Algorithmen und Datenstrukturen 1

ALGO1 · SoSe-2021 · tcs.uni-frankfurt.de/algo1

Wochenplan: Traversierung, Binäre Suchbäume

Revision 517827a

(2021-07-01)

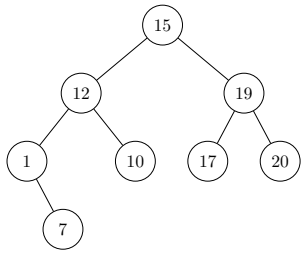
Vorbereitung

Lies CLRS Kapitel 12 ohne 12.4 und schau das Video der Woche.

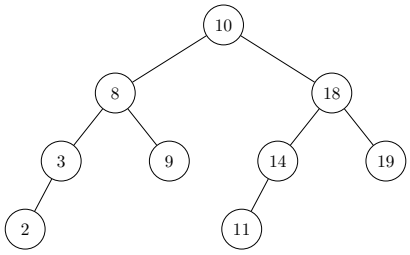
Dienstag

Aufgabe 1 (Binärbaumeigenschaften).

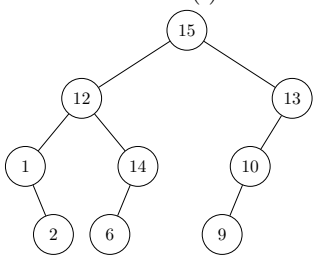
a) (einfach) Welche der folgenden Bäume sind binäre Suchbäume?



Baum (i)



Baum (ii)



Baum (iii)

b) (einfach) Wo in einem binären Suchbaum befinden sich die Elemente mit dem kleinsten und größten Schlüssel?

c) Betrachte die Schlüsselmenge $\{1, 4, 5, 10, 16, 17, 21\}$. Zeichne binäre Suchbäume der Höhe 2, 3, 4, 5 und 6, die jeweils genau diese Schlüssel enthalten.

d) (einfach) Gib die Reihenfolge an, in der die Knoten von Baum (ii) in inorder, preorder und postorder traversiert werden.

e) Vergleiche die Heap-Eigenschaft und die Suchbaum-Eigenschaft.

f) Schreibe Pseudocode für eine iterative Variante der Inorder-Traversierung.

g) Sei T ein binärer Suchbaum, in dem alle Schlüssel verschieden sind. Beweise die folgende Aussage mit einem Widerspruchsbeweis: Wenn ein Knoten v zwei Kinder hat, dann hat das Element mit dem nächstgrößeren Schlüssel kein linkes Kind und das Element mit dem nächstkleineren Schlüssel kein rechtes Kind.

Aufgabe 2 (AVL-Bäume).

a) (einfach) Füge die Elemente $6, 5, 2, 1, 3, 4$ in dieser Reihenfolge in einen zunächst leeren AVL-Baum ein. Zeichne den Baum nach jeder Einfügung.

b) (einfach) Schreib den fehlenden Pseudocode für `Rebalance`(y) im Fall, dass der Balancefaktor von y zwei ist.

c) (einfach) Füge die Elemente $1, 4, 5, 6, 3, 2$ in dieser Reihenfolge in einen zunächst leeren AVL-Baum ein. Zeichne den Baum nach jeder Einfügung.

d) Überlege dir, wie `Rebalance` die Höheninformationen `v.height` mit nur konstantem Mehraufwand aktuell halten kann.

e) Sei T ein binärer Suchbaum mit n Knoten. Beweise, dass die Höhe von T durch $O(\log n)$ beschränkt ist, wenn T die AVL-Eigenschaft erfüllt.

f) (schwer) Sei T ein binärer Suchbaum, sodass bis auf die Wurzel v alle Knoten die AVL-Eigenschaft erfüllen. An der Wurzel sei der Balancefaktor -2. Beweise, dass der Baum nach Ausführung von `Rebalance`(v) ein AVL-Baum ist.

Aufgabe 3 (Blätter und Höhe). Sei T ein Binärbaum mit n Knoten und Wurzel w.

a) Entwirf einen rekursiven Algorithmus, der für Eingabe w die Anzahl der Blätter in T ausgibt. Schreibe deine Lösung in Pseudocode auf.

b) Entwirf einen rekursiven Algorithmus, der für Eingabe w die Höhe von T ausgibt. Schreibe deine Lösung in Pseudocode auf.

c) Implementiere deine Lösungen in einer Sprache deiner Wahl.

Donnerstag

Aufgabe 4 (Traversierung von binären Suchbäumen).

a) Entwirf einen Algorithmus, der für einen binären Baum T (mit einem Schlüssel x.`key` an jedem Knoten) ermittelt, ob T ein binärer Suchbaum ist.

b) Entwirf einen Algorithmus, der für einen binären Suchbaum T einen *umgedrehten binären Suchbaum* T^R aufbaut: T^R soll ein binärer Baum sein, in dem genau dieselben Schlüssel vorkommen wie in T. Für jeden Knoten v in T^R soll zudem gelten, dass alle Knoten im linken Unterbaum von v Schlüssel haben, die größer gleich v.`key` sind, und dass alle Knoten im rechten Unterbaum von v Schlüssel haben, die kleiner gleich v.`key` sind.

c) (schwer) Entwirf einen Algorithmus, der zwei gegebene binäre Suchbäume T_1 und T_2 zu einem einzigen binären Suchbaum T mit denselben Elementen verschmilzt.

Aufgabe 5 (Vollständige binäre Suchbäume). Sei A ein sortiertes Feld mit $n = 2^{h+1} - 1$ paarweise verschiedenen Zahlen. In welcher Reihenfolge müssen wir die Elemente in einen zunächst leeren binären Suchbaum einfügen, sodass der Suchbaum am Ende ein *vollständiger* Binärbaum ist? Gib die Reihenfolge als eine Sequenz von Feldindizes an.

Aufgabe 6 (Rank/Select). Die binären Suchbäume implementieren eine dynamische Menge S, die die Operationen INSERT, DELETE, PREDECESSOR und SUCCESSOR in Zeit $O(h)$ unterstützt, wobei h die Höhe des Baums ist. Modifiziere die Datenstruktur nun so, dass die folgenden zwei Operationen ebenfalls in $O(h)$ Zeit unterstützt werden:

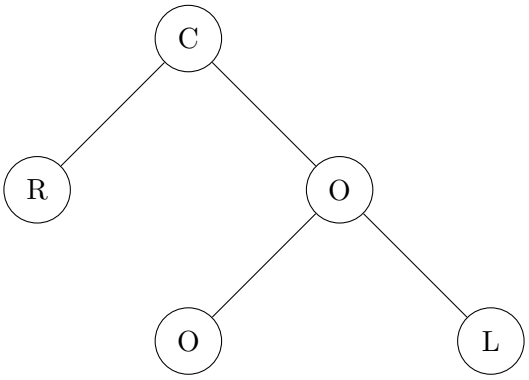
a) RANK(x) liefert die Zahl $\#\{y \mid y.\texttt{key} < x.\texttt{key}\}$, also die Anzahl der Elemente von S, die einen kleineren Schlüssel haben.

b) SELECT(i) liefert das $(i+1)$-te Element der Menge S, also das Element x mit RANK(x) $= i$.

Die existierenden Operationen INSERT und DELETE dürfen hierzu modifiziert werden, müssen aber weiterhin in Zeit $O(h)$ laufen.

Aufgabe 7 (Mehr Rekursionen auf Bäumen). Sei T ein Binärbaum. Jeder Knoten x von T hat die Eigenschaften x.`parent`, x.`left` und x.`right`, welche auf den Elternknoten sowie auf das linke und rechte Kind von x verweisen. Wenn der Knoten keine Kinder hat (z.B. die Blätter) oder keinen Elternknoten (Wurzel `root`) hat, wird der jeweilige Wert auf `null` gesetzt. Des Weiteren hat jeder Knoten x eine Eigenschaft x.`label`, die einen einzelnen Buchstaben speichert. Betrachte den folgenden Algorithmus und den Baum.

```
procedure PrintTree(x)
    if x ≠ null then
        print x.label
        if x.left ≠ null then
            PrintTree(x.left)
        if x.right ≠ null then
            PrintTree(x.right)
```



a) Wenn wir PrintTree mit der Wurzel des Baums aufrufen, wird „CROOL“ auf die Konsole ausgegeben. Wie muss PrintTree modifiziert werden, sodass wir bei derselben Eingabe stattdessen „COLOR“ erhalten?

b) Entwirf einen rekursiven Algorithmus Internal(x), der die Wurzel x des Baums als Eingabe erhält und die Anzahl der internen Knoten des Baums berechnet. Schreib deinen Algorithmus in Pseudocode auf und analysiere die Laufzeit als Funktion von n, wobei n die Anzahl der Knoten des Baums ist.

c) Wir sagen, dass ein Baum einen *R-Pfad* hat, wenn es einen Pfad von der Wurzel zu einem Blatt gibt, sodass alle Knoten v im Pfad v.`label` = `'R'` erfüllen. Entwirf einen rekursiven Algorithmus `RPfad(x)`, der für den gegebenen Wurzelknoten x ermittelt, ob es einen R-Pfad im Baum gibt. Schreib den Algorithmus in Pseudocode auf und analysiere die Laufzeit im Verhältnis zu $|T(x)|$.