NeuroScouting Generic Programming Exercise

Compiling the program:

This program is compiled using Microsoft Visual Studio 2015 C++ compiler. To run the source file, you can copy and paste the source file in a new project and then compile it. You can also try to compile it with a different C++ compiler, because there are no external dependencies (libraries or external files) attached to the program, except the default "iostream" header file. However, I have not tested compilation with any different compiler.

Total number of files needs to be compiled: 1

Overview of the logic:

Each node in the tree contains comprehensive information about its environment.

Each node stores the following information:

- Data
- Pointer to the Left child
- Pointer to the Right child
- Pointer to the Parent
- Pointer to the Left Neighbor
- Pointer to the Right Neighbor

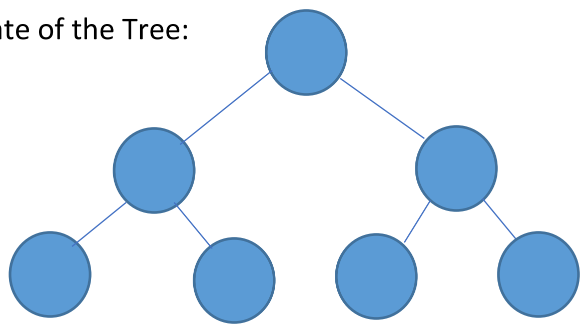
The height/depth of the tree is taken as an input from the user. After the input is taken, the program goes through following sequence of logic:

Example Input: "3"

1) Depending on the input, a "perfect" binary tree is crated with dummy values. This is done using Depth First Traversal. Each node in this tree has two children and while creating the nodes, we properly assign their child and parent pointers.

   (In-program function name: **bool buildTheTree(struct Node*, int , int);**)

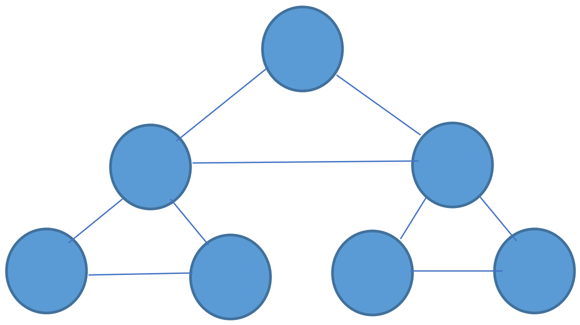
   Current State of the Tree:



2) Now, we have a framework of a binary tree containing dummy values with child-parent relationships properly set. However, the nodes are not aware of their neighbors. So, we now run Breadth First Traversal on the whole tree and while traversing we assign neighbors pointers to each other.

   (In-program function name: **void printGivenLevel(struct Node*, int);**)

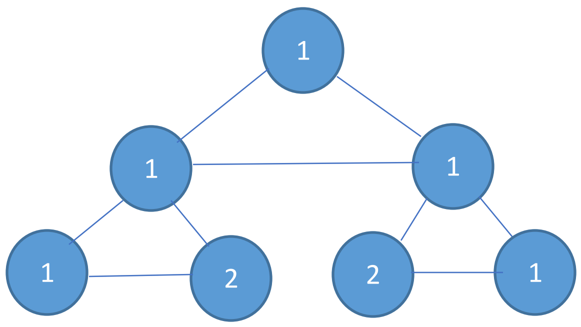
   Current State of the Tree:



3) At this point, we have a proper tree with all nodes aware of their children, parent and neighbors. Now we run Breadth First Traversal again and we set the values of nodes as per given rules. i.e. root value is always 1. If a node is a left child of its parent, then its value equals to addition of its parent's value and parents left neighbor's value. Same goes for right the child. If a child's parent doesn't have any neighbor, then its value is same as its parent's value.
   To do this we run the same function as we did in step 2 except this time we set a different value of a flag which changes flow of a program a little bit differently in the same function.

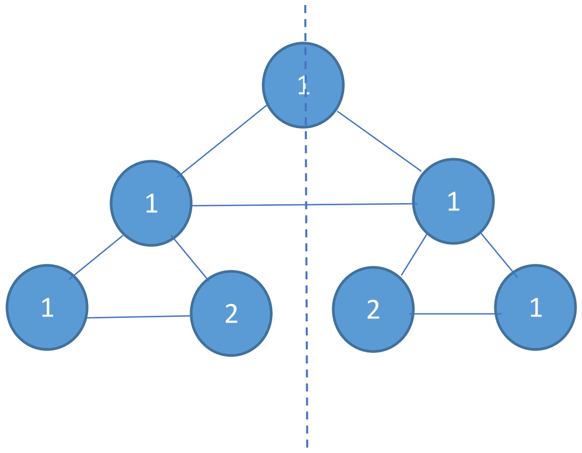
   (In-program function name: **void printGivenLevel(struct Node*, int);**)

   Current State of the Tree:



Optimization of the logic:
Unfortunately, I was not able to do any optimization due to time constraints, however, I would like to propose a logic which can brings time and space complexity of the program to its half.

If we slice the tree vertically in the middle, then we can see that the whole tree looks like its mirrored on both the sides.



So, what if we calculate and traverse only half of the tree, and while printing the values, we can just do a Reverse Breadth First Traversal on the half tree. I think this will heavily optimize the program if properly implemented.

Name: Shreevallabh Kulkarni

Email: shreevallabh29@gmail.com