

Задача А. TopSort

Имя входного файла: `topsort.in`
Имя выходного файла: `topsort.out`
Ограничение по времени: 3 секунды
Ограничение по памяти: 64 мегабайта

Дан ориентированный невзвешенный граф. Необходимо его топологически отсортировать.

Формат входных данных

В первой строке входного файла даны два натуральных числа N и M ($1 \leq N \leq 10^5, 1 \leq M \leq 10^5$) — количество вершин и рёбер в графе соответственно. Далее в M строках перечислены рёбра графа. Каждое ребро задаётся парой чисел — номерами начальной и конечной вершин соответственно.

Формат выходных данных

Вывести любую топологическую сортировку графа в виде последовательности номеров вершин. Если граф невозможно топологически отсортировать, требуется вывести -1 .

Примеры

| <code>topsort.in</code> | <code>topsort.out</code> |
|---|--------------------------|
| 6 6 1 2 3 2 4 2 2 5 6 5 4 6 | 4 6 3 1 2 5 |
| 3 3 1 2 2 3 3 1 | -1 |

Задача В. Поиск цикла

Имя входного файла: `cycle.in`
Имя выходного файла: `cycle.out`
Ограничение по времени: 2 секунды
Ограничение по памяти: 256 мегабайта

Дан ориентированный невзвешенный граф. Необходимо определить есть ли в нём циклы, и если есть, то вывести любой из них.

Формат входных данных

В первой строке входного файла находятся два натуральных числа N и M ($1 \leq N \leq 100\,000$, $M \leq 100\,000$) — количество вершин и рёбер в графе соответственно. Далее в M строках перечислены рёбра графа. Каждое ребро задаётся парой чисел — номерами начальной и конечной вершин соответственно.

Формат выходных данных

Если в графе нет цикла, то вывести «NO», иначе — «YES» и затем перечислить все вершины в порядке обхода цикла.

Примеры

| <code>cycle.in</code> | <code>cycle.out</code> |
|-----------------------|------------------------|
| 2 2 1 2 2 1 | YES 2 1 |
| 2 2 1 2 1 2 | NO |

Задача С. Конденсация графа

Имя входного файла: `condense2.in`
Имя выходного файла: `condense2.out`
Ограничение по времени: 0.65 секунда
Ограничение по памяти: 256 мегабайт

Требуется найти количество рёбер в конденсации ориентированного графа. Примечание: конденсация графа не содержит кратных рёбер и петель.

Формат входных данных

Первая строка входного файла содержит два натуральных числа n и m — количество вершин и рёбер графа соответственно ($n \leq 10\,000, m \leq 100\,000$). Следующие m строк содержат описание рёбер, по одному на строке. Ребро номер i описывается двумя натуральными числами b_i, e_i — началом и концом ребра соответственно ($1 \leq b_i, e_i \leq n$). В графе могут присутствовать кратные рёбра и петли.

Формат выходных данных

Первая строка выходного файла должна содержать одно число — количество рёбер в конденсации графа.

Примеры

| <code>condense2.in</code> | <code>condense2.out</code> |
|---------------------------------|----------------------------|
| 4 4 2 1 3 2 2 3 4 3 | 2 |

Задача D. Мосты

Имя входного файла: `bridges.in`
Имя выходного файла: `bridges.out`
Ограничение по времени: 2 секунды
Ограничение по памяти: 256 мегабайт

Дан неориентированный граф, не обязательно связный, но не содержащий петель и кратных рёбер. Требуется найти все мосты в нём.

Формат входных данных

Первая строка входного файла содержит два натуральных числа n и m — количества вершин и рёбер графа соответственно ($1 \leq n \leq 20\,000$, $1 \leq m \leq 200\,000$).

Следующие m строк содержат описание рёбер по одному на строке. Ребро номер i описывается двумя натуральными числами b_i, e_i — номерами концов ребра ($1 \leq b_i, e_i \leq n$).

Формат выходных данных

Первая строка выходного файла должна содержать одно натуральное число b — количество мостов в заданном графе. На следующей строке выведите b целых чисел — номера рёбер, которые являются мостами, **в возрастающем порядке**. Рёбра нумеруются с единицы в том порядке, в котором они заданы во входном файле.

Примеры

| <code>bridges.in</code> | <code>bridges.out</code> |
|-------------------------|--------------------------|
| 6 7 | 1 |
| 1 2 | 3 |
| 2 3 | |
| 3 4 | |
| 1 3 | |
| 4 5 | |
| 4 6 | |
| 5 6 | |

Задача Е. Компоненты реберной двусвязности

Имя входного файла: `bicone.in`
 Имя выходного файла: `bicone.out`
 Ограничение по времени: 2 секунды
 Ограничение по памяти: 64 мегабайта

Компонентой реберной двусвязности графа $\langle V, E \rangle$ называется подмножество вершин $S \subset V$, такое что для любых различных u и v из этого множества существует не менее двух реберно не пересекающихся путей из u в v .

Дан неориентированный граф. Требуется выделить компоненты реберной двусвязности в нем.

Формат входных данных

Первая строка входного файла содержит два натуральных числа n и m — количества вершин и ребер графа соответственно ($1 \leq n \leq 20\,000$, $1 \leq m \leq 200\,000$).

Следующие m строк содержат описание ребер по одному на строке. Ребро номер i описывается двумя натуральными числами b_i, e_i — номерами концов ребра ($1 \leq b_i, e_i \leq n$).

Формат выходных данных

В первой строке выходного файла выведите целое число k — количество компонент реберной двусвязности графа. Во второй строке выведите n натуральных чисел a_1, a_2, \dots, a_n , не превосходящих k , где a_i — номер компоненты реберной двусвязности, которой принадлежит i -я вершина.

Примеры

| <code>bicone.in</code> | <code>bicone.out</code> |
|------------------------|-------------------------|
| 6 7 | 2 |
| 1 2 | 1 1 1 2 2 2 |
| 2 3 | |
| 3 1 | |
| 1 4 | |
| 4 5 | |
| 4 6 | |
| 5 6 | |

Задача F. Компоненты вершинной двусвязности

Имя входного файла: `biconv.in`
 Имя выходного файла: `biconv.out`
 Ограничение по времени: 2 секунды
 Ограничение по памяти: 64 мегабайта

Компонентой вершинной двусвязности графа $\langle V, E \rangle$ называется максимальный по включению подграф (состоящий из вершин и ребер), такой что любые два ребра из него лежат на вершинно простом цикле.

Дан неориентированный граф без петель. Требуется выделить компоненты вершинной двусвязности в нем.

Формат входных данных

Первая строка входного файла содержит два натуральных числа n и m — количества вершин и ребер графа соответственно ($1 \leq n \leq 20\,000$, $1 \leq m \leq 200\,000$).

Следующие m строк содержат описание ребер по одному на строке. Ребро номер i описывается двумя натуральными числами b_i, e_i — номерами концов ребра ($1 \leq b_i, e_i \leq n$).

Формат выходных данных

В первой строке выходного файла выведите целое число k — количество компонент вершинной двусвязности графа. Во второй строке выведите m натуральных чисел a_1, a_2, \dots, a_m , не превосходящих k , где a_i — номер компоненты вершинной двусвязности, которой принадлежит i -е ребро. Ребра нумеруются с единицы в том порядке, в котором они заданы во входном файле.

Примеры

| <code>biconv.in</code> | <code>biconv.out</code> |
|------------------------|-------------------------|
| 5 6 | 2 |
| 1 2 | 1 1 1 2 2 2 |
| 2 3 | |
| 3 1 | |
| 1 4 | |
| 4 5 | |
| 5 1 | |

Задача G. Магнитные подушки

Имя входного файла: `city.in`
Имя выходного файла: `city.out`
Ограничение по времени: 2 секунды
Ограничение по памяти: 256 мегабайт

Город будущего застроен небоскребами, для передвижения между которыми и парковки транспорта многие тройки небоскребов соединены треугольной подушкой из однополярных магнитов. Каждая подушка соединяет ровно 3 небоскреба и вид сверху на нее представляет собой треугольник, с вершинами в небоскребах. Это позволяет беспрепятственно передвигаться между соответствующими небоскребами. Подушки можно делать на разных уровнях, поэтому один небоскреб может быть соединен различными подушками с парами других, причем два небоскреба могут соединять несколько подушек (как с разными третьими небоскребами, так и с одинаковым). Например, возможны две подушки на разных уровнях между небоскребами 1, 2 и 3, и, кроме того, магнитная подушка между 1, 2, 5.

Система магнитных подушек организована так, что с их помощью можно добраться от одного небоскреба, до любого другого в этом городе (с одной подушки на другую можно перемещаться внутри небоскреба), но поддержание каждой из них требует больших затрат энергии.

Требуется написать программу, которая определит, какие из магнитных подушек нельзя удалять из подушечной системы города, так как удаление даже только этой подушки может привести к тому, что найдутся небоскребы из которых теперь нельзя добраться до некоторых других небоскребов, и жителям станет очень грустно.

Формат входных данных

В первой строке входного файла находятся числа N и M — количество небоскребов в городе и количество работающих магнитных подушек соответственно ($3 \leq N \leq 100000$, $1 \leq M \leq 100000$). В каждой из следующих M строк через пробел записаны три числа — номера небоскребов, соединенных подушкой. Небоскребы пронумерованы от 1 до N . Гарантируется, что имеющиеся магнитные подушки позволяют перемещаться от одного небоскреба до любого другого.

Формат выходных данных

Выведите в выходной файл сначала количество тех магнитных подушек, отключение которых невозможно без нарушения сообщения в городе, а потом их номера. Нумерация должна соответствовать тому порядку, в котором подушки перечислены во входном файле. Нумерация начинается с единицы.

Примеры

| <code>city.in</code> | <code>city.out</code> |
|---|-----------------------|
| 3 1 1 2 3 | 1 1 |
| 3 2 1 2 3 3 2 1 | 0 |
| 5 4 1 2 3 2 4 3 1 2 4 3 5 1 | 1 4 |

Задача Н. Сигнализация

| | |
|-------------------------|--------------|
| Имя входного файла: | alarm.in |
| Имя выходного файла: | alarm.out |
| Ограничение по времени: | 4 секунды |
| Ограничение по памяти: | 512 мегабайт |

Подземный бункер состоит из n комнат, соединённых $n - 1$ коридорами. Каждый коридор соединяет две различные комнаты и имеет определённую длину. Бункер устроен таким образом, что из любой комнаты i можно прийти в любую другую комнату j . Заметим, что существует единственный такой путь, не проходящий по одному и тому же коридору дважды. Сумма длин коридоров, составляющих этот путь, называется расстоянием между комнатами i и j и обозначается $\rho(i, j)$.

Каждая комната бункера оборудована звуковой сигнализацией, состоящей из сирены и датчика звука, который её включает. Сирена, включённая в комнате i , активирует датчик звука в каждой комнате, расстояние до которой не превосходит расстояние d_i , определяемое мощностью этой сирены. Другими словами, включение сирены в комнате i автоматически включает сирену во всех комнатах j , таких что $\rho(i, j) \leq d_i$. Эта сирена, в свою очередь, может вызвать автоматическое включение других сирен и так далее.

В случае возникновения чрезвычайной ситуации некоторые сирены необходимо включить вручную, после чего звук от них автоматически включит сирены в других комнатах. Правила безопасности предписывают выбор такого набора сирен для ручного включения, который в конце концов приведёт к автоматическому включению сирен во всех комнатах.

Требуется написать программу, которая определяет минимальное количество сирен в наборе, удовлетворяющем правилам безопасности.

Формат входных данных

Первая строка входных данных содержит единственное число n — количество комнат ($1 \leq n \leq 3000$).

Вторая строка содержит последовательность из n целых чисел d_i , i -е из них равно максимальному расстоянию, на котором расположенная в комнате i сирена активирует датчики ($0 \leq d_i \leq 10^9$).

Последующие $n - 1$ строк описывают коридоры бункера. В i -й из них находятся три целых числа: u_i, v_i, l_i , где u_i, v_i — номера различных комнат, соединённых коридором i , а l_i — длина этого коридора ($1 \leq u_i, v_i \leq n; 1 \leq l_i \leq 10^9$).

Формат выходных данных

Выходные данные должны состоять из единственного числа — минимального количества сирен, которые необходимо включить вручную.

Примеры

| alarm.in | alarm.out |
|---------------------|-----------|
| 10 | 3 |
| 1 2 2 2 6 3 4 5 4 3 | |
| 1 2 5 | |
| 2 3 1 | |
| 2 4 5 | |
| 4 5 2 | |
| 4 6 4 | |
| 4 7 3 | |
| 1 8 1 | |
| 8 9 5 | |
| 8 10 4 | |

Задача I. Диаграммы Юнга выходят в интернет

| | |
|-------------------------|--------------|
| Имя входного файла: | dfs.in |
| Имя выходного файла: | dfs.out |
| Ограничение по времени: | 4 секунды |
| Ограничение по памяти: | 256 мегабайт |

Рома и Сеня дали констест из 10 задач про диаграмму Юнга и теперь вынуждены скрываться от ДемидКомНадзора. Но они слишком любят диаграммы Юнга и хотят делиться новыми открытиями. Однако держаться вместе очень рискованно, так как в случае чего повяжут их обоих, поэтому они вынуждены общаться через Интернет. Но "обычный Интернет" полностью контролируется ДемидКомНадзором, поэтому они пользуются даркнетом.

В даркнете любое сообщение может проделать длинный запутанный путь до получателя через множество серверов. Более того, оно даже может проходить через один и тот же сервер несколько раз. За счет этого сообщение сложнее отследить.

Компьютер Ромы связан с сервером 1, а компьютер Сени — с сервером n .

ДемидКомНадзор хочет перехватить Ромино сообщение с диаграммой Юнга. Для этого ему необходимо взломать такой сервер, что сообщение, посланное Ромой, по любому пути к Сене пройдет через этот сервер **ровно один** раз.

Найдите все подходящие сервера.

Формат входных данных

В первой строке файла дано количество тестовых примеров t ($1 \leq t \leq 500$).

Каждый тестовый пример выглядит так: в первой строке даны два числа: n и m ($2 \leq n \leq 5 \cdot 10^5$, $0 \leq m \leq 10^6$), число серверов и число прямых соединений между серверами.

В каждой из последующих m строк содержится упорядоченная пара чисел a и b ($1 \leq a, b \leq n$), это означает, что с сервера a можно переслать сообщение напрямую на сервер b .

Гарантируется, что эти упорядоченные пары не повторяются внутри одного тестового примера.

Так же гарантируется, что и сумма по n , и сумма по m по всем тестовым примерам не превосходит 10^6 .

Формат выходных данных

Для каждого тестового примера требуется вывести две строки: в первой число подходящих серверов, а во второй — номера этих серверов в порядке их следования на пути от a до b через пробел.

Примеры

| dfs.in | dfs.out |
|--------|---------|
| 4 | 4 |
| 4 3 | 1 3 2 4 |
| 2 4 | 0 |
| 1 3 | 0 |
| 3 2 | 2 |
| 2 2 | 1 4 |
| 1 2 | |
| 2 1 | |
| 3 1 | |
| 2 3 | |
| 4 4 | |
| 1 2 | |
| 2 4 | |
| 3 4 | |
| 1 3 | |