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
Обработка деревьев

Вариант 0

Студент Шатохина Т.П.

Группа ИУ7 – 31Б

Преподаватель Барышникова М.Ю.

2023 г.

Условие задачи:

Программа строит бинарное дерево, хэш-таблицы с открытой и закрытой адресацией из букв введенной строки. Реализовать удаление повторяющихся букв из бинарного дерева, поиск во всех видах структур, реструктуризацию обоих видов хэш-таблиц(с открытой и закрытой адресацией). Провести замеры среднего количества сравнений для поиска элементов во всех структурах и памяти, занимаемой структурами.

Пункты меню:

(чтобы выполнить какую-то команду, после предложения программы ввести номер этой команды)

0. Искусственное завершение программы.
1. Удаляет все повторяющиеся вершины из дерева и обновляет файл с графиком,
учитывая последние изменения.
2. Перестраивает хэш-таблицу с закрытой адресацией с учетом наиболее удобного для нас размера.
3. Перестраивает хэш-таблицу с открытой адресацией с учетом наиболее удобного для нас размера.
4. Выводит хэш-таблицу с закрытой адресацией.
5. Выводит хэш-таблицу с открытой адресацией.
6. Ищет букву в дереве и выводит, при наличие ее сыновей.
7. Ищет букву в хэш-таблице с закрытой адресацией, выводит ее и количество ее повторений.
8. Ищет букву в хэш-таблице с открытой адресацией, выводит ее и количество ее повторений.

Техническое задание:

1) Программа запускается из терминала посредством запуска исполняемого файла.

2) **Входные данные:** при запуске программа просит пользователя ввести строку. Строка состоит из любых символов, но структуры данных обрабатывают только заглавные и строчные английские буквы.

3) **Выходные данные:** они зависят от команды, которая будет дана программе:
   0. Сообщение “Программа успешно завершена.”.
   1. Сообщение о том что в дереве нет узлов для вывода или дерево в формате png.

2. и 3. Выходных данных нет.
4. Хэш-таблица с закрытой адресацией в порядке, котором она хранится, сначала 1й адрес и все его узлы, потом 2й и так далее.
5. Хэш-таблица с открытой адресацией, при проходе по адресам последовательно.
6. Выводит узел дерева, который искали или сообщение, что такого узла нет.
7. Выводит строчку хэш-таблицы с закрытой адресацией, в ней содержится буква и количество ее повторений или сообщение о том, что совпадений не найдено.
8. Выводит строчку хэш-таблицы с открытой адресацией, в ней содержится буква и количество ее повторений или сообщение о том, что совпадений не найдено.

4) Если не удалось построить таблицу с открытой адресацией, выводится сообщение: “При попытке заполнить хэш-таблицу с открытой адресацией произошло переполнение. Таким образом пользование этой таблицей невозможно. Для дальнейшего использования ее нужно реструктурировать.”
5) Реструктцризация хэш-таблицы с закрытой адресацией происходит тогда, когда среднее количество сравнений при поиске больше либо равно 4. Реструктуризация меняет среднее значение сравнений и делает его меньше 4.
6) Реструктуризация хэш-таблицы с открытой адресацией происходит когда есть незадействованная память. Реструктуризация меняет среднее значение сравнений и делает его равным 1.

Структуры данных:

```
typedef struct tree
{
bool sign;
char name;
struct tree *left;
struct tree *right;
int height;
} tree_n;
```

Описывает узел дерева

```
typedef struct table_node
{
int amount;
char letter;
struct table_node *next;

} table_n;
```

Описывает узел хэш-таблицы с закрытой адресацией

```
typedef struct
{
int amount;
char letter;
} table_s;
```

Описывает узел хэш-таблицы с открытой адресацией

```
typedef struct
{
table_n **array;
table_s **array_simple;
int num;
int s_num;
} table_mas;
```

Структура описывает 2 массива на которых построены 2 хэш-таблицы, array – хэш-таблица с закрытой адресацией, array_simple – хэш-таблица с открытой адресацией. Num – количество неповторяющихся букв в 1й хэш-таблице, s_num – во второй.

Аварийные ситуации:

1. Ввод пустой строки в начале
2. Некорректный номер команды

Тесты:

Входные данные	Выходные данные
Ввод пустой строки	Завершение программы с кодом 6
Поиск отсутствующго символа	Сообщение “Совпадений не найдено.”
Строчка, некорректная для построения таблицы с открытой адресацией.	Сообщение “При попытке заполнить хэш-таблицу с открытой адресацией произошло переполнение. Таким образом пользование этой таблицей невозможно. Для дальнейшего использования ее нужно реструктурировать.”
Некорректный номер команды	Завершение программы с кодом 1

Ввод некорректной максимальной длины таблицы	Завершение программы с кодом 1
Перестройка хэш-таблицы с закрытой адресацией со средним количеством сравнений при поиске не большим чем 4 и с минимальным для такого случая использованием памяти	Таблица не перестраивается
Перестройка хэш-таблицы с закрытой адресацией со средним количеством сравнений большим чем 4	Перестроит таблицу так чтобы среднее количество сравнений было не больше 4.
Перестройка таблицы с открытой адресацией	Таблица перестраивается так чтобы занимать минимально возможное количество маста
Поиск в дереве, при имеющемся узле	Выводит узел дерева
Поиск в дереве при отсутствии узла	Выводит сообщение “Совпадений не найдено”
Поиск в пустом дереве	Сообщение “Дерево пустое.”
Поиск в хэш-таблице при имеющемся в ней значении	Количество повторений и буква
Поиск в хэш-таблице при отсутствии значения в ней	Сообщение “Совпадений не найдено.”
Вывод пустой хэш-таблицы	Сообщение “Хэш-таблица пустая.”
Вывод хэш-таблицы	Таблица в порядке обхода от начала

Замеры по памяти и количеству сравнений:

Хэш-таблицы

Возьмем постоянное количество уникальных элементов – 10

Если взять шаг хэш-кодов в хэш-таблице с открытой адресацией – 10 с тем же количеством элементов

Максимальная длина таблицы	Память в байтах для хэш-таблицы с закрытой	Количество сравнений для хэш-таблицы с закрытой	Память в байтах для хэш-таблицы с открытой	Количество сравнений для хэш-таблицы с открытой

	адресацией	адресацией	адресацией	адресацией
2	160	5	160	5
5	160	2	400	2
10	160	1	1600	1
52	160	1	4160	1

Если взять шаг хэш-кодов в хэш-таблице с открытой адресацией – 5 с тем же количеством элементов

Максимальная длина таблицы	Память в байтах для хэш-таблицы с закрытой адресацией	Количество сравнений для хэш-таблицы с закрытой адресацией	Память в байтах для хэш-таблицы с открытой адресацией	Количество сравнений для хэш-таблицы с открытой адресацией
2	160	5	80	7
5	160	2	200	4
10	160	1	400	2
52	160	1	2080	1

Количество сравнений в случае с открытой адресацией увеличивается потому что про сравнении мы заходим на индексы других хэш-кодов.

Поиск в дереве всегда занимает одинаковое среднее количество сравнений и оно равно log(n), где n – количество элементов.

В таблицах не приведено время потому что для моего варианта с возможной максимальной длиной таблиц – 52, любое количество сравнений будет занимать очень маленькое количество времени. Вывод о затраченном времени будет

сделан на основании количества сравнений, так как очевидно затраченное время зависит от этой величины.

Вывод: Количество памяти в хэш-таблице с закрытой адресацией и в деревьях зависит от количества элементов добавленных в эти структуры и среднее количество сравнений зависит также от количества добавленных элементов. В случае хэш-таблицы с открытой адресацией мы сначала выделяем кусок памяти, потом работаем с ним, но могут возникнуть ситуации с недостатком памяти для хранения данных. При отсутствии реструктуризации хэш-таблиц количество сравнений при поиске в них имеет одинаковое среднее значение. При реструктуризации обоих среднее значение количества сравнений при поиске в хэш-таблице с открытой адресацией становится меньше 4, а в случае открытой адресации это значение равно 1.

Контрольные вопросы:

1. *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.

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

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.

3. 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.

4. Что такое коллизии? 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.

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

Хеш-таблицы перестают быть эффективными при большом количестве коллизий.

В таком случае количество сравнений будет значительно расти, независимо от способа их разрашения.

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

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.