Содержание:

Введение

С хешированием сталкиваются едва ли не на каждом шагу: при работе с браузером (список Web-ссылок), текстовым редактором и переводчиком (словарь), языками скриптов (Perl, Python, PHP и др.), компилятором (таблица символов). По словам Брайана Кернигана, это «одно из величайших изобретений информатики». Заглядывая в адресную книгу, энциклопедию, алфавитный указатель, мы даже не задумываемся, что упорядочение по алфавиту является не чем иным, как хешированием.

Хеширование применяется для быстрого поиска в структурах данных и в криптографии, а также для проверки на наличия ошибок.

Хеширование — это процесс получения уникального (чаще цифрового) идентификатора для объекта.

Хеширование

Хеширование (hash - смешивание, перемешивание,размешивание) — преобразование входного массива данных в короткое число фиксированной длины (которое называется хешем или хеш-кодом) таким образом, чтобы с одной стороны, это число было значительно короче исходных данных, а с другой стороны, с большой вероятностью однозначно им соответствовало.

Преобразование выполняется при помощи хеш-функции. В общем случае однозначного соответствия между исходными данными и хеш-кодом быть не может. Обязательно будут возможны массивы данных, дающих одинаковые хеш-коды, но вероятность таких совпадений в каждой конкретной задаче должна быть сведена к минимуму выбором хеш-функции.

Простым примером хеширования может служить нахождение циклической контрольной суммы, когда берётся текст (или другие данные) и суммируются коды входящих в него символов. Полученное число может являться примером хеш-кода исходного текста.

Это самый простой пример и тут же понятно, что будут коллизии — это когда одно и то же слово даст одно и тоже число. Например, слова дома и мода дадут одну и ту же цифру. Поэтому отсюда выход либо усложнять алгоритм для гарантии неповторимости, а значит важно и расположение букв в слове в плане порядка или при совпадении проверять уже саму строку непосредственно для гарантии того, что слова одинаковые. В любом случае ускорение выполнения значительно по причине того, что сравнивается все за один раз.

Хеш-функция (англ. hash function от hash — «превращать в фарш», «мешанина»), или функция свёртки — функция, осуществляющая преобразование массива входных данных произвольной длины в (выходную) битовую строку установленной длины, выполняемое определённым алгоритмом. Преобразование, производимое хеш-функцией, называется хешированием. Исходные данные называются входным массивом, «ключом» или «сообщением». Результат преобразования (выходные данные) называется «хешем», «хеш-кодом», «хеш-суммой», «сводкой сообщения».

Хеш-функции применяются в следующих случаях:

- при построении ассоциативных массивов;
- при поиске дубликатов в сериях наборов данных;
- при построении уникальных идентификаторов для наборов данных;
- при вычислении контрольных сумм от данных (сигнала) для последующего обнаружения в них ошибок (возникших случайно или внесённых намеренно), возникающих при хранении и/или передаче данных;
- при сохранении паролей в системах защиты в виде хеш-кода (для восстановления пароля по хеш-коду требуется функция, являющаяся обратной по отношению к использованной хеш-функции);
- при выработке электронной подписи (на практике часто подписывается не само сообщение, а его «хеш-образ»);
- и др.

В общем случае (согласно принципу Дирихле) нет однозначного соответствия между хеш-кодом (выходными данными) и исходными (входными) данными. Возвращаемые хеш-функцией значения (выходные данные) менее разнообразны,

чем значения входного массива (входные данные). Случай, при котором хеш-функция преобразует более чем один массив входных данных в одинаковые сводки, называется «коллизией». Вероятность возникновения коллизий используется для оценки качества хеш-функций.

Виды хеш-функций

«Хорошая» хеш-функция должна удовлетворять двум свойствам:

- быстрое вычисление;
- минимальное количество «коллизий».

Введём обозначения:

- K — количество «ключей» (входных данных);
- h(k) — хеш-функция, имеющая не более *M* различных значений (выходных данных).

То есть:

В качестве примера «плохой» хеш-функции можно привести функцию с M=1000, которая десятизначному натуральному числу K сопоставляет три цифры, выбранные из середины двадцатизначного квадрата числа K, Казалось бы, значения «хеш-кодов» должны равномерно распределяться между «000» и «999», но для «реальных» данных это справедливо лишь в том случае, если «ключи» не имеют «большого» количества нулей слева или справа.

Рассмотрим несколько простых и надёжных реализаций «хеш-функций».

«Хеш-функции», основанные на делении

1. «Хеш-код» как остаток от деления на число всех возможных «хешей»

Хеш-функция может вычислять «хеш» как остаток от деления входных данных на M:

$h(k) = k \bmod M$,

где M — количество всех возможных «хешей» (выходных данных).

При этом очевидно, что при чётном ***M*** значение функции будет чётным при чётном ***k*** и нечётным — при нечётном ***k***. Также не следует использовать в качестве ***M*** степень основания системы счисления компьютера, так как «хеш-код» будет зависеть только от нескольких цифр числа ***k***, расположенных справа, что приведёт к большому количеству коллизий. На практике обычно выбирают простое ***M***; в большинстве случаев этот выбор вполне удовлетворителен.

2. «Хеш-код» как набор коэффициентов получаемого полинома

Хеш-функция может выполнять деление входных данных на полином по модулю два. В данном методе ***M*** должна являться степенью двойки, а бинарные ключи ($K=k_{n-1}\ k_{n-2}\ \dots\ k_0$) представляются в виде полиномов, в качестве «хеш-кода» «берутся» значения коэффициентов полинома, полученного как остаток от деления входных данных K на заранее выбранный полином P степени m:

$K(x) \bmod P(x) = h_{m-1}\ x^{m-1} + \dots + h_1 x + h_0$

$h(x) = hm\text{-}1\ \dots\ h1\ h0$

При правильном выборе P(x) гарантируется отсутствие коллизий между почти одинаковыми ключами.

«Хеш-функции», основанные на умножении

Обозначим символом ***w*** количество чисел, представимых машинным словом. Например, для 32-разрядных компьютеров, совместимых с IBM PC, $w = 2^{32}$

Выберем некую константу ***A*** так, чтобы ***A*** была взаимно простой с ***w***. Тогда хеш-функция, использующая умножение, может иметь следующий вид:

В этом случае на компьютере с двоичной системой счисления ***M*** является степенью двойки, и ***h(K)*** будет состоять из старших битов правой половины произведения ***A * K.***

Среди преимуществ хеш-функций, основанных на делении и умножении, стоит отметить выгодное использование не случайности реальных ключей. Например, если ключи представляют собой арифметическую прогрессию (например, последовательность имён «Имя 1», «Имя 2», «Имя 3»), хеш-функция, использующая умножение, отобразит арифметическую прогрессию в приближенно арифметическую прогрессию различных хеш-значений, что уменьшит количество коллизий по сравнению со случайной ситуацией

Одной из хеш-функций, использующих умножение, является хеш-функция, использующая хеширование Фибоначчи. Хеширование Фибоначчи основано на свойствах золотого сечения. В качестве константы ***A*** здесь выбирается целое число, ближайшее к и взаимно простое с w, где φ — это золотое сечение.

Хеширование строк переменной длины

Вышеизложенные методы применимы и в том случае, если необходимо рассматривать ключи, состоящие из нескольких слов, или ключи переменной длины.

Например, можно скомбинировать слова в одно при помощи сложения по модулю ***w*** или операции «исключающее или». Одним из алгоритмов, работающих по такому принципу, является хеш-функция Пирсона.

Хеширование Пирсона — алгоритм, предложенный Питером Пирсоном (англ. Peter Pearson) для процессоров с 8-битовыми регистрами, задачей которого является быстрое преобразование строки произвольной длины в хеш-код. На вход функция получает слово ***W***, состоящее из ***n*** символов, каждый размером 1 байт, и возвращает значение в диапазоне от 0 до 255. При этом значение хеш-кода зависит от каждого символа входного слова.

Алгоритм можно описать следующим псевдокодом, который получает на вход строку ***W*** и использует таблицу перестановок ***T***:

```
h := 0
for each c in W loop
index := h xor c
h := T[index]
end loop
return h
```

Среди преимуществ алгоритма следует отметить:

- простоту вычисления;

- отсутствие таких входных данных, для которых вероятность коллизии наибольшая;
- возможность модификации в идеальную хеш-функцию.

Безопасное хранение паролей с помощью хеш-функций

Пусть есть у нас социальная сеть, база данных или прочее, где может быть вход по логину и паролю. Чтобы система проверила, правильно ли введён пароль, требуется где-то хранить пароли. Хранить пароль в открытом виде

не рекомендуется.

Причины:

- Любой файл можно открыть блокнотом и просмотреть.
- Если хранить в реестре, то также можно проследить обращение программы к реестру и обнаружить то место где хранятся пароли.
- Скомпилированную программу (exe файл) можно вскрыть дизассемблером (например ida pro) и отследить обращение к месту сравнения пароля (например soft-ice)

Таким образом если пароль хранится в открытом виде его легко можно обнаружить.

Чтобы хранить пароль в зашифрованном виде можно прибегнуть к хешированию - использовать хеш-функции с длинными хеш-значениями.

Вместо паролей будем хранить их хеш-значения. Пользователь вводит при входе логин и пароль. В файле по логину ищется нужный хеш. Он сравнивается с хешем того, что введено в поле "пароль" пользователем. Если равны, то пользователя пропускают, иначе - нет.

Если кто-то залезет в файл с данными пользователей (где хранится логин и пароли), вместо паролей он увидит хеши.

Наглядно эту схему защищённого хранения паролей с помощью хеширования можно нарисовать так:

В криптографии традиционно значение f пишут не как обычное десятично число, а в двоичной или 16-ичной системе(026f8e459c8f89ef75fa7a78265a0025), и это называют хеш-значением или просто хешем.

Делфи есть библиотека «IdHashMessageDigest» (если установлен пакет indy) она содержит функции с 128 битным алгоритмом хеширования.

Пример

```
Uses IdHashMessageDigest;

...

function md5(S: string): string;

var md5indy: TIdHashMessageDigest;

hash, HEXhash, Base: string;

begin

md5indy:=TIdHashMessageDigest5.Create;//создаем экземпляр объекта

hash:=StringOf(md5indy.HashString(S)); //получаем MD5-хэш

HEXhash:=md5indy.HashStringAsHex(Base);//тот же хэш, но в HEX-форме

end;
```

Результат: На входе строка на выходе 128 битный хеш.

Здесь метод HashString принимает на входе строку, вычисляет хэш и возвращает его в виде массива байтов (TidBytes). Поэтому дополнительно мы преобразуем этот массив в строку, используя функцию StringOf.

Метод HashStringAsHex также вычисляет MD5-хэш, но в дополнение сразу же его переводит в HEX-форму.

Примечание

MD5 - 128-битный алгоритм хеширования, разработанный профессором Рональдом Л. Ривестом из Массачусетского технологического института в 1991 году.

Контрольная сумма

Контрольная сумма файла (хеш) — это определенное значение, которое рассчитывается по набору данных с использованием определенного алгоритма. Она помогает проверить целостность данных при их хранении и передаче.Если у двух файлов совпадает контрольная сумма, это значит, что эти файлы идентичны по содержанию, даже если по какой-то причине имеют разные названия.

Например, вы скачали файл, а потом выяснили, что он дефектный (к примеру, программа, которой вы пытаетесь его открыть, выдает сообщение об ошибке, хотя остальные файлы этого же формата открывает «на ура»). Как проверить, был ли он дефектным изначально, или же произошли какие-то проблемы при скачивании? Для этого и нужна контрольная сумма файла.

Существуют различные алгоритмы хеширования для создания контрольных сумм. Скажем, программы-архиваторы используют так называемый циклический избыточный код (CRC). Он позволяет удостовериться, что распаковка файла из архива прошла без проблем, а полученный файл идентичен изначальному. Программа BitTorrent использует алгоритм SHA-1, чтобы проверять целостность загружаемых данных. Для проверки целостности скачанных файлов и поиска дубликатов файлов обычно используют алгоритм MD5.

Скажем, вы решили скачать дистрибутив операционной системы. Если при закачке произойдет какой-то сбой, операционная система может установиться «криво» или не установиться вообще. А контрольная сумма поможет определить, совпадает ли скачанный вами файл с изначальным. Для этих целей контрольную сумму обычно указывают на сайте, предоставляющем файлы для закачки. Вам нужно лишь узнать контрольную сумму скачанного вами файла и сравнить два значения. Если контрольные суммы совпадают, файлы идентичны.

Контрольная сумма определяется при помощи специальных программ. Одна из самых распространенных программ для проверки контрольных сумм файлов — HashTab