Введение в хеш-таблицы

bitsofmind.wordpress.com/2008/07/28/introduction_in_hash_tables/

Yaroslav Pogrebnyak 28.07.2008

Среди всех структур данных, имеющихся в распоряжении у замечательной науки информатики, есть одна, которой многие люди восхищаются больше, чем другими. Это – хеш-таблица (Hash Table), несомненное достижение в области компьютерных наук. Практически все современные языки программирования имеют реализации хеш-таблиц в своих библиотеках. Чаще всего мы работаем с ними в виде словарей (или ассоциативных массивов), представляющих собой контейнеры множества пар ключ-значение.

Так как же устроены хеш-таблицы, почему они удобны, эффективны, и зачем нам или пользоваться? Здесь я привожу краткое введение в хеш-таблицы для тех, кто не знает и по каким-то причинам не хочет или не может изучить классические труды (они приведены в конце статьи). Одна из причин, побудивших меня написать этот текст, так это то, как показал мой опыт, что тема хеш-таблиц достаточно часто встречается в вопросах на собеседованиях ;).

Хеш-таблица представляет собой обобщение обычного массива. Однако в то время как ключом массива может быть только число, для хеш-таблицы им может быть любой объект, для которого можно вычислить хеш-код. Но об этом позже. Итак, интерфейс хеш-таблицы предоставляет нам следующие операции:

- Добавление новой пары ключ-значение
- Поиск значения по ключу
- Удаление пары ключ-значение по ключу

Так, простой пример сессии работы с хеш-таблицой может выглядеть так (пример на Python):

```
>>> a = { "Vasya" : "123-23-12", "Petya" : "223-21-32" }
>>> a["Vasya"]
'123-23-12'
>>> a["Masha"] = "264-32-32"
>>> a["Vasya"] = "000-00-00"
>>> a
{'Vasya': '000-00-00', 'Petya': '223-21-32', 'Masha': '264-32-
32'}
```

Среднее время поиска значения по ключу в хеш-таблице равно *O*(1). Это значит, что в среднем, наш поиск не будет зависеть от количества элементов в хеш-таблице, и будет равен некоторому константному значению. В зависимости от самой внутренней реализации хеш-таблицы, время поиска для наихудшего случая может быть *O*(n), то есть линейно завесить от количества элементов в таблице, либо же оставаться *O*(1).

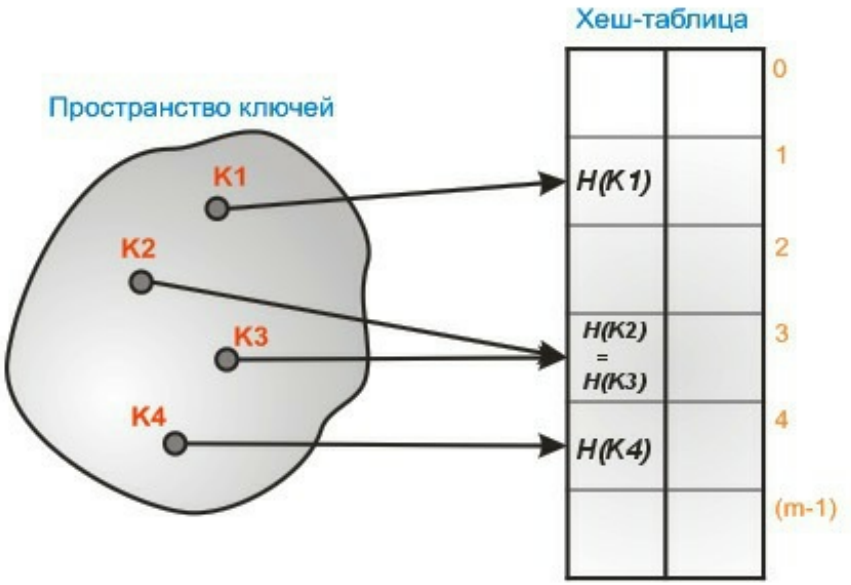
Что же такое хеширование? Идея хеширования основана на распределении ключей в обычном массиве H[0..m-1]. Распределение осуществляется вычислением для каждого ключа элемента некоторой хеш-функции h. Эта функция на основе ключа вычисляет целое число n, которое служит индексом для массива H. Конечно, необходимо придумать такую хеш-функцию, которая бы давала различный хеш-код для различных объектов.

Например, если в качестве ключа хеш-таблицы нам нужно использовать строки, то можно выбрать хеш-функцию, основанную на следующем алгоритме (пример на C):

```
int hash(char* str) {
    int h = 0;
    for (int i=0; i<strlen(str);
i++)
        h = (h*C + ord(str[i])) % m;
}
```

Где m – размер хеш-таблицы, C – константна, большая любого ord(c), а ord() – функция, возвращающая числовой код символа. Для каждого типа данных можно разработать свою хеш-функцию. Однако есть основные требования к хеш-функции: она должна распределять ключи по ячейкам хеш-таблицы как можно более равномерно, и должна просто вычисляться.

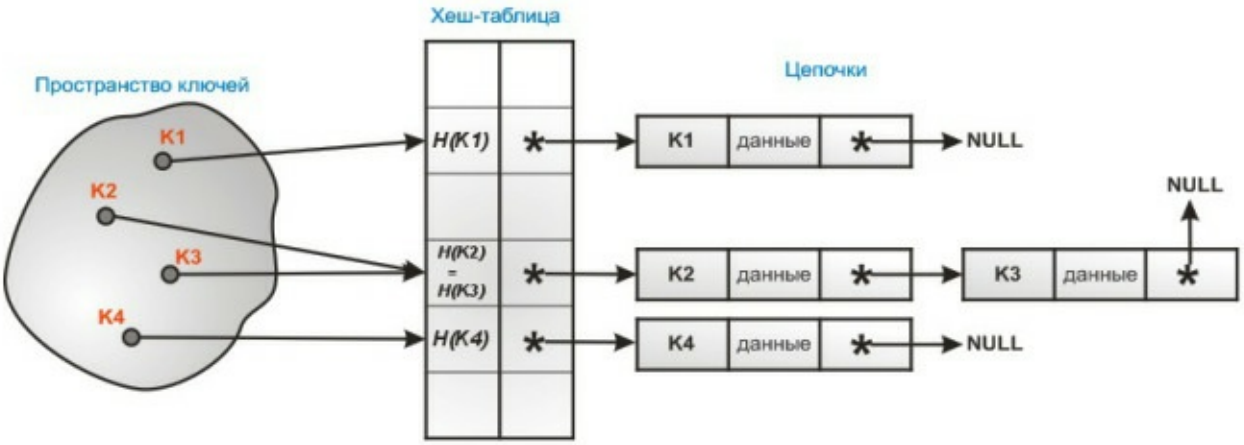
Вот иллюстрация хеш-таблицы. Мы видим, что индексами ключей в хеш-таблице является результат хеш-функции h, применённой к ключу.



Так же этот рисунок иллюстрирует одну из основных проблем. При достаточно маленьком значении m (размера хеш-таблицы) по отношению к n (количеству ключей) или при плохой хеш-функции, может случиться так, что два ключа будут хешированны в одну и ту же ячейку массива H. Такая ситуация называется коллизией. Хорошие хеш-функции стремятся минимизировать вероятность коллизий, однако, учитывая то, что пространство всех возможных ключей может быть больше размера нашей хеш-таблицы H, всё же избежать их вряд ли удастся. На этот случай имеются несколько технологий для разрешения коллизий. Основные из них мы и рассмотрим далее.

Хеширование с цепочками

В случае открытого хеширования (другое название хеширования цепочками), мы объедением элементы, хешированные в одну и ту же ячейку, в связный список. Следующий рисунок иллюстрирует это.



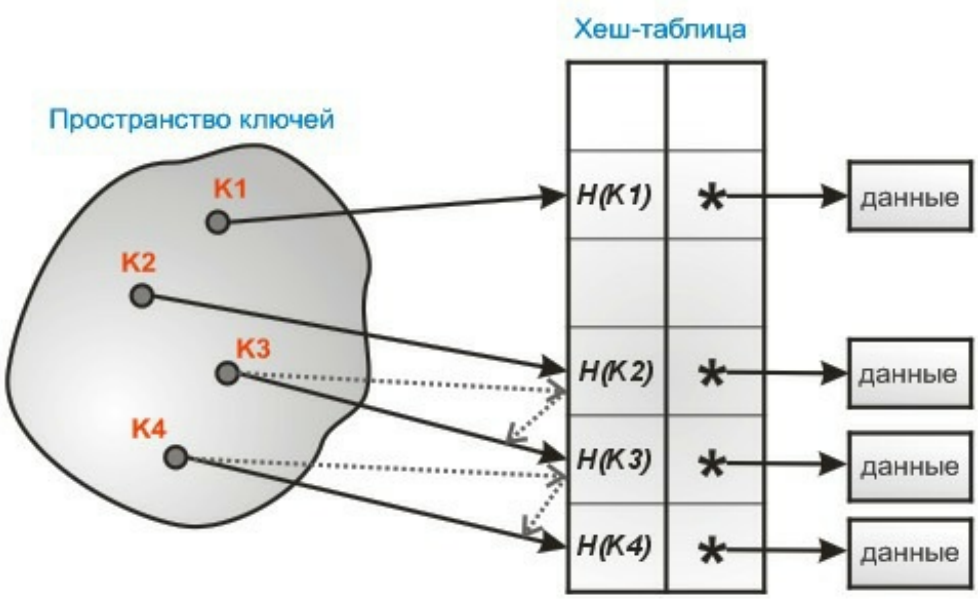
Так, идея достаточно проста. Если при добавлении в хеш-таблицу в заданную ячейку мы встречаем ссылку на элемент связного списка, то случается коллизия. Так, мы просто вставляем наш элемент как узел в список. При поиске мы проходим по цепочкам, сравнивая ключи между собой на эквивалентность, пока не доберёмся до нужного. При удалении ситуация такая же.

Процедура вставки выполняется даже в наихудшем случае за *O*(1), учитывая то, что мы предполагаем отсутствие вставляемого элемента в таблице. Время поиска зависит от длины списка, и в худшем случае равно *O*(n). Эта ситуация, когда все элементы хешируются в единственную ячейку. Если функция распределяем n ключей по m ячейкам таблицы равномерно, то в каждом списке будет содержаться порядка n/m ключей. Это число называется коэффициентом заполнения хеш-таблицы. Математический анализ хеширования с цепочками показывает, что в среднем случае все операции в такой хеш-таблице в среднем выполняются за время O(1). Ссылки на рассуждения и математические выкладки будут даны в конце статьи.

Хеширование с открытой адресацией

В случае метода открытой адресации (или по-другому: закрытого хеширования) все элементы хранятся непосредственно в хеш-таблице, без использования связанных списков. В отличии от хеширования с цепочками, при использовании метода открытой адресации может возникнуть ситуация, когда хеш-таблица окажется полностью заполненной, так что будет невозможно добавлять в неё новые элементы. Так что при возникновении такой ситуации решением может быть динамическое увеличение размера хеш-таблицы, с одновременной её перестройкой.

Для разрешения же коллизий применяются несколько подходов. Самый простой из них – это метод линейного исследования. В этом случае при возникновении коллизии следующие за текущей ячейки проверяются одна за другой, пока не найдётся пустая ячейка, куда и помещается наш элемент. Так, при достижении последнего индекса таблицы, мы перескакиваем в начало, рассматривая её как «цикличный» массив. Иллюстрация этого способа представлена на следующем рисунке:



Линейное хеширование достаточно просто реализуется, однако с ним связана существенная проблема – кластеризация. Это явление создания длинных последовательностей занятых ячеек, которое увеличивает среднее время поиска в таблице. Для снижения эффекта кластеризации используется другая стратегия разрешения коллизий – двойное хеширование. Основная идея заключается в том, что для определения шага смещения исследований при коллизии в ячейке используется другая хеш-функция, вместо линейного смещения на одну позицию.

Одной из сложных вопросов реализации хеширования с открытой адресацией – это операция удаления элемента. Дело в том, что если мы просто удалим некий элемент их хеш-таблицы, то сделаем невозможным поиск ключа, в процессе вставки которого текущая ячейка оказалась заполненной. Так, мы можем помечать очищенные ячейки какой-то меткой, чтобы впоследствии это учитывать.

Практика. Хеширование и ООП-языки

В некоторых популярных объектно-ориентированных языках программирования можно встретить методы вроде GetHashCode() и Equals() в интерфейсе корневого базового класса. Это даёт возможность использовать объекты любого класса в качестве ключей хеш-таблиц. В своих классах мы переопределяем эти методы в зависимости от наших потребностей, так GetHashCode() возвращает хеш-код нашего объекта, основываясь на его внутреннем состоянии. Так, вышеупомянутые методы должны подчинятся некоторым условиям:

- a.Equals(a) == true
- a.Equals(b) == b.Equals(a)
- Если a.Equals(b) и b.Equals(c), то a.Equals(c)
- Переопределяя в потомке Equals, следует переопределить и GetHashCode()
- Если a.Equals(b) == true, то a.GetHashCode() == b.GetHashCode()
- Значение a.GetHashCode() зависит только от внутреннего состояния объекта (его полней и свойств).

Так, реализация хеш-функции GetHashCode() зависит от класса объекта, и здесь мы можем использовать различные техники при расчётах.

Заключение

В статье было дано введение в хеш-таблицы. Однако многие темы остались за «бортом», например,

методология подбора хорошей хеш-функции, и идеальное хеширование, при котором поиск даже в наихудшем случае выполняется за $O(1)$. Так же формальный анализ и доказательства некоторых приведённых утверждений вы можете найти в литературе, обозначенной ниже.

Дополнительная литература

Тема хеширования достаточно широко описана в литературе. Следующие книги я могу рекомендовать в качестве авторитетных и достоверных источников информации:

- Кормен, Лейзерсон, Ривест, Штайн – Алгоритмы. Построение и анализ. Издание 2-е, Вильямс, 2007 г. – Глава 11, страница 282.
- Дональд Э. Кнут – Искусство программирования, том 2. Получисленные алгоритмы, Вильямс, 2007 г. – Глава 6.4.
- Ахо, Хопкрофт, Ульман – Структуры данных и алгоритмы, Вильямс, 2000 г.
- Левитин – Алгоритмы. Введение в разработку и анализ, Вильямс, 2006 г. – Глава 7.3, страница 323.

Пример реализации хеширования с цепочками можно найти в следующей книге:

- Керниган, Пайк — Практика программирования. Издание 4-е, Вильямс, 2004 г. — Глава 2.9, страница 72.