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: Бинарные деревья

Студент гр. 9382 _____ Герасев Г.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Изучить алгоритмы работы с бинарными деревьями.

Задание.

4 в) Для заданного бинарного дерева b типа BT с произвольным типом элементов определить, есть ли в дереве b хотя бы два одинаковых элемента

Реализовать дерево в векторном виде.

Основные теоретические положения.

Бинарное дерево – дерево, где у каждого узла есть 2 поддерева и в каждом узле хранится по одному значению.

Векторное представление бинарного дерева – одномерный массив, хранящий все значения узлов дерева сверху вниз, слева направо, в т.ч. пустые узлы.

Функции и структуры данных.

Создан класс бинарного дерева, при инициализации которого создается пустое дерево заданной глубины. Для добавления значений в узлах создаются 2 метода, для добавления значения по адресу в дереве и по адресу в массиве.

Создается метод, считающий количество переданного элемента в дереве рекурсивным методом. (очевидно проще и быстрее было бы просто пройти по массиву, хранящему значения в дереве, воспользовавшись преимуществом такой структуры дерева, но условие лабораторной запрещает не использование рекурсии). Чтобы реализовать рекурсию отслеживается номер узла в массиве, и при рекурсивном переходе этот номер меняется на номер левого и правого поддерева.

Позже из этого метода очевидно выводится метод проверки существования 2х элементов в дереве.

Описание алгоритма.

Метод применяется к дереву, в переменных которого за исключением искомого значения хранится номер ячейки массива, узел которого рассматривается. Метод начинает работу в 0, после чего возвращает количество искомого элемента в ячейке где он находится и в рекурсивных вызовах функции в левом и правом поддереве. Номер ячейки левого и правого поддерева находится по формулам x^2 + 1, x^2 + 2, и реализуется сдвигом влево. Каждый раз при вызове метода проверяется, что ячейка с таким номером существует в принципе, и если ее нет, то возвращается 0 (очевидно в пустом поддереве нет ни одного искомого элемента).

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	1 1 0 1 1 0 empty 0	inputValue can't be <= 0	
2.	1 1 0 1 1 0 empty 1	Call howManyRecursive Call howManyRecursiveHandler in node 0 Go left in node 1 Found value Go left in node 3 There is no such node in tree, go out of the function Go right in node 4	

		There is no such node in tree, go out of the function Returning sum of left and right Go right in node 2 Found value Go left in node 5 There is no such node in tree, go out of the function Go right in node 6 There is no such node in tree, go out of the function Returning sum of left and right Returning sum of left and right There are 2 or more of 1	
3.	1 1 0 1 1 0 empty 2	Call howManyRecursive Call howManyRecursiveHandler in node 0 Go left in node 1 Go left in node 3 There is no such node in tree, go out of the function Go right in node 4 There is no such node in tree, go out of the function Returning sum of left and right	

		Go right in node 2 Go left in node 5 There is no such node in tree, go out of the function Go right in node 6 There is no such node in tree, go out of the function Returning sum of left and right Returning sum of left and right There is 1 or less of 2	
4.	1 1 0 1 1 1 01 0 empty 1	Call howManyRecursive Call howManyRecursiveHandler in node 0 Go left in node 1 Found value Go left in node 3 There is no such node in tree, go out of the function Go right in node 4 There is no such node in tree, go out of the function Returning sum of left and right Go right in node 2 Found value	

		Go left in node 5 There is no such node in tree, go out of the function Go right in node 6 There is no such node in tree, go out of the function Returning sum of left and right Returning sum of left and right There are 2 or more of 1	
5.	2 1 0 1 1 1 01 1 10 0 empty 1	Call howManyRecursive Call howManyRecursiveHandler in node 0 Go left in node 1 Found value Go left in node 3 Go left in node 7 There is no such node in tree, go out of the function Go right in node 8 There is no such node in tree, go out of the function Returning sum of left and right Go right in node 4 Found value	

		Go left in node 9 There is no such node in tree, go out of the function Go right in node 10 There is no such node in tree, go out of the function Returning sum of left and right Returning sum of left and right Go right in node 2 Found value Go left in node 5 Found value Go left in node 11 There is no such node in tree, go out of the function Go right in node 12 There is no such node in tree, go out of the function Returning sum of left and right Go right in node 6 Go left in node 13 There is no such node in tree, go out of the function	

		Go right in node 14 There is no such node in tree, go out of the function Returning sum of left and right Returning sum of left and right Returning sum of left and right There are 2 or more of 1	
6.	3 1 00 1 000 1 010 1 110 1 011 0 empty 1	Call howManyRecursive Call howManyRecursiveHandler in node 0 Go left in node 1 Go left in node 3 Found value Go left in node 7 Found value Go left in node 15 There is no such node in tree, go out of the function Go right in node 16 There is no such node in tree, go out of the function Returning sum of left and right Go right in node 8 Go left	

		in node 17 There is no such node in tree, go out of the function Go right in node 18 There is no such node in tree, go out of the function Returning sum of left and right Returning sum of left and right Go right in node 4 Go left in node 9 Found value Go left in node 19 There is no such node in tree, go out of the function Go right in node 20 There is no such node in tree, go out of the function Returning sum of left and right Go right in node 10 Found value Go left in node 21 There is no such node in tree, go out of the function Go right	

		in node 22 There is no such node in tree, go out of the function Returning sum of left and right Returning sum of left and right Returning sum of left and right Go right in node 2 Go left in node 5 Go left in node 11 Go left in node 23 There is no such node in tree, go out of the function Go right in node 24 There is no such node in tree, go out of the function Returning sum of left and right Go right in node 12 Go left in node 25 There is no such node in tree, go out of the function Go right in node 26 There is no such node in tree, go out of the function	

		Returning sum of left and right Returning sum of left and right Go right in node 6 Go left in node 13 Found value Go left in node 27 There is no such node in tree, go out of the function Go right in node 28 There is no such node in tree, go out of the function Returning sum of left and right Go right in node 14 Go left in node 29 There is no such node in tree, go out of the function Go right in node 30 There is no such node in tree, go out of the function Returning sum of left and right Returning sum of left and right Returning sum of left and right Returning sum of left and right There are 2 or more of 1	

Выводы.

Был изучен алгоритм обработки дерева, была создана программа, которая создает список всех вхождений одного дерева в другое.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: BinTree.cpp

```
#include "BinTree.h"

int levelsToMasSize(int inp)
{
   if (inp == 0) { return 1;}
   inp--;
   return (1 + 2* levelsToMasSize(inp));
}

template <typename T>
BinTree<T>::BinTree(unsigned int inputLevels) // Counting from 0
{
   levels = inputLevels;
   treeData = new T[levelsToMasSize(inputLevels)];
}

template <typename T>
BinTree<T>::BinTree(const BinTree<T> & binTree) // Copy operator
{
   treeData = new T[levelsToMasSize(levels)];
   for (int i=0; i< levelsToMasSize(levels); i++)
   {
      treeData[i] = binTree.treeData[i];
   }
}

template <typename T>
BinTree<T>::~BinTree()
{
   if (treeData != nullptr)
   {
      delete treeData;
   }
}

template <typename T>
void BinTree<T>::viewData()
{
   int size = levelsToMasSize(levels);
   for (int i=0; i<size; i++)
   {
      cout << treeData[i] << ' ';
   }
   cout << '\n';
}


template <typename T>
void BinTree<T>::addByDataAddress(int address, T value)
{
   if (address <= levelsToMasSize(levels))
      treeData[address] = value;
}
```

```
template <typename T>
void BinTree<T>::addByTreeAddress(string address, T value)
{
   int dataAddress = treeAddressToDataAddress(address);
   addByDataAddress(dataAddress, value);
}

int binToDec(int inpBin)
{
   int decimalNumber = 0;
   int base = 1;
   int temp = inpBin;

   while (temp)
   {
      int lastDigit = temp % 10;
      temp = temp/10;
      decimalNumber += lastDigit*base;
      base = base*2;
   }
   return decimalNumber;
}

int treeAddressToDataAddress(string address)
{
   if (address.empty())
   {
      return 0;
   }

   int offset = (levelsToMasSize(address.size() -1)) ;
   int binAddress = atoi(address.c_str());
   return (offset + binToDec(binAddress));
}

unsigned int RECURSION_DEEPNESS = 0;

template <typename T>
int BinTree<T>::howManyRecursiveHandler(int node, T value)
{
   RECURSION_DEEPNESS++;
   string buffer(RECURSION_DEEPNESS, ' ');

   cout << buffer << "in node " << node << '\n';

   int result = 0;
   if (node >= levelsToMasSize(levels))
   {
      cout << buffer << "There is no such node in tree, go out of the function\n";
      RECURSION_DEEPNESS--;
      return result;
   }

   if (treeData[node] == value)
   {
      cout << buffer << "Found value\n";
      result++;
   }
```

```
    int left = (node<<1) +1;
    int right = (node<<1) +2;

    cout << buffer << "Go left\n";
    unsigned int res1 = howManyRecursiveHandler(left, value);
    cout << buffer << "Go right\n";
    unsigned int res2 = howManyRecursiveHandler(right, value);
    cout << buffer << "Returning sum of left and right\n";
    result += res1 + res2;

    RECURSION_DEEPNESS--;
    return result;
}

template <typename T>
int BinTree<T>::howManyRecursive(T value)
{
    if (value == 0)
        return 0;
    cout << "Call howManyRecursiveHandler\n";
    return howManyRecursiveHandler(0, value);
}

template <typename T>
bool BinTree<T>::isContainsAtLeastTwoRecursive(T value)
{
    cout << "Call howManyRecursive\n";
    return (howManyRecursive(value) >= 2);
}

void inputHandler(BinTree<int> tree, int inputValue)
{
    if (inputValue <= 0)
    {
        cout << "inputValue can't be <= 0\n";
        return;
    }

    if (tree.isContainsAtLeastTwoRecursive(inputValue)) {
        cout << "There are 2 or more of " << inputValue << '\n';
    } else {
        cout << "There is 1 or less of " << inputValue << '\n';
    }
}

void fileInputCase(string path)
{
    ifstream inFile;
    inFile.open(path);

    cout << "Tree deepth ";
    int inputLevels;

    while (inFile >> inputLevels)
    {
        BinTree<int> tree(inputLevels);

        string inputAddress = "empty";
        int inputValue = 0;
```

```
        while (inputValue != 0 || !(inputAddress.empty()))
        {
            tree.addByTreeAddress(inputAddress, inputValue);
            tree.viewData();
            cout << "\nNode value and address (0 empty to stop input, empty for empty address) ";
            inFile >> inputValue;
            inFile >> inputAddress;
            if (inputAddress == "empty") { inputAddress = ""; }
            cout << "\ninputValue is " << inputValue;
            cout << "\ninputAddress is " << inputAddress << '\n';
        }

        cout << "\nValue to find, is there >= 2 of them \n";
        if(inFile >> inputValue)
        {
            if (inputValue <= 0)
            {
                cout << "inputValue can't be <= 0\n";
            }
            else
            {
                if (tree.isContainsAtLeastTwoRecursive(inputValue)) {
                    cout << "There are 2 or more of " << inputValue << '\n';
                } else {
                    cout << "There is 1 or less of " << inputValue << '\n';
                }
            }
        }
        cout << "\n\nTree deepth ";
    }
    inFile.close();
}

void stdInputCase() // No obvious ways to unite std input and file input case
{
    cout << "Tree deepth ";
    int inputLevels;

    while (cin >> inputLevels && inputLevels >= 0)
    {
        BinTree<int> tree(inputLevels);

        string inputAddress = "empty";
        int inputValue = 0;

        while (inputValue != 0 || !(inputAddress.empty())) // empty exist because we need a way to call the first
node
        {
            tree.addByTreeAddress(inputAddress, inputValue);
            tree.viewData();
            cout << "\nNode value and address (0 empty to stop input, empty for empty address) ";
            cin >> inputValue;
            cin >> inputAddress;
            if (inputAddress == "empty") { inputAddress = ""; }
            cout << "\ninputValue is " << inputValue;
            cout << "\ninputAddress is " << inputAddress << '\n';
        }
```

```
        cout << "\nValue to find, is there >= 2 of them ";
        if (cin >> inputValue)
        {
            if (inputValue <= 0)
            {
                cout << "inputValue can't be <= 0\n";
            }
            else
            {
                if (tree.isContainsAtLeastTwoRecursive(inputValue)) {
                    cout << "There are 2 or more of " << inputValue << '\n';
                } else {
                    cout << "There is 1 or less of " << inputValue << '\n';
                }
            }
        }

        cout << "\n\nTree deepth ";
    }
}

void introductionMessageView()
{
    cout << "File input example: ./main -f test.txt\n\n";
    cout << "Examples of input see in inputExamples file\n";
    cout << "0 == empty node of tree\n";
    cout << "-1 in tree depth to exit\n";

    cout << "\n0 in address == left node\n1 == right node\n";
}

int main(int argc, char *argv[])
{
    cout << "\n\n\n";
    if (argc>= 2) // Arguments case
    {
        string flag(argv[1]);
        string path(argv[2]);
        if (flag.compare("-f") == 0)
            fileInputCase(path); // No obvious way to overload the function
        return 0;
    }
    introductionMessageView();
    stdInputCase();
    return 0;
}
```

Название файла: BinTree.h

```
#pragma once


#include <iostream>
#include <string.h>
#include <csignal>
```

```
#include <fstream>

using namespace std;

template <typename T>
class BinTree
{
private:
   int levels;
   T* treeData;

void addByTreeAddress(string address, int offset, T value);

public:
   BinTree(unsigned int levels = 0);
   BinTree(const BinTree<T> & binTree);
   ~BinTree();
   void addByDataAddress(int address, T value);
   void addByTreeAddress(string address, T value);
   void viewData();

   int howManyRecursiveHandler(int node, T value);
   int howManyRecursive(T value);

   bool isContainsAtLeastTwoRecursive(T value);};

int treeAddressToDataAddress(string address);
```