

Algorithmen und Datenstrukturen 1

ALGO1 · SoSe-2023 · tcs.uni-frankfurt.de/algo1/ · 2023-07-06 · ed6968a



Traversierung, Binäre Suchbäume (Woche 12)

Eigenständige Vorbereitung:

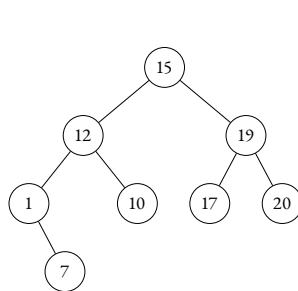
Lies CLRS Kapitel 12 ohne 12.4 und schau dir das Video der Woche an.

Zeichenlegende:

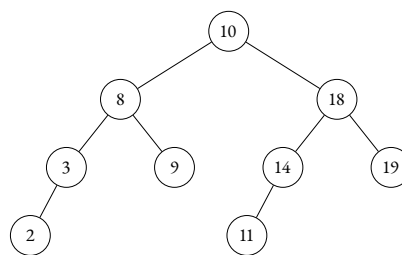
- Schriftliche Aufgabe, die du fristgerecht in Moodle abgibst. In der Klausur wirst du alle Aufgaben schriftlich bearbeiten, daher ist das Feedback der Tutoren wichtig, damit du deine Schreibfähigkeiten verbessern kannst.
- Diese Art von Aufgabe musst du sicher können, um die Klausur zu bestehen.
- Diese Art von Aufgabe musst du weitgehend können, um die Klausur zu bestehen.
- Diese Art von Aufgabe musst du können, um eine gute Note zu erhalten.
- Diese Aufgabe ist als Knobelspaß gedacht, der das algorithmische Verständnis vertieft.

Aufgabe 12.1 (Binärbaumeigenschaften).

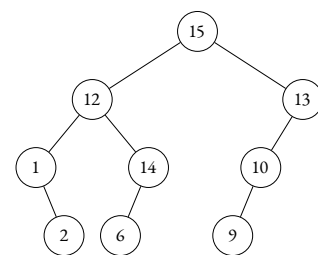
- a) Welche der folgenden Bäume sind binäre Suchbäume?



Baum (i)



Baum (ii)



Baum (iii)

- b) 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) 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 12.2 (AVL-Bäume).

- 👉 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.
- 👉 Schreib den fehlenden Pseudocode für `Rebalance(y)` im Fall, dass der Balancefaktor von y zwei ist.
- 👉 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.
- 👉 Überlege dir, wie `Rebalance` die Höheninformationen `v.height` mit nur konstantem Mehraufwand aktuell halten kann.
- 🔑 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.
- 🔪 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 12.3 (Blätter und Höhe 🔑). Sei T ein Binärbaum mit n Knoten und Wurzel w .

- Entwirf einen rekursiven Algorithmus, der für Eingabe w die Anzahl der Blätter in T ausgibt. Schreibe deine Lösung in Pseudocode auf.
- Entwirf einen rekursiven Algorithmus, der für Eingabe w die Höhe von T ausgibt. Schreibe deine Lösung in Pseudocode auf.
- Implementiere deine Lösungen in einer Sprache deiner Wahl.

Aufgabe 12.4 (Traversierung von binären Suchbäumen).

- 🔑 Entwirf einen Algorithmus, der für einen binären Baum T (mit einem Schlüssel $x.key$ an jedem Knoten) ermittelt, ob T ein binärer Suchbaum ist.
- 🔑 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.
- 🔪 Entwirf einen Algorithmus, der zwei gegebene binäre Suchbäume T_1 und T_2 zu einem einzigen binären Suchbaum T mit denselben Elementen verschmilzt.

Aufgabe 12.5 (Vollständige binäre Suchbäume 👉). Sei A ein sortiertes Feld mit $n = 2^{b+1} - 1$ 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.

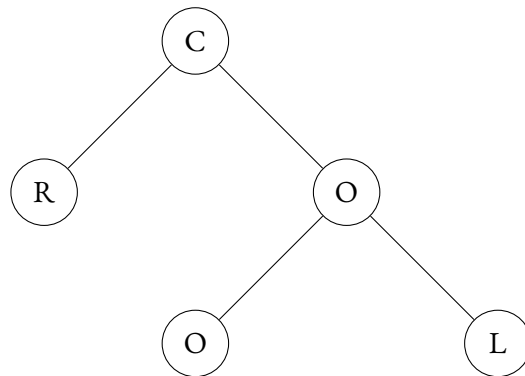
Aufgabe 12.6 (Preorder-Traversierung 🔑). Entwirf einen rekursiven Algorithmus, der die Preorder-Traversierung eines binären Baums T durchführt. Schreibe den Algorithmus in Pseudocode auf.

Aufgabe 12.7 (Rekursion 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  $x \neq \text{null}$  then
    print  $x.label$ 
    if  $x.left \neq \text{null}$  then
      PRINTTREE( $x.left$ )
    if  $x.right \neq \text{null}$  then
      PRINTTREE( $x.right$ )

```



- 👉 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?
- 🔑 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.
- 🔑 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 $|T(x)|$.