Обход в глубину (Depth-First-Search) и родственные задачи.

A. *Количество вершин в компоненте связности*

Дан неориентированный граф. Требуется найти количество вершин, лежащих в одной компоненте связности с данной вершиной (считая эту вершину).

В первой строке входных данных содержатся два числа: N и S ($1 \leqslant N \leqslant 100$; $1 \leqslant S \leqslant N$), где N — количество вершин графа, а S — заданная вершина. В следующих N строках записано по N чисел — матрица смежности графа, в которой 0 означает отсутствие ребра между вершинами, а 1 — его наличие. Гарантируется, что на главной диагонали матрицы всегда стоят нули.

Выведите одно целое число — искомое количество вершин.

Input	Output
3 1 0 1 1 1 0 0 1 0 0	3

B. *Проверить является ли граф деревом*

Дан неориентированный граф. Необходимо определить, является ли он деревом.

В первой строке входного файла содержится одно натуральное число N ($N \leqslant 100$) — количество вершин в графе. Далее в N строках записано по N чисел — матрица смежности графа.

На главной диагонали матрицы стоят нули. Матрица симметрична относительно главной диагонали.

Вывести `YES`, если граф является деревом, и `NO` в противном случае.

Input	Output
6 0 1 1 0 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0	NO
3 0 1 0 1 0 1 0 1 0	YES

C. *Компоненты связности*

Дан неориентированный граф. Необходимо посчитать количество его компонент связности и вывести их.

Во входном файле записано два числа N и M ($0 < N \leqslant 100000, 0 \leqslant M \leqslant 100000$). В следующих M строках записаны по два числа i и j ($1 \leqslant i, j \leqslant N$), которые означают, что вершины i и j соединены ребром.

В первой строчке выходного файла выведите количество компонент связности. Далее выведите сами компоненты связности в следующем формате: в первой строке количество вершин в компоненте, во второй — сами вершины в произвольном порядке.

Input	Output
6 4 3 1 1 2 5 4 2 3	3 3 1 2 3 2 4 5 1 6

D. *Цикл в ориентированном графе*

Дан ориентированный граф. Требуется определить, есть ли в нём цикл.

Циклом называется простой путь $v_1 \to v_2, v_2 \to v_3, \ldots, v_{n-1} \to v_n$ ($v_i \neq v_j$), такой, что существует также ребро $v_n \to v_1$.

В первой строке входного файла содержится одно натуральное число N ($N \leqslant 50$) — количество вершин в графе. Далее в N строках по N чисел — матрица смежности графа. Гарантируется, что на диагонали матрицы будут стоять нули.

Выведите `NO`, если в заданном графе цикла нет, и `YES`, если он есть.

Input	Output
3 0 1 0 0 0 1 0 0 0	NO
3 0 1 0 0 0 1 1 0 0	YES

E. *Цикл в неориентированном графе*

Дан неориентированный граф. Требуется определить, есть ли в нём цикл, и, если есть, вывести его.

Циклом в *неориентированном графе* называется простой путь, содержащий попарно различные рёбра $v_1 \leftrightarrow v_2, v_2 \leftrightarrow v_3, \ldots, v_{n-1} \leftrightarrow v_n$ ($v_i \neq v_j$), такой, что существует также ребро $v_n \leftrightarrow v_1$.

В первой строке дано одно число N ($1 \leqslant N \leqslant 500$) — количество вершин в графе. Далее в N строках задан сам граф матрицей смежности.

Если в исходном графе нет цикла, то выведите `NO`. Иначе, в первой строке выведите `YES`, во второй строке выведите число K — количество вершин в цикле, а в третьей строке выведите K различных чисел — номера вершин, которые принадлежат циклу в порядке его обхода (обход можно начинать с любой вершины цикла). Если циклов несколько — выведите любой.

Input	Output
3 0 1 1 1 0 1 1 1 0	YES 3 3 2 1
4 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 0	NO

F. *Проверка графа на двудольность*

На банкет были приглашены N человек. Были поставлены 2 стола. Столы достаточно большие, чтобы все посетители банкета могли сесть за любой из них. Проблема заключается в том, что некоторые люди не ладят друг с другом и не могут сидеть за одним столом. Вас попросили определить, возможно ли всех людей рассадить за двумя столами.

В первой строке входных данных содержатся два числа: N и M ($1 \leqslant N, M \leqslant 100$), где N — количество людей, а M — количество пар, которые не могут сидеть за одним столом. В следующих M строках записано по 2 числа — пары людей, которые не могут сидеть за одним столом.

Если существует способ рассадить всех присутствующих — выведите `YES` в первой строке и номера людей, которых необходимо посадить за первый стол, во второй строке. В противном случае в первой и единственной строке выведите `NO`.

Input	Output
3 2 1 2 1 3	YES 1

G. *Диаметр дерева*

Дано дерево, содержащее N вершин.

Диаметром дерева называется длина максимального пути (количество рёбер) между двумя его вершинами.

Вычислите диаметр данного дерева.

В первой строке входных данных содержатся число вершин дерева N ($1 \leqslant N \leqslant 10^4$). В следующих $N - 1$ строках записано по 2 числа, обозначающие рёбра дерева (вершины нумеруются числами от 1 до N).

Программа должна вывести одно число — диаметр данного дерева.

Input	Output
5 1 2 1 3 3 4 3 5	3

Замечание к тесту: диаметр дерева соответствует такому пути: $2 \to 1 \to 3 \to 5$

H. *Проверка обхода в глубину*

Дан неориентированный граф и некоторая последовательность номеров его вершин. Требуется определить, может ли эта последовательность вершин быть результатом работы программы, перечисляющей *все* вершины графа в порядке обхода в глубину?

Здесь имеется в виду такая реализация `DFS`, дополненная вызовами ранее непосещённых вершин в каком-то произвольном порядке:

```
def dfs(G, visited, v):
    print(v)
    visited[v] = True
    for u in G[v]:
        if not visited[u]:
            dfs(G, visited, u)
```

В первой строке через пробел записаны два целых числа N и M ($1 \leqslant n \leqslant 10^5, 0 \leqslant m \leqslant 10^6$) — количество вершин и ребер графа. В следующих M строчках записано по два целых числа a и b ($1 \leqslant a, b \leqslant n; a \neq b$) — номера вершин, которые соединяет описываемое ребро.

В следующей строке записано число Q — количество запросов. Наконец, в каждой из следующих Q строк записана последовательность чисел a_i ($1 \leqslant a_i \leqslant N$).

Программа должна вывести Q строк: в i-й строке следует вывести `YES`, если последовательность a_i может быть получена при выполнении обхода в глубину на данном графе начиная с какой-то вершины в каком-то порядке обхода и `NO`, если не может быть получена ни при каком порядке обхода.

Input	Output
3 2 1 2 3 2 2 1 2 3 3 1 2	YES NO
4 2 1 2 3 4 1 1 3 2 4	NO

I. *Максимальное количество рёбер*

Есть программа обхода графа в глубину (ниже представлены реализации на Python и C++):

```python
def dfs(G, visited, v):
    print(v + 1)
    visited[v] = True
    for u in G[v]:
        if not visited[u]:
            dfs(G, visited, u)
            print(v + 1)

V, E = map(int, input().split())
G = [[] for k in range(N)]
for k in range(E):
    a, b = map(int, input().split())
    G[a - 1].append(b - 1)
    G[b - 1].append(a - 1)
for k in range(len(G)):
    G[k].sort()
visited = [False] * N
for v in range(V):
    if not visited[v]:
        dfs(G, visited, v)
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

void DFS(const vector <vector <int>>& G, vector<bool>& visited, int v) {
    visited[v] = true;
    cout << v + 1;
    for (int u: G[v]) {
        if (!visited[u]) {
            DFS(G, visited, u);
            cout << v + 1;
        }
    }
}

int main(){
    int N, V;
    cin >> N >> V;
    vector <vector <int>> G(N);
    vector <bool> visited(N, false);
    for (int i = 0; i < V; i++) {
        int u, v;
        cin >> u >> v;
        G[u - 1].push_back(v - 1);
        G[v - 1].push_back(u - 1);
    }
    for (int k = 0; k < N; k++) {
        sort(G[k].begin(), G[k].end());
    }
    for (int v = 0; v < N; v++) {
        if (!visited[v]) {
            DFS(G, visited, v);
        }
    }
}
```

Граф хранится списком смежности (вершины графа пронумерованы от 1 до N, в каждом списке вершины упорядочены по возрастанию). В массиве `visited` помечается, в каких вершинах обход в глубину уже побывал.

Программу запустили для некоторого (неизвестного) неориентированного графа G с N вершинами и сохранили её вывод. Какое максимальное количество рёбер могло быть в графе G?

Первая строка входного файла содержит два целых числа: N и M — количество вершин в графе и количество чисел в выведенной последовательности ($1 \leqslant N \leqslant 300, 1 \leqslant M \leqslant 2N - 1$). Следующие M строк содержат вывод программы — по одному числу на строку. Гарантируется,

что существует хотя бы один граф, запуск программы на котором приводит к приведённому во входном файле выводу.

В первой строке программа должна вывести одно целое число P — максимальное возможное количество рёбер в графе.

Следующие P строк должны содержать по два целых числа: номера вершин, соединённых ребрами. В графе не должно быть петель и кратных рёбер.

Input	Output
6 10 1 2 3 2 4 2 1 5 6 5	6 1 2 1 3 1 4 2 3 2 4 5 6

J. *Топологическая сортировка*

Топологической сортировкой ориентированного графа называется такой порядок перечисления его вершин, что все рёбра графа идут из вершины с ме́ньшим номером в вершину с бо́льшим номером.

В создании сложных устройств (самолёты, автомобили) принимает участие много различных производств. Они могут зависеть (или не зависеть) друг от друга. Например, для сборки авиадвигателя требуются в числе прочего турбина и лопатки (они сначала должны быть произведены и поставлены), а производство кресел и стёкол для кабины пилотов — независимые процессы.

Все процессы некоторого производства пронумеровали и составили список связей между ними. Каждая такая связь это пара номеров процессов a b, где a обязательно предшествует b. Требуется написать программу, которая проверит реализуемость такого набора связей меду процессами.

В первой строке входных данных записаны два числа N и M ($1 < N \leqslant 100, 1 \leqslant M \leqslant 5000$) — количество процессов и количество известных связей. Далее в M строках записаны связи — пары чисел A и B по одной на строке ($1 \leqslant A, B \leqslant N$). Каждая такая строка означает, что процесс с номером A должен быть выполнен до процесса с номером B.

Не гарантируется, что все пары чисел во входных данных различны.

В первой строке выведите `YES`, если производство с такими зависимостями между процессами можно реализовать или `NO`, если это невозможно.

После ответа `YES` на следующей строке выведите N чисел, разделённых пробелами, — любую из возможных последовательностей выполнения процессов, которая не противоречит входным данным.

Input	Output
4 5 1 2 2 3 3 4 1 4 4 1	NO

K. *Алфавит*

Страна X использует при письме маленькие буквы латинского алфавита. Однако упорядочивают их в другом порядке.

Вам дан набор слов. Необходимо восстановить порядок букв в алфавите таким образом, чтобы данные слова оказались упорядоченными по возрастанию в лексикографическом порядке. Если порядок существует, но его нельзя восстановить однозначно — нужно вернуть любой подходящий порядок букв. Если не существует порядка букв, удовлетворящего условию задачи — вывести -1.

В первой строке входных данных указано количество слов ($N < 1000$). Затем N строк с одним словом на каждой. Суммарная длина слов не превосходит 10000.

Программа должна вывести одну строку: упорядоченную последовательность букв без разделителей, либо -1.

В последовательности букв должны встретиться обязательно все буквы из набора слов и никаких других букв.

Input	Output
3 abc bc bb	acb

L. *Конденсация графа*

Вам задан ориентированный граф с N вершинами и M ребрами ($1 \leqslant N \leqslant 20000, 1 \leqslant M \leqslant 200000$). Найдите компоненты сильной связности заданного графа и топологически отсортируйте его конденсацию.

Граф задан во входном файле следующим образом: первая строка содержит числа N и M. Каждая из следующих M строк содержит описание ребра — два целых числа из диапазона от 1 до N — номера начала и конца ребра.

На первой строке выведите число K — количество компонент сильной связности в заданном графе. На следующей строке выведите N чисел — для каждой вершины выведите номер компоненты сильной связности, которой принадлежит эта вершина. Компоненты сильной связности должны быть занумерованы таким образом, чтобы для любого ребра номер компоненты сильной связности его начала не превышал номера компоненты сильной связности его конца.

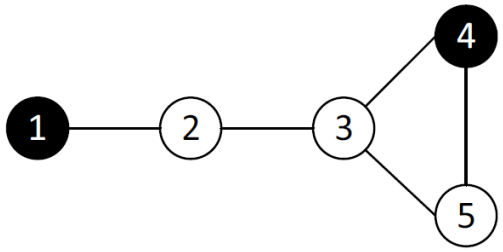
Input	Output
10 14 4 9 7 8 2 5 1 4 9 2 10 6 6 5 8 3 5 9 4 3 8 7 5 1 2 1 6 10	4 3 3 4 3 3 2 1 1 3 2

M. *Безопасность данных*

Телекоммуникационная сеть крупной IT-компании содержит n серверов, пронумерованных от 1 до n. Некоторые пары серверов соединены двусторонними каналами связи, всего в сети m каналов. Гарантируется, что сеть серверов устроена таким образом, что по каналам связи можно передавать данные с любого сервера на любой другой сервер, возможно с использованием одного или нескольких промежуточных серверов.

Множество серверов A называется отказоустойчивым, если при недоступности любого канала связи выполнено следующее условие: для любого не входящего в это множество сервера X существует способ передать данные по остальным каналам на сервер X хотя бы от одного сервера из множества A.

На иллюстрации ниже показан пример сети и отказоустойчивого множества из серверов с номерами 1 и 4. Данные на сервер 2 можно передать следующим образом. При недоступности канала между серверами 1 и 2 — с сервера 4, при недоступности канала между серверами 2 и 3 — с сервера 1. На серверы 3 и 5 при недоступности любого канала связи можно по другим каналам передать данные с сервера 4.



В рамках проекта группе разработчиков компании необходимо разместить свои данные в сети. Для повышения доступности данных и устойчивости к авариям разработчики хотят продублировать свои данные, разместив их одновременно на нескольких серверах, образующих отказоустойчивое множество. Чтобы минимизировать издержки, необходимо выбрать минимальное по количеству серверов отказоустойчивое множество. Кроме того, чтобы узнать, насколько гибко устроена сеть, необходимо подсчитать количество способов выбора такого множества, и поскольку это количество способов может быть большим, необходимо найти остаток от деления этого количества способов на число $10^9 + 7$.

Требуется написать программу, которая по заданному описанию сети определяет следующие числа: k — минимальное количество серверов в отказоустойчивом множестве серверов, c — остаток от деления количества способов выбора отказоустойчивого множества из k серверов на число $10^9 + 7$.

Первая строка входного файла содержит целые числа n и m — количество серверов и количество каналов связи соответственно ($2 \leqslant n \leqslant 200000, 1 \leqslant m \leqslant 200000$).

Следующие m строк содержат по два целых числа и описывают каналы связи между серверами. Каждый канал связи задается двумя целыми числами: номерами серверов, которые он соединяет.

Гарантируется, что любые два сервера соединены напрямую не более чем одним каналом связи, никакой канал не соединяет сервер сам с собой, и для любой пары серверов существует способ передачи данных с одного из них на другой, возможно с использованием одного или нескольких промежуточных серверов.

Выведите два целых числа, разделенных пробелом: k — минимальное число серверов в отказоустойчивом множестве серверов, c — количество способов выбора отказоустойчивого множества из k серверов, взятое по модулю $10^9 + 7$.

Input	Output
5 5 1 2 2 3 3 4 3 5 4 5	2 3

N. *Перегоны*

На некоторой железнодорожной ветке расположено N станций, которые последовательно пронумерованы числами от 1 до N. Известны расстояния между некоторыми станциями. Требуется точно вычислить длины всех перегонов между соседними станциями или указать, что это сделать невозможно (то есть приведенная информация является противоречивой или ее недостаточно).

Во входном файле записаны сначала числа N — количество станций ($2 \leqslant N \leqslant 10^5$) и E — количество пар станций, расстояния между которыми заданы ($0 \leqslant E \leqslant 10^5$). Далее идет E троек чисел, первые два числа каждой тройки задают номера станций (это числа из диапазона от 1 до N), а третье — расстояние между этими станциями (все эти расстояния заданы точно и выражаются вещественными неотрицательными числами не более чем с 3-я знаками после десятичной точки).

В случае, когда восстановить длины перегонов можно однозначно, в выходной файл выведите сначала число 1, а затем $N-1$ вещественное число. Первое из этих чисел должно соответствовать расстоянию от 1-й станции до 2-й, второе — от 2-й до 3-й, и так далее. Все числа должны быть выведены с точностью до 3-х знаков после десятичной точки.

Если приведенная информация о расстояниях между станциями является противоречивой или не позволяет однозначно точно восстановить длины перегонов, выведите в выходной файл одно число 2.

Input	Output
2 3 1 1 0 2 2 0 2 1 2.5	1 2.500
15 13 1 2 1 1 3 2 1 4 3 1 5 4 1 6 5 1 7 6 1 8 7 1 9 8 1 10 9 1 11 10 1 12 11 1 13 12 15 14 13	2

O. *Вершины на путях*

Дан неориентированный граф с N вершинами и M рёбрами, не содержащий петель и кратных рёбер. Кроме того дано несколько пар вершин (a_i, b_i). Для каждой пары требуется найти количество вершин c, таких, что существует путь из a_i в b_i, проходящий через вершину c.

Путь из a в b — это последовательность вершин, начинающаяся в a и заканчивающаяся в b, такая, что каждые две соседние вершины этой последовательности соединены ребром. Обратите внимание, что путь может проходить по одной и той же вершине более одного раза.

В первой строке через пробел записаны два целых числа n и m ($1 \leqslant n \leqslant 100; 0 \leqslant m \leqslant n(n-1)/2$) — количество вершин и рёбер графа. В следующих m строках записано по два целых числа u и v ($1 \leqslant u, v \leqslant n; u \neq v$) — номера вершин, которые соединяет описываемое ребро.

В следующей строке записано единственное целое число Q ($1 \leqslant Q \leqslant 10000$) — количество пар (a_i, b_i). В следующих Q строках описываются пары. В i-й из этих строк через пробел записаны целые числа a_i и b_i ($a_i \neq b_i; 1 \leqslant a_i, b_i \leqslant n$).

Для каждой описанной пары выведите на отдельной строке единственное число — количество вершин c, таких, что существует путь из a_i в b_i, проходящий через c.

Input	Output
4 3 1 2 2 3 3 4 2 1 2 1 3	4 4

P. *Циклы через рёбра*

Дан неориентированный граф с N вершинами и M ребрами, не содержащий петель и кратных рёбер. Найдите количество рёбер графа, через которые можно провести цикл.

В первой строке через пробел записаны два целых числа N и M ($1 \leqslant n \leqslant 100; 0 \leqslant m \leqslant n(n-1)/2$) — количество вершин и ребер графа. В следующих M строчках записано по два целых числа a и b ($1 \leqslant a, b \leqslant n; a \neq b$) — номера вершин, которые соединяет описываемое ребро.

Выведите единственное число — количество ребер графа, через которые можно провести цикл.

Input	Output
4 3 1 2 2 3 3 4	0
4 4 1 2 2 3 1 3 3 4	3

Q. *Важные вершины*

Дан неориентированный граф с N вершинами и M рёбрами, не содержащий петель и кратных рёбер. Так же вам дано несколько пар вершин (a_i, b_i). Для каждой пары требуется найти количество вершин c ($c \neq a_i; c \neq b_i$), таких, что любой путь из a_i в b_i, если такой существует, проходит через вершину c.

Путь из a в b — это последовательность вершин, начинающаяся в a и заканчивающаяся в b, такая, что каждые две соседние вершины этой последовательности соединены ребром. Обратите внимание, что путь может проходить по одной и той же вершине более одного раза.

В первой строке через пробел записаны два целых числа N и M — количество вершин и ребер графа, где $1 \leqslant N \leqslant 400; 0 \leqslant M \leqslant \dfrac{n(n-1)}{2}$.

В следующих M строках записано по два целых числа u и v ($1 \leqslant u, v \leqslant n; u \neq v$) — номера вершин, которые соединяет описываемое ребро.

В следующей строке записано единственное целое число Q ($1 \leqslant Q \leqslant 10000$) — количество пар (a_i, b_i). В следующих Q строках описываются пары. В i-й из этих строк через пробел записаны целые числа a_i и b_i ($a_i \neq b_i; 1 \leqslant a_i, b_i \leqslant n$).

Для каждой описанной пары выведите на отдельной строке единственное число — количество вершин c, таких, что любой путь из a_i в b_i, проходит через c.

Input	Output
4 3 1 2 2 3 3 4 2 1 2 1 3	0 1

R. *Поиск эйлерова пути*

В городе есть N площадей, соединённых улицами. При этом количество улиц не превышает 10^5 и существует не более трёх площадей, на которые выходит нечётное количество улиц. Для каждой улицы известна её длина. По каждой улице разрешено движение в обе стороны. В городе есть хотя бы одна улица. От каждой площади до любой другой можно дойти по улицам.

Почтальону требуется пройти хотя бы один раз по каждой улице. Почтальон хочет, чтобы длина его пути была минимальна. Он может начать движение на любой площади и закончить также на любой (в том числе и на начальной).

Помогите почтальону составить такой маршрут.

Сначала записано число N — количество площадей в городе ($2 \leqslant N \leqslant 1000$). Далее следуют N строк, задающих улицы. В i-ой из этих строк находится число m_i — количество улиц, выходящих из площади i. Далее следуют m_i пар натуральных чисел: в j-ой паре первое число — номер площади, в которую идет j-ая улица с i-ой площади, а второе число — длина этой улицы.

Между двумя площадями может быть несколько улиц, но не может быть улицы с площади на нее саму.

Все числа во входном файле не превосходят 10^5.

Если решение существует, то в первую строку выходного файла выведите одно число — количество улиц в искомом маршруте, а во вторую — номера площадей в порядке их посещения.

Если решения нет, выведите в выходной файл одно число -1.

Если решений несколько, выведите любое.

Input	Output
4 2 2 1 2 2 4 1 2 4 4 3 5 1 1 2 2 5 4 8 2 3 8 2 4	5 1 2 3 4 2 1