|  |  |
| --- | --- |
|  | **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) Сomparison 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

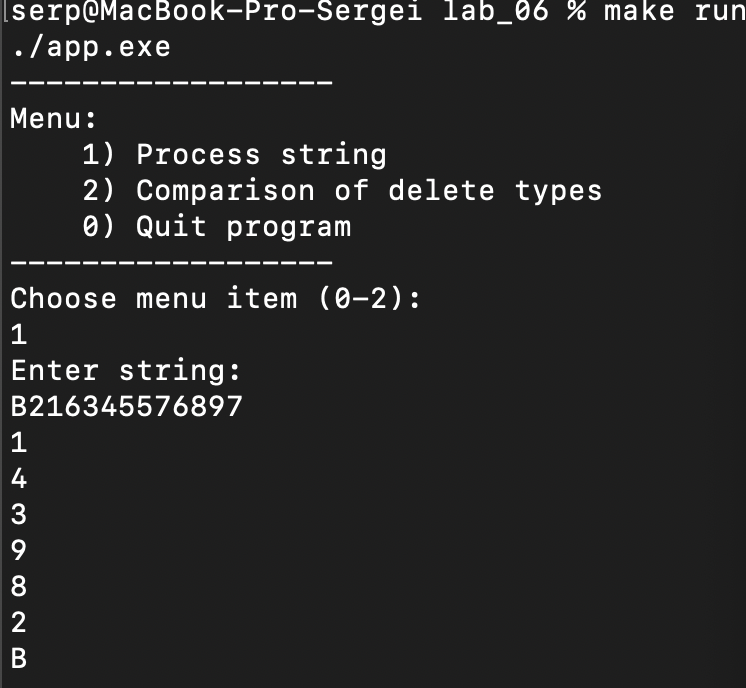
void tree\_apply\_pre(tree\_node\_t \*tree, void (\*f)(tree\_node\_t \*, void \*), void \*arg);

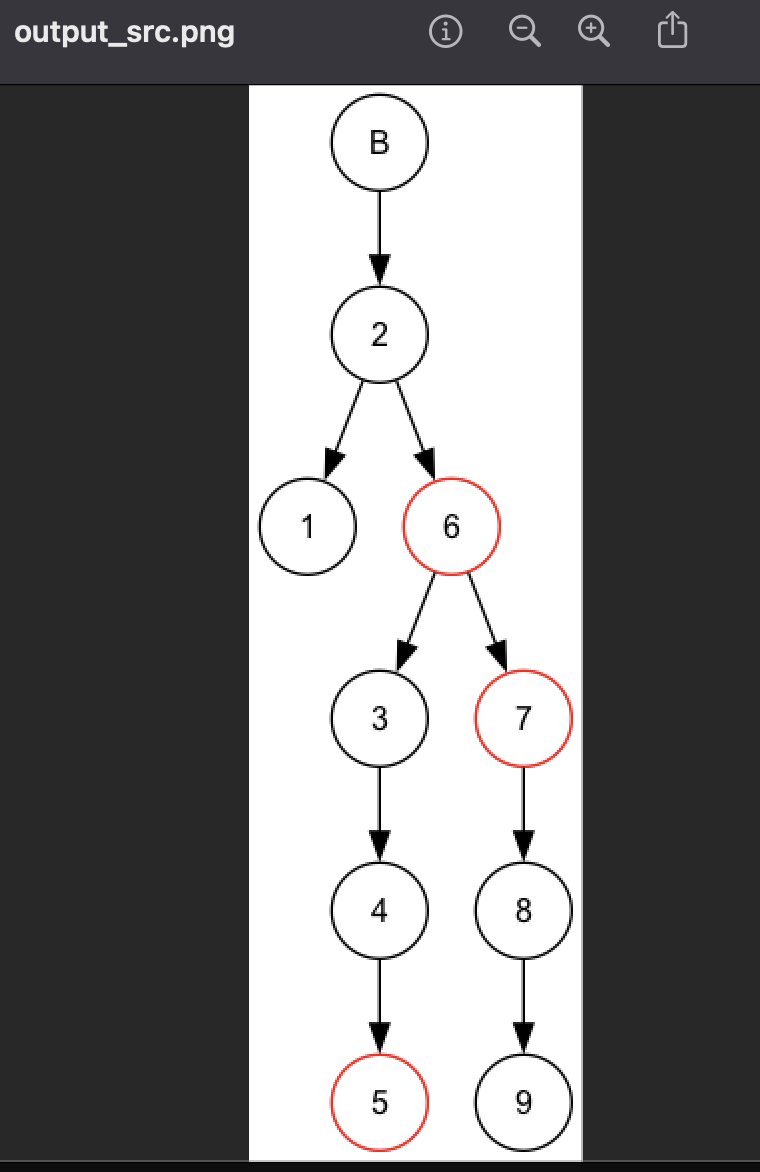
// Экспорт дерева в формат dot для отрисовки в виде картинки

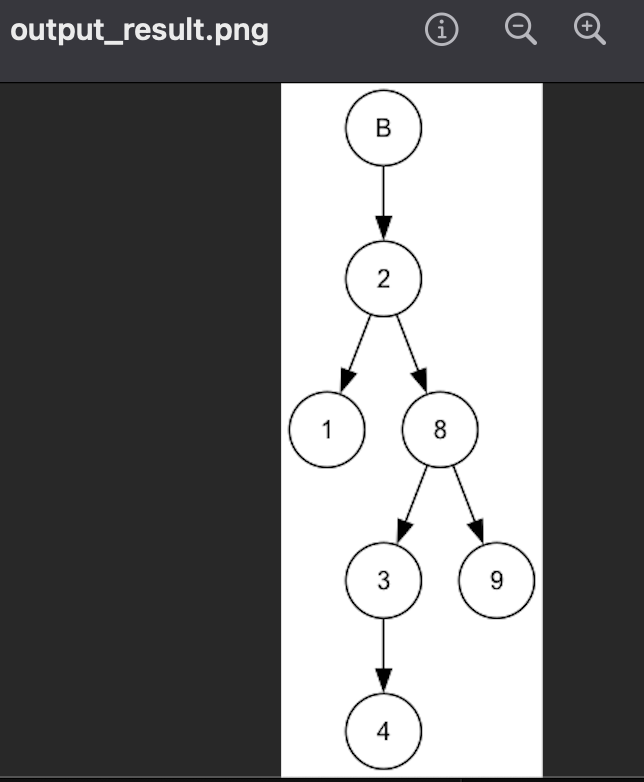
// Вход: указатель на файл, имя для дерева, указатель на корень,

void tree\_export\_to\_dot(FILE \*f, const char \*tree\_name, tree\_node\_t \*tree);

**Пример работы**

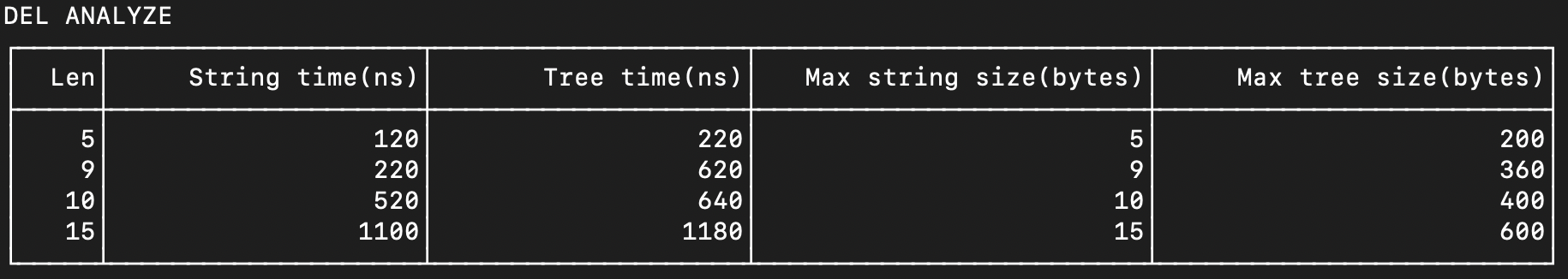


****

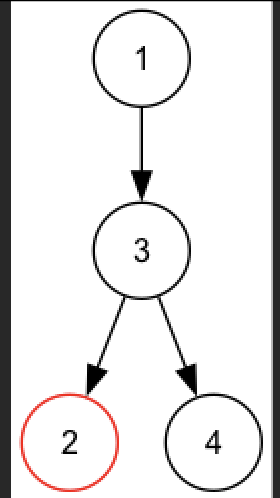
****

**Замеры**

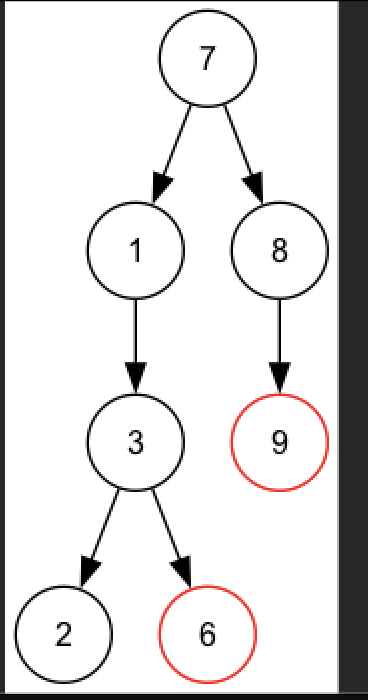
Производится 50 итераций при замерах.



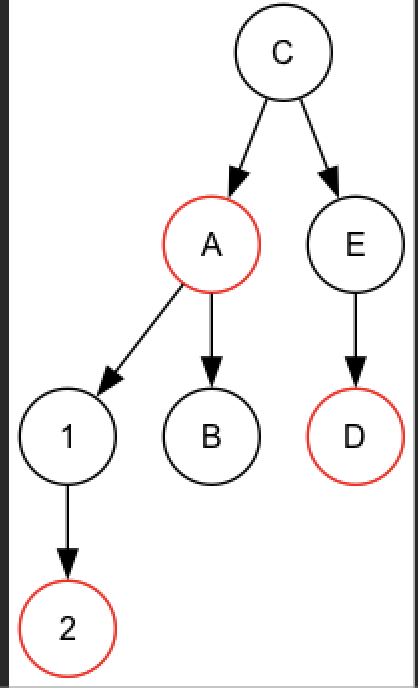
Дерево первого теста

****

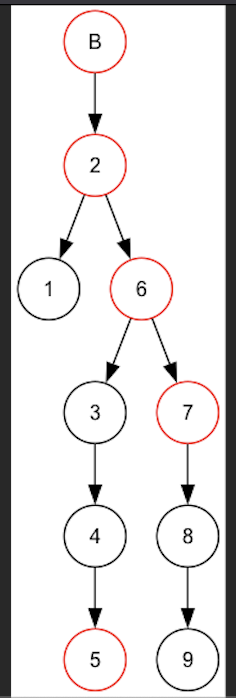
Дерево второго теста

****

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