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)

Отчет

по лабораторной работе №6
по теме
«Деревья, хеш-таблицы»
Вариант 5.

Дисциплина: Типы и структуры данных

Студент ИУ7-31Б:
Косарев Алексей
Проверила:

Москва, 2021

1. Описание условия задачи

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

- Описание исходных данных

Исходными данными являются числовые данные, введенные пользователем, для выбора пунктов меню, а также описание графа (связи вершин).

Меню:

1. Ввод данных из файла в бинарное\АВЛ дерево
2. Вывод бинарного дерева
3. Вывод АВЛ дерева
4. Ввод данных из файла в хэш-таблицу
5. Вывод хэш-таблицы
6. Реструктуризация хэш-таблицы
7. Добавление слова в бинарное\АВЛ дерево и хэш-таблицу
8. Анализ

0. Выход из программы

Также на вход подается файл со словами.

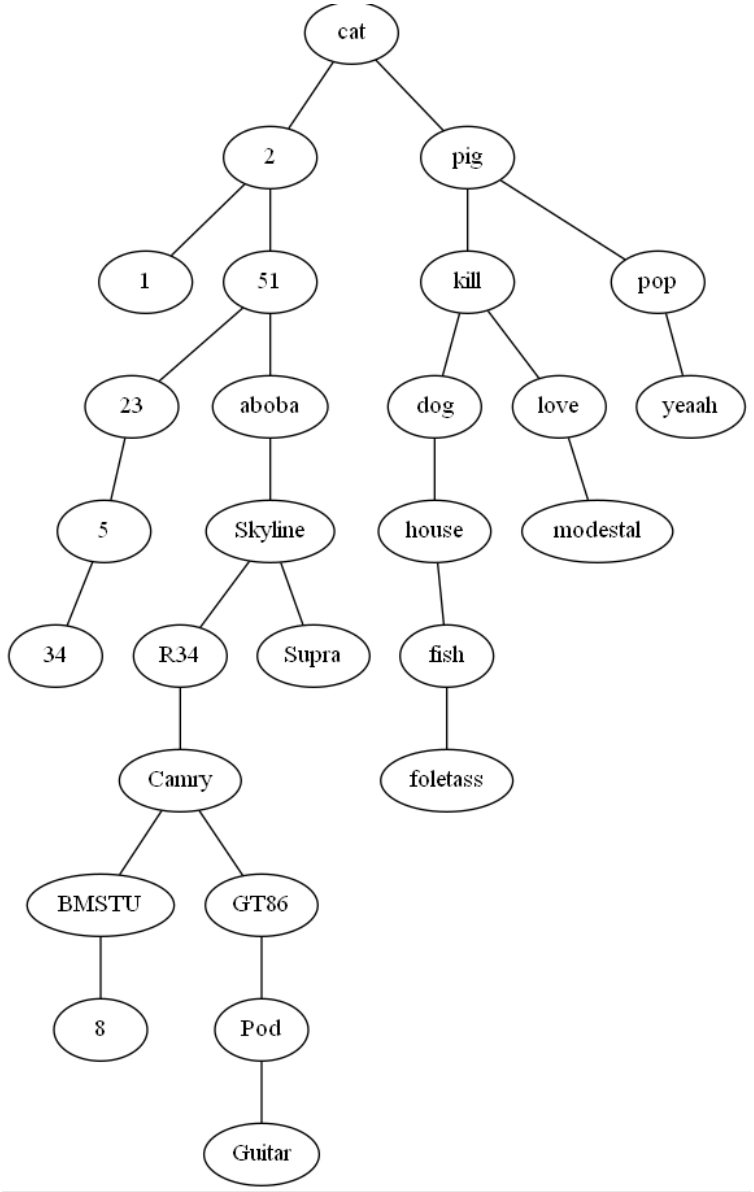
Пользователь вводит размерность хэш-таблицы и слово, которое необходимо добавить в деревья и хэш-таблицу.

При реструктуризации хэш-таблицы требуется ввести новое значение ее размерности и тогда создается новая хэш-таблица для всех слов, которые были в старой.

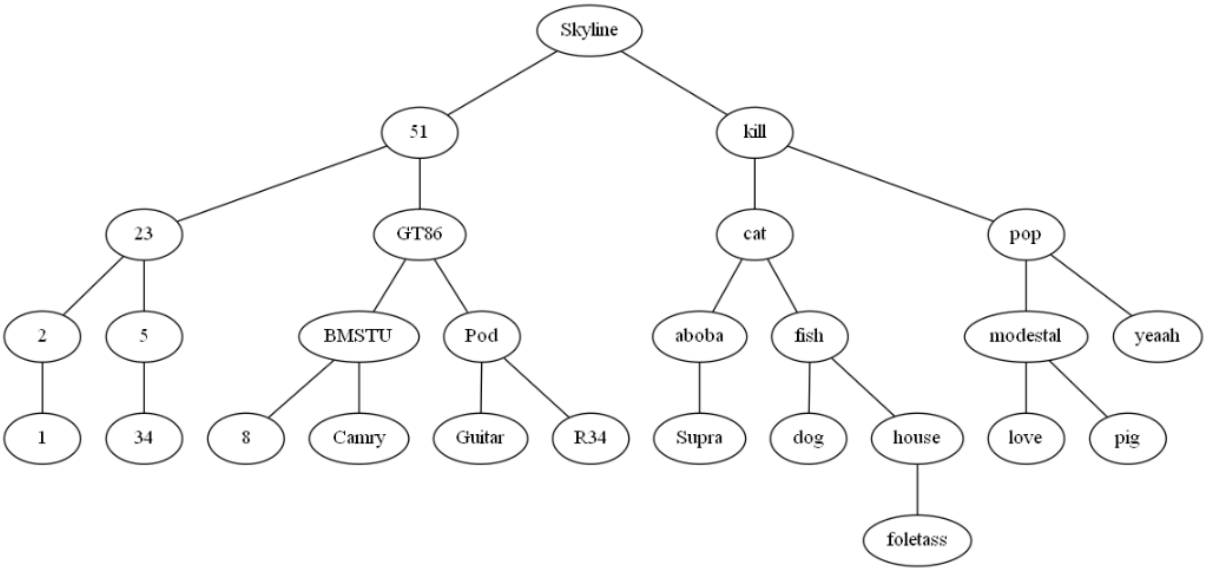
- Описание выходных данных

- DOT файл с информацией о бинарном дереве, из которого можно сделать png файл
- DOT файл с информацией об АВЛ дереве, из которого можно сделать png файл
- вывод хэш-таблицы
- анализ работы программы по времени и анализ памяти

- Пример работы (27 слов)
  Для бинарного дерева:



  АВЛ дерево:



Хэш-таблица размерности 22:

```
HASH table:
0:
1: GT86
2: Camry
3:
4: cat, Guitar
5: pop, 1
6: dog, 2
7: foletass
8: fish
9: 5, Skyline, R34
10: kill
11:
12: pig, 8
13: 23
14: 51, yeaah
15: 34
16:
17: aboba, Supra, Pod8
18:
19:
20: house, love
21: BMSTU, modestal
```

3. Описание задачи, реализуемой в программе

Программа реализует чтение слов из файла. На их основе она создает бинарное дерево, АВЛ дерево и хэш-таблицу. Для создания хэш-таблицы пользователь должен ввести ее размерность. Вывод деревьев происходит в формате экспорта в DOT файл для дальнейшего графического отображения, а хэш-таблица выводится в консоль.

Присутствует также возможность добавить введенное слово в бинарное и АВЛ дерево и в хэш-таблицу.

4. Способ обращения к программе

Обращение к программе происходит через консоль, путем запуска файла с расширением .exe (./main.exe) и указанием в параметрах командной строки названия файла со словами.

5. Описание возможных аварийных ситуаций и ошибок пользователя

```
// Коды ошибок
#define OK 0                        // Нет ошибок
#define INCORRECT_INPUT 1           // Некорректный ввод
#define EMPTY_STR 2                 // Пустая строка
#define FILE_OPEN_ERR 3             // Ошибка открытия файла
#define FILE_ERR 4                  // Ошибка данных файла
#define FILE_CLOSE_ERR 5            // Ошибка закрытия файла
#define MEMORY_ERR 6                // Ошибка выделения памяти
#define INCORRECT_ARGS 7            // Некорректные параметры командной
строки
#define EMPTY_FILE 8                // Пустой файл
```

Также при некорректном вводе (например, вводе буквы вместо размера хэш-таблицы или вводе несуществующего пункта меню) программа будет запрашивать ввод у пользователя, пока тот не введет корректное значение.

6. Описание внутренних структур данных

- Структура представления бинарного дерева:

```
typedef struct bin_tree_t bin_tree_t;

struct bin_tree_t
{
   char *word;          // Значение узла дерева
   bin_tree_t *left;    // Указатель на левое поддерево
   bin_tree_t *right;   // Указатель на правое поддерево
};
```

- Структура представления АВЛ дерева:

```
typedef struct avl_tree_t avl_tree_t;

struct avl_tree_t
{
   char *word;          // Значение узла дерева
   int height;          // Значение высоты дерева
   avl_tree_t *left;    // Указатель на левое поддерево
   avl_tree_t *right;   // Указатель на правое поддерево
};
```

- Структура представления хэш-таблицы:

```
typedef struct table_node_t table_node_t;

typedef struct
{
   table_node_t **table;  // Массив указателей на списки
   int size;              // Размер хэш-таблицы
} hash_table_t;

struct table_node_t
{
   char *word;            // Значение узла списка
   table_node_t *next;    // Указатель на следующий элемент списка
};
```

7. Алгоритмы

Алгоритм создания бинарного дерева:

1. Проверяем, есть ли корневой узел, если нет, то создаем новый узел с новым значением ключа.
2. Если уже есть корневой узел, то проверяем в какое поддерево перемещаться (если новое значение меньше значения корневого узла, то идем в левое поддерево, иначе в правое).
3. Переходим к пункту 1, пока не создадим новый узел или не определим, что такой уже существует.

Алгоритм создания АВЛ дерева:

1. Проверяем, есть ли корневой узел, если нет, то создаем новый узел с новым значением ключа.
2. Если уже есть корневой узел, то проверяем в какое поддерево перемещаться (если новое значение меньше значения корневого узла, то идем в левое поддерево, иначе в правое).

3. Переходим к пункту 1, пока не создадим новый узел или не определим, что такой уже существует.
4. После того, как ключ вставлен либо в правое, либо в левое поддерево, производим балансировку текущего узла.

Алгоритм балансировки для трех узлов:

1. Выбираем из трех вершин ту, которая имеет медианное значение ключа. Данная вершина будет новым узлом-родителем.
2. Узел с наименьшим значением ключа будет его левым потомком.
3. Узел с наибольшим значением ключа будет его правым потомком.
4. Прежних потомков нового узла-родителя нужно разместить так:
   Левое поддерево (если оно было и не равно связи с новыми потомками) становится правым поддеревом нового левого узла.
   Правое поддерево (если оно было и не равно связи с новыми потомками) становится левым поддеревом нового правого узла.

8. Тесты

Положительные тесты:

1. В файле только слова
2. В файле слова и числа
3. В файле все слова одинаковые
4. Добавление несуществующего в деревьях и хэш-таблице слова

Негативные тесты:

1. Несуществующий файл
2. Пустой файл
3. Некорректное количество параметров командной строки
4. Некорректный выбор пункта меню (ввод буквы).
5. Некорректный выбор пункта меню (ввод несуществующего номера пункта).
6. Некорректный ввод размерности хэш-таблицы (ввод буквы).
7. Некорректный ввод размерности хэш-таблицы (ввод отрицательного значения).

9. Временная эффективность и затраты памяти

Количество слов файле - 100. Размерность хэш-таблицы - 75

Structure	Create time, us	Comparisons while creating	Add time, ns	Comparisons while adding	Memory, bytes	Average number of comparisons
bin tree	0.048000	658	274	10	2400	7
avl tree	0.060000	536	343	7	3200	5
hash table	0.044000	78	140	0	1600	1
file	-	-	78980	-	638	-

Хэш-функция: (сумма кодов всех символов строки) % (размерность хэш-таблицы).

Время создания АВЛ дерева на 25% медленнее создания бинарного дерева из-за процессов балансировки, а время добавления элемента в АВЛ на 25% медленнее, чем в бинарное. При этом бинарное дерево экономит больше памяти за счет того, что в структуре узла бинарного дерева нет переменной для хранения высоты узла, в отличие от АВЛ дерева (бинарное занимает на 25% меньше памяти). Но среднее количество сравнений при поиске элементов при использовании АВЛ дерева на 40% меньше.

Хэш-таблица эффективнее бинарного дерева на 50% по памяти, а АВЛ дерева в 2 раза. При размерности хэш-таблицы равной 75% от количества элементов хэш-таблица создается быстрее, чем бинарное дерево на 10% и чем АВЛ дерево на 36%. По количеству сравнений хэш-таблица также выигрывает: она эффективнее в 5 раз, чем АВЛ дерево и в 7 раз, чем бинарное дерево.

Время добавления в файл самое большое, так как каждый раз приходится его открывать и закрывать. Добавление в файл медленнее в 288 раз, чем добавление в бинарное дерево и в 230 раз, чем добавление в АВЛ и в хэш-таблицу. Но файл эффективнее других структур по памяти: почти в 4 раза меньше, чем бинарное дерево; в 5 раз меньше, чем АВЛ дерево; в 2.5 раза меньше, чем хэш-таблица.

10. Вывод

В процессе выполнения данной лабораторной работы я научился работать с бинарными и АВЛ деревьями и хэш-таблицами.

Для представления деревьев в памяти я использовал списки, а для хэш-таблицы - массив указателей на списки (метод цепочек).

При анализе работы создания структур и добавления элементов в эти структуры, стало ясно, что при правильно подобранной хэш-функции хэш-таблица является наиболее эффективной структурой для добавления элемента.

11. Ответы на 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?
   Деревья можно представить в виде:

- связей с предками (в таком случае можно использовать статический или динамический массивы);
- связного списка потомков (массив односвязных списков, то есть память выделяется под каждого потомка)
- ДДП, память выделяется под каждую вершину дерева

3. Какие стандартные операции возможны над деревьями?
   Основные операции над деревьями: обход дерева, добавление, удаление, поиск элемента дерева.

4. Что такое дерево двоичного поиска?
   ДДП – это дерево, каждая вершина которого имеет не более двух потомков.

5. Чем отличается идеально сбалансированное дерево от АВЛ дерева?
   Идеально сбалансированное дерево – это дерево, у которого число вершин в левом и правом поддеревьях отличается не более, чем на 1, а АВЛ дерево – это дерево, у каждого узла которого высота двух поддеревьев отличается не более чем на единицу.

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

7. Что такое хеш-таблица, каков принцип ее построения?
   Массив, заполненный в порядке, определенным хеш-функцией, называется хеш-таблицей. Хеш-функция – это функция, по которой можно определять индекс элемента массива, в котором хранится информация.

8. Что такое коллизии? Каковы методы их устранения.
   Ситуация, когда разным ключам соответствует одно значение хеш-функции, то есть, когда $h(K1)=h(K2)$, в то время как $K1 \neq K2$, называется коллизией.
   Методы устранения коллизий:
   – внешнее (открытое) хеширование (метод цепочек);
   – внутреннее (закрытое) хеширование (открытая адресация)

9. В каком случае поиск в хеш-таблицах становится неэффективен?
   Если для поиска элемента необходимо более 3–4 сравнений, то эффективность использования такой хеш-таблицы пропадает и ее следует реструктуризировать (т.е. найти другую хеш-функцию), чтобы минимизировать количество сравнений для поиска элемента.

10. Эффективность поиска в АВЛ деревьях, в дереве двоичного поиска и в хеш-таблицах.
    Время поиска элемента в АВЛ дереве гораздо эффективнее, чем в ДДП дереве, так как требуемое количество сравнений в сбалансированном дереве равно высоте этого дерева в худшем случае. Но при этом АВЛ дерево уступает хеш-таблице (при условии хорошо подобранной хеш-функции), так как минимальная трудоемкость поиска в хеш-таблице равна $O(1)$.