**Лабораторная работа №6**

по курсу «Типы и структуры данных»

Обработка деревьев, работа с хеш-таблицами

Отчёт выполнила:

студент группы ИУ7-35Б

Лаврова Анастасия

**Цель работы**:

построить и обработать хеш-таблицы, сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска и в хеш-таблицах. Сравнить эффективность устранения коллизий при внешнем и внутренним хешировании.

**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

в хеш-таблице больше указанного, 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.

**Обращение к программе:**

Через консоль

**Описание задачи, реализуемой программой**

1. Построение ДДП из слов в файле.
2. Построение сбалансированного дерева.
3. Преобразование деревьев из формата .gv в .png и вывод на экран.
4. Построение хэш-таблицы.
5. Поиск введенного пользователем слова в ДДП, сбалансированном дереве, хеш-таблице и файле.
6. Сравнение времени поиска, объем памяти и количество сравнений при использовании различных структур данных.

**Внутренняя структура данных:**

Дерево:

**struct** tree\_node  
{  
 **char** \*word;  
 **int** balance;  
 **struct** tree\_node \*left;  
 **struct** tree\_node \*right;  
};

Хэш-таблица:  
**struct** node\_table  
{  
 **char** \*word;  
 **struct** node\_table \*next;  
};

**Функции**:

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

Вход: корень дерева, файл

Выход: созданное ДДП

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

Вход: корень дерева, файл

Выход: созданное АВЛ-дерево

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

Вход: файл, название дерева, дерево

Выход: сформированный файл gv

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

Вход: искомое слово, дерево, флаг о нахождении

Выход: измененное значение флага

node\_hash \*\*build\_table(FILE \*f, **int** \*len\_table);

Вход: файл, размер таблицы

Выход: сформированная хэш-таблица

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

Вход: хэш-таблица, размер таблицы

Выход: вывод на экран хэш-таблицы

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

Вход: искомое слово, хэш-таблица, размер таблицы

Выход: сообщение о нахождении слова

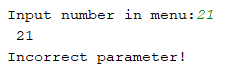
**int** search\_in\_file(FILE \*f, **char** \*s);

Вход: файл, искомое слово

Выход: сообщение о нахождении слова

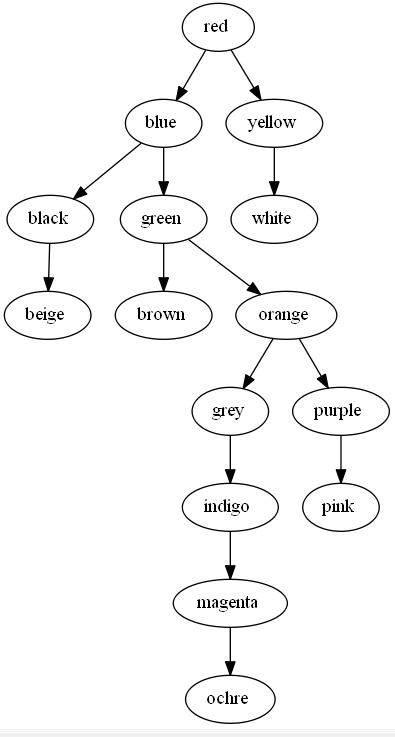
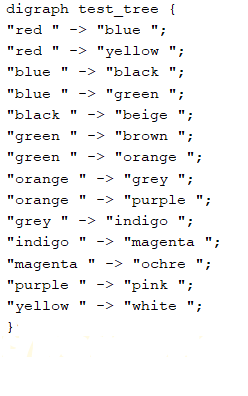
**Аварийные ситуации и ошибки пользователя:**

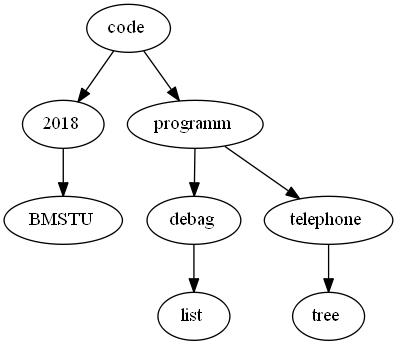
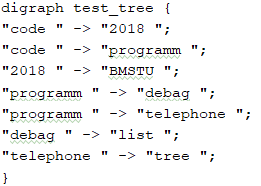
Некорректно введенные параметры



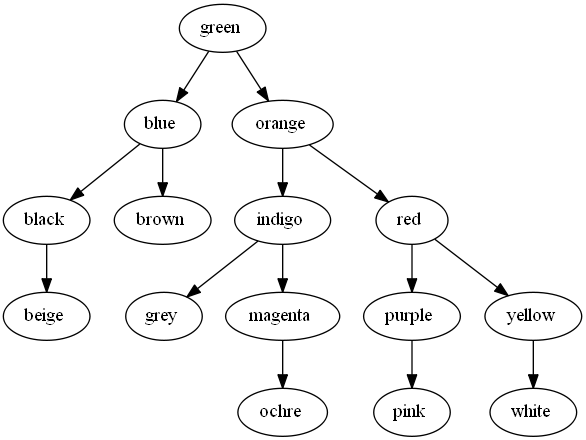
**Тесты:**

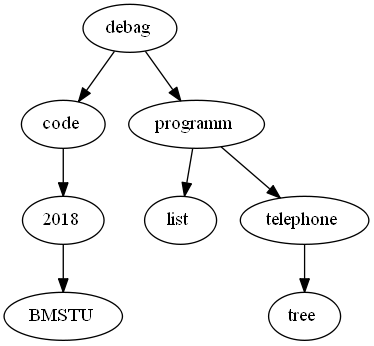
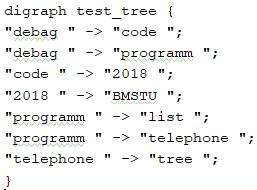
Вывод ДДП:

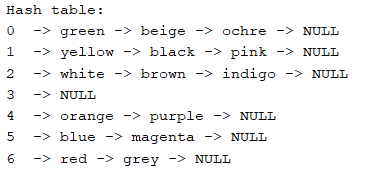
 

Вывод АВЛ-деревьев:

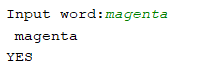
 

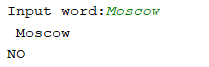
 

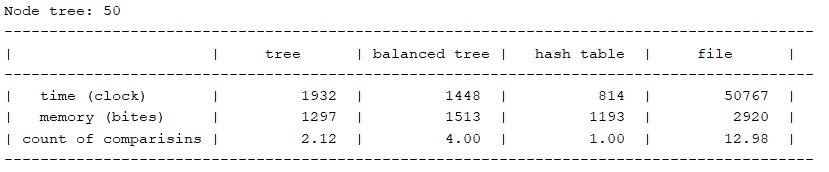
Вывод хеш-таблицы:



Поиск слова:





Эффективность:  


**Оценка эффективности:**

Для оценки эффективности используется файл с 50 записями.

Время работы: поиск слова в хэш-таблице работает на 98% быстрее чем поиск в файле; на 58% быстрее, чем в ДДП; на 44% 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.

**Вывод:**

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.

**Ответы на 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?

Выделение памяти под деревья определяется типом их представления. В памяти деревья можно представить в виде связей с предками (еще их называют родителями); связного списка потомков (сыновей) или структуры данных. Оба представления можно реализовать в виде матрицы или списка. При реализации списком память выделяется динамически, при реализации матрицей - статически.

3. Какие стандартные операции возможны над деревьями?

Обход дерева, поиск по дереву, включение и исключение элемента.

4. Что такое дерево двоичного поиска?

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

5. Чем отличается идеально сбалансированное дерево от АВЛ-дерева?

Дерево, у которого число вершин в левом и правом поддеревьях отличается не более, чем на единицу называется идеально сбалансированным. Двоичное дерево, у каждого узла которого высота двух поддеревьев отличается не более чем на единицу называется АВЛ-деревом.

6. Чем отличается поиск в АВЛ-дереве от поиска в дереве двоичного поиска?

Поиск в АВЛ-дереве имеет сложность О(log2n), а в ДДП - сложность О(n).

7. Что такое хеш-таблица, каков принцип ее построения?

Массив, заполненный в порядке, определенным хэш-функцией, называется хэш-таблицей. Хэш-функция – функция, которая ставит в соответствие каждому ключу индекс ячейки, где расположен элемент с этим ключом.

8. Что такое коллизии? Каковы методы их устранения.

Коллизия – ситуация, когда разным ключам соответствует одно значение хэш-функции. Существует несколько возможных вариантов разрешения коллизий: внешнее (открытое) хэширование (метод цепочек) и внутреннее (закрытое) хэширование (открытая адресация).

9. В каком случае поиск в хэш-таблицах становится неэффективен?

Поиск в хэш-таблице становится неэффективным при большом числе коллизий – сложность поиска возрастает.

При открытом хэшировании в случае, когда элемент таблицы с индексом, который вернула хэш-функция, уже занят, к нему присоединяется связный список. Тогда, если для нескольких различных значений ключа возвращается одинаковое значение хэш- функции, то по этому адресу находится указатель на связанный список, который содержит все значения. Поиск в этом списке осуществляется простым перебором, так как при грамотном выборе хеш-функции любой из списков оказывается достаточно коротким.

При закрытом хэшировании, если ячейка с вычисленным индексом занята, то можно просто просматривать следующие записи таблицы по порядку, до тех пор, пока не будет найден ключ или пустая позиция в таблице.

10. Эффективность поиска в АВЛ деревьях, в дереве двоичного поиска и в хеш-таблицах.

В хэш-таблице минимальное время поиска О(1). В АВЛ дереве О(log2n). В дереве двоичного поиска О(h), где h – высота дерева.