Хеш-таблицы

Хеш-таблица — это структура данных, реализующая интерфейс ассоциативного массива, то есть она позволяет хранить пары вида "ключ-значение" и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу. Хеш-таблица является массивом, формируемым в определенном порядке хеш-функцией.

Хеширование (hashing) — это преобразование входного массива данных определенного типа и произвольной длины в выходную битовую строку фиксированной длины. Такие преобразования также называются хеш-функциями или функциями свертки, а их результаты называют хешем или хеш-кодом.

Хеш-таблица содержит некоторый массив H, элементы которого есть пары (хеш-таблица с открытой адресацией) или списки пар (хеш-таблица с цепочками). Выполнение операции в хеш-таблице начинается с вычисления хеш-функции от ключа. Получающееся хеш-значение i является индексом в исходном массиве H. Затем выполняемая операция (добавление, удаление или поиск) перенаправляется объекту, который хранится в соответствующей ячейке массива H[i].

Количество хранимых элементов массива, деленное на число возможных значений хеш-функции, называется коэффициентом заполнения хеш-таблицы (load factor) и является важным параметром, от которого зависит среднее время выполнения операций.

Важное свойство хеш-таблиц состоит в том, что, при некоторых разумных допущениях, все три операции (поиск, вставка, удаление элементов) в среднем выполняются за время O(1). Но при этом не гарантируется, что время выполнения отдельной операции мало. Это связано с тем, что при достижении некоторого значения коэффициента заполнения необходимо осуществлять перестройку индекса хеш-таблицы: увеличить значение размера массива H и заново добавить в пустую хеш-таблицу все пары.

Принято считать, что хорошей с точки зрения практического применения является такая хеш-функция, которая удовлетворяет следующим условиям:

- функция должна быть простой с вычислительной точки зрения;
- функция должна распределять ключи в диапазоне выходных значений как можно более равномерно;
- функция не должна отображать какую-либо связь между значениями ключей в связь между значениями адресов;
- функция должна минимизировать число коллизий — то есть ситуаций, когда разным ключам соответствует одно значение хеш-функции.

Такие события коллизий не так уж и редки — например, при вставке в хеш-таблицу размером 365 ячеек всего лишь 23-х элементов вероятность коллизии уже превысит 50 % (если каждый элемент может равновероятно попасть в любую ячейку) — так называемый парадокс дней рождения. Поэтому механизм разрешения коллизий — важная составляющая любой хеш-таблицы.

Если бы все данные были случайными, то хеш-функции были бы очень простые (например, несколько битов ключа). Однако на практике случайные данные встречаются достаточно редко, и приходится создавать функцию, которая зависела бы от всего ключа. Если хеш-функция распределяет совокупность возможных ключей равномерно по множеству индексов, то хеширование эффективно разбивает множество ключей. Наихудший случай — когда все ключи хешируются в один индекс.

При возникновении коллизий необходимо найти новое место для хранения ключей, претендующих на одну и ту же ячейку хеш-таблицы. Причем, если коллизии допускаются, то их количество необходимо минимизировать. В некоторых специальных случаях удается избежать коллизий вообще. Например, если все ключи элементов известны заранее (или очень редко меняются), то для них можно найти некоторую инъективную хеш-функцию, которая распределит их по ячейкам хеш-таблицы без коллизий.

Хеширование полезно, когда широкий диапазон возможных значений должен быть сохранен в малом объеме памяти, и нужен способ быстрого, практически произвольного доступа. Хэш-таблицы часто применяются в базах данных, и, особенно, в языковых процессорах типа компиляторов и ассемблеров, где они повышают скорость обработки таблицы идентификаторов. В качестве использования хеширования в повседневной жизни можно привести примеры распределение книг в библиотеке по тематическим каталогам, упорядочивание в словарях по первым буквам слов, шифрование специальностей в вузах и т.д.

Хеш-таблицы можно использовать не только для реализации множеств, но и для реализации словарей — хранить в хеш-таблице пары (ключ, значение). Оно ещё называется "ассоциативные массивы".

Методы разрешения коллизий

Коллизии осложняют использование хеш-таблиц, так как нарушают однозначность соответствия между хеш-кодами и данными. Тем не менее, существуют способы преодоления возникающих сложностей:

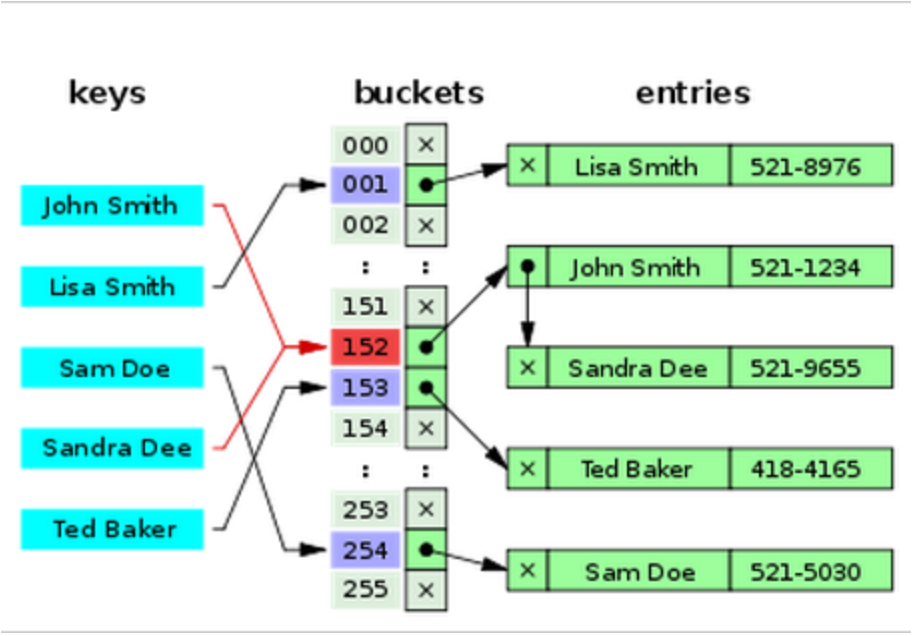
- метод цепочек;
- метод открытой адресации.

Метод цепочек

Каждая ячейка массива H является указателем на связный список (цепочку) пар ключ-значение, соответствующих одному и тому же хеш-значению ключа. Коллизии просто приводят к тому, что появляются цепочки длиной более одного элемента.

Операции поиска или удаления элемента требуют просмотра всех элементов соответствующей ему цепочки, чтобы найти в ней элемент с заданным ключом. Для добавления элемента нужно добавить элемент в конец или начало соответствующего списка, и в случае, если коэффициент заполнения станет слишком велик, увеличить размер массива H и перестроить таблицу.

При предположении, что каждый элемент может попасть в любую позицию таблицы H с равной вероятностью и независимо от того, куда попал любой другой элемент, среднее время работы операции поиска элемента составляет Θ(1 + α), где α — коэффициент заполнения таблицы.



Открытая адресация

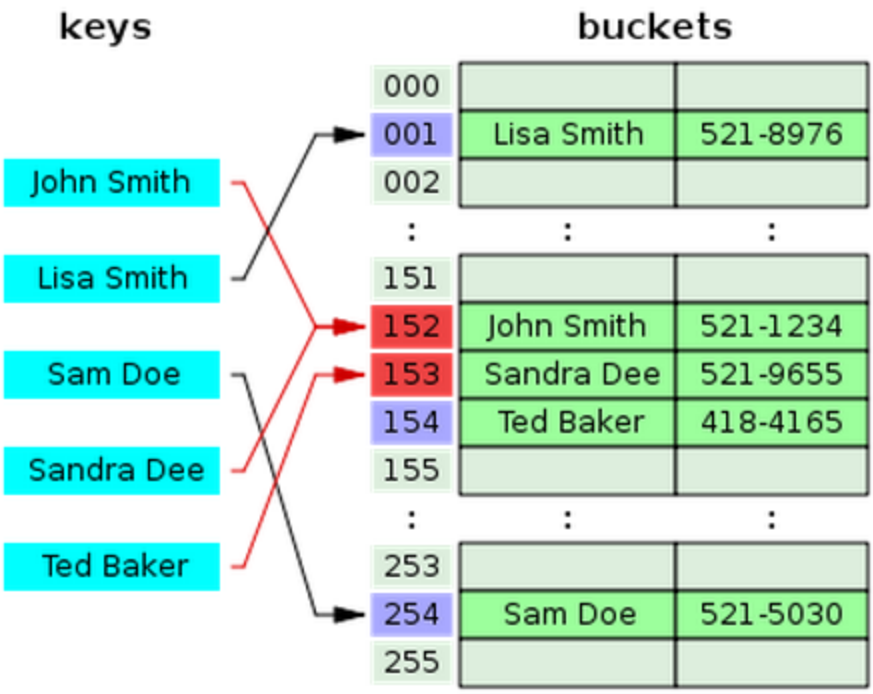
В массиве H хранятся сами пары ключ-значение. Алгоритм вставки элемента проверяет ячейки массива H в некотором порядке до тех пор, пока не будет найдена первая свободная ячейка, в которую и будет записан новый элемент. Этот порядок вычисляется на лету, что позволяет сэкономить на памяти для указателей, требующихся в хеш-таблицах с цепочками.

Последовательность, в которой просматриваются ячейки хеш-таблицы, называется *последовательностью проб*. В общем случае, она зависит только от ключа элемента, то есть это последовательность $h_0(x)$, $h_1(x)$, …, $h_{n-1}(x)$, где x — ключ элемента, а $h_i(x)$ —

произвольные функции, сопоставляющие каждому ключу ячейку в хеш-таблице. Первый элемент в последовательности, как правило, равен значению некоторой хеш-функции от ключа, а остальные считаются от него одним из приведённых ниже способов. Для успешной работы алгоритмов поиска последовательность проб должна быть такой, чтобы все ячейки хеш-таблицы оказались просмотренными ровно по одному разу.

Алгоритм поиска просматривает ячейки хеш-таблицы в том же самом порядке, что и при вставке, до тех пор, пока не найдется либо элемент с искомым ключом, либо свободная ячейка (что означает отсутствие элемента в хеш-таблице).

Удаление элементов в такой схеме несколько затруднено. Обычно поступают так: заводят булевый флаг для каждой ячейки, помечающий, удален ли элемент в ней или нет. Тогда удаление элемента состоит в установке этого флага для соответствующей ячейки хеш-таблицы, но при этом необходимо модифицировать процедуру поиска существующего элемента так, чтобы она считала удалённые ячейки занятыми, а процедуру добавления — чтобы она их считала свободными и сбрасывала значение флага при добавлении.



Последовательности проб

Ниже приведены некоторые распространенные типы последовательностей проб. Нумерация элементов последовательности проб и ячеек хеш-таблицы ведётся от нуля, а N — размер хеш-таблицы (и, как замечено выше, также и длина последовательности проб).

- *Линейное пробирование*: ячейки хеш-таблицы последовательно просматриваются с некоторым фиксированным интервалом k между ячейками (обычно, k = 1), то есть i-й элемент последовательности проб — это ячейка с номером (hash(x) + i*k) mod N. Для того, чтобы все ячейки оказались просмотренными по одному разу, необходимо, чтобы k было взаимно-простым с размером хеш-таблицы.
- *Квадратичное пробирование*: интервал между ячейками с каждым шагом увеличивается на константу. Если размер хеш-таблицы равен степени двойки ($N = 2^p$), то одним из примеров последовательности, при которой каждый элемент будет просмотрен по одному разу, является:
  hash(x) mod N, (hash(x) + 1) mod N, (hash(x) + 3) mod N, (hash(x) + 6) mod N, …
- *Двойное хеширование*: интервал между ячейками фиксирован, как при линейном пробировании, но, в отличие от него, размер интервала вычисляется второй, вспомогательной хеш-функцией, а значит может быть различным для разных ключей. Значения этой хеш-функции должны быть ненулевыми и взаимно-простыми с размером хеш-таблицы, что проще всего достичь, взяв простое число в качестве размера, и потребовав, чтобы вспомогательная хеш-функция принимала значения от 1 до N — 1.
- http://en.wikipedia.org/wiki/Cuckoo_hashing

Эффективность хеш-функций

Самое интересное — выбор хеш-функции. Всё это будет хорошо, только если элементы будут распределяться по сегментам равномерно. При этом хеш-функция должна считаться быстро, иначе особого преимущества не будет. Обычно h(x) — это "случайное" значение, практически не зависящее от x.

Например, на строках хеш-функция может быть такой:

```
int h(char *value)
{
 int result = 0;
 for (int i = 0; value[i] != '\0'; ++i)
   result = (result + value[i]) % hashSize;
 return result;
}
```

Т.е. сложили все буквы строки в кольце вычетов по модулю размера хеш-таблицы.

Операторы вставки и удаления элементов имеют среднее время выполнения O(1 + N/B), где B — число классов, а N — число добавляемых элементов. Это всё в предположении, что хеш-функция распределяет элементы равномерно. Та функция для строк выше таковым свойством не обладает, поскольку, например, строки вида A00, ..., A99 все попадут в 29 сегментов из 100. Более равномерное распределение даёт выдирание

средних цифр из квадрата числа:

$h(n) = (n^2 / C) \% B$, где C - такое, что $BC^2 = K^2$, где K — мощность множества, откуда берутся n.

Пример более достойной хеш-функции для строк:

```
unsigned char h = 0;
while (*str)
    h = rand8[h ^ *str++];
return h;
```

Бывает полезно уметь комбинировать значения хеш-функций. Например, есть структура, состоящая из полей, для каждого из которых определены хеш-функции. Для того, чтобы положить её в хеш-таблицу, надо определить хеш-функцию для неё. Обычно это делают, комбинируя хеш-функции для полей, например, исключающим или.

Ещё бывает полезно менять размеры хеш-таблиц на лету. Например, при $N > 2B$ для открытых хеш-таблиц и при $N > 0.9B$ для закрытых хеш-таблиц можно увеличить размеры таблицы вдвое (после чего потребуется перераспределить элементы).