Технокубок 2017/2018. Второй отборочный раунд. Условия и разбор задач.

A. Поиск красивых чисел

ограничение по времени на тест

1 секунда

ограничение по памяти на тест

256 мегабайт

ввод

стандартный ввод

вывод

стандартный вывод

Дано два списка различных ненулевых цифр.

Назовем число красивым, если в его записи (в системе счисления по основанию 10) присутствует хотя бы одна цифра из первого списка и хотя бы одна цифра из второго списка. Чему равно минимальное натуральное (положительное целое) красивое число?

Входные данные

В первой строке даны числа n, m ($1 \le n, m \le 9$) – длины первого и второго списка соответственно.

Во второй строке через пробел даны n различных целых цифр a_1, a_2, ..., a_n ($1 \le a_i \le 9$) – элементы первого списка.

В третьей строке через пробел даны m различных целых цифр b_1, b_2, ..., b_m ($1 \le b_i \le 9$) – элементы второго списка.

Выходные данные

Выведите минимальное натуральное красивое число.

Примеры

входные данные

```
2 3
4 2
5 7 6
```

выходные данные

```
25
```

входные данные

```
8 8
1 2 3 4 5 6 7 8
8 7 6 5 4 3 2 1
```

выходные данные

```
1
```

Примечание

В первом примере красивыми являются числа 25, 46, 24567 и многие другие. Из них минимальным является 25. 42 и 24 не являются красивыми, так как в них отсутствуют цифры из второго списка.

Во втором примере красивыми являются все числа, в чьей записи встречаются не только цифры 9. Очевидно, минимальным из таких чисел является 1, так как это минимальное натуральное число.

870A - Поиск красивых чисел

Заметим, что длина ответа не превосходит двух, так как мы можем взять по цифре из первого и второго списков и составить из них красивое число. Поэтому нам нужно проверить два случая:

1) Перебрать цифру из первого списка и составить число из нее, если цифра есть в обоих списках.

2) Перебрать цифру из первого списка и цифру из второго списка. Составить из них число двумя способами (сначала цифра из первого списка, потом из второго и наоборот).

Из всех вариантов нужно выбрать минимальный.

Код (C++)
Код (Python)

B. Максимум максимума из минимумов

ограничение по времени на тест

1 секунда

ограничение по памяти на тест

256 мегабайт

ввод

стандартный ввод

вывод

стандартный вывод

Дан массив $a_1, a_2, ..., a_n$ из n чисел, а также число k. Массив нужно разбить на ровно k непустых подотрезков. На каждом подотрезке вычисляется минимум, а из полученных k минимумов берется максимум. А какое максимальное число можно таким образом получить?

Определения подотрезка и разбиения массива на подотрезки находятся в примечаниях.

Входные данные

В первой строке записаны через пробел два целых числа n и k ($1 \le k \le n \le 10^5$) — размер массива a и количество подотрезков, на которые нужно разбить массив.

Во второй строке записано n целых чисел через пробел: $a_1, a_2, ..., a_n$ ($-10^9 \le a_i \le 10^9$).

Выходные данные

Выведите единственное число — максимальный максимум, который можно получить, разбив данный массив на k подотрезков и взяв максимум из минимумов на этих подотрезках.

Примеры

входные данные

```
5 2
1 2 3 4 5
```

выходные данные

```
5
```

входные данные

```
5 1
-4 -5 -3 -2 -1
```

выходные данные

```
-5
```

Примечание

Подотрезок $[l, r]$ ($l \le r$) массива a — это последовательность $a_l, a_{l+1}, ..., a_r$.

Разбиение массива a из n элементов на k подотрезков, $[l_1, r_1]$, $[l_2, r_2]$, ..., $[l_k, r_k]$ ($l_1 = 1$, $r_k = n$, $l_i = r_{i-1} + 1$ при $i > 1$), — kпоследовательностей $(a_{l_1}, ..., a_{r_1})$, ..., $(a_{l_k}, ..., a_{r_k})$.

В первом примере вы должны разбить массив на подотрезки $[1, 4]$ и $[5, 5]$, тогда вы получите последовательности $(1, 2, 3, 4)$ и (5). Минимумы равны $min(1, 2, 3, 4) = 1$ и $min(5) = 5$. Итоговый максимум равен $max(1, 5) = 5$. Очевидно, получить больший результат нельзя.

Во втором примере единственная ваша возможность — разбить массив на единственный подотрезок $[1, 5]$, тогда вы получите последовательность $(-4, -5, -3, -2, -1)$. Единственный минимум равен $min(-4, -5, -3, -2, -1) = -5$. Итоговый максимум равен -5.

870B - Максимум максимума из минимумов

Для решения задачи нужно рассмотреть 3 случая:

$k \geq 3$: тогда пусть *pos_max* - позиция элемента с максимальным значением, тогда всегда можно разделить массив на подотрезки так, чтобы один подотрезок содержал только это число, а значит ответ на задачу - a_{pos_max}.

k = 2: тогда все возможные разделения - некоторый префикс ненулевой длины и некоторый суффикс ненулевой длины. Позицию разделения можно перебрать, а минимумы на всех суффиксах и префиксах можно посчитать заранее. Ответ - максимум из получившихся ответов.

Также можно доказать что при *k = 2* ответ - максимум из первого и последнего элемента.

k = 1: тогда единственное возможное разделение - один отрезок равный всему массиву. А значит ответ - минимальное значение на всем массиве.

Код

C. Максимальное разбиение

ограничение по времени на тест

2 секунды

ограничение по памяти на тест

256 мегабайт

ввод

стандартный ввод

вывод

стандартный вывод

Вам задано некоторое количество запросов. В i-м запросе дано положительное целое число n_i. Надо представить n_i в виде суммы максимального количества составных слагаемых и вывести это количество, либо вывести -1, если такое представление невозможно.

Составными называются натуральные числа большие 1, не являющиеся простыми, то есть, имеющие натуральные делители, отличные от 1 и самого числа.

Входные данные

В первой строке дано число q ($1 \le q \le 10^5$) — количество запросов

Далее идет q строк. В $(i+1)$-й строке дано число n_i ($1 \le n_i \le 10^9$) — i-й запрос.

Выходные данные

Для каждого запроса выведите максимальное количество слагаемых в корректном разбиении на составные слагаемые или -1, если такого разбиения не существует.

Примеры

входные данные

```
1
12
```

выходные данные

```
3
```

входные данные

```
2
6
8
```

выходные данные

```
1
2
```

входные данные

```
3
1
2
3
```

выходные данные

```
-1
-1
-1
```

Примечание

$12 = 4 + 4 + 4 = 4 + 8 = 6 + 6 = 12$, но в первом разбиении больше количество слагаемых

$8 = 4 + 4$, 6 нельзя разбить на меньшие составные слагаемые.

$1, 2, 3$ меньше любого составного числа, поэтому для них не существует корректных разбиений.

870C - Максимальное разбиение

Заметим, что минимальное составное число равно четырем. Довольно логично, что четверка должна встречаться довольно часто в разбиении больших чисел. Давайте для достаточно малых чисел напишем динамику dp_n – количество слагаемых в разбиении числа n.

Если нам дано в запросе маленькое число, то выведем соответствующее значение динамики. Иначе уменьшим число на такое минимальное количество четверок, чтобы получить новое число, для которого ответ уже посчитан. Выведем как ответ значение динамики плюс количество четверок.

Динамику можно находить за $O(n^2)$, да и вообще за любую разумную сложность. В частности можно было разобрать все случаи руками, если выбрать $n = 15$ (далее будет доказано, что этого хватает.

В итоге у нас есть правильное решение, но не совсем очевидно, почему оно работает.

Доказательство. Оно не совсем красиво, но рассуждая похожим образом, можно было прийти к решению.

Давайте найдем ответ для чисел от 1 до 15. Несколько наблюдений:

1) В их разбиении встречаются только 4, 6 и 9

2) Невыгодно использовать 6 или 9 более раза, так как 6 + 6 = 4 + 4 + 4, 9 + 9 = 6 + 6 + 6

3) 12, 13, 14, 15 имеют корректные разбиения

Докажем, что для всех чисел больше 15 в разбиении будет четверка. Пусть ее нет. Если минимальное число в разбиении не 4, 6 или 9, то оно будем иметь разбиение на более чем одно слагаемое по индукции. Поэтому это невыгодно. Если 6 или 9, то уменьшая число на выбранное слагаемое, мы в конце концов получим число меньшее или равное 15. Либо разбиения такого числа не существует, либо в нем есть четверка, что противоречит минимальности выбранного числа, либо будет и шестерка, и девятка, что противоречит второму наблюдению. В том, что нет других случаев, можно убедиться разбирая все варианты от 1 до 15.

Код

D. Что-то там с xor запросами

ограничение по времени на тест

2 секунды

ограничение по памяти на тест

256 мегабайт

ввод

стандартный ввод

вывод

стандартный вывод

Это интерактивная задача.

Жюри загадало некоторую перестановку p из чисел от 0 до n - 1, про которую вам известна только ее длина n. Напомним, что в перестановке все числа различны.

Пусть b — обратная к p перестановка, то есть $p_{b_i} = i$ для всех i. Тогда единственное действие, которое вы можете выполнить — узнать `xor` элементов p_i и b_j, указав их индексы i и j (не обязательно различные). В результате запроса для индексов i и j вы получите значение $p_i \oplus b_j$, где символом $\oplus$ обозначена операция `xor`. Описание операции `xor` вы можете найти в примечаниях.

Заметим, что некоторые перестановки могут быть неотличимы от заданной, даже если сделать все различные n^2 запросов. Вам необходимо вычислить, сколько перестановок неотличимы от загаданной, а также выдать одну из таких перестановок, сделав не более $2n$ запросов.

Загаданная перестановка не зависит от ваших запросов.

Входные данные

В первой строке входных данных следует целое положительное число n ($1 \le n \le 5000$) — длина загаданной перестановки. В первую очередь ваша программа должна прочитать это число.

Выходные данные

Когда ваша программа будет готова вывести ответ, выведите три строки.

В первой строке выведите один символ « ! ».

Во второй строке выведите одно целое число $answers\_cnt$ — количество перестановок, неотличимых от загаданной жюри, считая загаданную.

В третьей строке выведите n целых чисел $p_0, p_1, ..., p_{n-1}$ ($0 \le p_i < n$, все p_i должны быть различны) — одну из перестановок, неотличимых от загаданной жюри.

После вывода ответа ваша программа должна завершиться.

Протокол взаимодействия

Чтобы узнать `xor` двух элементов, выведите строку вида «`? i j`», где i и j — целые числа от 0 до $n - 1$ — индекс элемента перестановки и индекс элемента обратной перестановки, `xor`-сумму которых ваша программа запрашивает. После этого выведите перевод строки и сделайте операцию `flush`.

После выполнения запроса ваша программа может считать целое число, равное $p_i \oplus b_j$.

Для перестановки длины n ваша программа должна сделать не более $2n$ запросов на `xor`-сумму. Обратите внимание, что вывод ответа не учитывается при подсчете количества запросов. Обратите внимание, что вы можете задать не более $2n$ запросов. В случае, если ваша программа сделает больше $2n$ запросов или сделает хотя бы один некорректный запрос, ваше решение получит вердикт «`Неправильный ответ`».

Если в какой-то момент ваша программа считывает `-1` как ответ, она должна немедленно завершиться (например, вызовом `exit(0)`). Вы получите вердикт «`Неправильный ответ`», и это будет означать, что вы задали больше $2n$ запросов или задали некорректный запрос. Если вы проигнорируете это, то можете получить любой вердикт, так как ваша программа продолжит читать из закрытого потока ввода.

Выше решение получит вердикт «`Решение зависло`», если вы не будете ничего выводить или забудете сделать операцию `flush` после вывода вопроса или ответа.

Чтобы выполнить операцию `flush`, можете использовать (сразу после вывода запроса и перевода строки):

- `fflush(stdout)` в C++;
- `System.out.flush()` в Java;
- `stdout.flush()` в Python;
- `flush(output)` в Pascal;
- Для других языков смотрите документацию.

Взломы

Используйте следующий формат для взломов:

n

$p_0\ p_1\ \ldots\ p_{n-1}$

Взламываемая программа не будет иметь доступа к этим данным.

Примеры

входные данные

```
3
0
0
3
2
3
2
```

выходные данные

```
? 0 0
? 1 1
? 1 2
? 0 2
? 2 1
? 2 0
!
1
0 1 2
```

входные данные

```
4
2
3
2
0
2
3
2
0
```

выходные данные

```
? 0 1
? 1 2
? 2 3
? 3 3
? 3 2
? 2 1
? 1 0
? 0 0
!
2
3 1 2 0
```

Примечание

Операцией `xor`, или побитовое исключающее ИЛИ, называется операция над двумя целыми числами, при которой i-й разряд результата в двоичной системе счисления будет равен 1 тогда и только тогда, когда ровно у одного из двух целых чисел в i-м разряде в двоичной системе счисления стоит 1. Больше информации смотрите по ссылке.

В первом примере при $p = [0, 1, 2]$, а значит $b = [0, 1, 2]$, на заданные запросы у этой перестановки совпадают все значения $p_i \oplus b_j$ для всех выведенных парах i, j. Кроме этой перестановки не существует других перестановок, подходящих под выданные ответы, поэтому это — ответ.

Ответы на запросы:

- $p_0 \oplus b_0 = 0 \oplus 0 = 0$,
- $p_1 \oplus b_1 = 1 \oplus 1 = 0$,
- $p_1 \oplus b_2 = 1 \oplus 2 = 3$,
- $p_0 \oplus b_2 = 0 \oplus 2 = 2$,
- $p_2 \oplus b_1 = 2 \oplus 1 = 3$,
- $p_2 \oplus b_0 = 2 \oplus 0 = 2$.

В втором примере при $p = [3, 1, 2, 0]$, а значит $b = [3, 1, 2, 0]$, на заданные запросы у этой перестановки совпадают все значения $p_i \oplus b_j$ для всех выведенных пар i, j. Однако кроме нее подходит также перестановка $p = [0, 2, 1, 3]$, $b = [0, 2, 1, 3]$, причем на всех n^2 возможных запросов у этих двух перестановок ответы будут совпадать, а у всех остальных перестановок ответы будут другие уже на заданных запросах.

870D - Что-то там с xor запросами

Утверждение: те и только те перестановки у которых ответы на запросы (0, *i*) и (*i*, 0) для всех *i* совпадут с ответами данными программе, подходят под все возможные запросы.

Доказательство: $p_i \oplus b_j = (p_i \oplus b_0) \oplus (p_0 \oplus b_j) \oplus (p_0 \oplus b_0)$ а значит по ответам на запросы (0, *i*) и (*i*, 0) можно восстановить ответы на все остальные запросы.

Если фиксировать значение b_0 то можно восстановить всю перестановку, так-как мы знаем ответы на запросы (*i*, 0), и $p_i = (p_i \oplus b_0) \oplus b_0$.

То-есть можно перебрать значение b_0, восстановить по нему всю перестановку, и если в ней не возникло противоречий (то-есть каждое число от 0 до *n* - 1 встречалось 1 раз) и для всех *i* значения $p_0 \oplus b_i$ и совпадают с ответами данными программе то данная перестановка подходит под ответ.

Ответ - количество подошедших перестановок, и любая из подошедших перестановок.

Код

E. Точки, прямые и неоригинальные названия

ограничение по времени на тест

2 секунды

ограничение по памяти на тест

256 мегабайт

ввод

стандартный ввод

вывод

стандартный вывод

На плоскости дано n различных точек с целыми координатами. Для каждой точки можно выполнить одно из трех действий: провести через нее вертикальную прямую, провести через нее горизонтальную прямую или ничего не делать.

Если рассматривать несколько совпадающих прямых как одну, то сколько различных картинок может получиться? Ответ вывести по модулю $10^9 + 7$.

Входные данные

В первой строке дано целое число n ($1 \le n \le 10^5$) — количество точек.

Далее идет n строк. В $(i + 1)$-й строке даны два целых числа x_i, y_i ($-10^9 \le x_i, y_i \le 10^9$) — координаты i-й точки.

Гарантируется, что все точки различны.

Выходные данные

Вывести количество различных картинок по модулю $10^9 + 7$.

Примеры

входные данные

```
4
1 1
1 2
2 1
2 2
```

выходные данные

```
16
```

входные данные

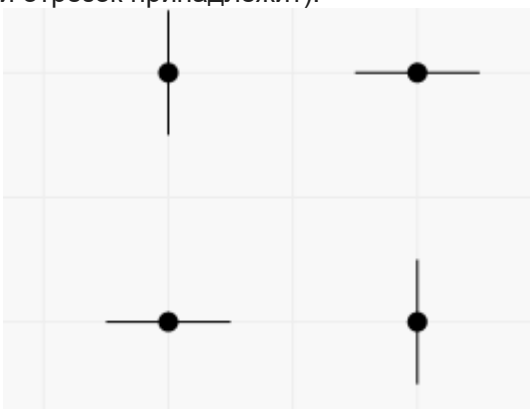
```
2
-1 -1
0 1
```

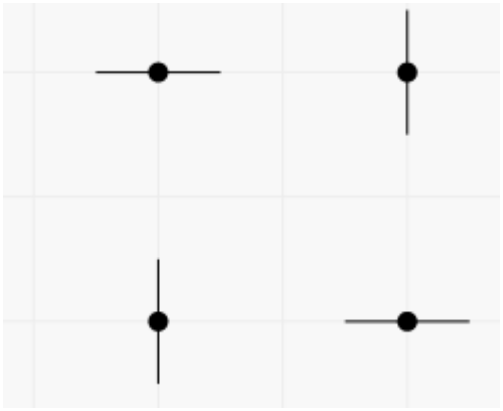
выходные данные

```
9
```

Примечание

В первом примере через точки проходят две вертикальные и две горизонтальные прямые. Существуют способы получить картинку из любого набора этих прямых. В частности, получить картинку на которой нарисованы все четыре прямые можно двумя способами (каждый отрезок обозначает прямую, которой отрезок принадлежит).

Первый способ:  Второй способ:



Во втором примере для каждой точки можно независимо выполнить одно из трех действий. Количество картинок равно $3^2 = 9$.

870E - Точки, прямые и неоригинальные названия

Построим граф на точках. Проведем ребро от каждой точки к ее ближайшему соседу сверху, снизу, слева и справа (если соответствующий сосед существует). Заметим, что можно решить задачу независимо для каждой компоненты связности и перемножить ответы для них. Не теряя общности будем считать, что граф связен.

Обозначим количество различных x-координат точек как *X*, количество различных y-координат точек как *Y*.

Пусть в графе есть цикл. Будем рассматривать его без промежуточных вершин (которые лежат на одной прямой с предыдущей и следующей вершиной цикла). Направим из каждой вершины цикла прямую в следующую вершину (из последней в

первую). Теперь мы получили все прямые соответствующие *x*-координатам и *y*-координатам вершин цикла. Докажем по индукции, что мы можем получить все такие прямые для всего графа.

Запустим обход в глубину из вершин цикла. Пусть мы пришли в какую-то вершину не из цикла. У нее должен быть хотя бы один посещенный сосед слева, справа, сверху или снизу. По предположению индукции для всех посещенных вершин мы можем получить полный набор прямых. Следовательно, текущую вершину уже должна пересекать хотя бы одна прямая. Проведем прямую из точки в другом направлении и продолжим обход. Так мы в итоге получим все прямые для всего графа. Заметим, что промежуточные вершины из цикла также будут обработаны корректно при обходе в глубину.

Если мы получим все прямые, то мы получим и все их подмножества, поэтому для графа с циклом ответ 2^{X+Y}.

Пусть в графе нет циклов (это дерево). Утверждается, что любой неполный набор прямых можно получить.

Зафиксируем набор и проведем фиктивную прямую, которой нет в наборе, не накладывая ограничений на точки. Аналогично случаю с циклами можно доказать по индукции, что можно получить все прямые (фиктивная не считается).

Теперь докажем, что полный набор прямых получить нельзя. Для одной вершины это известно. Иначе возьмем какой-то лист дерева. Мы можем провести прямую, не направленную из листа в другую вершину, таким образом перейдя к количеству вершин, меньшему на 1. Иначе мы не сможем провести эту прямую другим способом. Утверждение доказано.

Поэтому для дерева ответ $2^{X+Y} - 1$.

В итоге задача сводится к построению графа и проверке каждой компоненты на то, является ли она деревом.

Код

F. Пути

ограничение по времени на тест

4 секунды

ограничение по памяти на тест

512 мегабайт

ввод

стандартный ввод

вывод

стандартный вывод

Дано положительное целое число n. Построим граф на вершинах $1, 2, ..., n$ так, чтобы ребро между вершинами u и vсуществовало тогда и только тогда, когда $\gcd(u, v) \neq 1$.
Пусть $d(u, v)$ — кратчайшее расстояние между u и v или 0, если между ними нет пути.
Посчитайте сумму $d(u, v)$ для всех $1 \leq u < v \leq n$.

gcd или НОД (наибольший общий делитель) двух натуральных чисел — такое наибольшее натуральное число, которое делит оба этих числа нацело.

Входные данные

Целое число n ($1 \leq n \leq 10^7$).

Выходные данные

Выведите сумму $d(u, v)$ для всех $1 \leq u < v \leq n$.

Примеры

входные данные

```
6
```

выходные данные

```
8
```

входные данные

```
10
```

выходные данные

```
44
```

Примечание

Все кратчайшие пути в первом примере:

- $2 \to 6 \to 3$
- $2 \to 4$
- $2 \to 6$
- $3 \to 6 \to 4$

- $3 \to 6$
- $4 \to 6$

Между остальными парами вершин путь не существует.

Суммарное расстояние $2 + 1 + 1 + 2 + 1 + 1 = 8$.

870F - Пути

Будем считать число $1 \le x \le n$ плохим, если оно равно 1 или является простым, большим $n / 2$. Иначе будем считать его хорошим.

Путь между двумя вершинами u и v не существует, если хотя бы одно из них плохое.

Расстояние равно 0, если $u = v$

Расстояние равно 1, если u и v имеют общий делитель.

Расстояние равно 2, если $prime_u \cdot prime_v \le n$, где $prime_x$ равно минимальному простому делителю x.

Иначе расстояние равно 3 (мы всегда можем проделать путь $u \to 2 \cdot prime_u \to 2 \cdot prime_v \to v$).

Тривиально найти количество пар, между которыми нет пути.

Количество пар с расстоянием 1 равно сумме по всем хорошим x выражения $x - 1 - \varphi(x)$.

Количество пар с расстоянием 3 можно найти, вычтя из общего количества пар количество пар с расстоянием 0 и 1 и количество пар, между которыми пути нет.

Осталось найти количество пар с расстоянием 2. Разобьем пары на три типа

1) Пара составных чисел, не имеющих общего делителя

2) Хорошее простое число и любое составное

3) Два разных хороших простых числа.

Для 1 ответ равен сумме по всем составным числам $1 \le x \le n$ выражения $\varphi(x)$ - (количество составных чисел меньших x) + количество уникальных простых делителей x

Для 2 нужно просуммировать для всех хороших простых x количество хороших чисел t таких, что $prime_t * prime_x \le n$ и вычесть количество хороших простых p таких, что $prime_p * prime_x \le n$, чтобы учитывать только составные вторые числа в

парах. $prime_x$ везде можно заменить на x.

3 ищется тривиально, нужно просто перебрать все пары хороших простых чисел, произведение которых не превосходит n.

За остальными деталями смотрите авторский код.

Код

E. Восстановление дерева

ограничение по времени на тест
3 секунды
ограничение по памяти на тест
256 мегабайт
ввод
стандартный ввод
вывод
стандартный вывод

У Пети было дерево, состоящее из n вершин, пронумерованных целыми числами от 1 до n. Но по стечению обстоятельств он своё дерево потерял.

Петя помнит о k вершинах из своего дерева информацию о расстоянии от каждой из них до всех n вершин дерева.

Перед вами стоит задача восстановить любое дерево, удовлетворяющее информации, которую помнит Петя, либо сообщить, что это невозможно.

Входные данные

В первой строке следуют два целых числа n и k ($2 \le n \le 30\,000$, $1 \le k \le min(200, n)$) — количество вершин в дереве и количество вершин, о которых помнит Петя.

В следующих k строках содержится информация о вершинах, которую помнит Петя. В i-й строке содержится n целых чисел $d_{i,1}$, $d_{i,2}$, ..., $d_{i,n}$ ($0 \le d_{i,j} \le n$ - 1), где $d_{i,j}$ — расстояние до j-й вершины от i вершины, которую помнит Петя.

Выходные данные

Если не существует подходящего дерева, выведите `-1`.

В противном случае, выведите в n - 1-й строке по два целых числа — концы очередного ребра. Вы можете выводить ребра и концы ребер в любом порядке. Вершины дерева пронумерованы от 1 до n.

Если ответов несколько, разрешается вывести любой из них.

Примеры

входные данные

```
5 2
0 1 2 3 2
2 1 0 1 2
```

выходные данные

```
2 1
3 2
4 3
5 2
```

входные данные

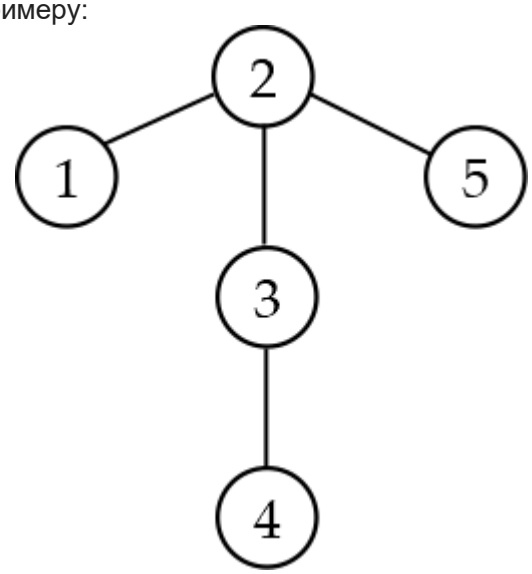
```
3 1
1 2 1
```

выходные данные

```
-1
```

Примечание

Иллюстрация к первому примеру:



871E - Восстановление дерева

В начале стоит заметить, что можно узнать какие номера у вершин, расстояния до которых заданы, номер *i*-ой заданной вершины равен id_i, такому что $d_{i, id_i} = 0$. Если такой вершины нет, или она не единственная то ответа не существует.

Зафиксируем корень *root* равный какой-нибудь вершине из тех, расстояния от которых заданы во входных данных. Допустим $root = id_1$. Для любой вершины id_i мы можем найти вершины лежащие на пути от root до этой вершины, так-как для таких и только для таких вершин выполняется $d_{1, v} + d_{i, v} = d_{1, id_i}$. И соответственно подходящее под это условия вершина *v* будет находиться на расстояние $d_{1, v}$ от *root*. А значит мы научились строить часть дерева, которое состоит из вершин, которые лежат на путях от *root* до какой-то вершины id_i. Если мы не смогли таким образом построить пути, то решения не существует. Это построение работает за $O(nk)$.

Теперь рассмотрим остальные вершины, по возрастанию глубины (расстоянию до корня). Пусть мы рассматриваем фиксированную вершину v, посмотрим на путь от нее до root, этот путь можно разбить на 2 части - *(root, u), (u, v)* где *u* - вершина из уже построенной части дерева, давайте из таких u найдем самую глубокую, это можно сделать за $O(k)$ воспользовавшись тем что *u* - самая глубокая вершина среди $lca(v, id_i)$, которое равно вершине на пути от *root* до id_i на глубине $d_{1, id_i} + d_{1, v} - d_{i, v}$. Тогда предок *v* - вершина, которая таким-же образом была добавлена в

поддерево *u* но с глубиной на 1 меньше, либо сама вершина *u* (если глубина *u* на 1 меньше глубины вершины *v*). Если такой вершины еще не добавили, то ответа не существует, так-как мы рассматривали вершины по возрастанию глубины. Добавление каждой вершины работает за *O(k)*.

Итоговое дерево и будет искомым. Весь алгоритм работает за *O(nk)*.
Код