|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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 И ХЕШ-ФУНКЦИЙ»**

Студент Лазутин Александр Владимирович

Группа ИУ7 – 33Б

*2022 г.*

# **Описание технического задания**

Используя предыдущую программу (задача №6), сбалансировать полученное дерево. Вывести его на экран в виде дерева. Построить хеш-таблицу из чисел файла. Осуществить поиск введенного целого числа в двоичном дереве поиска, в сбалансированном дереве и в хеш-таблице. Сравнить время поиска, объем памяти и количество сравнений при использовании различных структур данных

Входные данные:

1. Целое число, представляющее собой пункт меню:

целое число в диапазоне от 0 до 6

2. Дополнительный ввод: поле типа int в зависимости от команды

Выходные данные:

1. Результат выполнения команды

2. Сообщение об ошибке (при ее возникновении)

Функции программы:

1. Ввести данные из файла

2. Вывести двоичное дерево поиска

3. Вывести АВЛ дерево

4. Вывести ХЕШ-таблицу

5. Добавить элемент в ДДП, АВЛ, ХЕШ-таблицу

6. Поиск элемента

0. Выйти из программы

Аварийные ситуации:

1. Неверно введен пункт меню

(не число или число меньшее 0 или больше 6)

2. Введенный файл не существует

(неверное имя файла)

3. Неверно введено число в файле

(не число)

4. Неверно введено число допустимых сравнений в ХЕШ-таблице

(не число)

5. Число, которое нужно добавить, уже содержится в структурах программы

6. Число, которое нужно найти, не содержится в структурах программы

Описание структуры данных

*Структура для хранения массива чисел, считанных из файла*

typedef struct arr\_r

{

int \*data;

int len;

int capacity;

int max\_cap;

} arr\_t;

Поля структуры:

1. int \*data— массив значений из файла

2. int len — длина массива чисел

3. int capacity — текущий объем памяти, задействованной в массиве

4. int max\_cap - максимальный объем памяти, выделенной под массив

Структура для хранения хеш таблицы, где есть массив самих чисел в таблице (int \*data) и массив состояний, который отображает, заполнена данная ячейка массива или нет (int \*key) и размер таблицы (int size)

typedef struct hash\_r

{

int \*data;

int \*key;

int size;

} hash\_t;

typedef struct node\_r

{

int data;

unsinged char height;

struct node\_r \*left;

struct node\_r \*right;

} node\_t;

Поля структуры:

1. int data — значение текущего корня

2. unsinged char height — высота вершины относительно других вершин

3. struct node\_r \*left— указатель на левого предка корня

4. struct node\_r \*right— указатель на правого предка корня

# **Описание алгоритма**

1. Выводится меню программы

2. Пользователь вводит номер любой команды, которой соответствует свое назначение

3. Ввод осуществляется, пока не будет совершена ошибка при вводе (аварийная ситуация) или пока не будет введен 0 (означает выход из программы)

# 

# **Набор тестов**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Название теста** | **Пользовательский ввод** | **Результат** |
| 1 | Некорректный ввод пункта меню | aaa | Ошибка: пункты меню это числа от 0 до 10 |
| 2 | Нет файла, соответствующего введенному имени | Имя несуществующего файла | Ошибка: неверно введено имя файла |
| 3 | Число в файле введено неверно (не число) | aaa | Ошибка: неверно число в файле |
| 4 | Невозможно добавить два одинаковых числа в структуры данных | Попытка добавить уще имеющееся число в структурах данных | Ошибка: такое число уже существует |
| 5 | Неверное число при добавлении  (не число) | aaa | Ошибка: неверно введено число для добавления |
| 6 | Неверно введено число допустимых сравнений для ХЕШ таблицы  (не число или число, меньшее 1) | aaa | Ошибка: неверно введено количество сравнений |
| 7 | Ввод данных из файла | Команда 1  корректное имя файла | Данные из файла введены успешно |
| 8 | Вывести дерево двоичного поиска | Команда 2  данные из файла введены | Вывод ДДП |
| 9 | Вывести АВЛ дерево | Команда 3  данные из файла введены | Вывод АВЛ дерева |
| 10 | Вывести ХЕШ таблицу | Команда 4  данные из файла введены | Вывод ХЕШ таблицы |
| 11 | Поиск элемента | Команда 6  Некорректное число | Ошибка: введено число, которого нет в структурах |
| 12 | Выход из программы | Команда 0 | Выход из программы, очистка консоли |

# **Оценка эффективности**

Поиск элемента (в тиках)

|  |  |  |  |
| --- | --- | --- | --- |
| Количество элементов в структурах | Дерево двоичного поиска | АВЛ дерево | ХЕШ таблица |
| 10 | 468 | 388 | 161 |
| 100 | 1488 | 996 | 195 |
| 500 | 2122 | 1338 | 183 |
| 1000 | 2990 | 2350 | 242 |

(Приведены средние значения)

Память

|  |  |  |  |
| --- | --- | --- | --- |
| Количество элементов в структурах | Дерево двоичного поиска | АВЛ дерево | ХЕШ таблица |
| 10 | 264 | 264 | 108 |
| 100 | 2448 | 2448 | 828 |
| 500 | 12048 | 12048 | 4028 |
| 1000 | 24072 | 24072 | 8076 |

# **Ответы на контрольные вопросы**

*1. Чем отличается идеально сбалансированное дерево от АВЛ дерева?*

В идеально сбалансированном дереве количество вершин в каждом поддереве различается не больше, чем на 1.  
А в АВЛ дереве для каждой его вершины высота ее двух поддеревьев различается не более, чем на 1.

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

*поиска?*

В АВЛ дереве поиск происходит быстрее, чем в дереве двоичного поиска.

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

ХЕШ-таблица — массив, в котором каждому числу при включении элемента в ХЕШ-таблицу ХЕШ-функция ставит в соответствие определенный индекс.

Функция должна быть простой для вычисления, а также такой, чтобы давать наименьшее возможное число коллизий.

*4. Что такое коллизии? Каковы методы их устранения.*

Коллизия — ситуация, при которой разным вводимым элементам в ХЕШ-таблицу ХЕШ-функцией ставится в соответствие один индекс.  
Методы устранения коллизий: открытое и закрытое хеширования.

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

При большом числе коллизий поиск в ХЕШ-таблице становится неэффективен, поэтому нужна реструктаризация таблицы (ее перезаполнение) с помощью новой ХЕШ-функции, чтобы уменьшить число коллизий.

*6. Эффективность поиска в АВЛ деревьях, в дереве двоичного поиска и в*

*хеш-таблицах*

ХЕШ-таблица — О(1) - О(n)

АВЛ дерево — O(log2n)

Дерево двоичного поиска — O(log2n) - O(n)

# **Вывод**

В данной лабораторной работе сравнивается поиск элемента в 3 разных структуры данных.

Самой эффективной структурой данных является ХЕШ-табилца (при отсутствии коллизий количество сравнений для его поиска равно 1), поэтому по времени данный способ является самым эффективным, также при небольшой и верно подобранной ХЕШ-функции объем занимаемой памяти также выигрывает у реализации деревьев (~ в 3 раза меньше) (как АВЛ, так и ДДП).

ХЕШ-таблица быстрее при добавлении элемента, чем АВЛ – 7 раз, чем ДДП – 9 раз.

Если сравнивать две реализации деревьев, то АВЛ дерево явялется более эффективной по времени структурой данных для поиска элемента, так как его высота обычно меньше, чем высота ДДП, из-за чего нужно меньшее количество сравнений.

АВЛ быстрее ДДП – ~ в 1.5 раза.

АВЛ дерево и ДДП знаимают одинаковое количество памяти.