МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Отчет по выполнению практического задания №3

По дисциплине «Структуры и алгоритмы обработки данных»

Тема:

Нелинейные структуры данных. Бинарное дерево

Выполнил студент Сидоров С.Д.

группа ИКБО-20-21

Тема. Нелинейные структуры данных. Бинарное дерево

Цель. Получить навыки по разработке хеш-таблиц и их применении.

Задание: Разработать программу, которая создает идеально сбалансированное

дерево из n узлов и выполняет операции.

1. Реализовать операции общие для всех вариантов

1) Создать идеально сбалансированное бинарное дерево из n узлов. Структура узла дерева включает: информационная часть узла, указатель на левое и указатель на правое поддерево. Информационная часть узла определена вариантом.

2) Отобразить дерево на экране, повернув его справа налево.

2. Реализовать операции варианта.

3. Разработать программу на основе меню, позволяющего проверить выполнение всех операций на ваших тестах и тестах преподавателя.

4. Оформить отчет.

1) Для каждой представленной в программе функции предоставить отчет по ее разработке в соответствии с требованиями разработки программы (подпрограммы).

2) Представить алгоритм основной программы и таблицу имен, используемых в алгоритме.

Вариант 21 (1): Значение информационной части - целое число. Операции варианта: Определить высоту дерева. Определить длину пути дерева (количество ребёр), используя алгоритм прямого обхода. Вычисляет среднее арифметическое всех чисел в дереве.

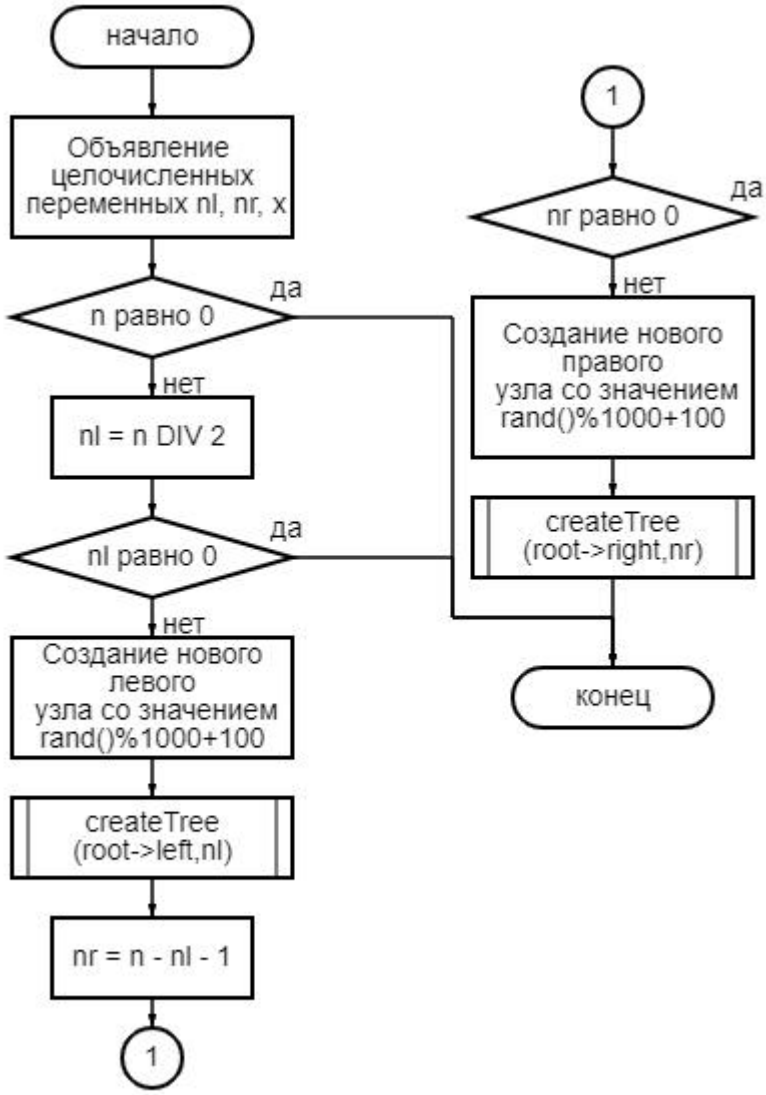
Разработка:

Функция создания дерева:

Прототип: void createTree(node* root)

Предусловие: node* root - указатель на текущий элемент дерева

Алгоритм:



Реализация: рекурсивно создание сбалансированного дерева(листинг 1)

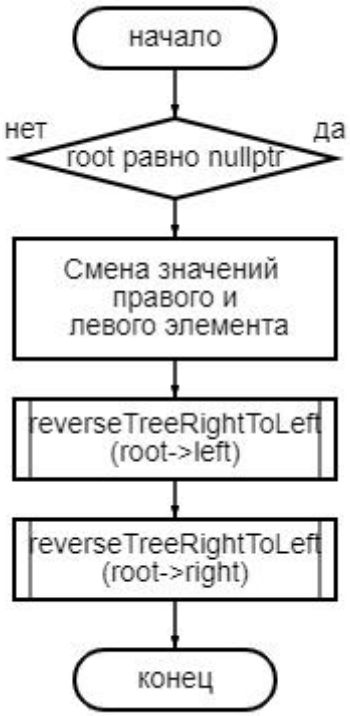
```
void createTree(node* root,int n) {
                    int nl, nr, x;
                    if (n == 0) return;

                    nl = n / 2;
                    if (nl == 0) return;
                    root->left = new node(rand()%1000+100);
                    createTree(root->left,nl);

                    nr = n - nl - 1;
                    if (nr == 0) return;
                    root->right = new node(rand() % 1000 + 100);
                    createTree(root->right, nr);
            }
```

Листинг 1

Функция переворота дерева с права на лево:

Прототип: reverseTreeRightToLeft(node* root)

Предусловие: node* root - указатель на текущий головной элемент.



Реализация: рекурсивная смена местами правого и левого листа каждого поддерева. (Листинг 2)

```
void print( node * root, int level) {


                    if (root == nullptr) return;
                    print( root->right, level + 1);
                    for (int i = 0; i < level; i++) cout << "\t\t";
                    cout << " "  << root->information_node << "\n";
                    cout;
                    print(root->left, level + 1);
            }
```

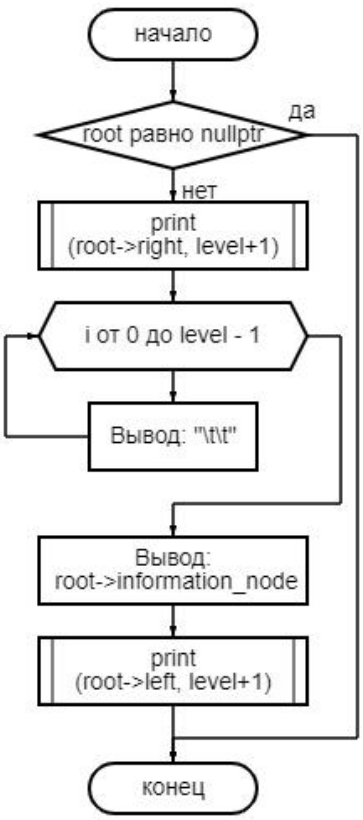
Листинг 2

Функция вывода дерева на экран:

Прототип: void print(node* root, int level);

Предусловие: node* root - указатель на текущий головной объект

Алгоритм:



Реализация: рекурсивный вывод дерева, начиная с крайнего правого поддерева (Листинг 3).

```
void print( node * root, int level) {
                    if (root == nullptr) return;
                    print( root->right, level + 1);
                    for (int i = 0; i < level; i++) cout << "\t\t";
                    cout << " "  << root->information_node << "\n";
                    print(root->left, level + 1);
             }
```

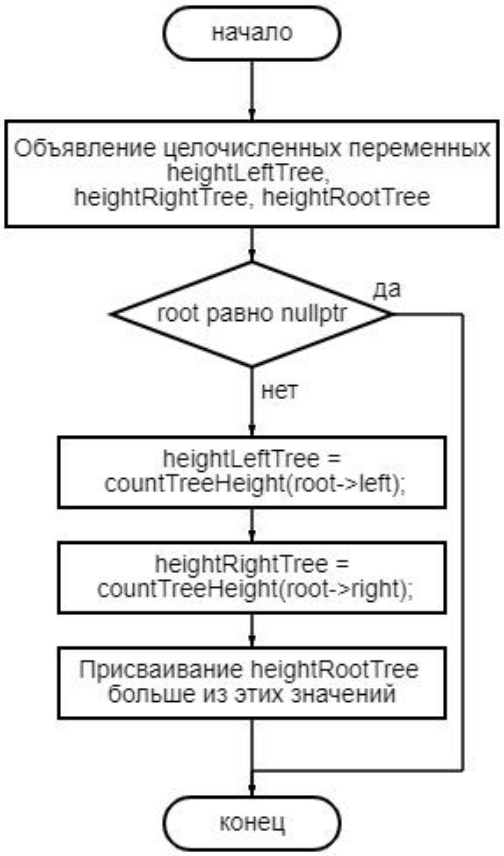
Листинг 3

Функция для подсчета высоты дерева:

Прототип: int countTreeHeight(node* root);

Предусловие: node* root - текущий головной элемент.

Алгоритм:



Реализация: рекурсивный подсчёт высоты каждого поддерева и выбор наибольшего из них.(Листинг 4)

```
int countTreeHeight(node* root) {
                    int heightLeftTree, heightRightTree, heightRootTree = 0;

                    if (root == nullptr) {
                            heightLeftTree = countTreeHeight(root->left);
                            heightRightTree = countTreeHeight(root->right);
                            heightRootTree = ((heightLeftTree > heightRightTree) ?
heightLeftTree : heightRightTree) + 1;
                    }

                    return heightRootTree;
            }
```

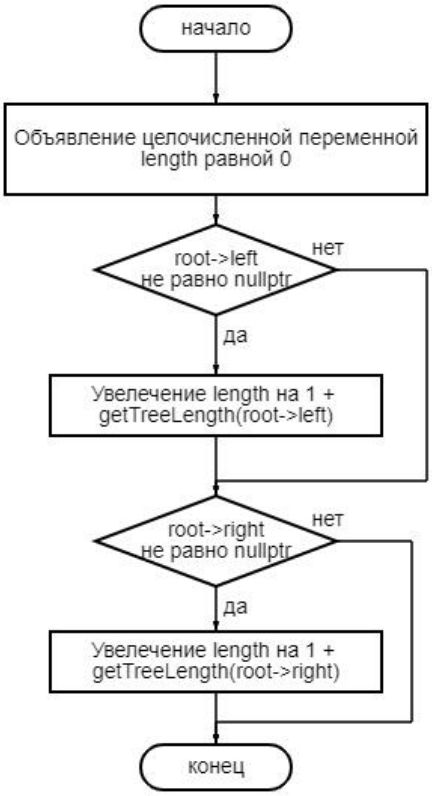
Листинг 4

Функция для вычисления длины дерева:

Прототип: int getTreeLength(node* root);

Предусловие: node* root - текущий головной элемент

Алгоритм:



Реализация: рекурсивный подсчёт каждого ребра в дереве (Листинг 5)

```
int getTreeLength(node* root) {
                    int length = 0;
                    if (root->left != nullptr)
                           length += 1 + getTreeLength(root->left);
                    if (root->right != nullptr)
                           length += 1 + getTreeLength(root->right);

                    return length;
             }
```

Листинг 5

Функция для подсчёта среднего арифметического от всех элементов:

Прототип: double getAverage(node* root);

Предусловие: node* root - текущий головной элемент.

Реализация: рекурсивное получение значения информационного узла и деление на количество элементов дерева(Листинг 6).

```
double getAverage() {
             return ((double)getValuesSum(this->root)) / (getTreeLength(this->root) + 1);
             }

int getValuesSum(node* root) {
             return (root == nullptr) ? 0 : root->information_node + getValuesSum(root->left) +
getValuesSum(root->right);
             }
```

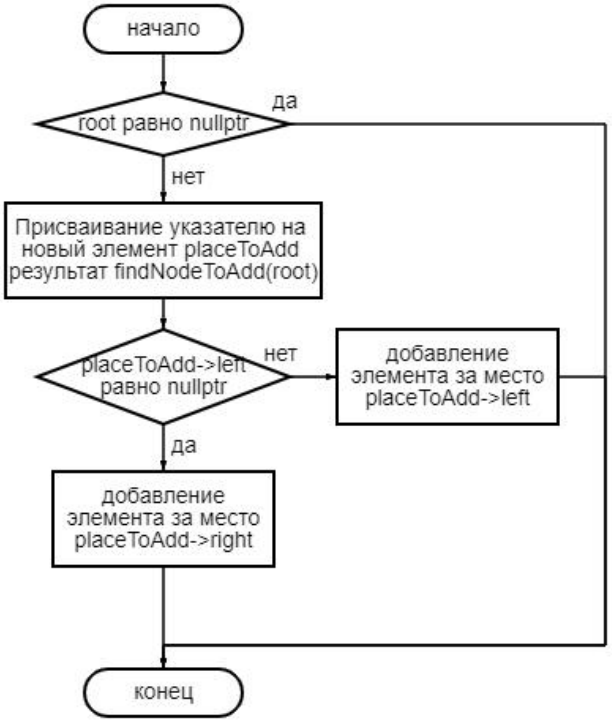
Листинг 6

Функция для добавления элемента в дерево:

Прототип: *void add(node* root)*;

Предусловие: node* root - текущий головной элемент

Алгоритм:



Реализация: создание нового элемента на место, найденное с помощью рекурсивной функции findNodeToAdd, описанной далее (Листинг 7).

```
void add(node* root) {
                    if (root == nullptr) return;


                    node* placeToAdd = findNodeToAdd(root);


                    if (placeToAdd->left == nullptr) placeToAdd->left = new
node(rand() % 1000 + 100);
                    else placeToAdd->right = new node(rand() % 1000 + 100);
            }
```

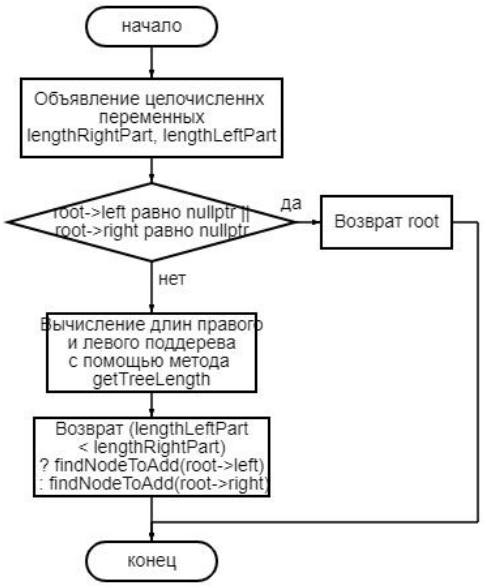
Листинг 7

Функция нахождения места для добавления элемента:

Прототип : node* findNodeToAdd(node* root);

Предусловие: node* root - текущий головной элемент;

Алгоритм:



Реализация: поиск места для добавления элемента, посредством вычисления длины каждого поддерева, и прохождении вниз по наименьшему пути (Листинг 8).

```
node* findNodeToAdd(node* root) {
	int lengthRightPart, lengthLeftPart;
	if (root->left == nullptr || root->right == nullptr) {
		return root;
	}
	lengthLeftPart = getTreeLength(root->left);
	lengthRightPart = getTreeLength(root->right);
	if (lengthLeftPart == lengthRightPart) {
		node* currentNode = root;
		while (currentNode->left != nullptr) {
			currentNode = currentNode->left;
		}
		return currentNode;
	}
	return (lengthLeftPart < lengthRightPart) ? findNodeToAdd(root->left) : findNodeToAdd(root->right);
}
```

Листинг 8

Код основной программы содержится в файле main.pdf и код содержащий класс Tree содержится в файле balanced_tree.pdf.

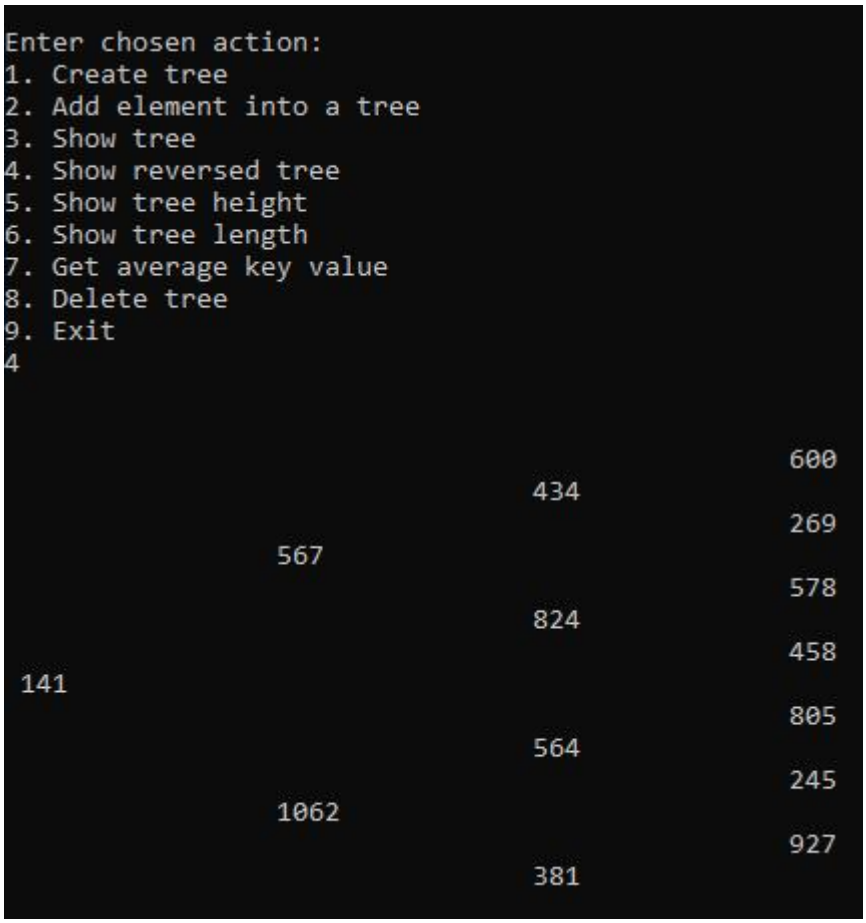
Тестирование:

Для тестирования каждая из необходимых команд была выполнена с помощью текстового меню для взаимодействия с пользователем. Результаты тестирования представлены далее.

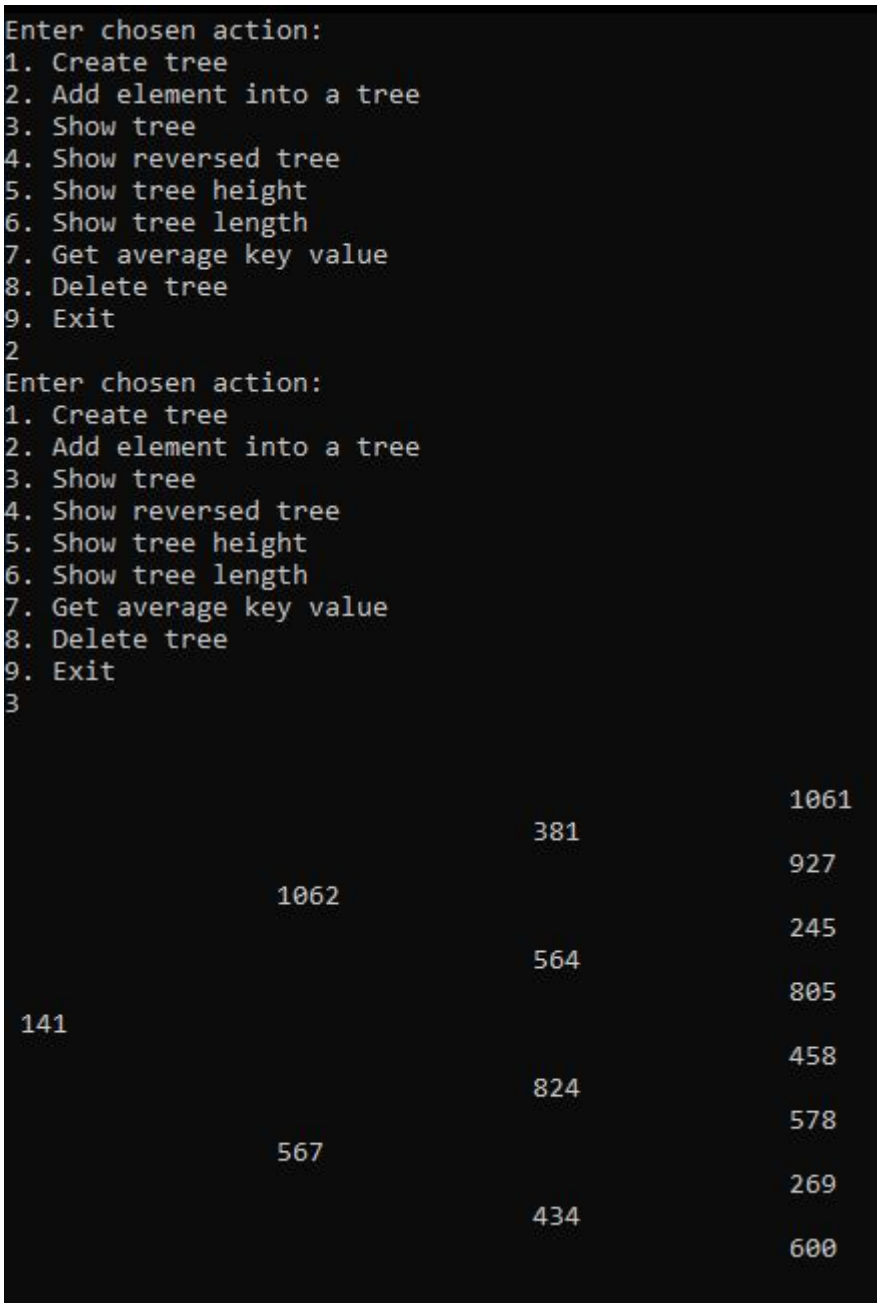
Создание и вывод дерева:

```
Enter chosen action:
1. Create tree
2. Add element into a tree
3. Show tree
4. Show reversed tree
5. Show tree height
6. Show tree length
7. Get average key value
8. Delete tree
9. Exit
1

Enter amount of elements:
14
Enter chosen action:
1. Create tree
2. Add element into a tree
3. Show tree
4. Show reversed tree
5. Show tree height
6. Show tree length
7. Get average key value
8. Delete tree
9. Exit
3


                              381
                                        927
                1062
                                        245
                              564
                                        805
 141
                                        458
                              824
                                        578
                567
                                        269
                              434
                                        600
```

Вывод перевёрнутого дерева:



Добавление элемента:



Операции варианта и удаление дерева:

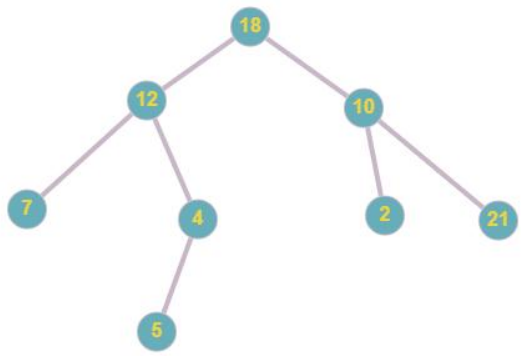
```
Tree height: 4
Enter chosen action:
1. Create tree
2. Add element into a tree
3. Show tree
4. Show reversed tree
5. Show tree height
6. Show tree length
7. Get average key value
8. Delete tree
9. Exit
6

Tree length: 14
Enter chosen action:
1. Create tree
2. Add element into a tree
3. Show tree
4. Show reversed tree
5. Show tree height
6. Show tree length
7. Get average key value
8. Delete tree
9. Exit
7

Average: 594.4
Enter chosen action:
1. Create tree
2. Add element into a tree
3. Show tree
4. Show reversed tree
5. Show tree height
6. Show tree length
7. Get average key value
8. Delete tree
9. Exit
8

Successfully deleted
Enter chosen action:
1. Create tree
2. Add element into a tree
3. Show tree
4. Show reversed tree
5. Show tree height
6. Show tree length
7. Get average key value
8. Delete tree
9. Exit
```

Ответы на вопросы:

1. Степень дерева - это максимальная степень вершин, входящих в дерево.
2. Степень сильноветвящегося дерева - произвольная.
3. Путь в дереве определяет последовательность узлов от корня до нужно узла.
4. Длину пути в дереве можно посчитать посчитав сумму длин пути его ребер.
5. Степень бинарного дерева - небольше двух.
6. Дерево может быть пустым, если оно не содержит ни одной вершины.
7. Бинарное дерево - дерево у каждого узла, которого не более двух потомков
8. Обход дерева - вид обхода графа, обуславливающий процесс посещения каждого узла структуры дерева ровно один раз.
9.

$$Hight(T) = \begin{cases} -1 \text{ если } T \text{ пусто} \\ 1 + \max(Hight(LeftTree(T)), Hight(RightTree(T))) \end{cases}$$

10.



11. Прямой ход: ABDGCEHIF, Обратный ход: GDBHIEFCA, Симметричный: DGBAHEICF
12. Очередь
13.

ШАГ	Очередь	Вывод
1	A	
2	BC	A
3	CD	AB
4	DEF	ABC

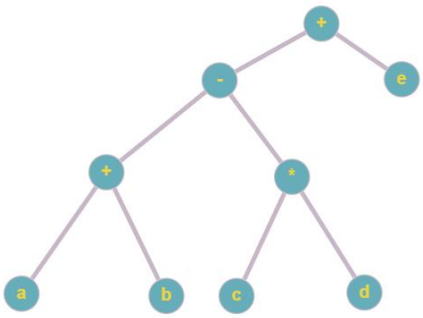
5	EFG	ABCD
6	FGHI	ABCDE
7	GHI	ABCDEF
8	HI	ABCDEF
9	I	ABCDEFH
10		ABCDEFHI

14. Стек

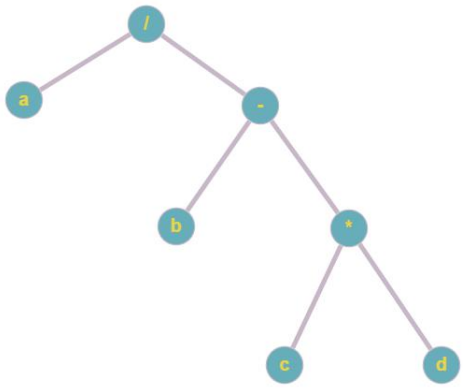
15. Прямой обход: **-+a/*bcde**, Обратный обход: **a+b*c/d-e,** Симметричный обход**: abc*d/+e-**

16.

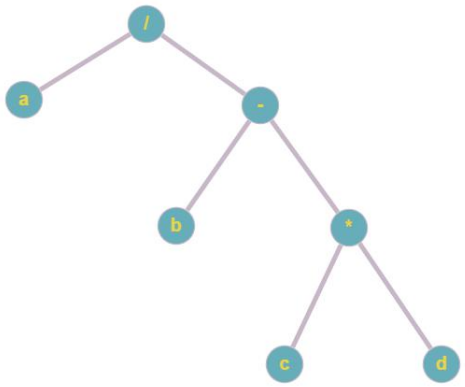
1.



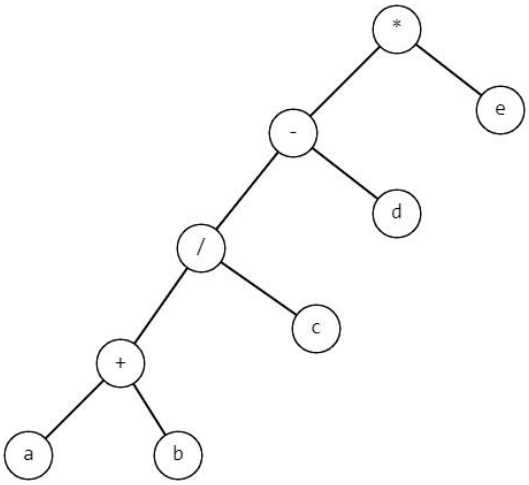
2.



3.



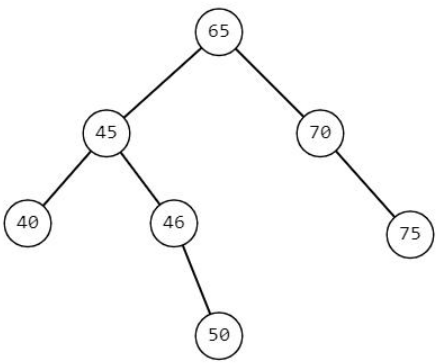
4.



17. Бинарное дерево будет обходится в прямом порядке

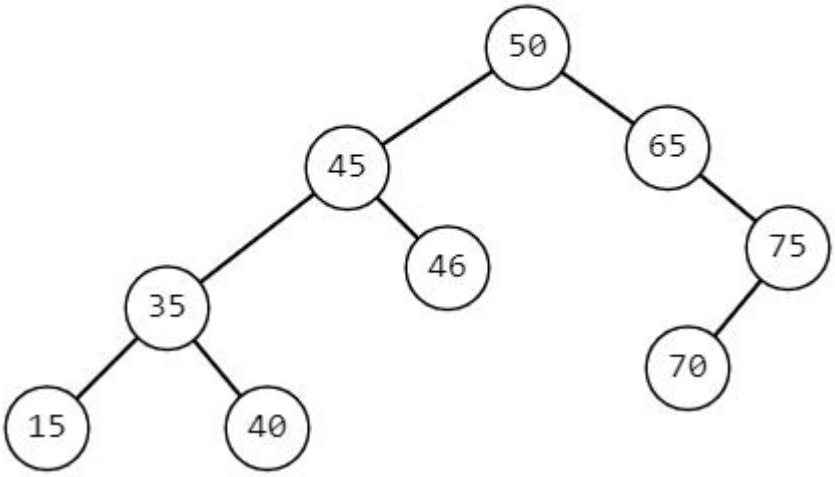
18.

```
40 45 46 50 65 70 75
```



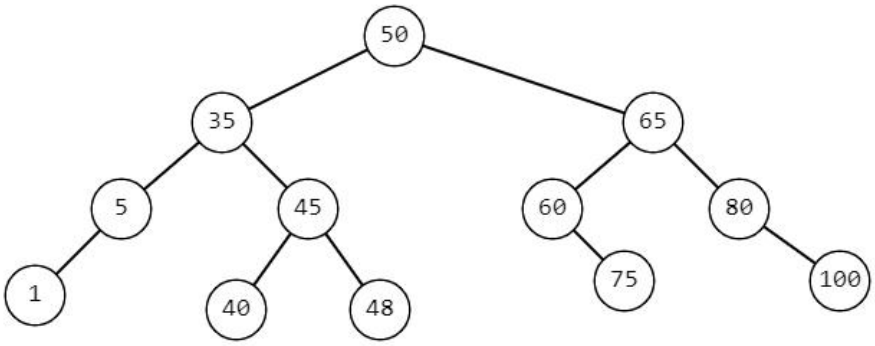
19.

50 45 35 15 40 46 65 75 70

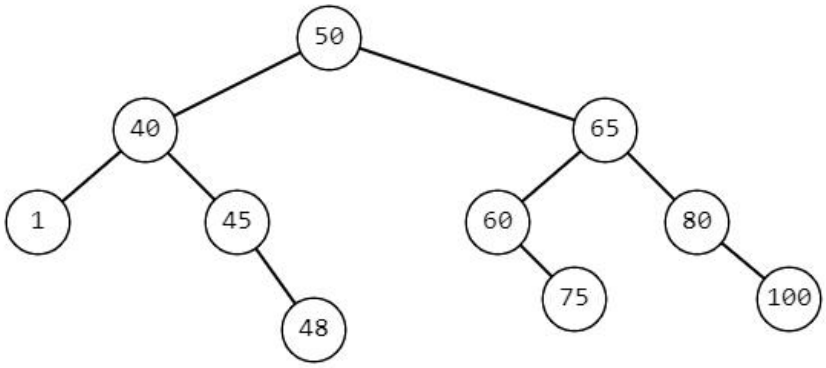


20.

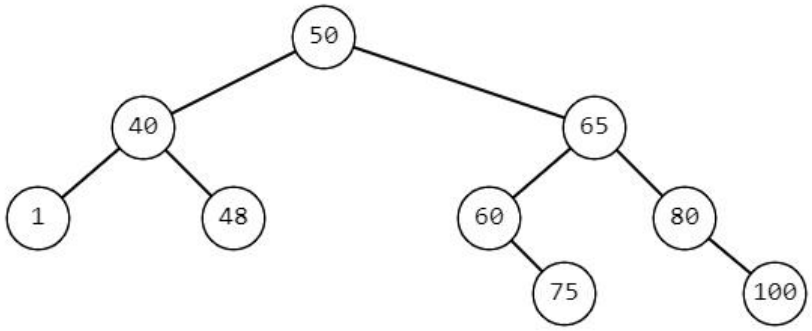
20.1.



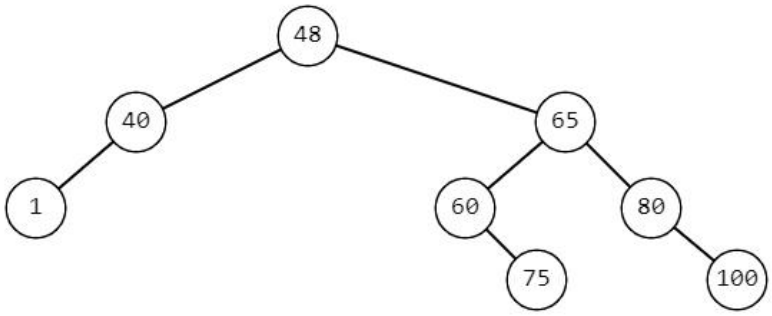
20.2.



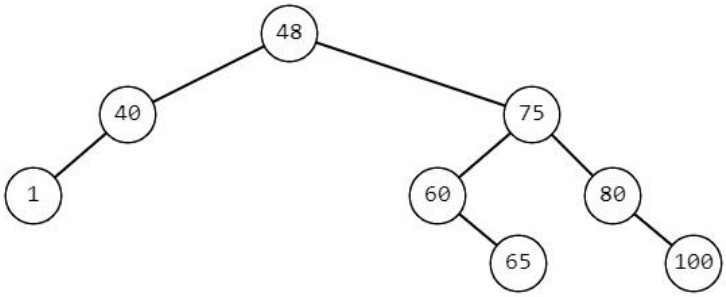
20.3.


50
40
65
1
48
60
80
75
100

20.4.


48
40
65
1
60
80
75
100

20.5.


48
40
75
1
60
80
65
100

Выводы:

В ходе выполнения данной работай был реализован класс представляющий собой структуру хранения - идеально сбалансированное дерево. Также были получены навыки и умения разработки и реализации операций над данной структурой.

Список литературы:

- Лекции по структурам и алгоритмам обработки данных Рысин М.Л.
- Методическое пособие по выполнению задания 2