Алгоритмы: дополнительные главы

Константин Макарычев

Содержание

1. Теория вероятностей

1.1. Напоминания

- Вероятностный алгоритм можно себе представлять так: до начала работы выбирается случайный объект (скажем, длинная битовая строка), и потом он используется в алгоритме.

- Случайный объект задаётся *вероятностным пространством* $(\Omega, \mathcal{F}, P)$. Здесь Ω — пространство исходов, которое мы будем считать конечным, $\mathcal{F}$ — некоторое семейство подмножеств Ω, которые называются *событиями*, а P — функция, которая каждому событию ставит в соответствие число. Для таких пространств есть аксиомы Колмогорова, но в конечном случае события — это любые подмножества Ω, каждому исходу из Ω присвоена некоторая неотрицательная вероятность, а $P(X)$ — это сумма вероятностей всех исходов из X.

  Например, если алгоритм использует N случайных битов, то Ω состоит из всех последовательностей N нулей и единиц, и каждый из 2^N исходов имеет вероятность 2^{-N}.

- Для бесконечных Ω — скажем, отрезка $[0, 1]$ с равномерным распределением — ситуация сложнее, уже не все подмножества будут входить в $\mathcal{F}$, а только измеримые (по Лебегу), а вероятность множества — его мера. Для распределения с некоторой плотностью надо брать не меру, а интеграл по (измеримому) множеству от функции плотности.

- Значение *случайной величины* зависит от того, какой исход выбран, то есть случайная величина есть функция $X : \Omega \to \mathbb{R}$ (для конечных пространств — любая). Например, для пространства битовых строк длины N число единиц в строке является случайной величиной. Если S — некоторое множество, то вероятность попадания X в S определяется как вероятность множества тех исходов, которые отображаются в S:
$$\Pr\{X \in S\} = \Pr\{\omega : X(\omega) \in S\}$$

- Если $X(\omega)$ — число единиц в трёхбитовой строке ω, то X принимает значение 2 с вероятностью 3/8, потому что есть ровно три строки с двумя единицами.

- Union bound: вероятность события "A или B" не больше суммы вероятностей событий A и B:
$$\Pr(A \cup B) \leqslant \Pr(A) + \Pr(B).$$
  (правая часть может быть больше за счёт одновременного выполнения A и B).

- *Условная вероятность* события A при условии события B определяется как $\Pr(A \cap B)/\Pr(B)$. Для равновероятных исходов это доля A-исходов среди B-исходов. Это имеет смысл при $\Pr(B) > 0$.

- События A и B независимы, если $\Pr(A|B) = \Pr(A)$, то есть $\Pr(A \cap B) = \Pr(A) \cdot \Pr(B)$.

- События $A_1, \ldots, A_n$ *попарно независимы*, если A_i и A_j независимы при любых $i \neq j$, то есть
$$\Pr(A_i \cap A_j) = \Pr(A_i) \cdot \Pr(A_j).$$

- События $A_1, \dots, A_n$ *независимы в совокупности*, если

$$\Pr\left(\bigcap_{i \in I} A_i\right) = \prod_{i \in I} \Pr(A_i)$$

- Случайные величины $X_1, \dots, X_n$ независимы, если для любых множеств $S_1, \dots, S_n$ выполняется равенство

$$\Pr(X_1 \in S_1, \dots, X_n \in S_n) = \Pr(X_1 \in S_1) \cdot \dots \cdot \Pr(X_n \in S_n).$$

- *Индикатором* события A называется функция $\mathbb{1}_A$, равная единице внутри A и нулю вне:

$$\mathbb{1}_A(\omega) = \begin{cases} 1, \omega \in A; \\ 0, \omega \notin A. \end{cases}$$

- События $A_1, \dots, A_n$ независимы тогда и только тогда, когда

$$\Pr(\mathbb{1}_{A_1} = \sigma_1, \dots, \mathbb{1}_{A_n} = \sigma_n) = \Pr(\mathbb{1}_{A_1} = \sigma_1) \cdot \dots \Pr(\mathbb{1}_{A_n} = \sigma_n),$$

  для всех $\sigma_1, \dots, \sigma_n \in \{0, 1\}$, что эквивалентно независимости индикаторов этих событий.

- Попарная независимость слабее независимости в совокупности (Пример: два независимых бита и их `xor`).

- *Математическое ожидание* случайной величины X определяется как

$$\mathbb{E}[X] = \sum_{\omega \in \Omega} X(\omega) Pr(\omega) = \sum_x \Pr(X = x) \cdot x.$$

  (Среднюю зарплату можно вычислять суммированием по работникам или как сумму произведений каждой зарплаты на долю работников с такой зарплатой.)

- Для бесконечных пространств и величин с бесконечным числом значений для вычисления математического ожидания нужно интегрировать.

- Математическое ожидание *линейно*:

$$\mathbb{E}[\alpha X] = \alpha \, \mathbb{E}[X]$$

  для случайной величины X и числа α, а также

$$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$$

- Как и для union bound, для линейности не нужна независимость (событий/величин).

- *Дисперсия* случайной величины X определяется как средний квадрат отклонения от среднего:

$$\mathrm{Var}[X] = \mathbb{E}[(X - \mathbb{E}[X])^2]$$

- Другое выражение:

$$\mathrm{Var}[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$$

- Для независимых величин X и Y:

$$\mathbb{E}[X \cdot Y] = \mathbb{E}[X] \cdot \mathbb{E}[Y];$$
$$\mathrm{Var}[X + Y] = \mathrm{Var}[X] + \mathrm{Var}[Y].$$

  Независимость существенна: у $X + X$ дисперсия вчетверо больше, чем у X.

- *Условное математическое ожидание* величины X при условии A (все в одном пространстве) определяется как

$$\mathbb{E}[X|A] = \sum_{\omega} X(\omega) \Pr(\omega \in A) = \sum_{\omega \in A} X(\omega) \frac{\Pr(\omega)}{\Pr(A)}$$

  (ограничиваемся исходами в A и корректируем вероятности делением на $P(A)$)

- Можно написать

$$\mathbb{E}[X|A] = \frac{\mathbb{E}[X \cdot \mathbb{1}_A]}{\Pr(A)}$$

- Можно рассмотреть математическое ожидание величины X, при условии что другая величина Y равна k:

$$\mathbb{E}[X|Y = k]$$

  Это математическое ожидание зависит от k, и если в качестве k взять значение Y, то получится новая случайная величина, которая обозначается $\mathbb{E}[X|Y]$. Другими словами, мы усредняем X по множествам уровня величины Y.

- *Неравенство Маркова*: для неотрицательной случайной величины X и любой границы $t > 0$ верна оценка:

$$\Pr\{X \geqslant t\} \leqslant \frac{\mathbb{E}\, X}{t}$$

  (доля людей с зарплатой втрое больше средней не превосходит $1/3$ — иначе уже они дали бы бóльшее среднее; всё это при условии, что зарплата неотрицательна).

- *Неравенство Чебышёва*: для любой случайной величины X и границы $t > 0$ выполнена оценка на вероятность отклонения t или больше:

$$\Pr[|X - \mathbb{E}\, X| \geqslant t] \leqslant \frac{\mathrm{Var}[X]}{t^2}$$

  (применяем неравенство Маркова к квадрату отклонения от среднего)

- Это неравенство позволяет доказать простой вариант *закона больших чисел*: если $X_1, \ldots, X_n$ — независимые одинаково распределённые случайные величины, с математическим ожиданием E (одинаковым), а $S_n = X_1 + \ldots + X_n$, то

$$\Pr\{|S_n/n - E| \geqslant \varepsilon\} \to 0$$

  при $n \to \infty$ для любого фиксированного ε. (Математическое ожидание S_n/n равно E, а дисперсия убывает пропорционально $1/n$.)

1.2. Задачи

1. Есть n независимых случайных величин, принимающих значения 1 и -1 с вероятностью $1/2$. Найдите математическое ожидание $(X_1 + ... + X_n)^k$ при $k = 1, 2, 3, 4$.

2. Последовательность R_n начиается с $R_1 = 1$, $R_2 = 5$, $R_3 = 12$, а затем $R_{n+3} = R_{n+2} + 3R_{n+1} + 5R_n$. Оцените рост R_n (докажите, например, что $R_n < 3^n$.

3. Найдите среднее число неподвижных точек в случайной перестановке (случайной биекции n-элементного множества на себя, все $n!$ вариантов равновероятны).

4. Честную монету бросают n раз (орлы и решки равновероятны, бросания независимы). Оцените сверху вероятность того, что доля орлов будет не менее 60%, показав, что она экспоненциально убывает с ростом n (то есть имеется оценка $O(c^n)$ при $c < 1$). Совет: сравните с другим распределением вероятностей, где вероятность орла 60%.

5. Имеется компрессор, который сжимает n-битовые строки без потерь (преобразует строку в другую, произвольной длины, по которой можно восстановить исходную — так что реально он может некоторые строки и удлинять); для простоты считаем, что на строках другой длины он не определён. «Экономией» для строки k считаем уменьшение длины при сжатии (и нуль, если строка осталась прежней длины или удлинилась). Покажите, что средняя экономия (по всем n-битовым строкам) есть $O(1)$.

2. Хеширование

2.1. Словари

Словарь (dictionary) — структура данных, хранящая пары (ключ, значение), изначально пустая, и поддерживающая операции:

- Add/Insert (key, value): добавить новую пару (ключ, значение) в словарь;
- Find (key): найти значение, соответствующее данному ключу;
- Delete/Remove (key): удалить пару с данным ключом;
- Update (key,value): обновить значение для ключа key, сделав его равным value.

Можно хранить пары в массиве, но поиск ключа долгий (в среднем половина длины массива).

Допустим, мы хотим хранить список пользователей: по идентификатору пользователя надо найти данные о нём. Это типичная задача для баз данных. Обычно для этого используются сбалансированные деревья поиска (скажем, red-black tree, или АВЛ-деревья, на практике при работе с диском удобны B-деревья), можно использовать skip-листы.

Обычно для этих структур данных время логарифмическое по числу ключей.

2.2. Хеш-таблицы

Можно ли придумать что-то быстрее (если структура данных в памяти)? Как, скажем, компилятор Clang (LLVM) хранит таблицу идентификаторов? Он использует хеш-таблицы, которые позволяют выполнять операции описанные выше за время $O(1)$ «в среднем».

Для начала предположим, что нам известно максимальное число m ключей. Создадим массив размера $n \approx m$ из «корзин» (buckets), и будем там хранить пары (key, value). Точнее, мы будем рассматривать вариант, когда в каждой ячейке хранится односвязный список тех пар, которые «положено» хранить в ней.

Пусть есть хеш-функция, то есть какая-то функция h, которая отображает возможные значения ключей в $[0, n)$, и договоримся, что пары с ключом x надо хранить в ячейке номер $h(x)$.

Добавление: узнаём куда и добавляем в соответствующий список (в любое место)

Поиск ключа: знаем, в каком списке искать — и ищем (линейным поиском)

Удаление элемента с ключом: знаем, в каком списке, ищем и удаляем (список односвязный, но всё равно $O(1)$ действий)

Что мы хотим от хеш-функции? Хорошо бы она отображала каждый ключ в свою ячейку («совершенная» хеш-функция). Тогда никаких проблем и $O(1)$ операций. Но это свойство не только функции, но и хранимых ключей — и гарантировать это можно, только если число возможных значений ключей меньше числа ячеек (иногда такое полезно, но редко).

Для любой хеш-функции с большим множеством ключей может оказаться, что все реально встретившиеся ключи попали в одну ячейку, и хеш-таблица превращается в список.

2.3. Случайные хеш-функции

Идея: будем выбирать хеш-функцию случайно в каком-то классе функций. Тогда число действий (при выполнении данной последовательности обращений к структуре) будет случайной величиной, и чтобы оценивать её, нужны предположения о том, как хеш-функция выбирается.

- В модели *случайного оракула* хеш-функция равновероятно выбирается из множества всех функций с n значениями: для каждого ключа x значение $h(x)$ равномерно распределено в $\{0, 1, \ldots, n-1\}$, и при разных x эти случайные величины независимы. Функций таких много, и хранить такую функцию сложнее, чем всю структуру данных, так что это скорее модель для анализа, чем реальный алгоритм.

- Можно взять меньшее семейство, сохранив некоторые полезные свойства. Пусть есть семейство функций H, из которого мы выбираем равновероятно некоторую функцию. Требование *универсальности*: для любой пары ключей $x \neq y$ вероятность коллизии для этой пары и случайно взятой функции из H не больше $1/n$:

$$\forall x \forall y \quad (x \neq y) \Rightarrow \Pr_{h \in H}[h(x) = h(y)] \leqslant 1/n.$$

  Случайный оракул этим свойством обладает: $h(x)$ и $h(y)$ независимы и равномерно распределены на множестве из n элементов и потому равны в точности с вероятностью $1/n$.

  Но можно построить гораздо меньшие универсальные семейства.

2.4. Оценка времени поиска

Пусть даны ключи $x_1, \ldots, x_m$ (различные) и ключ y (входящий или не входящий в этот список). Возьмём случайную хеш-функцию h из универсального семейства и посмотрим, сколько операций понадобится для поиска y после того, как в таблицу будет помещены ключи $x_1, \ldots, x_m$. Это число — случайная величина (на пространстве хеш-функций), для каждого набора ключей своя.

Теорема 1. *Математическое ожидание этой величины есть* $O(\frac{m-1}{n} + 1)$.

Доказательство. Оценим время поиска ключа y. Мы должны вычислить $h(y)$, обычно это несложно, так что считаем это за $O(1)$ операций. После этого надо искать ключ, и это время пропорционально размеру списка (корзины).[1] Поэтому надо понять, каково математическое ожидание размера списка в ячейке $h(y)$ (напомним, что y и x_i фиксированы, а матожидание берётся по h). Это удобно сделать с помощью индикаторов и линейности матожидания: размер списка (число x_i, попавших в ячейку к y) равен

$$\sum_{i=1}^{m} \mathbb{1}(h(x_i) = h(y)),$$

по линейности его математическое ожидание равно

$$\sum_{i=1}^{m} \mathbb{E}(\mathbb{1}(h(x_i) = h(y)),$$

а входящие в сумму математические ожидания индикаторов (то есть вероятности событий) не больше $1/n$, кроме того единственного места, где $x_i = y$, если такое есть — там это математическое ожидание равно 1. Поэтому сумма не больше $O(\frac{m-1}{n} + 1)$ (в константу можно поместить и $O(1)$ действий, о которых мы говорили раньше). В типичном случае $m \leqslant n$, и тогда выражение в скобках не больше 2 — получается $O(1)$ операций (в среднем по h). □

Та же оценка применима и к операциям удаления или добавления.

[1] Строго говоря, ещё нужно $O(1)$ операций на обработку конца списка — даже если список пуст, надо в этом убедиться.

2.5. Rehashing

Что делать, если заранее число ключей неизвестно (компилятор не знает, сколько будет идентификаторов)? Когда число ключей достигает размера таблицы, можно завести новую таблицу, скажем, вдвое большего размера, и все элементы по одному перенести в неё (rehashing). Это дорогое действие (ожидаемое время пропорционально числу элементов). Но зато она выполняется редко, и «амортизированное» число операций по-прежнему линейно.

Пусть мы переписываем таблицу, если число добавленных элементов достигнет степени двойки. Тогда размеры таблиц будут $1, 2, 4, 8, \ldots$ (кончается на степени двойки, большей m), и время на rehashing не больше $1 + 2 + 4 + \ldots + m \leqslant 2m$ (последнюю таблицу уже не надо переписывать). Так что суммарное время остаётся линейным по m.

3. Универсальные семейства: примеры

3.1. Афинные отображения по простому модулю

Пусть ключи выбираются из множества $\{0, 1, \ldots, N-1\}$, а таблица имеет размер $n \ll N$ и индексируется числами из $\{0, \ldots, n\}$.

Как построить универсальное семейство? Теорема Чебышёва (постулат Бертрана) позволяет выбрать простое число p от N до $2N$. В качестве хеш-функции возьмём функцию

$$x \mapsto h_{a,b}(x) = [(ax + b) \bmod p] \bmod n.$$

где a и b — случайные остатки по модулю p, причём $a \neq 0$ (всего $(p-1)p$ равновероятных вариантов.

Теорема 2. *Это универсальное семейство.*

Доказательство. Пусть $x, y \in \{0, 1, \ldots, N-1\}$ — два различных ключа. Надо оценить вероятность коллизии, то есть оценить число пар (a, b), при которых $h_{a,b}(x) = h_{a,b}(y)$.

Пусть сначала нет последней операции (деления на n с остатком). Получаются две случайных величины $u = ax + b \bmod p$ и $v = ay + b \bmod p$ — функции от исхода (a, b) (а ключи x и y фиксированы). Поскольку p простое, то тут всё очень регулярно, остатки по модулю p образуют поле $\mathbb{Z}_p$.

Как распределена пара (u, v)? В терминах линейной алгебры можно сказать, что отображение

$$\begin{pmatrix} a \\ b \end{pmatrix} \mapsto \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} ax + b \\ ay + b \end{pmatrix} = \begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$

является невырожденным линейным оператором (определитель равен $x - y \neq 0$), и потому является взаимно однозначным отображением $\mathbb{Z}_p \to \mathbb{Z}$ в себя, при котором подпространство $a = 0$ (которое мы выбросили) соответствует подпространству $u = v$. Если не верить в линейную алгебру, можно явно решить систему уравнений, и обратное отображение будет

$$\begin{pmatrix} a \\ b \end{pmatrix} = \frac{1}{x - y} \begin{pmatrix} 1 & -1 \\ -y & x \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

Поэтому нам осталось доказать такое утверждение: *берём два случайных различных элемента $\mathbb{Z}_p$; вероятность того, что они сравнимы по модулю n, не больше $1/n$.*

Это следует из того, что для любого элемента $u \in \mathbb{Z}$ вероятность того, что случайно взятый другой элемент сравним с u по модулю n, не больше $1/n$.

Элементы, сравнимые с u, образуют арифметическую прогрессию с шагом n. Сколько из них попадает в интервал $\{0, 1, \dots, p-1\}$? Пусть это

$$u', u'+n, u'+2n, \dots, \cancel{u}, \dots, u'+kn$$

при каком-то k. Тогда их k (было $k+1$ и один вычеркнут), и $kn \leqslant p-1$, то есть $k \leqslant (p-1)/n$. Всего элементов, отличных от u, имеется $p-1$ штук, то есть доля не больше $1/n$, что и требовалось доказать. □

Зачем нам было нужно, чтобы $N \leqslant p \leqslant 2N$? Первое неравенство нужно, чтобы разные ключи были разными элементами поля — а второе не нужно вовсе (но чем меньше p, тем меньше функций в семействе — нужно «меньше случайности»).

Первый шаг — переход к p — можно объяснить так: универсальное семейство останется универсальным, если некоторые ключи выбросить, так что можно начать с большего и «более регулярного» с алгебраической точки зрения множества.

3.2. Случайные линейные операторы

Пусть ключи представляют собой m-битовые строки, а ячейки индексируются d-битовыми строками (так что размер таблицы $n = 2^d$ является степенью двойки).

В качестве хеш-функции возьмём случайное линейное преобразования m-мерного пространства над $\mathbb{Z}_2$ в d-мерное над тем же полем:

$$h_A(x) = Ax$$

(умножение по модулю 2, то есть можно умножить в целых числах, а потом взять по модулю 2).

Теорема 3. *Семейство $h_A : \mathbb{Z}_2^m \to \mathbb{Z}_2^d$ для случайной матрицы A размером $d \times m$ универсально.*

Доказательство. Какая будет вероятность коллизии двух различных ключей x, y? Другими словами, какова вероятность того, что $Ax = Ay$ для фиксированных различных x, y и случайной матрицы A? По линейности это равносильно тому, что $Az = 0$ для вектора $z = y - x$; заметим, что $z \neq 0$, потому что $x \neq y$.

Случайная матрица A состоит из d независимых случайных битовых строк $a_1, \dots, a_d$. Каждая строка, a_i умножаясь на z, даёт одну из d координат вектора z, и координата эта представляет собой XOR нескольких битов из a_i (каких? тех, где у z стоят единицы). Сумма независимых случайных битов (нули и единицы равновероятны) сама представляет собой случайный бит, и потому все d координат вектора Az обращаются в нуль с вероятностью $1/2$ и независимо, так что одновременно это происходит с вероятностью 2^{-d}, что и требовалось. □

Семейство хорошее и простое, но довольно большое (2^{dm} функций, так что нужно dm случайных битов, чтобы задать хеш-функцию из семейства).

3.3. Задачи

1. В криптографии тоже используют хеш-функции, например, хранят не пароли пользователей, а значения какой-то заранее выбранной хеш-функции от этих паролей. Почему в этой схеме не стоит использовать хеш-функции из наших семейств?

2. Приведите пример четырёх случайных величин с двумя значениями, в котором любая тройка величин независима, но однозначно определяет значение отсутствующей величины.

3. Пусть p — простое число, и $k < p$. Постройте пример p случайных величин, равномерно распределённых в $\mathbb{Z}_p$, в котором любые k величин независимы, но однозначно определяют значения остальных $p - k$ величин.

4. Для случайных линейных операторов хеш-функция задаётся случайной матрицей, то есть md битами. Покажите, что можно обойтись меньшим числом битов ($m + d - 1$), сохраняя свойство универсальности. (Совет: рассмотрите матрицы, у которых в любой диагонали, параллельной главной, все элементы одинаковы.)

5. Дано двоичное дерево, в вершинах которого стоят целые числа. Требуется найти максимальное расстояние (по дереву) между вершинами с одинаковыми пометками за время $O(n \log n)$, где n — число вершин дерева. (Возможная мотивация: хорошо ли дерево кластеризует пометки.) Немного более простая задача: время работы $O(nh)$, где h — высота дерева.

6. В n-элементном множестве независимо случайно выбраны два k-элементных подмножества (все подмножества равновероятны). Каково математическое ожидание числа элементов в их объединении?

7. В хеш-таблицу из n ячеек независимо помещают n элементов, каждый элемент с равной вероятностью может попасть в любую ячейку. Ячейка может остаться пустой (ни один не попал) — каково математическое ожидание числа таких ячеек? Более сложный вопрос: назовём максимальным зазором наибольшее число подряд идующих пустых ячеек (считаем, что ячейки пронумерованы от 1 до n); оцените (с точностью до постоянного множителя) математическое ожидание максимального зазора.

4. Совершенное хеширование

Хеш-функция $h: U \to B$ *совершенна*, если нет коллизий, то есть разным ключам соответствуют разные ячейки (инъективное отображение). Существуют только при $|U| \leqslant |B|$, так что это исключительный случай, но если множество U известно заранее (скажем, какой-то набор строк), то такое может встретиться на практике. Есть свободно распространяемая реализация gperf (проект GNU), `https://www.gnu.org/software/gperf/`.

4.1. Таблица с запасом

Пусть для начала свободного места много.

Теорема 4. *Если $|B| \geqslant |U|^2$, то в любом универсальном семействе хеш-функций по крайней мере половина функций совершенные.*

Доказательство. Пусть x, y — пара различных ключей. Вероятность того, что $h(x) = h(y)$ для случайной функции h из семейства (то есть доля функций, у которых в этом месте коллизия), не больше $1/n$, где n — размер $|B|$ — по определению универсального семейства.

Таким образом, каждая пара x, y выбивает $1/n$ долю всех функций семейства, а таких пар $|U|(|U|-1)/2$, то есть меньше $n/2$ (напомним, что $|U|^2 \leqslant n$). Значит, всего выбито не больше половины.

В терминах теории вероятностей: событие "иметь коллизию в данной паре" имеет вероятность не более $1/n$, таких событий не более $n/2$, поэтому объединение имеет вероятность не более $1/2$. □

На практике можно несколько раз попробовать случайную функцию и проверять, не окажется ли она совершенной. Вероятность k неудач будет мала (не больше 2^{-k})).

4.2. Двухуровневое хеширование

Можно искать совершенную хеш-функцию для множества U из n элементов так. Возьмём случайно (в каком-то универсальном семействе) хеш-функцию g с n значениями, скажем, $\{0, 1, \dots, n-1\}$. Вряд ли она будет совершенной, скорее всего будут ячейки, куда попало больше одного элемента. Тогда можно использовать ещё одну хеш функцию, которая разделяет элементы этой ячейки (для каждой ячейки — свою).

Таким образом, у нас есть базовая хеш-функция $g: U \to \{0, \dots, n-1\}$, и для каждого i есть своя хеш-функция h_i, определённая на i-ой корзине (на тех элементах u, у которых $h(u) = i$, и итоговая хеш-функция будет такой:

$$i \mapsto (g(x), h_{g(x)}(x))$$

Тут, конечно, надо будет хранить много информации о функциях h_i (ведь для каждой ячейке своя), и значениями функции будут пары (так что индексировать значениями хеш-функции сложно), но давайте ради интереса оценим, сколько потребуется значений, если для каждой корзины использовать описанный в предыдущем размере метод с числом хеш-значений, равным квадрату размера корзины.

Теорема 5. *Математическое ожидание суммы квадратов размеров корзин для случайной функции универсального семейства (при $|U| = |B| = n$) не больше $2n$.*

Другими словами, математическое ожидание среднего квадрата размера корзины не превосходит 2 — при том, что средний размер корзины в точности равен единице (элементов столько же, сколько ячеек).

Доказательство. Пусть load_i — случайная величина, равная размеру i-й корзины (при случайном выборе функции из универсального семейства). Тогда нас интересует

$$\mathbb{E}\left[\sum_i \mathrm{load}_i^2\right] = \sum_i \mathbb{E}[\mathrm{load}_i^2].$$

Саму величину load_i (размер корзины) можно записать как сумму количеств элементов, пришедших из разных мест:

$$\mathrm{load}_i = \sum_{x \in U} \mathbb{1}\{g(x) = i\}.$$

Тогда

$$\mathrm{load}_i^2 = \sum_{x,y \in U} \mathbb{1}\{g(x) = i\} \cdot \mathbb{1}\{g(y) = i\}$$

Беря математические ожидания и переставляя суммирование, получаем

$$\sum_i \mathbb{E}[\mathrm{load}_i^2] = \sum_{x,y \in U} \sum_i \mathbb{E}[\mathbb{1}\{g(x) = i\} \cdot \mathbb{1}\{g(y) = i\}].$$

Но в последних квадратных скобках записан индикатор события $g(x) = g(y) = i$, и математическое ожидание равно вероятности этого события. При данных x и y и при разных i эти события не пересекаются и в сумме дают событие $g(x) = g(y)$, так что равенство можно продолжить:[2]

$$\ldots = \sum_{x,y \in U} \Pr[g(x) = g(y)]$$

В этой сумме есть n членов, равных единице (при $x = y$), и ещё $n^2 - n$ членов, которые не больше $1/n$ по условию универсальности. Всего не больше $2n$, что и требовалось. □

5. Хеш-таблицы на практике

Любопытно посмотреть, как используется хеширование, например, в проекте Chromium (на котором основаны браузеры Chrome, Edge). Файл `hash.h` содержит описания нескольких хеш-функций. Одна из них, `HashInts32`, кодирует пары 32-разрядных чисел (`unsigned`, то есть чисел от 0 до $2^{32} - 1$):

```
h32(x32,y32) = (h64(x32,y32)*rand_odd64 + rand16*2^16)%2^64 / 2^32
```

(текст из комментариев в коде). Видно, что из `x32` и `y32` собирается 64-битовое число, которое потом умножается на константу `rand_odd64` и прибавляется ещё 16-битовое число `rand16`, сдвинутое влево на 16 битов. Множитель и добавка (название которых намекает на то, что они выбраны случайно, хотя никто свечку не держал, вероятно) «зашиты» в текст программы (hardcoded):

```
uint64_t odd_random = 481046412LL << 32 | 1025306555LL;
uint32_t shift_random =10121U << 16
```

Затем выбираются 32 старших битов получившегося числа.

Другая функция хеширует пару 64-битовых чисел с помощью аналогичных битовых преобразований, включающих четыре 32-битовых «случайных» константы.

Можно надеяться, что эти преобразования выбраны не просто так, а обладают какими-то хорошими свойствами (для случайного выбора констант). Если бы ещё эти константы выбирались после того, как враги решили сломать Chrome, устроив большое число коллизий, то можно было бы надеяться, что вероятность успешной атаки мала. (Что, правда, в реальности не так: эти константы можно узнать до атаки.)

[2] В видео в этом месте вместо g написано h, это опечатка.

6. Сравнение множеств

6.1. Постановка задачи

Пусть имеются два набора чисел $[a_1, \dots, a_n]$ и $[b_1, \dots, b_n]$, хранящиеся в двух файлах. Требуется узнать, совпадают ли эти наборы как (мульти)множества — то есть получаются ли они друг из друга перестановкой. Для простоты будем рассматривать случай, когда все числа различны (и длина одинаковая, иначе ответ сразу отрицательный), то есть надо проверить равенство двух n-элементных множеств. Как это побыстрее сделать?

- Можно отсортировать оба набора и затем сравнить поэлементно. Это требует $O(n \log n)$ операций для сортировки (и ещё $O(n)$ для сравнения), и дополнительная память $O(n)$.
- Можно использовать хеширование. Поместим все элементы первого набора в хеш-таблицу по очереди. После этого проверим, что все элементы второго набора там есть. В наших предположениях этого достаточно: известно, что массивы равной длины и без повторений. Получается время $O(n)$ и память $O(n)$.

Но что делать, если памяти мало? Пусть есть, скажем, терабайтные файлы из 128-битовых чисел, реально ли такое сравнение?

Можно сложить все числа в одном файле в другом — и если суммы разные, то ясно, что файлы разные, а если одинаковые — сказать, что «скорее всего файлы одинаковые». Но тут есть проблема: пусть почему-то файлы содержат парами противоположные числа. Тогда сумма в обоих файлах будет нулевая, и мы не заметим разницы.

Можно улучшить алгоритм, вычисляя не только сумму, но и произведение — но и тут будет аналогичная проблема, если файлы содержат ещё и нули. Можно рассмотреть, помимо суммы и произведения, и другие способы вычислять то, что по-английски называется “sketch”, в нашей задаче — симметричную функцию от элементов массива. В нашем примере

$$\text{sketch}(X) = \left(\sum_i a_i, \prod_i a_i \right).$$

Имея такую функцию, можно объявить файлы равными, если равны их скетчи, и разными, если не равны. Во втором случае ответ гарантированно правильный, но в первом — нет. Возможных файлов больше, чем возможных скетчей (файлы длиннее — иначе смысла в скетчах нет). Поэтому есть два разных файла с одинаковыми скетчами.

Что же делать?

6.2. Вероятностный подход

Можно воспользоваться вероятностным алгоритмом и надеяться на малую вероятность ошибки *для любого входа*, то есть для любых двух списков $[a_1, \dots, a_n]$ и $[b_1, \dots, b_n]$. Более точно: если файлы совпадают, то алгоритм гарантированно скажет, что они равны. Если же файлы не совпадают, алгоритм в принципе может дать «ложно положительный» ответ, но с очень малой вероятностью. (В идеале для скетчей из d битов можно надеяться на вероятность ошибки 2^{-d}.)

Идея: возьмём случайную хеш-функцию, которая для каждого числа x (того размера, как в наших наборах) даёт некоторое число y из какого-то большого диапазона (d-битовое, то есть от 0 до $2^d - 1$). Это не совсем то, что было раньше — когда мы хотели, чтобы хеш-значения были существенно короче, чем аргументы, потому что иначе хеш-таблица не имеет смысла. Но теперь хеш-таблицы не будет, а важна нам именно случайность выбора.

(Практическое замечание: хеш-функцию с такими длинными значениями можно собрать как конкатенацию случайно выбранных более коротких хеш-функций.)

Теперь вместо сумм элементов сравним суммы хеш-значений

$$s_A = \sum_i h(a_i),\ s_B = \sum_i h(b_i).$$

Если они разные, то говорим, что массивы разные (и гарантированно правы), если одинаковые — то говорим, что массивы одинаковые (и возможна ошибка)

Какова вероятность этой ошибки, если хеш-функция выбирается в модели случайного оракула (все возможные функции равновероятны)? Если $A \neq B$, то есть какой-то элемент $a^* \in A \setminus B$ (или наоборот, это симметричный случай). Какова вероятность того, что для случайно выбранной хеш-функции происходит ошибка?

Заметим, что значение $h(a^*)$ входит в одну сумму, но не входит в другую. Можно переписать условие как

$$h(a^*) = \sum_{b \in B} h(b) \ - \sum_{a \in A, a \neq a^*} h(a).$$

Теперь видно, что значение $h(a^*)$ в правой части не встречается (она определяется остальными значениями хеш-функции), и поэтому как случайная величина $h(a^*)$ независима с остальными значениями. Отсюда следует, что вероятность события не больше 2^{-d}. (Не обязательно равна, если мы складываем значения как целые числа, а не по модулю: тогда может быть, что разность справа не попадает в диапазон возможных значений.)

Правда, этот анализ исходит из (совершенно нереалистического) предположения о случайном оракуле (значения хеш-функции во всех точках независимы — тут не хватит универсальности хеширования, или попарной независимости). Хранить такую функцию даже сложнее, чем сами массивы. Но на практике — в приложениях, не связанных с криптографией — можно использовать какие-то семейства хеш-функций меньшего размера, и это обычно сходит с рук.

6.3. Задачи

1. Что делать, если в массивах есть повторяющиеся элементы и их надо сравнивать как мультимножества (равны, если отличаются перестановкой — то есть надо сравнивать числа вместе с кратностями)?

2. Пусть нам надо сравнить две последовательности чисел буквально (а не с точностью до перестановки), но дают их не одновременно, а сначала одну, а потом вторую, и у нас недостаточно памяти, чтобы запомнить первую последовательность. Как решить эту задачу с малой вероятностью ошибки?

3. Корректность алгоритма сравнения массивов сложением скетчей была доказана лишь в предположении случайного оракула, так что для реально используемых хеш-функций мы ничего доказать не можем. Рассмотрим другой алгоритм: выбираем простое число p, много большее n, берём случайный вычет x по модулю p и вычисляем произведения $\prod_i (x - a_i)$ и $\prod_i (x - b_i)$, если совпадают, то считаем массивы равными. Покажите, что вероятность ошибки здесь не больше n/p.

7. Фильтры Блума

7.1. Постановка задачи

Пусть мы хотим хранить множество (обычно подмножество какого-то заранее выбранного «универсума») и выполнять такие операции:

- Add(e): добавить элемент e;
- Remove(e): удалить элемент e;
- Contains(e): принадлежит ли элемент e множеству?

Как это можно сделать? Если универсум состоит из небольшого числа объектов, пронумерованных натуральными числами от 0 до $N-1$, то можно хранить его подмножества как битовые строки. Скажем, атрибуты файла (указывающие, можно ли его читать, менять и т.п.) можно хранить как биты целого числа. Если N велико, понадобится несколько целых чисел — скажем, для хранения множества байтов (чисел от 0 до 255) нужно четыре 64-разрядных целых числа. Это легко эффективно реализовать (и по времени, и по памяти), используя битовые операции — но только если элементами хранимых множеств являются небольшие целые числа.

А что делать, если надо хранить множества строк (скажем, русских слов в некотором документе)? Тогда можно использовать хеш-таблицы или сбалансированные деревья поиска. Как это делать с хеш-таблицами, мы уже видели, и в некоторых предположениях о случайном выборе хеш-функций это даёт время работы $O(1)$ в среднем для всех операций. Главный недостаток — нужно много места. Размер таблицы должен быть порядка числа хранимых слов, и все эти слова нужно хранить (для сравнений).

Но если элементы удалять не требуется, и мы готовы смириться с некоторой небольшой вероятностью ошибки при проверке принадлжености, есть более эффективные способы. (Нечто похожее мы делали раньше при проверке равенства двух множеств сложением значений хеш-функции.) Один из таких способов называется «фильтры Блума».

Пусть можно добавлять элементы, *но нельзя удалять*. Мы хотим проверять принадежность множеству с помощью вероятностного алгоритма Test(x), который работает с односторонней ошибкой:

- Если $x \in S$, то Test(x) = True (наверняка);
- Если $x \notin S$, то Test(x) = False почти наверное (вероятность ответа True мала).

Прежде чем описывать реализацию, обсудим, для чего это может пригодиться. Если мы хотим найти вхождения какого-то слова в архиве документов, можно искать их по очереди в каждом документе. Но это долго. Если документы просмотреть предварительно и хранить для каждого документа список его слов в виде такого фильтра, то можно быстро отбросить большинство документов, где нужного слова нет. При этом мы гарантированно ничего не потеряем, но будут ложные срабатывания: слова нет, а нам скажут, что есть. Но если это редко, то ничего страшного (мы зря просмотрим какие-то документы, но немного).

7.2. Наивная реализация

Будем использовать два известных нам приёма: хранение массива битов (bit set) и хеш-таблицы. Пусть в множестве не более m элементов. Заведём таблицу с некоторым запасом,

взяв $n = mk$ ячеек для некоторого k. (Чем больше k, тем меньше будет вероятность ошибки — но тем больше нужно места.) В отличие от обычной хеш-таблицы, не будем хранить сам элемент x в ячейке $h(x)$, а будем хранить для каждой ячейки один бит: попал туда какой-то элемент или не попал. Другими словами, у нас будет массив битов T (на практике их хранят как разряды в массиве слов), и такие операции:

```
def Add (x):
    id=h(x)
    T[id]=True

def Test(x):
    id=h(x)
    return T[id]
```

Вначале (мы считаем, что множество изначально пустое) все биты в T равны нулю. Ясно, что все добавленные элементы будут правильно распознаны при тестировании, и реализация совсем простая. Единственная проблема в том, что отсутствующий элемент может быть объявлен присутствующим — если его хеш-значение случайно совпало с хеш-значением одного из реальных элементов. Насколько это вероятно?

Если мы храним не более n элементов, то заполнены не более $1/k$ всех ячеек таблицы. Отсюда легко следует оценка ошибки:

Лемма 1. *Фиксируем элементы $x_1, \ldots, x_m$ и некоторое $y \notin \{x_1, \ldots, x_m\}$. Мы добавляем все x_i и потом проверяем y. Тогда вероятность ошибки (ответа True) при использовании хеш-функции из универсального семейства с n значениями не превосходит $1/k$.*

Доказательство. В самом деле, ошибка может произойти только для тех функций, для которых $h(x_i) = h(y)$ при некотором i. Для универсального семейства это событие (при фиксированном i) имеет вероятность не больше $1/n$, поэтому объединение m таких событий имеет вероятность не более $m/n = 1/k$. □

Для простоты в лемме рассматривается случай множества максимального размера, но если в множестве меньше элементов, то вероятность ошибки будет только меньше.

Как можно улучшить этот наивный алгоритм (уменьшить вероятность ошибки, не сильно увеличивая размер таблицы)?

7.3. Более эффективный подход

Что мы недоиспользовали в описанном алгоритме? Битовый вектор был заполнен максимум на $1/k$. Если k больше 2 (а ошибка с вероятностью 1/2 нас вряд ли устроит), то большинство битов будет нулевыми, и мы используем память неэффективно. (Число состояний, в которых заполнено s битов в n-битовом векторе, равно C_n^s, и при s, существенно меньших $n/2$, это очень мало по сравнению с 2^n.)

Будем использовать несколько хеш-функций, выбираемых независимо. Пусть их k (хотя это не совсем оптимально), и алгоритмы операций такие:

```
def Add (x):
    for i=1,...,k:
    T[h_i(x)]=True

def Test(x):
    return (for all i=1,...k:  T[h_i(x)]=True)
```

Другими словами, у нас для каждого элемента есть k хеш-значений, и *все* они добавляются в таблицу (мы устанавливаем единичные биты в k местах битового вектора), при этом

мы не помним, какая именно хеш-функция их добавила. Проверяя принадлежность, мы смотрим на все хеш-значения для данного x. Если хотя бы одно отсутствует, то элемента заведомо не было. Но возможна и ошибка: элемента нет, а все его хеш-значения есть (потому что они были для других элементов). Насколько это вероятно?

Оценим эту вероятность в модели случайного оракула и покажем, что она экспоненциально убывает с ростом k.

Теорема 6. *Фиксируем элементы $x_1, \ldots, x_m$ и некоторое $y \notin \{x_1, \ldots, x_m\}$. Мы добавляем все x_i и потом проверяем y описанным способом. Тогда вероятность ошибки (ответа True) в модели случайного оракула (k случайных функций с n значениями) не превосходит*

$$\left(1 - \frac{1}{e} + \varepsilon_n\right)^k,$$

где $\varepsilon_n \to 0$ при $n \to \infty$, если k много больше n.

Набросок доказательства. В модели случайного оракула каждое хеш-значение (у нас есть $n = mk$ этих значений для $x_1, \ldots, x_m$, и ещё k значений для y) выбирается независимо от остальных. Плохо, если все k последних значений встречались среди n ранее выбранных, и надо оценить вероятность этого.

Можно сказать, что сначала мы n раз красим одну из n клеток (не обращая внимания на то, была ли она закрашена раньше), а потом k раз выбираем случайно клетку — какая вероятность, что все k раз попадётся закрашенная клетка?

Грубо можно рассуждать так. Найдём математическое ожидание числа незакрашенных клеток после n закрашиваний. Это случайная величина, которую мы обозначим F. Её можно записать как $F_1 + \ldots + F_n$, где F_i — индикатор события «i-я клетка не закрашена» (можно сказать, что F_i — это число незакрашенных клеток среди i-й клетки). По линейности достаточно найти математическое ожидание каждой из величин F_i, которое равно вероятности остаться незакрашенной, то есть $(1 - 1/n)^n$, и умножить на n.

Мы знаем, что $(1 - 1/n)^n \approx 1/e$. Помимо этого приближения, сделаем ещё одно допущение (более сомнительное). Будем считать, что число незакрашенных клеток в точности равно своему математическому ожиданию. Тогда вероятность попасть в закрашенную клетку равна $(1-1/e)$, а вероятность того, что это случится k раз подряд при k независимых попытках, равна $(1 - 1/e)^k$.

Наши допущения не в точности верны, поэтому в окончательном результате будет ещё и ε_n, а также ограничение $k \gg n$ (это нужно, поскольку в точной оценке будет и член, экспоненциально убывающий с ростом n и не зависящий от k). □

Несложно доказать строго немного худшую оценку для большего размера таблицы. Предположим, что мы храним множество из не более чем m элементов, используем k функций, но размер таблицы теперь вдвое больше: $n = 2mk$. Тогда закрашенные клетки (их не более чем mk) составляют не больше половины клеток. Когда они уже как-то закрашены, вероятность попасть в закрашенную клетку (при выборе хеш-значений в точке y) не больше $1/2$. Поэтому вероятность сделать это независимо[3] k раз не больше 2^{-k}.

Можно сравнить эту оценку с предыдущей: при том же размере таблицы надо взять вдвое меньшее k, так что честно нужно сравнивать

$$\left(\frac{1}{2}\right)^{k/2} \approx 0.7^k \text{ и } \left(1 - \frac{1}{e}\right)^k \approx 0.63^k,$$

видно, что получается немного хуже.

[3]Формально надо записать вероятность ошибки как среднее по всем возможным множествам закрашенных клеток условной вероятности ошибки при данном множестве, и заметить, что все эти условные вероятности не больше 2^{-k}.

7.4. Задачи

1. Оценка для фильтра Блума была получена в модели случайного оракула. Как модифицировать алгоритм и/или рассуждение, чтобы получить оценку в модели с независимым выбором k хеш-функций?

2. Оба предложенных алгоритма выглядят неоптимально, потому что в таблице ожидается перевес единиц над нулями или наоборот. Предложите изменение параметров и оцените (хотя бы без строгого обоснования) эффект от такого изменения с точки зрения соотношения ошибки и размера памяти.

3. Как можно модифицировать фильтр Блума, чтобы он обрабатывал и удаления элементов, заменив биты на счётчики? Оцените место, которое нужно отводить на каждый счётчик.

8. Быстрая сортировка: задачи

Алгоритм быстрой сортировки n объектов действует так: (а) выбираем случайный объект; (б) сравниваем все объекты с x и делим их на меньшие x и большие x; (в) рекурсивно вызываем алгоритм для каждой группы.

1. Оцените число действий алгоритма в худшем и лучшем случае (максимум и минимум числа сравнений по всем вариантам выбора случайного элемента).

2. Считая, что каждый раз выбор элемента для сравнения производится случайно (равновероятно в группе и независимо от результатов предыдущих выборов), докажите рекуррентную формулу для математического ожидания числа сравнений:

$$T(n) = (n-1) + \frac{2}{n}\sum_{i=0}^{n-1} T(i)$$

   с начальными условиями $T(0) = T(1) = 0$.

3. Докажите по индукции, что $T(n) \leqslant cn\log n$ для некоторого c и для всех n.

4. Рассмотрим другой вариант случайного выбора элементов: для каждого из n объектов выберем случайный «ранг» — равномерно распределённое действительное число в $[0, 1]$. После этого при любом из рекурсивных вызовов будем выбирать для сравнения элемент наименьшего ранга. (Не имеется в виду реализовывать это в алгоритме: это лишь эквивалентное представление вероятностного выбора). Пусть $x_1, \ldots, x_n$ — сравниваемые элементы в порядке возрастания. Покажите, что x_i и x_{i+1} сравниваются наверняка, x_i и x_{i+2} сравниваются с вероятность $2/3$ и вообще x_i и x_{i+k} (при $k \geqslant 1$) сравниваются с вероятностью $2/(k+1)$.

5. Выведите отсюда оценку для среднего числа сравнений

$$T(n) \leqslant 2n \cdot \left(\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\right) \leqslant O(n\log n).$$

6. Докажите, что новый способ случайного выбора элементов эквивалентен старому (даёт то же среднее число сравнений).

7. Покажите, что вероятность того, что быстрый алгоритм сортировки потребует более $dn\log n$ сравнений, не больше $O(1/d)$. Можно ли получить экспоненциально убывающую по d оценку?

9. Биномиальные коэффициенты: напоминание и оценки

- *Число сочетаний* из n по k есть число способов выбрать k разных элементов (без учёта порядка) в n-элементном множестве, то есть число k-элементных подмножеств n-элементного множества. Обозначение: $\binom{n}{k}$ (иногда C_n^k).

- Они появляются в разложении степени бинома

$$(a+b)^n = (a+b)(a+b)\dots(a+b) = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}.$$

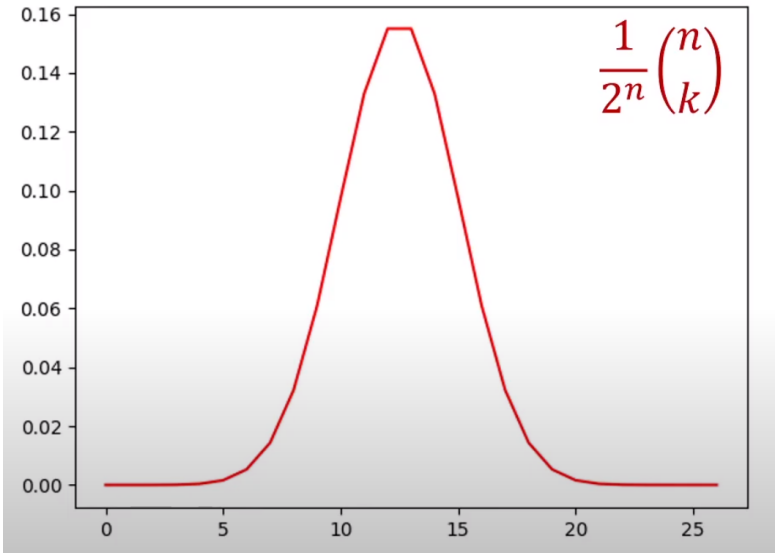
  В самом деле, после раскрытия скобок член $a^k b^{n-k}$ появляется, если мы выберем k скобок, их них возьмём a, а в остальных $n-k$ скобках возьмём b. Коэффициент (количество вхождений такого члена) равен числу способов выбрать эти k скобок, то есть $\binom{n}{k}$.

- *Биномиальное распределение*: выберем неотрицательные $p+q=1$ и будем независимо n раз бросать (кривую) монету X_i с вероятностью получить 1 с вероятностью p (тогда 0 имеет вероятность q). Распределение для числа единиц, то есть для суммы $X_1+\dots+X_n$, называется биномиальным и задаётся формулой

$$\Pr\left(\sum_{i=1}^{n} X_i = k\right) = \sum_{|S|=k} \Pr\left(X_i = 1 \textbf{ if } i \in S \textbf{ else } 0, \text{for all } i\right) = \binom{n}{k} p^k q^{n-k}.$$

  В самом деле, событие «выпало ровно k единиц» можно разбить на $\binom{n}{k}$ непересекающихся событий «единицы выпали на данных местах» (места единиц образуют S), и каждое из этих событий (единицы внутри S, нули вне) имеет вероятность $p^k q^{n-k}$ в силу независимости (оно является пересечением n событий вида «данный бросок дал данный результат»).

- График биномиального распределения с ростом n приближается к графику нормального распределения (на рисунке график при $n=25$).



Это утверждение называется теоремой Муавра–Лапласа (и является частным случаем центральной предельной теоремы про сумму независимых распределений)

- Формула для числа сочетаний:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

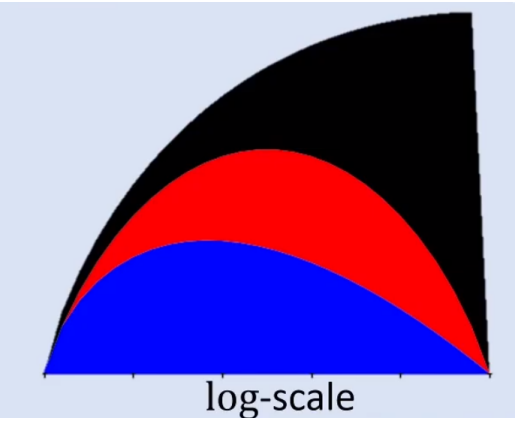
Если мы выбирали бы не просто k-элементные подмножества, а выбирали бы k элементов в некотором порядке (первый, второй,...), то это можно было бы сделать

$$n(n-1)(n-2)\dots(n-k+1) = \frac{n!}{(n-k)!}$$

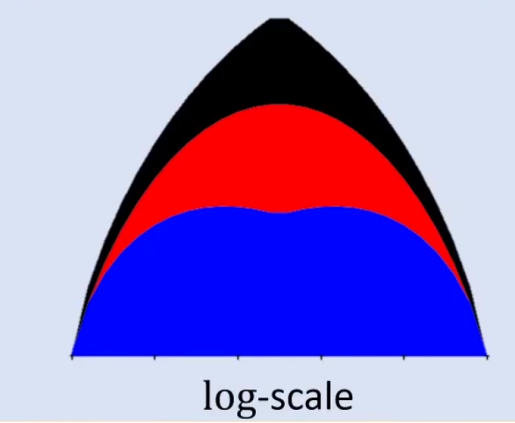
способами (первый элемент можно выбрать n способами, для каждого из них второй элемент можно выбрать $(n-1)$ способами, и так далее). Но при этом мы посчитаем каждое k-элементное подмножество много раз, а именно $k!$ раз (во всех возможных порядках). Соответственно надо ещё поделить на $k!$.

- Удобны следующие (довольно грубые) оценки:

$$\left(\frac{n}{k}\right)^k \leqslant \binom{n}{k} < \left(\frac{en}{k}\right)^k$$



(на рисунке показано их поведение при одном n и разных k, по вертикали откладываются логарифмы всех трёх выражений). Поскольку оцениваемая функция симметрична, то в правой половине лучше заменить k на $k' = n - k$:



- Первую (нижнюю) оценку можно доказать, если заметить, что в произведении

$$\binom{n}{k} = \frac{n(n-1)\dots(n-k+1)}{k(k-1)\dots 1} = \frac{n}{k} \cdot \frac{n-1}{k-1} \cdot \dots \cdot \frac{n-i}{k-i} \cdot \dots \cdot \frac{n-k+1}{1}$$

каждая дробь не меньше $\frac{n}{k}$ (уменьшение числителя и знаменателя на i относительно важнее в знаменателе, так как уменьшаемое меньше — формально мы пользуемся тем, что $(n-i)k \geqslant (k-i)n$).

- Чтобы доказать верхнюю оценку, можно заменить числитель тех же перемножаемых дробей на n и затем показать, что после этого увеличения верхняя оценка выполняется.

$$\binom{n}{k} \leqslant \frac{n^k}{k!} < \left(\frac{en}{k}\right)^k.$$

Второе неравенство (которое осталось доказать) означает, что

$$e^k > \frac{k^k}{k!}$$

и тут можно воспользоваться определением экспоненты как

$$e^k = 1 + k + \frac{k^2}{2} + \frac{k^3}{3!} + \dots + \frac{k^k}{k!} + \dots$$

так что мы сравниваем сумму с одним из её членов (кстати, это один из двух наибольших членов — начиная с него мы увеличиваем числитель в меньшее число раз, чем знаменатель). В других терминах, это нижняя оценка на $k!$: он больше $(k/e)^n$.

- Часто грубых оценок достаточно, но полезно иметь в виду и более точные приближения. Для факториалов это формула Стирлинга

$$k! = \sqrt{2\pi k}\left(\frac{k}{e}\right)^k (1 + o(1))$$

(правая скобка — множитель, стремящийся к единице при $k \to \infty$), а для биномиальных коэффициентов

$$\binom{n}{k} \approx 2^{nH(k/n)},$$

где приближённое равенство понимается с точностью до полиномиального от n множителя (который в логарифмической шкале почти незаметен), а $H(\cdot)$ — функция энтропии Шеннона, которая определяется как

$$H(p) = p\log\left(\frac{1}{p}\right) + (1-p)\log\left(\frac{1}{1-p}\right).$$

10. Балансировка нагрузки (load balancing)

10.1. Постановка задачи

Когда толпы людей заходят на популярный сайт, один компьютер не может их всех обслужить, так что нужно много серверов, на которые эти запросы перенаправляются. Как

определить, на какой из n имеющихся серверов надо перенаправить очередной поступивший запрос? Можно выбирать этот сервер случайно, но тогда из-за случайных флуктуаций некоторые серверы могут быть перегружены, так что это не очень хорошо. Можно выбирать наименее загруженный сервер, но запрашивать у всех серверов, насколько они загружены, долго.[4] Следующая простая идея ("the power of two choices approach") была предложена в статье 1999 года (Azar, Broder, Karlin, Upfal) и используется в `nginx`: мы случайно выбираем два сервера и направляем запрос тому из двоих, кто меньше загружен. Можно подумать, что это улучшит дело раза в два — но на самом деле эффект гораздо больше.

Мы будем рассматривать упрощённую модель с m шарами и n корзинами, в которые мы эти шары помещаем (как при хешировании). В первом варианте мы выбираем для каждого шара корзину случайно и независимо. Как будет распределена максимальная нагрузка, то есть максимальное число шаров в корзине? Мы будем рассматривать случай $m = n$. Оказывается, что с большой вероятностью самая загруженная корзина будет содержать порядка $\log n / \log\log n$ шаров.

Во втором варианте мы для каждого шара выбираем *две* случайных корзины и кладём шар в корзину, где шаров меньше. Тогда с большой вероятностью максимальное число шаров будет порядка всего лишь $\log\log n$ — гораздо меньше.

Хотя реальная ситуация отличается от модели (запросы обрабатываются и сервер готов принимать новые), но и в реальной ситуации выбор двух серверов оказывается существенно более эффективным.

10.2. Случайный выбор

Докажем, что в модели со случайных выбором шаров с большой вероятностью нагрузка всех корзин не превосходит $O(\log n / \log\log n)$.

С точки зрения одной корзины процесс происходит так: на каждом шаге, независимо от предыдущих, добавляется шар с вероятностью $1/n$ (а с вероятностью $1 - 1/n$ ничего не меняется). Другими словами, искомое число есть сумма n независимых случайных величин $X_1, \ldots, X_n$, каждая из которых равна 1 с вероятностью $1/n$ и 0 в остальных случаях. Это бернуллиево распределение «у левого края», и есть теорема Пуассона, которая говорит, что если в бернуллиевом распределении $p = \lambda/n$ для некоторой константы λ, то вероятность значения k стремится (при $n \to \infty$) к

$$\frac{e^{-\lambda}\lambda^k}{k!};$$

это выражение представляет собой долю k-го члена (с λ^k) в сумме ряда

$$e^\lambda = 1 + \lambda + \frac{\lambda^2}{2!} + \ldots + \frac{\lambda^k}{k!} + \ldots$$

Смысл этой теоремы в том, что при больших n и малых p важны не сами n и p, а их произведение $\lambda = np$.

Воспользуемся пока теоремой Пуассона (как если бы она была точным равенством) — у нас $\lambda = 1$ и вероятность обнаружить в корзине k шаров пропорциональна $1/k!$ (с коэффициентом пропорциональности $1/e$, который делает сумму вероятностей равной 1. Нас интересует, насколько быстро эти обратные факториалы убывают (с какого места их появление можно считать маловероятным). Точнее, нам надо узнать, с какого места хвост ряда

[4] Распределять запросы просто по кругу в порядке поступления тоже плохо, так как длительность обработки сильно зависит от запроса.

$\sum 1/i!$ составляет малую долю от всей суммы этого ряда. Члены ряда быстро убывают (мы делим на i при переходе к следующему члену), поэтому хвост ряда не больше последнего оставшегося члена (для геометрической прогрессии с убыванием вдвое было бы равенство). Поэтому можно просто оценивать $1/k!$. Мы знаем, что $k! > (k/e)^k$ (а если бы не было e в знаменателе, то была бы верхняя оценка), и этого достаточно, чтобы понять, что взяв[5]

$$k = c\frac{\log n}{\log\log n}$$

для некоторой константы c, можно получить $1/k! < 1/n$, а для большей константы можно получить и $1/k! < 1/n^2$. В самом деле,

$$\ln k! > \ln(k/e)^k = k\ln k - n = \frac{cn\log n}{\log\log n}[\ln c + \ln\log n - \ln\log\log n].$$

главный член тут $cn(\log n/\log\log n)\ln\log n$, и при подходящем c мы можем сделать это выражение больше n^2 (или вообще любой степени n). При этом c вероятность обнаружить в данной корзине больше $c\log n/\log\log n$ шаров не больше $1/n^2$, и (union bound) вероятность обнаружить превышение *хоть в какой-то* корзине не больше $1/n$.

Таким образом, мы «доказали» (для подходящего c), что с вероятностью как минимум $1-1/n$ максимальное число шаров в корзинах не больше $c\log n/\log\log n$ — в кавычках, потому что мы использовали приближение Пуассона. Чтобы сделать из этого строгое доказательство, есть разные подходы. Можно сравнивать бернуллиевскую величину (выпадение 1 с вероятностью $1/n$) с пуассоновской и показать, что при параметре $(1+1/n)/n$ вторая в некотором смысле мажорирует первую, а потом перенести это на сумму n независимых величин. Можно использовать оценки больших уклонений, о которых будет речь дальше. Но в данном случае проще всего сделать простую явную оценку: вероятность обнаружить в данной урне k шаров равна

$$\binom{n}{k}\left(\frac{1}{n}\right)^k\left(1-\frac{1}{n}\right)^{n-k} \leqslant \frac{n(n-1)\ldots(n-k+1)}{k!n^k} \leqslant \frac{1}{k!},$$

(мы написали формулу для биномиального коэффициента, заменили $1-\frac{1}{n}$ на 1, и, наконец, заменили все множители в числителе на n). Так что искомая оценка легко получается безо всякой теоремы Пуассона.

На самом деле эта оценка близка к точной — можно доказать, что при другой (меньшей) константе c вероятность того, что наиболее заполненная корзина будет содержать не меньше $c\log n/\log\log n$ шаров, больше $1-1/n$. Но этого мы делать не будем (тут нужно немного более сложное рассуждение).

10.3. The power of two choices

Теперь мы рассмотрим другой вариант распределения шаров по корзинам. Есть n корзин и m шаров. Мы кладём шары по очереди: для каждого шара выбираются случайно *две* корзины, и шар кладётся в менее загруженную (где меньше шаров):

```
for t in {1,...,m}
    выбрать два числа i,j в {1,...,n}
    if load(i) <= load(j):
        добавить шар t в корзину i
    else:
        добавить шар t в корзину j
```

[5] В видео описка на слайде: там этой дроби вместо n написано k.

В этом алгоритме предполагается, что числа i и j выбираются случайно и независимо (друг от друга, а также от всех предшествующих выборов). В частности, допустим случай $i = j$.

Этот процесс является упрощённой моделью распределения заданий по серверам (в реальности поток заданий не прерывается, а серверы их обрабатывают и убирают из очереди — а у нас имеется фиксированное число шаров и они накапливаются в корзинах). Эти упрощения позволят нам доказать следующую оценку:

Теорема 7. *Найдётся такое c, что вероятность события «максимальное число шаров в корзине больше $c \log\log n$» для n корзин и n/e^3 шаров стремится к 0 при $n \to \infty$.*

В нашем примере число шаров пропорционально числу корзин, но с малым коэффициентом $1/e^3$. (Ясно, что с ростом числа шаров вероятность переполнения корзин растёт — добавочные шары можно не класть, — поэтому результат верен и для любого меньшего числа шаров.)

$$\boxed{\boxed{\Pr\{\max_i \texttt{load}(i) \geqslant c \log\log n\} \to 0, \quad n \to \infty,\ m \leqslant n/e^3}}$$

На самом деле это утверждение верно и для $m = n$, и вообще для любого коэффициента пропорциональности, важно лишь, что $m = O(n)$ (хотя константа c, конечно, будет другой). Но мы для начала докажем это для $m = n/e^3 \approx n/20$.

Удобно представлять себе процесс так: мы берём граф с n вершинами (корзинами). Изначально в нём нет рёбер, но затем мы по очереди добавляем m рёбер, и каждое ребро соединяет случайно выбранные две вершины (возможно, оно будет петлёй, если вершины совпадут). В конце получится некоторый граф (возможно, с петлями и кратными рёбрами — хотя при нашем выборе параметров их будет мало). Такие случайные графы (и близкие к ним графы Эрдёша — Реньи, когда для каждой пары вершин независимо с вероятностью p возникает ребро между этими вершинами) хорошо изучены, и мы воспользуемся некоторыми вероятностными оценками, отложив их доказательство до следующего раздела. Эти оценки (леммы 2, 3) говорят, что с большой (стремящейся к единице) вероятностью в графе все связные компоненты имеют размер $O(\log n)$, а среднее число вершин в любом индуцированном подграфе не превосходит $O(1)$. Отсюда будет следовать[6], что с большой вероятностью максимальная загрузка вершины (число шаров в корзине) будет не больше $O(\log\log n)$.

Лемма 2. *С вероятностью, стремящейся к 1 при $n \to \infty$, все связные компоненты случайного графа содержат не более $5 \ln n$ вершин.*

В следующей лемме речь идёт про степени вершин в *индуцированном подграфе*. то означает, что выбирается некоторое множество вершин S и оставляются только эти вершины и соединяющие их рёбра (всё остальное, в том числе рёбра, соединяющие S с внешним миром, удаляется). Получается *индуцированный подграф* $G[S]$.

Лемма 3. *С вероятностью, стремящейся к 1 при $n \to \infty$, во всех индуцированных подграфах $G[S]$ (при любом $S \subset \{1, 2, \ldots, n\}$) средняя степень вершины не превосходит 5.*

Согласно этим леммам (и union bound, если быть педантичным), с вероятностью, стремящейся к 1, в случайно построенном в ходе алгоритма графе выполнены оба эти свойства, и остаётся доказать, что эти свойства случайного графа гарантируют, что в каждой

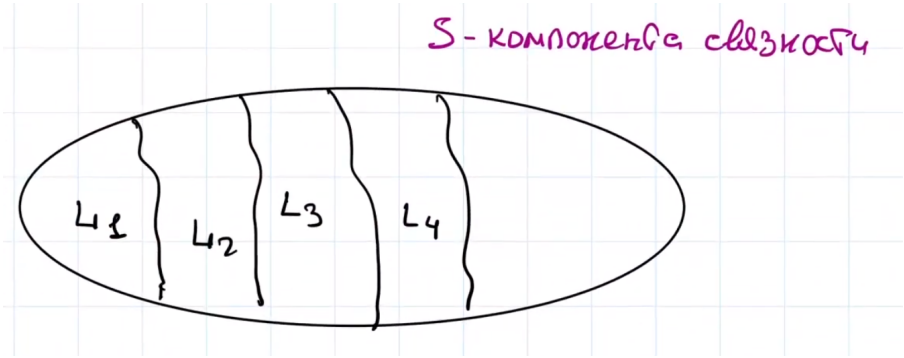
[6]Заметим, что сам по себе граф ещё не определяет загрузку — важно, в каком порядке добавляются рёбра. Но мы увидим, что если эти свойства графа выполнены, то при *любом* порядке добавления рёбер загрузка невелика.

корзине будет не более $O(\log\log n)$ шаров (независимо от того, в каком порядке рёбра появляются).

Вспомним, как происходит наш процесс: мы порождаем рёбра одно за другим, и у каждого ребра выбираем менее загруженный конец (и добавляем туда шар). В частности, вершины с малой степенью нам вообще не опасны: даже если во всех инцидентных с ними рёбрах будут выбраны именно они, шаров много не будет. Ясно и то, что каждую связную компоненту можно рассматривать отдельно (появление рёбер в других связных компонентах вообще на неё не влияет).

В каждой связной компоненте максимальная нагрузка не больше размера компоненты (и даже не больше максимальной степени вершины). Но этого мало — оценка для размера компонент только $O(\log n)$, а нам нужна существенно лучшая оценка. И мы знаем, что вполне могут быть вершины степени $\Omega(\log n/\log\log n)$, так что нужно более детально анализировать происходящее внутри компоненты.

Итак, будем рассматривать одну связную компоненту — которая совпадает с индуцированным графом на множестве её вершин. Средняя степень вершины в этом графе не больше 5 (лемма 3), поэтому вершины степени 10 и выше составляют не больше половины вершин (неравенство Маркова). Остальные вершины имеют степень не больше 9 и для нас безопасны: больше 9 шаров в них не будет точно. Опасные вершины можно расклассифицировать и дальше. Возьмём индуцированный подграф на опасных вершинах, отбросив безопасные вершины и рёбра, в них ведущие. Мы знаем, что после такого отбрасывания средняя степень индуцированного подграфа снова снизится до 5 (а до отбрасывания все вершины имели степень не меньше 10), значит, не меньше половины в нём составляют «относительно безопасные» вершины. Это расслоение можно продолжать. Более формально,



пусть S — компонента связности. В L_1 включим вершины степени меньше 10, а в $S_1 = S \setminus L_1$ — все оставшиеся вершины S. Теперь рассмотрим $G[S_1]$ и степени вершин *в нём*: те вершины, у кого степени меньше 10, включим в L_2, и так далее:

$$S_0 = S; \quad L_i = \{x \in S_{i-1} : \deg x < 10\}, \quad S_i = S_{i-1} \setminus L_i.$$

На каждом шаге размер множества S_i уменьшается по крайней мере вдвое, поэтому число слоёв не больше $\log|S| = \log(5\ln n)$ (вот тот двойной логарифм, который нам нужен — ясно, что $\log(5\ln n) = O(\log\log n)$ при больши́х n). Осталось оценить, сколько шаров может попасть в корзины из L_i: мы увидим, что не больше $20i$, и это завершит доказательство.

С L_1 всё просто: там вообще степени меньше 10. С L_2 надо уже разбираться: если $i \in L_2$, то в корзину i шары могут попадать двояко: из-за рёбер внутри S_1, а также из-за рёбер, соединяющих L_1 с L_2. Рёбер первого типа снова меньше 10 (потому что в $G[S_1]$ все вершины из L_2 имеют степень меньше 10), а вот рёбер второго типа может быть много: после того, как мы перешли к S_1, все эти рёбра были выброшены и не помешали включить i в L_2. Но (ключевой момент!) эти рёбра не могут привести к тому, что число шаров в i превысит 10:

наш алгоритм не будет класть одиннадцатый шар в i, поскольку противоположный конец ребра содержит менее 10 шаров (а в i их уже 10). Значит, увеличивать нагрузку сверх 10 могут только рёбра первого типа, а их меньше 10. Получаем, что число шаров в вершинах из L_2 меньше 20.

Дальше можно рассуждать по индукции: нагрузка вершин в L_i меньше $10i$. В самом деле, покажем, что нагрузка вершин в L_{i+1} меньше $10i+10$, считая, что для всех предыдущих это уже известно. Возьмём вершину из L_{i+1}. Её рёбра делятся на два типа: (1) внутри S_i и (2) рёбра в $L_1, \dots, L_i$. Рёбра второго типа не могут поднять нагрузку i выше $10i$ (потому что нагрузка с другого конца меньше $10i$ по предположению индукции), а рёбер первого типа меньше 10.

Теорема 7 доказана.

10.4. Задачи

1. Оцените вероятности появления петли и кратного ребра при случайном построении графа в теореме 7, показав, что они не превосходят $1/e^3$ и $2(n-1)/n^2$.

2. Попытаемся доказать теорему 7 для случая $m = O(n)$ так: разобьём добавляемые m шаров на порции по n/e^3; добавление каждой порции по нашему алгоритму по доказанному увеличивает нагрузку на $O(\log\log n)$, порций у нас $O(1)$, и всего получается нагрузка $O(1) \cdot O(\log\log n) = O(\log\log n)$, за исключением плохих случаев, которых в $O(1)$ раз больше, чем на одном шаге, поэтому вероятность неудачи стремится к нулю. Что неправильно в этом рассуждении? Как его можно исправить?

10.5. Отложенные оценки вероятности

Нам надо доказать, что в описанном процессе с большой вероятностью не образуется больши́х связных компонент или подмножеств с большой средней степенью в индуцированном графе. И то, и другое происходит, лишь если в индуцированном подграфе оказывается много рёбер: в связном графе с k вершинами не меньше $k-1$ рёбер, а средняя степень вершины равна удвоенному отношению числа рёбер к числу вершин (сумма степеней вершин равна удвоенному числу рёбер). Поэтому мы выведем оба результата из общей оценки типа «маловероятно, чтобы нашлось подмножество из k вершин, между которыми не менее l рёбер», взяв разные значения параметров.

Как оценить такую вероятность? Мы оценим её для одного подмножества S из k вершин и для одного списка из l шагов, на которых рёбра внутри S были проведены. Если нежелательное событие произошло, то это произошло для какого-то S и какого-то списка, поэтому можно применить union bound.

- количество всех k-элементных подмножеств S не больше

$$\binom{n}{k} \leqslant \left(\frac{ne}{k}\right)^k;$$

- количество всех вариантов выбрать l шагов из m возможных не больше

$$\binom{m}{l} \leqslant \left(\frac{me}{l}\right)^l;$$

- вероятность, что все рёбра с данных m шагов попадут в индуцированный на S граф (то есть все их $2m$ концов окажутся внутри S), равна

$$\left(\frac{k}{n}\right)^{2m}.$$

Всего получаем верхнюю оценку

$$\left(\frac{ne}{k}\right)^{k} \cdot \left(\frac{me}{l}\right)^{l} \cdot \left(\frac{k}{n}\right)^{2m}. \qquad (*)$$

Нас она интересует в ситуации $m = n/e^3$. Подставим это значение:

$$(*) = e^{k-2l} \cdot n^{k-l} \cdot k^{2l-k} l^{-l}$$

(легко по очереди проверить степени у каждого из множителей e, n, k, l). Теперь посмотрим, что это даёт для наших лемм. Заметим, что в этой оценке размер индуцированного подграфа фиксирован (равен l), а в наших леммах шла речь о существовании индуцированного подграфа неизвестного размера, так что нам понадобится ещё применить union bound к оценкам для разных l.

Доказательство леммы 2. Если в нашем случайном графе есть компонента из $k \geqslant 5\ln n$ вершин, то в этой компоненте не менее $l = k - 1$ рёбер. Какова вероятность такого события (наличия k вершин, между которыми не менее $l = k - 1$ рёбер) при данном k (и при $m = n/e^3$)? Используем нашу оценку при $k \geqslant 5\ln n$ и $l = k - 1$. Тогда выражение $(*)$ не превосходит

$$e^{k-2l} \cdot n^{k-l} \cdot k^{2l-k} l^{-l} = e^{2-k} \cdot n \cdot k^{k-2} l^{1-k} \leqslant \frac{e^2}{n^5} n \cdot \frac{1}{k} (k/l)^l < e^3/n^4.$$

(Мы выбрасываем множитель $1/k$ и пользуемся тем, что $(k/l)^l = (1 + 1/l)^l < e$.) Видно, что эта вероятность стремится к нулю, даже если просуммировать её по всем возможным значениям размера связной компоненты от $k = 5\ln n$ до $k = n$ (их не более n штук, а в знаменателе у нас n^4). Впрочем, можно и не суммировать, заметив, что в любом связном графе, где k или больше вершин, есть связный подграф с ровно k вершинами (возьмём произвольную вершину x и выбросим вершину, находящуюся на наибольшем от x расстоянии, граф останется связным, и это можно повторять). □

Доказательство леммы 3. Нам надо оценить вероятность того, что найдётся индуцированный подграф с некоторым числом k вершин, в котором средняя степень вершины не меньше 5, то есть число рёбер не меньше $l = 2.5k$. В предыдущей лемме у нас была меньшая граница $l = k - 1$, но там мы рассматривали случай $k \geqslant 5\ln n$, а сейчас у нас k может быть любым.[7] Снова оценим то же выражение:

$$e^{k-2l} \cdot n^{k-l} \cdot k^{2l-k} l^{-l} = e^{-4k} \cdot n^{-1.5k} \cdot k^{4k} \cdot (2.5k)^{-2.5k} = e^{-4k} \cdot (2.5)^{-2.5k} \left(\frac{k}{n}\right)^{1.5k}.$$

Теперь это надо просуммировать по всем k от 1 до n. Последняя скобка не больше 1, поэтому показатель степени $1.5k$ можно заменить на 1. При $k = 1$ выражение не больше $1/n$, а с ростом k на 1 выражение убывает не менее чем в 2 раза (последняя скобка возрастает не более чем в 2 раза, а множители перед ней убывают не менее чем в 4 раза). Поэтому вся сумма не больше $2/n$ (сравнение с геометрической прогрессией со знаменателем $1/2$), и этого достаточно для доказательства леммы.

□

[7] Можно считать, что $k \leqslant 5\ln n$, так как все остальные случаи покрываются оценкой из предыдущей леммы, но нам это не нужно.

11. Задачи: отрицательные ассоциации

1. Говорят, что два события A и B в одном вероятностном пространстве положительно [отрицательно] коррелированы, если

$$\Pr(A \wedge B) \geqslant \Pr(A) \cdot \Pr(B) \quad [\text{соответственно } \Pr(A \wedge B) \leqslant \Pr(A) \cdot \Pr(B)].$$

   Как сформулировать это в терминах вероятностей? Что происходит с этими свойствами, если одно из событий (или оба) заменить на дополнения?

2. Будем говорить, что две числовые случайные величины (на одном конечном пространстве) ξ и η *отрицательно ассоциированы*, если для любых порогов u и v события $\xi \geqslant u$ и $\eta \geqslant v$ отрицательно коррелированы. Покажите, что это равносильно такому условию: для любых двух неубывающих функций f и g выполняется неравенство

$$\mathbb{E}(fg) \leqslant \mathbb{E}\, f \cdot \mathbb{E}\, g.$$

   [Для начала можно рассмотреть случай функций с двумя значениями (скажем, 0 и 1), а затем воспользоваться линейностью математического ожидания.]

3. Обобщим это определение на случай нескольких числовых случайных величин: $\xi_1, \ldots, \xi_k$ *отрицательно ассоциированы*, если для любых двух числовых функций f, g, неубывающих по каждому аргументу, выполняется неравенство

$$\mathbb{E}(\, f(\xi_1, \ldots, \xi_k) g(\xi_{k+1}, \ldots, \xi_n)\,) \leqslant \mathbb{E}\, f(\xi_1, \ldots, \xi_k) \cdot \mathbb{E}\, g(\xi_{k+1}, \ldots, \xi_n)$$

   и аналогичные неравенства для произвольного разбиения величин на две непересекающиеся группы. Покажите, что это эквивалентно другому свойству: для любых монотонных вверх множеств $U \subset \mathbb{R}^k$ и $V \subset \mathbb{R}^{n-k}$ события $(\xi_1, \ldots, \xi_k) \in U$ и $(\xi_{k+1}, \ldots, \xi_n) \in V$ отрицательно коррелированы. (В обоих случаях — и для функций, и для событий — мы рассматриваем произвольное разбиение на две группы, не только на k первых и $n-k$ последних.)

4. Покажите, что если в группе отрицательно ассоциированных величин заменить несколько величин на неубывающую функцию от них (скажем, сумму), то полученная меньшая группа будет по-прежнему отрицательно ассоциированной.

5. Независимые величины (очевидно) отрицательно ассоциированы. Покажите, что верен и более общий факт: если есть несколько групп величин (на одном конечном пространстве), и каждая группа отрицательно ассоциирована, а сами группы (как векторные величины) независимы, то все величины отрицательно ассоциированы.

6. Выбираем случайное число от 1 до n и рассматриваем индикаторы событий «выбранное число», то есть n случайных величин, из которых ровно одна равна единице, а остальные нулю.[8] Покажите, что эти n случайных величин отрицательно ассоциированы.

7. Мы кладём по очереди m шаров в n корзин, причём каждый шар опускается независимо от предыдущих. После этого мы считаем шары в корзинах и получаем n случайных величин (их сумма равна общему числу шаров m). Покажите, что эти случайные величины отрицательно ассоциированы.

[8] «Ежели где убудет несколько материи, то умножится в другом месте» (Ломоносов).

8. (Продолжение) Пусть в этой задаче выбран некоторый порог, и корзина называется переполненной, если в ней шаров больше порога. Пусть p — вероятность корзины быть переполненной. Покажите, что вероятность того, что есть хотя бы одна переполненная корзина, не меньше $1 - (1 - p)^n$ (этот ответ соответствует случаю независимых корзин).

9. (Продолжение) Пусть выбрано s непересекающихся групп по k корзин в каждой. Пусть p — вероятность того, что данная группа корзин оказалась пустой (она одна и та же для всех групп из k корзин). Покажите, что вероятность того, что хотя бы одна из групп пуста, не превосходит $1 - (1 - p)^s$ (что соответствует случаю независимых групп).

10. Используйте эту технику для анализа зазоров: покажите, что при больших n вероятность того, что при размещении n шаров в n корзинах будет хотя бы один зазор размера $\ln n/2$ (столько подряд идущих пустых корзин), стремится к 1.

11. Восполните пробел в рассуждении из лекций и покажите, что для случая n корзин и n шаров вероятность того, что во всех корзинах будет менее $c \log n / \log \log n$ шаров, мала при малых c (скажем, что найдётся такое c, что эта вероятность меньше $1/n^2$ при всех n).