Студент: Маргарита Чудова
Группа: 2
Дата: 27 апреля 2022 г.

---

1. Положим все координаты стойл x_i в массив x и отсортируем его за $\mathcal{O}(m \log m)$. Пусть d — минимальное расстояние между коровами. Понятно, что если d очень маленькое и коров не больше, чем стойл, то все коровы поместятся, но, увеличивая d, мы дойдем до момента, когда разместить коров будет невозможно. С помощью процедуры, описанной ниже, будем проверять, **можно ли разместить коров при при d от 0 до** $x_{max} - x_0 + 1$, и бинарным поиском найдем в нем максимальное d, при котором задача решается.
   Как проверить, что коровы помещаются? Поставим первую корову в самое левое стойло, если расстояние между ним и следующим стойлом меньше d, то вторую корову ставим в следующее стойло, если нет - пропускаем стойло, повторяем до тех пор, пока стойла не кончатся.
   Псевдокод:

```
    counter = 1
    last = x[0]
    for item in x:
        if item - last >= d
            counter +=1
            last = item
        return counter >= k
```

   Такая процедура займет $\mathcal{O}(m)$ времени для конкретного d. Благодаря бинарному поиску можем уменьшить количество проверяемых d до $\mathcal{O}(\log x_{max})$. Итоговое время работы — $\mathcal{O}(m(\log m + \log x_{max}))$

2. (a) Посчитаем все возможные суммы $a_i + b_j$ простым перебором - это займет у нас $\mathcal{O}(n^2)$ (доппамяти здесь тоже $\mathcal{O}(n^2)$). Отсортируем то, что получилось, с помощью MergeSort – это займет $\mathcal{O}(n^2 \log(n^2)) = \mathcal{O}(n^2 \log(n))$

   (b) Отсортируем массивы a и b – это займет $\mathcal{O}(n \log n)$. Назовем массив дополнительной памяти массивом c (займет $\mathcal{O}(n)$) и массив итераторов массивом g (изначально состоит из n нулей, тоже займет $\mathcal{O}(n)$). Сначала заполним массив дополнительной памяти следующим образом: $c[i] = a[1] + b[i],\ i = 1 \ldots n$ (здесь имеется в виду, что массивы a, b уже отсортированы), и найдем в нем минимальный элемент с помощью обычного поиска минимума за $\mathcal{O}(n)$. Пусть это $c[j]$, выведем его, и на его место поставим $a_2 + b_j$, а к j-ому элементу массива итераторов прибавим единицу. Дальше делаем все то же самое - поиск минимума, вывод минимума, ставим на его место (пусть был минимум $a[i] + b[m]$) $a[i+1] + b[m]$, увеличиваем итератор, и так пока элементы не кончатся. Элементов $\mathcal{O}(n^2)$, значит результат займет $\mathcal{O}(n^3)$

   Почему это работает и мы получаем минимальный возможный элемент? Пусть на каком-то шаге мы выкинули элемент $c[j] = a[l] + b[j]$. Представим, что у нас есть массивы $d_i[k] = a[k]+b[i],\ k = 1 \ldots n$ – каждый такой массив можне сопоставить клетке массива c, и когда мы выкидываем элемент в c, мы как бы переходим к следующему элементу массива d_j. Нужно во-первых понять, почему элемент $d_j[k+1]$ будет меньше, чем все следующие элементы этого же массива - это очевидно по построению ($a[k+1] < a[k+2]$, $b[j]$ здесь выступает в роли константы). Во - вторых, нужно понять, почему имеет смысл взять $d_j[k]$, а не какой-то элемент из оставшихся массивов, например некий $d_l[z]$,. Ответ на этот вопрос такой: в массиве c есть элементы d_l с номером меньшим, чем z, а значит меньшие по построению. Имеет смысл сначала обработать их.

(c) Используем мин-кучу вместо массива дополнительной памяти и немного поменяем структуру данных — в куче в отличие от массива будут храниться не просто суммы $a[i]+b[j]$, а пары $(a[i]+b[j], j)$, j — это по сути указатель на массив d_j из предыдущего пункта (который мы держим в голове, но в памяти не храним), и все они умещаются в память $\mathcal{O}(n)$. В остальном все ровно то же самое, что в предыдущем пункте, используем массив итераторов для хранения номеров a, разница только в том, что раньше на d_j указывал номер клетки массива, а теперь отдельный указатель. Минимум будем узнавать за $\mathcal{O}(1)$, добавлять элементы и удалять минимум за $\mathcal{O}(\log n)$, элементов $\mathcal{O}(n^2)$. Итого результат $\mathcal{O}(n^2 \log n)$.

3. (a) Назовем первый массив a, второй — b. Возьмем элемент $a[i]$, $i = \frac{n}{2}$ и бинарным поиском найдем в массиве b элемент $b[j]$ — наибольший из элементов меньших или равных, чем $a[i]$, т.е последний из тех элементов, которые должны были бы стоять левее $a[i]$, если бы массивы были слиты и $b[m]$ — наименьший из элементов, больших или равных, чем $a[i]$.
Учтем случай одинаковых элементов в массивах: если $m \neq j$, $b[m] = b[j]$ и при этом $m \leq k - i \leq j$, то $b[j]$ — k-я статистика. Если это не выполняется, то сбивающих с толку одинаковых элементов нет, порядковая статистика элемента $a[i]$ равна $i + j$ (так как в объединении массивов до этого элемента стоит i элементов из a и j элементов из b).
Если $i + j = k$, то $a[i]$ является ответом. Если $i + j > k$, то повторим ту же процедуру, взяв вместо a массив $a[1 : i-1]$. Если $i + j < k$, то все тоже будет аналогично предыдущему шагу, но возьмем массив $a[i + 1 : n]$. Продолжаем в том же духе до тех пор, пока не попадем в k. Если так и не попадем, то, по всей видимости k-ая порядковая статистика лежит в массиве b. Проведем для него ту же процедуру.
Бинпоиск в каждом из массивов занимает $\mathcal{O}(\log n)$ и делается $\mathcal{O}(\log n)$ раз, так как массив на каждом шаге уменьшается вдвое, следовательно итоговая сложность $\mathcal{O}(\log^2 n)$.

(b) Использем допущение, о котором писали в чате, и будем считать, что все элементы в a и b различны. Пусть i — указатель на наибольший элемент, меньший либо равный k-ой порядковой статистике, пробегает по массиву a, j — по массиву b. Для k-ой порядковой статистики должны быть выполнены следующие условия:

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

$$b[j] < a[i] < b[j + 1] \text{ , если } k\text{-ая статистика равна } a[i] \tag{2}$$

$$a[i] < b[j] < a[i + 1] \text{ , если } k\text{-ая статистика равна } b[j] \tag{3}$$

0. Если выполнено (1) и одно из (2), (3), то нужную статистику мы нашли.

Будем подбирать i, j так, чтобы (1) выполнялось. Как и в пункте (b), хотим каким-то образом отсекать часть массива, в котором будем искать k, для этого рассмотрим разные соотношения между элементами и индексами. Попробуем индексами i, j разделять массивы на две части, одну которых проверять бессмысленно, будем отбрасывать ее на каждом шаге.

1. Сначала предположим, что $a[i] > b[j]$, (1) верно, но при этом (2), (3) не выполняется, то есть $b[j + 1] < a[i]$. Увидим, что для всех $\tilde{i} > i$ нужные условия тем более не будут не выполнены. В лучшем случае $a[i] < b[j + 2]$, тогда порядковая статистика этого элемента $i + j + 1 > i + j = k$. При больших i порядковая статистика будет только больше, так как $a[i+1] > a[i] \Rightarrow a[i+1] > b[j+1]$. Значит при таких условиях дальше можно искать статистику в массиве a только в части $a[1, i - 1]$.
Аналогичные рассуждения можно провести и для массива b: в лучшем случае для него будет выполняться

$$a[i - 1] < b[j] < a[i],\ i - 1 + j < i + j = k$$

То есть все элементы меньшие или равные j можно выкинуть и рассматривать $b[j + 1 : n]$.

2. Предположим, что $a[i] < b[j]$ верно, но при этом (2), (3) не выполняется, то есть $b[j] > a[i+1]$. Тогда, аналогично 1, в лучшем случае

$$a[i+1] < b[j] < a[i+2],\ i+1+j > i+j = k$$

то есть все большие j тем более не подходят. Можем рассматривать $b[1:j-1]$. Аналогично для массива a:

$$b[j-1] < a[i] < b[j],\ i+j-1 < i+j = k$$

то есть все меньшие i тем более не подходят. Можем рассматривать $a[i+1:n]$.

С учетом всего вышесказанного алгоритм выглядит так:

i. Берем $i = n//2$, по нему находим j из формулы (1).

ii. Проверяем условия (1), (2). Если выполнены - нашли порядковую статистику.

iii. Если нет, то:

  A. если $a[i] > b[j]$, то обрезаем a, b в соответствии с пунктом 1, подаем обратно на вход алгоритма

  B. если $a[i] < b[j]$, то обрезаем a, b в соответствии с пунктом 2, подаем обратно на вход алгоритма

  Сложность алгоритма $\mathcal{O}(\log n)$, так как на каждом шаге уменьшаем массивы вдвое, при этом в отличие от (b) не надо для второго массива искать соответствующий элемент бинарным поиском — благодаря (1) находим его за $\mathcal{O}(1)$, сравнения делаем за столько же.