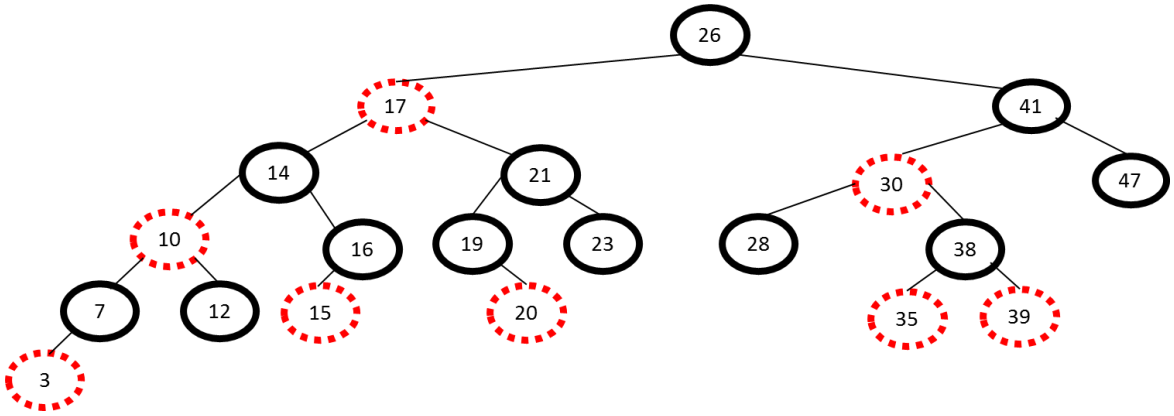
Übung 08: Rot-Schwarz-Bäume

Aufgabe 1: Rot-Schwarz-Bäume

a) Zeichnen Sie den perfekt ausbalancierten ***binären Suchbaum*** der Höhe 3 (T.NULL Blätter nicht zählen!), der die Schlüssel $\{1, 2, 3, \ldots, 15\}$ enthält.

b) Färben Sie nun den Baum aus a) auf 3 verschiedene Weisen zu einem gültigen ***Rot-Schwarz-Baum***, so dass die Schwarztiefe *bh(x)* an der Wurzel *x* den Wert 2, 3 bzw. 4 annimmt.
Hinweis: `bh(x)` wird auf Folie 7 der Vorlesung definiert. Ggfs. `T.NULL` Blätter ebenfalls einzeichnen.

c) Kennzeichnen Sie im folgenden Rot-Schwarz-Baum die Position, an der die 34 eingefügt wird.
Ist das Ergebnis ein Rot-Schwarz-Baum, falls
- der eingefügte Knoten 34 rot markiert wird?
- der eingefügte Knoten 34 schwarz gefärbt wird?



Aufgabe 2: Einfügen in Rot-Schwarz-Bäume[1]

Geben Sie den Rot-Schwarz-Baum an, der sich mit dem Algorithmus der Vorlesung ergibt, falls man ***der Reihe nach*** die folgenden Schlüssel ***in den Anfangs leeren Baum*** einfügt.

<41, 38, 31, 12, 19, 8>

Zeichnen Sie mindestens jeweils eine Skizze nachdem ein Element eingefügt wurde und die Rot-Schwarz-Baum Eigenschaft wiederhergestellt wurde.

Aufgabe 3: Iteratives Preorder in binären Suchbäumen

Implementieren Sie für die bekannte Klasse BST die Methode `public Iterable<Key> keysPreOrderIterative()`.

- Die Methode soll ein Objekt zurückliefern, dass die `Iterable`-Schnittstelle implementiert, möglich ist z.B. eine verkettete Liste oder eine Array-basierte Liste.
- Die Schlüssel (nicht die Nodes) sollen in der Preorder Reihenfolge durchlaufbar sein.
- Das Durchlaufen muss iterativ (nicht rekursiv) erfolgen. Welche Datenstruktur ist hilfreich?
- Testen Sie mit `BSTTest.java`. Dort wird mit dem Baum aus Übung 07/Aufgabe 2 getestet.

[Optional, für Interessierte]: Wie funktioniert das Löschen bei Rot-Schwarz-Bäumen?
https://www.geeksforgeeks.org/red-black-tree-set-3-delete-2/

[1] https://www.cs.usfca.edu/~galles/visualization/RedBlack.html