

SPb HSE, 1 курс, осень 2024/25

Практика по алгоритмам #4

Бинпоиск, два указателя

25 сентября

Собрано 25 сентября 2024 г. в 16:04

---

## Содержание

1. Бинпоиск, два указателя	1
2. Разбор задач практики	3
3. Домашнее задание	7
3.1. Обязательная часть . . . . .	7
3.2. Дополнительная часть . . . . .	9

# Бинпоиск, два указателя

## 1. C++.STL

```
1 priority_queue<int> q; q.push(1); q.top(); q.pop(); // max element
2 make_heap, pop_heap, push_heap, sort_heap; // делает ровно то, что обсудили на лекции
```

## 2. Strange bound

Дан  $x$  и сортированный массив. Найти  $\max i: a[i] \leq x$ . Используйте STL.

## 3. Амортизация

- У нас есть двоичное число длины  $k$  (массив из  $k$  нулей и единиц). Изначально число равно 0, затем к нему  $n$  раз прибавляется единица методом “в лоб” (идём с младших разрядов, делаем переносы). Доказать сложность  $\mathcal{O}(1)$  на операцию и прямым подсчетом, и методом потенциалов.
- Рассмотрим реализацию хеш-таблицы на списках с фиктивным удалением, в которой когда число добавленных элементов (в том числе помеченных удаленными), превышает число списков, происходит перевыделение и удвоение числа списков, затем копирование из старой таблицы в новую. Покажите амортизированную сложность добавления  $\mathcal{O}(1)$ .

## 4. Два указателя

- Найти в данном массиве число отрезков, содержащих ровно  $k$  единиц.  $\mathcal{O}(n)$ .
- Найти отрезок максимальной длины без повторяющихся элементов.  $\mathcal{O}(n)$ .
- Найти в массиве  $a$  позиции  $i, j, k: a[i] + a[j] + a[k] = S$  за  $\mathcal{O}(n^2)$ .

## 5. Поиск статистик

Даны два отсортированных массива длины  $n$ . Без дополнительного подсчета найти  $k$ -ю порядковую статистику в объединении массивов.

- За  $\mathcal{O}(\log n \cdot \log \text{MAX})$ . Значения элементов по модулю не превышают  $\text{MAX}$ .
- За  $\mathcal{O}(\log n)$ .

## 6. Для любителей статистики

Дан массив размера  $n$ . За  $\mathcal{O}(\log n)$  отвечать на запрос «сколько раз встречается число  $x$  на отрезке  $[L, R]$  массива?» Можно сделать преобработку массива.

## 7. Генерация сортировки пар

Даны два массива  $a$  и  $b$  длины  $n$ , сгенерировать все попарные суммы  $a_i + b_j$  в отсортированном порядке.

- За  $\mathcal{O}(n^2 \log n)$ .
- За  $\mathcal{O}(n^3)$  с использованием  $\mathcal{O}(n)$  дополнительной памяти.
- За  $\mathcal{O}(n^2 \log n)$  с использованием  $\mathcal{O}(n)$  дополнительной памяти.
- (\*) За  $\mathcal{O}(n^3)$  с использованием  $\mathcal{O}(1)$  дополнительной памяти.

**8. Поиск периода**

Дана последовательность чисел:  $x_0 = a$ ,  $x_{i+1} = f(x_i)$ .  
Найти длину периода  $T$  и длину предпериода  $L$ .

- За  $\mathcal{O}(L + T)$ , если у нас  $\mathcal{O}(L + T)$  доппамяти. Далее во всех пунктах  $\mathcal{O}(1)$  памяти.
- Пусть известен  $L$ , найти  $T$  за  $\mathcal{O}(L + T)$ .
- Найти  $T$  за  $\mathcal{O}((L + T) \log(L + T))$ .
- Найти  $T$  за  $\mathcal{O}(L + T)$ .
- Найти  $L$  за  $\mathcal{O}((L + T) \log(L + T))$ .
- Найти  $L$  за  $\mathcal{O}(L + T)$ .

**9. Поиск повтора**

Дан массив из  $n + 1$  целого числа от 1 до  $n$ . Массив доступен *только на чтение*, есть  $\mathcal{O}(1)$  дополнительной памяти. Найти за  $\mathcal{O}(n)$  любое число, которое встречается хотя бы два раза.

**10. (\*) Треугольники**

Даны  $N$  различных чисел. Рассмотрим треугольники с попарно различными сторонами. Сколько существует треугольников с такими длинами сторон?  $\mathcal{O}(N^2)$ .

**11. (\*) Среднее арифметическое**

Найти отрезок с максимальным средним арифметическим, при этом длины от  $L$  до  $R$ .

**12. (\*) Meet-in-the-middle**

Дана последовательность  $a_1, a_2, \dots, a_n \in \mathbb{N}$  и  $S \in \mathbb{N}$ . Нужно найти *подмножество* этой последовательности с суммой  $S$ . Существует решение за  $\mathcal{O}(n2^{n/2})$ . Сложим в **set** суммы всех подмножеств первых  $n/2$  чисел. Переберем все подмножества вторых  $n/2$  чисел, рассматривая сумму  $x$ , ищем в **set**-е число  $S - x$ . Придумайте, как решить эту задачу за  $\mathcal{O}(\text{poly}(n)2^{n/2})$  времени и  $\mathcal{O}(\text{poly}(n)2^{n/4})$  памяти.

**13. (\*) Быстрое сравнение матриц**

Даны матрицы  $A, B, C$  размера  $n \times n$  над  $\mathbb{F}_2$ . Проверить за  $\mathcal{O}(n^2)$ , верно ли, что  $AB = C$ .

Матрицы умножаются по правилу  $(AB)_{ij} = \sum_{t=1}^n A_{it}B_{tj}$ .

**14. (\*) Треугольники-2**

Даны  $N$  различных чисел. Рассмотрим треугольники с попарно различными сторонами.

- Найти треугольник минимальной площади с такими длинами сторон.  $\mathcal{O}(N^2)$ .
- Найти треугольник максимальной площади с такими длинами сторон.  $\mathcal{O}(N^2)$ .

# Разбор задач практики

## 1. C++.STL

**Пример** с `priority_queue`, **пример** с `make_heap`.

## 2. Strange bound

`upper_bound(x)` - 1 - последний  $\leq x$ .

## 3. Амортизация

а) В любом случае нужно смотреть за величиной  $\varphi$  «число единиц в двоичной записи». Заметим «за  $n$  запросов +1 добавится  $n$  единиц  $\Rightarrow$  удалится  $\leq n$  единиц  $\Rightarrow \mathcal{O}(n)$ ».

*Способ через потенциал.* Пусть мы делаем  $k$  переносов, тогда  $t_i = k + 1$ ,  $\Delta\varphi = -k + 1$ ,  $a_i = t_i + \Delta\varphi = 2$ ,  $\varphi_0 = 0, \varphi \geq 0 \Rightarrow \sum t_i \leq \sum a_i$ .

*Плохой потенциал.*  $\varphi =$  числу единиц с конца. Если число меняется как  $101111\dots1110 \rightarrow 101111\dots1111$ , то  $t_i = 1$ , а  $\Delta\varphi$  произвольное положительное число.

*Способ через сумму.* Крайний бит меняется каждую операцию, второй бит справа каждую вторую. Формально,  $k$ -й справа бит меняется каждые  $2^{k-1}$  операций, так как должен произойти полный цикл на предыдущих  $k - 1$  битах.

Итого  $\sum_{k=0}^{\log n} \frac{n}{2^k} = \Theta(n)$ .

б) **Хеш-таблица.** Пусть списков  $m$ , элементов  $n$ ,  $\varphi = -(m - n)$ .

При перевыделении  $a_i = m + (-(2m - m) - 0) = \mathcal{O}(1)$ .  $\varphi_0 - \varphi_N \leq 2N$ .

## 4. Два указателя

а) **Число отрезков, содержащих ровно  $k$  единиц.**

Три указателя: для каждого  $r$  находим минимальное и максимальное подходящее  $l$ .

```

1 int l1 = -1, l2 = -1, r = -1;
2 for (int cnt = 0; cnt < k && r < n; cnt += a[++r])
3     ;
4 for (; r < n; ++r) {
5     if (a[r] == 1) { // изменилось число единиц на отрезке!
6         l1 = l2;
7         for (l2++; a[l2] == 0; l2++)
8             ;
9     }
10    ans += l2 - l1;
11 }
```

Инварианты между итерациями:  $a[l_1] == a[l_2] == a[r] == 1$ , на отрезке  $[l_2, r]$  ровно  $k$  единиц, на отрезке  $[l_1, r]$  ровно  $k + 1$ .

Способ #2.

```

1 int c1 = 0, c2 = 0, l1 = 0, l2 = 0;
2 for (int r = 0; r < n; ++r)
3     if (a[r] == 1) ++c1, ++c2;
4     while (c1 > k) if (a[l1++] == 1) --c1;
5     while (c2 >= k) if (a[l2++] == 1) --c2;
6     ans += l2 - l1;
```

Способ #3.

```

1 int pos = -1, ans = 0;
2 vector<int> d;
3 for (int i = 0; i <= n; i++)
4     if (i == n || a[i] == 1)
5         d.push_back(i - pos), pos = i;
6 for (size_t j = k; j <= d.size(); j++)
7     ans += d[j - k] * d[j];

```

b) **Отрезок максимальной длины без повторяющихся элементов.**

```

1 int l = 0;
2 for (int r = 0; r < n; ++r) {
3     while (used[a[r]])
4         used[a[l++]] = false;
5     used[a[r]] = true;
6     ans = max(ans, r - l + 1);
7 }

```

При больших числах в качестве `used` используется хеш-таблица.

Способ #2.

```

1 int l = 0;
2 for (int r = 0; r < n; ++r) {
3     if (last[a[r]] >= l) l = last[a[r]] + 1;
4     last[a[r]] = r;
5     ans = max(ans, r - l + 1);
6 }

```

c) **3-SUM.**

Сортируем  $a$ . Перебираем  $k$ , ищем за  $\mathcal{O}(n)$   $a[i] + a[j] = S1 = S - a[k]$ .

```

1 int j = n - 1;
2 for (int i = 0; i < n; ++i) {
3     while (a[i] + a[j] > S1) --j;
4     if (a[i] + a[j] == S1) ok;
5 }

```

Кстати, полезно знать, что эта задача не решается за  $\mathcal{O}(n^{2-\epsilon})$ .

**5. Поиск статистик**

Задача переформулируется «выбрать  $k$  элементов, чтобы  $\max$  выбранных был поменьше».

a) *Кратко:*  $\mathcal{O}(\log n \cdot \log \text{MAX})$ : бинпоиск по ответу, внутри бинпоиск по массиву.

Делаем бинпоиск по  $x$ , внутри считаем в обоих массивах кол-во эл-ов  $\leq x$ .

Если насчитали хотя бы  $k$ , то ответ  $\leq x$ , иначе ответ  $> x$ .

b)  $\mathcal{O}(\log n)$ . Пусть в  $a$  взяли первых  $i$  элементов, тогда в  $b$  взяли  $k-i$  элементов.

Когда выгодно взять ещё один из  $a$  (добавить  $a[i]$  и выкинуть  $b[k-i-1]$ )?

iff  $a[i] < b[k-i-1] \Rightarrow$  ищем бинпоиском  $\min i: a[i] \geq b[k-i-1]$ .

Ответ =  $\max(a[i-1], b[k-i-1])$ .

**6. Для любителей статистики**

Отсортируем пары  $(a[i], i)$ . Тогда ответ это  $\text{upper\_bound}(x, R) - \text{lower\_bound}(x, L)$ .

*Другое решение:* `unordered_map<int, vector<int>>` – храним по числу индексы всех его вхождений в отсортированном порядке.

## 7. Генерация сортировки пар

a) За  $\mathcal{O}(n^2 \log n)$ : просто сгенерируем все пары и отсортируем.

b) За  $\mathcal{O}(n^3)$  и  $\mathcal{O}(n)$  памяти: ( $n$  указателей)

Отсортируем  $b$ . Для каждого  $a_i$  мы выведем суммы в порядке  $a_i + b_1, a_i + b_2 \dots \Rightarrow$  достаточно поддерживать для каждого  $a_i$   $\min$  индекс  $j_i$ : мы ещё не вывели в ответ  $a_i + b_{j_i}$ .

Алгоритм:  $n^2$  раз за  $\mathcal{O}(n)$  ищем  $i$ :  $a_i + b_{j_i} = \min$ , выводим, делаем  $j_i++$ .

c) За  $\mathcal{O}(n^2 \log n)$  с  $\mathcal{O}(n)$  дополнительной памяти: (need min? use heap!)

Улучшим предыдущее решение: будем хранить пары  $\langle a_i + b_{j_i}, i \rangle$  в куче.

Достали за  $\log n$  такое  $i$ :  $a_i + b_{j_i} = \min$ , вывели,  $j_i++$ , обновили за  $\mathcal{O}(\log n)$  кучу.

d) За  $\mathcal{O}(n^3)$  с  $\mathcal{O}(1)$  дополнительной памяти: отсортируем оба массива. После того, как вывели в ответ  $x$ , двумя указателями за  $\mathcal{O}(n)$  найдем минимальную сумму  $> x$ , и сколько раз она встречается.

## 8. Поиск периода

**С доп-памятью.** Храним хеш-мапу  $x_i \rightarrow i$ .

**С лишним логом.** Бинпоиск по длине предпериода  $L$ .

Tortoise-and-hare = черепаха-и-заяц = алгоритм Флойда.

a)  $x = x_0, y = f(x_0); \text{ while } (x \neq y) \ x = f(x), y = f(f(y));$

Каждый шаг расстояние между  $x$  и  $y$  увеличивается на один.

За  $\leq L + T$  шагов остановимся в точке на цикле. Обозначим её  $z$ .

b)  $x = f(a), T = 1; \text{ while } (x \neq z) \ x = f(x), ++T;$

**Способ #2:** алгоритм Брента.

Пытаемся угадать  $L + T$ , ищем такое  $k$ , что  $2^{k-1} \leq L + T \leq 2^k$ .

$x = f^{(2^k)}(x_0)$ , перебираем  $y$  от  $f(x)$  не более  $2^k$  шагов вперёд.

Время работы  $1 + 2 + 4 + \dots + 2^k = \mathcal{O}(L + T)$ .

```

1 x = x0, y = f(x), k = 0, steps = 2^k
2 while (x != y)
3     if (!--steps) k++, steps = 2^k, x = y;
4     y = f(y)

```

*Поиск предпериода:* возьмём два указателя на расстоянии ровно  $T$  и будем параллельно двигать вперёд.  $x = x_0, y = f^{(T)}(x_0); \text{ while } (x \neq y) \ x = f(x), y = f(y), L++;$

## 9. Поиск повтора

Последовательность  $x_1 = n + 1, x_{i+1} = a[x_i]$  периодична и имеет ненулевой предпериод, так как ни один элемент не равен  $n+1$ . Нам нужно начало периода, для этого достаточно узнать длину предпериода  $L$ , воспользуемся предыдущей задачей.

## 10. (\*) Треугольники

Сортируем стороны:  $x[i] < x[j] < x[k]$ . Для каждого  $i$  запускаем два указателя  $(j, k)$ :

$\forall j=i+1..n-1$  ищем  $\min k$ :  $x[k] \geq x[i] + x[j]$ . Увеличиваем  $ans$  на  $(k-j)$ .

## 11. (\*) Среднее арифметическое

Общая идея для минимизации частного: *бинпоиск по ответу*.

Среднее арифметическое  $> x \Leftrightarrow$  сумма для  $b_i = a_i - x$  положительна  $\Rightarrow$   
Найдём отрезок максимальной суммы длины от  $L$  до  $R$ .

## 12. (\*) Meet-in-the-middle

Рассмотрим другое решение за  $\mathcal{O}(n2^{n/2})$  времени и памяти:

- Сгенерируем суммы всех подмножеств первых  $\frac{n}{2}$  чисел, положим в массив  $a$ .
- Сгенерируем суммы всех подмножеств оставшихся  $\frac{n}{2}$  чисел, положим в массив  $b$ .
- Сортируем  $a$ ,  $b$  и двумя указателями ищем элементы двух массивов, дающие сумму  $S$ .

Вместо  $a$  и  $b$  длины  $2^{n/2}$  сохраним всевозможные суммы кусков длины  $\frac{n}{4}$  — массивы  $a_1, a_2, b_1, b_2$  длины  $2^{n/4}$ , отсортируем их.

Используем задачу про генерацию отсортированных по сумме пар за  $\mathcal{O}(k^2 \log k)$  с  $\mathcal{O}(k)$  памяти. Передвижение указателя по  $a$  заменяется на генерацию очередной пары из массивов  $a_1, a_2$ .

Время  $(2^{n/4})^2 \log(2^{n/4}) = \Theta(n2^{n/2})$ , память  $\Theta(2^{n/4})$ .

## 13. (\*) Быстрое сравнение матриц

Матрицу можно умножить на вектор за  $n^2$ , умножение матриц ассоциативно  $\Rightarrow$

Для нескольких случайных  $x$  проверим  $A(Bx) = Cx$ .

Если  $AB = D \neq C$ , то найдется  $D_{ij} \neq C_{ij} \Rightarrow$  вероятность  $D_i \cdot x = C_i \cdot x \leq \frac{1}{2}$ :  
если для какого-то  $x$  это верно, то для  $x$  с измененным  $j$ -м битом неверно.

## 14. (\*) Треугольники-2

Считаем, что стороны отсортированы и  $a < b < c$ .

- Минимальная площадь.* При фиксированных  $b$  и  $c$  высота, опущенная на сторону  $c$ , тем меньше, чем ближе  $a + b$  к  $c$ . Фиксируем  $b, c$ , выбираем  $\min a: a + b > c \wedge a < b$ .  $\mathcal{O}(n^2)$ .
- Максимальная площадь.* Рассуждение выше даст решение за  $\mathcal{O}(n^2)$ .

Можно показать, что на самом деле ответ – три последовательных элемента массива.

# Домашнее задание

## 3.1. Обязательная часть

### 1. (2) Параллельный минимум и максимум

Дан массив из  $2n$  чисел. Найти минимальное **И** максимальное за  $3n-2$  сравнения.

### 2. (4) Поиск статистики

Даны  $m$  сортированных массивов длины  $n$ . Нужно без дополнительного подсчета найти  $k$ -ю порядковую статистику. Числа от 0 до MAX.

- (2) За  $\mathcal{O}(m + k \log m)$ .
- (2) За  $\mathcal{O}(m \log \text{MAX} \log n)$ .
- (+1) допбалл: вероятностное  $\mathcal{O}(m \log^2 \text{poly}(n, m))$ .

### 3. (1) Ближайший по значению

Даны отсортированные массивы  $a$  и  $b$  длины  $n$ .

Для каждого элемента  $a$  найти ближайший по значению элемент  $b$ .  $\mathcal{O}(n)$ .

### 4. (2) В этом задании требуется прислать код на языке C/C++! <sup>1</sup>

Множество и мультимножество можно хранить в виде отсортированного массива. Даны два множества  $A$  и  $B$  в отсортированном виде, за  $\mathcal{O}(|A| + |B|)$  построить в таком же виде их множество-разность. Пример:  $\{1, 2, 3\} \setminus \{2, 4\} = \{1, 3\}$ .

TeX: пример вставки кода через пакет `verbatim` [ex1].

TeX: пример вставки кода через пакет `listing` [ex2].

C++: шаблон кода для сдачи этой задачи:

```
1 vector<int> setDifference(vector<int> a, vector<int> b) { ... }
```

### 5. (2) Ближайший по координате

Даны  $n \leq 10^6$  точек на плоскости, координаты целые до  $10^9$  по модулю. Приходят  $q \leq 10^6$  запросов: дана точка  $p$ , найти для нее ближайшие по  $X$  слева и справа точки с таким же  $Y$  и ближайшие по  $Y$  точки с таким же  $X$  сверху и снизу. Запросы online, нужно на каждый запрос отвечать сразу, а не на все вместе в конце.

### 6. (2) Амортизация не нужна

Избавиться от амортизации в «векторе с увеличением не в 2, а в  $C > 1$  раз».

<sup>1</sup>Можно прикрепить код, можно вставить его в tex. Код должен компилироваться и работать. Не пишите, пожалуйста, `int main() ...`, оформите код, как функцию.



## 7. (2) Отложенные операции

Вспомним структуру данных с лекции, хранящую мультимножество целых чисел  $S$  и умеющую обрабатывать два запроса:

- $S.count(L, R)$  – количество элементов в  $S$  от  $L$  до  $R$ .
- $S.add(x)$  – добавить  $x$  в  $S$ .

У нас было два решения, возьмём первое «корневую оптимизацию». Рассмотрим модификацию (выделена синим). Вся структура: храним отсортированный массив  $a$ , храним массив  $b$  в произвольном порядке, поддерживаем  $|b| \leq \sqrt{|a|}$ .

На запросы `count` отвечаем, делая бинпоиск в  $a$  и линейный проход в  $b$ .

`add(x)` добавляет элемент в конец  $b$  за  $\mathcal{O}(1)$ , если после этого  $|b| > \sqrt{|a|}$ , делаем `a = merge_arrays(a, sort(b)); b = empty`.

Найти и доказать амортизированное время обработки запросов. Нужно использовать потенциалы. За оценку времени без потенциалов можно получить частичный балл.

## 8. (2) Патчим структуру

Возьмём структуру из предыдущего задания дополните её функцией `merge(S1, S2)`, которая разрушает  $S_1$ ,  $S_2$  и создаёт  $S_1 \cup S_2$ . `count` должен работать за то же время, что и раньше. Оцените суммарное время  $n$  произвольных операций `add`, `merge` (изначально все множества пусты). Должно получиться  $o(n^2)$ .

*Подсказка:* обязательно почитайте нужный кусок конспекта.

## 3.2. Дополнительная часть

### 1. (3) Коробки с шарами

Дано  $2 \cdot n - 1$  коробок с черными и белыми шарами. В  $i$ -ой коробке находится  $w_i$  белых и  $b_i$  черных шаров. Всего в коробках находится  $W$  белых и  $B$  черных шаров. Требуется выбрать  $n$  коробок, чтобы суммарное число белых шаров в них было не менее  $\frac{W}{2}$ , а черных не менее  $\frac{B}{2}$ . Решить за  $\mathcal{O}(n \log n)$ .

### 2. (3) Коробки с предметами

Даны  $N$  предметов с весами  $w_i$  и бесконечный набор коробок размера  $W$ . Разложить предметы в минимальное число коробок при условии, что в одну коробку можно класть не более двух предметов.

### 3. (3) О трудности коммуникации

У Алисы есть массив  $a_1, a_2, \dots, a_{k_1}$ , у Боба есть  $b_1, b_2, \dots, b_{k_2}$ .

В каждом массиве числа от 1 до  $n$ .

В каждом массиве числа различны, но может быть одно и то же число в обоих массивах.

Алиса и Боб знают  $n$ , но не знают ничего про чужой массив.

Они хотят найти медиану объединения (merge) своих массивов, то есть  $\lfloor (k_1 + k_2)/2 \rfloor$  элемент отсортированного объединения (merge). Алиса и Боб могут общаться!

Придумайте стратегию, по которой они найдут медиану, если они могут переслать другу другу суммарно  $\mathcal{O}(\log n)$  бит. За  $\mathcal{O}(\log^2 n)$  бит можно получить (2) балла.