Формула оценки: **0.25 дз + 0.3 контест + 0.15 кр + 0.3 экзамен + Бонус**

Домашние задания: сдавать устно(раз в неделю ассисты устраивают доп пару, запись онлайн) или latex, дедлайн 10-21 день.

Контесты: Длинные(код ревью), короткие(раз в 2 недели), неточные, бонусные(идет к бонусу). Штрафов нет.

Контрольные работы: раз в модуль, тестовые вопросы.

Бонусы: бонусные контесты, ACM, работа на семинаре.

Материалы:

- Кормен
- en.wikipedia
- викиконспекты
- e-maxx
- Корте-Фанен Коммбинаторная оптимизация

**Теория вероятности**. $(\Omega, 2^{\Omega}, P)$ - вероятностная пространство.
$A \subset \Omega$, $P(A) = \sum_{w \in A} P(w)$.
Def: $A, B$ - события, $P(B) > 0$. **P(A|B)** - вероятность события A, если наступило событие B. Тогда $P(A|B) = \dfrac{\sum_{w \in A \cap B} P(w)}{\sum_{w \in B} P(w)} = \dfrac{P(A \cap B)}{P(B)}$.
Def: A и B независимые, если P(A|B) = P(A).
Тогда, если A и B независимые, то $P(A \cap B) = P(A) \cdot P(B)$.

$\xi : \Omega \to \mathbb{R}$ - случайная величина. $\xi(w)$ - значение, $w \in \Omega$.
Пример: Есть 5 марок автомобиля, их стоимости и их количества. A - 1000 - 100; B - 2000 - 5; C - 3000 - 5; D - 2000 - 20; E - 1500 - 30; Тогда нас интересуют $P(\xi = 1000) = \dfrac{100}{160}, P(\xi = 1500) = \dfrac{30}{160}, P(\xi = 2000) = \dfrac{25}{160}, P(\xi = 3000) = \dfrac{5}{160}$.

Матожидание $E(\xi) = \sum_{w \in \Omega} \xi(w) \cdot P(w) = \sum_x x \cdot P(\xi = x)$.

Индикаторная случайная величина: $I_A = \begin{cases} 1, w \in A \\ 0, w \notin A \end{cases}$ Тогда $E(I_A) = P(A)$.

Пусть есть 2 случайной величины $\xi_1$ и $\xi_2$. Тогда $E(\alpha\xi_1 + \beta\xi_2) = \alpha E(\xi_1) + \beta E(\xi_2)$.
$E(\alpha\xi_1 + \beta\xi_2) = \sum_{w \in \Omega}(\alpha\xi_1(w) \cdot P(w) + \beta\xi_2(w) \cdot P(w)) = \alpha \sum_{w \in \Omega} \xi_1(w) \cdot P(w) + \beta \sum_{w \in \Omega} \xi_2(w) \cdot P(w) = \alpha E(\xi_1) + \beta E(\xi_2)$

Две случайные величины называются независимые, если $\forall x, y : P(\xi_1 = x \text{ и } \xi_2 = y) = P(\xi_1 = x) \cdot P(\xi_2 = y)$.
$n$ случайных величин называются попарно независимыми, если любые 2 величины независимы.
*независимы в совокупности - см семинар*

$\xi_1$ и $\xi_2$ - случайные независимые величины. Тогда $E(\xi_1\xi_2) = E(\xi_1)E(\xi_2)$.
$E(\xi_1\xi_2) = \sum_{w \in \Omega} \xi_1(w)\xi_2(w)P(w) = \sum_x x \cdot P(\xi_1\xi_2 = x) = \sum_{(u,v)} uv \cdot P(\xi_1 = u \text{ и } \xi_2 = v) = [\xi_1 \text{ и } \xi_2 \text{ независимы}] = \sum_{(u,v)} uv \cdot P(\xi_1 = u) \cdot P(\xi_2 = v) = (\sum_u u \cdot P(\xi_1 = u)) \cdot (\sum_v v \cdot P(\xi_2 = v)) = E(\xi_1)E(\xi_2)$.

Задача о назначениях. Есть $n$ работников и $n$ работ. Есть таблица, где $a_{ij}$ - сколько i-ый работник берет за j-ую работу. Нужно распределить работников по работам так, чтобы суммарная плата за все работы была миниальна. Оценим матожидание затрат при случайном решении. $A_{ij}$ - событие, когда i-ый работник делает j-ую работу. $\xi = \sum_{(i,j)} I_{A_{ij}} \cdot a_{ij}$. Тогда $E(\xi) = \sum_{(i,j)} E(I_{A_{ij}} = \sum_{(i,j)} a_{ij} P(Aij) = \sum_{(i,j)} a_{ij} \cdot \frac{1}{n}$.

Найти максимальный разрез в неориентированном невзвешанном графе.

Будем строить случайный разрез(каждую вершину либо в $A$, либо в $\overline{A}$). Тогда $\xi$ - величина нашего разреза. $\xi = \sum_{e \in E(G)} I_{B_e}$, где $B_e$ - событие, когда $e$ лежит в разрезе. $P(e \in \text{разрез}) = \frac{1}{2}$. Тогда $E(\xi) = E(\sum_{e \in E(G)} I_{B_E}) = \sum E(I) = \dfrac{1}{2}|E(G)|$.

Есть перестановка $p_1 \dots p_n$. Алгоритм жадно набирает возрастающую подпоследовательность. Какое матожидание длины этой подпоследовательности?
Событие $A_i$ - алгоритм возьмет $p_i$. $E(\xi) = E(\sum I_{A_i}) = \sum P(A_i)$. $P(A_i) = P(\forall j < i : p_j < p_i) = \frac{1}{i}$. Тогда $E(\xi) = \sum_{i=1}^{n} \frac{1}{i} = \Theta(logn)$.

Дисперсия $D(\xi) = \sum_{w \in \Omega} P(w)(\xi(w) - E(\xi))^2$.
Свойства:

- $D(\xi_1 + \xi_2) = D(\xi_1) + D(\xi_2)$, $\xi_1$ и $\xi_2$ независимы

- $D(\lambda\xi_1) = \lambda^2 D(\xi_1)$

Неравенство Маркова. $\xi : \Omega \to \mathbb{R}_+$. $P(\xi(w) \geq E(\xi) \cdot k) \leq \dfrac{1}{k}$.

Неравенство Чебышева. $\xi : \Omega \to \mathbb{R}$. $P(|\xi - E(\xi)| \leq \alpha) \leq \dfrac{D(\xi)}{\alpha^2}$.

**Модели**

RAM-модель (Random Access Machine) Вопросы, возникающие при создании модели

1. адресация
2. какие инструкции
3. рекурсия
4. где лежат инструкции
5. размер данных
6. кол-во памяти
7. случайность

**Адресация** Есть ячейки, в которых можно хранить целые числа (ограничения на $MAXC$ разумные, и на них введена неявная адресация

*Замечание.* Явная адресация — при создании элемента получаем адрес и можем пользоваться только этим адресом. Неявно — можем получать адреса каким-то своим образом, к примеру, $ptr + 20$.

**Кол-во памяти** Неявное соглашение RAM — время работы не меньше памяти. По дефолту считаем, что мы его инициализируем мусором

**Где инструкции** Хранить инструкции можно в памяти и где-то снаружи. Мы будем хранить снаружи (внутри — RASP-модель). Иначе говоря, инструкции и данные отделены.

**Какие инструкции** В нашей модели есть инструкции следующих типов:

- работа с памятью
- ветвление
- передача управления ($=goto$),
- арифметика (at least $a + b, a - b, \frac{a}{b}, \cdot, mod, \lfloor\frac{a}{b}\rfloor$)
- сревнения (at least $a < b, a > b, a \leq b, a \geq b, a = b, a \neq b$)
- логические (at least $\wedge, \vee, \oplus, \neg$)
- битовые операции ($>>, <<, \&, |, \sim, \oplus$)
- математические функции (опять-таки, в рамках разумного)
- rand

Все инструкции работают от конечного разумного числа операндов (не умеем в векторные операции)

**Размер данных** $\exists\, C, k \,:\, C \cdot A^k \cdot n^k$ — верхнее ограничение на величины промежуточных вычислений.

**Рекурсия** Рекурсия всегда линейна по памяити относительно глубины.

**Случайность** Мы считаем, что у нас есть абсолютно рандомная функция. Будем полагать, что у нас есть источник энтропии, выдающий случайности в промежутке $[0, 1]$.

**Время работы.**

- наихудшее — $t = \max\limits_{input, random} t(input, random)$
- наилучшее — $t = \max\limits_{input} \min\limits_{random} t(input, random)$
- ожидаемое — $E\ t = \max\limits_{input} Average_{random} t(input, random)$
- на случайных данных — $t = Average_{input} Average_{random} t(input, random)$

**Алгоритмы**

Методы доказательства корректности алгоритма.

1. индукция
2. инвариант
3. от противного

Способы оценки времени работы:

- Прямой учет
- Рекурсивная оценка
- Амортизационный анализ

**Прямой учет** Время работы строчки — произведение верхних оценок по всем строчкам-предкам нашей.

К примеру

```
while (!is_sorted()) { // O(# inversions) = O(n^2)
    for (int i = 0; i + 1 < n; i++) { // O(n) * O(parent) = O(n^3)
        if (a[i] > a[i + 1]) {
            swap(a[i], a[i + 1]); // O(n^3)
        }
    }
}
```

**Рекурсивная оценка** Пример — сортировка слиянием

Нас интересует две вещи: инвариант и переход. Для оценки времени используем рекурренту вида $T(n) = O(f(n)) + \sum\limits_{n' \in calls} T(n')$

При этом если мы доказываем время работы, то показываем $T(n) \le c \cdot f(n)$, зная, что для $n'$ $\exists c$ : $T(n') \le c \cdot f(n)$

Важно, что $c$ глобальное и не должно увеличиваться в ходе доказательства

**stable sort** Делает сортировку, не меняя порядок равных элементов относительно исходной последовательности. Merge-sort стабилен.

**inplace-algorithm** Не требует дополнительной памяти и делает все прямо на данной памяти (у нас есть $\log n$ памяти на рекурсию). Quick-sort inplace.

**Время работы qsort**

$$T(n) = \max_{input} average_{rand} t(input, rand) = \max_{|input|=n} Et(input)$$

$$T(n) \le \Theta(n) + \frac{1}{n}\sum_{k=0}^{n-1}(T(k+1) + T(n-k)) \le \Theta(n) + \frac{2}{n}\cdot\sum_{k=1}^{n} T(k-1) \le a\cdot n + \frac{2}{n}\sum_{k=1}^{n} c\cdot(k-1)\cdot\log(k-1) \le$$

$$\le a\cdot n + \frac{2}{n}\sum_{k=1}^{\frac{n}{2}} c\cdot(k-1)\cdot\log n - \frac{2}{n}\sum_{k=1}^{\frac{n}{2}} c\cdot(k-1) + \frac{2}{n}\sum_{k=\frac{n}{2}+1}^{n} c\cdot(k-1)\cdot\log n \le$$

$$\le a\cdot n + \frac{2}{n}\cdot n^2\cdot\log n - \frac{2c}{n}\cdot\frac{(n-2)^2}{4} \le a\cdot n + cn\log n - \frac{c(n-2)}{4} \le cn\log n$$

$$\frac{c(n-2)}{4} \geq a \cdot n$$

$$c \cdot n - 2c \geq 4 \cdot a \cdot n$$

$$c \geq \frac{4 \cdot a \cdot n}{(n-2)}$$

, что верно для достаточно больших $n$.

**Ограничение на число сравнений в сортировке** Бинарные сравнения на меньше.

Рассмотрим дерево переходов. Для перестановки есть хотя бы один лист — листьев котя бы $n!$

$$L(T) \leq 2^x, d(T) \leq x$$

, если $x$ — ответ

$$L(T) \geq n!$$

$$d(T) \geq \log n! \geq \log \frac{n}{2}^{\frac{n}{2}} = \log 2^{(\log \frac{n}{2}) \cdot \frac{n}{2}} = \frac{n}{2} \cdot \log \frac{n}{2} = \Omega(n \log n)$$

# 1 Сортировки основанные на внутреннем виде данных

Имеем $n$ чисел $[0, U-1],\ U = 2^w$, числа укладываются в RAM-модель

**Сортировка подсчетом** Заводим массив $cnt[U],\ cnt[x] = |\{i : a_i = x\}|$. Дальше переводим $count \to pref,\ pref[x] = pref[x-1] + count[x]$

$O(n+U)$

**Поразрядная сортировка** $b_{ij}$—$j$-й бит $i$-го числа. (Сортируем бинарные строки длины $w$)

Поочередно сортируем строки, разбивая их на классы эквивалентности по $2^{iter}$ последним символам. После чего мы стабильно сортируем по $(iter+1)$-му символу.

$O(\log_n Un)$

**Bucket sort** Разбиваем множество на корзины, каждой корзине соответствует отрезок. В каждой корзине запускаемся рекурсивно.

$O(n \log U)$

В продакшне используют первую пару итераций, чтобы сильно снизить размерность на реальных данных.

Пусть мы хотим отсортить равновероятные числа из $[0, 1]$. В каждом бакете отсортируем за квадрат. Получим $O(n)$.

$$t(n) \leq \sum_{i=1}^{n} c \cdot (1 + E(cnt_i)^2) \leq c \cdot n + c \cdot \sum_{i=1}^{n} E(cnt_i^2) =$$

$$= c \cdot n + c \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} EI_{A_{ij}} = c \cdot n + c \cdot n^2 \cdot \frac{1}{n} \leq 2 \cdot c \cdot n$$

# 2 Иерархия памяти

Нас интересует задержка (latency), пропускная способность (throughput). Подгрузка $x$ данных занимает $l + \frac{x}{t}$

От долгой к быстрой

1. external machine / internet
2. HDD
3. SSD
4. RAM
5. L3
6. L2
7. L1
8. registers

# 3 Алгоритмы во внешней памяти

$n$ — размер задачи
$M$ — размер $RAM$
$B$ — блок данных
$B \ll M \ll n$
$\log n \ll B$
$B < \sqrt{M}$ (но это неявно и не факт)

**Mergesort во внешней памяти** Обычный mergesort, но три типа событий в $merge$:

1. Кончился первый буфер — подгружаем новый
2. Кончился второй — аналогично
3. Кончился буфер для слияния — выписываем обратно в RAM и сбрасываем

$O(\frac{n}{B}\log n) \to O(\frac{n}{B}\log \frac{n}{B}) \to O(\frac{n}{B}\log_{\frac{M}{B}} \frac{n}{B})$
$+2$ идеи:

1. Дошли до размера $M$ — явно посортим в RAM
2. Можем сливать сразу $\frac{M}{B}$ массивов