Лабораторная работа № 1  
Хеш-таблицы

**Цель работы**: изучить основные методы организации хеш-таблиц, получить представление о преимуществах и недостатках, присущих различным методам организации таблиц.

Для выполнения лабораторной работы требуется написать программу, которая получает на входе набор идентификаторов, организует таблицу по заданному методу и позволяет осуществить поиск идентификатора в этой таблице.

Список идентификаторов задан в виде текстового файла. Длина идентификаторов ограничена 32 символами.

**Хеш-таблица** — это структура данных, которая используется для эффективного хранения, поиска и извлечения данных. Основная идея хеш-таблицы заключается в использовании хеш-функции для преобразования ключа (например, строки или числа) в индекс массива, где будет храниться соответствующее значение.

**Основные компоненты хеш-таблицы:**

* **Хеш-функция** — это функция, которая принимает ключ и возвращает индекс, по которому значение этого ключа будет храниться в массиве. Хорошая хеш-функция равномерно распределяет ключи по доступным индексам, чтобы минимизировать вероятность коллизий.
* **Массив**: хеш-таблица обычно реализуется с использованием массива, где каждый элемент массива называется "ячейкой". Индексы этого массива генерируются хеш-функцией.
* **Ключи и значения**: в хеш-таблице каждый элемент (или запись) состоит из ключа и связанного с ним значения. Ключ используется для вычисления индекса при помощи хеш-функции, по которому затем можно быстро найти значение.

**Как работает хеш-таблица:**

* **Вставка**: когда нужно вставить новый элемент, хеш-функция вычисляет индекс на основе ключа. Затем значение сохраняется в массиве по этому индексу.
* **Поиск**: чтобы найти значение, хеш-функция снова используется для вычисления индекса из ключа. Затем можно быстро перейти к соответствующему индексу в массиве и извлечь значение.
* **Удаление**: для удаления элемента хеш-функция используется для нахождения индекса, по которому элемент хранится, и затем этот элемент удаляется.

**Коллизии**

Коллизия происходит, когда два разных ключа дают одинаковый индекс при использовании хеш-функции. Для разрешения коллизий используются различные методы, такие как:

* **Метод списков**: в каждой ячейке массива хранится список (или другая структура данных), в который помещаются все элементы, имеющие один и тот же хеш.
* **Открытая адресация**: при возникновении коллизии ищется другая свободная ячейка массива по определённому алгоритму (например, линейное или квадратичное пробирование).

**Преимущества и недостатки**

**Преимущества:**

* Быстрый доступ к данным: O(1) время в среднем случае для вставки, удаления и поиска.
* Простота реализации.

**Недостатки:**

* Неэффективное использование памяти, если размер массива сильно превышает количество элементов.
* Необходимость хорошей хеш-функции для равномерного распределения ключей и минимизации коллизий.
* В худшем случае время доступа может быть O(n), если все ключи попадают в одну и ту же ячейку.

Хеш-таблицы широко используются для реализации таких структур данных, как ассоциативные массивы, словари и множества.

**Порядок выполнения работы и требования к отчету**

Отчет по лабораторной работе должен содержать следующие разделы:

1. Задание по лабораторной работе.
2. Схему организации хеш-таблицы (в соответствии с вариантом задания).
3. Описание алгоритма поиска в хеш-таблице (в соответствии с вариантом задания).
4. Текст программы.
5. Скриншоты, демонстрирующую работу программы.
6. Выводы по проделанной работе.

Вариант лабораторной работы определяется преподавателем с помощью генератора случайных чисел.

**Варианты заданий**

* 1. Таблица организуется в виде упорядоченного списка. Поиск идет простым перебором. Подсчитывается число выполненных сравнений: в среднем и для каждого идентификатора.
  2. Таблица организуется в виде списка, упорядоченного в алфавитном порядке. Поиск - логарифмический. Подсчитывается число сравнений: в среднем и для каждого идентификатора.
  3. Таблица упорядочивается в обратном алфавитном порядке, при этом все буквы преобразуются в заглавные. Поиск - логарифмический с подсчетом числа сравнений.
  4. Таблица строится по методу цепочек с использованием хеш-функции, возвращающей код первой буквы идентификатора. При выполнении программы подсчитывается число коллизий.
  5. Таблица строится по методу цепочек с использованием хеш-функции, возвращающей сумму двух первых букв идентификатора. При выполнении должно подсчитываться число выполненных сравнений: в среднем и для каждого идентификатора.
  6. Таблица строится с использованием хеш-функции на основе суммы трех первых букв идентификатора. При этом все буквы переводятся в заглавные (большие). Одинаковые элементы помещаются в одну ячейку таблицы, внутри которой осуществляется поиск по простому перебору.
  7. Таблица строится с использованием хеш-функции из варианта №6. Одинаковые элементы помещаются в одну ячейку, внутри которой организуется упорядоченный список.
  8. Таблица строится с использованием хеш-функции из варианта №6. Одинаковые элементы помещаются в одну ячейку, внутри которой используется логарифмический поиск.
  9. Таблица строится по методу дерева. Программа должна подсчитывать число выполненных сравнений при поиске.
  10. Таблица строится по методу дерева с использованием хеш-функции при сравнении идентификаторов. В качестве хеш-функции выступает код первой буквы идентификатора. Внутри каждой ячейки дерева поиск идет простым перебором.
  11. Таблица по варианту №10. Внутри каждой ячейки дерева организуется упорядоченный список идентификаторов.
  12. Таблица строится по методу дерева с использованием хеш-функции при сравнении идентификаторов. В качестве хеш-функции выступает сумма двух первых букв идентификатора. Внутри каждой ячейки дерева организуется упорядоченный список идентификаторов.
  13. Придумать некоторую хеш-функцию и организовать для нее таблицу по методу цепочек. Подсчитать среднее число коллизий.
  14. Упорядоченный список с бинарным поиском: таблица хранится в виде списка, отсортированного в алфавитном порядке. Поиск выполняется методом бинарного поиска. Подсчитывается количество сравнений при поиске в среднем и для каждого идентификатора.
  15. Неупорядоченный список с подсчетом количества операций вставки: Таблица хранится в виде неупорядоченного списка. При добавлении нового идентификатора подсчитывается количество перемещений элементов, необходимых для поддержания порядка.
  16. Обратный алфавитный порядок с преобразованием регистра: Таблица организована в виде списка, упорядоченного в обратном алфавитном порядке. Все буквы преобразуются в строчные. Поиск осуществляется линейным перебором с подсчетом количества сравнений.
  17. Хеш-таблица с цепочками и учетом длины цепочек: таблица строится с использованием хеш-функции, возвращающей код первой буквы идентификатора. При коллизиях используется метод цепочек. Подсчитывается средняя длина цепочек и максимальная длина цепочки.
  18. Хеш-таблица с квадратичным пробированием: таблица использует хеш-функцию, которая вычисляет сумму кодов двух первых букв идентификатора. Коллизии разрешаются методом квадратичного пробирования. Подсчитывается количество шагов, необходимых для разрешения коллизий.
  19. Хеш-таблица с двойным хешированием: таблица организована с использованием хеш-функции на основе суммы трех первых букв идентификатора. При коллизиях используется метод двойного хеширования. Подсчитывается количество коллизий и количество сравнений при поиске.
  20. Двоичное дерево поиска с балансировкой: таблица организуется в виде двоичного дерева поиска. При добавлении новых идентификаторов выполняется балансировка дерева, если глубина превышает заданное значение. Подсчитывается количество операций балансировки и количество сравнений при поиске.