МГТУ им. Н. Э. Баумана

**Дисциплина типы и структуры данных**

**Лабораторный практикум №6**

**по теме: «Обработка деревьев, хеш-функций»**

**Вариант №3**

Работу выполнил:

студент группы ИУ7-35

Платонова Ольга

Работу проверил:

Москва, 2019 г.

***Цель работы***: получить навыки применения двоичных деревьев, реализовать основные операции над деревьями: обход деревьев, включение, исключение и поиск узлов; построить и обработать хеш-таблицы, сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска и в хеш-таблицах.

***Условие задачи***

Построить ДДП, сбалансированное двоичное дерево (АВЛ) и хеш-таблицу по указанным данным. Сравнить эффективность поиска в ДДП в АВЛ дереве и в хеш-таблице. Вывести на экран деревья и хеш-таблицу. Подсчитать среднее количество сравнений для поиска данных в указанных структурах. Произвести реструктуризацию хеш-таблицы, если среднее количество сравнений больше указанного. Оценить эффективность использования этих структур для поставленной задачи. Оценить эффективность поиска в хеш-таблице при различном количестве коллизий.

***Техническое задание***

1. Построить дерево, считав слова из файла.

2. Вывести дерево на экран.

3. Сбалансировать дерево.

4. Вывести сбалансированное дерево на экран.

5. Удалить указанное слово (принять с клавиатуры) в исходном и сбалансированном дереве.

6. Сравнить время удаления и объем памяти.

7. Построить хеш-таблицу, считав слова из файла и задав размер с клавиатуры.

8. Вывести таблицу на экран.

9. Удалить указанное слово (принять с клавиатуры) в таблице.

10. Сравнить время удаления, объем памяти и количество сравнений.

*Входные данные:*

Текстовый файл со словами.

Пользователь может самостоятельно выбрать реализацию:

Дерево:

Создать дерево — 1

Сбалансировать дерево — 2

Удалить элемент — 3

Хеш-функция:

Создать таблицу — 1 (с клавиатуры вводится размер таблицы в заданном диапазоне)

Удалить элемент — 2

Анализ

Программист должен рассмотреть ошибки, связанные с некорректным вводом, попыткой удалить отсутствующее слово, создание таблицы некорректного размера.

*Выходные данные*

ДДП и АВЛ-дерево, время и память, затраченные на удаление элемента.

Хеш-таблица, время, затраченное на удаление элемнта.

Время удаления элемента, память и количество сравнений.

*Структура данных*

*1. Дерево*

struct tree\_node  
 {  
 char \*element;  
 int height;  
 struct tree\_node \*left;  
 struct tree\_node \*right;  
 }

*2. Хеш-таблица*

*struct table\_node  
 {  
 char \*word;  
 struct table\_node \*next;  
 }*

***Описание алгоритма***

* Дерево

1. Создать узел.

struct tree\_node\* create\_node(char \*element)

2. Добавить созданный узел в дерево.

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

struct tree\_node\* insert(struct tree\_node \*tree, struct tree\_node \*node)

Если добавляемый элемент меньше текущего, то добавить в левый поддерево, иначе — в правое.

В случае пустого дерево — добавление в правое поддерево, так как нулевой указатель в левом.

3. Сбалансировать дерево.

node\_t\* balance(node\_t\* ptr)

Обход по дереву. Если текущая высота дерева больше двух и мы находимся в правом поддереве, то правый поворот дерева, иначе — левый.

Обход осуществляем в глубину, начиная с левого поддерева.

4. Удалить заданный элемент.

node\_t \*remove\_in\_tree(char \*rem\_element, node\_t \*tree, int \*done, int \*flag)

node\_t \*remove\_in\_bal\_tree(char \*rem\_element, node\_t \*tree, int \*done, int \*flag)

Осуществляется обход по дереву. Если не найдено заданного элемента, выдать сообщение на экран и завершить программу. Иначе — удалить узел, заменив его самым левым элементом в правом поддереве.

Выполнить проверку на пустоту дерево и на нулевых сыновей.

В случае сбалансированного дерева, необходимо после каждого удаления выполнять проверку на сбалансированность, и в случае необходимости, выполнять балансировку.

* Хеш-функция

1. Создать узел.

node\_hash\* create\_hash\_node(char \*element)

2. Добавить созданный узел в таблицу.

node\_hash \*\*create\_hash\_table(char \*file, int \*len, int stop\_len\_list)

void insert\_node(node\_hash \*\*head, node\_hash \*node)

Хеш-функция определяется как остаток от деления количества символов в строке на текущий размер таблицы.

sum += elemnt[i];

sum = sum % tbl\_len;

3. Удалить заданный элемент.

void remove\_in\_hash\_table(char \*del\_element, node\_hash \*\*table, int len\_table, int \*done)

Проход по всему списку, пока не найден заданный элемент. В случае совпадения, предыдущему элементу присвоить следующий и освободить память текущего элемента.

***Замечание***

Отметим, что оптимальным размером хеш-таблицы является число, которое примерно в 0.8 раз превосходит количество элементов и является простым. Именно при таком задании хеш-функция будет хорошей, то есть число коллизий минимально.

В моем примере на 20 строк — 17.

***Тесты***

1. Попытка считать информацию из пустого файла.

2. Попытка считать информацию из файла, который не задан.

3. Попытка удалить слова, которое отсутствует в файле.

4. Задание размера хеш-таблицы вне интервала допустимых значений.

***Вопросы***

*1. Что такое дерево?*

Дерево – это нелинейная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим». Состоит из узлов, которые хранят значение и ссылки на правую и левую ветвь. Каждая ветвь является деревом.

*2. Как выделяется память под представление деревьев?*

Выделение памяти под деревья зависит от их реализации. Это может быть как связный список сыновей, так и таблица связей. Оба варианта возможно реализовать как на массиве, так и на списке.

*3. Какие стандартные операции возможны над деревьями?*

Обход, поиск, удаление, добавление элементов.

*4. Что такое дерево двоичного поиска?*

Дерево двоичного поиска – это дерево, в котором все левые потомки меньше предка, а все правые – больше. Это свойство называется характеристическим свойством дерева двоичного поиска и выполняется для любого узла, включая корень.

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

В идеально сбалансированном дереве для каждой его вершины количество вершин в правом и левом поддереве различается не более, чем на 1. В АВЛ-дереве для каждой его вершине высота его двух поддеревьев различается не более, чем на 1.

АВЛ-дерево является идеально сбалансированным деревом по высоте.

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

Алгоритм поиска деревьев идентичен.

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

Хеш-таблица представляет из себя массив, заполненный в порядке, определенным хеш-функцией.

*8. Что такое коллизии? Каковы методы их устранения.*

*Коллизией называется ситуация, при которой разным ключам соотвествует одно ит тоже значение хеш-функции, то есть h(K1) = h(K2), в то время как K1 ≠ K2.*

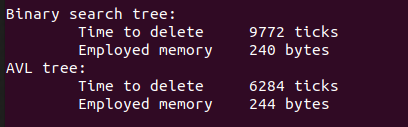
Сущетвует несколько способов разрешения коллизий. Это метод цепочек(длина одной цепочки больше единицы) и открытая адресация.

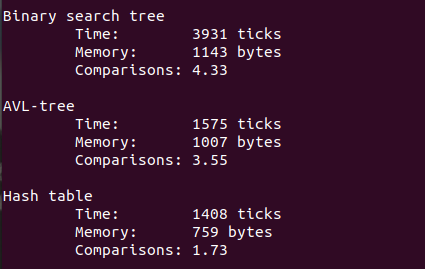
*9. В каком случае поиск в хеш-таблицах становится неэффективен?*

При большом количестве коллизий.

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

*таблицах.*





***Вывод***

На примере нашей задачи мы убедились, что реализация хеш-таблицы является эффективной как по времени, так и по памяти.

Что касается деревьев, поиск и удаление элементов эффективней в АВЛ-дереве.