Алгоритмы и структуры данных

Конспекты лекций

Лектор: Г.О. Евстропов

Под редакцией Валерии Маликовой и Глеба Новикова

НИУ ВШЭ, 2016-2017

Какое-то введение

В этой книжке отражено с относительной точностью содержание лекций Глеба Олеговича Евстропова, которые читаются на пилотном потоке в Высшей Школе Экономики на образовательной программе «Прикладная математика и информатика». Этот конспект лекций нельзя назвать «учебником» (надеюсь, что пока нельзя). Его автор первый раз сталкивается почти со всем, что пишет, поэтому, безусловно, текст требует правки.

Немного теории вероятностей

Disclaimer 1. Это первая из двух глав теории вероятностей. В них мы будем говорить исключительно о дискретной (с конечными вероятностными пространствами) вероятности, так как в алгоритмах ничего другого особо не понадобится. Кроме того, эти две главы не заслуживают даже названия «Начала теории вероятностей», так как являются совсем частным случаем общей теории вероятностей. Несмотря на это, изложенный в них кусочек начала теорвера достаточен для того, чтобы понимать, что происходит.

Disclaimer 2. Иногда (примерно всегда) мы будем ставить знак «=» между числами тогда, когда его ставить в строгом смысле не очень честно. Однако, поскольку нас всюду интересует ассимптотическое поведение величин, то этот знак будет стоять вполне себе честно.

Определение 2.1. *Вероятностное пространство* $\left(\Omega, 2^\Omega, \mathsf{P}\right)$ – структура, состоящая из:

- Ω – множество элементарных исходов (конечное);
- 2^Ω – множество всевозможных событий (наборов исходов);
- P – функция вероятности $\mathsf{P} : 2^\Omega \to [0, 1]$ такая, что $\mathsf{P}(\Omega) = 1$.

Теперь договоримся о некоторых обозначениях и лексике. Вместо $\mathsf{P}(\{\omega\})$ мы всегда будем писать $\mathsf{P}(\omega)$ и называть это *вероятностью исхода* ω. Если вероятность какого-то события $B \in 2^\Omega$ равна нулю, то есть $\mathsf{P}(B) = 0$, то событие B называется *невозможным*. Если же $\mathsf{P}(B) = 1$, то B называется *достоверным* событием.

Через $\mathsf{P}(A|B)$ будем обозначать вероятность события при условии наступления события B. Такая вероятность считается по формуле

$$\mathsf{P}(A|B) = \frac{\mathsf{P}(A \cap B)}{\mathsf{P}(B)}$$

Определение 2.2. *Независимыми* называются события A и B, если $\mathsf{P}(A|B) = \mathsf{P}(A)$ или $\mathsf{P}(A) \cdot \mathsf{P}(B) = \mathsf{P}(A \cap B)$.

Несовместными называются события, для которых $\mathsf{P}(A|B) = 0$.

Определение 2.3. $A_1, \ldots, A_n$ - множство попарно несовместных событий, то есть $\forall i, j \ \mathsf{P}(A_i|A_j) = 0$. Если $\mathsf{P}(\bigcup\limits_{i=1}^{n} A_i) = 1$, тогда набор $\{A_i\}_{i=1}^n$ называется *полной группой событий*.

Тогда если есть событие B и полная группа событий $\{A_i\}_{i=1}^n$, тогда $B = \bigcup\limits_{i=1}^{n} B \cap A_i$ и

$$\mathsf{P}(B) = \sum_{i=1}^{n} \mathsf{P}(B \cap A_i) = \sum_{i=1}^{n} \mathsf{P}(B|A_i) \cdot \mathsf{P}(A_i)$$

Больше теории вероятностей

Рассмотрим функцию ξ, определенную на вероятностном пространстве Ω:

$$\xi : \Omega \to \mathbb{R}$$

Математическим ожиданием ξ называется такая величина:

$$\mathsf{E}\xi = \sum_{\omega \in \Omega} \xi(\omega)\,\mathsf{P}(\omega)$$

Пример 3.1. Пусть вы - опытный продавец мороженного. Вам нужно заранее забронировать место в парке развлечений, поэтому вы хотите узнать, в какие дни стоит это делать. Вам известен прогноз погоды и то, сколько вы зарабатываете за сутки в определенную погоду.

Вероятностное пространство событий: $\Omega = \{\text{жара, облачно, дождь, ураган}\}$.

$\xi(\text{жара}) = 100\$,\ \xi(\text{облачно}) = 50\$,\ \xi(\text{дождь}) = 10\$,\ \xi(\text{ураган}) = 1\$$.

Аренда места обходится в 50\$ за сутки.

Пусть будет жарко с вероятностью 0.9 и облачно с вероятностью 0.1. Тогда:

$$\mathsf{E}\xi = 0{,}9 \cdot 100\$ + 0{,}1 \cdot 50\$ - 50\$ = 45\$$$

Если все исходы равновероятны, то:

$$\mathsf{E}\xi = 100 \cdot 0.25 + 50 \cdot 0.25 + 10 \cdot 0.25 + 1 \cdot 0.25 = 50\$$$

Введем определение индикаторной случайно величины.

Определение 3.2. Индикаторная случайная величина I_A события A - величина, принимающая значение 1, если событие A произошло, и 0 в обратном случае.

$$I_A = \begin{cases} 1, & \mathsf{P}(A) \\ 0, & 1 - \mathsf{P}(A) \end{cases}$$

Тогда $\mathsf{E}I_A = \mathsf{P}(A)$.

Заметим, что для любого $r \subset R$ верно, что $\mathsf{P}(\xi \in r) = \sum_{\xi(\omega) \in r} \mathsf{P}(\omega)$.

Мы уже знакомы с понятием «независимые события». Теперь мы хотим понять, какие случайные величины являются независимыми.

Определение 3.3. Случайные величины ξ и ψ независимы, если для любых $r_1 \in N(\xi)$ и $r_2 \in N(\psi)$, где $N(\ldots)$ - область значений величины, верно:

$$\begin{cases} A = \xi \in r_1 \\ B = \psi \in r_2 \end{cases} \Rightarrow \mathsf{P}(A)\,\mathsf{P}(B) = \mathsf{P}(A \cap B)$$

Определение 3.4. Случайные величины ξ и ψ независимы, если для $\forall x,y \in \mathbb{R}$, верно:

$$\begin{cases} A = (\xi \equiv x) \\ B = (\psi \equiv y) \end{cases} \Rightarrow \mathsf{P}(A)\,\mathsf{P}(B) = \mathsf{P}(A \cap B)$$

Докажем, что эти определения эквивалентны.

###
Я вообще не верю в то, что логика доказательства верна. Когда доказывается эквивалентность двух утверждений $A \Rightarrow B$ и $C \Rightarrow D$, то доказывается $A \Leftrightarrow C$.
###

Доказательство. Пусть $A_1 \dots A_n$, где n - число элементов в r_1. $A_i = (\xi = x_i), x_i \in r_1$.

$$\mathsf{P}(A) = \mathsf{P}(A_1) + \ldots + \mathsf{P}(A_n) \qquad\qquad \mathsf{P}(B) = \mathsf{P}(B_1) + ... + \mathsf{P}(B_m)$$

$$\begin{aligned}&(\mathsf{P}(A_1) + \mathsf{P}(A_2) + \ldots + \mathsf{P}(A_n))(\mathsf{P}(B_1) + \mathsf{P}(B_2) + \ldots + \mathsf{P}(B_m)) = \\ =&\sum_{i=1}^{n}\sum_{j=1}^{m}\mathsf{P}(A_i)\,\mathsf{P}(B_j) = \sum_{i=1}^{n}\sum_{j=1}^{m}\mathsf{P}(A_i \cap B_j) = \mathsf{P}(A \cap B)\end{aligned}$$

□

Теперь рассмотрим самое главное свойство случайной величины. Мы будем использовать это свойство на протяжении всего курса алгоритмов.

Теорема 3.5.

$$\mathsf{E}(\alpha\xi + \beta\psi) = \alpha\mathsf{E}\xi + \beta\mathsf{E}\psi$$

Доказательство.

$$\begin{aligned}\mathsf{E}(\alpha\xi + \beta\psi) =& \sum_{\omega\in\Omega}(\alpha\xi(\omega) + \beta\psi(\omega))\,\mathsf{P}(\omega) = \\ =&\alpha\sum\xi(\omega)\,\mathsf{P}(\omega) + \beta\sum\psi(\omega)\,\mathsf{P}(\omega) = \\ =&\alpha\mathsf{E}\xi + \beta\mathsf{E}\psi\end{aligned}$$

□

Пример 3.6. Найти математическое ожидание количества статичных точек($p_i = i$) в перестановке из n элементов.

$$\mathsf{E}\xi = \mathsf{E}(I_1 + ...I_n) = \sum_{i=1}^{n}\mathsf{E}I_i = \sum_{i=1}^{n}\mathsf{P}(I_i) = \sum_{i=1}^{n}\frac{1}{n} = 1$$

Мы теперь знаем, что $\mathsf{E}(\xi + \psi) = \mathsf{E}\xi + \mathsf{E}\psi$. Верно ли, что $\mathsf{E}(\xi\psi) = \mathsf{E}(\xi)\mathsf{E}(\psi)$?

Теорема 3.7. *Если ξ и ψ - независимые величины, то* $\mathsf{E}(\xi\psi) = \mathsf{E}(\xi)\mathsf{E}(\psi)$.

Доказательство.

$$\begin{aligned}\mathsf{E}\xi =& \sum_{\omega\in\Omega}\xi(\omega)\,\mathsf{P}(\omega) = \sum_{x\in N(\xi)} x\,\mathsf{P}(\xi = x) \\ \mathsf{E}(\xi\psi) =& \sum z(\xi\psi = z) = \sum_{x\in N(\xi)}\sum_{y\in N(\psi)}\mathsf{P}((\xi = x) \cap (\psi = y)) = \\ =&\sum_{x}\sum_{y} xy\,\mathsf{P}(\xi = x)\,\mathsf{P}(\psi = y) = \sum_{x} x\,\mathsf{P}(\xi = x)\sum_{y} y\,\mathsf{P}(\psi = y) = \mathsf{E}(\xi)\mathsf{E}(\psi)\end{aligned}$$

□

Определение 3.8. Дисперсией величины ξ называют такую величину $\mathsf{D}\,\xi$, что

$$\mathsf{D}\,\xi = \mathsf{E}(\xi - \mathsf{E}\xi)^2 = \mathsf{E}\xi^2 - (\mathsf{E}\xi)^2$$

Теорема 3.9. Неравенство Маркова (без док-ва)

$$\xi > 0,\ \mathsf{P}(\xi \geqslant a) \leqslant \frac{\mathsf{E}\xi}{a}$$

Теорема 3.10. Неравенство Чебышёва (без дока-ва)

$$\mathsf{P}(|\xi - \mathsf{E}\xi| \geqslant a) \leqslant \frac{\mathsf{D}\,\xi}{a^2}$$

Пример 3.11. Найти максимальную возрастающую подпоследовательность в случайной перестановке $a_1...a_n$, то есть найти $j_1,,j_k$ такие, что $a_{j_1} < a_{j_2} < ... < a_{j_k}$.

---

cur $= -1$
for $i : 1 \to n$ **do**
  if $a_i > cur$ **then**
    $cur = a_i$

###
Что такое A_k?
###

Пусть величина ξ – длина максимальной возрастающей последовательности. Тогда

$$\mathsf{E}\xi = \mathsf{E}(I_1 + ... + I_n) = \sum \mathsf{E}I_k = \sum \mathsf{P}(A_k) = \sum_{k=1}^{n} \frac{1}{k} = \log n.$$

Пример 3.12. Дан взвешенный граф $G(V,E), w(e) \geq 0$. Найти максимальный разрез. (разрез в графе – нетривиальное подмножество $A \subset V$ его вершин, величина разреза - сумма весов ребер, один конец которых лежит в A, другой - в $\overline{A}$). Тогда величина разреза это

$$w(A) = \sum_{u \in A v \in \not A} s(uv),$$

где U_A - множество ребер, лежащих на разрезе, индуцированном множеством A.

Решим задачу таким алгоримом: переберем все вершины, будем добавлять каждую из них в A с вероятностью $\frac{1}{2}$. Докажем, что математическое ожидание величины $w(A)$ это хотя бы половина оптимального ответа.

Вероятность того, что ребро лежит на оптимальном разрезе - $\frac{1}{2}$, потому либо оба конца лежат в A или $\overline{A}$, либо один из концов лежит в A, другой - в $\overline{A}$.

$$\mathsf{E}\xi = \sum \mathsf{E}I(e \in U_A)w(e) = \sum w(e)\,\mathsf{P}(e \in U_A) = \sum \frac{1}{2}w(e) = \frac{1}{2}\sum w(e)$$

for $i : 2 \to n$ **do**
  if $a_i > cur$ **then**
    $\text{parent}(i) = \text{rand}(1, i-1)$

Пример 3.13. Приведем пример алгоритма построения случайного дерева. Пусть 1-я вершина - это корень.

Докажем, что $\forall u,v\ \mathsf{E}g(v,u) = O(\log n)$.

$$\mathsf{E}g(v,u) \leqslant \mathsf{E}(d(v) + d(u)) = \mathsf{E}d(v) + \mathsf{E}d(u)$$
$$\mathsf{E}d(i) = \log i (i \geqslant 2, d(1) = 0)$$
$$\mathsf{E}d(i) = 1 + \sum_{j=1..i-1} \mathsf{E}d(j)\frac{1}{i-1}$$
$$\exists c : Ed(i) \leqslant c \log n$$
$$\mathsf{E}d(i) \leqslant 1 + \frac{1}{i-1}\sum c \log i =$$
$$= 1 + \frac{\sum_{j=1}^{\frac{i}{2}} c \log j}{i-1} + \frac{\sum_{j=\frac{i}{2}+1}^{i} c \log j}{i-1}$$
$$\mathsf{E}d(i) \leqslant 1 + \frac{1}{i-1}\sum c \log(i-1) + \frac{1}{i-1}\sum c \log i = 1 + \frac{1}{i-1}\sum c \log i$$

Методы анализа сортировок за квадратное (и не очень) время

Если мы скажем, что проходим на лекции бабл сорт, то нас засмеют, поэтому назовем нашу лекцию так.

Г.О. Евстропов

В процессе анализа алгоритма мы будем доказывать *корректность* алгоритма и анализировать *время его работы*.

Существуют хотя бы три метода доказательства корректности алгоритма (нам их должно хватить):

1. по индукции;
2. от противного;
3. поиск инварианта.

Определение 4.1. «Задача о сортировке». Есть n произвольных элементов $a_1, \ldots, a_n$. Хочется их упорядочить так, чтобы $a_{\pi(1)} \leq a_{\pi(2)} \leq \ldots \leq a_{\pi(n)}$.

Лирическое отступление: RAM-модель

В нашей RAM-подели будет:

- $\square$ – процессор. За единицу времени он умеет выполнять одно действие с целочисленными данными, а также запрос к одной ячейке пямяти.
- $\square\square\square\square\square\square\square\square\square\square$ – память. Её всегда ассимпотически достаточно для любых действий (но не слишком много). Например, при $P(n)$ действий памяти будет примерно $c \cdot P(n)$. Кроме того, любая работа с куском памяти выполняется не быстрее, чем за длину этого куса.

Что мы умеем хранить и делать в RAM-модели:

- целые числа;
- вещественные числа;
- $\pm,\ *, /, \%$ за $O(1)$;
- все битовые операции выполняются за $O(1)$;
- всякие $\sqrt{x}, \sin x, \cos x, \ldots$ мы тоже умеем вычислять за $O(1)$ (если противное не оговорено заранее).

Сортировка пузырьком (bubble sort)

Algorithm 1 Bubble sort

```
while NOT_SORTED do
    for i : 1 → n − 1 do
        if a_i > a_{i+1} then SWAP(a_i, a_{i+1})
```

Теперь нужно доказать корректность алгоритма. *Доказательство корректности.* На каждом шаге количество инверсий уменьшается хотя бы на 1, а значит за конечное число итераций алгоритм закончится.

Оценим количество операций. Оно, очевидно, не превосходит количество инверсий, а их не более, чем $\frac{n(n+1)}{2}$.

Сортировка выбором

Algorithm 2 Selection sort

```
for i : n → 1 do
    p = SELECT_MAX((1, i))
    SWAP(a_p, a_i)
```

Algorithm 3 Selection sort (recursive)

```
function REC(a, n)
    if n = 1 then return
    p = SELECT_MAX(1, i)
    SWAP(a_p, a_i)
    REC(a, n − 1)
```

Доказательство корректности. Докажем корректность индукцией по n.

1. База. $n = 1$. Один элемент уже отсортирован.
2. Допустим, для какого-то n алогоритм работает корректно, то есть сортировка работает для всех массивов длинны n. Рассмотрим массив из $n + 1$ элемента. Тогда на первой итерации алгоритма на место a_{n+1} встанет максимальный элемент (то есть он встанет на своё место). Тогда после первой итерации в остальных n элементах будут элементы меньше, чем a_{n+1}, а для них алогоритм работает корректно.

Теперь о времени работы. Докажем, что $t(n) \leq c_2 \cdot n^2$.

Доказательство корректности. Действительно, выразим величину $t(n)$ рекурсивно через прыдудещий шаг. На n-том шаге алгоритм ищет максимальный элемент (за $c_1 \cdot n$ времени). Тогда

$$t(n) \leq c_1 n + t(n-1).$$

Воспользуемся индукцией по n. База очевидна, та что напишем переход:

$$t(n) \leq c_1 n + c_2 (n-1)^2 = c_1 n + c_2 n^2 - 2nc_2 + c_2.$$

Возьмем $c_2 = c_1$ (нам же нужно выбрать c_2 такой, что...).
###
Тут я не уверен.
###
Тогда

$$t(n) \leq c_1 n + c_1 n^2 - 2nc_1 + c_1 = c_1 n^2 + c_1 - c_1 n \leq c_1 n^2 = c_2 n^2.$$

Сортировка вставками

Инвариант. На i-том шаге алгоритма префикс до $i-1$-го элемента отсортирован.

Algorithm 4 Insertion sort

for $i : 1 \to n$ **do** INSERT(a, i, a_i)

Доказательство корректности. Очевидно, что на $n+1$ шаге весь префикс будет отсортирован, а функция *insert* вставит элемент a_{n+1} на своё порядковое место внутри этого префикса.

Время работы. Всего выполняется n внешних итераций. Функция INSERT в худшем случае (при вставке элемента в самое начало) работает за i итераци й, где i – длина префикса. Таким образом получается $\frac{n(n+1)}{2}$ итераций, а значит $t(n) = O(n^2)$.

Свойства сортировок

1. *Stable sort* – сортировка, сохраняющая порядок равных элементов;
2. *Inplace sort* – сортировка (применимо и к другим алгоритмам), использующая $O(\log n)$ (или в редких случаях какую-то другую $o(n)$) памяти.

Сортировка слиянием

Algorithm 5 Merge sort

function MERGE_SORT$(a, begin, end)$
  $middle = \frac{begin+end}{2}$
  MERGE_SORT$(a, begin, middle)$
  MERGE_SORT$(a, middle, end)$
  MERGE$(a, begin, middle, middle, end)$
  ▷ MERGE creates sorted array from elements $a_{begin}, \ldots, a_{middle}, \ldots, a_{end}$

Доказательство корректности. База очевидна. В начале последнего шага предполагается, что промежутки $begin - middle$ и $middle - end$ отсортированы, а на самом шаге функция MERGE сливает эти промежутки в один отсортированный массив.

Время работы. На каждом шаге размер массива делится пополам, то есть глубина дерева рекурсии будет равна $\log_2 n$. Кроме того, на каждом шаге выполняется n опе раций с памятью (запись в новый массив). Таким образом $t(n) = O(n \log n)$.

Быстрая сортировка

Выберем с лучайным образом элемент в массиве. Пусть это будет a_k. Тогда переместим все элементы больше a_k в правую часть массива, а меньше или равные a_k в левую часть и отсортируем каждую часть.

Algorithm 6 Quick sort

```
function QUICK_SORT(a, begin, end)
    k = RAND(begin, end)
    m = a_k
    l = begin
    r = end
    while l ≤ r do
        while a_l < m do
            l = l + 1
        while a_r > m do
            r = r − 1
        SWAP(a_l, a_r)
        l = l + 1
        r = r − 1
    QUICK_SORT(a, begin, l)
    QUICK_SORT(a, l + 1, end)
    MERGE(a, begin, l, l + 1, end)        ▷ does the same as MERGE from MERGE_SORT
```

Время работы. Доказываем, что $t(n) \leq c_2 n \log n$.
###
Надо исправить
###

$$
\begin{aligned}
t(n) =& c_1 n + \frac{1}{n}\sum_{i=1}^{n}(t(i-1) + t(n-i)) = \\
=& c_1 n + \frac{1}{n}\sum_{i=1}^{n-1}(c_2(i-1)\log(i-1) + c_2(n-i)\log(n-i)) = \\
=& c_1 n + \frac{2c_2}{n}\sum_{i=1}^{n-1} i\log i \leq c_1 n + 2c_2\sum_{i=1}^{n-1}\log i = \\
=& c_1 n + 2c_2(n-1)\log(n-1)
\end{aligned}
$$