План лекции

- Графы. Представление графов.
- Обход графа. Поиск путей в графах. BFS. DFS.
- Топологическая сортировка.
- Компоненты связности. Алгоритм Косарайю.

Графы. Представление графов.

Графы: применение

- Географические карты. Какой маршрут из Москвы в Лондон требует наименьших расходов? Какой маршрут из Москвы в Лондон требует наименьшего времени? Требуется информация о связях между городами и о стоимости этих связей.
- Микросхемы. Транзисторы, резисторы и конденсаторы связаны между собой проводниками. Есть ли короткие замыкания в системе? Можно ли так переставить компоненты, чтобы не было пересечения проводников?
- Расписания задач. Одна задача не может быть начата без решения других, следовательно имеются связи между задачами. Как составить график решения задач так, чтобы весь процесс завершился за наименьшее время?

Графы: применение

- Компьютерные сети. Узлы — конечные устройства, компьютеры, планшеты, телефоны, коммутаторы, маршрутизаторы... Каждая связь обладает свойствами латентности и пропускной способности. По какому маршруту послать сообщение, чтобы она было доставлено до адресата за наименьшее время? Есть ли в сети «критические узлы», отказ которых приведёт к разделению сети на несвязные компоненты?
- Структура программы. Узлы — функции в программе. Связи — может ли одна функция вызвать другую (статический анализ) или что она вызовет в процессе исполнения программы (динамический анализ). Чтобы узнать, какие ресурсы потребуется выделять системе, требуется граф,

Графы: основные термины

- **Ориентированный граф**: $G = (V, E)$ есть пара из V — конечного множества и E — подмножества множества $V \times V$.
- **Вершины графа**: элементы множества V (*vertex, vertices*).
- **Рёбра графа**: элементы множества E (*edges*, *связи*).
- **Неориентированный граф**: рёбра есть неупорядоченные пары.
- **Петля:** ребро из вершины v_1 в вершину v_2, где $v_1 = v_2$.

Графы: основные термины

- **Смежные вершины**: v_i и v_j смежны, если имеется связь (v_i, v_j).
- **Множество смежных вершин**: обозначаем $Adj[v]$
- **Степень вершины**: величина $|Adj[v]|$
- **Путь из** v_0 **в** v_n: последовательность рёбер, таких, что $e_1 = (v_0, v_1)$, $e_2 = (v_1, v_2) \dots e_n = (v_{n-1}, v_n)$.
- **Простой путь**: путь, в котором все вершины попарно различны.
- **Длина пути**: количество n рёбер в пути.
- **Цикл**: путь, в котором $v_0 = v_n$.

Графы: основные термины

- **Неориентированный связный граф**: для любой пары вершин существует путь.
- **Связная компонента вершины** v: множество вершин неориентированного графа, до которых существует путь из v.
- **Расстояние между** $\delta(v_i, v_j)$: длина кратчайшего пути из v_i в v_j.

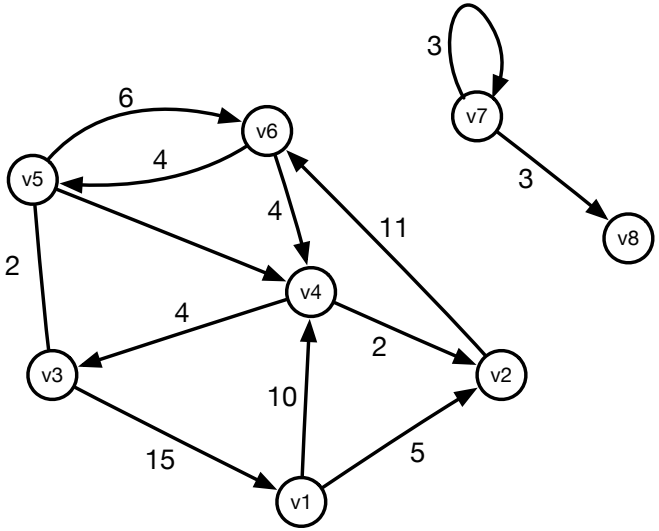
$$\delta(u, v) = 0 \Leftrightarrow u = v$$

$$\delta(u, v) \leqslant \delta(u, v') + \delta(v', v)$$

- **Дерево**: связный граф без циклов.
- **Граф со взвешенными рёбрами**: каждому ребру приписан вес $c(u, v)$.

Графы: основные термины

Ориентированный граф.



Типичные: задачи на графах

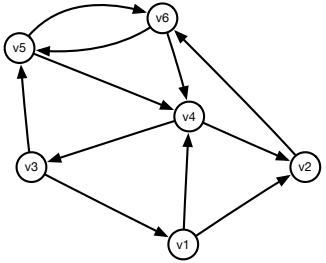
- Проверка графа на связность.
- Является ли граф деревом.
- Найти кратчайший путь из u в v.
- Найти цикл, проходящий по всем рёбрам ровно один раз (цикл Эйлера).
- Найти цикл, проходящий по всем вершинам ровно один раз (цикл Гамильтона).
- Проверка на планарность — определить, можно ли нарисовать граф на плоскости без самопересечений.

Представление графа в памяти.

- Каждой вершине сопоставляется множество смежных с ней.
- Всё представляется в виде матрицы смежности.
- Всё представляется в виде списка рёбер.

Представление графов в памяти

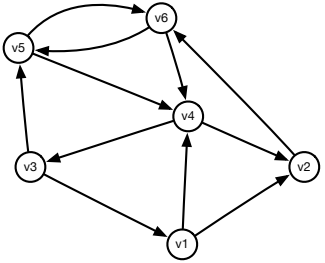
Представление графа в памяти в виде матрицы смежности



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	0	1
3	1	0	0	0	1	0
4	0	1	1	0	0	0
5	0	0	1	1	0	1
6	0	0	0	1	1	0

Представление графов в памяти

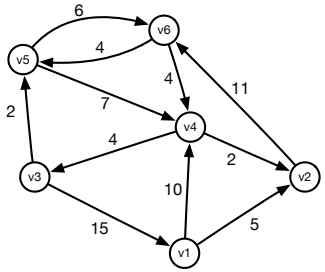
Представление графа в памяти в виде множеств смежности



$v_1 : \{2, 4\}$
$v_2 : \{6\}$
$v_3 : \{1, 5\}$
$v_4 : \{2, 3\}$
$v_5 : \{3, 4, 6\}$
$v_6 : \{4, 5\}$

Представление графов в памяти

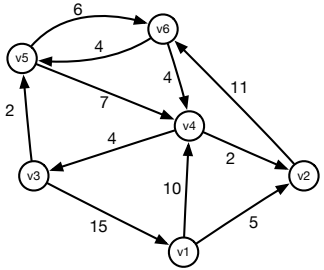
Представление взвешенного графа в памяти в виде матрицы смежности



	1	2	3	4	5	6
1	0	5	0	10	0	0
2	0	0	0	0	0	11
3	15	0	0	0	2	0
4	0	2	4	0	0	0
5	0	0	0	7	0	6
6	0	0	0	4	4	0

Представление графов в памяти

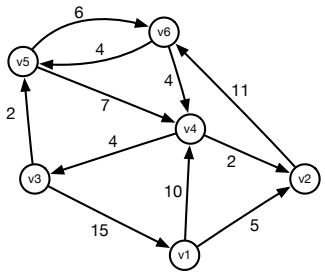
Представление взвешенного графа в памяти в виде множеств смежности



$v_1 : \{(2, 5), (4, 10)\}$
$v_2 : \{(6, 11)\}$
$v_3 : \{(1, 15), (5, 2)\}$
$v_4 : \{(2, 2), (3, 4)\}$
$v_5 : \{(4, 7), (6, 6)\}$
$v_6 : \{(4, 4), (5, 4)\}$

Представление графов в памяти

Представление взвешенного графа в памяти в виде списка рёбер



$$\{from, to, cost\} \dots$$

$$\{\{1,2,5\},\{1,4,10\},\{2,6,11\},\{3,1,15\},\{3,5,2\},$$
$$\{4,2,2\},\{5,4,7\},\{5,6,6\},\{6,4,4\},\{6,5,4\}\}$$

Представление графов в памяти

Преимущества и недостатки методов представления:

Представление	Матрица смежности	Множества смежности	Список рёбер
Занимаемая память	$O(\|V\|^2)$	$O(\|V\| + \|E\|)$	$O(\|E\|)$
Особенности	Простой доступ	Требует мало памяти для ряда графов	Можно иметь мультирёбра

Обход графа. Поиск путей в графах. BFS. DFS.

Поиск в ширину: алгоритм BFS

- Этот алгоритм сначала пытается обработать всех соседей текущей вершины.
- Используется абстракция *очередь* с методами **enqueue** и **dequeue**.
- Термин: **предшественник** $\pi(u)$ **на пути от** s: предпоследняя вершина в кратчайшем пути из s в u.
- Используются цвета:
  - Белый для непросмотренных вершин.
  - Серый для обрабатываемых вершин.
  - Чёрный для обработанных вершин.

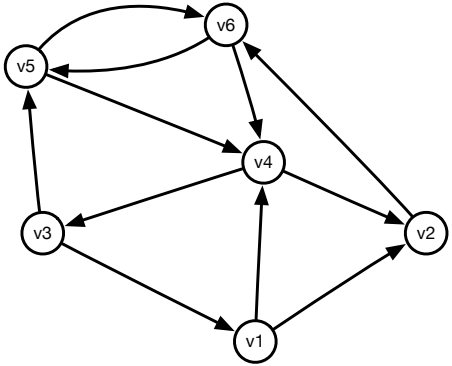
Обход графа: поиск в ширину от s, BFS

Просмотр вершин графа в порядке возрастания расстояния от s.

```
 1: procedure BFS(G : Graph, s : Vertex)
 2:     for all u ∈ V[G] \ {s} do
 3:         d[u] ← ∞;      c[u] ← white;      π[u] ← nil
 4:     end for
 5:     d[s] ← 0;    c[s] ← grey
 6:     Q.enqueue(s)
 7:     while Q ≠ ∅ do
 8:         u ← Q.dequeue()
 9:         for all v ∈ Adj[u] do
10:             if c[v] = white then
11:                 Q.enqueue(v)
12:                 d[v] ← d[u] + 1
13:                 π[v] = u;    c[v] = grey
14:             end if
15:         end for
16:         c[u] ← black
17:     end while
18: end procedure
```

Прогон алгоритма BFS

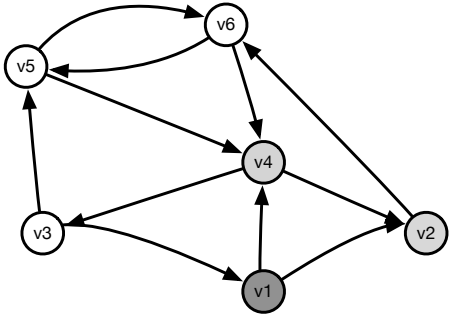
Начало алгоритма.



$d = \{0, \infty, \infty, \infty, \infty, \infty\}$
$\pi = \{nil, nil, nil, nil, nil, nil\}$
$Q = \{v_1\}$

Прогон алгоритма BFS

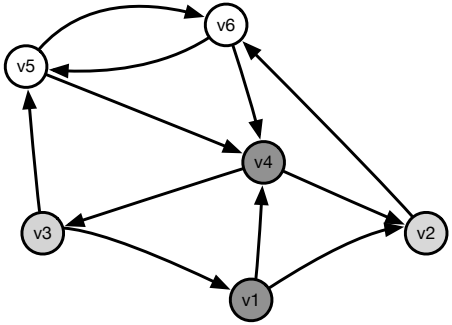
После первого прохождения цикла `While`



$d = \{0, 1, \infty, 1, \infty, \infty\}$
$\pi = \{nil, v_1, nil, v_1, nil, nil\}$
$Q = \{v_4, v_2\}$

Прогон алгоритма BFS

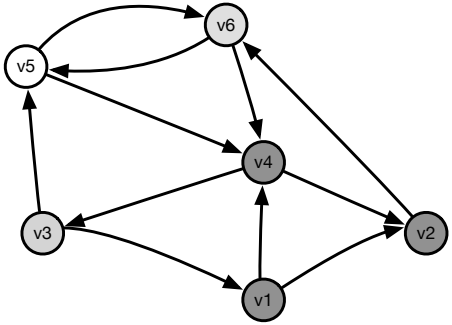
После второго прохождения цикла `While`



$d = \{0, 1, 2, 1, \infty, \infty\}$
$\pi = \{nil, v_1, v_4, v_1, nil, nil\}$
$Q = \{v_2, v_3\}$

Прогон алгоритма BFS

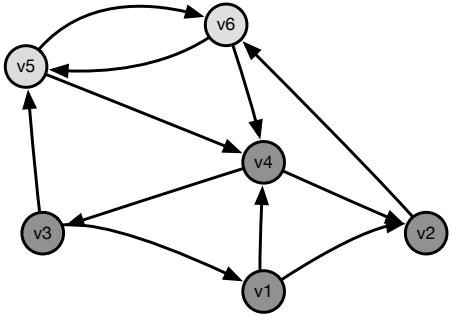
После третьего прохождения цикла `While`



$d = \{0, 1, 2, 1, \infty, 2\}$
$\pi = \{nil, v_1, v_4, v_1, nil, v_2\}$
$Q = \{v_3, v_6\}$

Прогон алгоритма BFS

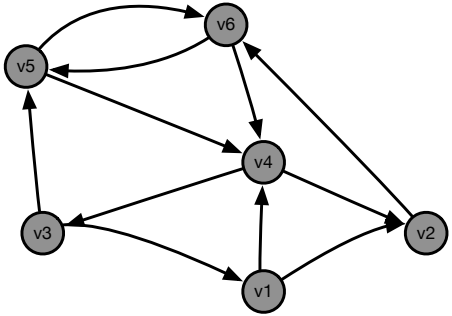
После четвёртого прохождения цикла `While`



$d = \{0, 1, 2, 1, 3, 2\}$
$\pi = \{nil, v_1, v_4, v_1, v_3, v_2\}$
$Q = \{v_6, v_5\}$

Прогон алгоритма BFS

Завершение алгоритма



$d = \{0, 1, 2, 1, 3, 2\}$
$\pi = \{nil, v_1, v_4, v_1, v_3, v_2\}$
$Q = \{\}$

Алгоритм BFS: свойства

Сложность алгоритма:

- представление в виде множества смежности:
  - Инициализация: $O(|V|)$
  - Каждая вершина обрабатывается не более одного раза. Проверяются все смежные вершины.
$$\sum_{v \in V} |Adj(v)| = O(|E|)$$
  - $T = O(|V| + |E|)$

Поиск в глубину: алгоритм DFS

- Этот алгоритм пытается идти вглубь, пока это возможно.
- Обнаружив вершину, алгоритм не возвращается, пока не обработает её полностью.
- Используются переменные
  - $time$ — глобальные часы.
  - $d[u]$ — время начала обработки вершины. u
  - $f[u]$ — время окончания обработки вершины. u
  - $\pi[u]$ — предшественник вершины u.

Алгоритм DFS

```
1:  procedure DFS(G : Graph)
2:      for all u ∈ V[G] do
3:          c[u] ← white;        π[u] ← nil
4:      end for
5:      time ← 0
6:      for all u ∈ V[G] do
7:          if c[u] = white then
8:              DFS-vizit(u)
9:          end if
10:     end for
11: end procedure
```

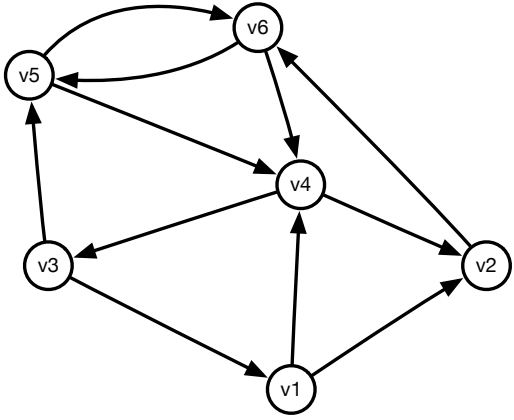
Алгоритм DFS

```
1: procedure DFS-VIZIT(u : Vertex)
2:     c[u] ← grey
3:     time ← time + 1
4:     d[u] ← time
5:     for all v ∈ Adj[u] do
6:         if c[v] = white then
7:             π[v] ← u
8:             DFS-vizit(v)
9:         end if
10:    end for
11:    c[u] ← black
12:    time ← time + 1
13:    f[u] ← time
14: end procedure
```

Прогон алгоритма DFS

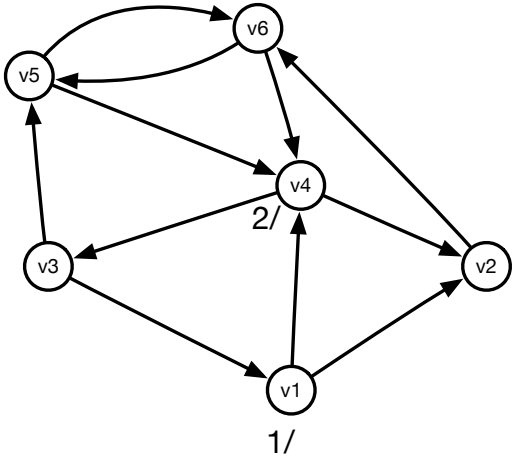
Начинается обход с вершины v_1

Около каждой вершины пишем два числа: время входа в вершину и через знак / — время выхода из вершины.



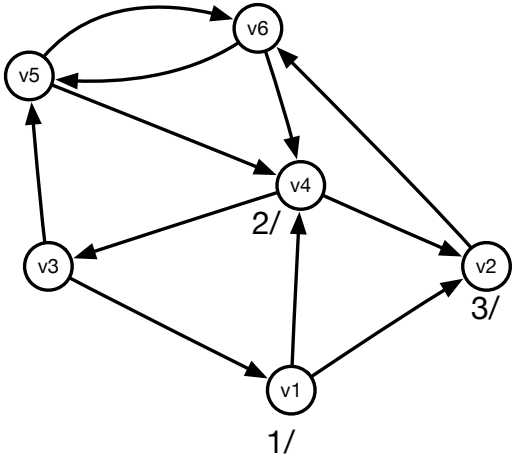
Прогон алгоритма DFS

Первый рекурсивный вызов DFS-vizit(v_4)



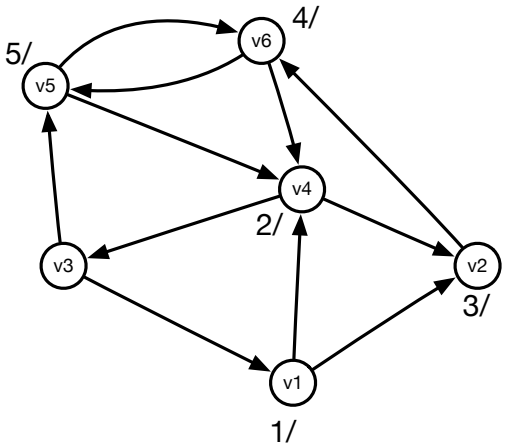
Прогон алгоритма DFS

Второй рекурсивный вызов DFS-vizit(v_2)



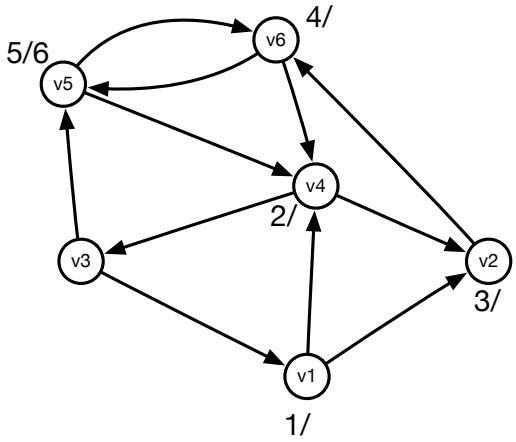
Прогон алгоритма DFS

Пятый рекурсивный вызов DFS-vizit(v_5)



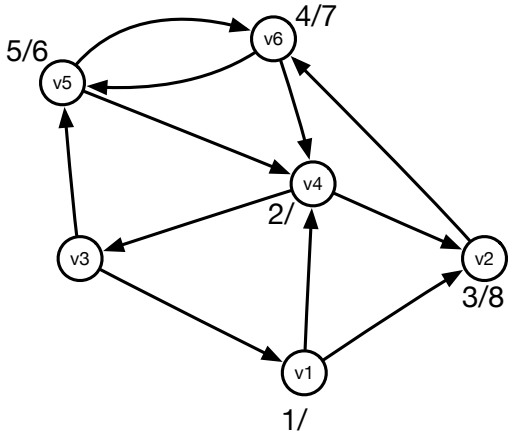
Прогон алгоритма DFS

Выход из пятой рекурсии вызова DFS-vizit(v_5)



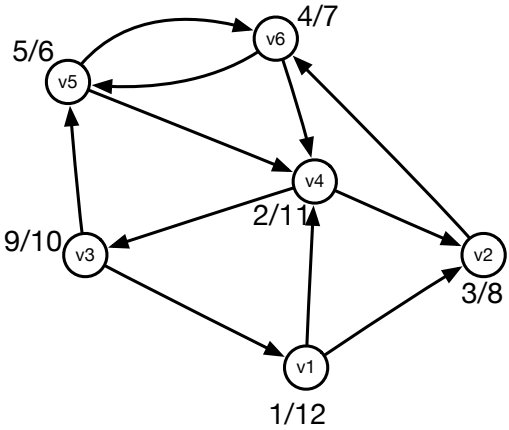
Прогон алгоритма DFS

Выход из рекурсии вызов DFS-vizit(v_2)



Прогон алгоритма DFS

Завершение алгоритма



Алгоритм DFS

- Сложность алгоритма для представления в виде множества смежности равна $O(|V| + |E|)$
- Этот алгоритм не находит кратчайшие маршруты!

Топологическая сортировка

Топологическая сортировка

Задача: имеется ориентированный граф $G = (V, E)$ без циклов.

Требуется указать такой порядок вершин на множестве V, что любое ребро ведёт из меньшей вершины к большей.
Требуемая структура данных: `L` — очередь с операцией `enqueue`.

Топологическая сортировка

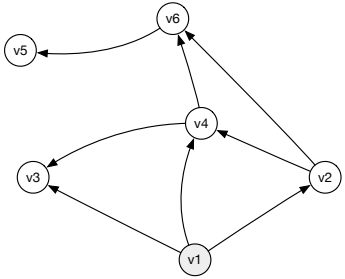
```
1: procedure TopoSort(G : Graph)
2:     L ← 0
3:     for all u ∈ V[G] do
4:         c[u] ← white;
5:     end for
6:     time ← 0
7:     for all u ∈ V[G] do
8:         if c[v] = white then
9:             DFS-vizit(u)
10:        end if
11:    end for
12: end procedure
```

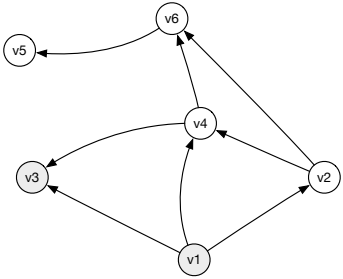
Топологическая сортировка

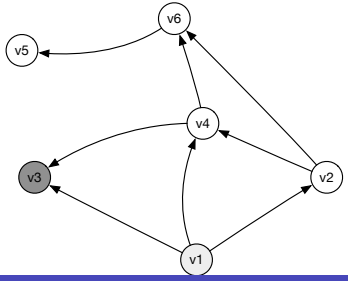
```
1: procedure DFS-VIZIT(u : Vertex)
2:     c[u] ← grey
3:     for all v ∈ Adj[u] do
4:         if c[v] = white then
5:             π[v] ← u
6:             DFS-vizit(u)
7:         end if
8:     end for
9:     c[u] ← black
10:    L.enqueue(u)
11: end procedure
```

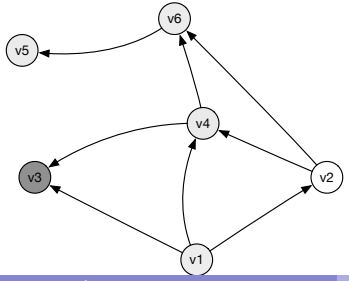
Прогон алгоритма топологической сортировки

Пусть обход начнётся с вершины v_1



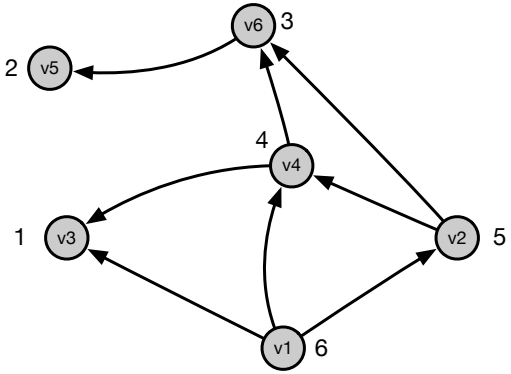






Прогон алгоритма топологической сортировки

Результат обхода.



Порядок вершин (добавляем **в начало** по номерам):
$V_1, V_2, V_4, V_6, V_5, V_3$

Поиск компонент связности

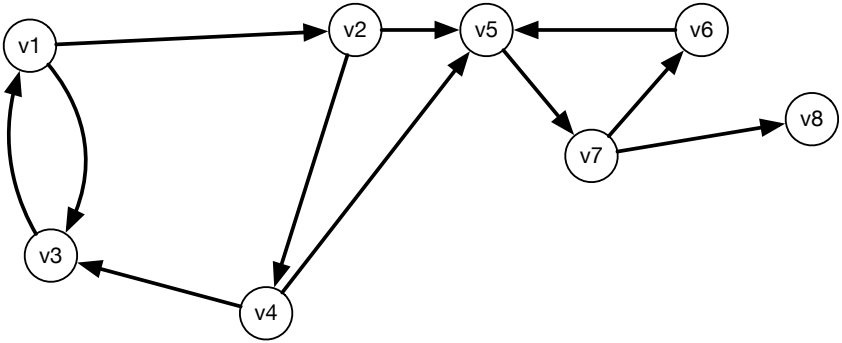
Поиск компонент связности

- Для неориентированных графов: запустив поиск `BFS` или `DFS`.
- Все выкрашенные по завершении поиска вершины образуют компоненту связности.
- Выбирается произвольным образом необработанная вершина и алгоритм повторяется, формируя другую компоненту связности.
- Алгоритм заканчивается, когда не остаётся необработанных вершин.

Поиск компонент связности

- Для ориентированных графов: результаты зависят от порядка обхода вершин.
- **Компонента сильной связности ориентированного графа:** максимальное по размеру множество вершин, взаимно достижимых друг из друга.

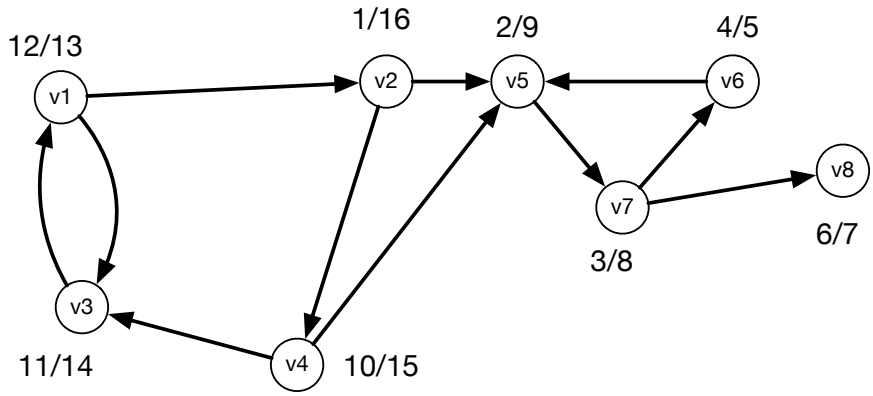
Поиск компонент связности: алгоритм Косарайю



Поиск компонент связности: алгоритм Косарайю

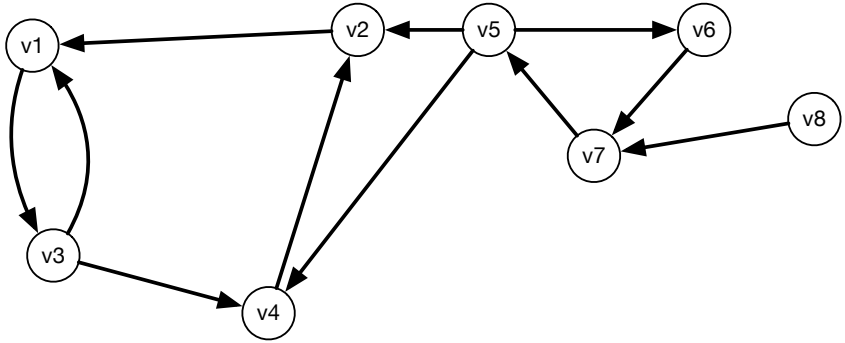
- Проведём полный DFS поиск.
- В алгоритме полного DFS не специфицировано, с какой вершины начинается поиск $\rightarrow$ можно выбрать произвольную.

Обход с вершины v_2:



Поиск компонент связности: алгоритм Косарайю

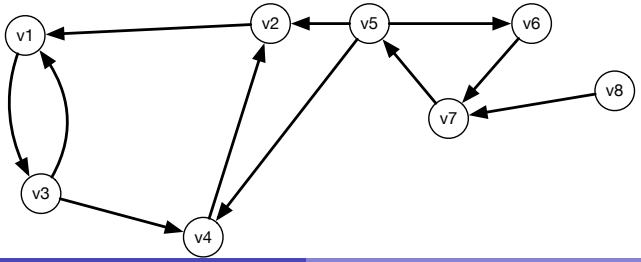
- Заменим направления всех рёбер (перевернём все стрелки).
- Каждое ребро $u \to v$ заменяется на $v \to u$.



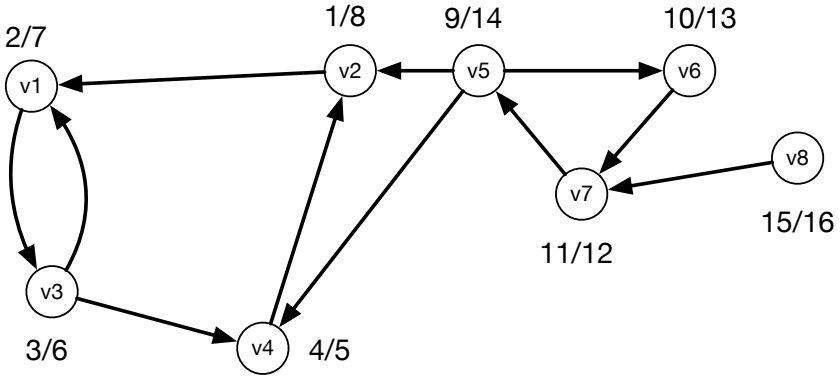
Поиск компонент связности: алг. Косарайю

- Обходим ещё раз. Начальная вершина — из необработанных, у которой наибольшие значение времени выхода.
- Обход из вершины 2 покрасил вершины v_1, v_2, v_3 и v_4.
- Остались непокрашенные вершины v_5, v_6, v_7 и v_8.
- Повторяем, пока останутся непокрашенные вершины.

Номер	1	2	3	4	5	6	7	8
Вход/выход	12/13	1/16	11/14	10/15	2/9	4/5	3/8	6/7

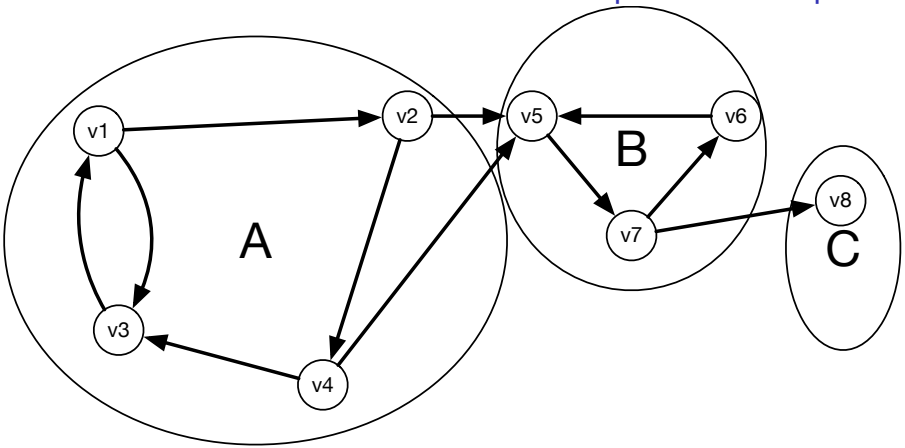


Поиск компонент связности: алгоритм Косарайю



- Каждый «малый» проход алгоритма DFS даст нам вершины, которые принадлежат одной компоненте сильной связности.

Поиск компонент связности: алгоритм Косарайю



- Рассматривая компоненту сильной связности как единую мета-вершину, мы получаем новый граф, который называется *конденсацией* исходного графа или *конденсированным графом*.