ДИСКРЕТНАЯ МАТЕМАТИКА

ФН, магистры - 2 семестр

Семинар 3. МЕТОДЫ СИСТЕМАТИЧЕСКОГО ОБХОДА ВЕРШИН ГРАФА. АЛГОРИТМЫ НА ГРАФАХ

Важными задачами теории графов являются **задачи глобального анализа** как *неориентированных*, так и *ориентированных графов*. К этим задачам относятся, например, задачи поиска *циклов* или *контуров*, вычисление *длин путей* между парами *вершин*, перечисление *путей* с теми или иными свойствами и т.п.

Необходимо уметь обходить все вершины графа таким образом, чтобы каждая вершина была отмечена ровно один раз. Обычно такое „путешествие“ по графу сопровождается нумерацией вершин графа в том порядке, в котором они отмечаются, а также определенной „маркировкой“ *ребер* (или *дуг*) графа. Существуют две основные стратегии таких обходов: **поиск в глубину** и **поиск в ширину**.

3.1. Алгоритм поиска в в глубину в неориентированном и в ориентированном графе

Поиск в глубину в неориентированном графе

Граф задан списками смежности, собранными в массив лидеров.

При поиске вершины графа нумеруются в порядке их посещения. Номер вершины v графа, присваиваемый ей при поиске в глубину, обозначим $D[v]$ и будем называть **D -номером**.

В процессе обхода будем находить **фундаментальные циклы** графа.

Пусть в неориентированном графе $G = (V, E)$ произвольно фиксирован **максимальный остовный лес**. Для связного графа это будет максимальное остовное дерево. Множество его ребер обозначим T. Все ребра из T назовем **древесными**, а ребра исходного графа G, не принадлежащие T, — **обратными**.

Любой цикл графа G, содержащий только одно обратное ребро, назовем фундаментальным.

Максимальный остовный лес, находимый с помощью алгоритма поиска в глубину, называют **остовным лесом поиска в глубину** или **глубинным остовным лесом**.

Классификация ребер зависит от хода работы алгоритма, который определяется стартовой вершиной и расположением вершин в списках смежности.

Для организации работы алгоритма поиска в глубину используется способ хранения данных, называемый **стеком**.

Элементы в стеке упорядочиваются в порядке поступления. В стек можно добавлять новые элементы и из него можно извлекать элементы. При этом доступен только последний добавленный элемент — **вершина стека**.

В алгоритме поиска в глубину используется стек вершин.

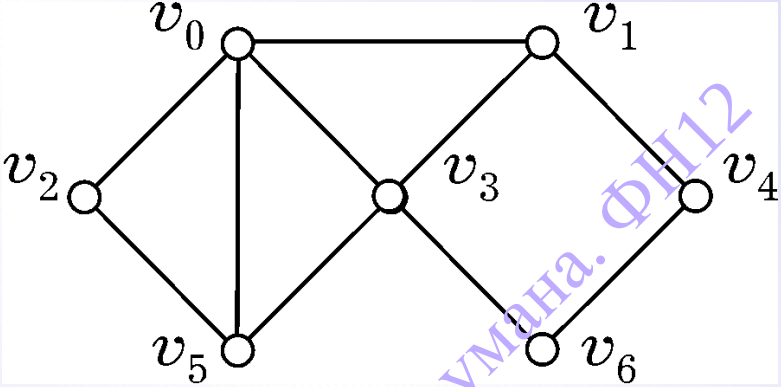


Рис. 1

Списки смежности

$V_0 \to (V_1, V_2, V_3, V_5)$
$V_1 \to (V_0, V_3, V_4)$
$V_2 \to (V_0, V_5)$
$V_3 \to (V_0, V_1, V_5, V_6)$
$V_4 \to (V_1, V_6)$
$V_5 \to (V_0, V_2, V_3)$
$V_6 \to (V_3, V_4)$

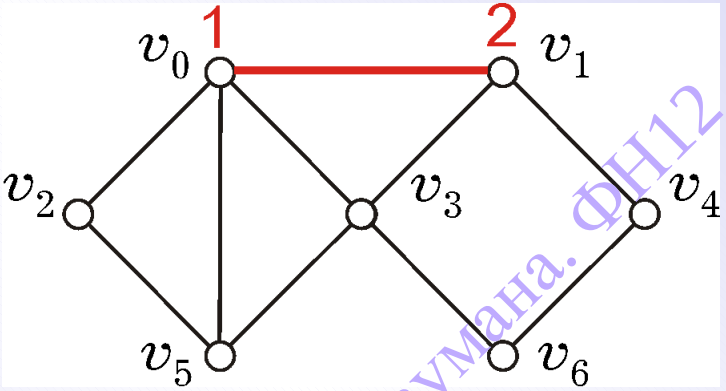


Рис. 2

Списки смежности

$V_0 \to (V_1, V_2, V_3, V_5)$
$V_1 \to (V_0, V_3, V_4)$
$V_2 \to (V_0, V_5)$
$V_3 \to (V_0, V_1, V_5, V_6)$
$V_4 \to (V_1, V_6)$
$V_5 \to (V_0, V_2, V_3)$
$V_6 \to (V_3, V_4)$

v_0

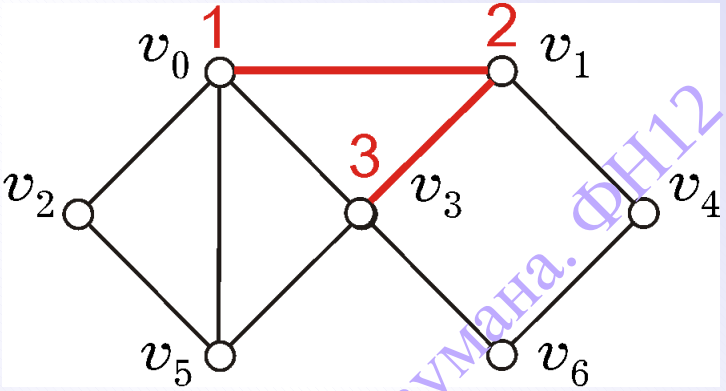


Рис. 3

Списки смежности

$V_0 \to (V_1, V_2, V_3, V_5)$
$V_1 \to (V_0, V_3, V_4)$
$V_2 \to (V_0, V_5)$
$V_3 \to (V_0, V_1, V_5, V_6)$
$V_4 \to (V_1, V_6)$
$V_5 \to (V_0, V_2, V_3)$
$V_6 \to (V_3, V_4)$

v_0
v_1

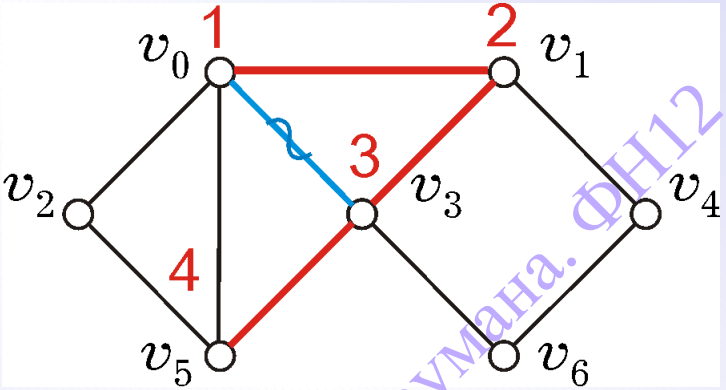


Рис. 4

Списки смежности

$V_0 \to (\not{V}_1, V_2, \not{V}_3, V_5)$
$V_1 \to (\not{V}_0, \not{V}_3, V_4)$
$V_2 \to (V_0, V_5)$
$V_3 \to (\not{V}_0, \not{V}_1, \not{V}_5, V_6)$
$V_4 \to (V_1, V_6)$
$V_5 \to (V_0, V_2, V_3)$
$V_6 \to (V_3, V_4)$

v_0
v_1
v_3

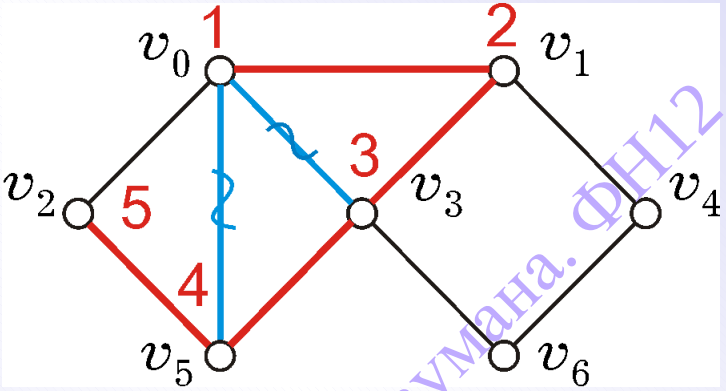
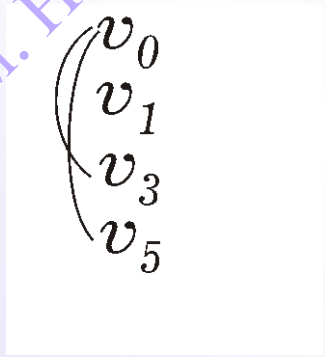


Рис. 5

Списки смежности

$V_0 \to (\not{N}_1, \not{N}_2, \not{N}_3, \not{N}_5)$
$V_1 \to (\not{N}_0, \not{N}_3, V_4)$
$V_2 \to (V_0, V_5)$
$V_3 \to (\not{N}_0, \not{N}_1, \not{N}_5, V_6)$
$V_4 \to (V_1, V_6)$
$V_5 \to (\not{N}_0, \not{N}_2, \not{N}_3)$
$V_6 \to (V_3, V_4)$



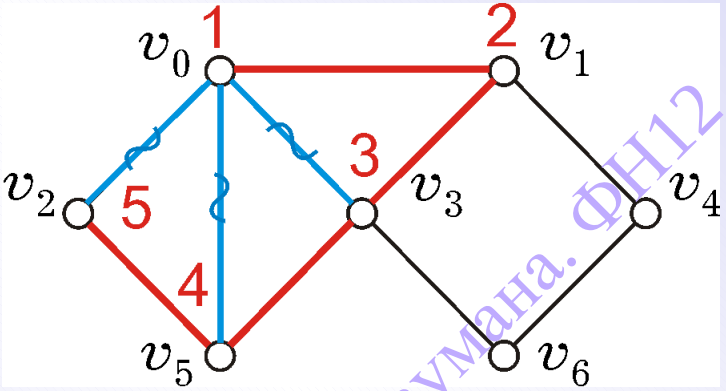
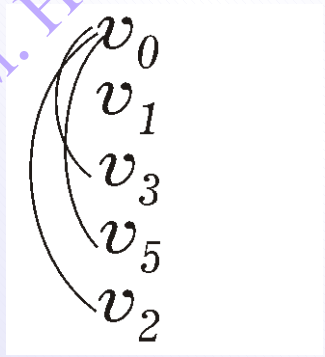


Рис. 6

Списки смежности

$V_0 \to (\not{V}_1, \not{V}_2, \not{V}_3, \not{V}_5)$
$V_1 \to (\not{V}_0, \not{V}_3, V_4)$
$V_2 \to (\not{V}_0, \not{V}_5)$
$V_3 \to (\not{V}_0, \not{V}_1, \not{V}_5, V_6)$
$V_4 \to (V_1, V_6)$
$V_5 \to (\not{V}_0, \not{V}_2, \not{V}_3)$
$V_6 \to (V_3, V_4)$



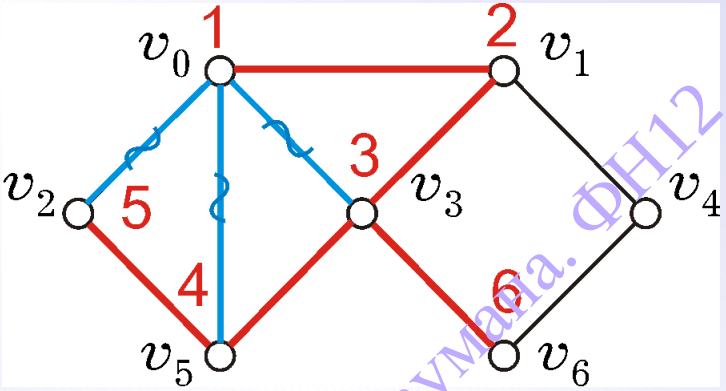
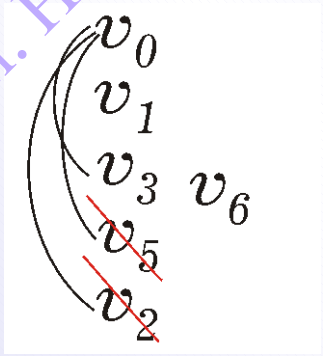


Рис. 7

Списки смежности

$V_0 \to (\not{V}_1, \not{V}_2, \not{V}_3, \not{V}_5)$
$V_1 \to (\not{V}_0, \not{V}_3, V_4)$
$V_2 \to (\not{V}_0, \not{V}_5)$
$V_3 \to (\not{V}_0, \not{V}_1, \not{V}_5, \not{V}_6)$
$V_4 \to (V_1, V_6)$
$V_5 \to (\not{V}_0, \not{V}_2, \not{V}_3)$
$V_6 \to (V_3, V_4)$



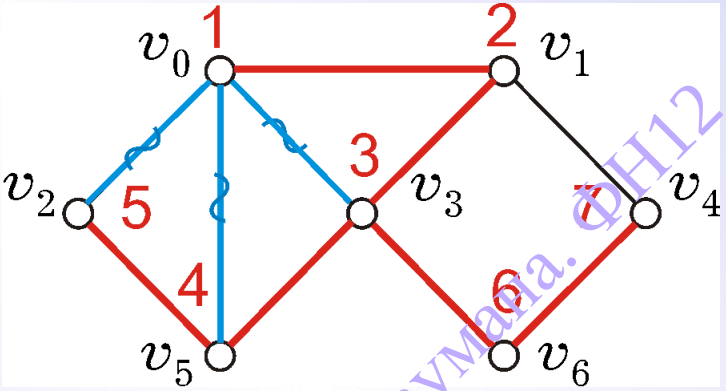
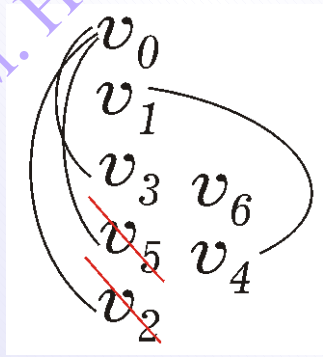


Рис. 8

Списки смежности

$V_0 \to (\not{V}_1, \not{V}_2, \not{V}_3, \not{V}_5)$
$V_1 \to (\not{V}_0, \not{V}_3, V_4)$
$V_2 \to (\not{V}_0, \not{V}_5)$
$V_3 \to (\not{V}_0, \not{V}_1, \not{V}_5, \not{V}_6)$
$V_4 \to (V_1, V_6)$
$V_5 \to (\not{V}_0, \not{V}_2, \not{V}_3)$
$V_6 \to (\not{V}_3, \not{V}_4)$



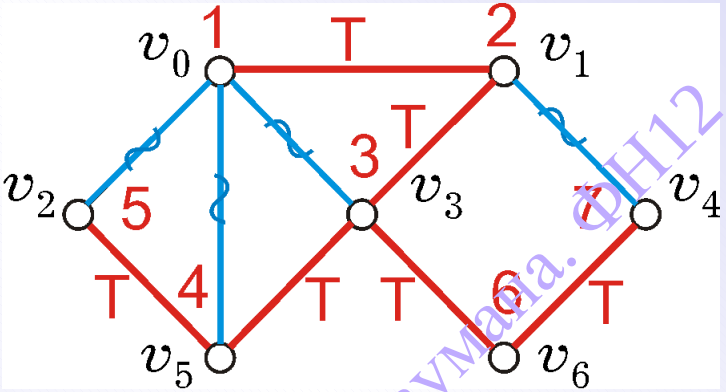


Рис. 2

Списки смежности

$V_0 \to (\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_5)$
$V_1 \to (\mathcal{N}_0, \mathcal{N}_3, \mathcal{N}_4)$
$V_2 \to (\mathcal{N}_0, \mathcal{N}_5)$
$V_3 \to (\mathcal{N}_0, \mathcal{N}_1, \mathcal{N}_5, \mathcal{N}_6)$
$V_4 \to (\mathcal{N}_1, \mathcal{N}_6)$
$V_5 \to (\mathcal{N}_0, \mathcal{N}_2, \mathcal{N}_3)$
$V_6 \to (\mathcal{N}_3, \mathcal{N}_4)$

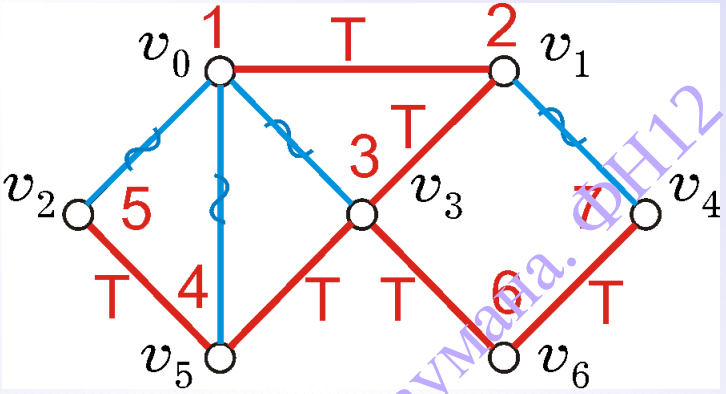


Рис. 13

Списки смежности

$V_0 \to (\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_5)$
$V_1 \to (\mathcal{N}_0, \mathcal{N}_3, \mathcal{N}_4)$
$V_2 \to (\mathcal{N}_0, \mathcal{N}_5)$
$V_3 \to (\mathcal{N}_0, \mathcal{N}_1, \mathcal{N}_5, \mathcal{N}_6)$
$V_4 \to (\mathcal{N}_1, \mathcal{N}_6)$
$V_5 \to (\mathcal{N}_0, \mathcal{N}_2, \mathcal{N}_3)$
$V_6 \to (\mathcal{N}_3, \mathcal{N}_4)$

В ориентированном графе вершинам также присваиваются D-номера. Но классификация дуг при поиске в глубину в ориентированном графе сложнее по сравнению с аналогичной классификацией ребер при поиске в глубину в неориентированном графе. Различают четыре класса дуг:

1) **древесные дуги** — каждая такая *дуга ведет* от *отца* к *сыну* в глубинном остовном лесу;

2) **прямые дуги** — каждая такая дуга ведет от *подлинного предка* к *подлинному потомку* (но не от отца к сыну) в глубинном остовном лесу;

3) **обратные дуги** — от *потомков к предкам* (включая все петли);

4) **поперечные дуги** — все дуги, не являющиеся ни древесными, ни прямыми, ни обратными.

В результате работы алгоритма будут получены множества $Tree$ — древесных дуг, $Back$ — обратных дуг, $Forward$ — прямых дуг, C — поперечных дуг и массив D, содержащий D-номера вершин.

В процессе работы алгоритма по сравнению с алгоритмом поиска в глубину в неориентированном графе имеется ряд особенностей. Так, если очередная вершина w, извлеченная из списка смежности текущей вершины v, новая, то дуга (v, w) является древесной.

Если вершина w не новая $(w \notin V_0)$, то дуга (v, w) будет либо прямой, либо обратной, либо поперечной.

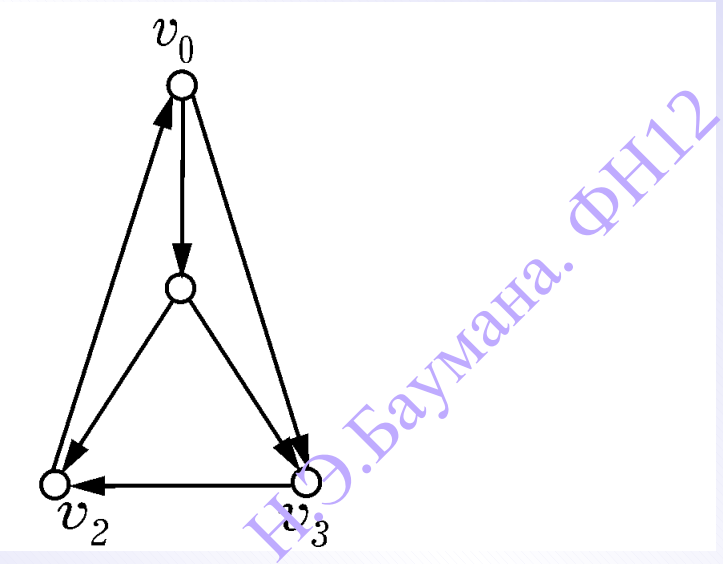
Если D-номер вершины v строго меньше D-номера вершины w ($D[v] < D[w]$), то дуга (v, w) является прямой.

Если D-номер вершины v не меньше D-номера вершины w ($D[v] \geq D[w]$), необходимо проверить, есть ли в стеке $STACK$ вершина w. Если вершина w находится в стеке, то дуга (v, w) является обратной. Если вершины w в стеке нет, то дуга является поперечной.

Если стек пуст, но не все вершины ориентированного графа обработаны, поиск продолжают из любой необработанной вершины.

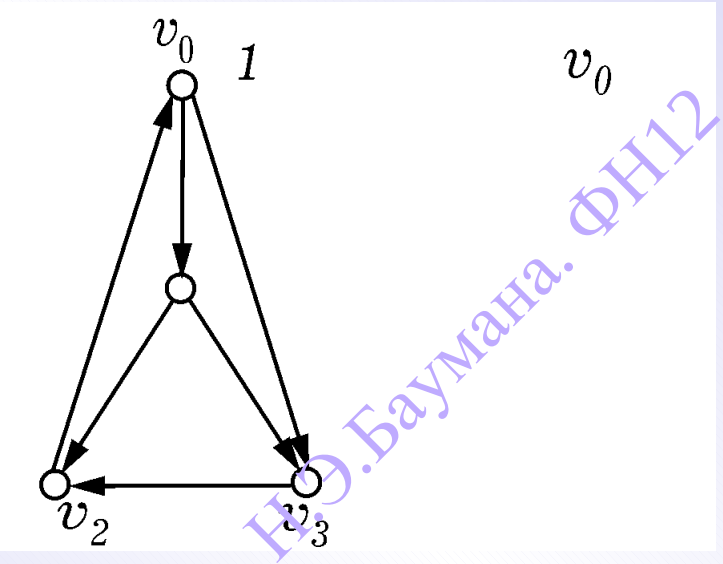
В случае ориентированного графа поиск контуров на базе поиска в глубину существенно сложнее.

Ориентированный граф является бесконтурным тогда и только тогда, когда при поиске в глубину от некоторой начальной вершины множество обратных дуг оказывается пустым.



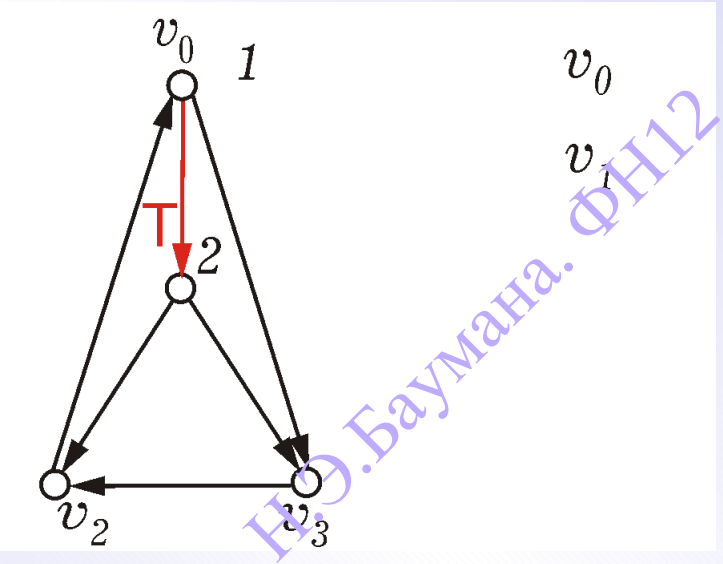
Списки смежности

$V_0 \to (V_1, V_3)$
$V_1 \to (V_2, V_3)$
$V_2 \to (V_0)$
$V_3 \to (V_2)$



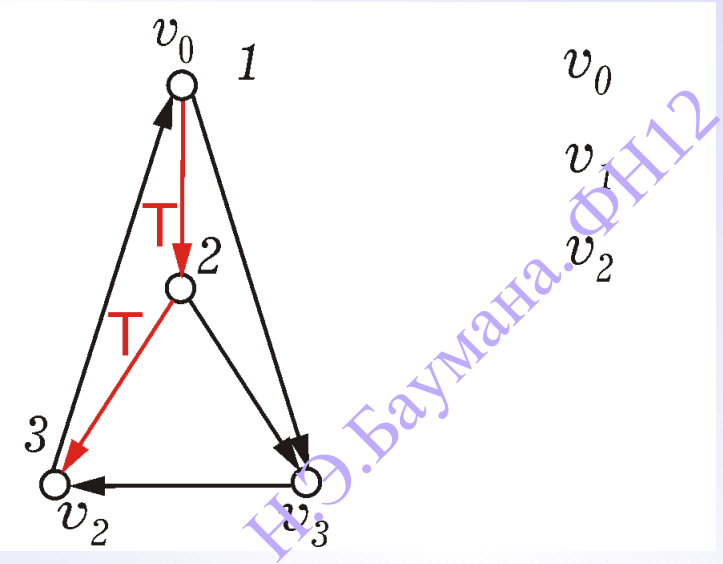
Списки смежности

$V_0 \to (V_1, V_3)$
$V_1 \to (V_2, V_3)$
$V_2 \to (V_0)$
$V_3 \to (V_2)$



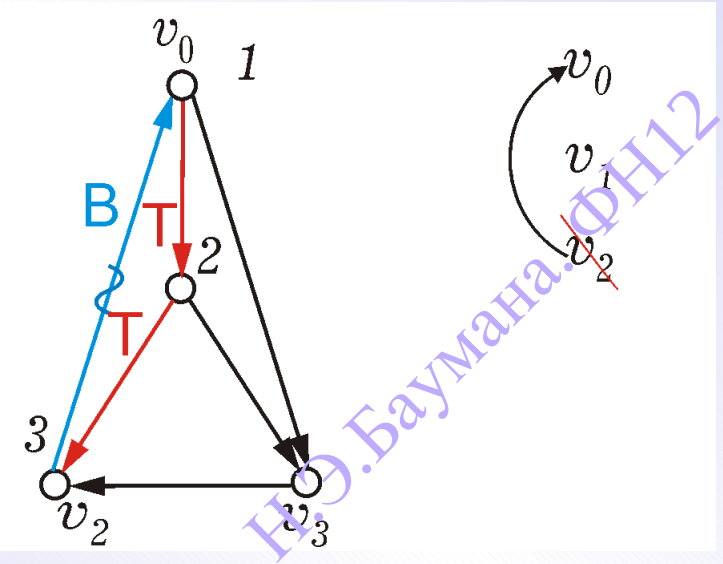
Списки смежности

$V_0 \to (V_1, V_3)$
$V_1 \to (V_2, V_3)$
$V_2 \to (V_0)$
$V_3 \to (V_2)$



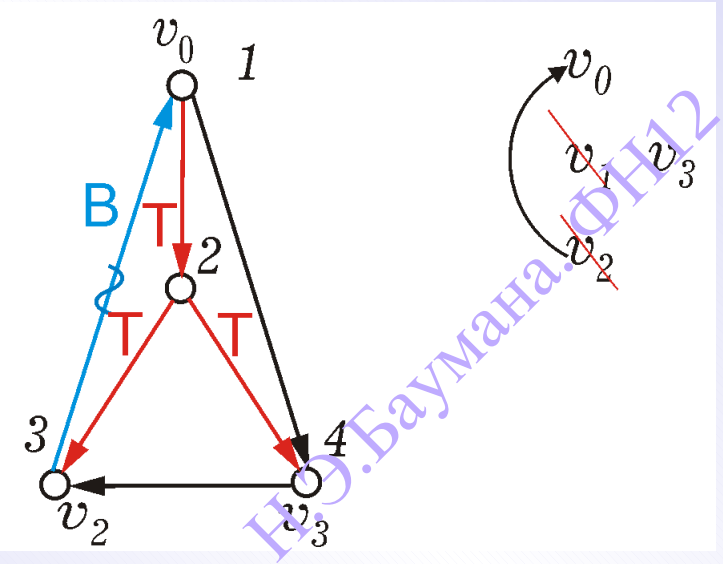
Списки смежности

$V_0 \to (\not{V}_1, V_3)$
$V_1 \to (\not{V}_2, V_3)$
$V_2 \to (V_0)$
$V_3 \to (V_2)$



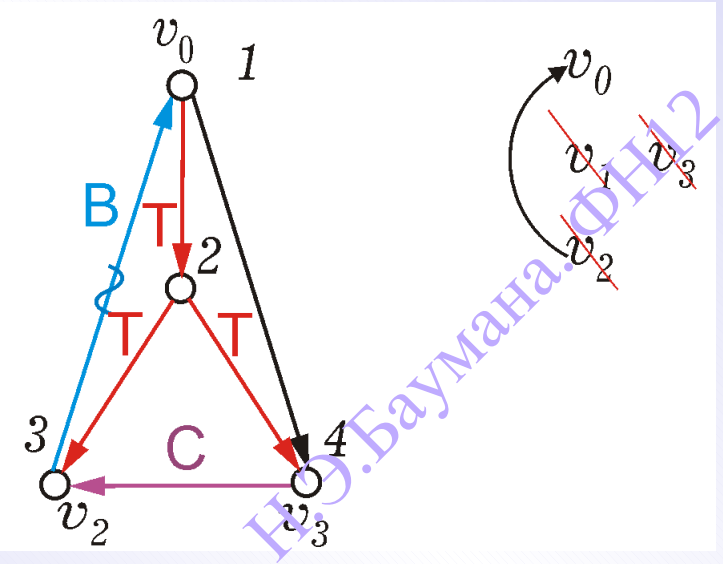
Списки смежности

$V_0 \to (\not{V}_1, V_3)$
$V_1 \to (\not{V}_2, V_3)$
$V_2 \to (\not{V}_0)$
$V_3 \to (V_2)$



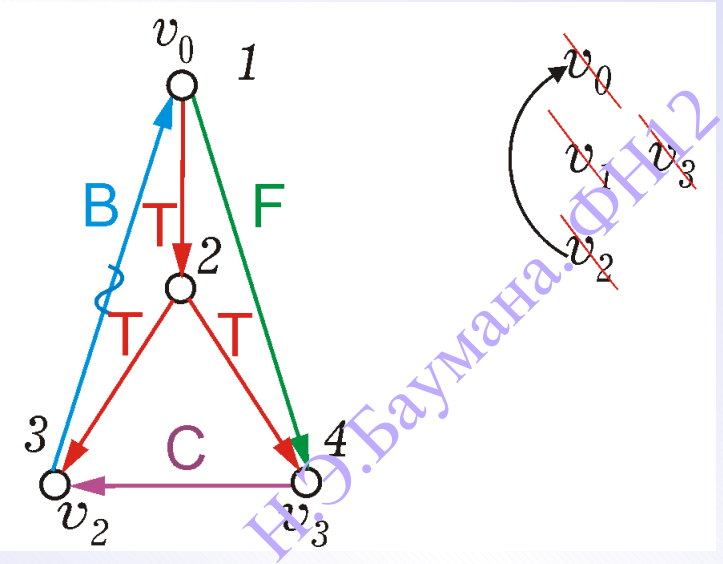
Списки смежности

$V_0 \to (\not{V}_1, V_3)$
$V_1 \to (\not{V}_2, \not{V}_3)$
$V_2 \to (\not{V}_0)$
$V_3 \to (V_2)$



Списки смежности

$V_0 \to (\not{V}_1, V_3)$
$V_1 \to (\not{V}_2, \not{V}_3)$
$V_2 \to (\not{V}_0)$
$V_3 \to (\not{V}_2)$



Списки смежности

$V_0 \to (\not{N}_1, \not{N}_3)$
$V_1 \to (\not{N}_2, \not{N}_3)$
$V_2 \to (\not{N}_0)$
$V_3 \to (\not{N}_2)$

3.2. Алгоритм поиска в ширину в ориентированном графе

Вход. Граф $G = (V,\ E)$, заданный списками смежности; v_0 — начальная вершина (не обязательно первый элемент массива лидеров).

Выход. Массив M меток вершин, где каждая метка равна длине пути от v_0 до v.

0. Очередь Q положить пустой $(Q := \varnothing)$. Все вершины пометить как недостижимые из вершины v_0, присваивая элементам массива M значение $+\infty$ $(M[v_i] := +\infty,\ i = \overline{1,\ N})$.
Стартовую вершину v_0 пометить номером 0, т.е. длину пути от стартовой вершины v_0 до самой себя положить равной 0 $(M[v_0] := 0)$. Поместить вершину v_0 в очередь Q. Перейти на шаг **1**.
1. Если очередь Q не пуста $(Q \neq \varnothing)$, то из „головы“ очереди извлечь (с удалением из очереди) вершину u и перейти на шаг **2**. Если очередь пуста, перейти на шаг **3**.
2. Если список смежности $L(u)$ вершины u пуст, вернуться на шаг **1**.
Если список смежности $L(u)$ вершины u не пуст, для каждой вершины w из списка смежности, где $M[w] = +\infty$, т.е. вершины, которую еще не посещали, положить длину пути из стартовой вершины v_0 до вершины w равной длине пути от v_0 до вершины u плюс одна дуга $(M[w] := M[u] + 1)$, т.е. отметить вершину w и поместить ее в очередь Q. После просмотра всех вершин списка смежности $L(u)$ вернуться на шаг **1**.
3. Распечатать массив M. Закончить работу.

Алгоритм поиска в ширину может быть дополнен процедурой „обратного хода“, определяющей номера вершин, лежащих на кратчайшем пути из вершины v_0 в данную вершину u.

Для этого необходимо завести массив PR размера $|V|$, каждый элемент $PR[w]$ которого содержит номер той вершины, из которой был осуществлен переход в вершину w при ее пометке.

Если вершина w находится в списке смежности $L(u)$ вершины u, заполнение элемента массива $PR[w]$ происходит при изменении метки вершины w $M[w]$ с $+\infty$ на единицу. При этом в элементе $PR[w]$ сохраняется номер вершины u $(PR[w] := u)$. Для начальной вершины $PR[v_0]$ можно положить равным 0, в предположении, что начальная вершина v_0 имеет номер 0 и остальные вершины пронумерованы от 1 до N.

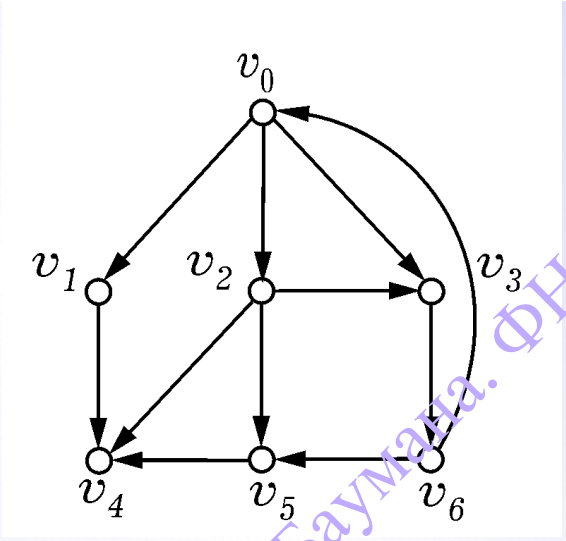


Рис. 11

Списки смежности

$v_0 \longrightarrow (v_1, v_2, v_3)$
$v_1 \longrightarrow (v_4)$
$v_2 \longrightarrow (v_4, v_5, v_3)$
$v_3 \longrightarrow (v_6)$
$v_4 \longrightarrow ()$
$v_5 \longrightarrow (v_4)$
$v_6 \longrightarrow (v_5, v_0)$

	v_0	v_1	v_2	v_3	v_4	v_5	v_6
1.	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

$Q = \{v_0\}$
$PR(v_0) = \varnothing$

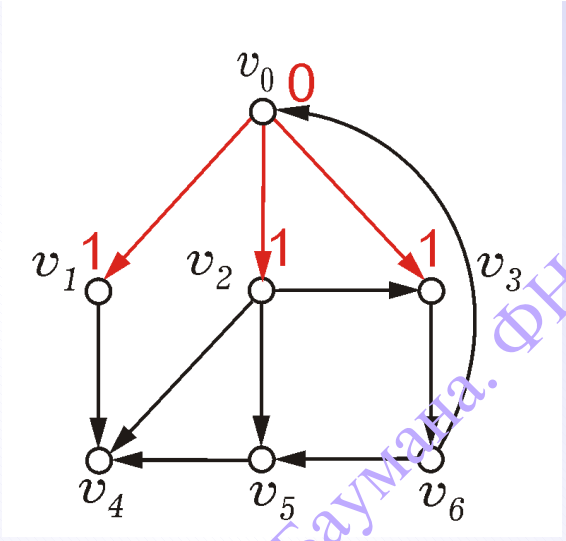


Рис. 12

	v_0	v_1	v_2	v_3	v_4	v_5	v_6
1.	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
2.	0	1	1	1	$+\infty$	$+\infty$	$+\infty$

Списки смежности

$v_0 \longrightarrow (v_1, v_2, v_3)$
$v_1 \longrightarrow (v_4)$
$v_2 \longrightarrow (v_4, v_5, v_3)$
$v_3 \longrightarrow (v_6)$
$v_4 \longrightarrow ()$
$v_5 \longrightarrow (v_4)$
$v_6 \longrightarrow (v_5, v_0)$

$Q = \{\not{v}_0, v_1, v_2, v_3\}$
$PR(v_0) = \varnothing, PR(v_1) = v_0, PR(v_2) = v_0,$
$PR(v_3) = v_0$

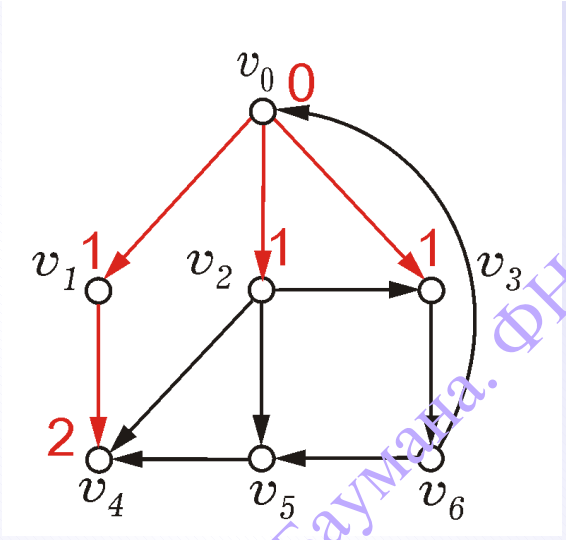


Рис. 13

	v_0	v_1	v_2	v_3	v_4	v_5	v_6
1.	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
2.	0	1	1	1	$+\infty$	$+\infty$	$+\infty$
3.	0	1	1	1	2	$+\infty$	$+\infty$

Списки смежности

$v_0 \to (v_1, v_2, v_3)$
$v_1 \to (v_4)$
$v_2 \to (v_4, v_5, v_3)$
$v_3 \to (v_6)$
$v_4 \to ()$
$v_5 \to (v_4)$
$v_6 \to (v_5, v_0)$

$Q = \{ \not{v}_0 \ \not{v}_1, v_2, v_3, v_4\}$
$PR(v_0) = \varnothing, PR(v_1) = v_0, PR(v_2) = v_0,$
$PR(v_3) = v_0, PR(v_4) = v_1$

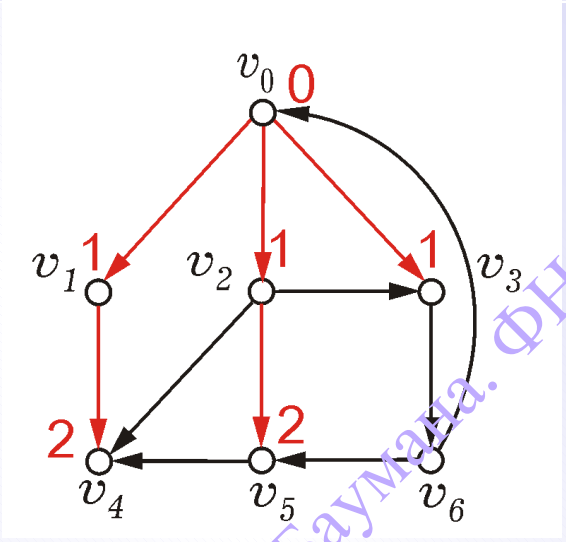


Рис. 14

Списки смежности

$v_0 \to (v_1, v_2, v_3)$
$v_1 \to (v_4)$
$v_2 \to (v_4, v_5, v_3)$
$v_3 \to (v_6)$
$v_4 \to ()$
$v_5 \to (v_4)$
$v_6 \to (v_5, v_0)$

	v_0	v_1	v_2	v_3	v_4	v_5	v_6
1.	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
2.	0	1	1	1	$+\infty$	$+\infty$	$+\infty$
3.	0	1	1	1	2	$+\infty$	$+\infty$
4.	0	1	1	1	2	2	$+\infty$

$Q = \{ \not{v}_0 \ \not{v}_1 \ \not{v}_2, v_3, v_4, v_5\}$
$PR(v_0) = \varnothing, PR(v_1) = v_0, PR(v_2) = v_0,$
$PR(v_3) = v_0, PR(v_4) = v_1, PR(v_5) = v_2$

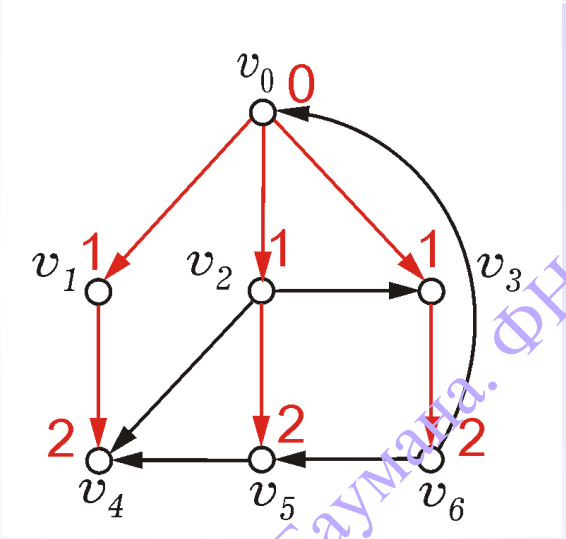


Рис. 13

Списки смежности

$v_0 \to (v_1, v_2, v_3)$
$v_1 \to (v_4)$
$v_2 \to (v_4, v_5, v_3)$
$v_3 \to (v_6)$
$v_4 \to ()$
$v_5 \to (v_4)$
$v_6 \to (v_5, v_0)$

	v_0	v_1	v_2	v_3	v_4	v_5	v_6
1.	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
2.	0	1	1	1	$+\infty$	$+\infty$	$+\infty$
3.	0	1	1	1	2	$+\infty$	$+\infty$
4.	0	1	1	1	2	2	$+\infty$
5.	0	1	1	1	2	2	2

$Q = \{\not{v_0}\ \not{v_1}\ \not{v_2}\ \not{v_3}, v_4, v_5, v_6\}$
$PR(v_0) = \varnothing, PR(v_1) = v_0, PR(v_2) = v_0,$
$PR(v_3) = v_0, PR(v_4) = v_1, PR(v_5) = v_2,$
$PR(v_6) = v_3$

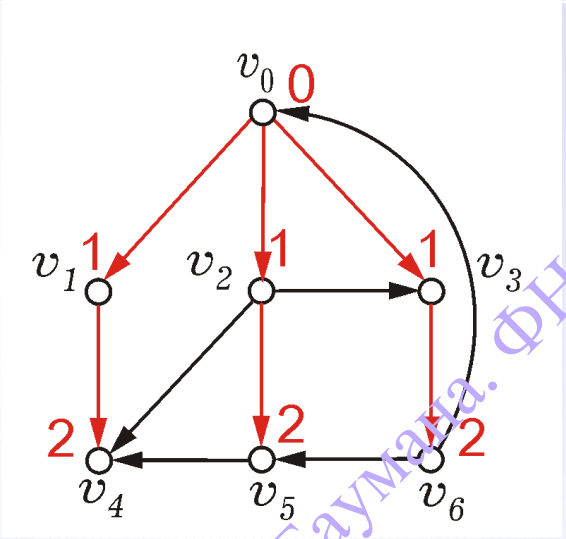


Рис. 16

Списки смежности

$v_0 \to (v_1, v_2, v_3)$
$v_1 \to (v_4)$
$v_2 \to (v_4, v_5, v_3)$
$v_3 \to (v_6)$
$v_4 \to ()$
$v_5 \to (v_4)$
$v_6 \to (v_5, v_0)$

	v_0	v_1	v_2	v_3	v_4	v_5	v_6
1.	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
2.	0	1	1	1	$+\infty$	$+\infty$	$+\infty$
3.	0	1	1	1	2	$+\infty$	$+\infty$
4.	0	1	1	1	2	2	$+\infty$
5.	0	1	1	1	2	2	2

$Q = \{ \not{v_0}\ \not{v_1}\ \not{v_2}\ \not{v_3}\ \not{v_4}\ \not{v_5}\ \not{v_6} \}$

$PR(v_0) = \varnothing, PR(v_1) = v_0, PR(v_2) = v_0,$
$PR(v_3) = v_0, PR(v_4) = v_1, PR(v_5) = v_2,$
$PR(v_6) = v_3$

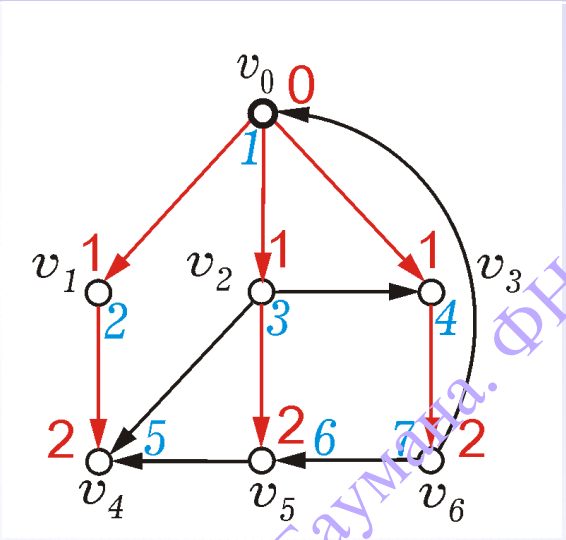


Рис. 17

Списки смежности

$v_0 \to (v_1, v_2, v_3)$
$v_1 \to (v_4)$
$v_2 \to (v_4, v_5, v_3)$
$v_3 \to (v_6)$
$v_4 \to ()$
$v_5 \to (v_4)$
$v_6 \to (v_5, v_0)$

	v_0	v_1	v_2	v_3	v_4	v_5	v_6
1.	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
2.	0	1	1	1	$+\infty$	$+\infty$	$+\infty$
3.	0	1	1	1	2	$+\infty$	$+\infty$
4.	0	1	1	1	2	2	$+\infty$
5.	0	1	1	1	2	2	2

$Q = \varnothing$

$PR(v_0) = \varnothing, PR(v_1) = v_0, PR(v_2) = v_0,$
$PR(v_3) = v_0, PR(v_4) = v_1, PR(v_5) = v_2,$
$PR(v_6) = v_3$

3.3. Остовное дерево наименьшего веса

Неориентированный (ориентированный) граф, у которого каждому ребру (дуге) сопоставлено некоторое действительное число, называют **взвешенным** или **размеченным** графом.

Это число называют **весом** или **меткой** ребра (дуги).

Алгоритм Краскала вычисляет для заданного взвешенного неориентированного графа G *остовное дерево* с наименьшей суммой весов ребер — **остовное дерево наименьшего веса**.

Алгоритм Краскала

Пусть дан связный неориентированный граф $G = (V, E)$ с числовыми неотрицательными весами ребер. Вес ребра e обозначим $\varphi(e)$.

В результате работы алгоритма получим остовное дерево $T = (V, H)$ графа G, такое, что сумма $\sum\limits_{e \in H} \varphi(e)$ является наименьшей.

Отсортируем все ребра исходного графа по возрастанию весов и сформируем из них **очередь**, в „голове“ очереди — ребро с наименьшим весом, в „хвосте“ — с наибольшим.

Метод состоит в „сшивании“ искомого дерева из *компонент* остовного леса. Первоначально *остовный лес* представляет собой множество изолированных вершин исходного графа, т.е. его множество ребер пусто. На первом шаге из очереди извлекается ребро наименьшего веса и добавляется к множеству ребер исходного дерева.

На последующих шагах алгоритма из очереди извлекается по одному ребру. Если это ребро соединяет вершины, принадлежащие разным компонентам текущего остовного леса, то оно добавляется к текущему множеству ребер искомого дерева, указанные компоненты сливаются в одну. Иначе ребро отбрасывается.

Процесс повторяется до тех пор, пока число компонент остовного леса не окажется равным 1. Можно показать, что эта компонента и будет искомым остовным деревом наименьшего веса.

1. Множество ребер H искомого остовного дерева полагаем пустым ($H = \varnothing$).

2. Формируем множество $V_S = \{\{v_1\}, \ldots, \{v_n\}\}$, элементами которого являются множества вершин, соответствующих компонентам исходного остовного леса. Каждая такая компонента состоит из единственной вершины.

3. Сортируем множество ребер E исходного графа по возрастанию весов и формируем очередь Q , элементами которой являются ребра графа G .

4. Если множество V_S содержит более одного элемента (т.е. остовный лес состоит из нескольких компонент) и очередь Q не пуста, переходим на шаг **5**, если иначе — на шаг **7**.

5. Извлекаем из очереди Q ребро e . Если *концы* ребра e принадлежат различным множествам вершин V_i и V_j из V_S , то переходим на шаг **6**, если иначе, то отбрасываем извлеченное ребро и возвращаемся на шаг **4**.

6. Объединяем множества вершин V_i и V_j (полагая $W = V_i \cup V_j$), удаляем множества V_i и V_j из множества V_S и добавляем в V_S множество W . Добавляем ребро e в множество H . Возвращаемся на шаг **4**.

7. Прекращаем работу. Множество H есть множество ребер полученного остовного дерева.

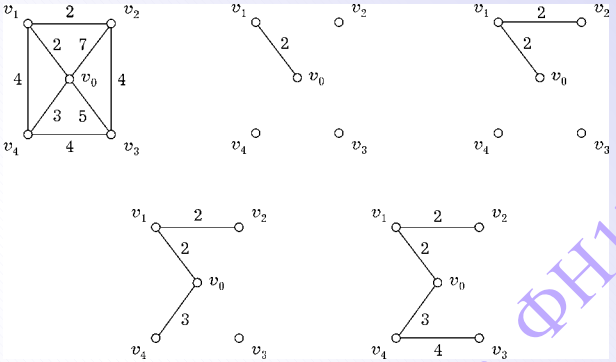


Рис. 18

Исходный граф изображен на рисунке *а*. На рисунке *б* проиллюстрирован результат выполнения первого шага алгоритма.

На *в* показан результат добавления следующего ребра $\{v_1,\ v_2\}$ с весом 2 из очереди. На *г* приведен результат добавления ребра $\{v_0,\ v_4\}$ с весом 3.

Если следующим в очереди ребром будет $\{v_1,\ v_4\}$, оно будет отброшено.

Дальнейший ход работы алгоритма зависит от того, в каком порядке в очереди размещены ребра $\{v_2,\ v_3\}$ и $\{v_3,\ v_4\}$ с весами 4. Любое из них может быть добавлено в множество ребер остовного дерева, и на этом алгоритм закончит работу. На *д* приведено остовное дерево, полученное после добавления ребра $\{v_3,\ v_4\}$.

Для приведенного графа оба ребра с весом 2 войдут в остовное дерево независимо от порядка их расположения в очереди после сортировки, а ребро $\{v_1,\ v_4\}$ не войдет ни в какое остовное дерево наименьшего веса.

3.4. Топологическая сортировка.

Определение 3.1. **Ориентированной сетью** (или просто **сетью**) называют **бесконтурный ориентированный граф**

Сеть является бесконтурным графом, поэтому существуют вершины (узлы) сети с **нулевой полустепенью исхода** , которые называют **стоками** или **выходами сети**, и вершины (узлы) с **нулевой полустепенью захода**, которые называют **источниками** или **входами сети**.

Определение 3.2.
Уровень вершины сети — это натуральное число, определяемое следующим образом:

1) если полустепень захода вершины равна 0, то ее уровень равен 0 и наоборот (т.е. нулевой уровень N_0 — это множество всех входов);
2) если множества N_i вершин уровня i определены для всех $i \leq k$, то уровень N_{k+1} содержит те и только те вершины, **предшественники** которых принадлежат любому из уровней с номером от 0 до k, причем существует хотя бы один предшественник уровня k, т.е.

$$N_{k+1} = \left\{ v \colon \Gamma^{-1}(v) \subseteq N_1 \cup \ldots \cup N_k,\ \Gamma^{-1}(v) \cap N_k \neq \varnothing \right\},$$

где $\Gamma^{-1}(v) = \{x \colon x \to v\}$ — множество предшественников вершины v.

Уровень вершины сети можно интерпретировать как **длину** максимального **пути** от входов сети до этой вершины.

Определение 3.3. **Порядковой функцией** сети $G = (V, E)$ называют отображение $\mathrm{ord}\colon V \to \mathbb{N}$, сопоставляющее каждой вершине сети номер ее уровня.

Во многих прикладных задачах возникает проблема такого упорядочения вершин сети, при котором вершины, принадлежащие одному уровню, располагаются друг под другом, а дуги ориентированного графа ведут в его изображении на плоскости от вершин с меньшим уровнем к вершинам с большим уровням слева направо.

Подобного рода проблема называется проблемой **топологической сортировки**, поскольку необходимо расположить вершины графа на плоскости так, чтобы отчетливо было видно распределение вершин по уровням.

Само расположение при этом может быть разным, лишь бы оно имело „слоистую“ структуру, в которой каждый слой составляют вершины одного уровня.

Топологическая сортировка применяется , например, при распараллеливании алгоритмов, когда по некоторому описанию алгоритма нужно составить граф зависимостей его операций и, отсортировав его топологически, определить, какие из операций являются независимыми и могут выполняться параллельно (одновременно).

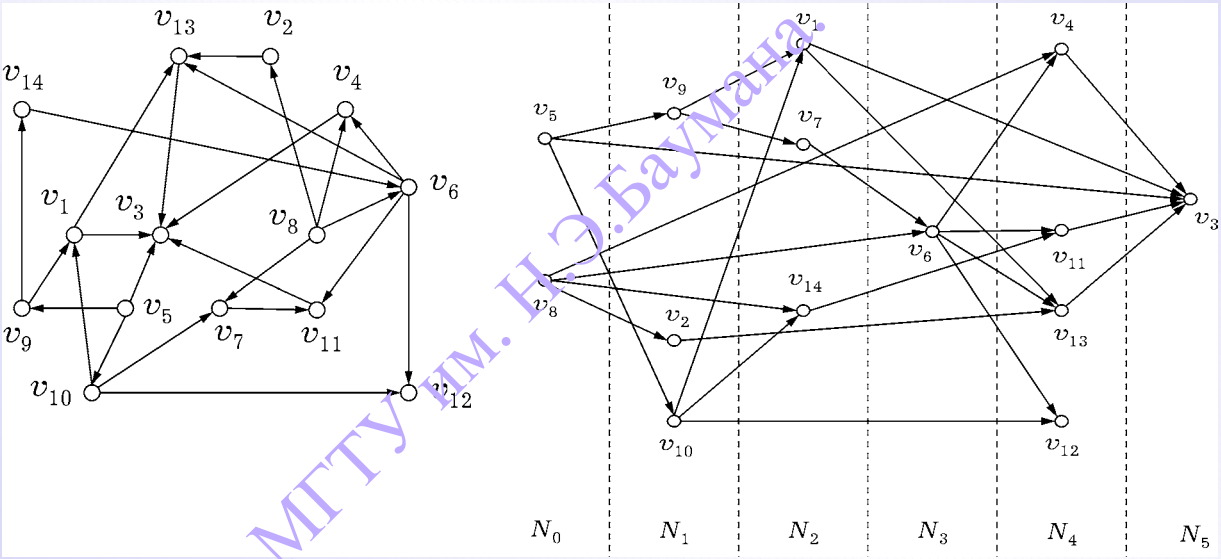


Рис. 19

Сеть и результат применения топологической сортировки сети.

Формально топологическую сортировку можно реализовать по-разному. Мы опишем один из возможных методов

Этот метод состоит в вычислении порядковой функции сети и известен как **алгоритм Демукрона**.

Предполагается, что вершины сети пронумерованы от 1 до n.

Наглядно процесс определения уровней вершин можно представить следующим образом.

Нулевой уровень образуют входы сети — вершины с полустепенью захода, равной 0.

Удалив из сети все вершины нулевого уровня и исходящие из них дуги, вновь получим сеть, входами которой будут вершины первого уровня исходной сети.

Указанный процесс „послойного“ удаления вершин следует продолжать до тех пор, пока все вершины исходной сети не будут распределены по уровням.

Алгоритм Демукрона использует матрицу смежности вершин B типа $n \times n$ в качестве средства представления сети и основан непосредственно на определении уровня вершины и свойствах матрицы B.

Сумма элементов k-го столбца матрицы B равна полустепени захода вершины v_k. Поэтому, просуммировав элементы матрицы по всем столбцам и выбрав вершины, соответствующие столбцам с нулевой суммой, получим множество вершин нулевого уровня — **входы** сети.

„Физически“ вершины и дуги из сети не удаляют, строки, соответствующие удаляемым вершинам,из матрицы B не вычеркивают.

Процесс вычисления порядковой функции.

Запишем **суммы элементов столбцов матрицы** B в **вектор** M длины n. Элемент m_k вектора M содержит полустепень захода вершины v_k.

Пусть из сети удалены вершина v_i и все исходящие из нее дуги.

Элемент b_{ik} равен 1, если из вершины v_i идет дуга в вершину v_k, и равен 0 в противном случае.

Чтобы пересчитать полустепени захода всех вершин сети, оставшихся в ней после удаления вершины v_i, надо из вектора M вычесть i-ю строку матрицы B.

Если на очередном шаге входами сети являются вершины $v_{i_1}, \ldots, v_{i_r}$, то для определения следующего „слоя“ вершин нужно из вектора M вычесть строки матрицы B с номерами $i_1, \ldots, i_r$ и зафиксировать**новые нулевые элементы** вектора M, появившиеся после вычитания. Фиксировать следует именно новые нулевые элементы, поскольку элементы вектора M, соответствующие вершинам, лежащим на предыдущих уровнях, стали равными 0 на предыдущих шагах алгоритма.

Порядковую функцию сети можно задать, указав множества вершин, принадлежащих каждому уровню, или сопоставив каждой вершине ее номер уровня. Первый способ более удобен при теоретических рассуждениях, второй — при вычислениях.

Алгоритм Демукрона вычисления порядковой функции сети

Алгоритм обрабатывает матрицу B смежности вершин графа порядка n. В результате работы алгоритма получаем массив Ord длины n, i-й элемент которого равен номеру уровня вершины v_i.

0. Сформировать множество V_1 вершин сети. Значение счетчика уровней k положить равным 0. Найти суммы элементов по всем столбцам матрицы B (полустепени захода вершин) и заполнить ими массив M.

1. Если множество V_1 не пусто, перейти на шаг **2**,если иначе, то перейти на шаг **5**.

2. Определить множество I номеров всех новых нулевых элементов массива M, т.е. таких, что соответствующие этим номерам вершины принадлежат множеству V_1.

3. Присвоить элементам массива Ord с номерами из множества I номер уровня k и удалить вершины с этими номерами из множества V_1 („замаскировать“ вершины). Вычесть из массива M строки матрицы B, соответствующие вершинам с номерами из множества I (т.е. вершинам последнего вычисленного уровня).

4. Увеличить счетчик уровней на 1 $(k := k + 1)$. Вернуться на шаг **1**.

5. Закончить работу.

Пример 3.1. Применим алгоритм Демукрона к сети, представленной на рисунке

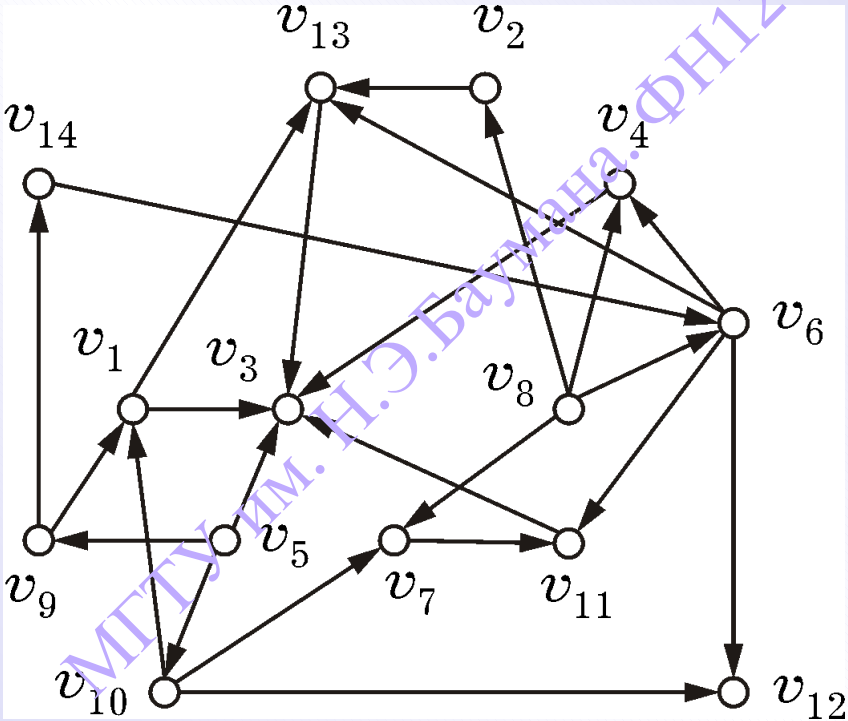


Рис. 20

Матрица смежности вершин сети имеет следующий вид:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1			1										1	
2													1	
3														
4			1											
5			1						1	1				
6				1							1	1	1	
7											1			
8		1		1		1	1							
9	1													1
10	1						1					1		
11			1											
12														
13			1											
14						1								

Покажем последовательность значений массива M, соответствующую итерациям алгоритма и множества N_i вершин i-го уровня. Прочерки соответствуют вершинам, не принадлежащим множеству V_1 („замаскированные“ вершины) на соответствующем этапе алгоритма.

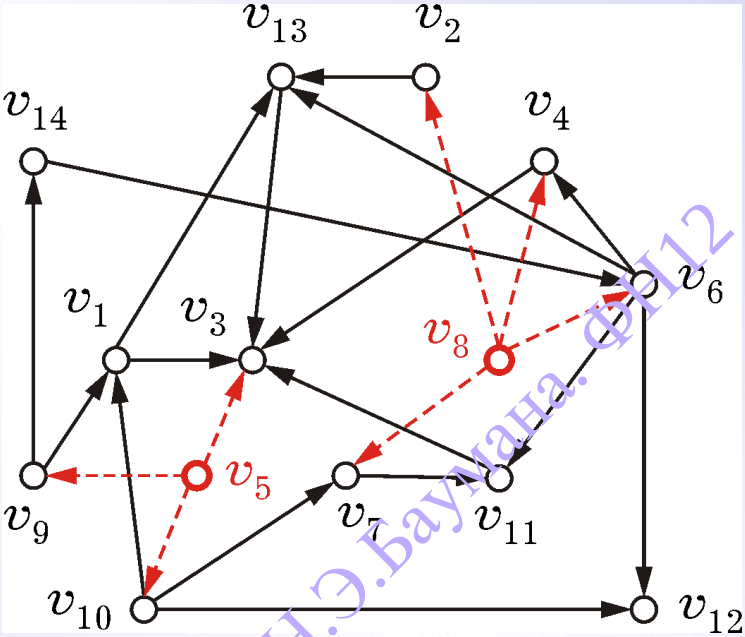


Рис. 21

1	2	3	4	5	6	7	8	9	10	11	12	13	14	
2	1	5	2	0	2	2	0	1	1	2	2	3	1	$N_0 = \{5, 8\}$

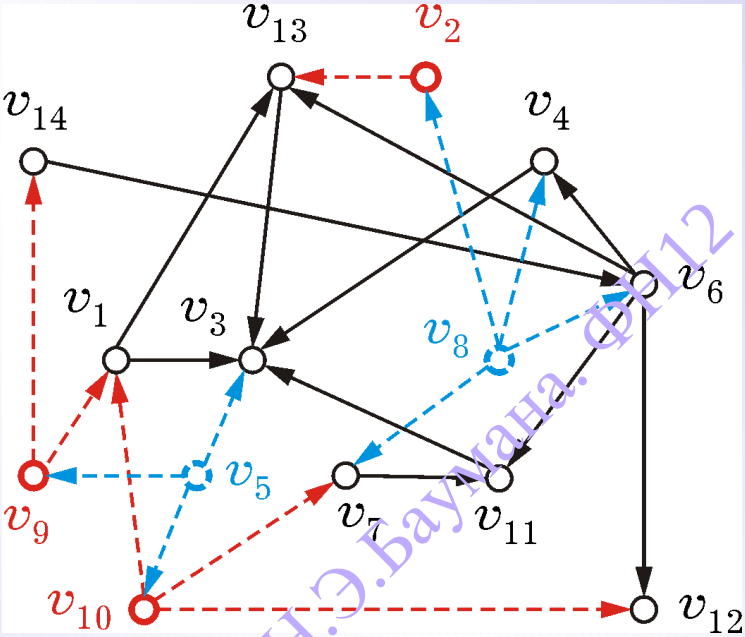


Рис. 22

1	2	3	4	5	6	7	8	9	10	11	12	13	14	
2	1	5	2	0	2	2	0	1	1	2	2	3	1	$N_0 = \{5, 8\}$
2	0	4	1	–	1	1	–	0	0	2	2	3	1	$N_1 = \{2, 9, 10\}$

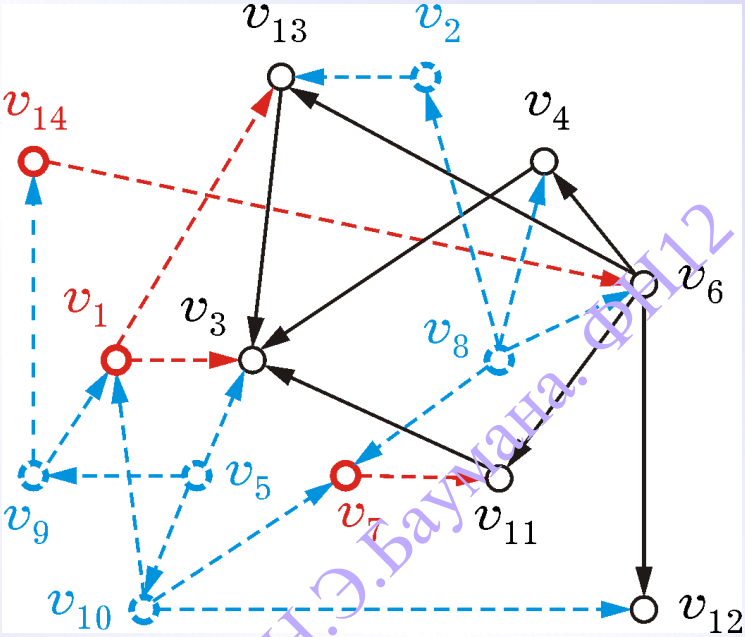


Рис. 23

1	2	3	4	5	6	7	8	9	10	11	12	13	14	
2	1	5	2	0	2	2	0	1	1	2	2	3	1	$N_0 = \{5, 8\}$
2	0	4	1	–	1	1	–	0	0	2	2	3	1	$N_1 = \{2, 9, 10\}$
0	–	4	1	–	1	0	–	–	–	2	1	2	0	$N_2 = \{1, 7, 14\}$

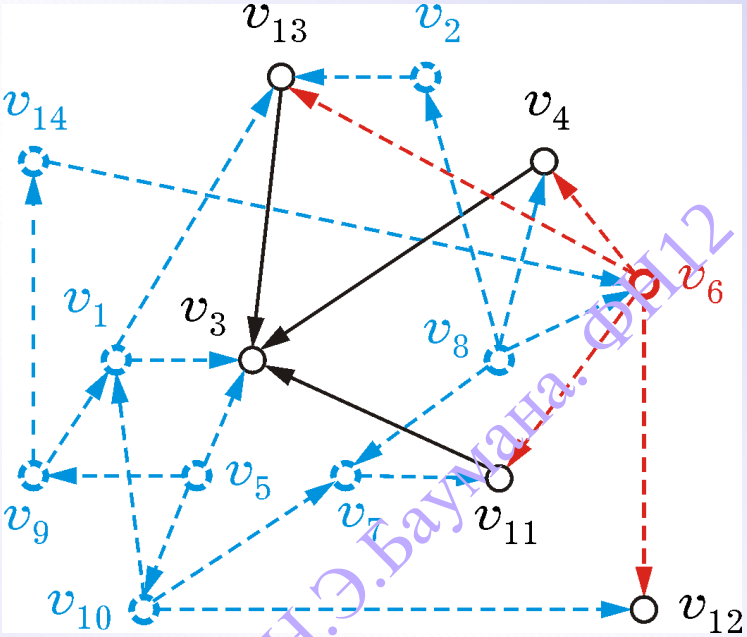


Рис. 24

1	2	3	4	5	6	7	8	9	10	11	12	13	14	
2	1	5	2	0	2	2	0	1	1	2	2	3	1	$N_0 = \{5, 8\}$
2	0	4	1	–	1	1	–	0	0	2	2	3	1	$N_1 = \{2, 9, 10\}$
0	–	4	1	–	1	0	–	–	–	2	1	2	0	$N_2 = \{1, 7, 14\}$
–	–	3	1	–	0	–	–	–	–	1	1	1	–	$N_3 = \{6\}$

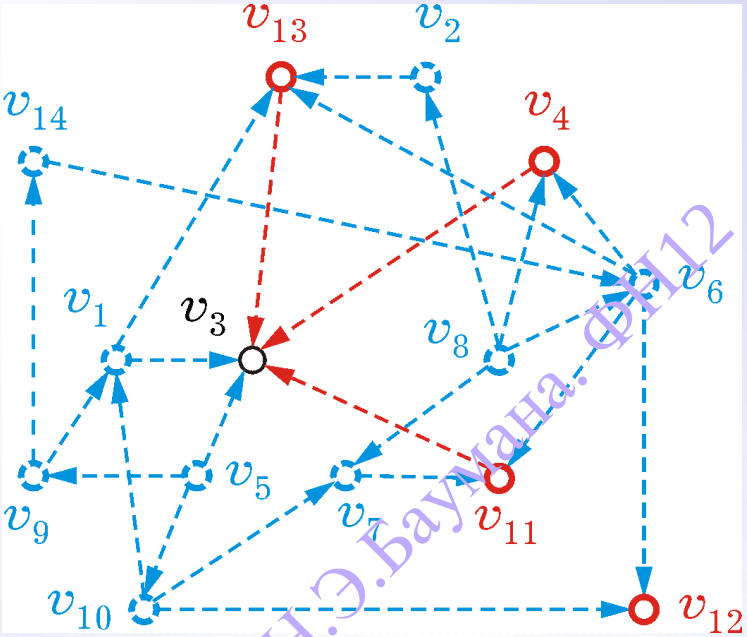


Рис. 25

1	2	3	4	5	6	7	8	9	10	11	12	13	14	
2	1	5	2	0	2	2	0	1	1	2	2	3	1	$N_0 = \{5, 8\}$
2	0	4	1	–	1	1	–	0	0	2	2	3	1	$N_1 = \{2, 9, 10\}$
0	–	4	1	–	1	0	–	–	–	2	1	2	0	$N_2 = \{1, 7, 14\}$
–	–	3	1	–	0	–	–	–	–	1	1	1	–	$N_3 = \{6\}$
–	–	3	0	–	–	–	–	–	–	0	0	0	–	$N_4 = \{4, 11, 12, 13\}$

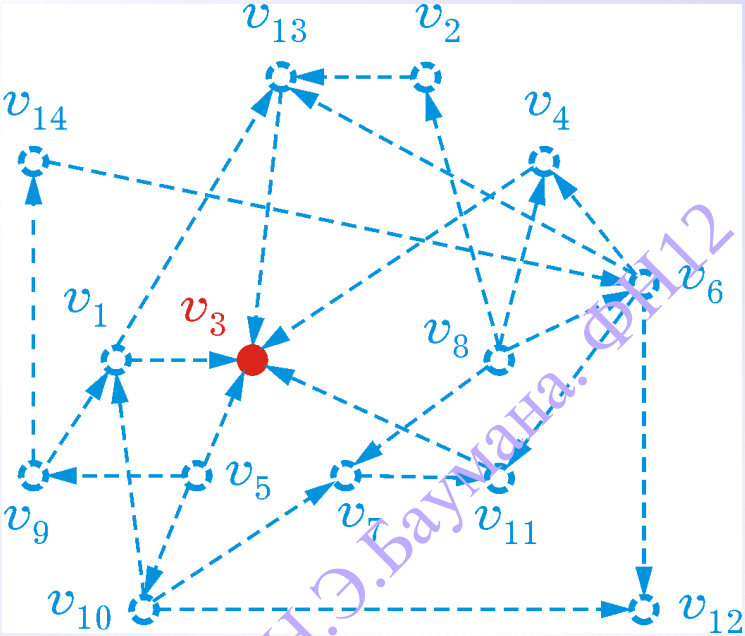


Рис. 26

1	2	3	4	5	6	7	8	9	10	11	12	13	14	
2	1	5	2	0	2	2	0	1	1	2	2	3	1	$N_0 = \{5, 8\}$
2	0	4	1	–	1	1	–	0	0	2	2	3	1	$N_1 = \{2, 9, 10\}$
0	–	4	1	–	1	0	–	–	–	2	1	2	0	$N_2 = \{1, 7, 14\}$
–	–	3	1	–	0	–	–	–	–	1	1	1	–	$N_3 = \{6\}$
–	–	3	0	–	–	–	–	–	–	0	0	0	–	$N_4 = \{4, 11, 12, 13\}$
–	–	0	–	–	–	–	–	–	–	–	–	–	–	$N_5 = \{3\}$

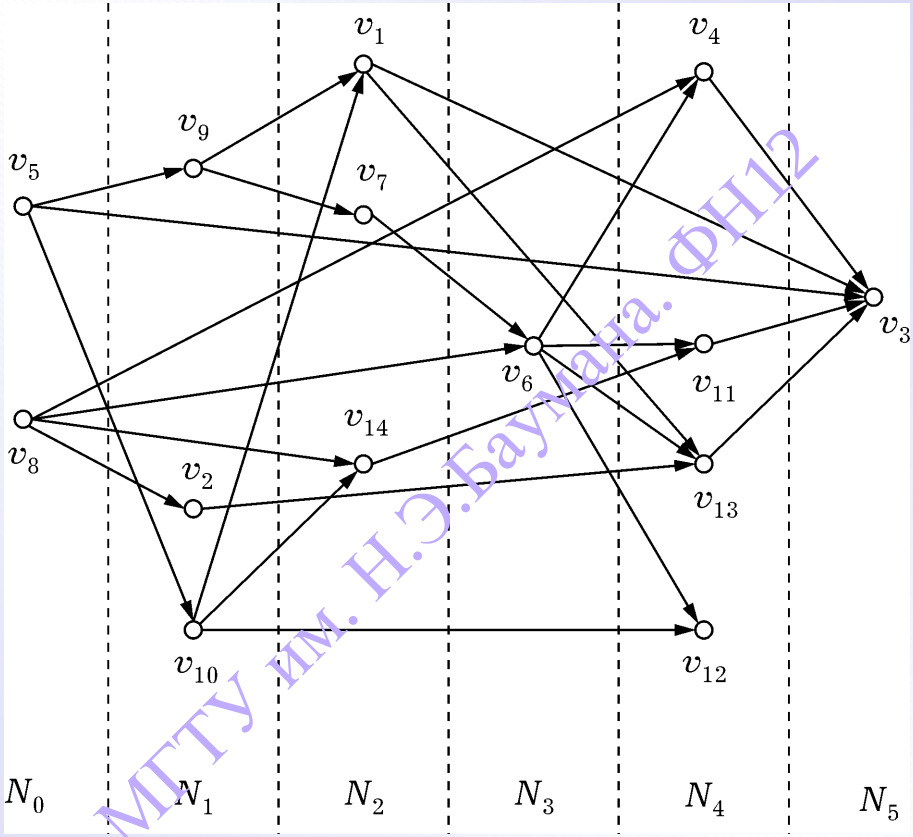


Рис. 27

Результат применения топологической сортировки сети.

Алгоритм Демукрона может быть модифицирован так, чтобы он останавливался, если ориентированный граф, поданный на вход, не является сетью, и сообщал об этом. Можно увидеть, что анализируемый граф не будет сетью тогда и только тогда, когда при очередном перевычислении массива M не появятся новые нули.