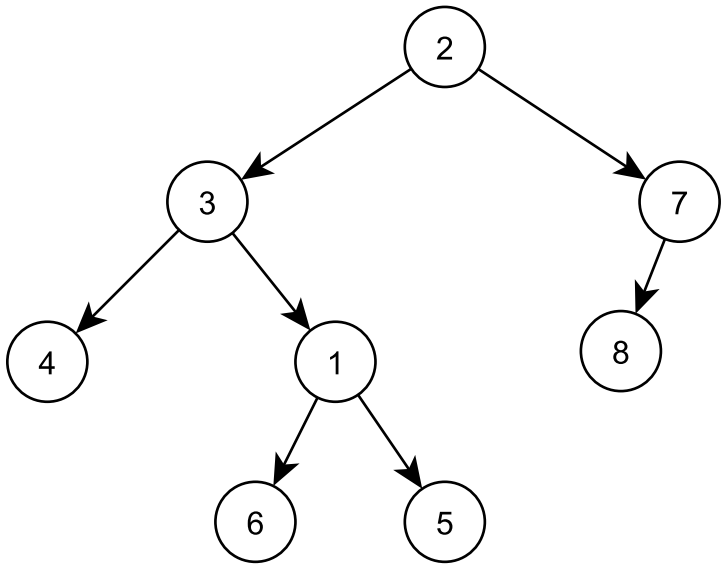
Ein *Breitendurchlauf* („BFS - Breadth First Search“) durch einen binären Baum liefert alle Knoten des Baums „ebenenweise“ von links nach rechts.
Der BFS-Algorithmus startet mit dem Wurzelknoten und fügt alle nicht-leeren Kindknoten des aktuell besuchten Knotens in eine *Schlange* s ein. Das jeweils erste Element der Schlange wird bearbeitet und danach aus der Schlange entfernt.

Beispiel: Für diesen Baum liefert BFS die Knoten in der Reihenfolge $2-3-7-4-1-8-6-5$.



Aufgabe 10.1
Ergänzen Sie die Klasse BinBaum<T> so, dass Sie das Interface Iterable <T> implementiert. Definieren Sie dazu in einer inneren Klasse bfsIterator einen Iterator, der die Knoten in BFS-Reihenfolge liefert.

Aufgabe 10.2 (Theorie)
Geben Sie für den Beispiel-Baum oben jeweils die preorder-, inorder- und postorder-Reihenfolge der Knoten an.

Wir haben bereits im zweiten Übungsblatt (Aufgabe 2.3) in einer Klasse MengeUtil eine Methode merge implementiert, die die „Vereinigung“ zweier Mengen berechnet. Gehen Sie für die folgenden Aufgaben nun zusätzlich davon aus, dass Mengen *iterierbar* sind, verwenden Sie also das folgende Interface als Grundlage:

```
1 public interface Menge<T> extends Iterable<T> {
2
3         int size();
4         boolean isEmpty();
5         T get();
6         void insert(T e);
7         void delete(T e);
8         boolean contains(T e);
9 }
```

Aufgabe 10.3

a) Implementieren Sie in MengeUtil die Methode **void** merge(Menge a, Menge b), die alle Elemente der Menge b der Menge a hinzufügt. Anders als in Kapitel 2 soll die Methode *nur* die Menge a verändern; die Menge b darf sich *nicht* verändern!

b) Implementieren Sie dieses Interface nun mittels eines Suchbaums (das setzt allerdings voraus, dass der Typ T über eine natürlich Ordnung verfügt).

c) Testen Sie die Methode merge am Beispiel der beiden Integer-Mengen $a = \{25, 7, 2024, 23, 0, 47\}$ und $b = \{47, 11, 0, 8, 15\}$, die beide mittels eines binären Suchbaums definiert sein sollen.
Verwenden Sie als Iterator den bfs-Iterator.

d) (Theorie:)
Stellen Sie graphisch dar, wie die Suchbäume der beiden Mengen und der Suchbaum der Vereinigung $a \cup b$ (also die Menge a nach Ausführung der merge-Methode) aussehen.
Die Einfüge-Reihenfolge sei dieselbe wie in der oben genannten Aufzählung.

Aufgabe 10.4

a) Implementieren Sie in MengeUtil eine weitere Methode cut(Menge a, Menge b), die die *Schnittmenge* $a \cap b$ zweier Mengen a und b bestimmt.
(Für Mathe-„Dummies“: Der Schnitt zweier Mengen besteht aus genau denjenigen Elementen, die sowohl in a als auch in b enthalten sind.)
Die Methode soll eine neue, dritte Menge liefern, die ebenfalls mittels eines Suchbaums implementiert sei.

b) (Theorie:)
Es sei n die Anzahl der Elemente in a und m die Anzahl der Elemente in b. Geben Sie in Abhängigkeit von n und m die Größenordnung (Komplexität) der Laufzeit Ihrer Methode an. (Gehen Sie dabei davon aus, dass die Einfüge-Reihenfolge der Elemente *balancierte* Suchbäume erzeugt.)

Wie wären die Laufzeiten, wenn Sie statt der Suchbäume eine Implementierung der Mengen als Dynamisches Array bzw als EVL gewählt hätten?

Aufgabe 10.5

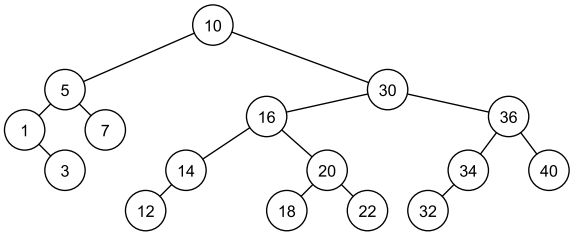
Wie Sie aus der Vorlesung wissen, liefert ein *inorder*-Durchlauf durch einen binären Suchbaum die Elemente in *sortierter Reihenfolge*.
Nutzen Sie dies aus, um in MengeUtil eine Sortier-Methode sort zu implementieren:
Die Methode erhält eine Menge als Eingabe; der Grundtyp soll über eine *innere Ordnung* verfügen.
Die Methode soll ein *dynamisches Array* liefern, das die Elemente in sortierter Reihenfolge enthält.

Aufgabe 10.6 (Theorie)
Fügen Sie in den unten angegebenen AVL-Baum einen Knoten mit Wert 17 ein. Führen Sie die notwendigen Rebalance-Operationen aus.

Markieren Sie dazu im Baum jeweils den (Vater-)Knoten, dessen Balance gestört ist und die beiden beteiligten Sohn- und Enkel-Knoten.

Zur besseren Übersicht können Sie auch die Teilbäume T_1 bis T_4 markieren (bzw die Wurzelknoten dieser Teilbäume bzw die Kanten, die zu diesen Teilbäumen hinführen).

Zeichnen Sie den Baum nach jeder Rotation neu.



Aufgabe 10.7 (Theorie)
Löschen Sie aus dem unten angegebenen AVL-Baum den Knoten mit Wert 33. Führen Sie die notwendigen Rebalance-Operationen aus.

Markieren Sie dazu im Baum jeweils den (Vater-)Knoten, dessen Balance gestört ist und die beiden beteiligten Sohn- und Enkel-Knoten.

Zur besseren Übersicht können Sie auch die Teilbäume T_1 bis T_4 markieren (bzw die Wurzelknoten dieser Teilbäume bzw die Kanten, die zu diesen Teilbäumen hinführen).

Zeichnen Sie den Baum nach jeder Rotation neu.