Задача A. Камни и бананчики

Автор задачи	Бабин Александр
Разработчик	Питер Лосев
Тема	Поиск в ширину

Если две клетки оказались на одном куске бумаги после того, как были выжжены клетки с камнями, то это значит, что есть простой путь соединяющий одну клетку с другой, соединяющий эти две клетки и не посещающий ни одну из клеток с камнями. Утверждение верно и в обратную сторону, так как клетки пути сами по себе образуют связное множество.

Поэтому попытаемся найти простой путь между клетками (x_1, y_1) и (x_2, y_2), который после перемещения камней не будет проходить через клетки с камнями. В задаче можно было переместить не более k камней, поэтому в исходном поле путь от клетки (x_1, y_1) до клетки (x_2, y_2) не может проходить более чем через k камней.

Но есть и дополнительное условие, если найденный путь посещает слишком много клеток, то его не получится *расчистить*, так как камни на этом пути будет попросту некуда деть. А именно, если в исходной таблице есть m свободных клеток, то и искомый путь не может проходить более чем через m клеток.

Чтобы определить существование такого пути, скомбинируем идеи динамического программирования и поиска в ширину. А именно предлагается вычислить $\mathrm{d}[t][i][j]$ — кратчайший путь от клетки (x_1, y_1) до клетки (i, j), проходящий не более чем через t камней. Чтобы вычислить значения этого массива при заданном t, достаточно написать поиск в ширину со следующим переходом:

- При переходе от тройки (t, i, j) к соседней клетке (i', j') следует полагать $t' = t$ в случае, если (i', j') не содержит камня и $t' = t + 1$ в ином случае.
- Во всех остальных деталях алгоритм не отличается от обычного поиска в ширину.

Далее следует проверить, что найденный путь достаточно короткий, и в случае успеха, восстановить сам путь, камни, которые на нем лежат, а также свободные клетки, лежащие вне этого пути. После чего легко восстанавливается ответ на задачу. Получили решение с временем работы $O(n^2k)$.

Задача B. Кодовый замок

Автор задачи	Заварин Алексей
Разработчик	Заварин Алексей
Тема	Динамическое программирование

Каждый набор отрезков можно *просканировать* слева-направо, поддерживая k — количество отрезков, в котором содержится очередной элемент i. В ходе этого процесса также можно поддерживать и количество отрезков: когда при переходе от i к $(i + 1)$ значение k увеличивается на x — это значит, что и количество отрезков увеличилось на x, так как в позиции $(i + 1)$ открылись x новых отрезков; а когда уменьшается — то суммарное количество отрезков не меняется, так как закрываются уже ранее подсчитанные отрезки.

Вычислим значения динамики $\mathrm{dp}[i][k]$ — минимальное количество отрезков, которое требуется для того, чтобы выставить правильные значения $a_1, \ldots, a_i$, при условии, что элемент i покрывается ровно k отрезками. Данный массив следует вычислять по слоям, чтобы сэкономить память.

Нетрудно показать, что ответ не превосходит $M = \sum_{i=1}^{n}(m_i - 1)$, поэтому нас интересуют только значения k на отрезке $[0, M]$. Преобразовать $\mathrm{dp}[i-1]$ в $\mathrm{dp}[i]$ можно с помощью трех линейных шагов:

1. Скопировать массив $\mathrm{dp}[i - 1]$ в массив $\mathrm{dp}[i]$.
2. В порядке $k = 1, 2, \ldots, M$ выполнить обновление $\mathrm{dp}[i][k] \leftarrow \min\{\mathrm{dp}[i][k], 1 + \mathrm{dp}[i][k - 1]\}$;
3. В порядке $k = M - 1, M - 2, \ldots, 0$ выполнить обновление $\mathrm{dp}[i][k] \leftarrow \min\{\mathrm{dp}[i][k], \mathrm{dp}[i][k + 1]\}$;

4. Профильтровать $\mathrm{dp}[i]$, а именно присвоить $\mathrm{dp}[i][k] \leftarrow \infty$ для всех $k \not\equiv a_i \pmod{m_i}$.

Второй шаг обрабатывает возможность открыть новые отрезки, третий обрабатывает закрытие отрезков, а четвертый — накладывает условие на то, что в позиции i замка будет введено нужное число. Нетрудно показать, что от первого шага можно отказаться и производить все вычисления на одном и том же массиве.

В итоге получили решение с временем работы $O(n \cdot \sum m_i)$ и занимаемой памятью $O(\sum m_i)$.

Задача C. Хорошие раскраски – 7

Автор задачи	Бабин Александр Романович
Разработчик	Бабин Александр Романович
Тема	Конструктивы и остовные деревья

Рассмотрим двудольный граф $G = (R, C, E)$, левой долей которого выступают строки таблицы $R = \{1, 2, ..., n\}$, правой долей столбцы $C = \{1, 2, ..., n\}$. Ребро из строки $i \in R$ в столбец $j \in C$ будем проводить в том случае, если клетка (i, j) белая.

За время $O(n^2)$ вычислим также следующие значения:

$r(i)$	количество красных клеток в строке i $(1 \leqslant i \leqslant n)$
$c(j)$	количество красных клеток в столбце j $(1 \leqslant j \leqslant n)$
d_1	количество красных клеток на главной диагонали
d_2	количество красных клеток на побочной диагонали

Пусть мы раскрасили белые клетки из множества U' в красный цвет, а остальные — в синий, при этом удовлетворив условиям задачи. Тогда клеткам U' соответствует набор ребер $U \subseteq E$ в графе G, для которого выполнены следующие условия:

- Для любой вершины-строки $i \in R$ четность числа инцидентных ребер из U с четностью $r(i)$.
- Для любой вершины-столбца $j \in R$ четность числа инцидентных ребер из U с четностью $c(j)$.
- Условие, соответствующее четности числа красных клеток на диагоналях после перекраски белых клеток, в терминах графа G сформулировать не получится. Мы разберем с ним позже.

Таким образом достаточно найти любое подходящее множество $U \subseteq E$ и искомый пример раскраски будет найден. Условия на строки и столбцы задают условия на четность вершин в графе $G_U = (R, C, U)$, откуда сразу можно получить ограничение:

Утверждение 1. Просуммируем значения $r(i)$ и $c(j)$ в произвольной компоненте связности графа G. Если полученное значение нечетно, то подобрать множество U не получится.

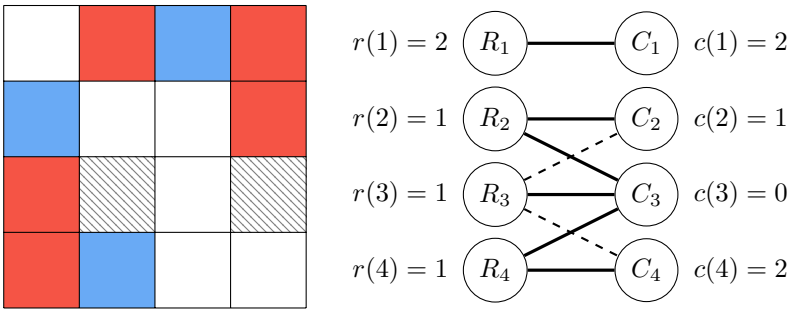
Доказательство. От противного, пусть нашлось подходящее множество U, тогда рассмотрим C — компоненту графа G, где сумма $r(i)$ и $c(j)$ нечетна. Тогда с одной стороны суммарная степень вершин из C в графе G_U четная, а сдругой стороны должна быть нечетной, так как совпадает с суммой $r(i)$ и $c(j)$). Пришли к противоречию.

Выделить компоненты связности и проверить для них это условие можно за время $O(n^2)$. Если условие не выполнилось хотя бы для одной компоненты, то следует вывести «No» и завершить обработку тестового набора.

В случае, если указанное условие выполняется для всех компонент графа, выделим произвольный *остовный лес* T в графе G, и заметим еще один факт:

Утверждение 2. Для любого множества ребер $A \subseteq E \backslash T$ найдется единственное множество ребер $B \subseteq T$ такое, что $U = A \cup B$ подходит под условие о четности степеней вершин.

Иными словами, после выделения остовного дерева все ребра разбились на два класса — *жесткие* (которые принадлежат остову) и *свободные* (не принадлежат остову). Сначала для каждого



Пример раскраски и соответствующего ей графа $G = (R, C, E)$, жесткие ребра выделены жирным, свободные — пунктиром.

свободного ребра мы можем свободно выбирать, войдет оно в множество U или нет, после чего для каждого *жесткого* ребра принадлежность к множеству U может быть однозначно восстановлена.

В любом случае придется написать код, который восстанавливает множество B по множеству A, поэтому придется разобраться, как это делать. Для каждой вершины $v \in R \cup C$ легко вычислить $s(v) \in \{0, 1\}$ — четность числа ребер из A, которые ей инцидентны при фиксированном B.

Заметим, что если выбрать произвольный путь $P \subseteq T$, а затем инвертировать ребра на этом пути в множестве A (то есть выкинуть те ребра, что были, а те, которых не было, добавить), то степени изменятся ровно у двух вершин — концов пути P. Поэтому можно, например, выделить в каждой компоненте корень, подвесить за него остовное дерево в этой компоненте, а затем просто инвертировать пути. При эффективной реализации это тратит $O(n^2)$ времени.

Доказать единственность тоже несложно, достаточно рассмотреть два подходящих множества A_1 и A_2 и обратить внимание, что в графе, построенном на ребрах $A_1 \Delta A_2$ (имеется ввиду симметрическая разность), все степени четные и при этом нет циклов, откуда следует $A_1 \Delta A_2 = \varnothing$.

Таким образом, мы научились легко генерировать множества ребер U, а значит и раскраски, удовлетворяющие первым двум условиям. Осталось только разобраться с тем, как удовлетворить условия для диагоналей. Здесь есть два подхода.

Первый подход. (б/д) Выделим остов T, затем каждое свободное ребро с вероятностью $\frac{1}{2}$ включим в множество B, после чего построим множество A, а затем восстановим раскраску. Утверждается, что если решение есть, то оно будет получено с вероятностью $\geqslant \frac{1}{4}$. Повторяем эту процедуру 60 раз, и если ни разу не нашли решения, то полагаем, что его не существует. Вероятность не найти ответ при его существовании составляет $\approx 3 \cdot 10^{-8}$.

Второй подход. Выделим остов T и построим множество A_0 по пустому множеству $B = \varnothing$. Теперь, если мы добавляем или удаляем ребро $e = \{u, v\}$ в множество B, получая $B' = \{e\} \Delta B$, то меняется степень только у 2 вершин. Пусть путь P соединяет эти две вершины, тогда множество A при таком изменении B заменяется на множество $A' = A \Delta P$.

Чтобы отслеживать четность диагоналей, сопоставим каждому ребру e (и жесткому, и свободному) 2-битную маску $x(e)$: 1-й бит этой маски равен единице, если клетка лежит на главной диагонали, 2-й бит равен единице, если клетка лежит на побочной.

Тогда последнее требование к раскраске в терминах выбора множества ребер U формулируется следующим образом:

$$x(U) = x(e_1) \oplus x(e_2) \oplus \ldots \oplus x(e_k) = (d_2 \bmod 2) \cdot 2 + (d_1 \bmod 2), \quad U = \{e_1, \ldots, e_k\}$$

Здесь $\oplus$ означает операцию побитового XOR. Теперь при добавлении или удалении ребра $e' = \{u, v\}$ величина $x(U)$ меняется следующим образом:

$$x(U') = \bigoplus_{e \in A' \cup B'} x(e) = \left[\bigoplus_{e \in U} x(e)\right] \oplus \underbrace{\left[\bigoplus_{e \in P} x(e)\right]}_{= y(e')} \oplus x(e').$$

Значения $y(e)$ для всех свободных ребер e легко вычислить за время $O(n^2)$. Но тогда пришли к такой постановке: изначально есть какое-то $x(U)$, и мы можем ксорить это значение с какими-то

$y(e)$, требуется получить 0.

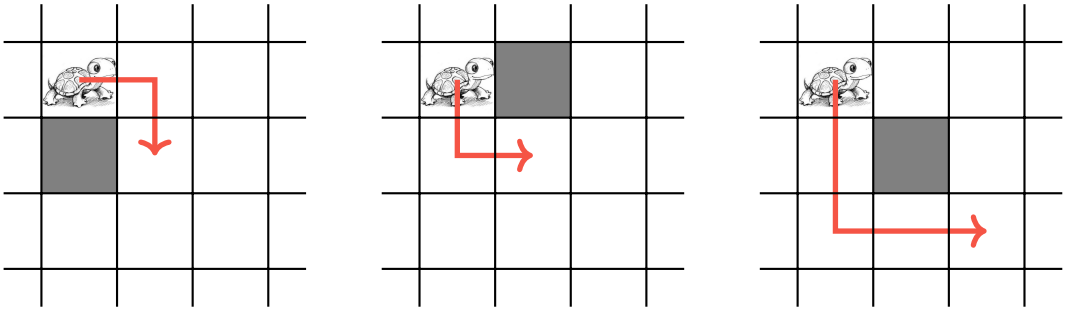
Но дело мы имеем с двухбитными масками, поэтому можно либо разбить задачу на случаи, либо написать перебор. Таким образом, подобрать нужный набор ребер из B можно за время $O(1)$.

После этого все восстанавливаются последовательно A, U, а затем и искомая раскраска.

Задача D. Как обидеть черепашку

Автор задачи	Бабин Александр
Разработчик	Бабин Александр
Тема	Конструктив

Сперва заметим, что черепашка может из клетки (x, y) всегда может попасть либо в клетку $(x + 1, y + 1)$, либо в клетку $(x + 2, y + 2)$. Всего есть 3 случая, которые изображены ниже:

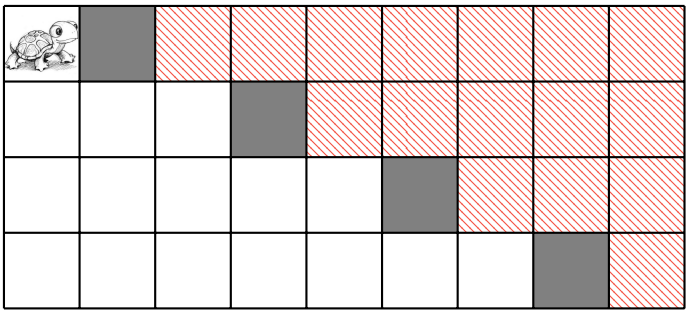


Не теряя общности положим, что $m \geqslant n$. Если $m \leqslant 2n$, то черепашка всегда может добраться из клетки $(1, 1)$ в клетку (n, m) с помощью жадного алгоритма:

- Если черепашка находится в клетке (x, y) и $y - x = m - n$, то она дальше продвигается по этой диагонали в клетку $(x + 1, y + 1)$, а если не может, то в клетку $(x + 2, y + 2)$.
- Если же $y - x < m - n$, то черепашка пытается пройти в клетку $(x, y + 1)$.
- Если у нее не получилось, то она может попасть в клетку $(x + 1, y + 2)$.

Легко видеть, что учитывая $m \leqslant 2n$, черепашка сможет добраться до клетки (n, m), так как в худшем случае она перемещается на соседнюю диагональ, спускаясь на одну клетку вниз.

С другой стороны, в случае $m > 2n$ легко предъявить пример, как можно расставить препятствия так, чтобы заблокировать путь черепашке. Для этого достаточно блокировать клетки вида $(k, 2k)$.



Полностью аналогично, рассматривается случай $n > m$.

Задача E. Бойцовский клуб

Автор задачи	Алексей Васильев
Разработчик	Алексей Васильев
Тема	Стек, префиксные максимумы

Пусть отрезок $[l, r]$ — честный и $a_l < a_r$, тогда a_r — ближайшие больший справа элемент по отношению к a_l, так как все элементы $a_{l+1}, \ldots, a_{r-1}$ меньше чем a_l. Аналогично, в случае $a_l > a_r$ элемент a_l является ближайшим большим слева элементом по отношению к a_l.

В другую сторону, пусть у элемента a_l есть ближайший справа больший и он равен a_r. Тогда до a_r нет элементов больше a_l, а значит отрезок $[l, r]$ — честный. Аналогичное рассуждение верно, в случае, если a_l — ближайшие слева больший элемент по отношению к a_r.

Таким образом, количество честных отрезков равно количеству элементов, для которых слева есть элементы больше него, плюс количество элементов, для которых справа есть элементы больше него.

Научимся поддерживать количество элементов для которых слева есть элементы больше него. Для этого будем поддерживать те элементы, которые являются префиксными максимумами. То есть это элементы, для которых слева нет больших элементов. Будем поддерживать их индексы в стеке слева направо. Когда элемент a_i стал новым максимумом, то он удаляет все префиксные максимумы правее него, оставляет прежние префиксные максимумы левее и делает себя префиксным максимумом. То есть мы легко можем поддерживать этот стек с амортизированным временем работы $O(n + q)$.

Таким образом, мы должны поддерживать два стека (префиксных максимумов и суффиксных максимумов) и ответом задачу будет $2n - cnt_{pref} - cnt_{suf}$, где cnt_{pref} — размер стека префиксных максимумов, а cnt_{suf} — размер стека суффиксных максимумов.

Задача F. Арсен и солдатики

Автор задачи	Хо Данг Зу
Разработчик	Хо Данг Зу
Тема	Эйлеровость графа

Заметим, что необходимым для условия баланса, является то, что каждый вид ружья должен быть выдан как утром, так и вечером четное количество раз. С помощью жадного алгоритма легко этого добиться или проверить, что это невозможно за линейное время.

Далее, рассмотрим двудольный граф в левой доли которого будут виды ружей, которые выданы утром, а в правой доли которого будут ружья, которые выданы вечером. Ребрами будут выступать солдаты: так, солдат получивший утром ружье u, а вечером ружье v будет представлен ребром, которое соединяет вершину u левой доли и вершину v правой доли.

Заметим, что каждая из компонент построенного графа является Эйлеровой, так как все степени вершин четны. Поэтому в каждой из компонент можно найти Эйлеров цикл и ориентировать все ребра в соответствии с этим циклом. Таким образом из каждой вершины будет исходить столько же ребер, сколько в нее входит.

Этим фактом можно воспользоваться и для последнего этапа решения задачи — распределения мыла. Выдадим мыло тем солдатам, соответствующие ребра, которых направлены из правой доли в левую, а остальным мыла выдавать не будем. Легко видеть, что условие задачи соблюдается.

Представленное решение можно легко реализовать за время $O(n)$. Долгое время работы на практике обусловлено тем, что для данный алгоритм делает много случайных обращений к памяти, из-за чего слишком часто не попадает в кеш.

Задача G. Геометрия!

Автор задачи	Понкратов Александр
Разработчик	Степанов Антон
Тема	Сортировка по углу, дерево отрезков

Если взять два многоугольника A и B, упорядочить их ребра по углу и выполнить обычную процедуру слияния двух отсортированных последовательностей, мы получим многоугольник $F(A, B)$: в нём последовательно идут все ребра из A и B с теми же длинами и ориентациями, что и исходные.

Вершина $F(A, B)$ учитывается функцией $G(A, B)$, если два инцидентных ей ребра пришли из разных многоугольников. Эквивалентно, если смотреть на круговой порядок всех ребер A и B после слияния, то $G(A, B)$ — это число переходов между цветами «A» и «B».

Пусть ребра A в этом порядке разбиваются на s_A непрерывных отрезков, а ребра B — на s_B отрезков. Тогда каждое внутреннее соседство двух ребер из A порождает вершину, которая **не** попадает в G. Таких соседств $k_A - s_A$, аналогично для B. Всего вершин в $F(A, B)$ равно $k_A + k_B$, поэтому

$$G(A, B) = (k_A + k_B) - (k_A - s_A) - (k_B - s_B) = s_A + s_B.$$

То есть для каждой пары многоугольников важно лишь, сколько раз ребра одного из них *разбиваются* ребрами другого.

Теперь рассмотрим сразу все многоугольники. Соберём все их ребра в один массив, отсортируем по полярному углу, записав для каждого ребра, какому многоугольнику оно принадлежит.

Теперь для произвольного многоугольника A возьмём его позиции в отсортированном массиве и посмотрим на соседние пары (p_j, p_{j+1}) (где $p_k = p_0$ для замыкания окружности). Если между этими двумя позициями встречаются ребра какого-то многоугольника B, значит именно B разбивает последовательность A и вклад в ответ от пары (A, B) увеличивается на 1. Повторив процедуру для всех многоугольников, мы сложим s_A по всем A и автоматически получим сумму $G(A, B)$ по всем парам.

Осталось эффективно посчитать число **разных** многоугольников, которые встретились между каждой парой p_j и p_{j+1}. Это классическая задача на запросы вида «количество различных на отрезке», которая может быть решена «offline» с помощью дерева отрезков и сортировки запросов по правой границе. Суммарное время работы составляет $O(m \log m)$, где под m подразумевается суммарное количество вершин во всех многоугольниках.

Задача H. Трамвайная система

Автор задачи	Александр Бабин
Разработчик	Александр Бабин
Тема	Палиндромы, таблицы

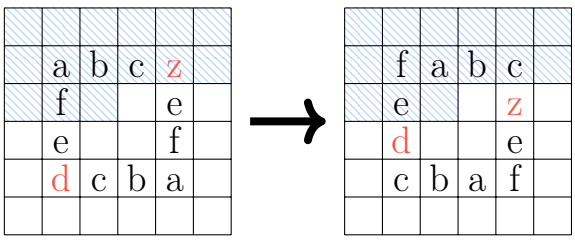
Таблица $n \times n$ по условию разделена на трамвайные пути, давайте для простоты называть их *квадратами*. Ясно, что любая конечная конфигурация трамваев может быть получена, если последовательно *вращать* квадраты, т.е. применять к ним циклические сдвиги по или против часовой стрелки.

Обратим внимание на *центральную симметрию* в представленной. Так клетка (x, y) будет центрально симметрична клетке $(n - 1 - x, n - 1 - y)$. Легко видеть, что после выписывания таблицы *в ряд* символы в этих клетках попадут на симметричные позиции.

Значением *мистики* — это как раз позиция первого символа i, после выписывания в ряд, которые не будет совпадать с симметричным ему символом, т.е. символом на позиции $n^2 - i - 1$.

Заметим, что при любом вращении квадратов центрально симметричные символы в таблице останутся центрально симметричными по отношению друг к другу, а поэтому имеет смысл разбить их на пары по этому принципу, при чем пары разделить на *хорошие* и *плохие*, то есть где символы в паре совпали и не совпали, соответственно.

Теперь запишем в клетки таблицы, которые принадлежат хорошим парам, значение 1, а в остальные клетки — значение 0. Задача свелась к тому, чтобы максимизировать позицию первого 0 после выписывания таблицы в ряд.



Картинка для понимания происходящего. Нашли наибольший отрезок `efabc` и пододвинули его так, чтобы плохая пара символов `z`–`d` встретились как можно позже после выписывания всего в ряд.

Заметим, что если отметить первые k элементов таблицы, то они высекут на каждом из квадратов непрерывные участки. Но тогда можно для каждого квадрата независимо вычислить наибольшее количество подряд идущих 1 (с учетом того, что квадрат образует цикл), а затем циклически сдвинуть этот квадрат так, чтобы первый 0 на этом квадрате встречался как можно позже в таблице. В итоге получили решение с временем работы $O(n^2)$.

Важный комментарий. Разумеется, описанное решение предполагает некоторые технические выкладки связанные с тем, чтобы правильно обработать каждый из квадратов, но можно существенно упростить реализацию, если не подбирать циклический сдвиг явно, а вместо этого просто пройтись по элементам таблицы и остановиться в тот момент, когда квадрат очередной клетки встретилась большее число раз, чем наибольшее количество подряд идущих 1 на этом квадрате.

Задача I. Оценки Рика

Автор задачи	Михаил Прохоров
Разработчик	Антон Ныйкин
Тема	Сортировка

Заменим все вхождения 0 в массиве $b_1, \ldots, b_n$ на 6. После такого преобразования необходимо перемешать элементы массива b таким образом, чтобы выполнялось $a_i \leqslant b_i$ для каждого i от 1 до n, а затем обратно заменить все вхождения 6 в массиве b на 0.

Если a, массив оценок Морти, упорядочен в порядке возрастания, то выгодно и массив b, оценки Рика после замены 0 на 6, упорядочить в порядке возрастания. Но изменять порядок элементов a нельзя, поэтому следует отсортировать массив a, но при этом запомнить для каждого элемента на какой позиции он стоял в исходном массиве. Решение за $O(n \log n)$ выглядит следующим образом:

```
t = int(input())
for _ in range(t):
    n = int(input())
    a = list(map(int, input().split()))
    b = list(map(int, input().split()))

    b = sorted([6 if x == 0 else x for x in b])
    u = sorted([(a[i], i) for i in range(n)])

    ans = True
    res = [0] * n
    for i in range(n):
        x, p = u[i]
        if x <= b[i]:
            res[p] = 0 if b[i] == 6 else b[i]
        else:
            ans = False
            break
    if ans:
        print(" ".join(map(str, res)))
    else:
        print(-1)
```

Также есть и более эффективное решение, которое работает за время $O(n)$ и использует тот факт, что участвующие в сортировках числа принадлежат отрезку $[0, 6]$. Уже это позволяет использовать цифровую сортировку или эффективно реализовать другой жадный подход: «Каждой оценке Морти сопоставить наименьшую, ранее неиспользованную оценку Рика».

Задача J. Упражнение для Дани

Автор задачи	Андрей Пархаев
Разработчик	Андрей Пархаев
Тема	Теория чисел, бор

Для краткости, будем пользоваться записью $f(x, y) = M\left(\frac{xy}{\gcd(x,y)^2}\right)$

Будем рассматривать каждое a_i как упорядоченную последовательность простых $p_1 \leqslant p_2 \leqslant \ldots \leqslant p_k$ из разложения a_i на простые множители (то есть $a_i = p_1 \cdot p_2 \cdot ... \cdot p_k$). Теперь давайте поймем, что мы пытаемся посчитать в терминах этих последовательностей.

Говоря простым языком, $f(a_i, a_j)$ — это наименьший простой делитель, который входит в разложения a_i и a_j **разное** количество раз. С другой стороны, если посмотреть на соответствующие последовательности простых для a_i и a_j:

$$a_i = q_1 \cdot q_2 \cdot \ldots \cdot q_m,\ q_1 \leqslant q_2 \leqslant \ldots \leqslant q_m$$
$$a_j = p_1 \cdot p_2 \cdot \ldots \cdot p_k,\ p_1 \leqslant p_2 \leqslant \ldots \leqslant p_m$$

Если t — длина наибольшего общего префикса у последовательностей p и q, то $f(a_i, a_j) = \min\{p_{t+1}, q_{t+1}\}$. В случае, если t совпадает с длиной одной из последовательностей, то в качестве минимума берется $(t+1)$-й элемент другой последовательности. Если t совпадает с длинами обеих последовательностей, то $f(a_i, a_j) = 0$.

Например, если $a_i = 12$, а $a_j = 20$, то им соответствуют последовательности $2, 2, 3$ и $2, 2, 5$. Наибольший общий префикс таких последовательностей равен 2, а значением функции от a_i и a_j является $\min\{3, 5\} = 3$.

Чтобы воспользоваться наблюдением выше, заведем структуру данных, которая умеет обрабатывать следующие запросы:

- Добавить в структуру число x;
- Для y вычислить сумму значений $f(x, y)$ по всем ранее добавленным в структуру числам x.

Каждое добавленное число x будем рассматривать как строку $p_1 \leqslant p_2 \ldots \leqslant p_k$ (его разложение на простые множители), также будем поддерживать бор на всех таких строках.

Чтобы ответить на второй запрос для заданного y, будем спускаться по соответствующей последовательности простых для числа y. Если мы находимся в какой-то вершине бора v и хотим спуститься по символу p (очередному простому делителю y), то мы тут же можем обработать все добавленные в структуру x, чей маршрут в боре совпадает с маршрутом y до вершины v, но далее расходится.

Пусть из вершины v есть переходы по символам $q_1 < q_2 < \ldots < q_l$ в поддеревья с размерами $s_1, \ldots, s_l$. Тогда можно вычислить сумму $f(x, y)$ по всем рассматриваемым x:

$$\sum f(x, y) = p \cdot \sum_{q_i > p} s_i + \sum_{q_i < p} q_i s_i$$

То есть сумма $f(x, y)$ по всем x, которые имеют заданные общий префикс с y сводится к двум запросам суммы на отрезке. Чтобы отвечать на запросы суммы эффективно, следует заранее построить бор для **всех** значений $a_1, \ldots, a_n$, отсортировать исходящие ребра из каждой вершины и построить дерево Фенвика для суммирования значений s_i и $q_i s_i$.

Пусть $\max\{a_1, \ldots, \} = C$, тогда, так как каждое число содержит не более $\log_2 C$ простых делителей, то полученное решение работает за время $O(n \log^2 C)$. Упомянем, что также допускались и другие реализации, использующие неявное дерево отрезков или даже декартово дерево.

Задача K. Белка и ступеньки

Автор задачи	Александр Бабин
Разработчик	Александр Макнил
Тема	Динамическое программирование

Задача решается с помощью техники динамического программирования. Вычислим значения $\mathrm{dp}[i][a][b]$ — количество маршрутов кошки, которые удовлетворяют следующим условиям:

- Все лапки находится на ступеньках i, $i-1$ или $i-2$.
- На ступеньке i находится a лапок, на ступеньке $(i-1)$ — b лапок, на $(i-2)$-й ступеньке $4-a-b$ лапок.
- Не фиксируется, какие именно лапки находятся на ступеньках $i, i+1, i+2$, но пары маршрутов, приводящие к тому, что хотя бы на одной ступеньке находятся различные множества лапок (например, в одном маршруте на ступеньке i находится левая передняя, а в другом — правая задняя) считаются различными.

Ясно, что имеет смысл рассматривать состояния, где $1 \leqslant i \leqslant \max n$, $1 \leqslant a \leqslant 4$ и $0 \leqslant 3 \leqslant b$ и $a + b \leqslant 4$, всего получается $10 \cdot \max n$ состояний динамики.

Для удобства семейство маршрутов, учитывающихся в одном состоянии ДП обозначать четверкой (i, a, b, c) $(c = 4 - a - b)$. Для пересчета такой динамики следует учитывать 3 вида переходов:

- Если $c = 0$, то есть переход $(i, a, b, 0) \to (i + 1, 1, a - 1, b)$ с кратностью a.
- Если $c > 0$, то есть переход $(i, a, b, c) \to (i, a, b + 1, c - 1)$ с кратностью c.
- Если $b > 0$, то есть переход $(i, a, b, c) \to (i, a + 1, b - 1, c)$ с кратностью b.

После вычисления ДП можно за $O(1)$ отвечать на каждый из запросов, используя значения $\mathrm{dp}[n][4][0]$. Разумеется все вычисления требуется проводить по модулю $10^9 + 7$.

Задача L. Векторная магия

Автор задачи	Бабин Александр
Разработчик	Грекова Дарья
Тема	Сортировка

Заметим, что выражение $\sqrt{b_1^2 + \ldots + b_n^2}$ может принимать ровно $n + 1$ различных значений.

Действительно, если среди чисел $b_1, \ldots, b_n$ ровно k чисел принимают значения ± 3, а остальные $n - k$ принимают значения ± 1, то $\sqrt{b_1^2 + \ldots + b_n^2} = \sqrt{n + 8k}$.

При фиксированном k легко максимизировать значение $a_1 b_1 + \ldots a_n b_n$. Видно, что для каждого i $(1 \leqslant i \leqslant n)$ знак b_i должен совпадать со знаком a_i (выгодно, чтобы каждое слагаемое было неотрицательным). Таким образом:

$$a_1 b_1 + \ldots + a_n b_n = |a_1| \cdot |b_1| + |a_2| \cdot |b_2| + \ldots + |a_n| \cdot |b_n|, \quad |b_i| \in \{1, 3\}$$

Осталось заметить, что выгодно выбрать $b_i = 3$ на тех k позициях, где $|a_i|$ максимальны.

Таким образом решение сводится к следующему:

- С помощью сортировки найти $p_1, \ldots, p_n$ — перестановку индексов, такую что $|a[p_1]| \geqslant \ldots \geqslant |a[p_n]|$
- Вычислить все префиксные и суффиксные суммы в массиве $|a[p_1]|, \ldots, |a[p_n]|$.
- Перебрать значение k и вычислить q_k для каждого k $(0 \leqslant k \leqslant n)$:

$$q_k = \frac{3 \cdot (|a[p_1]| + \ldots + |a[p_k]|) + (|a[p_{k+1}]| + \ldots + |a[p_n]|)}{\sqrt{n + 8k}}$$

- Среди $q_0, \ldots, q_n$ выбрать максимальное q_k и восстановить ответ.

Точности типа `double` хватает для тестов жюри, поэтому можно его использовать на последнем этапе вычислений при делении на радикал $\sqrt{1+8k}$. Но задачу можно решить в целых числах с помощью 64-битного целочисленного типа данных, используя следующий трюк:

$$a, b, c, d > 0 \colon \frac{a}{\sqrt{c}} < \frac{b}{\sqrt{d}} \iff a^2 d < b^2 c$$

Все вычисленные значения при использовании этого трюка не превосходят:

$$(\max |a_i| \cdot n)^2 \cdot 9n = 9n^3 (\max |a_i|)^2 \leqslant 9 \cdot 10^{12} \cdot 10^6 < 9 \cdot 10^{18} < 2^{63}$$