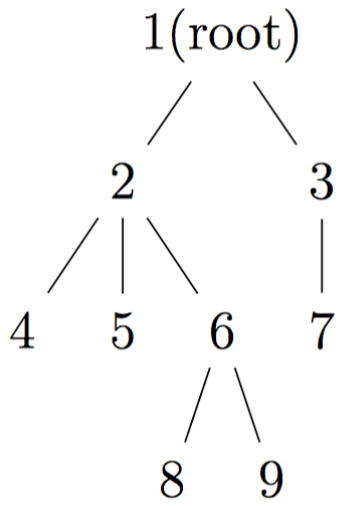
VE 280 Lab 8

Out: 00:01 am, July 5, 2022; **Due**: 11:59 pm, June 12, 2022.

Ex1. N-ary tree

Related Topics: *Dynamic Memory Allocation*, *overloading*, *default arguments*, *destructor*, *recursion*.

A tree ADT organizes and manages data in a hierarchical tree structure. An n-ary tree is a generalization of a binary tree where any node in the tree has exactly one parent, except one node called root node, and any node may have zero up to n children. For an example, see the following figure:



A tree ADT stores data in each node.

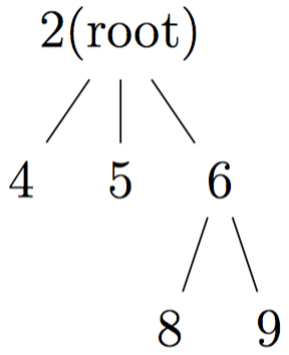
Terminology

Descendants

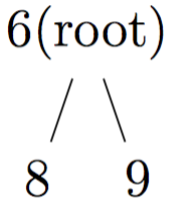
A `descendant` of a node X is a child of X or a descendant of a child of X.
For instance, the descendants of 2 in the previous figure are 4, 5, 6, 8, and 9.

Subtree

A `subtree` of the tree T is a tree consists of a node in T and all of this node's descendants.
For instance, if we name the tree above as T, then the tree rooted in 2 is a subtree of T.



Also, the the tree rooted in 6 is a subtree of T.



Leaf

A node with no child is called a `leaf` node. For example, node 4, 5, 8, 9 and 7 are leaf nodes.

Path

A `path` is a sequence of nodes such that a next node in the sequence is a child of the previous one.

For example 2->6->9 is a path and the length of this path is 2.

Height

The `height` of a node is the length of a longest path from the node to a leaf.

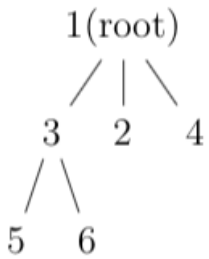
For example, height(1) = 3, height(9) = 0

Implementation

A node of a tree can be represented by the following class and a tree ADT can be represented by the node corresponding to its root.

```
class Node {
    // OVERVIEW: a node in the n-Ary tree, can also represent a n-ary tree rooted
at 'this'
private:
    int value;      // the integer value of this
    int child_num;  // the number of child of this
    int n;          // n for this n-Ary tree
    Node *parent;   // parent node of this, for root node, parent = NULL
    Node **children;
    // children is an array of pointer to Node. Therefore, children is a pointer
of pointer
    int height;     // height of this node

    void addChild(Node *child);
```

```
    // REQUIRES: n of the child node is the same with n of this
    // EFFECTS: add the node child to the children array
    //          throw an exception tooManyChildren when child_num exceed n
public:
    Node(int _value, int _n = 2);
    // EFFECTS: create a root node with value and n

    ~Node();
    // EFFECTS: destroy the (sub) tree rooted as this

    void addChild(int _value);
    // EFFECTS: create a child node with value and add it to the children array
    //          throw an exception tooManyChildren when child_num exceed n

    void traverse();
    // EFFECTS: print the value of the nodes using a pre-order traversal,
    //          separated by a space.
    //          A pre-order traversal print the value of the node
    //          and then traverse its child nodes
    //          according to the sequence in children array.
    //          For example, the output of the tree above is
    //          1 2 4 5 6 8 9 3 7
    //          And the output of the tree below is
    //          1 3 5 6 2 4

    bool contain(Node *sub);
    // EFFECTS: return whether the tree rooted at sub is a subtree of this

    int getHeight();
    // EFFECTS: return height of this

    Node &operator[](int i);
    // EFFECTS: return a reference of (i+1) th child node of this,
    //          e.g. node1[0] returns a reference of the first child node of
node1
    //          if i is invalid, throw an invalidIndex
};
```



Requirements:

Implement the node class.

Testing & Submitting

`lab8Test.cpp` and `test.out` are provided for your test.
`g++ -Wall -Werror -std=c++17 -fsanitize=leak -o lab8 lab8Test.cpp node.cpp` to compile and `./lab8` or use valgrind to check memory leak.
Please compress `node.h` and `node.cpp` and submit it onto JOJ.