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 №6 «ОБРАБОТКА ДЕРЕВЬЕВ И ХЕШ-ФУНКЦИЙ»

Студент Цветков Иван Алексеевич

Группа ИУ7 – 33Б

2020 г.

Описание условия задачи

Построить ДДП,сбалансированное двоичное дерево (АВЛ) и хеш-таблицу по указанным данным. Сравнить эффективность поиска в ДДП в АВЛ деревеи 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.

Описание технического задания

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 закрытое хеширование для устранения коллизий. Осуществить добавление введенного целого числа,если его там нет, в ДДП, в сбалансированное дерево, в хеш-таблицу и в файл.Сравнить время добавления, 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.

Входные данные:

1. Целое число, представляющее собой пункт меню:
целое число в диапазоне от 0 до 5

2. Дополнительный ввод: поле типа int или char в зависимости от требования

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 и файл (с выводом замеров времени и используемой памяти)

0. Выйти из программы

Обращение к программе:
Запускается через терминал командой ./app.exe

Аварийные ситуации:

1. Неверно введен пункт меню
   (не число или число меньшее 0 или больше 5)
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

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	iu	Ошибка: пункты меню это числа от 0 до 10
2	Нет файла, соответствующего	имя несуществуюшего	Ошибка: неверно

	введенному имени	файла	введено имя файла
3	Число в файле введено неверно (не число)	iu	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)	iu	Ошибка: неверно введено число для добавления
6	Неверно введено число допустимых сравнений для ХЕШ таблицы (не число или число, меньшее 1)	iu или -1	Ошибка: неверно введено количество сравнений
7	Ввод данных из файла	команда 1 и 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	Добавление элемента (количество сравнений не превышено)	Команда 5 допустимое количество сравнений для добавления элемента в ХЕШ таблицу не превышено	Вывод замеров времени, памяти, сравнений для добалвения элемента в файла
12	Добавление элемента (количество сравнений превышено)	Команда 5 допустимое количество сравнений для добавления элемента в ХЕШ таблицу превышено	Реструктуризация таблицы, пока не будет достигнуто допустимое значение для добавления элемента Затем вывод замеров времени, памяти, сравнений для добалвения элемента в файла
13	Добавление элемента (размер таблицы	Команда 5	Реструктуризация таблицы

	достиг максимума)	допустимое количество сравнений для добавления элемента в ХЕШ таблицу не превышено	Затем вывод замеров времени, памяти, сравнений для добалвения элемента в файла
14	Выход из программы	команда 0	Выход из программы, очистка консоли

Оценка эффективности

Время будет измеряться в тактах процессора на процессоре с частотой 35000000 Гц

Добавление элементов (в тиках)

Количество элементов в структурах	Дерево двоичного поиска	АВЛ дерево	ХЕШ таблица	Файл
10	1242	2004	154	26352
100	1762	2998	197	29195
500	2278	3812	187	31978
1000	2567	4159	360	33777

Приведены средние значения

Память

Количество элементов в структурах	Дерево двоичного поиска	АВЛ дерево	ХЕШ таблица	Файл
10	264	264	92	31
100	2400	2400	812	289
500	12024	12024	4028	1882
1000	24024	24024	8076	3763

Сравнения для добавления (добавлялось самое большое число из всех)

Количество элементов в структурах	Дерево двоичного поиска	АВЛ дерево	ХЕШ таблица	Файл
10	5	5	1	11
100	5	7	1	101
500	7	10	1	501
1000	10	11	1	1001

Пример работы программы

```
Введите элемент для добавления:  102


Введите резрешенное количество сравнений для ХЕШ таблицы:  2


Сравнений для добавления в ХЕШ таблицу: 5


Реструктаризация ХЕШ таблицы произведена по причине большого количества сравнений



Сравнений для добавления в новую таблицу: 1


Замеры для ХЕШ таблицы

Время для добавления (в тиках): 52
Памяти зайдествовано: 23б
Сравнений для вставки: 1



Замеры для файла

Время для добавления (в тиках): 23191
Памяти зайдествовано: 37
Сравнений для вставки: 13



Замеры для ДДП

Время для добавления (в тиках): 964
Памяти зайдествовано: 312
Сравнений для вставки: 4



Замеры для АВЛ

Время для добавления (в тиках): 2160
Памяти зайдествовано: 312
Сравнений для вставки: 4
```

Ответы на контрольные вопросы

1. Что такое дерево?

Дерево — структура данных (рекурсивная), используемая для представления иерархических связей (один ко многим)

2. Как выделяется память под представление деревьев?

Память выделяется как для связанного списка, то есть под каждый узел отдельно.

3. Какие стандартные операции возможны над деревьями?

Поиск по дереву, обход дерева, добавление элемента в дерево, удаление элемента из дерева

4. Что такое дерево двоичного поиска?

Двоичное дерево поиска (ДДП) — двоичное дерево
В нем для каждого узла выполняется условие, что левый потом больше или равен родителю, а правый потомок строго меньше родителя (или наоборот)

5. 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

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

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

7. 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 соотвествие определенный индекс

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

8. Что такое коллизии? Каковы методы их устранения.

Коллизия — ситуация, при котрой разным вводимым элементам в ХЕШ-таблицу ХЕШ-функцией ставится в соответствие один индекс

Методы устранения коллизий: открытое и закрытое хеширования

9. 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

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

ХЕШ-таблица — O(1) - O(n)
АВЛ дерево — O(log2n)
Дерево двоичного поиска — O(log2n) - O(n)

Вывод

В данной лабораторной работе сравнивается добавление элемента в 4 разных структуры данных

Самой эффективной структурой данных является ХЕШ-таблица (при отсутствии коллизий количество сравнений для нахождения добалвения элемента или 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 (примерно в 3 раза меньше) (как АВЛ, так и ДДП), но проигрывает файлу по занимаемой памяти (примерно в 3 раза)

ХЕШ-таблица быстрее при добавлении элемента

чем АВЛ – 15 раз

чем ДДП – 10 раз

Если сравнивать две реализации деревьев (АВЛ и ДДП), 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.

Но если речь идет о добавлении элемента в дерево, то АВЛ дерево начинает проигрывать по причине того, что при добавлении элемента нужна перебалансировка дерева, что занимает дополнительное время.

ДДП быстрее АВЛ – в 1.6-2 раза

АВЛ дерево и ДДП знаимают одинаковое количество памяти.