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 Бананов Б.Б.

группа БАНБ-АН-АН

Оглавление

Тема. Нелинейные структуры данных. Бинарное дерево.

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

Задание

Вариант: 17

Разработать программу, которая создает бинарное дерево поиска (БДП) и выполняет операции.

1. Реализовать операции общие для всех вариантов

   1) 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) 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.

   5) Для каждой представленной в программе функции предоставить отчет по ее разработке в соответствии с требованиями разработки программы (подпрограммы).
   6) Представить алгоритм основной программы и таблицу имен, используемых в алгоритме.

Вариант	Значение информационной части	Операции варианта
17	Целое число	1) Определить среднее арифметическое всех узлов дерева, используя алгоритм обхода в «ширину». 2) Определить количество узлов в дереве. 3) Удалить самый левый лист дерева 4) Определить уровень, на котором находиться заданное значение.

Функции

print_array_to_right(TreeNode* curr, int height, int step) – вывод дерева в правую сторону

print_array_to_left(TreeNode* curr, int height, int step) – вывод дерева в левую сторону

В обоих curr – корень дерева, height – уровень, на котором надо вывести узлы, step – рекурсивный шаг

print_tree(TreeNode* curr, int height) – deprecated

delete_tree(TreeNode* curr) – рекурсивное удаление дерева, начиная из корня curr

printArrayLeftToRight() – публичная функция для вывода разных уровней дерева в правую сторону

printArrayRightToLeft() – публичная функция для вывода разных уровней дерева в левую сторону

bool find(int key) – проверка наличия узла в дереве по значению key

insert(int key) – вставка узла со значением key в дерево

erase(int key) – удаление узла со значением key из дерева, если существует таковой.

int size() – GETTER m_size

double getAverage() – получение среднего значения из узлов деревьев

bool removeTheLeftNode(TreeNode* tree) – рекурсивная функция для удаления самого левого листа из дерева с корнем tree. В случае успешного удаления вернёт true, иначе false.

int findHeight(int key) – то же самое, что и функция find, но в процессе поиска считает кол-во спусков по наследников. Возвращает -1, если ошибка, и высоту от 0 узла с значением key в дереве.

Основной алгоритм

1) Ввод корневого значения дерева x
2) Создание объекта bt типа SearchBinaryTree с параметром x
3) Вывод меню

4) Ввод команды x. Если x из меню, то п.3, иначе выход.

5) Выполнение соответствующей команды(см. в int main() конструкцию switch-case). Переход на п.1

Таблица имён, используемые в основном алгоритме

Имя	Значение
int main()	Функция
X	Временная переменная для ввода
Bt	Переменная бинарное дерево поиска
SearchBinaryTree	Тип Бинарное Дерево Поиска(БДП)
Rand	Случайное положительное значение
bt.insert	Вставка элемента в дерево bt
bt.printArrayRightToLeft	Вывод дерева bt справа налево
bt.getAverage	Получение среднего значения bt
bt.size	Получение кол-ва узлов в дереве bt
bt.removeTheLeftNode	Удаление крайне левого листа в дереве bt
bt.findHeight	Получение высоту у узла с определённом значением в дереве bt

Разработка программы

Ex3.cpp

```
#include <memory>
#include <cassert>
#include <algorithm>

#include <vector>
#include <iostream>
#include <string>
#include <list>

using namespace std;



class SearchBinaryTree {
public:
    class TreeNode {
    public:
        int m_data;
        TreeNode* m_parent = NULL;
        TreeNode* m_left = NULL;
        TreeNode* m_right = NULL;
        TreeNode(int val) {
            m_parent = NULL;
            m_left = NULL; // В C++11 лучше использовать nullptr
            m_right = NULL;
```

```
            m_data = val;
        }
    };
private:
    TreeNode* m_root;
    int m_size;
    int m_height;

    void print_array_to_right(TreeNode*, int = 0, int = 0);
    void print_array_to_left(TreeNode*, int = 0, int = 0);
    void print_tree(TreeNode*, int = 0);
    void delete_tree(TreeNode*);
public:
    SearchBinaryTree(int);
    ~SearchBinaryTree();
    void printArrayLeftToRight();
    void printArrayRightToLeft();
    bool find(int);
    void insert(int);
    void erase(int);
    int size() { return m_size; };
    double getAverage() {
        list<TreeNode*> l;
        //Для начала поместим в очередь корень

        l.push_back(m_root);
        int sum = 0;
        int amount = 0;
        while (l.size() != 0) {
            TreeNode* tmp = l.front();
            l.pop_front();
            if (!tmp) continue;
            sum += tmp->m_data;
            amount++;
            //Если есть левый наследник, то помещаем его в очередь для дальнейшей
обработки
            if (tmp->m_left) {
                l.push_back(tmp->m_left);
            }
            //Если есть правый наследник, то помещаем его в очередь для дальнейшей
обработки
            if (tmp->m_right) {
                l.push_back(tmp->m_right);
            }
        }
        if (amount == 0) return 0;
        return sum*1. / amount;
    }
    bool removeTheLeftNode(TreeNode* tree = NULL) {
        if (!tree) tree = m_root;
        if (!tree) return false;
        bool o = false;
        if (tree->m_left) o = removeTheLeftNode(tree->m_left);
        if (o) return true;
        if (tree->m_right) o = removeTheLeftNode(tree->m_right);
        if (o) return true;
        erase(tree->m_data);
        return true;
    }
    int findHeight(int key) {
        TreeNode* curr = m_root;
        int height = 0;
        if (!curr) return -1;
        for (; curr && curr->m_data != key; height++) {
```

```
            if (curr->m_data > key)
                curr = curr->m_left;
            else
                curr = curr->m_right;
        }
        if (!curr) return -1;
        return height;
    }
};
SearchBinaryTree::SearchBinaryTree(int key) {
    m_root = new TreeNode(key);
    m_size = 1;
    m_height = 1;
}
SearchBinaryTree::~SearchBinaryTree() {
    delete_tree(m_root);
}

void SearchBinaryTree::delete_tree(TreeNode* curr) {
    if (curr) {
        delete_tree(curr->m_left);
        delete_tree(curr->m_right);
        delete curr;
        m_size--;
    }
}
void SearchBinaryTree::printArrayLeftToRight() {
    if (m_size < 1) {
        printf("Пусто!\n");
        return;
    }
    for (int i = 0; i <= m_height; i++) {
        print_array_to_right(m_root, i);
        cout << endl;
    }
}

void SearchBinaryTree::printArrayRightToLeft() {
    if (m_size < 1) {
        printf("Пусто!\n");
        return;
    }
    for (int i = 0; i <= m_height; i++) {
        print_array_to_left(m_root, i);
        cout << endl;
    }
}

void SearchBinaryTree::print_array_to_right(TreeNode* curr, int height, int step) {
    if (step >= height) {
        if (curr) printf("%2d  ", curr->m_data);
        else printf("%2s  ", "{}");
        return;
    }
    if (curr) {
        //cout << curr->m_data << "  ";
        print_array_to_left(curr->m_left, height, step + 1);
        print_array_to_left(curr->m_right, height, step + 1);
    }
}

void SearchBinaryTree::print_array_to_left(TreeNode* curr, int height, int step) {
    if (step >= height) {
        if (curr) printf("%2d  ", curr->m_data);
```

```
        else printf("%2s  ", "{}");
        return;
    }
    if (curr) {
        //cout << curr->m_data << "  ";
        print_array_to_left(curr->m_right, height, step + 1);
        print_array_to_left(curr->m_left, height, step + 1);
    }
}

void SearchBinaryTree::print_tree(TreeNode* curr, int height) {
    if (curr) // Проверка на ненулевой указатель
    {

        print_array_to_right(curr->m_left);
        cout << curr->m_data << " ";
        print_array_to_right(curr->m_right);
    }

}

bool SearchBinaryTree::find(int key) {
    TreeNode* curr = m_root;
    while (curr && curr->m_data != key) {
        if (curr->m_data > key)
            curr = curr->m_left;
        else
            curr = curr->m_right;
    }
    return curr != NULL;
}
void SearchBinaryTree::insert(int key) {
    TreeNode* curr = m_root;
    int height = 0;
    TreeNode* newNode = new TreeNode(key);
    if (!m_root) {
        m_root = newNode;
        m_size = 1;
        return;
    }
    while (curr && curr->m_data != key) {
        height++;
        if (curr->m_data > key && curr->m_left == NULL) {
            curr->m_left = newNode;
            newNode->m_parent = curr;
            ++m_size;
            if (m_height < height) m_height = height;
            return;
        }
        if (curr->m_data < key && curr->m_right == NULL) {
            curr->m_right = newNode;
            newNode->m_parent = curr;
            ++m_size;
            if (m_height < height) m_height = height;
            return;
        }
        if (curr->m_data > key)
            curr = curr->m_left;
        else
            curr = curr->m_right;
    }
}
void SearchBinaryTree::erase(int key) {
    TreeNode* curr = m_root;
```

```
    TreeNode* parent = NULL;
    while (curr && curr->m_data != key) {
        parent = curr;
        if (curr->m_data > key) {
            curr = curr->m_left;
        } else {
            curr = curr->m_right;
        }
    }
    if (!curr)
        return;
    if (curr->m_left == NULL) {
        // Вместо curr подвешивается его правое поддерево
        if (curr->m_right && curr->m_right->m_parent == curr)
            curr->m_right->m_parent = curr->m_parent;
        if (m_root == curr) m_root = curr->m_right;

        if (parent && parent->m_left == curr)
            parent->m_left = curr->m_right;
        if (parent && parent->m_right == curr)
            parent->m_right = curr->m_right;
        --m_size;
        delete curr;
        return;
    }
    if (curr->m_right == NULL) {
        // Вместо curr подвешивается его левое поддерево
        if (curr->m_left && curr->m_left->m_parent == curr)
            curr->m_left->m_parent = curr->m_parent;
        if (m_root == curr) m_root = curr->m_left;

        if (parent && parent->m_left == curr)
            parent->m_left = curr->m_left;
        if (parent && parent->m_right == curr)
            parent->m_right = curr->m_left;
        --m_size;
        delete curr;
        return;
    }
    // У элемента есть два потомка, тогда на место элемента поставим
    // наименьший элемент из его правого поддерева
    TreeNode* replace = curr->m_right;
    while (replace->m_left)
        replace = replace->m_left;
    int replace_value = replace->m_data;
    erase(replace_value);
    curr->m_data = replace_value;
}


int main() {
    setlocale(LC_ALL, "RUS");
    int a;
    cout << "Введите значение корня(0;99):\n";
    cin >> a;
    if (a < 0) a *= -1;
    SearchBinaryTree bt = SearchBinaryTree(a % 100);
    int choose = -1;
    int maxRepeats = 2;
    for (int repeat = maxRepeats; repeat > 0;) {
        printf("Меню:\n");
        printf("1. Добавить случайное число\n");
        printf("2. Добавить вводимое число\n");
        printf("3. Вывести дерево справа-налево\n");
```

```
        printf("4. Вывести среднее значение\n");
        printf("5. Вывести кол-во узлов в дереве\n");
        printf("6. Удалить самый левый лист дерева\n");
        printf("7. Найти уровень узла с вводимым значением\n");
        if (repeat < maxRepeats) printf("Вы хотите выйти? Напишите любой символ не из меню
ещё %d раз, чтобы выйти!\n", repeat);
        cin >> choose;
        int x;
        switch (choose) {
        case 1:
            x = rand() % 100;
            printf("Добавляем случайное число %d\n", x);
            bt.insert(rand() % 100);
            break;
        case 2:
            printf("Введите добавляемое число:\n");
            cin >> x;
            bt.insert(x);
            break;
        case 3:
            printf("Дерево, изображённое справа-налево:\n");
            bt.printArrayRightToLeft();
            break;
        case 4:
            printf("Среднее значение: %f\n", bt.getAverage());
            break;
        case 5:
            printf("Кол-во узлов в дереве: %d\n", bt.size());
            break;
        case 6:
            printf("Удаление самого левого листа дерева\n");
            bt.removeTheLeftNode();
            break;
        case 7:
            printf("Введите значение узла с искомым уровнем:\n");
            cin >> x;
            printf("Узел со значением %d имеет высоту: %d\n", x, bt.findHeight(x));
            break;
        default:
            repeat--;
            continue;
        }
        repeat = maxRepeats;
    }
    system("pause");
    return 0;
}
```

Зависимости

packages.config

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="boost" version="1.77.0.0" targetFramework="native" />
</packages>
```

Тесты

```
Введите значение корня(0;99):
50
Меню:
1. Добавить случайное число
2. Добавить вводимое число
3. Вывести дерево справа-налево
4. Вывести среднее значение
5. Вывести кол-во узлов в дереве
6. Удалить самый левый лист дерева
7. Найти уровень узла с вводимым значением
1
Добавляем случайное число 41
Меню:
1. Добавить случайное число
2. Добавить вводимое число
3. Вывести дерево справа-налево
4. Вывести среднее значение
5. Вывести кол-во узлов в дереве
6. Удалить самый левый лист дерева
7. Найти уровень узла с вводимым значением
2
Введите добавляемое число:
28
Меню:
1. Добавить случайное число
2. Добавить вводимое число
3. Вывести дерево справа-налево
4. Вывести среднее значение
5. Вывести кол-во узлов в дереве
6. Удалить самый левый лист дерева
7. Найти уровень узла с вводимым значением
3
Дерево, изображённое справа-налево:
50
67  28
Меню:
1. Добавить случайное число
2. Добавить вводимое число
3. Вывести дерево справа-налево
4. Вывести среднее значение
5. Вывести кол-во узлов в дереве
6. Удалить самый левый лист дерева
7. Найти уровень узла с вводимым значением
4
Среднее значение: 48,333333
Меню:
1. Добавить случайное число
2. Добавить вводимое число
3. Вывести дерево справа-налево
4. Вывести среднее значение
5. Вывести кол-во узлов в дереве
6. Удалить самый левый лист дерева
7. Найти уровень узла с вводимым значением
5
Кол-во узлов в дереве: 3
Меню:
1. Добавить случайное число
2. Добавить вводимое число
3. Вывести дерево справа-налево
4. Вывести среднее значение
5. Вывести кол-во узлов в дереве
6. Удалить самый левый лист дерева
7. Найти уровень узла с вводимым значением
3
Дерево, изображённое справа-налево:
50
67  28
Меню:
1. Добавить случайное число
2. Добавить вводимое число
3. Вывести дерево справа-налево
```

Ответы на вопросы

1. **Что определяет степень дерева?**

   Степень дерева определяет максимальную степень его узлов.

2. **Какова степень сильноветвящегося дерева?**

   Степень сильноветвящегося дерева больше 2.

3. **Что определяет путь в дереве?**

   Путь в дереве определяет последовательность узлов от корня до нужного узла.

4. **Как рассчитать длину пути в дереве?**

   Чтобы рассчитать длину пути в дереве нужно посчитать сумму длин его ребер. (Длина пути дерева определяется как сумма длин путей ко всем его вершинам.)

5. **Какова степень бинарного дерева?**

   Степень бинарного дерева равна 2.

6. **Может ли дерево быть пустым?**

   Дерево называется пустым, если оно не содержит ни одной вершины.

7. **Дайте определение бинарного дерева?**

   Бинарное дерево — это дерево у каждого узла, которого не более 2 потомков.

8. **Дайте определение алгоритму обхода.**

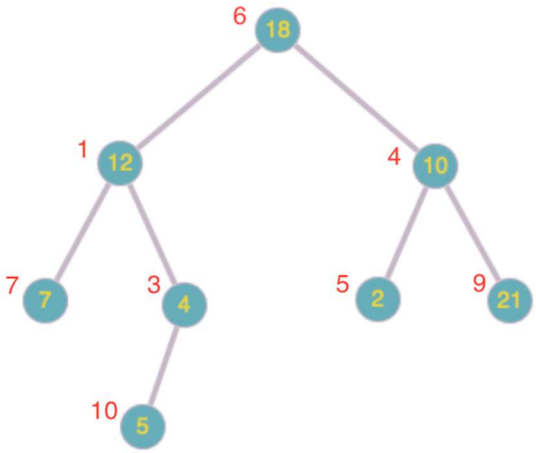
   Обход дерева – это упорядоченная последовательность вершин дерева, в которой каждая вершина встречается только один раз.

9. **Приведите рекуррентную зависимость для вычисления высоты дерева.**

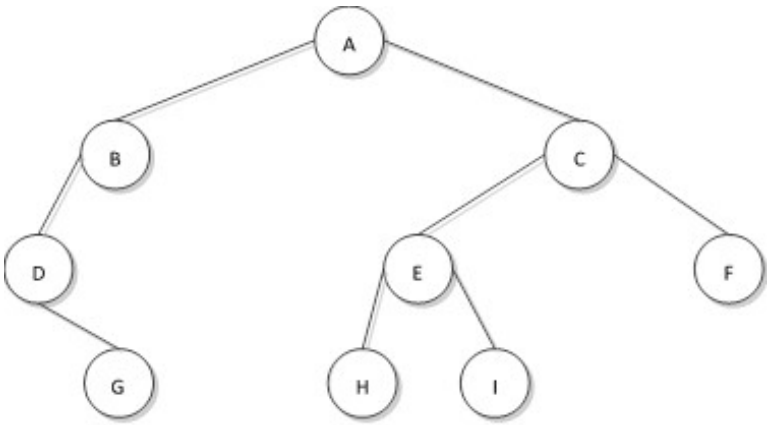
$$h(T) = \begin{cases} -1, & \text{если } T = NULL \\ 1 + max(h(T.left), h(T.right)), & \text{иначе} \end{cases}$$

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

Индекс	Key	left	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 алгоритму: прямой; обратный; симметричный



Путь обхода дерева в прямом порядке: ABDGCEHIF.

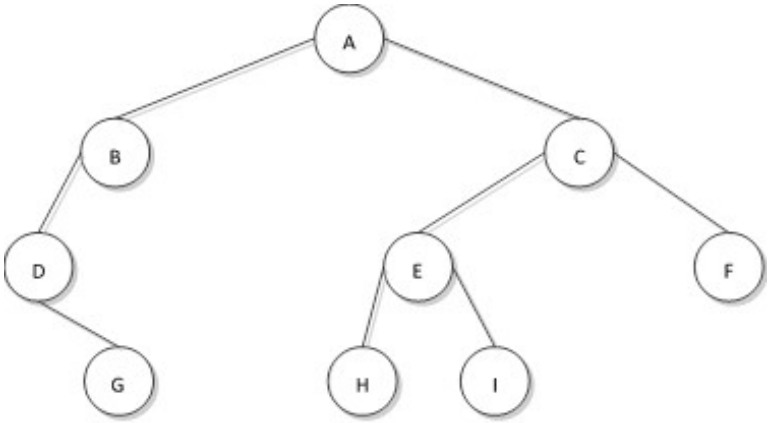
Путь обхода дерева в обратном порядке: GDBHIEFCA.

Путь обхода дерева в симметричном порядке: DGBAHEICF.

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

В алгоритме обхода дерева методом в «ширину» используется очередь.

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



Путь обхода дерева в «ширину»: ABCDEFGHI.

Шаг 1. Очередь: A.

Шаг 2. Очередь: BC. Вывод: A.

Шаг 3. Очередь: CD. Вывод: B.

Шаг 4. Очередь: DEF. Вывод: C.

Шаг 5. Очередь: EFG. Вывод: D.

Шаг 6. Очередь: FGHI. Вывод: E.

Шаг 7. Очередь: GHI. Вывод: F.

Шаг 8. Очередь: HI. Вывод: G.

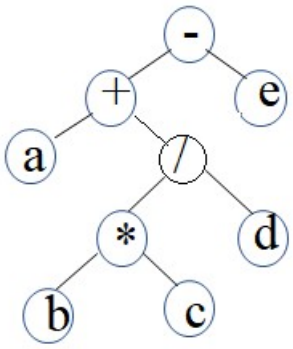
Шаг 9. Очередь: I. Вывод: H.

Шаг 10. Очередь: пусто. Вывод: I.

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

В не рекурсивном обходе дерева методом в «глубину» используется стек.

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



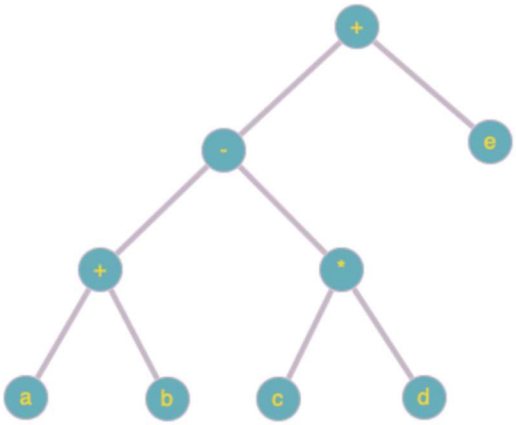
Путь обхода дерева в прямом порядке: -+a/*bcde.

Путь обхода дерева в симметричном порядке: a+b*c/d-e.

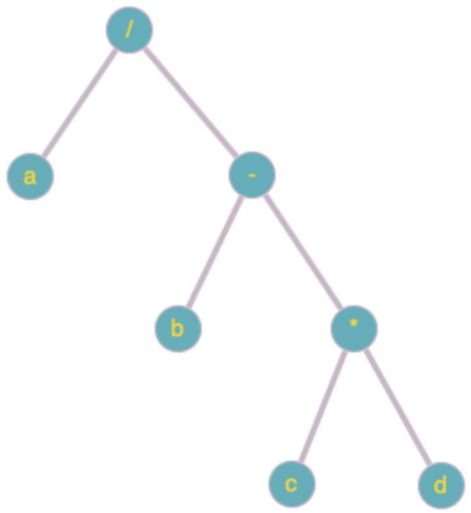
Путь обхода дерева в обратном порядке: abc*d/+e-.

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

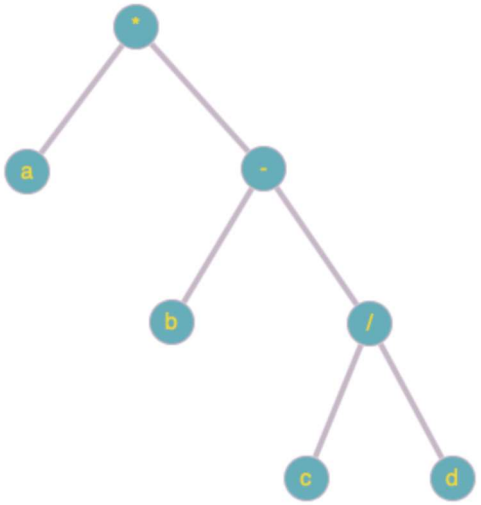
1. a+b-c*d+e



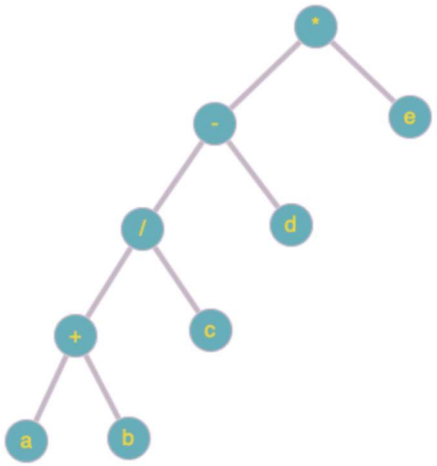
2. /a-b*c d



3. a b c d/ - *



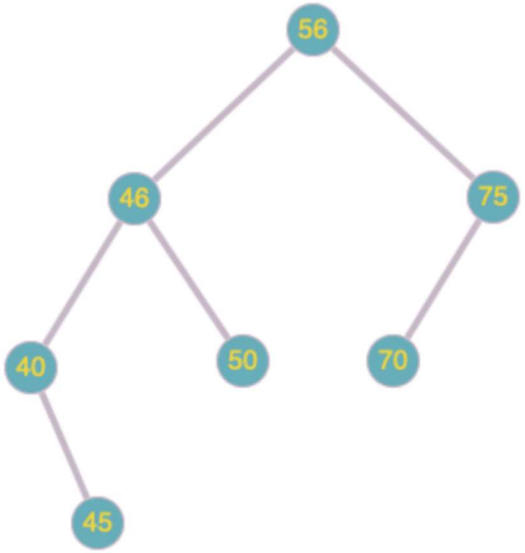
4. *-/+abcde



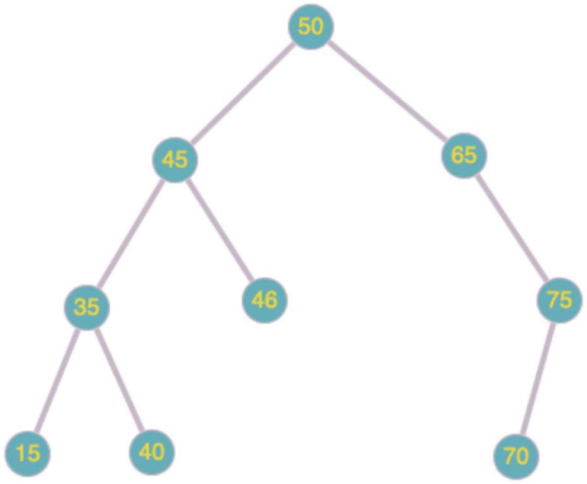
17.В каком порядке будет проходиться бинарное дерево, если алгоритм обхода в ширину будет запоминать узлы не в очереди, а в стеке?

Если алгоритм обхода в ширину будет запоминать узлы в стеке, то бинарное дерево будет проходиться в прямом порядке.

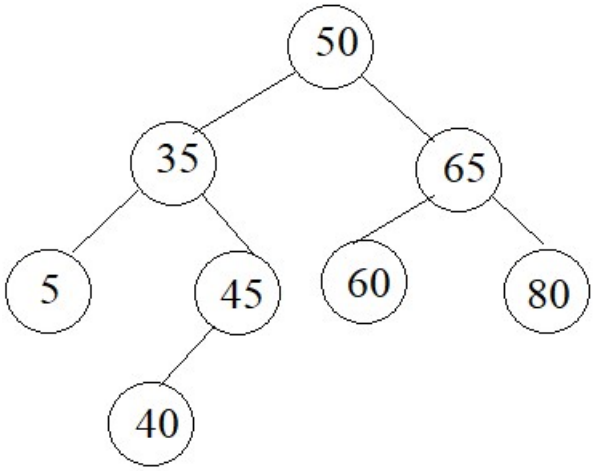
18. Постройте бинарное дерево поиска, которое в результате симметричного обхода дало бы следующую последовательность узлов: 40 45 46 50 65 70 75.



19. Последовательность {50 45 35 15 40 46 65 75 70} получена путем прямого обхода бинарного дерева поиска. Постройте это дерево.

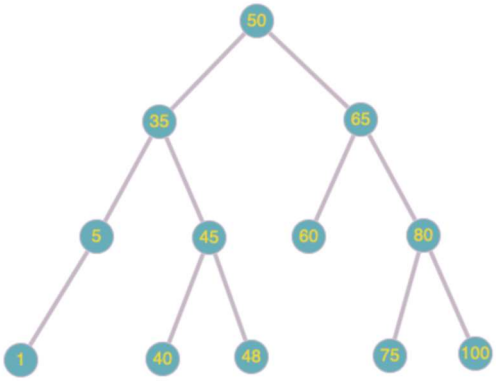


20. Дано бинарное дерево поиска.

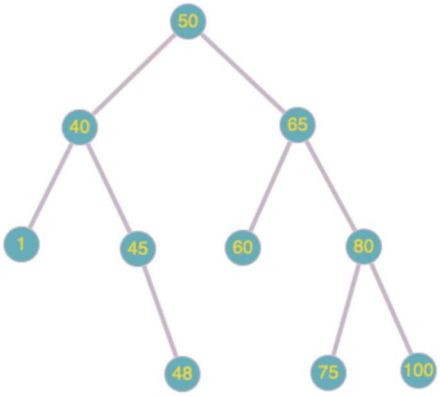


Выполните действия над исходным деревом и покажите дерево:

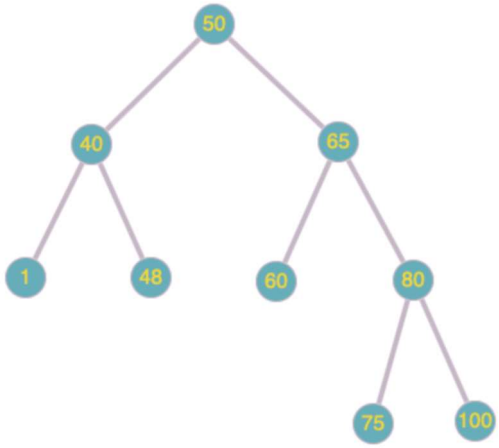
1) **после включения узлов 1, 48, 75, 100**



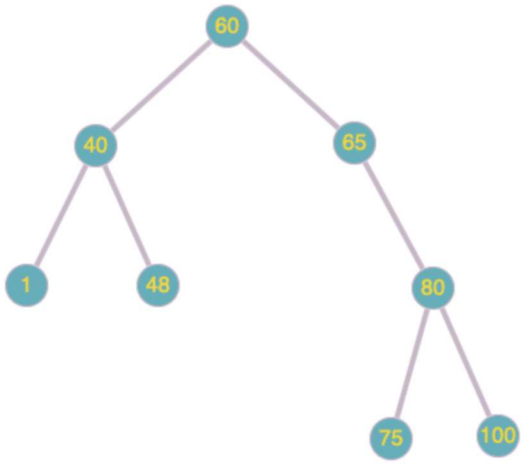
2) **после удаления узлов 5, 35**



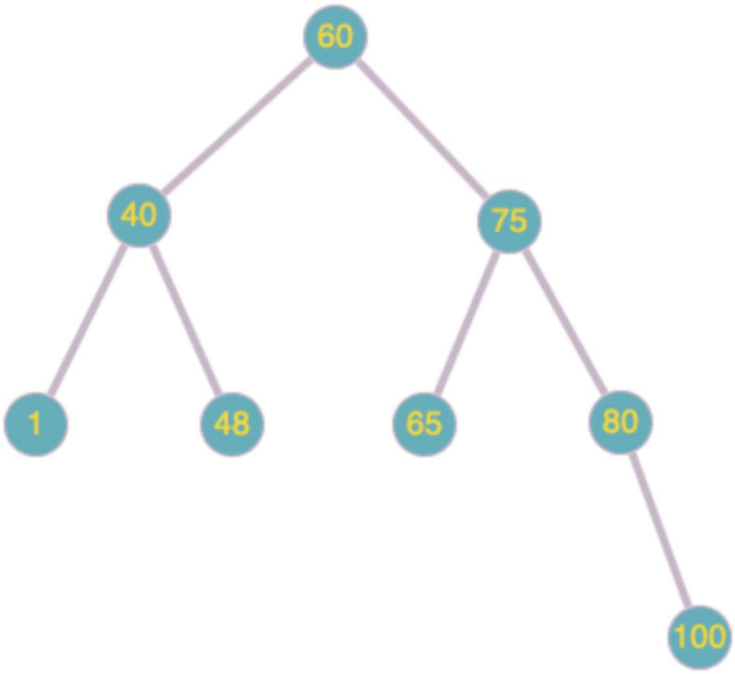
3) **после удаления узла 45**



4) после удаления узла 50



5) после удаления узла 65 и вставки его снова



Выводы

В результате проделанной работы, я получил умения и навыки разработки и реализации операций над структурой данных бинарное дерево. Была разработана программа, которая создает бинарное дерево поиска (БДП) и выполняет операции