Большинство методов поиска основывается на упорядочении данных. Однако эффективность поиска можно повысить, если вместо упорядоченных структур использовать так называемые беспорядочные или перемешанные структуры данных. Для работы с такими структурами предназначены методы хеширования. Среднее время поиска элемента этими методами есть $O(1)$, время для наихудшего случая – $O(n)$. Здесь n – общее число элементов.

Постановка задачи

Пусть задано некоторое количество записей, среди которых осуществляется поиск. Предположим, что записи размещены в оперативной памяти и, следовательно, каждой можно поставить в соответствие число a из диапазона $0…m$-1, которое назовем *адресом* записи. Например, записи размещены в одномерном массиве; в качестве значения a можно взять смещение записи относительно начала массива, а в качестве m – размер массива.

Определение. Назовём *ключом* записи некоторый числовой идентификатор, однозначно определяющий эту запись.

Обозначим через k – ключ записи с адресом a. Тогда *задача поиска* состоит в определении по заданному аргументу k адреса записи $a(k)$, содержащей заданный ключ.

Общие понятия

Определение. Организацию данных в виде таблицы назовём хеш-таблицей, если адрес каждой записи a этой таблицы определяется как значение некоторой функции (хеш-функции) $h(k)$, где k – значение ключа записи a.

Если число записей невелико и заранее известно, то можно построить функцию преобразования заданного множества ключей в различные адреса (и если возможно – в последовательные адреса, что выгодно при использовании виртуальной памяти). Если же число записей велико, то найти такую функцию оказывается сложно. В случае если число различных значений ключей, вероятность появления которых отлична от нуля, превышает размер хеш-таблицы, построение функции оказывается невозможным. В этом случае приходится отказываться от идеи однозначности и рассматривать хеш-функцию как функцию, рассеивающую множество ключей в множество адресов $0…m$-1.

Отказ от требования взаимно однозначного соответствия между ключом и адресом означает, что для двух различных ключей $k1 \neq k2$ значение хеш-функции может совпадать: $h(k1) = h(k2)$. Такая ситуация называется *коллизией*.

Определение. Ключи $k1$ и $k2$ называются *синонимами* хеш-функции h, если $h(k1) = h(k2)$.

Для метода хеширования главными задачами являются выбор хеш-функции h и нахождение способа разрешения возникающих коллизий.

Хеш-функции

При выборе хеш-функции следует учитывать скорость ее работы, а также равномерность распределения значений, которая позволяет не только сократить число коллизий, но и не допустить скучивания значений в отдельных частях таблицы.

В каждого конкретного множества возможных ключей можно «изобрести» (подобрать, найти) свою, возможно наилучшую, хеш-функцию распределения ключей по таблице. Но существуют и универсальные, дающие в большинстве случаев хорошие результаты, хеш-функции. Рассмотрим их.

Метод деления

В методе деления в качестве значения хеш-функции используется остаток от деления ключа на некоторое целое число M: $h(k)=k \bmod M$, где M обычно равняется размеру хеш-таблицы. Эффективность рассеивания ключей во многом зависит от значения M. Не стоит выбирать M, равное степени основания системы счисления, так как значением хеш-функции будут просто младшие разряды ключа. Например, для буквенных ключей не следует выбирать M, равное 2^8 или 2^{16}. В этом случае хеш-функция будет равняться одной или двум последним буквам ключа.

Для предотвращения скучивания ключей следует выбирать M, равное простому числу.

Метод деления часто используется после применения другой хеш-функции для соответствия полученных значений размеру хеш-таблицы.

Метод свертки (слияния)

Предположим, что ключ представлен в виде последовательности разрядов a_i: $k=a_1a_2a_3\ldots a_p$, где p кратно некоторому числу w. Тогда значением хеш-функции будет сумма $h(k)=a_1a_2\ldots a_w \oplus a_{w+1}a_{w+2}\ldots a_{2w} \oplus \ldots \oplus a_{p-w+1}a_{p-w}\ldots a_p$, где в качестве операции $\oplus$ может использоваться операция арифметического или побитового сложения, побитовая операция «исключающее или» и т.д.

Для буквенных ключей в качестве w удобно выбирать значения, кратные 8. Основной недостаток этого метода состоит в том, что он недостаточно «чувствителен» к порядку букв в слове. Но избавиться от этого недостатка просто. Допустим, что результат каждого последовательного применения операции $\oplus$ сохраняется в переменной h', конечное значение которой было результатом вычисления хеш-функции. Тогда для воздействия порядка букв в слове на значение $h(k)$ необходимо применять операцию циклического сдвига h' перед очередным применением операции $\oplus$.

Метод умножения

Представим значение ключа k в виде двоичного числа и примем m (размер хеш-таблицы) равным 2^p. Умножим дробь d на k и возьмем дробную часть числа, которую обозначим как $\{kd\}$, а в качестве значения хеш-функции используем p старших разрядов[1], т.е. $h(k)=\lfloor m\times\{kd\}\rfloor$, где $\lfloor x\rfloor$ – наибольшее целое число, не превосходящее x.

Рекомендуется в качестве значения d брать иррациональное число, например золотое сечение $(\sqrt{5}-1)/2$. При $d=1/m$ метод эквивалентен методу деления.

Метод «середины квадрата»

Пусть m (размер хеш-таблицы) равен 2^p. Обозначим $d=k^2$ и представим d в виде двоичного числа. Тогда значением хеш-функции $h(k)$ будет p битов средней части d.

Данный метод по многим параметрам уступает методу умножения.

Метод преобразования системы счисления

В основе метода лежит преобразование значения ключа k, выраженного в системе счисления с основанием p ($k=a_0+a_1p+a_2p_2+\ldots$), в систему счисления с основанием q ($h(k)=a_0+a_1q+a_2q_2+\ldots$) при условии, что $p<q$. Трудоемкость (число операций) этого метода оказывается большей, чем методов деления или умножения.

---

[1] Требование, чтобы при вычислении $h(k)$ размер хеш-таблицы равнялся степени 2, не является обязательным. Оно было использовано только для облегчения понимания работы метода.

Метод деления многочленов

Пусть k, выраженное в двоичной системе счисления, записывается как $k=2^n b_n+\ldots+2b_1+b_0$, и пусть m (размер хеш-таблицы) является степенью двойки $m=2^p$. Представим двоичный ключ k в виде многочлена вида $k(t)=b_n t^n+\ldots+b_1 t+b_0$. Определим остаток от деления этого многочлена на постоянный многочлен вида $c(t)=t^m+c_{m-1}t^{m-1}+\ldots+c_1 t+ c_0$. Этот остаток, опять же рассматриваемый в двоичной системе счисления, используется в качестве значения хеш-функции $h(k)$. Для вычисления остатка от деления многочленов используют полиномиальную арифметику по модулю 2. Если в качестве $c(t)$ выбрать простой неприводимый многочлен, то при условии близких, но не равных $k1$ и $k2$, обязательно будет выполняться условие $h(k1)\neq h(k2)$. Эта функция обладает сильным свойством рассеивания скученностей.

Методы разрешения коллизий

При работе с хеш-таблицей выделяются три основные операции: вставка, поиск и удаление элемента. Причем, существует круг задач, в которых используются только первые две. При решении задачи следует учитывать набор предполагаемых операций, так как, например, операция удаления может привести к изменению структуры данных, соответствующей используемому методу разрешения коллизий.

Методы разрешения коллизий можно разделить на два класса: метод цепочек и метод открытой адресации.

Метод цепочек

В данном методе для разрешения коллизий в каждую запись хеш-таблицы добавляется указатель для поддержания связанного списка. Сами списки могут размещаться как в памяти, принадлежащей хеш-таблице (внутренние цепочки), так и в отдельной памяти (внешние цепочки).

Внешние цепочки

Хеш-таблица представляет собой массив связанных списков размера m (элементы массива обычно имеют индексы от 0 до m-1). После вычисления значения хеш-функции $a=h(k)$ задача сводится к последовательному поиску в a-ом списке.

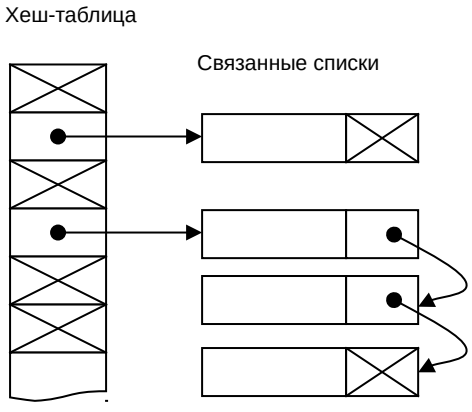

Рис. Метод внешних цепочек

Возможно, что список размещается во внешней памяти. В этом случае для ускорения поиска желательно, чтобы записи с ключами-синонимами при вставке попадали в один кластер файла.

Если число синонимов становится слишком большим, можно вместо линейных списков использовать дерево поиска.

Внутренние цепочки

В методе внутренних цепочек связанные списки для синонимов поддерживаются внутри таблицы. Для поиска свободного места в таблице можно использовать разные методы. Например, последовательный просмотр позиций таблицы.

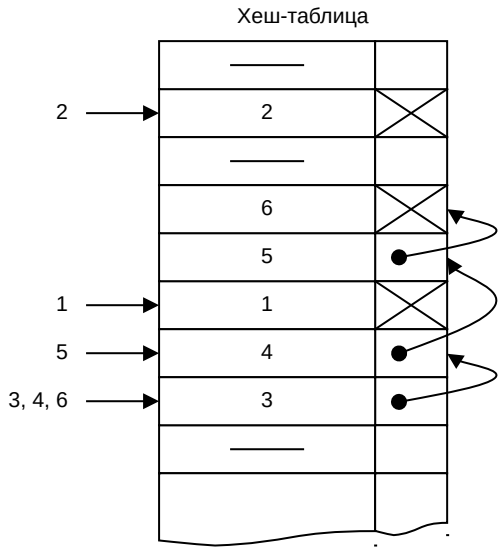

Рис. Метод внутренних цепочек

При вставке новых ключей появляется тенденция объединения хеш-адресов в группы, что проявляется в срастании списков. Таким образом, в один список могут попасть ключи, не являющиеся синонимами по хеш-функции, что удлиняет список. Часто данный метод называют методом *срастающихся цепочек*.

Метод открытой адресации

В данном методе все элементы хеш-таблицы хранятся в одномерном массиве. Если при добавлении нового элемента возникает коллизия, то производится поиск свободного места в следующей ячейке таблицы. Адрес следующей ячейки вычисляется при помощи некоторой функции, аргументами которой, в общем случае, являются: значение ключа, первичный хеш-адрес (полученный при первом применении хеш-функции к текущему ключу), номер шага при поиске свободной ячейки.

Введем обозначения:

k – значение ключа;

$h_0(k)$ – первичный хеш-адрес;

i – номер шага при поиске свободной ячейки, i=1, 2, …;

$h_i(k)$ – значение хеш-адреса, полученного на i-ом шаге.

Тогда алгоритм поиска/добавления элемента будет выглядеть так:

1. Полагаем i=1.
2. Вычисляем $a=h_i(k)$.
3. Если a свободно, то алгоритм завершается (процедура вставки сохраняет элемент в ячейке a, процедура поиска сообщает об отсутствии ключа в хеш-таблице).
4. Если ключ в ячейке a равен k, то алгоритм завершается (процедура вставки повторно не сохраняет элемент в ячейке, процедура поиска сообщает о найденном элементе).
5. Обнаружена коллизия. Полагаем $i:= i$+1 и переходим к пункту 2.

Линейное опробование

В данном методе $h_i(k)=h_0(k)+ci$, где c – константа. В простейшем случае c полагается равной 1 или –1. В этом случае возникает опасность образования скучивания синонимов (в случае использования виртуальной памяти это свойство будет преимуществом метода). Для устранения скучивания c и m (размер хеш-таблицы) должны быть взаимно простыми, а c – и не слишком малым числом.

Квадратичное опробование

В линейном опробовании значение $h_0(k)$, попавшее на элемент последовательности синонимов, всегда удлиняет эту последовательность. Устранить этот недостаток позволяет введение нелинейности в хеш-функцию: $h_i(k)=h_0(k)+ci+di^2$, где c и d – константы.

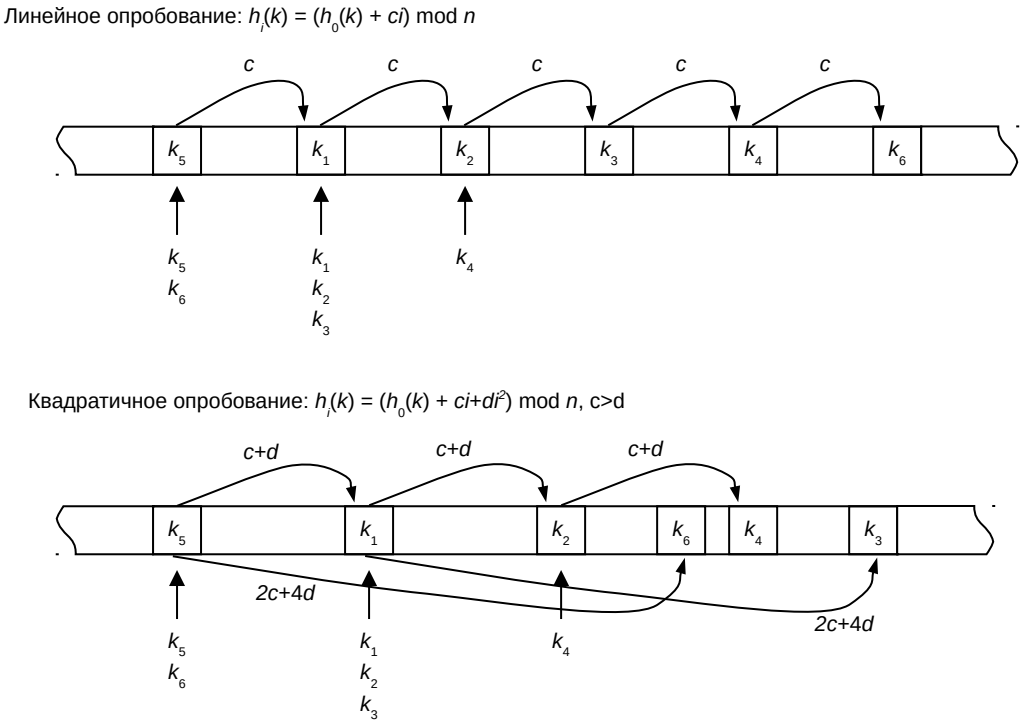

Рис. Пример работы методов линейного и квадратичного опробования

На рисунке приведен пример однотипной ситуации, при которой:

1. $h_0(k_1) = h_0(k_2) = h_0(k_3)$
2. $h_0(k_4) = h_1(k_1)$
3. $h_0(k_5) = h_0(k_6)$
4. $h_1(k_5) = h_0(k_1)$

Как видно из рисунка, в случае линейного опробования возникает 10 коллизий, а в случае квадратичного – 5.

Двойное хеширование

Здесь $h_i(k)=h_0(k)+ig(k)$, где $g(k)$ – хеш-функция, отличная от h_0.

Если m (размер хеш-таблицы) – простое число, то можно рекомендовать такой вид хеш-функции:

$h_i(k)=(k \bmod m) + i\,(1 + k \bmod (m – 2))$.

Для независимых функций h_0 и g вероятность коллизий при двойном хешировании будет составлять $1/m^2$.

Замечание к операции удаления элементов из хеш-таблицы

Во всех рассмотренных методах разрешения коллизий, за исключением метода, основанного на хеш-таблицах с внешними цепочками, удаление элемента из хеш-таблицы

не всегда является операцией, «обратной» вставке, поскольку при удалении нарушается связь между синонимами. Чтобы этого не произошло, можно вместо ключа, ставшего ненужным, не производя удаления записи, вписать специальный код, который во время поиска пропускается, но разрешает использовать эту позицию для вставки нового элемента. Таким образом, каждая позиция хеш-таблицы может находиться в одном из трех состояний: свободном, занятом или удаленном. Нужно учитывать, что если не предпринимать мер по переводу позиций из состояния «удалено» в состояние «свободно», то при постоянном изменении хеш-таблицы может наступить ситуация, при которой не останется свободных ячеек. Скорость поиска заметно ухудшится и будет соответствовать линейному поиску элемента в неупорядоченной последовательности.

Интерфейс модуля HashTable (хеш-таблица)

Определим интерфейс модуля, предоставляющего набор методов для работы с хеш-таблицей:

```
unit HashTable;
interface

   // предполагается наличие следующих типов:
   // THashTable – тип «хеш-таблица»
   // TKey – тип ключа
   // TItem – тип элемента данных

   // создание хеш-таблицы заданного размера size
   procedure Create (size: integer; var t: THashTable);

   // получение размера хеш-таблицы
   function GetSize (t: THashTable): integer;

   // удалить все элементы в хеш-таблице
   procedure Clear (var t: THashTable);

   // добавление элемента в хеш-таблицу.
   // Функция возвращает значение true в случае успешности выполнения операции вставки
   function Insert (var t: THashTable; k: TKey; i: TItem): boolean;

   // поиск элемента данных по ключу.
   // Функция возвращает значение true в случае успешности выполнения операции поиска
   function Find (t: THashTable; k: TKey; var i: TItem): boolean;

   // удаление элемента по заданному ключу.
   // Функция возвращает значение true в случае успешности выполнения операции удаления
   function Delete (var t: THashTable; k: TKey): boolean;

implementation
…
end.
```