Математическое программирование

Математические основы сетевого планирования

Математические основы сетевого планирования

- Цель: освоение теоретических основ и практических навыков решения задач с применением основ сетевого планирования.

- Задачи:

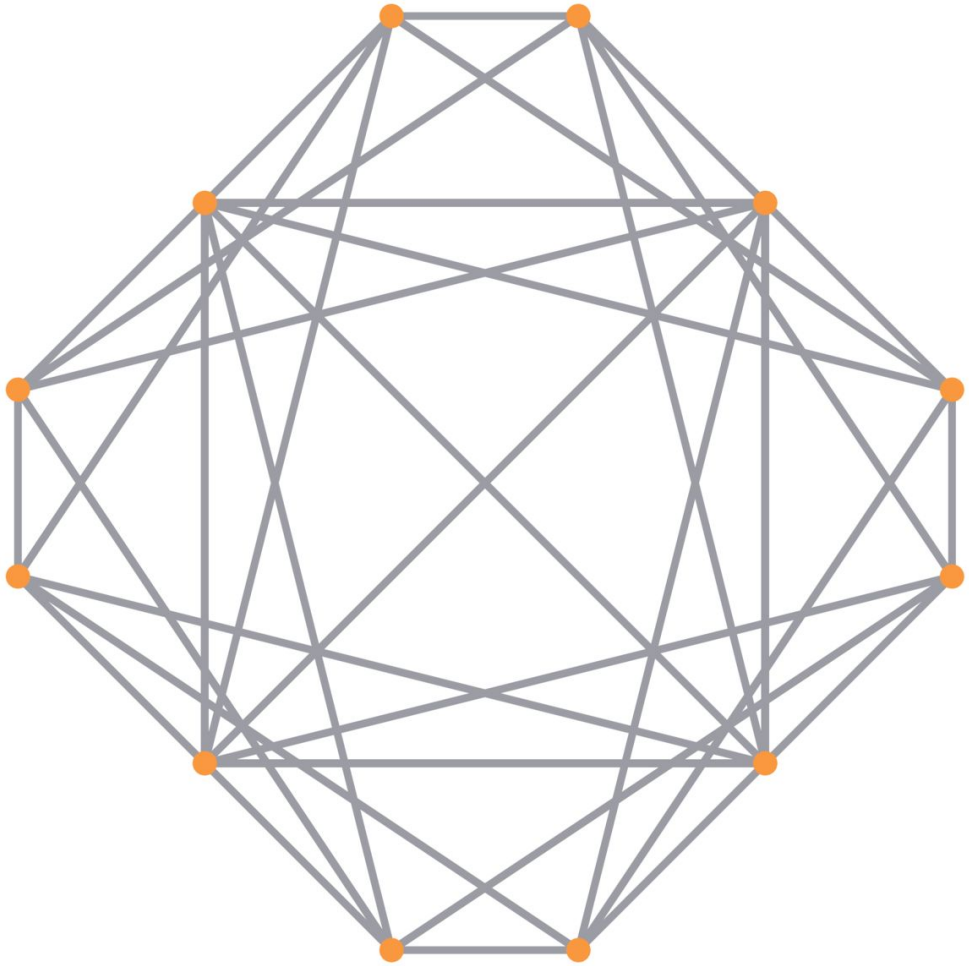
  - изучение основных понятий теории графов;
  - овладение навыками представления графов;
  - решение задач нахождения кратчайшего и максимального пути между вершинами графа.

План лекции

1. Основные понятия теории графов;
2. Способы представления графов;
3. Решение задачи нахождения кратчайшего пути между вершинами графа ;
4. Решение задач нахождения максимального пути между вершинами графа.

Основные понятия теории графов

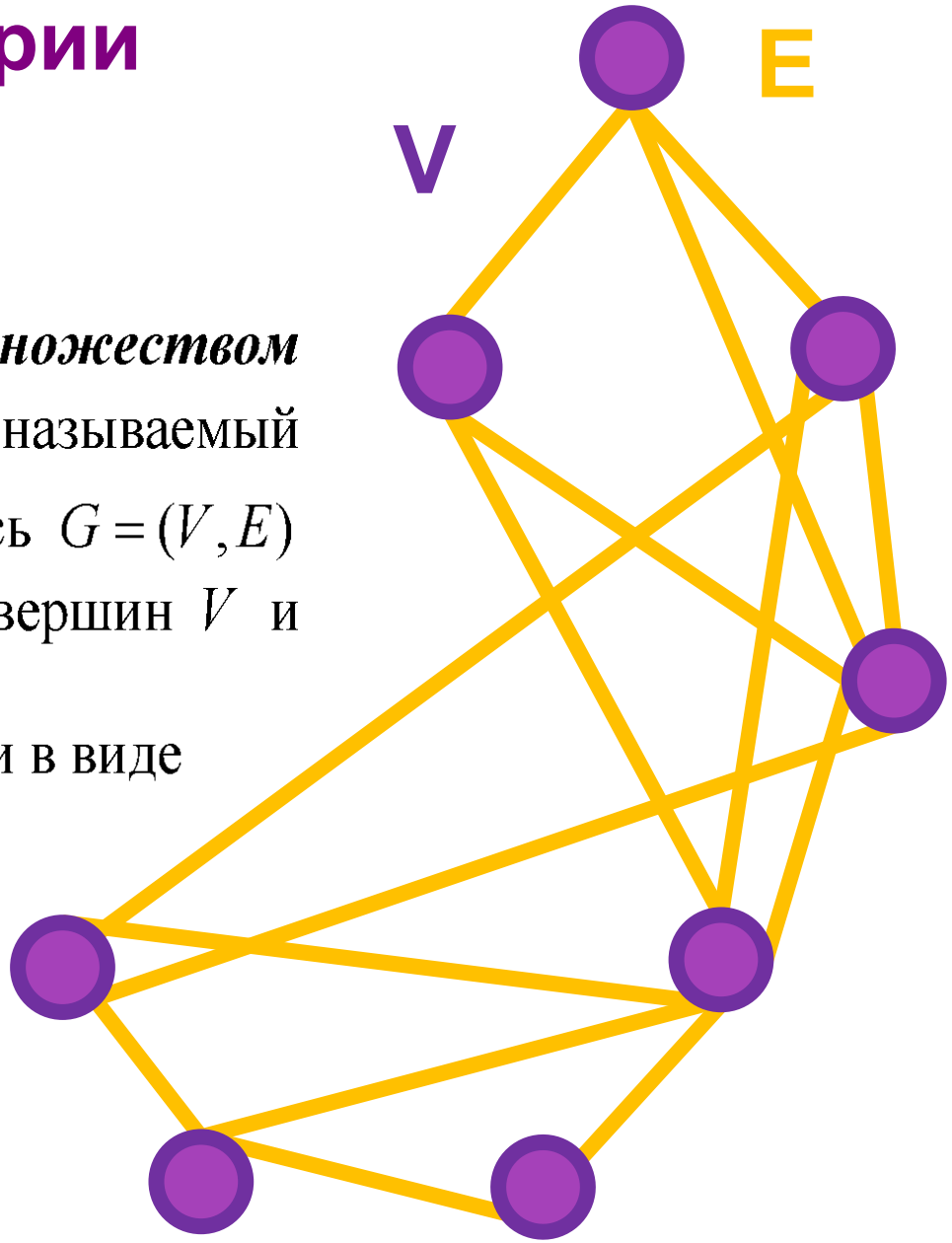
Граф – это математическая модель, с помощью которой удобно представлять бинарное отношение. Хотя теория графов получила свое развитие задолго до появления теории множеств как самостоятельной дисциплины, большое число задач теории отношений формулируются и решаются в рамках именно этой теории.

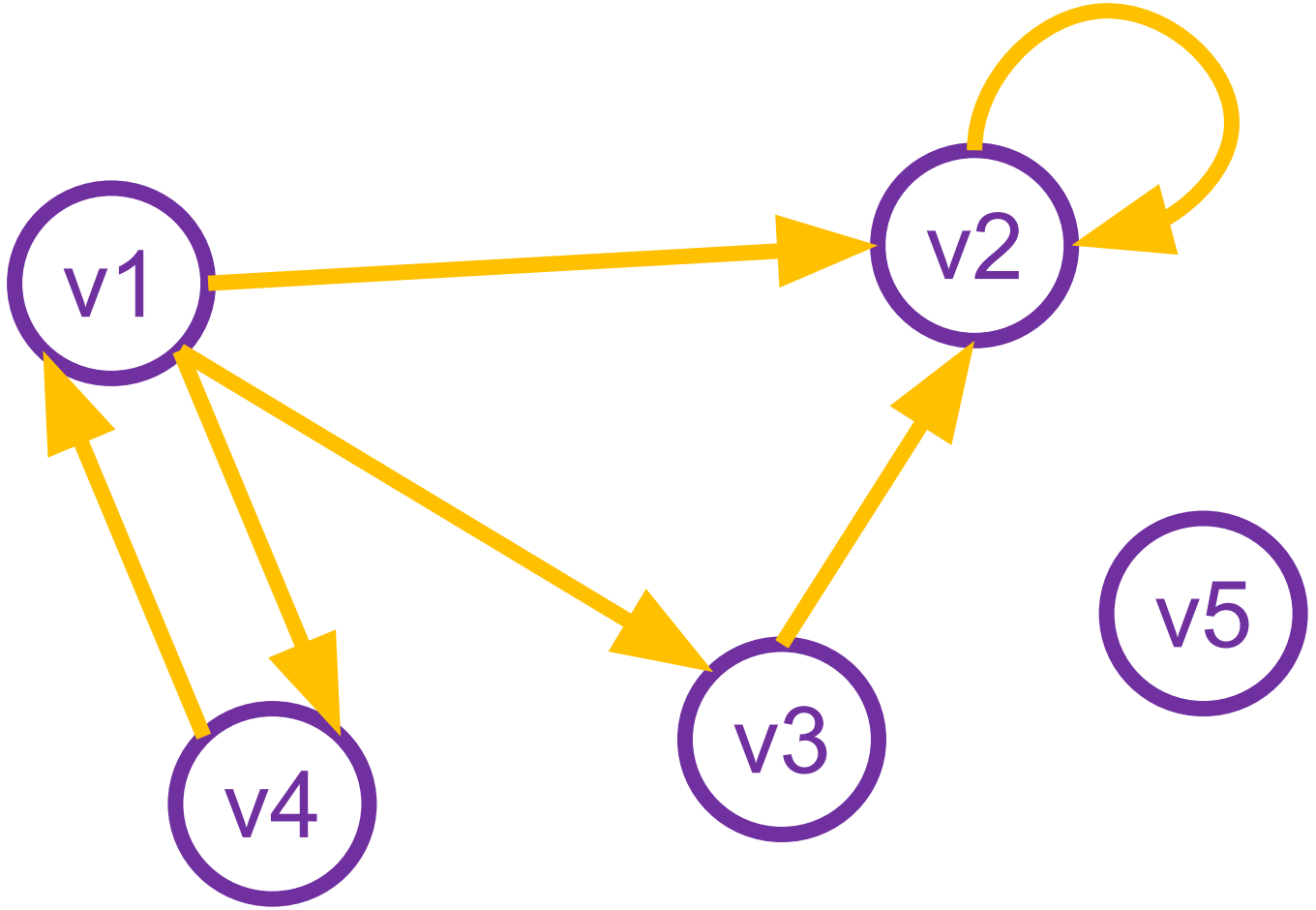


Основные понятия теории графов

Рассмотрим множество V, которое будем назвать ***множеством вершин***, и набор $E \subset V^2$ упорядоченных пар $\langle v_i, v_j \rangle \in V^2$, называемый ***множеством дуг***. Пара (V, E) называется ***графом***. Запись $G = (V, E)$ обозначает граф с именем G, состоящий из множества вершин V и множества дуг E.

Обычно вершины графа изображаются в виде точек, а дуги в виде соединяющих эти точки стрелок





G=(V,E), где $V=\{v_i \mid \overline{i=1,5}\}$ и

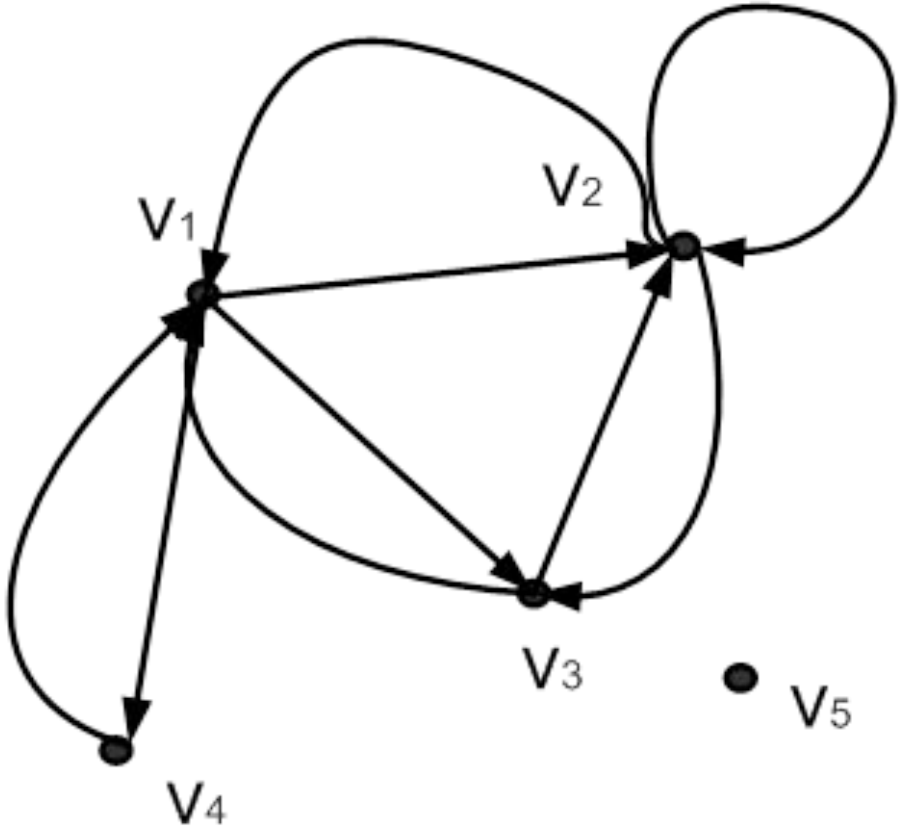
$E\{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_2\}, \{v_3, v_2\}, \{v_4, v_1\}\}$

Две соединенные дугой вершины графа, называются ***смежными вершинами***. Например, v_1 и v_2 смежные вершины графа $G = (V, E)$, т.к. есть дуга $\langle v_1, v_2 \rangle \in E$.

Считается, что дуга $\langle v_i, v_j \rangle$ ***выходит из вершины*** v_i и ***входит в вершину*** v_j. При этом говорят, что вершины v_i и v_j ***инцидентны*** дуге $\langle v_i, v_j \rangle$, а дуга $\langle v_i, v_j \rangle$ инцидентна вершинам v_i и v_j. Например, вершины v_2 и v_3 инцидентны дуге $\langle v_2, v_3 \rangle$, выходящей из v_2 и входящей в v_3.

Вершины v_i и v_j дуги $\langle v_i, v_j \rangle$ называют соответственно ***начальной*** и ***конечной*** вершинами этой дуги.

Ориентированный граф



Дуги, которые выходят и входят в одну и ту же вершину, называются ***петлями***.

$\langle v_2, v_2 \rangle$ - является петлей.

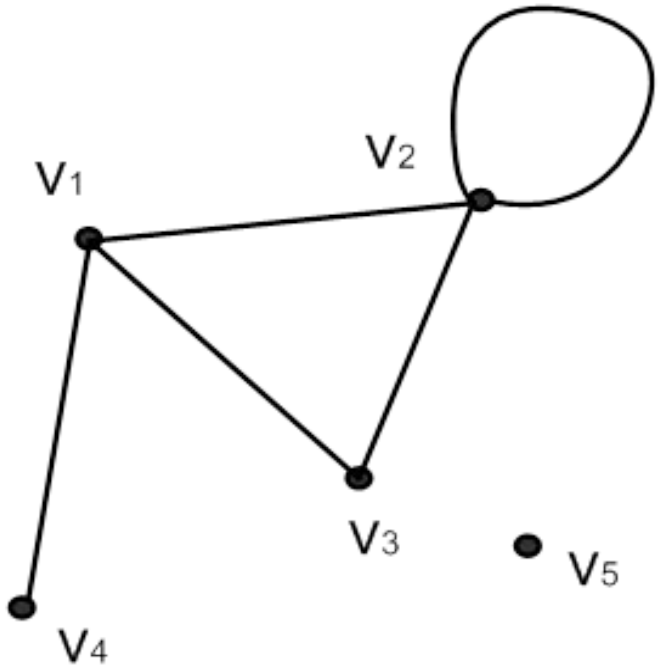
Вершины, не имеющие смежных, называются ***изолированными вершинами***.

v_5 –изолированная вершина.

Очевидно, что множество дуг ***E*** можно интерпретировать как бинарное отношение, а ***V*** – множество, на котором это бинарное отношение строится.

Если множество дуг ***E*** не является симметричным отношением, то такой граф называется ***ориентированным графом***.

Неориентированный граф



Матрица смежности

$$M_E = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Если множество дуг ***E*** – симметричное отношение, то соответствующий граф называется ***неориентированным графом***.

Симметричное отношение ***E***, которое может быть описано следующей матрицей M_E. Для того, чтобы разгрузить рисунок при изображении неориентированного графа, принято пару противоположных дуг изображать линией без стрелок.

Для того, чтобы подчеркнуть, что порядок вершин в неориентированном графе не имеет значения, при обозначении пар вершин, соединенных двумя противоположными дугами, используется запись с круглыми скобками: (v_i, v_j), а сами такие пары называют ***ребрами***.

В дальнейшем нас будут интересовать только ориентированные графы, т.к. именно они используются для моделирования сетевых графиков. Поэтому в дальнейшем изложение основ теории графов будет посвящено ориентированным графам.

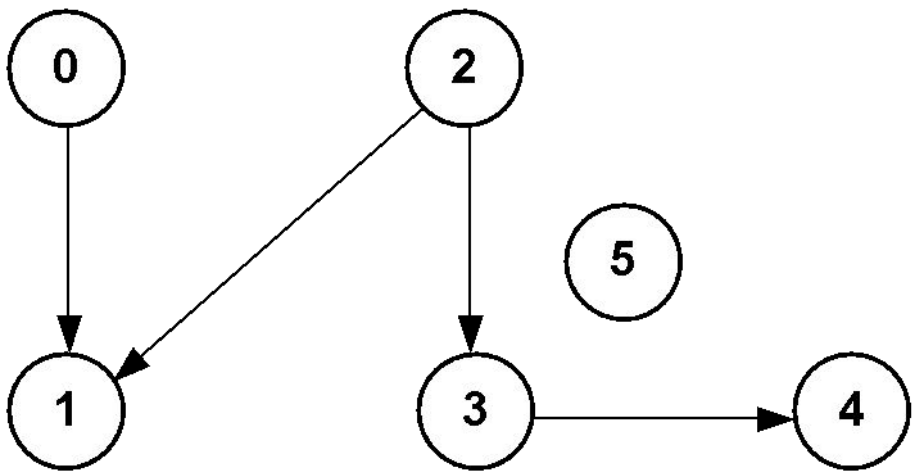
Существует три основных способа представления графов:

- ***матрица смежности,***
- ***матрица инцидентности***
- ***списки смежных вершин.***

Матрица инцидентности имеет размерность $m \times n$, где m – количество вершин, а n – количество дуг. Каждый (i, j) элемент матрицы принимает одно из трех значений: 0 – вершина i не инцидентна дуге j; 1 – дуга j выходит из вершины i; -1 – дуга j входит в вершину i.

$$G = (V, E), \quad V = \{0, 1, 2, 3, 4, 5\}, \quad E = \{\langle 0, 1\rangle_0, \langle 2, 1\rangle_1, \langle 2, 3\rangle_2, \langle 3, 4\rangle_3\}$$

а

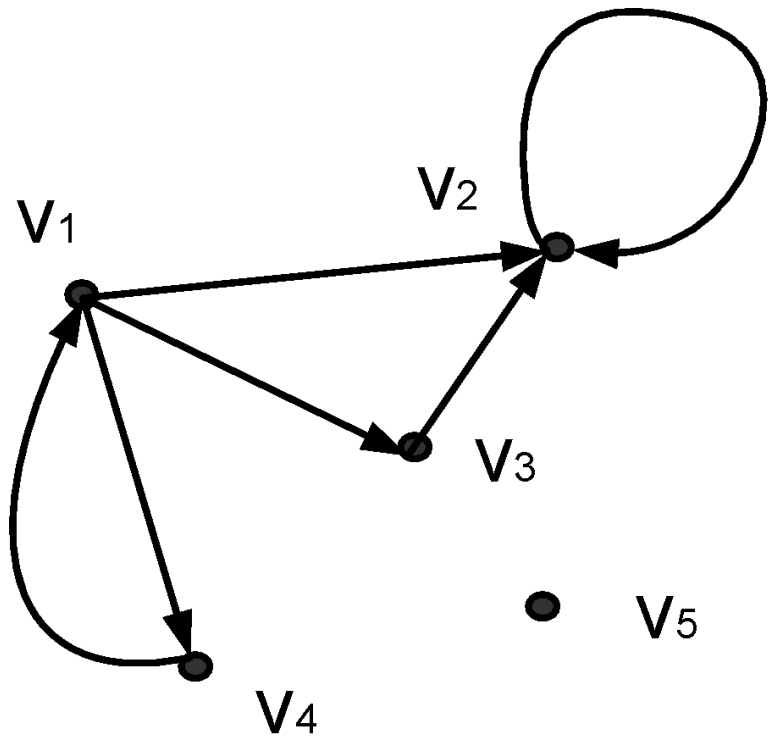


б

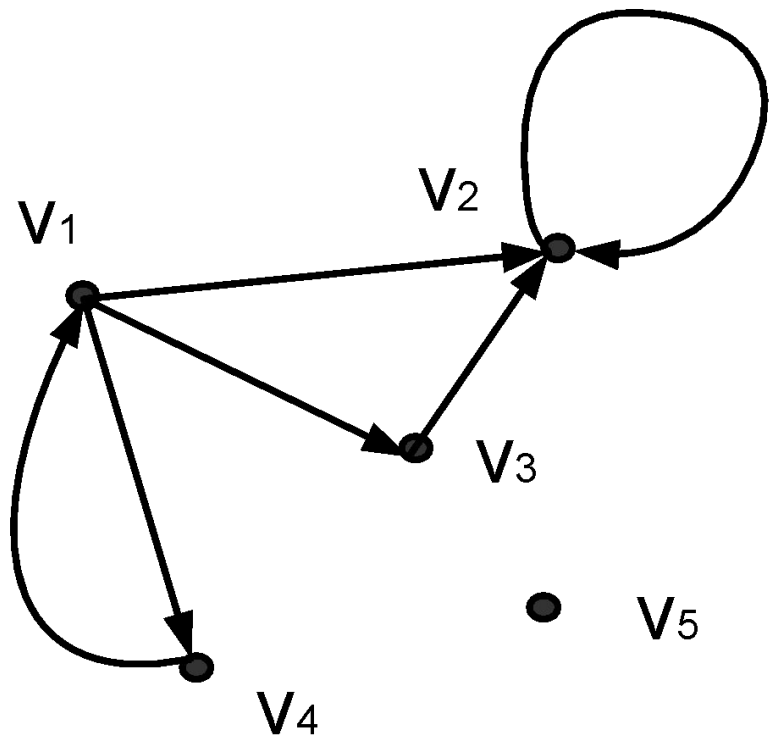
$$M_G = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

в

Следует обратить внимание, что петле в матрице ***M*** соответствует столбец с одной положительной единицей, что не всегда удобно при вычислениях. Например, при подсчете количества входящих в вершину дуг следует всегда учитывать, что может быть петля, которой нет соответствующей **-1**. Поэтому матрицы инцидентности применяются редко, особенно, если граф может иметь петли.



Списки смежных вершин представляют собой набор из m множеств, соответствующих m вершинам графа. Множество, соответствующее вершине i, $i = \overline{1,m}$ либо пустое, либо содержит номера вершин, в которые входит дуга, выходящая из вершины i. Например, списки смежных вершин для графа на рис. 1.1 можно представить в виде следующих пяти множеств: $S_1 = \{2,3,4\}$; $S_2 = \{2\}$; $S_3 = \{2\}$; $S_4 = \{1\}$; $S_5 = \varnothing$. Следует отметить, что этот способ является наиболее удобным при программировании большинства задач теории графов.



При программировании реальных задач теории графов, как правило, применяются матрица смежности и списки смежных вершин. При этом часто этой информации бывает недостаточно, т.к. она отражает только структуру графа. Если с вершинами и/или дугами графа связаны какие-то дополнительные характеристики, необходимо предусмотреть возможность их хранения.

Кратчайшие и максимальные пути между вершинами графа

Пусть $G=(V,E)$ – ориентированный граф, на множестве дуг которого определена функция $w: E \to \Re$. Функция w ставит каждой дуге $e_i \in E$ графа G в соответствие действительное число $x \in \Re$, которое часто называют ***весом дуги***. При этом саму функцию w обычно называют ***весовой функцией***, а граф – ***взвешенным графом***.

В общем случае граф $G=(V,E)$ может содержать несколько путей p_{ij}^1, p_{ij}^2, … p_{ij}^s из вершины v_i в вершину v_j. ***Весом*** $w(p_{ij}^k)$ ***пути*** p_{ij}^k ***взвешенного графа*** *называется сумма весов дуг, составляющих этот путь:* $w(p_{ij}^k)=\sum_{e_p \in p_{ij}^k} w(e_p)$.

Кратчайшим путем $\tilde{p}_{ij}$ ***из вершины*** v_i ***в вершину*** v_j *называется путь с минимальной весом* $\tilde{w}_{ij}=\min_k(w(p_{ij}^k))$. В общем случае в одном графе может быть несколько минимальных путей.

Поиск кратчайшего пути между двумя вершинами графа является одной из часто используемых в приложениях задач. Наиболее известными способами решения этой задачи являются ***алгоритмы Дейкстры***, ***Беллмана– Форда*** и ***Флойда – Уоршолла***.

Алгоритм Дейкстры

Алгоритм Дейкстры решает задачу о кратчайших путях во взвешенном графе $G=(V,E)$ с дугами неотрицательного веса $w(e_i)\geq 0$, $e_i \in E$, $i=\overline{1,|E|}$ из вершины s во все достижимые вершины этого графа. Алгоритм последовательно преобразовывает исходный граф G в дерево кратчайших путей G'. ***Дерево кратчайших путей*** – это граф $G'=(V',E')$, обладающий следующими свойствами:

1) $G' \subset G$;
2) G' – ориентированное дерево с корнем s;
3) V' – множество достижимых вершин графа G из s;
4) для каждой вершины $v_i \in V'$ путь из s в v_i в дереве G' является кратчайшим путем в графе G.

Алгоритм Дейкстры предполагает, что граф $G=(V,E)$ задан с помощью списков смежных вершин, а также известна весовая функция $W(v_i,v_j)=\begin{cases} w(v_i,v_j)\in\Re^+, \langle v_i,v_j\rangle\in E \\ +\infty, \langle v_i,v_j\rangle\notin E \end{cases}$, заданная на множестве $V\times V$. Функция W ставит в соответствие каждой паре вершин $\langle v_i,v_j\rangle$ вес $w(v_i,v_j)$ дуги, соединяющей эти вершины, или $+\infty$, если такой дуги нет в графе G. Кроме того, задана вершина s, относительно которой определяются все кратчайшие пути.

Для пояснения работы алгоритма Дейкстры будем использовать следующие обозначения.

$A[v]$ – множество вершин графа G, смежных с вершиной $v \in V$.

$d[v]$ – верхняя оценка кратчайшего пути из вершины s в вершину $v \in V$. В начале работы алгоритма $d[s] = 0$, а для всех остальных $v \in V \setminus \{s\}$ оценка $d[v] = +\infty$.

$p[v]$ – вершина, предшествующая вершине $v \in V$ в дереве кратчайших путей. В начале работы алгоритма для всех $v \in V$ $p[v] = nil$, где nil – специальный символ, обозначающий пустоту.

S – множество вершин $S \subset V$. Вначале множество S пустое: $S = \varnothing$. В процессе работы алгоритма S пополняется вершинами из V, для которых уже определен кратчайший путь из вершины s. После окончания работы алгоритма $S = V$.

Q – множество вершин $Q \subset V$. Вначале $Q = V$. В процессе работы алгоритма элементы этого множества – вершины, не добавленные во множество S: $Q = V \setminus S$. После окончания работы алгоритма $Q = \varnothing$.

extractQ – процедура извлечения элемента из множества Q с наименьшим текущим значением $d[v]$.

relax(u, v) – процедура релаксации, которая определена для произвольных двух вершин $\{u, v\} \subset V$ графа G. Релаксация состоит в следующем: значение $d[v]$ уменьшается до $d[u] + w(u, v)$ в том случае, если значение $d[u] + w(u, v)$ меньше текущего значения $d[v]$.

$\delta(v)$ – кратчайший путь из вершины s в вершину $v \in V$.

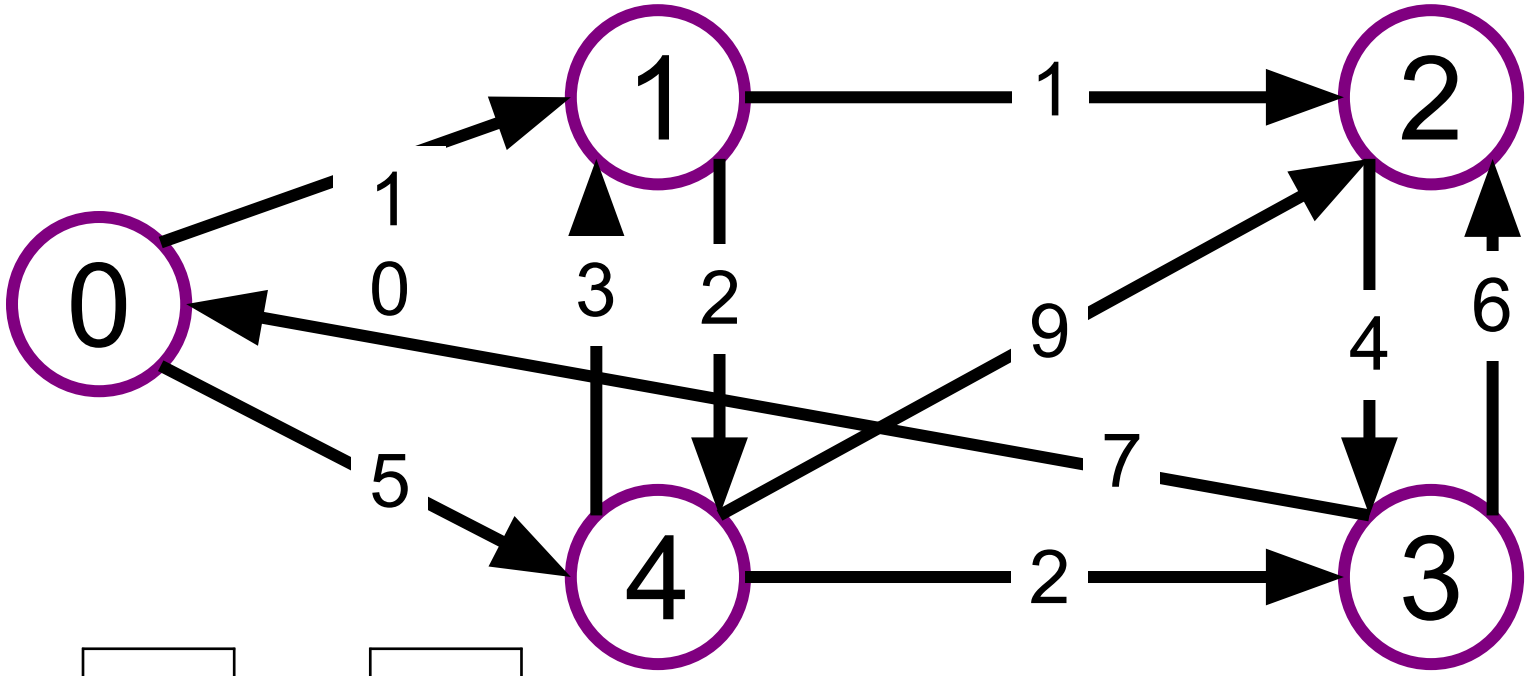
Выполнение алгоритма Дейкстры состоит из двух этапов. На первом этапе инициализируются S, Q, d и p значениями, описанными выше. Второй этап алгоритма опишем с помощью мнемокода, как показано на рис. 1.6. Для удобства строки мнемокода пронумерованы.

```
//-- Dijkstra
//-- вычисления кратчайших путей в графе
1  while Q ≠ ∅
2  {
3    u = extractQ
4    S = S ∪ {u}
5    for для всех v ∈ A(u)
6     {
7       relax(u,v)
8     }
9  }
```

Основной цикл алгоритма (строки 1–9) на рис. выполняется до тех пор, пока все вершины графа не будут извлечены из множества ***Q*** (строка 1). Извлечение вершин из множества ***Q*** осуществляется с помощью процедуры *extractQ* (строка 3).

Извлеченная вершина помещается сначала в переменную ***u***, затем во множество ***S*** (строка 4). Далее выполняется внутренний цикл (строки 5–8), в котором для всех вершин, имеющих входящие дуги с начальной вершиной ***u***, выполняется процедура релаксации.

Результатом выполнения алгоритма Дейкстры являются массивы ***p***[***v***] и ***d***[***v***], состоящие из |***V***| элементов. Массив ***p***[***v***] позволяет построить граф кратчайших путей, а каждый элемент массива ***d***[***v***] содержит вес кратчайшего пути между вершинами ***s*** и ***v***.



	D[v]
0	0
1	nil
2	nil
3	nil
4	nil

P[v]
nil
nil
nil
nil
nil

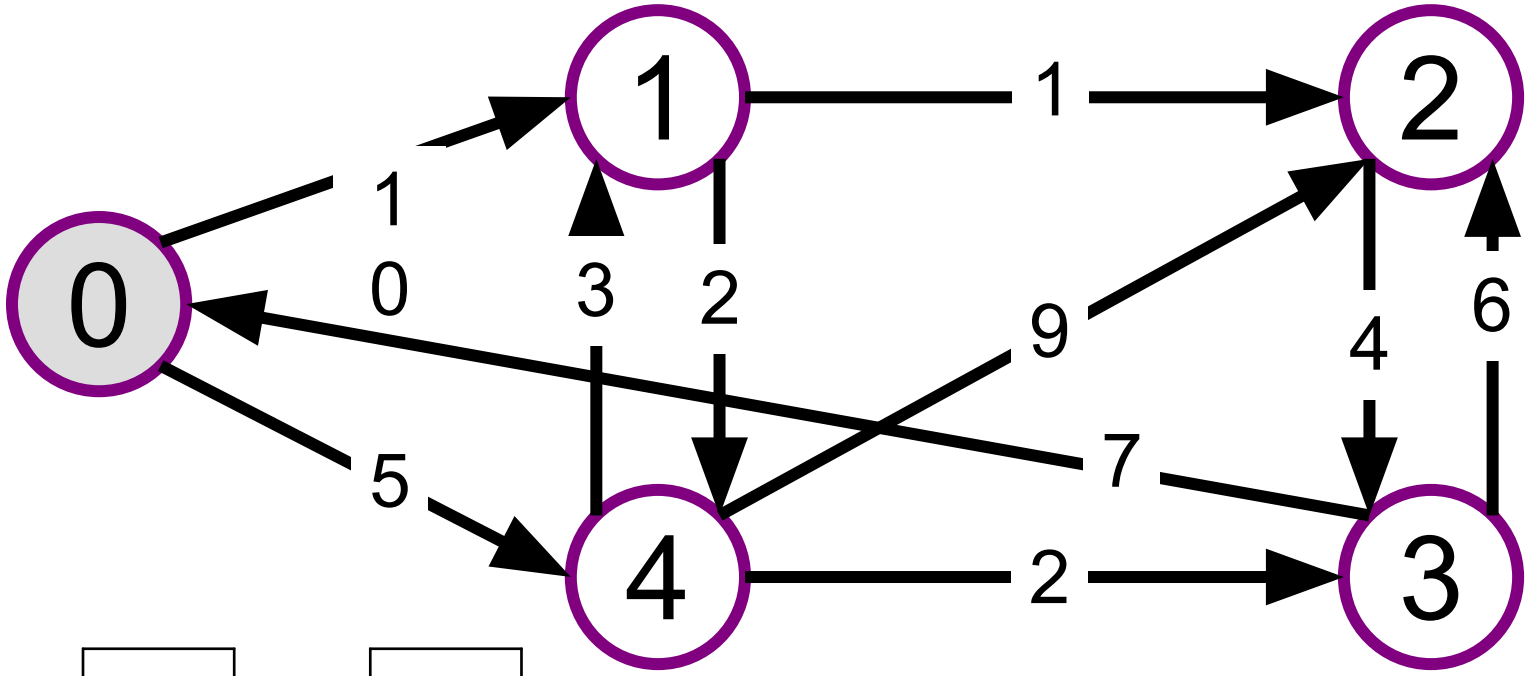
Q
0
1
2
3
4

S

Приведен пример решения задачи поиска кратчайшего пути в графе с помощью алгоритма Дейкстры. Изображен исходный граф и проинициализированные массивы ***d***, ***p***, ***Q*** и ***S***. В качестве меток для вершин графа используются числа от 0 до 4.

В примере на рис. осуществляется поиск кратчайших путей из вершины 0 до всех остальных вершин графа. По мере решения задачи, метки вершин графа перемещаются из массива ***Q*** в массив ***S***. В массиве ***d*** формируется вес пути для каждой вершины, а в массиве ***p*** – список предшествующих вершин.

Результат решения представляет собой дерево кратчайших путей. В этом дереве из вершины 0 до любой другой вершины графа существует единственный путь, который является кратчайшим.

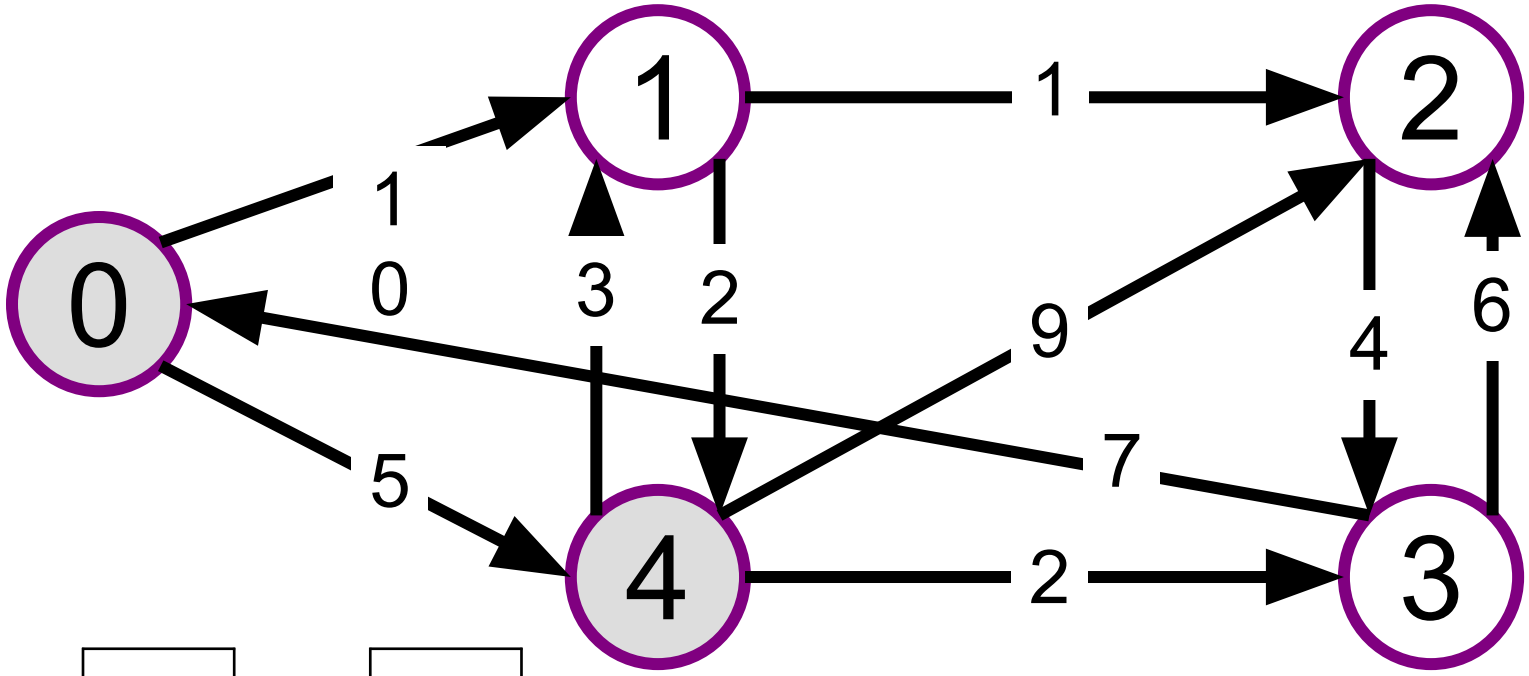


	D[v]
0	0
1	10
2	∞
3	∞
4	5

P[v]
nil
0
nil
nil
0

Q
0
1
2
3
4

S
0

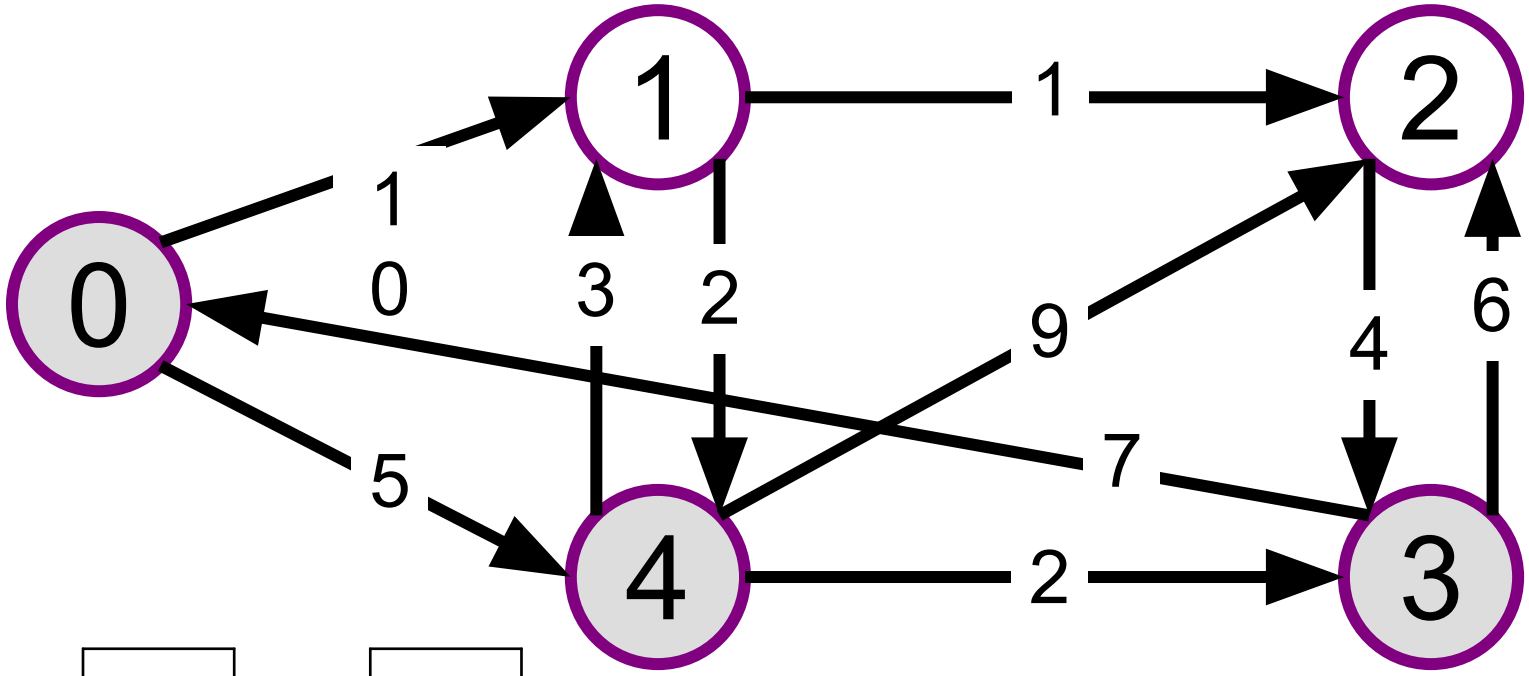


	D[v]
0	0
1	8
2	14
3	7
4	5

P[v]
nil
4
4
4
0

Q
1
2
3

S
0
4

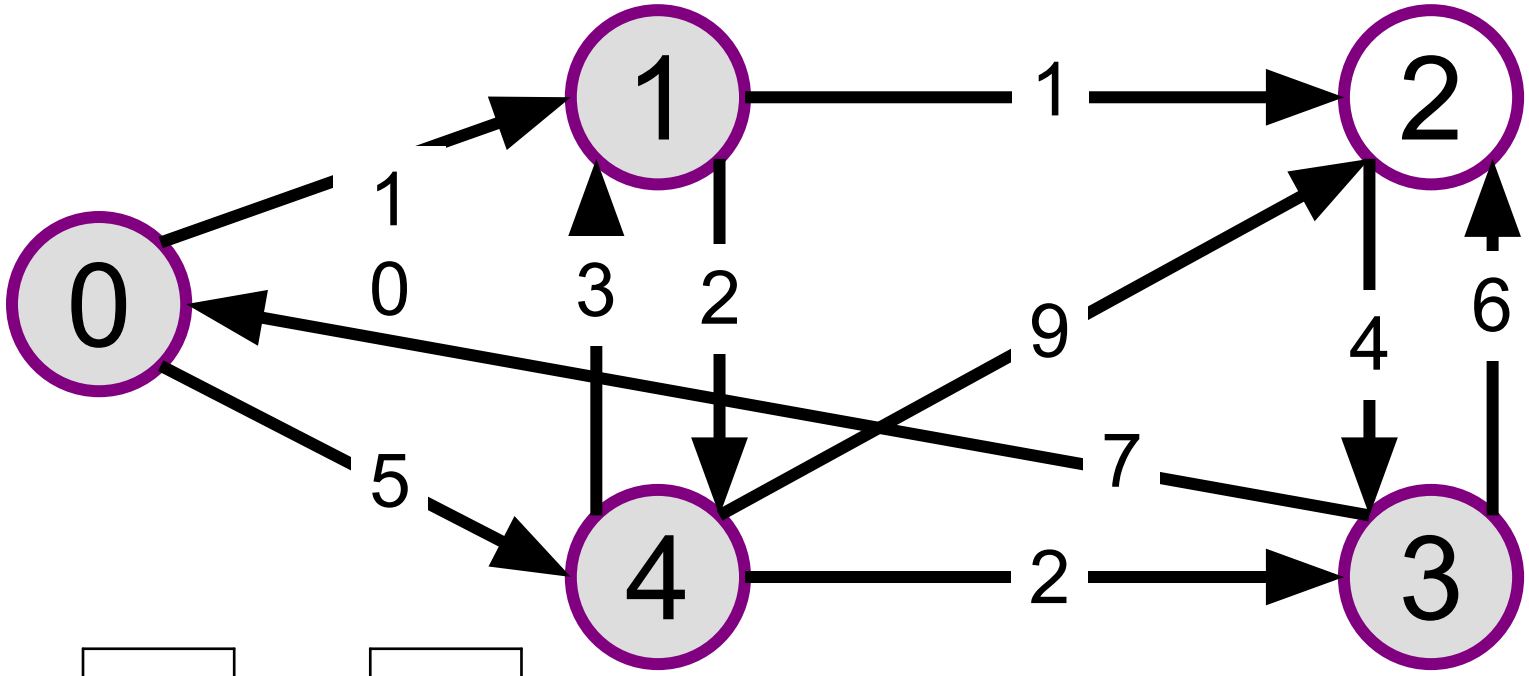


	D[v]
0	0
1	8
2	13
3	7
4	5

P[v]
nil
4
3
4
0

Q
1
2

S
0
4
3

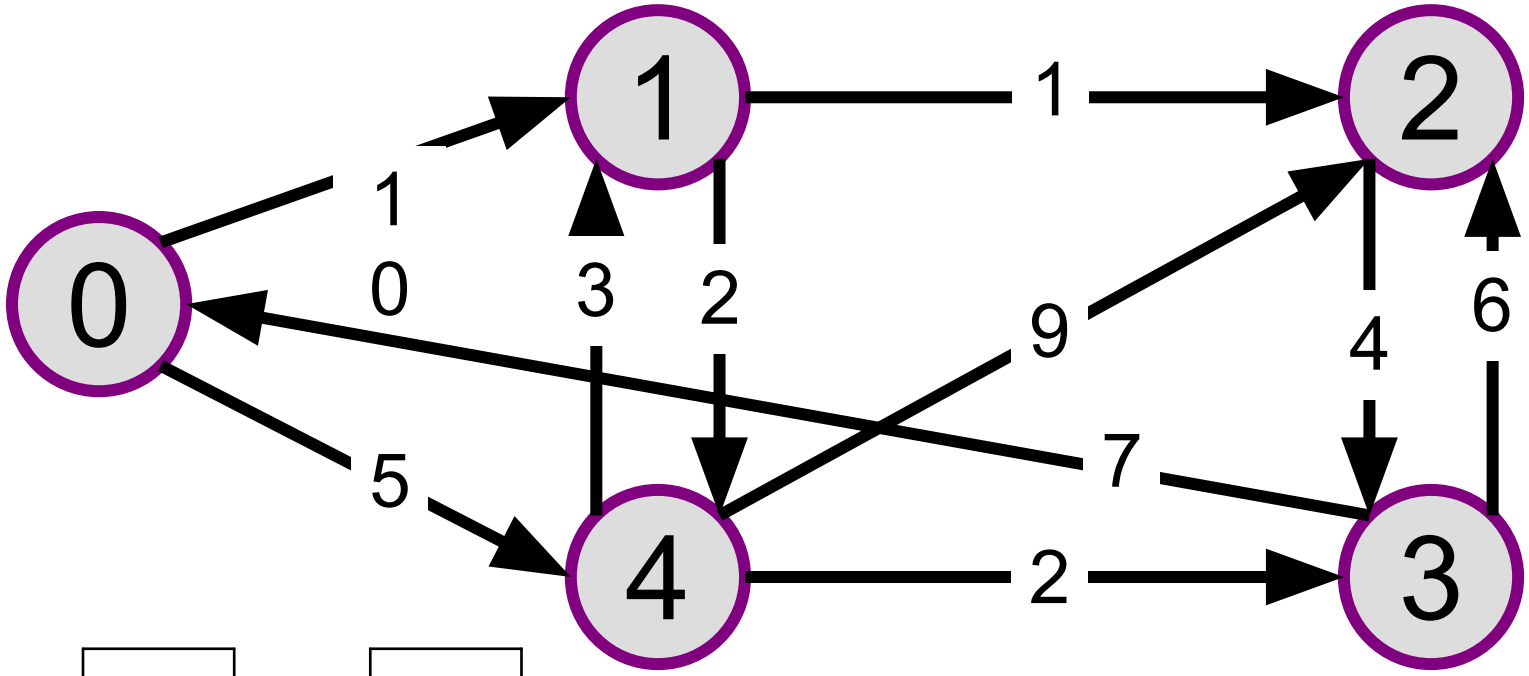


	D[v]
0	0
1	8
2	9
3	7
4	5

P[v]
nil
4
1
4
0

Q
2

S
0
4
3
1



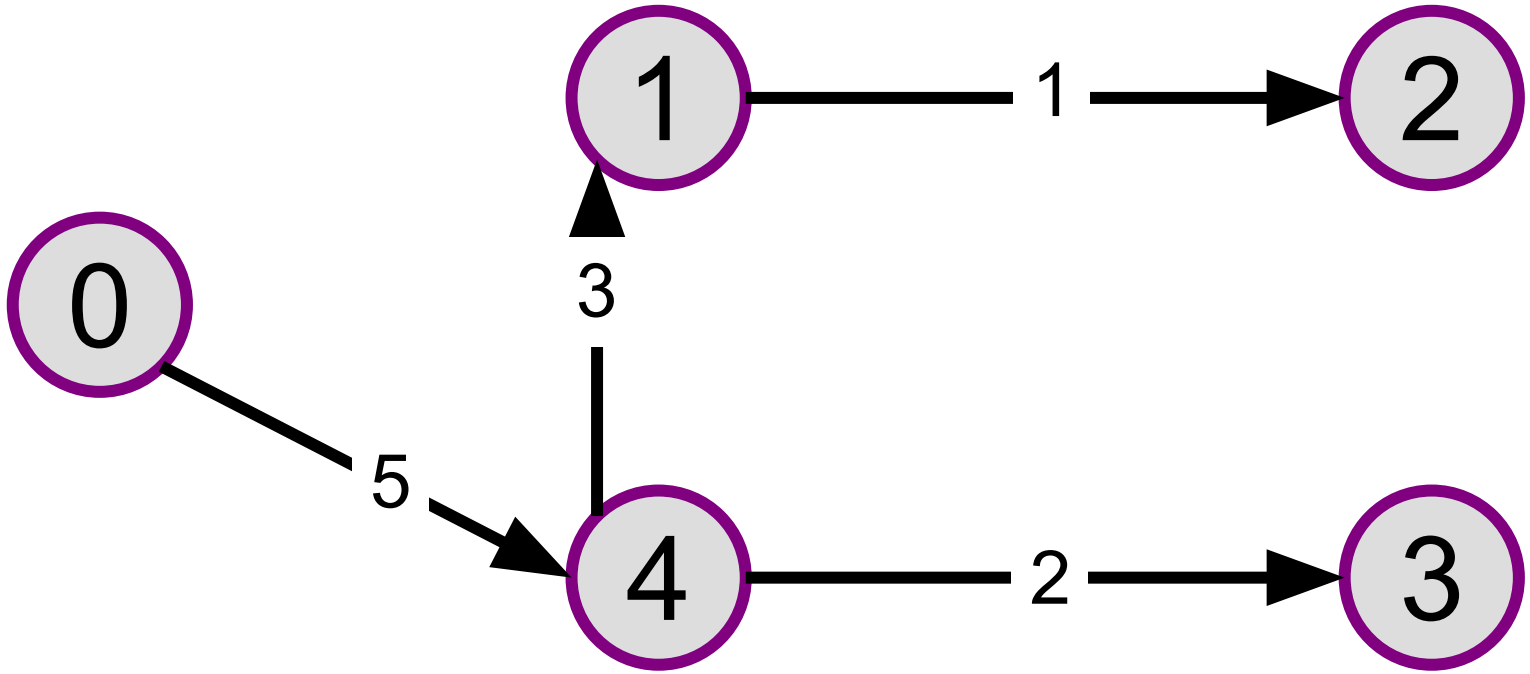
	D[v]
0	0
1	8
2	9
3	7
4	5

P[v]
nil
4
1
4
0

Q

S
0
4
3
1
2

P[v]
nil
4
1
4
0



При расчете временных характеристик сетевого графика, необходимо найти критический путь, определяющий минимальное время выполнения проекта. Отыскание критического, максимального, пути в графе сводится к поиску пути с самым большим весом, называемого максимальным путем в графе.

Максимальным путем $\bar{p}$ ***называется путь с максимальным весом*** $\bar{w} = \max_{k,i,j}(w(p_{ij}^{k}))$. В общем случае, в одном графе может быть несколько максимальных путей.

Рассматриваемый алгоритм основывается на рекуррентном выражении $d[v_j] = \max_{v_i \in V^-(v_j)} (d[v_i] + w(v_i, v_j))$, где $V^-(v_j)$ – множество начальных вершин дуг, входящих в вершину v_j. При этом предполагается, что для всех вершин $u \in V_0$ ранга 0 значение $d[u] = 0$.

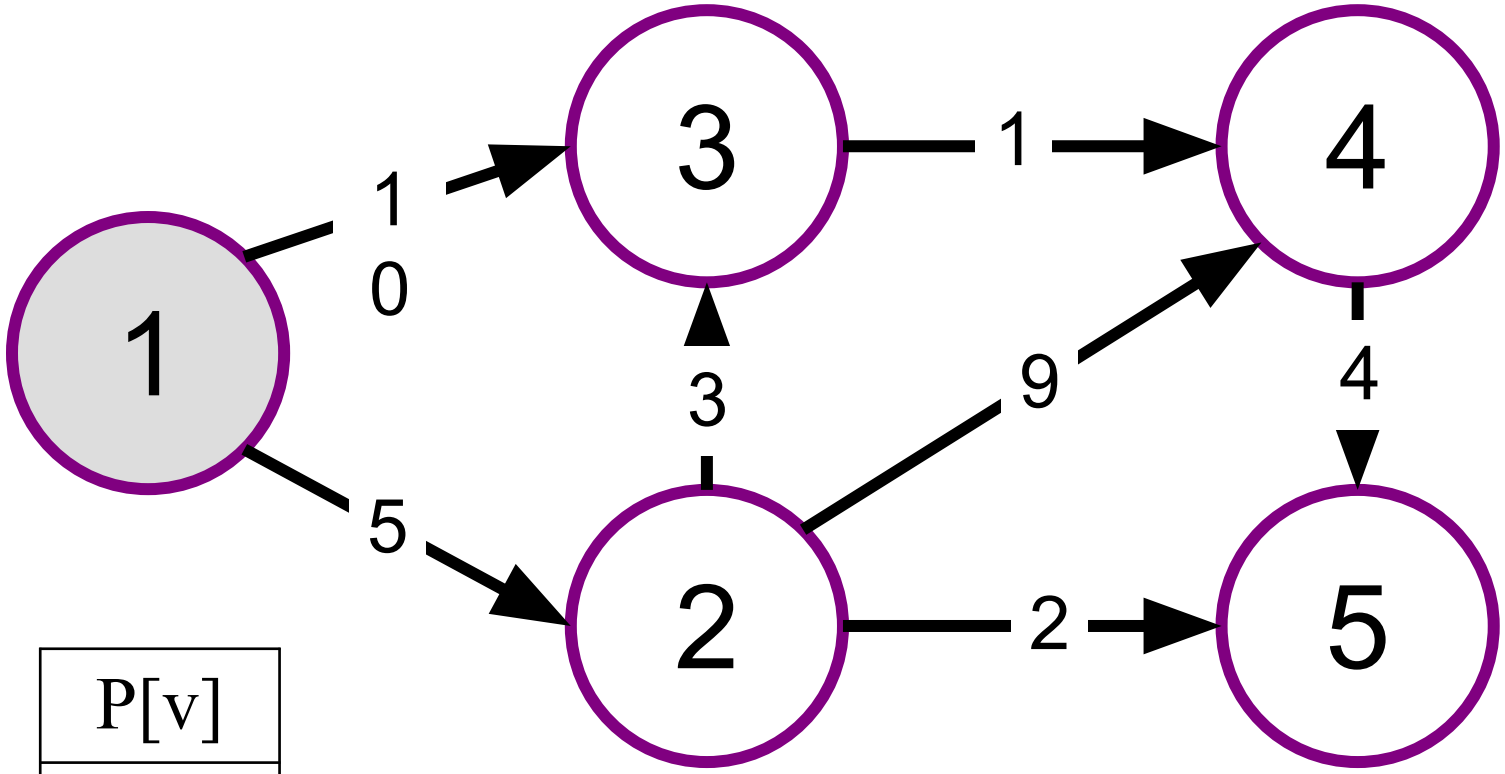
Вычисление значений $d[v]$ необходимо выполнить для каждой вершины графа в порядке возрастания номера. Каждое полученное значение $d[v]$ представляет собой максимальный вес пути в графе $G = (V, E)$ с конечной вершиной v.

Цикл вычисления $d[v]$ сопровождается построением массива элементов $p[v]$, которые формируются по тому же принципу, что и в алгоритме Дейкстры. Каждый элемент $p[v]$ соответствует вершине графа $v \in V$. Значение $p[v]$ – вершина (или ее номер), предшествующая вершине v в пути максимального веса с конечной вершиной v.

Представлен пример решения задачи поиска максимального пути в графе.

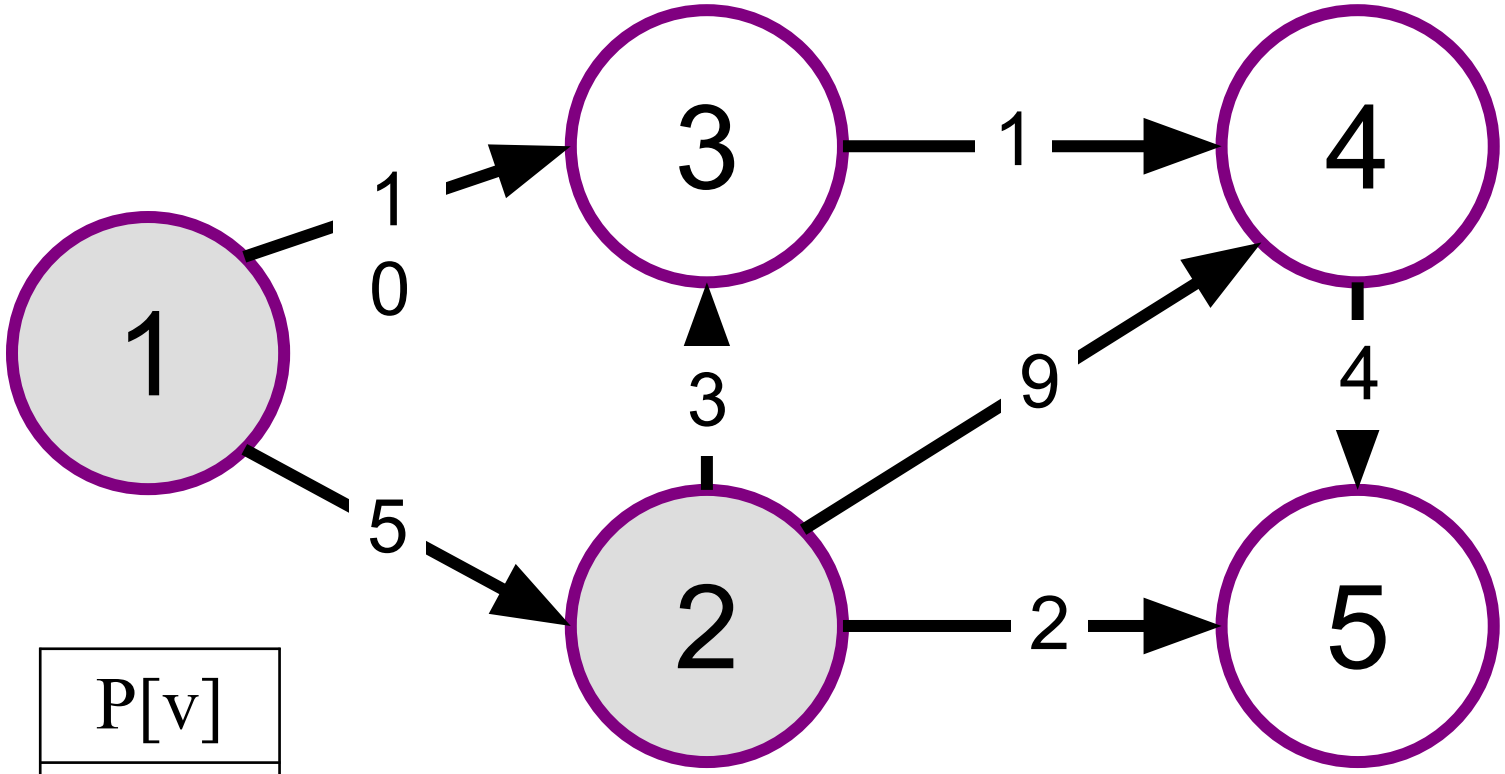
На рисунках изображен заданный граф и проинициализированные массивы $\boldsymbol{p}[\boldsymbol{v}]$ и $\boldsymbol{d}[\boldsymbol{v}]$.

Для построения максимального пути в графе необходимо найти максимальный элемент в $\boldsymbol{d}[\boldsymbol{v}]$ (в нашем случае – это 18) и обратным порядком построить все предшествующие $\boldsymbol{v}$ вершины по массиву $\boldsymbol{p}[\boldsymbol{v}]$ (в нашем случае – это вершины: 5, 4, 2, 1).



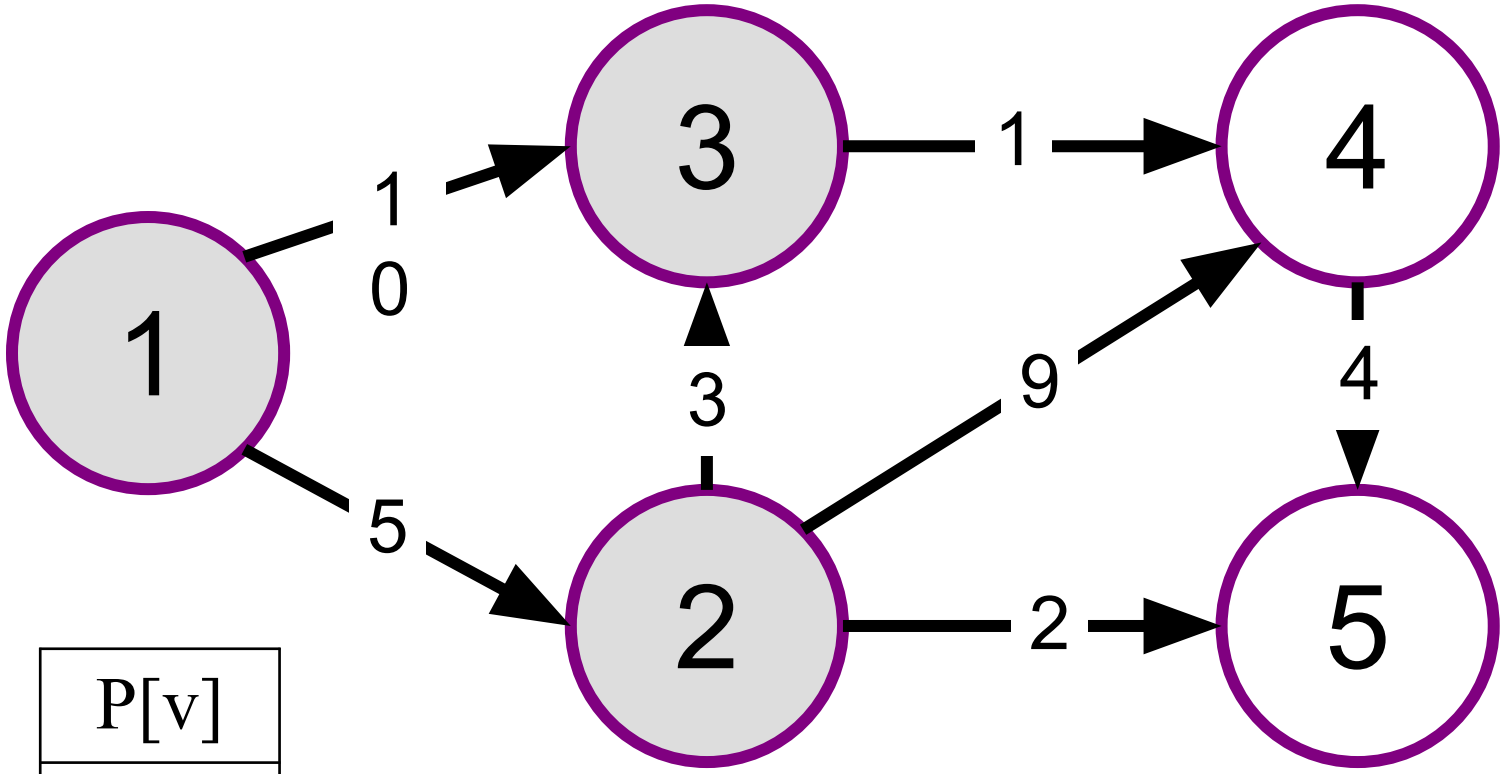
	D[v]
1	0
2	nil
3	nil
4	nil
5	nil

P[v]
nil
nil
nil
nil
nil



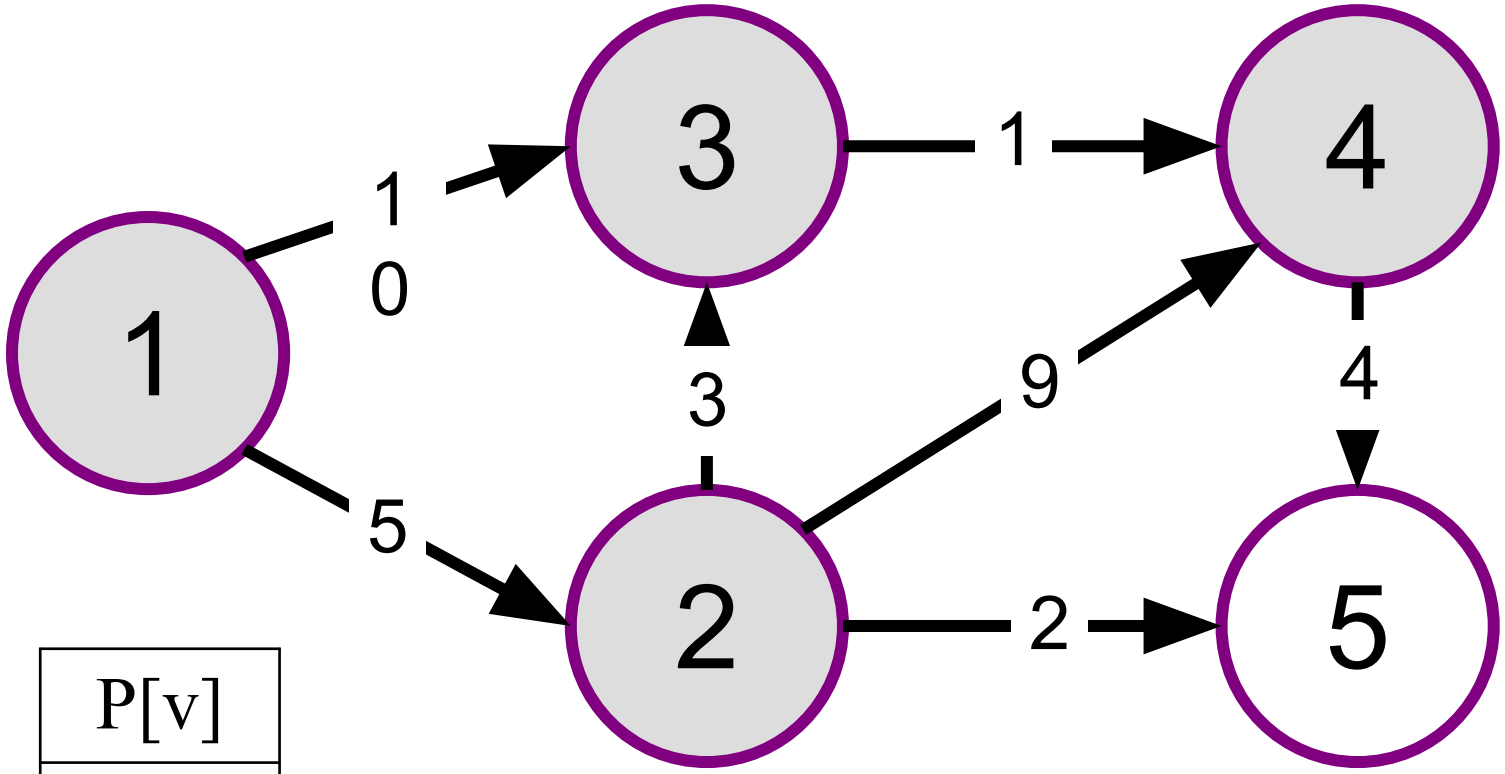
	D[v]
1	0
2	5
3	nil
4	nil
5	nil

P[v]
nil
1
nil
nil
nil



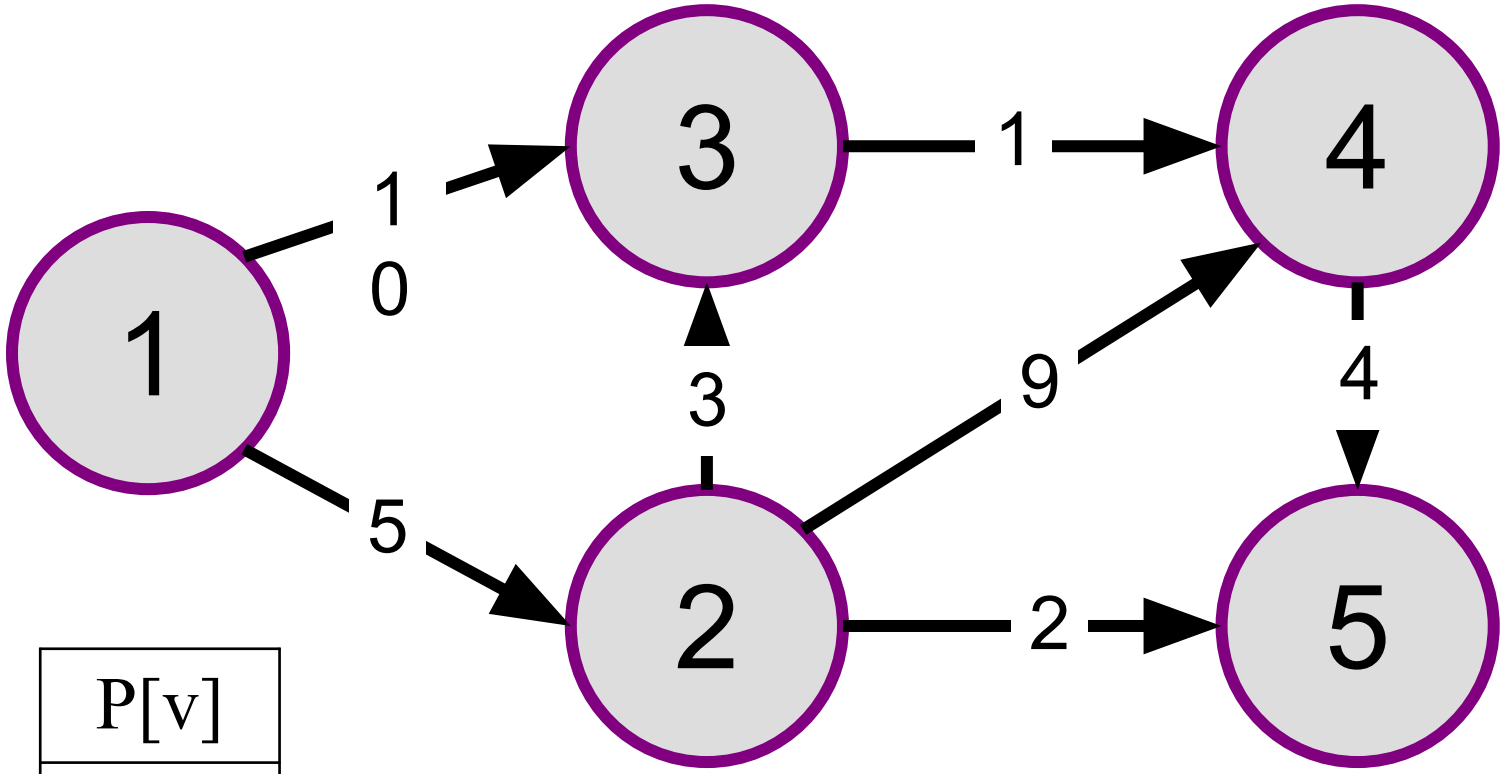
	D[v]
1	0
2	5
3	10
4	nil
5	nil

P[v]
nil
1
1
nil
nil



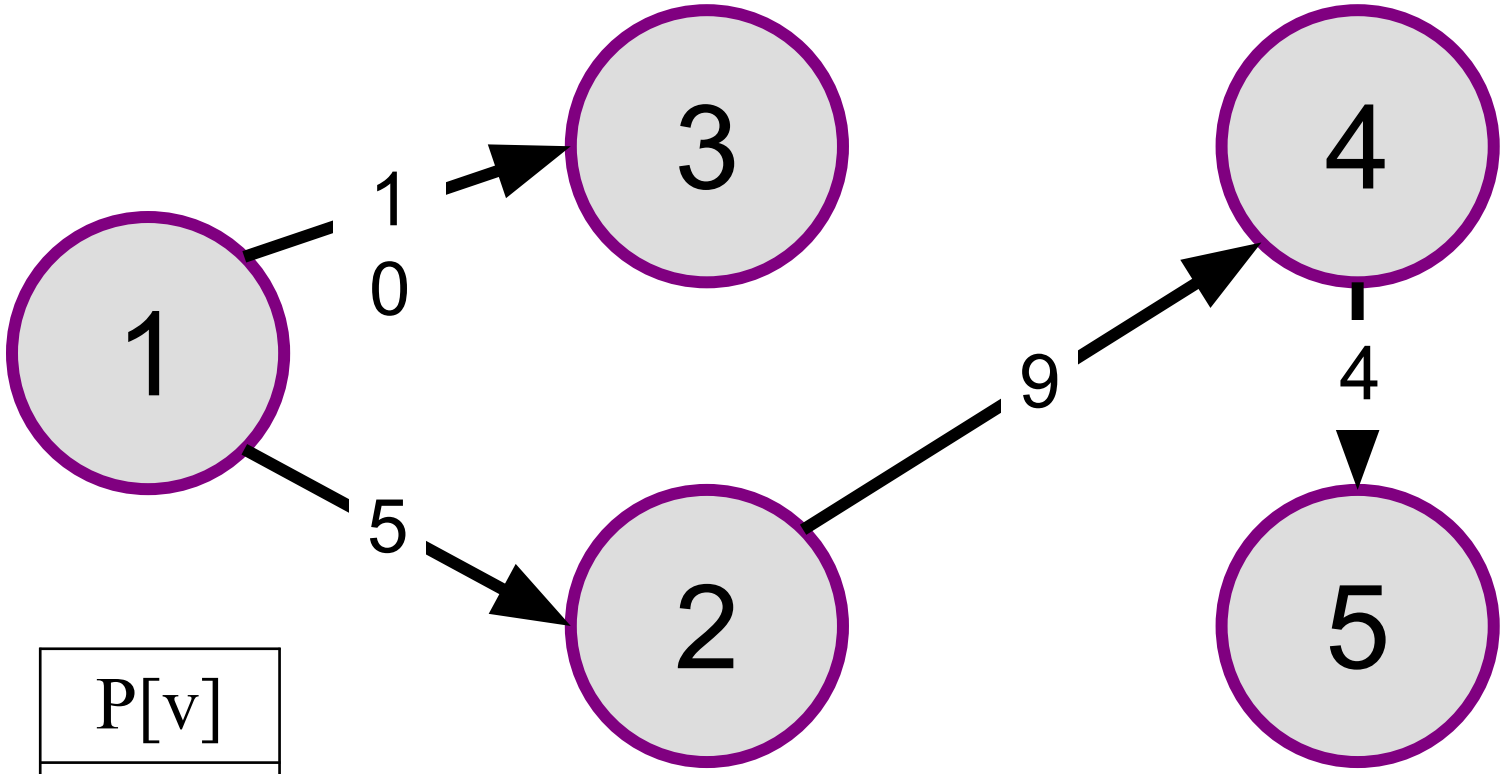
	D[v]
1	0
2	5
3	10
4	14
5	nil

P[v]
nil
1
1
2
nil



	D[v]
1	0
2	5
3	10
4	14
5	18

P[v]
nil
1
1
2
4



	D[v]
1	0
2	5
3	10
4	14
5	18

P[v]
nil
1
1
2
4