



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

Лещенко Вячеслав Романович

Проверил:

ассистент Муравьёва Е.А.

МОСКВА 2023 г.

Практическая работа № 4

Цель работы

Получение умений и навыков разработки и реализаций операций над структурой данных бинарное дерево.

Ход работы

Вариант 20

20	Символьное значение	Образовать префиксную форму выражения, содержащегося в дереве выражения, и записать ее в строку. Вычислить значение выражения по дереву. Вычислить значение выражения по префиксной форме.
----	---------------------	--

Задание 1

Формулировка задачи:

Ответы на вопросы:

Что определяет степень дерева?

Степень дерева определяется *количеством потомков*, или детей, у *каждой вершины (узла) в дереве*. Вершины дерева называются узлами, и степень узла определяет, сколько непосредственных потомков у этого узла.

Узел степени 0 (листовой узел): Узел, у которого нет потомков, называется листовым узлом. Листовые узлы находятся в конце ветвей дерева и не имеют дополнительных поддеревьев.

Узел степени 1: Узел, у которого есть только один непосредственный потомок, называется узлом степени 1. Такие узлы соединяют другие узлы в древовидной структуре.

Узел степени 2 и более: Узел, у которого есть два или более непосредственных потомка, называется узлом степени 2 или более.

Степень дерева — это максимальное количество потомков (детей), которое имеет узел в дереве. Она может варьироваться в зависимости от конкретной структуры дерева.

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.** Степень сильноветвящегося дерева может быть очень высокой, и она зависит от конкретной структуры дерева.

В примере сильноветвящегося дерева, степень каждой вершины может быть значительно больше, чем в типичных деревьях. Например, в бинарном дереве каждая вершина имеет не более двух потомков (степень 2), в то время как в сильноветвящемся дереве вершины могут иметь множество потомков. Степень сильноветвящегося дерева зависит от его конкретной структуры и потребностей задачи, для которой оно используется.

Что определяет путь в дереве?

Путь в дереве определяется последовательностью вершин, которые соединяются друг с другом от корня до определенной целевой вершины внутри дерева. Путь в дереве представляет собой **цепь или последовательность вершин и рёбер**, которые соединяются между собой.

Основные характеристики пути в дереве:

- **Начальная вершина (или корень):** Путь начинается с какой-то начальной вершины в дереве, которая может быть корневой вершиной или любой другой.
- **Целевая вершина:** Путь заканчивается в целевой вершине, которая может быть любой внутри дерева.
- **Вершины и рёбра на пути:** Путь включает в себя все вершины, которые проходят от начальной вершины к целевой вершине, а также все рёбра, которые соединяют эти вершины.
- **Длина пути:** Длина пути определяется количеством вершин или **рёбер** на этом пути. Например, длина пути может быть равна

количеству вершин минус один, или по-другому количеству рёбер.

Путь в дереве может быть использован для определения связей и отношений между вершинами внутри дерева. Также путь может использоваться для нахождения определенной информации или выполнения навигации в дереве, в том числе поиск пути между вершинами или вычисление расстояния между ними.

Как рассчитать длину пути в дереве?

- *По количеству вершин:* Длина пути в дереве может быть определена как количество вершин (узлов), которые включены в этот путь. Если есть последовательность вершин, соединенных рёбрами, можно просто подсчитать количество вершин на этом пути. Длина пути будет равна количеству вершин минус один, так как количество рёбер на пути всегда на один меньше количества вершин.
- *По количеству рёбер:* Длина пути также может быть рассчитана через количество рёбер на пути. Это делается путем подсчета всех рёбер, которые соединяют вершины на пути. Количество рёбер равно длине пути.
- *Весовой путь (взвешенное дерево):* если дерево имеет весовые рёбра (рёбра имеют числовые значения, называемые весами), то длина пути может быть определена как сумма весов рёбер на пути. В этом случае длина пути будет равна сумме весов всех рёбер, которые соединяют вершины на пути.

Выбор метода зависит от конкретного контекста и структуры дерева. Важно учесть, что в зависимости от задачи можно использовать один из этих методов для вычисления длины пути в дереве.

Какова степень бинарного дерева?

Степень бинарного дерева определяется максимальным числом потомков, которое может иметь вершина (узел) в этом дереве. В бинарном дереве каждая вершина может иметь не более двух потомков: левого и правого.

Следовательно, *степень бинарного дерева равна 2*, так как каждая вершина может иметь максимум два потомка. Это особенность бинарных деревьев и делает их структуру очень удобной для множества задач, включая поиск, сортировку и балансировку данных.

Может ли дерево быть пустым?

Да, дерево может быть пустым. Пустое дерево не содержит ни одной вершины (узла) и, следовательно, не имеет никакой структуры. В программировании и структурах данных пустые деревья часто используются как начальное состояние перед добавлением данных или как специальное состояние для представления отсутствия информации или структуры.

Пустое дерево полезно, когда вы хотите начать построение дерева или когда ваши данные могут отсутствовать. Когда дерево пусто, оно не содержит ни одной вершины, и его структура не определена. Как только вы начинаете добавлять вершины в дерево, оно начинает приобретать форму и структуру.

Пустые деревья широко используются в алгоритмах и структурах данных, таких как бинарные деревья поиска, кучи (heap), деревья AVL и других. Они представляют собой важный элемент для обеспечения корректной работы алгоритмов и обработки различных случаев.

Дайте определение бинарного дерева.

Бинарное дерево — это иерархическая структура данных, в которой каждая вершина (узел) имеет не более двух потомков: левого и правого. Бинарные деревья широко используются в информатике для различных задач, таких как поиск, сортировка, хранение данных и многие другие.

Основные характеристики бинарного дерева:

- *Корневая вершина (Root)*: Это вершина, которая находится в самом верхнем уровне бинарного дерева. Все остальные вершины происходят от корневой вершины.
- *Левый и правый потомки*: Каждая вершина может иметь не более двух потомков: левого и правого. Левый потомок находится слева от родительской вершины, а правый - справа.
- *Листовые вершины (Leaves)*: Листовые вершины — это вершины без потомков, то есть вершины, у которых не существует левого и правого поддерева. Они находятся в конце ветвей дерева.
- *Поддеревья*: Каждый узел бинарного дерева может быть корнем для своего собственного поддерева, включая все вершины, которые являются его потомками.

Бинарные деревья применяются для решения различных задач, таких как бинарные деревья поиска (BST), кучи (heap), деревья AVL и др. Каждый из этих видов бинарных деревьев имеет свои собственные правила и свойства, которые обеспечивают эффективную работу алгоритмов и структур данных.

Дайте определение алгоритму обхода.

Алгоритм обхода (или алгоритм прохода) — это специфическая процедура, которая определяет порядок посещения элементов (например, узлов или вершин) в структуре данных, такой как дерево, граф, массив и другие. Обход используется для систематического и последовательного доступа ко всем элементам структуры данных.

Алгоритмы обхода могут быть применены к различным задачам, таким как поиск, вывод данных, анализ и обработка информации внутри структуры. Обходы бывают двух видов: **в ширину и в глубину**.

Поиск в ширину (BFS) идет из начальной вершины, посещает сначала

все вершины, находящиеся на расстоянии одного ребра от начальной, потом посещает все вершины на расстоянии два ребра от начальной и так далее. Алгоритм поиска в ширину является по своей природе нерекурсивным (итеративным). Для его реализации применяется структура данных очередь (FIFO).

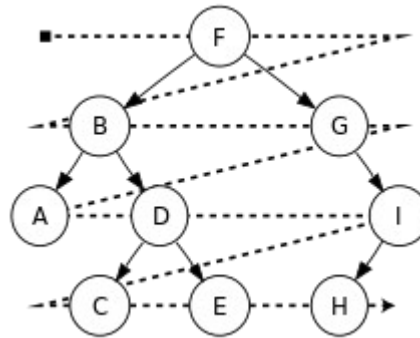


Рисунок 1 – пример обхода бинарного дерева в ширину

Поиск в глубину (DFS) идет из начальной вершины, посещая еще не посещенные вершины без оглядки на удаленность от начальной вершины. Алгоритм поиска в глубину по своей природе является рекурсивным. Для эмуляции рекурсии в итеративном варианте алгоритма применяется структура данных стек. Обходу в глубину в графе соответствуют три вида обходов бинарного дерева: *прямой* (pre-order), *симметричный* (in-order) и *обратный* (post-order).

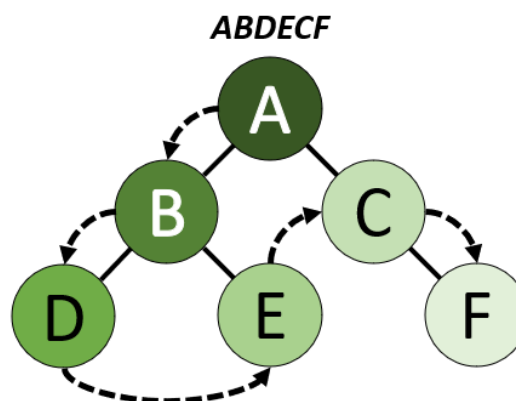


Рисунок 2 – Прямой обход дерева (корень, лево, право)

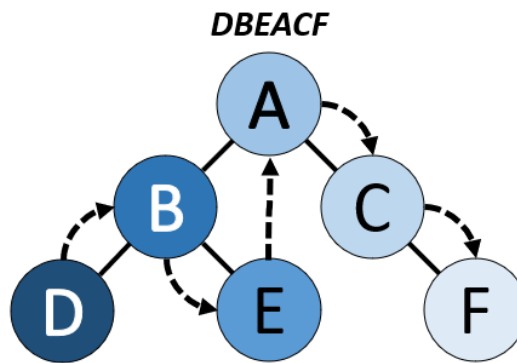


Рисунок 3 – Симметричный обход дерева (лево, корень, право)

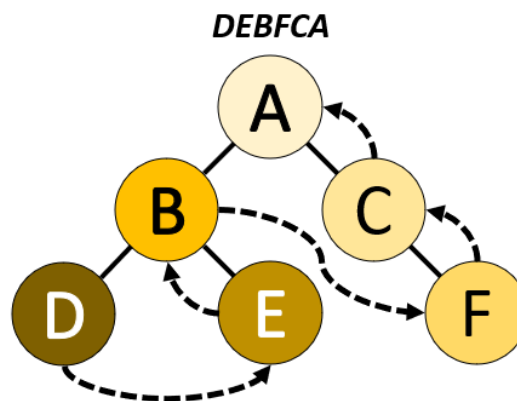


Рисунок 4 – Обратный обход дерева (лево, право, корень)

Каждый тип алгоритма обхода имеет свои уникальные применения и может использоваться в разных сценариях в зависимости от конкретной задачи и структуры данных.

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. Рекуррентная зависимость для вычисления высоты дерева может быть описана следующим образом:

Пусть *height(root)* обозначает высоту дерева с корнем в вершине *root*.

Если *root* равен *null* (то есть дерево пусто), то *height(root)* равна -1 (или

0, в зависимости от условия).

Иначе, $height(root)$ можно выразить как:

$$height(root) = \max(height(root.left), height(root.right)) + 1$$

Формула гласит, что высота дерева с корнем в вершине **root** равна максимальной из высот левого поддерева $height(root.left)$ и высоты правого поддерева $height(root.right)$, увеличенной на 1. Это означает, что для вычисления высоты дерева, мы выбираем более высокое из двух поддеревьев (левого и правого) и добавляем 1, чтобы учесть текущий уровень, на котором находится корневая вершина.

Эта рекуррентная зависимость рекурсивно вычисляет высоту дерева, начиная с корневой вершины и продолжая до листовых вершин. Результатом будет высота всего дерева.

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

<i>Индекс</i>	<i>key</i>	<i>left</i>	<i>right</i>
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

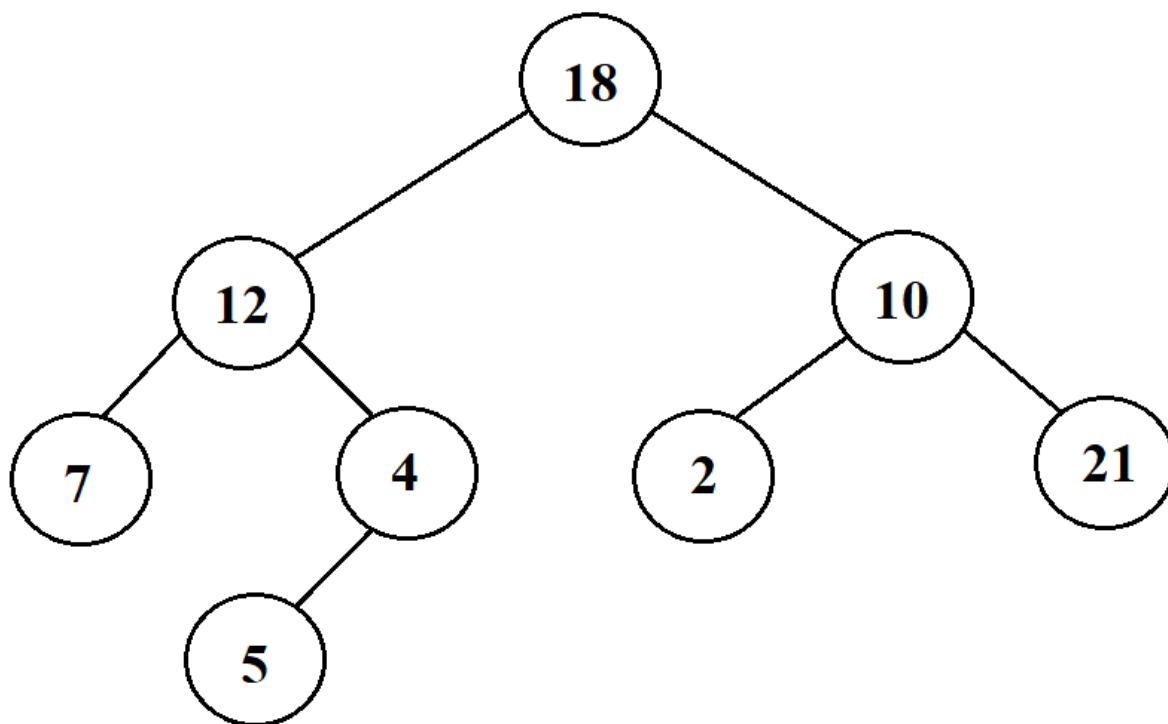


Рисунок 5 – бинарное дерево, представленное таблицей

Укажите путь обхода дерева по алгоритмам: прямой, обратный, симметричный

Представление трех видов обхода дерева в глубину:

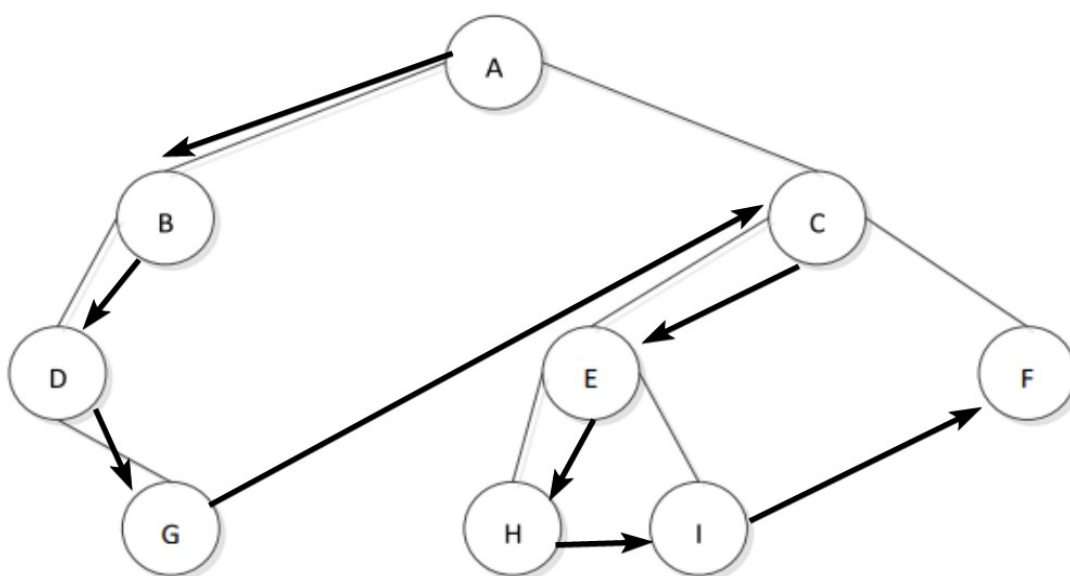


Рисунок 6 – Прямой обход дерева (**ABDGCENIF**)

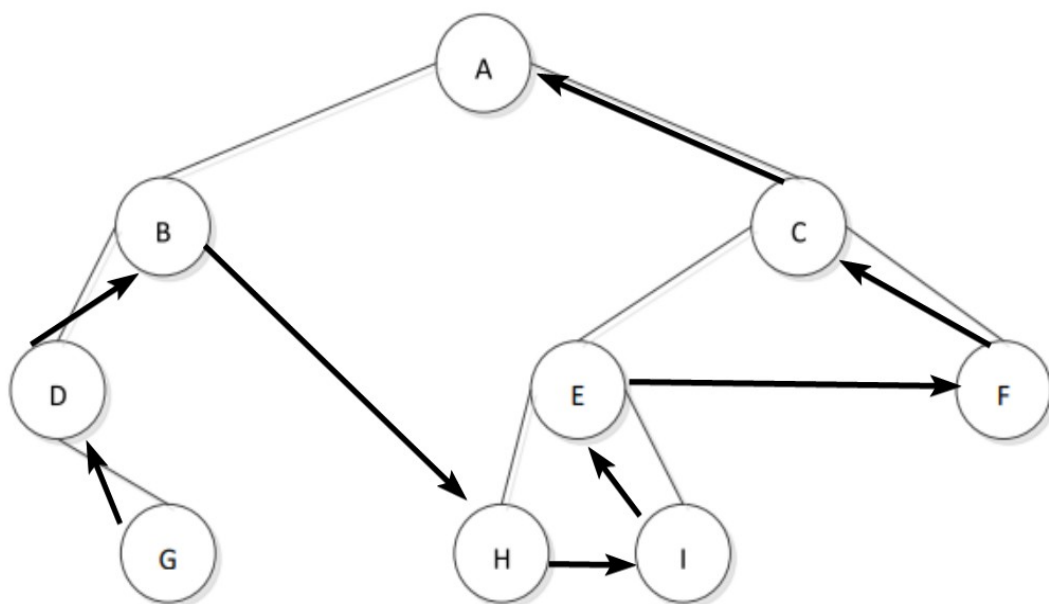


Рисунок 7 – Обратный обход дерева (**GDBHIEFCA**)

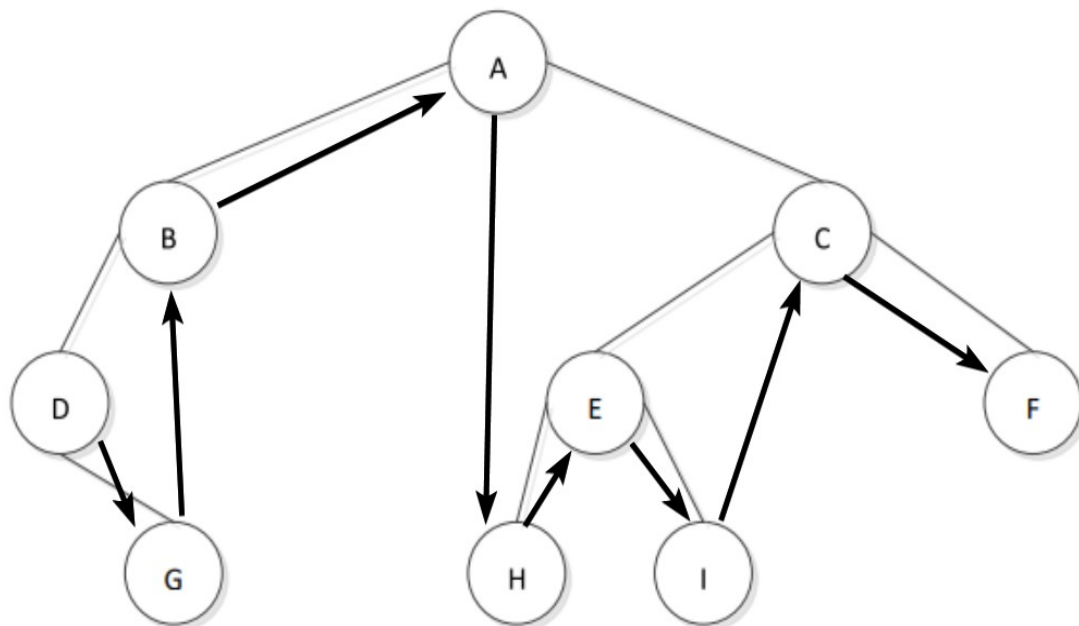


Рисунок 8 – Симметричный обход дерева (**DGBAHEICF**)

Какая структура используется в алгоритме обхода дерева методом в «ширину»

В алгоритме обхода дерева методом в "ширину" (Breadth-First Search, BFS) для хранения вершин, которые ожидают обработки, обычно используется структура данных, называемая *очередью* (queue). Очередь работает по принципу "первым пришел - первым ушел" (FIFO - First-In-First-Out). Это означает, что вершины добавляются в очередь в порядке их появления, и они извлекаются из очереди в том же порядке.

В контексте обхода дерева методом BFS, процесс может быть описан следующим образом:

1. Начинаем с корневой вершины дерева и помещаем ее в очередь.
2. Затем выполняем следующие шаги в цикле, пока очередь не опустеет:
 1. Извлекаем вершину из начала очереди.
 2. Обрабатываем эту вершину (например, выводим ее значение или выполняем другие операции).
 3. Добавляем в очередь всех непосещенных потомков этой вершины.
3. Повторяем шаги 2 до тех пор, пока в очереди не останется вершин для обработки.

Использование очереди в алгоритме BFS позволяет обеспечить обход вершин на текущем уровне перед переходом к вершинам следующего уровня. Это обеспечивает обход в "ширину", где сначала обрабатываются вершины на текущем уровне, затем на следующем уровне и так далее, пока не пройдены все уровни дерева.

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

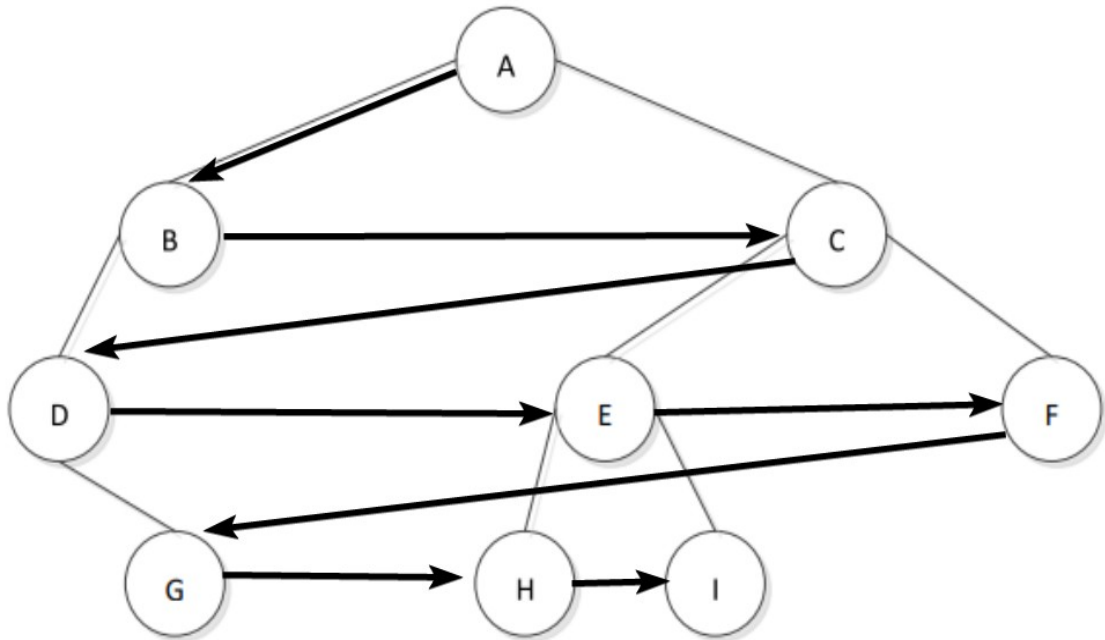


Рисунок 9 – обход дерева в ширину (ABCDEFGHII)

Структура обхода дерева в ширину заключается в добавлении в очередь каждого потомка искомого узла слева направо. То есть сначала в очереди у нас уже есть корневой узел, потом при обходе значений мы как-бы достаём его из очереди, производим с ним какие-либо операции (например, считываем) и засовываем в очередь его левого и правого потомка. Потом циклично повторяем тоже самое, доставая из очереди левых и правых потомков узла и уже помещая в очередь их потомков и т.д.

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

В не рекурсивном обходе дерева методом в "глубину" (Depth-First Search, DFS), часто используется структура данных, называемая **стеком** (stack), для выполнения обхода вершин. Стек работает по принципу

"последним пришел - первым ушел" (LIFO - Last-In-First-Out), что означает, что вершины добавляются и извлекаются из стека в обратном порядке.

Процесс не-рекурсивного обхода дерева методом в "глубину" с использованием стека может быть описан следующим образом:

1. Начинаем с корневой вершины дерева и помещаем ее в стек.
2. Затем выполняем следующие шаги в цикле, пока стек не опустеет:
 1. Извлекаем вершину из стека.
 2. Обрабатываем эту вершину (например, выводим ее значение или выполняем другие операции).
 3. Помещаем в стек всех непосещенных потомков этой вершины. При этом, вершины добавляются в стек в обратном порядке, так чтобы вершина, которая должна быть обработана следующей, оказывается на вершине стека.
3. Повторяем шаги 2 до тех пор, пока в стеке не останется вершин для обработки.

Использование стека в алгоритме DFS позволяет реализовать обход вершин на текущем уровне перед переходом к вершинам более глубокого уровня. Это обеспечивает обход в "глубину", где сначала обрабатываются вершины на текущем пути вниз по дереву, затем вершины на более глубоких уровнях.

Выполните прямой, симметричный, обратный методы обхода
дерева выражений

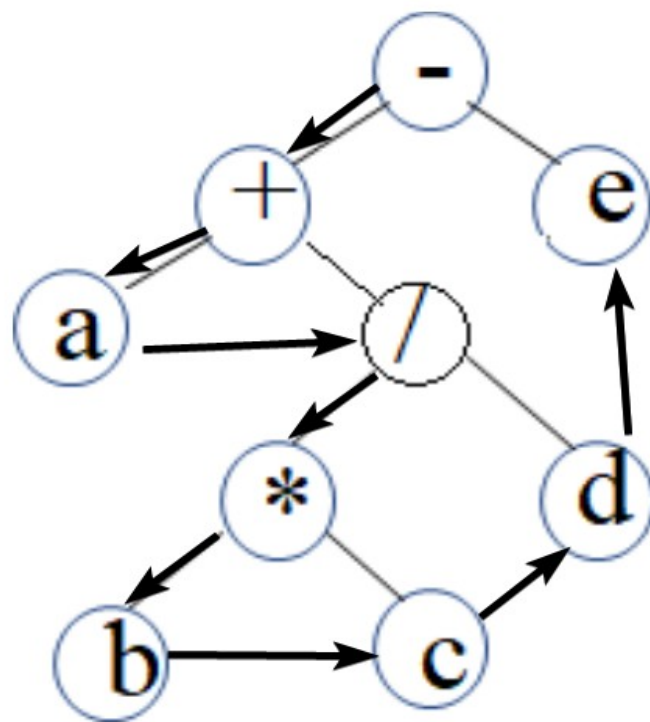


Рисунок 10 – Прямой обход дерева выражений $(-+a/bc*de)$

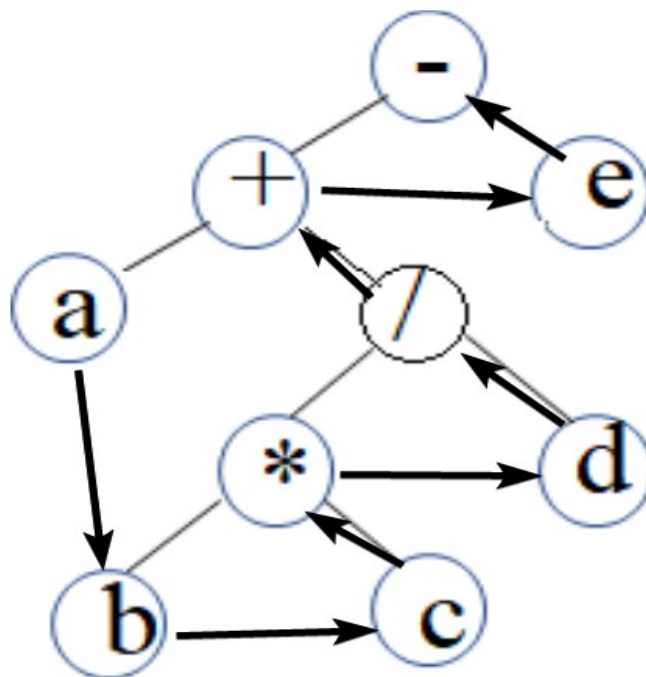


Рисунок 11 – Обратный обход дерева выражений $(abc/+de*-)$

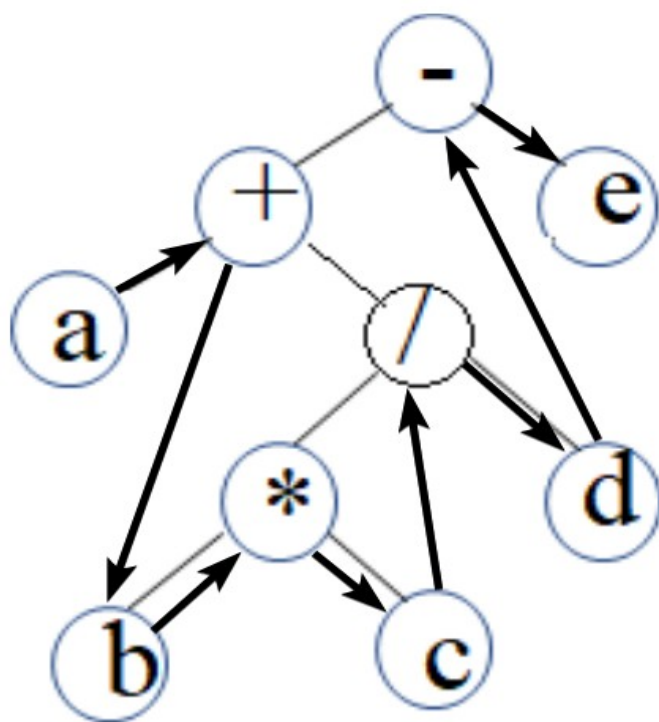


Рисунок 12 – Симметричный обход дерева выражений $(a+b/c-d*e)$

Для каждого заданного арифметического выражения постройте бинарное дерево выражений

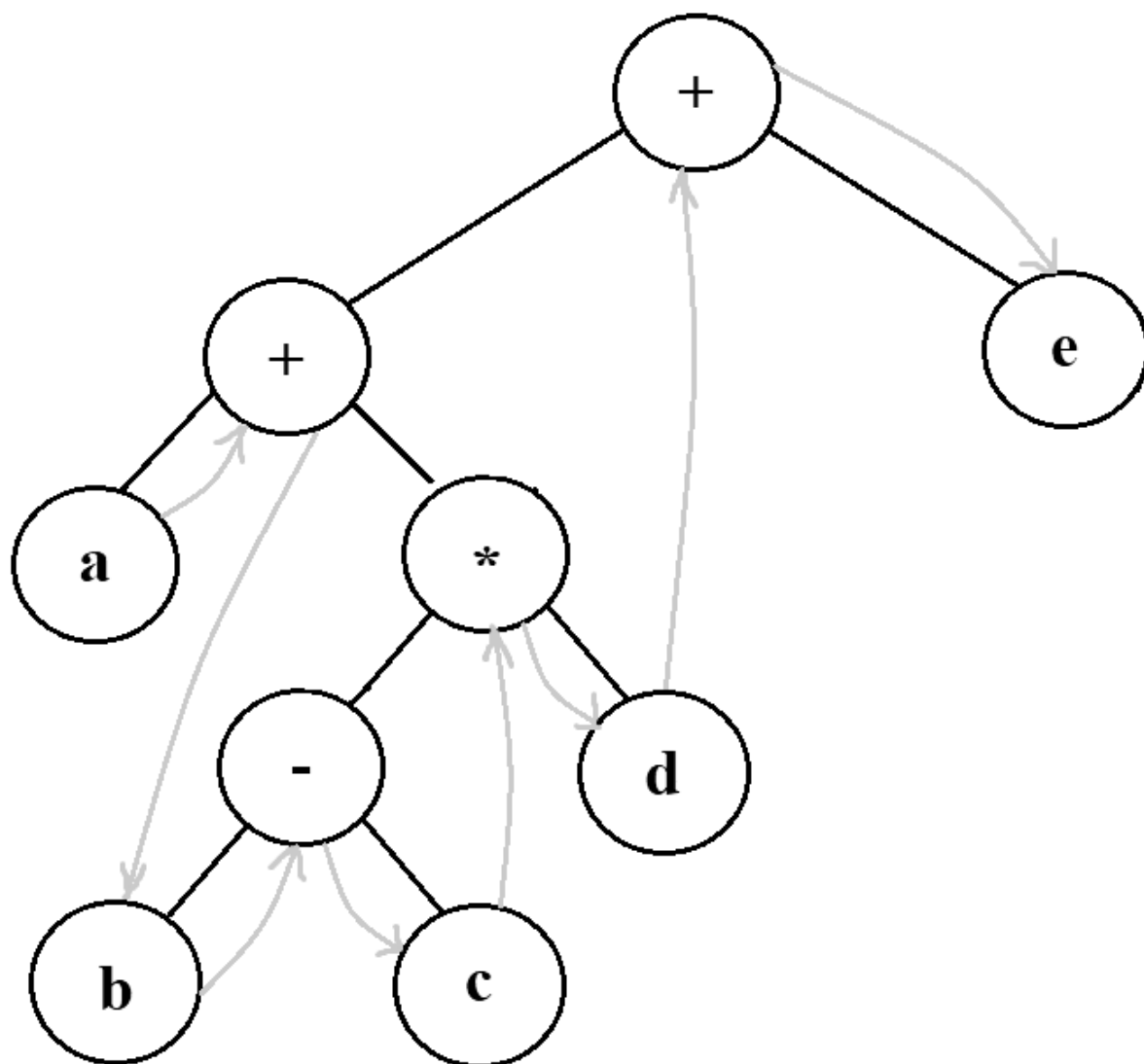


Рисунок 13 – Инфиксное выражение $(a+b-c*d+e)$

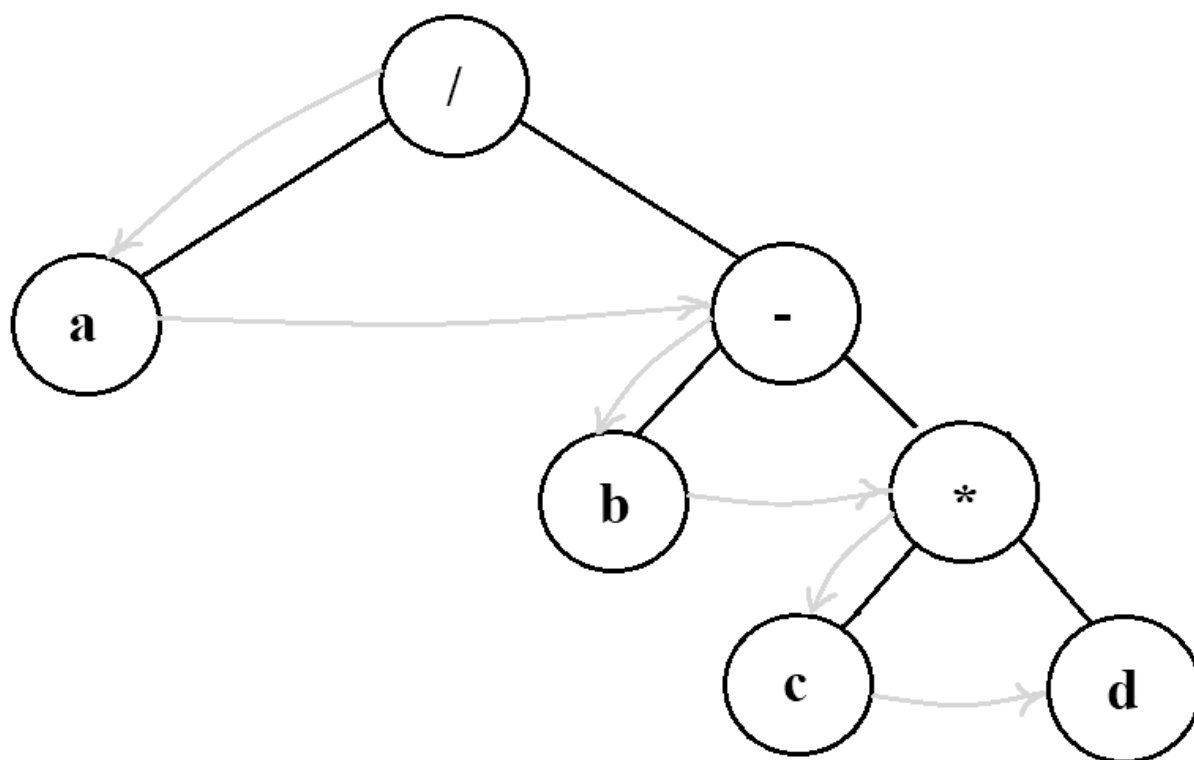


Рисунок 14 – Префиксное выражение (**/a-b*cd**)

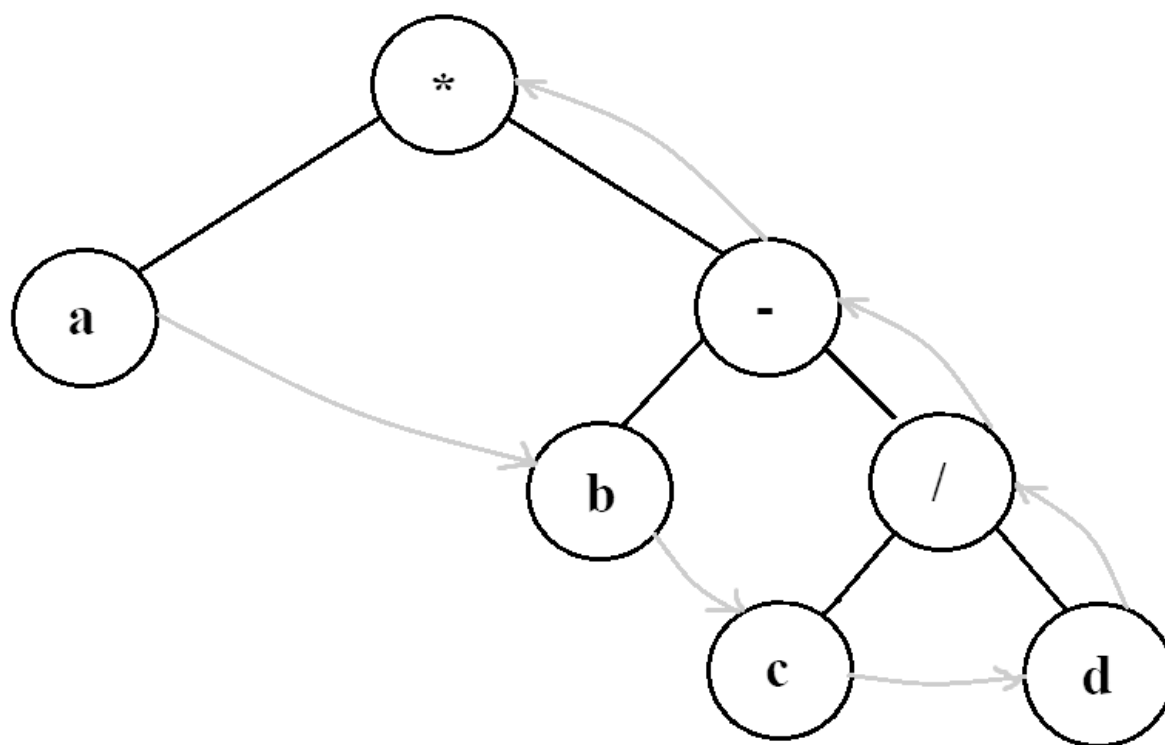


Рисунок 15 – Постфиксное выражение (**abcd/-***)

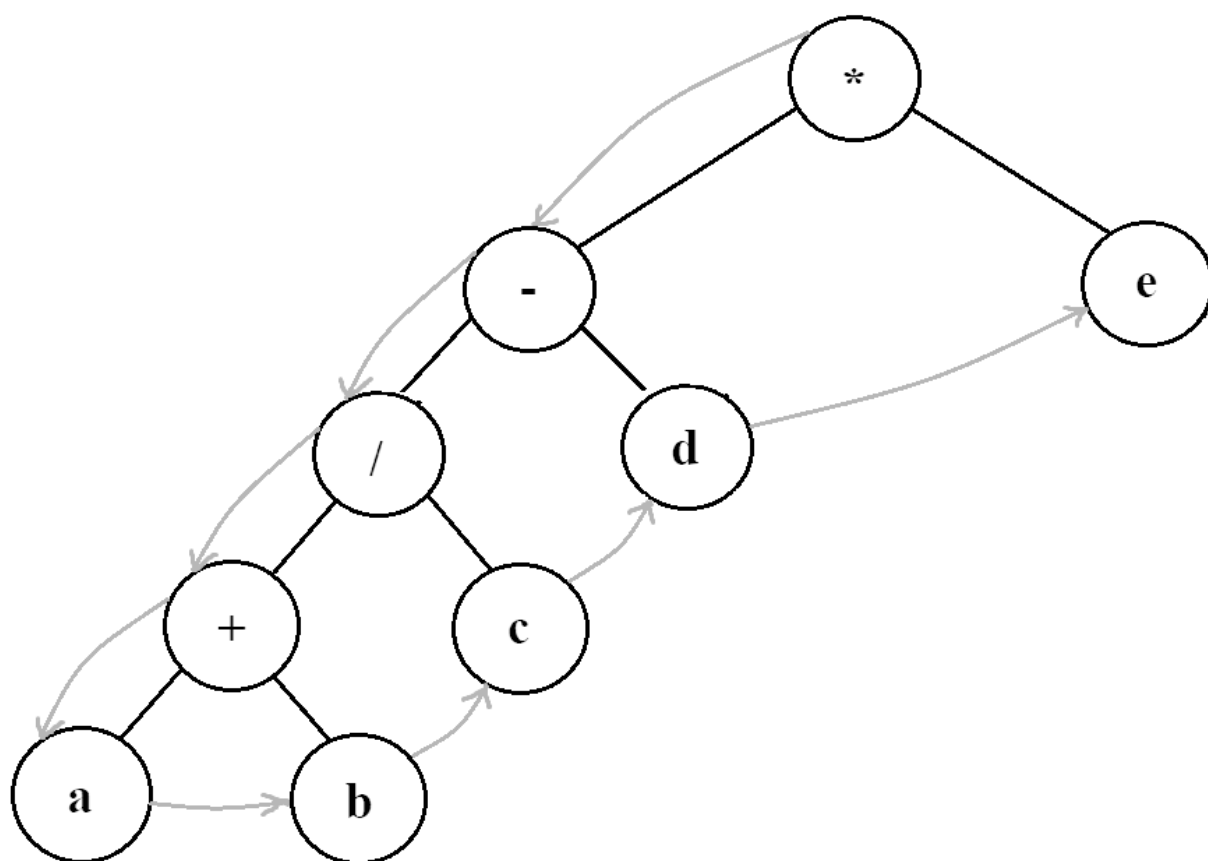


Рисунок 16 – Префиксное выражение $(*/-/ +abcde)$

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. Это произойдет из-за того, что стек работает по принципу "последним пришел - первым ушел" (LIFO - Last-In-First-Out).

Порядок обхода в ширину с использованием стека будет следующим:

1. Начнем с корневой вершины и добавим ее в стек.
2. Затем извлечем вершину из вершины стека, обработаем ее и добавим в стек ее потомков (сначала правого, затем левого).
3. Повторяем шаг 2 для вершины, находящейся на вершине стека.
4. Продолжаем извлекать вершины из вершины стека и добавлять их потомков до тех пор, пока стек не опустеет.

Итак, в этом случае обход будет происходить в порядке, обратном обходу в ширину. Первыми будут обработаны узлы на самом нижнем уровне, затем узлы на более высоких уровнях.

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

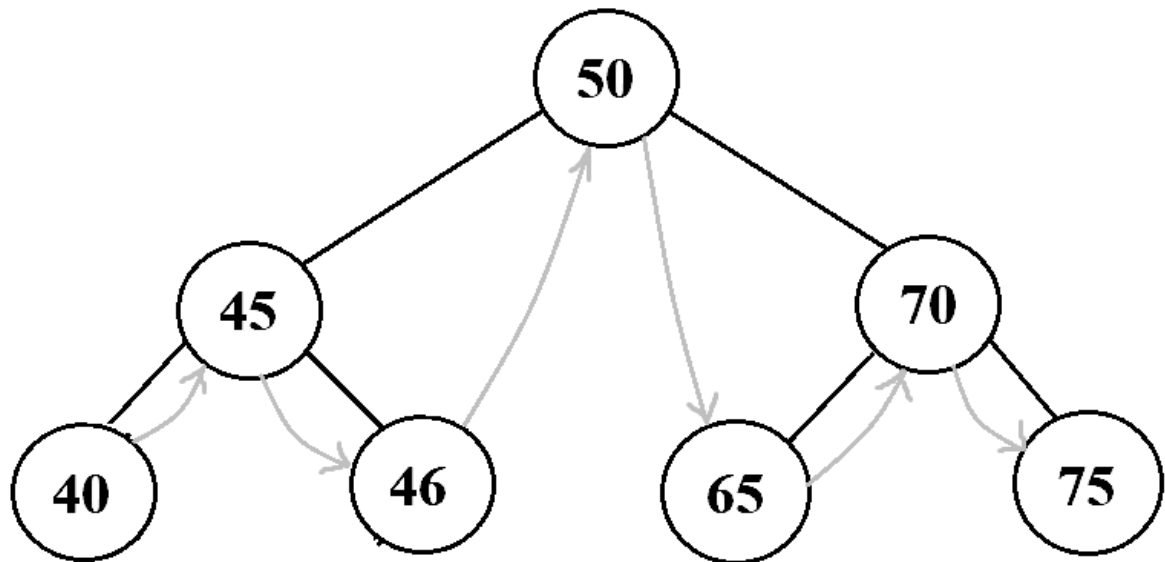


Рисунок 17 – ответ на задание 18

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

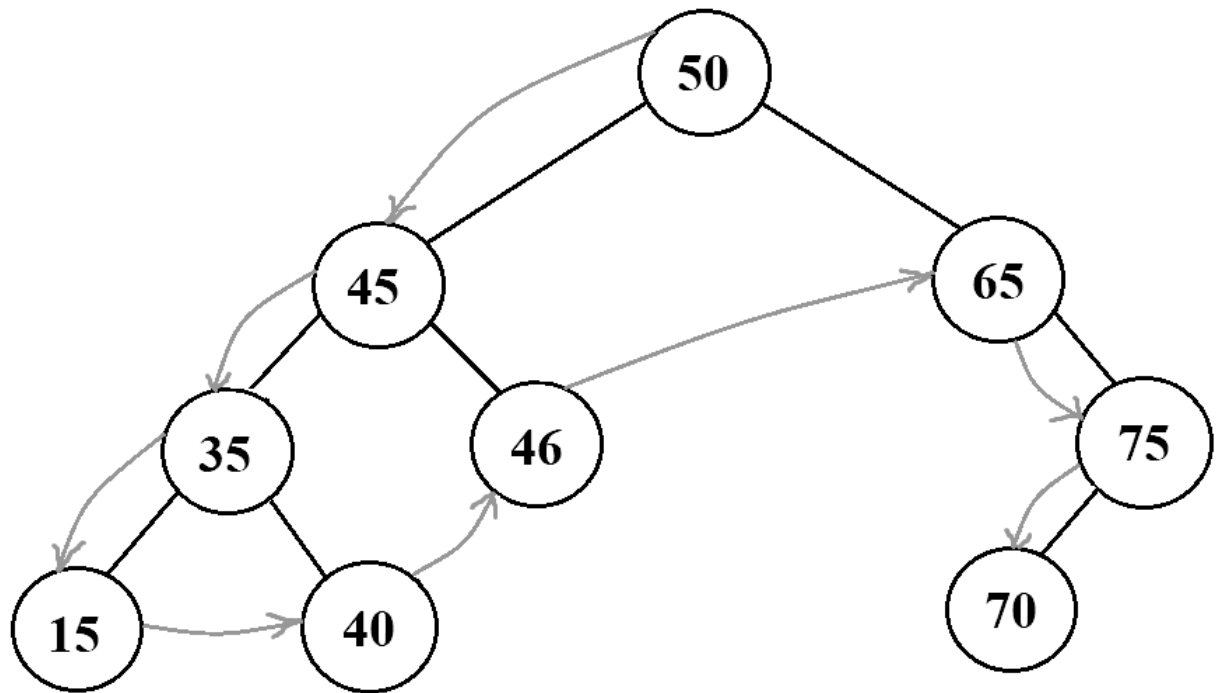


Рисунок 18 – Ответ на задание 19

Дано следующее бинарное дерево поиска

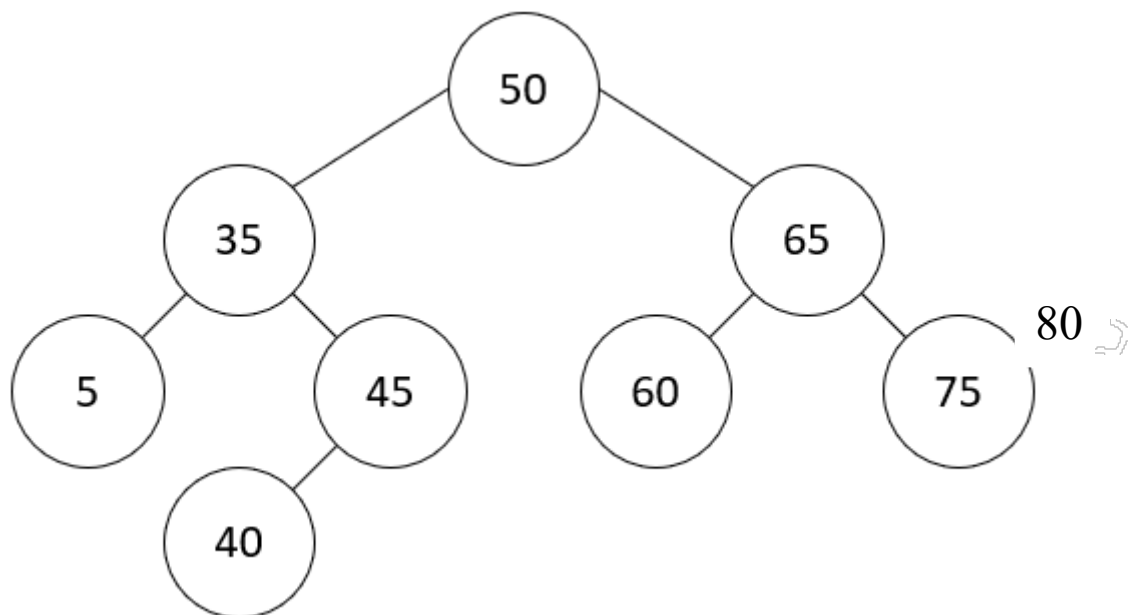


Рисунок 19 – Бинарное дерево поиска

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

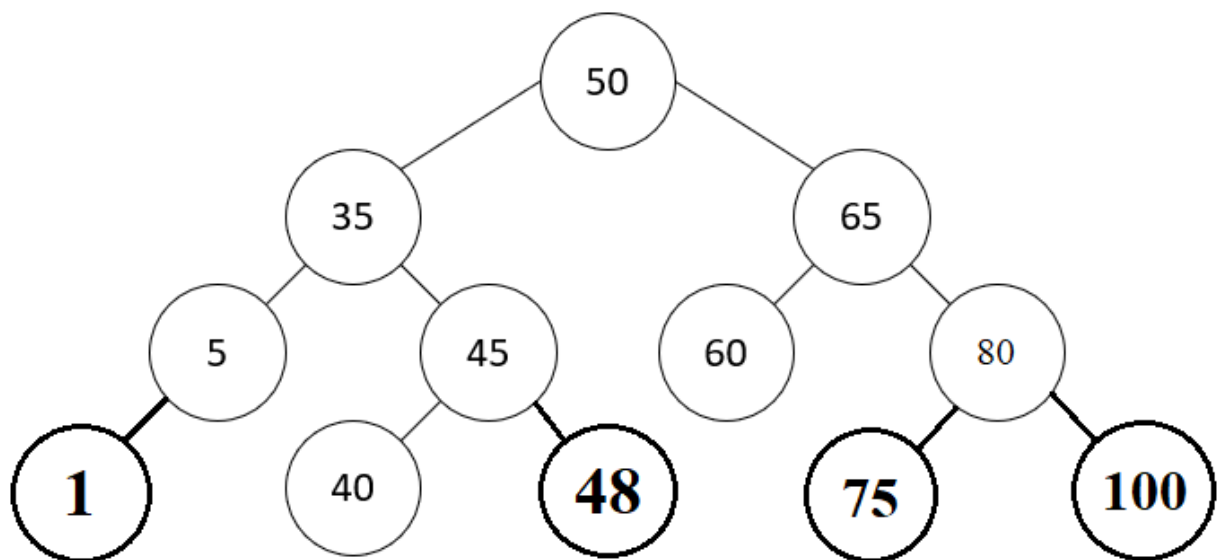


Рисунок 20 – Дерево поиска после добавления 1, 48, 75, 100

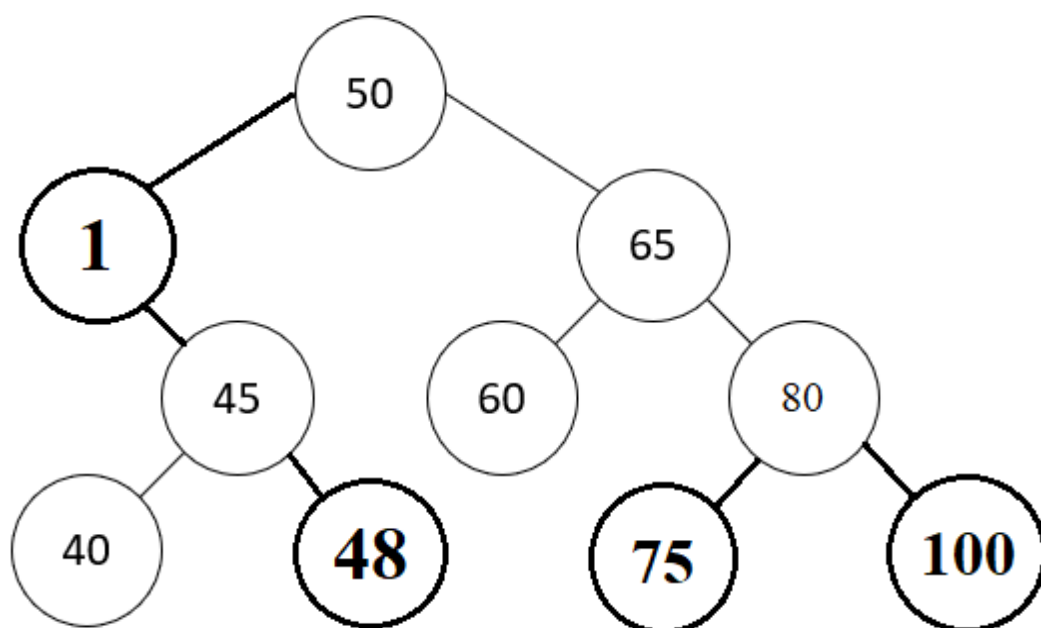


Рисунок 21 – Дерево поиска после удаления 5, 35

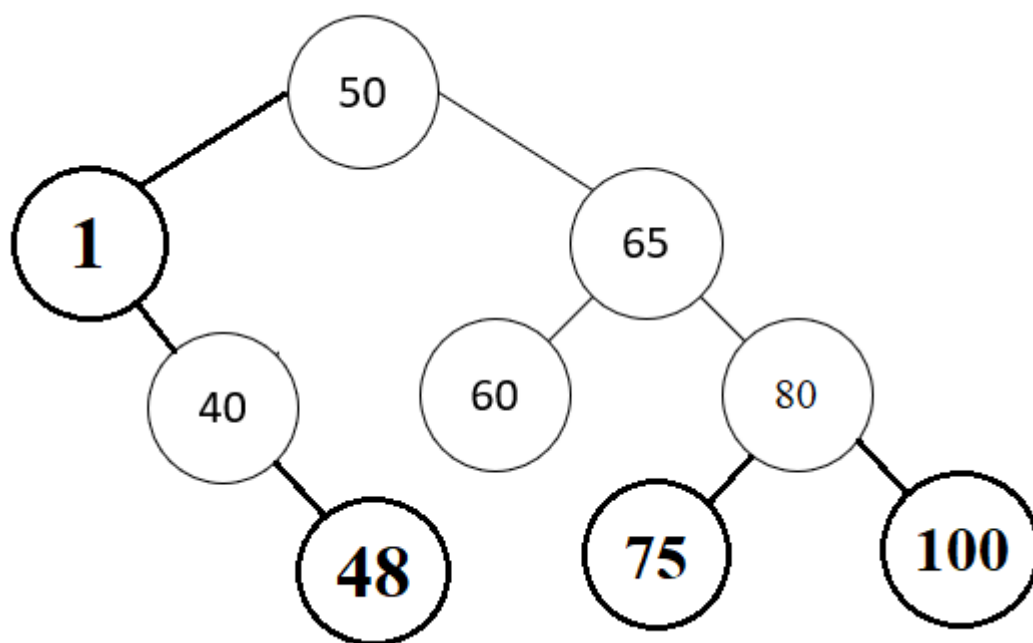


Рисунок 22 – Дерево поиска после удаления 45

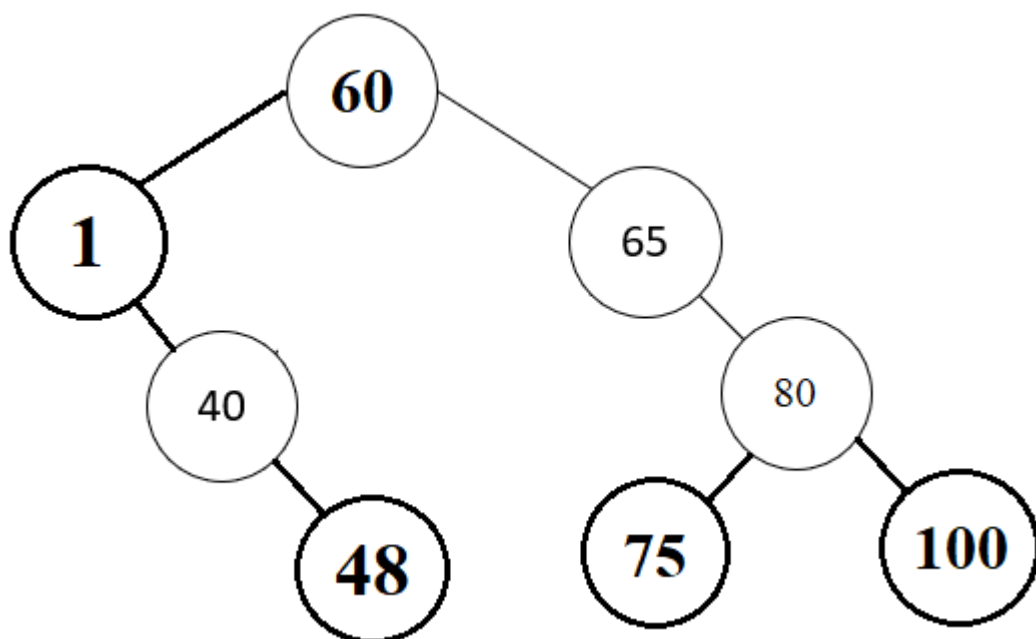


Рисунок 23 – Дерево поиска после удаления 50

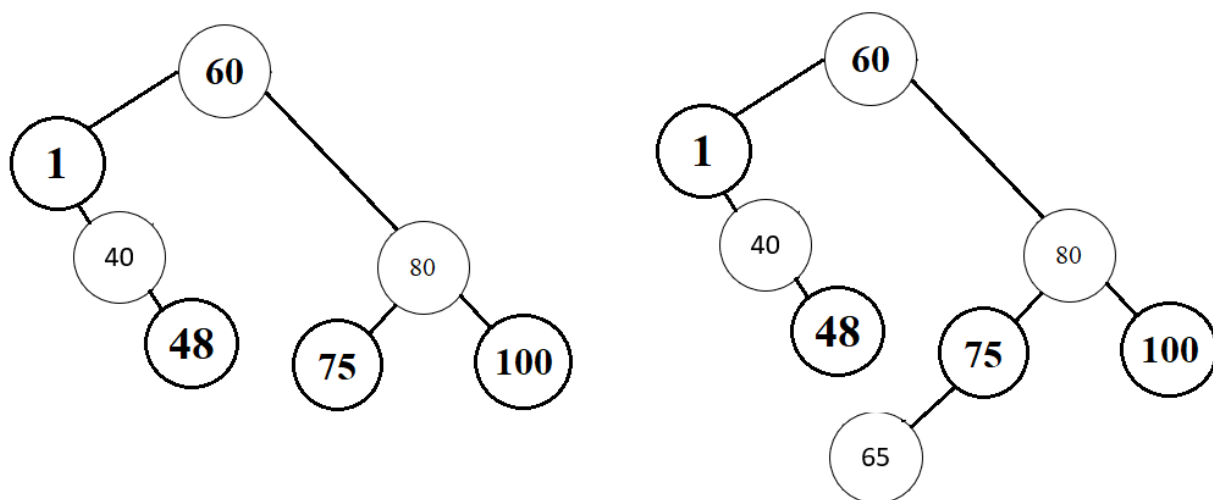


Рисунок 24 – Дерево поиска после удаления 65 и вставки его снова

Задание 2 (Вариант 20)

Формулировка задачи:

Вид дерева: дерево выражения.

1. Реализовать операции общие для вариантов с 11 по 20

Создать дерево выражений в соответствии с вводимым выражением.

Структура узла дерева включает: информационная часть узла – символьного типа: либо знак операции +, -, * либо цифра, указатель на левое и указатель на правое поддерево. В дереве выражения операнды в листьях дерева.

Исходное выражение имеет формат:

<формула>::=цифра|<формула><знак операции><формула>

2. Реализовать операции варианта.

Решение:

```
struct node
{
    char val;
    node* left;
    node* right;

    node(char userData)
    {
        this->val = userData;
        this->left = nullptr;
        this->right = nullptr;
    }
};

class exp_tree
{
private:
    node *root;

    int _recurr_calc(node* root)
    {
        char c = root->val;
        if (c == '+' || c == '-' || c == '*' || c == '/')
        {
            int leftValue = _recurr_calc(root->left);
            int rightValue = _recurr_calc(root->right);
            switch (c) {
                case '+':
                    return leftValue + rightValue;
                case '-':
                    return leftValue - rightValue;
                case '*':
                    return leftValue * rightValue;
                default:
                    return 0;
            }
        }
    }
};
```

```

    }
    else
    {
        return c - '0';
    }
}

std::string _recurr_str(node* root)
{
    std::string c(1, root->val);
    if (c == "+" || c == "-" || c == "*" || c == "/")
    {
        return c + this->_recurr_str(root->left) + this->_recurr_str(root->right);
    }
    else
    {
        return c;
    }
}

public:
exp_tree(std::string exp, std::string form="prefix")
{
    if (form == "prefix")
        for (size_t i = 0; i < exp.size() / 2; i++)
            std::swap(exp[i], exp[exp.size() - i - 1]);

    std::stack<node*> stack;
    for (char c : exp)
    {
        if (c == '+' || c == '-' || c == '*' || c == '/')
        {
            node* x = stack.top();
            stack.pop();
            node* y = stack.top();
            stack.pop();
            node* a = new node(c);

            if (form == "prefix")
            {
                a->left = x;
                a->right = y;
            }
            else
            {
                a->left = y;
                a->right = x;
            }

            stack.push(a);
        }
        else {
            stack.push(new node(c));
        }
    }
    this->root = stack.top();
}

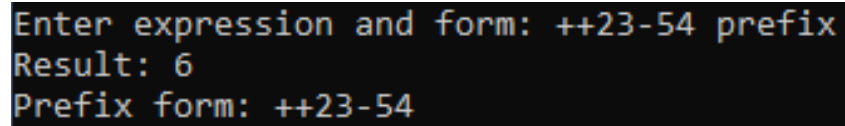
std::string get_prefix_form()
{
    return this->_recurr_str(this->root);
}

int calc()

```

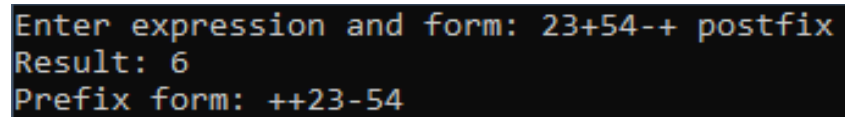
```
    {  
        return this->_recurr_calc(this->root);  
    }  
};
```

Листинг 1 – Код задачи 2



```
Enter expression and form: ++23-54 prefix  
Result: 6  
Prefix form: ++23-54
```

Рисунок 25 – результат тестирования



```
Enter expression and form: 23+54-+ postfix  
Result: 6  
Prefix form: ++23-54
```

Рисунок 26 – результат тестирования

Вывод

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