|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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»

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6**

**«ОБРАБОТКА ДЕРЕВЬЕВ»**

**по курсу «Типы и структуры данных»**

**Вариант 5**

Студент: Писаренко Дмитрий Павлович

Группа: ИУ7-34Б

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Писаренко Д.П.

*подпись, дата фамилия, и.о.*

Преподаватель \_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Рыбкин Ю.А.

*подпись, дата фамилия, и.о.*

# Цель работы

Цель работы: получить навыки применения двоичных деревьев, реализовать основные операции над деревьями: обход деревьев, включение, исключение и поиск узлов.

# Условие задачи

Построить словарь из слов текстового файла в виде дерева двоичного поиска. Вывести его на экран в виде дерева. Осуществить поиск указанного слова в дереве и в файле. Если слова нет, то (по желанию пользователя) добавить его в дерево и, соответственно, в файл. Сравнить время поиска слова в дереве и в файле.

# Техническое задание

## Исходные данные

*Выбор действия*: целое число от 0 до 5.

*0. Выход из программы*

Не требует ввода от пользователя.

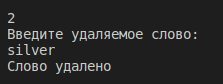
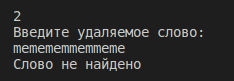
*1. Инициализировать дерево из файла*

Данные читаются из указанного файла в бинарное дерево. При успешном чтении пользователю сообщается о нем, иначе – программа завершается с ненулевым кодом возврата.



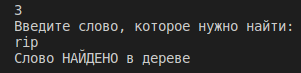
*2. Исключить узел*

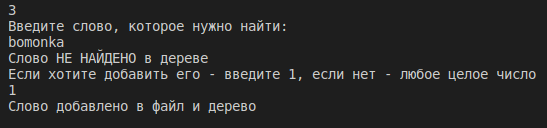
Пользователь вводит слово, которое хочет удалить. Если такое слово найдено – слово удаляется и на экран выводится “Слово удалено”, в ином случае – на экран выводится “Слово не найдено”.



*3. Найти указанное слово*

Пользователь вводит слово, которое ищется в дереве. При нахождении на экран выводится “Слово НАЙДЕНО в дереве”, в ином случае на экран выводится “Слово НЕ НАЙДЕНО в дереве” и пользователю предлагается добавить его с помощью ввода “1”, после добавления на экран выводится “Слово добавлено в файл и дерево”.

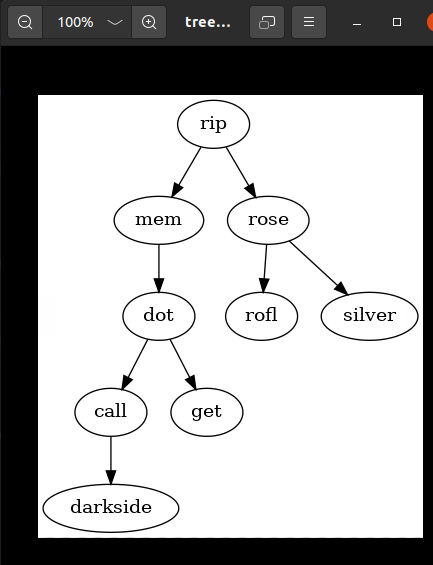




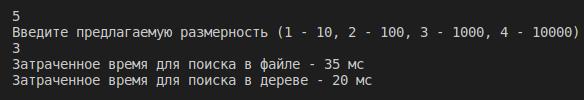
*4. Вывести дерево*

При помощи скрипта, языка описания графов DOT и программы Graphviz получаем выведенное дерево “tree.png”.

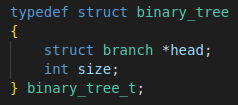
|  |
| --- |
| *#!/bin/sh*  dot -Tpng tree.gv -o tree.png  gio open tree.png |

Скрипт print\_tree.sh

*5. Сравнить время поиска в дереве и файле*

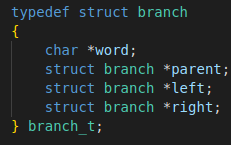
 Пользователю предлагается ввести размерность дерева (10/100/1000/10000 слов). После этого для данного количества слов сравнивается время поиска в дереве и файле.

## Структуры данных



|  |
| --- |
| struct branch \*head – указатель на корень дерева  int size – количество элементов |

Описание полей структуры binary\_tree\_t



|  |
| --- |
| char \*word - слово  struct branch \*parent – родительский узел вершины  struct branch \*left – левый узел вершины  struct branch \*right – правый узел вершины |

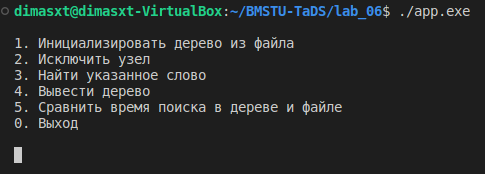
# Способ обращения к программе

Работа с программой осуществляется с помощью консоли.

Сборка осуществляется c помощью команды **make release**

Запуск выполняется с помощью команды **./app.exe**

Дальнейшая работа производится с помощью меню:



# Тестирование

## Позитивные тесты

|  |  |  |  |
| --- | --- | --- | --- |
| # | Входные данные | Выходные данные | Результат |
| 1 | Ключ = 1  Ключ = 3  Слово, которое нужно найти: rip | Сообщение:  Слово НАЙДЕНО в дереве | Ожидание следующего ключа |
| 2 | Ключ = 1  Ключ = 3  Слово, которое нужно найти: aaaa  1 | Сообщение:  Слово НЕ НАЙДЕНО в дереве  …  Слово добавлено в файл и дерево | Ожидание следующего ключа |
| 3 | Ключ = 1  Ключ = 2  Удаляемое слово: aaaa | Сообщение:  Слово не найдено | Ожидание следующего ключа |
| 4 | Ключ = 1  Ключ = 2  Удаляемое слово: silver | Сообщение:  Слово удалено | Ожидание следующего ключа |
| 5 | Ключ = 1  Ключ = 4 | Дерево в файле .png | Ожидание следующего ключа |
| 6 | Ключ = 5  Размерность: 3 | Время для поиска в дереве и файле | Ожидание следующего ключа |
| 7 | Ключ = 0 | Отсутствуют | Завершение программы |

## Негативные тесты

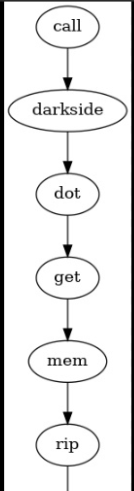
|  |  |  |  |
| --- | --- | --- | --- |
| # | Входные данные | Выходные данные | Результат |
| 1 | Ключ = 123 | Сообщение:  Номер меню - целое число от 0 до 5 | Код ошибки 2 |
| 2 | Ключ = 5  Ключ = 123 | Сообщение:  Размерность - число от 1 до 4 | Код ошибки 8 |
| 3 | Ключ = 1  Некорректное название файла | Сообщение:  Файл не найден | Код ошибки 5 |
| 4 | Ключ = dfwfd | Сообщение:  Номер меню - целое число | Код ошибки 1 |

# Таблица с результатами измерения времени и памяти

Время замерялось при 1000 выполнениях функций.

Время в таблице указано в мс.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Сбалансированные деревья** | | | **Максимальная глубина** | | |
| **Тип** | **Размер** | **Время** | **Тип** | **Размер** | **Время** |
| Файл | 10 | 2 | Файл | 10 | 2 |
| Дерево | 1 | Дерево | 0 |
| Файл | 100 | 3 | Файл | 100 | 6 |
| Дерево | 1 | Дерево | 1 |
| Файл | 1.000 | 26 | Файл | 1.000 | 73 |
| Дерево | 1 | Дерево | 1 |
| Файл | 10.000 | 287 | Файл | 10.000 | 613 |
| Дерево | 12 | Дерево | 6 |

В первом случае слово находилось в середине дерева и файла, во втором случае – в конце. Дерево второго случая выглядит примерно так:

# Контрольные вопросы

*1. Что такое дерево?*

Дерево – нелинейная структура данных, которая используется для представления иерархических связей «один ко многим». Дерево с базовым типом Т определяется рекурсивно: это либо пустая структура (пустое дерево), либо узел типа Т с конечным числом древовидных структур того же типа – поддеревьев.

*2. Как выделяется память под представление деревьев?*

Выделение памяти под деревья определяется типом их представления. Это может быть таблица связей с предками (№ вершины - № родителя), или связный список сыновей. Оба представления можно реализовать как с помощью матрицы, так и с помощью списков. При динамическом представлении деревьев (когда элементы можно удалять и добавлять) целесообразнее использовать списки – т.е. выделять память под каждый элемент динамически.

*3. Какие бывают типы деревьев?*

N-арное дерево, сбалансированное дерево, бинарное дерево, бинарное дерево поиска, дерево AVL, красное-черное дерево, 2-3 дерево.

*4. Какие стандартные операции возможны над деревьями?*

Обход, поиск, добавление и удаление элемента.

*5. Что такое дерево двоичного поиска?*

Дерево двоичного поиска – дерево, в котором все левые потомки «моложе» предка, а все правые – «старше».

# Вывод

В ходе выполнения лабораторной работы была освоена обработка деревьев. Экспериментальным путем доказано, что различные операции в дереве выполняются быстрее, чем в файле, независимо от количества элементов и позиции слова.