Datenstrukturen, Algorithmen und Programmierung 2

Amin Coja-Oghlan

June 20, 2022

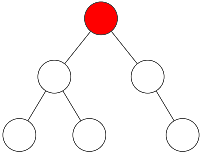
Lehrstuhl Informatik 2
Fakultät für Informatik

Binäre Suchbäume

Worum geht es?

- ein binärer Suchbaum ist eine Datenstruktur zur Speicherung von Objekten, die durch einen Schlüssel gekennzeichnet sind
- die Schlüssel sind total geordnet
- wir nehmen an, daß alle Schlüssel verschieden sind
- Operationen: `Insert`, `Delete`, `Search`, `Minimum`, `Maximum`, `Predecessor`, `Successor`
- die Laufzeit jeder Operation ist O(Höhe des Baums)

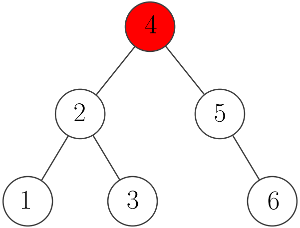
Binäre Suchbäume



Binärbäume

- ein Binärbaum ist ein gewurzelter Baum, d.h. ein Baum T mit einer ausgezeichneten Wurzel $r \in V(T)$
- die Wurzel hat Grad $d_T(r) \leq 2$
- jeder andere Knoten $v \in V(T) \setminus \{r\}$ hat Grad ≤ 3
- die Kinder eines Knotens v sind die Nachbarn $w \in \partial v$, die nicht auf dem kürzsten Pfad von v nach r liegen; jeder Knoten hat also höchstens zwei Kinder
- der Elternknoten von v ist der Nachbar auf dem kürzesten Pfad zu r, bzw. $\emptyset$ falls $v = r$

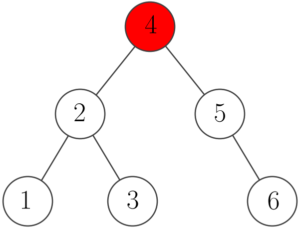
Binäre Suchbäume



Suchbäume

- die Knoten tragen vergleichbare Schlüssel $s(v)$
- sei v ein Knoten
- dann hat v höchstens ein Kind x mit $s(x) < s(v)$
- ferner hat v höchstens ein Kind y mit $s(y) > s(v)$
- wir nennen x das linke Kind von v und y das rechte Kind

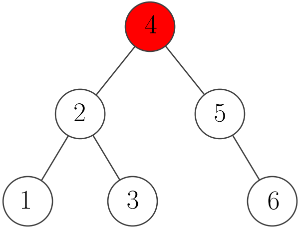
Binäre Suchbäume



Suchbäumeigenschaft

- der Knoten x und seine Kinder bilden den linken Unterbaum von v
- der Knoten y und seine Kinder bilden den rechten Unterbaum von v
- Suchbaumeigenschaft: für alle Knoten u im linken Unterbaum von v gilt $s(u) < s(v)$; für alle Knoten w im rechten Unterbaum von v gilt $s(v) < s(w)$

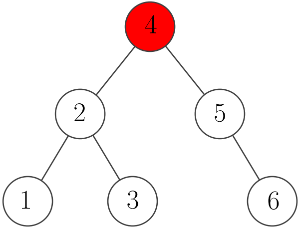
Binäre Suchbäume



Implementation

- jeder Knoten des Suchbaums enthält den Schlüssel und ggf. einen Zeiger auf das Objekt, das dieser Knoten repräsentiert
- jeder Knoten enthält einen Zeiger auf seinen Elternknoten (evtl. $\emptyset$)
- jeder Knoten enthält einen Zeiger auf das linke und einen auf das rechte Kind (ggf. $\emptyset$)

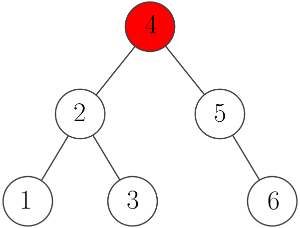
Binäre Suchbäume



Geordnete Ausgabe

- wir können die Elemente des Suchbaums ausgeben, indem wir von der Wurzel aus den Baum in Tiefensuchordnung durchlaufen
- dabei wird immer zuerst das linke Kind aufgesucht, wenn eines existiert

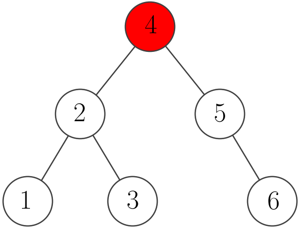
Binäre Suchbäume



Minimum finden

- um das Element mit minimalem Schlüssel zu finden, folgen wir von der Wurzel aus stets dem Zeiger auf das linke Kind
- der erste Knoten, dessen linkes Kind $\emptyset$ ist, ist das Minimum

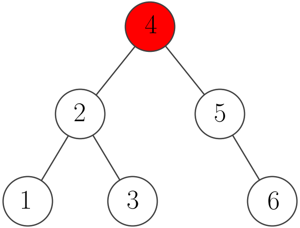
Binäre Suchbäume



Maximum finden

- um das Element mit maximalem Schlüssel zu finden, folgen wir von der Wurzel aus stets dem Zeiger auf das rechte Kind
- der erste Knoten, dessen rechtes Kind $\emptyset$ ist, ist das Maximum

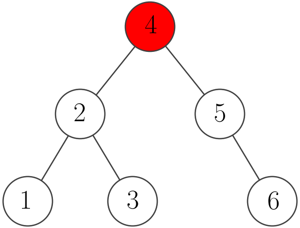
Binäre Suchbäume



Element mit einem gegebenem Schlüssel suchen

- die Operation `Search` erhält als Eingabe einen Schlüssel σ und sucht das Element mit diesem Schlüssel
- von der Wurzel $v = r$ aus wiederhole folgendes Verfahren

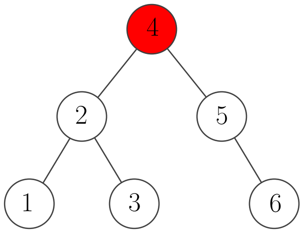
Binäre Suchbäume



Element mit einem gegebenem Schlüssel suchen

- falls $s(v) = \sigma$, gib v aus
- falls $s(v) > \sigma$, setze v auf das linke Kind u von v; falls $u = \emptyset$, gib “nicht vorhanden” aus
- falls $s(v) < \sigma$, setze v auf das rechte Kind w von v; falls $w = \emptyset$, gib “nicht vorhanden” aus

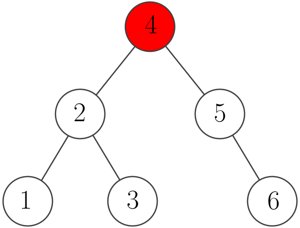
Binäre Suchbäume



`Successor`: **gegeben v finde z mit minimalem $s(z) > s(v)$**

- falls v ein rechtes Kind w hat, finde das Minimum im rechten Teilbaum
- sonst setze w auf den Elternknoten von v
- solange $w \neq \emptyset$ und v das rechte Kind von w ist, setze $v = w$ und w =Elternknoten von v
- gib schließlich w aus

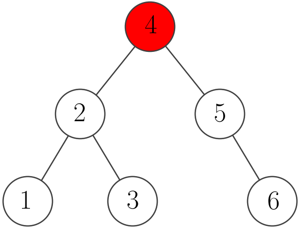
Binäre Suchbäume



`Predecessor`: **gegeben v finde z mit maximalem $s(z) < s(v)$**

- falls v ein linkes Kind w hat, finde das Maximum im linken Teilbaum
- sonst setze w auf den Elternknoten von v
- solange $w \neq \emptyset$ und v das linke Kind von w ist, setze $v = w$ und w =Elternknoten von v
- gib schließlich w aus

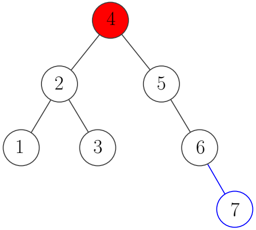
Binäre Suchbäume



Laufzeiten

- die Höhe $H(T)$ von T ist der maximale Abstand von r zu einem Blatt
- alle vorgenannten Operationen haben Laufzeit $O(H(T))$

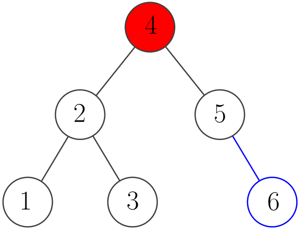
Binäre Suchbäume

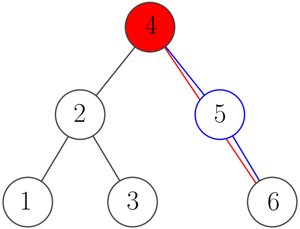


Einfügen

- um ein Element e mit einem gegeben Schlüssel $s(e)$ einzufügen, gehen wir so vor, als würden wir den Baum nach e durchsuchen
- weil wir annehmen, daß $s(e)$ nicht im Baum vorkommt, finden wir dabei schließlich einen $\varnothing$-Zeiger
- diesen ersetzen wir durch das neue Element

Binäre Suchbäume

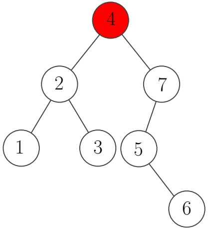


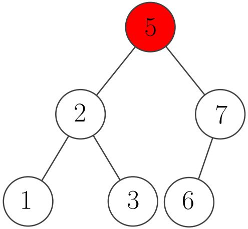


Entfernen eines Knotens v

- falls v kein Kind hat, wird v einfach gelöscht
- falls v nur ein Kind hat, nimmt dieses Kind die Position von v ein

Binäre Suchbäume





Entfernen eines Knotens v

- falls v zwei Kinder hat, finde den Nachfolger w von z; w hat nur ein Kind x
- w nimmt die Stelle von v ein, und x tritt an die Stelle von w

Binäre Suchbäume

Laufzeit einfügen/entfernen

- beide Operationen haben Laufzeit $O(H(t))$

Binäre Suchbäume

Zusammenfassung

- binäre Suchbäume erlauben effiziente Operationen, wenn $H(T)$ “klein” ist
- die optimale Höhe bei n Knoten ist $H(T) = O(\log n)$
- mit den beschriebenen Operationen kann jedoch der Baum zu einem Pfad “degenerieren”
- dann sind die Operationen nicht effizienter als bei einer verketteten Liste