das gegebene Paar ein
    - remove(key): entferne das Paar mit dem gegebenen Key
    - lookup(key): finde das Element mit dem gegebenen Key; falls es das nicht gibt, finde das Elemente mit dem kleinsten Key der > key ist
    - next / previous: für ein gegebenes Element, finde das mit dem nächstgrößeren / nächstkleineren Schlüssel; damit lässt sich insbesondere über alle Elemente iterieren

Wo braucht man das?

- Typisches Anwendungsbeispiel: Datenbanken
  - Eine große Menge von Records
  - Zum Beispiele Bücher, Produkte, Wohnungen, ...
  - Typische Suchanfrage: alle Wohnungen zwischen 600 und 800 Euro Monatsmiete
    - Ein sogenannter **range query (Bereichsanfrage)**
    - Das bekommt man mit lookup und next
    - Man beachte: es ist dafür nicht wichtig, dass es eine Wohnung gibt, die **genau** 600 Euro kostet
  - Wenn man ein paar records hinzufügt oder alte löscht, will man nicht jedes Mal erst alles wieder neu sortieren

Lösung 1 (schlecht): Einfache Arrays

- Mit einem einfachen Array bekommen wir
  - lookup in Zeit O(log n)
    - das geht mit binärer Suche
  - next und previous in Zeit O(1)
    - klar, sie stehen ja direkt nebeneinander
  - insert und remove in Zeit bis zu Θ(n)
    - bis zu Θ(n) Elemente müssen umkopiert werden

Binäre Suche Beispiel:

2 5 7 11 | 17 25 | 31 | 37

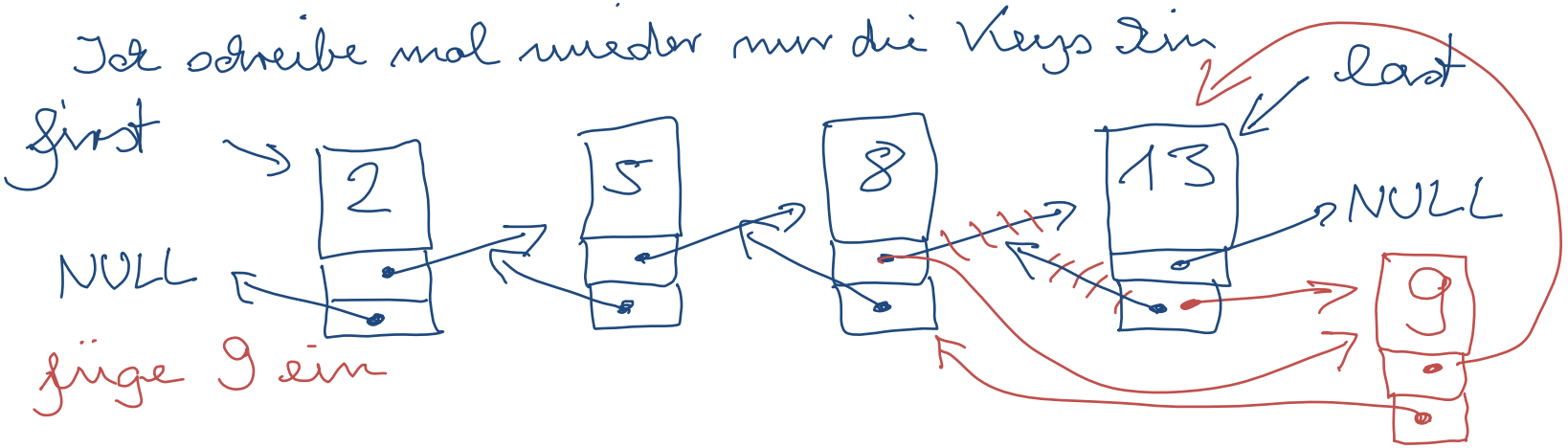
Suche 35

Lösung 2 (schlecht): Hashtabellen

- Mit einer Hashtabelle bekommt man
  - insert und remove in erwarteter Zeit $O(1)$
    - bei genügend großer Hashtabelle und guter Hashfunktion
  - lookup in erwarteter Zeit $O(1)$
    - aber nur wenn es ein Element mit dem Key gibt, sonst bekommt man gar nichts
  - next und previous in Zeit bis zu $\Theta(n)$
    - die Reihenfolge, in der die Elemente in einer Hashtabelle stehen hat nichts mit der Reihenfolge der Keys zu tun!

Lösung 3 (schlecht): Verkettete Listen

- Mit einer doppelt verketteten Liste bekommt man
  - next und previous in Zeit O(1)
    - jedes Element hat einen Zeiger zum Vorgänger / Nachfolger
  - insert und remove in Zeit O(1)
    - es müssen nur konstant viele Zeiger umgesetzt werden
  - lookup in Zeit bis zu Θ(n)
    - die Elemente stehen jetzt nicht mehr sortiert in einem Feld; man muss sie sich im schlechtesten Fall alle anschauen

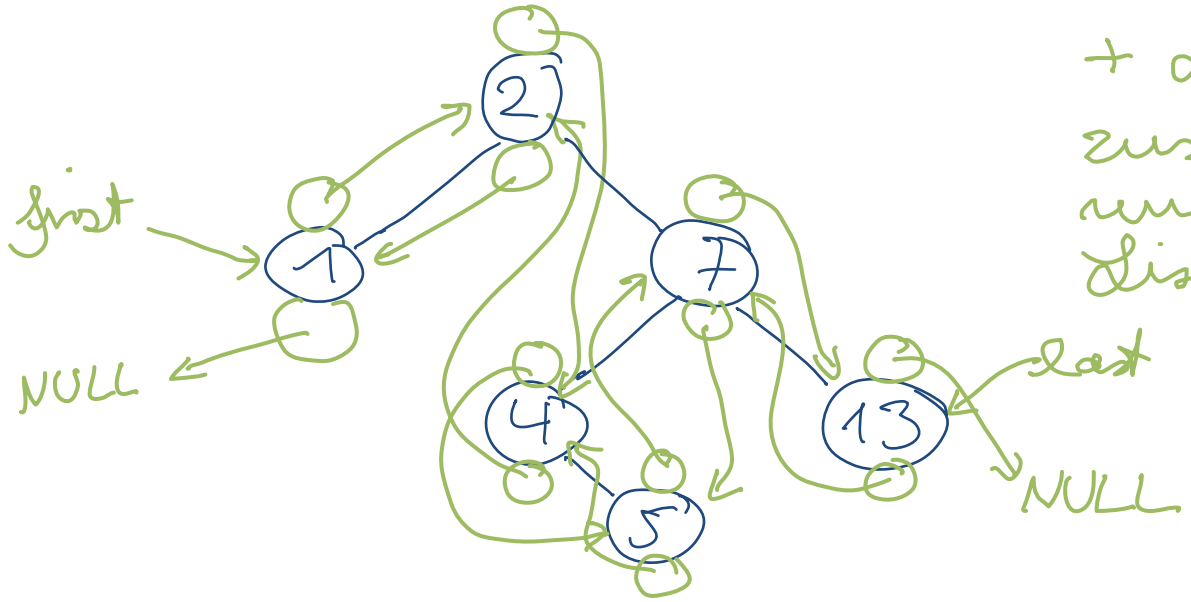


Lösung 4 (gut): Suchbäume

- Mit einem geeigneten Suchbaum bekommt man
  - next und previous in Zeit O(1)
    - entsprechende Zeiger wie bei der verketteten Liste
  - insert und remove in Zeit O(1)
    - ebenfalls wie bei der verketteten Liste
  - lookup in Zeit O(log n)
    - eine Baumstruktur hilft jetzt beim effizienten Suchen

Binäre Suchbäume — Idee

- Anordung ähnlich wie bei der Prioritätswarteschlange
  - Aber jetzt ganz sortiert!
  - Für jeden Knoten gilt: alle Elemente im linken Unterbaum haben einen kleineren Key + alle Elemente im rechten Unterbaum haben einen größeren Key
    - wir nehmen der Einfachheit halber an, dass alle Keys verschieden sind, es geht aber auch ohne diese Bedingung

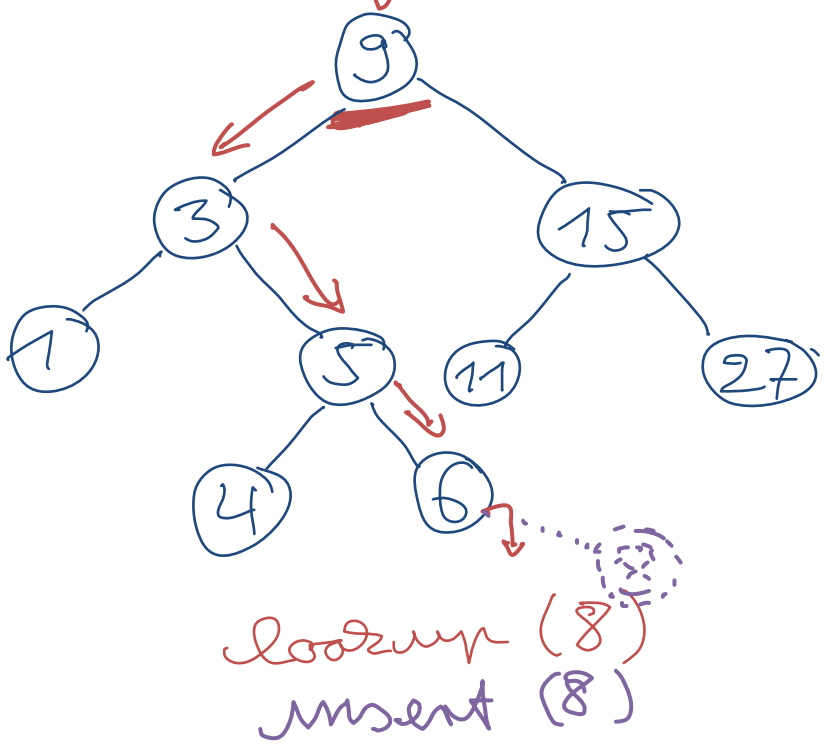


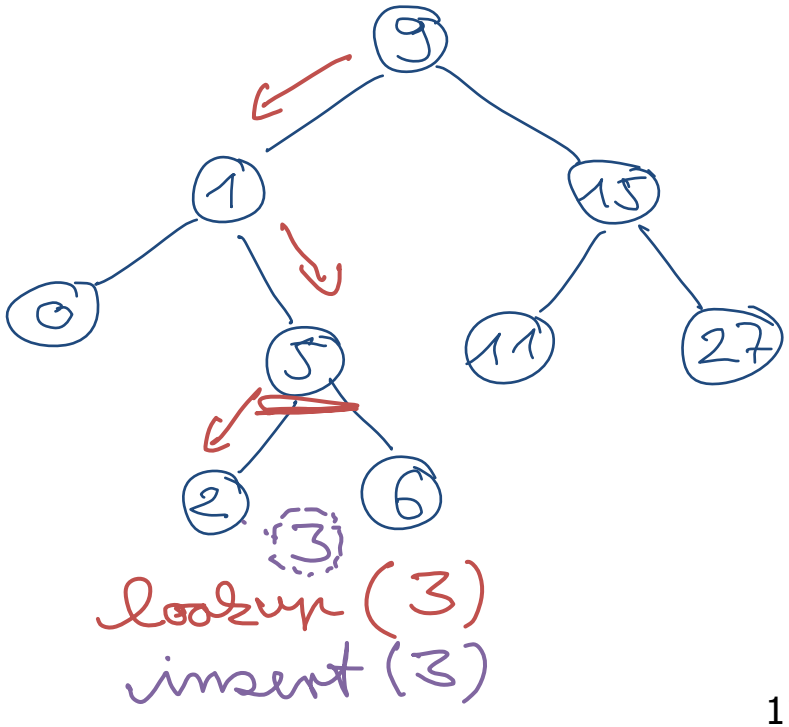
+ die Elemente sind zusätzlich verkettet wie in einer verketteten Liste

Binäre Suchbäume — Lookup

- Wir suchen einfach von der Wurzel abwärts
  - und gehen je nach Key links oder rechts
  - und merken uns dabei immer den letzten Knoten, wo wir nach links gegangen sind

überlegen warum!





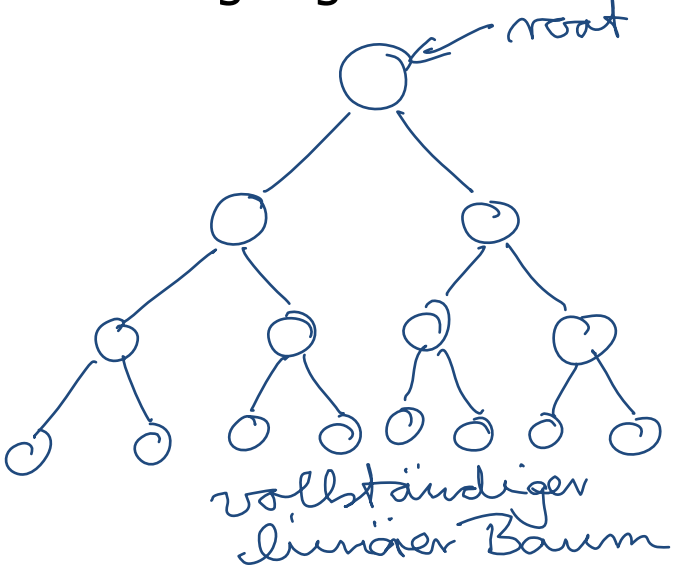
Binäre Suchbäume — Einfügen

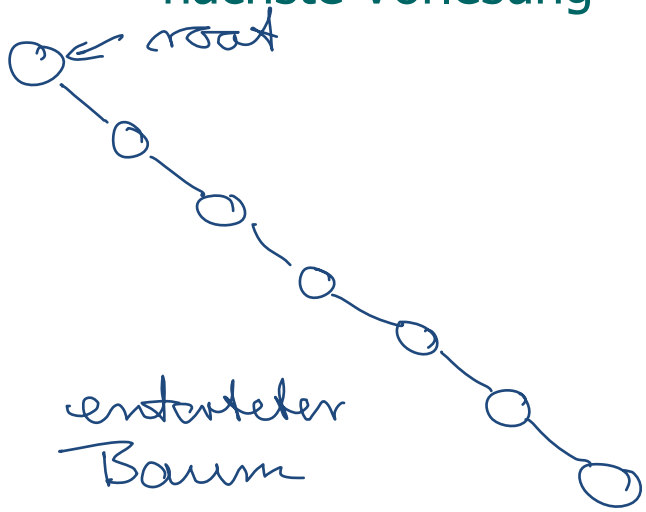
- Wir suchen erstmal den gegebenen Key
  - Wenn wir ihn gefunden haben, überschreiben wir einfach das Value an dem Knoten
  - Sonst fügen wir an geeigneter Stelle einen neuen Knoten ein, hier sind zwei Beispiele

siehe vorherige Folie!

Binäre Suchbäume — Komplexität

- Wie lange dauern insert und lookup ?
  - Bis zu Zeit $\Theta(d)$, wobei d die Tiefe des Baumes ist = die größte Tiefe eines Blattes
  - Im besten Fall ist das $\Theta(\log n)$, im schlechtesten Fall $\Theta(n)$, wobei n die Anzahl der Knoten im Baum ist
  - Wenn man immer $\Theta(\log n)$ will, muss man den Baum gelegentlich **rebalancieren** → nächste Vorlesung





Literatur / Links

- Suchbäume
  - In Mehlhorn/Sanders:

    7 Sorted Sequences
  - In Cormen/Leiserson/Rivest

    13 Binary Search Trees
  - In Wikipedia

    http://de.wikipedia.org/wiki/Binärer_Suchbaum

    http://en.wikipedia.org/wiki/Binary_search_tree
  - In C++ / Java
    - die Klasse Map ist typischerweise mit Suchbäumen implementiert