МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Деревья

Студентка гр. 9303 _____ Зарезина Е.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

Ознакомление со структурой бинарных деревьев, способом их реализации и разработка базовых функций для работы с деревьями.

Задание.

18д. Бинарное дерево называется бинарным деревом поиска, если для каждого его узла справедливо: все элементы правого поддерева больше этого узла, а все элементы левого поддерева – меньше этого узла. Бинарное дерево называется пирамидой, если для каждого его узла справедливо: значения всех потомков этого узла не больше, чем значение узла. Для заданного бинарного дерева с числовым типом элементов определить, является ли оно бинарным деревом поиска и является ли оно пирамидой.

Основные теоретические положения.

Дерево – конечное множество T, состоящее из одного или более узлов, таких, что

а) имеется один специально обозначенный узел, называемый корнем данного дерева;

б) остальные узлы (исключая корень) содержатся в m ³ 0 попарно не пересекающихся множествах T1, T2, ..., Tm, каждое из которых, в свою очередь, является деревом. Деревья T1, T2, ..., Tm называются поддеревьями данного дерева.

Каждый узел дерева является корнем некоторого поддерева. В том случае, когда множество поддеревьев такого корня пусто, этот узел называется концевым узлом, или листом.

Говорят, что каждый корень является отцом корней своих поддеревьев и что последние являются сыновьями своего отца и братьями между собой. Говорят также, что узел n – предок узла m (а узел m – потомок узла n), если n – либо отец m, либо отец некоторого предка m.

Если в определении дерева существен порядок перечисления поддеревьев T1, T2, ..., Tm, то дерево называют упорядоченным и говорят о «первом» (T1), «втором» (T2) и т. д. поддеревьях данного корня. Далее будем считать, что все рассматриваемые деревья являются упорядоченными, если явно не оговорено противное.

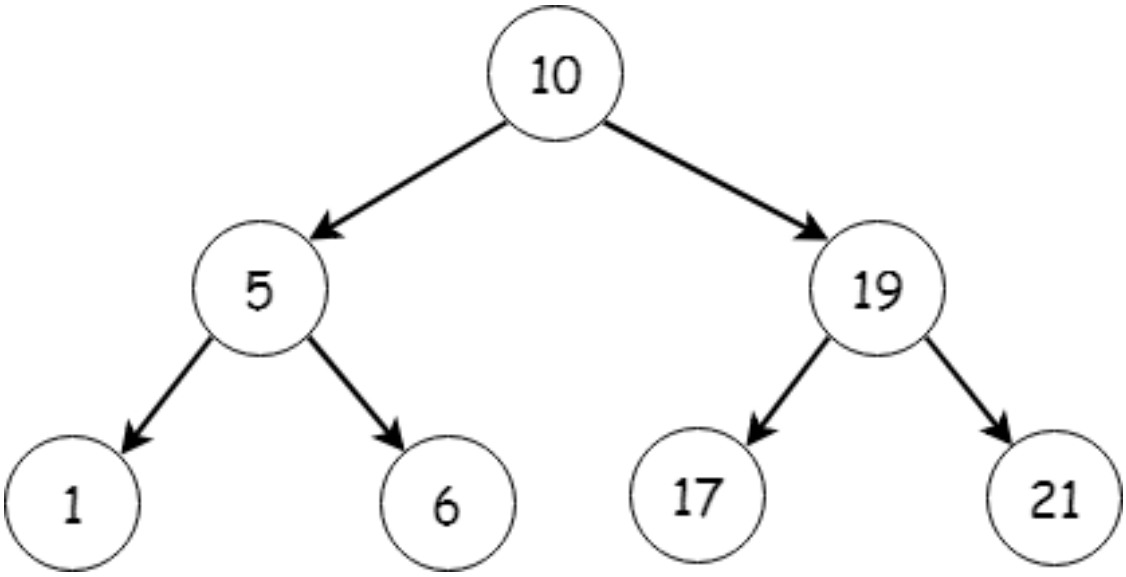
Лес – это множество (обычно упорядоченное), состоящее из некоторого (быть может, равного нулю) числа непересекающихся деревьев. Используя понятие леса, пункт б в определении дерева можно было бы сформулировать так: узлы дерева, за исключением корня, образуют лес.

Выполнение работы.

Пользователь вводит бинарное дерево в виде упрощённой последовательности, где отсутствие поддеревье обозначается нулём. В результате работы программы выводится информация о том, является ли введённое дерево пирамидой или деревом поиска.

На рисунке представлено дерево, соответсвующее упрощённой записи

10 5 1 0 0 6 0 0 19 17 0 0 21 0 0



Тестирование.

Ввод:	**Вывод:**
5 2 1 0 0 3 0 0 9 0 0	is Binary Search Tree? - YES is Pyramide Tree? - NO
30 15 10 0 0 3 0 0 16 0 0	is Binary Search Tree? - NO is Pyramide Tree? - YES
8 3 1 0 0 6 4 0 0 7 0 0 10 0 14 13 0 0 0	is Binary Search Tree? - YES is Pyramide Tree? - NO
4 2 1 0 0 3 0 0 6 5 0 0 7 0 0	is Binary Search Tree? - YES is Pyramide Tree? - NO
16 11 10 1 0 0 2 0 0 5 4 0 0 0 9 6 0 0 8 0 0	is Binary Search Tree? - NO is Pyramide Tree? - YES
5 5 0 0 5 0 0	is Binary Search Tree? - NO is Pyramide Tree? - YES
20 8 5 1 0 0 0 0 0	is Binary Search Tree? - YES is Pyramide Tree? - YES
16 0 9 0 8 0 0	is Binary Search Tree? - NO is Pyramide Tree? - YES

Выводы.

Было реализовано бинарное дерево и базовые функции для работы с ним. В отчёте представлены результаты тестирования, разработанный код программы представлен в Приложении А.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

NODETREE.H

```
#IFNDEF NODETREE_H
#DEFINE NODETREE_H

TEMPLATE<TYPENAME T> CLASS TREE;

TEMPLATE<TYPENAME T> CLASS NODETREE{
    FRIEND CLASS TREE<T>;

    T _DATA;
    NODETREE<T>* _LEFT;
    NODETREE<T>* _RIGHT;

PUBLIC:
    NODETREE():_LEFT(NULLPTR), _RIGHT(NULLPTR){}
             NODETREE(CONST   T   &DATA):   _DATA(DATA),
_LEFT(NULLPTR), _RIGHT(NULLPTR){}

    T GETDATA() CONST {
        RETURN _DATA;
    }

};

#ENDIF
```

TREE.H

```
#ifndef TREE_H
#define TREE_H

#include "NodeTree.h"
#include <sstream>
#include <iostream>
#include <vector>
#define EMPTY '_'

using namespace std;

template<typename T> class Tree{

    NodeTree<T>* _root;
```

```
    void deleteFunc(NodeTree<T>* node){
        if(node!=nullptr){
            deleteFunc(node->_left);
            deleteFunc(node->_right);

            delete node;
        }
    }



public:
    Tree(): _root(nullptr){}

      void readTree(NodeTree<T>**root, vector<long int>&
vec){


        int data = vec.back();
        vec.pop_back();
        *root = new NodeTree<T>(data);
        if(data == 0){
            return;
        }

        readTree(&((*root)->_left),vec);
        readTree(&(*root)->_right,vec);
    }

    NodeTree<T>* getRoot(){
        return _root;
    }

    void setRoot(NodeTree<T>* root){
        _root = root;
    }

    void preorder(NodeTree<T>*root){
        if(root == nullptr){
            cout<<"Sorry, root == nullptr"<<endl;
        } else {
            if(root->getData() !=0){
                cout<<root->_data<<" ";
                preorder(root->_left);
```

```
                preorder(root->_right);
            }
        }
    }

    ~Tree(){
        deleteFunc(_root);
    }

      void isBinSearch(NodeTree<T>** node,long int& min,
bool& flagBS){
        long int data = (*node)->getData();
        if(node == nullptr){
            cout<<"Sorry, node == nullptr"<<endl;
        }
        if((data != 0)&&(flagBS == true)){

        isBinSearch(&((*node)->_left), min, flagBS);
        flagBS &= (*node)->getData() > min;
        min = (*node)->getData();
        isBinSearch(&((*node)->_right), min, flagBS);
        }
    }

    void isPyramide(NodeTree<T>** node, bool& flagPyr){
        long int currData = (*node)->getData();
        long int leftData = (*node)->_left->getData();
        long int rightData = (*node)->_right->getData();
        if((leftData == 0)&&(rightData == 0)){
            return;
        }
        if(leftData == 0){
            flagPyr &= currData>=rightData;
            isPyramide(&((*node)->_right),flagPyr);
        } else if(rightData == 0){
            flagPyr &= currData>=leftData;
            isPyramide(&((*node)->_left),flagPyr);
        } else {
                                                flagPyr    &=
(currData>=leftData)&&(currData>=rightData);
            isPyramide(&((*node)->_left),flagPyr);
            isPyramide(&((*node)->_right),flagPyr);
        }
    }
```

```
};

vector<long int> treeVector(){
    long int num;
    vector<long int> vec;
    vector<long int> vecRes;
    cout<<"Enter your Tree:"<<endl;
    string n;
    do{
        cin >> n;
        num = stoi(n);
        vec.push_back(num);
    }while(cin.peek() != '\n');

    for(int i = vec.size()-1; i>-1;i--){

        vecRes.push_back(vec.at(i));
    }

    return vecRes;
}

void printResult(bool flagBS, bool flagPyr){
    cout<<"\n\tis Binary Search Tree?  -  ";
    if(flagBS){
        cout<<"YES"<<endl;
    } else {
        cout<<"NO"<<endl;
    }
    cout<<"\tis Pyramide Tree?  -  ";
    if(flagPyr){
        cout<<"YES"<<endl;
    } else {
        cout<<"NO"<<endl;
    }
}


#endif
```

MAIN.CPP

```
#include "NodeTree.h"
#include "Tree.h"

#include <iostream>
#include <sstream>
#include <climits>



using namespace std;

template<typename T> class Tree;

int main()
{
    vector<long int> vec = treeVector();
    Tree<long int>*binTree = new Tree<long int>();
    NodeTree<long int> *root = binTree->getRoot();
    binTree->readTree(&root,vec);
    binTree->setRoot(root);
    long int min = LONG_MIN;
    bool flagBS = true;
    bool flagPyr = true;
    binTree->isBinSearch(&root,min, flagBS);
    binTree->isPyramide(&root, flagPyr);
    printResult(flagBS,flagPyr);


    return 0;
}
```