Universitetet i Oslo
Institutt for Informatikk

Ragnhild Kobro Runde, Ingrid Chieh Yu, Stein Michael Storleer

IN2010—Algorithms and data structures

Series 1

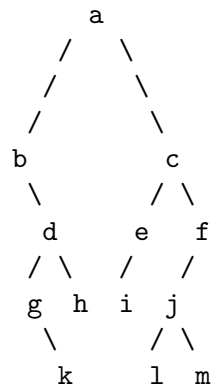
Topic Trees

Issued: 24.08.2018

Classroom

Exercise 1 (Terminology of trees and tree traversal) For the given tree, determine

- what is the root?
- which are the leaves
- what's the tree's height
- Give the result of preorder, postorder, and inorder traversal.
- For all the nodes of the tree
  - name the parent
  - list the children
  - list the siblings
  - compute the height, depth, and size (number of nodes in the subtree).



Exercise 2 (Binary search tree - insertion and deletion)

- Show the result of inserting 6, 4, 8, 5, 1, 9, 7, 11, 2 into an initially empty binary search tree.
- Show the result of first deleting 1 (from the previously constructed tree), and then 6.

Lab

Exercise 3 (Implementation of trees) Discuss how to implement a tree where each node may have an arbitrary number of children. The elements in the tree should be integers.

1. Write a method that returns the sum of all elements in the tree. Should the method be recursive or non-recursive?
2. Write a recursive method that computes the depth AND height for each of the nodes in the tree.

Remember to test the methods on some example trees.

Exercise 4 (Binary tree) Given a binary tree whose nodes are given as instances of the following class:

```
class BinNode {
  int data;
  BinNode left;
  BinNode right;
}
```

An empty tree is represented by the null reference.

1. Write a method `int number(BinNode t)` which gives back the *number of nodes*.
2. Write a method `int sum(BinNode t)` which gives back the sum of the integer data values of all nodes in the tree.

Exercise 5 (Binary trees (2)) Revisiting the binary trees and the `BinNode` data structure described in Exercise 4, this exercise here is to provide a slightly different way of solving the same 2 problems. Instead of the methods sketched in Exercise 4, provide two methods with the (alternative) interface

```
int number()

int sum()
```

so that they are local to class `BinNode`, i.e. one should be able to call functions as follows:

```
int number = root.number();

int sum = root.sum();
```

Exercise 6 (Binary search trees) In this exercise you are going to implement a binary search tree using two different approaches:

1. You are given a Tree class with an inner class Node:

```
public class Tree {
    Node root;

    private class Node {
        Node right;
        Node left;
        int value;

        Node(int value) {
            this.value = value;
        }
    }
}
```

Do the following exercises without changing the Node class, i.e. let all functions be a part of the Tree class:

(a) Implement a function that inserts a value in the BST.

(b) Implement a function that search for a value in the BST, returning a boolean value

(c) Implement a function that returns the smallest value in the BST.

(d) Implement a function that removes a value from the BST.

2. Assume now that you don't have a Tree class, i.e. only the structure

```
public class Node {
    Node right;
    Node left;
    int value;

    Node(int value) {
        this.value = value;
    }
}
```

An empty tree is refered to as a null pointer; the root is used to refer to the tree. Implement all of the above functions as recursive methods in the Node class.

Exercise 7 (Non-unique search keys) We use a binary search tree to store a number of elements containing an integer value (of type `int`) together with a number of other data. We assume that each node has a pointer to its left, resp. right child, as usual. Different from most examples in the lecture, we allow here that different elements can have the *same value* —the other data can be different— and they are supposed to be stored in *different nodes.*

1. A possible solution does the following when inserting a value: if reaching a node carrying the same value, continue further down the three, choosing the *right* subtree for equal values. Write an insert-method that implements this idea and sketch some typical trees that result in that implementation.

2. For a tree created with the above insertion method, if we print the nodes using *inorder* traversal, in which order will the nodes with the same values be printed? In which order would they have been printed, had we instead chosen to use the *left* subtree for equal values?

3. In this part of the exercise, the nodes with the same value should be put into a *list* starting at the first node with that value. This list, however, requires additional pointers, but we can do it as follows: If we insert an object and there is already *exactly one* with this value in the tree, we link that element between the old node and its right subtree. If later on more objects with the same value should be inserted, they will be linked into a list from that *second* object where the *left*-pointer is used as list-pointer. Sketch some examples, and write an insert method based in that idea.

4. when printing out trees constructed as described under 3 using *inorder* traveral, nodes with the same values end up in one batch. But in which order are they actually processed? Write a modified print-method which prints nodes with the same values in the order they had been inserted into the tree.

Exercise 8 (Frequency Tree) We want to use a binary search tree to analyze the play Vildanden from Henrik Ibsen. First read all the words which are separated by, e.g. ";", "?", etc, from the file and insert them into an initially binary search tree. The same word in upper case and lower case should count as the same word. Each node in the tree is corresponding to a unique word in the file. Each node should remember the frequency of the corresponding word appear in the play.

- Create a separate frequency tree which is sorted on the frequency of each word.
- Write a sorted list of the N most frequently used words (e.g. N = 20).
- Write a list of all words having frequency between X and Y (which may be equal).
- Calculate the depth of the left and right subtrees for both the binary search tree and the frequency tree.