УДК 004.021

Многоуровневые хэш-функции.

Н.Ю. Старичков, Е.А. Петухова

Московский физико-технический институт (государственный университет)

Фирма “1С”

В современном мире, несмотря на растущую производительность компьютеров, эффективность алгоритмов остается одной из ключевых характеристик программного обеспечения. Довольно часто требуется использование неупорядоченных словарей. Базовый подход к реализации неупорядоченных словарей – построение хэш-таблицы. В данной работе мы хотим предложить алгоритм, который позволял бы строить хэш-функции без коллизий для конечных множеств.

Определим задачу более строго. В приложениях часто встречается необходимость каким-либо образом проецировать некое множество A мощности n в множество целых чисел отрезка [0, n]. Например, у нас есть множество строк, которые мы используем для подписей элементов интерфейса нашего приложения. Во время работы программы хотелось бы не передавать эти строки непосредственно, а передавать, например, индекс этой строки в массиве строк. Такой подход отлично работает, если необходимо получать значение лишь из исходного множества по элементу отрезка [0, n]. Действительно, если массив строк – array, то такое преобразование эквивалентно array[i]. Однако такой подход не работает, если нам нужно по строке получить ее индекс. Потому что сложность этой операции сильно возрастает – нам нужно пройти по всему массиву строк, и сравнить искомую строку с каждой из строк в массиве, и при совпадении вернуть индекс найденной строки. Сложность сравнения строк – O(L), где L – длина строки. Также нам необходимо сравнить исходную строку со всеми, имеющимися в массиве, до тех пор, пока мы не найдем совпадение, это потребует O(n) операций сравнения. Если мы сделаем допущение, что все длина всех строк одинакова и равна L, то соответственно, сложность операции в целом будет O(L*n), что неприемлемо при большом количестве таких операций.

Допустим, что возможно предложить подход, при котором строки хранятся в неких структурах данных, которые имеют меньшую асимптотику поиска. Например, в ысбалансированном дереве поиска. Тогда нам потребуется O(n*logn) операций сравнения, т.е. сложность операции поиска будет O(L*n*logn), ноы при этом мы теряем возможность быстро по индексу получить непосредственно строку. Для этого нам придется либо дополнительно хранить

массив строк, либо тратить O(n) операций для прохода по дереву и сравнению индекса в вершине с искомым индексом). Обычно типичным подходом в таких ситуациях все же считается использование хэш-таблицы.

Ключевой элемент алгоритма построения и использования хэш-таблицы – это хэш-функция. Хэш-функция h(x) должна однозначно проецировать элемент x, принадлежащий множеству A, в множество целых чисел отрезка [0, n]. Фактически, алгоритм отображения множества строк в множество целых чисел, описанный выше, является хэш-функцией. Предлагается подход, в котором мы введем дополнительное ограничение, а именно, чтобы хэш-функция вычислялась исключительно по элементу x, без использования каких-либо структур данных и информации о других элементах множества.

Существует множество подходов построения таких функций. Применительно к строкам это функции, учитывающие длину строки, коды символов и т.д. Однако как и в случае со строками, так и с более сложными объектами, такие функции не гарантируют однозначности – такие ситуации называются коллизиямиы. Т.е. могут существовать, например, две разных строки, на которых хэш-фукнция h(x) принимает одно и то же значение. В общем случае избавиться от коллизий невозможно, можно лишь уменьшить их вероятность. Если множество, для которого необходимо построить хэш-функцию, фиксированно, то можно предложить алгоритм, строящий хэши без коллизий. Особо отметим, что предлагаемый в работе алгоритм не зависит от множества хэшируемых элементов как такового. Имеется только одно необходимое требование для корректной работы алгоритма – множество должно быть конечным и фиксированным, т.е. не изменяться в процессе работы с ним.

Алгоритм базируется не на одной хэш-функции, а на однопараметрическом семействе хэш-функций. Если говорить о строках, то в качестве хэша-функции можно использовать функцию, которая рассматривает строку как число в некоей системе счисления с основанием Y и переводит это число в десятичную систему счисления. Y выбирается простым числом. Тогда семейство данных хэш-функций, где каждая конкретная функция задается конкретным значением Y, можно рассматривать как однопараметрическое семейство. Хэш-функции, принадлежащие этому семейству, обозначим как h(Y, x), где Y – параметр, x – объект, для которого считается хэш.

Далее предположим, что у нас есть некое конечное фиксированное множество A, для которого нам необходимо построить хэш-функцию без коллизий. Предполагается, что элементы в множестве A уникальны, т.е. равных элементов нет, и соответственно, т.к. мы хотим избавиться от коллизий, т.е. чтобы разным элементам соответствовали различные значения хэш-

функции, каждому элементу из A необходимо сопоставить уникальный хэш.

Сначала выбирается какая-то базовая хэш-функция из имеющегося однопараметрического семейства – отвечающая параметру Y1. Для каждого элемента из множества A считается значение h(Y1, x). В этот момент отслеживаются коллизии, т.е. значения хэшей, которые одинаковы для нескольких разных элементов множества A, и сами элементы, на которых возникла коллизия. Далее, для этих элементов, которые не “разделились” первой хэш-функцией, выбирается некая другая, ранее не использовавшаяся, хэш-фукнция из однопараметрического семейства, назовем ее h(Y2, x). Считаются значения h(Y2, x) на элементах, которые вызывали коллизии на h(Y1, x). Если среди них есть элементы, которые вызывают коллизию и на h(Y2, x), то для них используется третья хэш-функция h(Y3, x) и т.д., до тех пор, пока все элементы не будут разделены. Тогда каждому из элементов из A будет сопоставлена своя цепочка хэшей. Например, элементам, которые не вызвали коллизий на уровне h(Y1, x), будет сопоставлена цеполчка из одного значения хэша. Элементам, которые были разделены лишь на уровне h(Y3, x), будет соответствовать цепочка h(Y3, x), h(Y2, x), h(Y1, x).

Далее, эти хэши могут быть приведены к единой длине и простой конкатенацией объединены в один хэш. Соответственно, по этому хэшу будет понятно, сколько хэш функций использовались для “выделения” именно этого элемента. Тогда если нужно быстро находить элемент, гарантированно принадлежащий множеству A, среди всех элементов этого множества, то это можно сделать с помощью данного алгоритма хэширования, без нужды в дополнительном непосредственном сравнении элементов в случае совпадения хэшей. Однако, надо заметить, что при проверке некоего элемента x на принадлежность множеству A непосредственное сравнение все равно необходимо, т.к. отсутствие коллизий гарантируется только для элементов множества A, и нет гарантии, что элемент не из множества A не может дать значения хэша, равное значению хэша для элемента из A. Другими словами, при проверке элемента на принадлежность множеству A возможны ложноположительные срабатывания.

Ценность данного алгоритма хэширования в том, что в ситуациях, когда необходимо быстро находить необходимый объект(например, чтобы вызвать его внутренний метод), такой подход позволяет избежать непосредственного сравнения объектов. Например, есть множество сервисов. Имя сервиса кодируется, например, уникальной строкой. Тогда, чтобы быстро найти необходимый сервис, достаточно посчитать хэш от его имени и гарантированно получить необходимый объект.

Сложность данного алгоритма зависит, в первую очередь, от сложности подсчета значения хэш-функции из однопараметрического семейства и качества этих хэш-функций. Т.к.

вторые, третьи и т.д. хэши считаются только для элементов, которые дали коллизию на предыдущих хэш-функциях, то меньшее число коллизий позволит уменьшит число операций по подсчету хэшей. С другой стороны, алгоритм можно настроить так(дав ему соответствующее семейство хэш-функций), чтобы каждая хэш-функция в отдельности считалась очень быстро, но, возможно, с большим количеством коллизий, тогда уровней хэшей будет, конечно же, больше, однако эффективность подобного решения может быть выше.

Также отметим, что возможно использования и другого семейства хэшей – например, совершенно разные по подходу хэш-функции(в контексте строк – берущие за основу длину и берущие за основу символы, четные, нечетные и т.д.), и параметризовать семейство просто индексом в массиве хэш-функций. Однако, чтобы достичь гарантий того, что на любом входном множестве будет построен хэш без коллизий, семейство хэш-функций не должно быть конечным.