Лабораторная работа №7

Янова Даниэлла, ИУ7-33

Сбалансированные деревья, хеш –таблицы

**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 (Вариант 2)​:**  Используя предыдущую программу (задача №6), сбалансировать полученное дерево. Вывести его на экран в виде дерева. Построить хеш-таблицу из слов текстового файла. Осуществить поиск введенного слова в двоичном дереве поиска, в сбалансированном дереве и в хеш-таблице. Сравнить время поиска, объем памяти и количество сравнений при использовании различных структур данных.

**Входные данные**:

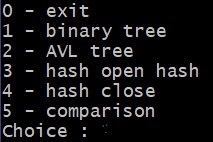
Программа предназначена для работы с текстовым файлом. Файл может содержать произвольные слова.

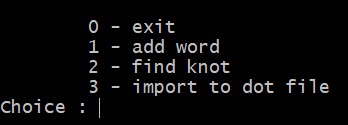
**Выходные данные:**

По требованию пользователя программа конвертирует дерево в .png, а также печатает хеш-таблицы (с открытой и закрытой адресацией). Для перечисленных структур реализованы поиск, добавление и удаление элементов. При необходимости, программа производит сравнение времени поиска в указанных структурах.

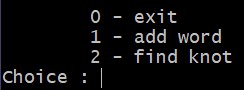
**Обращение к программе:** Через консоль

**Интерфейс программы:**



Для деревьев  


Для хеш-таблиц



**Внутренние структуры данных:**

*Бинарное дерево*:

struct tree\_node

{

const char \*word;

int count;

struct tree\_node \*left;

struct tree\_node \*right;

struct tree\_node \*parent;

};

*Сбалансированное дерево*:

struct avl\_tree

{

char key[MAX\_CHAR];

unsigned char height;

node\* left;

node\* right;

};

*Таблица с открытой адресацией*:

struct htype

{

char val[MAX\_CHAR]; /\* значение элемента данных \*/

struct htype \*next; /\* указатель на следующий элемент цепочки \*/

};

*Таблица с закрытой адресацией*:

struct close\_hash

{

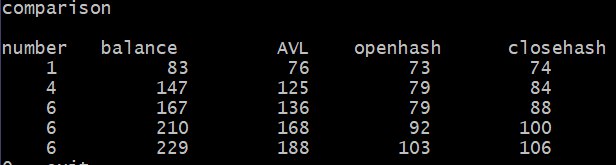
char val[MAX\_CHAR];

};

**Особенности реализации таблиц:**

Размер таблицы зависит от количества различных элементов, которые содержатся в таблице. Поэтому при превышении определенного количества, происходит перехеширование таблицы. Это необходимо для того, чтобы избавится от накопления коллизий. Для таблицы с открытой адресацией перехеширование происходит, если отношение кол-ва различных элементов к размеру таблицы равно 2.

**Сравнение поиска по времени:**



**Анализ памяти**:

Деревья - структуры, память на которые выделяется по мере добавление элементов. Объем памяти прямо пропорционален количеству элементов. Так как для реализации сбалансированного дерева поиска необходимо хранить дополнительные данные (высоту поддерева для каждой вершины), то для его хранения необходимо больше памяти. Так как кол-во элементов в таблице также зависит от количества элементов в данной реализации (при перехешировании размер таблицы удваивается), то количество памяти на хранение таблиц также линейно зависит от кол-ва элементов. Таблица с открытой адресацией требует больше памяти по сравнению с открытой, так как в ней выделяется константный объем памяти под все элементы, тогда как таблица с открытой адресацией может содержать лишь пустые указатели. “Лучшие” показатели памяти для таблиц по сравнению с деревьями можно объяснить тем, что в деревьях, помимо значения, необходимо хранить указатель на потомков.

**Вывод:**

Основным преимуществом рассмотренных структур данных является возможная высокая эффективность реализации алгоритмов добавления, поиска, удаления элементов. Среднее время выполнения этих операций для таблиц намного меньше, чем для деревьев, но у деревьев есть по крайней мере одно заметное преимущество по сравнению с хеш-таблицей. В них можно выполнить проход по возрастанию или убыванию ключей и сделать это быстро. Поэтому выбор структур данных напрямую зависит от области их применения.

**Теоретическая часть:**

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

Если при добавлении узлов в дерево располагать их равномерно слева и справа, то получится дерево, у которого число вершин в левом и правом поддеревьях отличается не более, чем на единицу. Такое дерево называется идеально сбалансированным. Идеальная балансировка дает наименьшую высоту дерева. В АВЛ дереве высота левого и правого поддеревьев отличается не более, чем на единицу.

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

Поиск в АВЛ дереве имеет трудоемкость О(log2n), в то время как в обычном ДДП может иметь O(n). АВЛ дерево никогда не вырождается в линейный список (исключение – дерево из двух элементов), в то время как «внешний вид» ДДП может зависеть от того, в каком порядке в него добавлялись элементы.

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

Хеш-таблицей называется массив, заполненный элементами в порядке, определяемом хеш-функцией. Хеш-функция каждому элементу таблицы ставит в соответствие некоторый индекс. ХФ должна быть простой для вычисления, распределять ключи в таблице равномерно и давать минимум коллизий.

**4. Что такое коллизии?**

Каковы методы их устранения. Коллизия – ситуация, когда разным ключам хеш-функция ставит в соответствие один и тот же индекс. Основные методы устранения коллизий: открытое и закрытое хеширование. При открытым хешировании, конфликтующие ключи просто добавляются в список, находящийся по их общему индексу. Поиск по ключу сводится к определению индекса, а затем к поиску ключа в списке перебором. При закрытом хешировании, конфликтующий ключ добавляется в первую свободную ячейку после «своего» индекса. Поиск по ключу сводится к определению начального приближения, а затем к поиску ключа методом перебора.

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

Поиск в ХТ становится неэффективен при большом числе коллизий – сложность поиска возрастает по сравнению с О(1). В этом случае требуется реструктуризация таблицы – заполнение её с использованием новой хеш-функции.

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

В хеш-таблице минимальное время поиска О(1). В АВЛ: О(log2n). В дереве двоичного поиска О(h), где h - высота дерева (от log2n до n)