

SPb HSE, 1 курс, осень 2021/22

Практика по алгоритмам #5

Бинпоиск и сортировка

1 октября

Собрано 2 декабря 2021 г. в 07:51

---

## Содержание

<b>1. Бинпоиск и сортировка</b>	<b>1</b>
<b>2. Разбор задач практики</b>	<b>3</b>
<b>3. Домашнее задание</b>	<b>6</b>
3.1. Обязательная часть . . . . .	6
3.2. Дополнительная часть . . . . .	7

# Бинпоиск и сортировка

## 1. Нижняя оценка на почти рабочую сортировку

Покажите, что любая сортировка, которая верно работает хотя бы на доле  $\frac{1}{100}$  от всех перестановок, работает за  $\Omega(n \log n)$ .

## 2. Почти отсортированный массив

В массиве длины  $n$  каждый элемент отстоит от своей правильной позиции на  $\leq k$ .

- Отсортируйте за  $\mathcal{O}(nk)$ .
- Отсортируйте за  $\mathcal{O}(n + I)$ , где  $I$  – число инверсий.
- Отсортируйте за  $\mathcal{O}(n \log k)$ .
- Докажите нижнюю оценку на время сортировки  $\Omega(n \log k)$ .

## 3. Anti-QuickSort test

Пусть в качестве разбивающего элемента всегда берется средний элемент:  $\lfloor (l + r)/2 \rfloor$ . Построить массив длины  $n$ , на котором QuickSort отработает за  $\Omega(n^2)$ .

## 4. Ломаная без самопересечений

Дано  $n$  точек на плоскости. За  $\mathcal{O}(n \log n)$  соединить их

- $(n - 1)$ -звенной ломаной без самопересечений (не замкнутой);
- $n$ -звенной ломаной без самопересечений (замкнутой).

## 5. Сканирование отрезков

Дан набор из  $n$  отрезков  $[a_i, b_i]$ , где  $a_i, b_i$  – вещественные.

- Выбрать максимальное число непересекающихся отрезков.  $\mathcal{O}(n \log n)$ . Жадно. Доказать!
- Найти такое вещественное  $x$ , что  $|\{i: x \in [a_i, b_i]\}|$  максимально.
- Для каждого  $k$  посчитать длину множества точек, покрытых ровно  $k$  отрезками.

## 6. Покраска забора

Есть  $q$  запросов вида  $\text{color}(l, r, c)$ : покраска отрезка  $[l, r]$  массива в цвет  $c$ .

В конце вывести получившийся массив. Решение в offline за  $\mathcal{O}(n + q \log q)$ .

(\*) Решите ещё быстрее ; -)

## 7. Коровы – в стойла!

Есть  $m$  стойл с координатами  $x_1, \dots, x_m$  и  $n$  коров. Расставить коров по стойлам так, чтобы минимальное расстояние между коровами было максимально.

## 8. K-best

Даны два массива из положительных чисел  $a$  и  $b$ .  $|a| = |b| = n \leq 10^5$ .

Выбрать массив  $p$ :  $k$  различных чисел от 1 до  $n$  так, чтобы  $\frac{\sum_{i=1}^k a_{p_i}}{\sum_{i=1}^k b_{p_i}} \rightarrow \max$ .

## 9. Поиск точки разреза

- Дан массив  $a_1 < a_2 < \dots < a_k > a_{k+1} > \dots > a_n$ . За  $\mathcal{O}(\log n)$  найти  $k$ .
- Дан массив  $a_1 \leq a_2 \leq \dots \leq a_k \geq a_{k+1} \geq \dots \geq a_n$ . За  $\mathcal{O}(\log n)$  найти  $k$ .
- (\*) Дан циклический сдвиг на  $k$  строго возрастающего массива. За  $\mathcal{O}(\log n)$  найти  $k$ .

**10. Сложность случайных сортировок**

Есть много разных сортировок. Есть даже такие, что работают дольше, чем квадрат. Оцените время работы в среднем следующих сортировок:

`while not sorted`

a) `i = rand(), j = rand(), if a[min(i,j)] > a[max(i,j)] swap`

b) (\*) `i = rand(), j = rand(), swap`

**11. (\*) Второй максимум**

Найти второй максимум в массиве за  $n + \mathcal{O}(\log n)$  сравнений.

**12. (\*) Binary search lower bound**

Представьте, что мы пишем «бинпоиск на односвязном списке». Делить можно не обязательно пополам, а по любой позиции  $i$ . Время обращение к  $i$ -му элементу всегда  $\Theta(i)$ .

Доказать, что такой «бинпоиск» работает за  $\Omega(n \log n)$  в худшем случае.

**13. (\*) Randomized sort lower bound**

Покажите, что не существует такой *вероятностной* сортировки, которая корректно сортирует массив с вероятностью не менее  $\frac{1}{2}$  и имеет среднее время работы (матожидание)  $o(n \log n)$ .

## Разбор задач практики

### 1. Нижняя оценка на почти рабочую сортировку

Чтобы различить  $\frac{1}{100}n!$  вариантов ответа, нужно хотя бы  $\log(\frac{1}{100}n!) = \Omega(n \log n)$  сравнений.

### 2. Почти отсортированный массив

- За  $\mathcal{O}(nk)$ . Сортировка вставками.  $k$  итераций пузырька. Сортировка выбором из  $k$  ближайших.
- За  $\mathcal{O}(n + I)$ : сортировка вставками (см. лекцию).
- За  $\mathcal{O}(n \log k)$ .

- **Способ #1.** Поддерживаем в куче элементы  $a[i..i + k]$ . На каждом шаге приписываем к результату минимум из кучи и кладем в кучу следующий  $(i + k + 1)$  элемент.

- **Способ #2.** Разбиваем массив на кусочки длины  $k$ , сортируем каждый, затем слева направо сливаем куски:  $\text{merge}(1, 2)$ ,  $\text{merge}(2, 3)$ ,  $\text{merge}(3, 4)$ , ...

**Корректность:** после первого  $\text{merge}$  все элементы первого куска стоят на своих местах, т.к. исходно находились или в первом, или во втором. Далее по индукции.

- Нижняя оценка  $\Omega(n \log k)$ : в  $i$ -м куске длины  $k$  может быть любая перестановка элементов  $ik+1, \dots, ik+k$ , на сортировку каждого куска нужно  $k \log k$  сравнений, кусков  $n/k$ . Иными словами: число перестановок, которые нужно уметь отличать, не менее  $(k!)^{n/k}$ . Получаем число сравнений не менее  $\log(k!)^{n/k} = \frac{n}{k} \log k! = \Omega(n \log k)$ .

### 3. Anti-QuickSort test

Чтобы QSort работал за квадрат, достаточно получить рекуррентность  $T(n) = T(n-1) + n$ .

- Если разбиваем по первому элементу, то подойдут  $\{1, 2, \dots, n\}$  и  $\{n, \dots, 2, 1\}$ .
- Если разбиваем по среднему, поставим в середину число  $n$ .

Нужно, чтобы после  $\text{partition}$  получился плохой массив размера  $n - 1$ .

Пусть  $\text{partition}$  реализован так: `code`  $\Rightarrow$

во время  $\text{partition}$  средний элемент  $n$  поменяется местами с последним.

Строим рекурсивно массив  $a$  длины  $n - 1$ , пишем в конец  $n$ , делаем  $\text{swap}(a[n/2], a[n - 1])$ .

### 4. Ломаная без самопересечений

- Ломаная:  $\text{sort}$  пар  $\langle x, y \rangle$ , соединим в таком порядке.
- Замкнутая ломаная: вокруг любой из данных точек  $\text{sort}$  по углу, при равенстве по возрастанию расстояния. Точки с самым большим углом нужно сортировать по убыванию расстояния.

### 5. Сканирование отрезков

- Непересекающиеся отрезки.** Отрезок  $= [L_i, R_i]$ . Сортируем по возрастанию  $R_i$ , жадно берём отрезки. Проверка, что можно взять:  $L_i > R_{\text{last}}$ .

**Корректность:** рассмотрим любой ответ, пусть в нём нет  $\text{min}$  по  $R_i$  отрезка. Тогда можно заменить самый левый на  $\text{min}$  по  $R_i$ , он заканчивается ещё левее  $\Rightarrow$  не пересечётся с оставшимися. Итого  $\exists$  оптимальный ответ, содержащий  $\text{min}$  по  $R_i$ .

Из оставшихся по индукции выгодно снова взять минимальный по  $R_i$ .

- События.** Пары  $(l_i, \text{open})$ ,  $(r_i, \text{close})$  назовём событиями, сортируем их ( $\text{open} < \text{close}$ ). Сканируем все события слева направо, поддерживая число открытых отрезков. Находясь

в точке  $x$ , мы прошли все события левее  $x \Rightarrow$  знаем, сколько отрезков накрывает  $x$ .

При целых координатах часто удобно делать  $(l_i, \text{open}), (r_i + 1, \text{close})$ .

- с) **То же самое.** В прошлом пункте обозначим за  $x_i$  координаты событий. Тогда весь промежуток  $(x_i, x_{i+1})$  покрыт одинаковым числом отрезков  $k_i$ .  $\text{ans}[k_i] += x_{i+1} - x_i$ .

## 6. Покраска забора

*Решение #1.* Если мы владеем деревом отрезков с присваиванием на отрезке, то можно напрямую применить его к этой задаче.  $\mathcal{O}(n + q \log n)$ . Если не владеем, это прекрасно, задачу можно решить проще!

*Решение #2.* Идем слева направо, обрабатываем события «начало отрезка» и «конец отрезка». Храним открытые отрезки в куче (**set**), сортирующей их по времени покраски.

Каждую точку красим в цвет открытого отрезка с самым поздним временем.

Для каждой точки цвет узнаем за  $\mathcal{O}(1)$  (посмотреть минимум), каждое начало и конец отрезка – положить или вынуть из кучи за  $\mathcal{O}(\log q)$ .

*Решение #3.* Будем выполнять запросы с конца. Выполнить запрос = покрасить все ещё не покрашенные клетки на отрезке. Пусть у каждой клетки  $i$  есть цвет  $c_i$  и указатель на ближайшую справа непокрашенную  $p_i$ . Тогда

```
1 int paint(int l, int r, int color):
2   if (l > r) return l;
3   if (c[l] == -1) c[l] = color;
4   return p[l] = paint(p[l], r, color);
```

`paint` возвращает клетку, до которой докрасил.

Заметим сходство этого кода с СНМ со сжатием путей. В таком виде он работает за  $\mathcal{O}((n+q) \log n)$ . Если добавить ранговую эвристику, будет  $\mathcal{O}((n+q)\alpha(n))$ .

## 7. Коровы – в стойла!

Бинпоиск по ответу. Проверка, что можно выбрать  $m$  стойл на расстоянии  $\geq d$ : самое левое берём, от него самое левое на расстояние хотя бы  $d$  и т.д.

## 8. K-best

$\frac{\sum_{i=1}^k a_{p_i}}{\sum_{i=1}^k b_{p_i}} \rightarrow \max$ . Бинпоиск по ответу:  $\frac{\sum_{i=1}^k a_{p_i}}{\sum_{i=1}^k b_{p_i}} \geq M \Leftrightarrow \sum_{i=1}^k (a_{p_i} - Mb_{p_i}) \geq 0$ ,

наличие таких можно проверить, взяв  $k$  индексов с максимальными  $a_j - Mb_j$ .

Чтобы взять  $k$  максимумов за  $\mathcal{O}(n)$ , выберем  $(n - k)$ -ю статистику и вернём все элементы после неё. Сложность  $\mathcal{O}(n \log \text{MAX})$ .

## 9. Поиск точки разреза

а) Предикат « $a_i > a_{i-1}$ » сперва **TRUE**, затем **FALSE**. Бинпоиском ищем границу.

б) Рассмотрим массив  $\{1, 1, \dots, 1, 1, 2, 1, 1, \dots, 1, 1\}$ . Мы ищем число 2, оно может быть на любой позиции. Если мы посмотрели не все ячейки, то число 2 может оказаться там, где не смотрели  $\Rightarrow$  в худшем случае нужно просмотреть все  $n$  ячеек.

с) Предикат « $a_i \geq a_0$ » сперва **TRUE**, затем **FALSE**. Бинпоиском ищем границу.

## 10. Сложность случайных сортировок

Если с вероятностью  $p$  монетка выпадет орлом, то среднее число подкидываний до первого орла  $E = \frac{1}{p}$ .

Доказательство:  $E = 1 + (1 - p)E$  (один бросок делаем всегда, с вероятностью  $1 - p$  не повезло,

и нужно повторить процесс).

Функцию `sorted` можно вызывать не каждый раз, а раз в  $n$  шагов, тогда её амортизированная сложность  $\mathcal{O}(1)$ .

а) Каждый `swap` уменьшает число инверсий  $I$  хотя бы на один.  $T(I)$  – время работы, если сейчас  $I$  инверсий.

Вероятность попасть в инверсию  $p = \frac{I}{n(n-1)/2} \Rightarrow$  среднее число проб до попадания в инверсию  $\frac{n(n-1)/2}{I}$ .

$$T(I) \leq \frac{n(n-1)/2}{I} + T(I-1) = \frac{n(n-1)}{2} \sum_{j=1}^I \frac{1}{j} = \mathcal{O}(n^2 \log n).$$

**Забавные факты.** Более точный анализ даст оценку  $\Theta(n^2)$ . Если, когда инверсий осталось  $\approx n \log^2 n$ , переключиться на сортировку вставками, время будет  $\Theta(n \log^2 n)$ .

б) На каждом шаге перестановка близка к случайной  $\Rightarrow$  с вероятностью  $\frac{1}{n!}$  отсортирована  $\Rightarrow$  среднее время работы  $\Theta(n!)$ .

## 11. (\*) Второй максимум

**Рандомизированное решение.** Сделаем `random_shuffle` массива, который делает 0 сравнений и работает за  $\mathcal{O}(n)$ . Теперь:

```
1 int m1 = a[1], m2 = INT_MIN;
2 for (int i = 2; i <= n; i++)
3   if (a[i] > m2)
4     if (a[i] > m1) m2 = m1, m1 = a[i];
5     else m2 = a[i];
```

На  $i$ -й итерации цикла мы войдём в `if` с вероятностью  $\frac{2}{i}$ , поэтому число сравнений равно  $n-1 + \sum_{i=2}^n \frac{2}{i} = n + 2 \ln n + \Theta(1)$ .

**Детерминированное решение кучеобразной структурой.**

Построить кучу мы за  $n$  сравнений не сможем, зато можем толкнуть каждый элемент вверх.

```
1 for (int i = n; i > 1; i++) if (a[i / 2] < a[i]) swap(a[i / 2], a[i]);
```

В результате максимум всплывёт вверх. Где искать второй максимум? Он не всплыл до корня, потому что где-то по дороге встретился с первым максимумом  $\Rightarrow$  он в детях вершин, по которым прошёл максимум, всего  $\log_2 n$  кандидатов.

**Детерминированное решение деревом отрезков.**

За  $n-1$  сравнение построим дерево отрезков. За  $\mathcal{O}(\log n)$  найдём в нём 2-й максимум.

## 12. (\*) Binary search lower bound

Пусть искомый элемент находится в последних  $\frac{n}{2}$  элементах. Тогда, чтобы его найти нам нужно сделать хотя бы  $\log \frac{n}{2}$  обращений к элементам, чтобы различить  $\frac{n}{2}$  возможных ответов. Каждое обращение работает хотя бы за  $\frac{n}{2}$ . Итого  $\Omega(n \log n)$ .

## 13. (\*) Randomized sort lower bound

Есть ещё более сложная задача: дать оценку  $\Omega(n \log n)$  на число сравнений.

Нам же нужно оценить лишь время работы, поэтому число случайных бит  $r$ , которые мы просмотрим, то же должно быть  $o(n \log n)$ . Итого  $k$  сравнений +  $r$  случайных бит дают  $2^{r+k}$  различных исходов  $\Rightarrow$  для корректной сортировки  $2^{r+k} \geq n! \Rightarrow r + k = \Omega(n \log n)$ .

# Домашнее задание

## 3.1. Обязательная часть

### 1. (3) Сложность сортировок

- (1) Покажите, что любая сортировка, которая верно работает хотя бы на доле  $\frac{1}{100^n}$  от всех перестановок, не может работать за  $o(n \log n)$  на всех тестах.
- (1) Покажите, что нельзя реализовать структуру данных, которая умеет то же, что и куча: `Add`, и `ExtractMin`, но за  $o(\log n)$  сравнений.
- (1) Покажите, что нельзя сделать `merge(a, a+n/2, a+n)` за  $o(n)$  сравнений. `merge` сливает левую и правую половины, результат кладёт в `[a, a+n)`

### 2. (5) Отрезки на прямой

Дано  $n$  отрезков  $[a_i, b_i]$ .

- (1.5) Найти длину объединения отрезков.
- (1.5) Выбрать минимальное число отрезков, покрывающих отрезок  $[0, M]$ .  $\mathcal{O}(n \log n)$ .  
В этой задаче важна строгость доказательства. Решение за  $\mathcal{O}(n^2)$  получит (1) балл.
- (2) Теперь пусть границы отрезка меняются со временем.  $i$ -й отрезок в момент времени  $t$  равен  $[x_i - t \cdot r_i, x_i + t \cdot r_i]$ . Найти первый момент времени, когда отрезки покроют  $[0, M]$ .

### 3. (2) Ускорение SiftUp

Модифицируйте операцию `SiftUp` для бинарной кучи так, чтобы она по-прежнему работала за  $\mathcal{O}(\log n)$ , но при этом делала лишь  $\mathcal{O}(\log \log n)$  сравнений.

### 4. (2) Ускорение SiftDown

Модифицируйте операцию `SiftDown` для бинарной кучи так, чтобы она по-прежнему работала за  $\mathcal{O}(\log n)$ , но при этом делала лишь  $\log_2 n + \mathcal{O}(\log \log n)$  сравнений.

### 5. (2) Anti-QuickSort test

Построить за  $\mathcal{O}(n^2)$  перестановку длины  $n$ , на котором `QuickSort` отработает за  $\Omega(n^2)$ , если разбивающего элемента берется следующим образом.

Есть некоторая произвольная детерминированная функция `pivot(l, r)`, она возвращает число от  $l$  до  $r$ ,  $x = a[\text{pivot}(l, r)]$ .

Есть произвольная детерминированная `partition(l, r, x)`. То, как она переставит элементы на отрезке  $[l, r]$ , зависит только от результатов сравнения элементов с  $x$ . Естественно, она делает `partition` – сначала ставит элементы  $< x$ , потом  $x$ , потом  $> x$ .

Алгоритм построения плохой перестановки имеет право вызывать и `pivot`, и `partition`.

В попытках объяснить, что именно вы делаете, можно использовать куски кода.

(\*) Можно получить (+1) балл для следующего случая: `pivot(l, r)` возвращает три индекса  $l \leq i, j, k \leq r$ . За  $x$  берется медиана чисел  $a[i], a[j], a[k]$ . Дальше то же самое.

## 3.2. Дополнительная часть

### 1. (2) Различные элементы

*Задача:* дан массив, проверить, что все элементы различны.

Пусть для работы с элементами массива нам доступна только операция сравнения.

Доказать оценку  $\Omega(n \log n)$  на число сравнений.

### 2. (2) Дополнительные отрезки на прямой

Есть  $n$  отрезков на прямой. Выбрать из них максимальное по размеру множество, покрывающее каждую точку не более, чем  $k$  раз. Важна строгость доказательства.

(1) балл можно получить за корректное решение без доказательства.

### 3. (2) Отрезки на окружности

Есть  $n$  отрезков на окружности. Выбрать из них максимальное по размеру множество, покрывающее каждую точку не более, чем один раз. Оцениваться будут решения за  $o(n^2)$ .