|  |  |
| --- | --- |
| **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 «СБАЛАНСИРОВАННЫЕ ДЕРЕВЬЯ, ХЕШ-ТАБЛИЦЫ»**

Студент Ланкин Дмитрий Леонидович

Группа ИУ7 – 34Б

Приняла: Барышникова Марина Юрьевна

*2022 г*

# Описание условия задачи

Построить и обработать хеш-таблицы, сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска и в хеш-таблицах.

# Техническое задание

**Вариант 2.** Используя предыдущую программу (задача №6), сбалансировать полученное дерево. Вывести его на экран в виде дерева. Построить хеш-таблицу из слов текстового файла, задав размерность таблицы с экрана. Осуществить поиск введенного слова в двоичном дереве поиска, в сбалансированном дереве и хеш-таблице. Сравнить время поиска, объем памяти и количество сравнений при использовании различных структур данных.

## Входные данные

**Целое число**: номер команды.

## Выходные данные

Двоичное дерево поиска, сбалансированное дерево и хеш-таблица. Сравнительный анализ времени поиска слов в ДДП, АВЛ и хеш-таблице.

## Описание задачи

Программа получает от пользователя номер команды и выполняет действия в соответствии с командами.

## Способ обращения к программе

Программа вызывается из консоли.

## Описание возможных аварийных ситуаций и ошибок пользователя

1. Некорректный ввод: введен буквенный символ.

Сообщение об ошибке: «Некорректный ввод.»

1. Некорректный ввод: номер команды не входит в соответствующий диапазон.

Сообщение об ошибке: «Неверная команда.»

1. Дерево пустое.

Сообщение об ошибке: «Дерево пустое.»

1. Ошибка выделения памяти.

Сообщение об ошибке: «Ошибка выделения памяти.»

# Внутренняя структура данных

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **typedef** **struct**  {  **char** name[MAX\_LEN];  } value\_s;  **typedef** **struct** **hash\_elem\_t**  {  **size\_t** hash\_index;  value\_s hash\_value;  **struct** **hash\_elem\_t** \*next;  } **hash\_elem\_t**;  **typedef** **struct** **hash\_elem\_step\_t**  {  **size\_t** hash\_index;  value\_s hash\_value;  } **hash\_elem\_step\_t**; |

Таблица 1. Реализация хеш-таблиц.

hash\_elem\_t – элемент хеш-таблицы, коллизии в которой решаются методом цепочек.

* hash\_index – значение хеша элемента;
* hash\_value – элемент хеш-таблицы;
* next – указатель на следующий элемент в односвязном списке для метода цепочек.

hash\_elem\_step\_t – элемент хеш-таблицы, коллизии в которой решаются методом внутреннего хеширования.

* hash\_index – значение хеша элемента;
* hash\_value – элемент хеш-таблицы.

# Особенности реализации алгоритма

1. Программа запрашивает у пользователя желаемое действие из следующего списка:
   1. Заполнить все типы данных из файла;
   2. Сбалансировать ДДП;
   3. Анализ количества сравнений для поиска в ДДП, АВЛ и хеш-таблицах;
   4. Вывести на экран ДДП;
   5. Вывести на экран АВЛ;
   6. Вывести на экран хеш-таблицу (метод цепочек);
   7. Вывести на экран хеш-таблицу (внутр. хеш-ие);
   8. Реструктуризировать хеш-таблицу (метод цепочек);
   9. Реструктуризировать хеш-таблицу (внутр. хеш-ие);
2. Завершить работу.
3. Работа программы продолжается до тех пор, пока пользователь не введет 0 при запросе команды.
4. ДДП и АВЛ выводятся с помощью GraphViz.

# Набор тестовых случаев

|  |  |  |  |
| --- | --- | --- | --- |
| № | Ситуация | Входные данные | Результат |
| 1 | Введен буквенный символ | a | Некорректный ввод. |
| 2 | Введена несуществующая функция | 10 | Некорректный ввод. |
| 3 | Заполнение дерева из несуществующего файла. | – | Ошибка при открытии файла. |
| 4 | Заполнение дерева из поврежденного файла. | – | Ошибка при открытии файла. |
| 5 | Некорректный ввод при запросе слова. | Fdm fd | Некорректный ввод. |
| 6 | Введена несуществующая команда. | 6 | Неверная команда. |

# Показатели эффективности

Оценка эффективности производилась методом запуска некоторого количества раз одной функции и нахождения среднего арифметического с целью избежания статистических выбросов и приблизиться к реальным условиям.

Для начала сравним скорость нахождения элементов в ДДП, АВЛ и хеш-таблицах с учетом, что при 100 элементах есть 49 коллизий.

Результаты поиска для элемента, являющегося четвертым в списке для метода цепочек.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Количество сравнений, шт | Время поиска, тики | Объём памяти, байтов |
| ДДП | 7 | 205 | 2 112 |
| АВЛ | 6 | 156 | 704 |
| Хеш-таблица (цепочки) | 4 | 64 | 27 200 |
| Хеш-таблица (внутреннее хеширование) | 21 | 128 | 26 400 |

Таблица 2. Результаты до реструктуризации при поиске элемента с глубиной 4 при коллизии.

Так как количество сравнений при поиске в хеш-таблице с методом цепочек равно 4, необходимо произвести реструктуризацию. Так же это необходимо сделать и в хеш-таблице с внутренним хешированием. Количество коллизий для внутреннего хеширования составляет всего 3.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Количество сравнений, шт | Время поиска, тики | Объём памяти, байтов |
| ДДП | 7 | 169 | 2 112 |
| АВЛ | 6 | 120 | 704 |
| Хеш-таблица (цепочки) | 1 | 32 | 69 904 |
| Хеш-таблица (внутреннее хеширование) | 1 | 32 | 114 312 |

Таблица 3. Результаты после реструктуризации.

## Выводы

Из проделанной работы стало понятно, что в случаях, когда есть необходимость осуществлять максимально быстрый поиск и затраты памяти не критичны, следует использовать хеш-таблицы в силу того, что минимальное время, необходимое для поиска, составляет О(1), т.е. реализуется произвольный доступ к элементу. Однако в случае возникновения коллизий есть 2 метода борьбы с ними.

В процессе выполнения лабораторной работы я выяснил, что метод цепочек, используемый для решения коллизий, является более удобным и предсказуемым в случаях, когда коллизий достаточно много. В список преимуществ данного метода можно внести и заметно меньшие затраты памяти.

Так же я узнал, что АВЛ-деревья являются лучшей альтернативой двоичному дереву поиска вследствие уменьшения затрат памяти (в данном случае, ровно в 3 раза) и увеличения скорости поиска необходимых элементов конкретно в случаях достаточно большой глубины в аналогичном ДДП.

# Контрольные вопросы

1. Чем отличается идеально сбалансированное дерево от АВЛ?

Идеально сбалансированное дерево: для каждой его вершины высоты двух её поддеревьев отличается не более, чем на 1.

А в АВЛ-дереве количество вершин в левом и правом поддеревьях может отличаться более, чем на 1.

1. Чем отличается поиск в АВЛ дереве от поиска в ДДП?

Поиск в АВЛ дереве происходит значительно быстрее в силу зачастую меньшей высоты самого дерева.

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

Хеш-таблица – массив элементов, порядок нахождения которых определяется конкретной хеш-функцией, связывающей значение элемента с его индексом в данном массиве.

1. Что такое колизии? Каковы методы их устранения?

Коллизия – это ситуация, при которой различным ключам ставится в соответствие один и тот же индекс.

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

В случае, если в хеш-таблице наблюдается большое количество коллизий, алгоритмическая сложность поиска возрастает относительно О(1) и эффективность теряется. Однако это решается путем реструктуризации таблицы с использованием иной, более эффективной хеш-функции.

1. Эффективность поиска в АВЛ деревьях, в ДДП, в хеш-таблицах и файле.

ДДП: минимальное время поиска O(log2n), максимальное О(n).

АВЛ: минимальное время поиска O(log2n), где n – количество вершин.

Хеш-таблица: минимальное время поиска О(1).