МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации
информационных технологий

Отчет по практической работе №4

по дисциплине «Структуры и алгоритмы обработки данных»
по теме «Нелинейные структуры данных. Бинарное дерево»

Выполнил:
Студент группы ИКБО-13-22 Тринеев Павле Сергееви

Проверил: ассистент Муравьёва Е.А.

МОСКВА 2023 г.

Практическая работа 4

Тема: Нелинейные структуры данных. Бинарное дерево.

Цель: получение умений и навыков разработки и реализаций операций над структурой данных бинарное дерево.

Задание 1. Ответьте на вопросы и выполните упражнения:

1. Что определяет степень дерева?

Степень дерева определяется количеством потомков, или детей, у каждой вершины (узла) в дереве.

2. Какова степень сильноветвящегося дерева?

Сильноветвящееся дерево — это дерево, у которого вершины имеют большое количество потомков (детей). Степень сильноветвящегося дерева определяется как максимальное количество потомков, которое имеет вершина в этом дереве.

3. Что определяет путь в дереве?

Путь в дереве определяется последовательностью вершин, которые соединяются друг с другом от корня до определенной целевой вершины внутри дерева.

4. Как рассчитать длину пути в дереве?

Длина пути в дереве может быть определена как количество вершин (узлов)

5. Какова степень бинарного дерева?

Степень бинарного дерева определяется максимальным числом потомков, которое может иметь вершина (узел) в этом дереве.

6. Может ли дерево быть пустым?

Да, дерево может быть пустым. Пустое дерево не содержит ни одной вершины (узла)

7. Дайте определение бинарного дерева?

Бинарное дерево — это иерархическая структура данных, в которой каждая вершина (узел) имеет не более двух потомков: левого и правого.

8. Дайте определение алгоритму обхода.

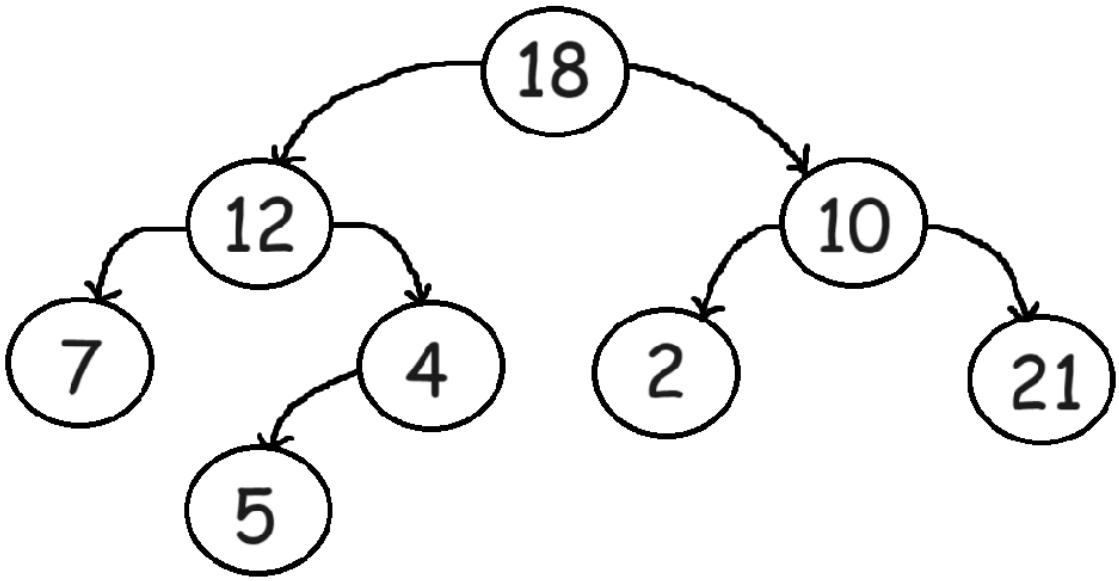
Алгоритм обхода — это специфическая процедура, которая определяет порядок посещения элементов

9. Приведите рекуррентную зависимость для вычисления высоты дерева.

Для вычисления высоты бинарного дерева можно использовать рекуррентную зависимость. Высота дерева определяется как максимальная длина пути от корневой вершины до самой удаленной листовой вершины.

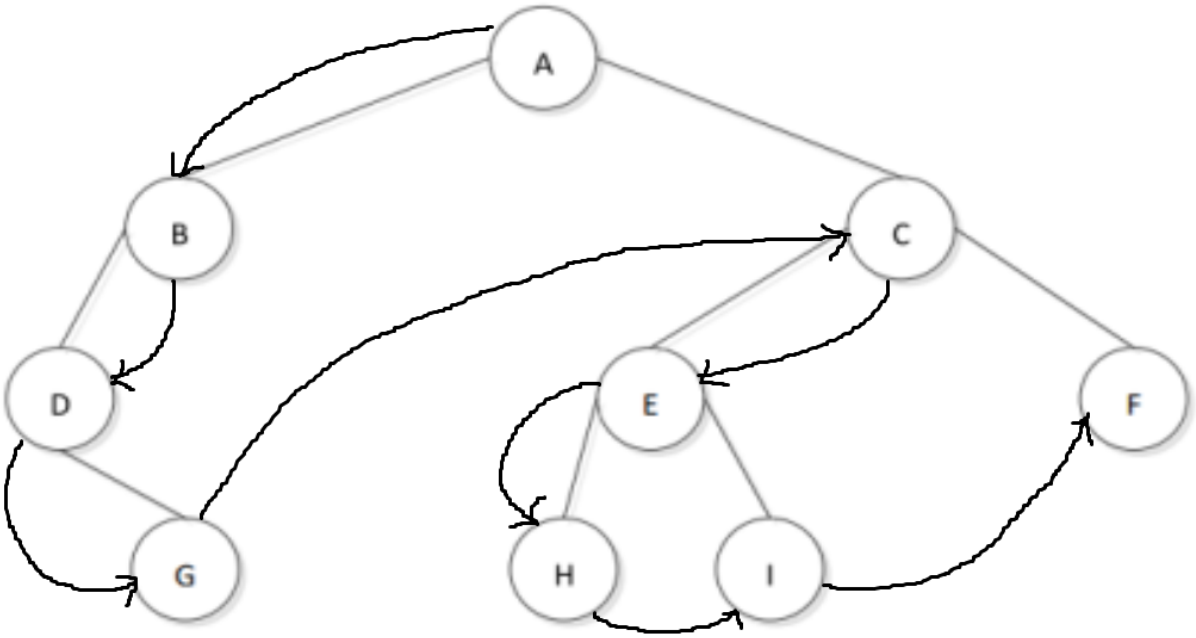
10. Изобразите бинарное дерево, корень которого имеет индекс 6, и которое представлено в памяти таблицей вида

Индекс	key	lef	right
1	12	7	3
2	15	8	NULL
3	4	10	NULL
4	10	5	9
5	2	NULL	NULL
6	18	1	4
7	7	NULL	NULL
8	14	6	2
9	21	NULL	NULL
10	5	NULL	NULL

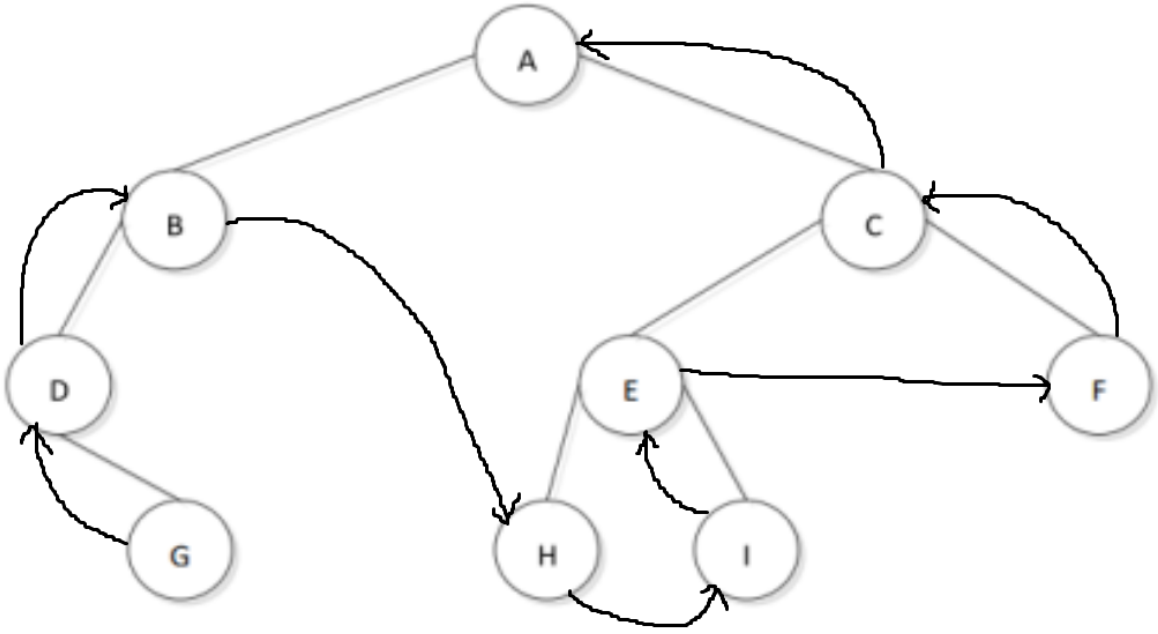


11. Укажите путь обхода дерева по алгоритмы: прямой, обратный, симметричный

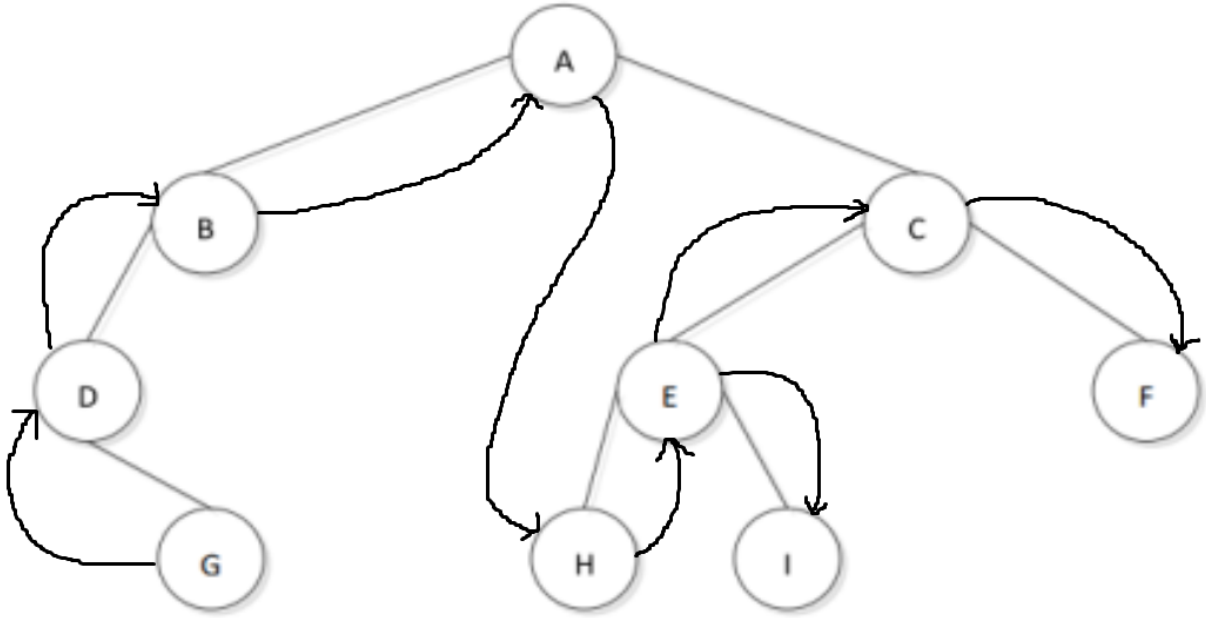
Прямой



Обратный



Симметричный

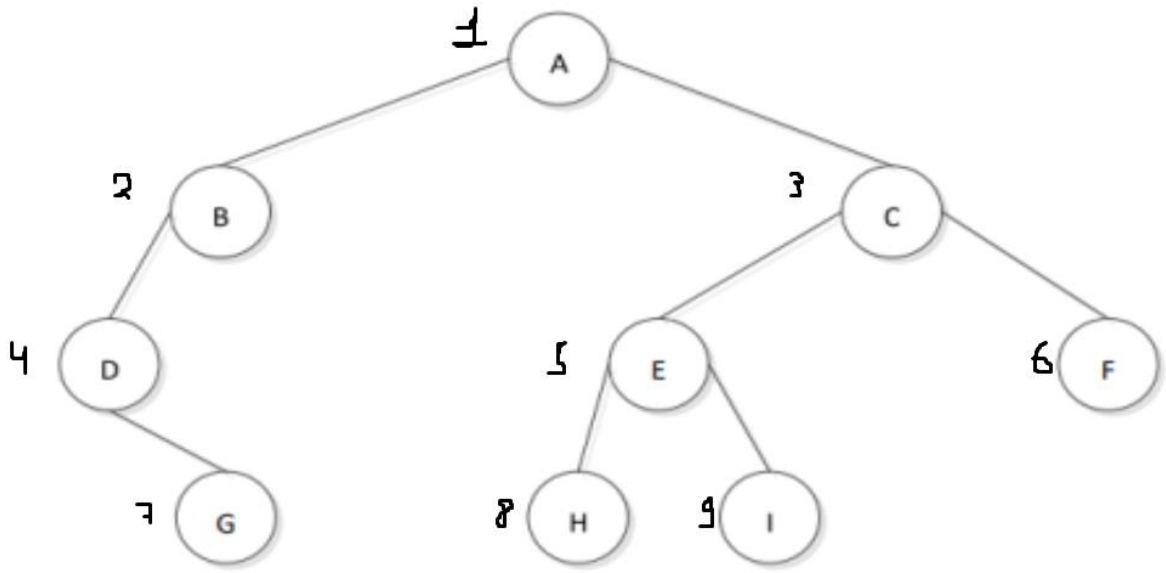


12. Какая структура используется в алгоритме обхода дерева методом в «ширину»?

В алгоритме обхода дерева методом в "ширину" для хранения вершин, которые ожидают обработки, обычно используется структура данных, называемая очередью (queue), но также можно использовать массивы.

13. Выведите путь при обходе дерева в «ширину». Продемонстрируйте использование структуры при обходе дерева.

A B C D E F G H I

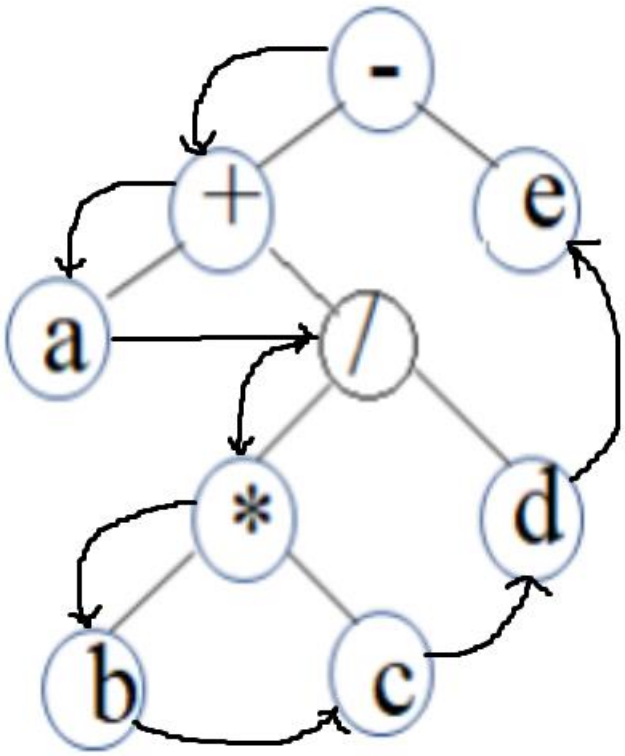


14. Какая структура используется в не рекурсивном обходе дерева методом в «глубину»?

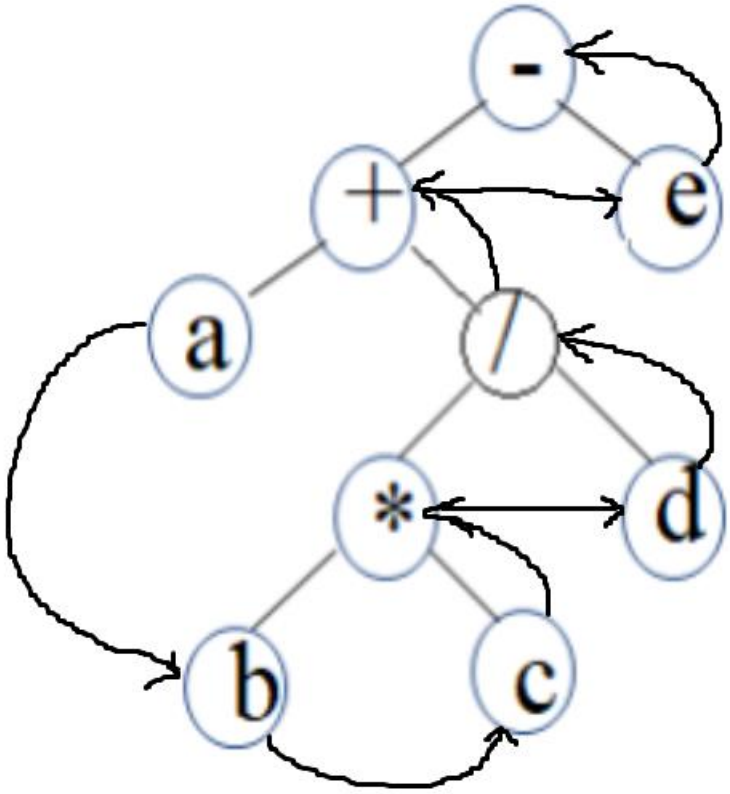
В не-рекурсивном обходе дерева методом в "глубину", часто используется структура данных, называемая стеком (stack), для выполнения обхода вершин.

15. Выполните прямой, симметричный, обратный методы обхода дерева выражений.

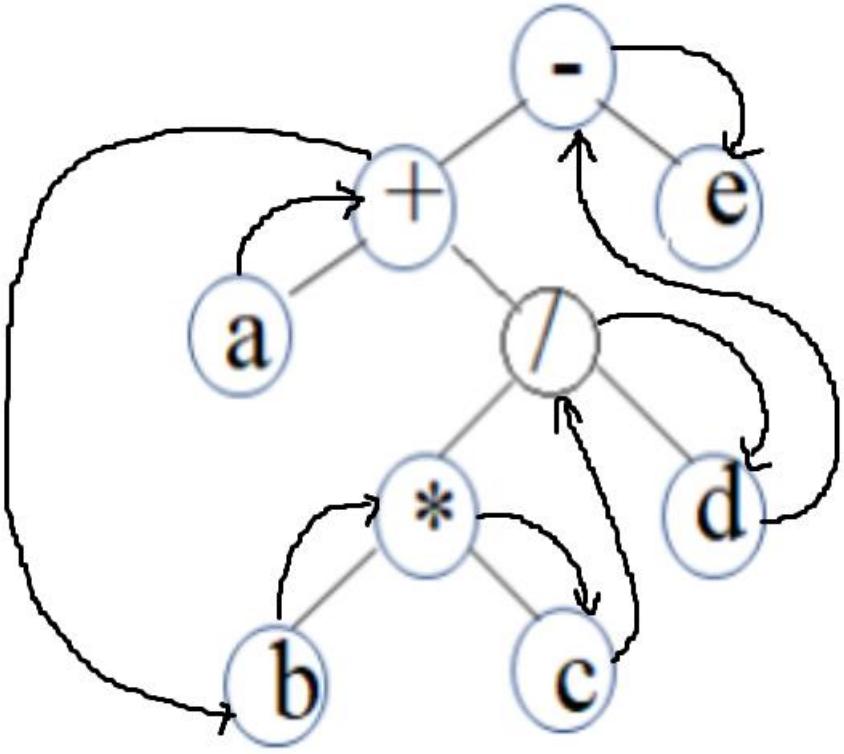
Прямой – префиксный



Обратный – постфиксный

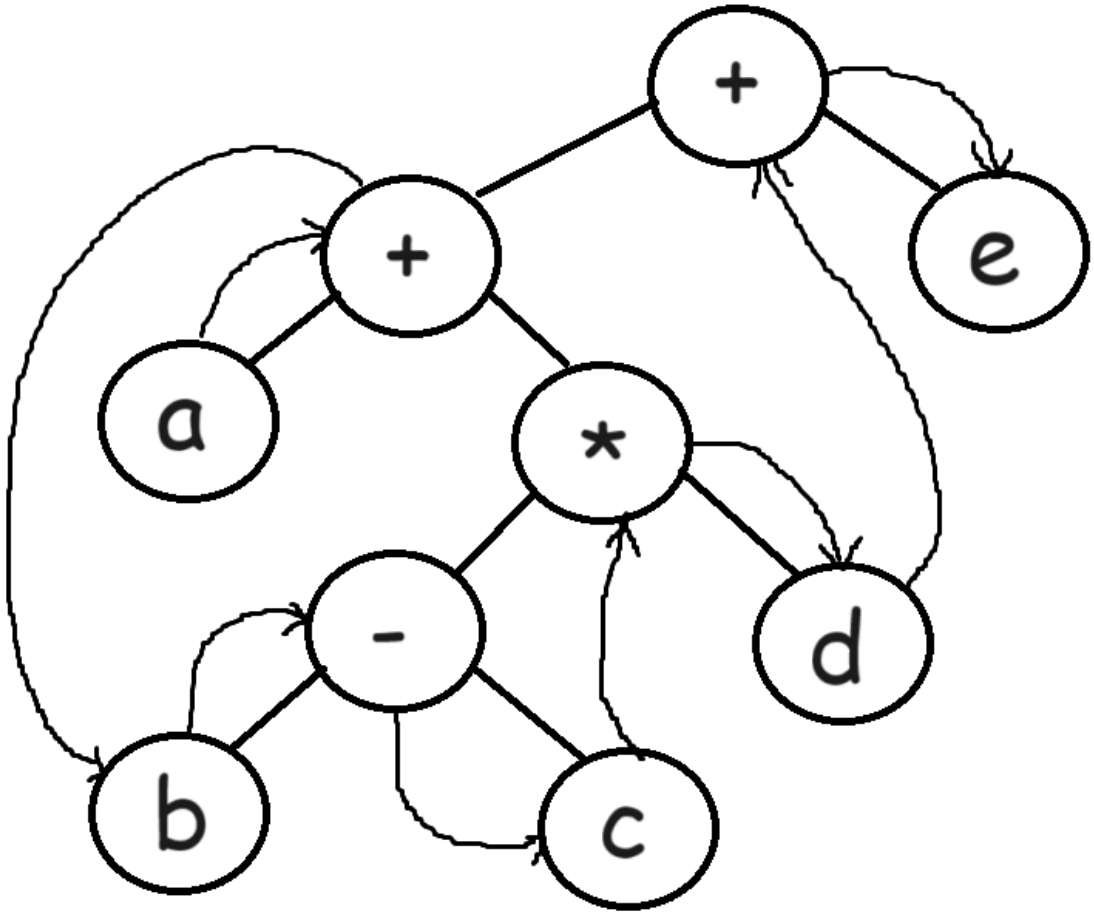


Обратный – инфиксный

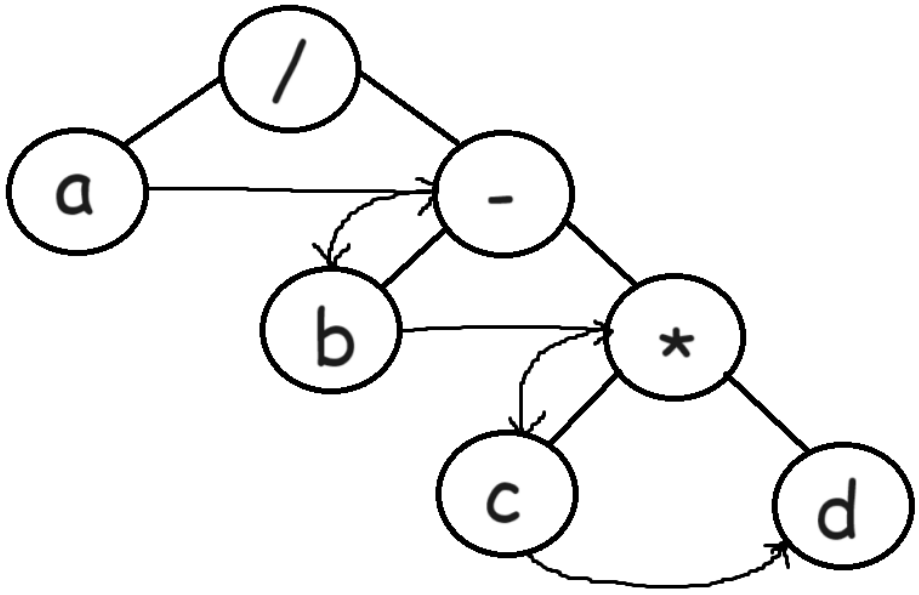


16. Для каждого заданного арифметического выражения постройте бинарное дерево выражений:

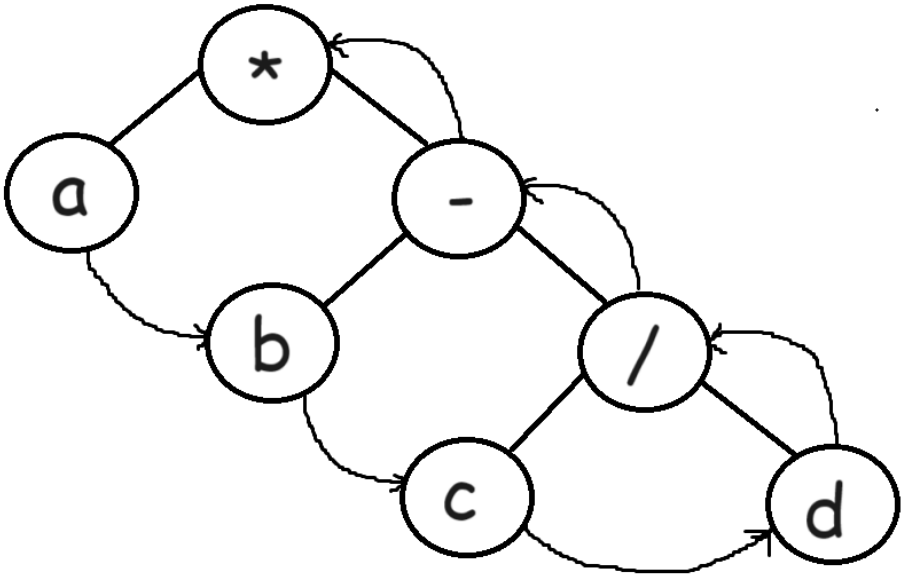
1) a+b-c*d+e



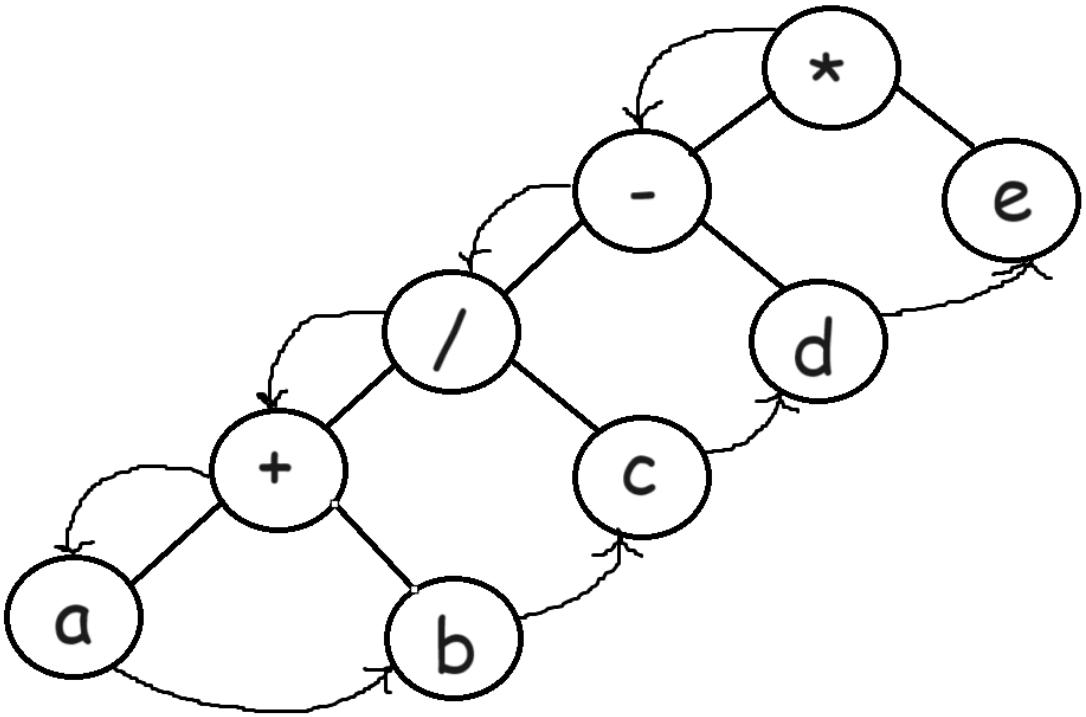
2) /a-b*c d



3) a b c d / - *



4) * - / + a b c d e

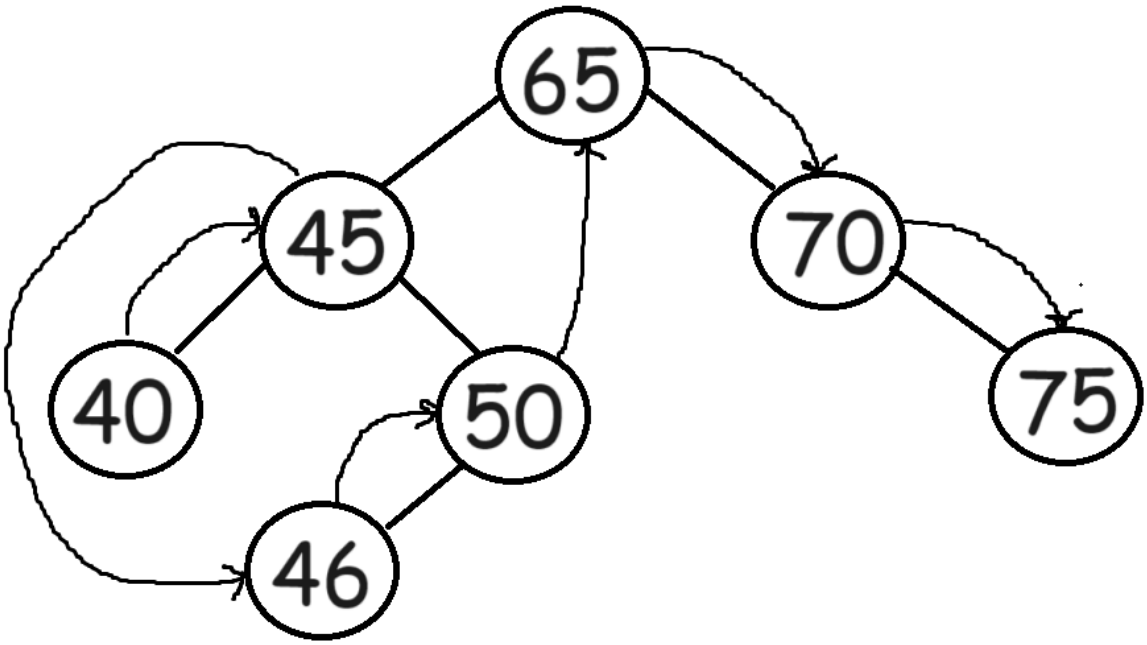


17. В каком порядке будет проходиться бинарное дерево, если алгоритм обхода в ширину будет запоминать узлы не в очереди, а в стеке?

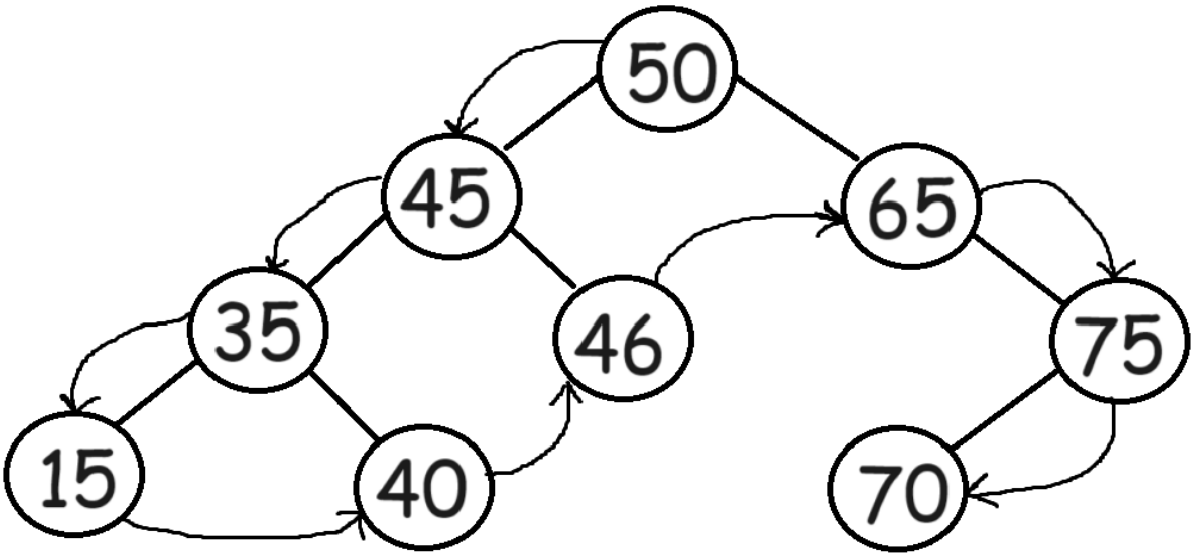
Если алгоритм обхода в ширину будет запоминать узлы не в очереди, а в стеке, то порядок обхода дерева изменится. Вместо классического порядка

по уровням, при котором ближайшие узлы к корню обрабатываются раньше, узлы будут обрабатываться в обратном порядке.

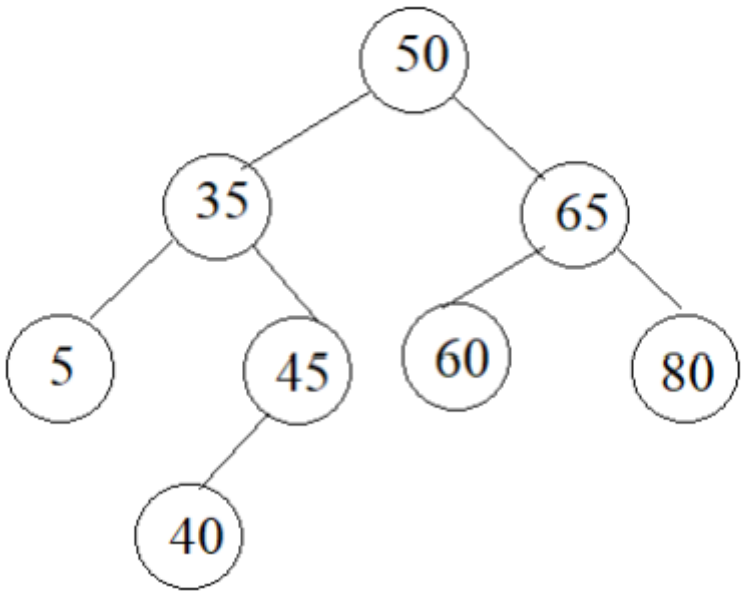
18. Постройте бинарное дерево поиска, которое в результате симметричного обхода дало бы следующую последовательность узлов? 40 45 46 50 65 70 75



19. Приведенная ниже последовательность получена путем прямого обхода бинарного дерева поиска. Постройте это дерево. 50 45 35 15 40 46 65 75 70



20. Дано следующее бинарное дерево поиска



Покажите дерево:

1) после включения узлов 1, 48, 75, 100

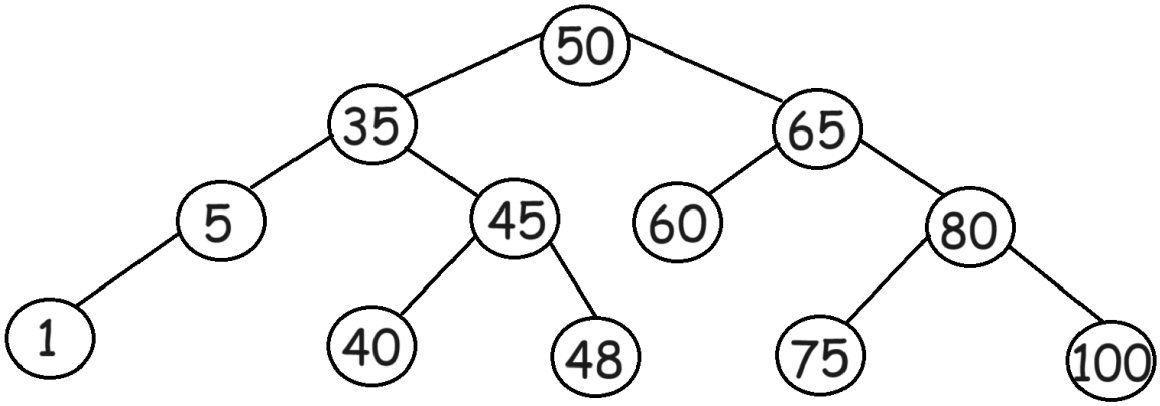
2) после удаления узлов 5, 35

3) после удаления узла 45

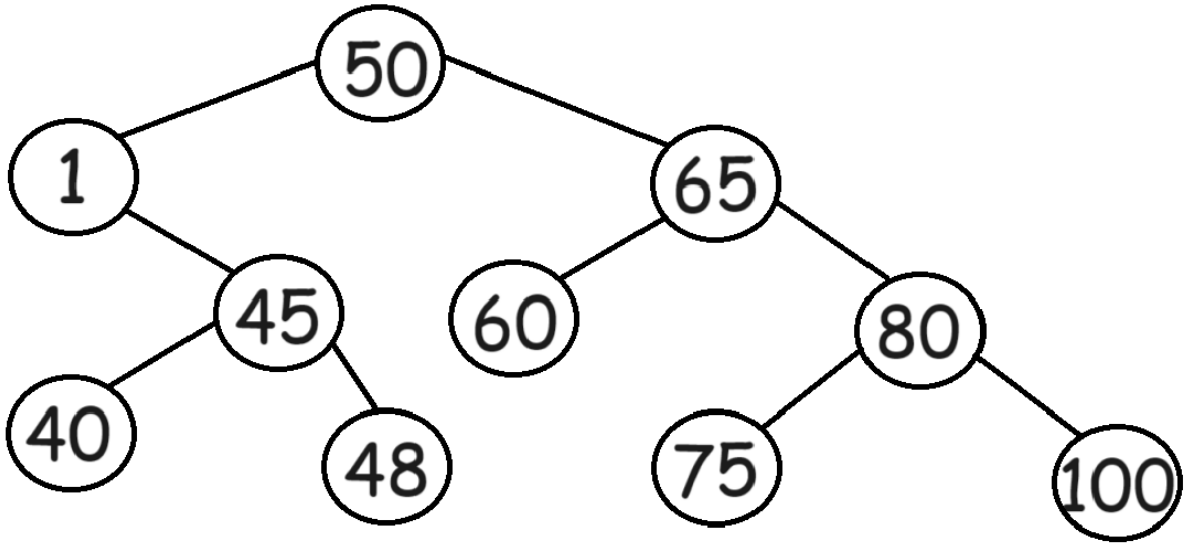
4) после удаления узла 50

5) после удаления узла 65 и вставки его снова

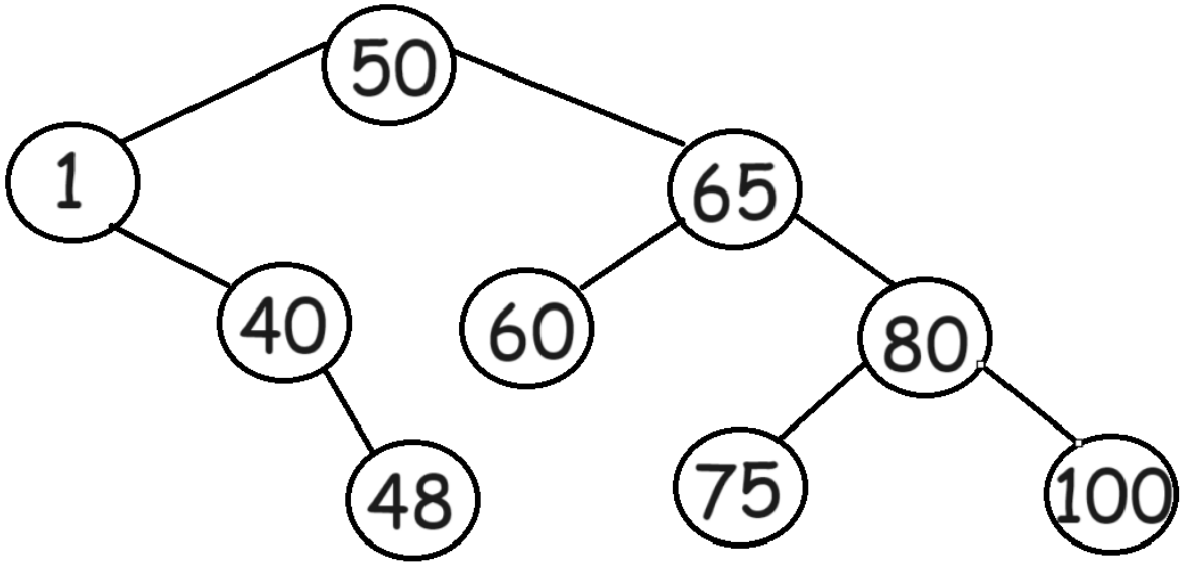
1)



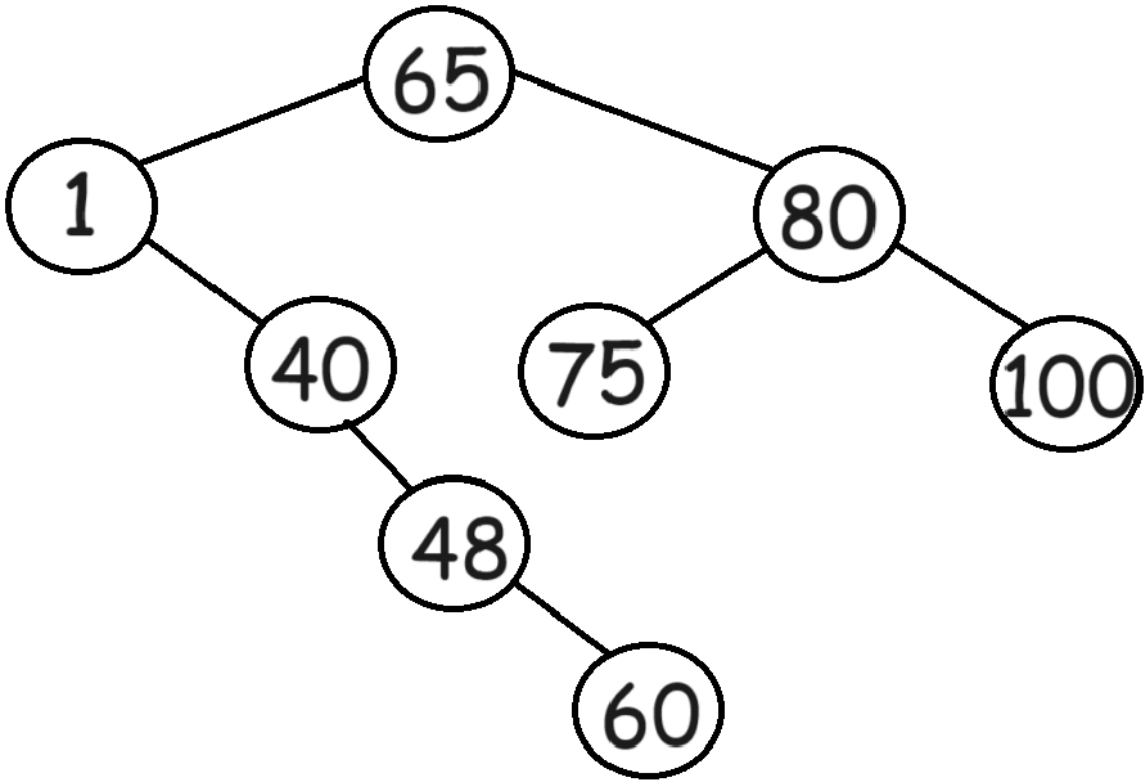
2)



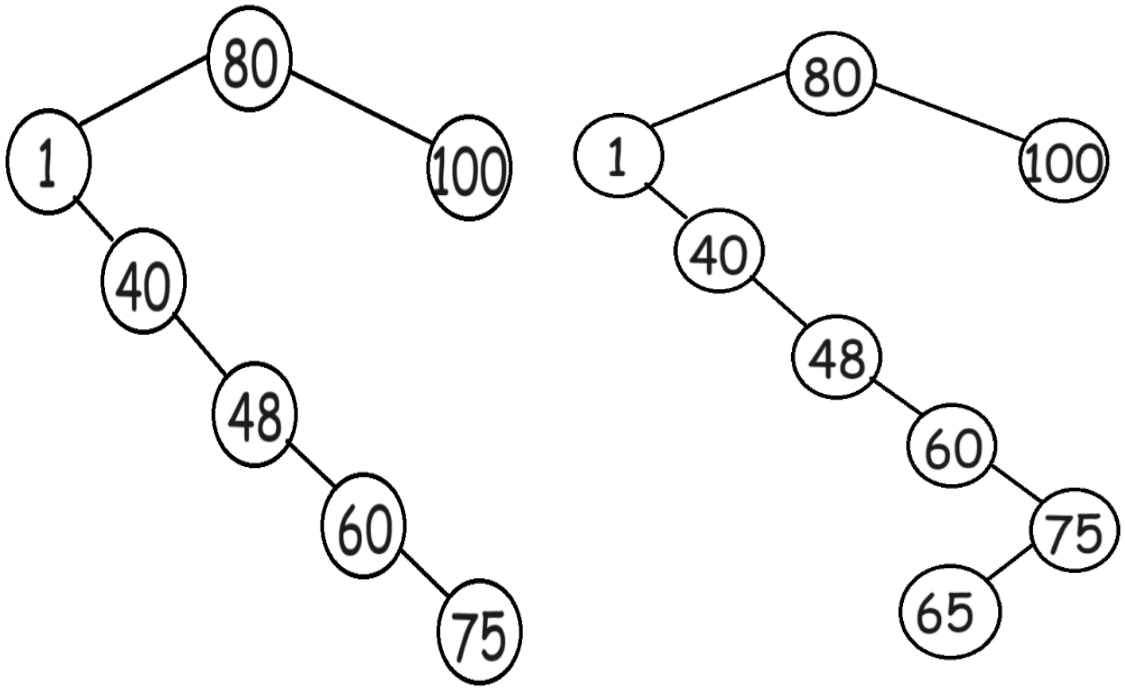
3)



4)



5)



Задание 2.

Разработать программу в соответствии с требованиями варианта. Варианты заданий представлены в таблице 1.

Совет. Выполните реализацию средствами ООП, операции будут методами класса.

Для вариантов с 21 по 30

Вид дерева: бинарное дерево поиска (БДП).

1. Реализовать операции общие для вариантов с 21 по 30 Создать бинарное дерево поиска (информационная часть узла определена вариантом). Для этого реализовать операцию вставки нового значения в БДП и использовать ее при создании дерева.

2. Реализовать операции варианта.

29	Символьное значение	Определить уровень, на котором находится узел с заданным значением. Вычислить значение выражения в левом поддереве. Определить, какое из поддеревьев выше.

Т.к. в варианте была допущена ошибка во втором пункте реализации дополнительных операций, задание было изменено на: Вернуть узел с максимальным значениям обходя дерево в ширину.

Ниже представлен полный листинг кода(Листинг 1).

Листинг 1.

```
#include <iostream>
#include <set>
#include <chrono>
#include <vector>
#define NULL 0
#define ONE 50

using namespace std;

struct node
{
    char data;
    node* left;
    node* right;
};
```

```
struct node* create_node(char data) {
    struct node* node = new (struct node);
    node->data = data;
    node->left = nullptr;
    node->right = nullptr;
    return node;
}

struct node* insert(char data, node* node) {

    if (node == NULL) {
        return create_node(data);
    }
    else {
        if ((char)data < (char)node->data) {
            node->left = insert(data, node->left);
        }
        if ((char)data > (char)node->data) {
            node->right = insert(data, node->right);
        }
        return node;
    }
}

void print_tree(node* node, int levl = 0) {

    if (node) {
        print_tree(node->right, levl + 1);
        for (int i = 0; i < levl; i++) cout << "    ";
        cout << (char)node->data << endl;
        print_tree(node->left, levl + 1);
    }
}


// поиск уровня
int search_levl(char data, node* node, int levl = 1) {
    if (node == nullptr) {
        return 0;
    }

    if (node->data == data) {
        return levl;
    }

    int left_levl = search_levl(data, node->left, levl + 1);
    if (left_levl != 0) {
        return left_levl;
    }

    int right_levl = search_levl(data, node->right, levl + 1);
    return right_levl;
}

// обход мудрого древа по ширине
//Мудрое слово DFS
void DFS(node* node) {
    struct node* p = node;
    vector<struct node*> v;
    v.push_back(node);
    char max_node = 'A';

    while (!v.empty())
    {
```

```
        vector<struct node*> vn;
        for (int i = 0; i < v.size(); i++) {
            auto node = v[i];
            if (node->data > max_node) {
                max_node = node->data;
            }
            //v.erase(v.begin());

            if (node->left != nullptr) {
                vn.push_back(node->left);
            }
            if (node->right != nullptr) {
                vn.push_back(node->right);
            }
        }
        v = vn;
    }
    cout << "Максимальное значение узла: " << max_node;
}


int max_node(node* node) {

    if (node == nullptr) {
        return 0;
    }

    int counter = 0;

    if (node->data) {
        counter++;
    }

    counter += max_node(node->left);
    counter += max_node(node->right);

    return counter;
}


int main()
{
    setlocale(LC_ALL, "ru");

    struct node* root = NULL;
    root = create_node('D');
    root = insert('A', root);
    root = insert('E', root);
    root = insert('C', root);
    root = insert('B', root);
    root = insert('G', root);
    root = insert('F', root);

    int x;

    while (true)
    {

        cout << endl << "Что вы хотите сделать? " << endl;
        cout << "1: Вывести дерево " << endl;
        cout << "2: Найти элемент " << endl;
        cout << "3: Вернуть узел с максимальным значением обходя дерево по ширене
" << endl;
        cout << "4: Вставить элемент" << endl;
        cout << "5: Какое из поддереввь выше?" << endl;
```

```
        cout << "6: выйти" << endl;
        cin >> x;
        cout << endl;

        switch (x)
        {
        case 1:
        {
            print_tree(root);
            break;
        }
        case 2:
        {
            cout << "Введите искомое значение: " << endl;
            char value;
            cin >> value;

            int levl = search_levl(value, root);
            if (levl != 0)
            {
                cout << "Значение находиться на уровне: " << levl << endl;
            }
            else {
                cout << "Не найдено" << endl;
            }
            break;
        }
        case 3:
        {
            DFS(root);
            break;
        }
        case 4:
        {
            char x;
            cout << "Введите символ: " << endl;
            cin >> x;
            root = insert(x, root);
            break;
        }
        case 5:
        {
            int L = max_node(root->left);
            int R = max_node(root->right);
            if (L == R) { cout << "Они равны" << endl; }
            else if (L > R) { cout << "Левое поддерево выше" << endl; }
            else { cout << "Правое поддерево выше" << endl; }
            break;
        }
        default:
        {
            return false;
            break;
        }
        }
    }
}
```

Вывод:

Были получены навыки разработки и реализаций операций над структурой данных бинарное дерево.