Design of our Binary Tree Program

Patzl, Steinberger

Introduction

This document describes the design of our terminal-based binary tree program. It contains a complexity analysis and describes the recursive functions of the program.

Complexity Analysis Of The Binary Tree

Single operations

Best Case

The best case is an empty tree. The first entry will always have following complexity:

$O(1)$

Average Case

The average case occurs if there are already other entries in the tree and a random entry is accessed or a new one is added somewhere in the tree:

$O(log(n))$

Worst Case

The worst case occurs if the tree is shaped like a linked list and the last entry is accessed or a new entry is added to the end of the 'list':

$O(n)$

Set Of Operations

Best Case

The best case is that the tree is AVL compliant - this results in inserts and searches with a complexity of:

$O(log(n))$

Worst Case

Because our tree does not re-balance itself to reach AVL compliance on insert, the worst case is that it is shaped like a linked list. Therefore the worst case complexity is:

$O(n)$

Function - insert

The insert function of the tree calls the recursive insertInner function which steps down the tree, checking if the new value is less or greater than the value of the current node and inserts a new node after reaching the end of the branch it stepped down.

Declaration:

```
void insert(int value);
void insertInner(int value, Node* node);
```

Function - printBalance

The printBalance function calls itself for the right node of the given node until it reaches the end of the tree. After reaching the end of the tree it prints the value of the last node and it's balance. If the balance breaks the AVL rules, the AVLflag of the tree is set to false. Next it steps back up and repeats the previous steps with the left branch. After all nodes beneath a node were printed the node from where they branched is printed.

Declaration:

```
int printBalance(Node* node)
```

Function - getMin

The getMin function recursively steps down the left most branch of the tree and returns the value of the last node of the branch.

Declaration:

```
int getMin(Node* node)
```

Function - getMax

The getMax function recursively steps down the right most branch of the tree and returns the value of the last node of the branch.

Declaration

```
int getMax(Node* node)
```

Function - getAverage

The getAverage function initiates a counter and calls the sumValues function with a reference to the counter. The sumValues function recursively sums up the values of all nodes of the tree and returns it to the getAverage function. Then the getAverage function divides the sum by the counter to get the average value of the tree.

Declaration:

```
double getAverage(Node* node) const
int sumValues(Node* node, int &count) const
```

Function - printTree

The printTree function calls the functions getBalance, getMin, getMax, getAverage and checks the AVLflag to print the final output.

Declaration:

```
void printTree();
```

Example

```
bal (5) = 0
bal (4) = 1
bal (3) = 2 (AVL Violation)
bal (2) = 3 (AVL Violation)
bal (1) = 4 (AVL Violation)
AVL: no
min: 1, max: 5, avg: 3
```