Отчет по 6 лабораторной работе
По дисциплине «Типы и структуры данных»

Подготовил Пересторонин Павел
Группа ИУ7-33Б
Вариант 15

Цель работы

Построить ДДП, сбалансированное двоичное дерево (АВЛ) и хеш-таблицу по указанным данным. Сравнить эффективность поиска в ДДП в АВЛ дереве и в хеш-таблице (используя открытую или 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.

Техническое задание

Построить ДДП, в вершинах которого находятся слова из текстового файла. Вывести его на экран в виде дерева. Сбалансировать полученное дерево и вывести его на экран. Удалить указанное слово в исходном и сбалансированном дереве. Сравнить время удаления и объем памяти. Построить хеш-таблицу из слов текстового файла, задав размерность таблицы с экрана, используя метод цепочек для устранения коллизий. Вывести построенную таблицу слов на экран. Осуществить удаление введенного слова, вывести таблицу. Сравнить время удаления, объем памяти и количество сравнений при использовании ДДП, сбалансированных деревьев, хеш-таблиц и файла.

Входные данные

Пункт в меню; элемент, который требуется удалить; имя файла.

Выходные данные

Статистика работы двоичного дерева поиска, АВЛ-дерева, хэш-таблицы и файла (время работы и кол-во сравнений). Вывод деревьев и хэш-таблицы. Информация о совершении той или иной операции. Статистика (среднее количество сравнений при поиске и т.п.)

Возможные аварийные ситуации

Некорректный ввод.

Структуры данных

Структура узла хэш-таблицы:

```
typedef struct node_h
{
   char *data;
   struct node_h *next;
} node_h;
} elem_t;
```

data — указатель на строку с данными.
next — указатель на следующий элемент в цепочке.

Структура хэш-таблицы:

```
typedef struct hash_t
{
   node_h **data;
   int base;
} hash_t;
```

base — размер хэш-таблицы (а также максимальная длина хэша = base — 1)
data — массив цепочек с данными.

Структура узла АВЛ-дерева:

```
typedef struct node_t
{
   char *value;
   unsigned int height;
   struct node_t *left;
   struct node_t *right;
} node_t;
```

value — указатель на строку с данными.
Height — высота узла (храню вместо коэффициента сбалансированности).
left, right — указатели на левое и правое поддеревья.

Структура узла двоичного дерева:

```
typedef struct node_tb
{
   char *value;
   struct node_tb *left;
   struct node_tb *right;
} node_tb;
```

value — указатель на строку с данными.
left, right — указатели на левое и правое поддеревья.

Алгоритм.

Хэширование.
В данной программе использовалось открытое хэширование (массив цепочек). Функция хэширования простая: взятие остатка от деления суммы кодов символов на размер массива цепочек. Цепочки реализованы в виде односвзязных списков.

Дерево.
При добавлении элемента в дерево он просто вставлялся (так как у нас либо уже есть элемент в дереве и мы его оставляем, либо мы добавляем листовую вершину и нам ничего делать не надо)
При удалении элемента из дерева элемент удаляется, а на его место ставится наименьший элемент из правого поддерева (таким образом главное свойство ДДП не нарушается)

АВЛ-дерево.
В АВЛ-дереве все происходит так же, как и в обычном дереве, только при изменении кол-ва элементов в каком-либо поддереве, это поддерево балансируется, а также балансируются все вышестоящие поддеревья, которые содержат данное.

Дополнительная информация.
Обход дерева реализован как «обход справа».
Удаление из файла не использует доп. файлов и не переписывает большой кусок файла: в моем случае удаляемые элементы просто затираются пробелами (выигрыш по времени).

Перехэширование.
В данной программе также указан максимальный размер коллизий. В начале размер массива цепочек выбирается равным минимальному простому числу, которое больше количества элементов в тестовом файле. В случае, если во время заполнения таблицы максимальный размер коллизий превышается, происходит перехэширование: размер массива цепочек увеличивается до ближайшего большего простого числа и пересчитываются хэши всех элементов. Если в новой таблице не удается удовлетворить условию количества коллизий, то берется следующее просто число и строится новая таблица (и т. д.).

Тесты.

1. Данные не выгружены.
1.1. Удаление (та же реакция при удалении несуществующего слова).

```
Enter word to delete: word
No such word in structures
```

1.2. Вывод структур.
<Пустой вывод>

1.3. «Перебазирование» хэш-таблицы.
<Пустой вывод> (без сообщений об ошибке)

1.4. Вывод статистики.

```
Average comparison number:
File: 11.0
Hash-table: 0.0
AUS-tree: 0.000000
Tree: 0.000000
```

2. Чтение из файла.

```
Input from file is done.
```

3. Бинарное дерево.

3.1. Вывод дерева.

3.2. Удаление «Ivanov»

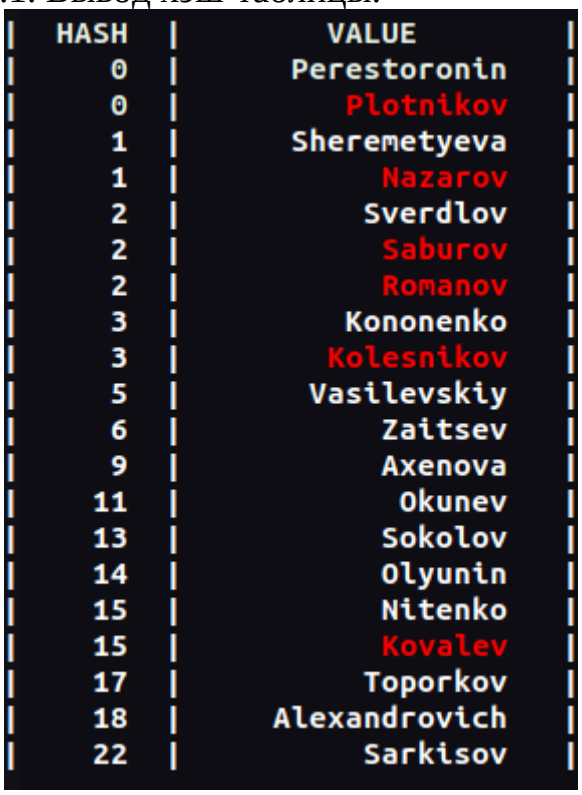
4. АВЛ-дерево.

4.1. Вывод дерева.

4.2. Удаление «Simonenko»

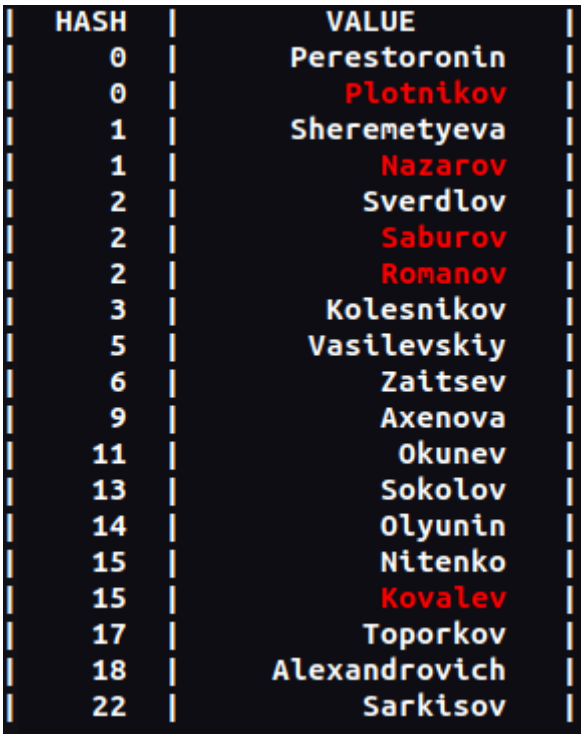
5. Хэш-таблица.

5.1. Вывод хэш-таблицы.



HASH	VALUE
0	Perestoronin
0	Plotnikov
1	Sheremetyeva
1	Nazarov
2	Sverdlov
2	Saburov
2	Romanov
3	Kononenko
3	Kolesnikov
5	Vasilevskiy
6	Zaitsev
9	Axenova
11	Okunev
13	Sokolov
14	Olyunin
15	Nitenko
15	Kovalev
17	Toporkov
18	Alexandrovich
22	Sarkisov

5.2. Удаление «Kononenko».



HASH	VALUE
0	Perestoronin
0	Plotnikov
1	Sheremetyeva
1	Nazarov
2	Sverdlov
2	Saburov
2	Romanov
3	Kolesnikov
5	Vasilevskiy
6	Zaitsev
9	Axenova
11	Okunev
13	Sokolov
14	Olyunin
15	Nitenko
15	Kovalev
17	Toporkov
18	Alexandrovich
22	Sarkisov

6. Ошибки «перебазировки».

Ошибка, связанная с неверно введенным числом.

```
Input new base of table: -1
Wrong value of base
```

Ошибка ввода (то есть ввод пустой (конец файла или «ctrl + D»)

```
Input new base of table: Can't read base value
```

Расчеты времени и памяти.

Время

Удаление элемента, время в тиках (при данных = 1000 элементов):

	файл	хэш-таблица	дерево	АВЛ-дерево
Среднее время	332000	1600	3100	6500**
Среднее количество сравнений	500	1.4*	20	10

* Данная величина очень сильно варьируется и в данной таблице представлена конкретно для моего файла (можете найти в папке с программой)
** Время получается большим в связи с тем, что после удаления нужно балансировать каждый узел, в той ветви, где было совершенно удаление.

Память

Накладная память.

1. Файл.
В случае файла мы тратим лишнюю память только на хранение разделителя между словами (то есть 1 байт на 1 элемент). Таким образом:
Memory_size = M байт. (M — количество слов)

2. Хэш-таблица.
Под каждый элемент выделяется узел. В узле, помимо данных, хранится указатель на следующий элемент + указатель на данные (16 байт). Хранение хэш-таблицы в виде массива указателей на цепочки тоже уходит некоторое количество памяти. При идеальном подборе соответствия хэш-функции и данных, длина каждой цепочки будет равна максимально допустимому количеству коллизий (в моем случае это количество равно 3 (однако меняется путем изменения одного параметра в define)), допустим это число равно N. Тогда для хранения массива указателей уходит M / N * 8 байт памяти, где M —

количество элементов в таблице.
Memory_size = M * 16 + M / N * 8 байт.

3. АВЛ-дерево.
В данном случае память отводится на хранение узлов. Узел = 2 указателя на ветви + указатель на данные + высота = 28 байт.
Memory_size = 28 * M байт.

4. Дерево.
Занимаемая память такая же, как и в случае обработки АВЛ-дерева, только мы не храним данные о высоте узла. Таким образом:
Memory_size = 24 * M байт.

Память, занимаемая СД (в данной таблице учитываются накладные расходы, то есть без учета веса самих слов):

Количество элементов	файл	хэш-таблица*	дерево	АВЛ-дерево
100	100	1800	2400	2800
1000	1000	18000	24000	28000
10000	10000 ~ 10 Кбайт	180000 ~ 180 Кбайт	240000 ~ 240 Кбайт	280000 ~ 280 Кбайт

* N = 4

Выводы по проделанной работе

В результате данной лабораторной работы можно сделать следующие выводы:
1) При сильной ограниченности по памяти желательно использовать файл;
2) При достаточной памяти максимальный выигрыш по времени удаления дает хэш-таблица (в том числе и выигрыш по времени доступа)
3) В плохом случае или при плохой хэш-функции (крайние случаи), хэш-функция может проиграть по времени доступа АВЛ-дереву (которому при количестве элементов 1.000.000 требуется всего около 21 сравнения), поэтому АВЛ-дерево тоже может быть выгодно использовать в некоторых задачах для частого обращения и нахождения элементов. Однако, как мы убедились, АВЛ-дерево, в следствие нужды в постоянной балансировке, очень долго обрабатывает удаление элементов.

Контрольные 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?

В самой популярной реализации дерева в виде узлов (СД, имеющая некоторую полезную нагрузку, а также указатель(-и) на подобные себе элементы), память выделяется на один узел при добавлении элемента, а дальше уже узлы могут перемешиваться, указывая друг на друга в ином порядке и т. п.

3. Какие стандартные операции возможны над деревьями?

Поиск, обход, включение в дерево, исключение в дерево.

4. 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.

5. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

В АВЛ-дереве критерий сбалансированности «высота 2 поддеревьев отличается не более, чем на 1», а в идеальном сбалансированном дереве - «количество элементов в левом и правом поддеревьях отличается не больше, чем на 1».

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

В частном случае простое ДДП (двоичное дерево поиска) может развернуться в линейный список (то есть каждый следующий элемент больше предыдущего). Преимущество АВЛ-дерева перед ДДП в том, что в первом такая ситуация невозможна (только при высоте 2*), однако в первом тратится больше времени на балансировку.

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

Массив, заполненный в порядке, определенным хеш-функцией, называется хеш-таблицей. Хэш-функция — функция, ставящая в соответствие какому-то объекту уникальный хэш (индекс). Хэш-функция находит некоторый «адрес» в хэш-таблице по значению элементов и помещает их по тому адресу. Таким

образом, чтобы получить адрес элемента, нужно просто рассчитать значение функции.

8. Что такое коллизии? Каковы методы их устранения.

Коллизии — совпадение значений хэшей для 2 и более разных объектов в хэш-таблице. При внешнем хэшировании используются цепочки (элементы с одинаковыми хэшами собираются в список), при закрытом — элемент ставится на первое (например; можно и не на первое определить) свободное место.

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

При большом количестве коллизий или при крайне малом объеме данных.

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

Поиск в АВЛ-дереве в общем случае лучше, чем в ДДП, так как в ДДП есть специфичные частные случаи и сложность поиска очень варьируется (см. п.6), таким образом поиск в АВЛ-дереве надежнее (быстрее в общем случае). Поиск в хэш-функции имеет временную сложность O(1) и считается самым эффективным по времени методом организации поиска, однако в частном случае может дать негативный результат (неправильно подобранная хэш-функция и т.п.)