День девятый (20.02.2011 г.)
Контест Гольдштейна Виталия Борисовича

Об авторе...

Гольдштейн Виталий Борисович, ассистент и аспирант Московского Физико-Технического Института. Член научного комитета всероссийской олимпиады школьников по информатике и сборов по подготовке к международной олимпиаде школьников. Стажер компании Google (зима-весна 2008). Завуч Летней Компьютерной Школы (август, 2009) и Летней Компьютерной Школы (зима, 2009-2010). Член жюри Московского и Саратовского четвертьфинала ACM ICPC, член жюри полуфинала NEERC ACM ICPC. Разработчик компании Яндекс. Лектор курса “Алгоритмы и структуры данных” базовой кафедры Яндекса “Анализ данных” на факультете Инноваций и высоких технологий в МФТИ. Член тренерской группы проекта “Яндекс-тренировки” в Москве.

Основные достижения:

- серебряная медаль чемпионата мира по программированию среди студентов (ACM ICPC Tokyo 2007);
- участник финалов TopCoder Open, TopCoder Collegiate Open, Global Google Code Jam, Google Code Jam Europe;
- тренер призеров (4-е место, золотая медаль) чемпионата мира по спортивному программированию ACM-ICPC World Finals 2009, Швеция, Стокгольм.

Теоретический материал. Применение обхода в глубину

Обход в глубину

Описание алгоритма

Обход в глубину — это рекурсивный алгоритм обхода графа. Самый простой способ описать этот алгоритм — это привести его реализацию:

```
def dfs(v):
    used[v] = True
    for u in incidents[v]:
        if not used[u]: dfs(u)
```

Кратко можно сказать, что этот алгоритм вызывает себя рекурсивно от смежных и еще непосещенных вершин. Несмотря на свою простоту, этот алгоритм имеет огромное количество применений, благодаря порядку, в котором обходятся вершины. Функция обхода в глубину для каждой вершины будет вызвана не более одного раза, поэтому суммарное время работы алгоритма равно сумме степеней всех вершин, то есть $O(E)$

Цвета вершин в процессе алгоритма

В процессе алгоритма каждой вершине соответствует цвет. Вершины, которые еще не были посещены алгоритмом, называют *белыми*. Вершины, которые еще обрабатываются алгоритмом (то есть алгоритм зашел в функцию, но еще не закончил), называют *серыми*. Полностью обработанные вершины, называют *черными*.

```
def dfs(v):
    color[v] = GRAY
    for u in incidents[v]:
        if not used[u]: dfs(u)
    color[v] = BLACK
```

В любой момент работы алгоритма серые вершины образуют некоторый путь от стартовой вершины до текущей. В дальнейшем мы будем использовать понятие цвета вершины в рассуждениях

Время начала и окончания обработки вершин

Для вершин так же можно определить время начала и окончания обработки алгоритмом. Будем считать, что время идет непрерывно вперед в процессе работы алгоритма. Нам не так важны числовые значения этих времен. В дальнейших рассуждениях мы будем использовать только соотношение этих времен.

```
def dfs(v):
    time_in[v] = cur_time++
    for u in incidents[v]:
        if not used[u]: dfs(u)
    time_out[v] = cur_time++
```

С первого взгляда кажется, что алгоритм закончит обработку вершины после того как обработает все достижимые из нее вершины. Однако это не так. Путь до некоторых вершин может быть загорожен серыми вершинами. Однако верна лемма о белом пути. Пусть произошел вызов алгоритма от некоторой вершины v, тогда все вершины, достижимые из v по белым вершинам будут обработаны до окончания обработки v.

Дерево обхода в глубину

Деревом обхода в глубину называют множество ребер, по которым прошел обход в глубину. То есть те ребра, которые при просмотре их обходом в глубину вели из текущей вершины в белую. Это понятие обычно применяют для неориентированных графов.

Классификация ребер

Ребра относительно любого остовного дерева в неориентированном графе классифицируют следующим образом:

1. Ребра дерева.
2. Перекрестные ребра — ребра, соединяющие разные поддеревья.
3. Возвратные ребра — ребра, ведущие от вершины к ее предку.

В ориентированном графе можно классифицировать ребра похожим образом.

1. Ребра дерева — из серой в белую вершину.
2. Перекрестные ребра — из серой в черную вершину.
3. Возвратные ребра — из серой в серую вершину.

Стоит обратить внимание, что при обходе в глубину в неориентированном графе перекрестных ребер нет.

Стандартные применения. Быстрый обзор.

Компоненты связности

Компонента связности в неориентированном графе — наибольшее по включению множество попарно достижимых вершин. При запуске обхода

в глубину из вершины v в неориентированном графе будут помечены все вершины компоненты связности.

Топологическая сортировка

Топологическая сортировка ациклического ориентированного графа — это упорядочивание вершин графа так, чтобы ребра были направлены от меньшего номера к большему. Топологическая сортировка эквивалентна сортировке вершин по возрастанию времени окончания обработки.

Мосты и точки сочленения

Мост в неориентированном графе — ребро, при удалении которого увеличивается количество компонент связности. Точка сочленения в неориентированном графе — вершина, при удалении которой увеличивается количество компонент связности.

Определим функцию $uptime(v)$ как наименьшее время входа вершины, в которую можно попасть из v путем спуска по дереву обхода в глубину и последующего подъема по одному возвратному ребру. Эту функцию можно вычислять в процессе обхода в глубину: $uptime(v) = min(tin[v], \min\limits_{u \in back(v)}(tin[u]), \min\limits_{u \in tree(v)}(uptime[u]))$, где $back(v)$ — множество вершин, в которые ведут возвратные ребра из v, а $tree(v)$ — множество потомков вершины v в дереве обхода в глубину.

Ребро (u, v) является мостом тогда и только тогда, когда оно ребро дерева (u предок v) и $uptime(v) > tin[u]$.

Вершина v является точкой сочленения в одном из двух случаев:

1. v — корень дерева и у него хотя бы 2 потомка.
2. Существует потомок u такой, что $uptime(v) \geq tin[v]$.

Компоненты вершинной и реберной двусвязности

- Компонентой реберной двусвязности называют наибольшее по включению множество вершин, такое что между любой парой есть два непересекающихся по ребрам пути.
- Компонентой вершинной двусвязности называют наибольшее по включению множество ребер, такое что в порожденном подграфе между любой парой вершин есть два непересекающихся по вершинам пути.

Компоненты сильной связности

В ориентированном графе понятие компоненты связности не определено. Однако абсолютно так же можно сформулировать определение компонент сильной связности.

Компонента сильной связности в ориентированном графе — наибольшее по включению множество попарно достижимых вершин. Для нахождения необходимо запустить обход в глубину из каждой не помеченной вершины. Упорядочить вершины по убыванию времени выхода. После чего запускать обход в глубину из вершин в полученном порядке. Каждый обход пометит вершины из очередной компоненты сильной связности. Кроме этого, компоненты сильной связности будут выписаны в порядке топологической сортировки.

Конденсация графа

Конденсацией графа G называют граф H, в котором некоторое множество вершин графа G объединяются в одну вершину графа H. В графе H вершины соединены ребром, если соединена хотя бы одна пара вершин из соответствующих множеств графа G.

Конденсация графа обычно лишена некоторого свойства. Так, например, конденсация компонент сильной связности является ациклическим графом. А конденсация компонент реберной двусвязности является деревом.

Дополнительные применения

Нахождение цикла

Цикл в графе существует тогда и только тогда, когда в процессе обхода в глубину было обнаружено возвратное ребро. Если ребро обнаружено, то цикл найден, а значит он существует. Несколько более сложно заметить, что при наличии цикла, он обязательно будет обнаружен. Рассмотрим произвольный цикл. Пусть, не ограничивая общности, первой мы посетили вершину u, тогда v — это предыдущая вершина этого цикла. По лемме о белом пути вершина v будет посещена до того, как закончит свою обработку вершина u. А следовательно, ребро (v, u) будет возвратным.

Отношение предка в дереве

Задача состоит в том, чтобы при некотором предподсчете за $O(1)$ проверять является ли одна вершина предком другой в дереве. Запустим обход в глубину из корня дерева и запомним для каждой вершины время входа и время выхода. Совершенно очевидно, что все потомки вершины p будут удовлетворять следующему свойству: $tin[p] \leq tin[v] \ \ \& \ \ tout[v] \leq tout[p]$. Времена входа и выхода в обходе в глубину образовывают правильную скобочную последовательность, поэтому достаточно проверить $tin[p] \leq tin[v] \leq tout[p]$.

Нахождение вершин и ребер, которые могут быть на пути от s ***к*** t

Задача состоит в том, чтобы найти вершины и ребра, которые в принципе могут находится на некотором (возможно не простом) пути от s к t. Для этого применяется стандартный прием поиска с двух концов. Найдем все вершины достижимые из s, после чего в транспонированном графе G^T найдем все вершины достижимые из t (то есть вершины, из которых достижима t в графе G).

- Вершина v может лежать на пути, если она достижима из s и из нее достижима t.
- Ребро (u, v) может лежать на пути, если u достижима из s и из v достижима t.

Проверка на сильносвязность

Задача состоит в том, чтобы проверить, является ли граф сильносвязным. Мы уже умеем выделять компоненты сильной связности, но для проверки на сильносвязность можно применить более простой алгоритм. Выберем произвольную вершину v. Есть два необходимых условия сильносвязности:

- из v достижимы все вершины;
- из всех вершин достижима вершина v.

Оказывается, что этих условий достаточно, так как из любой вершины можно будет добраться в любую другую вершину через v.

Проверка на "слабо"-связность

Назовем ориентированный граф "слабо"-связным, если для любой пары вершины u и v верно, что либо из u достижимо v, либо из v достижимо u.

Для решения этой задачи найдем конденсацию графа. Конденсация графа является ациклическим графом. При этом граф слабосвязан тогда и только тогда, когда слабосвязана конденсация. Очевидно, что ациклический ориентированный граф слабосвязан, если он представляет из себя ориентированную цепь.

SAT-2

Задача SAT-2 — это задача решения уравнения $\underset{t\in[1,m]}{\&}(x_{f_t}^{\alpha_t} \vee x_{s_t}^{\beta_t}) = 1$ в B_2. В этом уравнение n переменных.

Для решения этой задачи в первую очередь преобразуем формулу в другой вид $\&\limits_{t\in[1,m]}(\neg x_{f_t}^{\alpha_t} \to x_{s_t}^{\beta_t}) = 1$. А так же добавим симметричное утверждение $\&\limits_{t\in[1,m]}(\neg x_{s_t}^{\beta_t} \to x_{f_t}^{\alpha_t}) = 1$.

В этом виде очевидно, что нам необходимо выполнить $2m$ следствий. Построим граф из $2n$ вершин и $2m$ ребер. Каждой переменной будет соответствовать две вершины x_i и $\neg x_i$. Соединим ребром вершины, которые присутствуют в уравнении в одном следствии.

Теперь наша задача стоит в том, чтобы расставить около вершин 0 и 1 так, чтобы из 1 не было достижимо 0. Кроме этого, x_i и $\neg x_i$ должны соответствовать противоположные значения.

Выделим в полученном графе компоненты сильной связности. Каждая компонента должна быть отмечена одним числом. Если для некоторого i x_i и $\neg x_i$ попали в одну компоненту, то решения не существует. В противном случае все компоненты сильной связности распадутся на пары симметричных, в которых все переменные противоположны.

Будем двигаться в порядке, противоположном топологической сортировке. Если очередная компонента не отмечена числом, то отметим ее 1, а симметричную 0. Это можно делать, так как симметричная компонента не будет достижима только из отмеченных 0 компонент.

Задачи и разборы

Задача A. Цветные волшебники – 2

Имя входного файла:	`a.in`
Имя выходного файла:	`a.out`
Ограничение по времени:	2 с
Ограничение по памяти:	64 Мб

Сказочная страна представляет собой множество городов, соединенных дорогами с двухсторонним движением. Причем из любого города страны можно добраться в любой другой город либо непосредственно, либо через другие города. Известно, что в сказочной стране не существует дорог, соединяющих город сам с собой и между любыми двумя разными городами, существует не более одной дороги.

В сказочной стране живут желтый и синий волшебники. Желтый волшебник, пройдя по дороге, перекрашивает ее в желтый цвет, синий — в синий. Как известно, при наложении желтой краски на синюю, либо синей краски на желтую, краски смешиваются и превращаются в краску зеленого цвета, который является самым нелюбимым цветом обоих волшебников.

В этом году в столице страны (городе f) проводится конференция волшебников. Поэтому желтый и синий волшебники хотят узнать, какое минимальное количество дорог им придется перекрасить в зеленый цвет, чтобы добраться в столицу. Изначально все дороги не покрашены.

Начальное положение желтого и синего волшебников заранее не известно. Поэтому необходимо решить данную задачу для k возможных случаев их начальных расположений.

Формат входного файла

Первая строка входного файла содержит целые числа: n ($1 \le n \le 100\,000$) и m ($1 \le m \le 500\,000$) — количество городов и дорог в волшебной стране соответственно. Третья строка содержит одно целое число f ($1 \le f \le n$) — номер города, являющегося столицей сказочной страны. В следующих m строках находится описание дорог страны. В этих m строк записано по два целых числа a_i и b_i, означающих, что существует дорога, соединяющая города a_i и b_i. Следующая строка содержит целое число k ($1 \le k \le 100\,000$) — количество возможных начальных расположений волшебников. Далее следуют k строк, каждая из которых содержит два целых числа — номера городов, в которых изначально находится желтый и синий волшебники соответственно.

Формат выходного файла

Для каждого из k случаев ваша программа должна вывести в выходной минимальное количество дорог, которое придется покрасить в зеленый цвет волшебникам для того, чтобы добраться в столицу.

Пример

a.in	a.out
6 6 1 1 2 2 3 3 4 4 2 4 5 3 6 2 5 6 6 6	1 2

Разбор задачи A. Цветные волшебники – 2

Для решения задачи найдем конденсацию графа по компонентам реберной двусвязности. В любой компоненте двусвязности два волшебника с легкостью смогут разойтись, не пересекаясь по ребрам. Мосты — единственные ребра, которые они с неизбежностью должны вместе пройти. Конденсация является деревом, состоящим только из мостов. Требуемое количество ребер — это количество общих ребер в конденсации на пути к столице. Для нахождения количества общих ребер в пути, можно подвесить дерево за компоненту двусвязности с столицей. После этого количество общих ребер будет равно глубине наименьшего общего предка. Нужно использовать любой алгоритм нахождения минимального общего предка за $O(\log N)$.

Задача B. Граф операций

Имя входного файла:	`b.in`
Имя выходного файла:	`b.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Вам задан неориентированный граф. Каждому ребру (u, v) в этом графе приписана некоторая бинарная операция и некоторое число c. Число c может иметь значение 0 или 1, а операции могут быть такие: логическое умножение (AND), логическое сложение (OR) и сложение по модулю два (XOR). Ваша задача выяснить, можно ли присвоить каждой вершине u число 0 или 1 (обозначим это значение как $A(u)$) таким образом, чтобы для каждого ребра (u, v) результат приписанной ему бинарной операции над величинами $A(u)$ и $A(v)$ был равен написанному на нем числу c.

Правила вычисления указанных операций такие:

- `(0 AND 1) = (1 AND 0) = (0 AND 0) = 0, (1 AND 1) = 1`
- `(0 OR 1) = (1 OR 0) = (1 OR 1) = 1, (0 OR 0) = 0`
- `(0 XOR 1) = (1 XOR 0) = 1, (1 XOR 1) = (0 XOR 0) = 0`

Формат входного файла

В первой строке записано два целых числа n и m ($1 \le n \le 1000$, $0 \le m \le 300000$) — количество вершин и ребер в графе. Следующие m строк содержат описание ребер. Ребро задается номерами соединяемых

вершин u_i, v_i ($1 \le u_i, v_i \le n, u_i \ne v_i$), числом c_i и названием операции (AND, OR или XOR). Никакое ребро не дано более одного раза.

Формат выходного файла

Выведите "YES", если можно найти требуемый набор значений для вершин, и "NO" — в противном случае.

Примеры

b.in	b.out
3 3 1 2 1 OR 2 3 1 AND 1 3 1 XOR	YES
3 2 1 2 1 AND 2 3 0 OR	NO

Разбор задачи B. Граф операций

В задаче каждое ребро представляет собой некоторое уравнение из двух переменных. Любое логическое выражение из двух переменных можно привести к нормальной дизъюнктивной форме. После такого приведения задача сводится к задаче SAT-2. Описание решения задачи SAT-2 смотрите в лекции.

Задача C. Регулярное паросочетание

Имя входного файла:	c.in
Имя выходного файла:	c.out
Ограничение по времени:	3 с
Ограничение по памяти:	256 Мб

Максим с детства увлекается теорией графов. Сейчас его интересует все, что связано с двудольными графами. Двудольным графом называют граф, вершины которого можно разбить на две части так, что не существует ребер, соединяющих вершины из одной части. В одной замечательной книжке Максим прочитал, что в регулярном двудольном графе существует совершенное паросочетание. В этой же книге было написано, что регулярный граф — это граф, у которого степени всех вершин одинаковые, а совершенное паросочетание — разбиение **всех** вершин графа на пары соединенных ребром вершин. Однако, в этой книге ничего не было написано

как найти это паросочетание. Всю ночь Максим провел в поиске алгоритма, однако нашел только маленькую сноску в огромной книге, что эта задача легко решается для двудольных регулярных графов с степенью 2^k. Помогите ему найти совершенное паросочетание.

Формат входного файла

В первой строке задано число n ($1 \le n \le 50000$) — количество вершин в каждой доле графа. Во второй строке записана степень каждой вершины d ($d = 2^k$, где $0 \le k \le 5$). В последующих n строках для каждой вершины первой доли записано по d чисел – номера смежных вершин второй доли. Вершины первой и второй доли нумеруются независимо от 1 до n. В графе могут быть кратные ребра. Гарантируется, что степени всех вершин, как первой, так и второй доли, равны d. В графе допускаются кратные ребра, то есть пару вершин могут соединять более одного ребра.

Формат выходного файла

Выведите ровно n чисел — для каждой вершины первой доли номер вершины второй доли, с которой она будет объединена в совершенном паросочетании. В выводе должна быть некоторая перестановка чисел от 1 до n. Если решений несколько, выведите любое.

Примеры

c.in	c.out
2 1 1 2	1 2
2 2 2 1 1 2	1 2
7 4 5 4 2 6 3 7 5 7 1 3 1 6 2 3 5 1 3 7 6 1 4 6 2 4 2 7 4 5	6 5 3 1 7 2 4

Разбор задачи C. Регулярное паросочетание

Задача имеет неожиданное отношение к эйлерову циклу. Степень каждой вершины равна степени двойки 2^d. Если $d = 0$, то заданный граф и является паросочетанием. Пусть $d > 0$, тогда задачу можно свести к $d - 1$ следующим образом: найдем эйлеров цикл в графе, после чего каждое второе ребро удалим из графа. Полученный граф так же будет являться двудольным регулярным графом, степень каждой вершины которого является степенью двойки. Таким образом за d шагов мы придем к регулярному графу со степенями 1. Если первоначальное количество ребер E, то суммарное время работы $O(E)$, $E + \frac{E}{2} + \frac{E}{4} + ... + \frac{E}{2^d} \leq 2E$.

Задача D. Авиаперелеты

Имя входного файла:	`d.in`
Имя выходного файла:	`d.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Главного конструктора Петю попросили разработать новую модель самолета для компании “Air Бубундия”. Оказалось, что самая сложная часть заключается в подборе оптимального размера топливного бака.

Главный картограф “Air Бубундия” Вася составил подробную карту Бубундии. На этой карте он отметил расход топлива для перелета между каждой парой городов.

Петя хочет сделать размер бака минимально возможным, для которого самолет сможет долететь от любого города в любой другой (возможно, с дозаправками в пути).

Формат входного файла

Первая строка входного файла содержит натуральное число n ($1 \leq n \leq 1000$) — число городов в Бубундии. Далее идут n строк по n чисел каждая. j-ое число в i-ой строке равно расходу топлива при перелете из i-ого города в j-ый. Все числа не меньше нуля и меньше 10^9. Гарантируется, что для любого i в i-ой строчке i-ое число равно нулю.

Формат выходного файла

Первая строка выходного файла должна содержать одно число — оптимальный размер бака.

Пример

d.in	d.out
4 0 10 12 16 11 0 8 9 10 13 0 22 13 10 17 0	10

Разбор задачи D. Авиаперелеты

Задача “Авиаперелеты” — это пример задачи на применение двух стандартных алгоритмов. Представьте, что вы контролирующая организация и хотите проверить, можно ли при фиксированном размере бака добраться между любыми двумя городами. Рассмотрим граф, в котором города будут соединены ребром, когда бака хватает, чтобы совершить прямой рейс между ними. Тогда размер бака подходящий, если граф сильносвязан. В лекции описано, как проверить граф на сильносвязность. Теперь можно воспользоваться бинарным поиском по ответу. Выбрав некоторый размер бака, проверим подходит ли такой размер бака. Если размер подходит, то будем искать ответ с меньшим размером бака. Если же граф окажется не сильносвязным, то будем искать ответ с большим баком.

Задача E. Противопожарная безопасность

Имя входного файла:	`e.in`
Имя выходного файла:	`e.out`
Ограничение по времени:	2 с
Ограничение по памяти:	64 Мб

В городе Зеленоград n домов. Некоторые из них соединены дорогами с односторонним движением.

В последнее время в Зеленограде участились случаи пожаров. В связи с этим жители решили построить в городе несколько пожарных станций. Но возникла проблема — едущая по вызову пожарная машина, конечно, может игнорировать направление движения текущей дороги, однако, возвращающаяся с задания машина обязана следовать правилам дорожного движения (жители Зеленограда свято чтут эти правила!).

Ясно, что где бы ни оказалась пожарная машина, у неё должна быть возможность вернуться на ту пожарную станцию, с которой выехала. Но строительство станций стоит больших денег, поэтому на совете города

было решено построить минимальное количество станций таким образом, чтобы это условие выполнялось. Кроме того, для экономии было решено строить станции в виде пристроек к уже существующим домам.

Ваша задача — написать программу, рассчитывающую оптимальное положение станций.

Формат входного файла

В первой строке входного файла задано число n ($1 \le n \le 3\,000$). Во второй строке записано количество дорог m ($1 \le m \le 100\,000$). Далее следует описание дорог в формате a_i b_i, означающее, что по i-й дороге разрешается движение автотранспорта от дома a_i к дому b_i ($1 \le a_i, b_i \le n$).

Формат выходного файла

В первой строке выведите минимальное количество пожарных станций K, которые необходимо построить. Во второй строке выведите K чисел в произвольном порядке — дома, к которым необходимо пристроить станции. Если оптимальных решений несколько, выведите любое.

Пример

e.in	e.out
5 7 1 2 2 3 3 1 2 1 2 3 3 4 2 5	2 4 5

Разбор задачи E. Противопожарная безопасность

Требуется найти множество вершин, которые достижимы из всех остальных. Для этого нужно найти конденсацию графа. Из каждой компоненты сильной связности, из которой не выходит ни одного ребра, нужно выбрать по одной вершине. Тогда из любой вершины перемещаясь по произвольному ребру конденсации мы рано или поздно придем в такую компоненту, из которой не выходит ребер. Минимальность объясняется тем, что из вершин выбранных компонент не достижимы другие вершины.

Задача F. Островные государства

Имя входного файла:	`f.in`
Имя выходного файла:	`f.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Суровые феодальные времена переживала некогда великая островная страна Байтландия. За главенство над всем островом борются два самых сильных барона. Таким образом, каждый город страны контролируется одним из правителей. Как водится издревле, некоторые из городов соединены двусторонними дорогами. Бароны очень не любят друг друга и стараются делать как можно больше пакостей. В частности, теперь для того, чтобы пройти по дороге, соединяющей города различных правителей, надо заплатить пошлину — один байтландский рубль. Кроме этого, за выезд из городов с четными номерами берется удвоенная пошлина.

Программист Вася живет в городе номер 1. С наступлением лета он собирается съездить в город N на Всебайтландское сборище программистов. Разумеется, он хочет затратить при этом как можно меньше денег и помочь ему здесь, как обычно, предлагается Вам.

Формат входного файла

В первой строке входного файла записано два числа N и M ($1 \le N, M \le 100\,000$) — количество городов и количество дорог соответственно.

В следующий строке содержится информация о городах — N чисел 1 или 2 — какому из баронов принадлежит соответствующий город.

В последних M строках записаны пары $1 \le a, b \le N$, $a \ne b$. Каждая пара означает наличие дороги из города a в город b. По дорогам Байтландии можно двигаться в любом направлении.

Формат выходного файла

Если искомого пути не существует, выведите единственное слово `impossible`. В противном случае, в первой строке напишите минимальную стоимость и количество посещенных городов, а во вторую выведите эти города в порядке посещения. Если минимальных путей несколько, выведите любой.

Пример

f.in	f.out
7 8 1 1 1 1 2 2 1 1 2 2 5 2 3 5 4 4 3 4 7 1 6 6 7	0 5 1 2 3 4 7
5 5 1 1 1 2 1 1 2 2 3 3 4 4 5 2 4	3 5 1 2 3 4 5

Разбор задачи F. Островные государства – 2

Требуется найти кратчайший путь в графе, веса ребер которого 0, 1 или 2. Для этого можно воспользоваться модифицированным обходом в ширину. В середину каждого ребра веса 2 поставим фиктивную вершину. Тем самым одно ребро веса 2 превратится в два ребра веса 1. Тем самым мы свели задачу к поиску кратчайшего пути в 0-1 графе. Самым простым способом решить эту задачу — при каждом добавлении вершины в очередь обхода в ширину, запускать обход в глубину по 0 ребрам графа. Все новые вершины, которые мы смогли посетить, добавлять в очередь. Естественно каждую вершину должен посетить либо обход в глубину, либо обход в ширину, причем ровно один раз.

Задача G. Король

Имя входного файла:	`g.in`
Имя выходного файла:	`g.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

В Тридесятом царстве, Тридевятом государстве жил-был король. И было у короля n сыновей. В Тридесятом царстве жили n прекрасных девушек, и король знал, какие девушки нравятся каждому сыну (поскольку сыновья были молодыми и безшабашными, то им могли нравиться несколько девушек одновременно).

Однажды король приказал своему советнику подобрать для каждого сына прекрасную девушку, на которой тот сможет жениться. Советник выполнил приказ и подобрал для каждого сына для женитьбы прекрасную девушку, которая ему нравилась. Разумеется, каждая девушка может выйти замуж только за одного из сыновей.

Посмотрев на список невест, король сказал: “Мне нравится этот список, но я хочу знать для каждого сына список всех девушек, на которых он может жениться. Разумеется, при этом все сыновья также должны иметь возможность жениться на девушках, которые им нравятся”.

Эта задача оказалась для советника слишком сложной. Помогите ему избежать казни, решив ее.

Формат входного файла

Первая строка входного файла содержит число n — количество сыновей ($1 \le n \le 2\,000$). Следующие n строк содержат списки прекрасных девушек, которые нравятся сыновьям. В начале идет k_i — количество девушек, которые нравятся i-му сыну. Затем идут k_i чисел — номера девушек. Сумма k_i не превышает $200\,000$.

Последняя строка входного файла содержит список, составленный советником — n различных чисел от 1 до n: для каждого сына — номер прекрасной девушки, на которой он может жениться. Гарантируется, что список корректен, то есть каждому сыну нравится выбранная для него девушка.

Формат выходного файла

Выходной файл должен содержать n строк. Для каждого сына выведите l_i — количество различных девушек, на которых он может жениться. После этого выведите l_i чисел — номера девушек в произвольном порядке.

Пример

g.in	g.out
4 2 1 2 2 1 2 2 2 3 2 3 4 1 2 3 4	2 1 2 2 1 2 1 3 1 4

Разбор задачи G. Король

Требуется при заданном максимальном паросочетании найти все ребра, которые могут принадлежать какому-либо максимальному паросочетанию. Так как заданное паросочетание максимально, то не существует чередующегося удлинняющего пути. Ориентируем все ребра, входящие в паросочетание, в направлении от первой ко второй доле. А ребра, входящие в паросочетание из второй доли в первую. Рассмотрим произвольное ребро, не входящее в заданное максимальное паросочетание, ведущее из u первой доли в v второй доли. Оно может войти в максимальное паросочетание, если мы найдем чередующуюся цепочку от v к u. То есть фактически, ребро (u, v) находится в некотором цикле. Иначе говоря, u и v находятся в одной компоненте сильной связности. Таким образом, нужно выделить компоненты сильной связности. Все ребра, которые соединяют вершины одной компоненты и будут ответом на поставленный вопрос.

Задача H. Каток

Имя входного файла:	`h.in`
Имя выходного файла:	`h.out`
Ограничение по времени:	2 с
Ограничение по памяти:	64 Мб

Бизнесмен Николай очень любит кататься на коньках. Каждый год 31-го декабря он собирает компанию друзей, арендует каток и весело встречает там Новый Год. Недавно Николаю в голову пришла мысль, что гораздо приятнее было бы иметь собственный каток и бесплатно кататься на нем в любое удобное время. Недолго думая, он решил реализовать эту идею, то есть построить для себя каток, и нанял архитектора Дмитрия.

Архитектор узнал, что любимое число Николая — число P, а любимый способ катания на коньках — катание вдоль бортика. Чтобы угодить клиенту, Дмитрий решил построить такой каток, на котором Николай, прокатив-

шись вдоль всех бортиков, проедет ровно P сотен метров. Ему уже почти удалось нарисовать план такого катка, но Николай неожиданно предъявил новое требование — каток должен иметь площадь ровно S десятков тысяч квадратных метров (на меньшей площади не поместятся все друзья Николая, а большая слишком дорого обойдется).

Если некоторая площадь катка граничит с площадью, не принадлежащей катку, то между ними обязательно необходимо построить бортик. Таким образом, запрещается устанавливать бортики в произвольном месте катка, а разрешается лишь на его границе.

Новый каток будет построен недалеко от коттеджа Николая. В данный момент под этот проект выделили квадратный участок со стороной 400 метров. Проект катка должен быть таким, чтобы все углы катка были прямые, длины бортиков кратны 100 метрам, а сами бортики параллельны сторонам участка, на котором будет вестись строительство.

Помогите Дмитрию придумать план катка, который удовлетворял бы всем требованиям.

Формат входного файла

В единственной строке входного файла записаны два целых числа P и S ($1 \le P \le 40$, $1 \le S \le 16$) — периметр и площадь будущего катка, соответственно.

Формат выходного файла

В выходной файл выведите “Impossible”, если спроектировать каток, удовлетворяющий указанным требованиям невозможно.

В противном случае, выведите 4 строки по 4 символа в каждой — план нового катка. Каждый символ будет описанием квадрата 100×100 метров участка, на котором будет вестись строительство. Если соответствующий квадрат участка находится внутри катка, то выведите “*”, иначе — “.”.

Примеры

h.in	h.out
18 8	**** *.*. **.. ....
3 1	Impossible

Разбор задачи H. Каток

Требуется построить конфигурацию катка в квадрате 4×4 с заданной площадью и периметром. Размер квадарата достаточно маленький, поэтому можно перебрать все возможные конфигурации катка, которых 2^{16}. После этого необходимо проверить связность графа и соответствие количества клеток катка и требуемой площади. Для подсчета периметра достаточно посчитать количество полосок сетки, которые соединяют клетку катка и внешности катка. Если периметр будет равен требуемому, то выедем искомую конфигурацию.

Задача I. Предок

Имя входного файла:	`i.in`
Имя выходного файла:	`i.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Напишите программу, которая для двух вершин дерева определяет, является ли одна из них предком другой.

Формат входного файла

Первая строка входного файла содержит натуральное число n ($1 \le n \le 100\,000$) — количество вершин в дереве. Во второй строке находится n чисел, i-ое из которых определяет номер непосредственного родителя вершины с номером i. Если это число равно нулю, то вершина является корнем дерева.

В третьей строке находится число m ($1 \le m \le 100\,000$) — количество запросов. Каждая из следующих m строк содержит два различных числа a и b ($1 \le a, b \le n$).

Формат выходного файла

Для каждого из m запросов выведите на отдельной строке число `1`, если вершина a является одним из предков вершины b, и `0` — в противном случае.

Пример

i.in	i.out
6 0 1 1 2 3 3 5 4 1 1 4 3 6 2 6 6 5	0 1 1 0 0

Разбор задачи I. Предок

В этой задаче требуется быстро определять, является ли одна вершина предком другой в дереве.

Решение задачи было описано в лекции.

Задача J. Неизбежность

Имя входного файла:	`j.in`
Имя выходного файла:	`j.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Вася живет в первой вершине связного неориентированного графа, состоящего из n вершин и m ребер. Каждый день он ходит в школу, находящуюся в вершине с номером n. Вася старается каждый день ходить в школу новым маршрутом, однако однажды он заметил, что некоторые ребра он проходит каждый день, независимо от того, каким маршрутом идет. Помогите Васе найти все такие ребра.

Формат входного файла

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

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

Формат выходного файла

Первая строка выходного файла должна содержать одно натуральное число b — количество ребер, которые неизбежно встречаются на пути Васи. На следующей строке выведите b целых чисел — номера этих ребер в возрастающем порядке. Ребра нумеруются с единицы в том порядке, в котором они заданы во входном файле.

Пример

j.in	j.out
7 8 1 2 2 3 3 1 4 3 5 4 5 6 4 6 6 7	2 4 8
4 4 1 2 2 4 1 4 2 3	0

Разбор задачи J. Неизбежность

Требуется найти ребра, которые гарантированна лежат на пути между двумя заданными вершинами s и t. Не трудно понять, что такие ребра — это мосты. Однако не все мосты, а только находящиеся на пути. Для решения задачи необходимо найти мосты. После этого нужно найти произвольный простой путь из s в t. Мосты, лежащие на этом пути, и будут решением задачи.

Задача K. Мосты

Имя входного файла:	`k.in`
Имя выходного файла:	`k.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Дан неориентированный граф. Требуется найти все мосты в нем.

Формат входного файла

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

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 выходного файла

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

Пример

k.in	k.out
6 7 1 2 2 3 3 4 1 3 4 5 4 6 5 6	1 3

Разбор задачи K. Мосты

Нужно просто реализовать алгоритм нахождение мостов, точек сочленения, эйлерова пути, конденсации графа и топологической сортировки.

Решение задачи было описано в лекции.

Задача L. Цветные волшебники

Имя входного файла:	`l.in`
Имя выходного файла:	`l.out`
Ограничение по времени:	2 с
Ограничение по памяти:	64 Мб

Сказочная страна представляет собой множество городов, соединенных дорогами с двухсторонним движением. Причем из любого города страны

можно добраться в любой другой город либо непосредственно, либо через другие города. Известно, что в сказочной стране не существует дорог, соединяющих город сам с собой, и между любыми двумя разными городами, существует не более одной дороги.

В сказочной стране живут желтый и синий волшебники. Желтый волшебник, пройдя по дороге, перекрашивает ее в желтый цвет, синий — в синий. Как известно, при наложении желтой краски на синюю, либо синей краски на желтую, краски смешиваются и превращаются в краску зеленого цвета, который является самым нелюбимым цветом обоих волшебников.

В этом году в столице страны (городе f) проводится конференция волшебников. Поэтому желтый и синий волшебники хотят узнать, какое минимальное количество дорог им придется перекрасить в зеленый цвет, чтобы добраться в столицу. Изначально все дороги не покрашены.

Начальное положение желтого и синего волшебников заранее не известно. Поэтому необходимо решить данную задачу для k возможных случаев их начальных расположений.

Формат входного файла

Первая строка входного файла содержит целые числа: n ($1 \leq n \leq 100$) и m ($1 \leq m \leq 500$) — количество городов и дорог в волшебной стране соответственно. Третья строка содержит одно целое число f ($1 \leq f \leq n$) — номер города, являющегося столицей сказочной страны. В следующих m строках, находится описание дорог страны. В этих m строк записано по два целых числа a_i и b_i, означающих, что существует дорога, соединяющая города a_i и b_i. Следующая строка содержит целое число k ($1 \leq k \leq 100$) — количество возможных начальных расположений волшебников. Далее следуют k строк, каждая из которых содержит два целых числа — номера городов, в которых изначально находится желтый и синий волшебники соответственно.

Формат выходного файла

Для каждого из k случаев, ваша программа должна вывести в выходной минимальное количество дорог, которое придется покрасить в зеленый цвет волшебникам для того, чтобы добраться в столицу.

Пример

l.in	l.out
6 6 1 1 2 2 3 3 4 4 2 4 5 3 6 2 5 6 6 6	1 2

Разбор задачи L. Цветные волшебники

Для решения задачи найдем конденсацию графа по компонентам реберной двусвязности. В любой компоненте двусвязности два волшебника с легкостью смогут разойтись, не пересекаясь по ребрам. Мосты — единственные ребра, которые они с неизбежностью должны вместе пройти. Конденсация является деревом, состоящее только из мостов. Требуемое количество ребер — это количество общих ребер в конденсации на пути к столице. Для нахождения количества общих ребер в пути можно подвесить дерево за компоненту двусвязности с столицей. После этого количество общих ребер будет равно глубине наименьшего общего предка. Можно искать наименьшего общего предка за линейное время.

Задача M. Точки сочленения

Имя входного файла:	`m.in`
Имя выходного файла:	`m.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Дан неориентированный граф. Требуется найти все точки сочленения в нем.

Формат входного файла

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

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

Формат выходного файла

Первая строка выходного файла должна содержать одно натуральное число b — количество точек сочленения в заданном графе. На следующей строке выведите b целых чисел — номера вершин, которые являются точками сочленения, в возрастающем порядке.

Пример

m.in	m.out
9 12 1 2 2 3 4 5 2 6 2 7 8 9 1 3 1 4 1 5 6 7 3 8 3 9	3 1 2 3

Разбор задачи M. Точки сочленения

Нужно просто реализовать алгоритм нахождение мостов, точек сочленения, эйлерова пути, конденсации графа и топологической сортировки.

Решение задачи было описано в лекции.

Задача N. Почтальон

Имя входного файла:	`n.in`
Имя выходного файла:	`n.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

В городе есть n площадей, соединенных улицами. При этом количество улиц не превышает ста тысяч и существует не более трех площадей, на

которые выходит нечетное число улиц. Для каждой улицы известна ее длина. По улицам разрешено движение в обе стороны. В городе есть хотя бы одна улица. От любой площади до любой можно дойти по улицам.

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

Формат входного файла

Первая строка входного файла содержит натуральное число n — количество площадей в городе ($1 \le n \le 1\,000$). 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 положительных 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 улиц с площади на нее саму.

Все числа во входном файле не превосходят 10^5.

Формат выходного файла

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

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

Если решений несколько, выведите любое.

Пример

n.in	n.out
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

Разбор задачи N. Почтальон

Нужно просто реализовать алгоритм нахождение мостов, точек сочленения, эйлерова пути, конденсации графа и топологической сортировки.

Решение задачи было описано в лекции.

Задача O. Конденсация графа

Имя входного файла:	`o.in`
Имя выходного файла:	`o.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Требуется найти количество ребер в конденсации ориентированного графа. Примечание: конденсация графа не содержит кратных ребер.

Формат входного файла

Первая строка входного файла содержит два натуральных числа n и m — количество вершин и ребер графа соответственно ($n \leq 10\,000$, $m \leq 100\,000$). Следующие m строк содержат описание ребер, 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 ребра и петли.

Формат выходного файла

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

Пример

o.in	o.out
4 4 2 1 3 2 2 3 4 3	2

Разбор задачи O. Конденсация графа

Нужно просто реализовать алгоритм нахождение мостов, точек сочленения, эйлерова пути, конденсации графа и топологической сортировки.

Решение задачи было описано в лекции.

Задача P. Топологическая сортировка

Имя входного файла:	`p.in`
Имя выходного файла:	`p.out`
Ограничение по времени:	2 с
Ограничение по памяти:	256 Мб

Дан ориентированный невзвешенный граф. Необходимо его топологически отсортировать.

Формат входного файла

В первой строке входного файла даны два натуральных числа N и M ($1 \le N \le 100\,000, M \le 100\,000$) — 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 выходного файла

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

Пример

p.in	p.out
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

Разбор задачи P. Топологическая сортировка

Нужно просто реализовать алгоритм нахождения мостов, точек сочленения, эйлерова пути, конденсации графа и топологической сортировки.

Решение задачи было описано в лекции.