Technische Universität München

Informatics I for MSE

MSE IN8011
Prof. Bjoern Menze, Giles Tetteh
Philipp Kaeß

WS 19/20
Homework 9

Due Date: 26.01.20, 23:59

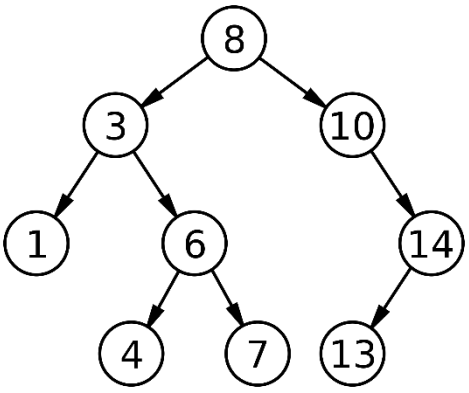
Release Date: 17.01.20

Homework 9: Sorted Tree

Topics: Binary Search Tree, Recursion, Depth-First Traversal, Sorting

Introduction

In the tutorials you were introduced to the concept of trees in C. The advantage of using trees instead of e.g. lists becomes clearer if we look at a special kind of tree, namely a binary search tree. As the name already indicates, this tree is binary (max. of two child nodes per node) and, more importantly, sorted. You can see an example of a binary search tree of integers on the right.



Binary Search Tree with Integer Values

It becomes obvious that an inorder depth-first traversal yields the correct, ascending sequence of the values. We notice that for every node holds, that all descendants of the left subtree of a node are smaller than, and all descendants of the right subtree are higher than the current node.

Imagine you want to search for a specific value in the tree. You compare that value to the root, and if it is larger you pass it on to the right child, if it is smaller you pass it on to the left child. You then repeat this process, until you reach the searched value or a NULL pointer (which indicates that the value is not stored in the tree). For every “step” further down the tree you exclude about half of the remaining possible nodes, as we know that they are all smaller (or higher) than our searched value. Therefore, the number of necessary comparisons to search for a specific value in the tree is proportional to $\log_2 n$, where n is the number of elements in the tree. (Think about it: How would this proportionality look for a list?)

Knowing the definition of a binary search tree from above, you should now be able to create one on your own. Your task will be to code a function to insert a new node into a binary search tree.

Specifications

The function `insert_value()` with the input parameters

- `value` , the `int`-value that should be inserted into the tree
- `n` , a pointer to the root node of the search tree

should (dynamically) create a node with the given value and insert this node into the search tree, so that the tree is still sorted after the insertion.

If a node with this value does already exist in the tree, the function should **not** insert it a second time, but only return `1`. If the value is not found, the function should insert it and return `0`. You may always assume that "unused" child pointers (e.g child pointers of leaves) point to NULL, and of course you are expected to maintain that convention throughout your insertion process. You may assume that your function only gets valid input (`n` points to root of correctly initialized and sorted tree, `value` is in `int`-range).

Tasks

We will provide a template called ***sorted_tree.c***. Complete the function `insert_value()` so that it works as described above. Please note:

- Do not print anything

- You are provided with the corresponding header file ***sorted_tree.h***. It contains the `struct` definition of the tree nodes and the function declarations.

- To encourage you to test and examine your function by yourself, we will not provide any main file this time. For your convenience, the function `print_inorder()` is predefined in the template file. To test your progress, you will have to create your own main function, define and initialize a root node (which by definition is a sorted tree), insert various values into the tree by repeatedly calling your insert function and test the results by printing the tree inorder with the provided function. Your inserted values should then be printed in ascending order. Also do not forget to test the correctness of the return values. (e.g. insert a value twice and check if the second try returns 1) **IMPORTANT: DO NOT FORGET TO DELETE YOUR MAIN FUNCTION AT THE END, AS WE WILL TEST YOUR FUNCTION WITH AN EXTRA MAIN FILE! YOUR C-FILE MUST NOT CONTAIN A MAIN FUNCTION WHEN BEING UPLOADED!**

- You are allowed to code outside the marked spots and e.g. define own functions, but you are of course not allowed to include any additional libraries.

- Your file will be compiled together with a main file and the flags `-Wall -Werror` enabled.

- When you are finished, put ***sorted_tree.c*** directly in a zip-File named "HW9_Mustermann_Max" (with your name of course) and upload this file to Moodle. Do not upload ***sorted_tree.h***, and do not put any additional folders into the zip-File.

Threshold
To pass this homework, your program has to fulfill the following conditions:

- Your file ***sorted_list.c*** must be compilable and runnable (together with our main file) without any errors and with the flags -Wall -Werror enabled

Your functions will be tested on a total of 5 cases. To pass this assignment, you have to pass at least 80% (=4) of the test cases.

Information on Plagiarism and Copyright

As already announced in the lecture and the tutorials, you are supposed to do this homework alone. This homework is an official part of the exam at the end of this semester, and therefore fraud or plagiarism will be pursued and punished.

You are encouraged to discuss **specific** problems with other students (like e.g. “I don’t know how to print a string, can you help me?”), but you are forbidden to share ideas on the approach/structure of your functions, and of course to show other students your solution. This will result in a similar control-flow and structure of your submissions, which we will be able to detect even if you change variable names or loops etc.

We will compare your submissions with a code-optimized plagiarism scanner and if we find plagiarized code, you will fail this submission. If you are found guilty of plagiarism twice, you will lose the homework bonus, and in severe cases we will take further steps.

This homework is intellectual property of the TUM, namely the Chair of Image-Based Biomedical Computing. Publishing either this homework task itself or your solutions to it in any form (including on the MSE-Cloud) is strictly forbidden and can be pursued TUM-internally as well as legally.