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 «ДЕРЕВЬЯ, СБАЛАНСИРОВАННЫЕ ДЕРЕВЬЯ, ХЕШ-ТАБЛИЦЫ»

Студент Тузов Даниил Александрович

Группа ИУ7 – 32Б

Преподаватель Барышникова Марина Юрьевна
Силантьева Александра Васильевна

2023 г.

Оглавление

ОПИСАНИЕ УСЛОВИЯ ЗАДАЧИ

Построить хеш-таблицу по указанным данным. Сравнить эффективность поиска в сбалансированном двоичном дереве, в двоичном дереве поиска и в хеш-таблице (используя открытую и 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 по памяти) для поставленной задачи. Оценить эффективность поиска в хеш-таблице при различном количестве коллизий и при различных методах их разрешения.

Построить хеш-таблицу для зарезервированных слов языка C++ (не менее 20 слов), содержащую HELP для каждого слова. Выдать на экран подсказку по введенному слову. Выполнить программу для различных размерностей таблицы и сравнить время поиска и количество сравнений. Для указанных данных создать сбалансированное дерево. Добавить подсказку по вновь введенному слову, используя при необходимости реструктуризацию таблицы. Сравнить эффективность добавления ключа в таблицу или ее реструктуризацию для различной степени заполненности таблицы.

ОПИСАНИЕ ТЕХНИЧЕСКОГО ЗАДАНИЯ

Входные данные:
Файл с зарезервированными словами и их описаниями, ключ и значения для вставки, ключ для поиска, файл для вывода

Выходные данные:
DOT-файлы с полученными деревьями, файл с хеш-таблицей, время поиска элемента

Меню:
1 — Вставка в ДДП
2 — Вставка в АВЛ
3 — Вставка в хеш-таблицу
4 — Вывод ДДП в файл
5 — Вывод АВЛ в файл

6 — Вывод хеш-таблицы в dot-файл
7 — Поиск в ДДП
8 — Поиск в АВЛ
9 — Поиск в хеш-таблице
0 — завершение программы

Описание программы:
Программа имитирует работу с двоичным деревом поиска, с AVL-деревом и с хеш-таблицей на примере поиска элемента.

Способ обращения к программе:
Взаимодействие через консоль.

Аварийные ситуации:

1. Ошибка ввода
   Код ошибки — 1
2. Ошибка памяти
   Код ошибки — 2
3. Ошибка файла
   Код ошибки — 3
4. Ошибка вставки в дерево
   Код ошибки — 4

ОПИСАНИЕ СТРУКТУР ДАННЫХ

Описание пары слово-его описание
word – зарезервированное слово
description – описание этого зарезервированного слова

```
typedef struct type_item
{
char *word;
char *description;
} item_t;
```

Описание узла двоичного дерева поиска
item – пара, находящаяся в узле
left – указатель на левое поддерево (меньшее)
right – указатель на правое поддерево (правое)

```
typedef struct type_tree
{
```

```
item_t *item;
struct type_tree *left;
struct type_tree *right;
} tree_t;
```

Описание узла AVL-дерева
item – пара, находящаяся в узле
depth – глубина вершины
left – указатель на левое поддерево (меньшее)
right – указатель на правое поддерево (правое)

```
typedef struct type_avl
{
item_t *item;
int depth;
struct type_avl *left;
struct type_avl *right;
} avl_t;
```

Описание списка ключей с равными индексами
item – пара, хранящаяся в узле
next – указатель на следующий элемент

```
typedef struct type_list
{
item_t *item;
struct type_list *next;
} list_t;
```

Описание хеш-таблицы
size – размер таблицы
count – количество элементов в таблице
overflow – список элементов с совпавшими индексами
items – массив элементов, расположенных в соответсвующих индексах

```
typedef struct type_hash
{
int size;
int count;
list_t **overflow;
item_t **items;
} hash_t;
```

ОПИСАНИЕ АЛГОРИТМА

1. Пользователь выбирает действие

2. Согласно выбранному действию вызывается соответствующая функция
    1. Вставка в АВЛ И ДДП производится рекурсивно в лист, вставка в хеш-таблицу производится в ячейку таблицы равную сумме символов строки по модулю размера строки
    2. Поиск в АВЛ и ДДП осуществляется рекурсивно, в хеш-таблице по индексу, соответствующему хеш-функции
3. Выполняется логика этой функции
4. Если во время выполнения функции возникла ошибка, то выводится сообщение, поясняющее эту ошибку. Программа аварийно завершается
5. Если ошибки не возникло, возвращаемся к 1 пункту

ОПИСАНИЕ ФУНКЦИЙ

`void discription();` - функция выводит описание программы

`void menu();` - функция выводит меню пользователю

`item_t *item_create(void);` - функция создания пары ключ-значение
`int item_input(FILE *f, item_t *item);` - ввод пары из файла

`void item_free(item_t *item);` - очистка памяти из-под пары

`int item_cmp(item_t *pl, item_t *pr);` - сравенение двух пар по ключу

`list_t *allocate_list(void);` - создание узла списка для хеш-таблицы

`list_t *list_insert(list_t *list, item_t *item);` - вставка в список для хеш-таблицы

`void list_free(list_t *list);` - очистка памяти из-под узла списка

`list_t **create_overflow(hash_t *table);` - создание списка элементов с повторяющимися индексами

`void free_overflow(hash_t *table);` - очистка памяти из-под списка

`int hash_func(char *str);` - хеш-функция

`hash_t *hash_create(int size);` - создание объекта хеш-таблица

`void hash_free(hash_t *table);` - очистка памяти из-под хеш-таблицы

`void hash_collision(hash_t *table, int index, item_t *item);` - функция разрешения коллизий при вставке

`int hash_insert(hash_t *table, item_t *item);` - вставка в хе-таблицу

`char *hash_find(hash_t *table, char *word);` - поиск в хеш-таблице

`void hash_print(FILE *f, hash_t *table);` - печать хеш-таблицы

`int read_avl(FILE *f, avl_t **avl);` - чтение авл-дерева из файла

`int read_tree(FILE *f, tree_t **tree);` - чтение бинарного дерева из файла

`int read_hash(FILE *f, hash_t **tree);` - чтение хеш-таблицы из дерева

`int read_str(char **buf);` - чтение строки

`tree_t *tree_create(item_t *item);` - создание объекта двоичное дерево поиска на основе пары

`void tree_free(tree_t *pt);` - очистка памяти из-под узла двоичного дерева

`void tree_destroy(tree_t *pt);` - удаление объекта двоичное дерево поиска

`void tree_print(FILE *f, tree_t *pt);` - печать двоичного дерева в файл

`void tree_print_dot(FILE *f, tree_t *pt);` - экспорт двоичного дерева в дот-файл

`tree_t *tree_find(tree_t *pt, tree_t *el, int (*comp)(item_t *, item_t *));` - поиск в двоичном дереве поиска

`tree_t *tree_insert(tree_t *pt, tree_t *el, int (*comp)(item_t *, item_t *));` - вставка в двоичное дерево поиска

`avl_t *avl_create(item_t *item);` - создание объекта авл-дерево на основе пары

`void avl_free(avl_t *pt);` - очистка памяти из-под узла авл-дерева

`void avl_destroy(avl_t *pt);` - уничтожение авл-дерева

`void avl_print(FILE *f, avl_t *pt);` - печать авл-дерева в файл

`void avl_print_dot(FILE *f, avl_t *pt);` - экспорт авл-дерева в дот-файл

`avl_t *avl_find(avl_t *pt, avl_t *el, int (*comp)(item_t *, item_t *));` - поиск в авл-дереве

`avl_t *avl_insert(avl_t *pt, avl_t *el, int (*comp)(item_t *, item_t *));` - вставка в авл-дерево

НАБОР ТЕСТОВ

№	Описание теста	Ввод пользователя	Ответ программы
1	Выбор действия	0	Выход из программы
2	Выбор действия	1	Вставка в ДДП
3	Выбор действия	2	Вставка в АВЛ
4	Выбор действия	3	Вставка в хеш-таблицу
5	Выбор действия	4	Экспорт ДДП в дот-файл

6	Выбор действия	5	Экспорт АВЛ в дот-файл
7	Выбор действия	6	Печать хеш-таблицы на экран
8	Выбор действия	7	Поиск в ДДП
9	Выбор действия	8	Поиск в АВЛ
10	Выбор действия	9	Поиск в хеш-таблице
11	Ввод файла для экспорта	a.dot	Успешно экспортировано
12	Ввод файла для экспорта	Поврежденный файл	Ошибка файла
13	Ввод ключа для вставки	for	Введите описание
14	Ввод описания для вставки	Цикл с итерацией	For - цикл с итерацией в 14 ячейку таблицы
15	Ввод существующего ключа для поиска	if	Ветка правды в условном операторе
16	Ввод несуществующего ключа для поиска	abc	Ничего не происходит

ЗАМЕРЫ ВРЕМЕНИ ПОИСКА В ОПИСАННЫХ СТРУКТУРАХ

В ходе эксперимента для получения результатов проводилось 1000000 замеров по времени. Усредненный результат представлен в таблице

Числа в таблице представлены в **микросекундах**, ЛС – лучший случай (в среднем 0.12 сравнений, размер таблицы 50), ХС – худший случай (в среднем 5.37 сравнений, размер таблицы равен количеству элементов в таблице и равен 25)

	ДДП	АВЛ	Хеш-открытая	Хеш-закрытая
Время	0.043	0.031	0.024 – ЛС 0.051 – ХС	0.04 – ЛС 0.06 – ХС

ЗАМЕРЫ РАСХОДОВ ПАМЯТИ В ОПИСАННЫХ СТРУКТУРАХ

Числа в таблицах представлены в **байтах**

	ДДП	АВЛ	Хеш
Память	24	28	24

ВЫВОД

В ходе лабораторной работы я познакомился с такими структурами данных, как АВЛ-деревья и Хеш-таблицы. А также сравнил их скорость реализации на примере поиска значения по ключу. Я выяснил:

- Поиск в АВЛ-дереве почти всегда быстрее поиска в ДДП, потому что в случае АВЛ-дерева высота = log(n), а в случае ДДП больше либо равна log(n), соответственно в среднем АВЛ-дерево работает на 28% быстрее
- В случае, когда хеш-таблица имеет небольшое количество коллизий (в среднем 0.12), она превосходит по скорости АВЛ-дерево, потому что в таком случае поиск осуществляется в среднем за O(1)
- В случае, когда хеш-таблица (25 элементов) имеет большое количество коллизий (в среднем 5.37), она проигрывает даже ДДП, потому что в таком случае элемент ищется в среднем за 5 сравнений, тогда как в АВЛ и ДДП примерно за log25=4.6 сравнения

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

В ИСД количество вершин в поддеревьях отличается не более, чем на 1

В АВЛ высоты поддеревьев отличаются не боле, чем на 1

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

Ничем

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

Таблица, построенная на хеш-функции — хеш-таблица

Принцип построения: хеш-функция определяет индекс элемента в таблице, в этот индекс записывается сам элемент (ключ-значение). Соответственно по ключу можно за O(1) получить значение

4. Что такое коллизии? Каковы методы их устранения.

Коллизия возникает, когда одному индексу соответствует несколько элементов. Методы устранения: метод цепочек (список), метод линейной адресации (ищется следующая незаполненная ячейка)

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

В случае, когда появляется много коллизий

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

Хеш-таблица наиболее эффективна при небольшом количестве коллизий, поиск осуществляется в среднем за O(1) операций. Далее идет АВЛ-дерево, потому что поиск и в лучшем, и в худшем случае составляет логарифм от высоты дерева. ДДП менее эффективны, чем АВЛ-деревья, так как поиск в них в лучшем случае — двоичный логарифм от высоты, а в худшем случае аж O(n), где n — количество вершин