Министерство образования и науки 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 (ИУ)

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ7)

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6
ДЕРЕВЬЯ, ХЕШ-ТАБЛИЦЫ
ДИСЦИПЛИНА: ТИПЫ И СТРУКТУРЫ ДАННЫХ

Студент, группа **Боренко А. ИУ7-32Б**

2020 г.

Цель работы

Построить дерево, вывести его на экран в виде дерева, реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов, сбалансировать дерево, сравнить эффективность алгоритмов сортировки и поиска в зависимости от высоты деревьев и степени их ветвления; построить хеш-таблицу и вывести ее на экран, устранить коллизии, если они достигли указанного предела, выбрав другую хеш-функцию и реструктуризировав таблицу; сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска (ДДП), в хеш-таблицах и в файлах. Сравнить эффективность реструктуризации таблицы для устранения коллизий и поиска в ней с эффективностью поиска в исходной таблице.

Описание условия задачи

Вариант 6

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.

Описание Технического Задания

Описание исходных данных и результатов

Исходные данные:

Файл

Реультаты:

Дерево двочиного поиска в файле dot, дерево АВЛ в файле dot, хеш-таблица.

Описание задачи, реализуемой программой

Выбор пользователя. Создание ДДП, АВЛ, хеш-таблицы. Вывод ДДП,АВЛ, хеш-таблицы. Добавление в ДДП, АВЛ, хеш-таблицу. Сравнение добавления в ДДП и АВЛ. Сравнение добавления в ДДП, АВЛ, хеш-таблицу и файл.

Способ обращения к программе

Обращение к программе происходит через терминал. В командную строку вводится «./app.exe».

Описание возможных аварийных ситуаций и ошибок пользователя

1. Ввод не существующего файла

2. Выбор не из меню
3. Некорректный ввод ранг таблицы
4. Некорректный ввод файла

Описание внутренних структур данных

Дерево двоичного поиска и АВЛ

```
struct tree_node
{
	//tree_node_t *parent;
	tree_node_t *right;
	tree_node_t *left;
	int height;
	char *word;
};
```

Хеш-таблица

```
struct hash_element_node_t
{
	char *data;
	hash_element_node_t *next;
};

struct hash_table_t
{
	hash_element_node_t **array;
	int len;
};
```

Вспомогательные структуры данных для обработки графа

```
struct data_node
{
	char *data;
	data_node_t *next;
};
```

Описание алгоритма

Общий алгоритм

1. Считать имя файла, введенное пользователем.
2. Вывести меню
3. Пока ввод пользователя корректен, выполнить действие, указанное пользователем
4. Выйти

Алгоритм Добавления в ДДП

1. Получить корень и новое слово
2. Если корень больше нового слова
    1. Если существует левый элемент идти вправо
    2. Иначе добавить лист с новым словом
3. Если корень меньше нового слова
    1. Если существует правый элемент идти вправо
    2. Иначе добавить лист с новым словом
4. Если корень равен новому слову
    1. Выйти с кодом ошибки

Алгоритм Добавления в АВЛ

5. Получить корень и новое слово
6. Если корень больше нового слова
    1. Если существует левый элемент идти вправо
    2. Иначе добавить лист с новым словом
7. Если корень меньше нового слова
    1. Если существует правый элемент идти вправо
    2. Иначе добавить лист с новым словом
8. Если корень равен новому слову
    1. Выйти с кодом ошибки
9. сбалансировать корень

Алгоритм Добавления в Хеш-таблицу

10. Получить таблицу и новое слово
11. Вычислить хеш с помощью хеш-функции

12. Если в таблице под хеш-ключем что-то есть
    1. Перебрать элементы цепочки до последнего
    2. Добавить в цепочку новый элемент
13. Иначе
    1. Добавить элемент в хеш-таблицу

Хеш-функция

Функция берущая остаток от деления числа на ранг хеш-таблицы.

```
int hash_fuction(char *str, int m)
{
	int p = 'z' + 1 - 'a';
	unsigned long long int p_pow = 1;
	unsigned long long hash = 0;
	//printf("m = %d\n", m);
	for (int i=0; i < strlen(str); ++i)
	{
		//printf("previous hash = %lld\n", hash);
		hash += (str[i] - 'a' + 1) * p_pow % m;
		//printf("str[i] = %c, p = %d, hash_cur = %lld, %d, %d, %lld\n",
str[i], p, (str[i] - 'a' + 1) * p_pow % m, str[i], 'a', p_pow);
		//printf("hash = %lld\n", hash);
		p_pow *= p;
	}
	return hash % m;
}
```

Алгоритм реструкторизации

1. Узнать размер данных, по которым строится хеш-таблица.
2. Удалить старую хеш-таблицу
3. Создать новую, размер которой равен размеру данных.

Основные функции

Работа с ДДП

```
int create_tree_node(tree_node_t **new_node, char *word);
void free_tree_node(tree_node_t **root);
void free_tree(tree_node_t **root);
void find_near_node(tree_node_t **root, char *element, tree_node_t
**return_node);
int add_tree_node(tree_node_t **root, char *element);
void make_tree_from_data(tree_node_t **root, data_node_t *array);
// OUTPUT
void output_tree_as_dot(tree_node_t *root, char *filename);
// С подсчетом сравнений
int add_tree_node_counter(tree_node_t **root, char *element, int *i);
```

Работа с АВЛ

```
int fix_height(tree_node_t **root);
int get_balance_factor(tree_node_t *node);
void right_rotate(tree_node_t **root);
void left_rotate(tree_node_t **root);
void balance(tree_node_t **root);
void make_by_add_balanced_tree_from_data(tree_node_t **root, data_node_t *data);
int add_balance_node(tree_node_t **root, char *element);
// С подсчетом сравнений
int add_balance_node_counter(tree_node_t **root, char *element, int *i);
```

Работа с хеш-таблицей

```
int hash_fuction(char *str, int m);

int make_hash_table_from_data(hash_table_t *hash_table, data_node_t *data, int
m);
void output_hash_table(hash_table_t hash_table);
void free_hash_table(hash_table_t *hash_table);
int add_element_in_hash_table(hash_table_t *hash_table, char *data);
int create_hash_table(hash_table_t *hash_table, int m);
int create_hash_element_node(hash_element_node_t **res, char *data);
// С подсчетом сравнений
int add_element_hash_counter(hash_table_t *hash_table, char *data, int *i);
```

Тестирование

```
void testing_trees_time(char *test_filename);
void testing_all(int i, char *test_filename);
```

Тесты

Негативные

Некорректный файл

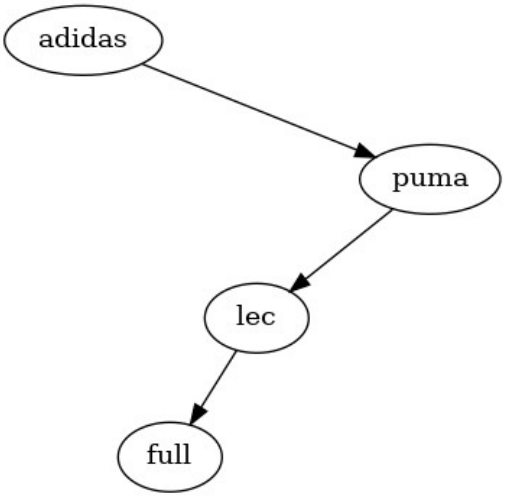
Отсутствующий файл

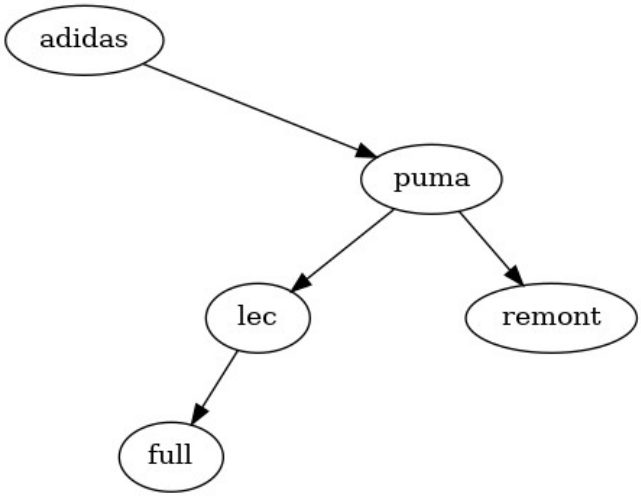
Некорректный выбор меню — выход

Некорректный ранг хеш-таблицы — выход в главное меню

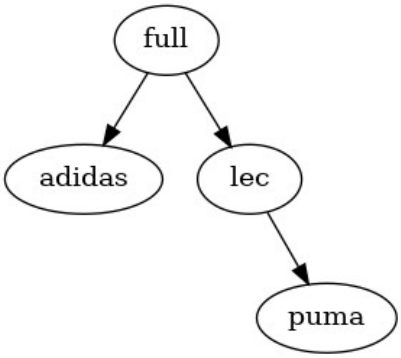
Позитивные

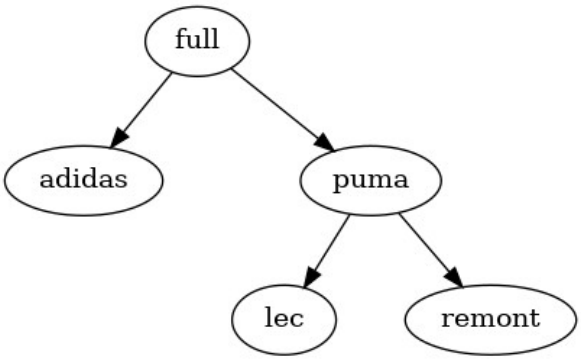
Вывод ДДП, Добавление в ДДП





Вывод АВЛ, Добавление в АВЛ





Вывод хеш-таблицы

```
|  key|              values
|    0|full
|    1|adidas
|    2|puma lec
|    3|
```

Добавление в хеш-таблицу

```
|  key|              values
|    0|full remont
|    1|adidas
|    2|puma lec
|    3|
```

Сравнение ДДП и АВЛ

```
Результат теста (тест проводился на файле test_trees_time.txt):
|                №|              ДДП|              АВЛ|
|             time|               13|               21|
|           memory|             1368|             1520|
```

Сравнение ДДП, АВЛ, хеш-таблицы и файла.

```
Результат теста (тест проводился на файле test_trees_time.txt):
|          №|    ДДП|    АВЛ|    Хеш|   Файл|
|       time|     17|     23|      9|     12|
|     memory|   1368|   1520|     16|    ---|
| comparing|8.63157|3.97368|0.05263|    ---|
Добавлено 38 слов(а).
```

Выводы о проделанной работе

```
Результат теста (тест проводился на файле test_trees_time.txt):
|                №|              ДДП|              АВЛ|
|             time|                13|                21|
|           memory|              1368|              1520|

Результат теста (тест проводился на файле test_trees_time.txt):
|          №|    ДДП|    АВЛ|    Хеш|   Файл|
|      time|     17|     23|      9|     12|
|    memory|   1368|   1520|     16|    ---|
| comparing|8.63157|3.97368|0.05263|    ---|
Добавлено 38 слов(а).

Результат теста (тест проводился на файле testing_data.txt):
|          №|    ДДП|    АВЛ|    Хеш|   Файл|
|      time|    357|    622|    167|     60|
|    memory|  36000|  40000|  16000|    ---|
| comparing|10.92800|8.53400|1.14900|    ---|
Добавлено 1000 слов(а).
(Среди слов были повторяющиеся)
```

(Время в мсек)

Добавление в файл самое быстрое, т. к. оно не подразумевает никакой упорядоченности. Найти в файле потому будет очень сложно.

Хеш-таблица наиболее выгодна, как и ожидалось.

Добавление в ДДП быстрее чем в АВЛ, т. к. не требуется балансировка.

Ддля деревьев добавление эффективней в ДДП, поскольку не требует дополнительной балансировки.

Ответы на вопросы

1. Что такое дерево?

Дерево — это абстрактный тип данных, связный ациклический граф.

2. Как выделяется память под представление деревьев?

Выделяется динамическая память под листья.

3. Какие стандартные операции возможны над деревьями?

Добавление, удаление листа, поиск.

4. Что такое дерево двоичного поиска?

Дерево двоичного поиска — это двоичное дерево (у одного листа не может быть больше 2 потомков), для которого выполняются следующие дополнительные условия (*свойства дерева поиска*):

- Оба поддерева — левое и правое — являются двоичными деревьями поиска.
- У всех узлов *левого* поддерева произвольного узла X значения ключей данных *меньше либо равны*, нежели значение ключа данных самого узла X.
- У всех узлов *правого* поддерева произвольного узла X значения ключей данных *больше либо равны*, нежели значение ключа данных самого узла X.

5. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

Идеально сбалансированное дерево — дерево, для каждой вершины которого размеры левого и правого поддеревьев отличаются не более чем на 1.

АВЛ дерево — это дерево, у которого высоты двух поддеревьев каждой из его вершин отличаются не более чем на единицу.

Идеально сбалансированное дерево имеет в среднем меньшее кол-во сравнений для всех своих вершин, чем АВЛ-дерево.

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

В АВЛ поиск обычно происходит быстрее, потому что глубина дерева меньше, чем ДДП, для тех же данных.

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

Хеш-таблица - ассоциативный массив, в который можно выполнить добавление, удаление, поиск данных. Особенность хеш-таблицы — это возможность найти ключ с помощью хеш-функции.

8. Что такое коллизии? Каковы методы их устранения.

Коллизия (конфликт) — ситуация: один хеш–адрес для разных ключей.

Метод цепочек

- Данные хранятся не в самой таблице, а в связных списках. В таблице – ссылка на первый элемент списка
- Создается цепочка (список) элементов с одним ключом.
- Поиск в списке осуществляется простым перебором, т.к.при грамотном выборе хеш-функции любой из списков оказывается достаточно коротким.

Закрытое хеширование

При добавлении проверяется занята ли ячейка, и если занята, то данных заносятся в следущую пустую ячейку.

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

В случае если коллизий слишком много. Обычно это происходит, если ранг таблицы значительно меньше количества занесенных данных.

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

Наиболее эффективен поиск в хеш-таблицах с правильно подобранным рангом и хеш-функцией. Поиск в АВЛ деревьях немного эффективнее поиска в ДДП.