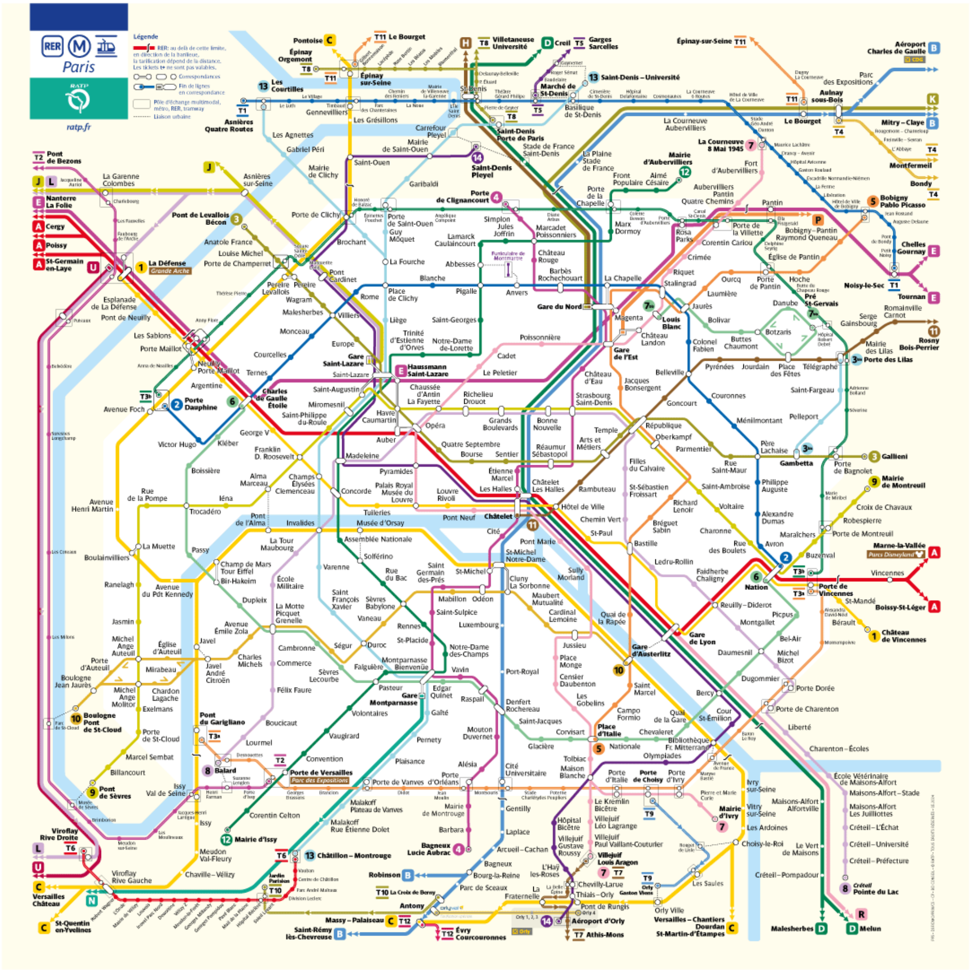
# 第2章 连通和遍历

程龚
南京大学 计算机科学与技术系
gcheng@nju.edu.cn
http://ws.nju.edu.cn/~gcheng

# 本章内容

# 如何判定两个顶点是否连通？如何判定一个图是否连通？

- 从一个站点到另一个站点存在路线吗？



https://www.ratp.fr/plan-de-ligne/img/metro/Plan-Metro.1722015166.png

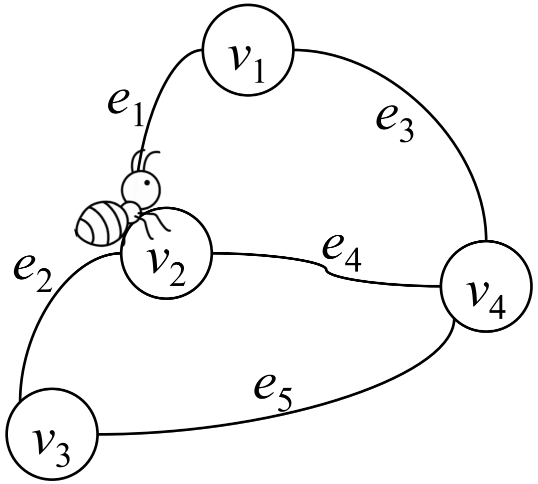
# 如何判定两个顶点是否连通？如何判定一个图是否连通？

- 从一个站点到另一个站点存在路线吗？

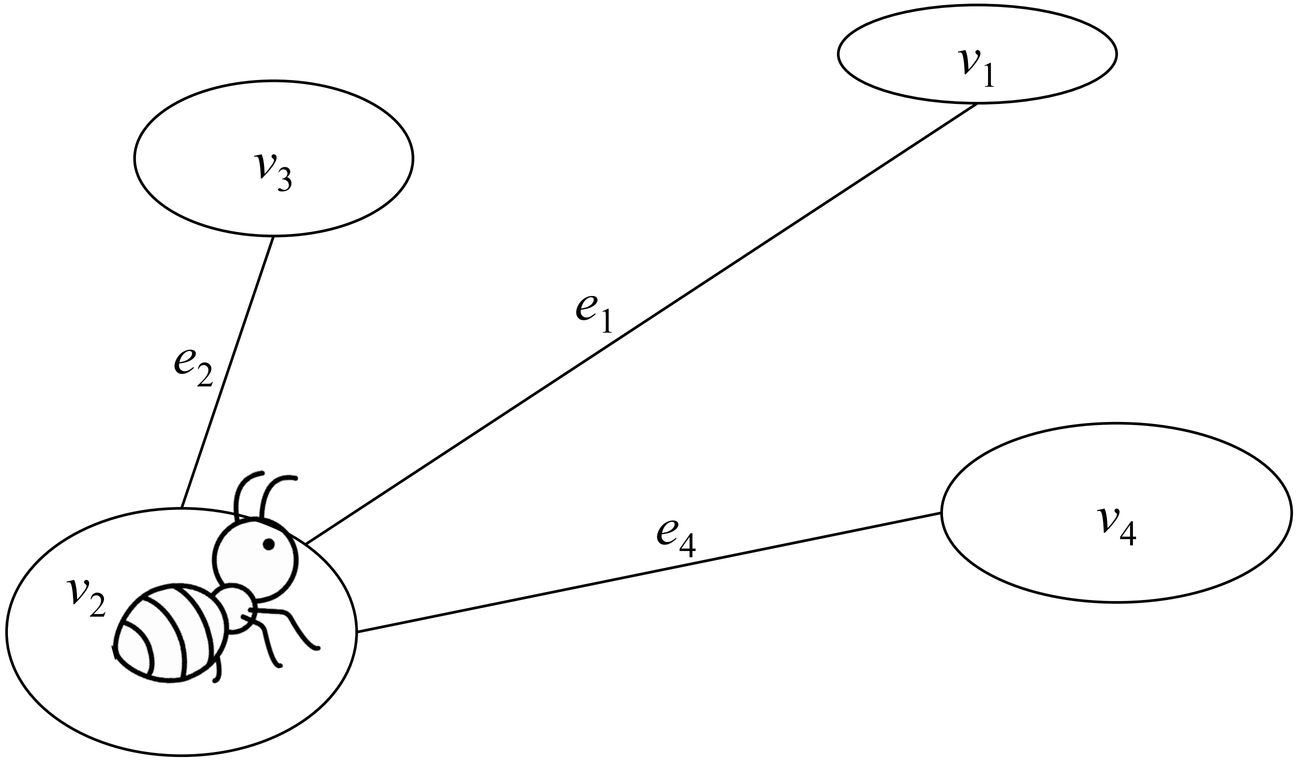

https://en.wikipedia.org/wiki/Paris_M%C3%A9tro#/media/File:Station_M%C3%A9tro_Jaur%C3%A8s_Ligne_2_-_Paris_X_(FR75)_-_2022-04-28_-_8.jpg

# 如何判定两个顶点是否连通？如何判定一个图是否连通？

- 从一个顶点到另一个顶点存在路线吗？




上帝视角




蚂蚁视角

# DFS算法（深度优先搜索算法）

- 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
  - 深度优先：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 算法伪代码

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

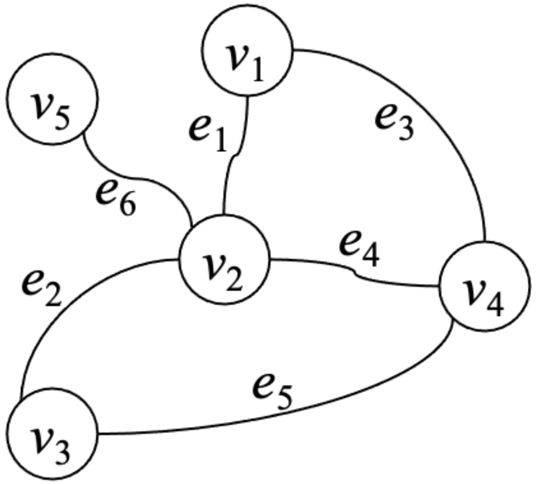
**初值:** 顶点集 $V$ 中所有顶点的 visited 初值为 false

$u$.visited $\leftarrow$ true;

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

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

        DFS($G$, $v$);



# DFS算法

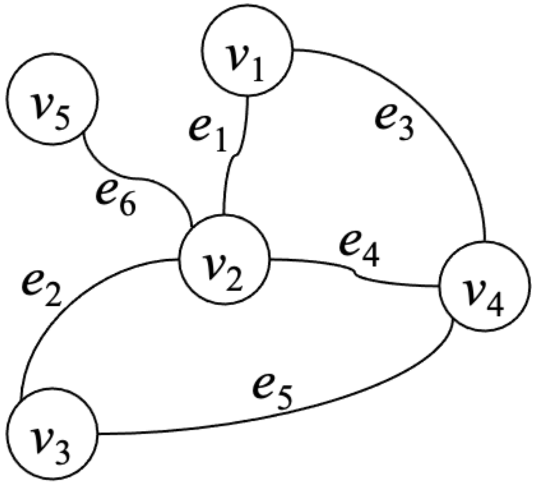
- 每个顶点的visited属性：布尔型变量，表示该顶点是否被访问过

**算法 2.1:** DFS 算法伪代码

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

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

$u$.visited ← true;
**foreach** $(u, v) \in E$ **do**
    **if** $v$.visited = false **then**
        DFS($G$, $v$);



# DFS算法

- 基本思路：通过递归调用，访问每个未被访问过的邻点

**算法 2.1:** DFS 算法伪代码

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

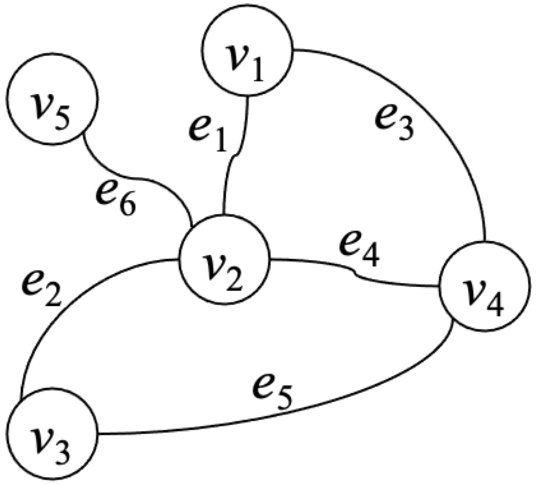
**初值:** 顶点集 $V$ 中所有顶点的 visited 初值为 false

$u$.visited $\leftarrow$ true;

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

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

    DFS($G$, $v$);



# DFS算法

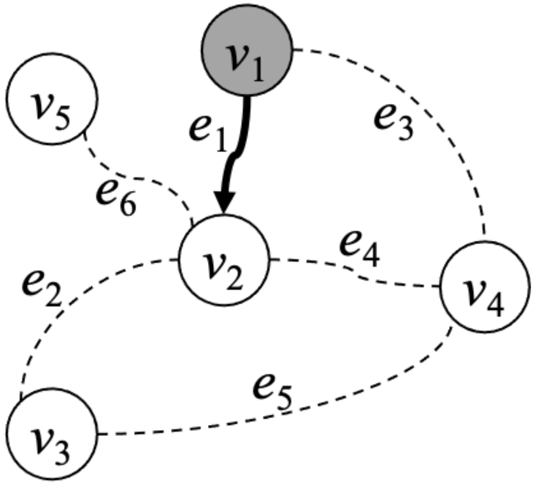
- 例如：从$v_1$出发，调用DFS($G$, $v_1$)
  - $v_1$.visited ← true
  - 判断$v_1$的邻点$v_2$.visited为false，递归调用DFS($G$, $v_2$)

**算法 2.1:** DFS 算法伪代码

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

**初值:** 顶点集 $V$ 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);
```




白色顶点：DFS未调用
灰色顶点：DFS已调用未结束
黑色顶点：DFS调用已结束

虚线：未发生DFS调用的相邻顶点
粗实线箭头：下一步DFS调用
细实线箭头：已发生的DFS调用

# DFS算法

- 递归调用DFS($G$, $v_2$)
  - $v_2$.visited ← true
  - 判断$v_2$的邻点$v_1$.visited为true
  - 判断$v_2$的邻点$v_3$.visited为false，递归调用DFS($G$, $v_3$)

---

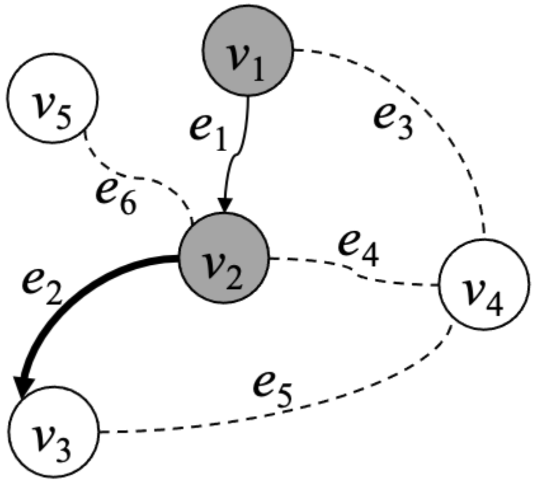
**算法 2.1:** DFS 算法伪代码

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

**初值:** 顶点集 $V$ 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);
```

---




白色顶点：DFS未调用
灰色顶点：DFS已调用未结束
黑色顶点：DFS调用已结束

虚线：未发生DFS调用的相邻顶点
粗实线箭头：下一步DFS调用
细实线箭头：已发生的DFS调用

# DFS算法

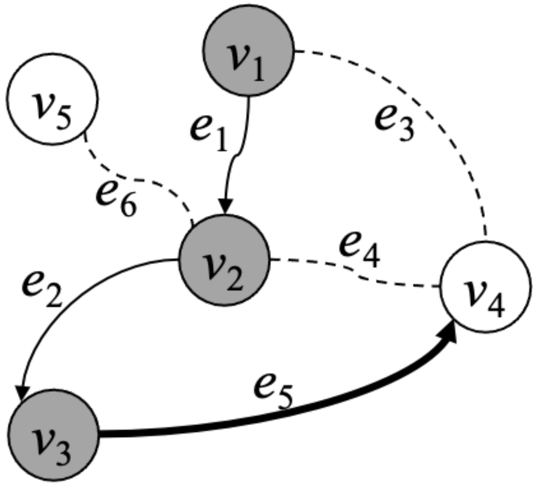
- 递归调用DFS($G$, $v_3$)
  - $v_3$.visited ← true
  - 判断$v_3$的邻点$v_2$.visited为true
  - 判断$v_3$的邻点$v_4$.visited为false，递归调用DFS($G$, $v_4$)

**算法 2.1:** DFS 算法伪代码

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

**初值:** 顶点集 $V$ 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);
```




白色顶点：DFS未调用
灰色顶点：DFS已调用未结束
黑色顶点：DFS调用已结束

虚线：未发生DFS调用的相邻顶点
粗实线箭头：下一步DFS调用
细实线箭头：已发生的DFS调用

# DFS算法

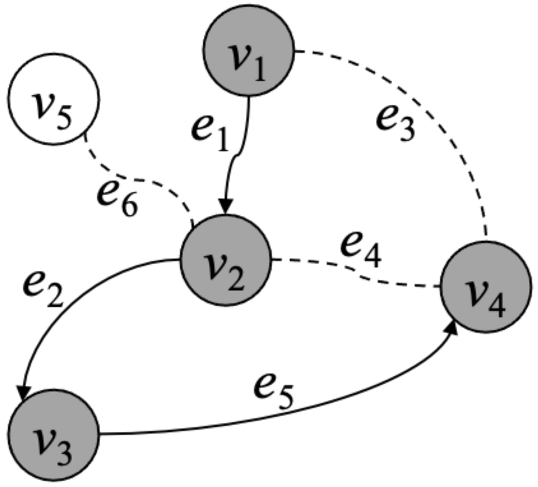
- 递归调用DFS($G$, $v_4$)
  - $v_4$.visited ← true
  - 判断$v_4$的邻点$v_1$.visited为true
  - 判断$v_4$的邻点$v_2$.visited为true
  - 判断$v_4$的邻点$v_3$.visited为true

**算法 2.1:** DFS 算法伪代码

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

**初值:** 顶点集 $V$ 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);
```




白色顶点：DFS未调用
灰色顶点：DFS已调用未结束
黑色顶点：DFS调用已结束

虚线：未发生DFS调用的相邻顶点
粗实线箭头：下一步DFS调用
细实线箭头：已发生的DFS调用

# DFS算法

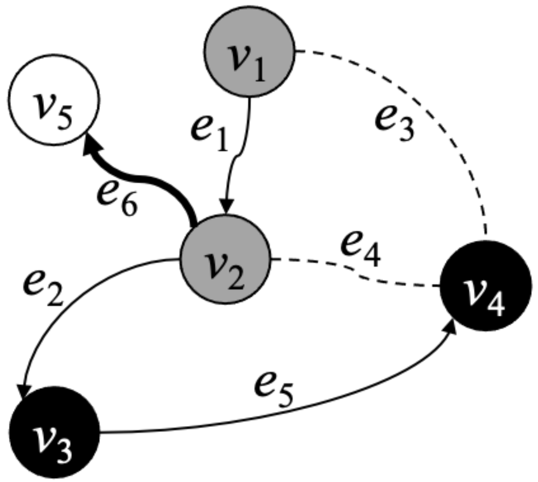
- DFS($G$, $v_4$)结束
- DFS($G$, $v_3$)结束
- DFS($G$, $v_2$)尚未结束
  - 判断$v_2$的邻点$v_4$.visited为true
  - 判断$v_2$的邻点$v_5$.visited为false，递归调用DFS($G$, $v_5$)

**算法 2.1:** DFS 算法伪代码

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

**初值:** 顶点集 $V$ 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);
```




白色顶点：DFS未调用
灰色顶点：DFS已调用未结束
黑色顶点：DFS调用已结束

虚线：未发生DFS调用的相邻顶点
粗实线箭头：下一步DFS调用
细实线箭头：已发生的DFS调用

# DFS算法

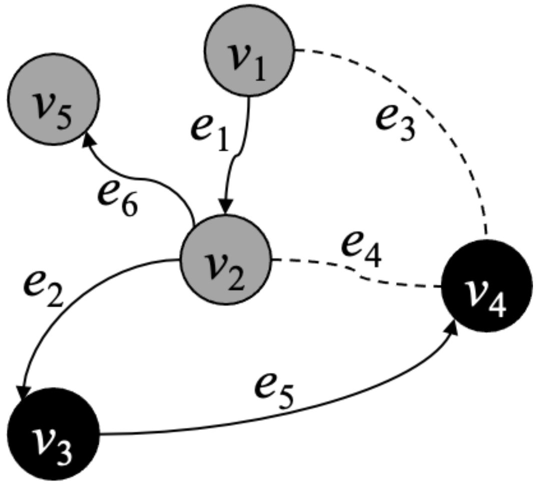
- 递归调用DFS($G$, $v_5$)
  - $v_5$.visited ← true
  - 判断$v_5$的邻点$v_2$.visited为true

**算法 2.1:** DFS 算法伪代码

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

**初值:** 顶点集 $V$ 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);
```




白色顶点：DFS未调用
灰色顶点：DFS已调用未结束
黑色顶点：DFS调用已结束

虚线：未发生DFS调用的相邻顶点
粗实线箭头：下一步DFS调用
细实线箭头：已发生的DFS调用

# DFS算法

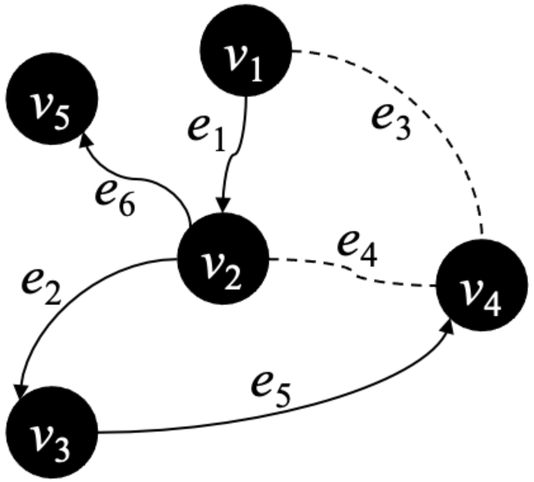
- DFS($G$, $v_5$)结束
- DFS($G$, $v_2$)结束
- DFS($G$, $v_1$)尚未结束
  - 判断$v_1$的邻点$v_4$.visited为true
- DFS($G$, $v_1$)结束

**算法 2.1:** DFS 算法伪代码

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

**初值:** 顶点集 $V$ 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);
```




白色顶点：DFS未调用
灰色顶点：DFS已调用未结束
黑色顶点：DFS调用已结束

虚线：未发生DFS调用的相邻顶点
粗实线箭头：下一步DFS调用
细实线箭头：已发生的DFS调用

# DFS算法

- 不同次运行DFS算法，各顶点的访问顺序可能不同
  - 取决于邻点的访问顺序

**算法 2.1:** DFS 算法伪代码

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

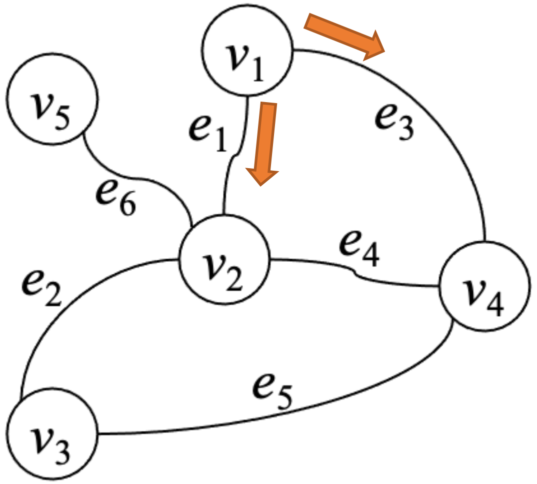
**初值:** 顶点集 $V$ 中所有顶点的 visited 初值为 false

$u$.visited $\leftarrow$ true;

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

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

    DFS($G$, $v$);



# 定理2.2

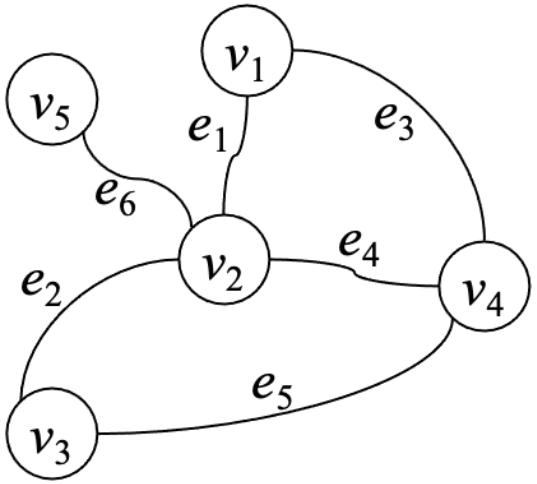
- 从顶点$u$出发运行DFS算法，恰能访问与$u$连通的所有顶点。

**算法 2.1:** DFS 算法伪代码

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

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

$u$.visited $\leftarrow$ true;<br>
**foreach** $(u, v) \in E$ **do**<br>
    **if** $v$.visited $=$ false **then**<br>
        DFS($G$, $v$);



# 定理2.2

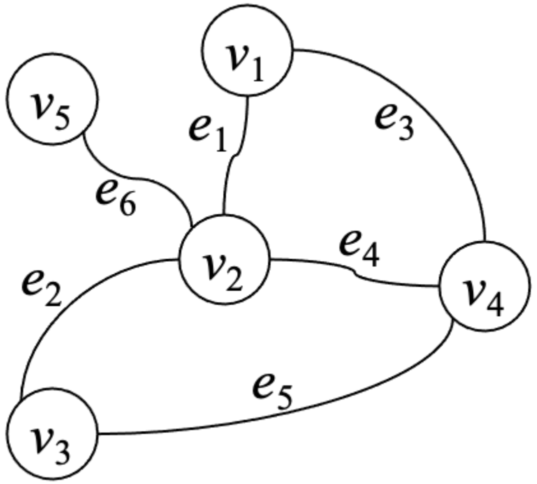
- 从顶点$u$出发运行DFS算法，恰能访问与$u$连通的所有顶点。
  - 与$u$连通的所有顶点都被访问过：
  - 被访问过的所有顶点都与$u$连通：

**算法 2.1:** DFS 算法伪代码

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

**初值:** 顶点集 $V$ 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);
```



# 定理2.2

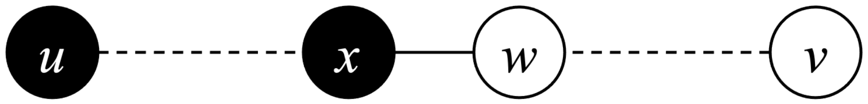
- 从顶点$u$出发运行DFS算法，恰能访问与$u$连通的所有顶点。
  - 与$u$连通的所有顶点都被访问过：
    采用反证法
    - 假设与$u$连通的顶点$v$未被访问过
    - 则任取一条$u$-$v$路$P$，其经过的第一个未被访问过的顶点记作$w$（可能是$v$）
    - 根据DFS算法，$u$被访问过，因此，$w \neq u$，即$w$不是$P$经过的第一个顶点
    - $P$经过的$w$的前一个邻点$x$（可能是$u$）被访问过
    - 根据DFS算法，$w$也应被访问过，矛盾
  - 被访问过的所有顶点都与$u$连通：

**算法 2.1:** DFS 算法伪代码

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

**初值:** 顶点集 $V$ 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);
```

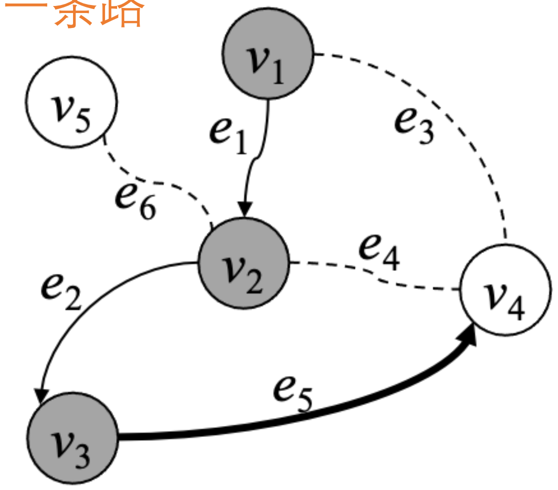


# 定理2.2

- 从顶点$u$出发运行DFS算法，恰能访问与$u$连通的所有顶点。
  - 与$u$连通的所有顶点都被访问过：<br>采用反证法
    - 假设与$u$连通的顶点$v$未被访问过
    - 则任取一条$u$-$v$路$P$，其经过的第一个未被访问过的顶点记作$w$（可能是$v$）
    - 根据DFS算法，$u$被访问过，因此，$w \neq u$，即w不是$P$经过的第一个顶点
    - $P$经过的$w$的前一个邻点$x$（可能是$u$）被访问过
    - 根据DFS算法，$w$也应被访问过，矛盾
  - 被访问过的所有顶点都与$u$连通：递归调用链对应一条路

**算法 2.1:** DFS 算法伪代码

```
输入: 图 G = ⟨V, E⟩, 顶点 u
初值: 顶点集 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);
```



# DFS算法

- 时间复杂度：$O(n+m)$

**算法 2.1:** DFS 算法伪代码

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

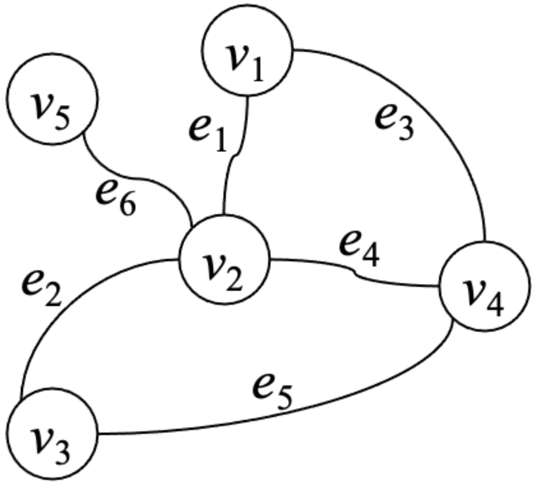
**初值:** 顶点集 $V$ 中所有顶点的 visited 初值为 false

$u$.visited $\leftarrow$ true;

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

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

        DFS($G$, $v$);



# DFS算法

- 利用DFS算法判定顶点$u$和$v$是否连通：
- 利用DFS算法判定图$G$是否连通：

**算法 2.1:** DFS 算法伪代码

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

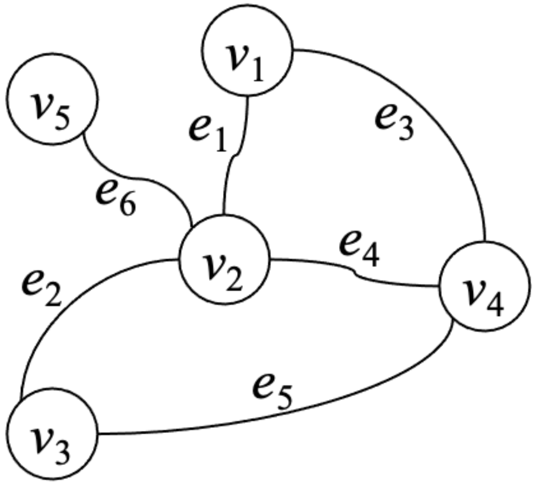
**初值:** 顶点集 $V$ 中所有顶点的 visited 初值为 false

$u$.visited $\leftarrow$ true;

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

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

        DFS($G$, $v$);



# DFS算法

- 利用DFS算法判定顶点$u$和$v$是否连通：
  - 从$u$出发运行一次算法，若访问过$v$，则$u$和$v$连通，否则不连通
- 利用DFS算法判定图$G$是否连通：

**算法 2.1:** DFS 算法伪代码

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

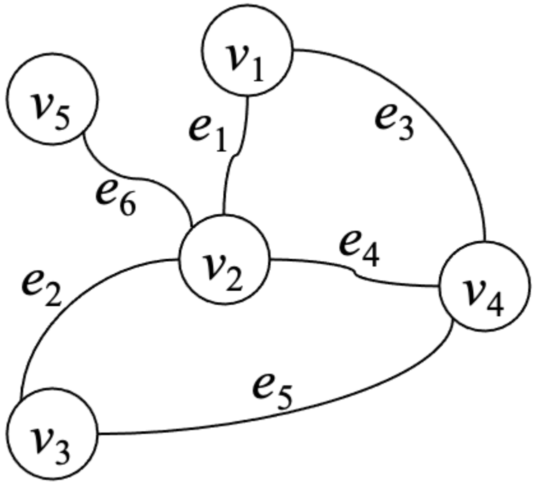
**初值:** 顶点集 $V$ 中所有顶点的 visited 初值为 false

$u$.visited $\leftarrow$ true;

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

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

    DFS($G$, $v$);

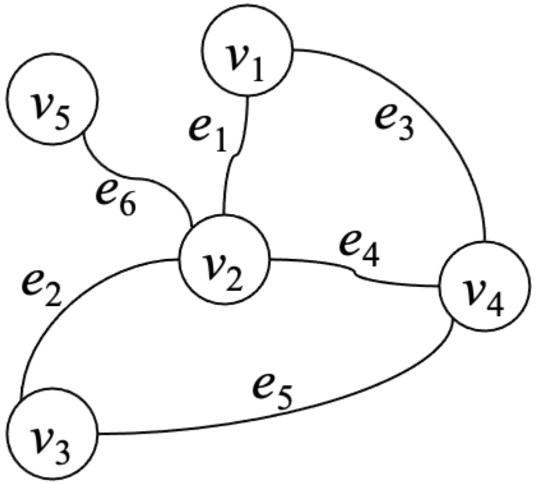


# DFS算法

- 利用DFS算法判定顶点$u$和$v$是否连通：
  - 从$u$出发运行一次算法，若访问过$v$，则$u$和$v$连通，否则不连通
- 利用DFS算法判定图$G$是否连通：
  - 从任意一个顶点出发运行一次算法，若访问过图中所有顶点，则$G$连通，否则不连通

**算法 2.1:** DFS 算法伪代码

```
输入: 图 G = ⟨V, E⟩，顶点 u
初值: 顶点集 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);
```



请认真完成课后练习