Лабораторная работа №7
по дисциплине «Типы и структуры данных»
Тема :Деревья.Хеш-таблицы.

Условие задачи:
Построить хеш-таблицу по указанным данным. Сравнить эффективность поиска в сбалансированном двоичном дереве, в двоичном дереве поиска и в хеш-таблице (используя открытую и закрытую адресацию). Вывести на экран деревья и хеш-таблицы. 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 (по времени и памяти) для поставленной задачи. Оценить эффективность поиска в хеш-таблице при различном количестве коллизий и при различных методах их разрешения.

Вариант 9 (1).
Используя предыдущую программу (задача №6), сбалансировать полученное дерево. Вывести его на экран в виде дерева. Построить хеш-таблицу из чисел файла. Осуществить поиск введенного целого числа 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 строит сбалансированное двоичное дерево, двоичное дерево поиска и хеш таблицы по информации:

1. полученной из файла. Файл содержит целые числа, разделенные пробелом или Tab.
2. добавленной пользователем через консоль (с помощью меню пункт (3)). Пользователь вводит целое число после сообщения “Введите число: ”

Выходные данные:
Программа печатает:

- Деревья (двоичное сбалансированное и двоичное поиска) в графическом виде.

Голова дерева - элемент, в который идет →.

Верхний потомок (↗) больше родителя.
Нижний потомок (↘) меньше родителя.

Для каждой вершины сбалансированного бинарного дерева

в скобках указана высота данной вершины.

- Хеш-таблицы с открытой и закрытой адресацией

```
Хеш-таблица с закрытой адресацией
  Коллизий - 4
   Хеш   |   Элемент
     0   |     15
     1   |      1
     2   |      2
     3   |     30
     5   |      5
     6   |     20
    10   |     10
    11   |     25
    12   |     26
Хеш-таблица с открытой адресацией
  Коллизий - 3
   Хеш   |   Элементы
     1   |    25     1
     2   |    10    26     2
     4   |    20
     5   |     5
     6   |    30
     7   |    15
```

При выводе на печать хеш-таблиц также выводится количество коллизий.

В хеш-таблице с закрытой адресацией во второй колонке находится одно число.

В хеш-таблице с открытой адресацией во второй колонке может находиться несколько чисел. Они разделены пробелами.

- Для поиска элемента, введенного пользователем, в различных структурах выводится таблица со временем поиска и количеством сравнений.

```
      Структура        | Время(тик) | Сравнений
     Дерево поиска     |       1002 |     3
Сбалансированное дерево |        798 |     4
 Хеш-таблица (Закрытая) |       1278 |     2
 Хеш-таблица (Открытая) |        858 |     1
```

- Таблица, содержащая среднее время поиска элементов и среднее количество сравнений.

```
     Структура   | Среднее время поиска | Среднее количество сравнений
  Дерево поиска  |          506         |         3.10
 Сбалансир.дерево|          388         |         2.80
 Хеш-таблица(cl) |          552         |         1.60
 Хеш-таблица(op) |          389         |         1.50
```

Меню программы.

Работа с программой осуществляется с помощью меню:

```
МЕНЮ:
(1) Загрузить информацию из файла
(2) Вывести на экран деревья и хеш-таблицы
(3) Добавить элемент
(4) Удалить элемент
(5) Найти элемент и провести сравнение в различных структурах
(6) Подсчитать среднее количество сравнений для поиска
(0) Завершение работы программы
Выберите пункт меню: 0
```

Пользователь вводит номер пункта меню на запрос “Выберите пункт меню: ”

(1) Загружает информацию из текстового файла. Генерирует дерево двоичного поиска, сбалансированное двоичное дерево и две хеш-таблицы на основе этой информации.
(2) Выводит на экран деревья (двоичного поиска и сбалансированное) и хеш-таблицы (с открытой и закрытой адресацией).
(3) Запрашивает у пользователя число, которое нужно добавить к структурам. Если до выбора этого пункта информация из файла загружена не была, генерирует два новых дерева и две хеш-таблицы с введенным элементом.
(4) Запрашивает у пользователя число, которое нужно удалить из структур.
(5) Запрашивает у пользователя число, которое нужно найти в структурах. При его наличии в структурах на экран выводится время, за которое был совершен поиск и количество сравнений.
(6) Ищет в дереве N элементов (т. е. процедура поиска N раз). Выводит на экран количество затраченного времени / N (среднее время поиска) и количество сравнений / N (среднее количество сравнений) для каждой структуры данных.

Структуры данных:
Узел дерева:

```
typedef struct tree_node
{
    int data;                  data - число
    int count;                 count - количество встреч
    unsigned char height;      height - высота узла
    struct tree_node *left;    left - левый потомок
    struct tree_node *right;   right - правый потомок
} tree_node;
```

Оболочка хеш-таблицы с открытой адресацией:

```
typedef struct OpenShell
{
    hash_open *data;       data - хеш-таблица
    size_t size;           size - её размер
    size_t collisions;     collisions - количество коллизий
} hash_table_open;
```

Элементы хеш-таблицы с открытой адресацией:

```
typedef struct HashTable_open
{
    struct HashTable_open *next;   next - ссылка на следующий эл-т
    int info;                      info - число
} hash_open;
```

Оболочка хеш-таблицы с закрытой адресацией:

```
typedef struct CloseShell
{
    hash_close *data;
    size_t array_s;
    size_t size;
    size_t collisions;

} hash_table_cl;
```

data - хеш-таблица
array_s - количество элементов
size - размер таблицы
collisions - количество коллизий

Элементы хеш-таблицы:

```
typedef struct HashTable_close
{
    int info;
    int status;
} hash_close;
```

info - число
status - статус

Хеш-функция:

```
unsigned int hash(int info, int key)
{
    return (unsigned int) info % key;
}
```

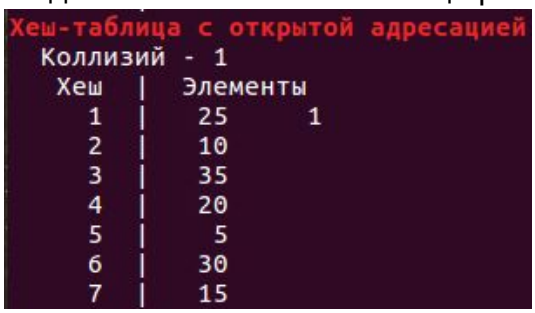
info - число
key - размер таблицы

Реструктуризация хеш-таблиц:
При достижении количества коллизий больше установленного значения, хеш-таблицы реструктуризируются. При этом размер изменяется на константное значение, что приводит к изменению хеш-функции.
Например:
Элементы: 1 5 10 15 20 25 30
Максимально допустимое количество коллизий = 5
В хеш-таблице с открытой адресацией Хеш = 0 соответствует 6 элементов => 5 коллизий. Размер таблицы = 5

```
Хеш-таблица с открытой адресацией
  Коллизий - 5
   Хеш   |   Элементы
     0   |     5     10     15     20     25     30
     1   |     1
```

Тогда при добавлении элемента 35 Хеш = 0 будет соответствовать 7 элементов. Число коллизий увеличится до 6. Таблица реструктуризируется. Новый key (из хеш-функции) станет равным 5 + C (в данном примере C = 3) = 8

Тогда элементы в хеш-таблице расположатся:



Число коллизий после реструктуризации таблицы равно 1.

Сравнение поиска в различных структурах данных:

Для каждой ячейки Элемент-Структура первое число - количество сравнений, второе число - время поиска.

Элемент	Сбалансированное дерево		Двоичное дерево поиска		Хеш-таблица с открытой адресацией		Хеш-таблица с закрытой адресацией	
Вершина двоичного дерева поиска	**3**	**636**	**1**	**400**	**2**	**1028**	**1**	**816**
Вершина AVL-дерева	**1**	**332**	**4**	**1010**	**1**	**940**	**1**	**840**
Средний узел дерева	**3**	**512**	**4**	**1156**	**2**	**1228**	**2**	**1024**
Лист двоичного дерева поиска	**4**	**1136**	**5**	**1040**	**1**	**996**	**4**	**1144**
Лист AVL-дерева	**4**	**954**	**3**	**1099**	**1**	**966**	**2**	**976**
Элемент коллизии для з.а.	**2**	**780**	**4**	**1058**	**3**	**1188**	**1**	**754**
Элемент коллизии для о.а	**4**	**1099**	**5**	**1007**	**1**	**996**	**4**	**1200**

Время поиска элемента в хеш-таблицах относительно стабильно.
Однако оно зависит от количества коллизий в таблице: чем больше коллизий, тем больше времени потребуется. По результатам замеров получилось, что время поиска в таблице с открытой адресацией при одинаковом количестве коллизий несущественно больше, чем время поиска в таблице с закрытой адресацией. Это может зависеть от конкретной реализации поиска элемента в таблице.
Время поиска в деревьях зависит прежде всего от расположения искомого элемента. Если искомый элемент - вершина, то время на его поиск мало, если искомый элемент - лист дерева, время на его поиск больше, чем время, потраченное на поиск какого-либо среднего узла в дереве.

Анализ затраченной памяти:
Память под деревья выделяется по мере добавления в них элементов. Следовательно объем затраченной памяти пропорционален количеству элементов в дереве. Для сбалансированного дерева требуется больше памяти для хранения высот его вершин.
Количество памяти под хранение хеш-таблиц также линейно зависит от количества элементов. Количество элементов может увеличиваться при реструктуризации таблицы. Таблица с закрытой адресацией требует больше памяти, так как в ней выделяется константное количество памяти под все элементы, а в таблице с закрытой реализацией изначально выделяется на массив пустых указателей.

Вывод:
Основное преимущество рассмотренных структур - высокая эффективность алгоритмов добавления, поиска и удаления элементов.
Выбор структуры данных напрямую зависит от области ее применения.
Время выполнения стандартных операций в хеш-таблицах меньше, чем у деревьев. В таблицах можно хранить статические данные с возможностью быстрого доступа к ним по ключу. А в деревьях удобно сортировать информацию.

Вопросы к лабораторной работе:
1. Чем отличается идеально сбалансированное дерево от АВЛ дерева?
Если при добавлении узлов в дерево мы будем их равномерно располагать 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. Адельсон-Вельский и Ландис сформулировали менее жесткий критерий сбалансированности таким образом: двоичное дерево называется сбалансированным, если у каждого узла дерева высота двух поддеревьев отличается не более чем на единицу. Такое дерево называется АВЛ-деревом.

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

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

3. 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.*

4. Что такое коллизии? Каковы методы их устранения.

Может возникнуть ситуация, когда разным ключам соответствует одно значение хеш-функции, то есть, когда h(K1)=h(K2), в то время как K1 ≠ K2. Такая ситуация называется коллизией. 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. Поиск в этом списке осуществляется простым перебором, так как при грамотном выборе хеш-функции любой из списков оказывается достаточно коротким. Другой путь решения проблемы, связанной с коллизиями – 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. При этом, если индекс следующего просматриваемого элемента определяется добавлением какого-то постоянного шага (от 1 до n), то данный способ разрешения коллизий называется линейной адресацией.

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

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

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

Хеш-таблица -от O(1) до O(n)
АВЛ-дерево - O(log2n)
Дерево двоичного поиска – от O(log2n) до O(n).