南京大学 NANJING UNIVERSITY

Websoft

第2章 连通和遍历

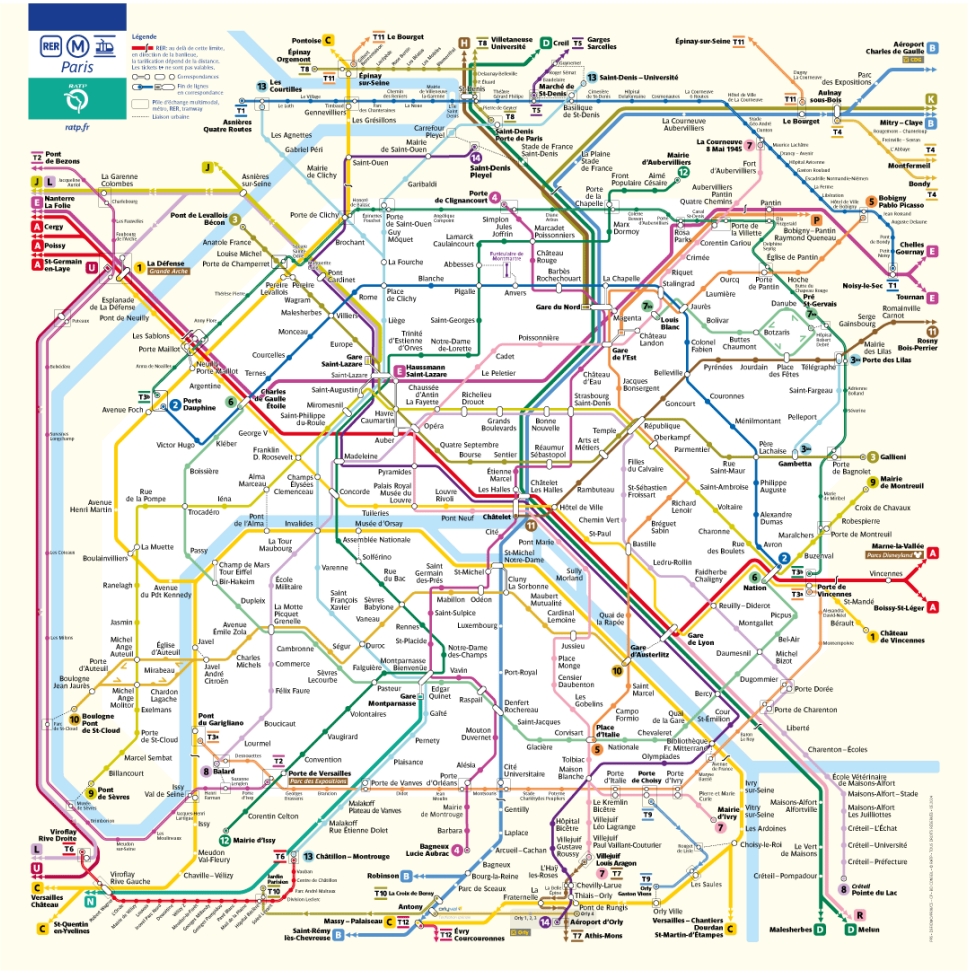
程龚
南京大学 计算机科学与技术系
gcheng@nju.edu.cn
http://ws.nju.edu.cn/~gcheng

本章内容

- 第2.1节 连通和DFS
- 第2.2节 割点和割边
  - 第2.2.1节 理论
  - **第2.2.2节 算法**
- 第2.3节 距离和BFS

如何判定一个顶点是否为割点？如何找出图中所有割点？

- 这座地铁站无法绕过吗？
  哪些地铁站无法绕过？



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

DFSCV算法（扩展DFS算法用于找割点）

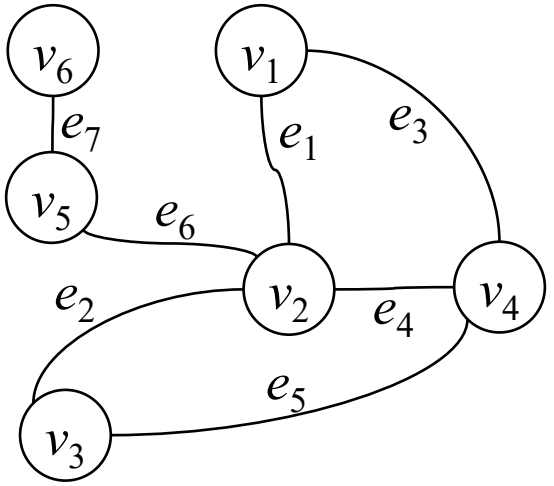
- 从连通图中任意一个指定顶点出发，按DFS的方式有序地遍历所有顶点，并找出所有割点

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11         if u.parent = null 且 u.children ⩾ 2 then
12             u.isCutVertex ← true;
13         else if u.parent ≠ null 且 v.low ⩾ u.d then
14             u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

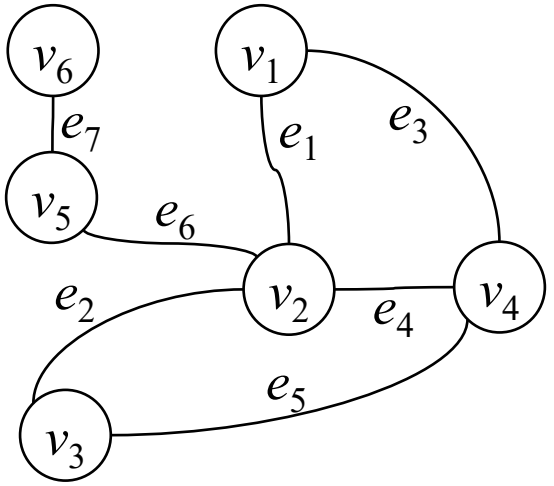
- 基于DFS算法

算法 2.2: DFSCV 算法伪代码

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

初值: 顶点集 V 中所有顶点的 visited 初值为 false，

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



DFSCV算法

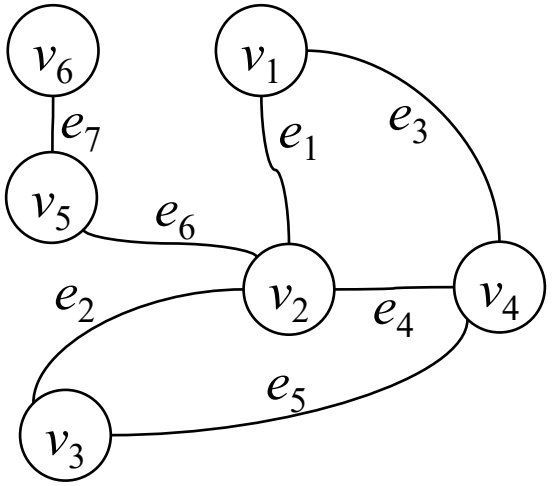
- 第1项扩展：记录每个顶点被访问的次序
  - 每个顶点的d属性：整数型变量，表示该顶点被访问的次序
  - time变量：整数型变量，初值为0，每访问一个顶点便将time加1

算法 2.2: DFSCV 算法伪代码

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

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

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



DFSCV算法

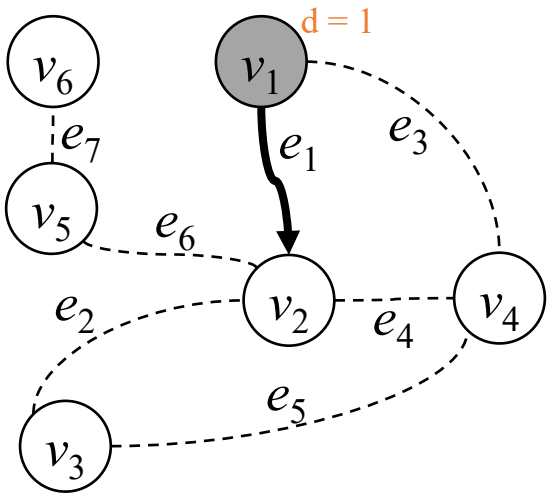
- 例如：从v_1出发，调用DFSCV(G, v_1)
  - time ← 1
  - v_1.d ← 1

算法 2.2: DFSCV 算法伪代码

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

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

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



DFSCV算法

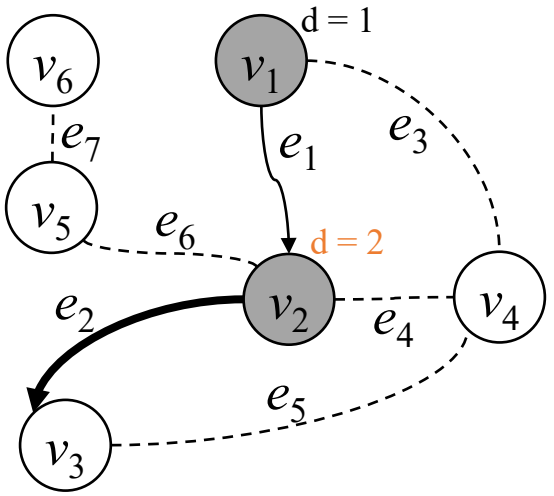
- 递归调用DFSCV(G, v_2)
  - time ← 2
  - v_2.d ← 2

算法 2.2: DFSCV 算法伪代码

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

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

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



DFSCV算法

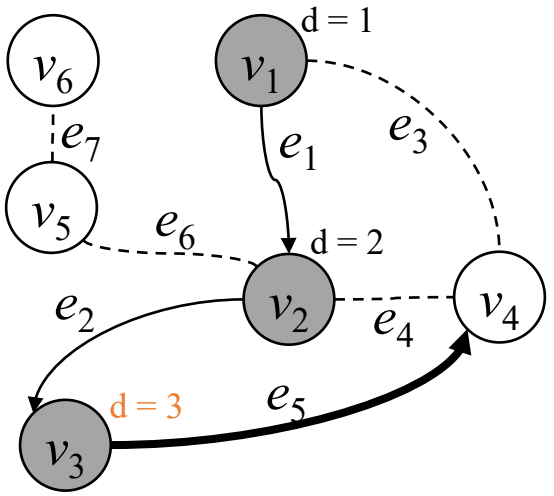
- 递归调用DFSCV(G, v_3)
  - time ← 3
  - v_3.d ← 3

算法 2.2: DFSCV 算法伪代码

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

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

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



DFSCV算法

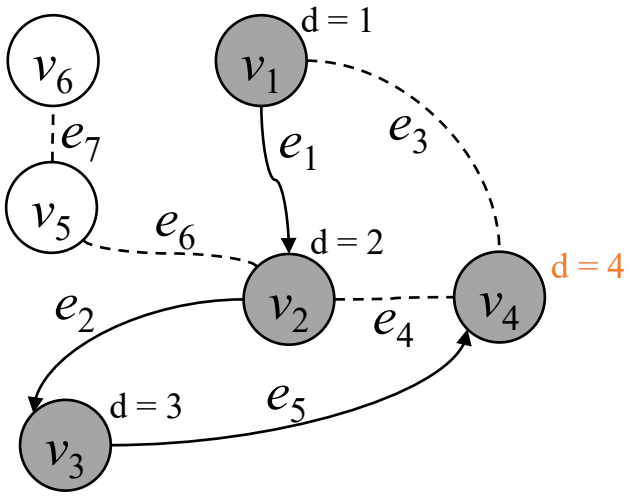
- 递归调用DFSCV(G, v_4)
  - time ← 4
  - v_4.d ← 4

算法 2.2: DFSCV 算法伪代码

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

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

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



DFSCV算法

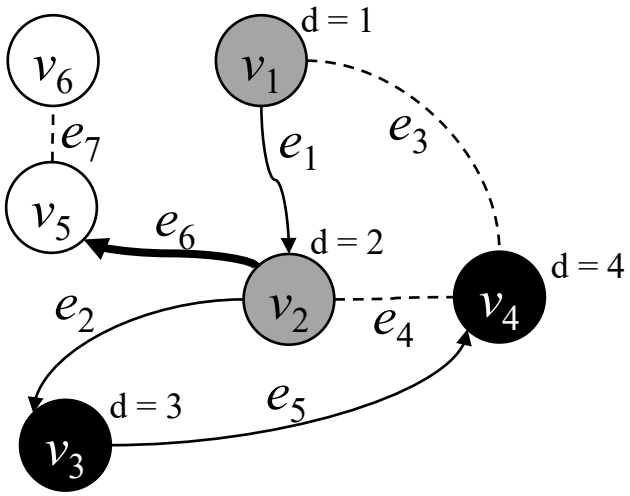
- DFSCV(G, v_4)结束
- DFSCV(G, v_3)结束

算法 2.2: DFSCV 算法伪代码

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

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

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



DFSCV算法

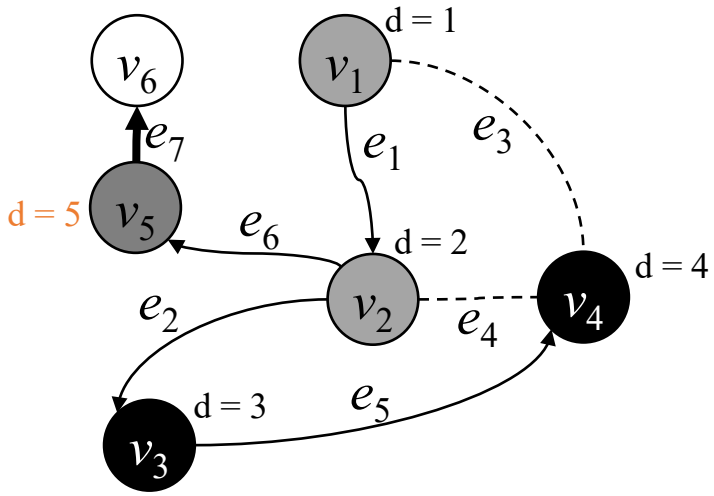
- 递归调用DFSCV(G, v_5)
  - time ← 5
  - v_5.d ← 5

算法 2.2: DFSCV 算法伪代码

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

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

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



DFSCV算法

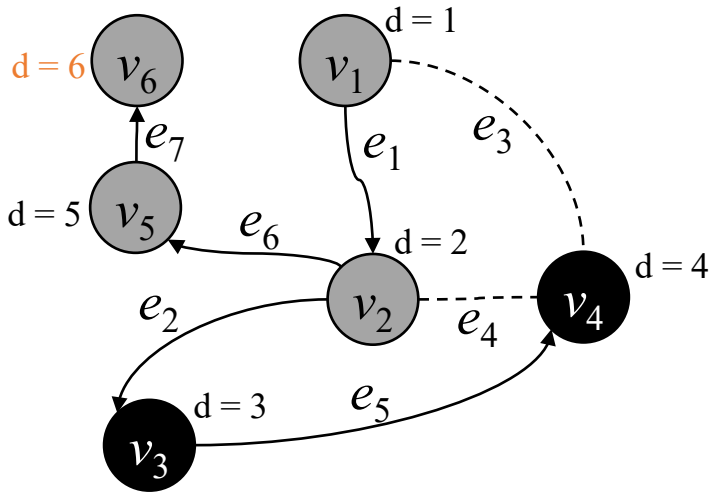
- 递归调用DFSCV(G, v_6)
  - time ← 6
  - v_6.d ← 6

算法 2.2: DFSCV 算法伪代码

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

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

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



DFSCV算法

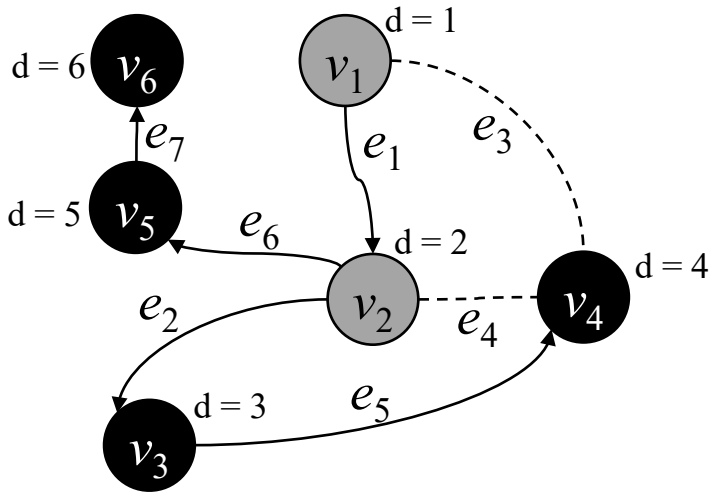
- DFSCV(G, v_6)结束
- DFSCV(G, v_5)结束

算法 2.2: DFSCV 算法伪代码

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

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

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



DFSCV算法

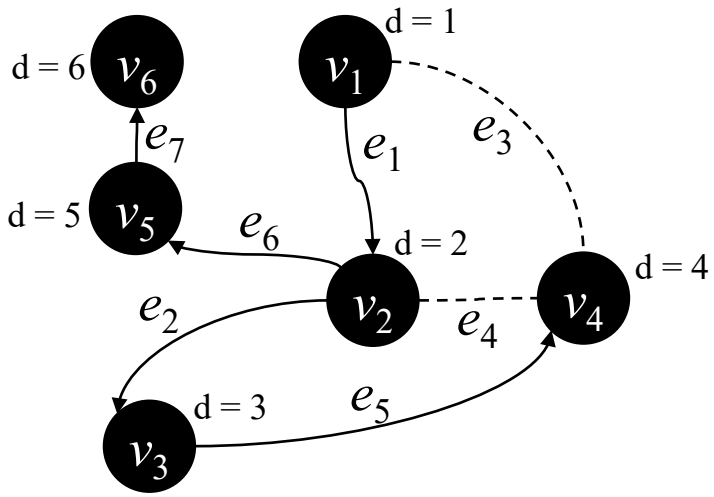
- DFSCV(G, v_2)结束
- DFSCV(G, v_1)结束

算法 2.2: DFSCV 算法伪代码

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

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

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



DFSCV算法

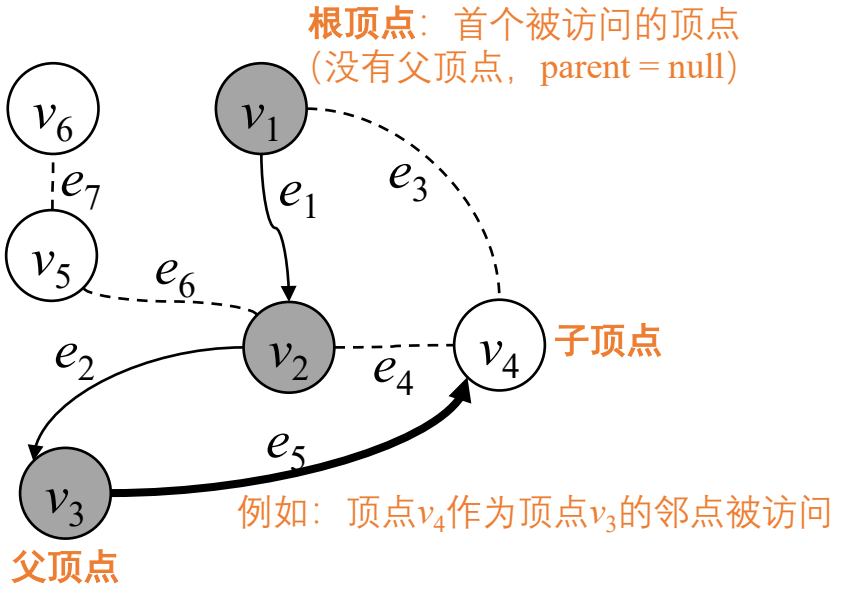
- 第1项扩展：记录每个顶点被访问的次序，以及它的父顶点
  - 每个顶点的parent属性：顶点型变量，初值为null，表示该顶点的父顶点
  - 每个顶点的children属性：整数型变量，初值为0，表示该顶点的子顶点数量

算法 2.2: DFSCV 算法伪代码

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

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

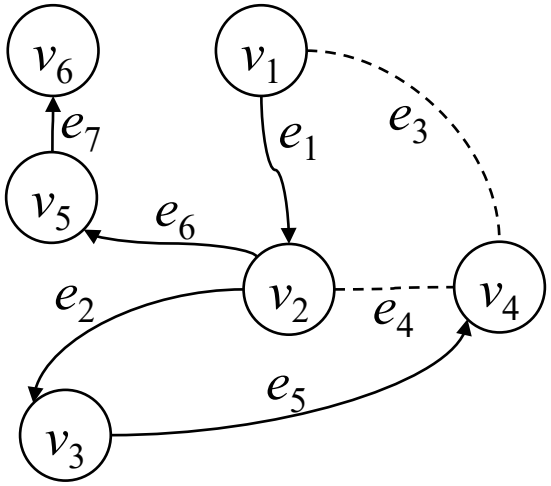
```
1  time ← time + 1;
2  u.d ← time;
3
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10
11
12
13
14
15
16
```



DFSCV算法

- 第1项扩展：记录每个顶点被访问的次序，以及它的父顶点
- **DFS树**：所有树边的边导出子图
  - **树边**：从父顶点访问子顶点经过的边
  - **后向边**：其它边

顶点	d	parent	chidren
v_1	1	null	1
v_2	2	v_1	2
v_3	3	v_2	1
v_4	4	v_3	0
v_5	5	v_2	1
v_6	6	v_5	0

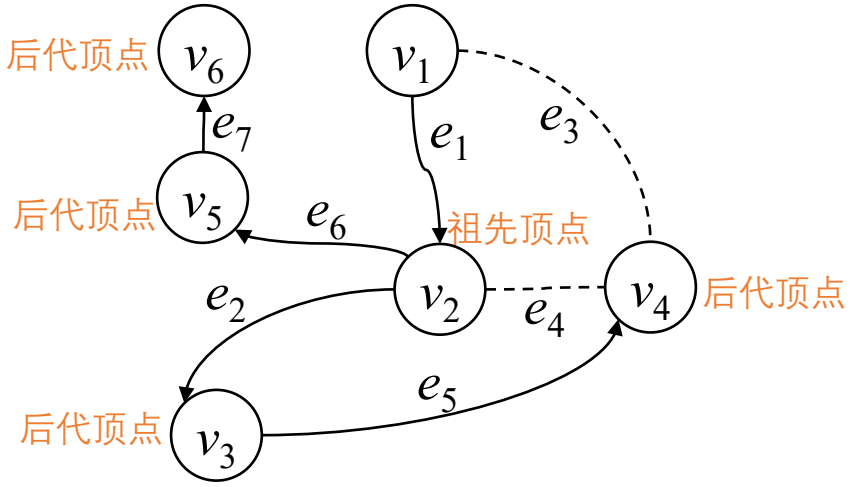


实线箭头：树边
虚线：后向边

DFSCV算法

- 第1项扩展：记录每个顶点被访问的次序，以及它的父顶点
- DFS树：所有树边的边导出子图
  - 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**
  - 例如：v_2是v_3, v_4, v_5, v_6的祖先顶点

顶点	d	parent	chidren
v_1	1	null	1
v_2	2	v_1	2
v_3	3	v_2	1
v_4	4	v_3	0
v_5	5	v_2	1
v_6	6	v_5	0

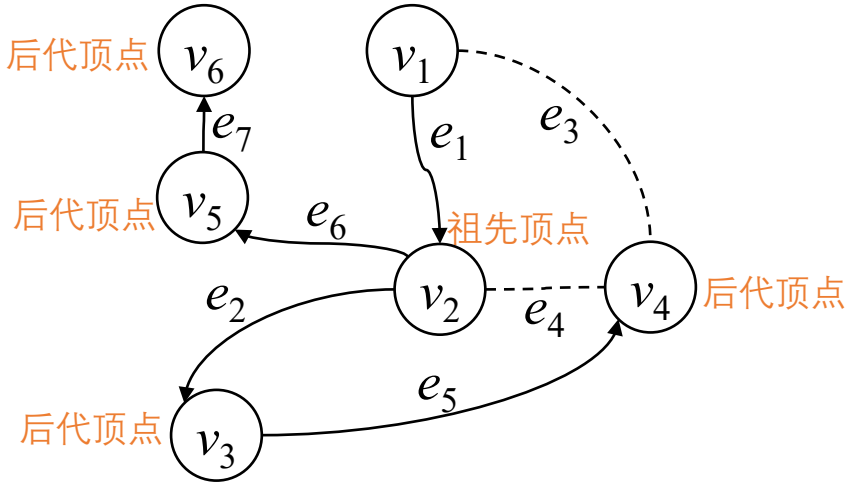


实线箭头：树边
虚线：后向边

DFSCV算法

- 第1项扩展：记录每个顶点被访问的次序，以及它的父顶点
- DFS树：所有树边的边导出子图
  - 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
  - 对祖先顶点的访问早于后代顶点
  - 对祖先顶点的递归调用结束晚于后代顶点
    （DFS树刻画了递归调用关系）

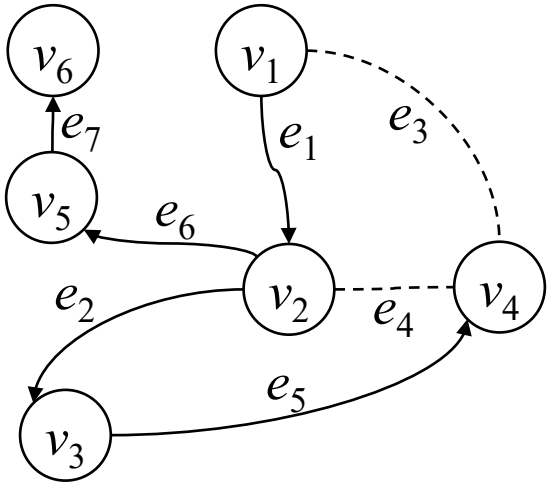
顶点	d	parent	chidren
v_1	1	null	1
v_2	2	v_1	2
v_3	3	v_2	1
v_4	4	v_3	0
v_5	5	v_2	1
v_6	6	v_5	0



实线箭头：树边
虚线：后向边

引理2.1

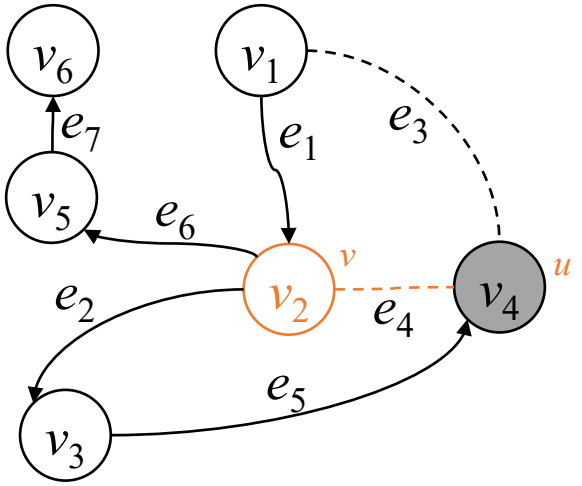
- 后向边关联一对祖先-后代顶点。



实线箭头：树边
虚线：后向边

引理2.1

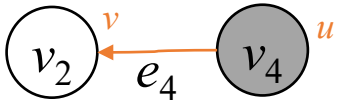
- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:



实线箭头：树边
虚线：后向边

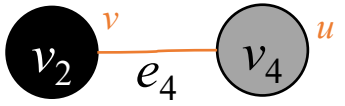
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。



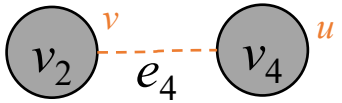
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。
    - 若v的visited属性值为true且对其DFS调用已结束，则对v的DFS调用已访问过边(u, v)，与首次访问矛盾。



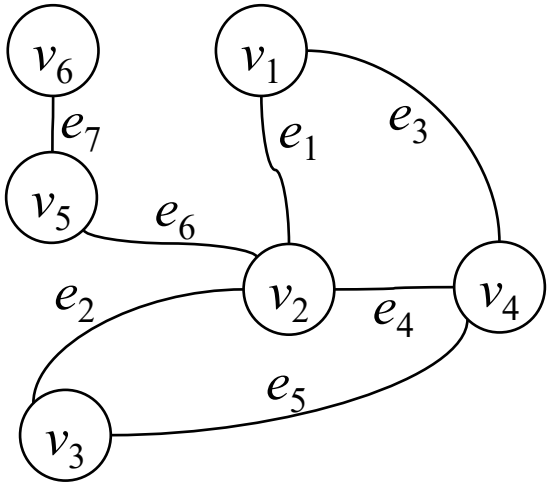
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。
    - 若v的visited属性值为true且对其DFS调用已结束，则对v的DFS调用已访问过边(u, v)，与首次访问矛盾。
    - 因此，v的visited属性值为true且对其DFS调用未结束，
      而DFS算法运行过程中所有visited属性值为true且对其DFS调用未结束的顶点
      都在同一条从根顶点到当前访问顶点的祖先-后代路上，
      因此，v是u的祖先顶点。



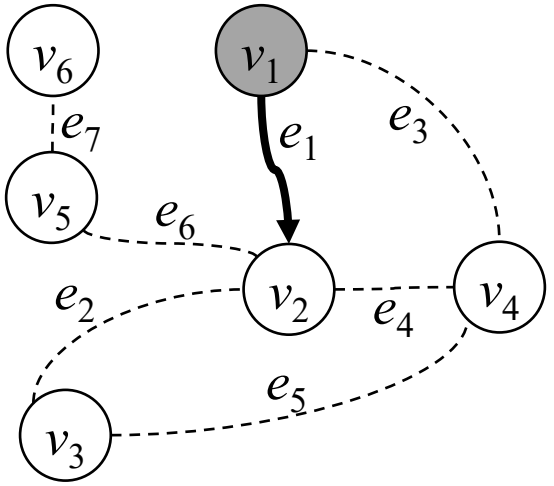
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。
    - 若v的visited属性值为true且对其DFS调用已结束，则对v的DFS调用已访问过边(u, v)，与首次访问矛盾。
    - 因此，v的visited属性值为true且对其DFS调用未结束，
      而DFS算法运行过程中所有visited属性值为true且对其DFS调用未结束的顶点
      都在同一条从根顶点到当前访问顶点的祖先-后代路上，
      因此，v是u的祖先顶点。
      （可以用数学归纳法证明，这里仅举例说明）



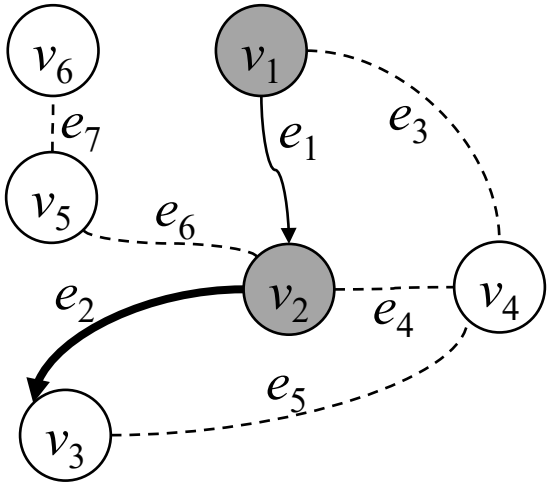
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。
    - 若v的visited属性值为true且对其DFS调用已结束，则对v的DFS调用已访问过边(u, v)，与首次访问矛盾。
    - 因此，v的visited属性值为true且对其DFS调用未结束，
      而DFS算法运行过程中所有visited属性值为true且对其DFS调用未结束的顶点
      都在同一条从根顶点到当前访问顶点的祖先-后代路上，
      因此，v是u的祖先顶点。
      （可以用数学归纳法证明，这里仅举例说明）



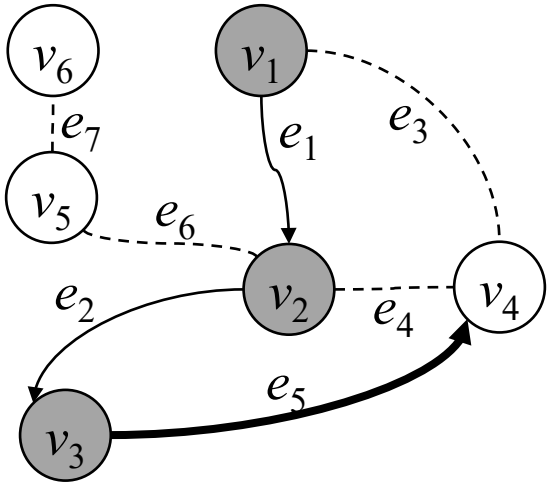
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。
    - 若v的visited属性值为true且对其DFS调用已结束，则对v的DFS调用已访问过边(u, v)，与首次访问矛盾。
    - 因此，v的visited属性值为true且对其DFS调用未结束，
      而DFS算法运行过程中所有visited属性值为true且对其DFS调用未结束的顶点
      都在同一条从根顶点到当前访问顶点的祖先-后代路上，
      因此，v是u的祖先顶点。
      （可以用数学归纳法证明，这里仅举例说明）



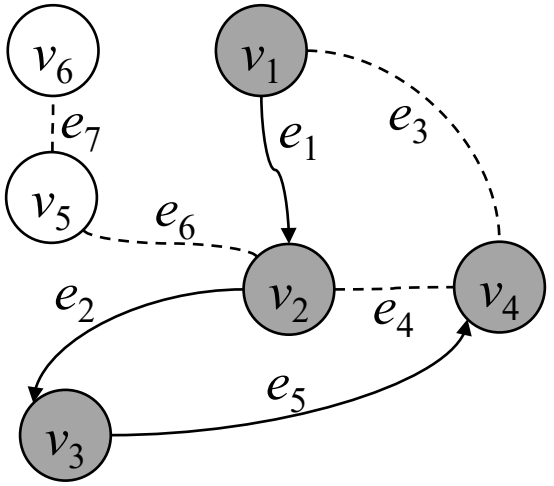
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。
    - 若v的visited属性值为true且对其DFS调用已结束，则对v的DFS调用已访问过边(u, v)，与首次访问矛盾。
    - 因此，v的visited属性值为true且对其DFS调用未结束，
      而DFS算法运行过程中所有visited属性值为true且对其DFS调用未结束的顶点
      都在同一条从根顶点到当前访问顶点的祖先-后代路上，
      因此，v是u的祖先顶点。
      （可以用数学归纳法证明，这里仅举例说明）



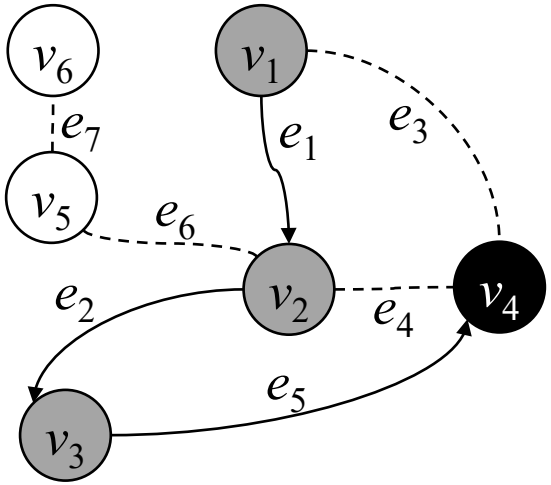
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。
    - 若v的visited属性值为true且对其DFS调用已结束，则对v的DFS调用已访问过边(u, v)，与首次访问矛盾。
    - 因此，v的visited属性值为true且对其DFS调用未结束，
      而DFS算法运行过程中所有visited属性值为true且对其DFS调用未结束的顶点
      都在同一条从根顶点到当前访问顶点的祖先-后代路上，
      因此，v是u的祖先顶点。
      （可以用数学归纳法证明，这里仅举例说明）



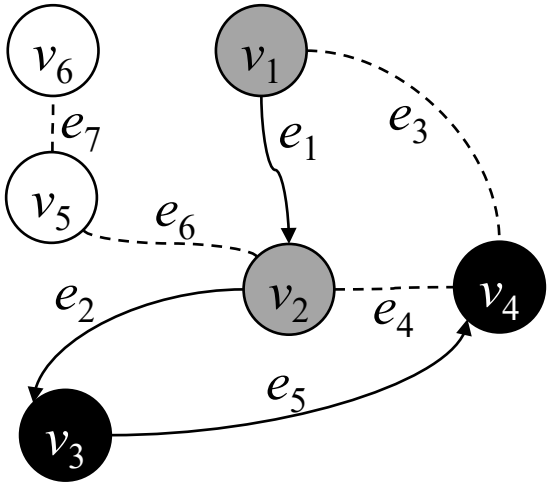
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。
    - 若v的visited属性值为true且对其DFS调用已结束，则对v的DFS调用已访问过边(u, v)，与首次访问矛盾。
    - 因此，v的visited属性值为true且对其DFS调用未结束，
      而DFS算法运行过程中所有visited属性值为true且对其DFS调用未结束的顶点
      都在同一条从根顶点到当前访问顶点的祖先-后代路上，
      因此，v是u的祖先顶点。
      （可以用数学归纳法证明，这里仅举例说明）



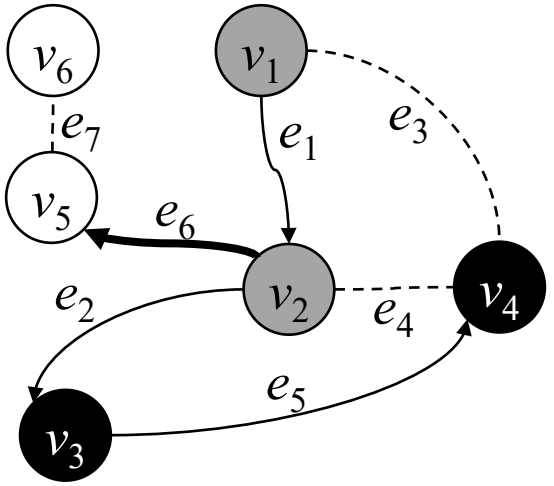
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。
    - 若v的visited属性值为true且对其DFS调用已结束，则对v的DFS调用已访问过边(u, v)，与首次访问矛盾。
    - 因此，v的visited属性值为true且对其DFS调用未结束，
      而DFS算法运行过程中所有visited属性值为true且对其DFS调用未结束的顶点
      都在同一条从根顶点到当前访问顶点的祖先-后代路上，
      因此，v是u的祖先顶点。
      （可以用数学归纳法证明，这里仅举例说明）



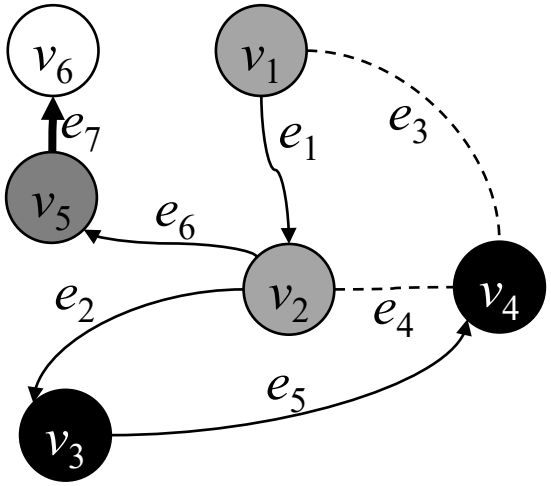
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。
    - 若v的visited属性值为true且对其DFS调用已结束，则对v的DFS调用已访问过边(u, v)，与首次访问矛盾。
    - 因此，v的visited属性值为true且对其DFS调用未结束，
      而DFS算法运行过程中所有visited属性值为true且对其DFS调用未结束的顶点
      都在同一条从根顶点到当前访问顶点的祖先-后代路上，
      因此，v是u的祖先顶点。
      （可以用数学归纳法证明，这里仅举例说明）



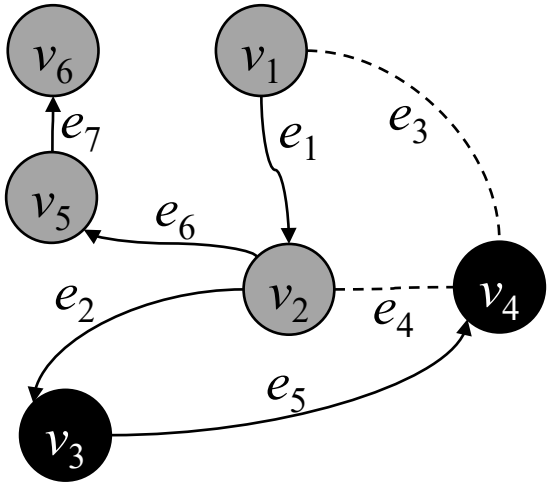
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。
    - 若v的visited属性值为true且对其DFS调用已结束，则对v的DFS调用已访问过边(u, v)，与首次访问矛盾。
    - 因此，v的visited属性值为true且对其DFS调用未结束，
      而DFS算法运行过程中所有visited属性值为true且对其DFS调用未结束的顶点
      都在同一条从根顶点到当前访问顶点的祖先-后代路上，
      因此，v是u的祖先顶点。
      （可以用数学归纳法证明，这里仅举例说明）



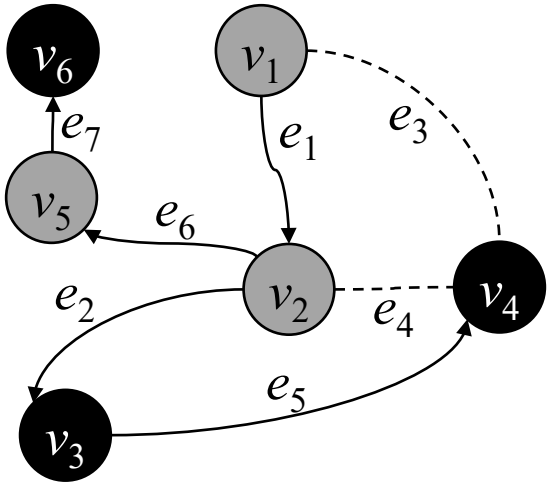
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。
    - 若v的visited属性值为true且对其DFS调用已结束，则对v的DFS调用已访问过边(u, v)，与首次访问矛盾。
    - 因此，v的visited属性值为true且对其DFS调用未结束，
      而DFS算法运行过程中所有visited属性值为true且对其DFS调用未结束的顶点
      都在同一条从根顶点到当前访问顶点的祖先-后代路上，
      因此，v是u的祖先顶点。
      （可以用数学归纳法证明，这里仅举例说明）



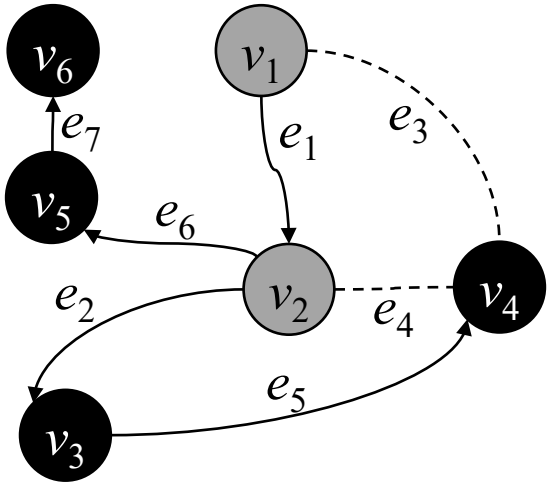
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。
    - 若v的visited属性值为true且对其DFS调用已结束，则对v的DFS调用已访问过边(u, v)，与首次访问矛盾。
    - 因此，v的visited属性值为true且对其DFS调用未结束，
      而DFS算法运行过程中所有visited属性值为true且对其DFS调用未结束的顶点
      都在同一条从根顶点到当前访问顶点的祖先-后代路上，
      因此，v是u的祖先顶点。
      （可以用数学归纳法证明，这里仅举例说明）



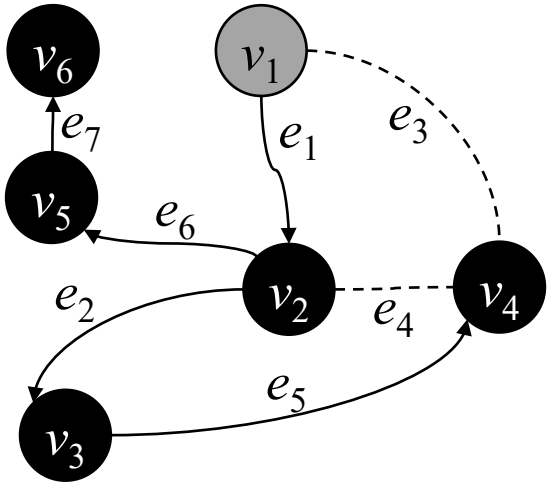
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。
    - 若v的visited属性值为true且对其DFS调用已结束，则对v的DFS调用已访问过边(u, v)，与首次访问矛盾。
    - 因此，v的visited属性值为true且对其DFS调用未结束，
      而DFS算法运行过程中所有visited属性值为true且对其DFS调用未结束的顶点
      都在同一条从根顶点到当前访问顶点的祖先-后代路上，
      因此，v是u的祖先顶点。
      （可以用数学归纳法证明，这里仅举例说明）



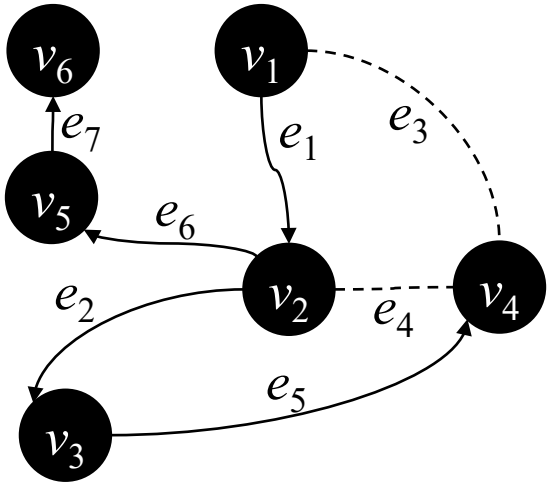
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。
    - 若v的visited属性值为true且对其DFS调用已结束，则对v的DFS调用已访问过边(u, v)，与首次访问矛盾。
    - 因此，v的visited属性值为true且对其DFS调用未结束，
      而DFS算法运行过程中所有visited属性值为true且对其DFS调用未结束的顶点
      都在同一条从根顶点到当前访问顶点的祖先-后代路上，
      因此，v是u的祖先顶点。
      （可以用数学归纳法证明，这里仅举例说明）



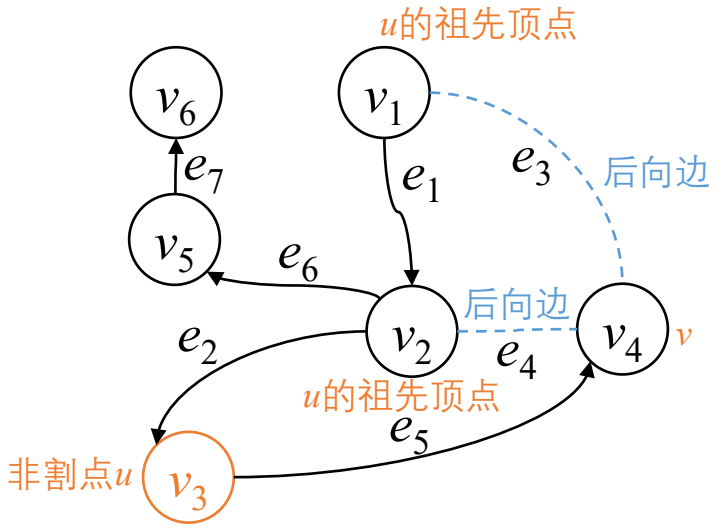
引理2.1

- 后向边关联一对祖先-后代顶点。
  - 不失一般性，假设后向边(u, v)在对顶点u的DFS调用中首次被访问，则分情况讨论顶点v:
    - 若v的visited属性值为false，则边(u, v)是树边，与是后向边矛盾。
    - 若v的visited属性值为true且对其DFS调用已结束，则对v的DFS调用已访问过边(u, v)，与首次访问矛盾。
    - 因此，v的visited属性值为true且对其DFS调用未结束，
      而DFS算法运行过程中所有visited属性值为true且对其DFS调用未结束的顶点
      都在同一条从根顶点到当前访问顶点的祖先-后代路上，
      因此，v是u的祖先顶点。
      （可以用数学归纳法证明，这里仅举例说明）



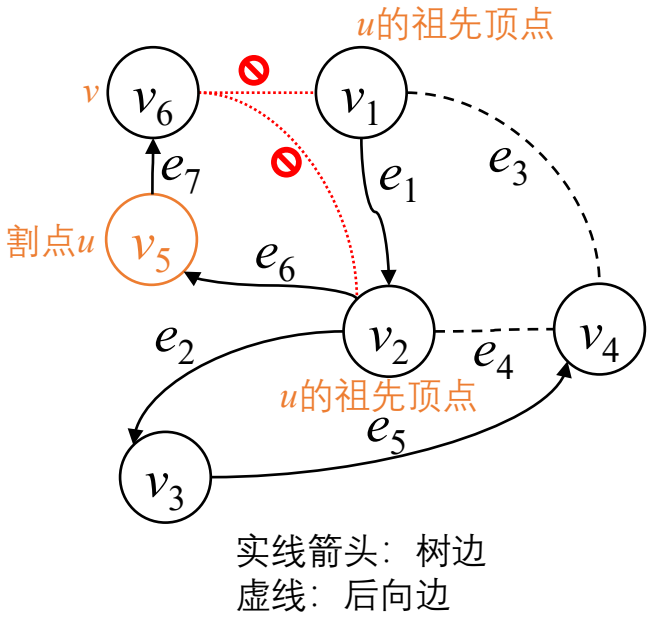
定理2.5（判断非根顶点是否为割点）

- 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。
  - 例如：非根顶点v_3不是割点



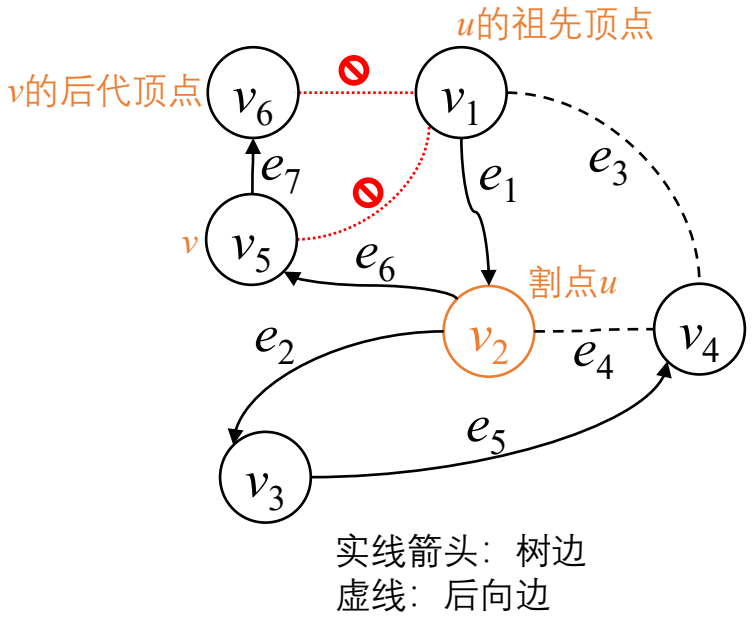
定理2.5（判断非根顶点是否为割点）

- 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。
  - 例如：非根顶点v_5是割点



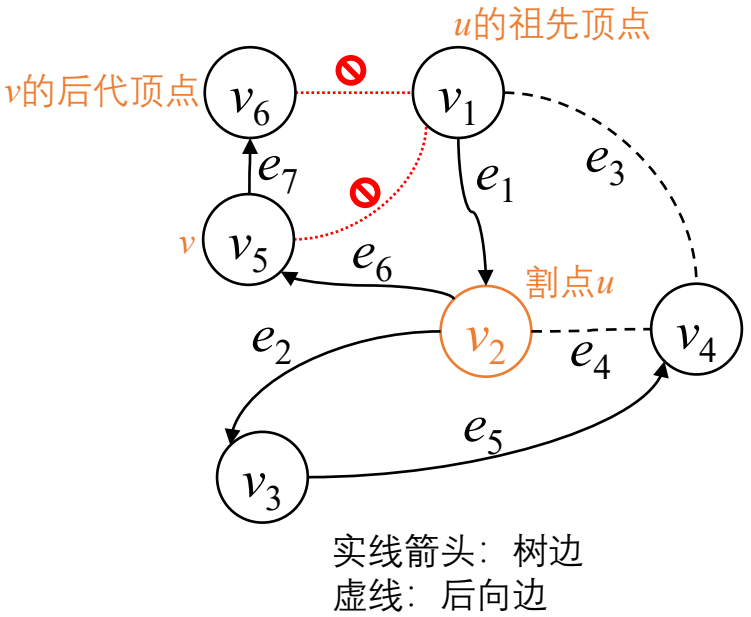
定理2.5（判断非根顶点是否为割点）

- 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。
  - 例如：非根顶点v_2是割点



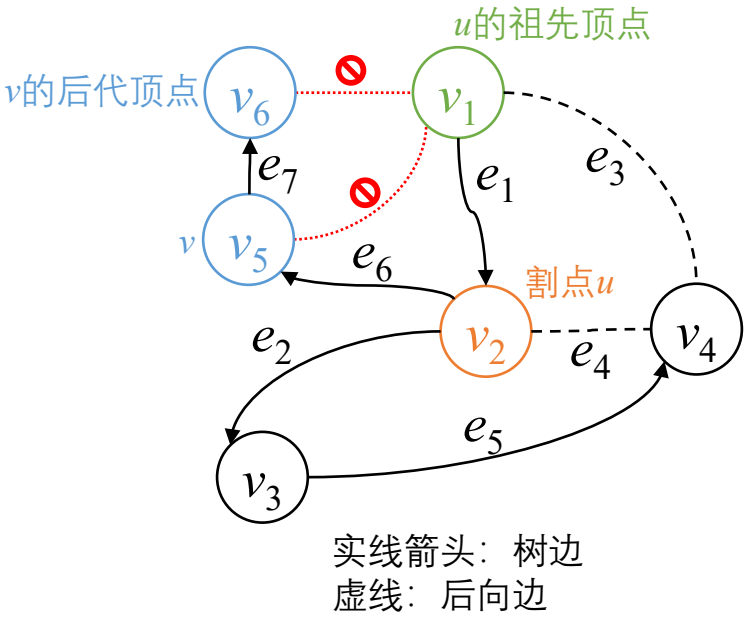
定理2.5（判断非根顶点是否为割点）

- 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。
  - u割开了哪些顶点？



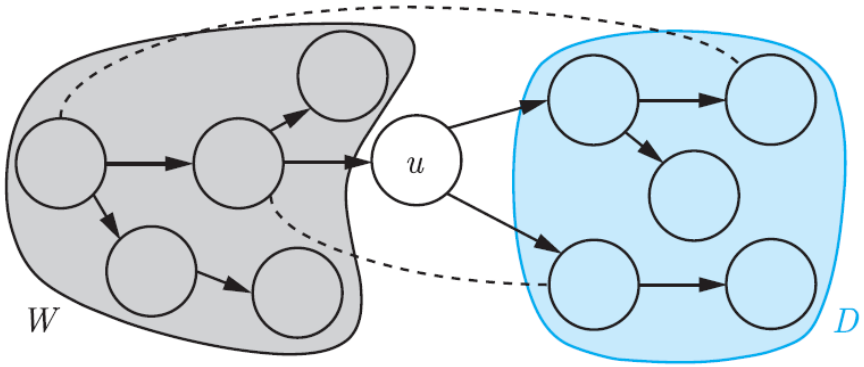
定理2.5（判断非根顶点是否为割点）

- 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。
  - u割开了哪些顶点：v及其后代顶点 和 u的祖先顶点



定理2.5（判断非根顶点是否为割点）

- 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。
  - 必要性：
    - 将顶点u的所有后代顶点的集合记作D，将集合$V \setminus (\{u\} \cup D)$记作$W \neq \emptyset$。
    - 采用反证法，假设u的每个子顶点都不满足，
      则u的每个后代顶点都与W中u的某个祖先顶点间存在一条经过后向边而不经过u的路；
    - 根据DFS树，W中任意两个顶点间存在一条不经过u的路。
    - 综上，集合$D \cup W = V \setminus \{u\}$中任意两个顶点间存在一条不经过$u$的路，与$u$是割点矛盾。

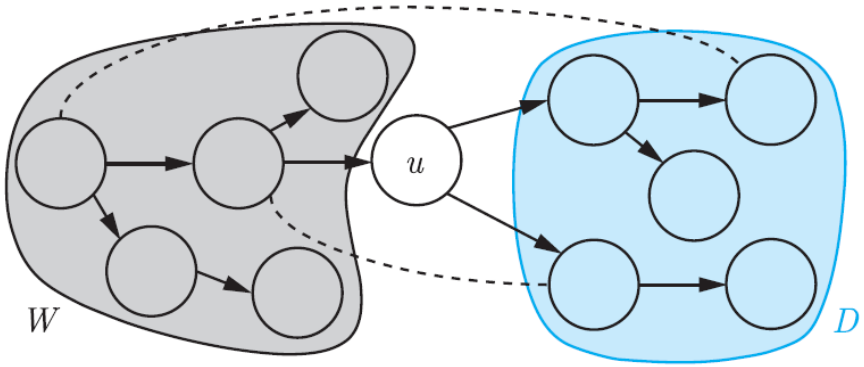


实线箭头：树边
虚线：后向边

定理2.5（判断非根顶点是否为割点）

- 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。
  - 充分性：
    - 由引理2.1，子顶点v及其后代顶点不与集合W中任何顶点相邻，
      即v及其后代顶点与W中顶点间的每条路都经过顶点u，因此，u是割点。

引理2.1 后向边关联一对祖先-后代顶点。



实线箭头：树边
虚线：后向边

DFSCV算法

- 第2项扩展：实现了上述对非根顶点的判定方法
  - 每个顶点的isCutVertex属性：布尔型变量，初值为false，表示该顶点是否为割点
  - 每个顶点的low属性：整数型变量，初值为该顶点的d属性值，
    表示该顶点及其后代顶点通过后向边关联的邻点的最先访问次序（即最小的d属性值）

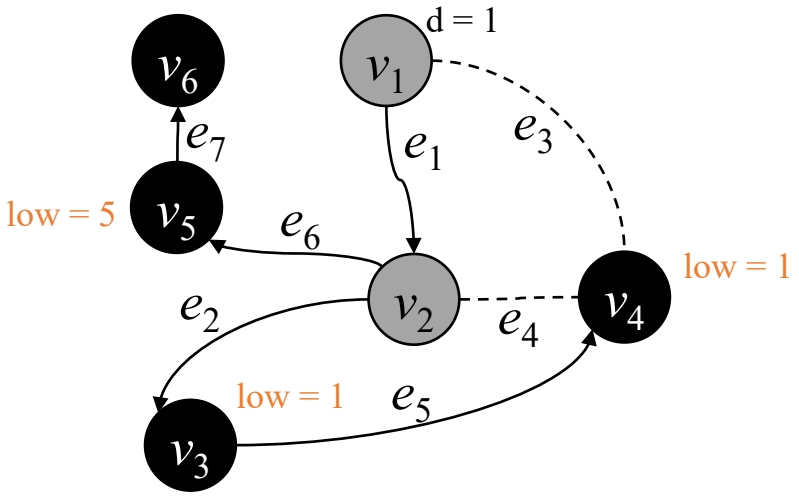
算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10
11
12
13
14
15
16
```

定理2.5 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。



DFSCV算法

- 第2项扩展：实现了上述对非根顶点的判定方法
  - 每个顶点的isCutVertex属性：布尔型变量，初值为false，表示该顶点是否为割点
  - 每个顶点的low属性：整数型变量，初值为该顶点的d属性值，
    表示该顶点及其后代顶点通过后向边关联的邻点的最先访问次序（即最小的d属性值）

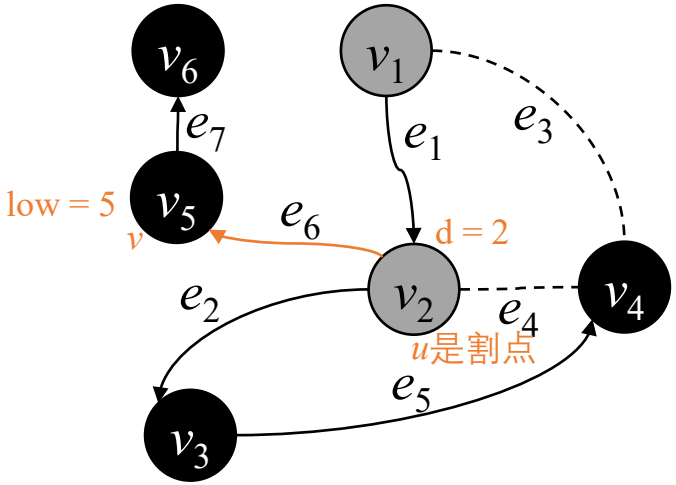
算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14             u.isCutVertex ← true;
15
16
```

定理2.5 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。



DFSCV算法

- 第2项扩展：实现了上述对非根顶点的判定方法
  - 每个顶点的isCutVertex属性：布尔型变量，初值为false，表示该顶点是否为割点
  - 每个顶点的low属性：整数型变量，初值为该顶点的d属性值，
    表示该顶点及其后代顶点通过后向边关联的邻点的最先访问次序（即最小的d属性值）

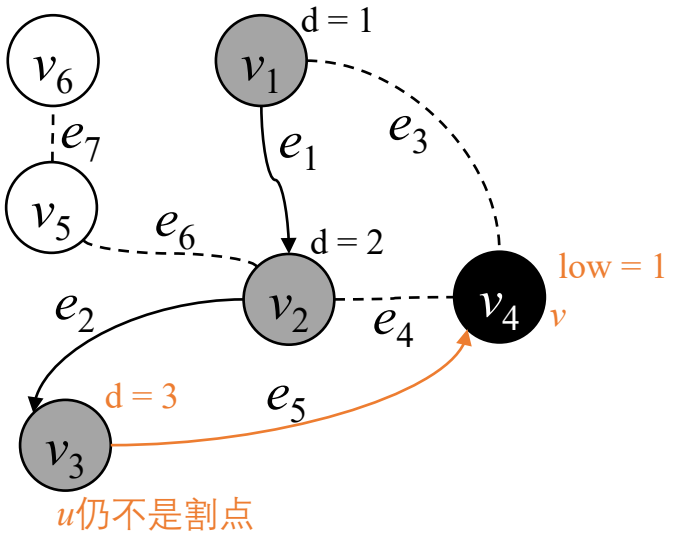
算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14             u.isCutVertex ← true;
15
16
```

定理2.5 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。



DFSCV算法

- 第2项扩展：实现了上述对非根顶点的判定方法
  - 每个顶点的isCutVertex属性：布尔型变量，初值为false，表示该顶点是否为割点
  - 每个顶点的low属性：整数型变量，初值为该顶点的d属性值，
    表示该顶点及其后代顶点通过后向边关联的邻点的最先访问次序（即最小的d属性值）

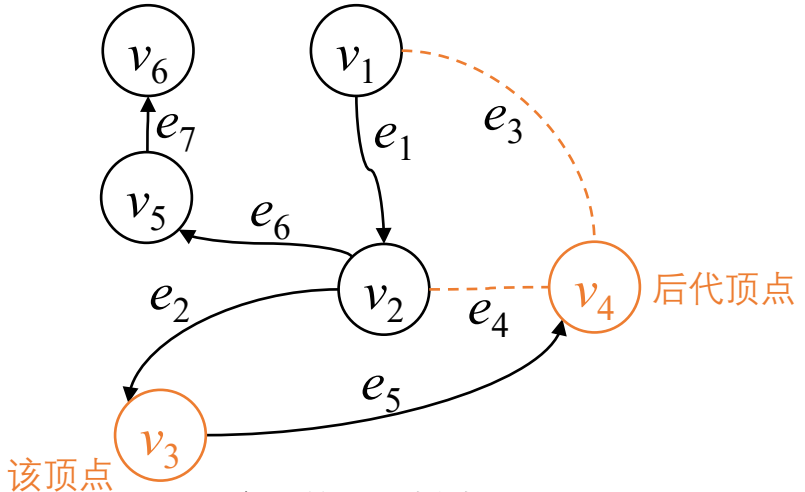
算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15
16
```

对祖先顶点的递归调用结束晚于后代顶点



实线箭头：树边
虚线：后向边

DFSCV算法

- 第2项扩展：实现了上述对非根顶点的判定方法
  - 每个顶点的isCutVertex属性：布尔型变量，初值为false，表示该顶点是否为割点
  - 每个顶点的low属性：整数型变量，初值为该顶点的d属性值，
    表示该顶点及其后代顶点通过后向边关联的邻点的最先访问次序（即最小的d属性值）

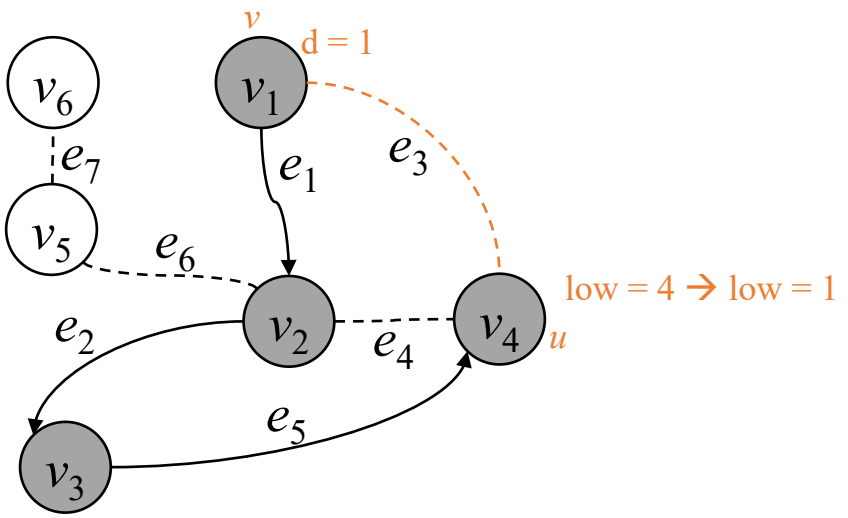
算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15     else if v ≠ u.parent then          后向边
16         u.low ← min{u.low, v.d};
```

对祖先顶点的递归调用结束晚于后代顶点



DFSCV算法

- 第2项扩展：实现了上述对非根顶点的判定方法
  - 每个顶点的isCutVertex属性：布尔型变量，初值为false，表示该顶点是否为割点
  - 每个顶点的low属性：整数型变量，初值为该顶点的d属性值，
    表示该顶点及其后代顶点通过后向边关联的邻点的最先访问次序（即最小的d属性值）

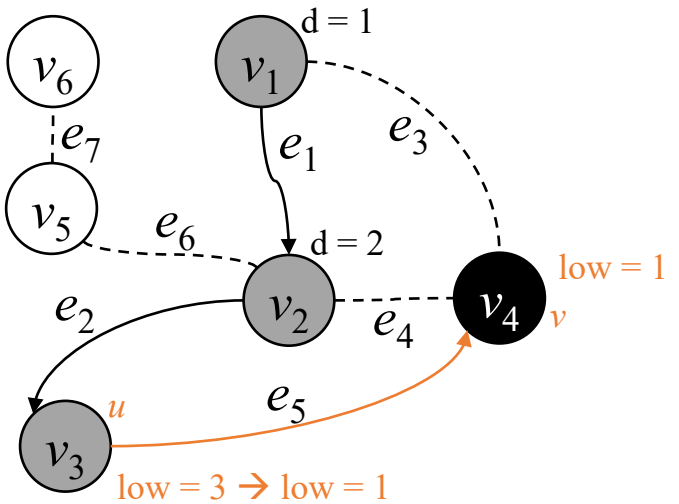
算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then                    树边
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```

对祖先顶点的递归调用结束晚于后代顶点



DFSCV算法

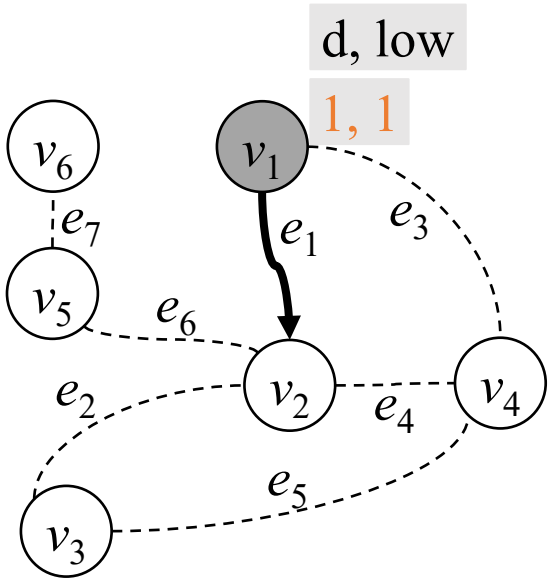
- 例如：从v_1出发，调用DFSCV(G, v_1)
  - v_1.d ← 1
  - v_1.low ← v_1.d

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14             u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

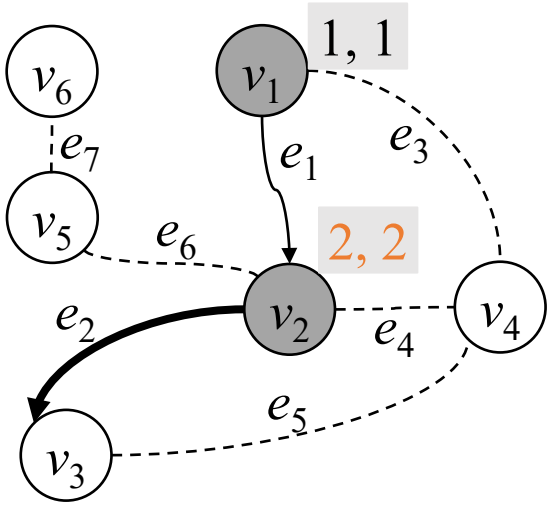
- 递归调用DFSCV(G, v_2)
  - v_2.d ← 2
  - v_2.low ← v_2.d

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

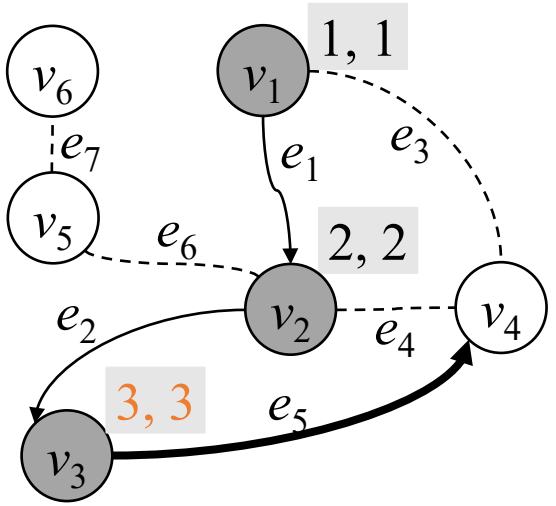
- 递归调用DFSCV(G, v_3)
  - v_3.d ← 3
  - v_3.low ← v_3.d

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

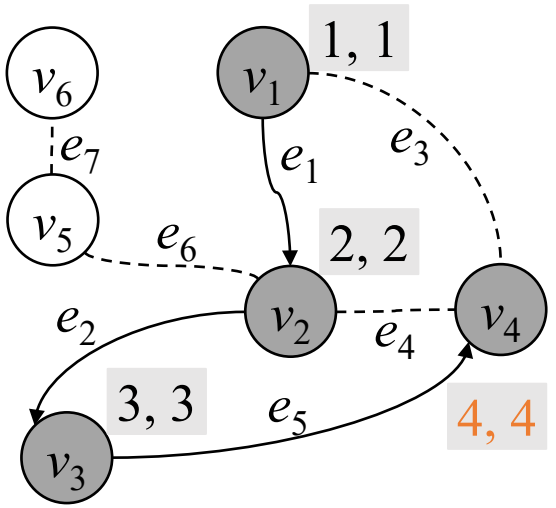
- 递归调用DFSCV(G, v_4)
  - v_4.d ← 4
  - v_4.low ← v_4.d

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

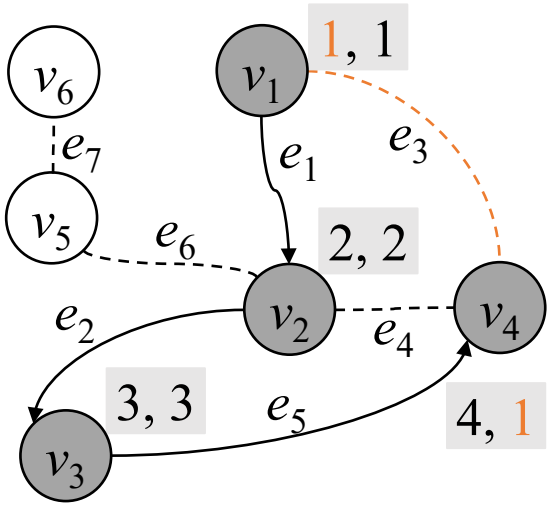
- v_4.low ← v_1.d

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

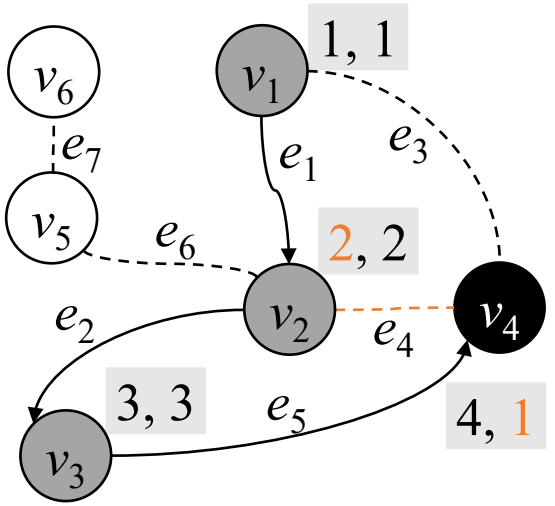
- v_4.low不变
- DFSCV(G, v_4)结束

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

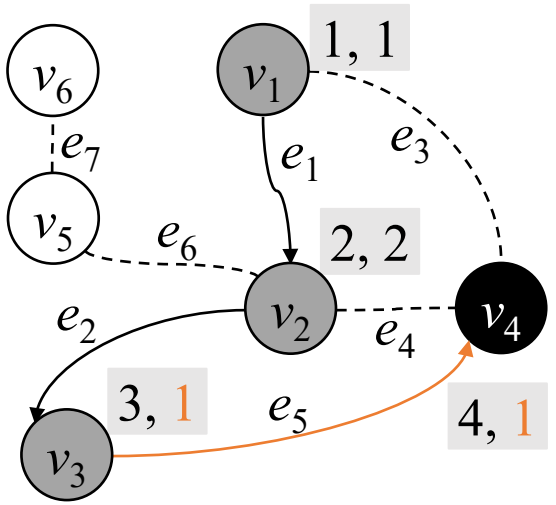
- v_3.low ← v_4.low

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

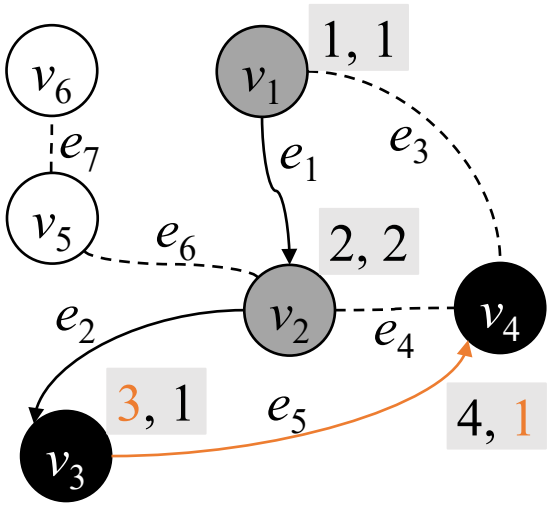
- v_3仍不是割点
- DFSCV(G, v_3)结束

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

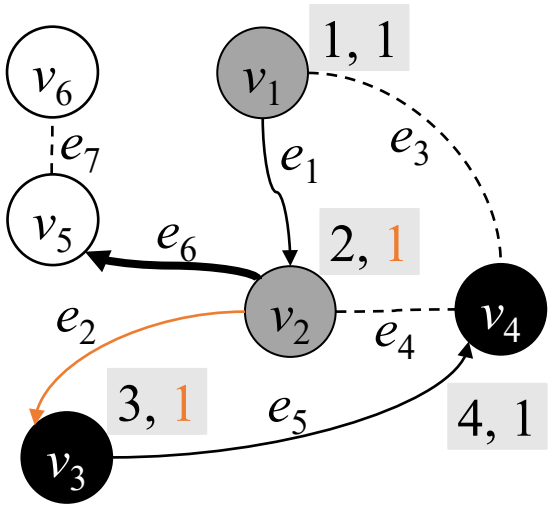
- v_2.low ← v_3.low

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

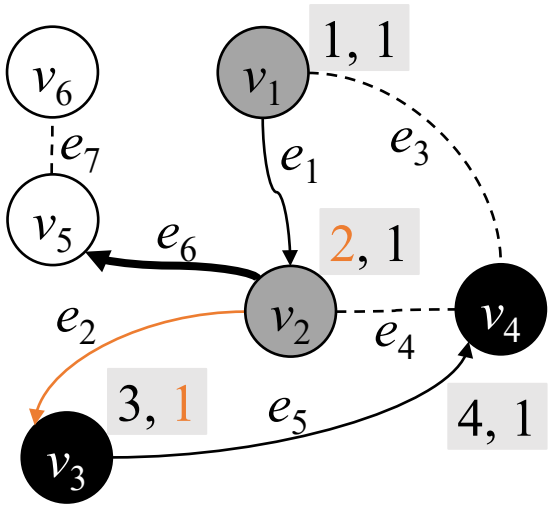
- v_2仍不是割点
- DFSCV(G, v_2)尚未结束

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14             u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

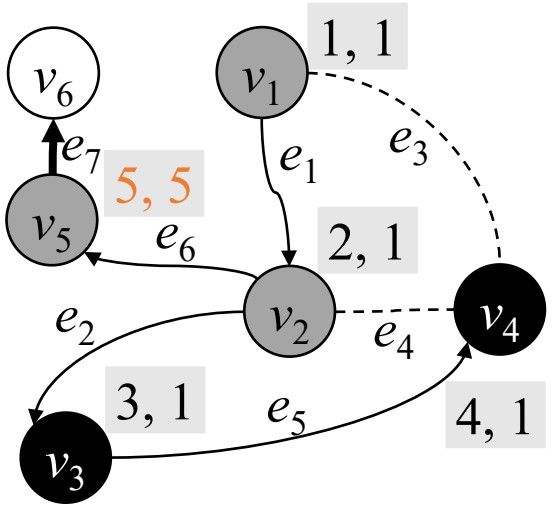
- 递归调用DFSCV(G, v_5)
  - v_5.d ← 5
  - v_5.low ← v_5.d

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

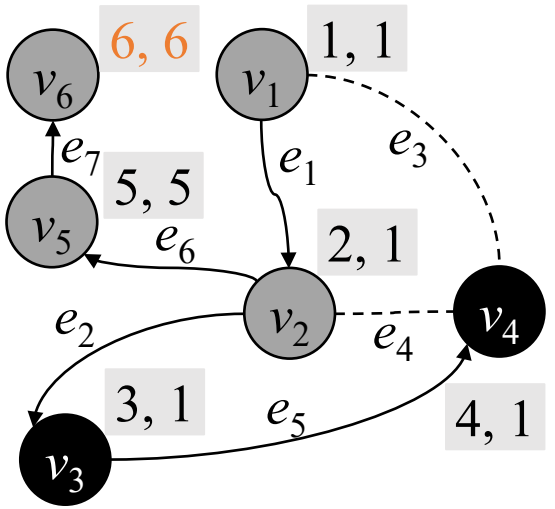
- 递归调用DFSCV(G, v_6)
  - v_6.d ← 6
  - v_6.low ← v_6.d

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

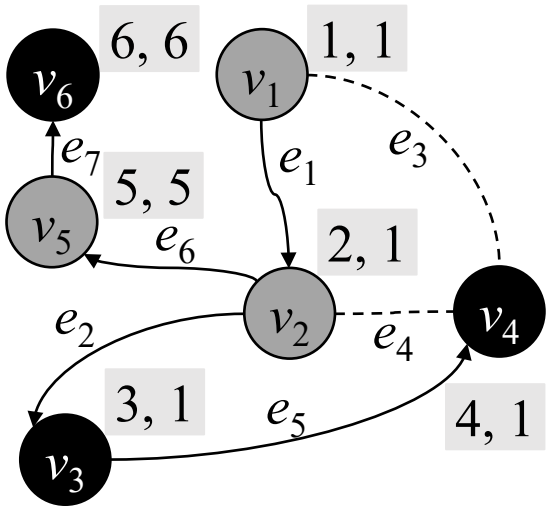
- DFSCV(G, v_6)结束

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

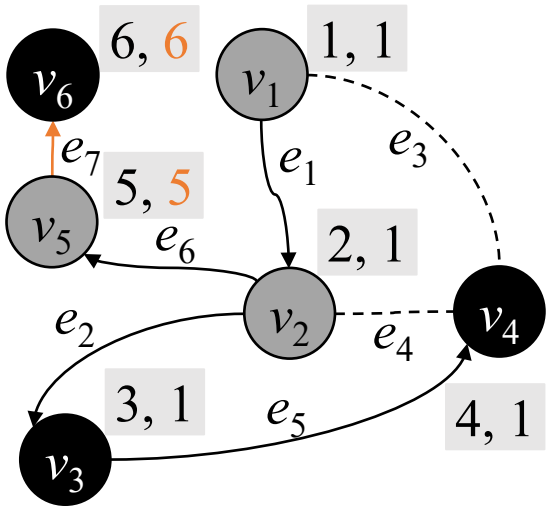
- v_5.low不变

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

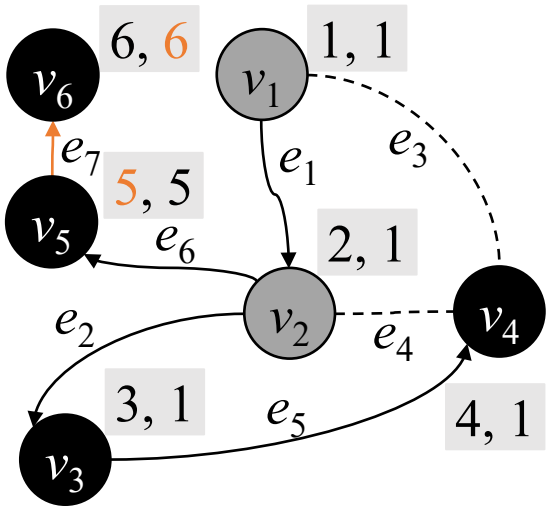
- v_5是割点
- DFSCV(G, v_5)结束

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14             u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

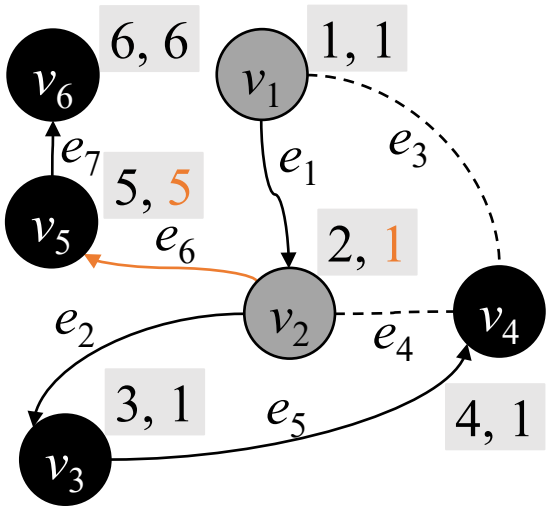
- v_2.low不变

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

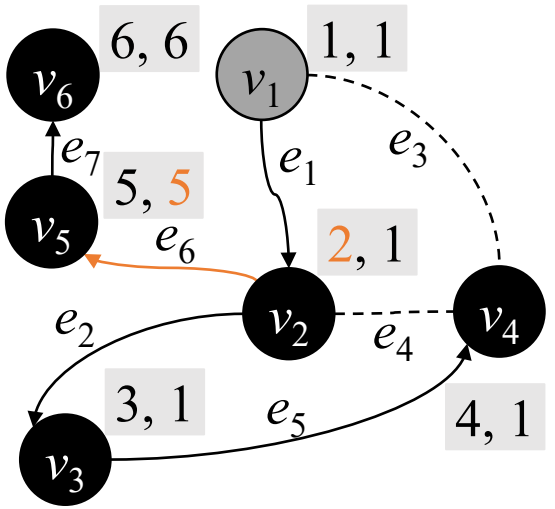
- v_2是割点
- DFSCV(G, v_2)结束

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14             u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

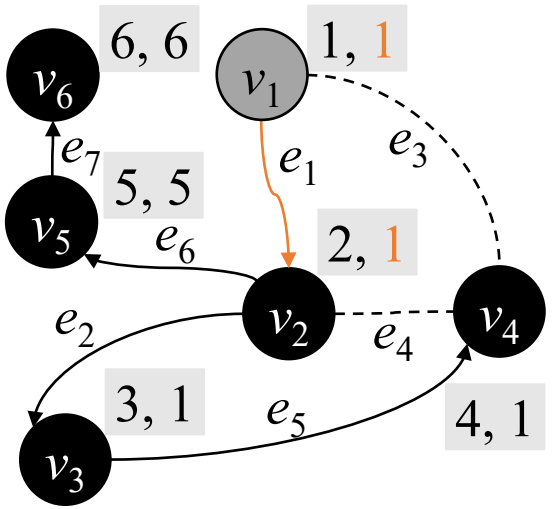
- v_1.low不变

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11
12
13             if u.parent ≠ null 且 v.low ⩾ u.d then
14                 u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

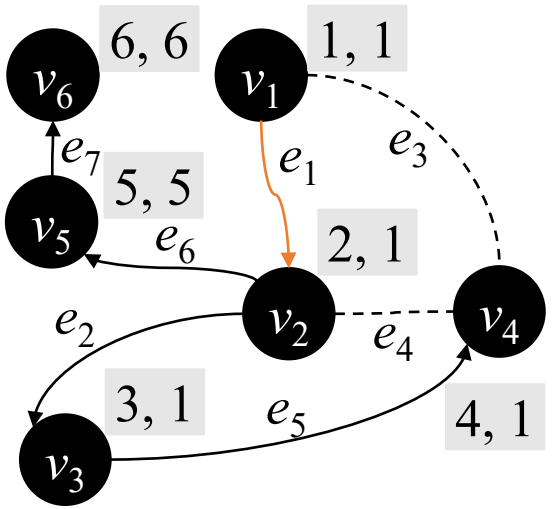
- v_1仍不是割点
- DFSCV(G, v_1)结束

算法 2.2: DFSCV 算法伪代码

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

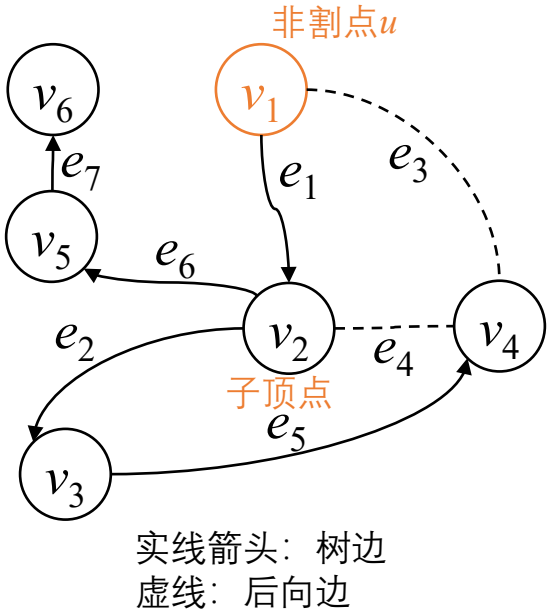
初值: 变量 time 初值为 0；顶点集 V 中所有顶点的 visited 初值为 false，parent 初值为 null，children 初值为 0，isCutVertex 初值为 false

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



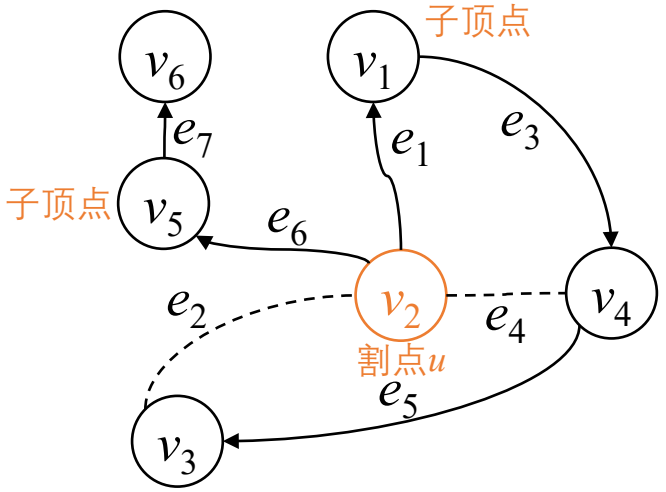
定理2.6（判断根顶点是否为割点）

- 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。
  - 例如：根顶点v_1不是割点



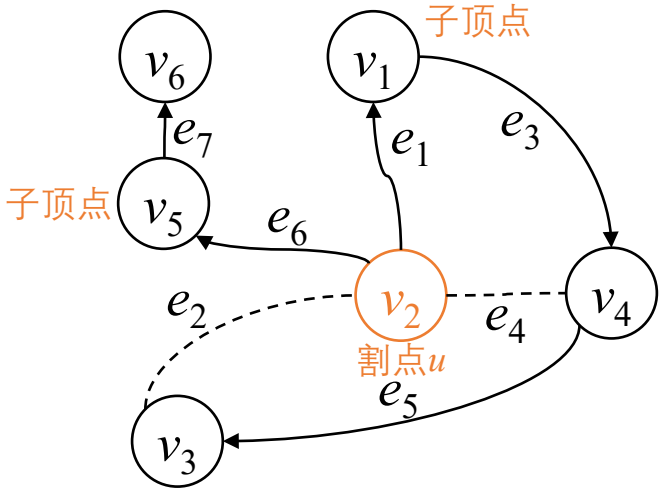
定理2.6（判断根顶点是否为割点）

- 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。
  - 例如：根顶点v_2是割点



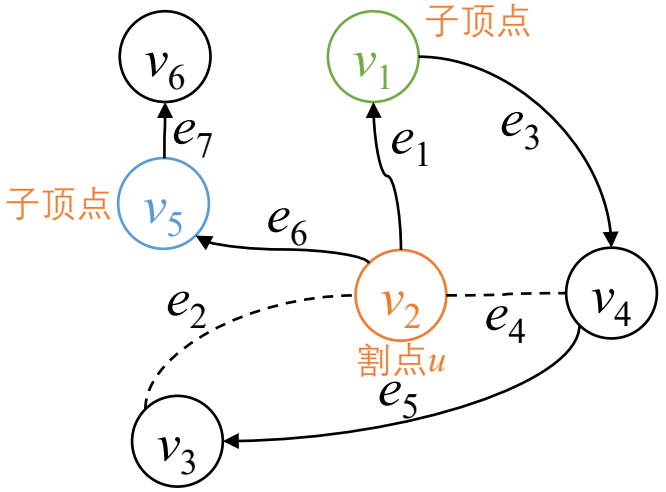
定理2.6（判断根顶点是否为割点）

- 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。
  - u割开了哪些顶点？



定理2.6（判断根顶点是否为割点）

- 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。
  - u割开了哪些顶点：u的各子顶点



实线箭头：树边
虚线：后向边

定理2.6（判断根顶点是否为割点）

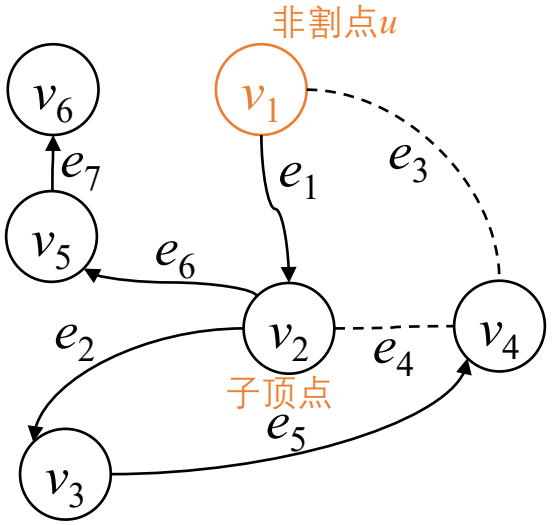
- 若顶点u是根顶点，则u是割点当且仅当u有至少2个子顶点。
  - 必要性：
    - 采用反证法，假设根顶点u没有子顶点，则图G是平凡图，与u是割点矛盾；

非割点u

平凡图

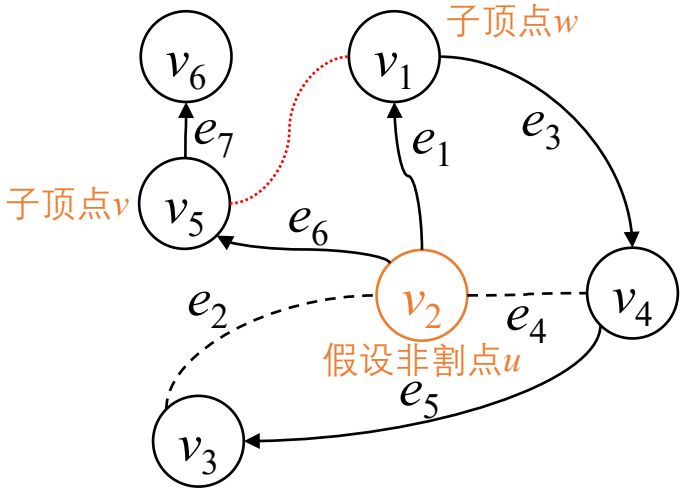
定理2.6（判断根顶点是否为割点）

- 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。
  - 必要性：
    - 采用反证法，假设根顶点u没有子顶点，则图G是平凡图，与u是割点矛盾；
    - 假设u只有1个子顶点，则根据DFS树，集合$V \setminus \{u\}$中任意两个顶点间存在一条不经过u的路，与u是割点矛盾。



定理2.6（判断根顶点是否为割点）

- 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。
  - 充分性：
    - 采用反证法，假设根顶点u不是割点，则u的第一个被访问的子顶点v和另一个子顶点w间存在一条不经过u的路，
    - 因此，对v的DFS调用结束时，w已被访问过，即w是v的后代顶点，与w是u的子顶点矛盾。



实线箭头：树边
虚线：后向边

DFSCV算法

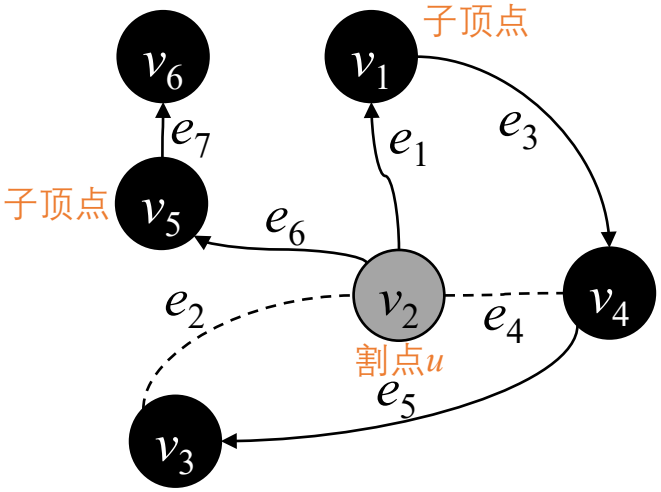
- 第3项扩展：实现了上述对根顶点的判定方法

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11         if u.parent = null 且 u.children ⩾ 2 then
12             u.isCutVertex ← true;
13         else if u.parent ≠ null 且 v.low ⩾ u.d then
14             u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

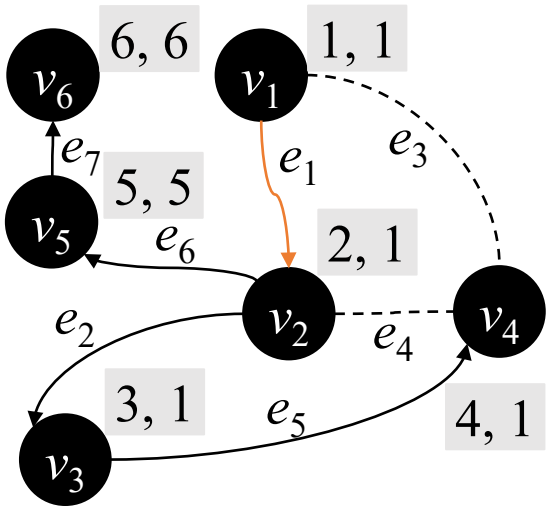
- 例如：v_1仍不是割点，DFSCV(G, v_1)结束

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11         if u.parent = null 且 u.children ⩾ 2 then
12             u.isCutVertex ← true;
13         else if u.parent ≠ null 且 v.low ⩾ u.d then
14             u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

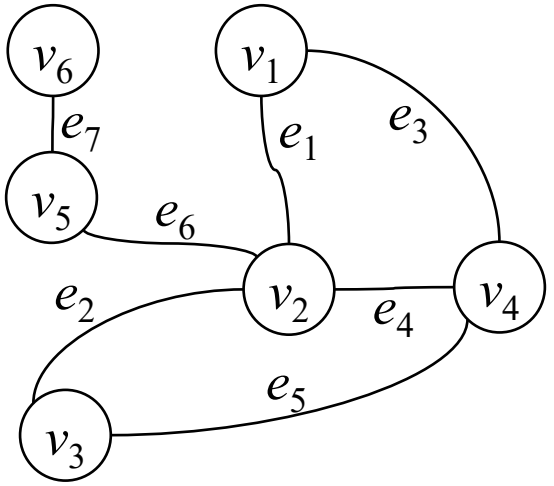
- 时间复杂度：$O(n+m)$

算法 2.2: DFSCV 算法伪代码

输入: 连通图 $G=\langle V,E\rangle$，顶点 u

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11         if u.parent = null 且 u.children ⩾ 2 then
12             u.isCutVertex ← true;
13         else if u.parent ≠ null 且 v.low ⩾ u.d then
14             u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



DFSCV算法

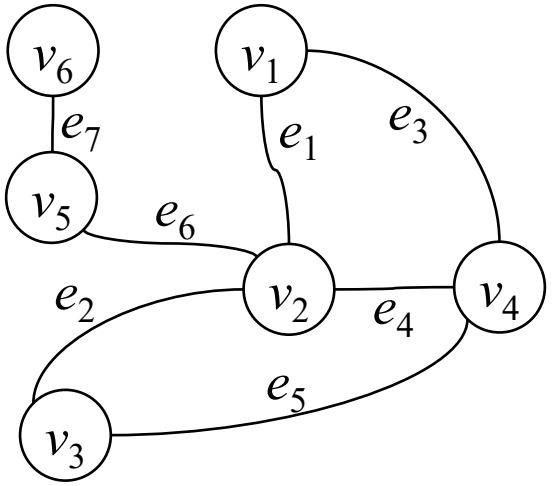
- DFSCV算法略作修改便可用于找出图中所有的割边，作为练习

算法 2.2: DFSCV 算法伪代码

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

初值: 变量 time 初值为 0；顶点集 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 ← time + 1;
2  u.d ← time;
3  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7          v.parent ← u;
8          u.children ← u.children + 1;
9          DFSCV(G, v);
10         u.low ← min{u.low, v.low};
11         if u.parent = null 且 u.children ⩾ 2 then
12             u.isCutVertex ← true;
13         else if u.parent ≠ null 且 v.low ⩾ u.d then
14             u.isCutVertex ← true;
15     else if v ≠ u.parent then
16         u.low ← min{u.low, v.d};
```



请认真完成课后练习