Middle East Technical University — Department of Computer Engineering

CENG 140

C Programming

Spring '2019-2020
Take-Home Exam 1

Yusuf Mücahit Çetinkaya
yusufc@ceng.metu.edu.tr
Due date: April 13, 2020, Monday, 23:59

1 Overview

Recursion is a concept in computer science that a function calls itself by breaking the input into pieces directly or indirectly to solve problems. These functions are called recursive functions. The recursion method perfectly fits for some problems such as Towers of Hanoi, tree traversals, graph searches, etc. However, sometimes iterative solutions are simpler and more appropriate.

In this assignment, you are expected to implement some functions with iterative and recursive approaches. Signatures of the functions are given in *the1.h* header file. Additionally, stub codes are available for the source file.

2 Tasks

Binary tree is a data structure where each node in the tree has up to two child nodes. It is a recursive data structure by its nature. More clearly, each node in the tree can be accepted as a root of subtree. In this assignment, the values in the nodes of the tree are integer and stored in an array with the size of *MAX_LENGTH* defined in the header file. Each node's left child and right child can be accessed with 2*node_index+1 and 2*node_index+2. Leaf nodes that do not have any child still preserves a place in the array for each children with -1 value. and Figure 1 visualizes a binary tree that is stored in an array. You will implement insert node, delete node, tree initialization, calculating the height of the tree, finding the min/max in the tree, breadth-first and depth-first search functions.

- You will implement eight methods related to binary tree;
  - **void initialize_the_tree(int binary_tree[MAX_LENGTH], int get_values_from_the_user)**,
  - **void insert_node(int binary_tree[MAX_LENGTH], int parent, char where, int value)**,
  - **void delete_node(int binary_tree[MAX_LENGTH], int node)**,
  - **void draw_binary_tree_rec(int binary_tree[MAX_LENGTH], int root, int depth)**,
  - **int find_height_of_tree_rec(int binary_tree[MAX_LENGTH], int root)**,
  - **int find_min_of_tree_rec(int binary_tree[MAX_LENGTH], int root)**,
  - **int breadth_first_search_itr(int binary_tree[MAX_LENGTH], int root, int value)**,
  - **int depth_first_search_rec(int binary_tree[MAX_LENGTH], int root, int value)**.
- **void initialize_the_tree(int binary_tree[MAX_LENGTH], int get_values_from_the_user)** function gets a parameter that indicates whether user will give initial values or not. If it evaluates false, you will simply set all values of the array -1. Otherwise, you will scan an integer from the user stating the number of values that s/he will enter. User will enter that many integer index and value pairs. If the value for that index is inserted before, or the index or the value is out of the range, you will simply ignore it.
- **void insert_node(int binary_tree[MAX_LENGTH], int node, char where, int value)** function gets index of the node, **'l'**eft, **'r'**ight or **'i'**tself for where to insert the integer value. If there is a value other than -1 for the node to insert, it ignores and does nothing.
- **void delete_node(int binary_tree[MAX_LENGTH], int node)** method gets the index of the node to be deleted. If a node is to be deleted, all of its descendants will be also purged. Since each node in the tree is a subtree, all nodes belonging to that root should be deleted. Deleting means putting -1 value to the proper area in the array.

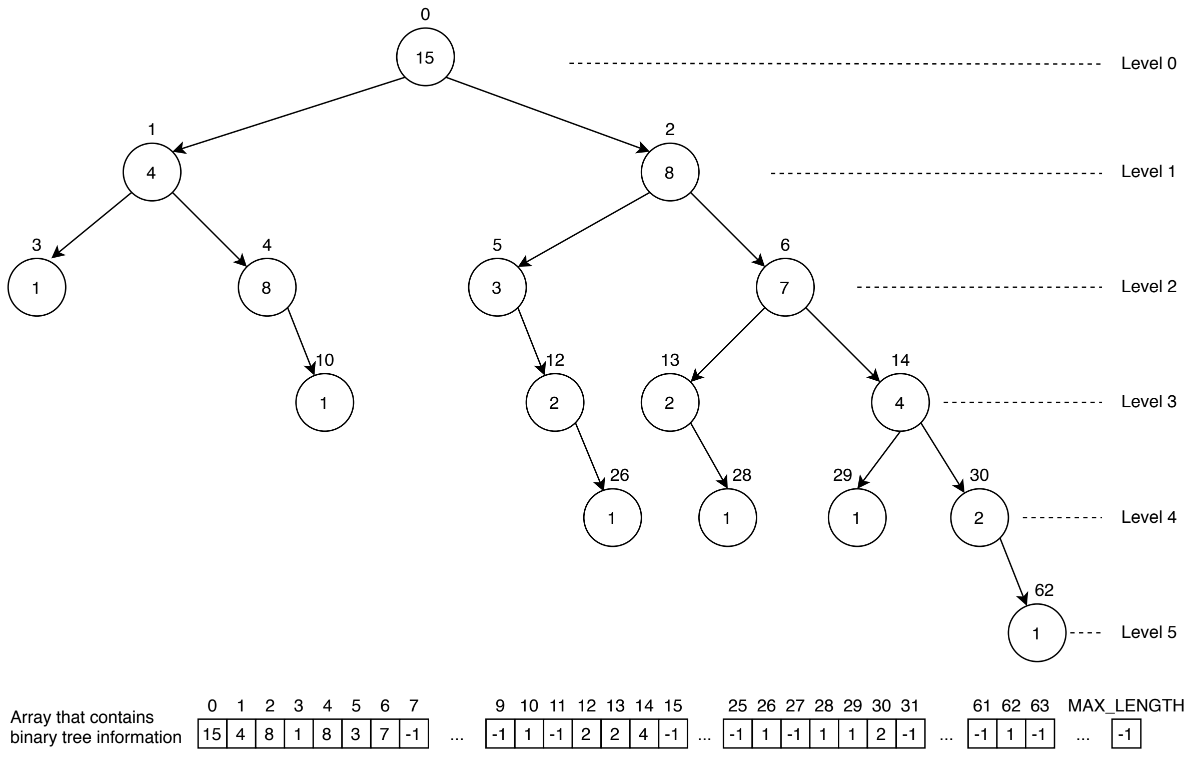

Figure 1: Binary tree that is stored in an array. Node values are integer. The left child of a node is 2*(node_index)+1 and the right child is 2*(node_index)+2. The height of the tree is 5.

- **void draw_binary_tree_rec(int binary_tree[MAX_LENGTH], int root, int depth)** is a recursive function that prints the tree in in-order fashion. In other words, it will print the nodes starting from the left-most child and traverse the rest of the tree in that manner. Printing order will be <left-child, root, right-child>. It gets the index of the root and the depth value as control variable. Initial value of the depth will be the height of the tree. Be careful, any sub-tree can be given to the function. For example, calling this function on the tree visualized in Figure 1 with 14 as root index and 2 as depth will come up with following output;

```
          1
                    4
          2
1
```

- **int find_height_of_tree_rec(int binary_tree[MAX_LENGTH], int root)** is a recursive function that returns the height of the tree. If given root does not have any child, its height is 0. Be careful, any sub-tree can be given to the function.
- **int find_min_of_tree_rec(int binary_tree[MAX_LENGTH], int root)** is a recursive function that returns the minimum value given tree contains. Be careful, any sub-tree can be given to the function.
- **int breadth_first_search_itr(int binary_tree[MAX_LENGTH], int root, int value)** is an iterative function that performs breadth-first search (BFS) on the given tree. BFS is an algorithm for searching the tree in level-wise order. After one level is completed, another level is explored with keeping the order. Detailed information can be found here and here. If the value does not appear in the given tree, it returns -1. Otherwise, it returns the index of the first observation of the value. It gets the index of the root and the integer value that is to be searched. Be careful, any sub-tree can be given to the function and you will apply level wise search in the tree. For example, searching a value in the tree visualized in Figure 1 with the root index of 6 will explored node indices with the order is as follows;

```
6 13 14 28 29 30 62
```

- **int depth_first_search_rec(int binary_tree[MAX_LENGTH], int root, int value)** is a recursive function that performs depth-first search (DFS) on the given tree. DFS is an algorithm for searching the tree with exploring the farthest point starting from root. Detailed information can be found here and here. If the value does not appear in the given tree, it returns -1. Otherwise, it returns the index of the first observation of the value. It gets the index of the root and the integer value that is to be searched. Be careful, any sub-tree can be given to the function. The tree traversal of your implementation should be in-order. For example, consider searching a value in the tree visualized in Figure 1 with the root index of 6. Explored node indices with the order is as follows;

```
13 28 6 29 14 30 62
```

- Sample main functions are provided to you for debugging but you will be graded with hidden main functions. the1.h file contains some definitions like MAX_VAL, MIN_VAL, MIN and MAX macros, etc. If you need, you can use them.
- Using any high-level data structure (pointers, structs, etc.) is strictly **forbidden**.
- You must use recursion for recursive functions and iterative approach for iterative function. Implementing these functions with another techniques will not give you any points. The recursive functions must not include any loop or global, static variables.

3 Regulations

- **Input-Output format:** You will not implement a main function. You will print something in only draw_binary_tree_rec function. The sample output of this function with initial parameters root=0 and depth=5 is provided below. Other functions' side effects are different or simply returns its result.
  Output:

```
                        1
                                4
                        8
                1
                                        15
                        3
                2
        1
                                8
                2
        1
                        7
        1
                4
        2
1
```

- **Programming Language:** C
- **Libraries and Language Elements:**
  You should not use any library other than *"stdio.h"* and *"math.h"*. You can use conditional clauses (switch/if/else if/else), loops (for/while). **You can NOT use any further elements beyond that (this is for students who repeat the course).** You can define your own helper functions.
- **Compiling and running:**
  DO NOT FORGET! YOU WILL USE ANSI-C STANDARTS. You should be able to compile your codes and run your program with given **Makefile**:

```
>_ make the1
>_ ./the1
```

  If you are working with ineks or you are working on Ubuntu OS, you can feed your program with input files instead of typing inputs. This gives the input from stdin and an equivalent of typing inputs. This way you can test your inputs faster:

```
>_ ./the1 < inp1.txt
>_ ./the1 < inp2.txt
```

- **Submission:**
  You will use CengClass system for the homework just like Lab Exams. You can use the system as editor or work locally and upload the source files. Late submission IS NOT allowed, it is not possible to extend the deadline and **please do not ask for any deadline extensions**.
- **Evaluation:** Your codes will be evaluated based on several input files including, but not limited to the test cases given to you as example. You can check your grade with sample test cases via CengClass system but do not forget it is not your final grade. Your output must give the exact output of the expected outputs. It is your responsibility to check the correctness of the output with the invisible characters. Otherwise, you can not get grade from that case. If your program gives correct outputs for all cases, you will get 100 points.
- **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations and will get 0. Sharing code between each other or using third party code is strictly forbidden. Even if you take a "part" of the code from somewhere/somebody else - this is also cheating. Please be aware that there are "very advanced tools" that detect if two codes are similar. So please do not think you can get away with by changing a code obtained from another source.