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
КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

ОТЧЕТ ПО УЧЕБНОЙ ДИСЦИПЛИНЕ “ТИПЫ И СТРУКТУРЫ ДАННЫХ” ЛАБОРАТОРНАЯ РАБОТА №6

Студент Ляпина Наталья Викторовна
фамилия, имя, отчество

Группа ИУ7-32Б

Вариант 0

Студент _____ Ляпина Н.В.
подпись, дата *фамилия, и.о.*

Преподаватель _____ Силантьева А.В.
подпись, дата *фамилия, и.о.*

2021 г.

Оглавление

Задание

1. **Общее задание**
   построить ДДП, сбалансированное двоичное дерево (АВЛ) и хеш-таблицу по указанным данным. Сравнить эффективность поиска в ДДП в АВЛ дереве и в хеш-таблице (используя открытую или 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
2. **Задание по варианту**

   В текстовом файле содержатся целые числа. Построить ДДП из чисел файла. Вывести его на экран в виде дерева. Сбалансировать полученное дерево и вывести его на экран. Построить хеш-таблицу из чисел файла. Использовать метод цепочек для устранения коллизий. Осуществить поиск введенного целого числа в ДДП, в сбалансированном дереве, в хеш-таблице и в файле. Сравнить время поиска, объем памяти и количество сравнений при использовании различных (4-х) структур данных. Если количество сравнений в хеш-таблице больше указанного (вводить), то произвести реструктуризацию таблицы, выбрав другую функцию.

3. **Входные данные**

- Команда для выбора действия (от 0 до 5)
  1. Печать бинарного дерева
  2. Печать сбалансированного бинарного дерева
  3. Печать хеш-таблицы
  4. Поиск указанного числа
  5. Сравнение операции поиска по времени и по памяти
  0. Выход из программы

- Команда для выбора структуры данных
- Элемент для поиска
- Максимальное количество сравнений в хеш-таблице

- **Требования к входным данным**

1) Валиндая команда
2) Количество сравнений больше 0
3) Число для поиска типа int

4. Выходные данные

- бинарное дерево
- сбалансированное бинарное дерево
- хеш-таблица
- информация о сравнении
- информация о поиске

5. Действие программы

Программа выполняет обработку данных из файла с помощью 3 типов данных

5. Обращение к программе

Программа может быть вызвана через консоль компилятора с помощью команды “./app.exe”

6. Аварийные ситуации

В случае аварийной ситуации выводится сообщение об определенной ошибке.

Неверный ввод:

- Ошибка при вводе команды
- Ошибка при вводе элемента для поиска
- Ошибка при вводе количества сравнений

Структура данных

<u>Элемент дерева</u>

```
typedef struct tree_node
{
        int num;
        tree_node_t *left;
        tree_node_t *right;
} tree_node_t;
```

num - данные узла
left - указатель на левый узел
right - указатель на правый узел

<u>Элемент хеш-таблицы</u>

```
typedef struct
{
        int value;
        struct table_node *next;
```

} table_node_t;

value - значение узла
next - указатель на следующий элемент, если они расположены по одному индексу

Элемент списка
typedef struct
{
table_node_t **table;
int size;
int hash;
} hash_table_t;

table - непосредственно хеш-таблица
size - количество элементов в хеш-таблице
hash - простое число (хеш-функция)

Интерфейс модулей

Для обработки типа tree_node_t используются функции:

```
tree_node_t *create_node(const int elem)
// Функция создания элементов дерева
// elem - значение элемента

tree_node_t *insert_tree(tree_node_t *tree, tree_node_t *node)
// Функция добавления элемента в дерево
// tree - корень дерева, node - узел для вставки

int fill_tree_by_file(tree_node_t **tree, const char *filename)
// Функция заполнения дерева из файла
// tree - корень дерева, filename - имя файла

void print_tree(const tree_node_t *root, int step)
// Функция печати дерева
// root - корень дерева, step - шаг

tree_node_t* find_tree(tree_node_t *root, const int num, int *compare_num)
// Функция поиска элемента в дереве
```

// root - корень дерева, num- число , compaare_num - количество сравнений

tree_node_t *do_a_balance_tree(tree_node_t *root, int *elems)
// Функция сбалансирования дерева
// root - корень, elems - количество элементов

tree_node_t *fill_tree_by_buf(buf_t *buf, int start, int end)
// Функция заполения дерева из буфера
// buf - буфер, start - индекс начала, end - индекс конца

void fill_buf_by_nodes(buf_t *buf, tree_node_t *root)
// Функция заполнения буфера из дерева
// buf -буфер , root - корень

void shift_and_push_buf(buf_t *buf, tree_node_t *el)
// Функция записи в буфер элемента
// buf - буфер, el - элемент

Для обработки типа hash_table_t используются функции:

int hash_f(int size, int coef)
// хеш-функция
// size - размер таблицы ,coef - коэффициент

int get_table(hash_table_t *table, char *filename)
// Функция получения хеш-таблицы
// table - таблица, filename - имя файла

hash_table_t get_new_table(hash_table_t *table)
// Функция реструктаризации хеш-таблицы
// table - таблица

void print_table(hash_table_t table)
// Функция печати хеш-таблицы
// table - таблица

table_node_t *find_table(lhash_table_t table, int num, int *comp_num)
// Функция добавления рандомных элементов в очередь
// table - таблица, num - число, comp_num - количество сравнений

Описание алгоритма

1. Вводится команда для выполнения определенной функции программы.
2. Выполняется введенная функция
    - Печать дерева
    - Печать сбалансированного дерева
    - Печать хеш-таблицы
    - Поиск элемента в 4-х структурах
    - Сравнение поиска по времени и памяти
3. При ошибке выполнения функции выводится сообщение об ошибке и программа завершается с ненулевым кодом возврата.

Тесты

Негативные тесты

№	Описание	Вводимая структура	Результат
1	Недопустимый символ команде меню	a2	Ошибка
2	Несуществующая команда меню	10	Ошибка
3	Недопустимое количество сравнений	-2	Ошибка
4	Несуществующий файл	123.ttt	Ошибка
5	Недопустимое значение элемента	1.3	Ошибка

Позитивные тесты

№	Описание	Входные данные	Результат
1	Правильный пункт меню	comand = 1	Выбор действия
2	Правильный файл	test.txt	Успешное чтение файла
3	Правильное количество	2	Успешный поиск

	сравнений		
4	Правильный элемент	3	Успешный поиск

Оценка эффективности

Сравнение эффективности

Поиск в тактах 1 элемента 1000 повторений

Кол-во элементов	Бинарное дерево	Сбалансированное дерево	Хеш-таблица	Файл
100	106	141	62	201183
250	164	85	50	400946
500	145	230	52	480056
1000	306	138	54	95118

Кол-во сравнений (среднее)

Кол-во элементов	Бинарное дерево	Сбалансированное дерево	Хеш-таблица	Файл
100	6	6	1	100
250	9	8	1	250
500	10	9	1	292
1000	13	10	1	247

Память с учетом большого количества колизий

Кол-во элементов	Бинарное дерево	Сбалансированное дерево	Хеш-таблица	Файл
100	2400	2400	3192	388
250	6000	6000	7896	1138
500	12000	12000	16024	3429
1000	24000	24000	31992	6887

Вывод

Хеш-таблица всегда эффективнее по времени (Для АВЛ в 2 раза, для ДДП в 15 раз). При этом не всегда эффективна по памяти т.к. при возникновении большого количества коллизий приходится реструктуризировать таблицу. АВЛ дерево всегда выигрывает по времени и памяти у ДДП дерева. Использование файла для поиска элемента имеет место быть только в том случае, когда нам не важно насколько быстро будет осуществлен поиск, главное, что файл занимает намного меньше памяти, чем остальные структуры.

Ответы на контрольные вопросы

1. Что такое дерево?

Дерево — одна из наиболее широко распространенных структур данных в информатике, эмулирующая древовидную структуру в виде набора связанных узлов.

2. Как выделяется память под представление деревьев?

В виде связного списка — динамически под каждый узел.

3. Какие стандартные операции возможны над деревьями?

Обход дерева, поиск по дереву, включение в дерево, исключение из дерева.

4. Что такое дерево двоичного поиска?

Двоичное дерево поиска - двоичное дерево, для каждого узла которого сохраняется условие: левый потомок больше или равен родителю, правый потомок строго меньше родителя (либо наоборот).

5. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

У АВЛ дерева для каждой его вершины высота двух её поддеревьев различается не более чем на 1, а у идеально сбалансированного дерева различается количество вершин в каждом поддереве не более чем на 1.

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

Поиск в ДДП может занять время O(n) в худжем случае. Поиск в АВЛ деревьях выполняется за O(log2(n)).

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

Хеш-таблицей называется массив, заполненный элементами в порядке,

определяемом хеш-функцией. Хеш-функция каждому элементу таблицы ставит в соответствие некоторый индекс. Функция должна быть простой для вычисления,распределять ключи в таблице равномерно и давать минимум коллизий.

8. Что такое коллизии? Каковы методы их устранения?

Коллизия – ситуация, когда разным ключам хеш-функция ставит в соответствие один и тот же индекс. Основные методы устранения коллизий: открытое и закрытое хеширование. При открытом хешировании к ячейке по данному ключу прибавляется связанны список, при закрытом – новый элемент кладется в ближайшую свободную ячейку после данной.

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

Поиск в хеш-таблице становится неэффективен при большом числе коллизий –сложность поиска возрастает по сравнению с O(1). В этом случае требуется реструктуризация таблицы – заполнение её с использованием новой хеш-функции.

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

В хеш-таблице минимальное время поиска O(1). В АВЛ: O(log2n). В дереве двоичного поиска O(h), где h - высота дерева (от log2n до n).