МГТУ им. Н. Э. Баумана

Отчет по лабораторной работе №6 по предмету «Типы и структуры данных»
«Деревья, хеш-таблицы»

Выполнила Параскун София,
ИУ7-34Б
Проверила
Никульшина Т. А.

Москва, 2020 г.

1. Описание условия задачи

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

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. Добавить указанное слово, если его нет в дереве (по желанию пользователя) в исходное и сбалансированное дерево. Сравнить время добавления 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 добавление введенного слова, вывести таблицу. Сравнить время добавления, объем памяти и количество сравнений при использовании ДДП, сбалансированных деревьев, хеш-таблиц и файла.

2. Описание ТЗ, включающего внешнюю спецификацию

a) описание исходных данных и результатов

Основная программа получает на вход выбранный пользователем пункт меню (целое число). В соответствии с ним производятся операции моделирования процесса обслуживания очереди или вывода состояния.

Программа имеет на входе файл, содержащий слова, которые являются вершинами бинарного дерева.

В зависимости от выбранного режима результатом являются сообщения о состоянии выполнения операции, ДДП или АВЛ в формате png, хеш-таблица или результаты эффективности.

b) Способ обращения к программе

Программа «app.exe» запускается через консоль, после чего можно увидеть меню программы, состоящее из 5 пунктов:

1 — Считать слова из файла,
2 — Добавить слово,
3 — Вывести ДДП,
4 — Вывести АВЛ-дерево,
5— Вывести хеш-таблицу,
0 — Выход из программы.

Все вводы и выводы также осуществляются в консоли, вывод деревьев осуществляется в формате png.

c) Описание возможных аварийных ситуаций и ошибок пользователя

№ ситуации	Аварийная ситуация	Реакция программы
1	Ввод некорректного пункта меню	Сообщение «Its end of working the program!», завершение программы
2	Слово больше 10 символов	Сообщение «Too many symbols.» возврат в главное меню
3	Некорректный размер хеш-таблицы	Сообщение «Hash size is not acceptable.», возврат в главное меню

3. Описание внутренних СД

Зададим максимально возможную длину слова MAX_LEN 10. Перейдем к структурам.

```
struct binary_tree
{
    char value[MAX_LEN + 1];
    struct binary_tree *left;
    struct binary_tree *right;
};
```

Структура binary_tree, отвечающая за элемент ДДП. Здесь value – значение элемента, left и right – указатели на левого и правого потомков.

Структура avl_tree, отвечающая за элемент АВЛ-дерева. Здесь value – значение элемента, balance – состояние сбалансированности высот правой и левой ветвей, left и right – указатели на левого и правого потомков.

```
struct avl_tree
{
    char value[MAX_LEN + 1];
    int balance;
    struct avl_tree *left;
    struct avl_tree *right;
};
```

```
struct hash_slot
{
    char value[MAX_LEN + 1];
    struct hash_slot *next;
};
```

Структура hash_slot отвечает за элемент хеш-таблицы. Здесь value – значение элемента, next – укзатель на следующий элемент таблицы с таким же значением хеш-функции.

Структура tree, отвечающая за объединенное хранение всех форм бинарного дерева. Здесь count – количество элементов, binary – указатель на корень ДДП, avl – указатель на корень АВЛ-дерева, hash_size – размер хеш-таблицы, rand – массив сгенерированных величин для хеш-таблицы, hash_table – указатель на массив элементов хеш-таблицы.

```
struct tree
{
    int count;
    struct binary_tree *binary;
    struct avl_tree *avl;
    int hash_size;
    int *rand;
    struct hash_slot **hash_table;
};
```

4. Описание алгоритма

Исходное состояние — пустое дерево, все используемые структуры пусты. Чтение элементов из файла продолжается до конца файла.

При добавлении элемента в ДДП и АВЛ происходит сравнение добавляемого элемента со встречающимися ему на пути при обходе слева-направо. Однако, если в случае ДДП можно просто добавить элемент после найденного, АВЛ-дереву необходима перебалансировка, котоаря осуществляется правым или левым поворотом (в зависимости от степени сбалансированности).

При добавлении элемента в хеш-таблицу осуществляется вычисление значения хеш-функции. Так как мы работаем со строками, хеш-функцией был выбран метод исключающего или (или же XOR). В случае возникновения коллизий, они устраняются методом цепочек — добавлением в список текущего элемента.

5. Описание используемых функций

В используемых ниже функциях tree – общая структура дерева, file – исходный файл со словами.

*Void read_tree(FILE *file, struct tree *tree)* – производит считывание слов из файла и из запись в ДДП, АВЛ-дерево и хеш-таблицу.

*Unsigned long place_leaf(sruct binary_tree *leaf, struct tree *tree, int *compare)* – помещает элемент в ДДП, здесь leaf – элемент ДДП.

*Struct avl_tree *search_inc(struct avl_tree *leaf, struct avl_tree *node, int *compare)* – помещает элемент в АВЛ-дерево, выполняю перебалансировку, если это необходимо. Здесь leaf – элемент АВЛ-дерева, node – указатель на корень АВЛ-дерева.

*Struct avl_tree *balance(struct avl_tree *node)* – осуществляет балансировку дерева.

*Struct avl_tree *rotright(struct avl_tree *node)* – осуществляет правый поворот.

*Struct avl_tree *rotleft(struct avl_tree *node)* – осуществляет левый поворот.
*Unsigned long hash_func(char *leaf, struct tree tree)* – вычисляет значение хеш-функции по ключу и размещает элемент в хеш-таблицу. Здесь

Void binary_out(struct tree tree) – выводит ДДП в формате png-файла.

Void avl_out(struct tree tree) – выводит АВЛ-дерево в формате png-файла.

Void hash_out(struct tree tree) – выводит хеш-таблицу в консоль.

*Void add_leaf(FILE *file, struct tree *tree)* – добавление нового слова во все структуры хранения: ДДП, АВЛ-дерево, хеш-таблица, файл. Также выводит результаты эффективности добавления по времени и по памяти.

6. Результаты эффективности

Результатом эффективности данной программы является сравнение времени добавления, количества сравнений и затрачиваемой памяти при разных реализациях хранения дерева. Время указано в тактах.

Способ хранения	**Время**	**Кол-во сравнений**	**Память**
ДДП	11880	3	32
АВЛ-дерево	76960	4	32
Хеш-таблица	4136	1	24
Файл	40806	0	0

7. Выводы по проделанной работе

В ходе этой лабораторной удалось поработать с таким типом данных как дерево. Оно было реализовано с помощью 3 структур: ДДП, АВЛ-дерево, хеш-таблица.

На основе этих структур проводилось сравнение эффективности при добавлении элемента.

Результаты сравнительной эффективности добавления элемента в дерево показали, что намного эффективнее по времени использовать хеш-таблицу, однако при выборе плохой хеш-функции или большом количестве коллизий, процесс поиска значительно замедляется. При обычном поиске элемента также рационально использовать АВЛ-дерево, так как оно дает минимальное количество сравнений (особенно в случае неудачно хеш-функции).

Ответы на контрольные вопросы

1. Что такое дерево?

Дерево — нелинейная структура данных, используемая при представлении иерархических связей, имеющих отношение «один ко многим». Также деревом называется совокупность элементов, называемых узлами или вершинами, и отношений («родительских») между ними, образующих иерархическую структуру узлов.

2. Как выделяется память под представление дерева?

Деревья могут представляться как списком, так и массивом. При реализации списком соответственно память выделяется под каждый элемент (для значения, правого и левого потомков), при реализации массивом выделяется с запасом фиксированная длина. При этом размер массива выбирается исходя из максимально возможного количества уровней двоичного дерева, и чем менее полным является дерево, тем менее рационально используется память.

3. Какие стандартные операции возможны над деревьями?

-Обход по дереву (посещение вершин)
-Поиск по дереву
-Включение узла в дерево
-Удаление узла из дерева
-Перебалансировка (в случае балансированных деревьев)

4. Что такое дерево двоичного поиска?

Каждая вершина дерева двоичного поиска (ДДП) имеет значение, которое больше, чем содержание любой из вершин его левого поддерева и меньше, чем содержание любой из вершин его правого поддерева.

5. Чем отличается идеально сбалансированное дерево от АВЛ-дерева?

Критерий для АВЛ-дерева: дерево называется сбалансированным тогда и только тогда, когда высоты двух поддеревьев каждой из его вершин отличаются не более чем на единицу.
В случае идеально сбалансированного дерева не более чем на единицу должно отличаться число вершин в левом и правом поддеревьях.

6. Чем отличается поиск в АВЛ-дереве от поиска в ДДП?
Так как высоты поддеревьев АВЛ-дерева отличаются не более чем на 1, количество сравнений в таком дереве заметно уменьшается. ДДП же в худшем случае (дерево представляет из себя линейный список) имеет количество сравнений равное количеству элементов, что значительно замедляет процесс поиска.

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

Хеш-таблица — массив, заполненный в порядке, определенным хеш-функцией, т.е. это структура данных вида «ассоциативный массив», которая ассоциирует ключи со значениями.
Для каждого исходного элемента вычисляется значение хеш-функции, в соответствии с которым элемент записывается в определенную ячейку хеш-таблицы.

8. Что такое коллизия? Каковы методы их устранения?

Коллизия — совпадение хеш-адресов для разных ключей.
Устранение коллизий можно производить методом цепочек (также открытое хеширование). Его суть заключается в том, что каждая ячейка хеш-таблицы представляет собой связный список, содержащий все элементы, значение хеш-функции которых совпадает с текущим.
Также можно бороться с коллизиями методом закрытого хеширования. Хеш-таблица в данном случае представляет из себя обычный массив, который заполняется по такому принципу: если ячейка со значением хеш-функции свободна, туда записывается элемент, иначе проверяется другая ячейка. При этом адресация может быть линейной, квадратичной или произвольной.

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

Хеш-таблицы перестают быть эффективными при большом количестве коллизий. В таком случае количество сравнений будет значительно расти, независимо от способа их разрашения.

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

Нетрудно понять, что поиск в АВЛ-дереве намного эффективнее по времени, чем поиск в ДДП. Однако даже АВЛ проигрывает хеш-таблице в случае малого количества коллизий, так как в идеале количество сравнений в хеше будет равно 1. При большом количестве коллизий хеш-таблица может стать даже хуже ДДП.