**Отчет лабораторной работы №6**

по курсу «Типы и структуры данных»

Обработка деревьев, работа с хеш-таблицами

Отчёт выполнила:

Кондрашова Ольга

Группа ИУ7-35Б

**Цель работы**:

построить и обработать хеш-таблицы, сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска и в хеш-таблицах. Сравнить эффективность устранения коллизий при внешнем и внутренним хешировании.

**Задание**:

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, то произвести реструктуризацию таблицы, выбрав другую функцию.

**Входные данные:**

Файл с целыми числами

Пункт меню:

1. Добавить число в двоичное дерево

2. Добавить число в сбалансированное дерево

3. Добавить число в хеш-таблицу

4. Добавить число в файл

5. Вывод хеш-таблицы

6. Эффективность поиска

7. Эффективность добавления

8. Среднее число сравнений

0. Выход

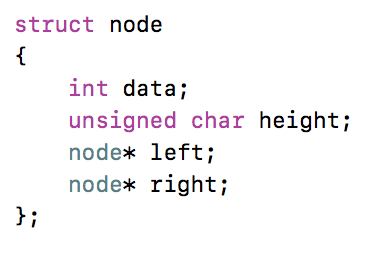
**Выходные данные:**

Двоичное дерево, сбалансированное дерево, хеш-таблица, оценка эффективности добавления и поиска элемента (для дерева, хеш-таблицы и файла), среднее кол-во сравнений.

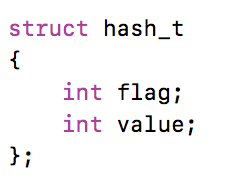
**Обращение к программе:**Через консоль

**Внутренняя структура данных:**

Дерево:



Хеш-таблица:



**Описание задачи, реализуемой программой**

1. Построение ДДП из чисел в файле.
2. Построение сбалансированное дерево.
3. Преобразование деревьев из формата .dot в .png и вывод на экран.
4. Построение хеш-таблицы.
5. Добавление введенного пользователем целого числа в ДДП, сбалансированное дерево, хеш-таблицу и файл.
6. Сравнение времени добавления, поиска, объем памяти и количество сравнений при использовании различных структур данных.
7. Реструктуризация хеш-таблицы по требованию пользователя.

**Алгоритм**

1 – добавление элемента в ДДП

2 – добавление элемента с сбалансированное дерево

3 – добавление в хеш-таблицу

4 – добавление в файл

5 – печать хеш-таблицы

6 – вывод времени поиска элементов для всех типов данных

7 – вывод времени добавления элементов для всех типов данных

8 – вывод среднего кол-ва сравнений при поиске – возможна реструктуризация хеш-таблицы

0 – завершение работы программы

**Функции**

node\* insert(node \*head, node \*elem, int\* back)

Вход: корень дерева, элемент, флаг наличия числа

Выход: добавление элемента в ДДП

node\* insert\_balance(node\* p, node\* k, int\* back)

Вход: корень дерева, ключ, флаг наличия числа

Выход: добавление элемента в АВЛ

void add\_to\_hash\_table(int key, my\_int\*\* table, int len, int\* back)

Вход: ключ, пустая таблица, длина таблицы, флаг наличия элемента.

Выход: добавление в хеш-таблицу

void add\_to\_file(int key, char\* file\_name, int\* back)

Вход: ключ, название файла, флаг наличия элемента

Выход: добавление в файл

void print\_hash\_table(my\_int\*\* table, int len)

Вход: таблица, длина таблицы

Выход: печать хеш-таблицы

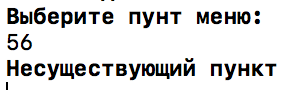
void export\_to\_dot(FILE \*f, node \*head)

Вход: файл, корень дерева

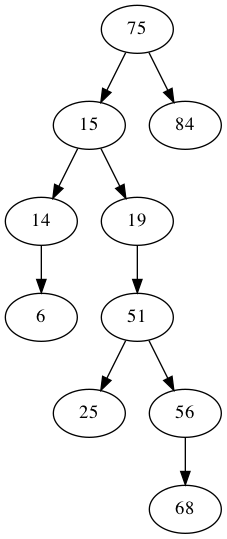
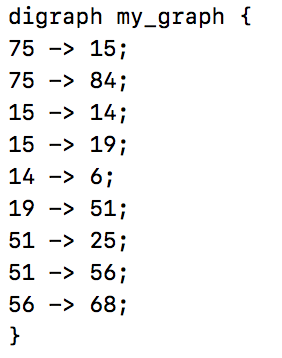
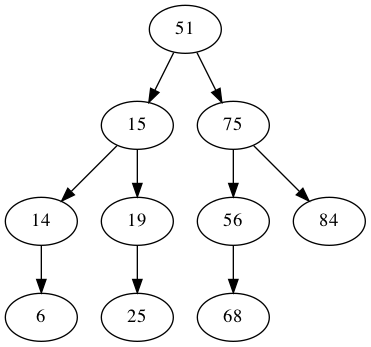
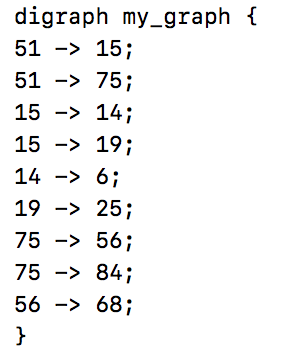
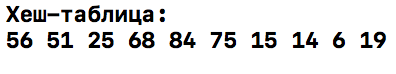
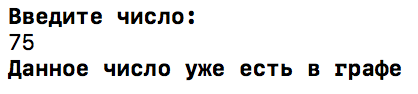
Выход: файл с расширением .gv

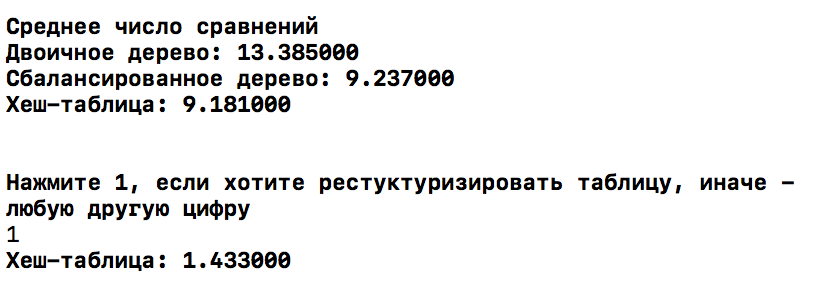
**Аварийные ситуации и ошибки пользователя**

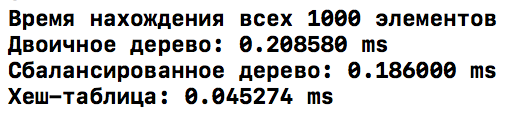
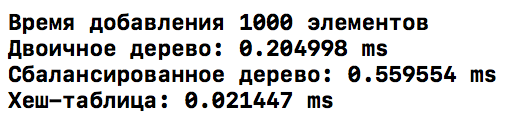
Ввод несуществующей команды меню



**Тесты**

1. Вывод деревьев   
     
   
2. Вывод хеш-таблицы  
   
3. Попытка добавить число, которое уже находится в структуре данных  
   
4. Вывод кол-ва сравнение и реструктуризация хеш-таблицы



1. Вывод результатов тестирования поиска  
   
2. Вывод результатов тестирования добавления  
   

**Оценка эффективности**

Тестирование проведено для 1000 рандомных элементов, единица измерения – мкс.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ДДП | АВЛ | Хеш-таблица | Файл |
| Поиск | 0.210999 | 0.168735 | 0.062785 | 13.776384 |
| Добавление | 0.132901 | 0.396467 | 0.057528 | 19.444354 |
| Среднее кол-во сравнений | 12.206000 | 9.155000 | 12.489000 | 50.511067 |

Наиболее эффективны в использовании хеш-таблицы: поиск и добавление в хеш-таблице происходит быстрее, чем в ДДП и АВЛ.

Поиск в хеш-таблице в 3.5 раза эффективнее, чем в ДДП, в 2.6 раз эффективнее, чем в АВЛ и в 224 раз эффективнее, чем в файле.

Добавление в хеш-таблицу в 2.6 раза эффективнее, чем в ДДП, в 7.8 раз эффективнее, чем в АВЛ и в 398 раз эффективнее, чем в файле.

**Вывод:**

Основным преимуществом деревьев является возможная высокая эффективность реализации основанных на нём алгоритмов поиска и сортировки. Но при удалении и добавлении элементов необходимо корректировать балансировку.

Хеш таблицы используют меньше памяти, и требуют минимального количества операций сравнения при поиске. Таблицы также требуют качественной хеш-функции для избегания колизий.Для того, чтобы в хеш-таблице быстрее осуществлялся поиск, должно быть как можно меньше коллизий. Количество коллизий зависит от того, сколько было выделено памяти, следовательно, для большей эффективности по времени придется выделить больше памяти.

**Ответы на вопросы**

1. Что такое дерево?

Дерево – это нелинейная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим».

2. Как выделяется память под представление деревьев?

Выделение памяти под деревья определяется типом их представления. Это может быть таблица связей с предками или связный список сыновей. Оба представления можно реализовать в виде матрицы или списка. При реализации списком память выделяется динамически, при реализации матрицей статически.

3. Какие стандартные операции возможны над деревьями?

Основные операции с деревьями: обход дерева, поиск по дереву, включение в дерево, исключение из дерева.

4. Что такое дерево двоичного поиска?

Дерево двоичного поиска – это такое дерево, в котором все левые потомки моложе предка, а все правые – старше.

5. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

Дерево, у которого число вершин в левом и правом поддеревьях отличается не более, чем на единицу называется идеально сбалансированным. Двоичное дерево, у каждого узла которого высота двух поддеревьев отличается не более чем на единицу называется АВЛ-деревом.

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

Поиск в АВЛ дереве имеет сложность О(log2n), в то время как в обычном ДДП сложность О(n).

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

Массив, заполненный в порядке, определенным хеш-функцией, называется хеш- таблицей.

Хеш-функция – функция, которая ставит в соответствие каждому ключу индекс ячейки, где расположен элемент с этим ключом.

8. Что такое коллизии? Каковы методы их устранения.

Коллизия – ситуация, когда разным ключам соответствует одно значение хеш-функции. Существует несколько возможных вариантов разрешения коллизий:  
внешнее (открытое) хеширование (метод цепочек) и внутреннее (закрытое) хеширование (открытая адресация).

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

Поиск в хэш-таблице становится неэффективным при большом числе коллизий – сложность поиска возрастает.

При открытом хэшировании в случае, когда элемент таблицы с индексом, который вернула хеш-функция, уже занят, к нему присоединяется связный список. Таким образом, если для нескольких различных значений ключа возвращается одинаковое значение хеш- функции, то по этому адресу находится указатель на связанный список, который содержит все значения. Поиск в этом списке осуществляется простым перебором, так как при грамотном выборе хеш-функции любой из списков оказывается достаточно коротким.

При закрытом хэшировании в этом случае, если ячейка с вычисленным индексом занята, то можно просто просматривать следующие записи таблицы по порядку, до тех пор, пока не будет найден ключ K или пустая позиция в таблице.

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

В хэш-таблице минимальное время поиска О(1). В АВЛ дереве О(log2n). В дереве двоичного поиска О(h), где h – высота дерева.