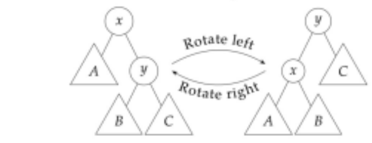
AlgoDat: 6.Hausaufgabe () - Cora Zeitler

Donnerstag, 25. Mai 2023 08:41

Σ: 23/30

Aufgabe 3:
Fügen Sie die Schlüssel ~~20~~, ~~11~~, ~~7~~, ~~15~~, ~~17~~, ~~55~~, ~~37~~ und ~~40~~ in dieser Reihenfolge in einen anfangs leeren Rot-Schwarz-Baum T ein. Geben Sie nach jeder Einfügeoperation den Baum T an. Wie groß ist die Schwarzhöhe von T nachdem alle Schlüssel eingefügt sind? (8 Punkte)

Lokale Rotation, die die Suchbaumeigenschaft erhält.



A, B, C sind Suchbäume, die nicht verändert werden.

- für LeftRotate muss right[x] $\neq$ NIL gelten
- für RightRotate muss left[x] $\neq$ NIL gelten

4 scenarios / Cases

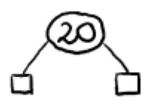
0. Z = root –> *color black*
1. Z.uncle = red –> *recolor*
2. Z.uncle = black (triangle) –> *rotate Z.parent*
3. Z.uncle = black (line) –> *rotate Z.grandparent & recolor*

↳ Elternteile rot

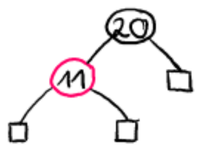
8/8

3) • Anfang: □ (leer

• 20 einfügen:

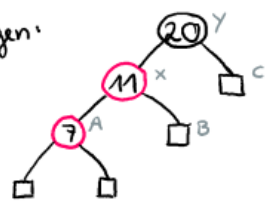


• 11 einfügen:

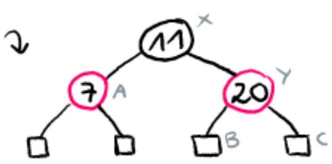


→ Elternteil schwarz
(kein Fixen benötigt)

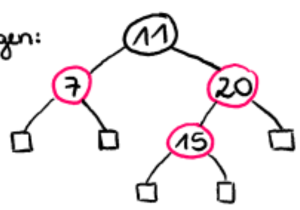
• 7 einfügen:



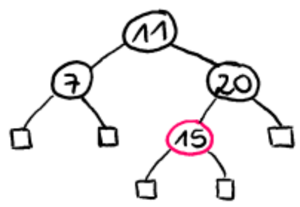
→ case 3
→ Elternteil rot
→ RBInsert Fixup
→ RIGHTROTATION um 20
→ 11 wird schwarz
→ 20 wird rot



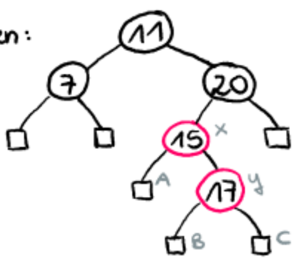
• 15 einfügen:



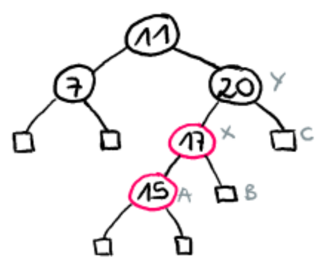
→ case 1
→ Onkel rot
→ RBInsertFixup
→ Kinder von 11 werden schwarz
→ 11 wird rot
→ am Ende wird 11 wieder schwarz, weil keine Elternteile vorhanden



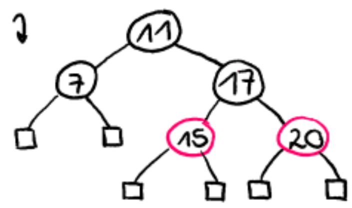
• 17 einfügen:



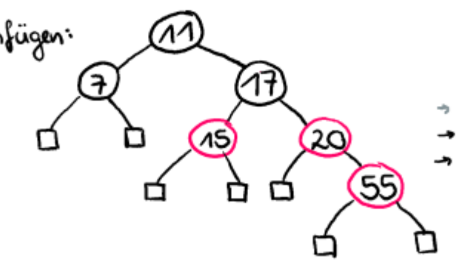
→ case 2
→ Elternteil rot
→ RBInsertFixup
→ LEFTROTATE um 15



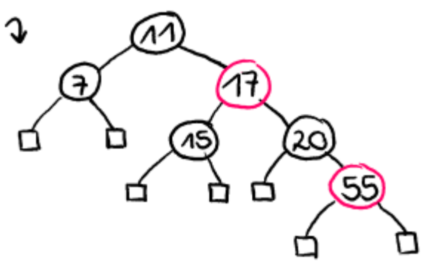
→ case 3
→ Elternteil rot
→ RBInsertFixup
→ RIGHTROTATION um 20
→ 17 wird schwarz
→ 20 wird rot



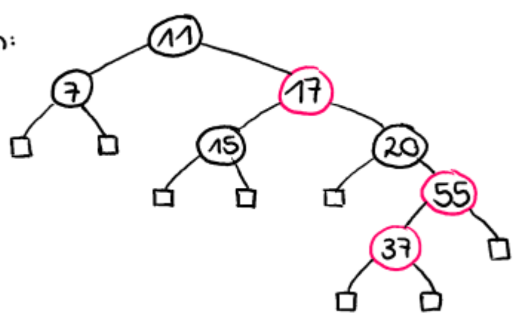
• 55 einfügen:



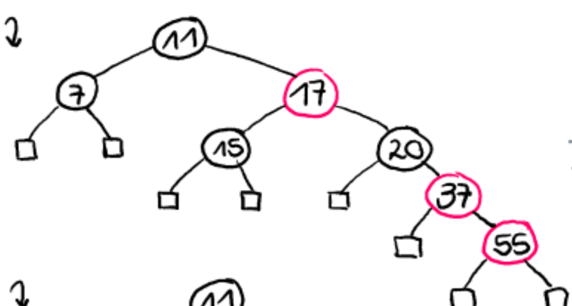
→ case 1
→ Onkel rot
→ RBInsertFixup
→ 20 wird schwarz
→ 15 wird schwarz
→ 17 wird rot



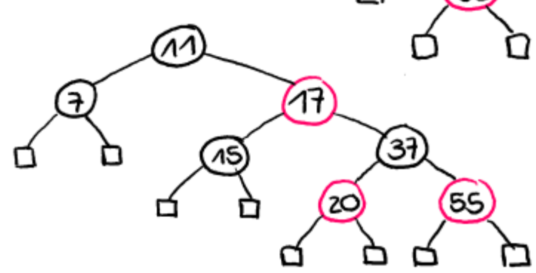
• 37 einfügen:



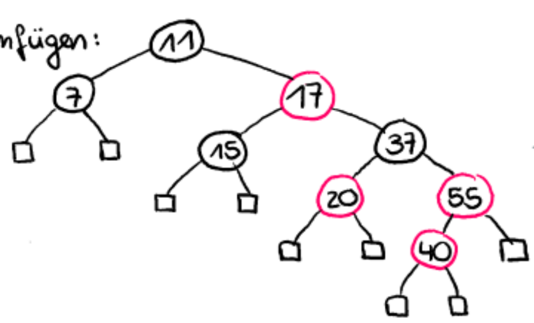
→ case 2
→ Elternteil rot
→ RBInsertFixup
→ RIGHTROTATE um 55



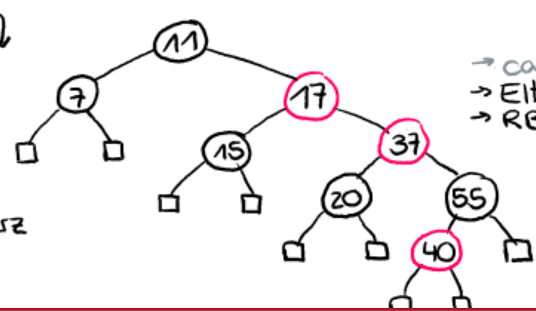
→ case 3
→ Elternteil rot
→ LEFTROTATE um 20
→ 20 wird rot
→ 37 wird schwarz



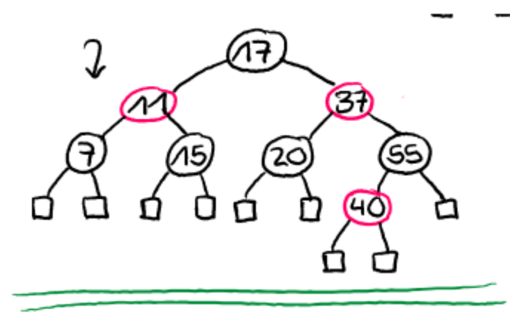
• 40 einfügen:



→ case 1
→ Onkel rot
→ RBInsert Fixup
→ 37 wird rot
→ 20 und 55 werden schwarz



→ case 3
→ Elternteil rot
→ RBInsertFixup
→ LEFTROTATION um 11
→ 17 wird schwarz
→ 11 wird rot
→ 15 anpassen



↳ Schwarzhöhe von T
(Pfad von Wurzel bis zu untersten schwarzen Element)

↳ Pfad von 11 bis 55
↳ Anzahl der Knoten im Pfad

↳ Schwarzhöhe von T = 2

Aufgabe 1:
Berechnen Sie die Anzahl der Permutationen der Zahlen $1, \ldots, 2^k - 1$, so dass beim Einfügen, der durch die Permutation gegebenen Folge, in einen Suchbaum ein vollständiger Binärbaum entsteht für $k = 1, 2, 3$ und finden Sie eine Rekursionsgleichung für den allgemeinen Fall.
(7 Punkte)

1) $p(k)$ = Anzahl der Permutation

$k=1 \rightarrow 2^1 - 1 = 1$, also $1, \ldots, 1 \rightarrow p(1) = 1$

$k=2 \rightarrow 2^2 - 1 = 3$, also $1, \ldots, 3 \rightarrow p(2) = 2$ (213 und 231)

$k=3 \rightarrow 2^3 - 1 = 7$, also $1, \ldots, 7 \rightarrow p(3) = 80$ ✓

6/7

↳ 4 vorne festgelegt ↳ die $n=6$ anderen Positionen können frei belegt werden

↳ dabei gibt es jeweils 2 Mgl.keiten für den linken und den rechten Teilbaum (Bsp. 213 und 231 im linken Teilbaum)

↳ Um die Elemente im Knoten auszuwählen: $n = 6$, $k=3$ ↳ $\binom{6}{3}$

↳ $2 \cdot 2 \cdot \binom{6}{3} = 80$ ✓

Begründung?

↳ Allgemeiner Fall: $\binom{(2^k-1)-1}{2^{k-1}-1}$ Möglichkeiten, welche Positionen von Elementen des linken Teilbaumes belegt sind

↳ es gibt für die Reihenfolge jeder dieser Belegungen jeweils für den linken und rechten Teilbaum $p(n-1)$ Möglickeiten

↳ $p(k) = p(k-1)^2 \cdot \binom{(2^k-1)-1}{2^{k-1}-1}$ ✓

Aufgabe 2:
Zeigen Sie, dass jeder beliebige binäre Suchbaum mit n Knoten durch $O(n)$ Rotationen in jeden anderen binären Suchbaum überführt werden kann. (7 Punkte)

2) → Baum in eine „rechtsläufige" Kette umwandeln

→ die Wurzel und alle ihre Kinder im rechten Teilbaum bilden das Grundgerüst der Kette

→ jeder Knoten, der linkes Kind eines Knotens innerhalb der Kette ist, kann durch RECHTSROTATION in Kette gebracht werden
(somit können neue Linksknoten der Kette entstehen
↳ wieder RECHTSROTATION bis keine Linksknoten mehr vorhanden)

Dass das funktioniert wäre zu zeigen (bzw wie genau was rotiert wird)

→ Transformation benötigt höchstens $n-1$ (worst case: Wurzel hat kein rechtes Kind)

→ Kette kann nun in jeden beliebigen anderen Baum transformiert werden (auch höchstens $n-1$ Rotationen)

⟹ höchstens $2n-2$ Rotationen → $\in O(n)$ ✓

5/7

Aufgabe 4:
Gegeben ist eine Folge $K = \langle a_1, \ldots, a_n \rangle$ von n paarweise verschiedenen ganzen Zahlen. Gesucht ist ein Algorithmus der das zweitkleinste Element der Folge K ermittelt. Dieser Algorithmus soll möglichst wenige Vergleiche zwischen Elementen aus K ausführen (nicht nur asymptotisch). Erläutern Sie zuerst an einem selbstgewählten Beispiel mit $n = 8$ wie Sie vorgehen. Argumentieren Sie dann, dass Ihre Vorgehensweise auch für beliebige Eingaben das Gewünschte leistet und geben Sie eine Abschätzung für die Anzahl der Vergleiche zwischen Elementen aus K an. (8 Punkte)

Algorithmus:

1. Initialisiere zwei Variablen min1 und min2 mit den größtmöglichen Werten (zum Beispiel min1 = +∞ und min2 = +∞)
2. Gehe jedes Element a[i] der Folge K durch
3. Vergleiche a[i] mit min1:
   Wenn a[i] kleiner als min1 ist, dann setze min2 auf den aktuellen Wert von min1 und min1 auf a[i]
   Ansonsten, wenn a[i] größer als min1 und kleiner als min2 ist, setze min2 auf a[i]
4. Das zweitkleinste Element ist min2.

4/8

Laufzeit: O(n) --> führt maximal n-1 Vergleiche aus f
Jedes Elem > min1 : $2n-1$ Vgl
↳ Das geht schneller

Beispiel: mit n=8

1. Initialisierung: Setze min1 = +∞ und min2 = +∞.
2. Betrachte das erste Element 9. Da min1 == +∞, setze min1 = 9.
3. Betrachte das zweite Element 5. Da 5 < min1, setze min2 = min1 (min2 = 9) und min1 = 5.
4. Betrachte das dritte Element 3. Da 3 < min1, setze min2 = min1 (min2 = 5) und min1 = 3.
5. Betrachte das vierte Element 7. Da 7 > min1 und 7 < min2, setze min2 = 7.
6. Betrachte das fünfte Element 2. Da 2 < min1, setze min2 = min1 (min2 = 3) und min1 = 2.
7. Betrachte das sechste Element 8. Da 8 > min1 und 8 < min2, setze min2 = 8.
8. Betrachte das siebte Element 6. Da 6 > min1 und 6 < min2, setze min2 = 6.
9. Betrachte das achte Element 1. Da 1 < min1, setze min2 = min1 (min2 = 2) und min1 = 1.

Vorgehensweise:

- in jedem Schritt wird ein Element a[i] mit min1 und min2 verglichen
- wenn a[i] < min1 ist, wird min2 auf aktuellen Wert von min1 gesetzt und min1 auf a[i]
  ->somit ist sicher, dass min1 bisher immer das kleinste Element ist und min2 das zweitkleinste Element
- wenn a[i] > min1 ist, wird geprüft, ob a[i] < min2 ist, und gegebenenfalls wird min2 aktualisiert
  -> somit bleibt min2 das zweitkleinste Element, selbst wenn a[i] größer als min1 ist
- am Ende hat man min1 als das kleinste Element und min2 als das zweitkleinste Element

--> da jedes Element der Folge K nur einmal betrachtet wird, werden insgesamt maximal n-1 Vergleiche zw. den Elementen durchgeführt