|  |  |
| --- | --- |
| 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\_\_\_\_

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6**

**«Обработка деревьев, работа с хеш-таблицами»**

Студент\_\_\_\_\_\_Чыонг Ван Хао\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*фамилия, имя, отчество*

Группа\_\_\_\_\_*ИУ7И-31Б\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

Вариант 1

Принял : Силантьева А. В.

**Описание условия задачи**

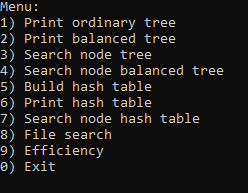
**Техническое задание**

Построить ДДП, в вершинах которого находятся слова из текстового файла. Вывести его на экран в виде дерева. Сбалансировать полученное дерево и вывести его на экран. Построить хеш-таблицу из слов текстового файла. Использовать метод цепочек для устранения коллизий. Осуществить поиск введенного слова в ДДП, в сбалансированном дереве, в хеш-таблице и в файле. Сравнить время поиска, объем памяти и количество сравнений при использовании различных (4-х) структур данных. Если количество сравнений вхеш-таблице больше указанного (вводить), то произвести реструктуризацию таблицы, выбрав другую функцию.

**Входные данные**

Файл со словами.

Пункты меню:



1. Вывести ДДП
2. Вывести АВЛ-дерево
3. Поиск слова в ДДП
4. Поиск слова в АВЛ-дереве
5. Построить хэш-таблицу
6. Вывести хэш-таблицу
7. Поиск слова в хэш-таблице
8. Поиск слова в файле
9. Эффективность программы
10. Выход

**Выходные данные**

Двоичное дерево, сбалансированное дерево, хеш-таблица, оценка эффективности добавления и поиска элемента (для деревьев, хеш-таблицы и файла), среднее кол-во сравнений.

**Функция программы**

Работа с деревом слов — создание и вывод дерева двоичного поиска, создание и  
вывод сбалансированного дерева, построение, вывод и реструктуризация (при  
необходимости) хеш-таблицы. Поиск слова в двоичном дереве поиска слов, в  
сбалансированном дереве слов, в хеш-таблице и в файле.

**Обращение к программе**

Программа запускается из терминала в директории с проектом при помощи  
команды «./app.exe».

**Возможные аварийные ситуации и ошибки пользователя**

1. команда введена неправильно.

На выходе сообщение: «Wrong choice!.»

2. ввод неверного размера таблицы.

На выходе сообщение: « Error input!!»

3. распечатать пустую таблицу.

На выходе сообщение: « Empty table »

**Внутренняя структура данных**

Для реализации очереди при помощи массива используется структура:

typedef struct tree\_node node\_t;

struct tree\_node

{

    char \*word;

    int balance;

    struct tree\_node \*left;

    struct tree\_node \*right;

};

Её поля:

*char \*word* — значение текущей вершины;  
*int balance* — высота вершины относительно других вершин;  
*struct tree\_t \*left* — указатель на левого потомка;  
*struct tree\_t \*right* — указатель на правого потомка;

typedef struct node\_table node\_hash;

struct node\_table

{

    char \*word;

    struct node\_table \*next;

};

Её поля:

*char \* word* — значение текущего элемента списка

*struct node\_table \*next* — указатель на следующий элемент списка;

**Функции программы**

void menu(void);

Описание: параметры печати на экране.

struct tree\_node\* create\_node(char \*word);

Описание: создать узел для дерева

node\_t\* create\_tree(node\_t \*root, FILE \*f);

Описание: созданное ДДП.

node\_t\* create\_bal\_tree(node\_t \*root, FILE \*f);

Описание: созданное АВЛ-дерево.

node\_t \*search(char \*rem, node\_t \*tree, int \*done, int \*flag);

Описание: сделай поиск в дереве.

node\_hash\* add\_end(node\_hash \*head, node\_hash \*elem);

Описание: добавить узел в хеш.

node\_hash\* create\_hash\_node(char \*word);

Описание: создать хеш-узел.

void print\_table(node\_hash \*\*table, int len\_table);

Описание: печать на экран хеш-таблицы.

void search\_in\_hash\_table(char \*word, node\_hash \*\*table, int len\_table, int \*done);

Описание: выполнить поиск в хеш-таблице.

int search\_in\_file(FILE \*f, char \*s);

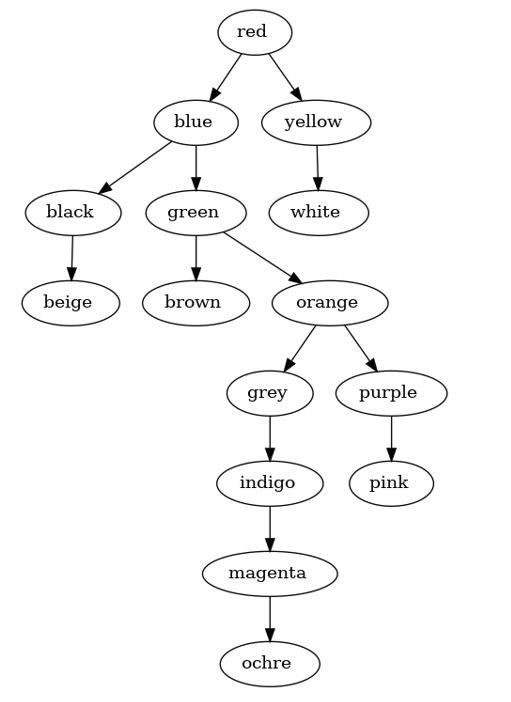
Описание: выполните поиск в файле.

**Тесты**

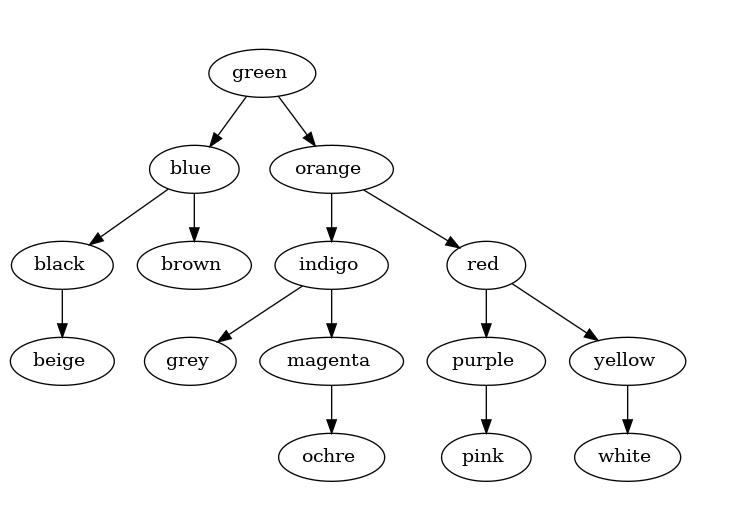
Некорректно введенные параметры



Вывод ДДП:

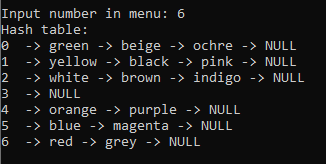


Вывод АВЛ-деревьев:



Вывод хеш-таблицы:



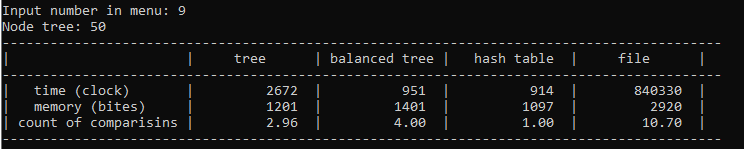


Поиск слова:





**Оценка эффективности:**



Для оценки эффективности используется файл с 50 записями.

Время работы: поиск слова в хэш-таблице работает на много раз быстрее чем поиск в файле; на 3 раз быстрее, чем в ДДП; на 4% быстрее, чем в АВЛ-дереве  
Использование памяти: хэш-таблица затрачивает меньше памяти, чем файл на 60%; чем ДДП на 9%; чем АВЛ-дерево на 22%

Количество сравнений: меньше всего сравнений для поиска требует хэш-таблица.

**Выводы**

Основным преимуществом деревьев является высокая эффективность реализации основанных на нём алгоритмов поиска и сортировки.

Хэш-таблицы используют меньше памяти и требуют минимального количества операций сравнения при поиске. Таблицы также требуют качественной хэш-функции для избегания колизий. Для того, чтобы в хеш-таблице быстрее осуществлялся поиск, должно быть как можно меньше коллизий. Количество коллизий зависит от того, сколько было выделено памяти, следовательно, для большей эффективности по времени придется выделить больше памяти.

**Контрольные вопросы**

**1. Что такое дерево?**Дерево – это нелинейная структура данных, используемая для представления  
иерархических связей, имеющих отношение «один ко многим».  
**2. Как выделяется память под представление деревьев?**Дерево реализуется при помощи односвязного списка, поэтому память  
выделяется для каждого узла отдельно.  
**3. Какие стандартные операции возможны над деревьями?**Обход дерева, поиск по дереву, включение в дерево, исключение из дерева.  
**4. Что такое дерево двоичного поиска?**Дерево двоичного поиска – это такое дерево, в котором все левые потомки  
моложе предка (меньше, либо равны), а все правые – старше (больше, либо  
равны). Это свойство называется характеристическим свойством дерева  
двоичного поиска и выполняется для любого узла, включая корень.  
**5. Чем отличается идеально сбалансированное дерево от АВЛ-дерева?**У идеально сбалансированного дерева число вершин в левом и правом  
поддеревьях отличается не более, чем на единицу. У каждого узла АВЛ-дерева  
высота двух поддеревьев отличается не более, чем на единицу.  
**6. Чем отличается поиск в АВЛ-дереве от поиска в дереве двоичного поиска?**Поиск в АВЛ-дереве происходит быстрее, чем поиск в дереве двоичного поиска  
и с меньшим числом сравнений.  
**7. Что такое хеш-таблица, каков принцип ее построения?**Хеш-таблица - массив, заполненный в порядке, определенным хеш-функцией.  
Принцип построения: хеш-функция ставит в соответствие каждому ключу ki  
индекс ячейки j, где расположен элемент с этим ключом. Таким образом:  
h (ki) = j, если j=(1, m), где j принадлежит множеству от 1 до m, а m. – размерность массива.  
**8. Что такое коллизии? Каковы методы их устранения?**Коллизии - ситуации, когда разным ключам соответствует одно значение хешфункции, то есть, когда h(K1)=h(K2), в то время как K1 ≠ K2.  
Методы устранения:  
1) Внешнее (открытое) хеширование (метод цепочек). В случае, когда элемент  
таблицы с индексом, который вернула хеш-функция, уже занят, к нему  
присоединяется связный список. Таким образом, если для нескольких различных  
значений ключа возвращается одинаковое значение хеш-функции, то по этому  
адресу находится указатель на связанный список, который содержит все  
значения.  
2) Внутреннее (закрытое) хеширование (открытая адресация). В этом случае,  
если ячейка с вычисленным индексом занята, то можно просто просматривать следующие записи таблицы по порядку (с шагом 1), до тех пор, пока не будет найден ключ K или пустая позиция в таблице. При этом, если индекс следующего просматриваемого элемента определяется добавлением какого-то постоянного шага (от 1 до n), то данный способ разрешения коллизий называется линейной адресацией. Для вычисления шага можно также применить формулу: h = h + a 2 , где a – это номер попытки поиска ключа. Этот вид адресации называется квадратичной или произвольной адресацией.  
**9. В каком случае поиск в хеш-таблицах становится неэффективен?**Если для поиска элемента необходимо более 3–4 сравнений, то эффективность  
использования хеш-таблицы пропадает.  
**10. Эффективность поиска в АВЛ деревьях, в дереве двоичного поиска и в хештаблицах.**АВЛ-деревья : O(log2(n))  
ДДП : O(log2(n)) - O(n)  
Хеш-таблица: O(1)