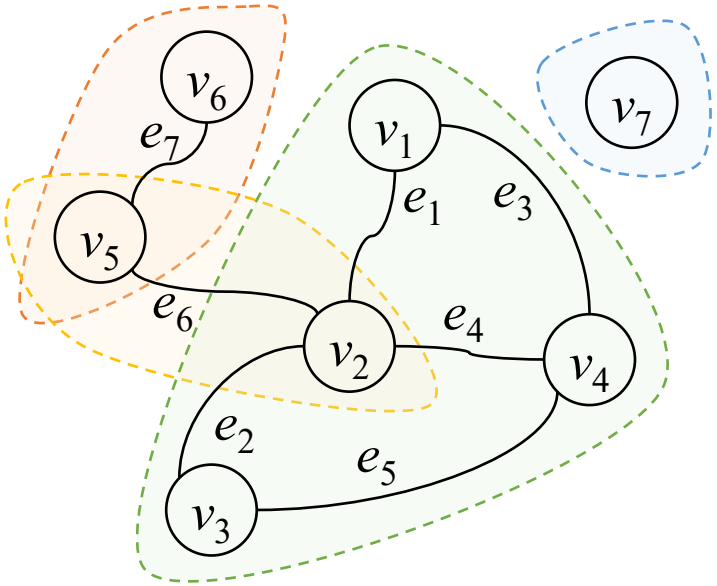
第4章 连通度

程龚
南京大学 计算机学院
gcheng@nju.edu.cn
http://ws.nju.edu.cn/~gcheng

本章内容

- 第4.1节 块
  - 第4.1.1节 理论
  - **第4.1.2节 算法**
- 第4.2节 割集和连通度

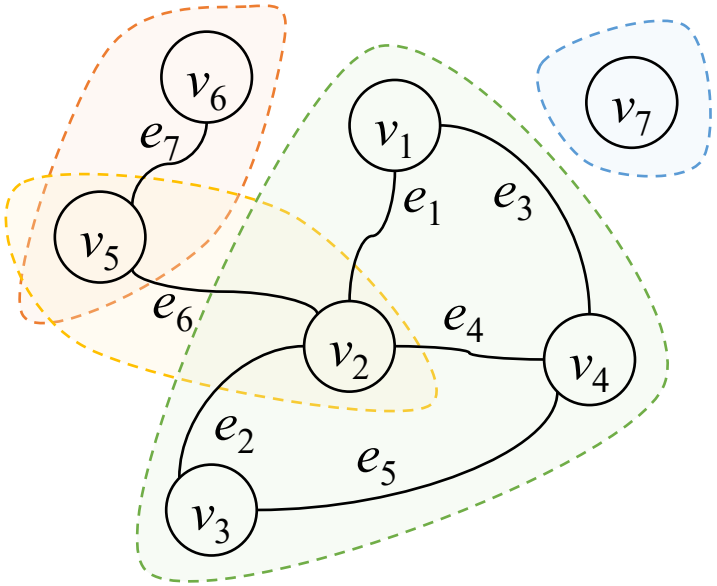
如何找出图中所有块?



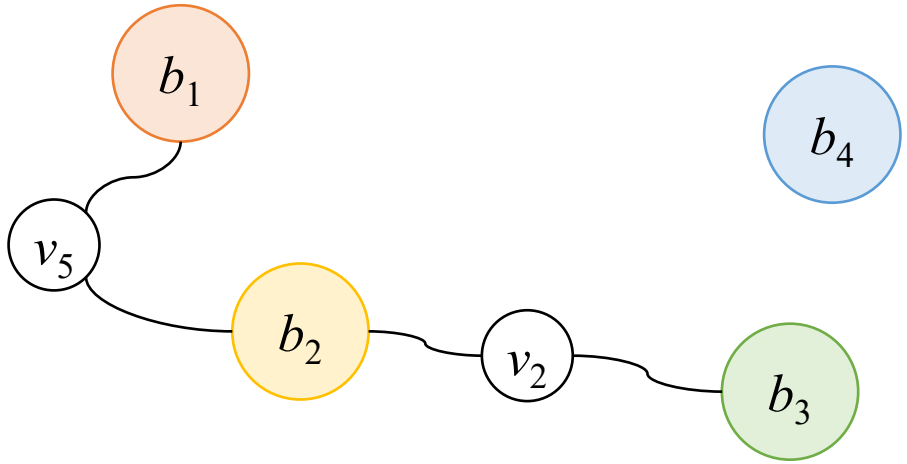
图G

如何找出图中所有块?

- 修改DFSCV算法：判定割点 → 找到块



图G



块-割点图H

DFSBlk算法（扩展DFS算法用于找块）

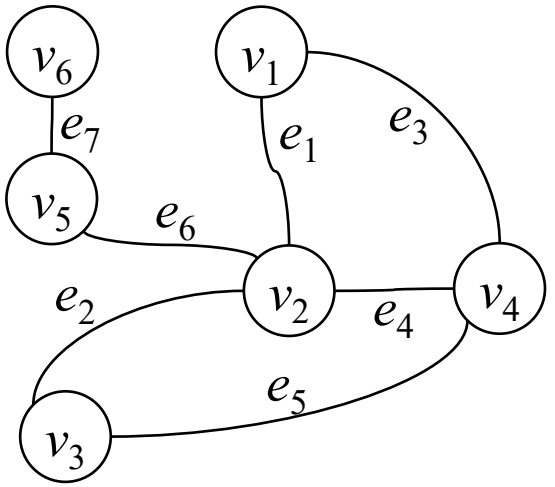
- 基本思路：从非平凡连通图中任意一个指定顶点出发，按DFS的方式有序地遍历所有顶点，并找出所有块。

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

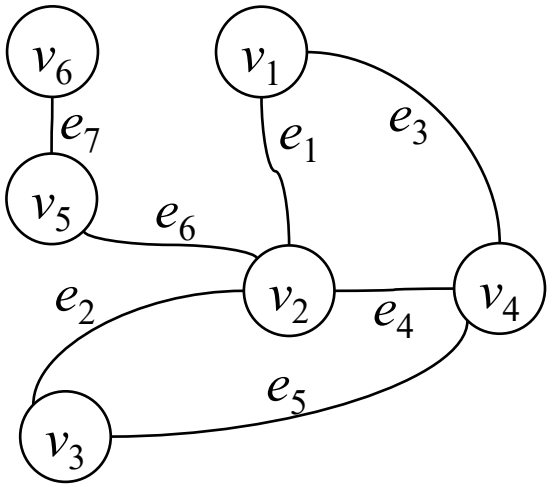
- 基于DFSCV算法

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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  u.low ← u.d;
4  u.visited ← true;
5  foreach (u, v) ∈ E do
6      if v.visited = false then
7
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13
14
15
16
17
18     else if v ≠ u.parent then
19
20
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

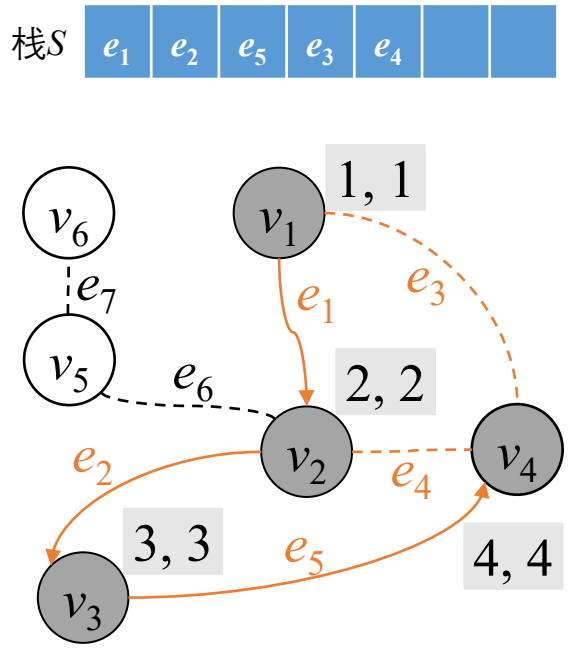
- 第1项修改：发现每条边后，将其增加到一个初值为空的栈S的栈顶。

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));                    树边
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13
14
15
16
17
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));                后向边
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

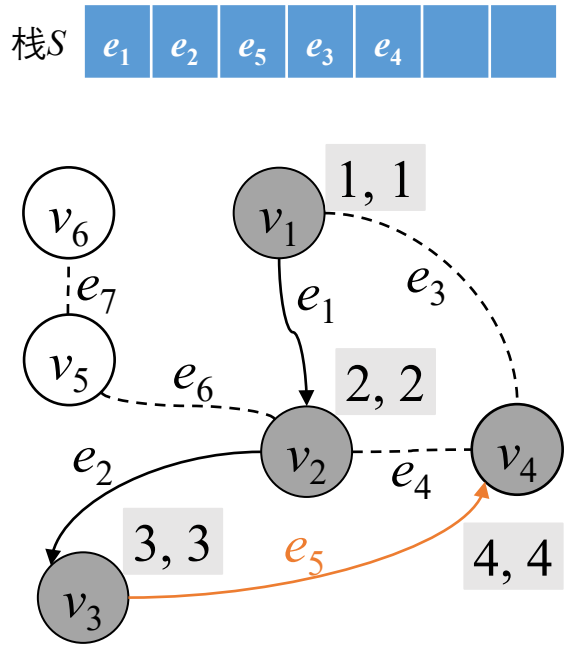
- 第1项修改：发现每条边后，将其增加到一个初值为空的栈S的栈顶。

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13
14
15
16
17
18     else if v ≠ u.parent then        避免树边再次入栈
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

- 第1项修改：发现每条边后，将其增加到一个初值为空的栈S的栈顶。

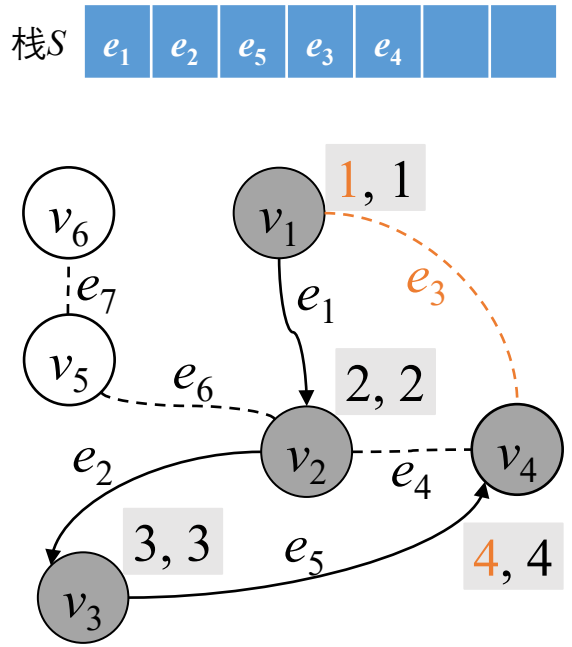
算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13
14
15
16
17
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```

避免后向边再次入栈



DFSBlk算法

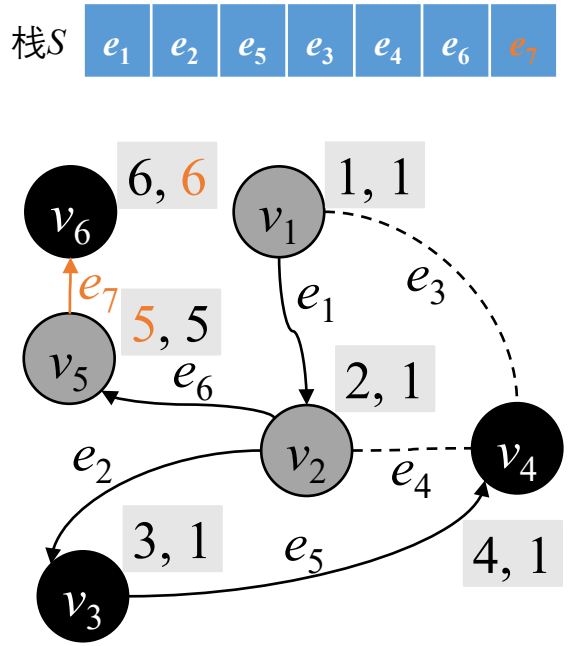
- 第2项修改：将割点判定改为输出一个块的边集，
  即反复出栈并输出出栈的边，直至该块中最早入栈的边(u, v)。

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

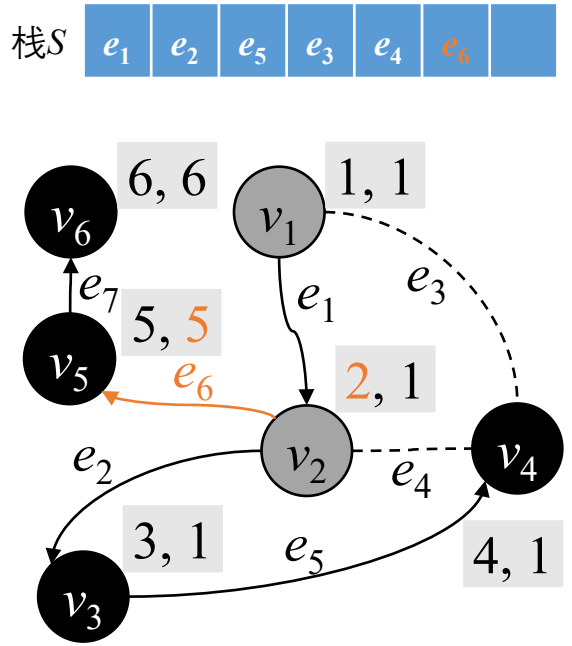
- 第2项修改：将割点判定改为输出一个块的边集，
  即反复出栈并输出出栈的边，直至该块中最早入栈的边(u, v)。

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

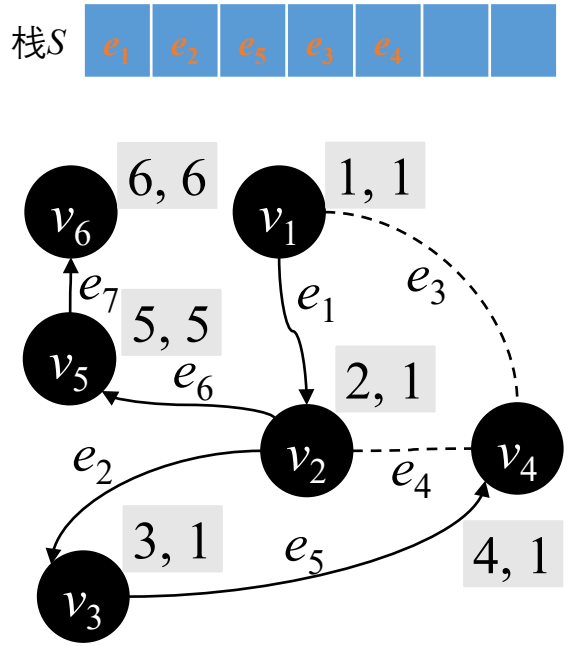
- 若算法的出发点不是割点，
  则算法运行结束时S不为空，而是存储了出发点所在块的边集，
  因此，还需额外输出这个块的边集，这一点未体现在算法4.1中。

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

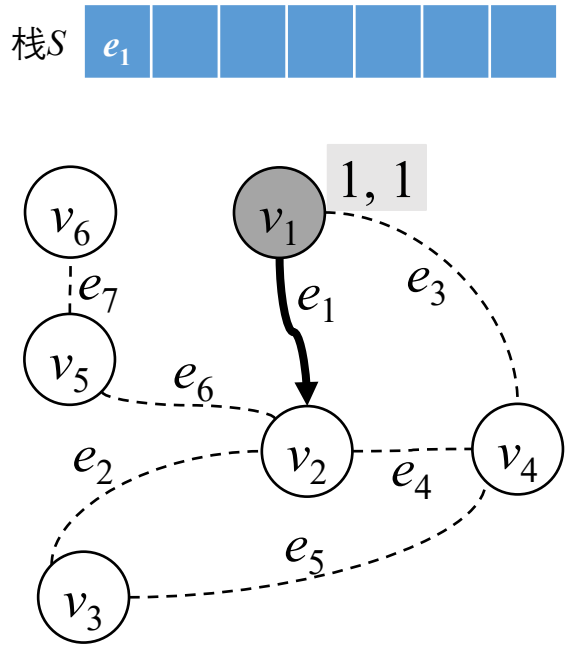
- 例如：从v_1出发，调用DFSBlk(G, v_1)，e_1入栈
  - v_1.d ← 1
  - v_1.low ← v_1.d

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

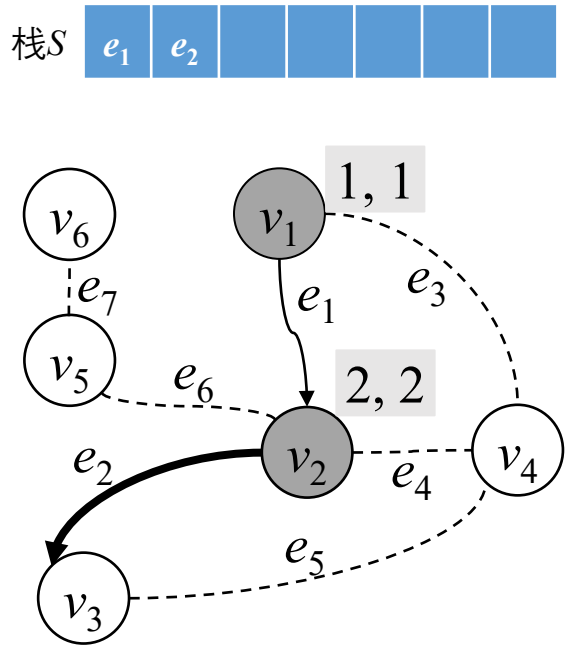
- 递归调用DFSBlk(G, v_2)，e_2入栈
  - v_2.d ← 2
  - v_2.low ← v_2.d

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

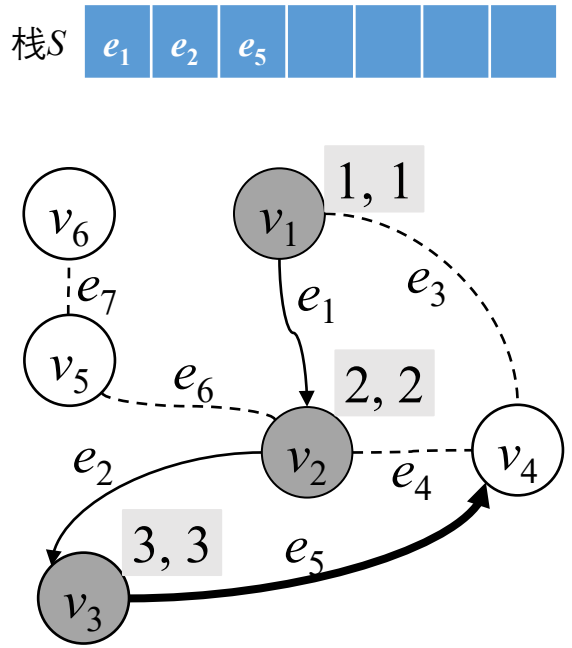
- 递归调用DFSBlk(G, v_3)，e_5入栈
  - v_3.d ← 3
  - v_3.low ← v_3.d

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

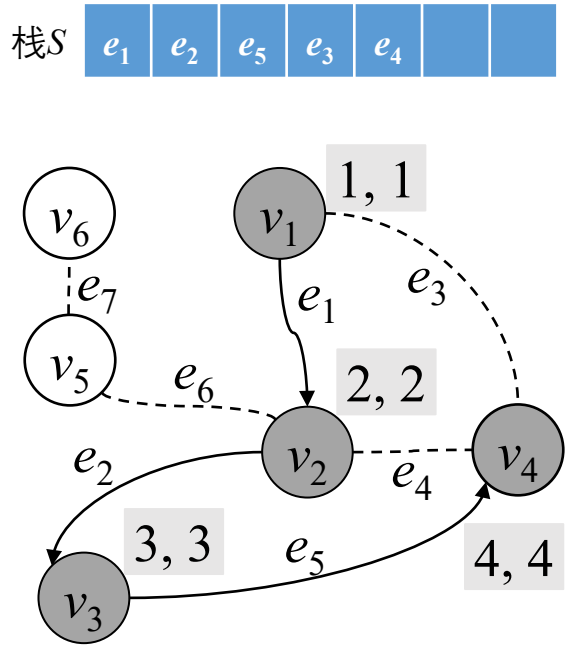
- 递归调用DFSBlk(G, v_4)，e_3、e_4入栈
  - v_4.d ← 4
  - v_4.low ← v_4.d

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

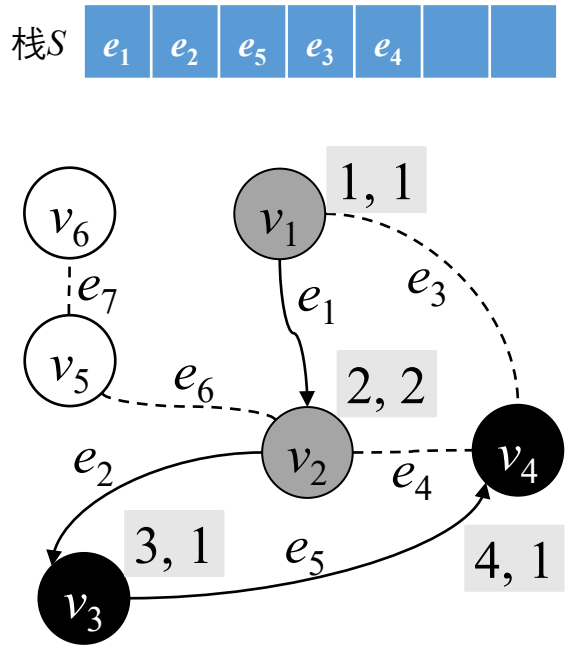
- DFSBlk(G, v_4)结束
  - v_4.low ← v_1.d
- DFSBlk(G, v_3)结束
  - v_3.low ← v_4.low

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

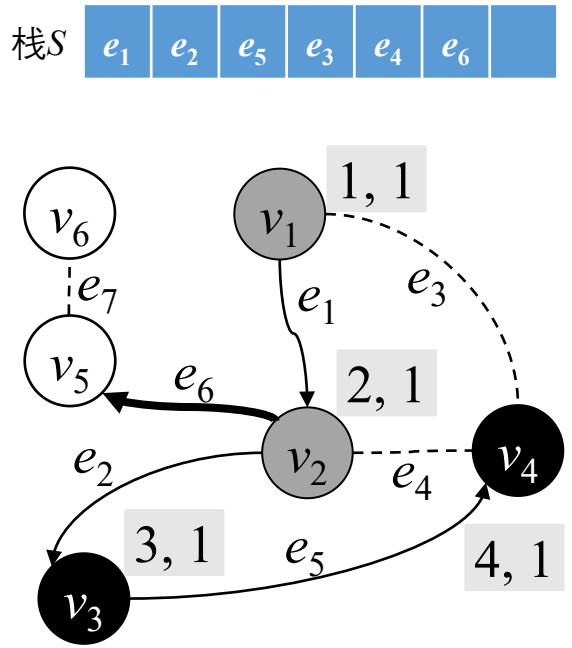
- 继续递归调用DFSBlk(G, v_2)，e_6入栈
  - v_2.low ← v_3.low

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

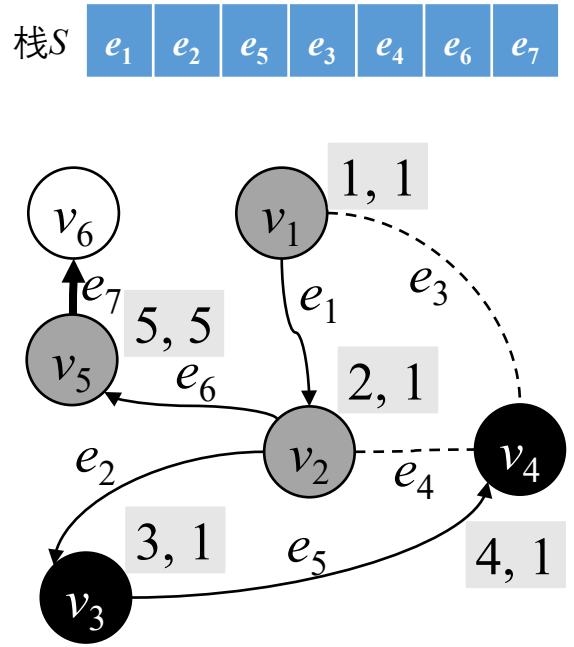
- 递归调用DFSBlk(G, v_5)，e_7入栈
  - v_5.d ← 5
  - v_5.low ← v_5.d

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

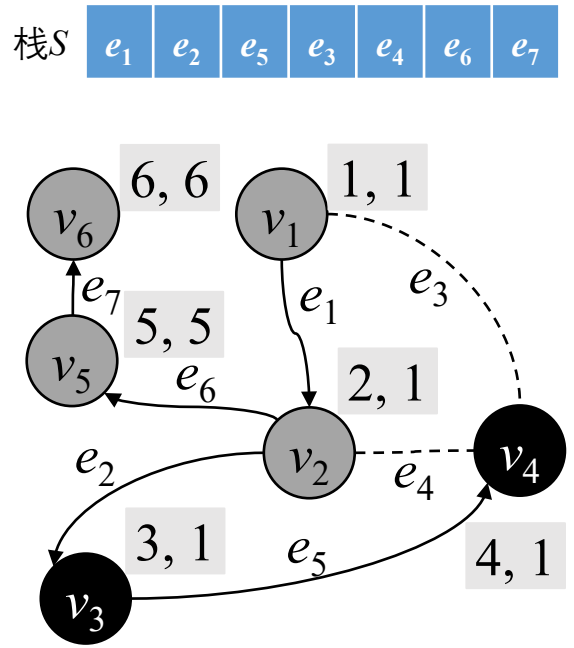
- 递归调用DFSBlk(G, v_6)
  - v_6.d ← 6
  - v_6.low ← v_6.d

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

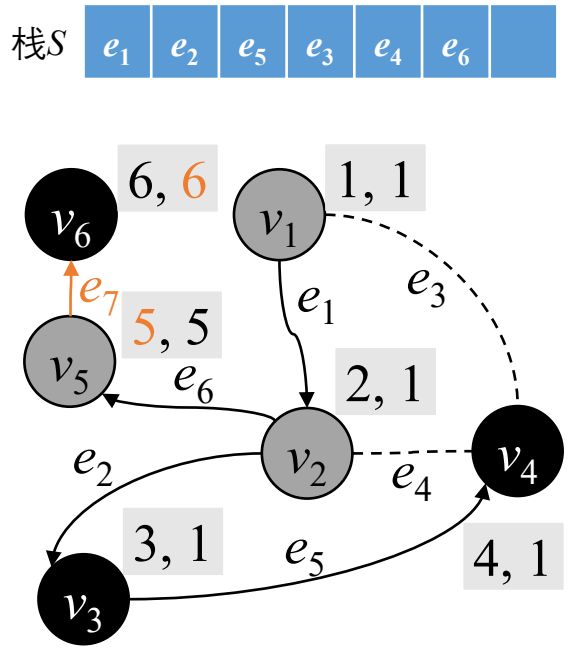
- DFSBlk(G, v_6)结束
- e_7出栈，输出一个块的边集$\{e_7\}$

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E\rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

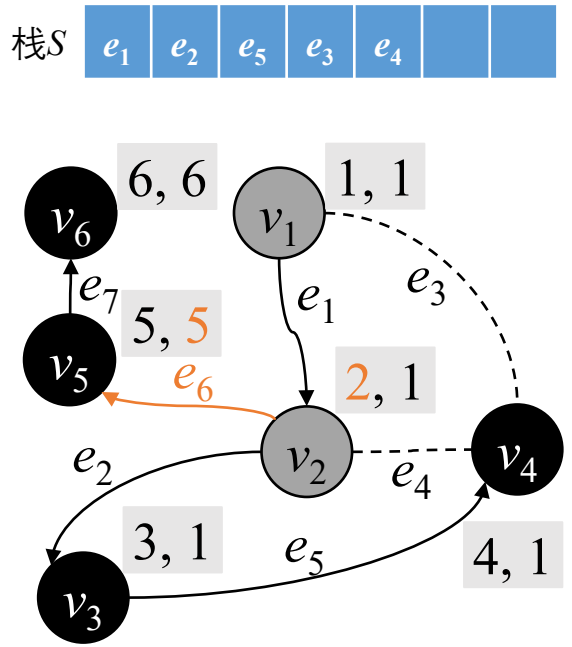
- DFSBlk(G, v_5)结束
- e_6出栈，输出一个块的边集$\{e_6\}$

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E\rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



DFSBlk算法

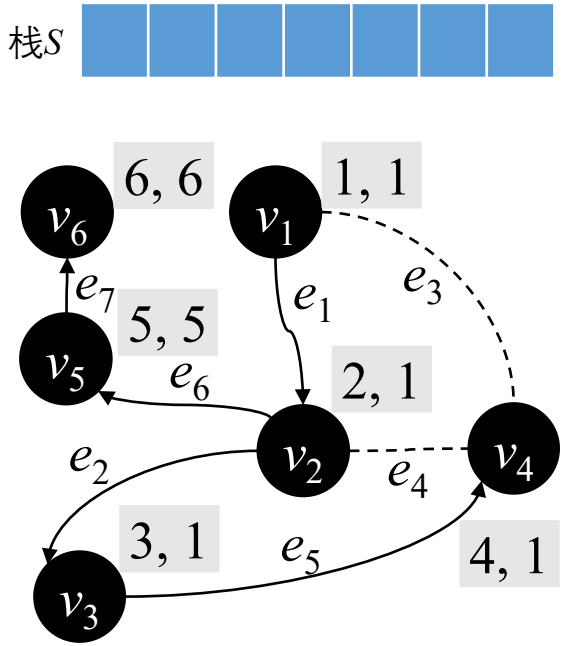
- DFSBlk(G, v_2)结束
- DFSBlk(G, v_1)结束
- S不为空，全部出栈，输出一个块的边集$\{e_4, e_3, e_5, e_2, e_1\}$

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



DFSBlk算法的正确性

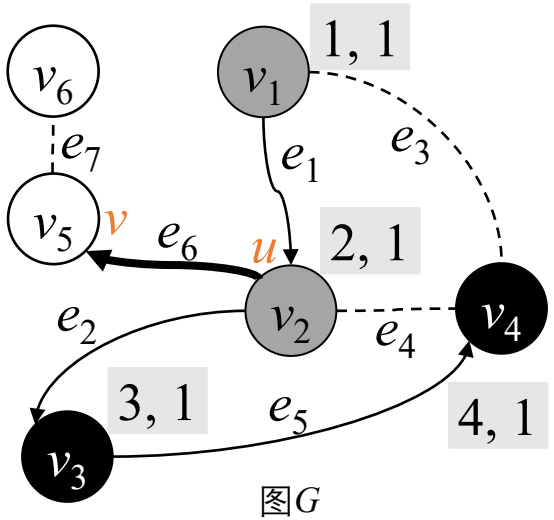
- 同步分析DFSCV算法和DFSBlk算法运行的下述时间段：
  - 开始时刻是发现边(u, v)并从顶点u递归调用访问顶点v，
    结束时刻是完成对v的递归调用且DFSCV算法判定u为割点。

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G = \langle V, E \rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



图G

DFSBlk算法的正确性

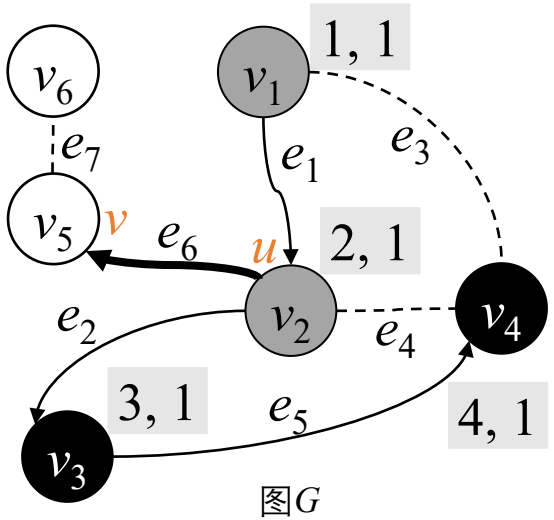
- 对于非平凡连通图G，其块-割点图$H = <B \cup C, E'>$是树。
- 以割点$u \in C$为界，H可分解为至少两棵含公共顶点u的树，
  其中恰有一棵树$T = <B_T \cup C_T, E_T>$中的一个顶点$b \in B_T$表示的块含顶点v和边(u, v)。

算法 4.1: DFSBlk 算法伪代码

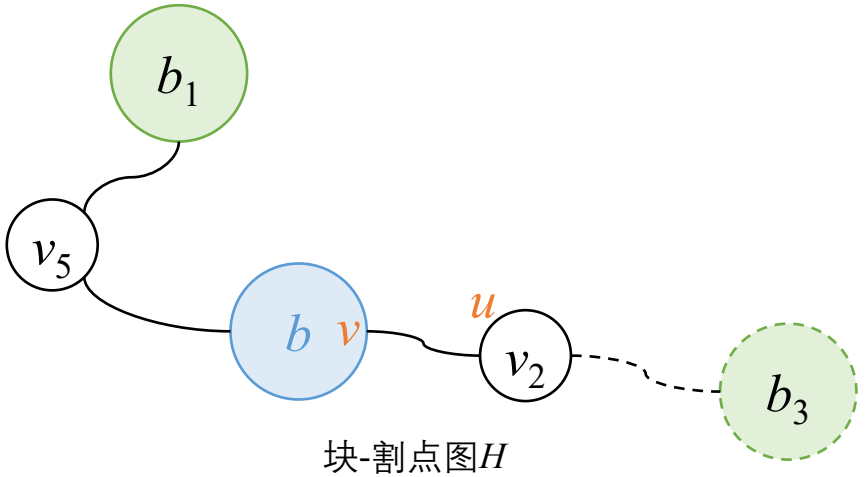
输入: 非平凡连通图 $G = \langle V, E \rangle$，顶点 u

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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



图G



块-割点图H
（实线：T）

定理4.4

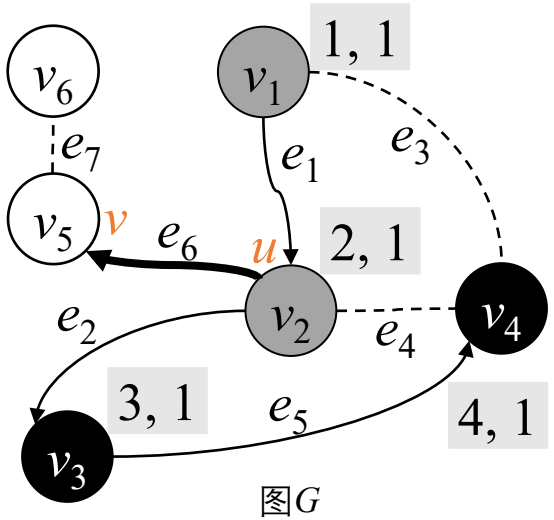
- 上述时间段内发现的边恰组成集合B_T中的顶点表示的所有块。

算法 4.1: DFSBlk 算法伪代码

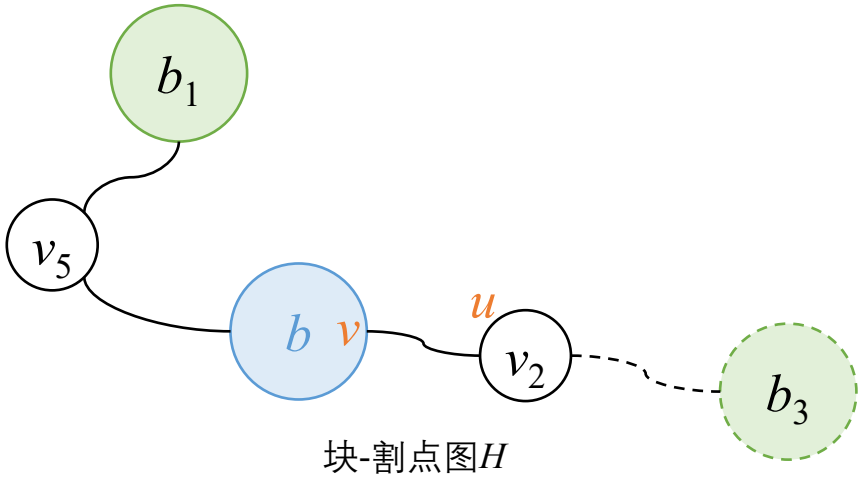
输入: 非平凡连通图 $G = \langle V, E \rangle$，顶点 u

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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



图G



块-割点图H
（实线：T）

定理4.4

- 上述时间段内发现的边恰组成集合B_T中的顶点表示的所有块。
  - 将上述时间段内发现的边的集合记作E_1，将集合B_T中的顶点表示的所有块的边集的并集记作E_2。
  - 先证$E_1 \subseteq E_2$，由割点的性质，即定理2.5，得证。
  - 再证$E_2 \subseteq E_1$，由DFS算法的性质，即定理2.2，得证。

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。

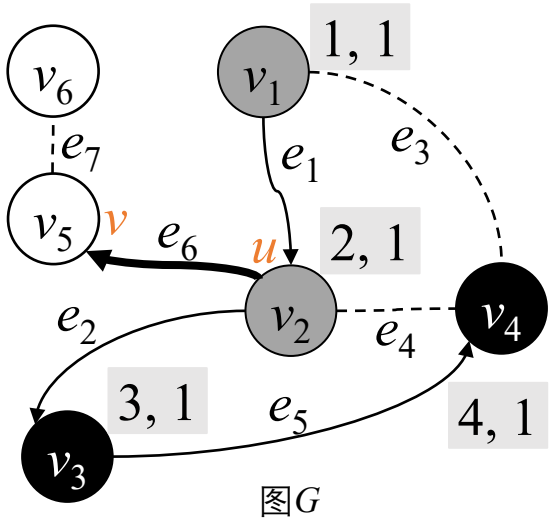
定理2.2 从顶点u出发运行DFS算法，恰能访问与u连通的所有顶点。

算法 4.1: DFSBlk 算法伪代码

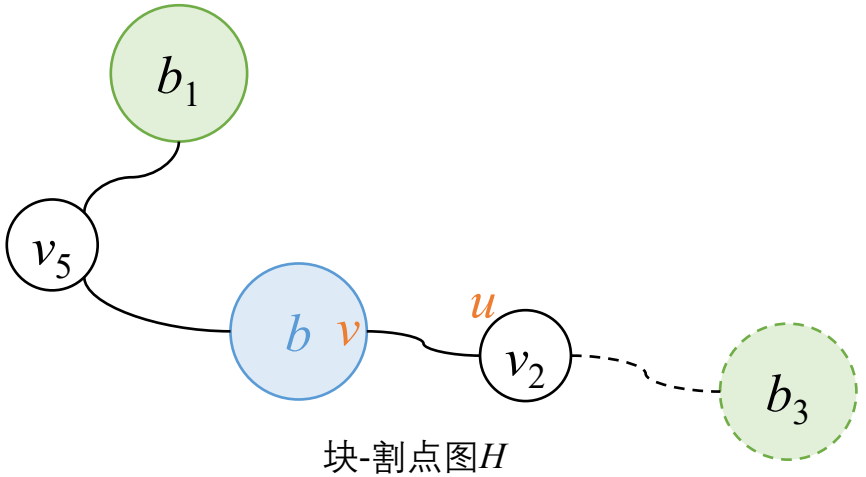
输入: 非平凡连通图 $G = \langle V, E \rangle$，顶点 u

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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



图G



块-割点图H
（实线：T）

定理4.4

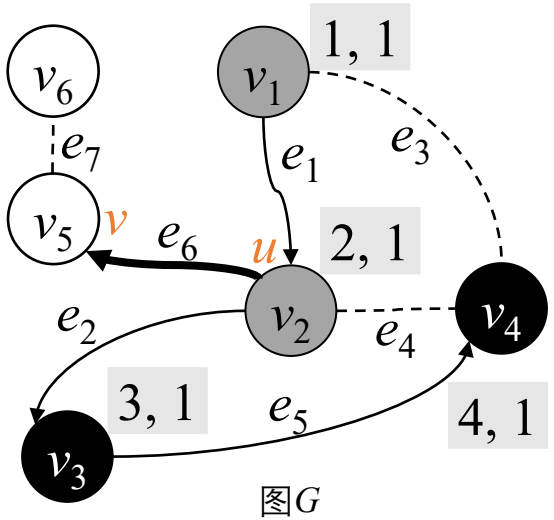
- 上述时间段内发现的边恰组成集合B_T中的顶点表示的所有块。
- 因此，在上述时间段内：
  - DFSBlk算法恰将这些块中的边增加到栈S中，其中最先增加的边是(u, v)。
  - 同时，在DFS树中，顶点v的后代顶点恰由这些块的顶点集（除顶点u, v外）组成。

算法 4.1: DFSBlk 算法伪代码

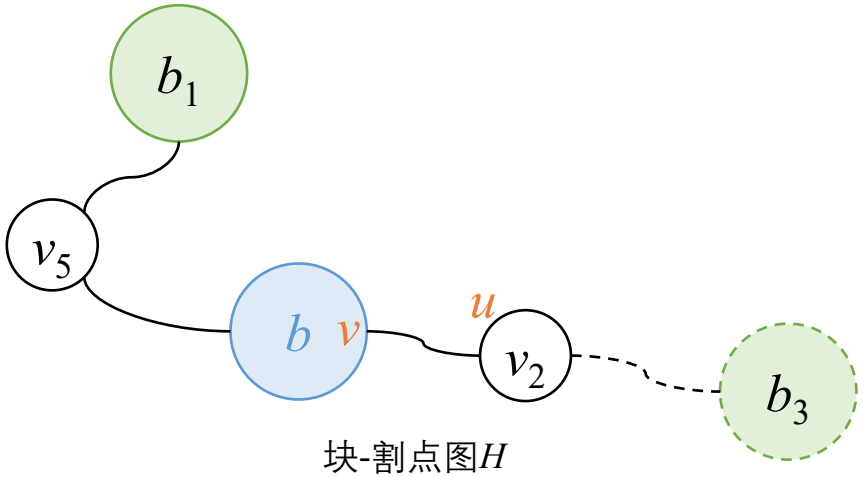
输入: 非平凡连通图 $G = \langle V, E \rangle$，顶点 u

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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



图G



块-割点图H
（实线：T）

定理4.5

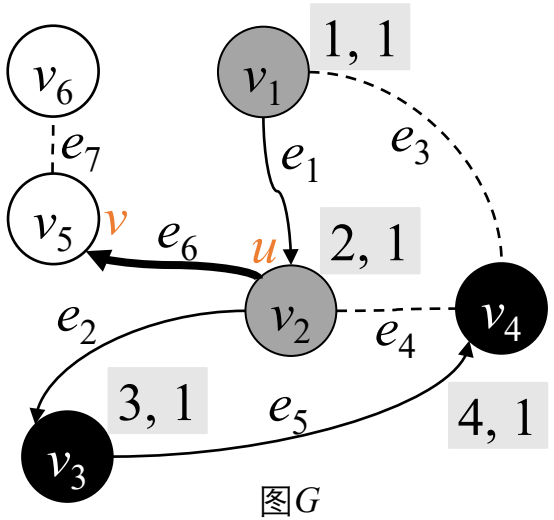
- 在上述时间段的结束时刻，边(u, v)在栈S中，且从栈顶到(u, v)恰存储了顶点b表示的块的边集。

算法 4.1: DFSBlk 算法伪代码

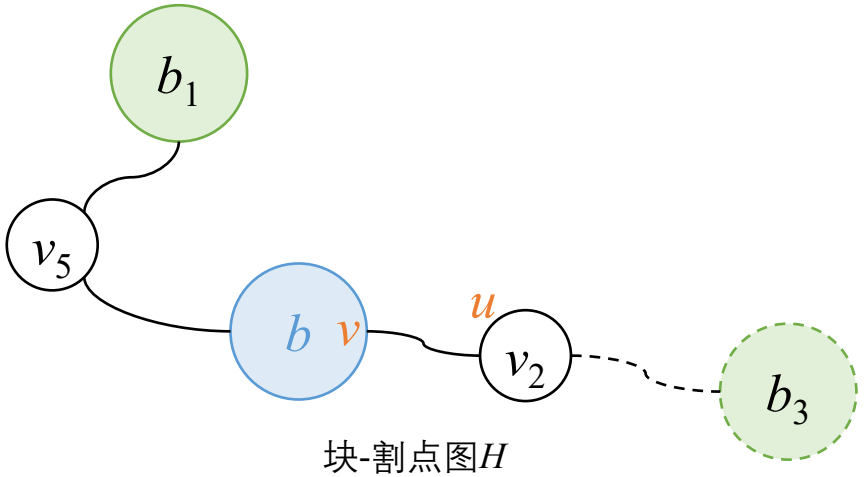
输入: 非平凡连通图 $G = \langle V, E \rangle$，顶点 u

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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



图G



块-割点图H
（实线：T）

定理4.5

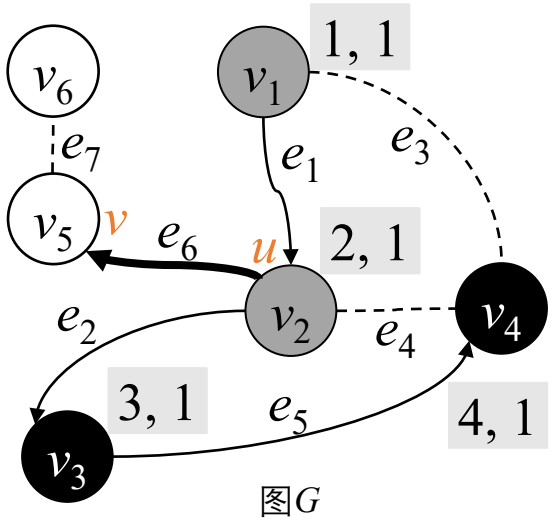
- 在上述时间段的结束时刻，边(u, v)在栈S中，且从栈顶到(u, v)恰存储了顶点b表示的块的边集。
  - 采用数学归纳法，对上述时间段内发现的边的集合E_1的大小进行归纳。
    - 当$|E_1| = 1$时，$E_1 = \{(u, v)\}$，成立。
    - 假设$|E_1| \leqslant k$时成立，则$|E_1| = k + 1$时，由归纳假设，集合$B_T \setminus \{b\}$中的顶点表示的所有块的边集均已出栈并输出，得证。

算法 4.1: DFSBlk 算法伪代码

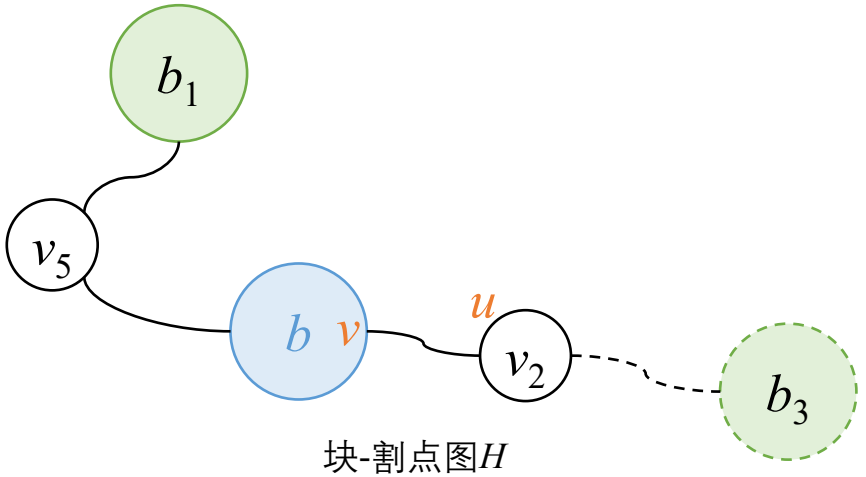
输入: 非平凡连通图 $G = \langle V, E\rangle$，顶点 u

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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



图G



块-割点图H
（实线：T）

定理4.5

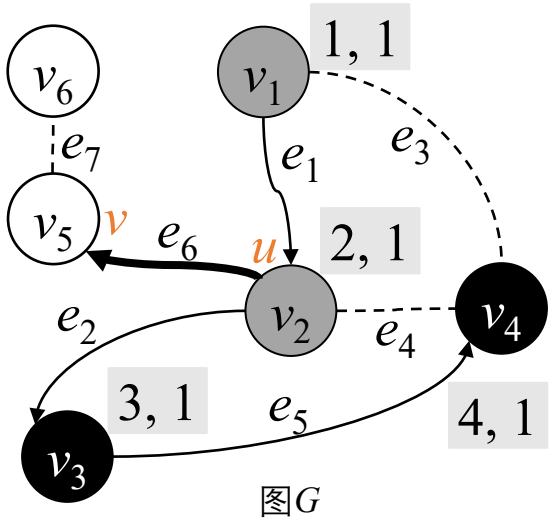
- 在上述时间段的结束时刻，边(u, v)在栈S中，且从栈顶到(u, v)恰存储了顶点b表示的块的边集。
- 因此，在上述时间段的结束时刻，DFSBlk算法出栈并且输出的正是顶点b表示的块的边集。

算法 4.1: DFSBlk 算法伪代码

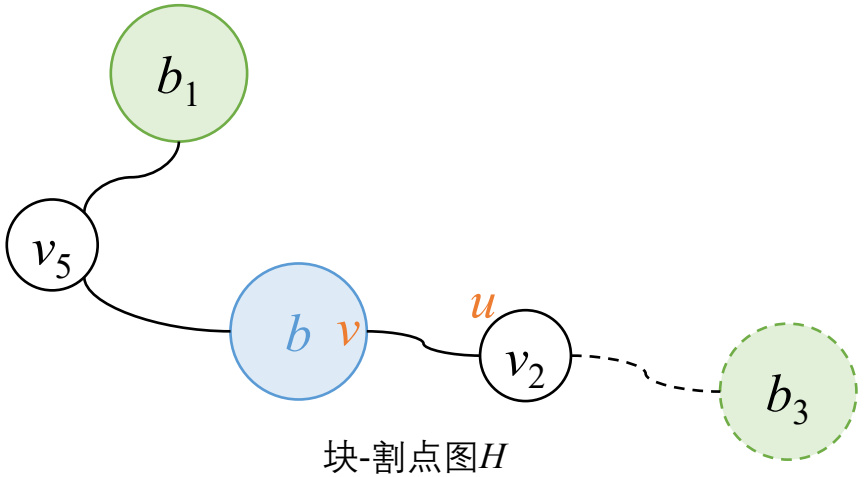
输入: 非平凡连通图 $G = \langle V, E \rangle$，顶点 u

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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```



图G



块-割点图H
（实线：T）

DFSBlk算法

- 时间复杂度：$O(n+m)$

算法 4.1: DFSBlk 算法伪代码

输入: 非平凡连通图 $G=\langle V,E\rangle$，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；栈 S 初值为空

```
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          入栈 (S, (u, v));
8          v.parent ← u;
9          u.children ← u.children + 1;
10         DFSBlk(G, v);
11         u.low ← min{u.low, v.low};
12         if (u.parent = null 且 u.children ⩾ 2) 或 (u.parent ≠ null 且 v.low ⩾ u.d) then
13             输出 (以下边组成一个块);
14             do
15                 (x, y) ← 出栈 (S);
16                 输出 ((x, y));
17             while (x, y) ≠ (u, v);
18     else if v ≠ u.parent then
19         if u.d > v.d then
20             入栈 (S, (u, v));
21         u.low ← min{u.low, v.d};
```

请认真完成课后练习