# D4 Symbol Tables

**Problem 1**

A)

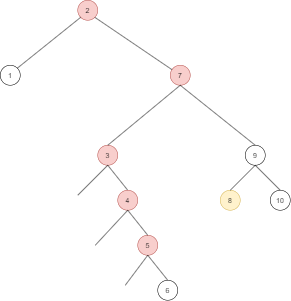
We chose the letter A, since Æ doesn’t exist in the English vocabulary. From that we got the most used word, “another” with 132 occurrences.

B)

We added an argument to the main function called “first\_letter” which is a char. In the for loop iterating over the words we added this to the if sentence:

&& first\_letter == word.charAt(0)

**Problem 2**



The 4th pattern cannot be replicated in a BST.

By asserting that the root is 2 and the right node of the root is 7 we can construct the rest of the node approximately as it would enfold. Some variations are possible on the rightmost sub-tree.

Primarily the given search pattern is incorrect since **the 8 would never be on the left side of the seven**.

The red nodes are the closest replication possible to pattern 4 and the yellow node is what is missing with respect to the given pattern.

**Problem 3**

Submitted separately

(BSTConstructor.java, Node.java)

**Problem 4**

We assume that pop needs to pop the oldest item and push needs to push at the back just like a queue. We would use a data structure that we call Doubly linked list red black tree. It uses the same node for both the tree and the linked list. The node would have following attributes: value, leftchild, rightchild, next, previous.

The left child and right child attributes are for the red black tree and the next and previous attributes are for the doubly linked list.

When we need to pop, then we pop the front most node using the doubly linked list implementation and simultaneously do a remove(node) operation on the popped node in the red black tree implementation.

And if we are removing an item (remove(item x)), then we would find it in the tree and connect the next and previous nodes around the current node before executing the tree reordering operation for removing an item in a red black tree.

This way we can have both pop, push functionality and also the quickness of searching in a red black tree. And of course the contains() function would just go into the tree and try to find it and return true or false if it contains it.

The searching mechanism would therefore take log(n) time

The pop mechanism would take log(n)+1 time

The push mechanism would take

**Problem 5**

We don't have full understanding on what the problem is about. But we did some research on google and looked through the book and the conclusion we came to was that the best data structure would be balanced binary search tree and each node is sorted by the start point and the end point. Then you could take a value x and go through the tree and count how many intervals x overlaps. We would need more research and/or teaching to get full grasp on the full concept and implement this in code. But we have rough idea on how this works.

**Problem 6**

The red black BST will have height 2LogN when the sequence of insertion is nodes in sorted order. A sorted sequence in either direction where N = 2^k − 2, for some k.

**Problem 7**

