Universitetet i Oslo
Institutt for Informatikk
I. Yu, D. Karabeg

INF2220: algorithms and data structures

Series 2

Topic Balanced trees

Issued: 31. 08. 2016

Classroom

Exercise 1 (Trees) Given a binary tree (not necessesarily a binary search tree), with an arbitary number of nodes. Implement/write a function which

(a) returns the smallest value in the tree

(b) returns the largest value in the tree

(c) returns the length of the longest path from the root to a null pointer

(d) returns the length of the shortest path from the root to a null pointer

Exercise 2 (Nodes in a binary tree)

1. Show that in a (non-empty) binary tree with N nodes, there are $N + 1$ `null` links representing children.
2. A *full* node is a node with 2 children. Prove that the number of full nodes + 1 is equal to the number of leaves in a non-empty binary tree.

Exercise 3 (Red-black tree) Build, step by step, red-black trees that result from inserting the following sequences of elements:

1. `41 38 31 12 19 8`
2. `A L G O R I T H M`

Exercise 4 (B-trees)

1. Assume an empty *B-Tree* with $M = 4$ and $L = 4$. Insert the following values in the given order:

`A B C D G H K M R W Z`

Show how the tree changes step by step.

2. Assume an empty *B-Tree* with $M = 5$ and $L = 5$. Insert the following values in the given order:

   `2 6 17 20 24 25 27 29 30 31 32 5 21 1 40 45 50 70`

   Show how the tree changes step by step.

3. Assume an empty *B-Tree* with $M = 3$ and $L = 4$.

   - Insert the following values:

     `61 27 19 5 7 25 36 4 42 2 13 44 62 98 43 16 24 29 15`

     Show how the tree changes step by step.

4. Assume an empty *B-Tree* with $M = 3$ and $L = 2$.

   - Insert the following values:

     `3 14 1 59 26 5 89 79`

     Show how the tree changes step by step.
   - Delete `59, 5, 3, 1` and `26`. Draw the tree after each deletion.

Exercise 5 (Rotations - theory) What do we mean by a *single roation*, and what do we mean by a *double rotation*? Give a few examples on each.

Exercise 6 (AVL trees (extra exercise)) Build, step by step, *AVL* trees that result from inserting the following sequences of elements:

- `41 38 21 12 19 8`
- `A L G O R I T H M`

Lab

Exercise 7 (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.

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 8 (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 9 (Binary trees (2)) Revisiting the binary trees and the `BinNode` data structure described in Exercise 8, this exercise here is to provide a slightly different way of solving the same 2 problems. Instead of the methods sketched in Exercise 8, 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 number = root.sum();
```

Exercise 10 (B-Trees) Write a *general* implementation of insertion for a B-Tree. Note that you have to restructure the tree in case the leaf is full after the insertion.

Exercise 11 (Rotations - programming (can also be blackboard exercises)) Implement a function which makes V the new root node (for a subtree), without destroying the properties of the BST.

References