Lab 7

**Mouna Kacimi, Werner Nutt,
Camilo Thorne**

Traversal of Binary Search Trees

1. Successors in Trees with and without Parent Pointers

We want to write methods to navigate in binary search trees. We suppose that a class `Tree` is defined as

```
class Tree {
Node root;
```

Depending on the definition of the class Node, we distinguish between two kindes of trees:

- trees where nodes have an integer key and two pointers left and right that point to the left and right child of the node
- trees where, in addition, nodes have also a parent pointer.

In the first case, we speak about trees *without* parent pointers, and in the second case we speak about trees *with* parent pointers.

1. Write a recursive method

   ```
   Node minNode(Node node)
   ```

   that returns the node with the minimal key in the subtree rooted at the input node.

2. For trees with parent pointers, write a recursive version of the method

   ```
   Node succ(Node node)
   ```

that retrieves the node with least key value greater than the key value of the input node (we assume that all nodes have different keys), if such a node exists, and null otherwise. In other words, the call `succ(node)` returns the node with the next value after `node.key` among the keys in the tree of node.

Hint: Remember the two cases that we discussed in the lecture.

3. Write a recursive method

   - `Node succ(Node node)`

   for trees without parent pointer.

   Hint 1: The two cases to distinguish are still the same as for the previous exercise, however, in the second case one cannot walk up the tree because there are no parent pointers.

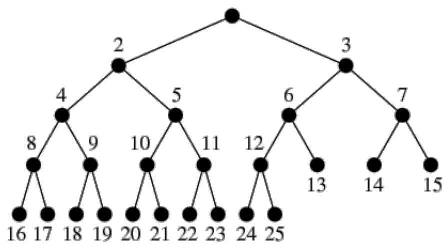
   Hint 2: Note that in the second case of our case analysis, the sucessor of node lies on the path from the input node to the root of the tree. Therefore, one can find the successor of the input node by traversing the tree on the path from the root to the input node. Write an auxiliary recursive methode for this case.

   Hint 3: Every node is the successor of the maximal node in its left subtree. Use this fact as a recursion invariant for the auxiliary method mention in Hint 2.

4. Write an iterative version of the successor method for trees without parent pointers.

2. Breadth-First Traversal of a Binary Search Tree

On Exercise Sheet 7, we introduced a numbering scheme for the nodes of a complete binary tree:



That is, the nodes are numbered as one would encounter them when traversing the tree one level after the other, from left to right.
In an incomplete binary tree, one can still use this scheme for attaching numbers to nodes of the tree. However, since some nodes are missing in an incomplete tree, there will be gaps in the sequence of numbers for which nodes exist in the tree. We call this scheme the "BF" scheme, where BF stands for "breadth first".

1. Design a method

   ```
   void bFTraversal()
   ```

   for binary trees that prints out the keys in the tree nodes as they are encountered in a BF traversal.

   Hint 1: You will need an auxiliary data structure that maintains a queue of nodes to be processed. As discussed in the lecture, you can implement such a queue using HT lists with operations like `enqueue()`, `top()`, `dequeue()`, or `pop()`.

   Hint 2: The query will no more be a queue of integers, but a queue of tree nodes. Take care to distinguish between queue nodes and and tree nodes in your implementation.

2. Modify the method `bFTraveral()` you just developed in such a way that not only the keys are printed out, but also the positions at which you found the keys.

   Hint 1: Modify the queue nodes in such a way that a queue node contains not only a tree node, but also the position of that node according to the BF scheme.

   Hint 2: Add methods to the queue to retrieve both, the top node and position of the top node according to the BF scheme.