

ФАКУЛЬТЕТ Информатика и системы управления \_ КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

**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)**

ТИПЫ И СТРУКТУРЫ ДАННЫХ

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6

“Деревья, хэш-таблицы”

ВАРИАНТ 0

Студент\_\_

Гурова Наталия Алексеевна

*фамилия, имя, отчество*

Группа ИУ7-34Б

Выполнил

Гурова Н.А.

*подпись, дата фамилия, и.о.*

Принял

Силантьева А.В.

*подпись, дата фамилия, и.о.*

*2021 г.*

**Цель работы**

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), в хеш-таблицах и в файлах. Сравнить эффективность реструктуризации таблицы для устранения коллизий и поиска в ней с эффективностью поиска в исходной таблице.

**Задание**

В текстовом файле содержатся целые числа. Построить ДДП из чисел файла. Вывести его на экран в виде дерева. Сбалансировать полученное дерево и вывести его на экран. Построить хеш-таблицу из чисел файла. Использовать метод цепочек для устранения коллизий. Осуществить поиск введенного целого числа 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 (вводить), то произвести реструктуризацию таблицы, выбрав другую функцию.

**Входные данные**

* Для пунктов меню 0, 1, 2, 3, 4, 5, 6 вводится данный пункт в виде целого
* Для пункта 2 вводится также целое число, которое нужно вставить в дерево и хэш таблицу
* Для пункта 3 вводится также целое число, которое требуется найти
* Для пункта 5 вводится также целое число, обозначающее максимальное среднее количество сравнений для хеш-таблицы
* Для пункта 6 вводятся также параметры исследования памяти и эффективности (количество измерений, количество элементов для каждого измерения)

**Выходные данные**

* Результат выполнения определенной команды:
  1. Сообщение о том, что данные успешно считаны из файла, либо о том, что в процессе чтения произошла ошибка
  2. Сообщение о том, что вставка элемента произошла успешно, либо о том, что в процессе произошла ошибка
  3. Интерфейс ввода числа, информация об успешности его нахождения, в также количество сравнений, произведенных в каждой структуре данных.
  4. Двоичное дерево, abl дерево и хэш-таблица
  5. Информация о текущем среднем количестве сравнений. Интерфейс ввода числа, информация об успешности реструктуризации таблицы
  6. Отчет о количестве запрашиваемой памяти, эффективности и количестве сравнений.

**Возможности программы**

0 – выход из программы

1 - Загрузить данные из файла

2 – Выставить элемент

3 – Найти элемент

4 – Печать

5 – Реструктуризация таблицы

6 – Проведение исследоования

**Способ обращения к программе** Программа может быть вызвана через консоль с помощью команды app.exe

**Аварийные ситуации**

* Некорректный ввод номера команды (введено не число)
* Некорректные данные в файле
* Некорректный ввод нового количества сравнений
* Некорректный ввод нового элемента (повтор)

**Описание алгоритма**

* 1. Выводится меню данной программы.
  2. Пользователь вводит номер команды из предложенного меню.
  3. Пока пользователь не введет 0 (выход из программы), ему будет предложено вводить номера команд и выполнять действия по выбору.
  4. Для реструктуризации хеш-таблицы применяется инкрементация хеш-константы в цикле до тех пор, пока среднее количество сравнений не станет меньше указанного предела.

**Структуры данных**

// Обычное дерево

typedef int tree\_data\_t;  
  
typedef struct tree\_node tree\_node\_t;  
struct tree\_node  
{  
 tree\_data\_t data; // данные узла  
 struct tree\_node \*left; // левое поддерево  
 struct tree\_node \*right; // правое поддерево  
};

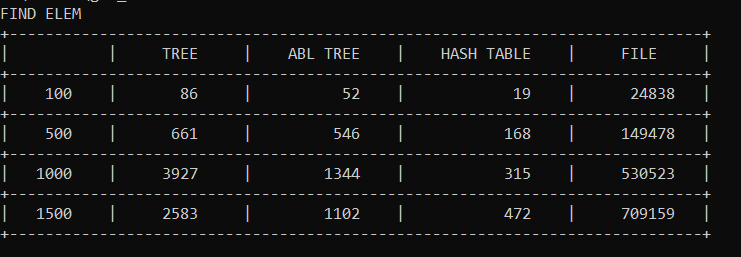
// ABL-дерево

typedef struct abl\_node abl\_node\_t;  
struct abl\_node  
{  
 int data; // данные узла  
 int height; // высота узла   
 abl\_node\_t\* left; // левое поддерево  
 abl\_node\_t\* right; // правое поддерево  
};

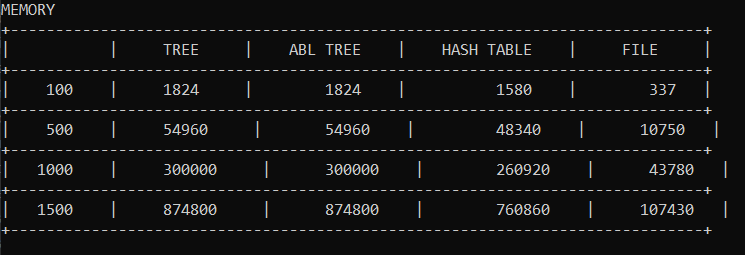
//элемент хеш-таблицы  
typedef int node\_data\_t;  
typedef struct node\_t  
{  
 struct node\_t \*next; // указатель на следующий элемент  
 bool is\_data\_fill; // флаг, заполнена ли ячейка  
 node\_data\_t data; // данные  
} node\_t;

//структура хеш-таблицы  
typedef node\_t\* list\_node\_t;  
typedef struct  
{  
 int count\_indexes; // количество индексов  
 int count\_collision; // количество коллизий  
 list\_node\_t \*array; // массив данных  
 int (\*hash\_func)(int n, ...); // хэш-функция (n – кол-во параметров)  
} hash\_table\_t;

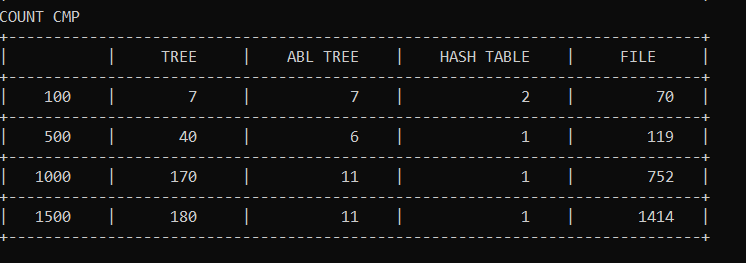
**Оценка эффективности**

Время поиска элемента (в тактах процессора, 5000 итераций):

Память (в байтах):



Количество сравнений:



**Вывод**

Основным преимуществом файла является то, что файл занимает меньше памяти (примерно в 5 раз).

Основным преимуществом хэш-таблицы является то, что поиск элемента в ней занимает меньше времени (если хэш-функция подобрана хорошо) (в 3 и более раза).

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

Между сбалансированным и несбалансированным лучше выбирать сбалансированное дерево. Хоть это и увеличит сложность обработки, но даст возможность сделать поиск более эффективным за счет уменьшения высоты дерева и, следовательно, количества сравнений. (в 6 и более раз)

Из приведенной выше оценки эффективности можно сделать вывод, что лучше всего и по памяти, и по времени работает хеш-таблица. Поиск элемента в хэш-таблице занимает меньше времени, чем в дереве.

**Контрольные вопросы**

# Что такое дерево?

Дерево – это нелинейная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим». Дерево с базовым типом Т определяется рекурсивно либо как пустая структура (пустое дерево), либо как узел типа Т с конечным числом древовидных структур этого же типа, называемых поддеревьями.

# Как выделяется память под представление деревьев?

Способ выделения памяти под деревья определяется способом их представления в программе. C помощью матрицы или списка может быть реализована таблица связей с предками или связный список сыновей. Целесообразно использовать списки для упрощенной работы с данными, когда элементы требуется добавлять и удалять, т. е. выделять память под каждый элемент отдельно. При реализации матрицей память выделяется статически.

# Какие стандартные операции возможны над деревьями?

Основные операции с деревьями: обход дерева, поиск по дереву, включение в дерево, исключение из дерева. Обход вершин дерева можно осуществить следующим образом:

* + сверху вниз (префиксный обход)
  + слева направо (инфиксный обход)
  + снизу вверх (постфиксный обход)

# Что такое дерево двоичного поиска?

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

Это свойство называется характеристическим свойством дерева двоичного поиска и выполняется для любого узла, включая корень. С учетом этого свойства поиск узла в двоичном дереве поиска можно осуществить, двигаясь от корня в левое или правое поддерево в зависимости от значения ключа поддерева.

# Чем отличается идеально сбалансированное дерево от АВЛ дерева?

Идеально сбалансированное дерево: при добавлении узлов в дерево мы будем их равномерно располагать слева и справа, и получится дерево, у которого число вершин в левом и правом поддеревьях отличается не более, чем на единицу.

АВЛ-дерево: двоичное дерево называется сбалансированным, если у каждого узла дерева высота двух поддеревьев отличается не более чем на единицу.

# Чем отличается поиск в АВЛ-дереве от поиска в дереве двоичного по- иска?

Временная сложность поиска элемента в АВЛ дереве – О(log2n)

Временная сложность поиска элемента в дереве двоичного поиска –от О(log2n) до O(n).

# Что такое хеш-таблица, каков принцип ее построения?

Массив, заполненный в порядке, определенным хеш-функцией, называется хеш-таблицей. Функцию, по которой можно вычислить этот индекс. называется хеш-функцией. Принято считать, что хорошей является такая функция, которая удовлетворяет следующим условиям:

* + функция должна быть простой с вычислительной точки зрения;
  + функция должна распределять ключи в хеш-таблице наиболее равномерно.
  + функция должна минимизировать число коллизий

# 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.

Первый метод – 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.

Второй метод - внутреннее (закрытое) хеширование (открытая адресация). Оно, состоит в том, чтобы полностью отказаться от ссылок. 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.

# В каком случае поиск в хеш-таблицах становится неэффективен?

Поиск в хеш-таблицах становится менее эффективен, если наблюдается большое число коллизий. Тогда вместо ожидаемой сложности О(1) получим сложность O(n).

В первом методе - поиск в списке осуществляется простым перебором, так как при грамотном выборе хеш-функции любой из списков оказывается достаточно коротким.

Во втором методе – необходимо просматривать все ячейки, если есть много коллизий.

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

Хеш-таблица - от О(1) до O(n)

АВЛ-дерево - О(log2n)

Дерево двоичного поиска – от О(log2n) до O(n).