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

ОТЧЕТ

По лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: **Программирование алгоритмов с бинарными деревьями**

Студент гр. 9381 Судаков Е.В.

Преподаватель Фирсов М.А.

Санкт-Петербург
2020

1. Цель работы.

Ознакомиться с понятием деревьев в компьютерных науках, реализовать соответствующие алгоритмы и структуры данных.

2. Задание.

Вариант 4в.

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

3. 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 0 попарно не пересекающихся множествах T_1, T_2, ..., T_m, каждое из которых, в свою очередь, является деревом. Деревья T_1, T_2, ..., T_m называются *поддеревьями* данного дерева.

В данной работе дерево представляется на базе целочисленного массива. Таким образом, чтобы узнать индекс потомка элемента по индексу x, нужно взять элементы tree[2*x] и tree[2*x + 1] для левого и правого потомка соответственно.

Наиболее эффективным видом дерева для решения задачи данной работы есть **Бинарное дерево поиска**.

Бинарное дерево поиска обладает следующим свойством: если x — узел бинарного дерева с ключом k, то все узлы в левом поддереве должны иметь ключи, меньшие k, а в правом поддереве большие k.

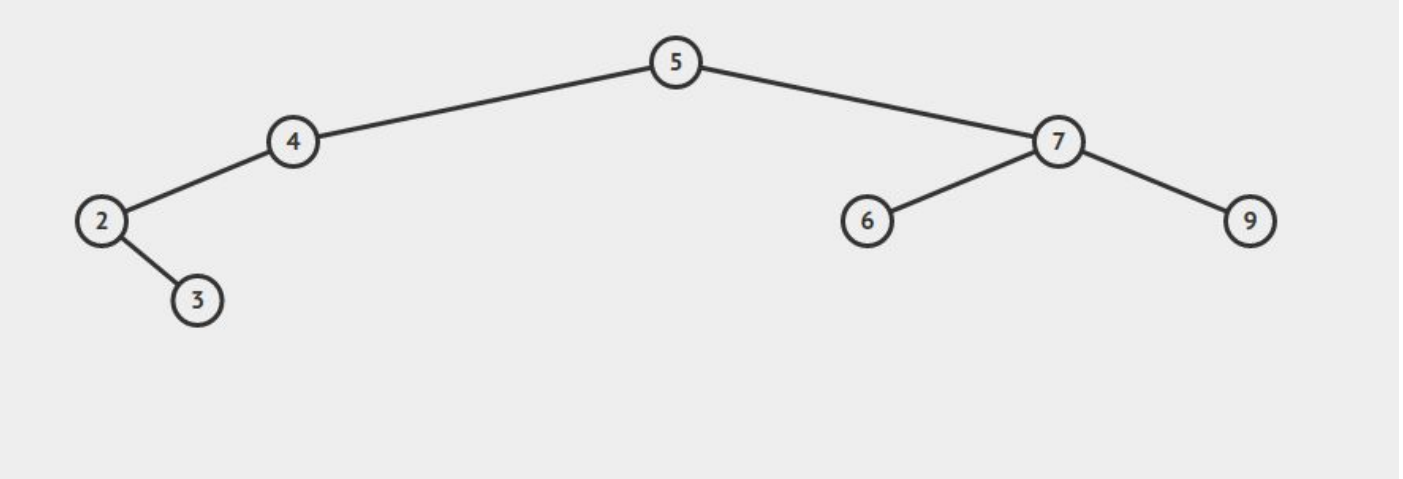

Рисунок 1. Бинарное дерево поиска из основного теста

4. Пример работы программы:

Основной тест №1:

Входные данные: 5 4 7 2 6 3 9

Выходные данные (с промежуточной информацией):

```
Вставка : 5

Создан корень 5



Вставка : 4

Спускаемся в левый потомок. Текущий корень : 5

      Добавлен элемент 4 слева от 5



Вставка : 7

Спускаемся в правый потомок. Текущий корень : 5

      Добавлен элемент 7 справа от 5



Вставка : 2
```

Спускаемся в левый потомок. Текущий корень : 5

Спускаемся в левый потомок. Текущий корень : 4

Добавлен элемент 2 слева от 4

Вставка : 6

Спускаемся в правый потомок. Текущий корень : 5

Спускаемся в левый потомок. Текущий корень : 7

Добавлен элемент 6 слева от 7

Вставка : 3

Спускаемся в левый потомок. Текущий корень : 5

Спускаемся в левый потомок. Текущий корень : 4

Спускаемся в правый потомок. Текущий корень : 2

Добавлен элемент 3 справа от 2

Вставка : 9

Спускаемся в правый потомок. Текущий корень : 5

Спускаемся в правый потомок. Текущий корень : 7

Добавлен элемент 9 справа от 7

Все элементы в дереве различны

(5, 1) (4, 1) (7, 1) (2, 1) (0, 0) (6, 1) (9, 1) (0, 0) (3, 1)

Process finished with exit code 0

Дополнительное тестирование :

Номер теста	Входные данные	Результат
2	5 4 7	Все элементы в дереве различны (5, 1) (4, 1) (7, 1)
3	5 1 6 3 0 5	Дерево содержит одинаковые элементы ! (5, 2) (1, 1) (6, 1) (0, 1) (3, 1)
4	1 1 1	Дерево содержит одинаковые элементы ! (1, 3)
5	1 2 3	Все элементы в дереве различны (1, 1) (0, 0) (2, 1) (0, 0) (0, 0) (0, 0) (3, 1)

4. Выполнение программы:

1. Для ввода информации из файла необходимо ввести “1” на вопрос программы “Строка из консоли или из файла (0/1)?”.
2. Для ввода информации через консоль необходимо ввести “0” на вопрос программы “Строка из консоли или из файла (0/1)?”.

Программе подается на вход единственная строка для разбора.

5. Описание функций:

Все функции описаны в исходном коде в стиле Javadoc.

`friend ostream &operator<<(ostream &output, const Node &node);` -

Перегружненный оператор вывода элемента в строковой поток.

@param output ссылка на поток

@param Node - объект класса элемента

BST::BST(vector<int> input) -Конструктор бинарного дерева поиска

@param input интовый вектор с элементами. По факту бинарное дерево.

Из этого бинарного дерева делаем бинарное дерево поиска.

void BST::addLeave(Node &root, Node add, Node parent, bool fromLeft) - Функция добавление листа с удобным выводом.

@param root куда добавляем

@param add собственно лист

@param parent предок листа

@param fromLeft слева или справа ?

void BST::insert(Node node) - Функция вставки элемента в бинарное дерево поиска

Сравнивая элемент для вставки с текущим рутом спускаемся вниз до тех пор,

пока не обнаружим у элемента отсутствие ребенка, тогда нужно подвесить на него вставляемый элемент.

@param node элемент для вставки

6. Описание структур данных

Класс элемента дерева. Оператор вывода перегружен.

```
class Node {

public:

    int count = 0; // количество вхождений в исходное бинарное
дерево

    int value;  //значение

    bool created = false; // создан ли элемент по этому индексу в
бинарном дереве поиска

    Node(int x) : value(x), count(0) {};

    Node() : value(0), count(0) {};

    friend ostream &operator<<(ostream &output, const Node &node);

};
```

Класс Бинарного дерева поиска на векторе. Левый потомок элемента tree[x] := tree[2*x + 1]

Правый потомок элемента tree[x] := tree[2*x + 2]

```
class BST {
public:
    BST(vector<int> input);
```

```
    void print();


private:
    Node tree[sz];
    int currentSize = 0;
    bool isSimple = true;
    void insert(Node node);
    void addLeave(Node &node, Node add, Node parent, bool
fromLeft);
};
```

8. Вывод:

В ходе выполнения лабораторной работы была создана программа, реализующее бинарное дерево поиска.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```
 #include <bits/stdc++.h>


using namespace std;


#define sz 10005


/**
 * Класс элемента дерева
 */
class Node {
public:
    int count = 0; // количество вхождений в исходное бинарное дерево
    int value;  //значение
    bool created = false; // создан ли элемент по этому индексу в бинарном дереве поиска


    Node(int x) : value(x), count(0) {};
    Node() : value(0), count(0) {};


    friend ostream &operator<<(ostream &output, const Node &node);
};


/**
 * Перегружненный оператор вывода элемента в строковой поток.
 * @param output output ссылка на поток
 * @param node  объект класса элемента
 */
ostream &operator<<(ostream &output, const Node &node) {
```

```
    output << "(" << node.value << ", " << node.count << " ) ";
    return output;
}

/**
 * Реализация Бинарного дерева поиска на векторе.
 * Левый потомок элемента tree[x] := tree[2*x + 1]
 * Правый потомок элемента tree[x] := tree[2*x + 2]
 */
class BST {
public:

    BST(vector<int> input);

    void print();

private:
    Node tree[sz];
    int currentSize = 0;
    bool isSimple = true;

    void insert(Node node);

    void addLeave(Node &node, Node add, Node parent, bool fromLeft);
};

/**
 * Конструктор бинарного дерева поиска
 * @param input интовый вектор с элементами. По факту бинарное дерево.
 * Из этого бинарного дерева делаем бинарное дерево поиска.
 */
```

```
BST::BST(vector<int> input) {
    for (auto x : input) {
        BST::insert(x);
    }
    if (!isSimple)
        cout << "\nДерево содержит одинаковые элементы !\n";
    else
        cout << "\nВсе элементы в дереве различны \n";
}

void printDepth(int depth) {
    for (int i = 0; i < depth; i++) cout << "\t";
}

/**
 * Функция добавление листа с удобным выводом.
 * @param root куда добавляем
 * @param add  собственно лист
 * @param parent предок листа
 * @param fromLeft слева или справа ?
 */
void BST::addLeave(Node &root, Node add, Node parent, bool fromLeft) {
    if (parent.value == 0) cout << "Создан корень " << add.value << "\n\n";
    else
        cout << "Добавлен элемент " << add.value << (fromLeft ? " слева от " : " справа
от ") << parent.value << "\n\n";
    root = add;
    root.created = true;
    root.count++;
}

/**
```

```
 * Функция вставки элемента в бинарное дерево поиска

 * Сравнивая элемент для вставки с текущим рутом спускаемся вниз до тех пор,

 * пока не обнаружим у элемента отсутствие ребенка, тогда нужно подвесить на него вставляемый элемент.

 * @param node элемент для вставки

 */

void BST::insert(Node node) {

    int root = 1;

    int level = 0;

    int parent = root;

    bool fromLeft = false;

    cout << "Вставка : " << node.value << "\n";

    while (true) {

        printDepth(level);

        int right = root * 2 + 1;

        int left = root * 2;

        if (tree[root].value < node.value) {

            if (right < sz && tree[root].created) {

                cout << "Спускаемся в правый потомок. Текущий корень : " << tree[root].value << "\n";

                parent = root;

                root = right;

                fromLeft = false;

            } else {// иначе корень - уже лист. Вставляем

                addLeave(tree[root], node, tree[parent], fromLeft);

                currentSize = max(currentSize, root);

                break;

            }

        } else if (tree[root].value > node.value) {

            if (left < sz && tree[root].created) {

                cout << "Спускаемся в левый потомок. Текущий корень : " << tree[root].value << "\n";

                parent = root;
```

```
                root = left;
                fromLeft = true;
            }
        } else {
            tree[root].count += 1; // увеличиваем число вхождений
            cout << "Элемент со значением " << node.value << " встречается уже " <<
tree[root].count << " раз !\n";
            isSimple = false;
            break;
        }
        level++;
    }

}


void BST::print() {
    for (int i = 1; i <= currentSize; i++) {
        cout << tree[i] << " ";
    }
}


int main() {
    int f;
    cout << "Строка из консоли или из файла (0/1)?\n";
    cin >> f;
    if (f == 1) {
        freopen("input.txt", "r", stdin);
    } else cout << "Введите строку :\n";
    char c;
    cin >> c;
```

```
    string input;
    getline(cin, input);
    input = c + input;
    std::stringstream iss(input);


    int number;
    std::vector<int> treeNumbers;
    while (iss >> number)
        treeNumbers.push_back(number);


    BST tree(treeNumbers);
    tree.print();
    return 0;
}
```