**МГТУ им. Н.Э.Баумана**

**Дисциплина “Типы и структуры данных”**

**Отчет по Лабораторной работе №6 по теме “** **Деревья,**

**хеш –таблицы** **”.**

Работу выполнил:

Студент: Зейналов Зейнал

ИУ7-31Б

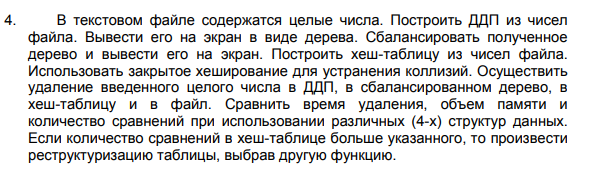
Варинат 14(4)

Москва, 2018.

**Работа № 6**

**Цель работы – построить дерево, вывести его на экран в виде дерева, реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов, сбалансировать дерево, сравнить эффективность алгоритмов сортировки и поиска в зависимости от высоты деревьев и степени их ветвления; построить хеш-таблицу и вывести ее на экран, устранить коллизии, если они достигли указанного предела, выбрав другую хеш-функцию и реструктуризировав таблицу; сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска (ДДП) и в хеш-таблицах. Сравнить эффективность реструктуризации таблицы для устранения коллизий с эффективностью поиска в исходной таблице.**

Описание условия задачи



Описание ТЗ

Общая концепция системы

**Программа выполняет операции над ДДП в 4 структурах (Файл, сбалансированное дерево, не сбалансированное дерево, хеш-таблица).**

**Пользователь может удалить число из структур, введя само это число и минимальное количество сравнений. В случае необходимости, будет произведена реструктуризация хеш-таблицы(изменение хеш-функции).**

### Требования к функциональным характеристикам

При разработке интерфейса программы следует предусмотреть:

• указание типа, формата и диапазона вводимых данных,

• указание действий, производимых программой,

• наличие пояснений при выводе результата,

• вывод дерева осуществить в графическом виде (или предложить иную визуализацию в виде дерева)

• вывод на экран хеш-таблицы;

• возможность изменения хеш-функции при необходимости реструктуризации таблицы;

• вывод времени и количества сравнений при поиске одних и тех же данных в различных структурах данных.

При тестировании программы необходимо:

o проверить правильность ввода и вывода данных (т.е. их соответствие требуемому типу и формату). Обеспечить адекватную реакцию программы на неверный ввод данных;

o обеспечить вывод сообщений при отсутствии входных данных («пустой ввод»);

o проверить правильность выполнения операций; o обеспечить возможность добавления узла в пустое дерево;

o предусмотреть вывод сообщения при попытке удаления узла из пустого дерева;

o проверить различные варианты включения и исключения узла в существующее дерево;

o проверить поиск существующего узла и поиск несуществующего узла в дереве и хеш-таблице;

o создать левостороннее или правостороннее дерево, проверить время обработки узла в нем;

o предусмотреть вывод сообщения при необходимости реструктуризации хеш-таблицы.

На **вход** программа получает название текстового файла, в котором хранится граф.

Затем число, которое необходимо удалить и минимально допустимое количество сравнений.

**Выход** должен быть представлен в виде набора данных, представляющих собой 4 структуры.

Также программа должна вывести данные о времени выполнения алгоритмов.

Программа должна выдавать корректный ответ при вводе любых данных. Если произошла ошибка ввода, программа должна сообщить об этом.

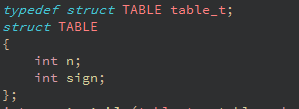
## Способ обращения к программе

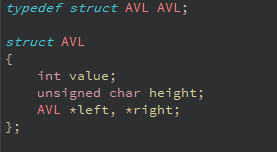
Программа представляет собой файл app.exe. Запускается в консоли. Для запуска достаточно команды ./app.exe. Если файл отсутствует можно собрать его с помощью утилиты make app.exe .

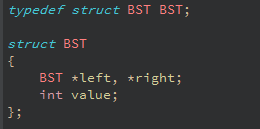
Аварийные ситуации:

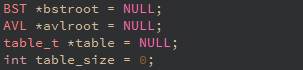
* Ввод несуществующего в данной директории файла  
  Программа выведет информацию о неверном вводе названия файла
* Введение неверного минимального количества сравнений   
  Программа выведет информацию об ошибке ввода

Описание структур данных









 **Элемент который ищем, максимальное кол-во сравнений.**



Описание алгоритма

Добавление элемента:

BST:

Пока не пустая вершина

Если элемент равен вершине

выйти из цикла

Если элемент больше вершины

рассмотреть правую вершину

Иначе

рассмотреть левую вершину

Вставить элемент на пустое место

AVL:

Пока не пустая вершина

Если элемент равен вершине

выйти из цикла

Если элемент больше вершины

рассмотреть правую вершину

Иначе

рассмотреть левую вершину

Вставить элемент на пустое место

Провести балансировку дерева

Хеш-функция:

Найти индекс элемента

Если ячейка пуста, вставить элемент  
 Иначе   
 пройти по массиву вправо в поиске пустой ячейки и вставить

Если конец массива

Начать с начала массива

Удаление элемента:

ДДП:

Найти элемент и его предка

Если элемент последний

Удалить элемент

Если у элемента один потомок

Заменить элемент на потомка

Если у элемента два потомка

Найти наименьший элемент в правом поддереве

Заменить элемент на найденный

Рекурсивно удалить найденный элемент

AVL:

Найти элемент и его предка

Если элемент последний

Удалить элемент

Если у элемента один потомок

Заменить элемент на потомка

Если у элемента два потомка

Найти наименьший элемент в правом поддереве

Заменить элемент на найденный

Рекурсивно удалить найденный элемент

Провести балансировку дерева

Хеш-функция:

Найти индекс элемента

Найти элемент в списке

Удалить элемент из списка

Балансировка:

Если разность высот правого и левого поддеревьев стала равна 2

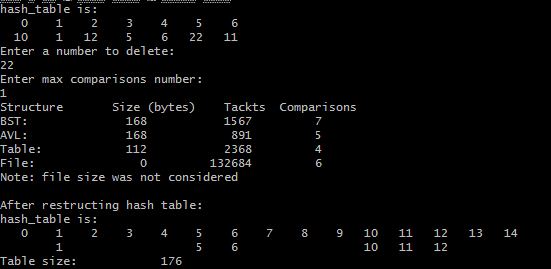
Повернуть влево

Если разность высот правого и левого стала равна -2

Повернуть вправо

Входные данные

В качестве входных данных участвует файл содержащий вершины графа, элемент который необходимо удалить и максимальное кол-во сравнений для хеш-таблицы.



Тестовые данные

1. Ввод несуществующего файла  
     
   Программа выведет информацию о том, что такого файла не существует
2. Некорректный ввод ( maxcmp <= 0)  
   Программа выведет информацию о том, что количество сравнений не может быть меньше либо равным 0
3. Введен не существующий элемент в дереве  
     
   Программа выведет информацию о том, что такого элемента не существует.
4. Пусть дерево представлено в виде:

1->2

1->3

3->4

1. Удаление “листа”

Вход: 4

Вывод:

1->2

1->3

Удаление узла с одним предком

Вход: 3

Выход:   
 1->2

1->4

Удаление узла с двумя предками

Вход: 1

Выход:

2->3

3->4

Оценка эффективности

|  |  |  |  |
| --- | --- | --- | --- |
| Удаление элемента  BST-дерево  t, такты | Удаление элемента  AVL-дерево  t, такты | Удаление элемента  Хеш-таблица  t, такты | Удаление элемента  Файл |
| 2970 | 6530 | 4600 | - |
| Память  BST-дерево | Память  AVL-дерево | Память  Хеш-таблица | Память  файл |
| 1344 | 1344 | 560 | - |
| Среднее количество сравнений  BST-дерево | Среднее количество сравнений  АВЛ-дерево | Среднее количество сравнений  Хеш-таблица | Среднее количество сравнений  Файл |
| 4.6 | 4.1 | 2.6 | 13.5 |

Выводы по проделанной работе

В ходе выполнения работы была реализована программа, работающая с двумя типами деревьев и хеш-таблицами. Были реализованы функции добавления, удаления и поиска для деревьев и хеш-таблицы.

Удаление элемента в хеш-таблице происходит на 80% быстрее, чем в обычном дереве, и на 35% быстрее, чем в АВЛ-дереве.

ДДП и АВЛ-дерево требуют примерно одинаковое количество памяти. а Хеш-таблица требует на 50% меньше памяти.

Эффективность хеш-таблицы зависит от среднего количества допустимых коллизий. При большом значении этого показателя, хеш-таблицы становятся не эффективными, увеличивается время удаления и среднее количество необходимых сравнений.

Таким образом, представление данных в виде хеш-таблицы эффективнее, чем в виде деревьев, а АВЛ-дерево эффективнее, чем дерево двоичного поиска, так как оно сбалансированное и требует меньшее количество сравнений.

# 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?

Память под деревья выделяется динамически. Память выделяется под элемент и два указателя на правое и левое поддерево.

3. Какие стандартные операции возможны над деревьями?

Обход, поиск, добавление и удаление элемента.

4. Что такое дерево двоичного поиска?

Дерево двоичного поиска - это бинарное дерево(у каждого элемента не больше двух потомков), в котором все левые потомки меньше предка, а все правые больше.

5. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

В идеально сбалансированном дереве количество элементов в правом и левом поддеревьях отличается не больше, чем на единицу. В АВЛ дереве, не больше, чем на единицу отличается высота дерева.

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

Поиск в АВЛ дереве гораздо быстрее посика в ДДП, так как зачастую приходится применять меньше операций сравнения, чтобы найти элемент.

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

Массив, заполненный в порядке, определенным хеш-функцией, называется хеш-таблицей.

8. Что такое коллизии? Каковы методы их устранения.

Коллизия - ситуация, когда разным ключам соответствует одно значение

хеш-функции, то есть, когда h(K1)=h(K2), в то время как K1 ≠ K2.

Методы устранения: внешнее(метод цепочек) и внутреннее(открытая адресация) хеширование.

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

Когда среднее количество коллизий становится больше 3-4, поиск в хеш-таблицах становится неэффективен.

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

таблицах

Эффективность хеш-таблицы зависит от среднего количества допустимых коллизий. При большом значении этого показателя, хеш-таблицы становятся не эффективными, увеличивается время удаления и среднее количество необходимых сравнений.

Авл-деревья эффективнее, чем деревья двоичного поиска, так как требуется меньше времени и сравнений, чтобы найти каждый элемент.