Тема 12. Графы

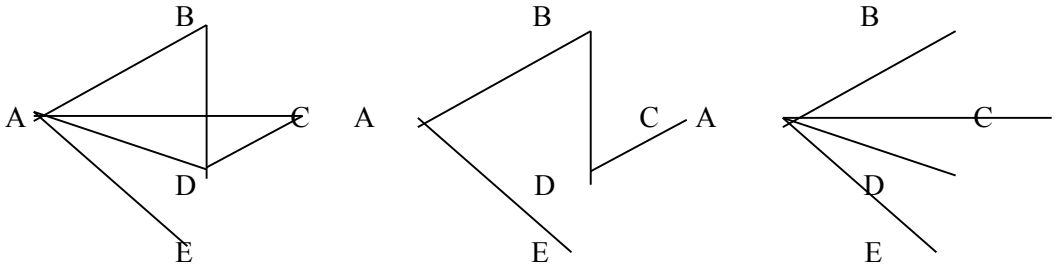
Содержание

В теории алгоритмов графы – это одна из универсальных тем. С помощью графов исследуются транспортные системы, электрические цепи, сети коммуникаций. Наиболее распространены задачи поиска путей на графах. Однако применение теории графов значительно шире.

В настоящем разделе описываются некоторые распространенные задачи на графах: поиск остовных деревьев, методы исследования связности графов, проверка ацикличности, топологическая сортировка ориентированных графов. Приводится ряд задач, которые не имеют выраженной “графовой” ориентации, но, тем не менее, сводятся к графам. Подобные графы иногда называют скрытыми или неявными.

12.1. Остовные деревья

Рассмотрим неориентированный связный граф. Остовное (каркасное) дерево – подграф, являющийся деревом и содержащий все его вершины. Для графа в левой части рисунка справа показаны примеры остовных деревьев

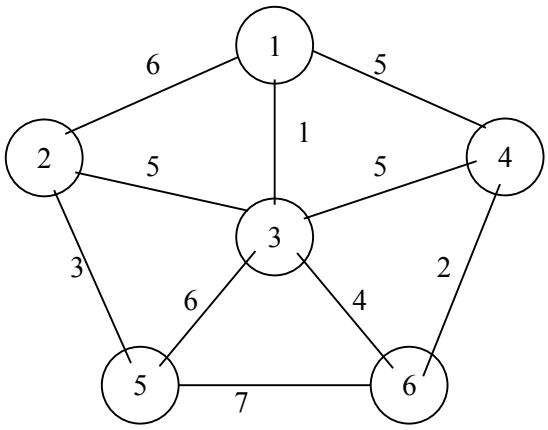


Здесь рассматриваются свободные (бескорневые) деревья, в которых корнем можно считать любую вершину.

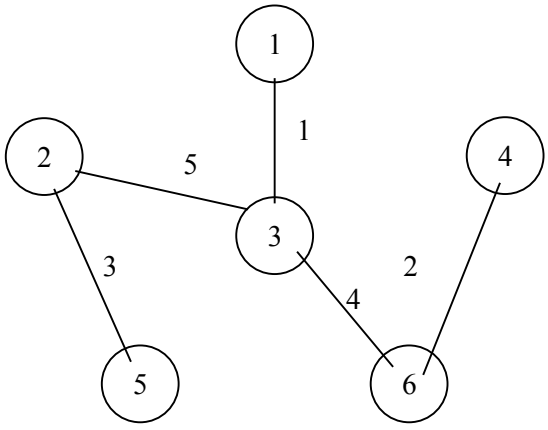
Ряд практических задач связан с нахождением остовных деревьев. Например, множество населенных пунктов требуется связать между собой дорогами, телефонной связью, водопроводом и т. п. Если в графе заданы стоимости ребер, то встает естественная задача нахождения остовного дерева с минимальной суммарной стоимостью ребер. Наиболее распространены два алгоритма нахождения остовных деревьев: Прима и Краскала [4, 14].

Алгоритм Прима начинает построение минимального остовного дерева U с включения в него произвольной вершины u. Далее находится ребро (u, v) минимальной стоимости, связывающее множество вершин U с вершинами, не входящими в U. Вершина v и ребро (u, v) включаются в U. Процесс продолжается, пока в U не войдут все вершины графа.

Рассмотрим для примера следующий граф.



Пусть выбрана начальная вершина 1. В остовное дерева будут последовательно добавляться вершина 3 и ребро (1, 3), вершина 6 и ребро (3, 6), вершина 4 и ребро (6, 4), вершина 2 и ребро (3, 2), вершина 5 и ребро (2, 5). Получим следующее минимальное остовное дерево



Алгоритм Прима можно реализовать аналогично алгоритму Дейкстры. Будем в каждой вершине проставлять два типа числовых меток: временную и окончательную (постоянную).

1. Первая вершина включается в остовное дерево. Ей присваивается окончательная метка 0, остальным вершинам – временные метки ∞.

2. Всем преемникам последней вершины с окончательной меткой, имеющим временные метки, присваивается новая временная метка в виде меньшего значения ее прежней величины и стоимости ведущего в вершину ребра.
3. Минимальная из временных меток становится окончательной, а ребро вместе с новой вершиной включается в остовное дерево.
4. Переход к 2, если остаются вершины с временными метками.

После N-1 шага формирование остовного дерева заканчивается. Этапы расстановки меток по приведенному примеру представим в виде таблицы. Первые 6 колонок соответствуют вершинам. Окончательную метку будем отмечать жирным шрифтом и подчеркиванием. В скобках указывается предыдущая вершина с окончательной меткой. В последней колонке содержится общая стоимость остовного дерева в процессе формирования. Она представляет собой сумму окончательных меток.

№	1	2	3	4	5	6	Ребро	Сумма
0	**0**	∞	∞	∞	∞	∞		
1	**0**	6(1)	1(1)	5(1)	∞	∞		
2	**0**	6(1)	**1(1)**	5(1)	∞	∞	1-3	1
3	**0**	5(3)	**1(1)**	5(1)	6(3)	4(3)		
4	**0**	5(3)	**1(1)**	5(1)	6(3)	**4(3)**	3-6	5
5	**0**	5(3)	**1(1)**	2(6)	6(3)	**4(3)**		
6	**0**	5(3)	**1(1)**	**2(6)**	6(3)	**4(3)**	6-4	7
7	**0**	5(3)	**1(1)**	**2(6)**	6(3)	**4(3)**		
8	**0**	**5(3)**	**1(1)**	**2(6)**	6(3)	**4(3)**	3-2	12
9	**0**	**5(3)**	**1(1)**	**2(6)**	3(2)	**4(3)**		
10	**0**	**5(3)**	**1(1)**	**2(6)**	**3(2)**	**4(3)**	2-5	15

Трудоемкость алгоритма Прима составляет $O(N^2)$, где N – число вершин графа. Для разреженных графов существуют другие реализации алгоритма, основанные на более сложных структурах данных, таких как бинарная куча.

В **алгоритме Краскала** первоначально считается, что граф состоит из N компонент, представленных вершинами графа. На очередном шаге добавляется ребро минимальной стоимости, соединяющее две разные компоненты. Эти компоненты объединяются. В конце остается единственная компонента, которая и будет являться минимальным остовным деревом.

В приведенном примере к вершинам графа будут добавляться ребра в порядке (1, 3), (4, 6), (2, 5), (3, 6), (2, 3). В результате получается то же минимальное остовное дерево, что и в алгоритме Прима.

Выбор минимального ребра, соединяющего две разные компоненты, достигается с помощью меток. Первоначально в каждой вершине проставляется метка, совпадающая с номером вершины. После выбора ребра A-B вершина B получает метку вершины A. Более того, такую же метку получают все вершины, имеющие метки вершины B. Чтобы избежать перебора при перекраске меток, используют дополнительные списки.

Трудоемкость алгоритма Краскала составляет $O(M \times logM)$, где M – количество дуг графа. Конечно, предполагается применение рационального алгоритма сортировки ребер по их стоимости. Если M существенно меньше N^2, то есть граф разреженный, то выгоднее применение алгоритма Краскала, а не Прима.

12.2. Алгоритм Беллмана-Форда

Алгоритм носит имя двух американских учёных: Ричарда Беллмана (Richard Bellman) и Лестера Форда (Lester Ford). Форд фактически изобрёл этот алгоритм в 1956 г. при изучении другой

математической задачи, подзадача которой свелась к поиску кратчайшего пути в графе, и Форд дал набросок решающего эту задачу алгоритма. Беллман в 1958 г. опубликовал статью, посвященную конкретно задаче нахождения кратчайшего пути, и в этой статье он четко сформулировал алгоритм в том виде, в котором он известен нам сейчас. В 1969 г. алгоритм лег в основу протокола маршрутизации RIP (Routing Information Protocol).

Как и алгоритм Дейкстры, алгоритм Беллмана-Форда вычисляет во взвешенном графе кратчайшие пути от заданной вершины S до всех остальных вершин. Он подходит для работы с графами, в которых имеются дуги отрицательной длины. Задача становится неопределенной, если из начальной вершины достижим циклический путь отрицательной длины. В этом случае алгоритм выделяет один из таких циклов, что является его дополнительным достоинством.

Трудоемкость алгоритма оценивается величиной $O(M \times N)$, где M – количество дуг, а N – число вершин графа. Как видно из этой оценки, алгоритм выгодно использовать для разреженных графов. В отличие от матричных алгоритмов дуги могут задаваться в произвольном порядке обычным списком.

Заведем двумерный массив D. В элементе $D[i, j]$ будет храниться значение кратчайшего пути из вершины S в вершину i, состоящего не более чем из j дуг. Изначально, присвоим элементам массива D значения, равные условной бесконечности (например, число заведомо большее суммы всех весов), а все элементы $D[S, j]$ приравняем к нулю. Это соответствует тому, что наилучший путь из вершины S в нее же саму равен 0, а другие вершины графа из вершины S пока недоступны.

Итак, есть граф $G = (V, E)$, и $N = |V|$, а $M = |E|$. Обозначим смежные вершины этого графа символами u и v ($u, v \in V$), а вес ребра (u, v) через $w(u, v)$. Проведем поиск в ширину от вершины S по увеличению числа входящих в пути дуг. Во внешнем цикле по i будем изменять количество дуг от 1 до N-1. Во внутреннем цикле для каждой дуги (u, v) при условии конечности значения $D[u, i\text{-}1]$ проведем перерасчет по формуле

$D[v, i] = \min (D[v, i], D[u, i\text{-}1] + w(u, v))$.

Этот алгоритм можно ускорить: зачастую ответ находится уже за несколько шагов, а за оставшиеся шаги лишь впустую просматриваются все ребра. Поэтому можно хранить флаг того, изменилось ли что-то на текущем шаге фазе или нет. Если на каком-то шаге ничего не произошло, то алгоритм можно останавливать. Эта оптимизация не улучшает асимптотику, но значительно ускоряет поведение алгоритма "в среднем", то есть на случайных графах.

Рассмотрим теперь, как можно модифицировать алгоритм Форда-Беллмана так, чтобы он не только находил длины кратчайших путей, но и позволял восстанавливать сами кратчайшие пути. Для этого заведем еще один двумерный массив P, в котором для каждой i-ой вершины будем хранить её "предка", т.е. предпоследнюю вершину в кратчайшем пути. Как и в массиве D, второй индекс будет соответствовать числу дуг. С помощью массива P кратчайшие пути восстанавливаются в обратном порядке.

Выше мы везде считали, что отрицательного цикла в графе нет (уточним, нас интересует отрицательный цикл, достижимый из стартовой вершины). Если не ограничивать число шагов, то алгоритм будет работать бесконечно, постоянно улучшая расстояния до всех вершин цикла.

Отсюда мы получаем **критерий наличия достижимого цикла отрицательной длины**. Если после N-1-го шага мы выполним еще один шаг, и на нем произойдет хотя бы одно улучшение, то граф содержит цикл отрицательной длины, достижимый из начальной вершины. В противном случае такого цикла нет.

Более того, если такой цикл обнаружился, то алгоритм Форда-Беллмана можно модифицировать таким образом, чтобы он выводил сам этот цикл в виде последовательности вершин, входящих в него. Для этого достаточно запомнить номер вершины u, в которой произошло улучшение на N-ом шаге. Эта вершина будет либо лежать на цикле отрицательной длины, либо она достижима из него. Чтобы получить вершину, которая гарантированно лежит на цикле, достаточно, например, N раз пройти по предкам, начиная от вершины u. Получив номер вершины v, лежащей на цикле, надо пройтись от этой вершины по предкам, пока мы не вернёмся в эту же вершину (а это обязательно произойдёт, потому что улучшения в цикле отрицательной длины происходят по кругу).

Оказывается, что массивы D и P могут быть одномерными. Снова внешний цикл повторяется N раз. При этом последний шаг служит для выявления циклического пути отрицательного длины. Формула расчета кратчайших расстояний практически не изменяется, принимая вид

$D[v] = \min (D[v], D[u] + w(u, v))$.

Однако сейчас уже не будет классического поиска в ширину по увеличению числа дуг. На одном шаге внешнего цикла могут определяться кратчайшие пути из разного количества дуг в зависимости

от очередности их просмотра. Тем не менее, после выполнения k шагов внешнего цикла находятся все кратчайшие пути, число дуг которых не превосходит k.

Приведем текст программы для следующего формата данных.

Ввод из файла INPUT.TXT. Первая строка входного файла INPUT.TXT содержит 4 числа: N - количество вершин графа ($3 \le N \le 30$), M – дуг ($3 \le M \le 30$), S – номер начальной вершины. В следующих M строках по 3 числа, задающих дуги: A - номер начальной вершины, B - номер конечной вершины, C – длина (вес).

Вывод в файл OUTPUT.TXT. В i-й строке выводится длина минимального пути из вершины S до i-й вершины, затем количество вершин минимального пути и а далее номера вершин минимального пути. Все числа разделены пробелами. Если пути в некоторую вершину не существует, то в соответствующей строке выводится слово No. Если в графе имеется достижимый из начальной вершины цикл отрицательной длины, то вывод состоит из двух строк. В первой строке выводится слово No, а во второй –количество и номера вершин обнаруженного цикла через пробел, начиная с его любой вершины и заканчивая ей же. При наличии нескольких циклов вывести информацию о любом из них.

Примеры ввода-вывода

Ввод 1
```
4 7 1
1 2 3
1 3 7
2 3 2
2 4 1
3 2 2
3 4 2
4 3 2
```

Вывод 1
```
0 0
3 2 1 2
5 3 1 2 3
4 3 1 2 4
```

Ввод 2
```
4 2 1
1 2 3
3 4 4
```

Вывод 2
```
0 1 1
3 2 1 2
No
No
```

Ввод 3
```
4 4 1
1 2 2
2 3 4
3 1 -9
2 4 5
```

Вывод 3
```
No
4 2 3 1 2
```

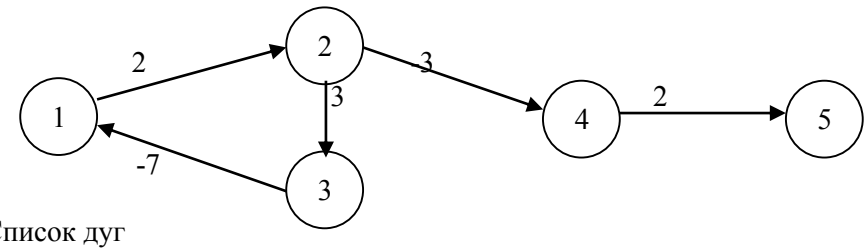
Пример реализации на языке C++

Реализация алгоритма Беллмана-Форда размещена на Github: https://github.com/PS-Group/algorithm-samples/tree/master/graphs

- Первая реализация создана с использованием стандартной библиотеки STL и покрыта тестами на boost.test. Реализация читает и пишет данные согласно описанному текстовому формату.
- Вторая реализация создана с использованием библиотеки boost.graph и покрыта тестами на boost.test. Реализация также способна читать и писать в вышеописанном текстовом формате, но кроме этого умеет читать графы на языке пакета Graphviz (dot-файлы).

Подробное объяснение алгоритма

Рассмотрим несколько примеров. Имеется следующий граф



Список дуг

1 2 2

2 3 3

2 4 -3
3 1 -4
2 4 2

Первые два числа задают дугу, а третье ее длину. Будем искать кратчайшие пути из вершины 1. Шаги алгоритма показаны в таблице. Номер соответствует значению параметра внешнего цикла *i*. В скобках указаны номера предыдущих вершин.

№	1	2	3	4	5
0	0(1)	∞	∞	∞	∞
1	-2(3)	2(1)	5(2)	-1(2)	1(4)
2	-2(3)	2(1)	5(2)	-1(2)	1(4)

Все кратчайшие расстояния получены на первом шаге. На втором шаге значения меток не изменились, поэтому вычисления заканчиваются. Кратчайшие пути восстанавливаются в обратном порядке по номерам предыдущих вершин.

Если список дуг задан в обратном порядке, то таблица примет следующий вид:

№	1	2	3	4	5
0	0(1)	∞	∞	∞	∞
1	0(1)	2(1)	∞	∞	∞
2	0(1)	2 (1)	5(2)	-1(2)	∞
3	0(1)	2 (1)	5(2)	-1(2)	1(4)
4	0(1)	2 (1)	5(2)	-1(2)	1(4)

Сейчас кратчайшие расстояния найдены на третьем шаге, что определяется на следующем шаге.

Рассмотрим, наконец, случай отрицательного цикла. Восстановим первоначальный порядок списка дуг, но заменим вес дуги 1-3 с -4 на -7. В итоге получится следующая таблица результатов.

№	1	2	3	4	5
0	0(1)	∞	∞	∞	∞
1	-2(3)	2(1)	5(2)	-1(2)	1(4)
2	-4(3)	0(1)	3(2)	-3(2)	-1(4)
3	-6(3)	-2(1)	3(2)	-5(2)	-3(4)
4	-8(3)	-4(1)	1(2)	-7(2)	-5(4)
5		-6(1)			

На последнем контрольном шаге уменьшилась метка вершины 2, поэтому делается вывод о наличии отрицательного цикла. Одна из вершин этого цикла находится проходом в обратном направлении из вершины 2. Для этого потребуется не более N шагов. В данном примере обнаруживается цикл 2 – 1 – 3 – 2.

Заметим, что вершина с меньшим значением метки может не входить в цикл отрицательной длины. При другом порядке списка дуг первой такой вершиной могла оказаться вершина 5. В общем случае после нахождения одной из вершин цикла нужно еще раз пройти по предыдущим вершинам до его обнаружения.

12.3. Связность графов

Приведем некоторые определения. Неориентированный граф **связен**, если существует хотя бы один путь между каждой парой вершин.

Ориентированный граф связен, если связен неориентированный граф, получающийся путем удаления ориентации ребер.

Ориентированный граф **сильно связен**, если для каждой пары вершин существуют пути как из первой вершины во вторую, так и из второй вершины в первую.

Максимальный связный подграф называется **связной компонентой** графа. По аналогии, максимальный сильно связный подграф называется **сильно связной компонентой** графа.

Связность графа выявляется проще всего обычным поиском в глубину из произвольной начальной вершины. Посещаемые вершины помечаются с тем, чтобы повторно в них не заходить. Если в конце все вершины оказываются помеченными, то граф связен. Поскольку при поиске в глубину каждое ребро проходится только дважды, то общая трудоемкость этого метода составляет $O(V+E)$, где V – количество вершин графа, а E – количество ребер.

Если граф несвязен, то часто требуется выделить все компоненты связности. Для этого достаточно использовать различные метки. Выберем произвольную начальную вершину и обойдем в глубину компоненту связности, включающую эту вершину. Затем выберем любую непомеченную вершину и выделим таким же образом следующую компоненту. Будем повторять эти действия, пока не окажутся помеченными все вершины графа.

Выделение сильно связных компонент происходит несколько сложнее. Пусть T – некоторая вершина ориентированного графа. Обозначим через $R(T)$ множество вершин, достижимых из вершины T, а через $Q(T)$ – множество вершин, из которых существует путь в вершину T. Оказывается, что пересечение множеств $R(T)$ и $Q(T)$ вместе с соответствующими дугами является множеством вершин графа, составляющим вместе с инцидентными им дугами сильно связную компоненту [4]. После пометки всех ее вершин можно находить следующие сильно связные компоненты.

Иногда недостаточно знать, что граф связен. Может возникнуть вопрос, насколько “сильно” связан граф. Например, в графе может существовать вершина, удаление которой вместе с инцидентными ей ребрами разъединяет оставшиеся вершины. Такая вершина называется точкой сочленения. Граф без точек сочленения называется **двусвязным**. Максимальный двусвязный подграф графа называется **двусвязной компонентой**.

Точки сочленения и двусвязные компоненты проще всего находить простым перебором. Удалим какую-либо вершину вместе с инцидентными ей ребрами и проверим связность получившегося графа. Если связность нарушена, то эта точка является точкой сочленения. После удаления точек сочленения легко выделить двусвязные компоненты.

Трудоемкость этого метода $O(V^2+VE)$. Существует алгоритм выделения точек сочленения и двусвязных компонент с трудоемкостью $O(V+E)$ [1, 8].

Рассмотрим следующую задачу.

Жизнь на Марсе. При высадке на Марс было основано N поселений. Их координаты заданы. Каждое поселение равномерно расширялось во все стороны на L марсианских миль в месяц. Постепенно поселения начали сливаться друг с другом, получая общее название. Какое минимальное время с момента высадки потребуется для того, чтобы на Марсе осталось не более K поселений?

Будем рассматривать граф, соответствующий марсианским поселениям. Если нет совпадающих по координатам поселений, то сначала он состоит только из N вершин. Два поселения соприкоснутся в середине соединяющего их отрезка через время $T=D/2L$, где D – расстояние между поселениями. Назовем это событие встречей. Будем встречу поселений отмечать ребром графа.

Поселения, расположенные ближе друг от друга, будут встречаться раньше. Нельзя считать, что при каждой встрече число поселений уменьшается на 1. Во-первых, при равных ребрах некоторые встречи будут происходить одновременно. Во-вторых, появление нового ребра может не приводить к уменьшению числа поселений, если обе вершины на концах этого ребра уже связаны через более короткие ребра.

Задача сводится к поиску такого минимального множества самых коротких расстояний между точками, что граф из N вершин и соответствующих ребер будет состоять из K или менее компонент связности. Полная постановка задачи имеется в конце раздела.

Следующая задача может быть решена путем нахождения всех циклических путей на ориентированном графе, но проще всего использовать понятие сильной связности.

Рекурсия. Имеется программа, состоящая из n процедур P_1, P_2 ,…, P_n. Пусть для каждой процедуры известны процедуры, которые она может вызывать. Процедура P называется потенциально рекурсивной, если существует такая последовательность процедур Q_0, Q_1, …, Q_k, что $Q_0 = Q_k = P$, $k \geq 2$ и для $i = 1,\ldots, k–1$ процедура Q_{i-1} может вызвать процедуру Q_i. Требуется определить для каждой из заданных процедур, является ли она потенциально рекурсивной.

Для решения этой задачи необходимо построить ориентированный граф, вершинами которого являются процедуры. Между двумя вершинами должна быть дуга, если одна процедура может вызвать другую. После этого необходимо найти компоненты сильной связности этого графа и все вершины, которые лежат в компонентах сильной связности размером больше единицы.

12.4. Циклы и ацикличность. Неявные графы

Поиск циклов на ориентированном графе. Многие практические задачи сводятся к поиску циклов, то есть путей, начинающихся и заканчивающихся в одной и той же вершине. Отметим два момента. Во-первых, очевидно есть смысл искать элементарные циклы, то есть пути, не содержащие циклов внутри себя. Во-вторых, каждый цикл из K вершин порождает еще K-1 цикл путем выбора другой начальной вершины. Например, если путь из вершин $a – b – c$ образует цикл, то циклами будут и пути $b – c – a$, $c – a – b$. Простейший способ избежать такого повторения – нумеровать вершины графа и считать, что цикл начинает вершина с минимальным номером.

Циклы можно находить различными способами. Выберем, например, начальную вершину S и обходом в глубину будем искать пути $S – T – R –\ldots– S$ с дополнительными условиями, чтобы номера вершин T, R, … были большими, чем номер вершины S, а отличные от S вершины в пути не повторялись. Перебирая начальные вершины, можно найти все циклы.

Часто по смыслу задачи граф не должен иметь циклов. Рассмотрим, например, следующую задачу [3, 11].

Стройка. При строительстве дома есть несколько видов работ. Некоторые из них можно выполнять только после завершения других, а некоторые не зависят одна от другой. Например, нельзя возводить стены, пока не закончен фундамент, но электротехнические и водопроводные работы можно выполнять одновременно. Проектная организация предоставила строителям перечень работ и их зависимость друг от друга. Требуется проверить корректность предоставленной информации либо выявить хотя бы одну ошибку.

Итак, виды работ частично упорядочены. Представив работы вершинами графа, а зависимости между ними дугами, направленными от более ранних работ к более поздним, получим ориентированный граф.

Эта модель появилась в конце 50-х годов прошлого столетия и легла в основу направления “Сетевое планирование”. Таким образом, например, планировался американский проект пилотируемого полета на Луну. Иногда работам ставят в соответствие не вершины, а дуги графа. Каждый способ имеет свои достоинства и недостатки.

Полученный граф может содержать большое количество вершин и ребер, но не должен иметь циклов. При наличии циклов решение задач сетевого планирования становится невозможным. Поэтому требуется проверить ацикличность ориентированного графа либо указать какую-либо вершину, входящую в цикл.

Если в графе есть цикл, содержащий некоторую вершину A, то при обходе в глубину с начальной вершиной V обязательно встретится дуга цикла, ведущая в A. С другой стороны, в графе может быть цикл, не содержащий вершины A и такой, что при обходе с начальной вершиной A он не найдется. Значит, для полного исследования графа можно запустить обход в глубину с каждой непосещенной ранее вершиной в качестве начальной.

Топологическая сортировка. Предположим, что ацикличность графа, соответствующего нашей стройке, проверена. Расширим постановку задачи. Пусть единственная бригада выполняет все виды работ. Требуется спланировать последовательность работ так, чтобы зависимые работы выполнялись после тех, от которых они зависят.

Другими словами, нужно пронумеровать вершины ациклического ориентированного графа так, чтобы номер каждой вершины был больше номеров вершин, из которых в нее ведут дуги.

Указанная нумерация вершин называется **топологической сортировкой**. Она задает линейное упорядочение вершин и может быть не единственной.

Топологическую сортировку можно провести на основе обхода в глубину, помечая посещенные вершины. Номера присваиваются в порядке убывания, но не при первом приходе в вершину, а перед тем, как эта вершина удаляется из текущего пути.

Если после очередного обхода в графе остались непомеченные вершины, берется следующая начальная вершина, и обход повторяется. Возможно, из новых вершин достижимы старые, помеченные на предыдущих обходах. Однако номера новых вершин меньше, поэтому правильность нумерации не нарушается. Если в процессе обхода обнаружится цикл, работа после выдачи диагностического сообщения прекращается.

Нередко формулировка проблемы в терминах теории графов появляется не сразу. Рассмотрим следующую задачу.

Ременная передача. На плоскости расположены N валов. Заданы координаты их центров и радиусы. Некоторые валы связаны ременными передачами. Валы, имеющие общий центр вращения, жестко связаны, то есть имеют одинаковую угловую скорость. Задаваемая система геометрически правильная, то есть валы не пересекаются между собой, а валы, имеющие общий центр вращения, не могут быть соединены ремнем.

Вал с номером M вращается по часовой стрелке со скоростью 1000 оборотов в минуту. Требуется для каждого вала найти его угловую скорость. Необходимо отметить случай, когда заданная система противоречива, то есть на один вал передается вращение от разных источников с разными угловыми скоростями.

Для перехода к графу сопоставим вершинам валы, а ребрам - ременные передачи или жесткое сцепление. Тогда проводя поиск в ширину от начального вала можно рассчитать движение каждого вала.

В следующей задаче также можно использовать графы [11].

Обмен квартир. В файле записаны предложения по обмену жилплощадью в пределах некоторого города. Имеются варианты размена одной квартиры на две других либо наоборот. Требуется по заявке клиента предложить способы обмена. Предусмотреть возможность нахождения циклических обменов, в которых участвуют более двух сторон. Найденные варианты выдать в порядке возрастания числа участвующих в обмене сторон.

Вершинам ориентированного графа поставим в соответствие владельцев квартир. Если владельца A устраивает для обмена квартира (квартиры) владельца B, то в графе окажется дуга AB.

Учтем в описанном графе условия и пожелания нового клиента X. Простой обмен сводится к нахождению всех вершин, которые связаны с вершиной X взаимными дугами. Более сложные обмены обеспечивает поиск циклов X - C_1 - C_2 -...- C_K – X. Предполагается, что клиент переедет в квартиру владельца C_1, тот в свою очередь переедет в квартиру владельца C_2 и, в конце концов, владелец C_K займет квартиру владельца X. Задача может быть существенно усложнена, если допустить, что возможны обмены квартиры на две других, которые могут принадлежать разным владельцам.

Рассмотрим еще одну задачу с применением графов.

Проверка веб-страниц. Работая в Интернете, многим случалось сталкиваться со ссылками на несуществующие документы. Входной файл содержит название одного или нескольких документов и их содержимое. В тексте могут присутствовать ссылки на другие документы данного сервера. Требуется найти общее количество ссылок на несуществующие документы и количество документов, до которых нельзя добраться, начав с первого документа и переходя по ссылкам.

В качестве вершин графа возьмем имена заданных документов, а также имена файлов, на которые есть ссылки. В качестве дуг примем ссылку на файл. Тогда общее количество ссылок на несуществующие документы – это сумма всех ссылок на вершины, соответствующие таким файлам, имена которых не фигурируют как имена документов.

Количество документов, на которые нельзя попасть по ссылкам, начиная с первого документа, можно определить с помощью матрицы достижимости. Эта матрица, называемая также транзитивным замыканием, легко получается с помощью алгоритма Уоршела [1, 11]. По матрице достижимости находится количество вершин, не достижимых из первой вершины.

Иногда в дополнение к графу, который следует из условий задачи, необходимо строить некий модифицированный граф. Рассмотрим подобную задачу.

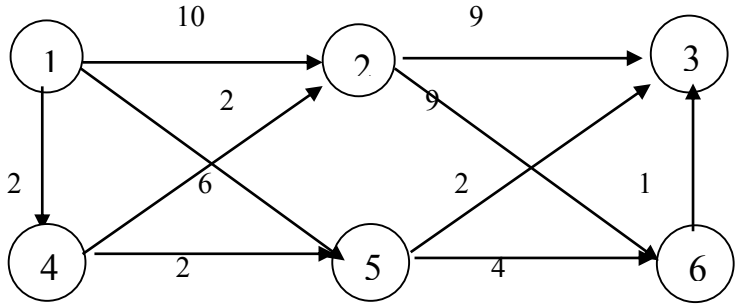
Шпионские страсти. Резидент разведки руководит сетью секретных агентов, контактируя только с некоторыми из них. Каждый агент имеет право передавать информацию определенному

кругу других агентов. Полученные данные могут распространяться далее и, в конечном счете, доставляются резиденту. Резидент оплачивает каждую передачу сведений от одного агента другому. Величина оплаты зависит от пары агентов и от того, кто из них принимает информацию. Таким образом, общая оплата доставки информации зависит только от цепочки агентов, участвующих в доставке, и действующих расценок.

Определенный агент добыл ценные сведения. Резидент решил получить информацию по двум независимым каналам. Требуется найти минимальный размер затрат резидента для оплаты передачи информации по двум цепочкам агентов, которые начинаются с удачливого агента и заканчиваются резидентом, не пересекаясь друг с другом.

Итак, требуется найти на ориентированном взвешенном графе два непересекающихся пути из начальной вершины *A* в конечную *B* наименьшей суммарной стоимости.

Во-первых, убедимся, что нельзя использовать "жадный" алгоритм, то есть найти кратчайший путь, затем удалить из графа вершины этого пути вместе с входными и выходными дугами и снова найти кратчайший путь. Рассмотрим следующий граф:



Пусть требуется найти два пути минимальной суммарной стоимости из вершины 1 в вершину 3.

Кратчайший путь 1-4-5-3 имеет стоимость 6. Следующим минимальным путем, не пересекающимся с первым, является путь 1-2-3 стоимости 19. Суммарная стоимость двух путей 25. Однако путь 1-5-6-3 стоимости 11 и 1-4-2-3 стоимости 13 имеют суммарную стоимость 24. Более того, путь 1-4-2-3 стоимости 13 и 1-5-3 стоимости 8 имеют суммарную стоимость 21.

Перебор всех путей и выбор лучшей пары из них бесперспективен. Для графа из 30 вершин, в котором все вершины связаны дугами, только количество путей с включением всех вершин равно 28!, что составляет более 10^{29}. А если рассматривать все пути и перебирать пары путей?!

Будем одновременно отслеживать два пути, продвигаясь от *A* к *B* то по одному из них, то по второму (необязательно поочередно). Для этого в процессе решения будем неявно строить новый граф, вершинами которого являются пары вершин исходного графа. В каждой паре, кроме пар из двух начальных и двух конечных вершин, окажутся разные вершины, т. к. пути не должны пересекаться. Для определенности будем считать, что первой в паре располагается вершина с меньшим номером.

Дуга из некоторой первой пары вершин в какую-либо вторую пару при поиске двух путей из *A* в *B* описывается следующими правилами:

- в исходном графе есть дуга *L*, соединяющая одну из вершин первой пары с одной из вершин второй пары;
- две другие вершины из этих пар совпадают;
- два пути, восстановленные в исходном графе из второй пары в обратном направлении к паре вершин (*A, A*), где *A* - начальная вершина, не пересекаются;
- стоимостью дуги между парами вершин является стоимость дуги *L*.

В приведенном примере дуга (1, 1) - (1, 4) имеет стоимость 2, а дуга (2, 5) - (5, 6) – стоимость 9.

Задача сводится к нахождению кратчайшего пути в новом графе из вершины (*A, A*) в вершину (*B, B*), где *A* и *B* - начальная и конечная вершины исходного графа соответственно. Поскольку стоимости дуг положительны, то удобно применить алгоритм Дейкстры (поиск в ширину кратчайшего пути). В новом графе не более N^2 вершин, поэтому общая трудоемкость алгоритма $O(N^4)$, что вполне допустимо для заданной в условии размерности графа.

В нашем примере кратчайшим путем будет (1, 1) - (1, 4) - (1, 2) - (2, 5) - (2, 3) - (3, 3) стоимости 21. По этому пути восстанавливаются два пути в исходном графе 1-4-2-3 – стоимости 13 и 1-5-3 – стоимости 8. Суммарная стоимость этих двух путей составляет указанную выше величину 21.

Заметим в заключение, что граф по парам вершин не нужно явно размещать в памяти в виде матрицы стоимостей или какой-либо другой структуры. Формат данных для этой задачи приведен ниже.

Задачи для самостоятельного решения

12.1. Дороги Прима (5)

В государстве Прим имеется ряд населенных пунктов, связанных грунтовыми дорогами. Между любыми двумя пунктами может быть не более одной дороги. Решено заасфальтировать некоторые дороги. Стоимость асфальтирования каждой дороги известна. Требуется выбрать дороги для асфальтирования так, чтобы можно было проехать из каждого населенного пункта в любой другой по асфальту, а общая стоимость работы была минимальной. Использовать алгоритм Прима.

Ввод из файла INPUT.TXT. В первой строке вводятся значение N ($1 \le N \le 500$) и M – количество населенных пунктов и дорог. В каждой из следующих M строк вводится через пробел пара номеров пунктов в порядке возрастания в паре и стоимость асфальтирования дороги между ними.

Вывод в файл OUTPUT.TXT. В первой строке выводится общая стоимость работы. В следующих N -1 строках содержатся пары населенных пунктов, определяющие выбранные дороги. В каждой паре меньший номер должен быть первым. Список дорог должен быть отсортирован по возрастанию первого в паре номера, а при равном первом номере – по второму. В случае нескольких ответов выдать любой из них.

Пример

Ввод

```
4 5
2 3 5
1 4 5
1 2 6
1 3 1
3 4 3
```

Вывод

```
9
1 3
2 3
3 4
```

12.2. Королевские дороги (7)

В некотором королевстве N городов соединены N-1 дорогой. Имеется ровно один путь между любой парой городов. Дороги иногда выходят из строя. Требуется построить минимальное количество новых дорог так, чтобы каждая пара городов оказалась бы связана не менее чем двумя путями.

Ввод. В первой строке N ($2 \le N \le 100000$). В следующих N-1 строках – дороги в виде пар номеров городов A_i, B_i ($1 \le A_i, B_i \le N$).

Вывод. В первой строке минимальное число K новых дорог. В следующих K строках – пары городов, задающих новые дороги.

Примеры

Ввод 1

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

Вывод 1

```
2
1 4
4 5
```

Ввод 2

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

Вывод 2

```
2
3 2
1 4
```

12.3. Максимальный груз (6)

Имеются города с номерами от 1 до N и дороги между некоторыми из них. По дороге можно ехать в любом направлении. Известно, какой максимальный груз можно провезти по каждой из дорог. Нужно узнать, какие максимальные грузы можно доставить из города 1 в остальные города.

Ввод из файла INPUT.TXT. Первая строка содержит количество городов N и дорог M через пробел. В каждой из следующих M строк находится по 3 числа. Первые два из них - i и j- задают дорогу, а третье C_{ij} – ее грузоподъемность.

Ограничения: $2 \le N \le 50$; $0 \le C_{ij} \le 10^5$; время работы программы до 2 с.

Вывод в файл OUTPUT.TXT. В i-й строке выводится значение максимального груза, который может быть доставлен их города 1 до i+1-го города. Таким образом, файл OUTPUT.TXT состоит из N-1 строки. Если в какой-либо город пути нет, в соответствующей строке вывести -1.

Пример

Ввод

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

Вывод

```
6
6
3
7
```

Подсказка. Модифицировать алгоритм Дейкстры.

12.4. Эх, дороги (7)

В тридевятом царстве имеется сеть автомобильных дорог, связывающая столицу со всеми другими городами. Длины участков дорог известны. Однажды царь приказал министру гражданской обороны составить список кратчайших расстояний от столицы до всех остальных городов. Усердный министр решил перевыполнить приказ и поручил программистам дать дополнительно список длин вторых по минимальности путей. Требование на отсутствие циклов в путях было забыто, поэтому некоторые пути второго списка содержали повторяющиеся города, включая столицу и пункты назначения. Требуется вывести полученный список длин вторых по минимальности путей. В некоторые города вторых путей может не оказаться.

Ввод из файла INPUT.TXT. Первая строка содержит количество городов N и дорог M через пробел. В каждой из следующих M строк находится по три числа. Первые два из них - i и j - задают дорогу, а третье C_{ij} - ее длину. По некоторым дорогам разрешено движение только в одном направлении. Столица имеет номер 1.

Вывод в файл OUTPUT.TXT. В i-й строке выводится длина второго по минимальности пути из столицы до i+1-го города. Если кратчайших путей несколько, то эта длина совпадает с длиной кратчайшего пути. Повторение городов в пути допускается. Если другого пути не существует, выводится строка со словом No. Таким образом, файл OUTPUT.TXT состоит из N-1 строки.

Ограничения: $2 \le N \le 50$; $M \ge 2$; $0 \le C_{ij} \le 1000$.

Примеры

Ввод 1

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

Ввод 2

```
4
1 2 3
1 3 2
2 3 3
2 4 3
3 4 5
4 3 4
4 1 6
```

Вывод 1	Вывод 2
No	15
6	6
7	7

Подсказка. Модифицировать алгоритм Дейкстры.

12.5. Диспетчер процессов (5)

В любой момент времени операционная система может исполнять произвольное количество независимых процессов, и время, за которое каждый процесс будет выполнен, не зависит от того, сколько процессов выполнялось одновременно с ним. Однако одновременно могут выполняться только независимые процессы. В некоторых случаях процесс *A* может использовать данные, полученные процессом *B* (и произвольным количеством других процессов). Тогда к моменту старта процесса *A* все процессы, от которых он зависит, уже должны отработать.

По времени исполнения нескольких процессов и таблице зависимости между этими процессами нужно вычислить время исполнения заданной группы процессов.

Ввод из файла INPUT.TXT. В первой строке находится число N ($2 \le N \le 20$) — количество процессов, которые нужно выполнить. Во второй строке через пробел записаны N чисел от 1 до 1000; *i*-е число - это время в миллисекундах, требуемое для исполнения *i*-го процесса. Далее идут N строк, описывающих зависимости между процессами. В каждой из них находятся по N символов 'Y' и 'N' (заглавных латинских букв). Символ 'Y' в *i*-й из этих строк на *j*-м месте означает, что для запуска *i*-го процесса требуется завершить *j*-й. Символ 'N' означает, что *i*-й процесс не зависит от *j*-го напрямую.

Вывод в файл OUTPUT.TXT.В единственной строке вывести минимальное время в миллисекундах, требуемое для выполнения всех процессов, или –1, если при заданной таблице зависимостей группу процессов нельзя выполнить.

Примеры

Ввод 1	Ввод 2
3	2
100 200 300	10 10
NNN	NY
NNN	YN
YYN	
Вывод 1	**Вывод 2**
500	-1

12.6. Жизнь на Марсе (8)

При высадке на Марс было основано N поселений. Каждое из них равномерно расширялось во все стороны на L марсианских миль в месяц. Постепенно поселения начали сливаться друг с другом, получая общее название. Какое минимальное время с момента высадки потребуется для того, чтобы на Марсе осталось не более K поселений?

Ввод из файла INPUT.TXT. В первой строке задаются через пробел три целых положительных значения: начальное количество поселений N ($1 \le N \le 1000$), число K ($1 \le K \le 10$, $K < N$) и скорость роста поселений L ($1 \le L \le 100$). Далее в следующих N строках содержатся через пробел целые координаты поселений X_i, Y_i ($-1000 \le X_i, Y_i \le 1000$) в марсианских милях.

Вывод в файл OUTPUT.TXT.В единственной строке вывести с точностью до 2 знаков минимальное время в месяцах с момента высадки, необходимое для того, чтобы в результате слияния осталось не более K поселений.

Ограничения. Время работы на одном тесте до 2 с. Объем используемой памяти: 64 мегабайта.

Пример

Ввод
3 2 1
-1 1
2 1
2 5

Вывод
```
1.50
```

12.7. Шпионские страсти (9)

Резидент разведки руководит сетью секретных агентов, контактируя только с некоторыми из них. Каждый агент имеет право передавать информацию определенному кругу других агентов. Множество тех партнеров, которым агент передает сведения, не обязано совпадать с множеством принимающих от него информацию агентов. Полученные данные могут распространяться далее и, в конечном счете, доставляются резиденту. Резидент оплачивает каждую передачу сведений от одного агента другому. Величина оплаты зависит от пары агентов и от того, кто из них принимает информацию. Таким образом, общая оплата доставки информации зависит только от цепочки агентов, участвующих в доставке, и действующих расценок.

Стало известно, что определенный агент добыл ценные секретные сведения. В целях надежности резидент решил получить информацию по двум независимым каналам, то есть через разных агентов. Требуется найти минимальный размер затрат резидента для оплаты передачи информации по двум цепочкам агентов, которые начинаются с удачливого агента и заканчиваются резидентом, не пересекаясь друг с другом.

Ввод из файла INPUT.TXT. Первая строка содержит три целых положительных числа N, A и B, разделённых пробелом: N – количество агентов, включая резидента, A – номер начального агента, а B – номер конечного агента, то есть резидента. Далее следуют N строк, описывающих связи агентов в виде матрицы стоимости C_{ij}. В i-й строке задаются через пробел стоимости передачи информации от агента с номером i агентам с номерами 1, 2, …, N соответственно в виде целых положительных чисел, то есть значения $C_{i1}, C_{i2}, \ldots, C_{iN}$. Бесплатной передачи информации между агентами не существует. Если i-й агент не связан с j-м, то в соответствующем месте ставится 0. Связи каждого агента с самим собой заполняются нулями, то есть главная диагональ матрицы стоимостей состоит из нулей.

Ограничения: $N \le 50$, $0 \le C_{ij} \le 10000$, время 2 с.

Вывод в файл OUTPUT.TXT. В единственной строке выводится минимальная суммарная стоимость передачи информации по двум непересекающимся цепочкам агентов. Если таких цепочек не находится, в файл OUTPUT.TXT выводится строка со словом No.

Примеры

Ввод 1	**Ввод 2**
4 1 4	4 1 3
0 2 3 9	0 5 2 0
1 0 0 6	2 0 4 7
1 2 0 5	0 1 0 0
0 0 0 0	3 7 0 0
Вывод 1	**Вывод 2**
16	No

12.8. Учебный план (6)

Учебный план включает перечень дисциплин. Задан список пар дисциплин. Отдельная пара показывает, что вторая дисциплина должна изучаться после первой. Составить список дисциплин учебного плана в порядке их изучения. В том случае, когда задание некорректно, т.е. в списке пар имеются циклы, выдать хотя бы один из них.

Ввод из файла INPUT.TXT. В первой строке задается число пар дисциплин N ($1 \le N \le 300$). В каждой из следующих N строк указываются через пробел два натуральных числа X_i, Y_i (X_i, $Y_i \le$ 1000), определяющих номера дисциплин. Первая дисциплина должна изучаться раньше второй.

Вывод в файл OUTPUT.TXT. В первой строке вывести Yes либо No – возможность расположения в списке дисциплин в порядке их изучения. При наличии такой возможности во второй строке выводится через пробел искомый список. Если задание некорректно, т.е. имеется цикл, то во второй строке выдается список номеров, образующих цикл. Первый и последний номера в этом списке должны совпадать.

Примеры

Ввод 1	**Ввод 2**

```
7                 8
1 2               1 2
1 3               1 3
2 5               2 5
3 4               3 4
4 2               4 2
3 2               3 2
6 4               6 4
                  5 3
Вывод 1           Вывод 2
Yes               No
1 6 3 4 2 5       2 5 3 4 2
```

12.9. Морские дьяволы (6)

Полигон для тренировки морских десантников представляет собой площадку в форме прямоугольника с водоемами и задается матрицей размером *M* x *N*. Каждый элемент матрицы содержит либо символ '@', обозначающий участок водной поверхности, либо символ '.' (точка), обозначающий участок суши. Подразделение морских дьяволов находится в клетке, соответствующей левому верхнему углу матрицы. Ему поставлена задача достичь участка, соответствующего правому нижнему углу матрицы. Десантники могут передвигаться в направлениях вдоль сторон полигона, не выходя за его пределы. Они планируют преодолеть как можно меньше клеток, занятых водой. Если это можно сделать по-разному, то предпочтительнее такой вариант, когда путь включает меньшее количество клеток суши.

Ввод из файла INPUT.TXT. В первой строке содержатся числа *M* и *N* ($1 \le M, N \le 50$), разделенные пробелами. В следующих *M* строках находится матрица, представляющая полигон, по *N* подряд идущих символов в строке. Гарантируется, что левый верхний и правый нижний углы матрицы соответствуют участкам суши.

Вывод в файл OUTPUT.TXT. В единственной строке вывести через пробел наименьшее число клеток *K*, которое подразделение должно преодолеть по воде, и для найденного значения *K* минимальное число клеток по суше *L,* включая начальную и конечную клетки.

Примеры

```
Ввод 1          Ввод 2
7 7             7 6
..@@...         .@@@..
..@@@..         ......
@.@@..@         @.@@.@
@@@...@         @@@..@
..@....         ..@...
..@...@         ..@...
....@..         ....@.
Вывод 1         Вывод 2
1 14            0 12
```

12.10. Детали (7)

Некоторое предприятие выпускает двигатели для автомобилей. Двигатель состоит ровно из *n* деталей, пронумерованных от 1 до *n*, при этом деталь с номером *i* изготавливается за p_i секунд. Специфика предприятия заключается в том, что одновременно может изготавливаться лишь одна деталь двигателя. Для производства некоторых деталей необходимо иметь предварительно изготовленный набор других деталей. Генеральный директор поставил перед предприятием задачу: за наименьшее время изготовить деталь с номером 1, чтобы представить ее на выставке. Требуется написать программу, которая по заданным зависимостям порядка производства между деталями найдет наименьшее время, за которое можно произвести деталь с номером 1.

Ввод из файла INPUT.TXT. Первая строка содержит число *n* ($1 \le n \le 100000$) – количество деталей двигателя. Вторая строка содержит *n* натуральных чисел $p_1, p_2 \ldots p_n$, определяющих время

изготовления каждой детали в секундах. Время для изготовления каждой детали не превосходит 10^9 секунд. Каждая из последующих n строк входного файла описывает характеристики производства деталей. Здесь i-ая строка содержит число деталей k_i, которые требуются для производства детали с номером i, а также их номера. Сумма всех чисел k_i не превосходит 200000. Известно, что не существует циклических зависимостей в производстве деталей.

Вывод в файл OUTPUT.TXT. В первой строке выходного файла должны содержаться два числа: минимальное время (в секундах), необходимое для скорейшего производства детали с номером 1 и число k деталей, которые необходимо для этого произвести. Во второй строке требуется вывести через пробел k чисел – номера деталей в том порядке, в котором следует их производить для скорейшего производства детали с номером 1.

Примеры

Ввод 1	**Ввод 2**	**Ввод 3**
3	2	4
100 200 300	2 3	2 3 4 5
1 2	1 2	2 3 2
0	0	1 3
2 2 1		0
		2 1 3
Вывод 1	**Вывод 2**	**Вывод 3**
300 2	5 2	9 3
2 1	2 1	3 2 1

12.11. Дороги Краскала (6)

В государстве Краскал имеется ряд населенных пунктов, связанных грунтовыми дорогами. Между любыми двумя пунктами может быть не более одной дороги. Решено заасфальтировать некоторые дороги. Стоимость асфальтирования каждой дороги известна. Требуется выбрать дороги для асфальтирования так, чтобы можно было проехать из каждого населенного пункта в любой другой по асфальту, а общая стоимость работы была минимальной. Использовать алгоритм Краскала.

Ввод из файла INPUT.TXT. В первой строке вводятся значение N ($1 \le N \le 10000$) и M ($1 \le M \le 100000$) – количество населенных пунктов и дорог. В каждой из следующих M строк вводится через пробел пара номеров пунктов в порядке возрастания в паре и стоимость асфальтирования дороги между ними.

Вывод в файл OUTPUT.TXT. В первой строке выводится общая стоимость работы. В следующих N -1 строках содержатся пары населенных пунктов, определяющие выбранные дороги. В каждой паре меньший номер должен быть первым. Список дорог должен быть отсортирован по возрастанию первого в паре номера, а при равном первом номере – по второму. В случае нескольких ответов выдать любой из них.

Пример

Ввод

```
4 5
2 3 5
1 4 5
1 2 6
1 3 1
3 4 3
```

Вывод

```
9
1 3
2 3
3 4
```

12.12. Олимпиада (7)

На олимпиаде по программированию предложено N задач. Требуется решить как можно больше задач. Если два участника решили одинаковое количество задач, то впереди оказывается тот, который имеет меньшее суммарное штрафное время. Это время определяется как сумма по всем зачтенным задачам времен их сдачи в минутах. Если, например, участник сдал 3 задачи на 10-й, 45-й и 123-й минутах, то штрафное время будет равно 10 + 45 +123 =178 минут.

Один из участников точно определил время, необходимое для решения каждой задачи. Кроме того он выявил, что решение части задач потребуется для сдачи других более сложных задач, поэтому эти простые задачи должны быть решены раньше. Каждое такое условие задается упорядоченной парой (a, b), которая определяет, что задача a должна быть решена раньше задачи b (если обе они будут решены). В этом случае $t_a \le t_b$, где t_a и t_b – необходимые затраты времени для решения задач a и b. Участник решает задачи последовательно, приступая к следующей после полного завершения предыдущей. Выведите наибольшее количество задач, которое сможет решить участник за время T, суммарное штрафное время при их решении и номера задач в порядке их решения.

Ввод. В первой строке входного файла содержатся два натуральных числа N и T ($1 \le N \le 200$; $1 \le T \le 10^6$). Во второй строке записаны N чисел t_i ($1 \le t_i \le 1000$), где t_i – время решения i-й задачи. В третьей строке записано число M ($0 \le M \le 1000$) – количество условий. Далее вводится M строк, каждая из которых содержит условие в виде двух целых чисел a_i и b_i ($1 \le a_i, b_i \le N$; $a_i \ne b_i$), которое обозначает, что задача a_i должна решаться раньше задачи b_i. Гарантируется, что время решения задачи a_i не больше времени решения задачи b_i. Все числа во входном файле целые.

Вывод. В первой строке выходного файла выведите два целых числа A и B, где A – наибольшее количество задач, а B – минимальное штрафное время. Во второй строке должны содержаться номера задач в порядке их решения. Если вариантов несколько, выведите любой из них.

Примеры

Ввод 1	Ввод 2	Ввод 3
1 1	3 10	4 2
1	1 1 1	1 2 1 2
0	3	1
	1 2	1 3
	2 3	
	3 1	
Вывод 1	**Вывод 2**	**Вывод 3**
1 1	0 0	2 3
1		1 3

12.13. Контрабандисты (10)

В некоторой местности имеется сеть автомобильных дорог. Каждая дорога соединяет два разных населенных пункта. Пункты пронумерованы от 1 до N. Известно, какой максимальный груз можно провезти по каждой из дорог. Движение по дорогам возможно в обоих направлениях.

Контрабандисты мечтают доставить как можно больше своего нелегального товара из пункта A в пункт B. Им стало известно, что полиция готовит засаду для конфискации товара. Засада может быть организована на какой-либо дороге или в некотором населенном пункте, исключая пункты A и B. Контрабандисты решили отправить товар двумя партиями по двум разным путям, которые не имеют общих населенных пунктов и дорог. Какой наибольший груз можно гарантированно доставить из пункта A в пункт B?

Ввод. В первой строке содержатся через пробел значения N, M, A и B, где N – количество населенных пунктов ($3 \le N \le 30$), M – число дорог, A и B – номера начального и конечного населенных пунктов ($A \ne B$). В каждой из следующих M строк находятся через пробел три значения U_i, V_i, H_i, задающие параметры очередной дороги. Значения U_i и V_i ($1 \le U_i < V_i \le 30$) определяют номера населенных пунктов на концах дороги, а H_i ($1 \le H_i \le 10^5$) указывает максимальный груз, который можно провезти по этой дороге. Все значения целые и положительные.

Вывод. Единственная строка должна содержать величину наибольшего гарантированного груза, который можно доставить из пункта A в пункт B с учетом пропускной способности дорог и

возможности перехвата полицией одной из партий. Если двух различных путей из пункта *A* в пункт *B* не существует, вывести в выходной файл No.

Пример

Ввод 1	**Ввод 2**	**Ввод 3**
3 3 3 1	4 2 3 4	3 2 1 2
1 3 5	1 2 5	1 2 5
1 2 7	1 3 3	1 3 3
3 2 4		
Вывод 1	**Вывод 2**	**Вывод 3**
4	No	No

12.14. Осмотр (4)

Для того, чтобы быть допущенным к занятиям в бассейне, необходимо предъявить справку, на которой должны поставить свои подписи *K* врачей. Некоторые врачи отказываются ставить подписи на справке до тех пор, пока на ней не распишется другой врач. Например, стоматолог отказывается ставить подпись, пока не будет принесена справка от психиатра, потому что однажды его укусил психически неуравновешенный студент. Составлен список, какому врачу нужны какие справки. Нужно определить, можно ли получить все справки, а также в какой последовательности необходимо обходить врачей.

Ввод. В первой строке файла INPUT.TXT содержится общее количество врачей ($1 \le K \le 100$). В следующих K строках описываются необходимые справки. Первое число *j* в *i*+1 строке входного файла означает, сколько справок нужно *i*-му врачу. Затем в той же строке, содержится *j* чисел - номера врачей, чьи подписи надо предварительно поставить, чтобы получить подпись *i*-го врача.

Вывод. Если подписи всех врачей собрать невозможно, то в выходной файл OUTPUT.TXT следует вывести NO. Если же все справки собрать возможно, то в первой строке выходного файла должно содержаться YES, а в следующих *K* строках - последовательность, в которой нужно получать справки. В случае нескольких решений выдать любое из них.

Пример

Ввод

```
4
1 2
0
2 1 4
1 1
```

Вывод

```
YES
2
1
4
3
```

12.15. Центр дерева (8)

В офисе фирмы Megasoft установлены *N* компьютеров с номерами от 1 до *N*, некоторые из них соединены между собой. Сообщение между соединенными компьютерами проходит в любом из двух направлений за 1 с. Компьютер, получив сообщение, сразу отправляет его всем соединенным с ним компьютерам. Компьютерная сеть устроена так, что между любыми двумя компьютерами есть путь, причем только один.

Найти номера всех компьютеров, с которых главный программист Гилл Бейтс может отправить сообщение так, чтобы максимальная задержка в получении сообщения была как можно меньше.

Ввод из файла INPUT.TXT. В первой строке вводится значение *N* ($1 \le N \le 100000$). В каждой из следующих *N*-1 строк вводится через пробел пара номеров компьютеров, обозначающая соединение.

Вывод в файл OUTPUT.TXT. В первой строке выводится значение *M* – количество искомых компьютеров. Во второй строке выдаются через пробел в порядке возрастания номера искомых компьютеров.

Пример

Ввод

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

Вывод

```
2
2 3
```

Указание. Предложить структуру данных, обеспечивающую быстрое нахождение листьев бескорневого дерева из условия задачи.