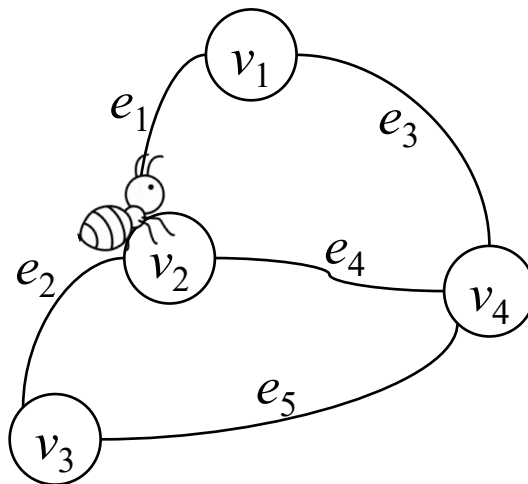


# 第2章 连通和遍历

程龚

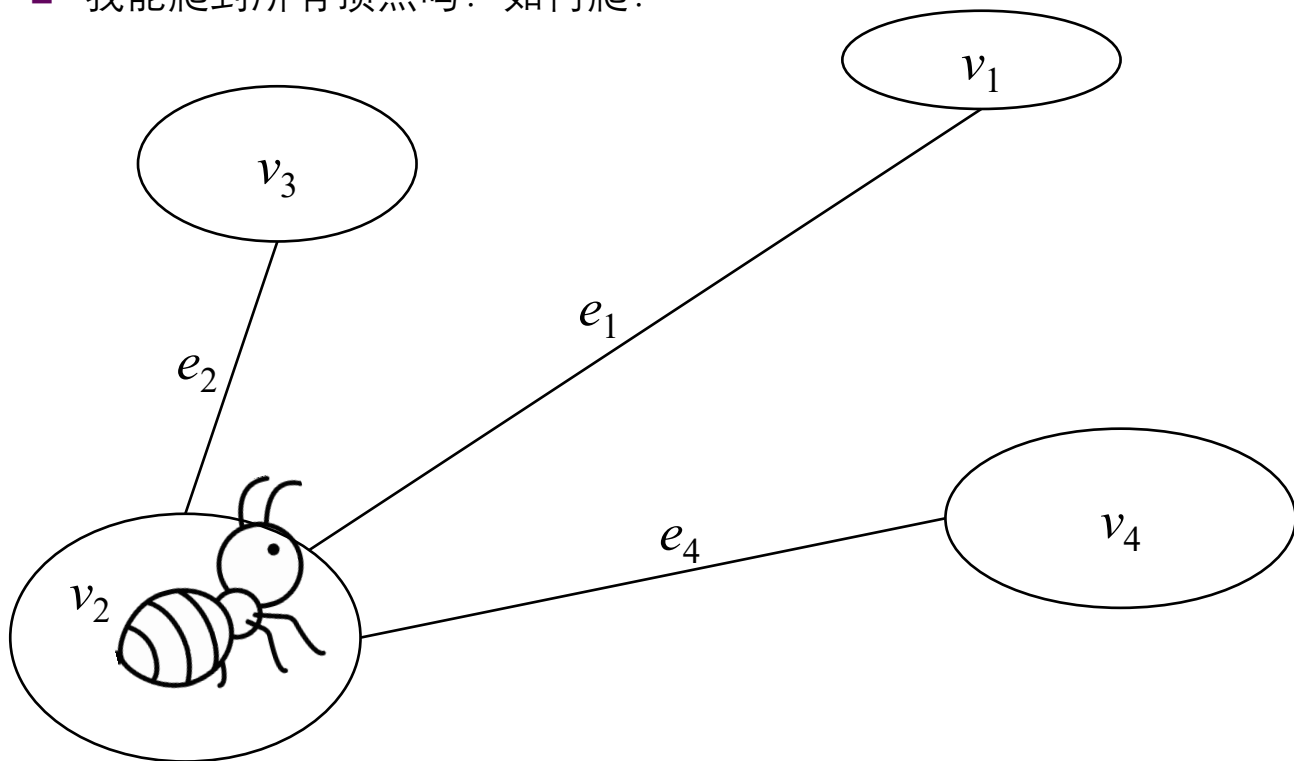
# 遍历——上帝视角

- 它能爬到所有顶点吗？如何爬？

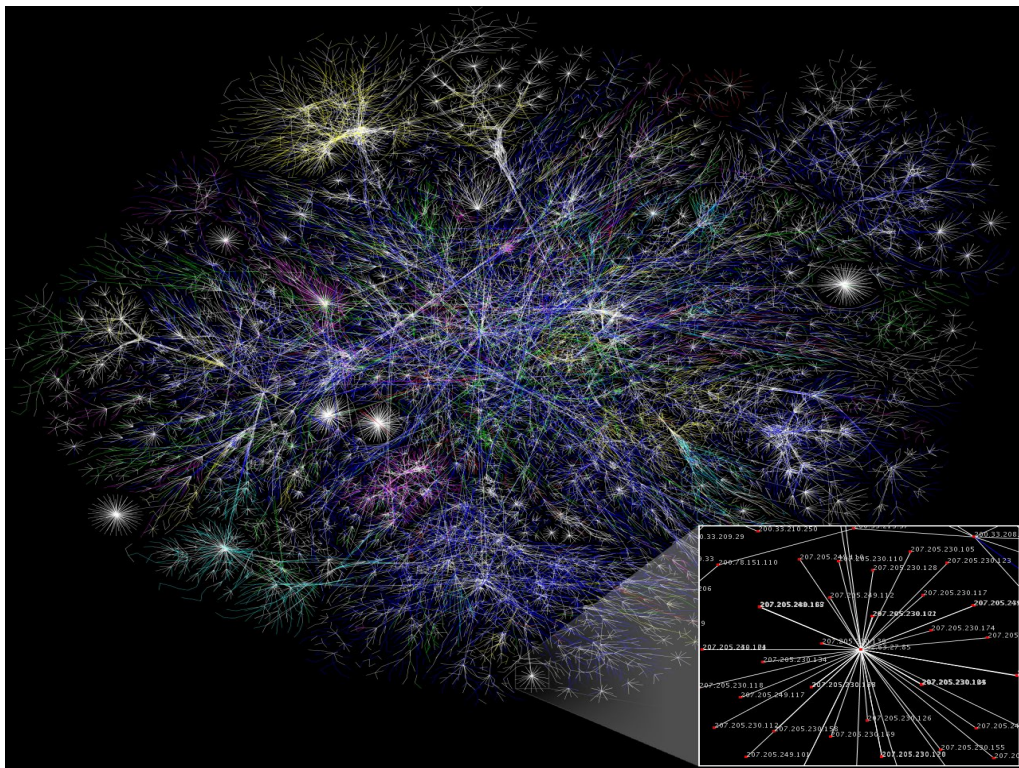


## 遍历——蚂蚁视角

- 我能爬到所有顶点吗？如何爬？

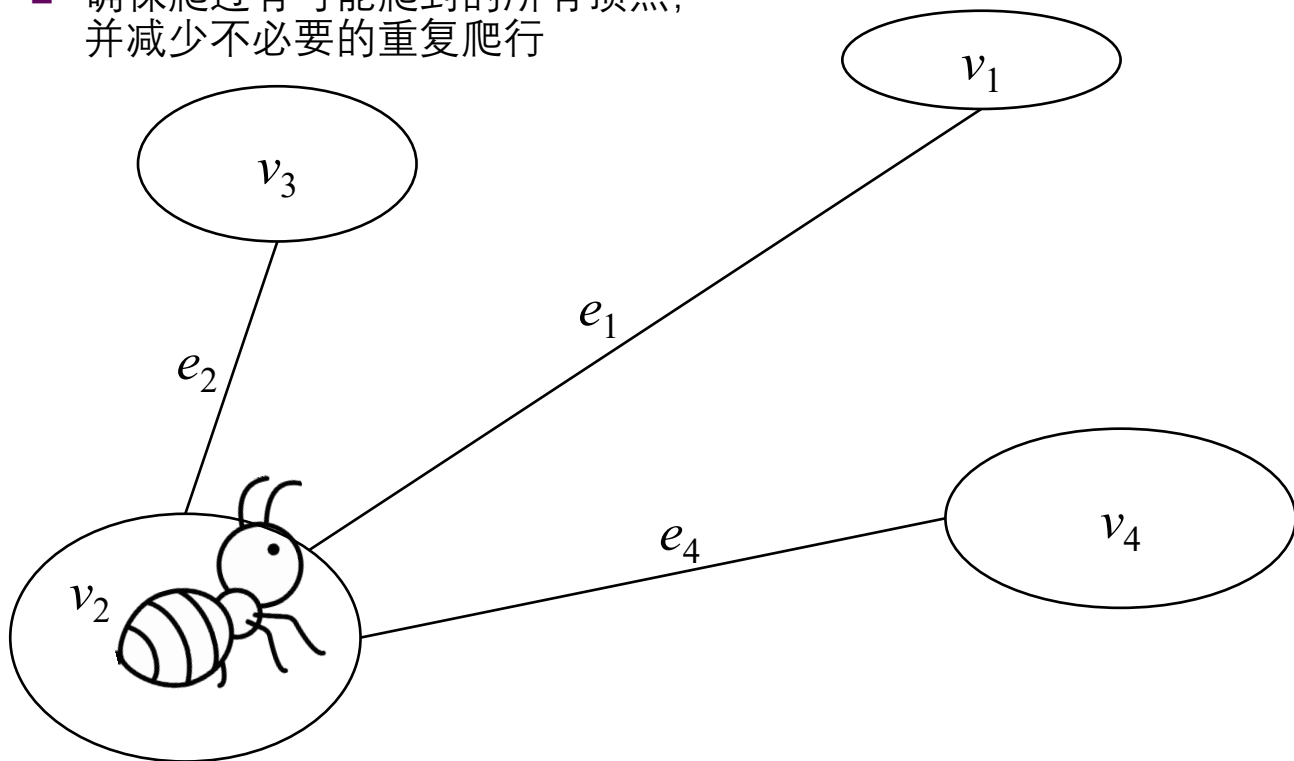


# 遍历——蚂蚁视角



## 遍历——蚂蚁视角

- 确保爬过有可能爬到的所有顶点，并减少不必要的重复爬行





# 遍历——第一人称视角



[https://www.njmetro.com.cn/njdtweb/dtweb/images/map\\_new.jpg](https://www.njmetro.com.cn/njdtweb/dtweb/images/map_new.jpg)  
<http://www.dianping.com/review/833785054>

# 本次课的主要内容

2.1 连通和DFS

2.2 割点和割边

2.3 距离和BFS

# 本次课的主要内容

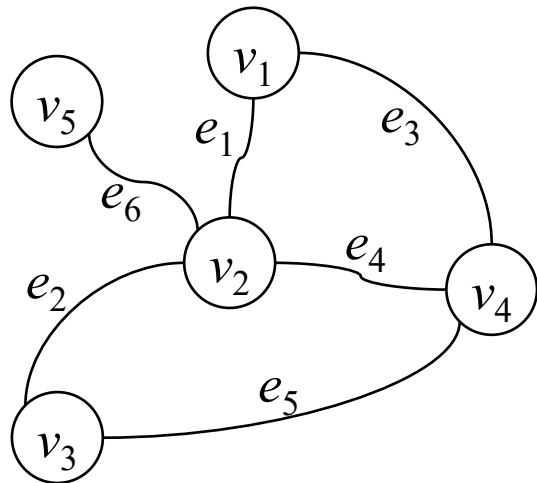
2.1 连通和DFS

2.2 割点和割边

2.3 距离和BFS

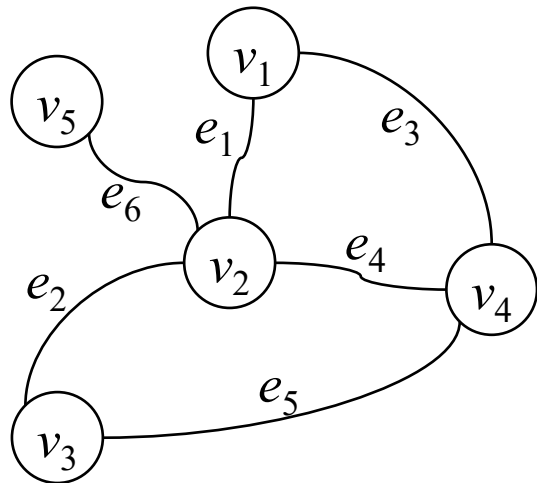
# 连通和DFS

- **路线**: 以顶点开始、顶点和边交替出现、以顶点结束的序列  $v_0, e_1, v_1, \dots, e_l, v_l$ , 其中每条边  $e_i$  的两个端点恰为顶点  $v_{i-1}$  和  $v_i$ 
  - 起点:  $v_0$
  - 终点:  $v_l$
  - 长度:  $l$



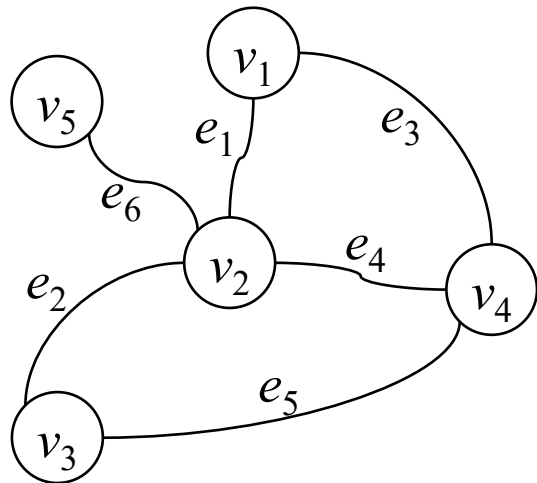
# 连通和DFS

- 路线：以顶点开始、顶点和边交替出现、以顶点结束的序列  $v_0, e_1, v_1, \dots, e_l, v_l$ ，其中每条边  $e_i$  的两个端点恰为顶点  $v_{i-1}$  和  $v_i$ 
  - 起点：  $v_0$
  - 终点：  $v_l$
  - 长度：  $l$
- 平凡路线：  $l = 0$



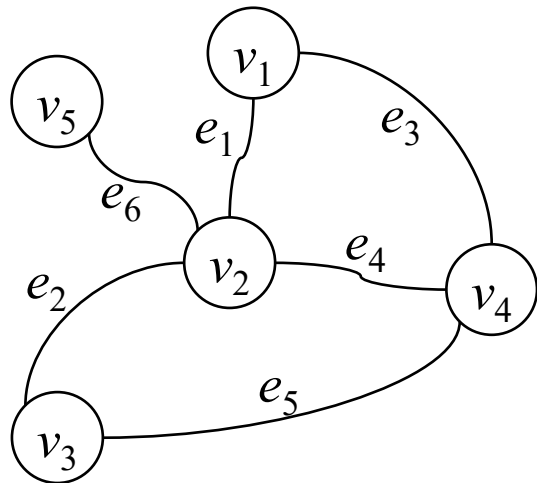
# 连通和DFS

- 路线：以顶点开始、顶点和边交替出现、以顶点结束的序列  $v_0, e_1, v_1, \dots, e_l, v_l$ ，其中每条边  $e_i$  的两个端点恰为顶点  $v_{i-1}$  和  $v_i$ 
  - 起点：  $v_0$
  - 终点：  $v_l$
  - 长度：  $l$
- 平凡路线：  $l=0$
- 迹：路线，且边在序列中不重复出现



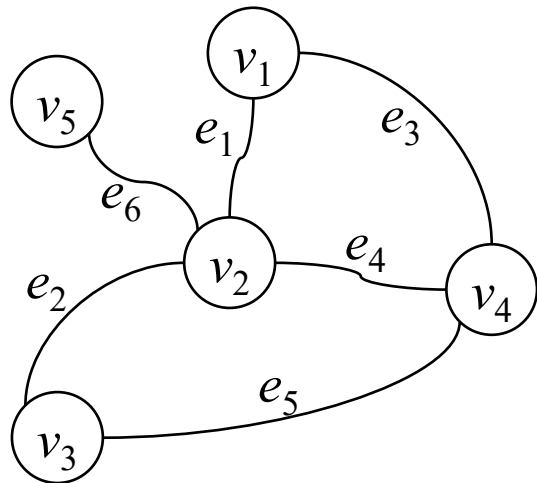
# 连通和DFS

- 路线：以顶点开始、顶点和边交替出现、以顶点结束的序列  $v_0, e_1, v_1, \dots, e_l, v_l$ ，其中每条边  $e_i$  的两个端点恰为顶点  $v_{i-1}$  和  $v_i$ 
  - 起点：  $v_0$
  - 终点：  $v_l$
  - 长度：  $l$
- 平凡路线：  $l = 0$
- 迹：路线，且边在序列中不重复出现
- 路：迹，且顶点在序列中不重复出现
  - 内顶点：路中除起点和终点外的其它顶点



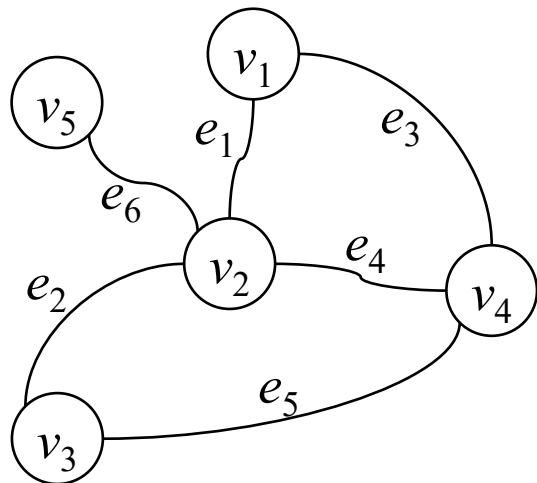
# 连通和DFS

- 若图中存在 $u$ - $v$ 路线，则一定存在 $u$ - $v$ 迹吗？
  - $v_3, e_2, v_2, e_4, v_4, e_3, v_1, e_1, v_2, e_4, v_4$
- 若图中存在 $u$ - $v$ 迹，则一定存在 $u$ - $v$ 路吗？



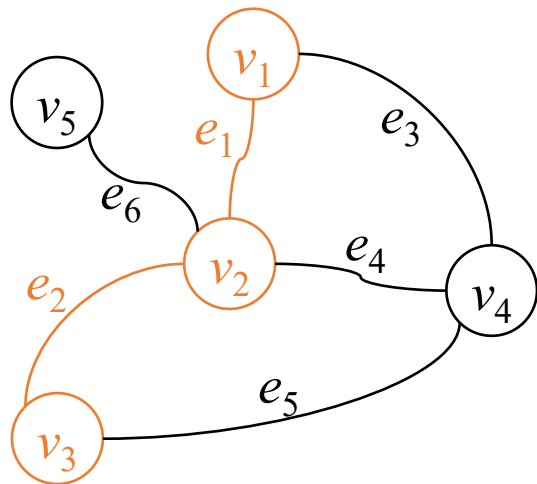
# 连通和DFS

- 若图中存在 $u-v$ 路线，则一定存在 $u-v$ 迹吗？
  - $v_3, e_2, v_2, e_4, v_4, e_3, v_1, e_1, v_2, e_4, v_4$
- 若图中存在 $u-v$ 迹，则一定存在 $u-v$ 路吗？
- 若图中存在 $u-v$ 路线和 $v-w$ 路线，则一定存在 $u-w$ 路线吗？



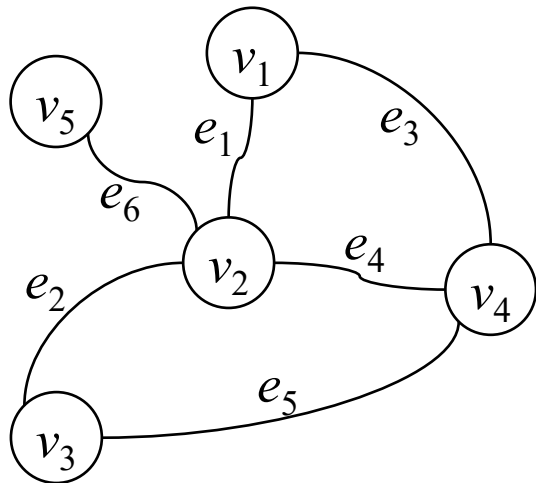
# 连通和DFS

- 顶点 $u$ 和 $v$ 连通：存在 $u-v$ 路



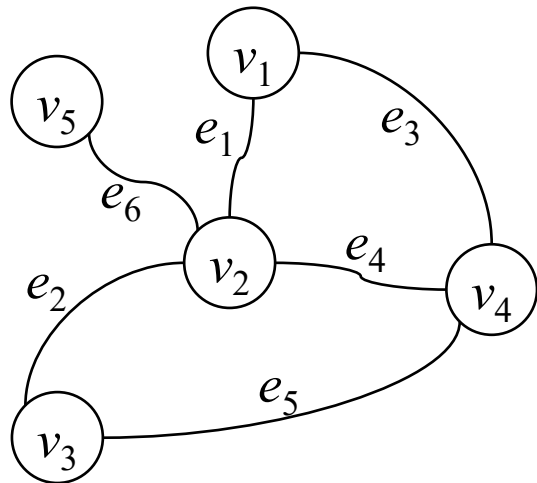
# 连通和DFS

- 顶点 $u$ 和 $v$ 连通：存在 $u$ - $v$ 路
- 连通关系是定义在顶点集上的等价关系。



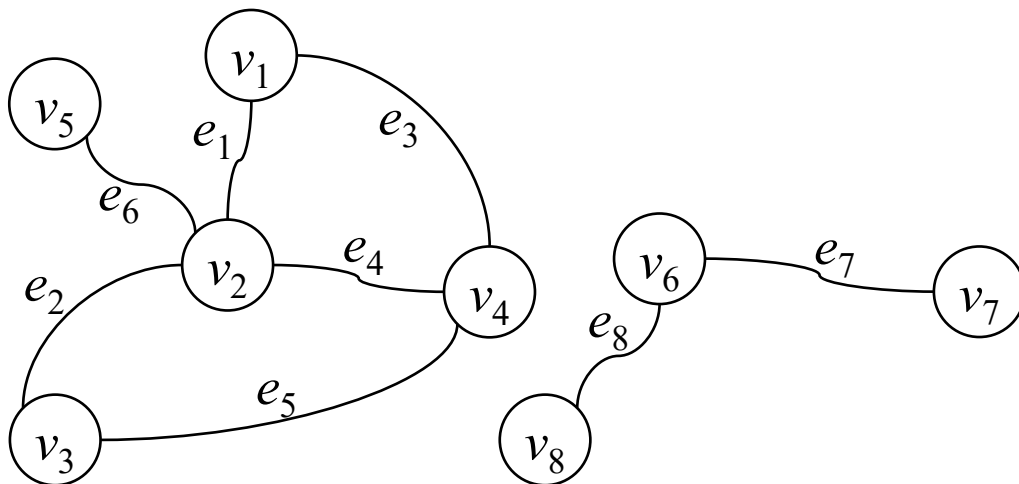
## 连通和DFS

- 顶点 $u$ 和 $v$ 连通：存在 $u$ - $v$ 路
- 连通关系是定义在顶点集上的等价关系。
- 图 $G = \langle V, E \rangle$  **连通**：  $V$ 中每对顶点都连通



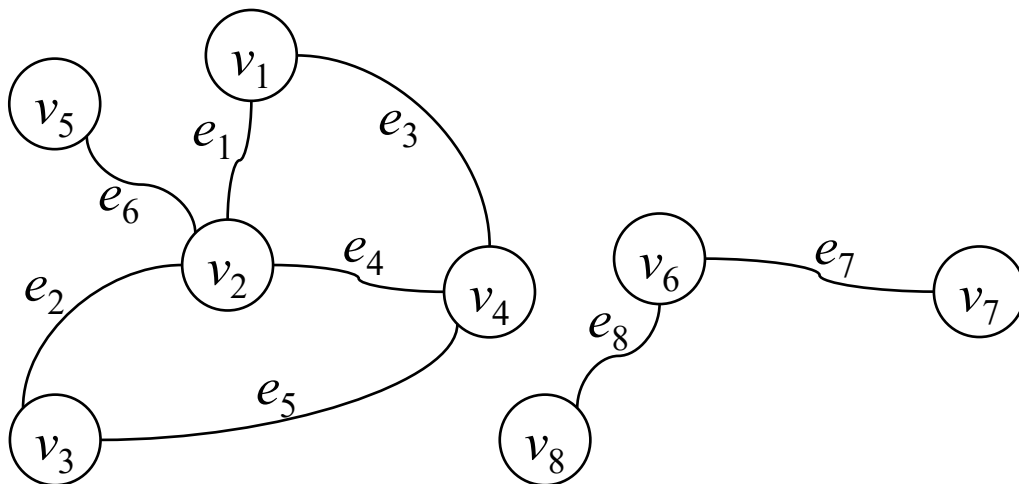
# 连通和DFS

- 连通分支：极大连通子图
- 平凡连通分支：阶为1



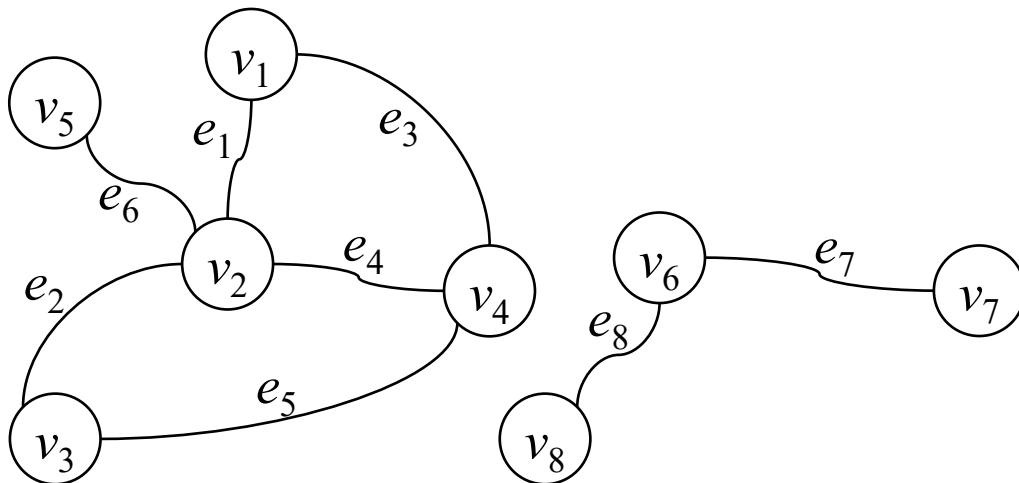
# 连通和DFS

- 连通分支：极大连通子图
- 平凡连通分支：阶为1
- 顶点集 $V$ 上的连通关系将 $V$ 划分为若干子集，每个子集 $V_i \subseteq V$ 的点导出子图 $G[V_i]$ 形成一个连通分支



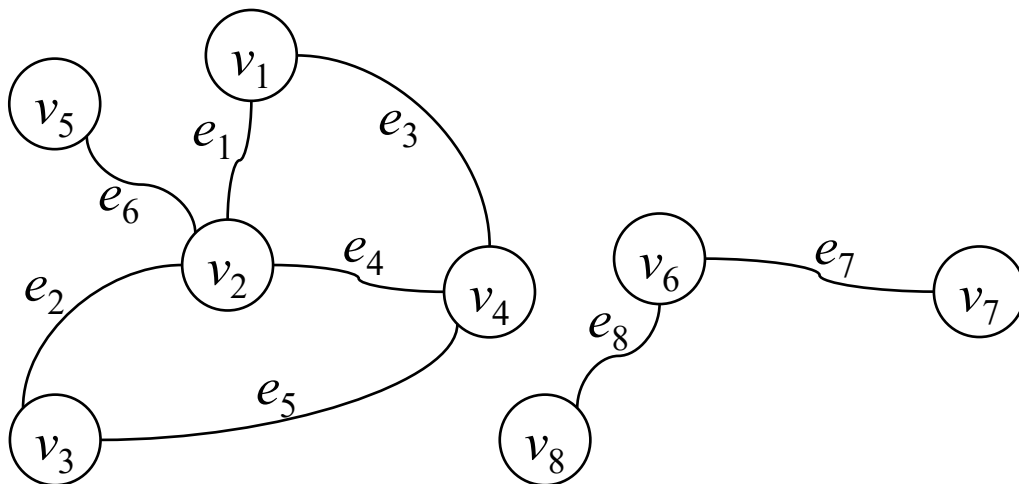
## 连通和DFS

- 若图 $G$ 连通，则 $\bar{G}$ 连通吗？若 $G$ 不连通，则 $\bar{G}$ 连通吗？



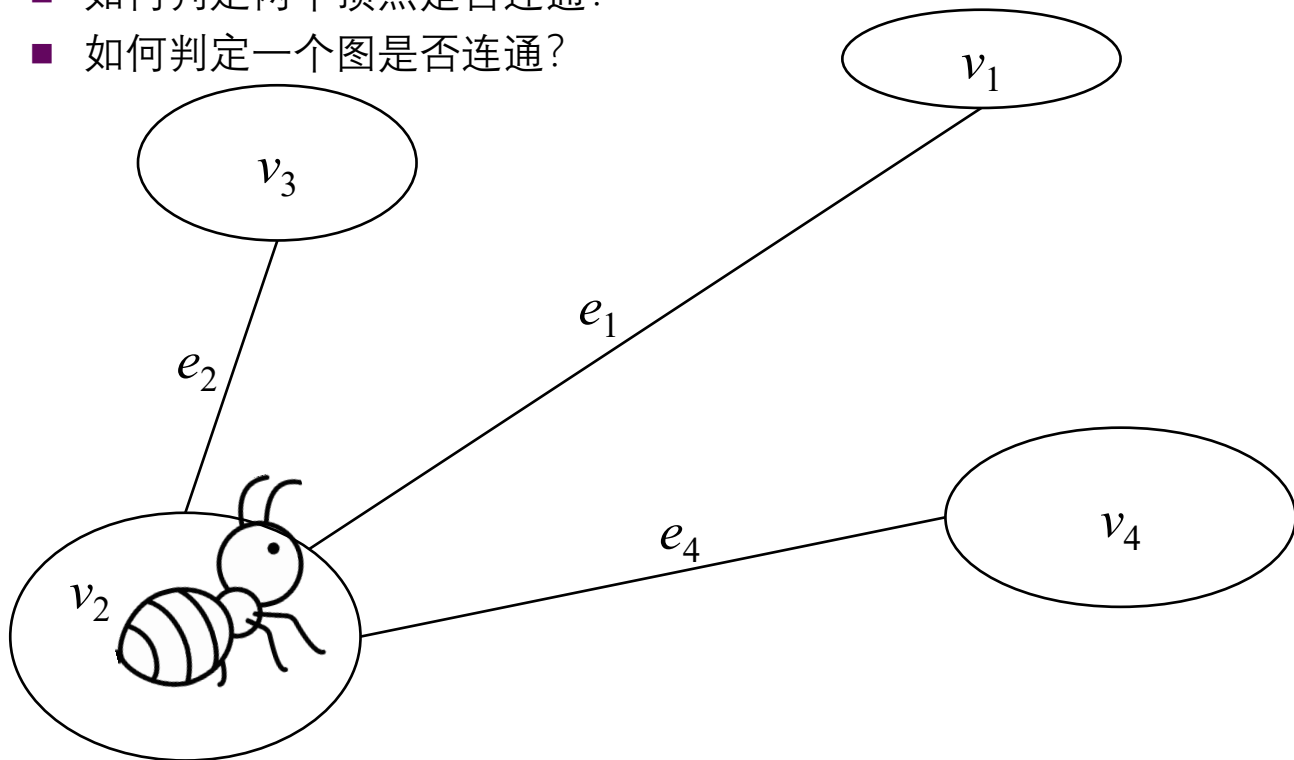
## 连通和DFS

- 若图 $G$ 连通，则 $\bar{G}$ 连通吗？若 $G$ 不连通，则 $\bar{G}$ 连通吗？
- 自补图是连通图吗？



## 连通和DFS

- 如何判定两个顶点是否连通？
- 如何判定一个图是否连通？



# 连通和DFS

## ■ 深度优先搜索 (DFS) 算法

- 从图中的一个指定顶点出发, 有序地遍历和该顶点连通的所有顶点
- 遍历顶点的顺序是优先向图的“深处”访问, 即倾向于远离出发点

---

### 算法 2.1: DFS

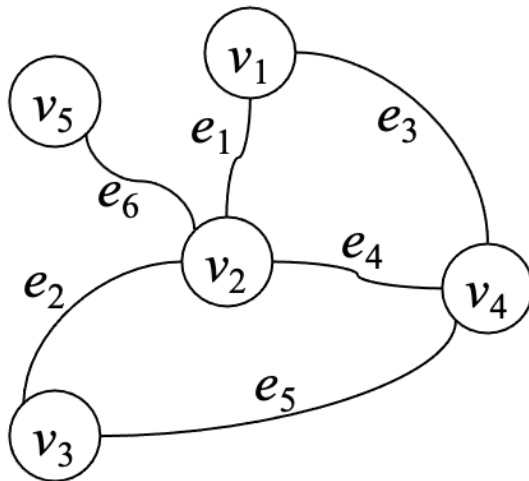
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 `visited` 初值为 `false`

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 连通和DFS

## ■ 深度优先搜索 (DFS) 算法

- 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
- 遍历顶点的顺序是优先向图的“深处”访问，即倾向于远离出发点

---

### 算法 2.1: DFS

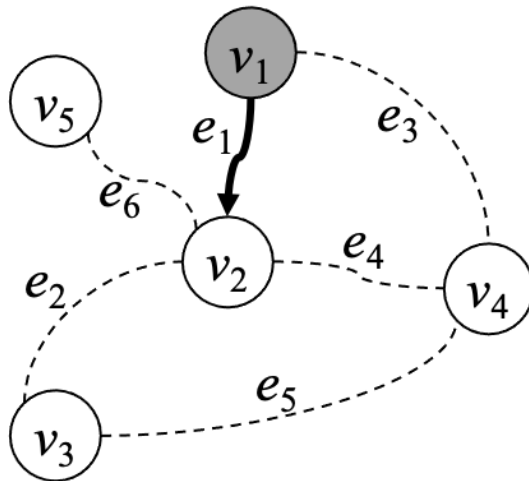
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 `visited` 初值为 `false`

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 连通和DFS

## ■ 深度优先搜索 (DFS) 算法

- 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
- 遍历顶点的顺序是优先向图的“深处”访问，即倾向于远离出发点

---

### 算法 2.1: DFS

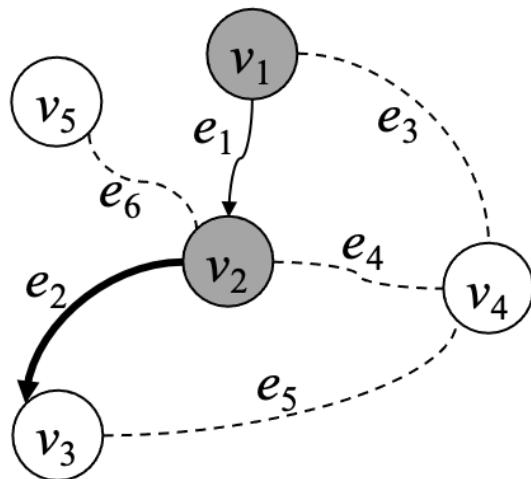
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 `visited` 初值为 `false`

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 连通和DFS

## ■ 深度优先搜索 (DFS) 算法

- 从图中的一个指定顶点出发, 有序地遍历和该顶点连通的所有顶点
- 遍历顶点的顺序是优先向图的“深处”访问, 即倾向于远离出发点

---

### 算法 2.1: DFS

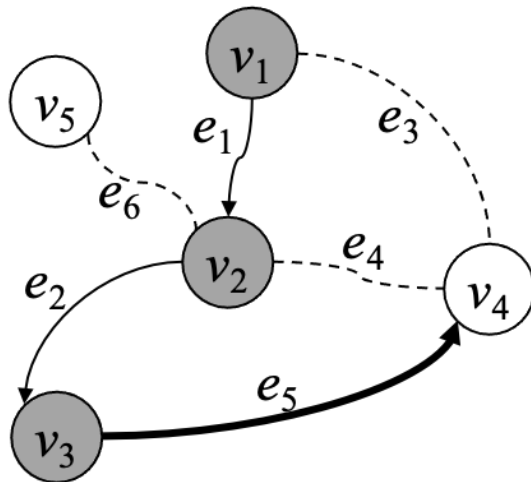
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 `visited` 初值为 `false`

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 连通和DFS

## ■ 深度优先搜索 (DFS) 算法

- 从图中的一个指定顶点出发, 有序地遍历和该顶点连通的所有顶点
- 遍历顶点的顺序是优先向图的“深处”访问, 即倾向于远离出发点

---

### 算法 2.1: DFS

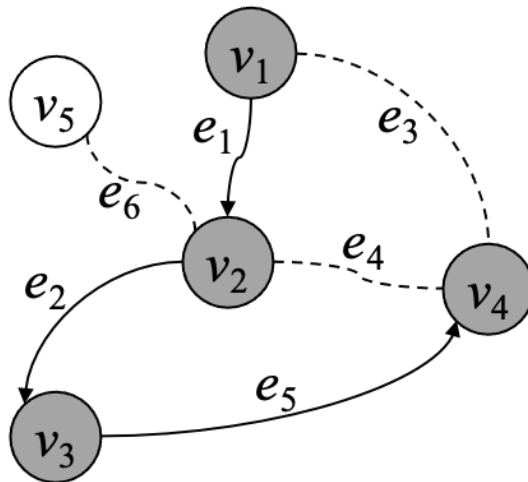
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 `visited` 初值为 `false`

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 连通和DFS

## ■ 深度优先搜索 (DFS) 算法

- 从图中的一个指定顶点出发, 有序地遍历和该顶点连通的所有顶点
- 遍历顶点的顺序是优先向图的“深处”访问, 即倾向于远离出发点

---

### 算法 2.1: DFS

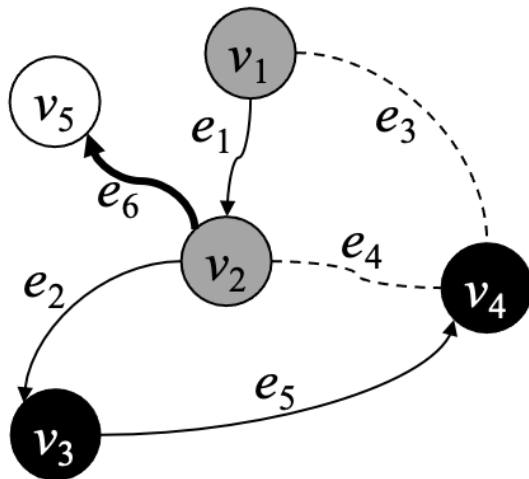
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 `visited` 初值为 `false`

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 连通和DFS

## ■ 深度优先搜索 (DFS) 算法

- 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
- 遍历顶点的顺序是优先向图的“深处”访问，即倾向于远离出发点

---

### 算法 2.1: DFS

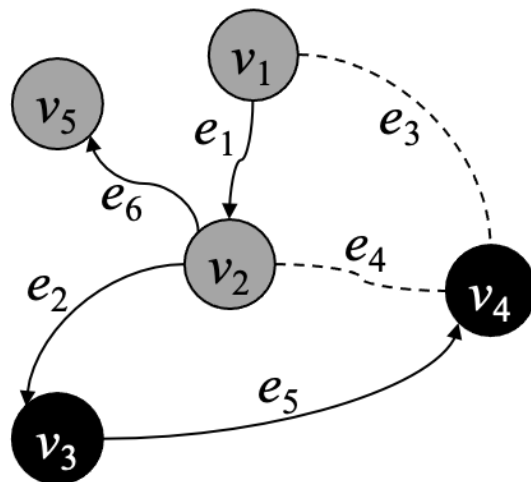
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 `visited` 初值为 `false`

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 连通和DFS

## ■ 深度优先搜索 (DFS) 算法

- 从图中的一个指定顶点出发, 有序地遍历和该顶点连通的所有顶点
- 遍历顶点的顺序是优先向图的“深处”访问, 即倾向于远离出发点

---

### 算法 2.1: DFS

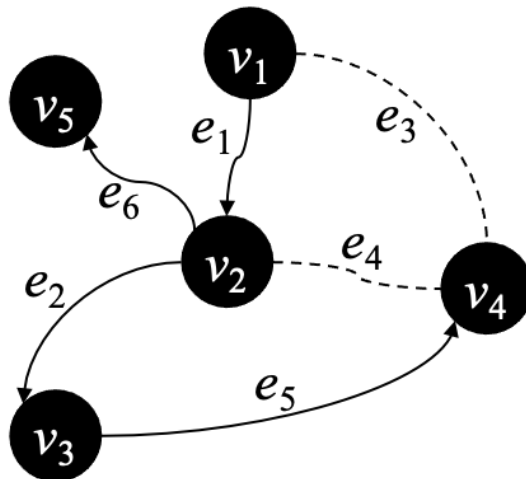
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 `visited` 初值为 `false`

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 连通和DFS

- 从顶点 $u$ 出发运行DFS算法，为什么恰能访问与 $u$ 连通的所有顶点？
  - 为什么访问的都连通？
  - 为什么连通的都能访问？

---

## 算法 2.1: DFS

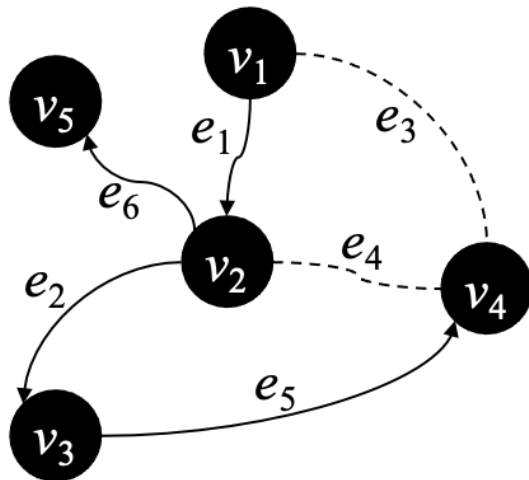
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 `visited` 初值为 `false`

```
1  $u.\text{visited} \leftarrow \text{true};$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.\text{visited} = \text{false}$  then  
4   |   |  $\text{DFS}(G, v);$ 
```

---



# 连通和DFS

- 从顶点 $u$ 出发运行DFS算法，为什么恰能访问与 $u$ 连通的所有顶点？
  - 为什么访问的都连通？
  - 为什么连通的都能访问？

---

## 算法 2.1: DFS

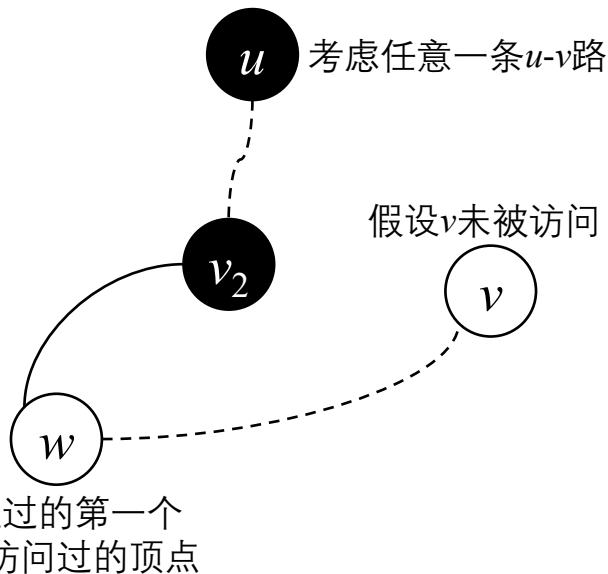
---

**输入:** 图  $G = \langle V, E \rangle$ , 顶点  $u$

**初值:**  $V$  中所有顶点的 `visited` 初值为 `false`

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 连通和DFS

- 时间复杂度:  $O(n + m)$

---

## 算法 2.1: DFS

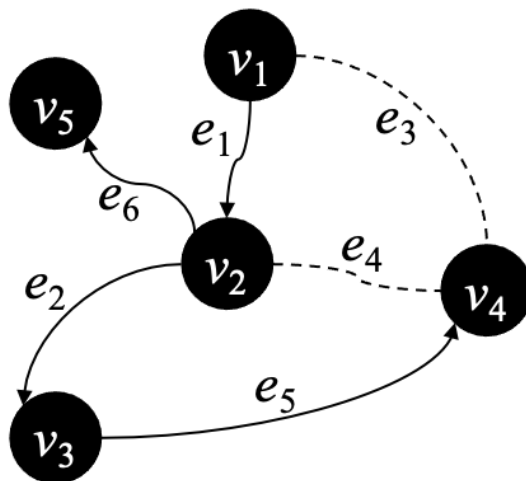
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 本次课的主要内容

2.1 连通和DFS

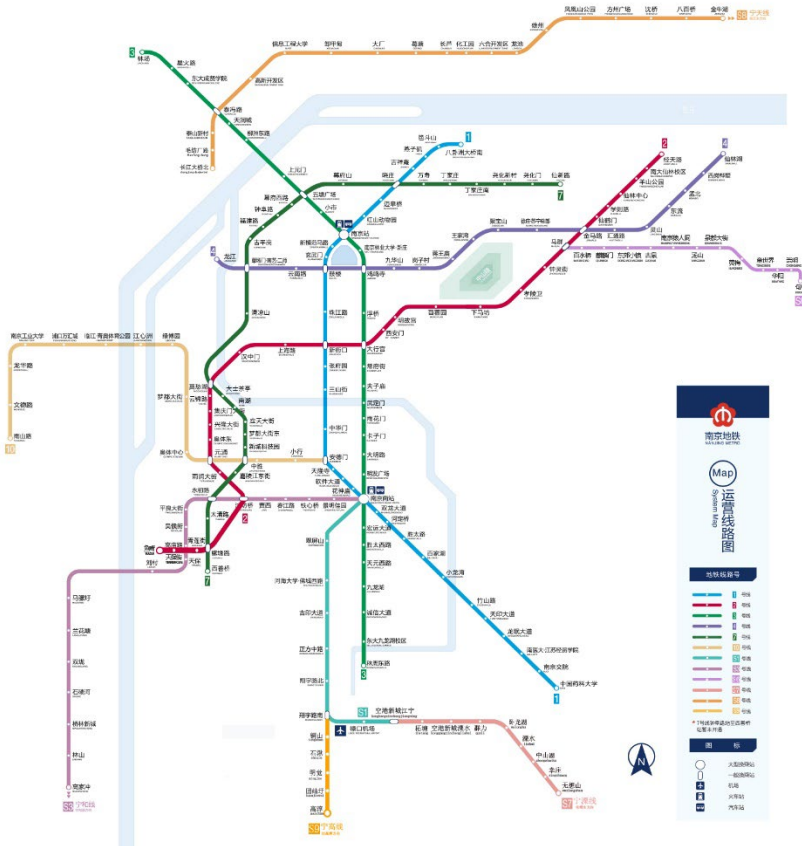
2.2 割点和割边

2.3 距离和BFS



# 割点和割边

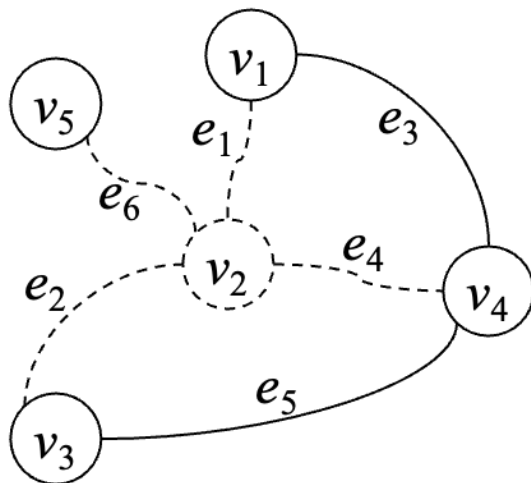
- 哪座地铁站、哪段线路最薄弱?



# 割点和割边

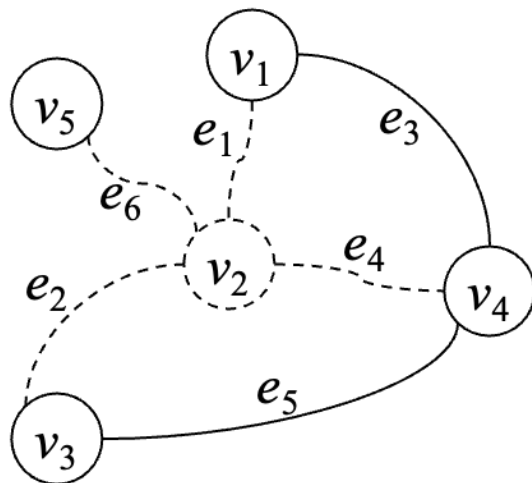
## ■ 割点 (关节点)

- 狭义定义: 对于连通图 $G = \langle V, E \rangle$ 和顶点 $v \in V$ ,  $G - v$ 不连通
- 广义定义: 对于图 $G = \langle V, E \rangle$ 和顶点 $v \in V$ ,  $G - v$ 的连通分支数量大于 $G$ 的



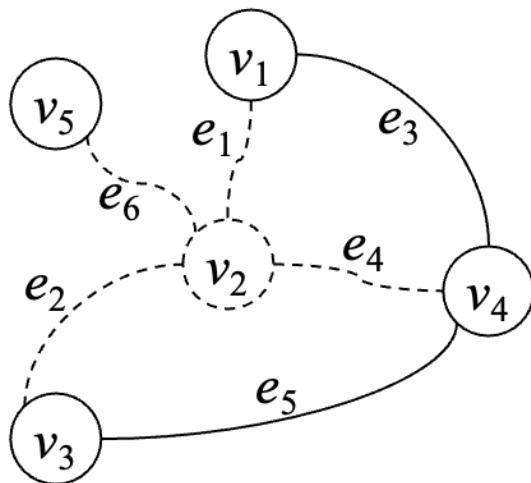
## 割点和割边

- 割点的度的下界是多少？



# 割点和割边

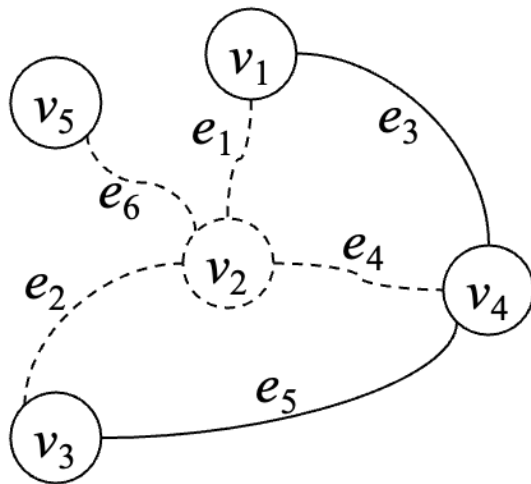
- 割点的度的下界是多少？
- 割点的充要条件：对于连通图 $G = \langle V, E \rangle$ 和顶点 $v \in V$ ， $v$ 是 $G$ 的割点当且仅当存在 $V$ 的两个不相交的非空子集 $V_i$ 和 $V_j$ ，对于任意顶点 $u \in V_i$ 和 $w \in V_j$ ，每条 $u-w$ 路都经过 $v$ 。



# 割点和割边

- 割点的度的下界是多少？
- 割点的充要条件：对于连通图 $G = \langle V, E \rangle$ 和顶点 $v \in V$ ， $v$ 是 $G$ 的割点当且仅当存在 $V$ 的两个不相交的非空子集 $V_i$ 和 $V_j$ ，对于任意顶点 $u \in V_i$ 和 $w \in V_j$ ，每条 $u-w$ 路都经过 $v$ 。
- 若顶点 $v$ 是连通图 $G$ 的割点，则 $v$ 也是 $\bar{G}$ 的割点吗？

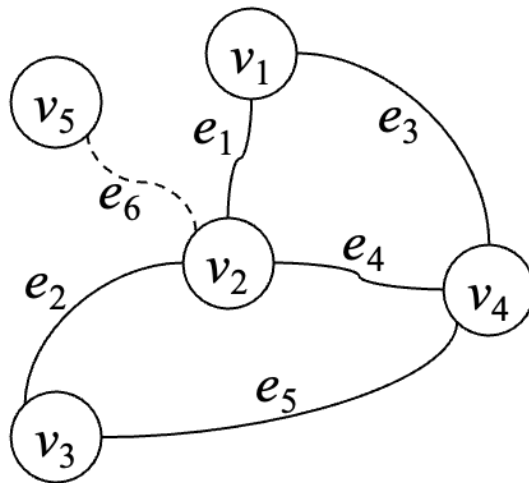
## 随堂小测



# 割点和割边

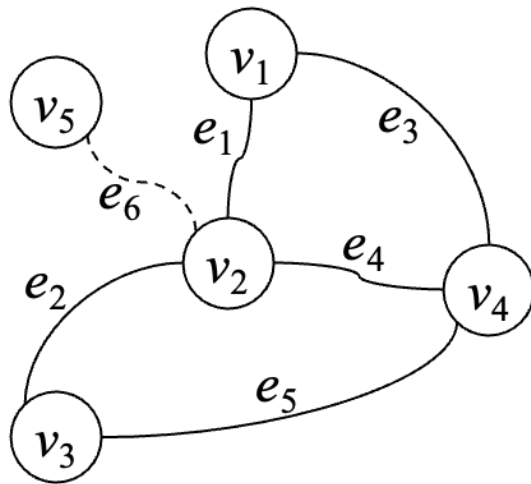
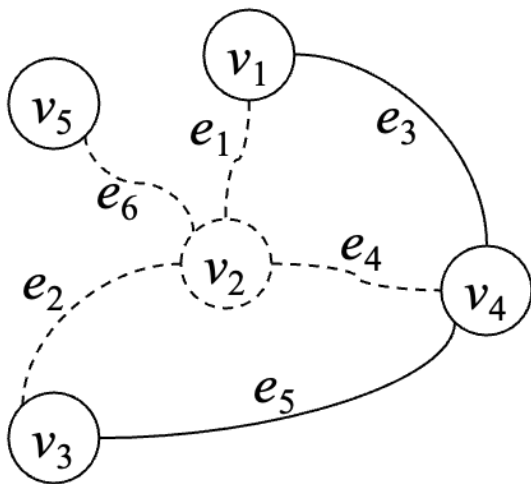
## ■ 割边 (桥)

- 广义定义: 对于图 $G = \langle V, E \rangle$ 和边 $e \in E$ ,  $G - e$ 的连通分支数量大于 $G$ 的



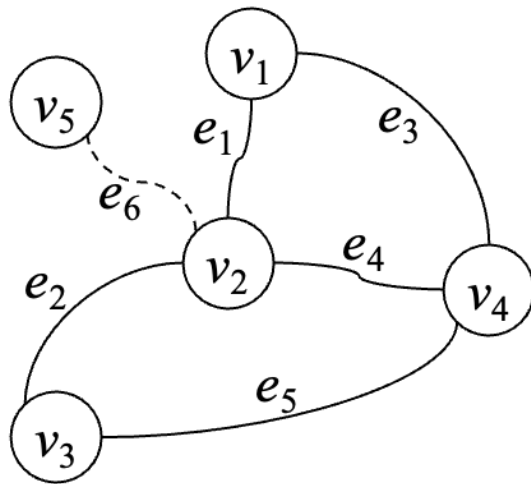
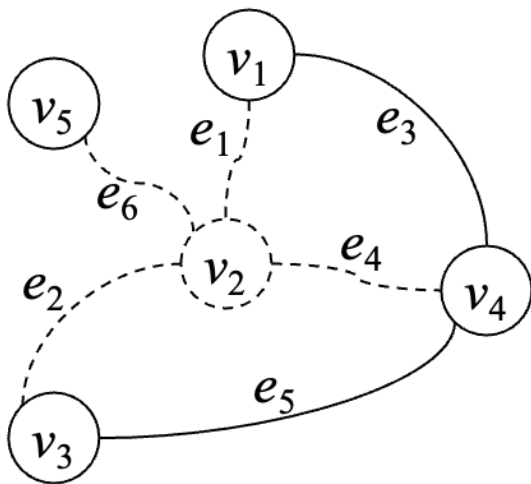
## 割点和割边

- 割点关联的边是割边吗？ 割边的端点是割点吗？



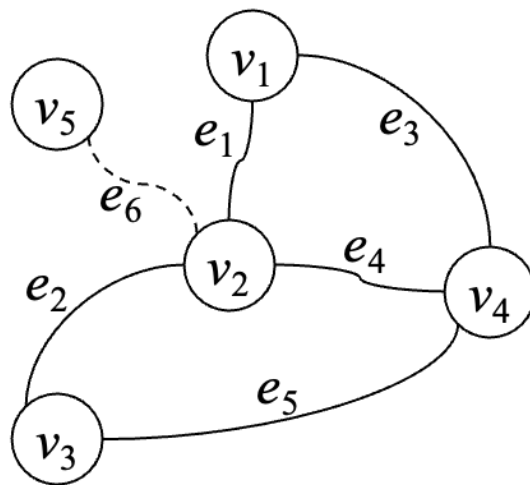
## 割点和割边

- 割点关联的边是割边吗？割边的端点是割点吗？
- 有割点的图一定有割边吗？有割边的图一定有割点吗？



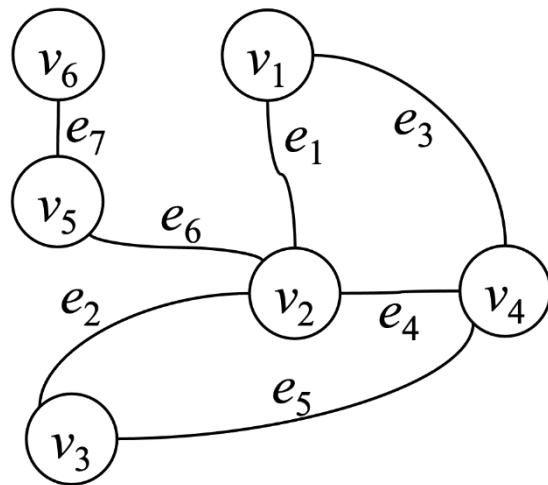
## 割点和割边

- 割点关联的边是割边吗？割边的端点是割点吗？
- 有割点的图一定有割边吗？有割边的图一定有割点吗？
- 割边的充要条件：对于连通图 $G = \langle V, E \rangle$ 和边 $e \in E$ ， $e$ 是 $G$ 的割边当且仅当存在 $V$ 的两个不相交的非空子集 $V_i$ 和 $V_j$ ，对于任意顶点 $u \in V_i$ 和 $w \in V_j$ ，每条 $u-w$ 路都经过 $e$ 。



## 割点和割边

- 如何判定一个顶点是否为割点？
- 如何找出图中所有割点？



## 割点和割边

- John Edward Hopcroft, 1939年出生于美国, 1986年获得图灵奖
- Robert Endre Tarjan, 1948年出生于美国, 1986年获得图灵奖



[https://en.wikipedia.org/wiki/John\\_Hopcroft](https://en.wikipedia.org/wiki/John_Hopcroft)  
[https://en.wikipedia.org/wiki/Robert\\_Tarjan](https://en.wikipedia.org/wiki/Robert_Tarjan)

# 割点和割边

## ■ 扩展DFS算法

---

### 算法 2.2: DFSCV

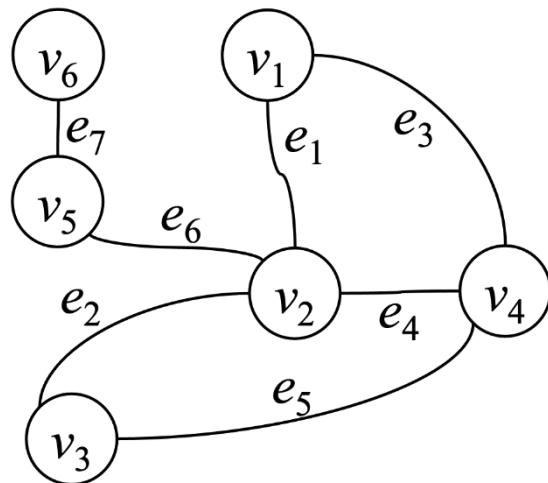
---

输入: 连通图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false,

```
1
2
3
4  $u.visited \leftarrow true;$ 
5 foreach  $(u, v) \in E$  do
6   if  $v.visited = false$  then
7     |
8     |
9     |  $DFSCV(G, v);$ 
10    |
11    |
12    |
13    |
14    |
15    |
16
```

---



# 割点和割边

## ■ 扩展DFS算法：记录每个顶点被访问的次序

---

### 算法 2.2: DFSCV

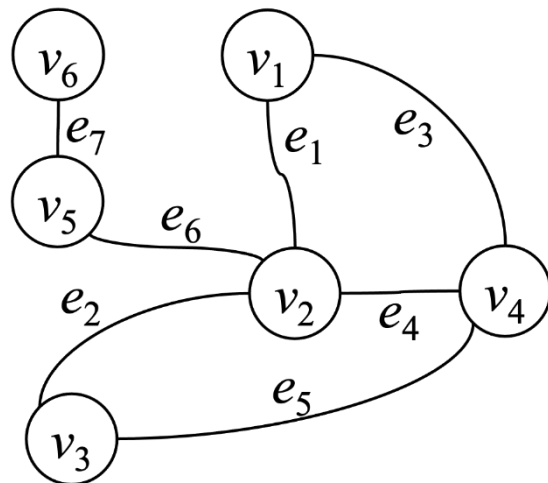
---

输入: 连通图  $G = \langle V, E \rangle$ , 顶点  $u$

初值: time 初值为 0;  $V$  中所有顶点的 visited 初值为 false,

```
1 time  $\leftarrow$  time + 1;  
2 u.d  $\leftarrow$  time;  
3  
4 u.visited  $\leftarrow$  true;  
5 foreach  $(u, v) \in E$  do  
6   if v.visited = false then  
7     |  
8     |  
9     | DFSCV( $G, v$ );  
10    |  
11    |  
12    |  
13    |  
14    |  
15    |  
16    |
```

---



# 割点和割边

- 扩展DFS算法：记录每个顶点被访问的次序，以及它的父顶点

---

## 算法 2.2: DFSCV

---

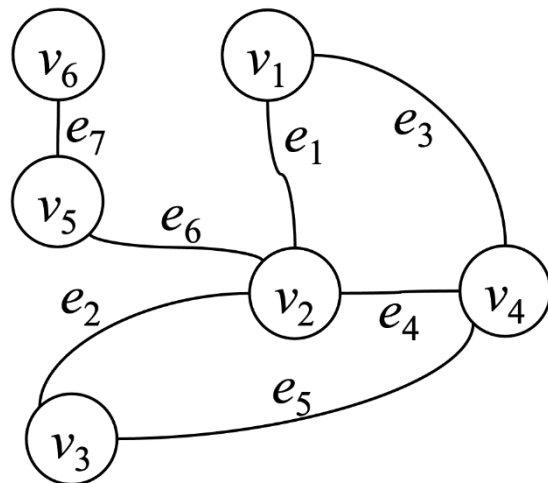
输入: 连通图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $time$  初值为 0;  $V$  中所有顶点的  $visited$  初值为  $false$ , parent 初值为  $null$ , children 初值为 0,

```
1  $time \leftarrow time + 1;$ 
2  $u.d \leftarrow time;$ 
3
4  $u.visited \leftarrow true;$ 
5 foreach  $(u, v) \in E$  do
6   if  $v.visited = false$  then
7      $v.parent \leftarrow u;$ 
8      $u.children \leftarrow u.children + 1;$ 
9     DFSCV( $G, v$ );
10
11
12
13
14
15
16
```

---

(根顶点: 首个被访问的顶点)

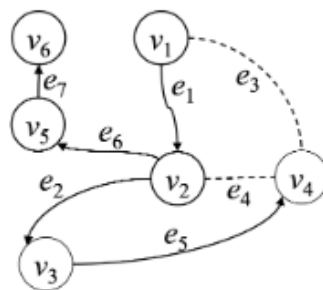


# 割点和割边

- 扩展DFS算法：记录每个顶点被访问的次序，以及它的父顶点
- DFS树
  - 树边：从父顶点访问子顶点经过的边
  - 后向边：其它边

	d	parent	children	low
$v_1$	1	null	1	1
$v_2$	2	$v_1$	2	1
$v_3$	3	$v_2$	1	1
$v_4$	4	$v_3$	0	1
$v_5$	5	$v_2$	1	5
$v_6$	6	$v_5$	0	6

(a)



(b)

图 2.8 DFSCV 算法运行结果示例

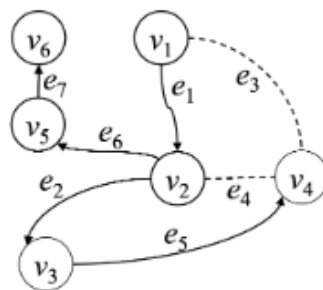
(a) 顶点的属性值；(b) 树边和后向边

## 割点和割边

- 扩展DFS算法：记录每个顶点被访问的次序，以及它的父顶点
- DFS树
  - 树边：从父顶点访问子顶点经过的边
  - 后向边：其它边
- 基于父顶点和子顶点可以分别递归定义**祖先顶点**和**后代顶点**。

	d	parent	children	low
$v_1$	1	null	1	1
$v_2$	2	$v_1$	2	1
$v_3$	3	$v_2$	1	1
$v_4$	4	$v_3$	0	1
$v_5$	5	$v_2$	1	5
$v_6$	6	$v_5$	0	6

(a)



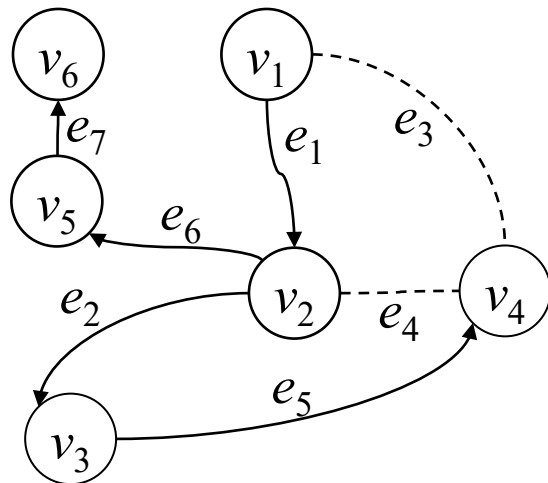
(b)

图 2.8 DFSCV 算法运行结果示例

(a) 顶点的属性值；(b) 树边和后向边

## 割点和割边

- 为什么后向边关联一对祖先-后代顶点？



# 割点和割边

- 为什么后向边关联一对祖先-后代顶点？
- 每个顶点有3种状态
  - 白：  $visited = false$  (DFS调用前)
  - 灰：  $visited = true$ , 且DFS调用未结束
  - 黑：  $visited = true$ , 且DFS调用已结束

---

## 算法 2.1: DFS

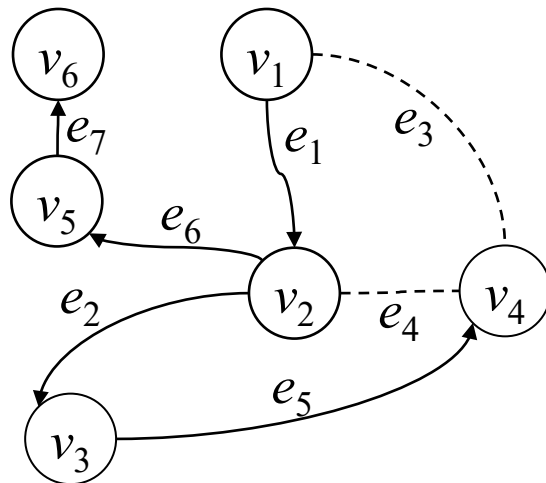
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的  $visited$  初值为  $false$

```
1  $u.visited \leftarrow true$ ;  
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v)$ ;
```

---



# 割点和割边

- 为什么后向边关联一对祖先-后代顶点？
- 每个顶点有3种状态
  - 白：visited = false (DFS调用前)
  - 灰：visited = true, 且DFS调用未结束
  - 黑：visited = true, 且DFS调用已结束
- 任意时刻，所有灰点组成一条祖先-后代路

---

## 算法 2.1: DFS

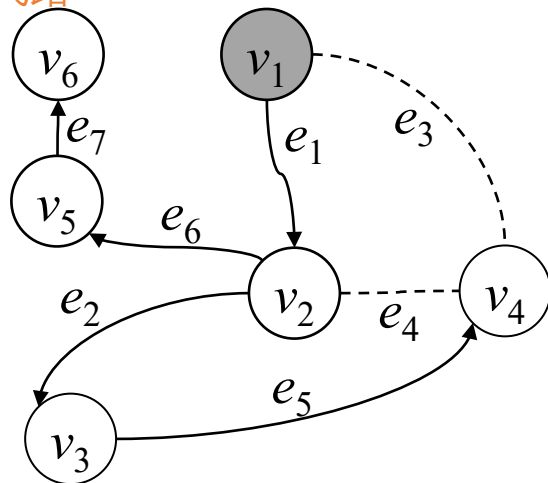
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false

```
1  $u.visited \leftarrow true$ ;  
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v)$ ;
```

---



# 割点和割边

- 为什么后向边关联一对祖先-后代顶点？
- 每个顶点有3种状态
  - 白：visited = false (DFS调用前)
  - 灰：visited = true, 且DFS调用未结束
  - 黑：visited = true, 且DFS调用已结束
- 任意时刻，所有灰点组成一条祖先-后代路

---

## 算法 2.1: DFS

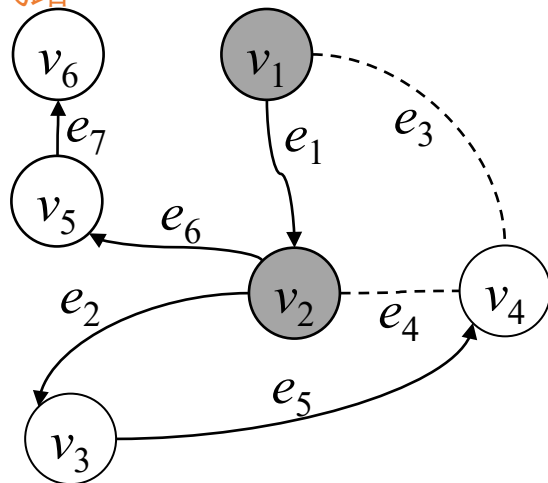
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 割点和割边

- 为什么后向边关联一对祖先-后代顶点？
- 每个顶点有3种状态
  - 白：visited = false (DFS调用前)
  - 灰：visited = true, 且DFS调用未结束
  - 黑：visited = true, 且DFS调用已结束
- 任意时刻，所有灰点组成一条祖先-后代路

---

## 算法 2.1: DFS

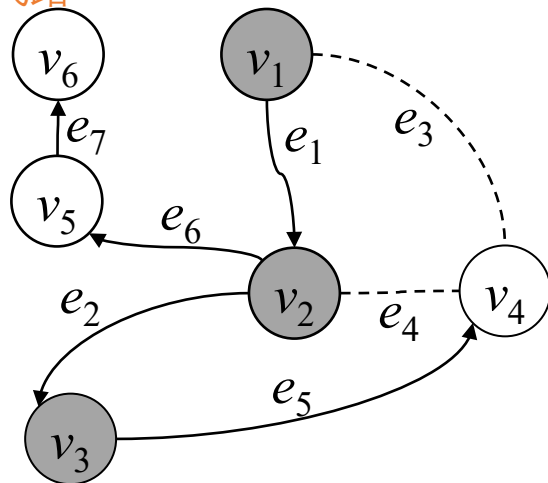
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 割点和割边

- 为什么后向边关联一对祖先-后代顶点？
- 每个顶点有3种状态
  - 白:  $visited = false$  (DFS调用前)
  - 灰:  $visited = true$ , 且DFS调用未结束
  - 黑:  $visited = true$ , 且DFS调用已结束
- 任意时刻, 所有灰点组成一条祖先-后代路

---

## 算法 2.1: DFS

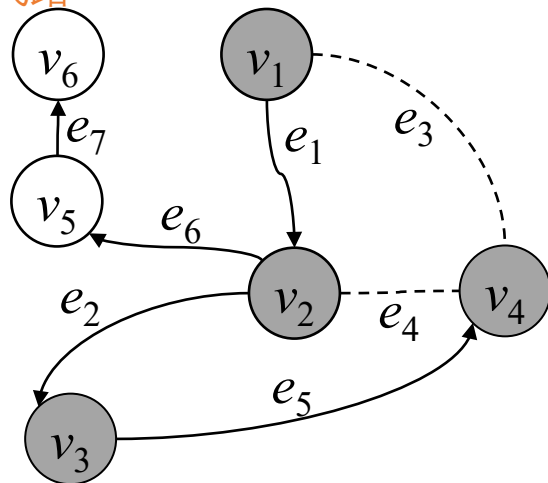
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的  $visited$  初值为  $false$

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 割点和割边

- 为什么后向边关联一对祖先-后代顶点？
- 每个顶点有3种状态
  - 白：visited = false (DFS调用前)
  - 灰：visited = true, 且DFS调用未结束
  - 黑：visited = true, 且DFS调用已结束
- 任意时刻，所有灰点组成一条祖先-后代路

---

## 算法 2.1: DFS

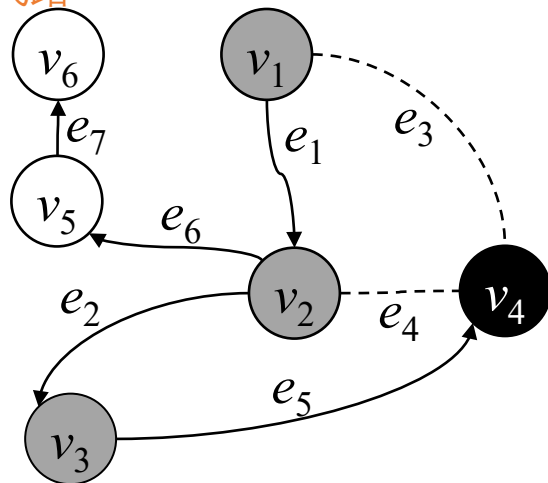
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



## 割点和割边

- 为什么后向边关联一对祖先-后代顶点？
- 每个顶点有3种状态
  - 白：visited = false (DFS调用前)
  - 灰：visited = true, 且DFS调用未结束
  - 黑：visited = true, 且DFS调用已结束
- 任意时刻，所有灰点组成一条祖先-后代路

---

### 算法 2.1: DFS

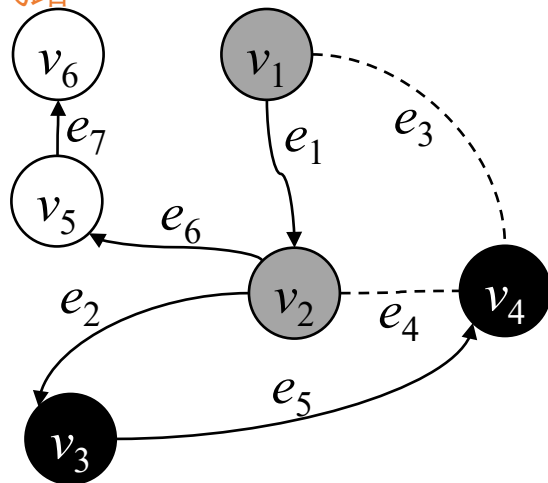
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 割点和割边

- 为什么后向边关联一对祖先-后代顶点？
- 每个顶点有3种状态
  - 白:  $visited = false$  (DFS调用前)
  - 灰:  $visited = true$ , 且DFS调用未结束
  - 黑:  $visited = true$ , 且DFS调用已结束
- 任意时刻, 所有灰点组成一条祖先-后代路

---

## 算法 2.1: DFS

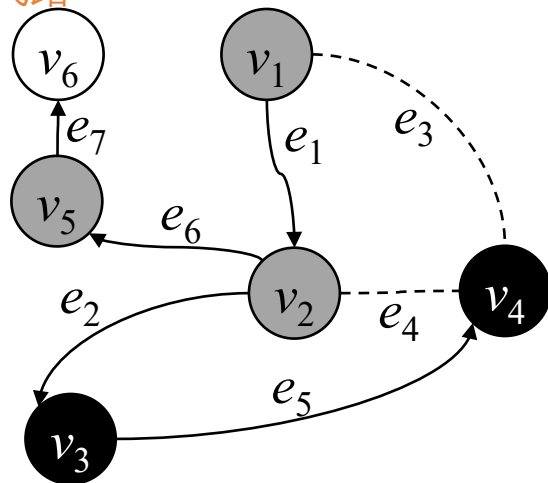
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的  $visited$  初值为  $false$

```
1  $u.visited \leftarrow true$ ;  
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v)$ ;
```

---



# 割点和割边

- 为什么后向边关联一对祖先-后代顶点？
- 每个顶点有3种状态
  - 白:  $visited = false$  (DFS调用前)
  - 灰:  $visited = true$ , 且DFS调用未结束
  - 黑:  $visited = true$ , 且DFS调用已结束
- 任意时刻, 所有灰点组成一条祖先-后代路

---

## 算法 2.1: DFS

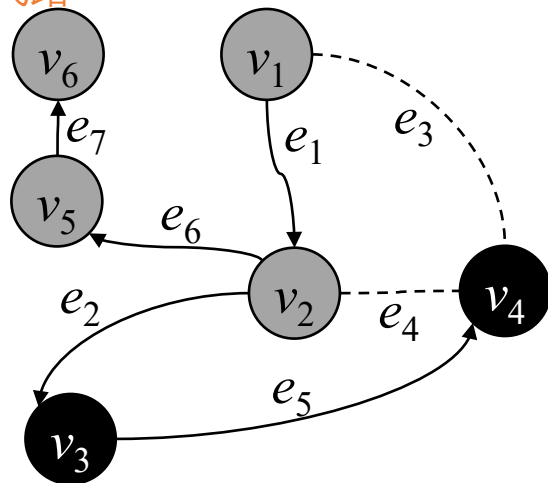
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的  $visited$  初值为  $false$

```
1  $u.visited \leftarrow true$ ;  
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v)$ ;
```

---



# 割点和割边

- 为什么后向边关联一对祖先-后代顶点？
- 每个顶点有3种状态
  - 白：visited = false (DFS调用前)
  - 灰：visited = true, 且DFS调用未结束
  - 黑：visited = true, 且DFS调用已结束
- 任意时刻，所有灰点组成一条祖先-后代路

---

## 算法 2.1: DFS

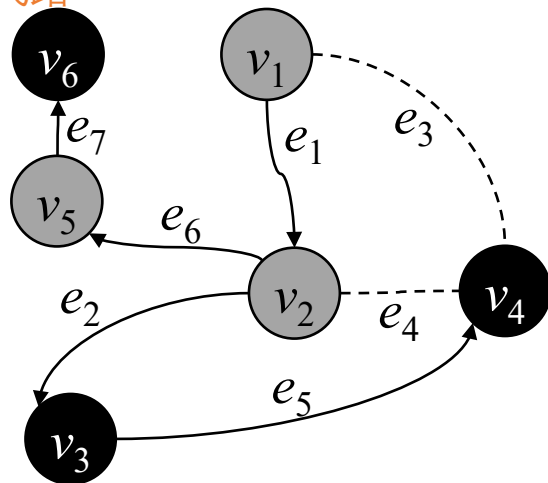
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 割点和割边

- 为什么后向边关联一对祖先-后代顶点？
- 每个顶点有3种状态
  - 白：  $visited = false$  (DFS调用前)
  - 灰：  $visited = true$ ，且DFS调用未结束
  - 黑：  $visited = true$ ，且DFS调用已结束
- 任意时刻，所有灰点组成一条祖先-后代路

---

## 算法 2.1: DFS

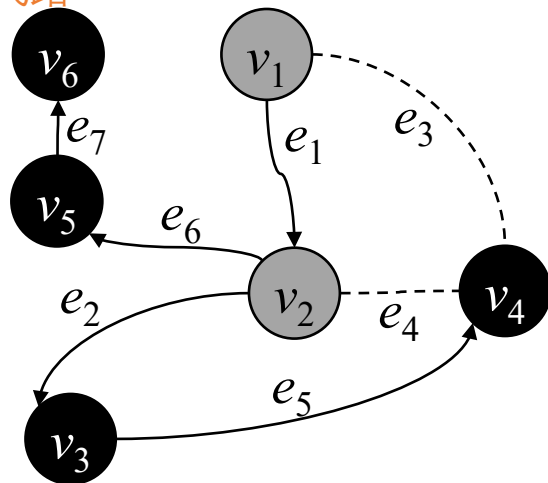
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的  $visited$  初值为  $false$

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 割点和割边

- 为什么后向边关联一对祖先-后代顶点？
- 每个顶点有3种状态
  - 白：visited = false (DFS调用前)
  - 灰：visited = true, 且DFS调用未结束
  - 黑：visited = true, 且DFS调用已结束
- 任意时刻，所有灰点组成一条祖先-后代路

---

## 算法 2.1: DFS

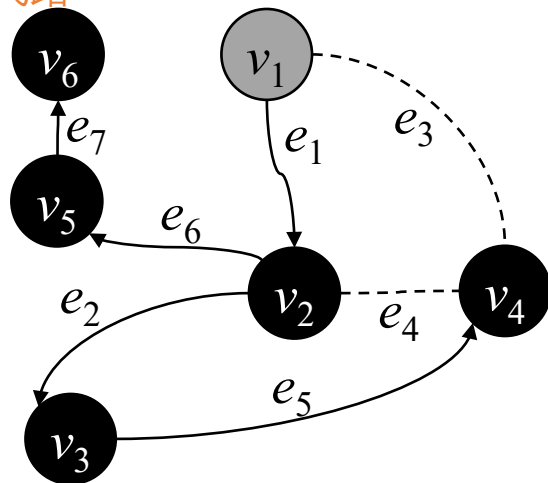
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false

```
1  $u.visited \leftarrow true;$   
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v);$ 
```

---



# 割点和割边

- 为什么后向边关联一对祖先-后代顶点？
- 每个顶点有3种状态
  - 白：  $visited = false$  (DFS调用前)
  - 灰：  $visited = true$ ，且DFS调用未结束
  - 黑：  $visited = true$ ，且DFS调用已结束
- 任意时刻，所有灰点组成一条祖先-后代路

---

## 算法 2.1: DFS

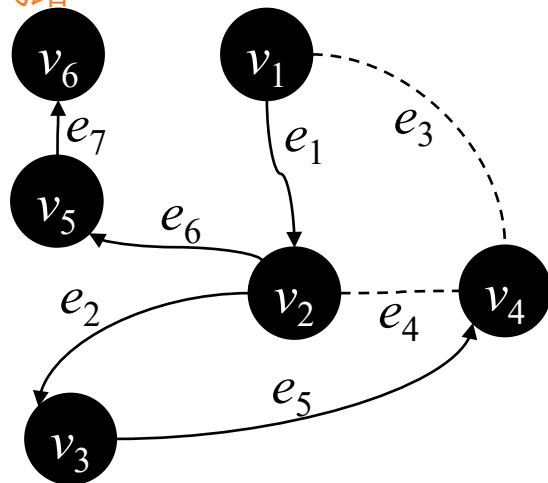
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的  $visited$  初值为  $false$

```
1  $u.visited \leftarrow true$ ;  
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v)$ ;
```

---



# 割点和割边

- 为什么后向边关联一对祖先-后代顶点？
- 每个顶点有3种状态
  - 白：visited = false (DFS调用前)
  - 灰：visited = true, 且DFS调用未结束
  - 黑：visited = true, 且DFS调用已结束
- 任意时刻，所有灰点组成一条祖先-后代路
- 后向边的首次访问必从灰到灰 (祖先)

---

## 算法 2.1: DFS

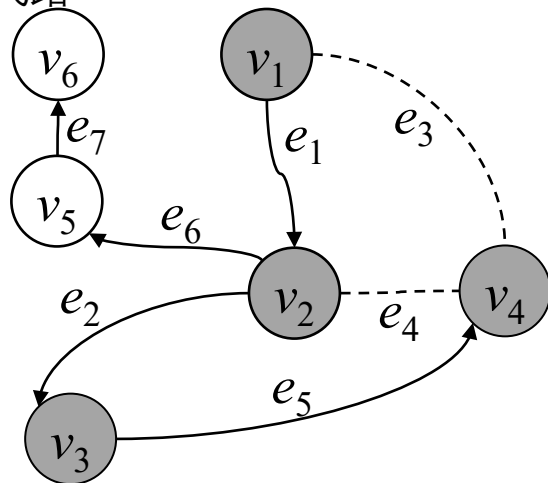
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false

```
1  $u.visited \leftarrow true$ ;  
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v)$ ;
```

---



# 割点和割边

- 为什么后向边关联一对祖先-后代顶点？
- 每个顶点有3种状态
  - 白：visited = false (DFS调用前)
  - 灰：visited = true, 且DFS调用未结束
  - 黑：visited = true, 且DFS调用已结束
- 任意时刻，所有灰点组成一条祖先-后代路
- 后向边的首次访问必从灰到灰 (祖先)
  - 为什么不能到白？

---

## 算法 2.1: DFS

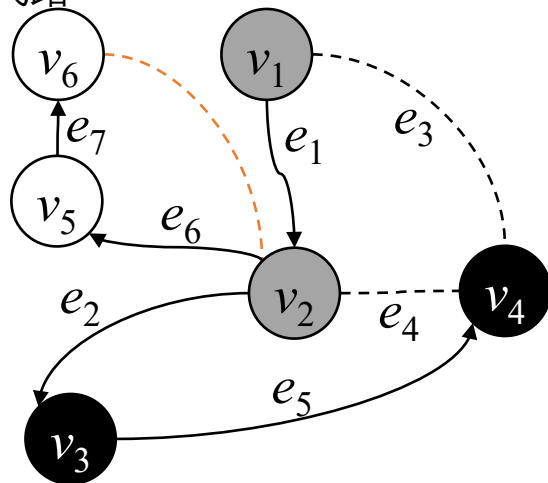
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false

```
1  $u.visited \leftarrow true$ ;  
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v)$ ;
```

---



# 割点和割边

- 为什么后向边关联一对祖先-后代顶点？
- 每个顶点有3种状态
  - 白：visited = false (DFS调用前)
  - 灰：visited = true, 且DFS调用未结束
  - 黑：visited = true, 且DFS调用已结束
- 任意时刻，所有灰点组成一条祖先-后代路
- 后向边的首次访问必从灰到灰 (祖先)
  - 为什么不能到黑？

---

## 算法 2.1: DFS

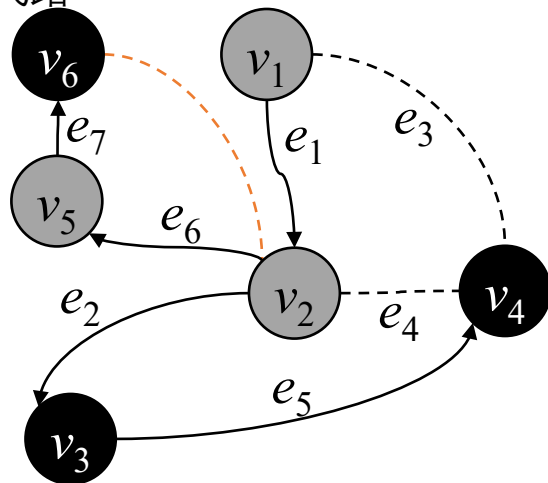
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false

```
1  $u.visited \leftarrow true$ ;  
2 foreach  $(u, v) \in E$  do  
3   | if  $v.visited = false$  then  
4   |   |  $DFS(G, v)$ ;
```

---



# 割点和割边

## ■ 扩展DFS算法：判定非根顶点是否为割点

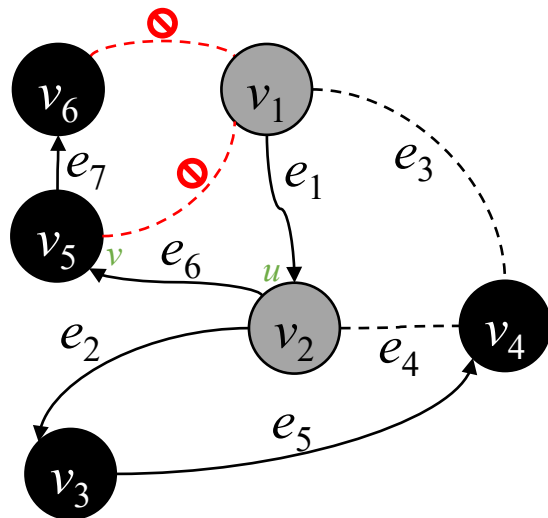
### 算法 2.2: DFSCV

输入: 连通图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $time$  初值为 0;  $V$  中所有顶点的  $visited$  初值为 false,  $parent$  初值为 null,  $children$  初值为 0,

```
1  $time \leftarrow time + 1;$ 
2  $u.d \leftarrow time;$ 
3
4  $u.visited \leftarrow true;$ 
5 foreach  $(u, v) \in E$  do
6   if  $v.visited = false$  then
7      $v.parent \leftarrow u;$ 
8      $u.children \leftarrow u.children + 1;$ 
9     DFSCV( $G, v$ );
10
11
12
13
14
15
16
```

若顶点  $u$  不是根顶点, 则  $u$  是割点当且仅当  $u$  有子顶点  $v$  满足: 不存在这样一条后向边, 其一个端点是  $v$  或其后代顶点, 另一个端点是  $u$  的祖先顶点。



# 割点和割边

## ■ 扩展DFS算法：判定非根顶点是否为割点

### 算法 2.2: DFSCV

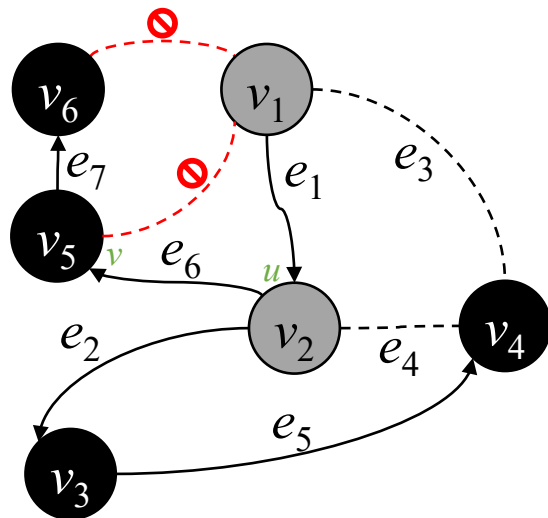
输入: 连通图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $time$  初值为 0;  $V$  中所有顶点的  $visited$  初值为 false,  $parent$  初值为 null,  $children$  初值为 0,

```
1  $time \leftarrow time + 1;$   
2  $u.d \leftarrow time;$   
3  
4  $u.visited \leftarrow true;$   
5 foreach  $(u, v) \in E$  do  
6   if  $v.visited = false$  then  
7      $v.parent \leftarrow u;$   
8      $u.children \leftarrow u.children + 1;$   
9     DFSCV( $G, v$ );  
10  
11  
12  
13  
14  
15  
16
```

$u$  割开了谁和谁?

若顶点  $u$  不是根顶点, 则  $u$  是割点当且仅当  $u$  有子顶点  $v$  满足: 不存在这样一条后向边, 其一个端点是  $v$  或其后代顶点, 另一个端点是  $u$  的祖先顶点。



# 割点和割边

## ■ 扩展DFS算法：判定非根顶点是否为割点

### 算法 2.2: DFSCV

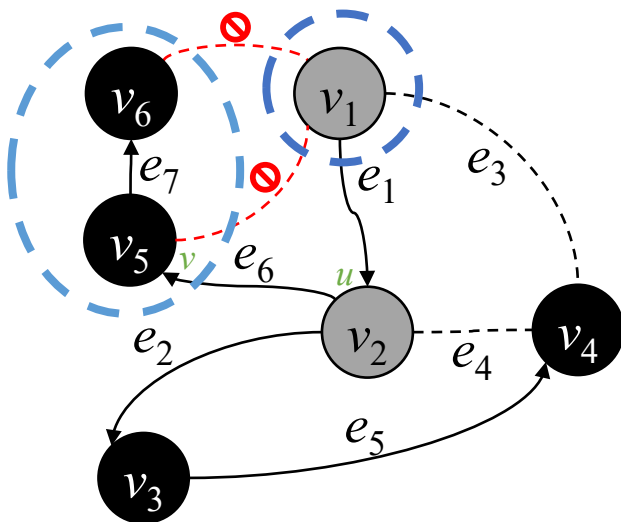
输入: 连通图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $time$  初值为 0;  $V$  中所有顶点的  $visited$  初值为 false,  $parent$  初值为 null,  $children$  初值为 0,

```
1  $time \leftarrow time + 1;$   
2  $u.d \leftarrow time;$   
3  
4  $u.visited \leftarrow true;$   
5 foreach  $(u, v) \in E$  do  
6   if  $v.visited = false$  then  
7      $v.parent \leftarrow u;$   
8      $u.children \leftarrow u.children + 1;$   
9     DFSCV( $G, v$ );  
10  
11  
12  
13  
14  
15  
16
```

$u$  割开了谁和谁?

若顶点  $u$  不是根顶点, 则  $u$  是割点当且仅当  $u$  有子顶点  $v$  满足: 不存在这样一条后向边, 其一个端点是  $v$  或其后代顶点, 另一个端点是  $u$  的祖先顶点。



# 割点和割边

## ■ 扩展DFS算法：判定非根顶点是否为割点

### 算法 2.2: DFSCV

输入: 连通图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $time$  初值为 0;  $V$  中所有顶点的  $visited$  初值为  $false$ ,  $parent$  初值为  $null$ ,  $children$  初值为 0,  $isCutVertex$  初值为  $false$

1  $time \leftarrow time + 1$ ;

2  $u.d \leftarrow time$ ;

3

4  $u.visited \leftarrow true$ ;

5 **foreach**  $(u, v) \in E$  **do**

6     **if**  $v.visited = false$  **then**

7          $v.parent \leftarrow u$ ;

8          $u.children \leftarrow u.children + 1$ ;

9         DFSCV( $G, v$ );

10

11

12

13     **else if**  $u.parent \neq null$  且  $v.low \geq u.d$  **then**

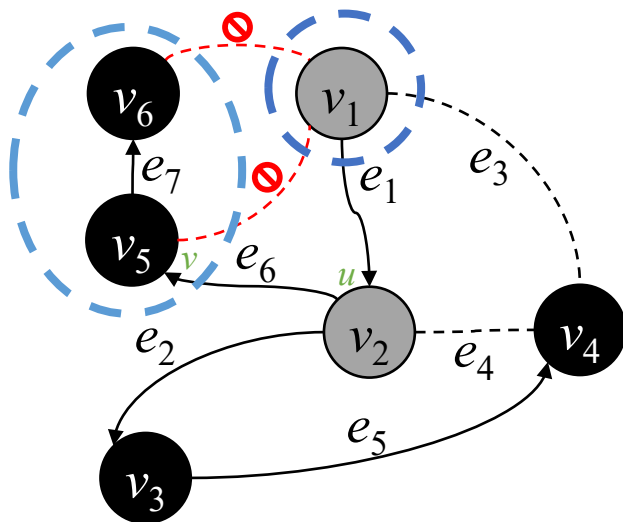
14          $u.isCutVertex \leftarrow true$ ;

15

16

$low$ : 该顶点及其后代顶点通过后向边关联的邻点的最小  $d$  值

若顶点  $u$  不是根顶点, 则  $u$  是割点当且仅当  $u$  有子顶点  $v$  满足: 不存在这样一条后向边, 其一个端点是  $v$  或其后代顶点, 另一个端点是  $u$  的祖先顶点。



# 割点和割边

## ■ 扩展DFS算法：判定非根顶点是否为割点

### 算法 2.2: DFSCV

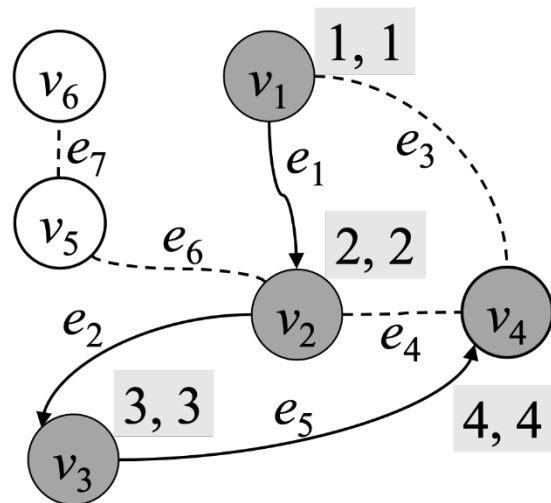
输入: 连通图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $time$  初值为 0;  $V$  中所有顶点的  $visited$  初值为  $false$ ,  $parent$  初值为  $null$ ,  $children$  初值为 0,  $isCutVertex$  初值为  $false$

```
1  $time \leftarrow time + 1$ ;  
2  $u.d \leftarrow time$ ;  
3  $u.low \leftarrow u.d$ ;  
4  $u.visited \leftarrow true$ ;  
5 foreach  $(u, v) \in E$  do  
6   if  $v.visited = false$  then  
7      $v.parent \leftarrow u$ ;  
8      $u.children \leftarrow u.children + 1$ ;  
9     DFSCV( $G, v$ );  
10  
11  
12  
13   else if  $u.parent \neq null$  且  $v.low \geq u.d$  then  
14      $u.isCutVertex \leftarrow true$ ;  
15  
16
```

**low**: 该顶点及其后代顶点通过后向边关联的邻点的最小d值

若顶点  $u$  不是根顶点, 则  $u$  是割点当且仅当  $u$  有子顶点  $v$  满足: 不存在这样一条后向边, 其一个端点是  $v$  或其后代顶点, 另一个端点是  $u$  的祖先顶点。



# 割点和割边

## ■ 扩展DFS算法：判定非根顶点是否为割点

### 算法 2.2: DFSCV

输入: 连通图  $G = \langle V, E \rangle$ , 顶点  $u$

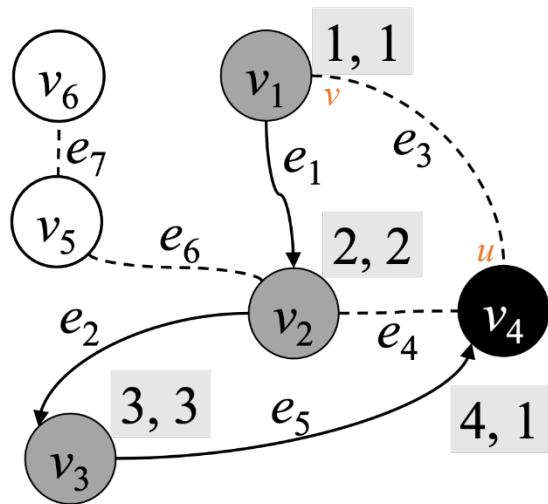
初值:  $time$  初值为 0;  $V$  中所有顶点的  $visited$  初值为  $false$ ,  $parent$  初值为  $null$ ,  $children$  初值为 0,  $isCutVertex$  初值为  $false$

```
1  $time \leftarrow time + 1$ ;  
2  $u.d \leftarrow time$ ;  
3  $u.low \leftarrow u.d$ ;  
4  $u.visited \leftarrow true$ ;  
5 foreach  $(u, v) \in E$  do  
6   if  $v.visited = false$  then  
7      $v.parent \leftarrow u$ ;  
8      $u.children \leftarrow u.children + 1$ ;  
9     DFSCV( $G, v$ );  
10  
11  
12  
13   else if  $u.parent \neq null$  且  $v.low \geq u.d$  then  
14      $u.isCutVertex \leftarrow true$ ;  
15   else if  $v \neq u.parent$  then  
16      $u.low \leftarrow \min\{u.low, v.d\}$ ;
```

low: 该顶点及其后代顶点通过后向边关联的邻点的最小d值

后向边( $u, v$ )

若顶点  $u$  不是根顶点, 则  $u$  是割点当且仅当  $u$  有子顶点  $v$  满足: 不存在这样一条后向边, 其一个端点是  $v$  或其后代顶点, 另一个端点是  $u$  的祖先顶点。





# 割点和割边

## ■ 扩展DFS算法：判定根顶点是否为割点

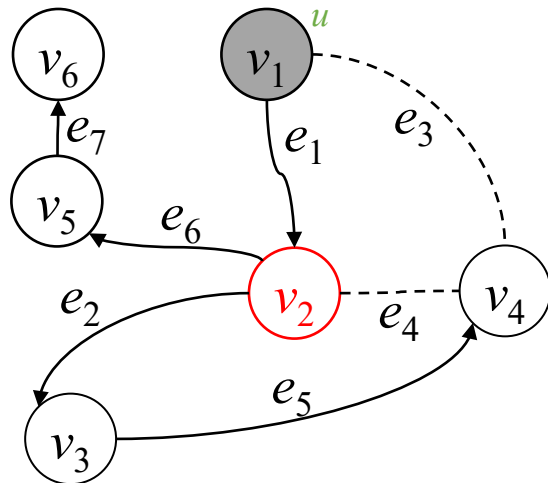
### 算法 2.2: DFSCV

输入: 连通图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $time$  初值为 0;  $V$  中所有顶点的  $visited$  初值为  $false$ ,  $parent$  初值为  $null$ ,  $children$  初值为 0,  $isCutVertex$  初值为  $false$

```
1  $time \leftarrow time + 1$ ;  
2  $u.d \leftarrow time$ ;  
3  $u.low \leftarrow u.d$ ;  
4  $u.visited \leftarrow true$ ;  
5 foreach  $(u, v) \in E$  do  
6   if  $v.visited = false$  then  
7      $v.parent \leftarrow u$ ;  
8      $u.children \leftarrow u.children + 1$ ;  
9      $DFSCV(G, v)$ ;  
10     $u.low \leftarrow \min\{u.low, v.low\}$ ;  
11    if  $u.parent = null$  且  $u.children \geq 2$  then  
12       $u.isCutVertex \leftarrow true$ ;  
13    else if  $u.parent \neq null$  且  $v.low \geq u.d$  then  
14       $u.isCutVertex \leftarrow true$ ;  
15    else if  $v \neq u.parent$  then  
16       $u.low \leftarrow \min\{u.low, v.d\}$ ;
```

若顶点  $u$  是根顶点, 则  $u$  是割点当且仅当  $u$  有至少 2 个子顶点。



# 割点和割边

## ■ 扩展DFS算法：判定根顶点是否为割点

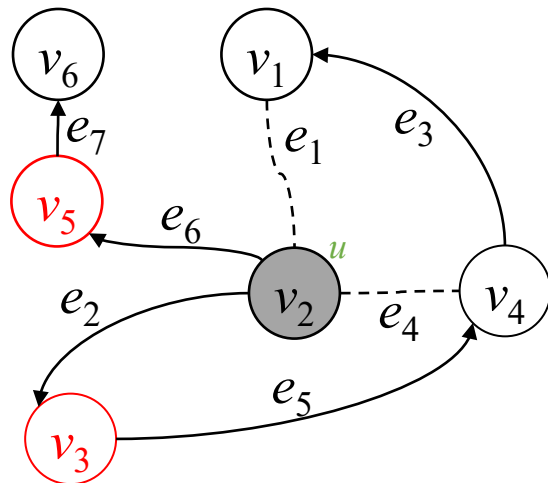
### 算法 2.2: DFSCV

输入: 连通图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $time$  初值为 0;  $V$  中所有顶点的  $visited$  初值为  $false$ ,  $parent$  初值为  $null$ ,  $children$  初值为 0,  $isCutVertex$  初值为  $false$

```
1  $time \leftarrow time + 1$ ;  
2  $u.d \leftarrow time$ ;  
3  $u.low \leftarrow u.d$ ;  
4  $u.visited \leftarrow true$ ;  
5 foreach  $(u, v) \in E$  do  
6   if  $v.visited = false$  then  
7      $v.parent \leftarrow u$ ;  
8      $u.children \leftarrow u.children + 1$ ;  
9      $DFSCV(G, v)$ ;  
10     $u.low \leftarrow \min\{u.low, v.low\}$ ;  
11    if  $u.parent = null$  且  $u.children \geq 2$  then  
12       $u.isCutVertex \leftarrow true$ ;  
13    else if  $u.parent \neq null$  且  $v.low \geq u.d$  then  
14       $u.isCutVertex \leftarrow true$ ;  
15    else if  $v \neq u.parent$  then  
16       $u.low \leftarrow \min\{u.low, v.d\}$ ;
```

若顶点  $u$  是根顶点, 则  $u$  是割点当且仅当  $u$  有至少 2 个子顶点。



# 割点和割边

## ■ 扩展DFS算法：判定根顶点是否为割点

### 算法 2.2: DFSCV

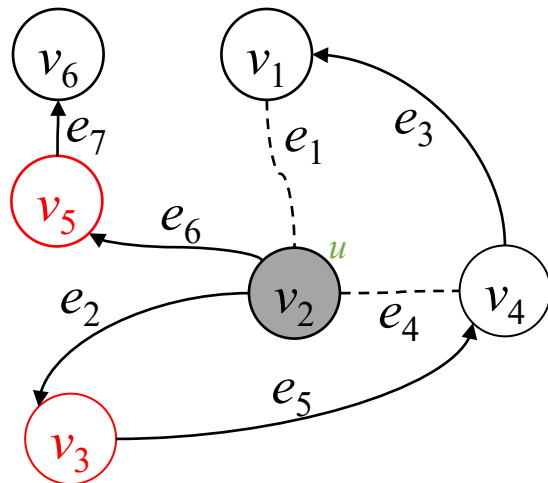
输入: 连通图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $time$  初值为 0;  $V$  中所有顶点的  $visited$  初值为  $false$ ,  $parent$  初值为  $null$ ,  $children$  初值为 0,  $isCutVertex$  初值为  $false$

```
1  $time \leftarrow time + 1;$ 
2  $u.d \leftarrow time;$ 
3  $u.low \leftarrow u.d;$ 
4  $u.visited \leftarrow true;$ 
5 foreach  $(u, v) \in E$  do
6   if  $v.visited = false$  then
7      $v.parent \leftarrow u;$ 
8      $u.children \leftarrow u.children + 1;$ 
9     DFSCV( $G, v$ );
10     $u.low \leftarrow \min\{u.low, v.low\};$ 
11    if  $u.parent = null$  且  $u.children \geq 2$  then
12       $u.isCutVertex \leftarrow true;$ 
13    else if  $u.parent \neq null$  且  $v.low \geq u.d$  then
14       $u.isCutVertex \leftarrow true;$ 
15  else if  $v \neq u.parent$  then
16     $u.low \leftarrow \min\{u.low, v.d\};$ 
```

$u$  割开了谁和谁?

若顶点  $u$  是根顶点, 则  $u$  是割点当且仅当  $u$  有至少 2 个子顶点。



# 割点和割边

## ■ 扩展DFS算法：判定根顶点是否为割点

### 算法 2.2: DFSCV

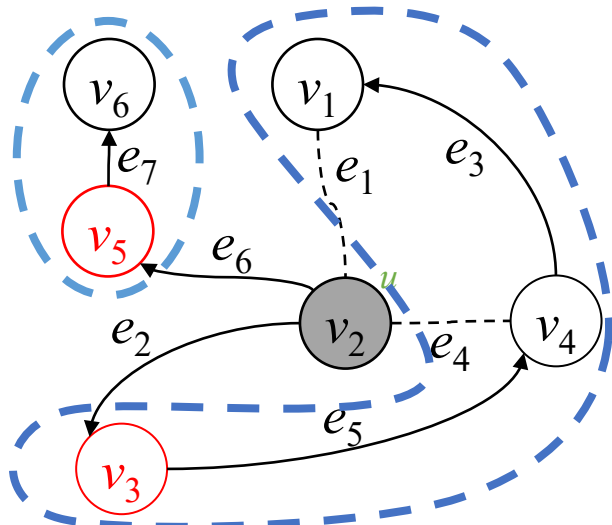
输入: 连通图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $time$  初值为 0;  $V$  中所有顶点的  $visited$  初值为  $false$ ,  $parent$  初值为  $null$ ,  $children$  初值为 0,  $isCutVertex$  初值为  $false$

```
1  $time \leftarrow time + 1$ ;  
2  $u.d \leftarrow time$ ;  
3  $u.low \leftarrow u.d$ ;  
4  $u.visited \leftarrow true$ ;  
5 foreach  $(u, v) \in E$  do  
6   if  $v.visited = false$  then  
7      $v.parent \leftarrow u$ ;  
8      $u.children \leftarrow u.children + 1$ ;  
9     DFSCV( $G, v$ );  
10     $u.low \leftarrow \min\{u.low, v.low\}$ ;  
11    if  $u.parent = null$  且  $u.children \geq 2$  then  
12       $u.isCutVertex \leftarrow true$ ;  
13    else if  $u.parent \neq null$  且  $v.low \geq u.d$  then  
14       $u.isCutVertex \leftarrow true$ ;  
15  else if  $v \neq u.parent$  then  
16     $u.low \leftarrow \min\{u.low, v.d\}$ ;
```

$u$  割开了谁和谁?

若顶点  $u$  是根顶点, 则  $u$  是割点当且仅当  $u$  有至少 2 个子顶点。



# 割点和割边

## ■ 时间复杂度: $O(n + m)$

---

### 算法 2.2: DFSCV

---

**输入:** 连通图  $G = \langle V, E \rangle$ , 顶点  $u$

**初值:**  $time$  初值为 0;  $V$  中所有顶点的  $visited$  初值为 false,  $parent$  初值为 null,  $children$  初值为 0,  $isCutVertex$  初值为 false

```
1  $time \leftarrow time + 1;$ 
2  $u.d \leftarrow time;$ 
3  $u.low \leftarrow u.d;$ 
4  $u.visited \leftarrow true;$ 
5 foreach  $(u, v) \in E$  do
6   if  $v.visited = false$  then
7      $v.parent \leftarrow u;$ 
8      $u.children \leftarrow u.children + 1;$ 
9     DFSCV( $G, v$ );
10     $u.low \leftarrow \min\{u.low, v.low\};$ 
11    if  $u.parent = null$  且  $u.children \geq 2$  then
12       $u.isCutVertex \leftarrow true;$ 
13    else if  $u.parent \neq null$  且  $v.low \geq u.d$  then
14       $u.isCutVertex \leftarrow true;$ 
15  else if  $v \neq u.parent$  then
16     $u.low \leftarrow \min\{u.low, v.d\};$ 
```

---

# 本次课的主要内容

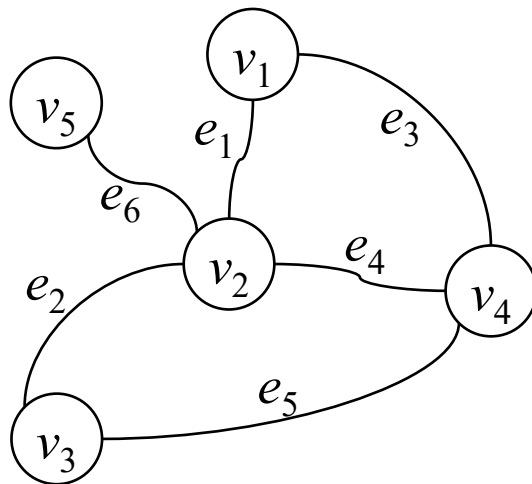
2.1 连通和DFS

2.2 割点和割边

2.3 距离和BFS

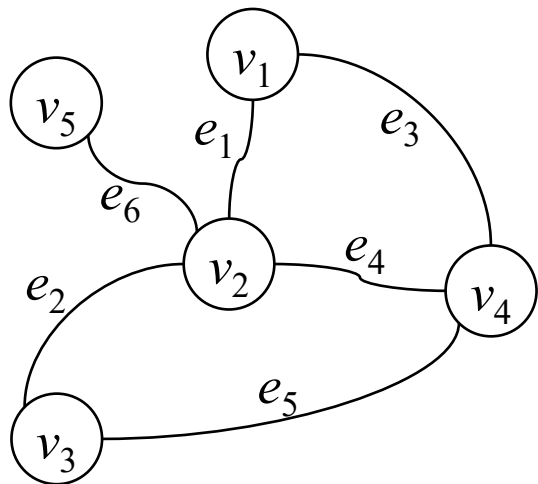
## 距离和BFS

- 顶点 $u$ 和 $v$ 间的**最短路**：长度最小的 $u$ - $v$ 路



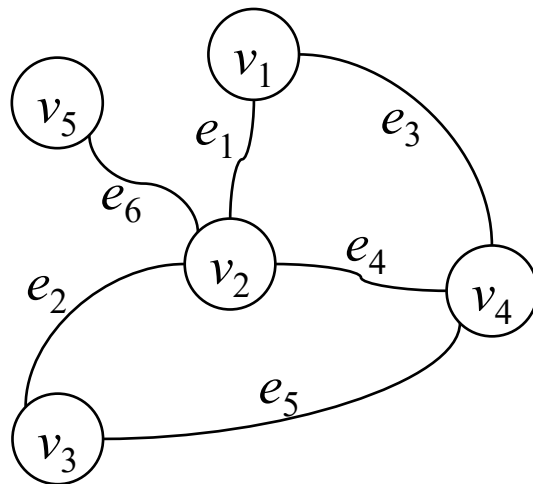
## 距离和BFS

- 顶点 $u$ 和 $v$ 间的最短路：长度最小的 $u$ - $v$ 路
- 两个顶点间一定有最短路吗？若有，则唯一吗？



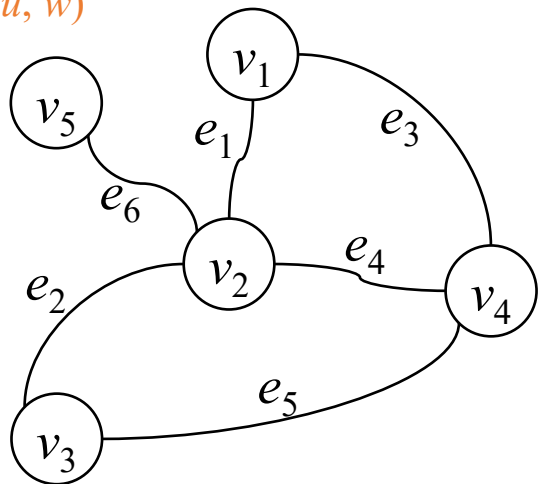
# 距离和BFS

- 顶点 $u$ 和 $v$ 间的最短路：长度最小的 $u$ - $v$ 路
- 两个顶点间一定有最短路吗？若有，则唯一吗？
- 顶点 $u$ 和 $v$ 间的**距离**： $u$ 和 $v$ 间的最短路的长度，记作 $dist(u, v)$ 
  - 不连通： $dist = \infty$



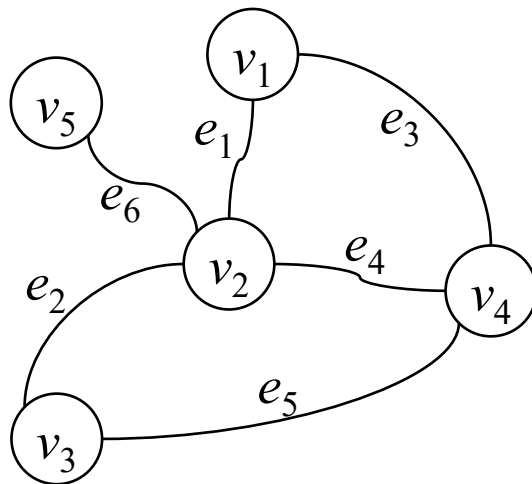
# 距离和BFS

- 顶点 $u$ 和 $v$ 间的最短路：长度最小的 $u$ - $v$ 路
- 两个顶点间一定有最短路吗？若有，则唯一吗？
- 顶点 $u$ 和 $v$ 间的距离： $u$ 和 $v$ 间的最短路的长度，记作 $dist(u, v)$ 
  - 不连通： $dist = \infty$
- 证明：对于连通图 $G = \langle V, E \rangle$ ，距离函数 $dist$ 满足三角不等式，即 $\forall u, v, w \in V, dist(u, v) + dist(v, w) \geq dist(u, w)$



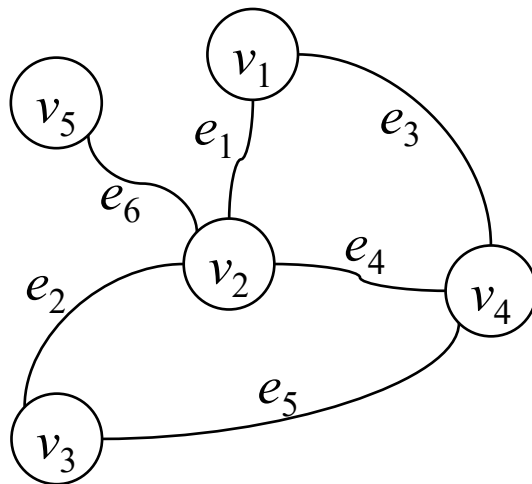
## 距离和BFS

- 顶点 $v$ 的**离心率**：  $v$ 和所有顶点间的距离的最大值，记作 $ecc(v)$



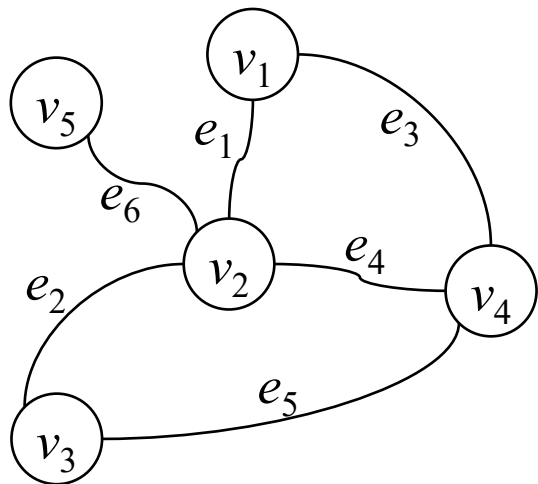
## 距离和BFS

- 顶点 $v$ 的离心率： $v$ 和所有顶点间的距离的最大值，记作 $ecc(v)$
- 证明：对于连通图 $G = \langle V, E \rangle$ 和两个相邻顶点 $u, v \in V$ ， $|ecc(u) - ecc(v)| \leq 1$ 。



## 距离和BFS

- 顶点 $v$ 的离心率： $v$ 和所有顶点间的距离的最大值，记作 $ecc(v)$
- 证明：对于连通图 $G = \langle V, E \rangle$ 和两个相邻顶点 $u, v \in V$ ， $|ecc(u) - ecc(v)| \leq 1$ 。
- 证明：对于连通图 $G = \langle V, E \rangle$ 和两个顶点 $u, v \in V$ ， $|ecc(u) - ecc(v)| \leq dist(u, v)$ 。

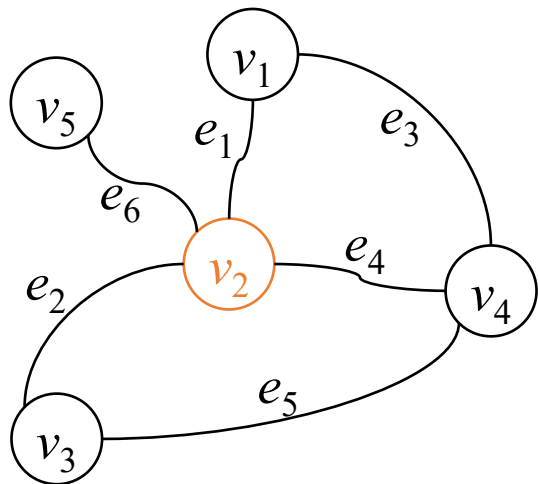


## 距离和BFS

- 顶点 $v$ 的离心率： $v$ 和所有顶点间的距离的最大值，记作 $ecc(v)$
- 证明：对于连通图 $G = \langle V, E \rangle$ 和两个相邻顶点 $u, v \in V$ ， $|ecc(u) - ecc(v)| \leq 1$ 。
- 证明：对于连通图 $G = \langle V, E \rangle$ 和两个顶点 $u, v \in V$ ， $|ecc(u) - ecc(v)| \leq dist(u, v)$ 。

### 图 $G$ 的

- **中心点**：离心率最小的顶点
- **半径**：中心点的离心率，记作 $rad(G)$
- **中心**：所有中心点的集合

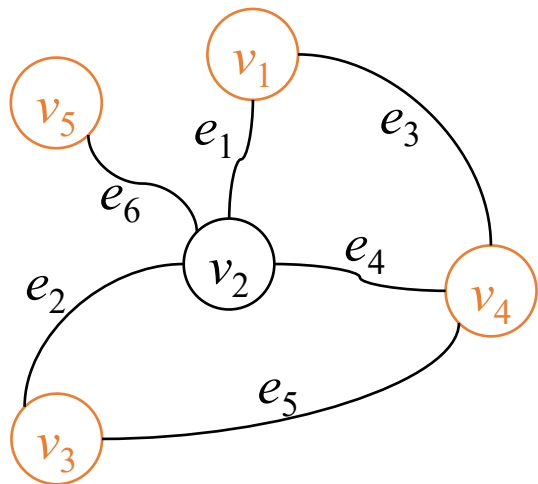


## 距离和BFS

- 顶点 $v$ 的离心率：  $v$ 和所有顶点间的距离的最大值，记作 $ecc(v)$
- 证明： 对于连通图 $G = \langle V, E \rangle$ 和两个相邻顶点 $u, v \in V$ ， $|ecc(u) - ecc(v)| \leq 1$ 。
- 证明： 对于连通图 $G = \langle V, E \rangle$ 和两个顶点 $u, v \in V$ ， $|ecc(u) - ecc(v)| \leq dist(u, v)$ 。

图 $G$ 的

- 中心点： 离心率最小的顶点
- 半径： 中心点的离心率，记作 $rad(G)$
- 中心： 所有中心点的集合
- **边缘点**： 离心率最大的顶点
- **直径**： 边缘点的离心率，记作 $diam(G)$

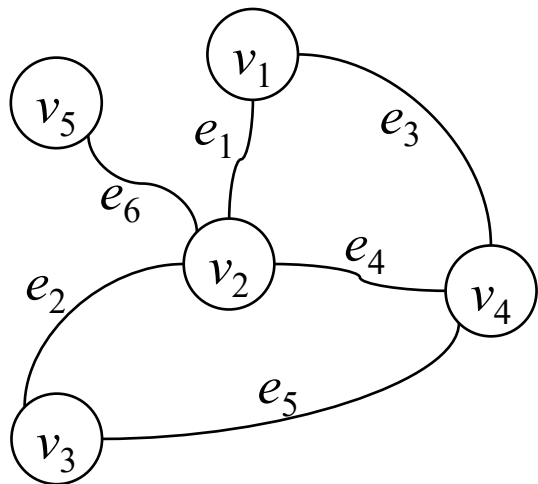


## 距离和BFS

- 顶点 $v$ 的离心率：  $v$ 和所有顶点间的距离的最大值，记作 $ecc(v)$
- 证明： 对于连通图 $G = \langle V, E \rangle$ 和两个相邻顶点 $u, v \in V$ ， $|ecc(u) - ecc(v)| \leq 1$ 。
- 证明： 对于连通图 $G = \langle V, E \rangle$ 和两个顶点 $u, v \in V$ ， $|ecc(u) - ecc(v)| \leq dist(u, v)$ 。

图 $G$ 的

- 中心点： 离心率最小的顶点
- 半径： 中心点的离心率，记作 $rad(G)$
- 中心： 所有中心点的集合
- 边缘点： 离心率最大的顶点
- 直径： 边缘点的离心率，记作 $diam(G)$
- 连通图的直径是图中哪条路的长度？



## 距离和BFS

- 对于连通图 $G$ ,  $rad(G) \leq diam(G) \leq 2 \cdot rad(G)$ 。

## 距离和BFS

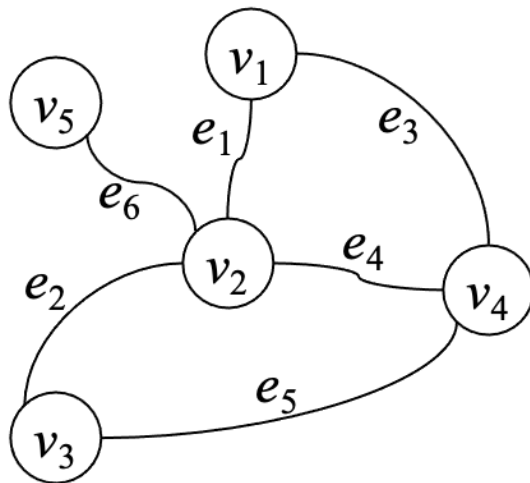
- 对于连通图 $G$ ,  $rad(G) \leq diam(G) \leq 2 \cdot rad(G)$ 。

取中心点 $w$ :

$$\begin{aligned} diam(G) &= dist(u, v) \\ &\leq dist(u, w) + dist(w, v) \\ &\leq ecc(w) + ecc(w) \\ &= 2 \cdot ecc(w) \\ &= 2 \cdot rad(G). \end{aligned}$$

## 距离和BFS

- 如何计算两个顶点间的距离?
- 如何计算一个顶点和图中所有顶点间的距离?



# 距离和BFS

## ■ 宽度优先搜索（BFS）算法

- 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
- 遍历顶点的顺序是优先向图的“浅处”访问，即倾向于贴近出发点

---

### 算法 2.3: BFS

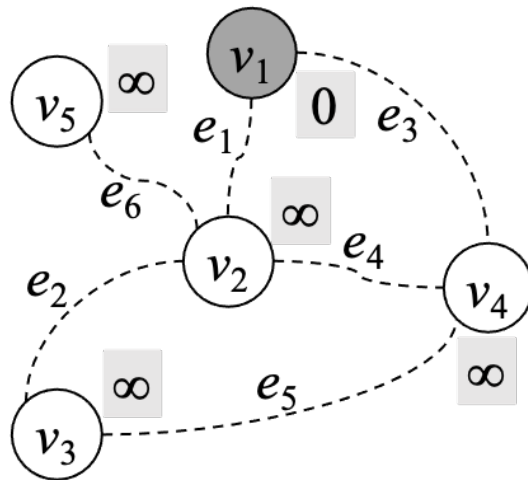
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false,  $d$  初值为  $\infty$ ;  $Q$  初值为空队列

```
1  $u.visited \leftarrow true$ ;
2  $u.d \leftarrow 0$ ;
3 入队列 ( $Q, u$ );
4 while  $Q$  非空 do
5    $v \leftarrow$  出队列 ( $Q$ );
6   foreach  $(v, w) \in E$  do
7     if  $w.visited = false$  then
8        $w.visited \leftarrow true$ ;
9        $w.d \leftarrow v.d + 1$ ;
10      入队列 ( $Q, w$ );
```

---



# 距离和BFS

## ■ 宽度优先搜索（BFS）算法

- 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
- 遍历顶点的顺序是优先向图的“浅处”访问，即倾向于贴近出发点

---

### 算法 2.3: BFS

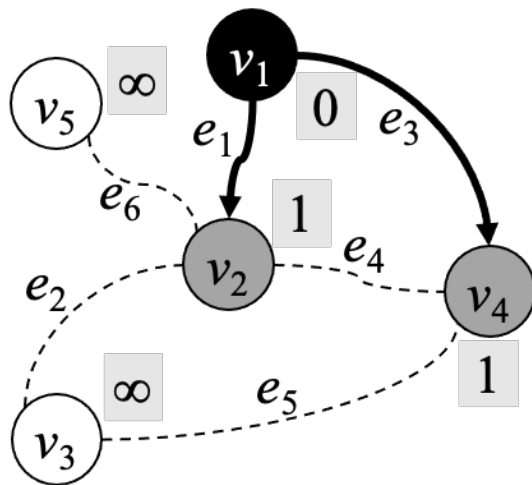
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false,  $d$  初值为  $\infty$ ;  $Q$  初值为空队列

```
1  $u.visited \leftarrow true$ ;
2  $u.d \leftarrow 0$ ;
3 入队列 ( $Q, u$ );
4 while  $Q$  非空 do
5    $v \leftarrow$  出队列 ( $Q$ );
6   foreach  $(v, w) \in E$  do
7     if  $w.visited = false$  then
8        $w.visited \leftarrow true$ ;
9        $w.d \leftarrow v.d + 1$ ;
10      入队列 ( $Q, w$ );
```

---



# 距离和BFS

## ■ 宽度优先搜索（BFS）算法

- 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
- 遍历顶点的顺序是优先向图的“浅处”访问，即倾向于贴近出发点

---

### 算法 2.3: BFS

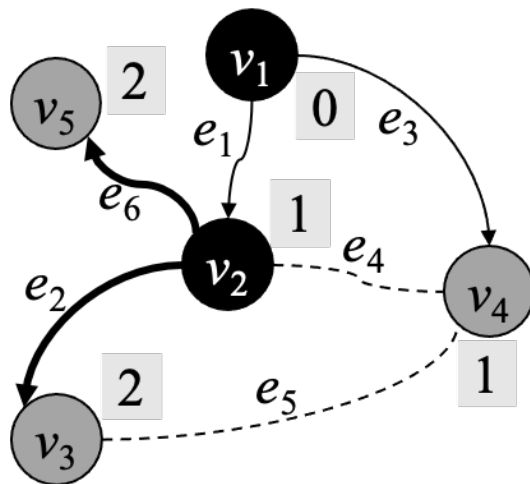
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false,  $d$  初值为  $\infty$ ;  $Q$  初值为空队列

```
1  $u.visited \leftarrow true$ ;  
2  $u.d \leftarrow 0$ ;  
3 入队列 ( $Q, u$ );  
4 while  $Q$  非空 do  
5    $v \leftarrow$  出队列 ( $Q$ );  
6   foreach  $(v, w) \in E$  do  
7     if  $w.visited = false$  then  
8        $w.visited \leftarrow true$ ;  
9        $w.d \leftarrow v.d + 1$ ;  
10      入队列 ( $Q, w$ );
```

---



# 距离和BFS

## ■ 宽度优先搜索（BFS）算法

- 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
- 遍历顶点的顺序是优先向图的“浅处”访问，即倾向于贴近出发点

---

### 算法 2.3: BFS

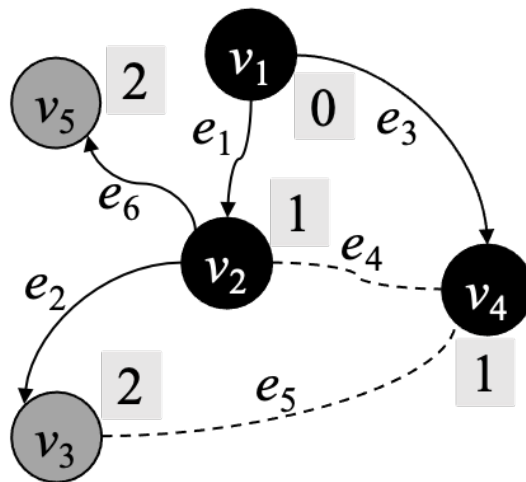
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false,  $d$  初值为  $\infty$ ;  $Q$  初值为空队列

```
1  $u.visited \leftarrow true$ ;
2  $u.d \leftarrow 0$ ;
3 入队列 ( $Q, u$ );
4 while  $Q$  非空 do
5    $v \leftarrow$  出队列 ( $Q$ );
6   foreach  $(v, w) \in E$  do
7     if  $w.visited = false$  then
8        $w.visited \leftarrow true$ ;
9        $w.d \leftarrow v.d + 1$ ;
10      入队列 ( $Q, w$ );
```

---



# 距离和BFS

## ■ 宽度优先搜索（BFS）算法

- 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
- 遍历顶点的顺序是优先向图的“浅处”访问，即倾向于贴近出发点

---

### 算法 2.3: BFS

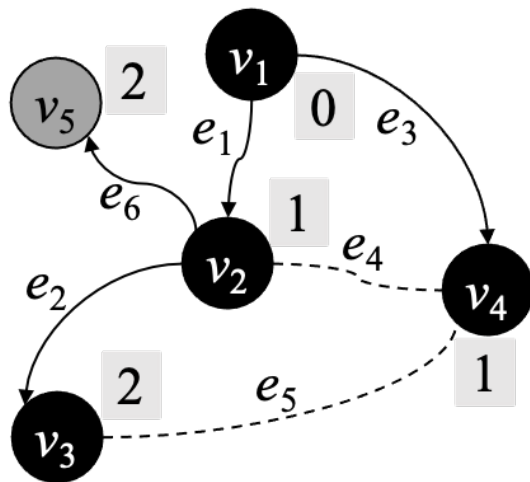
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false,  $d$  初值为  $\infty$ ;  $Q$  初值为空队列

```
1  $u.visited \leftarrow true$ ;
2  $u.d \leftarrow 0$ ;
3 入队列 ( $Q, u$ );
4 while  $Q$  非空 do
5    $v \leftarrow$  出队列 ( $Q$ );
6   foreach  $(v, w) \in E$  do
7     if  $w.visited = false$  then
8        $w.visited \leftarrow true$ ;
9        $w.d \leftarrow v.d + 1$ ;
10      入队列 ( $Q, w$ );
```

---



# 距离和BFS

## ■ 宽度优先搜索（BFS）算法

- 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
- 遍历顶点的顺序是优先向图的“浅处”访问，即倾向于贴近出发点

---

### 算法 2.3: BFS

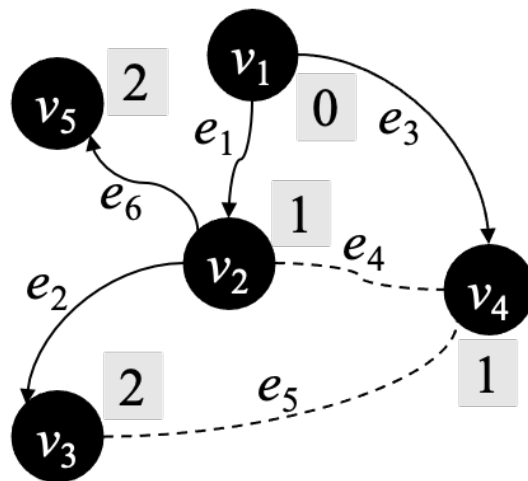
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 visited 初值为 false,  $d$  初值为  $\infty$ ;  $Q$  初值为空队列

```
1  $u.visited \leftarrow true$ ;
2  $u.d \leftarrow 0$ ;
3 入队列 ( $Q, u$ );
4 while  $Q$  非空 do
5    $v \leftarrow$  出队列 ( $Q$ );
6   foreach  $(v, w) \in E$  do
7     if  $w.visited = false$  then
8        $w.visited \leftarrow true$ ;
9        $w.d \leftarrow v.d + 1$ ;
10      入队列 ( $Q, w$ );
```

---



# 距离和BFS

- 从顶点 $u$ 出发运行BFS算法，为什么恰能访问与 $u$ 连通的所有顶点？

---

## 算法 2.3: BFS

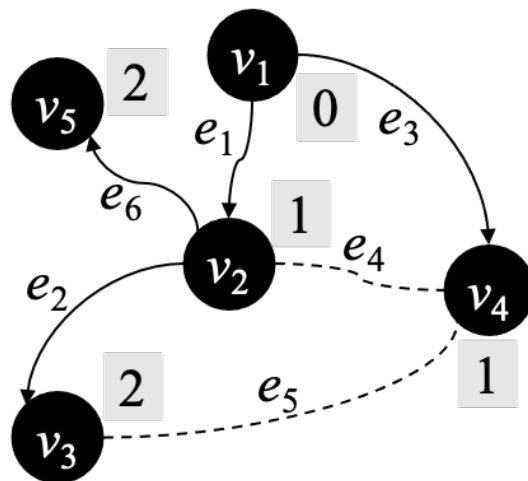
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的 `visited` 初值为 `false`, `d` 初值为  $\infty$ ;  $Q$  初值为空队列

```
1  $u.visited \leftarrow true$ ;  
2  $u.d \leftarrow 0$ ;  
3 入队列 ( $Q, u$ );  
4 while  $Q$  非空 do  
5    $v \leftarrow$  出队列 ( $Q$ );  
6   foreach  $(v, w) \in E$  do  
7     if  $w.visited = false$  then  
8        $w.visited \leftarrow true$ ;  
9        $w.d \leftarrow v.d + 1$ ;  
10      入队列 ( $Q, w$ );
```

---



# 距离和BFS

- 从顶点 $u$ 出发运行BFS算法，为什么恰能访问与 $u$ 连通的所有顶点？
- 算法运行结束时，每个顶点的 $d$ 属性值是其和出发点间的距离的上界。

---

## 算法 2.3: BFS

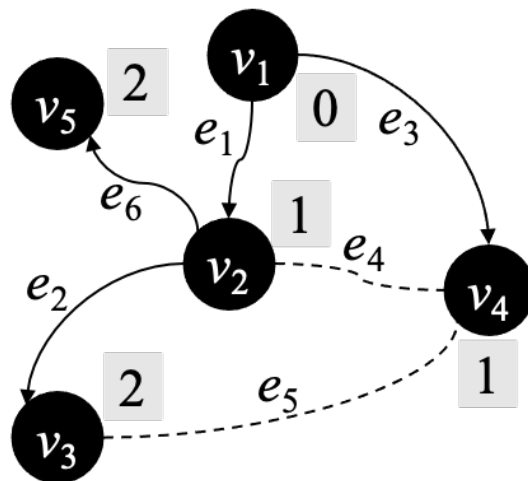
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的  $visited$  初值为  $false$ ,  $d$  初值为  $\infty$ ;  $Q$  初值为空队列

```
1  $u.visited \leftarrow true$ ;  
2  $u.d \leftarrow 0$ ;  
3 入队列 ( $Q, u$ );  
4 while  $Q$  非空 do  
5    $v \leftarrow$  出队列 ( $Q$ );  
6   foreach  $(v, w) \in E$  do  
7     if  $w.visited = false$  then  
8        $w.visited \leftarrow true$ ;  
9        $w.d \leftarrow v.d + 1$ ;  
10      入队列 ( $Q, w$ );
```

---



# 距离和BFS

- 从顶点 $u$ 出发运行BFS算法，为什么恰能访问与 $u$ 连通的所有顶点？
- 算法运行结束时，每个顶点的 $d$ 属性值是其和出发点间的距离的上界。
- 算法运行过程中，队列中所有顶点的 $d$ 属性有至多2种取值。

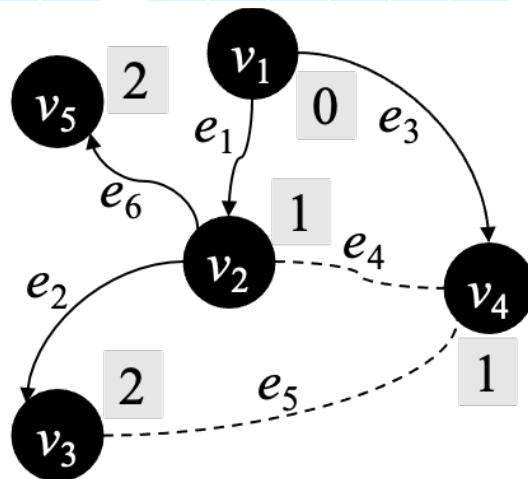


## 算法 2.3: BFS

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的  $visited$  初值为  $false$ ,  $d$  初值为  $\infty$ ;  $Q$  初值为空队列

```
1  $u.visited \leftarrow true$ ;  
2  $u.d \leftarrow 0$ ;  
3 入队列 ( $Q, u$ );  
4 while  $Q$  非空 do  
5    $v \leftarrow$  出队列 ( $Q$ );  
6   foreach  $(v, w) \in E$  do  
7     if  $w.visited = false$  then  
8        $w.visited \leftarrow true$ ;  
9        $w.d \leftarrow v.d + 1$ ;  
10      入队列 ( $Q, w$ );
```



# 距离和BFS

- 从顶点 $u$ 出发运行BFS算法，为什么恰能访问与 $u$ 连通的所有顶点？
- 算法运行结束时，每个顶点的 $d$ 属性值是其和出发点间的距离的上界。
- 算法运行过程中，队列中所有顶点的 $d$ 属性有至多2种取值。
- 按BFS算法的访问顺序，所有顶点的 $d$ 属性值组成非减序列。

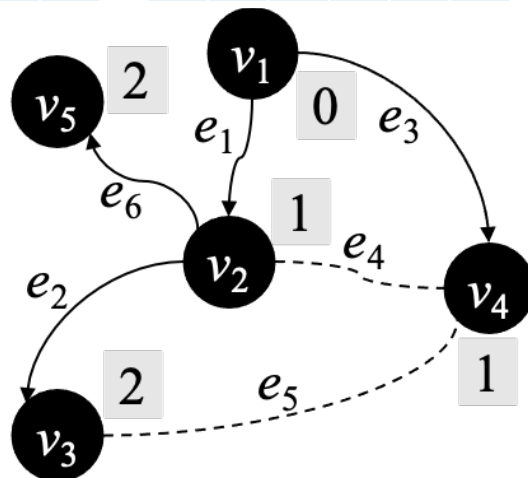


## 算法 2.3: BFS

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的  $visited$  初值为  $false$ ,  $d$  初值为  $\infty$ ;  $Q$  初值为空队列

```
1  $u.visited \leftarrow true$ ;  
2  $u.d \leftarrow 0$ ;  
3 入队列  $(Q, u)$ ;  
4 while  $Q$  非空 do  
5    $v \leftarrow$  出队列  $(Q)$ ;  
6   foreach  $(v, w) \in E$  do  
7     if  $w.visited = false$  then  
8        $w.visited \leftarrow true$ ;  
9        $w.d \leftarrow v.d + 1$ ;  
10      入队列  $(Q, w)$ ;
```



# 距离和BFS

- 从顶点 $u$ 出发运行BFS算法，为什么恰能访问与 $u$ 连通的所有顶点？
- 算法运行结束时，每个顶点的 $d$ 属性值是其和出发点间的距离的上界。
- 算法运行过程中，队列中所有顶点的 $d$ 属性有至多2种取值。
- 按BFS算法的访问顺序，所有顶点的 $d$ 属性值组成非减序列。
- 算法运行结束时，每个顶点的 $d$ 属性值为其和出发点间的距离。

---

## 算法 2.3: BFS

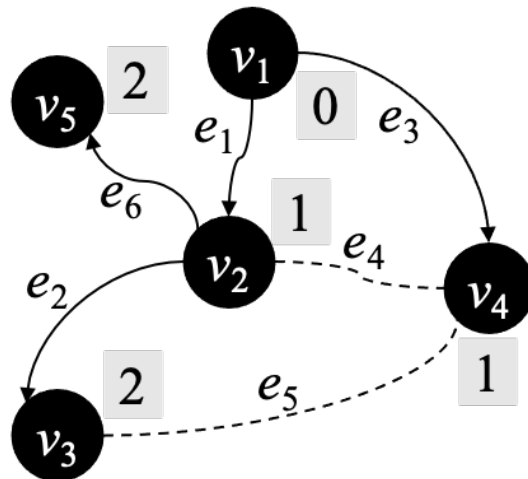
---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

初值:  $V$  中所有顶点的  $visited$  初值为  $false$ ,  $d$  初值为  $\infty$ ;  $Q$  初值为空队列

```
1  $u.visited \leftarrow true$ ;  
2  $u.d \leftarrow 0$ ;  
3 入队列 ( $Q, u$ );  
4 while  $Q$  非空 do  
5    $v \leftarrow$  出队列 ( $Q$ );  
6   foreach  $(v, w) \in E$  do  
7     if  $w.visited = false$  then  
8        $w.visited \leftarrow true$ ;  
9        $w.d \leftarrow v.d + 1$ ;  
10      入队列 ( $Q, w$ );
```

---



# 距离和BFS

- 从顶点 $u$ 出发运行BFS算法，为什么恰能访问与 $u$ 连通的所有顶点？
- 算法运行结束时，每个顶点的 $d$ 属性值是其和出发点间的距离的上界。
- 算法运行过程中，队列中所有顶点的 $d$ 属性有至多2种取值。
- 按BFS算法的访问顺序，所有顶点的 $d$ 属性值组成非减序列。
- 算法运行结束时，每个顶点的 $d$ 属性值为其和出发点间的距离。

---

## 算法 2.3: BFS

---

输入: 图  $G = \langle V, E \rangle$ , 顶点  $u$

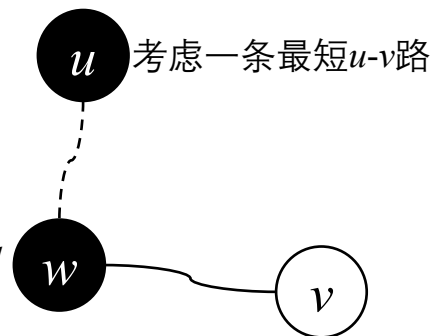
初值:  $V$  中所有顶点的  $visited$  初值为  $false$ ,  $d$  初值为  $\infty$ ;  $Q$  初值为空队列

```
1  $u.visited \leftarrow true$ ;  
2  $u.d \leftarrow 0$ ;  
3 入队列 ( $Q, u$ );  
4 while  $Q$  非空 do  
5    $v \leftarrow$  出队列 ( $Q$ );  
6   foreach  $(v, w) \in E$  do  
7     if  $w.visited = false$  then  
8        $w.visited \leftarrow true$ ;  
9        $w.d \leftarrow v.d + 1$ ;  
10      入队列 ( $Q, w$ );
```

---

$v.d > dist(u, v) > dist(u, w) = w.d$   
即  $v.d > w.d + 1$

当 $w$ 出队列时,  
无论 $v$ 是白/灰/黑, 都矛盾



# 距离和BFS

- 时间复杂度:  $O(n + m)$

---

## 算法 2.3: BFS

---

**输入:** 图  $G = \langle V, E \rangle$ , 顶点  $u$

**初值:**  $V$  中所有顶点的 `visited` 初值为 `false`, `d` 初值为  $\infty$ ;  $Q$  初值为空队列

```
1  $u.visited \leftarrow true$ ;  
2  $u.d \leftarrow 0$ ;  
3 入队列 ( $Q, u$ );  
4 while  $Q$  非空 do  
5    $v \leftarrow$  出队列 ( $Q$ );  
6   foreach  $(v, w) \in E$  do  
7     if  $w.visited = false$  then  
8        $w.visited \leftarrow true$ ;  
9        $w.d \leftarrow v.d + 1$ ;  
10      入队列 ( $Q, w$ );
```

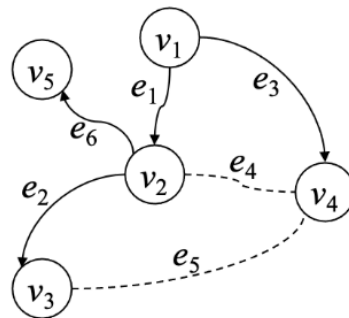
---

# 距离和BFS

## ■ BFS树

	d	parent
$v_1$	0	null
$v_2$	1	$v_1$
$v_3$	2	$v_2$
$v_4$	1	$v_1$
$v_5$	2	$v_2$

(a)



(b)

图 2.10 BFS 算法运行结果示例

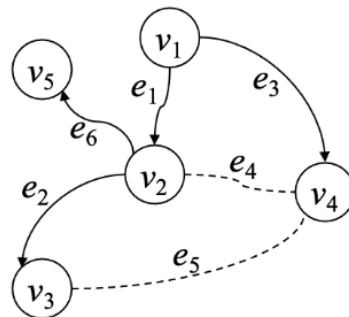
(a) 顶点的属性值; (b) 树边和其它边

# 距离和BFS

- BFS树
- 在BFS树中，以根顶点为起点的每条路都是最短路。

	d	parent
$v_1$	0	null
$v_2$	1	$v_1$
$v_3$	2	$v_2$
$v_4$	1	$v_1$
$v_5$	2	$v_2$

(a)



(b)

图 2.10 BFS 算法运行结果示例  
(a) 顶点的属性值；(b) 树边和其它边

# 距离和BFS

## ■ DFS树和BFS树的对比

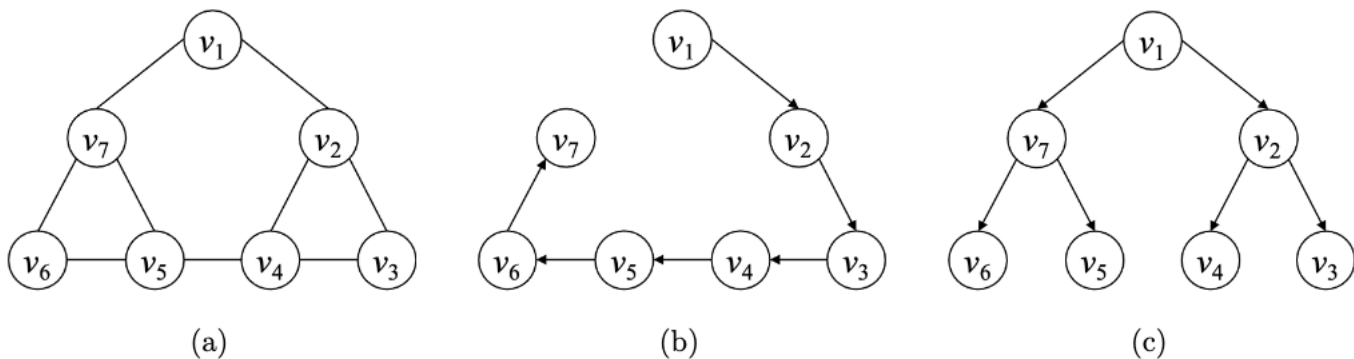
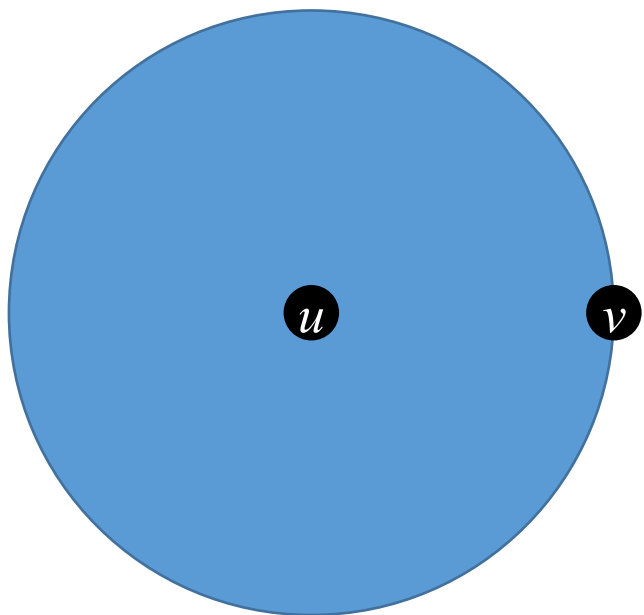


图 2.11 DFS 树和 BFS 树的对比

(a) 图  $G$ ; (b)  $G$  的 DFS 树; (c)  $G$  的 BFS 树

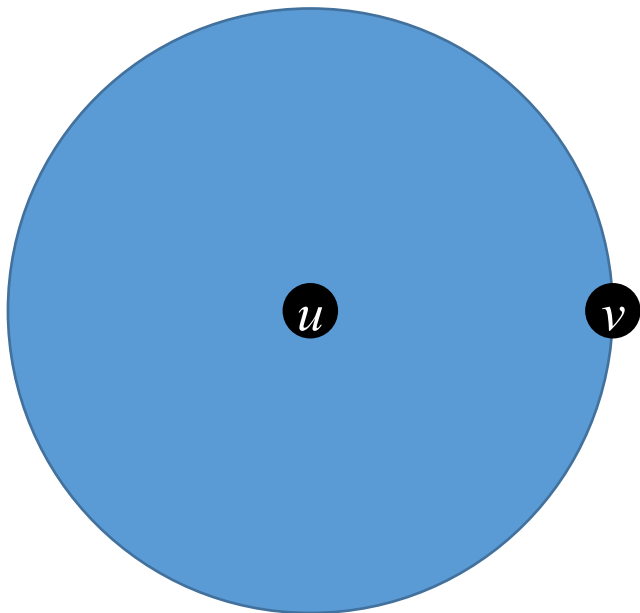
# 距离和BFS

- 单向搜索



# 距离和BFS

- 单向搜索
- 双向搜索



# 书面作业

- 练习2.1、2.2、2.3
- 练习2.8
- 练习2.11