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)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7 «СБАЛАНСИРОВАННЫЕ ДЕРЕВЬЯ, ХЕШ-ТАБЛИЦЫ»
Вариант 6

Студент Кладницкий А. Б.

Преподаватель Силантьева А. В.

Группа ИУ7 – 32Б

2022 г.

1. **Описание условия 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-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 (по времени и памяти) для поставленной задачи. Оценить эффективность поиска в хеш-таблице при различном количестве коллизий и при различных методах их разрешения.

2. **ТЗ**

   Построить хеш-таблицу для слов текстового файла (задача №6). Осуществить поиск указанного слова в двоичном дереве поиска (ДДП) и в хеш-таблице, если его нет, то добавить его (по желанию пользователя) в дерево и, соответственно, в таблицу. При необходимости использовать реструктуризацию таблицы. Сбалансировать дерево. Сравнить время поиска, объем памяти и количество сравнений при использовании ДДП, сбалансированных деревьев и хеш-таблиц. Сравнить эффективность добавления ключа в таблицу или ее реструктуризацию для различной степени заполненности таблицы

   1. Исходные данные:
      - Пункт меню
      - Имя файла
      - Данные для поиска
      - Меню:
        1. Open file
        2. Show graphs
        3. Search
        4. Check efficiency of different structs
        5. Check efficiency of different hash tables
        6. Print menu again
        7. Set restructurising flag (default = yes)
        0. Exit
   2. Результирующие данные:
      - В зависимости от пункта меню:
        - Вывод деревьев и таблицы
        - Вставка элемента в деревья и таблицы
        - Результаты замеров
   3. Задача программы:
      - Построение дерева двоичного поиска, AVL-дерева и хеш-таблицы
      - Поиск и добавление данных в структурах
      - Замер эффективности для разных структур при выполнении задачи

4. Способ обращения к программе:
    - Запуск через терминал (./app.exe)
5. Возможные ошибки:
    - Ошибка выделения динамической памяти
    - Ошибка чтения из потока
    - Ошибка чтения целого числа
    - Ошибка открытия файла
    - Вывод деревьев и таблицы без предварительного открытия файла

3. Описание внутренних структур данных

Тип данных `tree_t`:

```
typedef struct tree_node tree_t;
struct tree_node
{
  char *data;
  tree_t *right;
  tree_t *left;
};
```

Тип содержит в себе данные, хранящиеся в узле дерева, и указатели на его левого и правого потомков.

Тип данных `tree_avl_t`:

```
typedef struct tree_avl_node tree_avl_t;
struct tree_avl_node
{
  char *data;
  int diff;
  tree_t *right;
  tree_t *left;
};
```

Тип содержит в себе данные, хранящиеся в узле дерева, и указатели на его левого и правого потомков, а также разность высот потомков.

Тип данных `hash_elem`:

```
typedef struct hash_elem
{
  char *key;
  char *data;
} elem_t;
```

Тип содержит в себе пару ключ-значение.

Тип данных hash_table:

```
typedef struct hash_table
{
  elem_t **data;
  list_t **links;
  int size;
  int count;
} hash_t;
```

Тип содержит в себе указатель на область памяти, в которой хранятся элементы таблицы, указатель на область, в которой хранятся дополнительные списки (для открытого разрешения коллизий), размер таблицы и текущей кол-во элементов в ней.

Тип данных list_t:

```
typedef struct list list_t;
struct list
{
  elem_t *elem;
  list_t *next;
};
```

Тип содержит в себе значение узла списка и указатель на его следующий узел.

4. **Алгоритм**
    1. Считать пункт меню
    2. Выполнить действие в соответствии с пунктом меню (чтение из файла, вывод, замеры и т.д.)
5. **Набор тестов**

	Описание	Входные данные	Результат
1.	Неверный пункт меню	q, 100, -5 и т.д.	EXIT_WRONG_MENU_NUM
2.	Ошибка выделения динамической памяти	???	EXIT_ALLOCATE
3.	Неверное имя файла	Имя несуществующего файла	EXIT_BAD_FILE
4.	Пустой ввод	-	EXIT_WRONG_READ
5.	Вывод незаполненных деревьев и таблицы	Запрос вывода без запроса ввода	EXIT_NEED_PREP
6.	Ошибка чтения флага	1hg12, 7.82, wjkgh и т.д.	EXIT_BAD_INT
7.	Чтение из файла	Пункт меню 1 Имя файла	Слова из файла записаны в структуры
8.	Вывод структур в графическом виде	Пункт меню 2	Деревья и таблица в графическом виде
9.	Поиск в структурах с добавлением	Пункт меню 3 Запрос на добавление	Результат поиска (да/нет) Добавление данного в структуры
10.	Поиск в структурах без добавления	Пункт меню 3	Результат поиска (да/нет)
11.	Сравнение эффективности разных структур	Пункт меню 4	Результаты замеров
12.	Сравнение эффективности разных таблиц	Пункт меню 5	Результаты замеров
13.	Изменение флага	Пункт меню 7	Изменение флага реструктуризации

6. Оценка эффективности

Эффективность поиска в дереве в разных структурах:

Рассматривается дерево бинарного поиска, AVL-дерево, а также хеш-таблицы с открытым и закрытым способом разрешения коллизий.

Время (мкс), память (байт):

Элементы	Дерево бин. поиска		AVL-дерево		Хеш-таблица (откр.)		Хеш-таблица (закр.)	
	Время	Память	Время	Память	Время	Память	Время	Память
25	254	600	159	800	228	1240	217	912
100	317	2400	228	3200	212	5144	200	4320
250	412	6000	307	8000	238	14136	214	12528
500	404	12000	288	16000	216	32920	192	27648

Эффективность добавления элемента в хеш-таблицу в зависимости от размера и использования реструктуризации:

Рассматриваются хеш-таблицы с открытым и закрытым способом разрешения коллизий и с использованием реструктуризации или без нее. Реструктуризация используется только при необходимости.

Время (мкс):

Элементы	С реструктуризацией		Без реструктуризации	
	Открытая	Закрытая	Открытая	Закрытая
25	75	84	55	65
100	60	69	56	56
250	56	57	60	54
500	51	61	66	60

7. Вывод

Хеш-таблицы, в отличие от деревьев, производят поиск за примерно одинаковое время не зависимо от размера, в то время как время поиска в деревьях преимущественно возрастает с увеличением кол-ва элементов.

AVL-деревья в большинстве случаев оказываются быстрее деревьев бинарного поиска при поиске элемента.

Начиная примерно со 100 элементов, хеш-таблица становится эффективнее деревьев по времени, но всегда проигрывает им по занимаемой памяти.

Таблица, использующая открытый способ разрешения коллизий, занимает больше памяти и немного больше времени на поиск, чем таблица с закрытым способом.

При добавлении элемента в хеш-таблицу наблюдается следующая закономерность: без реструктуризации, времена добавления практически не отличаются (небольшие отличия из-за возможных коллизий). При

использовании реструктуризации для некоторых размеров наблюдается увеличение времени работы, что объясняется необходимостью реструктуризации. Добавление в таблица с закрытым способом разрешения коллизий происходит немного дольше, чем для открытого способа.

8. Контрольные вопросы

1) Чем отличается идеально сбалансированное дерево от АВЛ дерева?
   В идеально сбалансированном дереве не более чем на единицу отличается число вершин слева и справа, а в АВЛ-дереве не более чем на единицу отличаются высоты левого и правого поддеревьев.
2) Чем отличается поиск в АВЛ-дереве от поиска в дереве двоичного поиска?
   Он, в общем случае, выполняется быстрее, т.к. меньше высота самого дерева.
3) Что такое хеш-таблица, каков принцип ее построения?
   Хеш-таблица – массив, заполненный в порядке, определенном хеш-функцией.
   Для построения выбирается некоторая хеш-функция. Результат применения этой функции к ключу – индекс в массиве, в который нужно записать данные (соответствующие ключу).
4) Что такое коллизии? Каковы методы их устранения?
   Коллизия – ситуация, когда разным ключам соответствует одно и то же значение хеш-функции.
   Два способа: закрытый и открытый.
   При закрытом идет добавление в следующий пустой элемент (или шаги квадратами чисел).
   При открытом создаются дополнительные списки.
5) В каком случае поиск в хеш-таблицах становится неэффективен?
   При большом количестве коллизий (тогда нужно провести реструктуризацию) или при маленьком количестве элементов.
6) Эффективность поиска в АВЛ деревьях, в дереве двоичного поиска, в хеш-таблицах и в файле.
   По времени (для больших объемов данных):
   Файл > ДДП > АВЛ > хеш-таблица
   По памяти (для больших объемов данных):
   Хеш-таблица > АВЛ > ДДП > файл
   Зависимости обратные