Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6 ПО ДИСЦИПЛИНЕ ТИПЫ И СТРУКТУРЫ ДАННЫХ

Студент **Паншин Сергей Константинович**

Группа **ИУ7-33Б**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Паншин С. К.**

Преподаватель _____ **Никульшина Т. А.**

Преподаватель _____ **Барышникова М. Ю.**

Оценка _____

2023 г.

Условие задачи

Рассмотреть применение двоичных деревьев, реализовать основные операции над деревьями: обход деревьев, включение, исключение и поиск узлов.

Техническое задание

Построить двоичное дерево поиска из букв вводимой строки. Вывести его на экран в виде дерева. Выделить цветом все буквы, встречающиеся более одного раза. Удалить из дерева эти буквы. Вывести оставшиеся элементы дерева при постфиксном его обходе. Сравнить время удаления повторяющихся букв из дерева и из строки.

Входные данные:

Пункт меню (число от 0 до 2):

```
Menu:
    1) Process string
    2) Comparison of delete types
    0) Quit program
```

Также обрабатываемая строка

Выходные данные:

Картинка изначального графа и конечного, а также конечный граф в постфиксном обходе.

Возможные аварийные ситуации:

Некорректный ввод: пункта меню, пустая строка

Способ обращения к программе

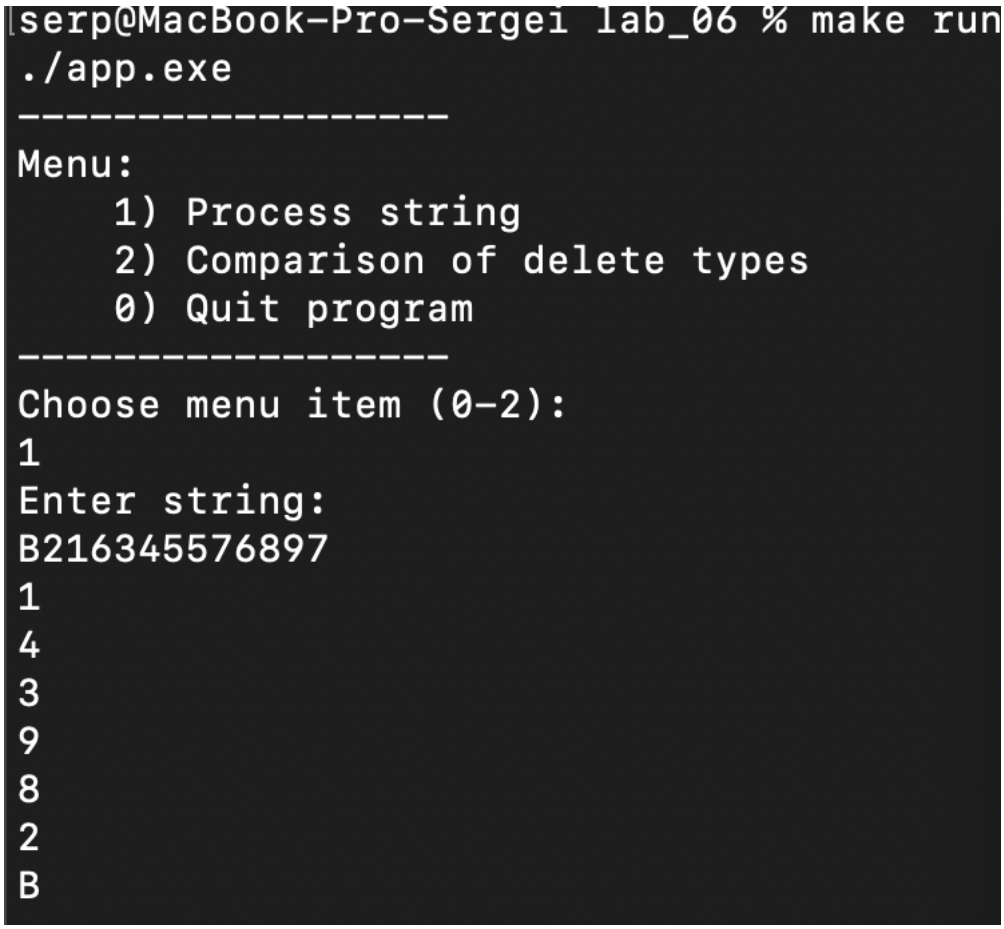
В папке с программой запустить команду *make run*.

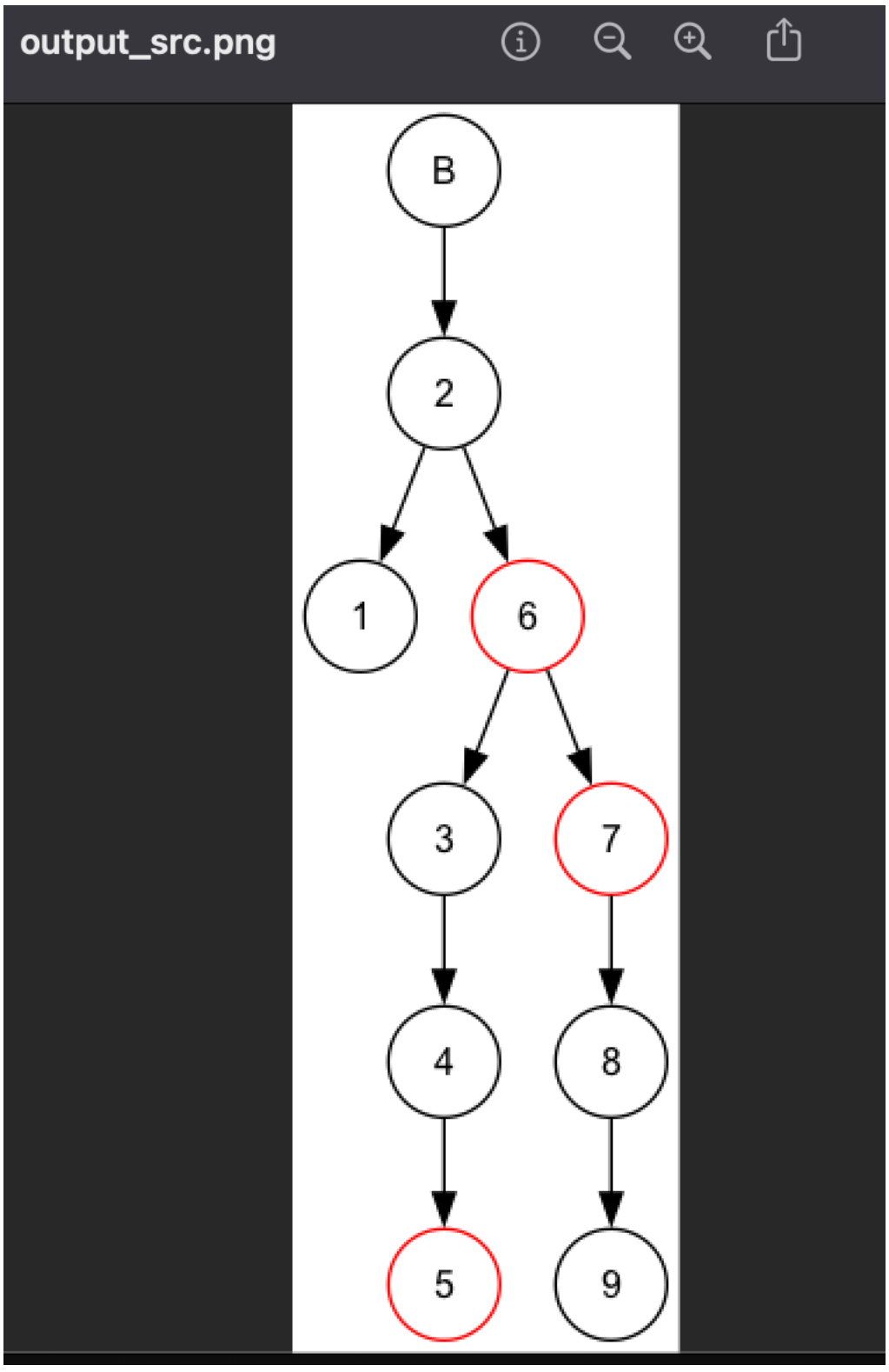
Структуры данных

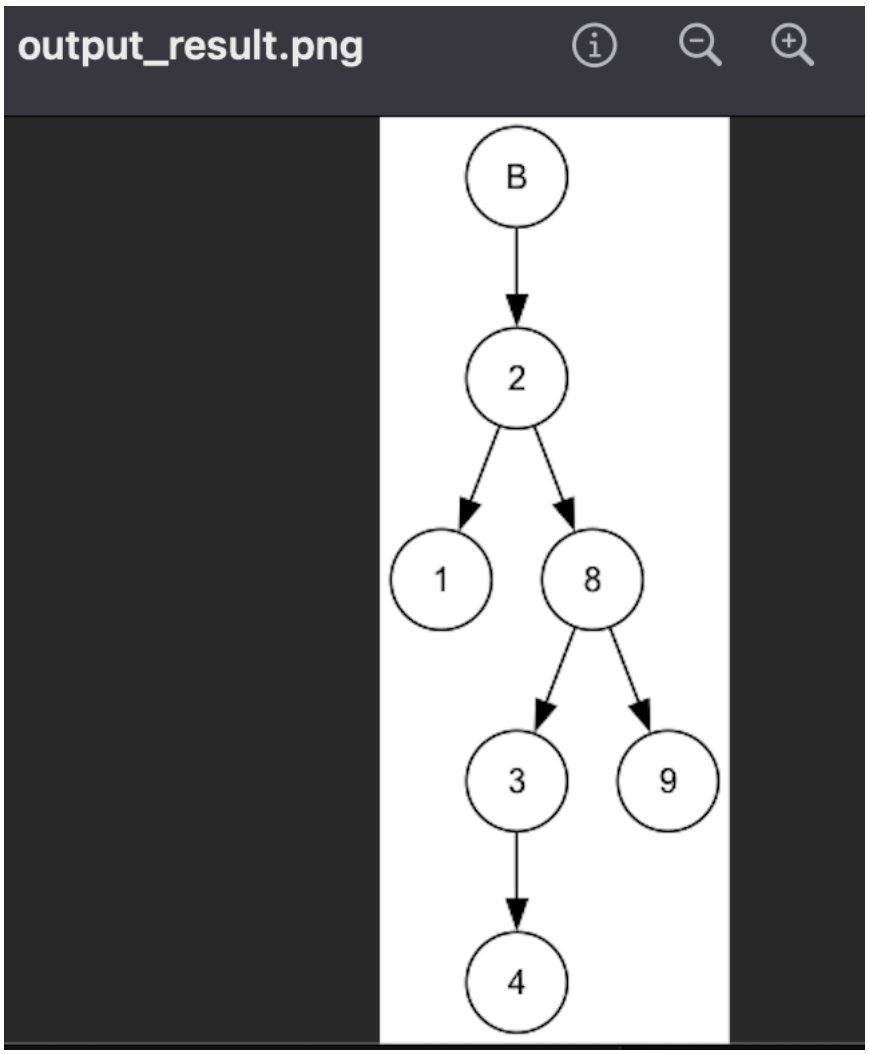
```
// узел дерева
typedef struct tree_node
{
    char value;
    size_t count;
    struct tree_node *parent;
    struct tree_node *left;
    struct tree_node *right;
} tree_node_t;


// Создания узла дерева
// Вход: указатель на указатель по которому хранится узел,
значение
// Выход: код возврата
int node_init(tree_node_t **node, char value);

// Освобождение памяти из-под узла дерева
// Вход: указатель, по которому хранится узел
void node_free(tree_node_t *node);

// Освобождение памяти из-под дерева
// Вход: указатель, по которому хранится узел
void tree_free(tree_node_t **tree);

// Создания дерева из элементов строки
// Вход: указатель на строку, размер строки
// Выход: указатель на корень дерева
tree_node_t *tree_create_from_str(char *str, size_t len);

// Добавление узла в дерево
// Вход: указатель на указатель по которому хранится корень,
значение
// Выход: код возврата
int tree_add_el(tree_node_t **tree, char value);

// Вывод дерева при постфиксном обходе
// Вход: указатель на корень
void tree_print_post_order(tree_node_t *tree);

// Удаление не уникальных узлов
// Вход: указатель на указатель по которому хранится корень
void tree_del_not_unique_nodes(tree_node_t **tree);
```

```
// Применение функции к дереву при префиксном обходе
// Вход: указатель на указатель на корень, указатель на функцию, указатель на аргументы
void tree_apply_pre(tree_node_t *tree, void (*f)(tree_node_t *, void *), void *arg);

// Экспорт дерева в формат dot для отрисовки в виде картинки
// Вход: указатель на файл, имя для дерева, указатель на корень,
void tree_export_to_dot(FILE *f, const char *tree_name, tree_node_t *tree);
```

Пример работы




output_src.png
B
2
1
6
3
7
4
8
5
9



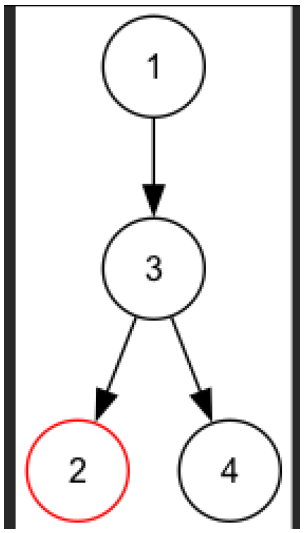
Замеры

Производится 50 итераций при замерах.

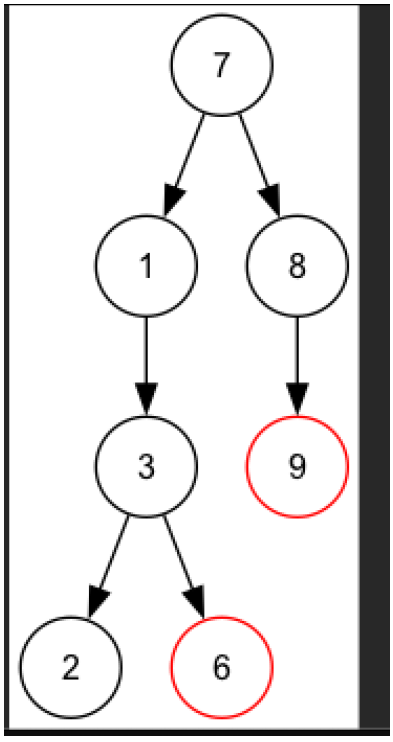
DEL ANALYZE

Len	String time(ns)	Tree time(ns)	Max string size(bytes)	Max tree size(bytes)
5	120	220	5	200
9	220	620	9	360
10	520	640	10	400
15	1100	1180	15	600

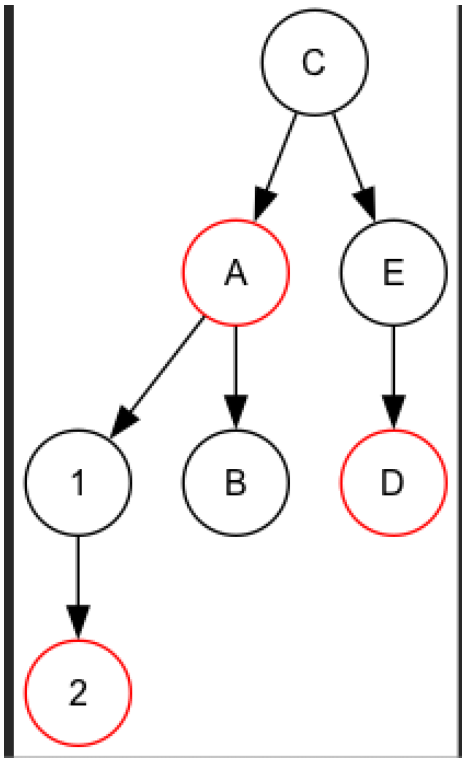
Дерево первого теста



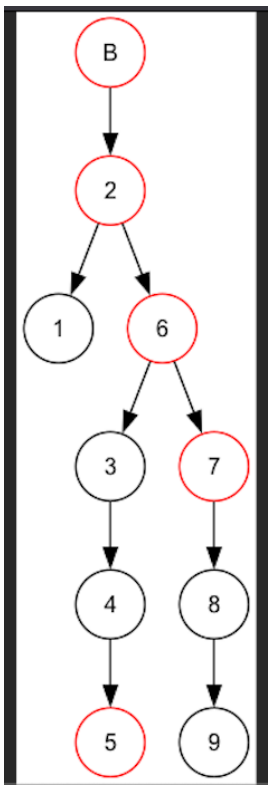
Дерево второго теста



Дерево третьего теста



Дерево четвертого теста



Выводы по проделанной работе

Деревья удобно применять при хранении иерархических данных или в случаях, когда нужен быстрый поиск каких-то элементов, так как в бинарном дереве он быстр. Но при использовании деревьев надо учитывать, что удаление узлов дерева может занять продолжительное время, так как там идет рекурсивное перестроение дерева в случае удаления узла, у которого есть оба потомка.

Также в моей реализации алгоритма удаления из-за того что он сначала рекурсивно ищет по дереву узел, который надо удалить, а потом при удалении рекурсивно также идет до него, то при замере работы всей этой функции получается, что она всегда проигрывает по времени удалении повторений из строки.

Контрольные вопросы

1. Что такое дерево? Как выделяется память под представление деревьев?

Дерево – это нелинейная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим».

В виде связного списка, динамически под каждый узел

2. Какие бывают типы деревьев?

Двоичное дерево - иерархическая структура данных, в которой каждый узел имеет не более двух потомков

АВЛ-дерево — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

Красно-чёрное дерево — один из видов самобалансирующихся двоичных деревьев поиска, гарантирующих логарифмический рост высоты дерева

от числа узлов и позволяющее быстро выполнять основные операции дерева поиска: добавление, удаление и поиск узла.

И другие.

3. Какие стандартные операции возможны над деревьями?

Обход дерева, включение, исключение и поиск узлов

4. Что такое дерево двоичного поиска?

Двоичное дерево, для каждого узла которого сохраняется условие: Левый потомок меньше родителя, правый потомок больше родителя