Студент: Тимур Хабибуллин
Группа: 1
Дата: 5 мая 2022 г.

---

1. **Задача про n коров в m стоилах**
   Считаем координаты стоил в массив и за $O(m\log(m))$ отсортируем его в порядке позрастания:

$$\texttt{A}: \ \ x_1 < x_2 < ... < x_m$$

   После это воспользуемся бинарным поиском по ответам: определим функцию (тест), которая будет по данному δx проверять, можно ли расставить коров по стойлам с минимальным расстоянием $\Delta_{min} \geqslant \delta x$. Для этого мы сразу проверяем, что количество стоил не меньше чем коров и если все хорошо, то заведем переменные

$$\texttt{last\_cow} = x_1, \text{ координата последней коровы}$$

$$\texttt{cows} = n - 1, \text{ сколько осталось коров}$$

   Первую корову мы сразу кладем в самое левое стойло, потому что тем самым мы оставляем справа от него наибольшее количество пустых стоил, в которые можно расставить остальных коров. После этого пока мы не дошли до конца массива или не пересчитали всех коров, проходимся по всем точкам x_i и на каждом шаге считаем расстояние $\texttt{cur} = x_i - \texttt{last\_cow}$ и если $\texttt{cur} \geqslant \delta x$, то помещаем в это стойло корову:

$$\texttt{last\_cow} = x_i$$

$$\texttt{cows} -= 1$$

   и двигаемся дальше. Т.е. мы ставим следующую корову в ближайшую позицию, удаленную от предыдущей не менее, чем на δx. В конце проверяем, смогли ли мы посадить всех коров и если да, то возращаем `True`. Этот тест работает за $O(m)$, поскольку мы просто проходимся по массиву. С помощью этого теста и бин поиска ищем самое правое значение $\delta x \in [0, x_m - x_1 + 1]$, при котором тест еще дает `True` и возвращаем это δx. На бин поиск по ответам тратим $O(m\log(x_m - x_1)) = O(m\log(x_{max})$ времени.
   Вся программа работает за $T = O(m[\log(m) + \log(x_{max})])$

2. **Задача про всевозможные суммы элементов двух массивов**

   (a) Считаем оба массива и заведем новый массив размера n^2, в который мы запишем всевозможные суммы $a_i + b_j, \ i, j = 1...n$. После этого отсортируем его за $O(n^2\log(n^2) = O(n^2\log(n)$ и выведем. При решении нам потребовалось $O(n^2)$ памяти.

   (b) Сначала сортируем оба массива по возрастанию за $O(n\log(n))$ времени, это не повлияет на итоговую асимптотику:

$$a: \ a_1 \leqslant a_2 ... \leqslant a_n$$

$$b: \ b_1 \leqslant b_2 ... \leqslant b_n$$

   Создадим два массива - массив индексов $\texttt{inds}[i] = 1, \ i = 1...n$ (здесь мы будем хранить индекс i сумм $a_i + b_j$), и массив `sums`, куда мы будем класть сами суммы $a_i + b_j$ по следующему правилу. В начале мы заполняем `sums` следующим образом:

$$\texttt{sums}[j] \ = \ a_1 + b_j, \ j = 1...n$$

После этого проходимся по этому массиву и за $O(n)$ находим минимум и выводим его. Пусть он достигается в некотором $\texttt{sums}[j]$. Тогда в этой ячейке массива после вывода минимума записываем $a_2 + b_j$ и в массиве индексов в j позиции увеличиваем значение на один:

$$\texttt{inds}[j] \mathrel{+}= 1$$

$$\texttt{sums}[j] \; = \; a_2 + b_j$$

Что здесь произошло? Если представить матрицу M_{ij} размера $n*n$, в которую мы вписываем суммы $a_i + b_j$, то изначально в $\texttt{sums}$ мы вписали ее первую строку $M_{1j}, j = 1...n$. Так как

$$a_i + b_j \leqslant a_{i+1} + b_j \leqslant a_{i+2} + b_j \leqslant ...$$

(мы отсортировали массивы a b), то $a_2 + b_j$ есть значение, которое может быть выведено после $a_1 + b_j$. Мы не можем записать в $\texttt{sums}[j]$ никакое другое значение из других столбиков M_{kl}, потому что в $\texttt{sums}$ уже есть значения из $l-$го столбца, которые не больше и их надо вывести раньше.

После этого мы продолжаем находить минимумы, их извлекать и менять значения в $\texttt{inds}$, $\texttt{sums}$, пока не переберем все значения $a_i + b_j$, которых n^2. В итоге получается асимптотика $O(n^3)$ и используемая память $O(n)$.

(c) Для того, чтобы получить асимптотику $O(n^2 \log(n)$ достаточно завести массив $\texttt{sums}$, записывать в него тройки $(i, j, a_i + b_j)$ по тому же правилу, что и в предыдущем пункте, и использовать его как min-кучу. Тогда на каждом шаге при поиске минимума мы будет тратить $O(\log(n)$ времени и добавлять в кучу $(i + 1, j, a_{i+1} + b_j)$. В результате получим необходимую асимптотику и линейный расход памяти.

(d) empty

3. (a) Пусть у нас есть два отсортированных по возрастанию массивы $A : \{a_i\}$, $B : \{b_j\}$, $i, j = 1...n$, считаем все элементы различными. Обозначим C - слитый из A и B массив. Заметим, что если $b_j : \; b_j < a_i$ - ближайший слева, то a_i соответствует $k = i + j$ номеру в C ($k-$порядковой статистике A и B). Это выполняется, потому что если слить массивы в один, то в промежутках между i элементами массива A находятся j элементов массива B. А если нет таких $b_j : b_j < a_i$, то a_i стоит на i месте в C.

Возьмем $i = n//2$ и найдем с помощью бин поиска ближайший слева $b_j : \; b_j < a_i$. Соответственно, если $k = i + j$, то мы нашли $k-$й элемент в C ($k-$порядковую статистику). Если $k < i + j$, то нуждый нам элемент находится левее в C; если $k > i + j$, то правее. Если выполняется первый вариант, тогда сузим отрезок индексов $i \in [1, n//2 - 1]$, если второй - $i \in [n//2 + 1, n]$. Затем будем аналогично искать на полученном промежутке - берем его середину, ищем ближайший слева $b_j : \; b_j < a_i$, сравниваем $i + j$ и k, сужаем рассматриваемый промежуток i и тд. Если нам не повезет, мы дойдем до одного значения a_i. Здесь есть несколько крайних вариантов:

i. $k = i + j$, все окей

ii. $k \neq i + j$, обсудим ниже

iii. если значения $b_j : \; b_j < a_i$ просто нет. Это значит, что в начале C стоят одни элементы A и нужный нам элемент - a_k

Обсудим второй вариант. Если так произошло, то это значит, что $k-$порядковая статистика находится в элементе массива B. в этом случае можно поменять ролями массивы A и B и аналогичной процедурой деления интервала $j \in [1, n]$ и нахождением ближайшего слева к b_j элемента a_i пройти по массиву B.

Корректность алгоритма объясняется тем, что на каждом шаге мы фактически сужаем область поиска нужного индекса i (если не повезет, то во втором проходе j) в два раза, что обеспечивает сходимость к нужной паре индексов.

Время работы $T = O(\log^2(n))$, потому что делаем $O(\log(n))$ шагов (деление отрезка пополам) и на каждом шаге алгоритма мы делаем бин поиск за $O(\log(n))$.

Что делать если есть повторяющиеся элементы
Если есть повторяющиеся элементы, можно на каждом шаге искать ближайшие слева $b_j : b_j \leqslant a_i$. Тогда в результате прогона получим два значения

(b) Будем считать, что элементы A и B различны. Для $k-$порядковой статистики должны выполняться условия:

$$i + j = k \tag{1}$$

$$b_j < a_i < b_{j+1}, \text{ если } k-\text{статистика соответствует } a_i \tag{2}$$

$$a_i < b_j < a_{i+1}, \text{ если } k-\text{статистика соответствует } b_i \tag{3}$$

Если выполнено (1) и (2) или (3), то нашли $k-$порядковую статистику.
Хотим научиться отсекать лишние части массива

i. Пусть $a_i > b_j$, выполнено (1), но (2) не выполнено, т.е. $a_i > b_{j+1}$. Тогда если в лучшем случае $a_i < b_{j+2}$, порядковая статистика a_i равна

$$i + j + 1 > i + j = k,$$

а значит для больших i порядковая статистика будет еще больше, поскольку $a_{i+1} > a_i > b_{j+1}$, а значит порядковая статистика $i + j + 2 > k$. Поэтому надо искать $k-$порядковую статистику при $i \in [1, i-1]$.
Аналогично для элементов массива B: в лучшем случае:

$$a_{i-1} < b_j < a_i$$

и тогда номер порядковой статистики для $b_j : i+j-1 < k$. Т.е. надо искать $k-$порядковую статистику при $j \in [j+1, n]$

ii. Пусть $a_i < b_j$, выполнена связь (1), но (3) не выполняется. Тогда $b_j > a_{i+1}$ и аналогично в лучшем случае

$$a_{i+1} < b_j < a_{i+2}$$

и номер порядковой статистики для $b_j : i + j + 1 > k$, т.е. надо смотреть $j \in [1, j-1]$.
Аналогично в лучшем случае

$$b_{i-1} < a_j < b_i$$

и номер порядковой статистики для $a_i : \ i + j - 1 < k$, т.е. надо смотреть $i \in [i+1, n]$

В результате имеем следующий алгоритм действий:

i. Берем $i = n//2$, их связи 1 находим j
ii. Проверяем условия (2) и (3). Если одно из них выполнено - нашли $k-$порядковую статистику, а если нет то:
iii. в зависимости от соотношения $a_i > b_j$ или $a_i < b_j$, сужаем область рассмотрения индексов как в описано в соответсвующем пункте

Время работы алгоритма $O(\log(n))$, поскольку на каждом шаге уменьшаем области рассматриваемых индексов в двое. Кроме того не надо искать подходящий элемент бинарным поиском, сейчас это делается за $O(1)$.