Тема 3.3. Поиск кратчайших маршрутов

Аннотация: Задачи поиска маршрутов в графе. Поиск маршрутов с минимальным числом ребер. Связные компоненты графа. Слабые и сильные орграфы. Вершинная связность и реберная связность. Расстояния в графе. Диаметр, радиус и центр графа. Расстояния между вершинами графа. Эксцентриситет вершины. Диаметр графа. Центр графа. Обходы графов. Эйлеровы графы, эйлеровы цепи. Гамильтоновы графы.

Расстояние в графах

Пусть $G = \langle V, E \rangle$ связный неориентированный граф, v_i, v_j – две его не совпадающие вершины. Длина кратчайшего (v_i, v_j)– маршрута называется ***расстоянием*** между вершинами v_i и v_j и обозначаются через $\rho(v_i, v_j)$. Положим $\rho(v, v) = 0$. Очевидно, что введенное таким образом расстояние удовлетворяет следующим аксиомам метрики:

$\rho(v_i, v_j) \geq 0$;

$\rho(v_i, v_j) = 0 \Leftrightarrow v_i = v_j$;

$\rho(v_i, v_j) = \rho(v_j, v_i)$ (симметричность);

$\rho(v_i, v_j) \leq \rho(v_i, u) + \rho(u, v_j)$ (неравенство треугольника).

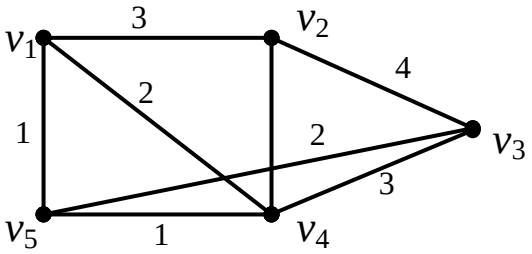
Матрицей расстояний называется матрица $P = (p_{ij})$, в которой $p_{ij} = \rho(v_i, v_j)$. Заметим, что $P^{\mathrm{T}} = P$, т.е. матрица P симметрична.

Зафиксируем вершину v_i. Для этой вершины величина $e(v_i) = \max\{\rho(v_i, v_j)\}$ называется ***эксцентриситетом вершины*** v_i. Т.е., эксцентриситет вершины равен расстоянию от данной вершины до наиболее удаленной от нее.

Максимальный среди всех эксцентриситетов вершин называется ***диаметром графа*** G и обозначается через $d(G) : d(G) = \max\{e(v_i)\}$. Вершина u называется периферийной, если $e(u) = d(G)$.

Минимальный из эксцентриситетов графа G называется ***радиусом*** и обозначается через $r(G) : r(G) = \min\{e(v_i)\}$. Вершина u называется центральной, если $e(u) = r(G)$. Множество всех центральных вершин графа называется его центром.

Пример: Найдем диаметр графа, изображенного на рисунке:



Решение: Матрица расстояний P имеет вид

$$\begin{pmatrix} 0 & 3 & 5 & 2 & 1 \\ 3 & 0 & 4 & 2 & 3 \\ 5 & 4 & 0 & 3 & 2 \\ 2 & 2 & 3 & 0 & 1 \\ 1 & 3 & 2 & 1 & 0 \end{pmatrix},$$

отсюда $e(v_1)=5,\quad e(v_2)=4,\quad e(v_3)=5,\quad e(v_4)=3,\quad e(v_5)=3$ и, следовательно $d(G)=5$. Вершины 1 и 3 являются периферийными.

Пример: Радиус графа из предыдущего примера равен 3, а его центром является множество $\{v_4, v_5\}$.

Будем предполагать, что в G отсутствуют контуры с отрицательным весом, поскольку, двигаясь по такому контуру достаточное количество раз, можно получить маршрут, имеющий вес меньше любого заведомо взятого числа, и тем самым задача нахождения расстояния становится бессмысленной.

Алгоритм Форда–Беллмана

В отличие от алгоритма Дейкстры, который будет рассмотрен позднее, алгоритм Форда–Беллмана допускает рёбра с отрицательным весом. Предложен независимо Ричардом Беллманом и Лестером Фордом.

Идея алгоритма

Шаг 1. На этом шаге инициализируются расстояния от исходной вершины до всех остальных вершин, как бесконечные, а расстояние от источника (обозначим v_0 принимается равным 0. Все расстояния помещаются в строку, в которой расстояния от источника – 0, а до всех остальных вершин равны w_{ij}, т.е. значениям матрицы расстояний (весов).

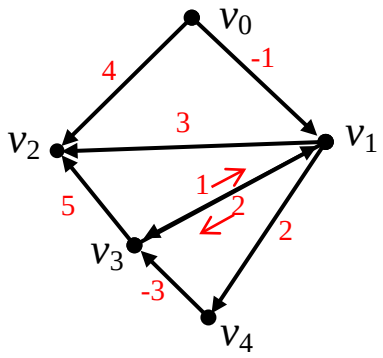
Шаг 2. Вычисляются самые короткие расстояния ($\rho(v_0,v)$). Для каждого ребра (u,v): если $\rho(v_0,v) > \rho(v_0,u) + w_{uv}$, то заменяем $\rho(v_0,v) = \rho(v_0,u) + w_{uv}$.

Шаг 2 выполняется $|V| - 1$ раз.

Шаг 3. Для каждого ребра (u,v) выполняем проверку $\rho(v_0,v) > \rho(v_0,u) + w_{uv}$. Если это условие выполняется, то в графе цикл отрицательного веса.

Идея шага 3 заключается в том, что шаг 2 гарантирует кратчайшее расстояние, если граф не содержит цикла отрицательного веса. Если мы снова переберем все ребра и получим более короткий путь для любой из вершин, это будет сигналом присутствия цикла отрицательного веса.

Пример: Найти кратчайшие расстояния от v_0 в графе:



Последовательно в таблице будем показывать задаваемые метки для вершин:

Шаг	v_0	v_1	v_2	v_3	v_4	Примечание
1	0	-1	4	∞	∞	Инициализация. Подставляем первую строку.
2.1.1	0	-1	2	∞	∞	Начинаем пересчет расстояний до каждой из вершин. Обновляем расстояние до v_2 (-1+3)
2.1.2	0	-1	2	∞	1	Вычисляем расстояние до v_4
2.1.3	0	-1	2	1	1	Вычисляем расстояние до v_3
2.2	0	-1	2	-2	1	Пересчитываем расстояние до v_3 (1-3), до v_2 и v_1 (они не меняется)

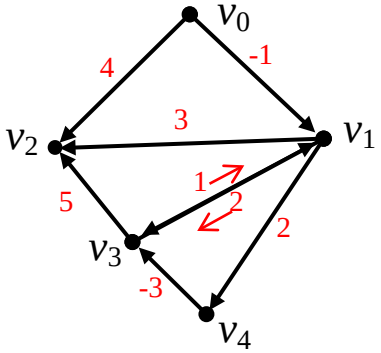
Третья и четвертая итерации ничего не меняют. Решение окончено.

Алгоритм Форда-Беллмана в матричном виде

Зададим строку $D^{(1)} = (d_1^{(1)}, d_2^{(1)}, \dots d_n^{(1)})$, полагая $d_1^{(1)} = 0$, $d_j^{(1)} = w_{ij}$, $i \neq j$. В этой строке $d_j^{(1)}$ $(j \neq i)$ есть вес w_{ij} дуги (v_i, v_j), если (v_i, v_j) существует, и $d_j^{(1)} = \infty$, если дуги (v_i, v_j) нет. Теперь определим строку $D^{(2)} = (d_1^{(2)}, d_2^{(2)}, \dots d_n^{(2)})$, полагая $d_j^{(2)} = \min\left\{d_j^{(1)}, d_k^{(1)} + w_{kj}\right\}_{k=1,\dots,n}$. Нетрудно заметить, что $d_j^{(2)}$ – минимальный из весов (v_i, v_j) – маршрутов, состоящих не более чем из двух дуг.

Продолжая процесс, на шаге s определим строку $D^{(s)} = (d_1^{(s)}, d_2^{(s)}, \dots d_n^{(s)})$, полагая $d_j^{(s)} = \min\left\{d_j^{(s-1)}, d_k^{(s-1)} + w_{kj}\right\}_{k=1,\dots,n}$. Т.е. вычисляя, например третий элемент в строке $d_j^{(s)}$ необходимо выбрать минимальный элемент среди соответствующего (третьего) элемента предыдущей строки $d_j^{(s-1)}$ и всеми суммами элементов третьего столбца матрицы $\left(w_{ij}\right)$ и соответствующих (по порядку следования) элементов предыдущей строки $d_j^{(s-1)}$. Искомая строка w-расстояний получается при $s = n-1$: $d_j^{(n-1)} = \rho(v_i, v_j)$. Действительно, на этом шаге из весов всех (v_i, v_j) –маршрутов, содержащих не более $n-1$ дуг, выбирается наименьший, а каждый маршрут более чем с $n-1$ дугами содержит контур, добавление которого к маршруту не уменьшает w –расстояние, так как мы предположили отсутствие контуров отрицательного веса. Работу алгоритма можно завершить на шаге k, если $D^{(k)} = D^{(k+1)}$.

Пример: Найти кратчайшие расстояния от v_0 в графе из предыдущего примера:



Матрица весов:

$$W = \begin{pmatrix} 0 & -1 & 4 & \infty & \infty \\ \infty & 0 & 3 & \infty & 2 \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & 5 & 0 & \infty \\ \infty & \infty & \infty & -3 & 0 \end{pmatrix}.$$

Первая строка $D^{(1)} = (0, -1, 4, \infty, \infty)$ - строка матрицы весов.

Начинаем пересчет: $D^{(2)} = (0, -1, 2, \infty, \infty)$. Третий элемент заменился, т.к. мы искали $\min\{4; 4+0; 3-1; 0+4; 5+\infty; \infty+\infty\} = 2$ (в этом выражении у нас первый элемент – это соответствующий элемент предыдущей строки $D^{(1)}$, а все следующие – это суммы элементов соответствующего (в данном случае третьего) столбца матрицы весов и соответствующих по порядку элементов строки $D^{(1)}$, например $d_1^{(1)} + w_{13}$ и $d_2^{(1)} + w_{23}$ и $d_3^{(1)} + w_{33}$ и т.д.

Далее: $D^{(3)} = (0, -1, 2, \infty, 1)$. Последний элемент заменился, т.к. $\min\{\infty; \infty+0; 2-1; \infty+2; \infty+\infty; 0+1\} = 1$.

Последняя строка: $D^{(4)} = (0, -1, 2, -2, 1)$. Дальше уже ничего меняться не будет (можете проверить). Это ответ.

Алгоритм Дейкстры

Алгоритм Дейкстры — алгоритм на графах, изобретённый нидерландским ученым Э. Дейкстрой в 1959 году. Находит кратчайшее расстояние от одной из вершин графа до всех остальных. Работает только для графов без рёбер отрицательного веса.

Алгоритм Дейкстры является более эффективным, чем алгоритм Форда-Беллмана, но используется только для взвешенных графов, в которых веса всех дуг не отрицательны.

Основная идея

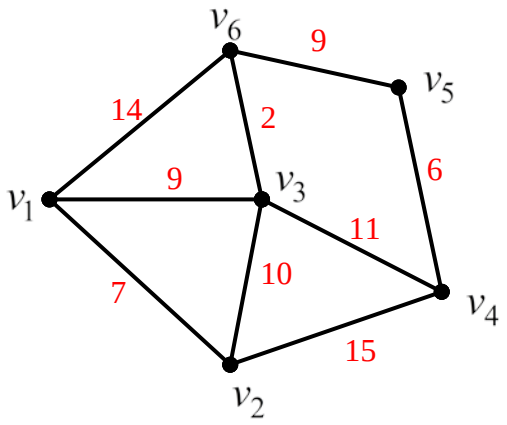
Его идея следующая: перед каждым этапом известно множество **отмеченных вершин** S, для которых кратчайшие пути найдены ранее; тогда на очередном этапе к нему добавляется вершина v, с самым коротким путем из v_0, проходящим по множеству S; после этого пересчитываются длины кратчайших путей из v_0 в оставшиеся вершины из $V \setminus S$ с учетом новой вершины v. Длина текущего кратчайшего пути из v_0 в v, проходящего по множеству S, заносится

в строку $D^{(k)}$. В конце работы в этой строке отыскиваются длины соответствующих кратчайших путей.

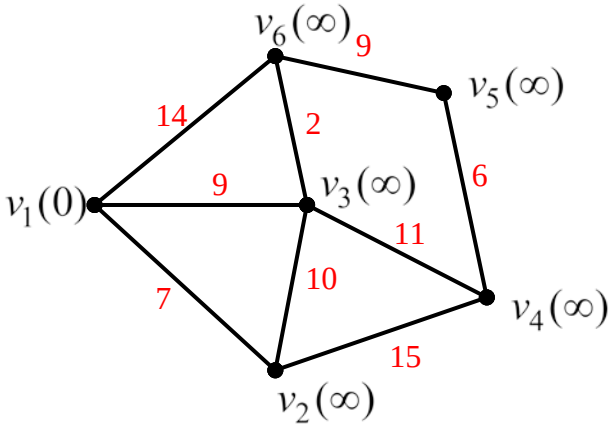
Графическая реализация

Рассмотрим графическую реализацию на примере.

Пример: Пусть требуется найти кратчайшие расстояния от 1-й вершины до всех остальных.



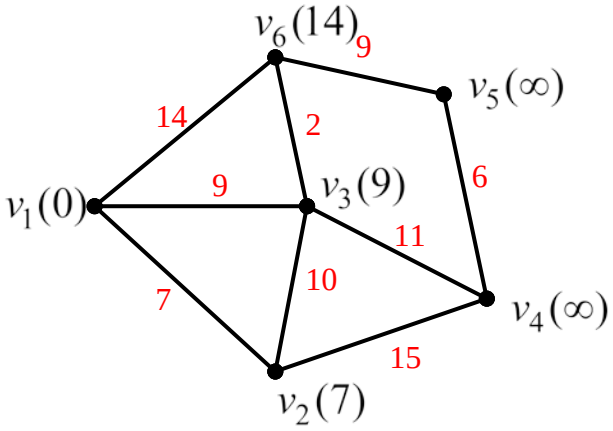
Инициализация: Метка самой вершины 1 полагается равной 0, метки остальных вершин – бесконечность (это вариант для составления программы для компьютера, для решения задачи на графах берут сразу расстояния до соседних вершин, т.е. то, что получится на следующем шаге).



Шаг 1. Соседи вершины с минимальной меткой (вершина v_1 с меткой 0) являются вершины 2, 3 и 6. Обходим соседей вершины по очереди.

Первый сосед v_1 – вершина v_2, потому что длина пути до неё минимальна. Длина пути $\rho(v_1,v_2)$ равна сумме кратчайшего расстояния до вершины v_1 (значению её метки, т.е. 0) и длины ребра, (v_1,v_2), то есть 0 + 7 = 7. Это меньше текущей метки v_2 (∞), поэтому новая метка 2-й вершины равна 7.

Аналогично находим длины пути для всех других соседей (вершины 3 и 6).



Все соседи v_1 проверены. Текущее минимальное расстояние до v_1 считается окончательным и пересмотру не подлежит. Вершина v_1 отмечается как посещенная (возьмем ее в квадратные скобки).

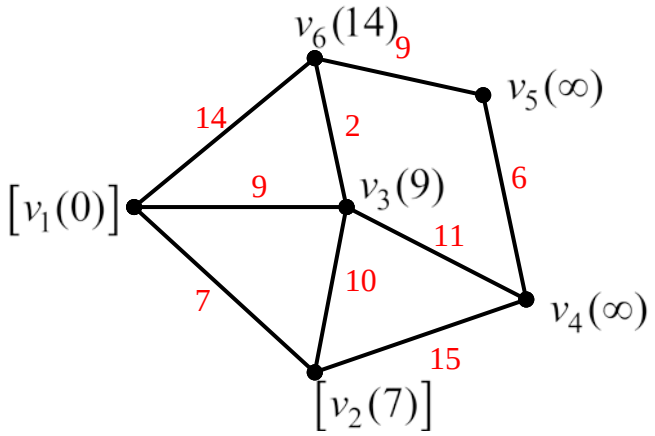
Шаг 2. Шаг 1 алгоритма повторяется. Снова находим «ближайшую» из непосещенных вершин. Это v_2 с меткой 7.

Снова пытаемся уменьшить метки соседей выбранной вершины, пытаясь пройти в них через v_2. Соседями вершины 2 являются вершины v_1, v_3 и v_4.

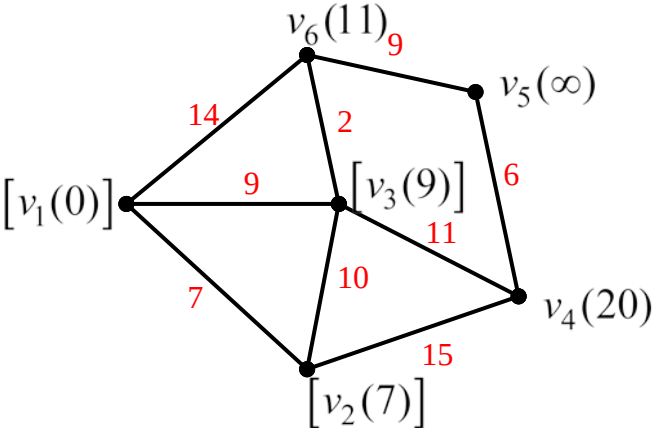
Вершина v_1 уже посещена. Следующий сосед вершины v_2 — вершина v_3, так как имеет минимальную метку из вершин, отмеченных как не посещённые. Если идти в неё через 2, то длина такого пути будет равна 17 (7 + 10 = 17). Но текущая метка третьей вершины равна 9, а $9 < 17$, поэтому метка не меняется.

Ещё один сосед v_2 — вершина v_4. Если идти в неё через 2-ю, то длина такого пути будет равна 22 (7 + 15 = 22). Поскольку $22 < \infty$, устанавливаем метку v_4 равной 22.

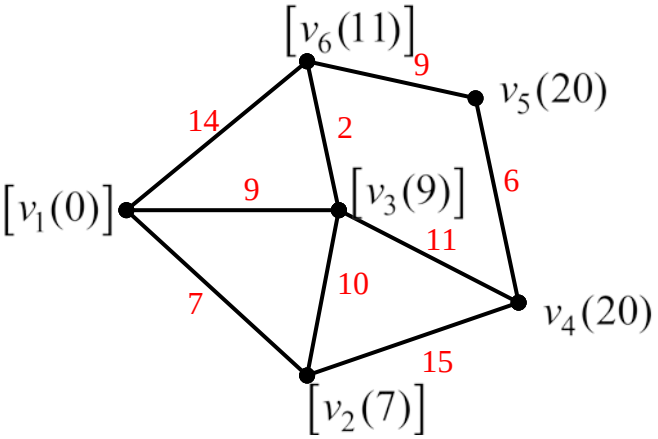
Все соседи вершины 2 просмотрены, помечаем её как посещенную.



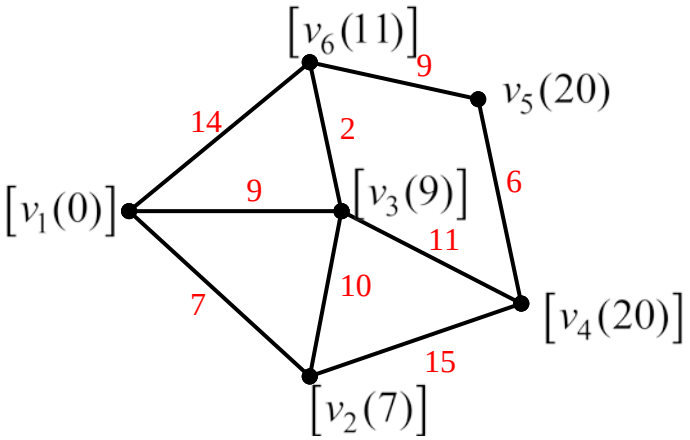
Шаг 3. Повторяем шаг алгоритма, выбрав вершину v_3. Получим:



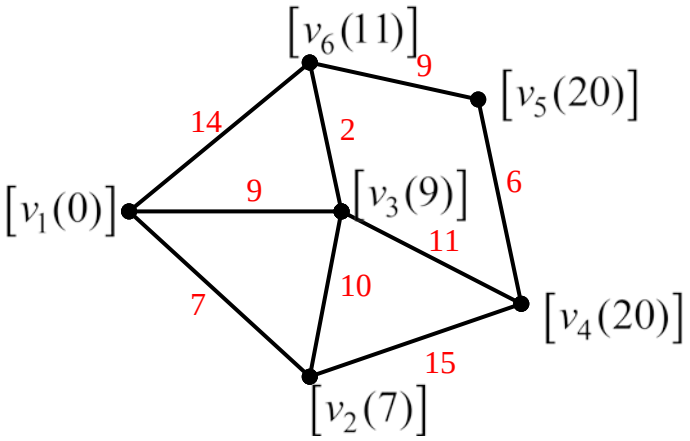
Шаг 4:



Шаг 5:



Шаг 6:



Таким образом, кратчайшие пути из вершины 1: (0,7,9,20,20,11).

Вывод кратчайшего пути получается разворачивая данную схему в обратную сторону. Рассматриваем конечную вершину (v_5 в нашей предыдущей схеме), и для всех вершин, с которой она связана, находим длину пути, вычитая вес соответствующего ребра из длины пути конечной вершины.

Так, v_5 имеет длину пути 20. Она связана с вершинами 6 и 4.

Для v_6 получим вес 20 — 9 = 11 (совпал).

Для v_4 получим вес 20 — 6 = 14 (не совпал).

Если в результате мы получим значение, которое совпадает с длиной пути рассматриваемой вершины (в данном случае v_6), то именно из нее был осуществлен переход в конечную вершину. Отмечаем эту вершину на искомом пути.

Далее определяем ребро, через которое мы попали в v_6. И так пока не дойдем до начала.

Если в результате такого обхода у нас на каком-то шаге совпадут значения для нескольких вершин, то можно взять любую из них — несколько путей будут иметь одинаковую длину.

Алгоритм Дейкстры в матричном виде.

Итак, пусть G – взвешенный граф, $W = (w_{ij}$ - матрица весов графа G, где $w_{ij} \geq 0$; v_i – источник. Зададим строку $D^{(1)} = (d_1^{(1)}, d_2^{(1)}, \ldots d_n^{(1)})$, полагая, как и в алгоритме Форда-Беллмана, $d_1^{(1)} = 0, d_j^{(1)} = w_{ij}, i \neq j$. Обозначим через T_1 множество вершин $M \setminus \{v_i\}$. Предположим, что на шаге s уже определены строка $D^{(s)} = (d_1^{(s)}, \ldots d_n^{(s)})$ и множество вершин T_S. Выберем теперь вершину $v_j \in T_s$ так, что $d_j^{(s)} = \min\left\{d_j^{(s)} \middle| v_k \in T_s\right\}$. Положим $T_{s+1} = T_s \setminus \{v_j\}$, $D_j^{(s+1)} = \left(d_1^{(s+1)}, \ldots, d_n^{s+1}\right)$, где $d_k^{(s+1)} = \min\left\{d_k^{(s)}, d_j^{(s)} + w_{jk}\right\}$, если $v_k \in T_{s+1}$, и $d_k^{(s+1)} = d_k^{(s)}$, если $a_k \notin T_{s+1}$. На шаге $s = n-1$ образуется строка $D^{(n-1)}$ w–расстояние между вершиной v_i и остальными вершинами графа: $d_j^{(n-1)} = \rho(v_i, v_j)$.

Отметим, что для реализации алгоритма Форда-Беллмана требуется порядка n^3 операций, тогда как в алгоритме Дейкстры необходимо выполнить порядка n^2 операций.

Пример: По заданной матрице весов графа найти величину минимального пути от вершины v_1 до каждой из вершин по алгоритму Дейкстры (в матричном виде):

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	11	∞	14	15	∞
v_2	∞	0	13	∞	∞	∞
v_3	∞	∞	0	∞	∞	15
v_4	8	7	11	0	9	∞
v_5	∞	12	10	∞	0	14
v_6	∞	∞	9	∞	∞	0

Решение:

Построим строку $T_1 = \{2,3,4,5,6\}$ - номера вершин до которых нужно вычислить длину пути и $D^{(1)} = (0,\underline{11},\infty,14,15,\infty)$ - расстояния

от x_1 до этих вершин (первоначально совпадает с первой строкой матрицы весов). Находим минимальный элемент (подчеркнут) и удаляем его номер из строки T. Пересчитываем D по правилу: $D^{(s)} = (d_1^{(s)}, \dots d_n^{(s)})$, где $d_k^{(s+1)} = \min\left\{d_k^{(s)}, d_j^{(s)} + w_{jk}\right\}$, (т.е., если мы считаем k-ый элемент в строк D, то мы выбираем минимальное значение среди того элемента, который занимал эту позицию в предыдущей строке D, а также среди всех сумм элементов столбца с номером k матрицы весов и соответствующих, по порядку следования, значений предыдущей строки D) если $a_k \in T_{s+1}$, и $d_k^{(s+1)} = d_k^{(s)}$, если $a_k \notin T_{s+1}$. Получим:

$T_2 = \{3,4,5,6\}$.

$D^{(2)} = (0,11,24,\underline{14},15,29)$.

$T_3 = \{3,5,6\}$.

$D^{(3)} = (0,11,24,14,15,29)$.

Строка $D^{(3)}$ не отличается от $D^{(2)}$, поэтому решение закончено даже несмотря на то, что в строке T остались элементы.

Ответ: минимальные расстояния от вершины 1 до всех остальных: $(0,11,24,14,15,29)$.

Опишем теперь алгоритм нахождения кратчайших маршрутов в бесконтурном графе. Так же, как и в алгоритме Дейкстры, для выполнения этого алгоритма необходимо порядка n^2 операций.

Прежде всего заметим, что в конечном бесконтурном графе $G = \langle V, E \rangle$ вершины можно перенумеровать так, что каждая дуга будет иметь вид (v_i, v_j), где $i < j$. Действительно в конечном бесконтурном графе всегда существует вершина v, в которую не заходит ни одна дуга. Обозначим эту вершину через v_1 и рассмотрим граф G_1, полученный из G удалением вершины v_1. Граф G_1 также является бесконтурным и, следовательно, содержит вершину, в которую не заходит ни одна дуга графа G_1. Продолжая это процесс для полученного графи и всех следующих, получим в результате искомую нумерацию $v_1, v_2, \dots, v_n$ вершин графа G.


Теорема

После того как какая-либо вершина v становится помеченной, текущее расстояние до неё $d_v^{(k)}$ уже является кратчайшим, и, соответственно, больше

меняться не будет.

Доказательство:

➢ Воспользуемся методом математической индукции.

Для первой итерации справедливость его очевидна — для вершины v_1 имеем $d_1^{(1)} = 0$, что и является длиной кратчайшего пути до неё.

Пусть теперь это утверждение выполнено для всех $k-1$ предыдущих итераций, т.е. всех уже помеченных вершин; докажем, что оно не нарушается после выполнения текущей (k-ой) итерации. Пусть v — вершина, выбранная на текущей итерации, т.е. вершина, которую алгоритм собирается пометить. Докажем, что $d_v^{(k)}$ действительно равно длине кратчайшего пути до неё (обозначим эту длину через $l(v)$).

Рассмотрим кратчайший путь P до вершины v. Понятно, этот путь можно разбить на два пути: P_1, состоящий только из помеченных вершин (как минимум стартовая вершина v_1 будет в этом пути), и остальная часть пути P_2 (она тоже может включать помеченные вершины, но начинается обязательно с непомеченной). Обозначим через p первую вершину пути P_2, а через q — последнюю вершины пути P_1.

Докажем сначала наше утверждение для вершины p, что соответствует $d_p^{(k)} = l(p)$. Однако это практически очевидно: ведь на одной из предыдущих итераций мы выбирали вершину q и находили кратчайшие маршруты из нее в соседние вершины. Поскольку (в силу самого выбора вершины p) кратчайший путь до p равен кратчайшему пути до q плюс ребро (p,q), то при поиске кратчайших маршрутов в соседние вершины из q величина $d_p^{(k)}$ действительно установится в требуемое значение.

Вследствие неотрицательности стоимостей рёбер длина кратчайшего пути $l(p)$ (а она по только что доказанному равна $d_p^{(k)}$) не превосходит длины $l(v)$ кратчайшего пути до вершины v. Учитывая, что $l(v) \le d_v^{(k)}$ (ведь алгоритм Дейкстры не мог найти более короткого пути, чем это вообще возможно), в итоге получаем соотношения:

$$d_p^{(k)} = l(p) \le l(v) \le d_v^{(k)}$$

С другой стороны, поскольку и p, и v — вершины непомеченные, то так как на текущей итерации была выбрана именно вершина v, а не вершина p, то получаем другое неравенство: $d_p^{(k)} \ge d_v^{(k)}$.

Одновременно эти неравенства могут выполниться только если $d_p^{(k)} = d_v^{(k)}$, а тогда из найденных до этого соотношений получаем и $d_v^{(k)} = l(v)$, что и требовалось доказать.◅

Эйлеровы граф и путь

Вернемся к задаче, с которой началась теория графов: «Задача о семи кенигсбергских мостах»: как пройти по всем городским мостам, не проходя ни по одному из них дважды:

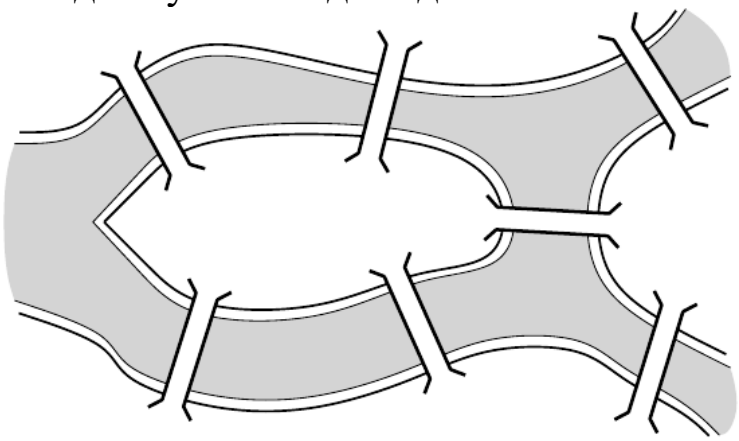
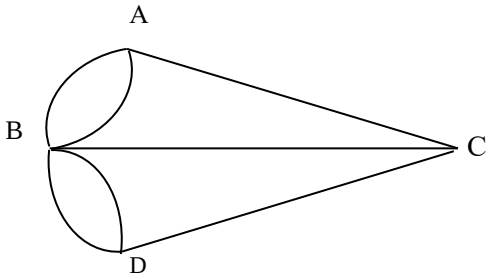


Рисунок 1 Задача о семи кенигсбергских мостах

Так как нас интересуют только переходы по мостам, мы можем считать A, B, C, D вершинами некоторого графа, ребра которого отвечают соответствующим мостам. Этот граф изображен ниже. Эйлер показал, что задача не имеет решения, т.е. не существует цикла, проходящего по всем ребрам точно по одному разу. Ведь если бы такой цикл существовал, то в каждой вершине графа было бы столько входящих в нее ребер, сколько и выходящих из нее, т.е. в каждой вершине графа было бы четное число ребер. Однако это условие, очевидно, не выполнено для графа, представляющего карту Кенигсберга.



Изложив решение задачи о кенигсбергских мостах, Эйлер в своей работе перешел к следующей общей проблеме теории графов: на каких графах можно найти цикл, содержащий все ребра графа, причем каждое ребро в точности по одному разу? Такой цикл называется ***эйлеровой линией*** или ***эйлеровым циклом***, а граф, обладающий эйлеровой линией, – ***эйлеровым графом***.

Эйлеровы графы можно построить, не отрывая карандаша от бумаги и не проводя одну и ту же линию дважды.

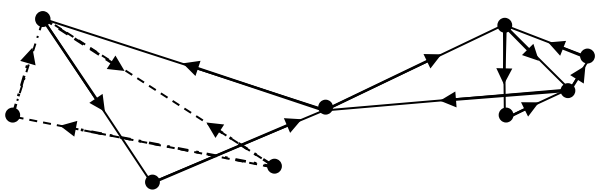
Теорема

Для того, чтобы граф был эйлеровым необходимо и достаточно, чтобы он был связным и степени всех его вершин были четные.

Доказательство:

- **Необходимость**. Для того, чтобы граф имел эйлерову линию, он должен быть **связным**. Как и в задаче о кенигсбергских мостах, ясно, что каждая эйлерова линия должна входить в каждую вершину и выходить из нее одно и то же число раз, т.е. степени всех вершин графа должны быть четными. Значит, чтобы граф был **эйлеровым**, необходимы два условия: **связность графа и четность степеней** всех его вершин.

Достаточность. Если начать путь из произвольной вершины графа G, то найдется цикл, содержащий все ребра графа. Пусть v_0 произвольная вершина графа G.



Из v_0 начнем путь 1 по одному из ребер и продолжим его, проходя каждый раз по новому ребру. Так как число ребер конечно, то этот путь должен окончиться, причем в вершине v_0 (на рисунке

выше путь l и направление его обхода показаны стрелками). Он не может закончиться в другой вершине, так как в этом случае все ребра инцидентные этой вершине окажутся пройденными по разу и их число будет нечетным. Этого быть не может, так как степени вершин четные.

Если путь l, замкнувшийся в v_0, проходит через все ребра графа, то мы получим искомый эйлеров цикл.

Если остались не пройденные ребра, то должна существовать вершина (не теряя общности назовем ее v_1), принадлежащая l и ребру, не вошедшему в l, так как в противном случае граф окажется не связанным, потому что вершина, соответствующая не пройденному ребру, не будет связана маршрутом с любой вершиной пути l. Так как степень вершины v_1 – четная, то число ребер, которым принадлежит v_1 и которые не вошли в путь l, тоже четное.

Начнем новый путь 2 из v_1 и используем только ребра, не принадлежащие 1. Этот путь кончится в v_1 (на рисунке путь 2 обозначен пунктирными стрелками). Это доказывается рассуждением, аналогичным тому, как доказывалось, что путь кончится в v_0.

Объединим теперь оба цикла: из v_0 пройдем по пути l к v_1, затем по циклу 2 и, вернувшись в v_1, пройдем по оставшейся части пути l обратно в v_0.

Если снова найдутся ребра, которые не вошли в путь, то найдем новые циклы. Число ребер и вершин конечно, процесс закончится.

Итак, приведено конструктивное доказательство достаточного условия существования эйлерового графа, т.е. приведен **алгоритм**, позволяющий отыскать эйлеров цикл, и показано, что он применим во всех случаях, допускаемых условиями теоремы.

Теорема доказана. ◅

Помимо конструктивного алгоритма, изложенного при доказательстве теоремы, рассмотрим еще один.

Алгоритм Флери построения эйлерового цикла

Пусть G – эйлеров граф; тогда следующая процедура всегда возможна и приводит к эйлеровому циклу. Выходим из произвольной

вершины и идем по ребрам графа произвольным образом, соблюдая лишь следующие правила:

1. стираем ребра по мере их прохождения и стираем также изолированные вершины, которые при этом образуются;
2. на каждом этапе идем по мосту только тогда, когда нет других возможностей. (Мост – это ребро, которое после удаления делает граф не связным. Хотя эйлеров граф не содержит мостов, но после удаления ребер, могут образоваться мосты!)

Можно ввести понятие ***эйлерова пути*** – это путь, когда все рёбра проходятся по одному разу, но без возвращения в исходную точку.

Если граф не обладает эйлеровым циклом, то можно поставить задачу об отыскании одного эйлерова пути или нескольких эйлеровых путей, в совокупности содержащих все рёбра графа.


Теорема

Для того, чтобы граф обладал эйлеровым путем необходимо и достаточно, чтобы он был связным и степени всех его вершин, кроме быть может двух, были четные. При этом одна из вершин с нечетной степенью будет началом эйлерова пути, а вторая – его концом.

Доказательство данной теоремы очевидно с учетом доказательства предыдущей теоремы.

Гамильтонов цикл и путь. Задача коммивояжера

Гамильтоновым циклом является такой цикл, который проходит через каждую вершину данного графа ровно по одному разу.

Гамильтоновым путём называется простой путь (путём без петель), проходящий через каждую вершину графа ровно один раз. Гамильтонов путь отличается от цикла тем, что у пути начальные и конечные точки могут не совпадать, в отличие от цикла. Гамильтонов цикл является гамильтоновым путём.

Название «гамильтонов цикл» произошло от задачи «Кругосветное путешествие» предложенной ирландским математиком Вильямом Гамильтоном в 1859 году. Нужно было, выйдя из исходной вершины графа, обойти все его вершины и вернуться в исходную точку. Граф представлял собой укладку додекаэдра, каждой из 20 вершин графа было приписано название крупного города мира.

Теорема (Дирака)

Если в простом графе с $n > 3$ *вершинами* $p(v) > n/2$ *для любой вершины v, то граф G является гамильтоновым.*

Доказательство:

➢ Добавим к нашему графу k новых вершин, соединяя каждую из них с каждой вершиной из G. Будем предполагать, что k - наименьшее число вершин, необходимых для того, чтобы полученный граф G стал гамильтоновым. Затем, считая, что $k > 0$, придем к противоречию.

Пусть ($v;p;w;\ldots;v$) гамильтонов цикл в графе G, где v, w - вершины из G, а p - одна из новых вершин. Тогда w не является смежной с v, так как в противном случае мы могли бы не использовать вершину p, что противоречит минимальности k. Более того, вершина, скажем, w, смежная вершине w, не может непосредственно следовать за вершиной v, смежной вершине v, потому что тогда мы могли бы заменить ($v;p;w;\ldots;v;w;v$) на ($v;v;\ldots;w;w;\ldots;v$), перевернув часть цикла, заключенную между w и v. Отсюда следует, что число вершин графа G, не являющихся смежными с w, не меньше числа вершин, смежных с v (то есть равно, по меньшей мере, $n/2+k$); с другой стороны, очевидно, что число вершин графа G, смежных с w, тоже равно, по меньшей мере, $n/2+k$. А так как ни одна вершина графа G не может быть одновременно смежной и не смежной вершине w, то общее число вершин графа G, равное $n+k$, не меньше, чем $n+2k$. Это и есть искомое противоречие. Теорема доказана. ◅

Вопросы для самопроверки:

1. Расстояние в графах. Основные понятия.
2. Задачи поиска маршрутов в графе.
3. Алгоритм Форда-Беллмана.
4. Алгоритм Декстры (графическая реализация).
5. Алгоритм Дейкстры в матричном виде.
6. Эйлеровы граф и путь. Алгоритмы построения эйлерового цикла.
7. Гамильтонов цикл и путь. Задача коммивояжера.