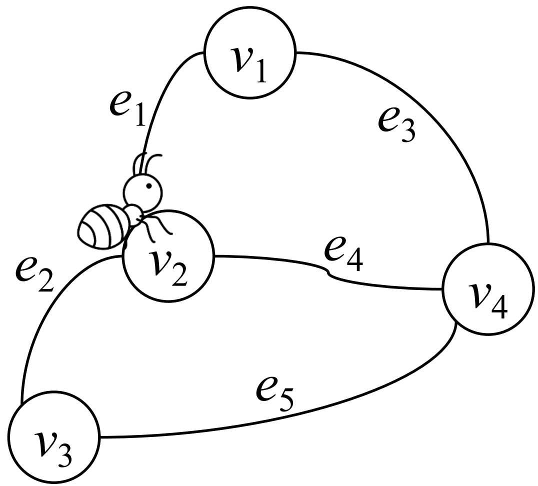
2 连通和遍历

程龚

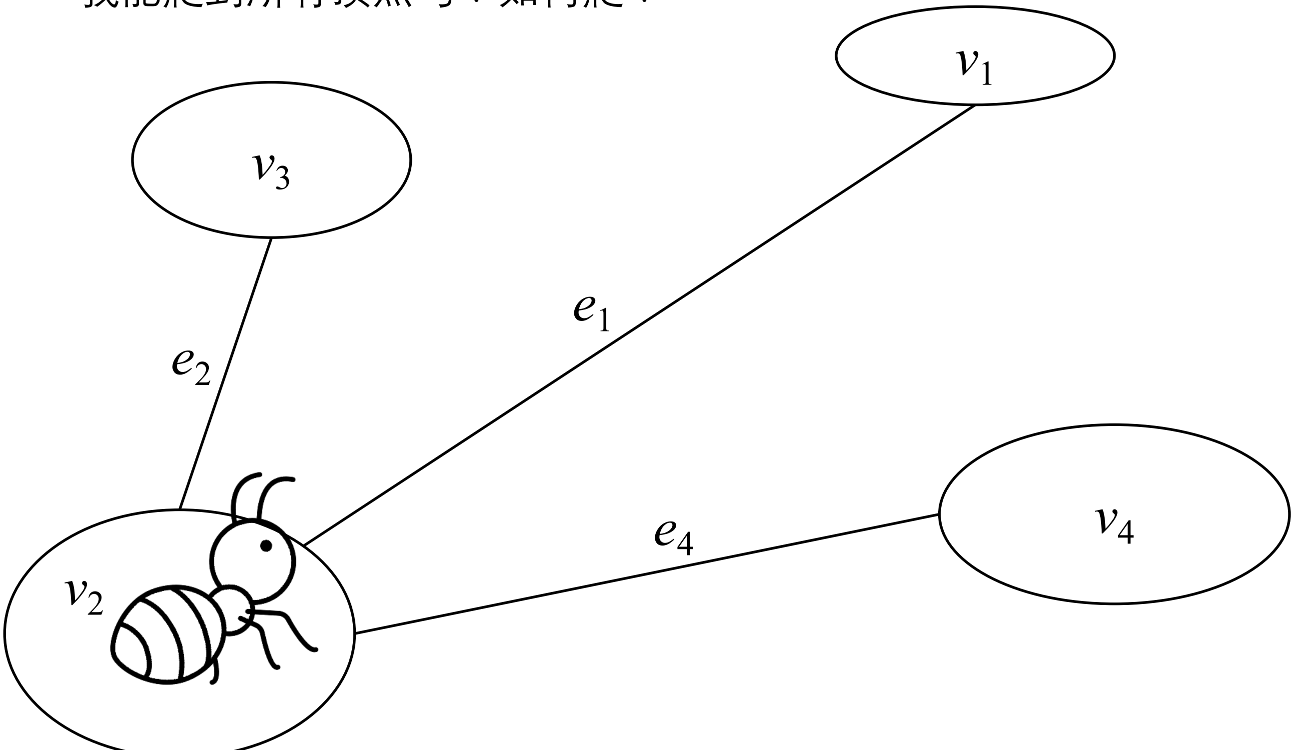
遍历——上帝视角

- 它能爬到所有顶点吗？如何爬？



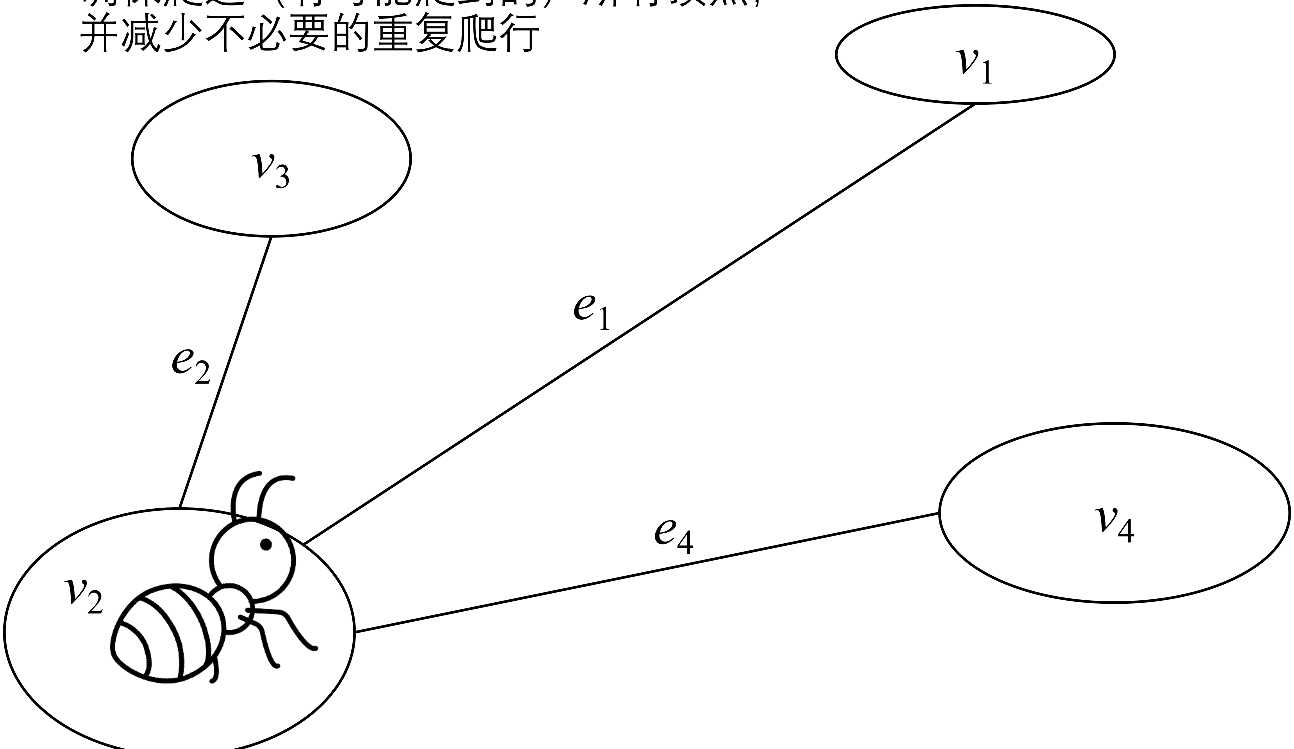
遍历——蚂蚁视角

- 我能爬到所有顶点吗？如何爬？



遍历——蚂蚁视角

- 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

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

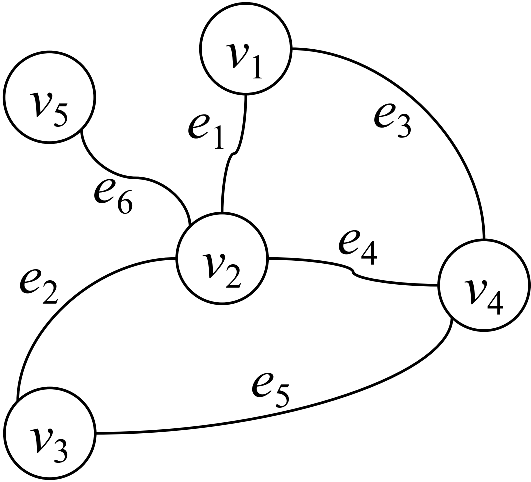
2.1 连通和DFS

2.2 割点和割边

2.3 距离和BFS

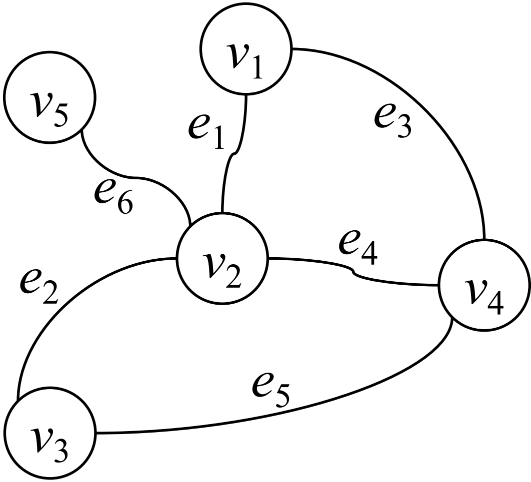
连通和DFS

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



连通和DFS

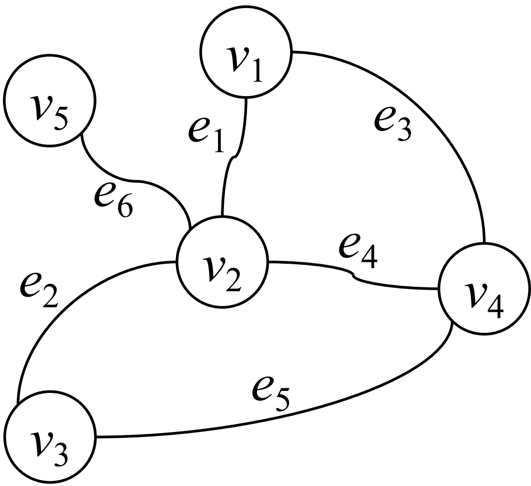
- 路线：以顶点开始、顶点和边交替出现、以顶点结束的序列 $v_0, e_1, v_1, \ldots e_l, v_l$，其中每条边$e_i$的两个端点恰为顶点$v_{i-1}$和$v_i$
  - 起点：v_0
  - 终点：v_l
  - 长度：l
- **平凡路线**：$l = 0$



连通和DFS

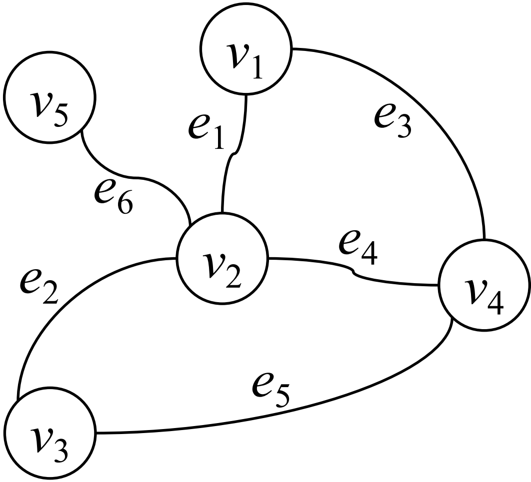
- 路线：以顶点开始、顶点和边交替出现、以顶点结束的序列 $v_0, e_1, v_1, \ldots e_l, v_l$，其中每条边$e_i$的两个端点恰为顶点$v_{i-1}$和$v_i$
  - 起点：v_0
  - 终点：v_l
  - 长度：l
- 平凡路线：$l = 0$

- **迹**：路线，且边在序列中不重复出现



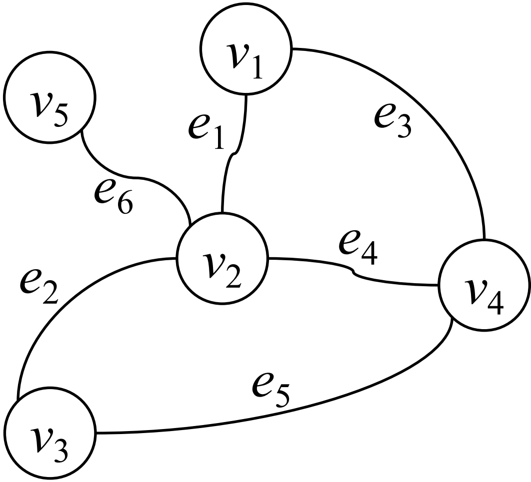
连通和DFS

- 路线：以顶点开始、顶点和边交替出现、以顶点结束的序列 $v_0, e_1, v_1, \ldots e_l, v_l$，其中每条边$e_i$的两个端点恰为顶点$v_{i\text{-}1}$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**：路中除起点和终点以外的其它顶点



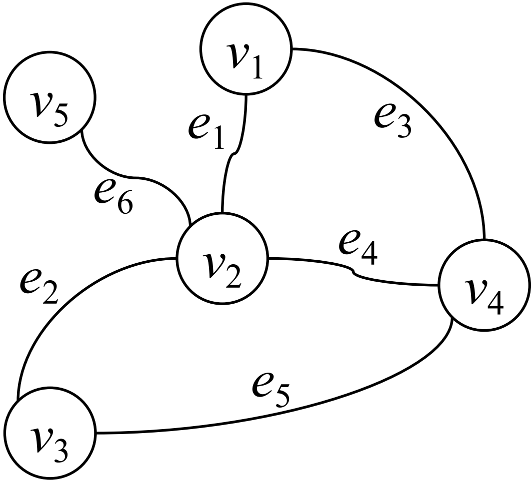
连通和DFS

- 若图中存在u-v路线，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，一定存在u-v路吗？



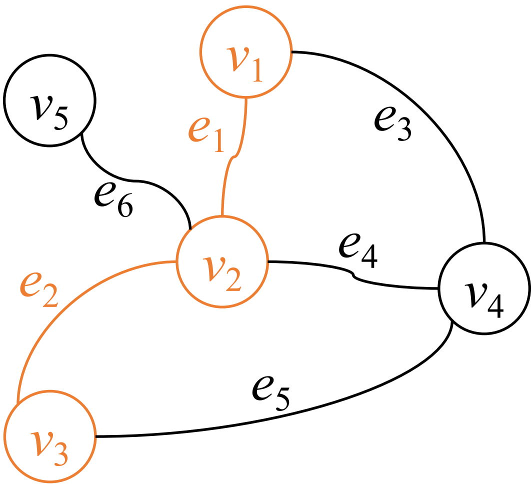
连通和DFS

- 若图中存在*u*-*v*路线，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，一定存在*u*-*v*路吗？
- 若图中存在*u*-*v*路线和*v*-*w*路线，一定存在*u*-*w*路线吗？



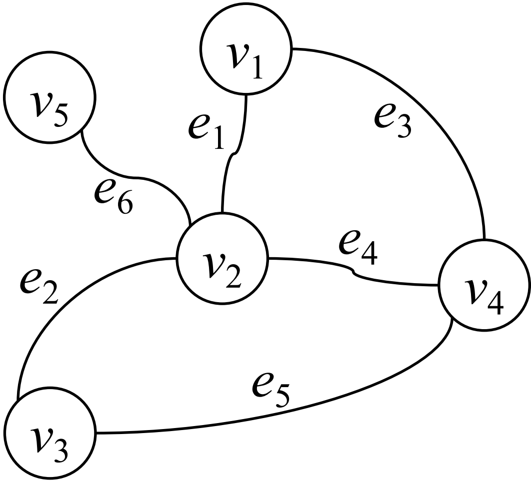
连通和DFS

- 顶点u和v**连通**：存在u-v路



连通和DFS

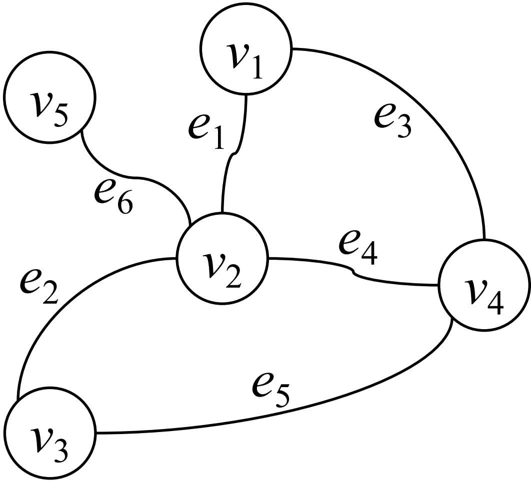
- 顶点u和v连通：存在u-v路
- 连通关系是定义在顶点集上的等价关系。



连通和DFS

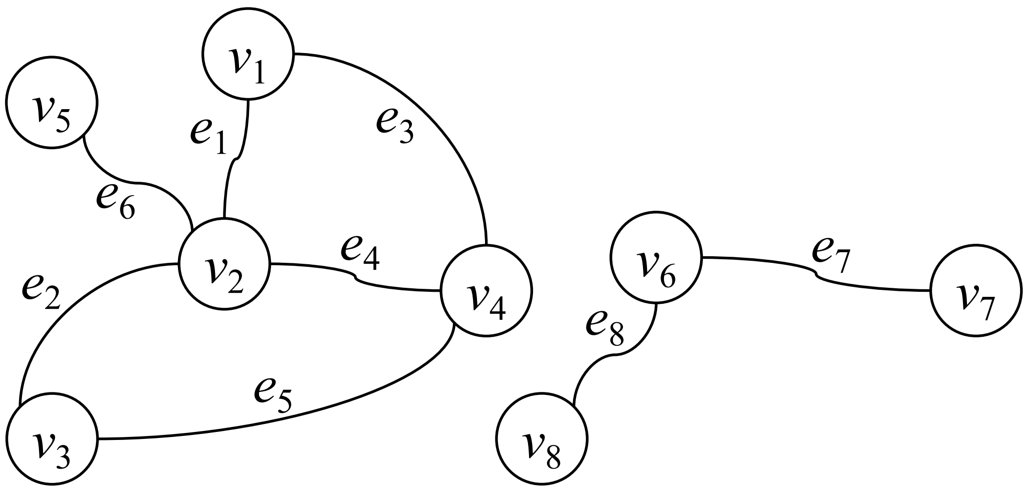
- 顶点u和v连通：存在u-v路
- 连通关系是定义在顶点集上的等价关系。

- 图$G = <V, E>$**连通**：V中每对顶点都连通



连通和DFS

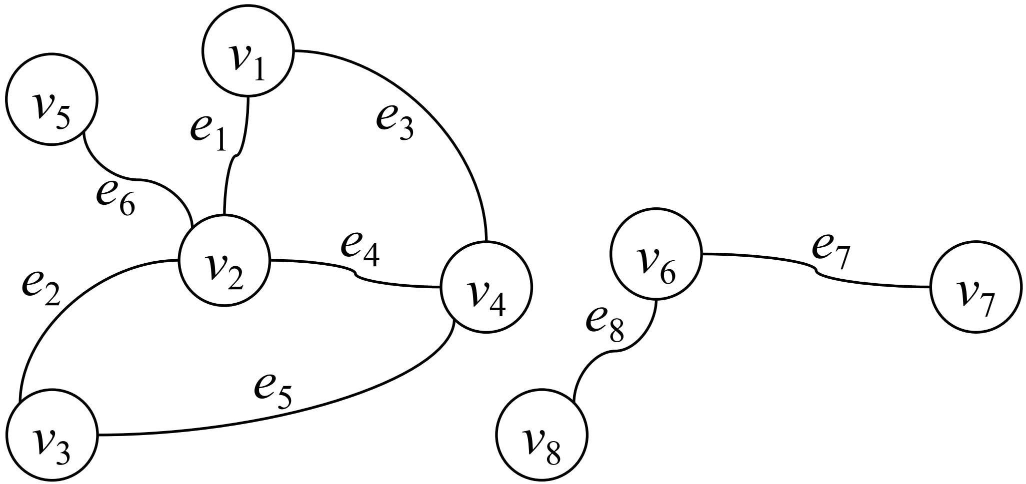
- **连通分支**：极大连通子图
- **平凡连通分支**：连通分支，且阶为1



连通和DFS

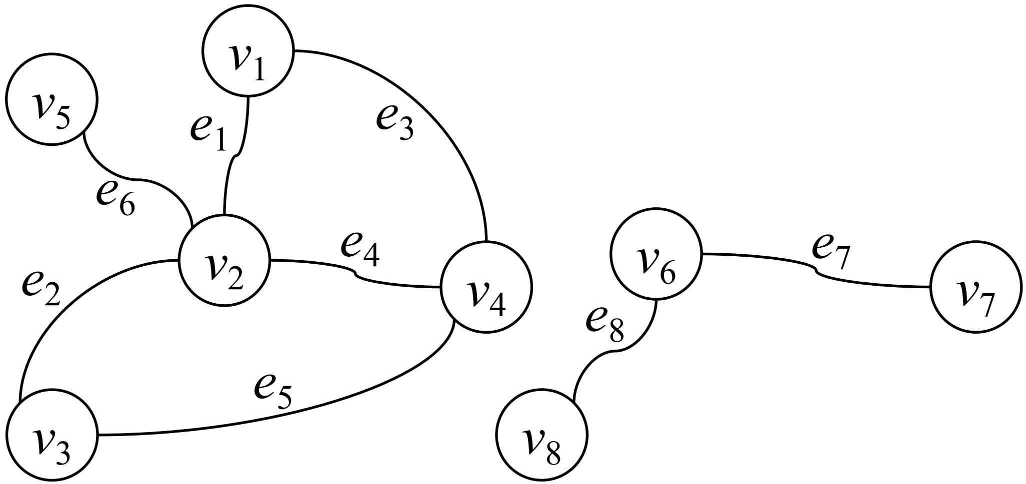
- 连通分支：极大连通子图
- 平凡连通分支：连通分支，且阶为1

- 顶点集V上的连通关系将V划分为等价类，
每个等价类V_i的点导出子图$G[V_i]$形成一个连通分支



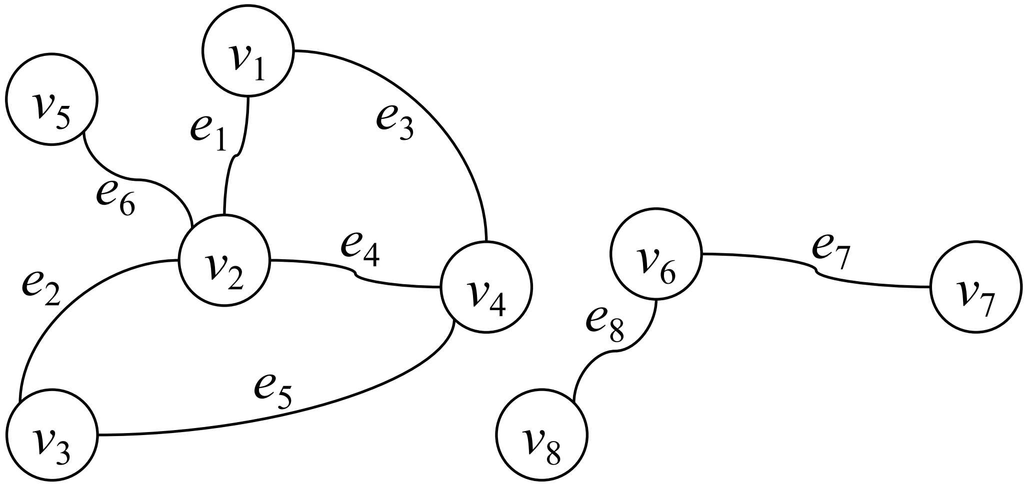
连通和DFS

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



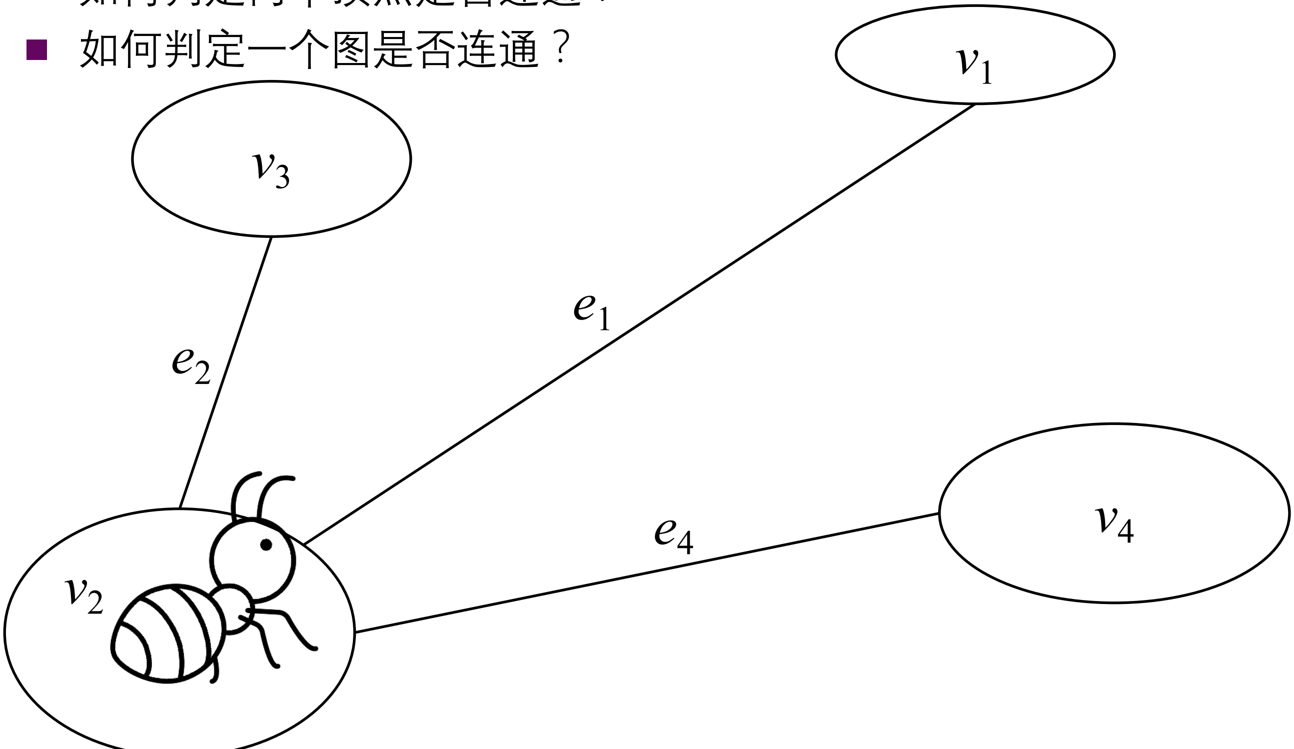
连通和DFS

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

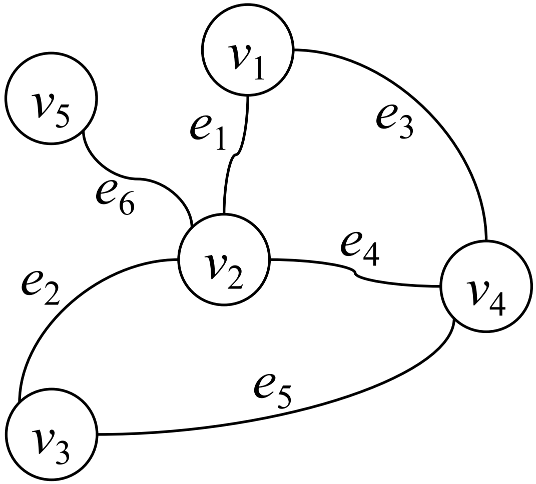
- 深度优先搜索（DFS）算法
  - 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
  - 遍历顶点的顺序是优先向图的“深处”访问，即倾向于远离出发点

算法 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

```
u.visited ← true;
foreach (u, v) ∈ E do
    if v.visited = false then
        DFS(G, v);
```



连通和DFS

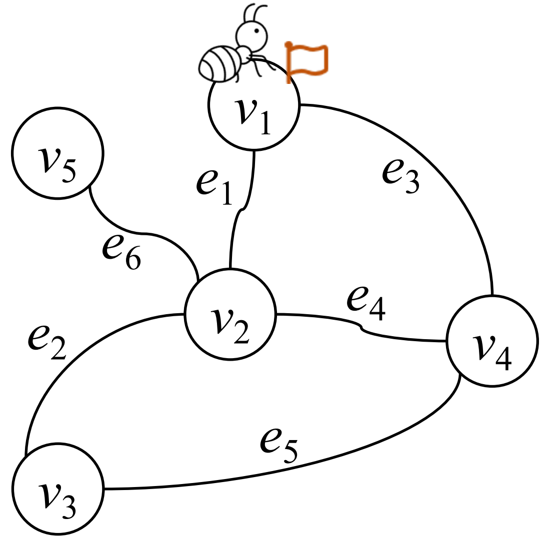
- 深度优先搜索（DFS）算法
  - 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
  - 遍历顶点的顺序是优先向图的“深处”访问，即倾向于远离出发点

算法 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

```
u.visited ← true;
foreach (u, v) ∈ E do
    if v.visited = false then
        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 1: DFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

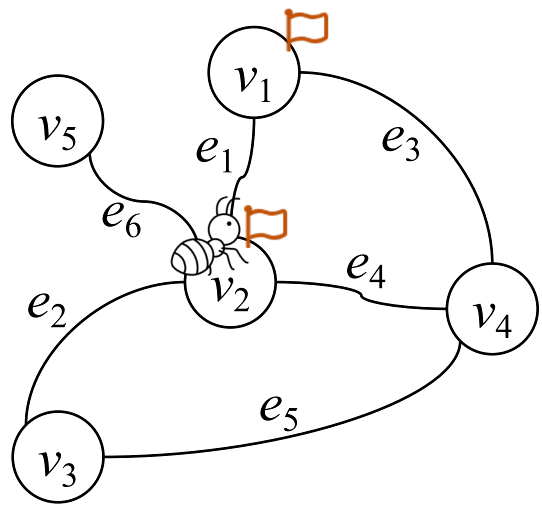
初值: V 中所有顶点的 visited 初值为 false

u.visited $\leftarrow$ true;

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

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

        `DFS(G, v)`;



连通和DFS

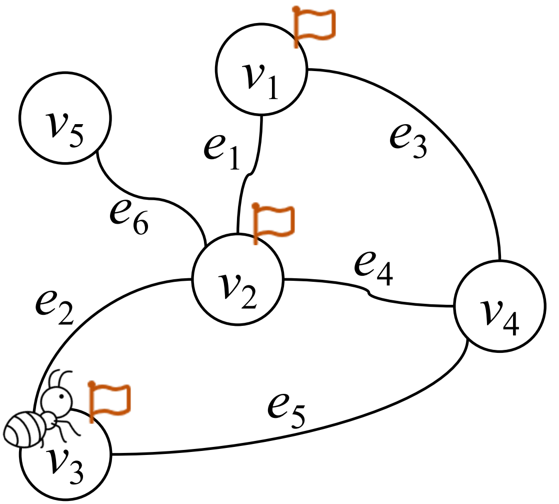
- 深度优先搜索（DFS）算法
  - 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
  - 遍历顶点的顺序是优先向图的“深处”访问，即倾向于远离出发点

算法 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

```
u.visited ← true;
foreach (u, v) ∈ E do
    if v.visited = false then
        DFS(G, v);
```



连通和DFS

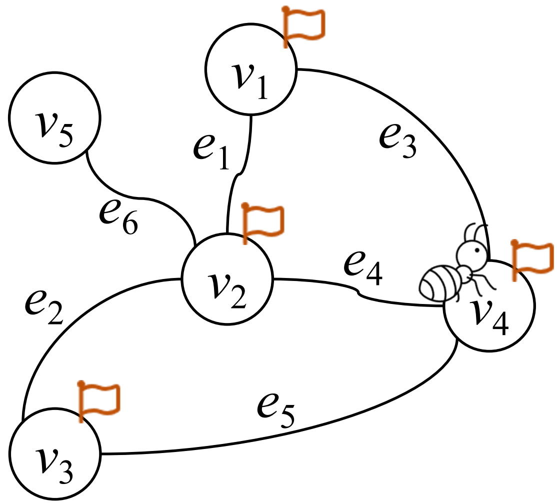
- 深度优先搜索（DFS）算法
  - 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
  - 遍历顶点的顺序是优先向图的“深处”访问，即倾向于远离出发点

算法 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

```
u.visited ← true;
foreach (u, v) ∈ E do
    if v.visited = false then
        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 1: DFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

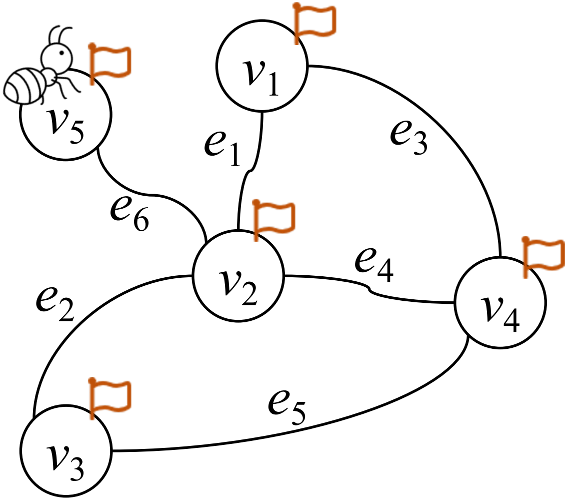
初值: V 中所有顶点的 visited 初值为 false

u.visited $\leftarrow$ true;

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

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

    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
- 时间复杂度：O($n + m$)

算法 1: DFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

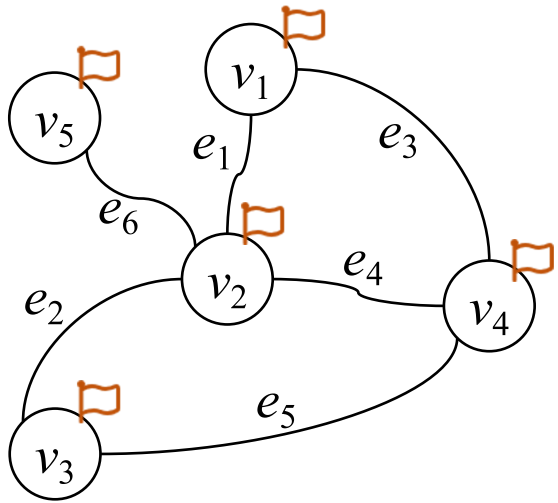
初值: V 中所有顶点的 visited 初值为 false

u.visited $\leftarrow$ true;

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

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

    DFS(G, v);



本次课的主要内容

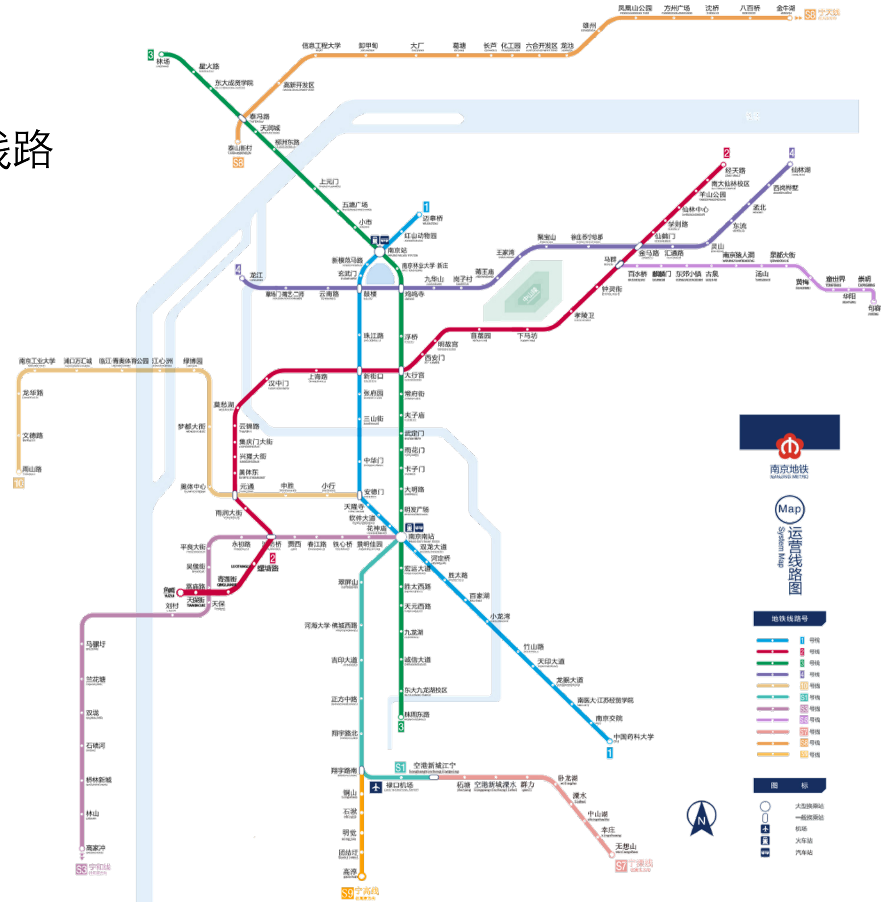
2.1 连通和DFS

2.2 割点和割边

2.3 距离和BFS

割点和割边

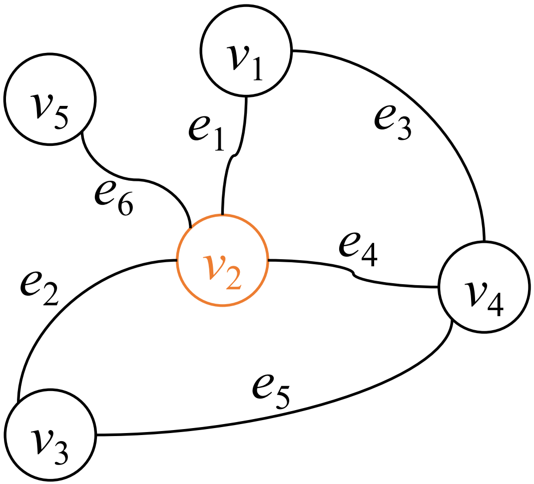
- 哪座地铁站、哪段线路最薄弱？



https://www.njmetro.com.cn/njdtweb/dtweb/images/map_new.jpg

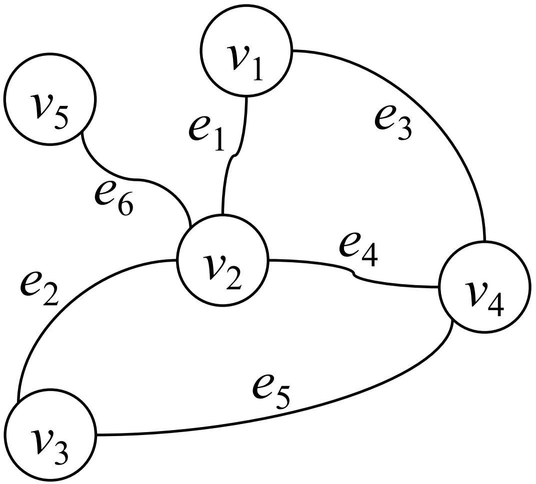
割点和割边

- **割点**（关节点）
  - 狭义定义：对于连通图$G = <V, E>$和顶点$v \in V$，$G - v$不连通
  - 广义定义：对于图$G = <V, E>$和顶点$v \in V$，$G - v$的连通分支数量大于G



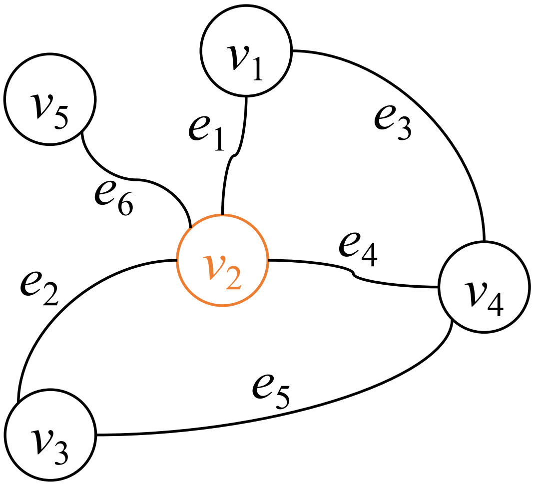
割点和割边

- 割点的度的下界是多少？



割点和割边

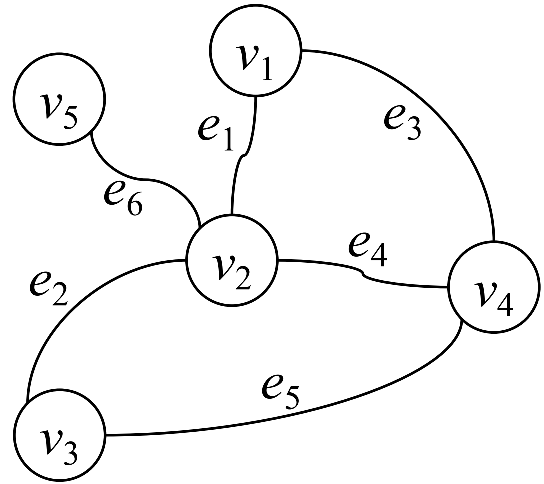
- 割点的度的下界是多少？
- 割点的等价定义：对于连通图$G = <V, E>$和顶点$v \in V$，v是G的割点当且仅当存在V的两个不相交的非空子集V_i和V_j，对于任意顶点$u \in V_i$和$w \in V_j$，每条u-w路都经过v。



割点和割边

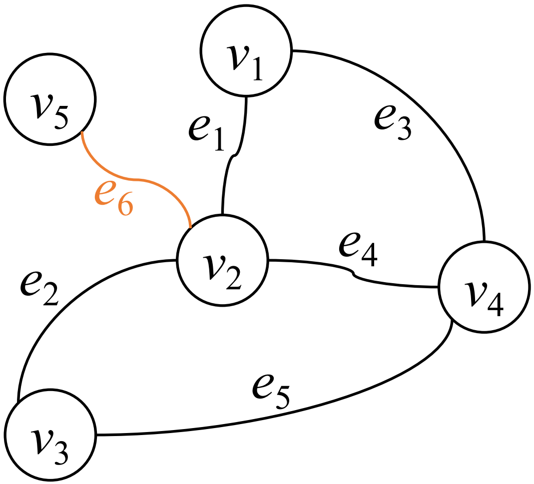
- 割点的度的下界是多少？
- 割点的等价定义：对于连通图$G=<V, E>$和顶点$v \in V$， v是G的割点当且仅当存在V的两个不相交的非空子集V_i和V_j，对于任意顶点$u \in V_i$和$w \in V_j$，每条u-w路都经过v。
- 若顶点v是连通图G的割点，则v也是$\overline{G}$的割点吗？

随堂小测



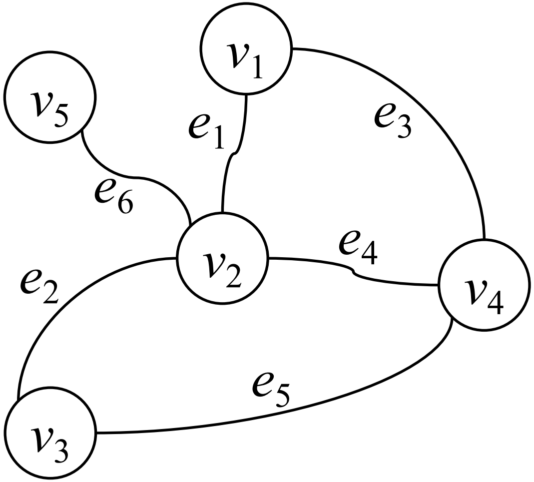
割点和割边

- **割边**（桥）
  - 广义定义：对于图$G = <V, E>$和边$e \in E$，$G - e$的连通分支数量大于G



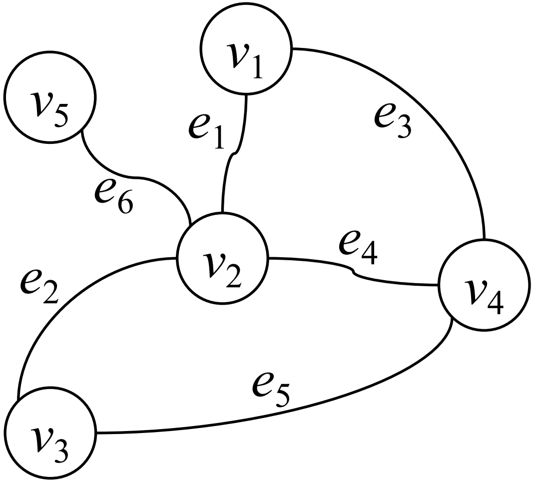
割点和割边

- 割点关联的边是割边吗？割边的端点是割点吗？



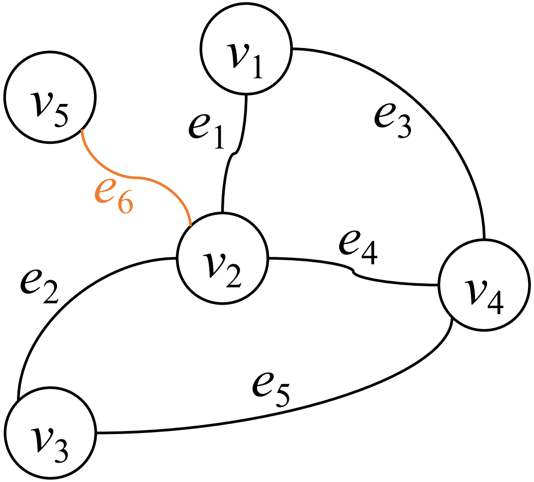
割点和割边

- 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？



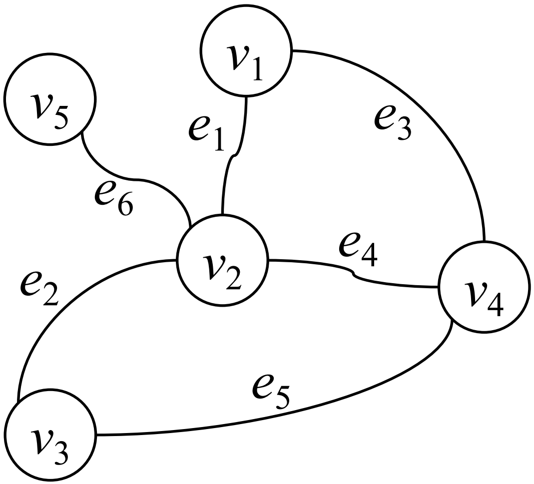
割点和割边

- 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？
- 割边的等价定义：对于连通图$G = <V, E>$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？
- 如何找出图中的所有割点？



割点和割边

■ 扩展DFS算法

算法 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

$time \leftarrow time + 1$;

u.d $\leftarrow$ $time$;

u.low $\leftarrow$ u.d;

u.visited $\leftarrow$ true;

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

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

    v.parent $\leftarrow$ u;

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

    DFSCV(G, v);

    u.low $\leftarrow$ min{u.low, v.low};

    if $u.parent = null$ 且 $u.children \geq 2$ **then**

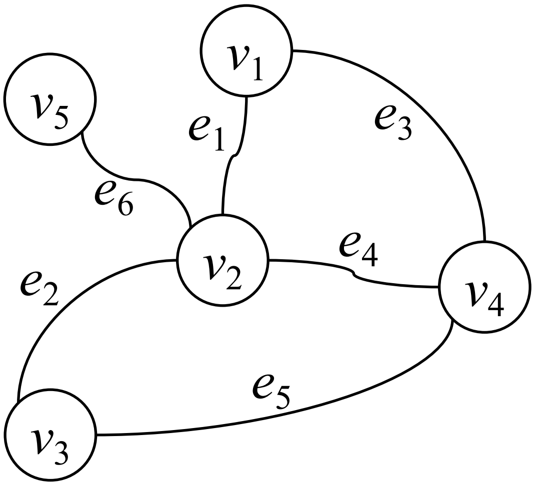
      u.isCutVertex $\leftarrow$ true;

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

      u.isCutVertex $\leftarrow$ true;

  else if $v \neq u.parent$ **then**

    u.low $\leftarrow$ min{u.low, v.d};



割点和割边

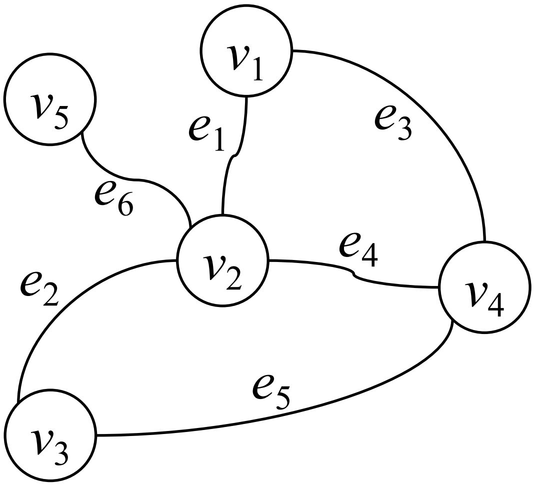
- 扩展DFS算法：记录每个顶点被访问的次序

算法 2: 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

$time \leftarrow time + 1$;
u.d $\leftarrow time$;
u.low $\leftarrow u$.d;
u.visited $\leftarrow$ true;
foreach $(u, v) \in E$ **do**
  if $v.visited = false$ **then**
    v.parent $\leftarrow u$;
    u.children $\leftarrow u$.children + 1;
    `DFSCV`(G, v);
    u.low $\leftarrow$ min{u.low, v.low};
    if $u.parent = null$ 且 $u.children \geq 2$ **then**
      u.isCutVertex $\leftarrow$ true;
    else if $u.parent \neq null$ 且 $v.low \geq u.d$ **then**
      u.isCutVertex $\leftarrow$ true;
  else if $v \neq u.parent$ **then**
    u.low $\leftarrow$ min{u.low, v.d};



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

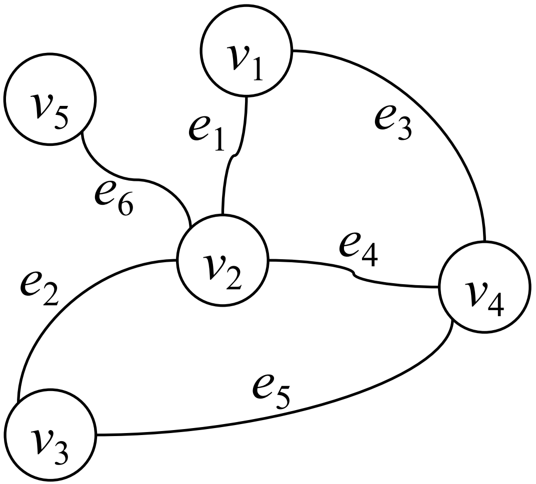
（根顶点：首个被访问的顶点）

算法 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

```
time ← time + 1;
u.d ← time;
u.low ← u.d;
u.visited ← true;
foreach (u, v) ∈ E do
    if v.visited = false then
        v.parent ← u;
        u.children ← u.children + 1;
        DFSCV(G, v);
        u.low ← min{u.low, v.low};
        if u.parent = null 且 u.children ≥ 2 then
            u.isCutVertex ← true;
        else if u.parent ≠ null 且 v.low ≥ u.d then
            u.isCutVertex ← true;
    else if v ≠ u.parent then
        u.low ← min{u.low, v.d};
```

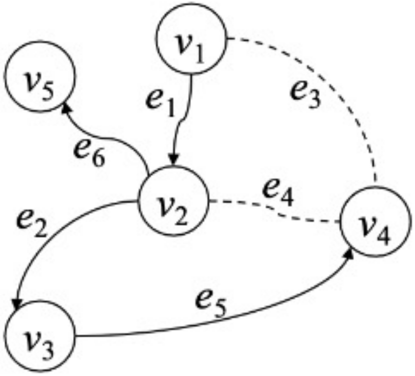


割点和割边

- 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	0	5

(a) 顶点的属性值



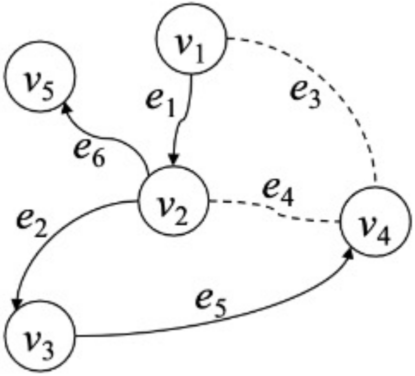
(b) 树边（实线）和后向边（虚线）

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	0	5

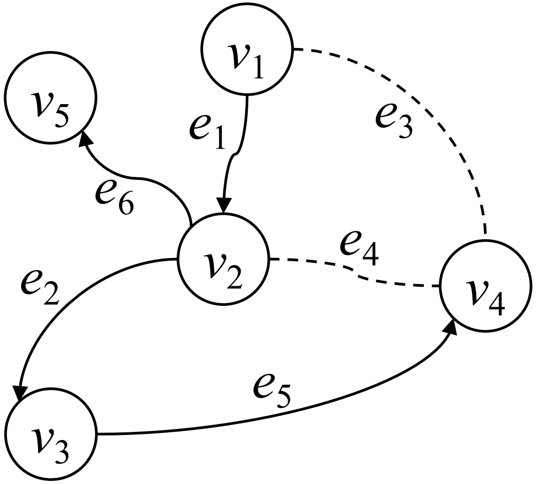
(a) 顶点的属性值



(b) 树边（实线）和后向边（虚线）

割点和割边

- 后向边关联一对祖先-后代顶点。



割点和割边

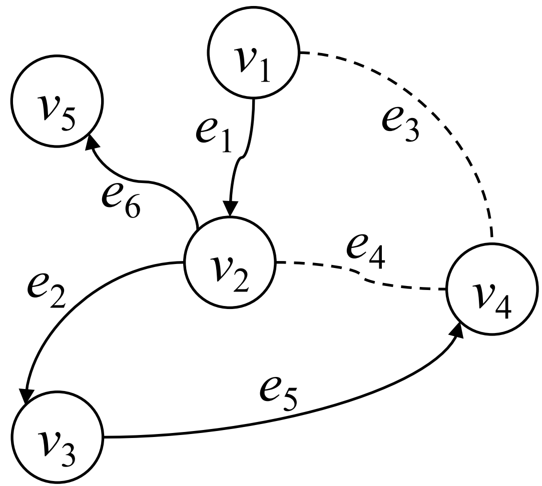
- 后向边关联一对祖先-后代顶点。
- 每个顶点有3种状态
  - 白：visited = false（DFS调用前）
  - 灰：visited = true，且DFS调用未结束
  - 黑：visited = true，且DFS调用已结束

算法 1: DFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

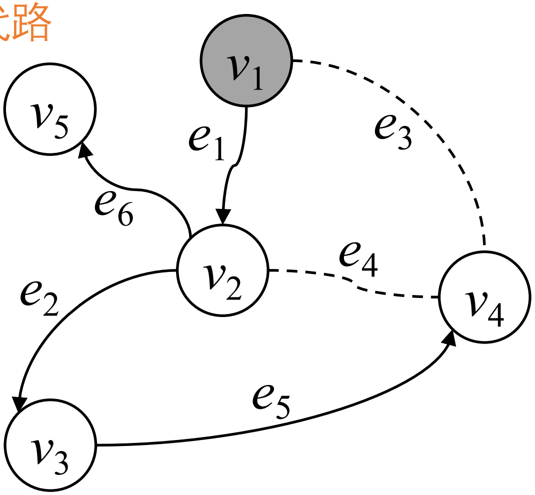
初值: V 中所有顶点的 visited 初值为 false

```
u.visited ← true;
foreach (u, v) ∈ E do
    if v.visited = false then
        DFS(G, v);
```



割点和割边

- 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 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

```
u.visited ← true;
foreach (u, v) ∈ E do
    if v.visited = false then
        DFS(G, v);
```

割点和割边

- 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 1: DFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

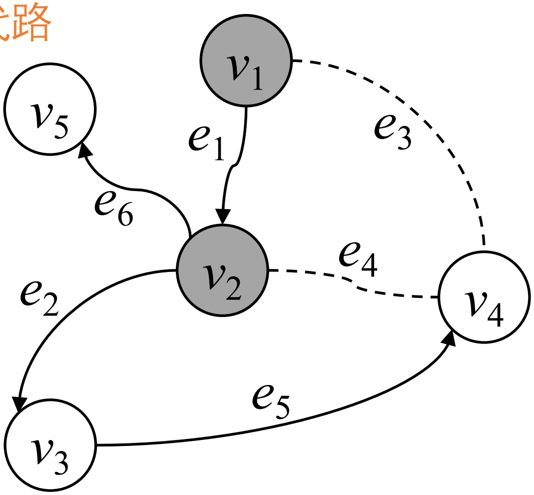
初值: V 中所有顶点的 visited 初值为 false

u.visited $\leftarrow$ true;

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

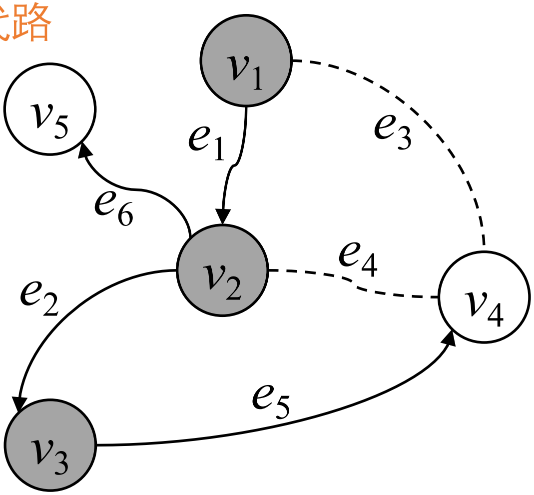
  if $v.visited = false$ **then**

    DFS(G, v);



割点和割边

- 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 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

u.visited $\leftarrow$ true;

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

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

        DFS(G, v);

割点和割边

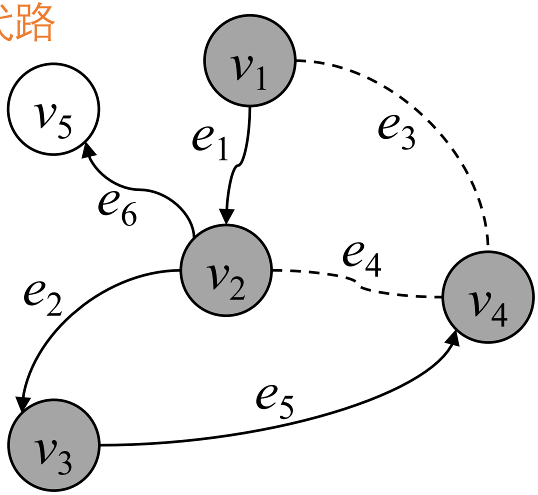
- 后向边关联一对祖先-后代顶点。
- 每个顶点有3种状态
  - 白：visited = false（DFS调用前）
  - 灰：visited = true，且DFS调用未结束
  - 黑：visited = true，且DFS调用已结束
- 任意时刻，所有灰点组成一条祖先-后代路

算法 1: DFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

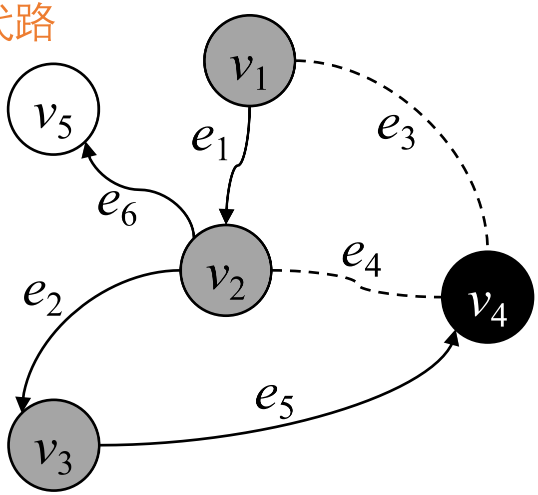
初值: V 中所有顶点的 visited 初值为 false

```
u.visited ← true;
foreach (u, v) ∈ E do
    if v.visited = false then
        DFS(G, v);
```



割点和割边

- 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 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

```
u.visited ← true;
foreach (u, v) ∈ E do
    if v.visited = false then
        DFS(G, v);
```

割点和割边

- 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 1: DFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

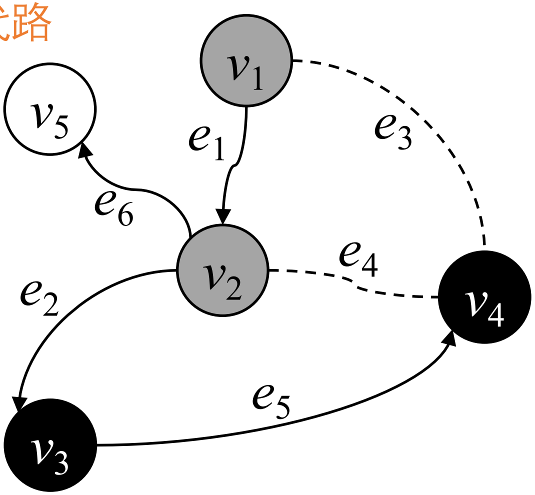
初值: V 中所有顶点的 visited 初值为 false

u.visited $\leftarrow$ true;

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

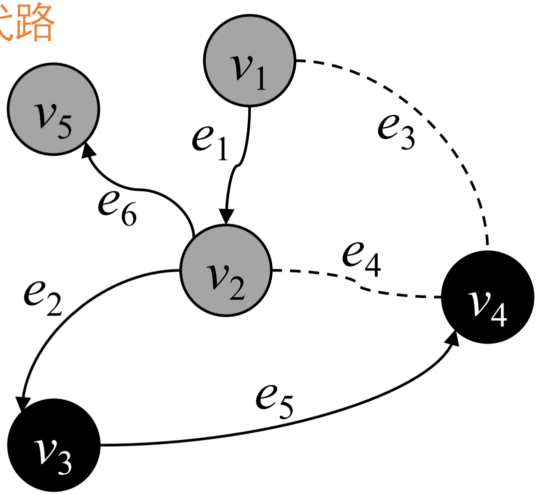
    if $v.visited = false$ **then**

        DFS(G, v);



割点和割边

- 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 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

u.visited $\leftarrow$ true;

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

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

    DFS(G, v);

割点和割边

- 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 1: DFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

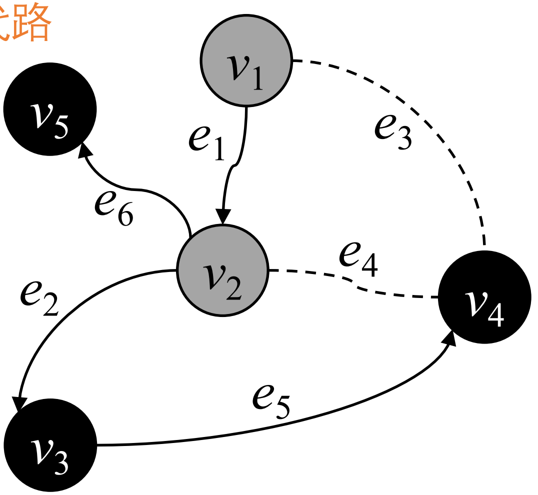
初值: V 中所有顶点的 visited 初值为 false

u.visited $\leftarrow$ true;

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

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

    DFS(G, v);



割点和割边

- 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 1: DFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

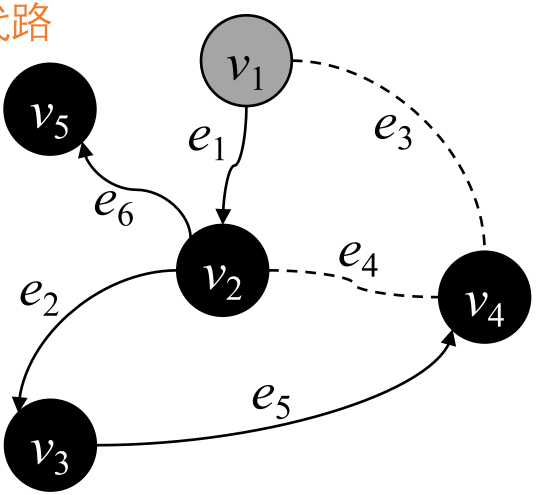
初值: V 中所有顶点的 visited 初值为 false

u.visited $\leftarrow$ true;

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

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

        DFS(G, v);



割点和割边

- 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 1: DFS

输入: 图 $G=\langle V,E\rangle$，顶点 u

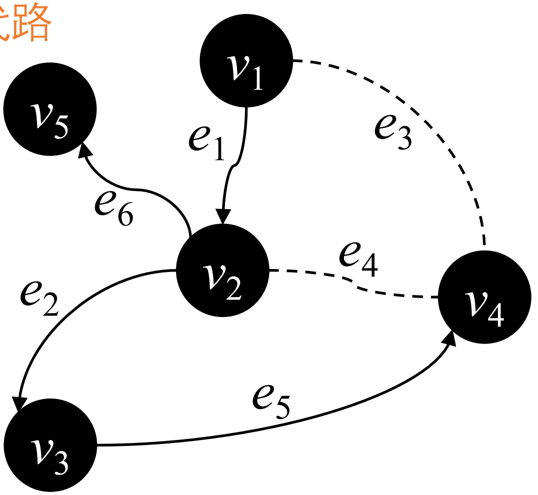
初值: V 中所有顶点的 visited 初值为 false

u.visited $\leftarrow$ true;

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

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

    DFS(G, v);



割点和割边

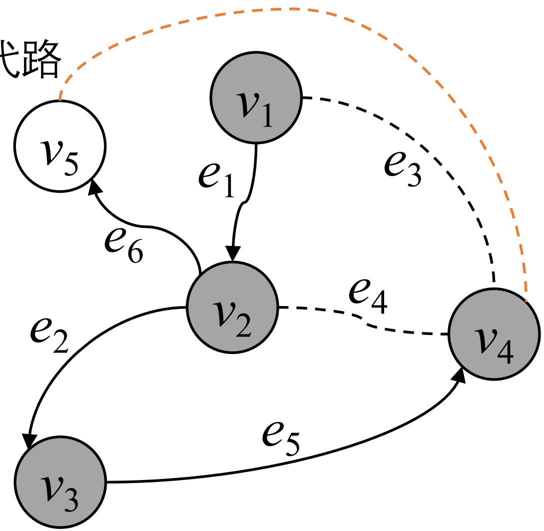
- 后向边关联一对祖先-后代顶点。
- 每个顶点有3种状态
  - 白：visited = false（DFS调用前）
  - 灰：visited = true，且DFS调用未结束
  - 黑：visited = true，且DFS调用已结束
- 任意时刻，所有灰点组成一条祖先-后代路
- 后向边的首次访问必从灰到灰（祖先）
  - 为什么不能到白？

算法 1: DFS

输入: 图 $G = \langle V, E \rangle$，顶点 u

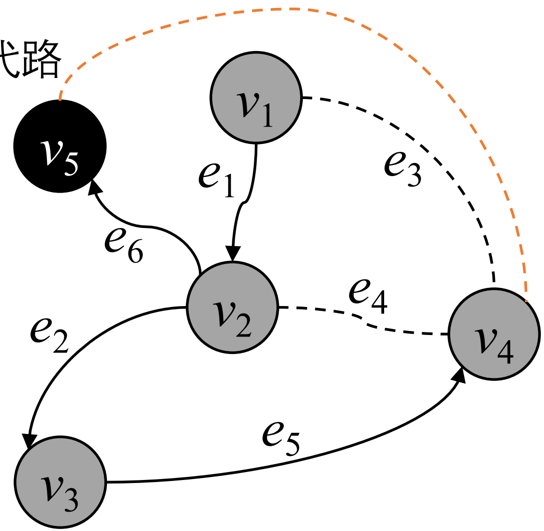
初值: V 中所有顶点的 visited 初值为 false

```
u.visited ← true;
foreach (u, v) ∈ E do
    if v.visited = false then
        DFS(G, v);
```



割点和割边

- 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 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

```
u.visited ← true;
foreach (u, v) ∈ E do
    if v.visited = false then
        DFS(G, v);
```

割点和割边

- 扩展DFS算法：判定非根顶点是否为割点

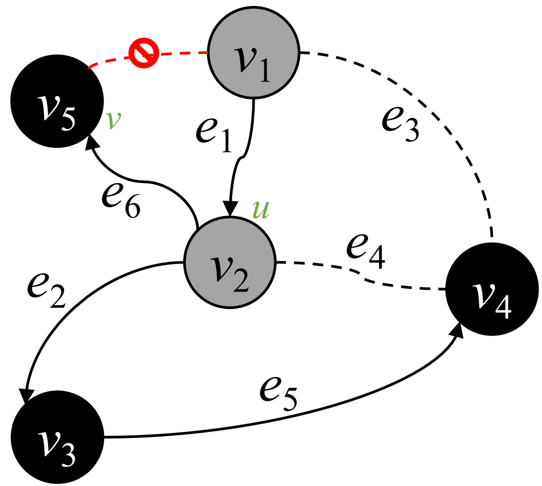
算法 2: DFSCV

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

```
time ← time + 1;
u.d ← time;
u.low ← u.d;
u.visited ← true;
foreach (u, v) ∈ E do
    if v.visited = false then
        v.parent ← u;
        u.children ← u.children + 1;
        DFSCV(G, v);
        u.low ← min{u.low, v.low};
        if u.parent = null 且 u.children ≥ 2 then
            u.isCutVertex ← true;
        else if u.parent ≠ null 且 v.low ≥ u.d then
            u.isCutVertex ← true;
    else if v ≠ u.parent then
        u.low ← min{u.low, v.d};
```

若顶点u不是根顶点，则u是割点的充要条件是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: DFSCV

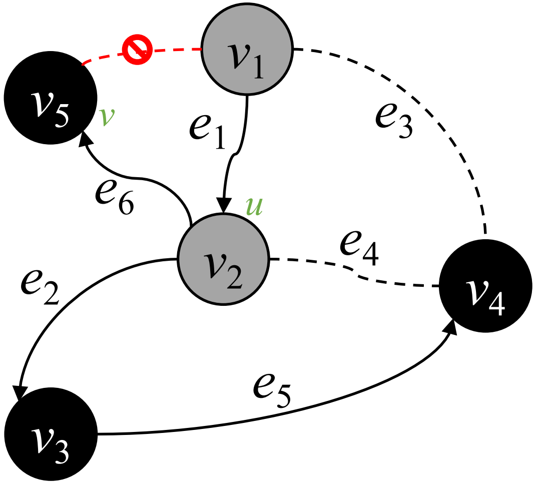
输入: 连通图 $G = \langle V, E\rangle$，顶点 u

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

$time \leftarrow time + 1$;
u.d $\leftarrow time$;
u.low $\leftarrow u$.d;
u.visited $\leftarrow$ true;
foreach $(u, v) \in E$ **do**
    if $v.visited = false$ **then**
        v.parent $\leftarrow u$;
        u.children $\leftarrow u$.children + 1;
        DFSCV(G, v);
        u.low $\leftarrow$ min{u.low, v.low};
        if $u.parent = null$ 且 $u.children \geq 2$ **then**
            u.isCutVertex $\leftarrow$ true;
        else if $u.parent \neq null$ 且 $v.low \geq u.d$ **then**
            u.isCutVertex $\leftarrow$ true;
    else if $v \neq u.parent$ **then**
        u.low $\leftarrow$ min{u.low, v.d};

u割开了谁和谁？

若顶点u不是根顶点，则u是割点的充要条件是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: DFSCV

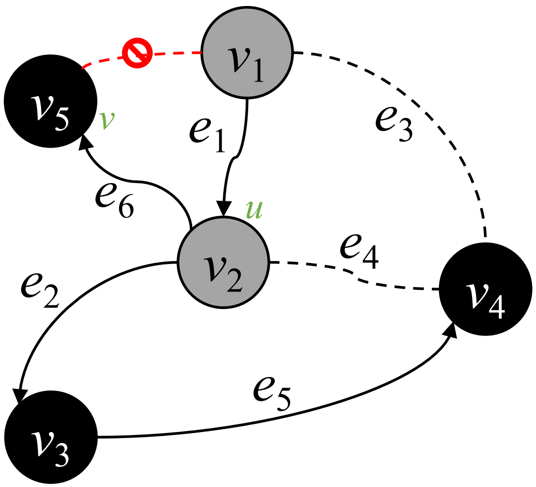
输入: 连通图 $G = \langle V, E \rangle$，顶点 u

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

```
time ← time + 1;
u.d ← time;
u.low ← u.d;
u.visited ← true;
foreach (u, v) ∈ E do
    if v.visited = false then
        v.parent ← u;
        u.children ← u.children + 1;
        DFSCV(G, v);
        u.low ← min{u.low, v.low};
        if u.parent = null 且 u.children ≥ 2 then
            u.isCutVertex ← true;
        else if u.parent ≠ null 且 v.low ≥ u.d then
            u.isCutVertex ← true;
    else if v ≠ u.parent then
        u.low ← 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

若顶点u不是根顶点，则u是割点的充要条件是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: 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

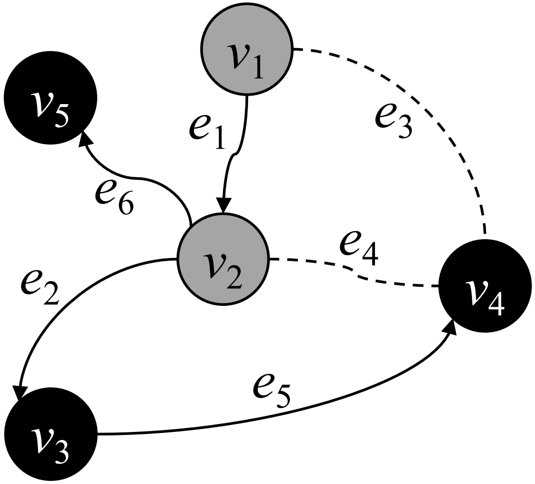
$time \leftarrow time + 1$;
u.d $\leftarrow time$;
u.low $\leftarrow u$.d;
u.visited $\leftarrow$ true;
foreach $(u, v) \in E$ **do**
  if $v.visited = false$ **then**
    v.parent $\leftarrow u$;
    u.children $\leftarrow u$.children + 1;
    DFSCV(G, v);
    u.low $\leftarrow$ min{u.low, v.low};
    if $u.parent = null$ 且 $u.children \geq 2$ **then**
      u.isCutVertex $\leftarrow$ true;
    else if $u.parent \neq null$ 且 $v.low \geq u.d$ **then**
      u.isCutVertex $\leftarrow$ true;
  else if $v \neq u.parent$ **then**
    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

树边(u, v)

后向边(u, v)

若顶点u不是根顶点，则u是割点的充要条件是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

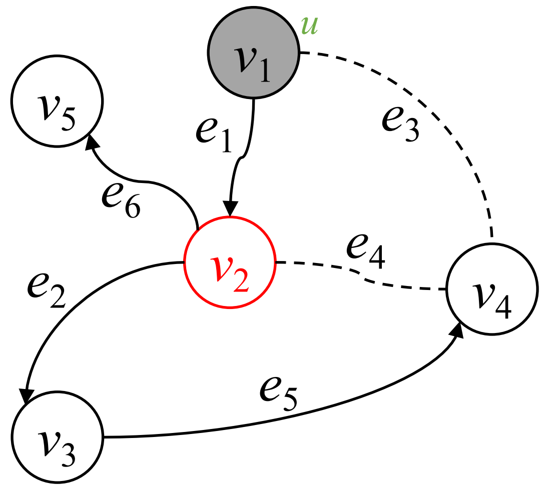
算法 2: DFSCV

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

$time \leftarrow time + 1$;
u.d $\leftarrow time$;
u.low $\leftarrow u$.d;
u.visited $\leftarrow$ true;
foreach $(u, v) \in E$ **do**
  if $v.visited = false$ **then**
    v.parent $\leftarrow u$;
    u.children $\leftarrow u$.children + 1;
    DFSCV(G, v);
    u.low $\leftarrow \min\{u.\text{low}, v.\text{low}\}$;
    if $u.parent = null$ 且 $u.children \geq 2$ **then**
      u.isCutVertex $\leftarrow$ true;
    else if $u.parent \neq null$ 且 $v.low \geq u.d$ **then**
      u.isCutVertex $\leftarrow$ true;
  else if $v \neq u.parent$ **then**
    u.low $\leftarrow \min\{u.\text{low}, v.\text{d}\}$;

若顶点u是根顶点，则u是割点的充要条件是它有至少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

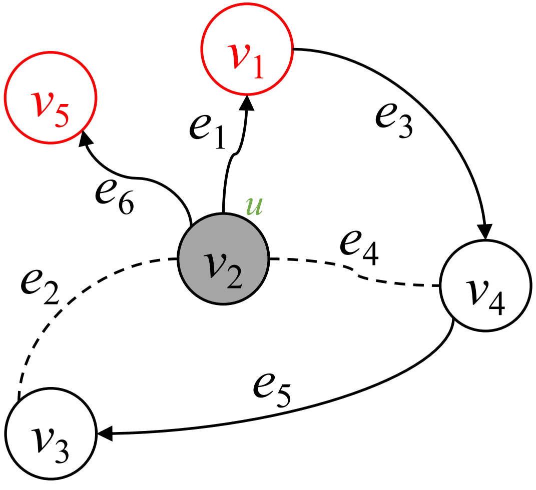
算法 2: DFSCV

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

```
time ← time + 1;
u.d ← time;
u.low ← u.d;
u.visited ← true;
foreach (u, v) ∈ E do
    if v.visited = false then
        v.parent ← u;
        u.children ← u.children + 1;
        DFSCV(G, v);
        u.low ← min{u.low, v.low};
        if u.parent = null 且 u.children ≥ 2 then
            u.isCutVertex ← true;
        else if u.parent ≠ null 且 v.low ≥ u.d then
            u.isCutVertex ← true;
    else if v ≠ u.parent then
        u.low ← min{u.low, v.d};
```

若顶点u是根顶点，则u是割点的充要条件是它有至少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: DFSCV

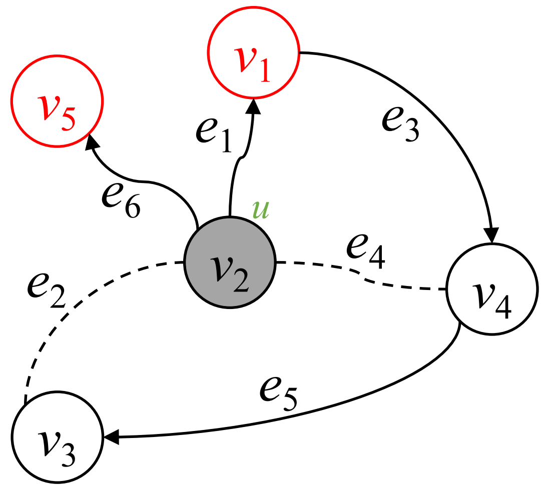
输入: 连通图 $G = \langle V, E \rangle$，顶点 u

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

$time \leftarrow time + 1$;
u.d $\leftarrow$ $time$;
u.low $\leftarrow$ u.d;
u.visited $\leftarrow$ true;
foreach $(u, v) \in E$ **do**
    if $v.visited = false$ **then**
        v.parent $\leftarrow$ u;
        u.children $\leftarrow$ u.children + 1;
        DFSCV(G, v);
        u.low $\leftarrow$ min{u.low, v.low};
        if $u.parent = null$ 且 $u.children \geq 2$ **then**
            u.isCutVertex $\leftarrow$ true;
        else if $u.parent \neq null$ 且 $v.low \geq u.d$ **then**
            u.isCutVertex $\leftarrow$ true;
    else if $v \neq u.parent$ **then**
        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是割点的充要条件是它有至少2个子顶点。



本次课的主要内容

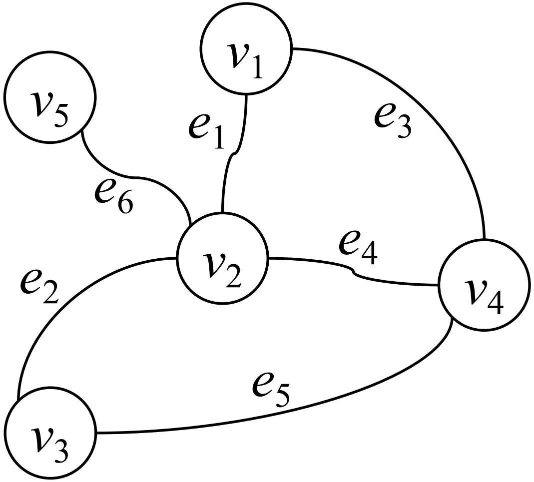
2.1 连通和DFS

2.2 割点和割边

2.3 距离和BFS

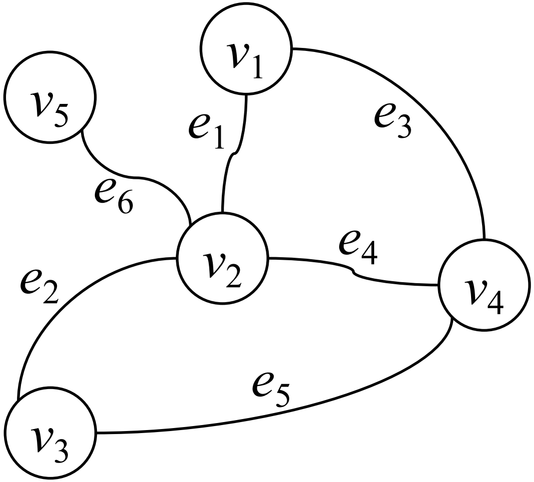
距离和BFS

- 顶点u和v间的**最短路**：长度最小的u-v路



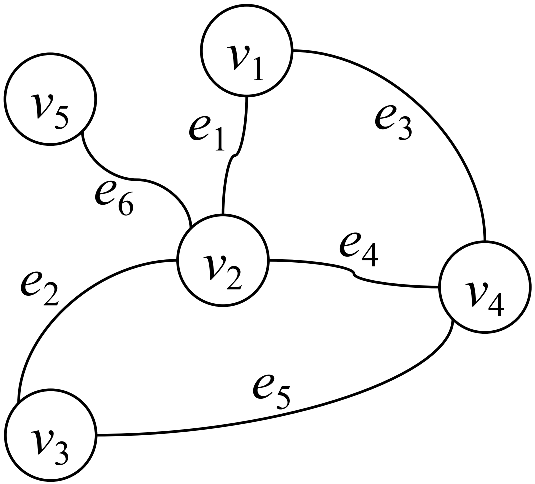
距离和BFS

- 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，唯一吗？



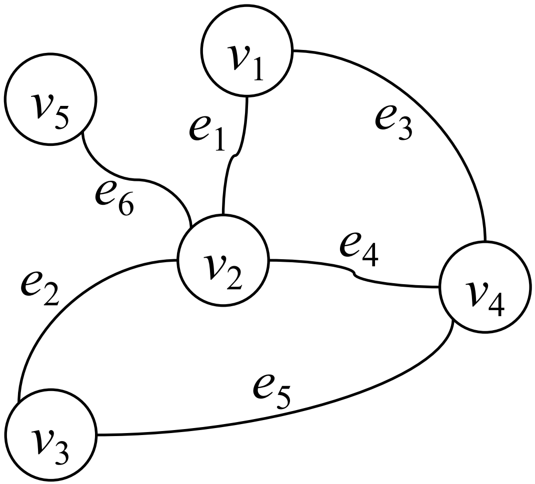
距离和BFS

- 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，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 = ∞



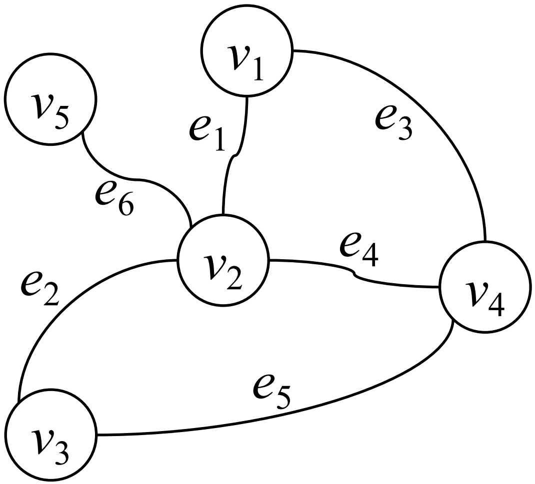
距离和BFS

- 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，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$\text{dist}(u, v)$
  - 不连通：$\text{dist} = \infty$
- 证明：对于连通图$G = \langle V, E\rangle$和顶点$u,v,w \in V$, $\text{dist}(u, v) + \text{dist}(v, w) \geq \text{dist}(u, w)$。



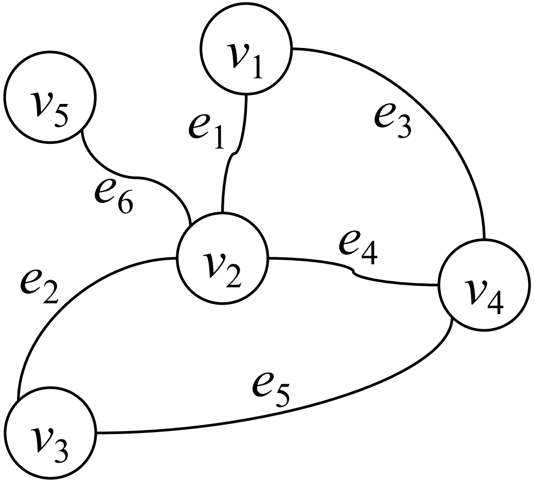
距离和BFS

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



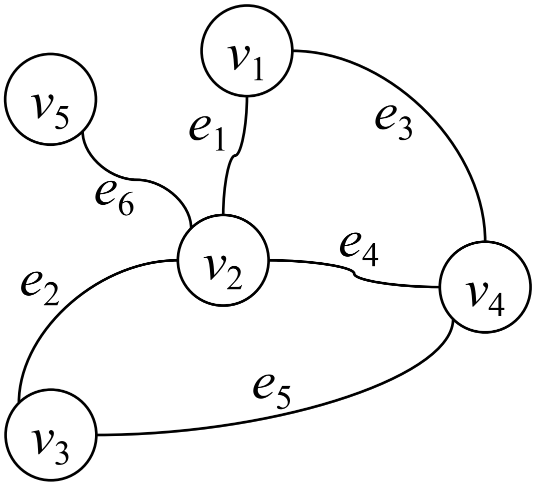
距离和BFS

- 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$G = <V, E>$和两个相邻顶点$u,v \in V$，$|\mathrm{ecc}(u) - \mathrm{ecc}(v)| \leq 1$。



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$\mathrm{ecc}(v)$
- 证明：对于连通图$G = \langle V, E\rangle$和两个相邻顶点$u,v \in V$, $|\mathrm{ecc}(u) - \mathrm{ecc}(v)| \le 1$。
- 证明：对于连通图$G = \langle V, E\rangle$和两个顶点$u,v \in V$, $|\mathrm{ecc}(u) - \mathrm{ecc}(v)| \le \mathrm{dist}(u, v)$。

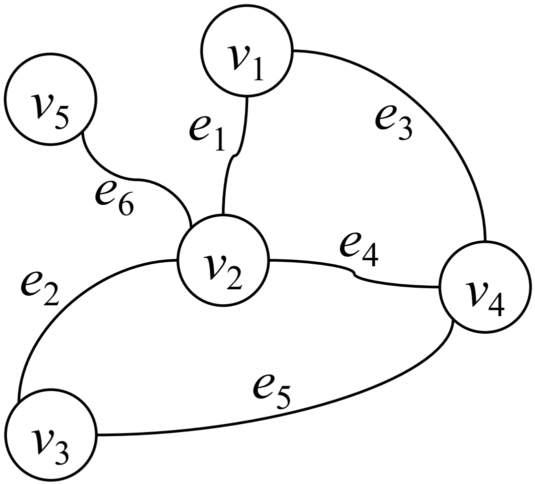


距离和BFS

- 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$G = <V, E>$和两个相邻顶点$u,v \in V$，$|\mathrm{ecc}(u) - \mathrm{ecc}(v)| \le 1$。
- 证明：对于连通图$G = <V, E>$和两个顶点$u,v \in V$，$|\mathrm{ecc}(u) - \mathrm{ecc}(v)| \le \mathrm{dist}(u, v)$。

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**：所有中心点形成的集合

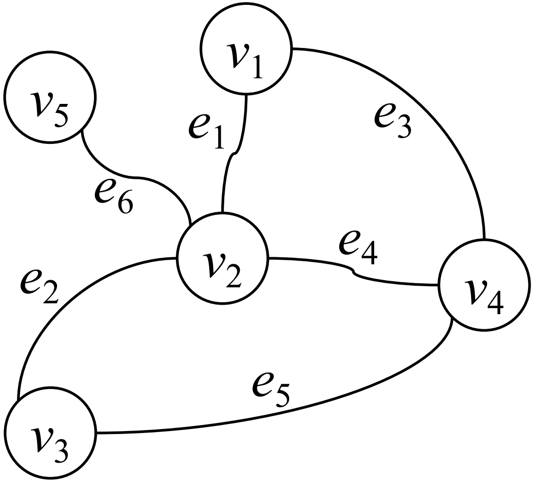


距离和BFS

- 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$G = <V, E>$和两个相邻顶点$u,v \in V$，$|\mathrm{ecc}(u) - \mathrm{ecc}(v)| \leq 1$。
- 证明：对于连通图$G = <V, E>$和两个顶点$u,v \in V$，$|\mathrm{ecc}(u) - \mathrm{ecc}(v)| \leq \mathrm{dist}(u, v)$。

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：所有中心点形成的集合
- **边缘点**：离心率最大的顶点
- **直径**：边缘点的离心率，记作diam(G)

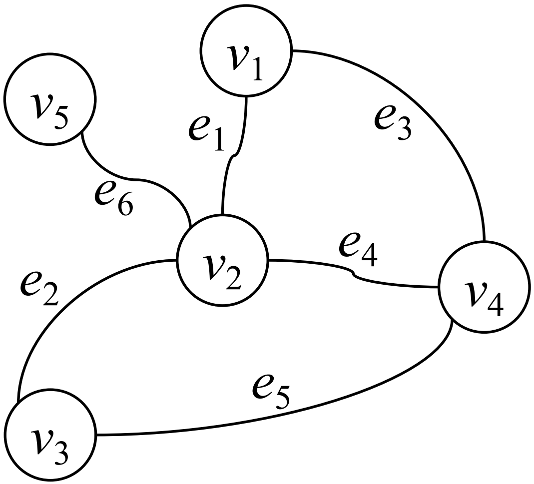


距离和BFS

- 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$u,v \in V$，$|\mathrm{ecc}(u) - \mathrm{ecc}(v)| \le 1$。
- 证明：对于连通图$G = \langle V, E \rangle$和两个顶点$u,v \in V$，$|\mathrm{ecc}(u) - \mathrm{ecc}(v)| \le \mathrm{dist}(u, v)$。

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：所有中心点形成的集合
- 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，$\mathrm{rad}(G) \le \mathrm{diam}(G) \le 2\mathrm{rad}(G)$。

距离和BFS

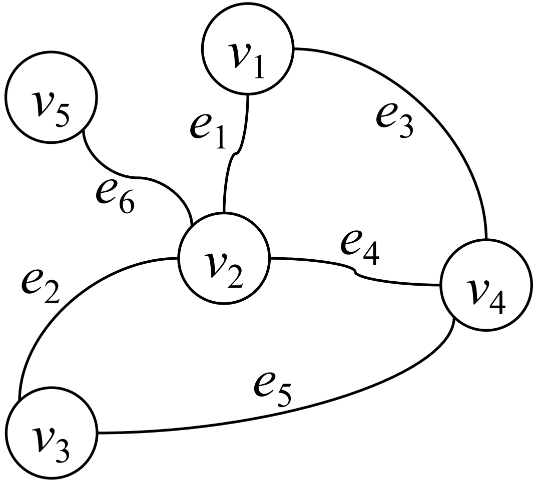
- 对于连通图G，$\text{rad}(G) \le \text{diam}(G) \le 2\text{rad}(G)$。

  取中心点w：

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

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

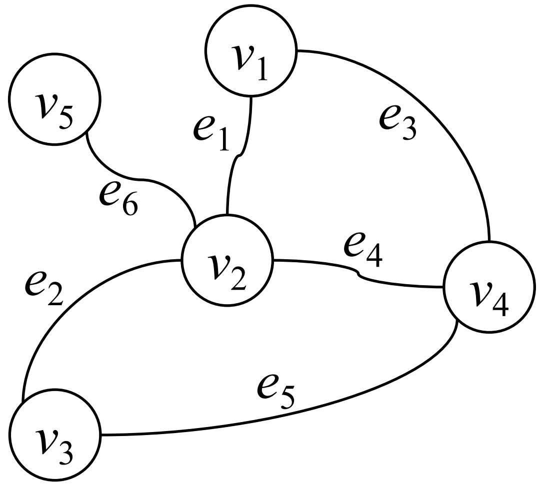
- 宽度优先搜索（BFS）算法
  - 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
  - 遍历顶点的顺序是优先向图的“浅处”访问，即倾向于贴近出发点

算法 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 ∞；Q 初值为空队列

```
u.visited ← true;
u.d ← 0;
入队列 (Q, u);
while Q 非空 do
    v ← 出队列 (Q);
    foreach (v, w) ∈ E do
        if w.visited = false then
            w.visited ← true;
            w.d ← v.d + 1;
            入队列 (Q, w);
```



距离和BFS

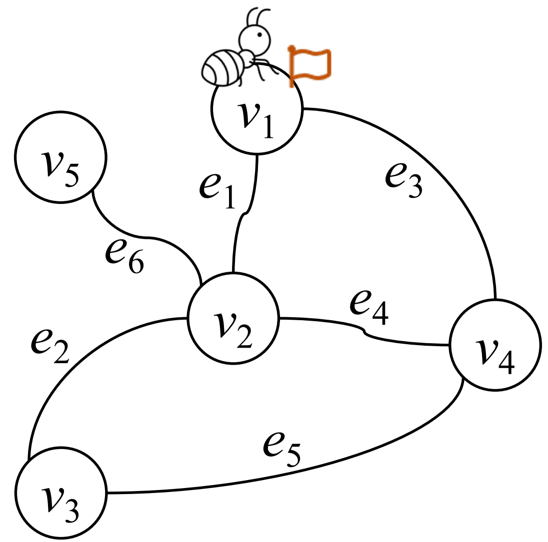
- 宽度优先搜索（BFS）算法
  - 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
  - 遍历顶点的顺序是优先向图的“浅处”访问，即倾向于贴近出发点

算法 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 ∞；Q 初值为空队列

u.visited $\leftarrow$ true;
u.d $\leftarrow$ 0;
入队列（Q, u）;
while Q 非空 **do**
  $v \leftarrow$ 出队列（Q）;
  foreach $(v, w) \in E$ **do**
    if $w.visited = false$ **then**
      w.visited $\leftarrow$ true;
      w.d $\leftarrow$ v.d + 1;
      入队列（Q, w）;



距离和BFS

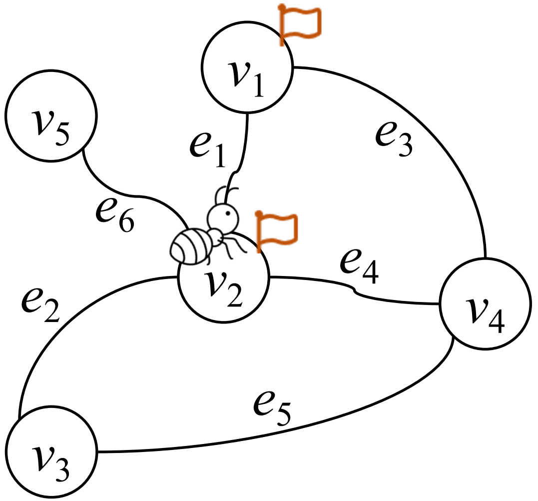
- 宽度优先搜索（BFS）算法
  - 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
  - 遍历顶点的顺序是优先向图的“浅处”访问，即倾向于贴近出发点

算法 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 ∞；Q 初值为空队列

```
u.visited ← true;
u.d ← 0;
入队列（Q, u);
while Q 非空 do
    v ← 出队列（Q);
    foreach (v, w) ∈ E do
        if w.visited = false then
            w.visited ← true;
            w.d ← v.d + 1;
            入队列（Q, w);
```



距离和BFS

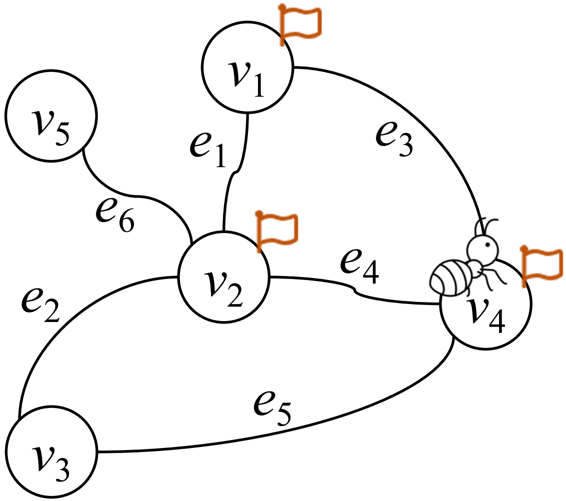
- 宽度优先搜索（BFS）算法
  - 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
  - 遍历顶点的顺序是优先向图的“浅处”访问，即倾向于贴近出发点

算法 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 ∞；Q 初值为空队列

u.visited $\leftarrow$ true;
u.d $\leftarrow$ 0;
入队列（Q, u）;
while Q 非空 **do**
    $v \leftarrow$ 出队列（Q）;
    foreach $(v, w) \in E$ **do**
        if $w.visited = false$ **then**
            w.visited $\leftarrow$ true;
            w.d $\leftarrow v$.d + 1;
            入队列（Q, w）;



距离和BFS

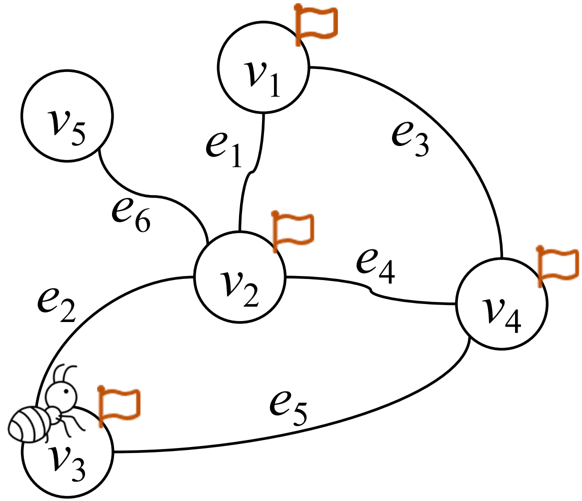
- 宽度优先搜索（BFS）算法
  - 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
  - 遍历顶点的顺序是优先向图的“浅处”访问，即倾向于贴近出发点

算法 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 ∞；Q 初值为空队列

u.visited $\leftarrow$ true;
u.d $\leftarrow$ 0;
入队列（Q, u）;
while Q 非空 **do**
  $v \leftarrow$ 出队列（Q）;
  foreach $(v, w) \in E$ **do**
    if $w.visited = false$ **then**
      w.visited $\leftarrow$ true;
      w.d $\leftarrow$ v.d + 1;
      入队列（Q, w）;



距离和BFS

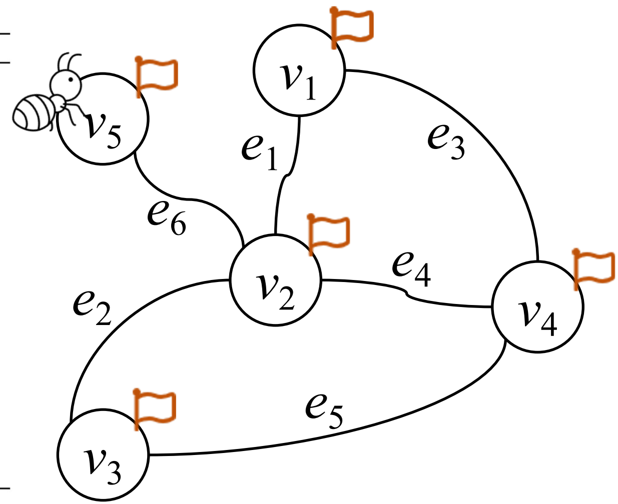
- 宽度优先搜索（BFS）算法
  - 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
  - 遍历顶点的顺序是优先向图的“浅处”访问，即倾向于贴近出发点

算法 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 ∞；Q 初值为空队列

```
u.visited ← true;
u.d ← 0;
入队列 (Q, u);
while Q 非空 do
    v ← 出队列 (Q);
    foreach (v, w) ∈ E do
        if w.visited = false then
            w.visited ← true;
            w.d ← v.d + 1;
            入队列 (Q, w);
```



距离和BFS

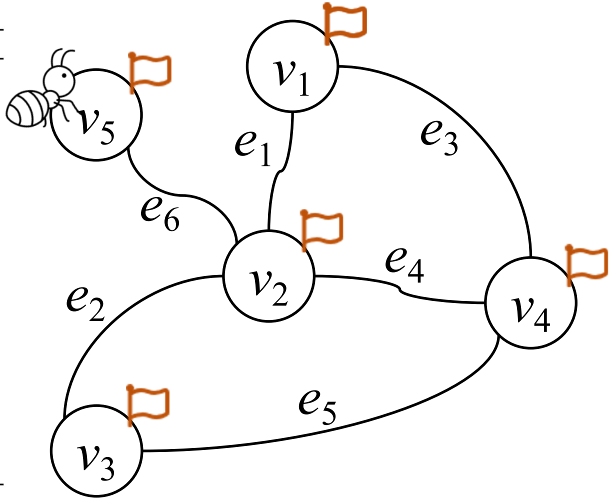
- 宽度优先搜索（BFS）算法
  - 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
  - 遍历顶点的顺序是优先向图的“浅处”访问，即倾向于贴近出发点
- 时间复杂度：$O(n+m)$

算法 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 ∞；Q 初值为空队列

```
u.visited ← true;
u.d ← 0;
入队列（Q, u);
while Q 非空 do
    v ← 出队列（Q);
    foreach (v, w) ∈ E do
        if w.visited = false then
            w.visited ← true;
            w.d ← v.d + 1;
            入队列（Q, w);
```



距离和BFS

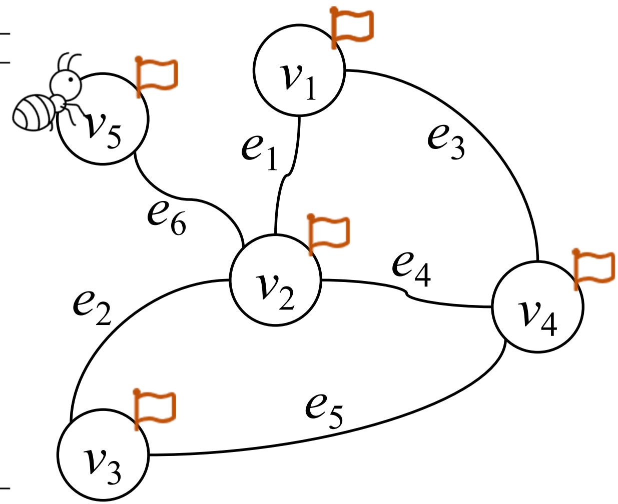
- 宽度优先搜索（BFS）算法
  - 从图中的一个指定顶点出发，有序地遍历和该顶点连通的所有顶点
  - 遍历顶点的顺序是优先向图的“浅处”访问，即倾向于贴近出发点
- 时间复杂度：O($n + m$)
- 可否省略visited属性？

算法 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 ∞；Q 初值为空队列

u.visited $\leftarrow$ true;
u.d $\leftarrow$ 0;
入队列（Q, u);
while Q 非空 **do**
  $v \leftarrow$ 出队列（Q);
  foreach $(v, w) \in E$ **do**
    if $w.visited = false$ **then**
      w.visited $\leftarrow$ true;
      w.d $\leftarrow$ v.d + 1;
      入队列（Q, w);

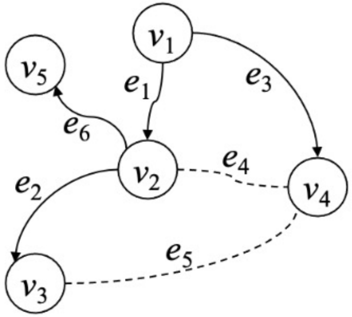


距离和BFS

- 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
- 时间复杂度：$O(n+m)$
- 可否省略visited属性？
- BFS树

	d	parent
v_1	1	null
v_2	2	v_1
v_3	3	v_2
v_4	2	v_1
v_5	3	v_2

(a) 顶点的属性值



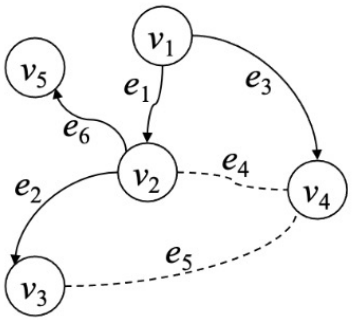
(b) 树边（实线）和其它边（虚线）

距离和BFS

- 按BFS算法的访问顺序，所有顶点的d属性值组成非减序列。

	d	parent
v_1	1	null
v_2	2	v_1
v_3	3	v_2
v_4	2	v_1
v_5	3	v_2

(a) 顶点的属性值



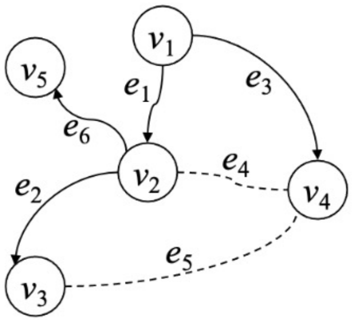
(b) 树边（实线）和其它边（虚线）

距离和BFS

- 按BFS算法的访问顺序，所有顶点的d属性值组成非减序列。
- 在BFS树中，以根顶点为起点的每条路都是最短路。

	d	parent
v_1	1	null
v_2	2	v_1
v_3	3	v_2
v_4	2	v_1
v_5	3	v_2

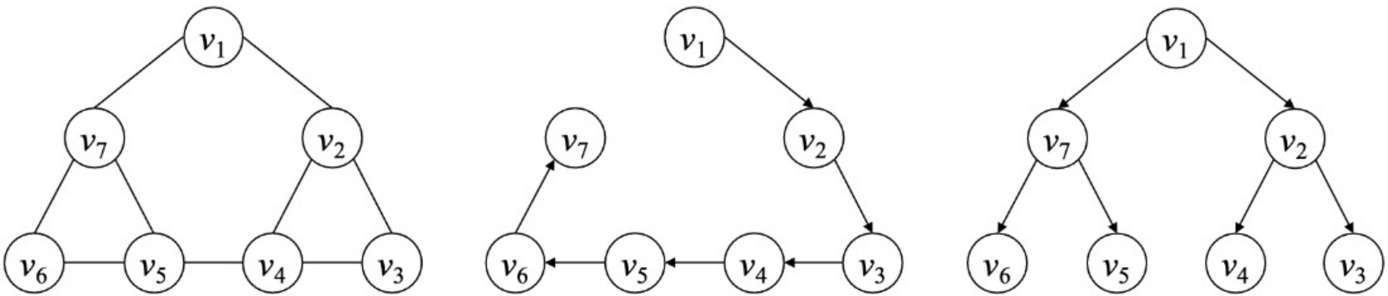
(a) 顶点的属性值



(b) 树边（实线）和其它边（虚线）

距离和BFS

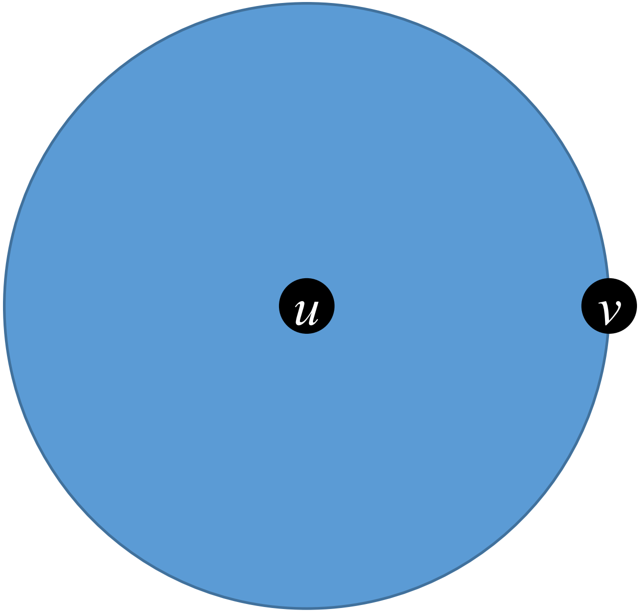
- DFS树和BFS树的对比



(a) 一个图　　(b) 一棵 DFS 树　　(c) 一棵 BFS 树

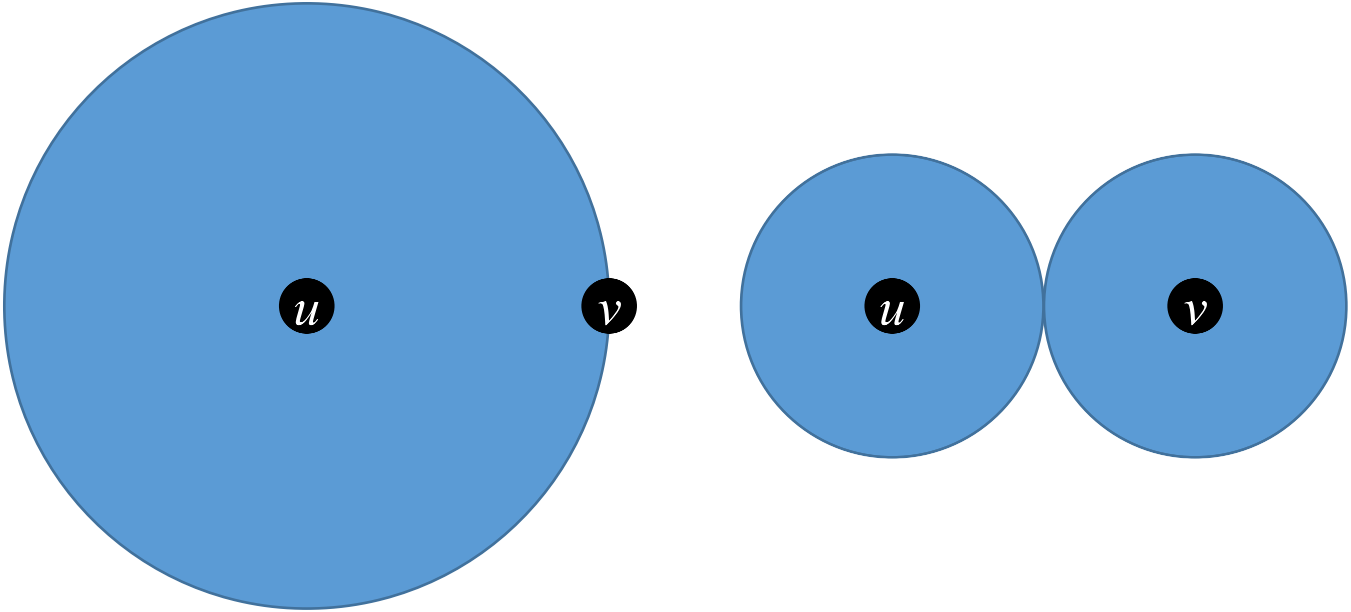
距离和BFS

- 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、2.8、2.11

实战应用

若间接认识也算认识，则两个人互相“认识”可以如下递归定义：他们互相直接认识，或者认识同一个人。基于该定义，回答下述问题。

练习 2.1. n 位同学互相认识，其中至少有几对同学互相直接认识？

练习 2.2. $2n+1$ 位同学中，每位同学至少直接认识 n 位其他同学，这 $2n+1$ 位同学互相认识吗？

练习 2.3. n 位同学中，只有 2 位同学直接认识的其他同学数量是奇数，这 2 位同学互相认识吗？

实战应用

基于练习 2.1–2.3对“认识”的定义，回答下述问题。

练习 2.8. n 位同学互相认识但非全部互相直接认识 $(n \geq 3)$，能否找出 *3* 位同学，其中 *2* 对互相直接认识，*1* 对互相不直接认识？

实战应用

某国的任意两座城市间，或者开通了直飞航线，或者开通了直达班车，但不兼有两者。请证明或否定下述结论。

练习 2.11. 若存在两座城市，无法通过至多一次转机到达，则任意两座城市间，可通过至多两次转车到达。

练习 2.12. 若存在两座城市，无法通过至多两次转机到达，则任意两座城市间，可通过至多一次转车到达。

练习 2.13. 若对于每座城市，都存在另一座无法通过至多一次转机到达的城市，则存在一座城市，到其它每座城市，都可通过至多一次转车到达。