4.5 Хеширование: теория чисел встречается рандомизацией

В предыдущих параграфах мы видели, как теория чисел даёт «жёсткие» гарантии безопасности: вычислить нужный ответ практически невозможно из-за вычислительной сложности задачи факторизации. Сейчас мы рассмотрим совсем другой тип использования теории чисел — **вероятностные алгоритмы**, в которых правильность работы достигается не «навсегда», а *в среднем*, с очень высокой вероятностью.

Главная мысль: добавив в алгоритм *случайный выбор* (на этапе настройки, ещё до запуска), можно гарантировать, что для *любых* входных данных алгоритм работает в среднем хорошо. Принципиальное отличие от наивного «алгоритм работает хорошо для случайных данных» в том, что входные данные могут быть какими угодно — даже подобраны злоумышленником, — а случайность спрятана *внутри* алгоритма.

Этот параграф следует изложению из классической книги С. Дасгупты, Х. Пападимитриу и У. Вазирани «Алгоритмы»[^1], в котором роль теории чисел особенно прозрачна.

Задача быстрого поиска

Представьте сервис электронной почты, в котором зарегистрировано сто миллионов пользователей. Каждый пользователь идентифицируется адресом, скажем, длиной до 30 символов. При каждом входе нужно *мгновенно* (за время, не зависящее от размера базы) определить: есть ли такой пользователь, и если есть — найти его данные.

Технически: имеется большая «вселенная» возможных ключей U (всех допустимых адресов), и интересующее нас подмножество $S \subseteq U$ из n реальных пользователей. Мы хотим организовать структуру данных, позволяющую за $O(1)$ операций проверять, лежит ли $x \in S$, и получать данные, связанные с x.

Прямая адресация. Самое простое решение — завести массив, индексируемый *всеми* элементами U. Такой массив имеет длину $|\,U\,|$. Если $|\,U\,|$ велико (а оно *колоссально*: количество всевозможных 30-символьных строк — это 26^{30}, что больше 10^{42}), это решение нереализуемо: не хватит памяти всей планеты.

Идея хеширования. Вместо того, чтобы выделять ячейку под каждый возможный элемент, заведём массив размера $m \approx n$ (того же

порядка, что число реальных пользователей), и зададим **хеш-функцию** $h : U \to \{0, 1, ..., m-1\}$, которая каждому ключу сопоставляет номер ячейки.

i Определение 4.25. хеш-функция и хеш-таблица

Хеш-функция — это произвольная функция $h : U \to \{0, 1, ..., m-1\}$. По ней строится **хеш-таблица**: массив $T[0], T[1], ..., T[m-1]$. Ключ $x \in U$ хранится в ячейке $T[h(x)]$.

Если двум разным ключам $x \neq y$ присвоилось одно и то же значение $h(x) = h(y)$, мы говорим, что произошла **коллизия**.

Иллюстрация работы хеш-функции и появления коллизий приведена на рис. 4.16.

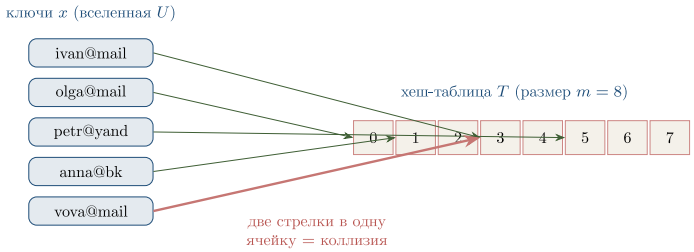

Рисунок 1. Рис. 4.16. Хеш-функция «утрамбовывает» большую вселенную ключей в маленькую таблицу. Иногда возникают коллизии.

Если коллизий нет (или их мало), мы успешно сжали огромную вселенную ключей в небольшую таблицу. Если же коллизий много, эффективность падает: на одну ячейку приходится много ключей, поиск замедляется.

Беда детерминированного хеширования

Зафиксируем какую-нибудь конкретную хеш-функцию — например, $h(x) = x \bmod m$ (в предположении, что ключи x представлены как целые числа). На большинстве входов работает прекрасно. Но злоумышленник, зная нашу функцию, может специально подобрать набор ключей $S = \{0, m, 2m, 3m, ...\}$ — все они дают одинаковый остаток 0. Все они попадут в одну ячейку. Хеш-таблица деградирует до линейного списка, и каждый поиск занимает $O(n)$ вместо $O(1)$.

Этот сценарий не теоретический: реальные сетевые сервисы испытывали атаки именно такого рода — атаки «алгоритмическим усложнением», когда поток специально подобранных запросов приводил к замедлению сервера в тысячи раз.

Важно: принципиальное наблюдение

Для *любой* фиксированной хеш-функции h найдётся «плохой» набор ключей, на котором коллизий очень много. Никакая «гениальная» формула h не спасает.

Спасает другое: *не фиксировать h заранее*, а выбирать её случайно из специального семейства функций.

Универсальное семейство хеш-функций

Идея, восходящая к Картеру и Вегману (1977), состоит в следующем. Договоримся об *целом семействе* хеш-функций $\mathcal{H} = \{h_a\}$. При создании хеш-таблицы выберем из $\mathcal{H}$ *случайную* функцию h_a. Эта функция и будет использоваться. Поскольку выбор делается уже после того, как зафиксированы данные (или независимо от них), злоумышленник заранее не знает, какая h_a будет применена, и не может «подстроиться» под неё.

Чтобы это сработало, семейство $\mathcal{H}$ должно обладать специальным свойством.

i Определение 4.26. универсальное семейство

Семейство хеш-функций $\mathcal{H} = \{h : U \to \{0, \dots, m-1\}\}$ называется **универсальным**, если для любых двух различных ключей $x, y \in U$ $(x \neq y)$ вероятность коллизии при случайном выборе $h \in \mathcal{H}$ не превосходит $1/m$:

$$\Pr_{h \in \mathcal{H}}[\, h(x) = h(y) \,] \leq \frac{1}{m}. \tag{1}$$

Поясним: если бы h присваивала каждому ключу совершенно случайный номер ячейки из m возможных, вероятность совпадения двух конкретных ключей в одной ячейке составила бы ровно $1/m$. Универсальное семейство просит лишь того же порядка — вероятность *не больше* $1/m$.

⚠ Теорема 4.27. оценка коллизий

Если хеш-функция выбрана из универсального семейства, то математическое ожидание числа коллизий заданного ключа x с прочими $n-1$ ключами таблицы не превосходит $(n-1)/m$. В частности, если $m \geq n$, среднее число коллизий не превосходит 1.

Доказательство. Введём индикаторы: $X_y = 1$, если $h(x) = h(y)$, иначе $X_y = 0$. Общее число коллизий ключа x есть $\sum_{y \in S, y \neq x} X_y$. По линейности математического ожидания:

$$\mathbb{E}\left[\sum X_y\right] = \sum \mathbb{E}[X_y] = \sum \Pr[h(x) = h(y)] \leq (n-1) \cdot \frac{1}{m}. \qquad (2)$$

▫

Таким образом, для $m \approx n$ *в среднем* на одну ячейку приходится не более одной–двух коллизий, и время поиска остаётся константным.

Где встречается случайность. Подчеркнём: математическое ожидание берётся *по случайному выбору* h, а не по случайному выбору ключей! Сами ключи могут быть «худшими из возможных» — утверждение всё равно справедливо. Это и есть сила рандомизации.

Конструкция универсального семейства через теорию чисел

Теперь самый интересный вопрос: *как же построить* такое семейство $\mathcal{H}$? И вот тут на сцену выходит теория чисел.

Шаг 1: представление ключа. Будем считать, что каждый ключ $x \in U$ — это *вектор* из k небольших чисел: $x = (x_1, x_2, \ldots, x_k)$. Например, если ключ — это IP-адрес вида 192.168.0.1, его естественно представить как четвёрку чисел $(192, 168, 0, 1)$. Если ключ — это строка, разобьём её на четвёрки или восьмёрки байтов.

Пусть каждое x_i лежит в диапазоне $0 \leq x_i < p$, где p — **простое число**, чуть большее размера наших значений.

Шаг 2: выбор размера таблицы. Положим $m = p$. (В этом разделе для простоты принимаем такое допущение; в общем случае схема легко обобщается.)

Шаг 3: семейство $\mathcal{H}$. Для каждого вектора $a = (a_1, \ldots, a_k)$, где $0 \leq a_i < p$, определим хеш-функцию:

$$h_a(x) = \left(a_1 x_1 + a_2 x_2 + \ldots + a_k x_k \right) \bmod p. \qquad (3)$$

Семейство $\mathcal{H} = \{h_a : a \in \{0, \dots, p-1\}^k\}$. Размер семейства — p^k.

Шаг 4: случайный выбор. При построении таблицы выбираем a случайно равновероятно из $\{0, \dots, p-1\}^k$ (например, генератором случайных чисел).

> ⚠ Теорема 4.28. Картер–Вегман
>
> Описанное выше семейство $\mathcal{H}$ *универсально.*

Доказательство. Возьмём два разных ключа $x = (x_1, \dots, x_k)$ и $y = (y_1, \dots, y_k)$, $x \neq y$. Поскольку $x \neq y$, найдётся хотя бы один индекс i, в котором $x_i \neq y_i$; без ограничения общности будем считать $i = 1$.

Коллизия $h_a(x) = h_a(y)$ означает:

$$a_1 x_1 + a_2 x_2 + \dots + a_k x_k \equiv a_1 y_1 + a_2 y_2 + \dots + a_k y_k \pmod p, \tag{4}$$

или, после перестановки,

$$a_1(x_1 - y_1) \equiv -\sum_{i=2}^{k} a_i (x_i - y_i) \pmod p. \qquad (*) \tag{5}$$

Обозначим правую часть как R. Заметим, что R зависит от $a_2, \dots, a_k$, но *не зависит* от a_1.

Здесь и появляется теория чисел. Поскольку p — *простое*, а $x_1 - y_1 \neq 0 \pmod p$ (по нашему предположению $x_1 \neq y_1$, и оба они в диапазоне $[0, p)$, значит их разность ненулевая по модулю p), у числа $(x_1 - y_1)$ *существует и единственен* обратный элемент $(x_1 - y_1)^{-1} \bmod p$. Это — ровно то свойство модулярной арифметики над простым модулем, которое мы доказали в §4.1 (расширенный алгоритм Евклида).

Следовательно, уравнение $(*)$ имеет ровно одно решение относительно a_1 при фиксированных $a_2, \dots, a_k$:

$$a_1 = R \cdot (x_1 - y_1)^{-1} \bmod p. \tag{6}$$

Среди p возможных значений a_1 только одно приводит к коллизии. Поэтому условная вероятность коллизии (при фиксированных $a_2, \dots, a_k$) равна $1/p$. Усреднив по $a_2, \dots, a_k$, получаем

$$\Pr_a[h_a(x) = h_a(y)] = \frac{1}{p} = \frac{1}{m}. \tag{7}$$

▫

Важно: почему именно простое p

Ключевую роль играет то, что p — *простое*. Будь p составным, у числа $x_1 - y_1$ *могло бы не быть обратного* (если бы оно делилось на общий с p множитель), и тогда уравнение $a_1 \cdot (x_1 - y_1) \equiv R \pmod p$ либо не имело бы решения, либо имело несколько — и аккуратной оценки $1/p$ не получилось бы.

Так теорема единственности обратного элемента по простому модулю, открытая в античности, оказывается фундаментом современных структур данных, ежесекундно работающих во всех серверах мира.

Пример: маленькая хеш-таблица

Пусть простое $p = 7$ (значит, и $m = 7$), а ключи — двойки чисел из $\{0, 1, \ldots, 6\}$, то есть $k = 2$. Возьмём пять «настоящих» ключей:

$$S = \{(1,2), (3,4), (5,5), (2,6), (4,1)\}. \tag{8}$$

Случайно выбираем $a = (a_1, a_2)$. Например, пусть случайный генератор выдал $a = (3, 5)$. Тогда $h_a(x) = (3x_1 + 5x_2) \bmod 7$:

ключ x	$3x_1 + 5x_2$	$\bmod 7$
$(1,2)$	$3 + 10 = 13$	6
$(3,4)$	$9 + 20 = 29$	1
$(5,5)$	$15 + 25 = 40$	5
$(2,6)$	$6 + 30 = 36$	1
$(4,1)$	$12 + 5 = 17$	3

Получаем коллизию: $(3,4)$ и $(2,6)$ попали в ячейку 1. Если бы случайный генератор выдал другое a, например $a = (2,1)$, картина была бы другой:

ключ x	$2x_1 + x_2$	$\bmod 7$
$(1,2)$	4	4
$(3,4)$	10	3
$(5,5)$	15	1
$(2,6)$	10	3
$(4,1)$	9	2

Снова коллизия (правда, в другой ячейке: $(3,4)$ и $(2,6)$ дали 3). Удивительно: эти два ключа коллидируют при разных a. Дело в том, что

$(3,4)-(2,6)=(1,-2)$, и многие a удовлетворяют $a_1-2a_2\equiv 0 \pmod 7$. Однако в среднем по всем a вероятность коллизии любых двух конкретных ключей — ровно $1/7$, как и обещает теорема.

Применения хеширования

Универсальное хеширование — одна из самых востребованных конструкций в современной информатике. Вот лишь несколько мест, где оно работает:

- **Поиск дубликатов и быстрая выборка.** Все базы данных, все таблицы соответствий в операционных системах, все словари в языках программирования (Python: dict, JavaScript: Map, Java: HashMap) построены на хешировании.
- **Проверка целостности файлов.** Контрольные суммы при скачивании файла (CRC, MD5, SHA) — по сути, хеш-функции. Если хеш скачанного файла не совпадает с заявленным, файл повреждён.
- **Цифровая подпись.** Как мы видели в §4.5, RSA-подпись на самом деле ставится не на весь документ, а на его хеш.
- **Антивирусы и системы обнаружения дубликатов.** База вирусных сигнатур — это хеши известных вредоносных файлов; чтобы проверить, не заражён ли файл, антивирус считает его хеш и ищет в базе.
- **Blockchain.** В криптовалютах хеш каждого блока включается в следующий блок, образуя цепочку, в которой невозможно незаметно подправить «прошлое».

Это интересно: криптографические хеш-функции

В этом параграфе мы говорили о «простых» хеш-функциях, минимизирующих коллизии. В криптографии используется родственное, но более сильное понятие — **криптографическая хеш-функция**. От неё требуется не только малое число коллизий, но и *вычислительная невозможность найти* два разных входа с одинаковым хешем (*collision resistance*), а также найти прообраз заданного хеша (*preimage resistance*). Примеры таких функций — SHA-256, BLAKE2, в России — «Стрибог» (ГОСТ Р 34.11–2012). Внутри они устроены сложнее (и не на основе модулярной арифметики), но используют тот же общий принцип: смешать входные биты так, чтобы малейшее изменение во входе приводило к радикальному изменению в выходе.

! Задачи для самостоятельной работы

1. Для простого $p = 5$, $k = 1$ и $a = 3$ найдите $h_a(x) = ax \bmod p$ для всех $x \in \{0, 1, 2, 3, 4\}$.
2. Пусть ключи — пары (x_1, x_2), $p = 11$, $a = (4, 7)$. Найдите коллидирует ли пара $(1, 3)$ с парой $(8, 1)$ при таком a.
3. Объясните, что произойдёт с оценкой $1/m$, если в семействе Картера–Вегмана взять p составным (например, $p = 6$).
4. Почему в формуле хеш-функции $h_a(x) = (a_1x_1 + \ldots + a_kx_k) \bmod p$ нет «свободного члена» a_0, как в линейной функции $ax + b$? Что изменится, если его добавить?
5. * Покажите, что если выбирать a всего лишь из множества $\{1\}$ (одно значение), то семейство не универсально.
6. * Покажите, что вероятность того, что у заданного ключа *не будет* коллизий ни с каким из $n - 1$ других ключей в таблице размера $m \geq n$, не меньше $1/2$ (используйте неравенство Маркова).