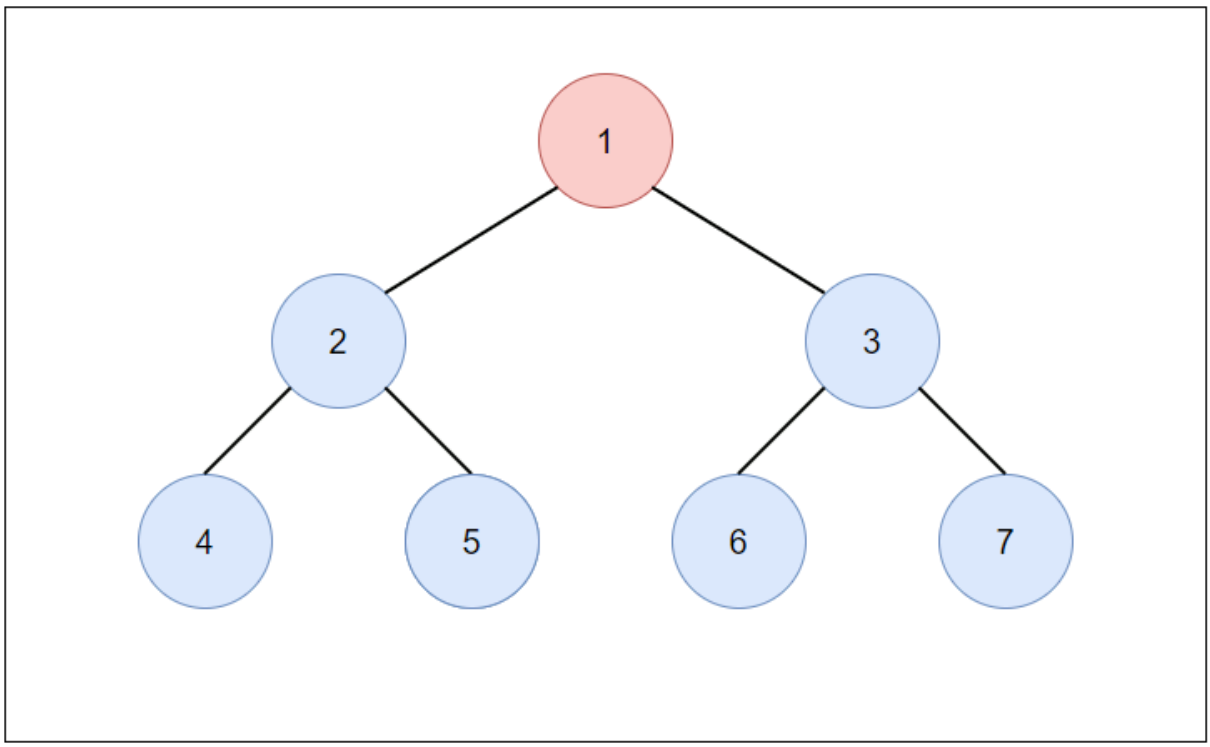
[A05] Breitensuche

Grafische Beschreibung



Textuelle Beschreibung

Bei unserem Breitensuche Algorithmus handelt es sich um einen Binärbaum, welcher genau einen Wurzelknoten hat, sowie maximal 2 Nachfolgeknoten. Der Wurzelknoten wird auch „Root“ bezeichnet. Die Nachfolgeknoten werden als „Child“ definiert. Vorgängerknoten können auch als „Parent“ bezeichnet werden. Die Binärbäume werden in hierarchischen Ebenen dargestellt. In der oben veranschaulichten Grafik ist „1“ der Root Knoten, welcher sich auf Ebene 1 befindet. Die Knoten „2“ und „3“ sind die Kindelemente von Knoten „1“ und befinden sich Hierarchisch auf Ebene 2. Die Knoten „4 – 7“ sind wiederum die Kindelemente von den Knoten „2“ und „3“ und befinden sich auf Ebene 3.
Der Ablauf der Breitensuche ist folgender:

- Starte beim Root Knoten
- Suche nach Kindknoten von links nach rechts Ebene für Ebene durch
- Speichere die Kindknoten in einer „Queue“ zwischen und füge diese dann in eine Liste „besuchte Knoten“
- Wiederhole dies solange bis alle Knoten besucht wurden und alle zwischengespeicherten Knoten von der „Queue“ in die Liste übertragen wurden

Laufzeit

Durchschnittsfall: $O(|V| + |E|)$
Wobei $O(|E|)$ zwischen $O(V^2)$ und $O(1)$ wandert

Wichtige Elemente

```java
public List<Integer> getBreadthFirstOrder(Node<Integer> start) {

   List<Integer> visited = new ArrayList<>();          // visited list for result
   LinkedList<Node<Integer>> queue = new LinkedList<>();  // queue for chaching childs

   queue.add(start);                                 // add start node

   while (!queue.isEmpty()){                          // repeat untief the queue is empty
      Node<Integer> tmpNode = queue.poll();           // cache the node

      if (tmpNode.getLeft()!=null)                    // check left child is existing
         queue.add(tmpNode.getLeft());                // add child to visited list

      if (tmpNode.getRight()!=null)                   // check right child is existing
         queue.add(tmpNode.getRight());               // add child to visited list

      visited.add(tmpNode.getValue());                // add the cached node to the visted list

   }

   return visited;
}
```

```java
public class Node<Type> {

   /**
    * Level des Nodes basierend auf Root
    */
   private int level;

}
```

```java
public List<Integer> getBreadthFirstOrderForLevel(Node<Integer> start, int level) {

   List<Integer> levelList = new ArrayList<>();         // create list for return values
   LinkedList<Node<Integer>> queue = new LinkedList<>();
// create queue for nodes which must be checked

   start.setLevel(1);       // set level of start node to 1
   queue.add(start);        // add start node to queue (=first one to be checked)

   while (!queue.isEmpty()){                 // while queue is NOT empty run through loop
      Node<Integer> tmpNode = queue.poll();
// get element (FIFO), store in temporary variable and remove from queue

      if (tmpNode.getLeft() != null)          // check if left child exists
         queue.add(tmpNode.getLeft());        // if it does add to queue

      if (tmpNode.getRight() != null)             // check if right child exits
         queue.add(tmpNode.getRight());       // if it does add to queue

      if (tmpNode.getParent() != null)            // check if current element has a parent
         tmpNode.setLevel(tmpNode.getParent().getLevel() + 1);
// if it has get level from parent add 1 to it and set it as it's own level

      if (tmpNode.getLevel() == level)
// check if level of node is target level from method parameter
         levelList.add(tmpNode.getValue());
// if it is add it to the list which is returned from the method
   }

   return levelList;  // return complete list of of level

}
```