Задача A. Keep Talking and Nobody Explodes

Автор задачи и разработчик: Даниил Орешников

Идея задачи следующая: запросы первого типа почти никогда не выгодно делать. Сначала поймем, что позволяет получить запрос первого типа. Если мы узнаем, правда ли $t \in [\sqrt{p}, p]$, то это то же самое, что выяснить, правда ли, что $t \leqslant p \leqslant t^2$. То есть мы можем проверить принадлежность p некоторому отрезку целых чисел.

Если текущие возможные границы для p — это $[l, r]$, а мы можем проверить принадлежность p некоторому $[t, t^2]$, то мы выясним, что либо p принадлежит $[l, r] \cap [t, t^2]$, либо он принадлежит $[l, r] \setminus [t, t^2]$. Размер одной из этих двух половин будет не меньше $\frac{r-l+1}{2}$, так как обе не могут быть одновременно меньше половины исходного отрезка. Таким образом, каждый запрос позволяет нам «уточнить» значение p не более, чем в 2 раза. Для этого понадобится $\log_2(p_{\max}) \approx 40$ запросов.

Теперь посмотрим на запросы второго типа. Заметим, что мы можем делать запросы второго типа t, только если мы гарантированно знаем, что $t \leqslant p$, иначе есть ненулевая вероятность попасть в число, которое больше p. Давайте оценим, какую информацию мы получаем из запросов. Пусть из запроса второго типа для числа t мы получили число x. Тогда мы знаем, что

$$\frac{p}{t} \leqslant x \leqslant \frac{p^2}{t^2}.$$

Тогда если перенести в обоих неравенствах все, кроме p, в другую сторону, мы получим $p \leqslant tx$ и $p \geqslant t\sqrt{x}$. То есть

$$t\sqrt{x} \leqslant p \leqslant tx.$$

И можно из нашего текущего приближения p, равного t, получить приближение $p \geqslant t\sqrt{x}$, то есть более точное, и следующий запрос уже можно делать с $t_{\text{new}} = \texttt{ceil}(t\sqrt{x})$.

Поскольку x равновероятно выбирается из $\left[\frac{p}{t}, \frac{p^2}{t^2}\right]$, то матожидание получаемого значения будет равно $M = \frac{1}{2}\left(\frac{p}{t} + \frac{p^2}{t^2}\right) > \frac{p^2}{2t^2}$. Тогда с вероятностью хотя бы $\frac{1}{2}$ мы получаем новую оценку на p, равную $t\sqrt{M} > \frac{p}{\sqrt{2}}$.

После этого $\frac{p}{t} \approx \sqrt{2} < 2$. Обозначим $\frac{p}{t}$ за $1 + \varepsilon$. Воспользуемся приближением, что $(1+\varepsilon)^2 = 1 + 2\varepsilon + \varepsilon^2 \approx 1 + 2\varepsilon$ при $\varepsilon \ll 1$. Тогда $M \approx 1 + \frac{3}{2}\varepsilon$. Мы переходим от этого к оценке $p \geqslant t\sqrt{M} \approx \frac{p}{1+\varepsilon} \cdot (1 + \frac{3}{4}\varepsilon) \approx \frac{p}{1+\frac{1}{4}\varepsilon}$. То есть вот это отношение $\frac{p}{t} - 1$, которое мы обозначили за ε, в среднем с каждым новым запросом уменьшается в 4 раза.

Если мы хотим добиться точности x порядка $\frac{1}{p_{\max}}$, то есть 10^{-12}, то нам для этого достаточно $\log_4(10^{12}) \approx 20$. На самом деле, учитывая тот факт, что мы привели достаточно грубые оценки, хватит меньшего числа запросов — это можно спокойно проверить симуляцией на своем компьютере.

Таким образом, полное решение выглядит так:

```
t = 1
do {
    x = EXPERIMENT(t)
    t = ceil(t * sqrt(x))
} while (x > 1)
```

Задача B. Кадровые перестановки

Автор задачи: Александр Гордеев, разработчик: Егор Юлин

Перефразируем задачу следующим образом: есть массив, состоящий из чисел a_i. Мы можем делать следующую операцию несколько раз:

- прибавить 1 к a_i и вычесть 1 из a_{i-1} для $2 \leqslant i \leqslant n$;
- прибавить 1 к a_{i-1} и вычесть 1 из a_i для $2 \leqslant i \leqslant n$.

Тогда нам нужно решить следующую задачу: за минимальное число действие сделать все элементы массива, кроме одного, равные 0. Или же, иными словами, перенести все числа в один элемент

массива Обозначим за $\texttt{pref}_i$ сумму a_j для $1 \leqslant j < i$, иными словами — префиксные суммы. Посмотрим теперь как выражается ответ для фиксированного i. Для фиксированного i ответ будет равен

$$\sum_{j=1}^{n} |i-j| \cdot a_j,$$

что можно переписать как

$$\sum_{j=1}^{i-1} (i-j) \cdot a_j + \sum_{j=i+1}^{n} (j-i) \cdot a_j.$$

Найдем ответ для $i = 1$. Для этого просто посчитаем эту сумму за время $\mathcal{O}(n)$. Теперь заметим, что при переходе от i к $i+1$

- «левая сумма» увеличивается на $a_1 + a_2 + \ldots + a_i$, так как все коэффициенты при них стали на 1 больше;
- «правая сумма» уменьшается на $a_{i+1} + a_{i+2} + \ldots + a_n$, так как все коэффициенты при них уменьшились на 1.

Обе эти суммы можно стандартным образом выразить через префиксные суммы: $a_l + \ldots + a_{r-1} = \texttt{pref}_r - \texttt{pref}_l$. То есть для каждого следующего i ответ можно пересчитать за $\mathcal{O}(1)$, и суммарное время решения равно $\mathcal{O}(n)$.

Задача C. Барби в реальном мире

Автор задачи и разработчик: Даниил Шевнин, модификация: Даниил Орешников

Задача имеет достаточно простой критерий того, будет ответ «`YES`» или «`NO`». Во-первых, если $\sum\limits_{i=1}^{n} a_i$ не делится на m, то, очевидно, раздать куклы поровну не получится, поэтому ответ будет «`NO`». На самом деле этот критерий также является достаточным: если количество делится, то всегда можно раздать поровну. Более того, это задача-шутка: расстановка детей в таком случае не имеет значения, можно показать, что в любом случае всем детям достанется поровну кукол.

Давайте докажем этот факт. Посмотрим на первый момент, когда какому-то ребенку не хватает куклы в его ряду. Пусть у i-го ряда сейчас стоит b_i детей, и в нем осталось c_i кукол. Если изначально количество кукол в сумме делилось на m, и каждый ребенок забирал по одной, то их количество все еще делится на m. Если их осталось 0, то все уже получили поровну, и утверждение доказано. А иначе их хотя бы m, а значит $\sum\limits_{i=1}^{n} b_i = m$ и $\sum\limits_{i=1}^{n} c_i \geqslant m$, или же $\sum\limits_{i=1}^{n} c_i \geqslant \sum\limits_{i=1}^{n} b_i$.

Если суммарное число кукол не меньше, чем суммарное число детей, но в каком-то ряду детей больше, то найдется и ряд, в котором кукол строго больше. Значит ребенку, которому пока куклы не хватает, найдется куда переместиться, чтобы получить еще одну.

И так далее — пока кукол не 0, всегда можно перемещениями добиться того, чтобы каждому ребенку хватило еще по одной. Поэтому ответ — «`YES`» тогда и только тогда, когда сумма a_i делится на m.

Задача D. Прелестная рассадка

Автор задачи: Даниил Орешников, разработчик: Константин Бац

Будем решать задачу жадным алгоритмом. Давайте последовательно для каждого стула с номерами от 1 до n искать наиболее подходящего гостя. Идея такая: на первый стул можно посадить любого гостя с $l_i = 1$, и, очевидно, выгодно выбрать из них того, у которого r_i минимально, тогда у остальных будет больше возможных вариантов.

Для этого будем поддерживать структуру данных, которая позволяет добавлять еще необработанных гостей и доставать гостя с минимальным r_i. В качестве такой структуры данных можно взять кучу.

Отсортируем гостей по возрастанию l_i. Выбирая гостя, которого лучше всего посадить на место k, добавим в кучу всех гостей с $l_i \leqslant k$. Это не обязательно делать заново каждый раз с нуля: при

переходе к следующему k достаточно добавить в уже имеющуюся кучу всех гостей с таким l_i. Далее возьмем из кучи гостя с минимальным r_i.

- Если $r_i \geqslant k$, то посадим этого гостя на место k.
- Заметим, что если $r_i < k$, то этого гостя некуда посадить. Действительно, в данный момент на него место уже точно не найдется, а если посадить его на какое-то из предыдущих мест, то гостя, который сидит на нем, некуда пересаживать.

Так как на каждом шаге мы выбираем гостя с наименьшим r_i, то можно показать по индукции, что если какая-то прелестная рассадка существует, то такой алгоритм позволит получить правильную рассадку.

В итоге за все время мы добавим n гостей в кучу, а затем каждого один раз в какой-то момент удалим. Таким образом, время работы решения — $\mathcal{O}(n \log n)$.

Задача E. Цепная реакция

Автор задачи: Даниил Орешников, разработчик: Константин Бац

Граф возможных взаимодействий атомов представляет из себя дерево. Давайте обойдем это дерево при помощи поиска в глубину (`dfs`). При этом, для каждого атома будем поддерживать максимальный заряд, который он может получить. Для этого поймем, что максимальный заряд в определенном атоме можно представить как сумму весов ребер на пути до него, при чем на каждом ребре (u, v) можно поставить вес либо $a_v - a_u$, либо $b_v - b_u$.

Начнем обход из атома с номером s, его максимально возможный заряд — 1. При переходе к очередному атому v от атома u будем выбирать такой способ передачи заряда, который будет максимально увеличивать заряд v. Иными словами, будем искать заряд атома, к которому переходим, по формуле $f_v = f_u + \max(a_v - a_u, b_v - b_u)$.

Поскольку дерево не содержит циклов и петель, ты мы никогда не будем пересчитывать f_v больше одного раза: каждая вершина будет посещена ровно один раз, и ей будет присвоено значение максимального возможного пути из s до нее. Можно доказать через индукцию по дереву, что таким образом мы для каждого атома правильно посчитаем максимально возможную силу заряда.

Временная сложность такого решения равна сложности поиска в глубину, то есть $\mathcal{O}(n + m)$.

Задача F. Самый милый дом

Автор задачи и разработчик: Даниил Голов

Заметим, что так как все комнаты одинаковы по длине, то длину самого длинного возможного коридора можно измерить в количестве комнат, идущих подряд, а затем домножить на b.

Высота каждой комнаты a ни на что не влияет, какова бы она ни была: если ограничения высоты на текущую комнату (l_i, h_i) как интервал пересекаются с ограничениями высоты следующей комнаты (l_{i+1}, h_{i+1}), между этими комнатами можно разместить коридор высоты как минимум 1 на высоте пересечения. Чтобы коридор можно было разместить между более чем двумя комнатами, необходимо, чтобы все интервалы ограничений высот этих комнат попарно пересекались друг с другом (тогда найдется полоса по всем комнатам высоты хотя бы 1).

Таким образом, наша задача — найти как можно более длинную последовательность подряд идущих интервалов, где все друг с другом попарно пересекаются. Эту задачу можно решить, например, методом двух указателей с помощью структуры, помогающей поддерживать минимум на скользящем окне. Действительно, пересечение $[l_i, h_i] \cap \ldots \cap [l_j, h_j] = [\max\limits_{t=i}^{j} l_t, \min\limits_{t=i}^{j} h_t]$.

Можно взять очередь с минимумом для всех h_i и очередь с максимумом для всех l_i. Для каждого возможного начала отрезка i будем искать максимальный конец отрезка $\texttt{to}(i)$, для которого пересечение всех ограничений от i до $\texttt{to}(i)$ не пустое. Для этого просто будем добавлять следующие за i пары (l_j, h_j) в очередь, пока $\max l_j < \min h_j$. Метод двух указателей заключается в том, что $\texttt{to}(i+1) \geqslant \texttt{to}(i)$ — действительно, если пересечение было не пусто, и мы удалили первое ограничение, пересечение не уменьшится. Таким образом, при переходе от i к $i+1$ достаточно начать перебор с того состояния очереди, которое осталось с прошлой итерации.

Суммарное время работы такого решения — $\mathcal{O}(n)$. Можно было вместо очереди с минимумом воспользоваться структурой данных `set` или `priority_queue` (дерево поиска и куча), и асимптотика времени работы была бы $\mathcal{O}(n \log n)$, чего тоже могло быть достаточно для получения вердикта `OK` при применении различных оптимизаций.

Задача G. Прогулка с Барби

Автор задачи: Мария Иерусалимова, разработчик: Константин Бац

Для начала рассмотрим простую версию задачи, в которой hw достаточно небольшое, чтобы можно было целиком поместить весь пляж в двумерный массив. Если бы можно было перемещаться только вправо и вниз, и были бы запрещены переходы влево, можно было бы решить задачу стандартным динамическим программированием, в котором $\texttt{dp}[i][j]$ пересчитывается через $\texttt{dp}[i][j-1]$ и $\texttt{dp}[i-1][j]$. Однако в данном случае придется считать динамику чуть сложнее.

Важное наблюдение заключается в том, что не запрещается посещать одну и ту же клетку несколько раз. Посмотрим на клетку (i, j) и найдем такие $l < j$ и $r > j$, что (i, l) и (i, r) содержат валун, и среди всех таких они ближайшие к (i, j). Тогда, попав в клетку (i, j), можно собрать все интересные вещи на отрезке $(i, l+1), \ldots, (i, r-1)$. Будем считать ту же динамику: $\texttt{dp}[i][j]$ — максимальная симпатия, которую можно набрать, придя в (i, j). Тогда $\texttt{dp}[i][j]$ можно обновить через $\texttt{dp}[i-1][j] + \sum\limits_{t=l+1}^{r-1} d_{i,t}$.

Однако мы пока не учли только способ попасть в $\texttt{dp}[i][j]$ из клетки на один выше. Могло быть и так, что мы спустились в другую клетку этого отрезка $(i, l+1), \ldots, (i, r-1)$, обошли его, и потом решили остановиться в (i, j). Таким образом,

$$\texttt{dp}[i][j] = \left(\max_{t=l+1}^{r-1} \texttt{dp}[i-1][t]\right) + \left(\sum_{t=l+1}^{r-1} d_{i,t}\right).$$

В обычной таблице эту величину можно было бы для каждой клетки посчитать, используя префиксные суммы по каждой строке для нахождения второго слагаемого. А первое слагаемое для всех клеток на отрезке было бы одинаковым, потому что у любой клетки на этом отрезке одни и те же l и r. Получается, что можно было бы на каждом отрезке строки i за длину отрезка посчитать это первое слагаемое за время, пропорциональное длине отрезка, один раз, и в сумме все такие вычисления тоже работали бы за $\mathcal{O}(wh)$.

Теперь попробуем из полученного решения сделать решение для версии задачи с большим h. Поскольку валунов не больше n, и в каждой строке x валунов будут приводить к не более чем $x+1$ отрезков, в сумме не более $n + h \leqslant 2 \cdot 10^5$ отрезков. Поскольку на каждом отрезке значения `dp` получаются одинаковыми,

1. сгруппируем валуны по строкам и отсортируем;
2. выделим в каждой строке отрезки между валунами в порядке их следования слева направо;
3. будем для каждого отрезка считать значение `dp` на нем.

В каждый отрезок можно попасть из некоторого количества подряд идущих отрезков предыдущей строки. И при переходе к следующему отрезку это «окно» отрезков предыдущей строки, из которых можно в него попасть, сдвигается вправо. Будем для пересчета динамики очередного отрезка текущей строки поддерживать два указателя на отрезки предыдущей строки, из которых в него можно попасть. Эти два указателя будут двигаться только вправо, поэтому обработать пересчет динамики на отрезках i-й строки получится за $\mathcal{O}$(кол-ва отрезков предыдущей строки). В сумме это не больше $n + h$.

В итоге мы получаем решение за время $\mathcal{O}(h + n \log n)$ — сортировка особых клеток в каждой строке займет $\mathcal{O}(n \log n)$, а пересчет динамики — $\mathcal{O}(n + h)$. Ответ динамики будет находиться в том отрезке, который содержит клетку (h, w).

Задача H. Конференция

Автор задачи: Даниил Голов, разработчик: Константин Бац

В этой задаче нужно было выбрать вершину в графе так, чтобы сумма кратчайших расстояний до выбранной вершины, умноженных на количество людей в соответствующей вершине, была минимальной.

Ограничения на входные данные в этой задачи позволяли воспользоваться алгоритмом Флойда для нахождения кратчайших расстояний от каждой до каждой вершины. Тогда для нахождения ответа достаточно перебрать все вершины и явно просуммировать произведения расстояний и количеств жителей в каждом городе, после чего выбрать среди таких сумм минимальную.

Асимптотика нахождения кратчайших расстояний — $\mathcal{O}(n^3)$, выбора города и вычисления ответа для него — $\mathcal{O}(n^2)$. Тогда итоговая асимптотика такого решения равна $\mathcal{O}(n^3)$, и почти любая корректная реализация должна была укладываться в ограничения по времени.

Задача I. Интерпретация

Автор задачи и разработчик: Даниил Орешников

Для начала посмотрим на команды, вид которых почти не меняется при преобразовании из одного кода в другой. Это команды чтения и записи, а также команды вычисления новых значений переменных. Для первых двух надо просто убрать открывающую и закрывающую скобки, команду написать заглавными буквами, а между командой и аргументом поставить пробел. Для выражений все еще проще — надо '=' заменить на строку «GETS».

Дальше сложнее. Надо научиться раскрывать и преобразовывать циклы так, чтобы не было вложенности. Но это можно сделать следующим образом: цикл

```
for <var> = <from>...<to> {
    <do something>
}
```

можно заменить на

```
MACRO macro:
    <do something>
    <var> GETS <var> + 1

<var> = <from>
repeats = <to> - <from> + 1
REPEAT macro repeats
```

Иными словами, любой цикл `for` можно преобразовать в цикл `while` с приращением подлежащей переменной. Ее исходное значение равно `<from>` и всего происходит `<to> - <from> + 1` итераций. Величину `<to> - <from> + 1` можно посчитать за одно или два арифметических выражения (если хотя бы одно из <from> и <to> — константа, то можно сразу ее увеличить или уменьшить на 1, иначе — в первой операции вычесть их, а во второй добавить 1).

Осталось только показать, как осуществлять само преобразование. Но заметим, что

1. по строке всегда однозначно понятно, какую операцию она задает — либо можно посмотреть на префикс до первого пробела или первой скобки, либо вообще сделать сопоставление с помощью регулярных выражений;

2. по началу цикла легко определить его конец: фигурные скобки образуют правильную скобочную последовательность;

3. можно «разбирать» выражение как дерево вложенных блоков: для этого достаточно завести рекурсивный парсер, и при встрече `for`, заходить в рекурсию, парсить его, и возвращать соответствующий разобранному циклу макро, когда встречаем закрывающую фигурную скобку;

4. можно не беспокоиться о конфликте имен, когда мы заводим новую переменную под счетчик количества итераций: гарантируется, что в исходном коде переменные не начинаются на букву 'x', так что можно «зарезервировать» все такие переменные под счетчики, и по очереди создавать новые по мере необходимости;

5. во время «разбора» цикла достаточно хранить массив строк, из которых будет состоять соответствующий ему макро; когда мы зашли во вложенный цикл, разобрали его, и вернулись, надо просто в массив строк текущего цикла добавить строки, инициализирующие переменную цикла и количество итераций, и строку, вызывающую соответствующий разобранному внутреннему циклу макро.

Весь алгоритм, таким образом, представится в виде одного рекурсивного парсера, который на выходе из рекурсии создает макро, соответствующие циклу, и передает «наружу» информацию о том, как его надо вызвать в том месте, где в исходном коде находился этот цикл. За примером решения можете обратиться в архивы жюри.

Задача J. Опасные опыты

Автор задачи: Максим Рысков, разработчик: Егор Юлин

Для решения задачи рассмотрим два случая:

- Среди чисел a_i есть хотя бы одно, равное 0. Тогда при любом значении критическая масса в данном опыте будет достигнута, значит минимальная подходящая равна 0.
- Все числа a_i не равны 0, тогда наибольшее не критическое значение в опыте i будет равно $a_i - 1$. Легко построить пример, когда сумма равна $\sum_{i=1}^{n}(a_i - 1) = S$, и ни в одном опыте масса не превышает критическую, то есть в точности равна $a_i - 1$. Разумеется, и для любой меньшей суммы тоже не факт, что хотя бы в одном опыте наберется критическая масса, значит нам подойдет только большее значение.

  Тогда возьмем сумму равную, $S + 1$, и по принципу Дирихле хотя бы в одном опыте масса превысит критическую. Значит такая сумма нам подходит, и является минимальной подходящей.

Задача K. Идеальная пара

Автор задачи и разработчик: Егор Юлин

Перефразируем задачу следующим образом: у нас есть два множества строк, и можно в каждое добавлять строки или удалять их. Будем решать задачу при помощи бора, а именно — будем хранить все строки вместе в одном боре. Перед спуском в бор, если приходит запрос для второго множества, то перевернем строку. Далее для второго множества будет рассматриваться перевернутая строка.

Про вершины бора будем хранить следующую информацию: $\texttt{pal}_1[i]$ — количество путей от вершины i до листа в боре, являющихся палиндромами, в первом множестве $\texttt{end}_1[i]$ — количество строк из первого множества, которые заканчиваются в данной позиции. Аналогично для второго массива.

Теперь заметим, что если конкатенация s_1 и s_2 является палиндромом, и, не теряя общности, $|s_1| \geqslant |s_2|$, то если развернуть $s_1 = s_2^{\text{rev}} + p$, где p — суффикс s_1 длины $|s_1| - |s_2|$, являющийся палиндромом. В обратную сторону тоже верно. А в терминах нашего бора, каждой строке первого множества, которая заканчивается в состоянии i, в пару можно поставить $\texttt{pal}_2[i]$ строк из второго множества (у них совпадают префиксы, а оставшийся суффикс является палиндромом), и наоборот. Поэтому ответ равен

$$\sum_{i=1}^{s} \texttt{end}_1[i] \cdot \texttt{pal}_2[i] + \texttt{end}_2[i] \cdot \texttt{pal}_1[i].$$

Теперь рассмотрим изменение переменных, когда нам задаются запросы на изменение первого множества:

1. Для позиции f, в которой строка из запроса заканчивается, $\texttt{end}_1[f]$ изменится на 1 в нужную сторону, и $\texttt{pal}_1[f]$ тоже, так как пустой суффикс мы считаем палиндромом. Поскольку изменился $\texttt{end}_1[f]$, то ответ увеличится или уменьшится на $\texttt{pal}_2[f]$.
2. Для позиций i, для которых продолжение строки из запроса является палиндромом: ответ изменяется на $\pm\texttt{end}_2[i]$, так как $\texttt{pal}_1[i]$ изменяется на ± 1.

3. Во всех остальных вершинах бора ничего не меняется.

При запросе ко второму множеству переменные будут меняться симметричным образом. Тем самым, нам осталось только быстро понимать, является ли суффикс, начиная с некоторой позиции, палиндромом. Это можно сделать, например, с помощью хэшей или z-функции.

Таким образом, если мы за $\mathcal{O}(|s|)$ умеем считать для каждой позиции, является ли суффикс палиндромом, где $|s|$ — длина строки s, то такое решение работает за сложность $\mathcal{O}(\text{total length})$.

Задача L. Беспорядки в Барбилэнде

Автор задачи и разработчик: Даниил Орешников

Заметим, что если из графа можно удалить ребро, не потеряв связность, суммарный вес вершин (суммарное недовольство) гарантированно неувеличится. Поэтому в итоге мы хотим построить минимальное остовное дерево, с критерием минимальности по сумме весов вершин, которые зависят от выбранных ребер.

Покажем, как свести имеющуюся у нас задачу к обычной задаче о поиске остовного дерева с минимальной суммой весов **ребер**. Для этого запишем сумму весов

$$\sum_{v=1}^{n} \sum_{(u,v)} p_v \oplus d_{u,v},$$

что можно упростить и перегруппировать по ребрам:

$$= \sum_{e \in T} d_e \oplus p_{e.v} + d_e \oplus p_{e.u}.$$

Действительно, в исходную сумму каждое ребро вошло ровно дважды: с каждым из своих концов. В обоих случаях его исходный вес изменяется применением `xor` с весами его концов. И взятие ребра (u, v) означает добавление веса $du, v \oplus p_u + d_{u,v} \oplus p_v$ в итоговую сумму.

Таким образом, давайте при чтении каждому ребру назначим вес по рассмотренной выше формуле, останется только построить минимальный остов во взвешенном графе. Это можно сделать за $\mathcal{O}(m \log m)$, чего хватает, чтобы уложиться в ограничения.