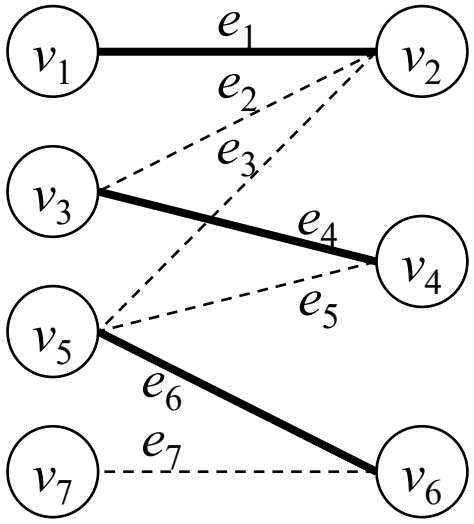
第5章 匹配

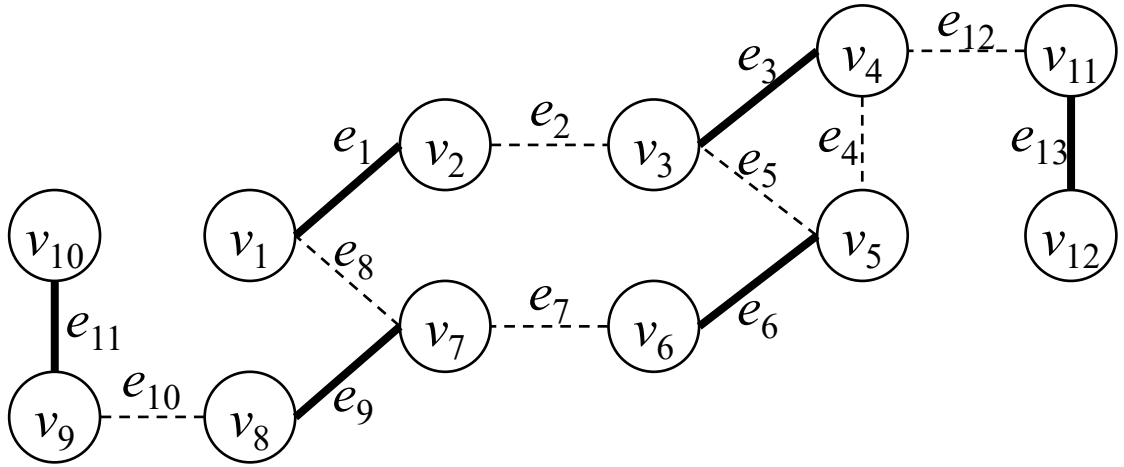
程龚
南京大学 计算机学院
gcheng@nju.edu.cn
http://ws.nju.edu.cn/~gcheng

本章内容

- 第5.1节 匹配和最大匹配
  - 第5.1.1节 理论
  - **第5.1.2节 算法**
- 第5.2节 完美匹配

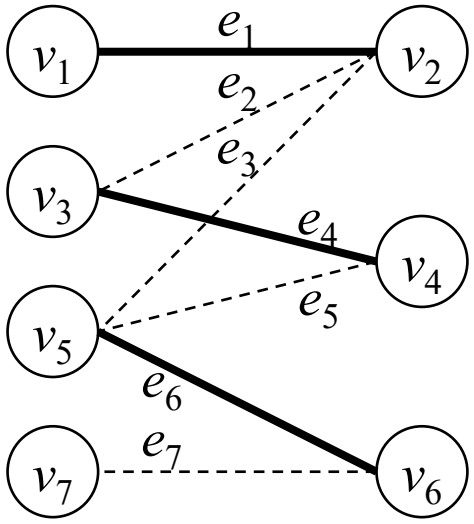
如何找出图中的最大匹配?

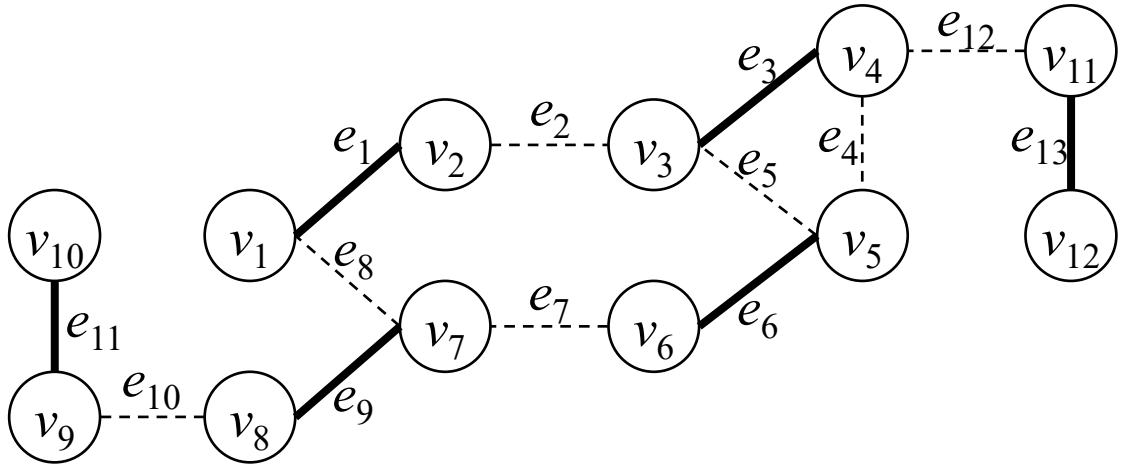




如何找出图中的最大匹配?

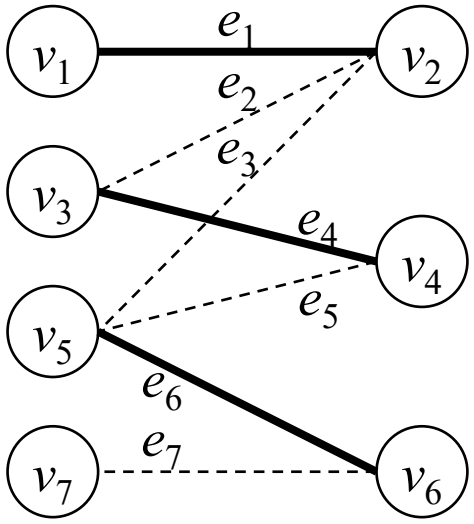
- 对于二分图
  1. 匈牙利算法
  2. 霍普克罗夫特-卡普算法
- 对于非二分图
  3. 花算法

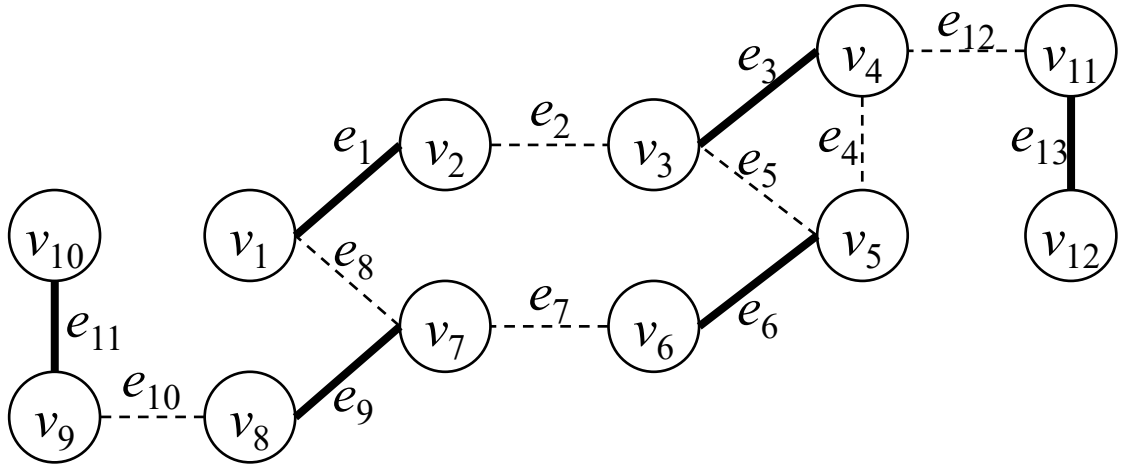




如何找出图中的最大匹配?

- 对于二分图
  1. **匈牙利算法**
  2. 霍普克罗夫特-卡普算法
- 对于非二分图
  3. 花算法





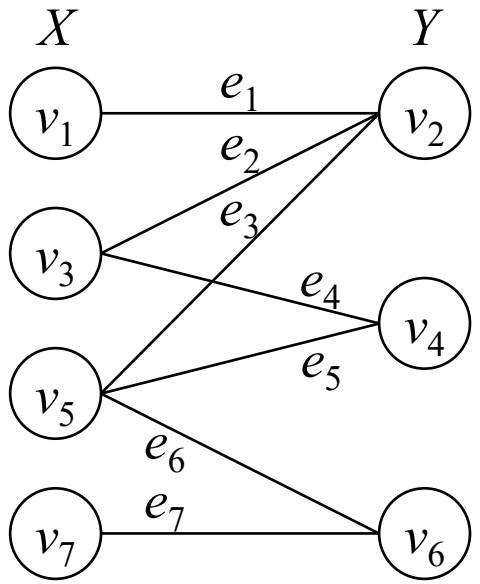
匈牙利算法

- 基本思路：逐步构造最大匹配，每步利用当前匹配的一条增广路得到一个更大的匹配。

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$
初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



匈牙利算法

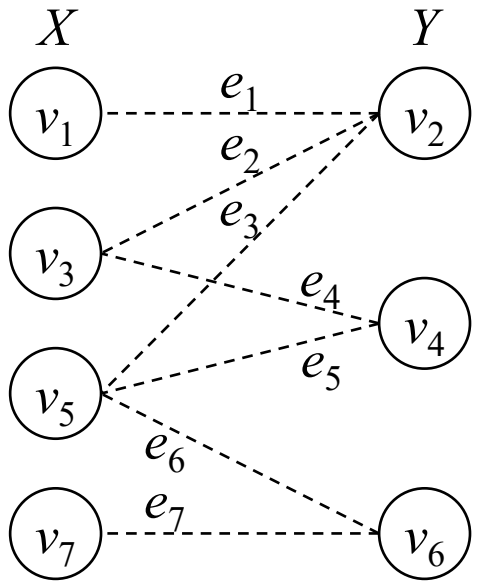
- 从初值为空集的匹配M开始

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



匈牙利算法

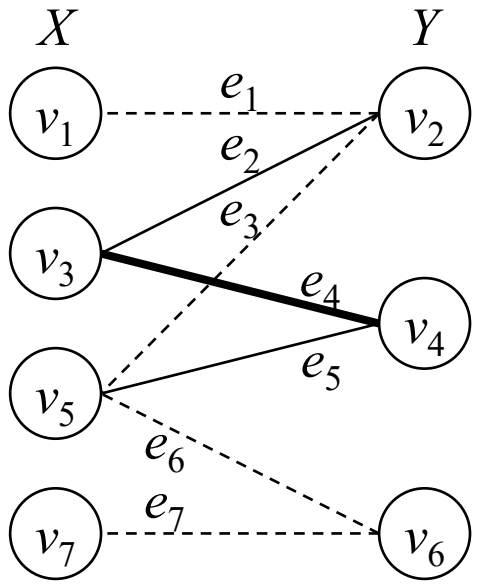
- 每轮do-while循环尝试找一条M增广路P，直至G中不存在M增广路，即P为null:

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



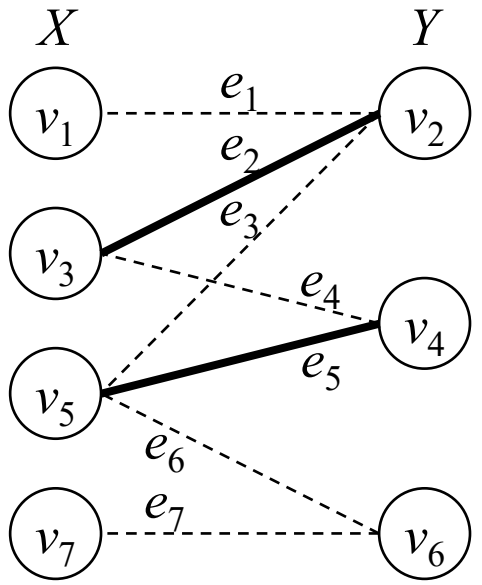
匈牙利算法

- 每轮do-while循环尝试找一条M增广路P，直至G中不存在M增广路，即P为null：
  - 若能找到，即P不为null，则计算P经过的边的集合和M的对称差，得到一个包含边的数量更多的匹配，本轮尝试提前中止并进入下轮尝试

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$
初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



匈牙利算法

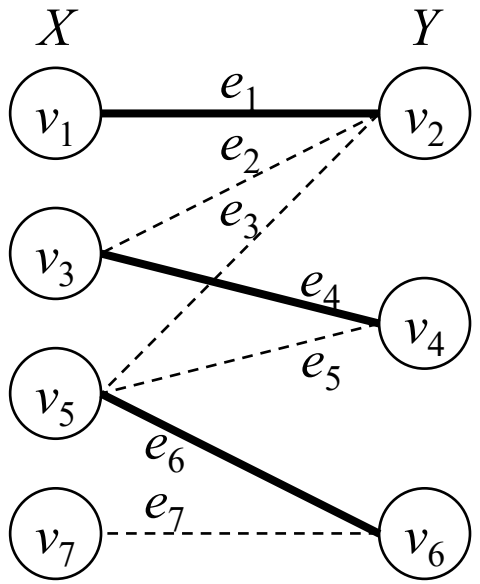
- 算法运行结束时，输出最大匹配M。

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



匈牙利算法

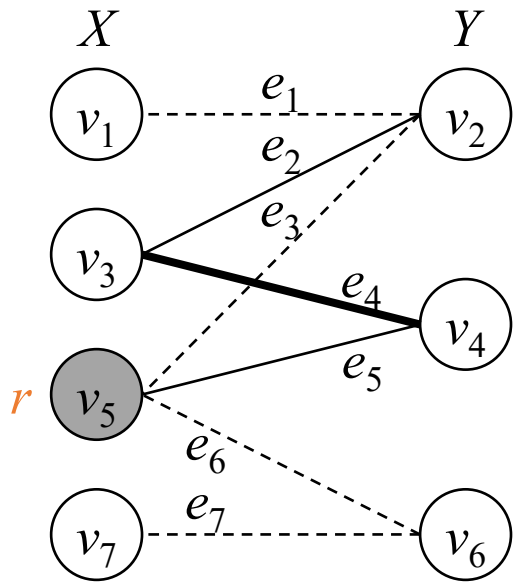
- 找M增广路的方法是调用DFSAP算法，这是一种扩展的DFS算法。
  - 从顶点子集X中每个在本轮do-while循环中未被DFSAP算法访问过（即visited属性值为false）且未被M饱和的顶点r出发运行DFSAP算法，尝试找一条以r为起点的M增广路P。

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



思考题5.11

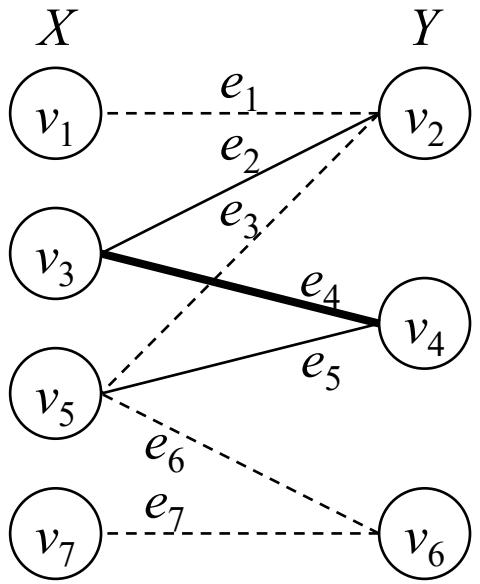
- do-while循环运行多少轮?

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



思考题5.11

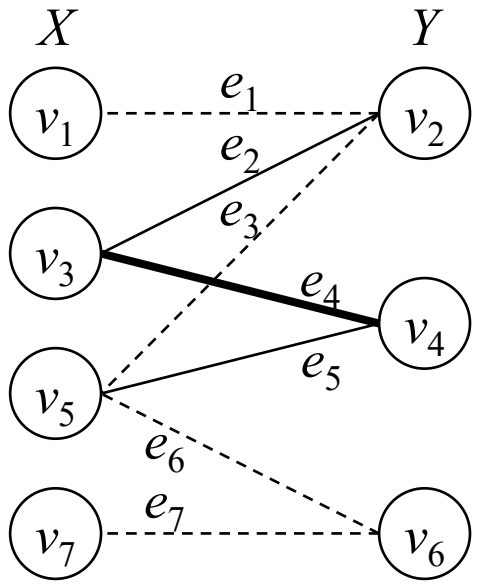
- do-while循环运行多少轮？
  - 最大匹配的大小 + 1

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



匈牙利算法

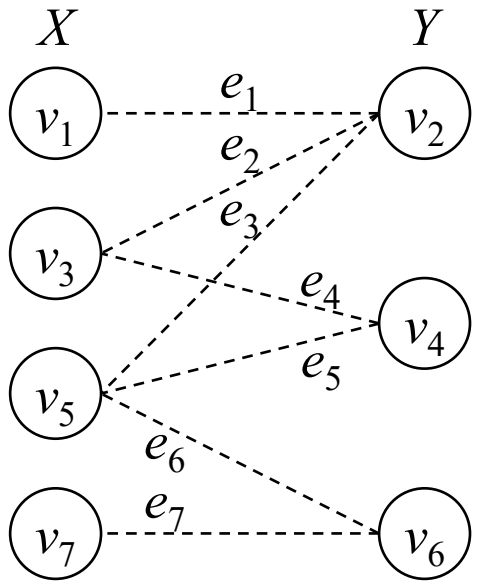
- 例如：第1轮do-while循环
  - 开始前：$M \leftarrow \emptyset$

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



匈牙利算法

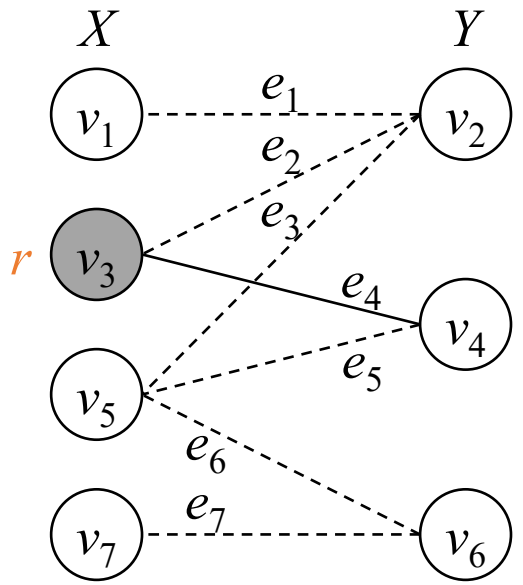
- 例如：第1轮do-while循环
  - 开始前：$M \leftarrow \emptyset$
  - 开始后：$P \leftarrow v_3, v_4$

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



匈牙利算法

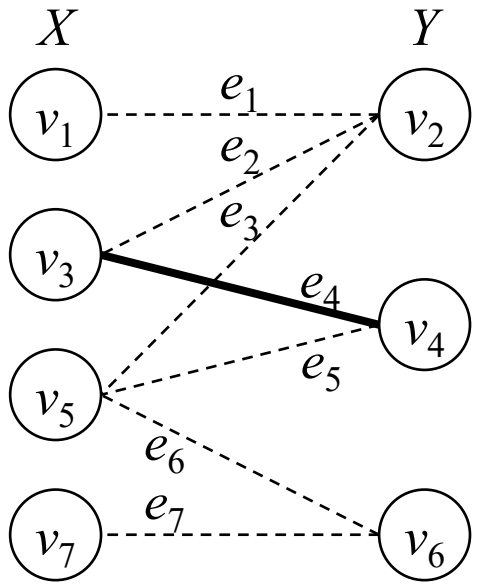
- 第2轮do-while循环
  - 开始前：$M \leftarrow \{e_4\}$

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



匈牙利算法

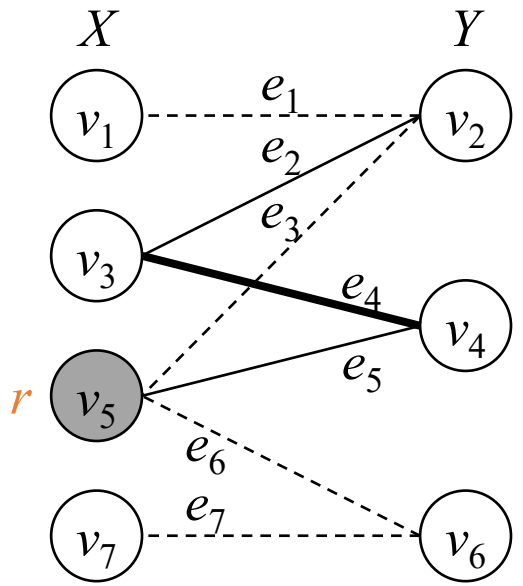
- 第2轮do-while循环
  - 开始前：$M \leftarrow \{e_4\}$
  - 开始后：$P \leftarrow v_5, v_4, v_3, v_2$

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



匈牙利算法

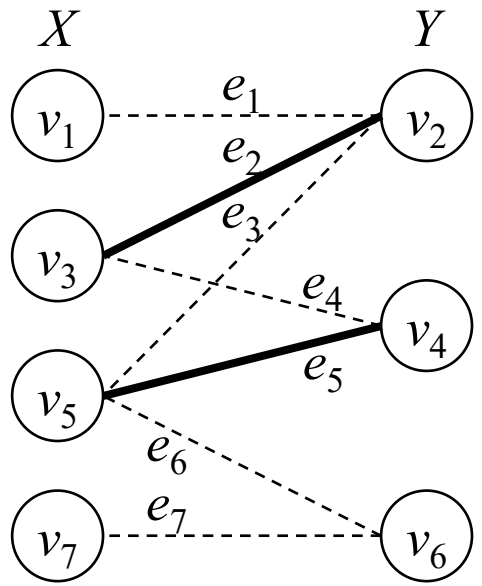
- 第3轮do-while循环
  - 开始前：$M \leftarrow \{e_2, e_5\}$

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



匈牙利算法

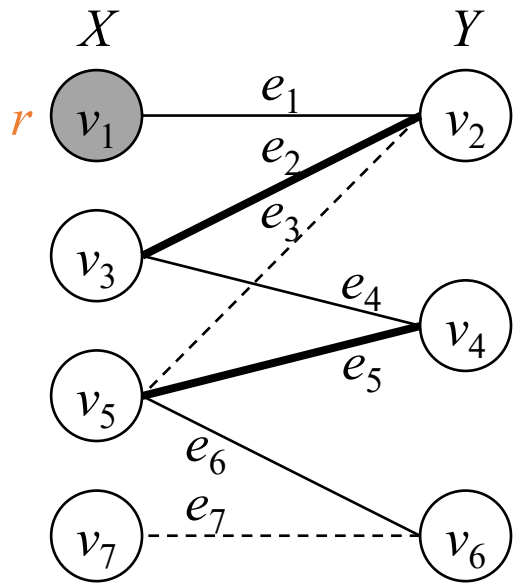
- 第3轮do-while循环
  - 开始前：$M \leftarrow \{e_2, e_5\}$
  - 开始后：$P \leftarrow v_1, v_2, v_3, v_4, v_5, v_6$

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



匈牙利算法

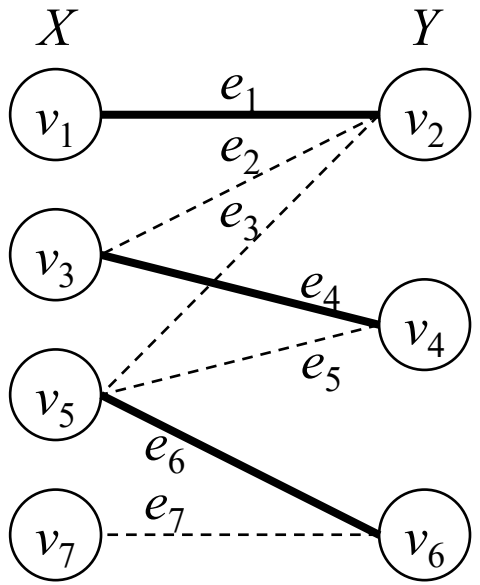
- 第4轮do-while循环
  - 开始前：$M \leftarrow \{e_1, e_4, e_6\}$

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



匈牙利算法

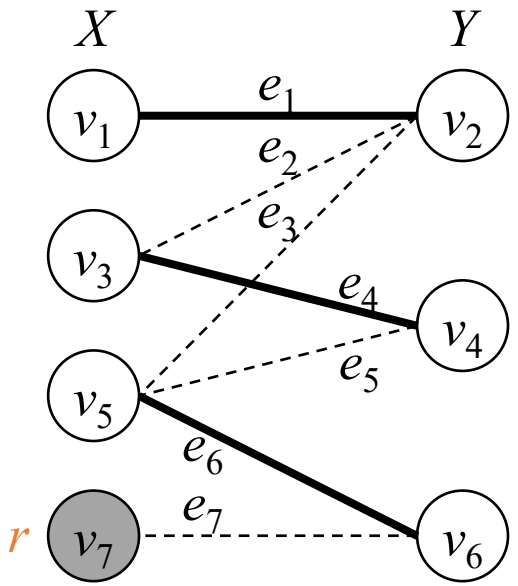
- 第4轮do-while循环
  - 开始前：$M \leftarrow \{e_1, e_4, e_6\}$
  - 开始后：$P \leftarrow$ null

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



匈牙利算法

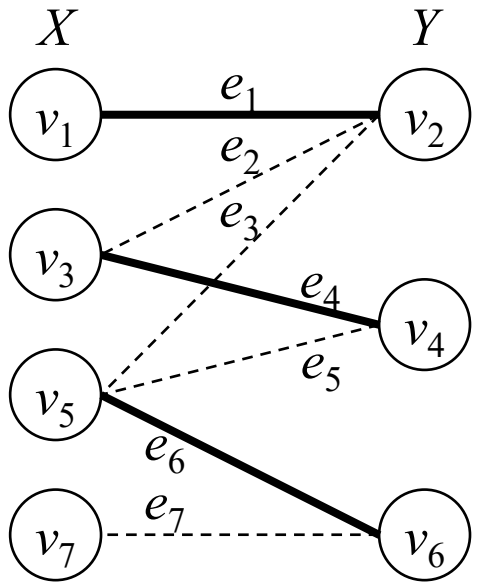
- 第4轮do-while循环
  - 开始前：$M \leftarrow \{e_1, e_4, e_6\}$
  - 开始后：$P \leftarrow$ null
- 算法运行结束，输出$M = \{e_1, e_4, e_6\}$

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



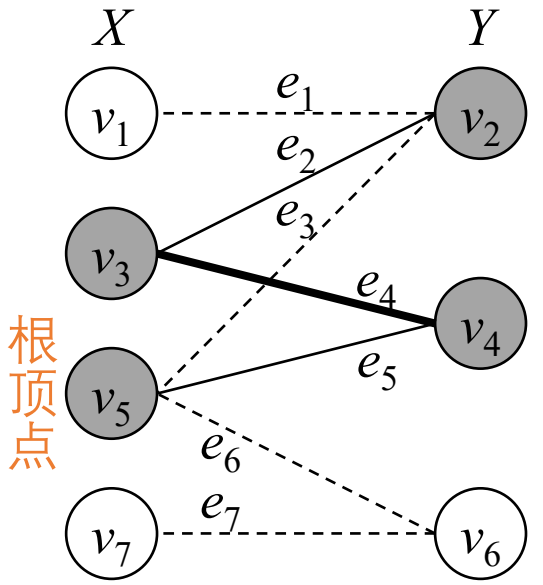
DFSAP算法（扩展DFS算法用于找增广路）

- 基本思路：从图中的一个指定顶点u出发，按DFS的方式有序地遍历图，并返回一条找到的M增广路，或返回null表示未找到。

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



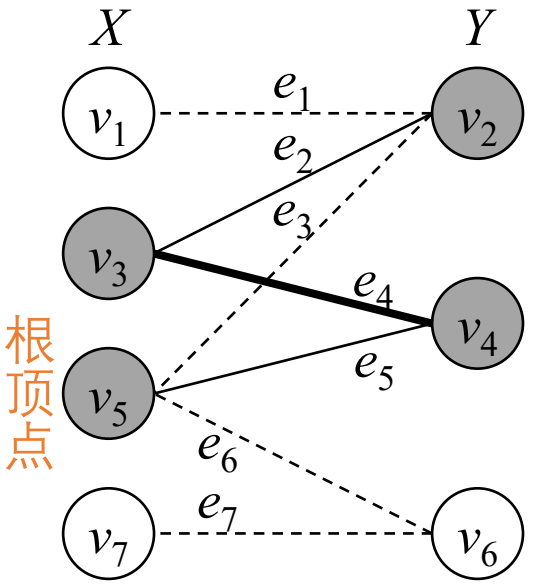
DFSAP算法

- 基于DFS算法

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2
3
4
5  foreach (u, v) ∈ E do
6      if v.visited = false
           then
7              DFSAP(G, v, M);
8
9
10
```



DFSAP算法

- 第1项扩展：限制了可访问的邻点，
  使DFS树中以根顶点（即匈牙利算法中的顶点r）为起点的每条路都是M交错路。

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

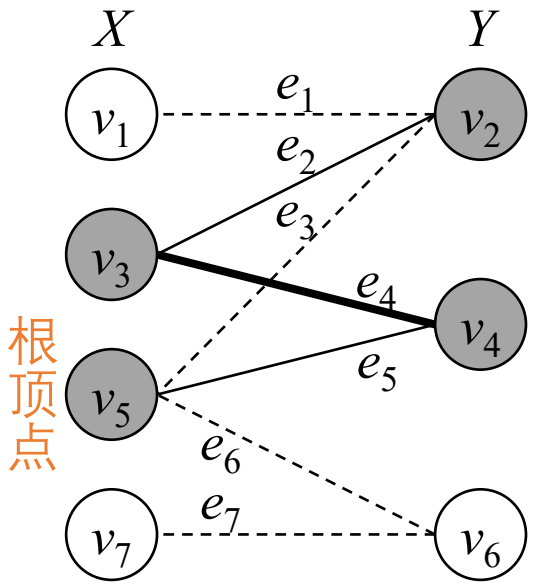
```
1  u.visited ← true;
2
3
4
5  foreach (u, v) ∈ E do
6      if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
        错路 then
7          DFSAP(G, v, M);
8
9
10
```

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



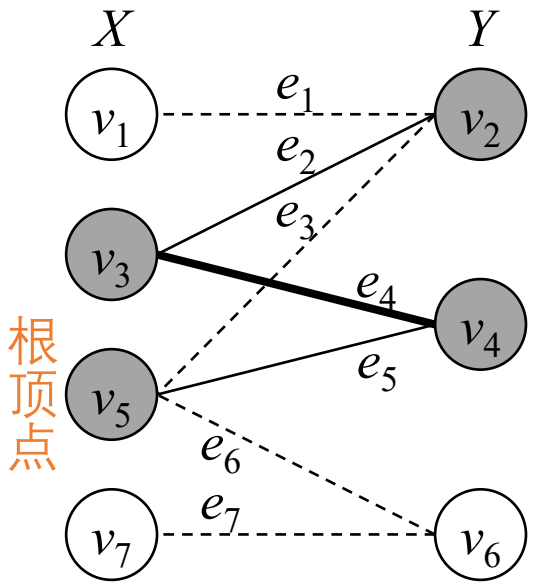
思考题5.12

- 如何高效地判定DFS树中从根顶点到顶点v的路是M交错路?

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2
3
4
5  foreach (u, v) ∈ E do
6      if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交错路 then
7          DFSAP(G, v, M);
8
9
10
```



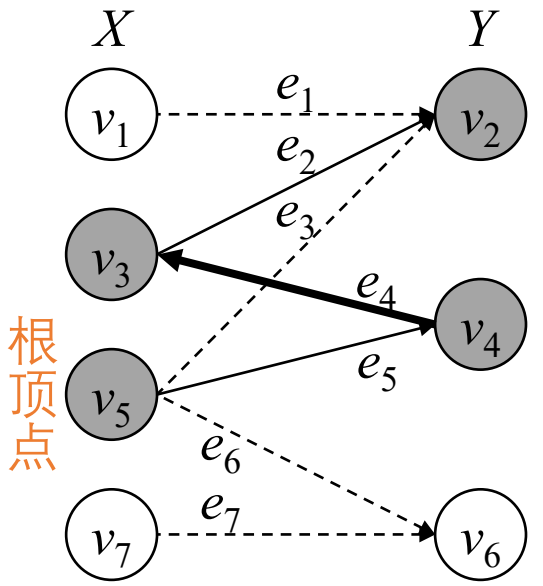
思考题5.12

- 如何高效地判定DFS树中从根顶点到顶点v的路是M交错路?
  - 为匹配M中的边赋予从集合Y中顶点到集合X中顶点的方向，
    为不在M中的边赋予从X中顶点到Y中顶点的方向，
    只按方向访问。

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2
3
4
5  foreach (u, v) ∈ E do
6      if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
        错路 then
7          DFSAP(G, v, M);
8
9
10
```



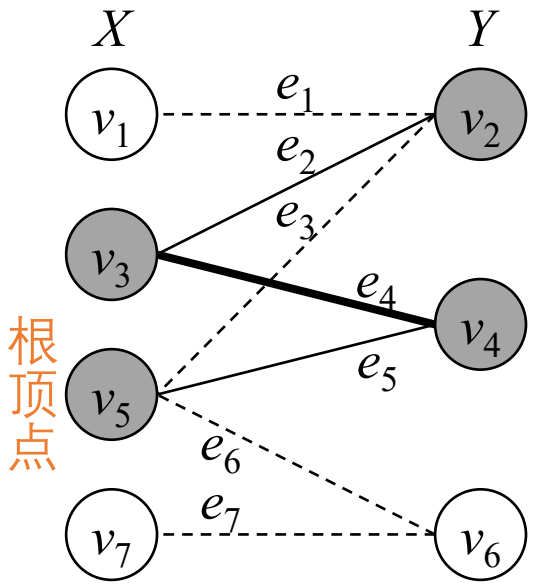
DFSAP算法

- 第2项扩展：判定并返回增广路。

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2
3
4
5  foreach (u, v) ∈ E do
6      if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
        错路 then
7          DFSAP(G, v, M);
8
9
10
```



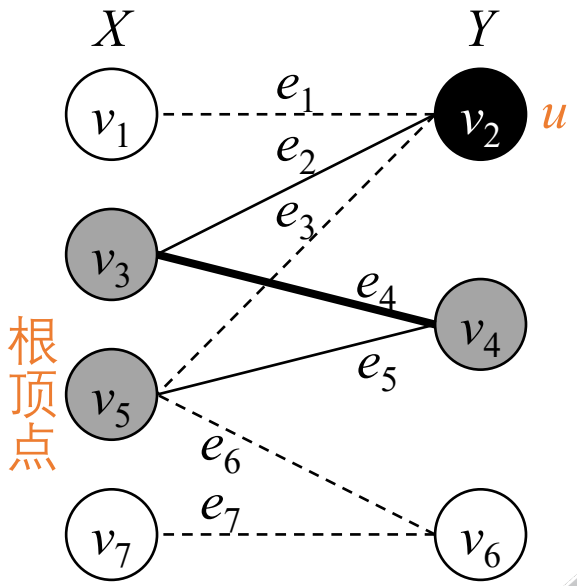
DFSAP算法

- 第2项扩展：判定并返回增广路。
  - 若非根顶点u未被M饱和，则DFS树中从根顶点到u的M交错路是M增广路，算法返回这条路；

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3      return DFS 树中从根顶点到 u 的路;
4  else
5      foreach (u, v) ∈ E do
6          if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
            错路 then
7              DFSAP(G, v, M);
8
9
10
```



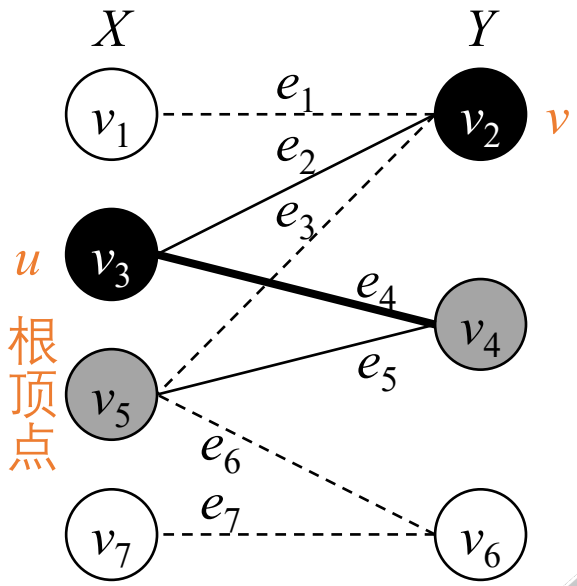
DFSAP算法

- 第2项扩展：判定并返回增广路。
  - 若非根顶点u未被M饱和，则DFS树中从根顶点到u的M交错路是M增广路，算法返回这条路；
  - 否则，若从u对邻点v的递归调用找到M增广路P_v，则算法返回P_v；

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |
```



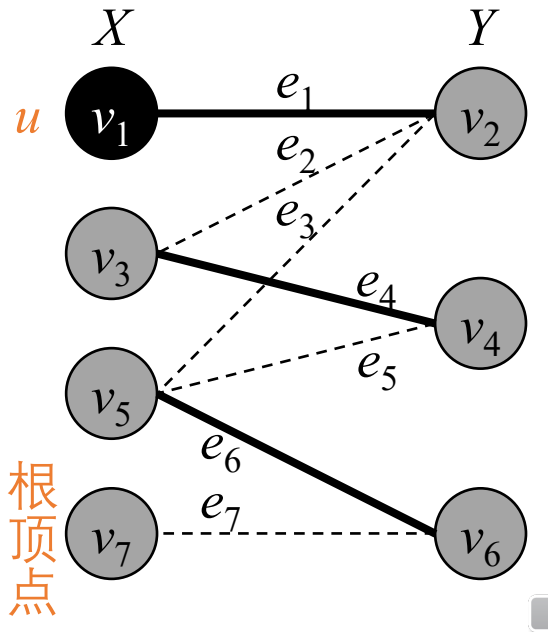
DFSAP算法

- 第2项扩展：判定并返回增广路。
  - 若非根顶点u未被M饱和，则DFS树中从根顶点到u的M交错路是M增广路，算法返回这条路；
  - 否则，若从u对邻点v的递归调用找到M增广路P_v，则算法返回P_v；
  - 否则，未找到M增广路，算法返回null。

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



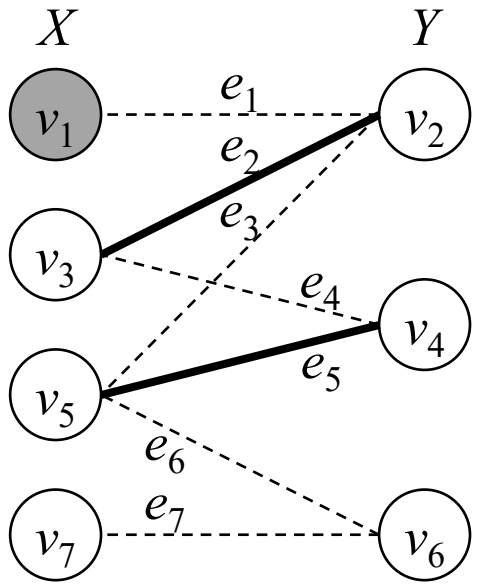
DFSAP算法

- 例如：从v_1出发，调用DFSAP(G, v_1, M)
  - 判断v_2.visited为false 且 从v_1到v_2的路是M交错路
  - 递归调用DFSAP(G, v_2, M)

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |    错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



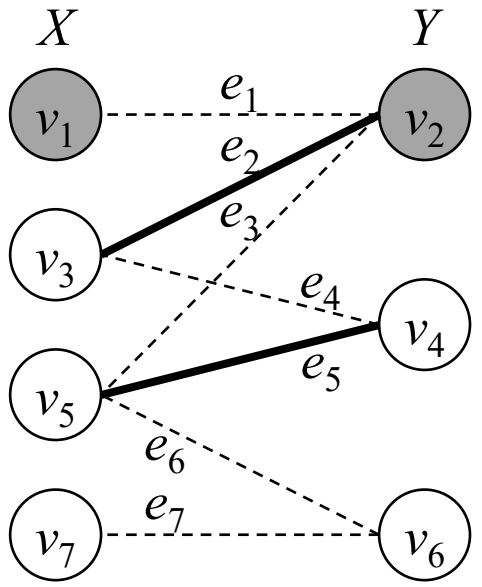
DFSAP算法

- 递归调用DFSAP(G, v_2, M)
  - 判断v_3.visited为false 且 从v_1到v_3的路是M交错路
  - 递归调用DFSAP(G, v_3, M)

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |    错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



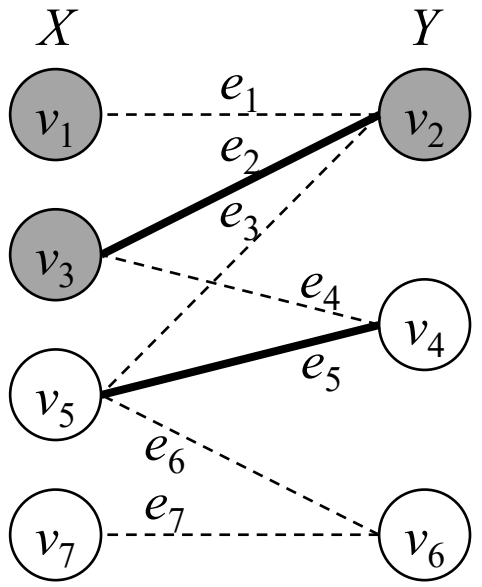
DFSAP算法

- 递归调用DFSAP(G, v_3, M)
  - 判断v_4.visited为false 且 从v_1到v_4的路是M交错路
  - 递归调用DFSAP(G, v_4, M)

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |    错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



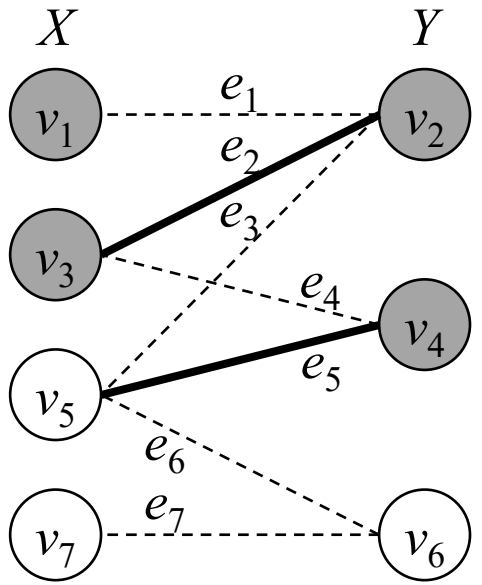
DFSAP算法

- 递归调用DFSAP(G, v_4, M)
  - 判断v_5.visited为false 且 从v_1到v_5的路是M交错路
  - 递归调用DFSAP(G, v_5, M)

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |    错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



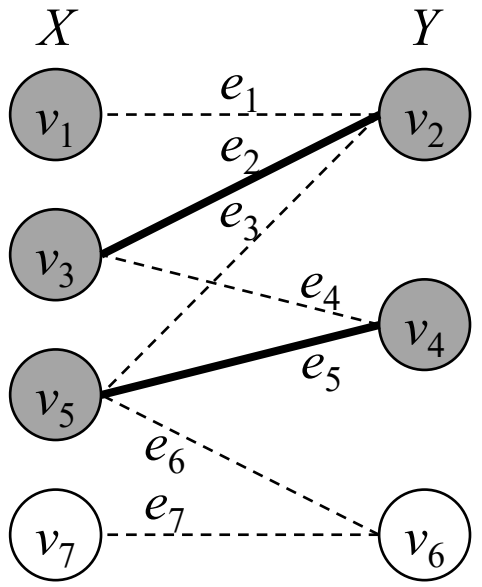
DFSAP算法

- 递归调用DFSAP(G, v_5, M)
  - 判断v_6.visited为false 且 从v_1到v_6的路是M交错路
  - 递归调用DFSAP(G, v_6, M)

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |    错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



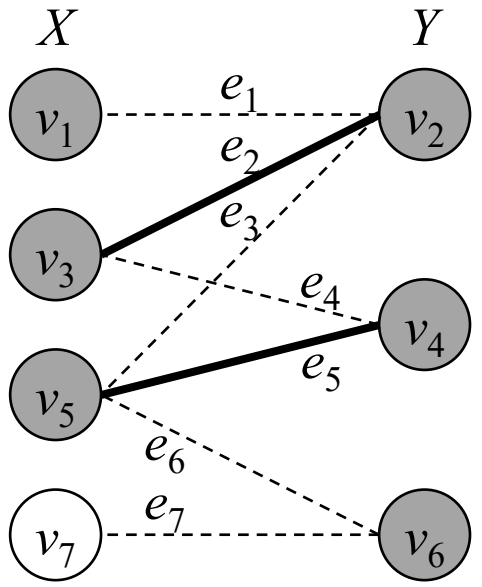
DFSAP算法

- 递归调用DFSAP(G, v_6, M)
  - 返回路v_1, v_2, v_3, v_4, v_5, v_6

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



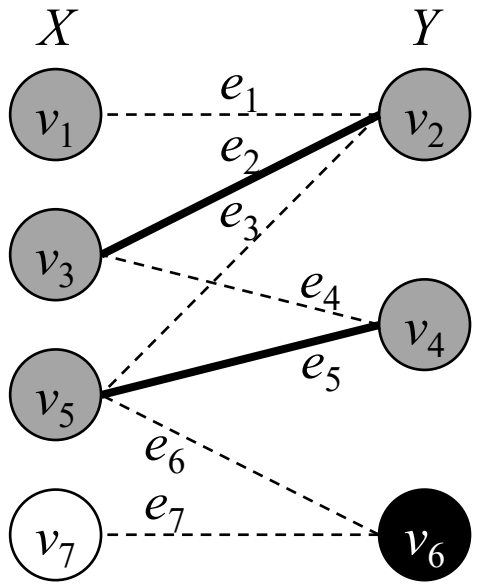
DFSAP算法

- DFSAP(G, v_6, M)结束
- DFSAP(G, v_5, M)尚未结束
  - 返回路v_1, v_2, v_3, v_4, v_5, v_6

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



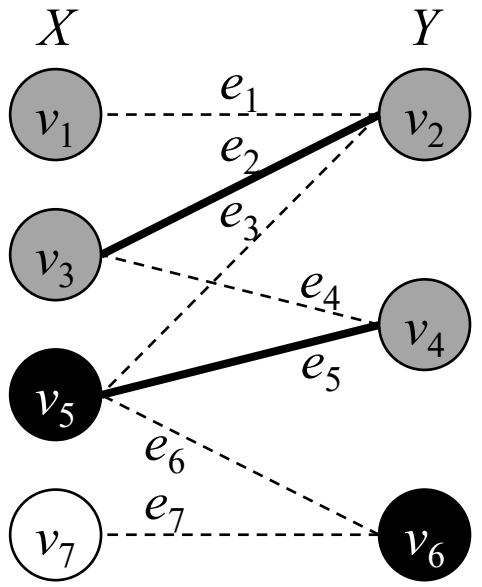
DFSAP算法

- DFSAP(G, v_5, M)结束
- DFSAP(G, v_4, M)尚未结束
  - 返回路v_1, v_2, v_3, v_4, v_5, v_6

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



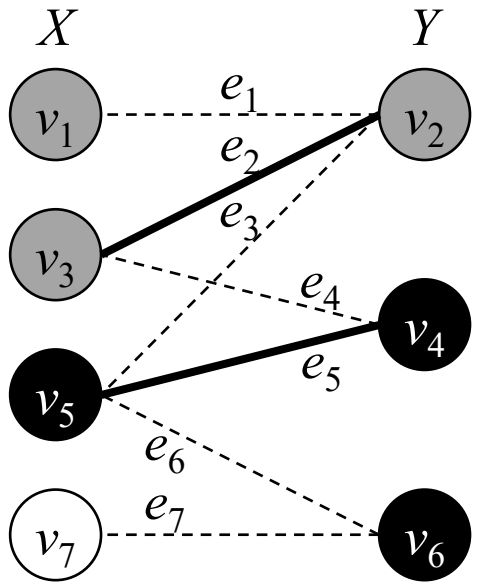
DFSAP算法

- DFSAP(G, v_4, M)结束
- DFSAP(G, v_3, M)尚未结束
  - 返回路v_1, v_2, v_3, v_4, v_5, v_6

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



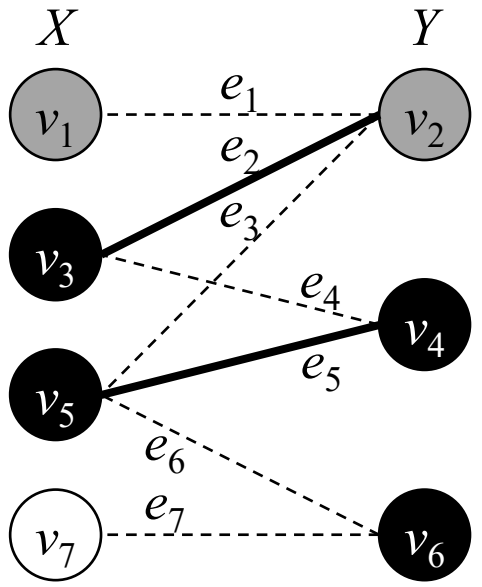
DFSAP算法

- DFSAP(G, v_3, M)结束
- DFSAP(G, v_2, M)尚未结束
  - 返回路v_1, v_2, v_3, v_4, v_5, v_6

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



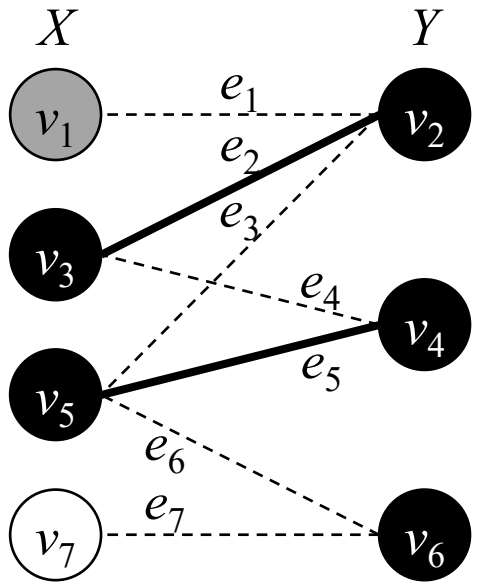
DFSAP算法

- DFSAP(G, v_2, M)结束
- DFSAP(G, v_1, M)尚未结束
  - 返回路v_1, v_2, v_3, v_4, v_5, v_6

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



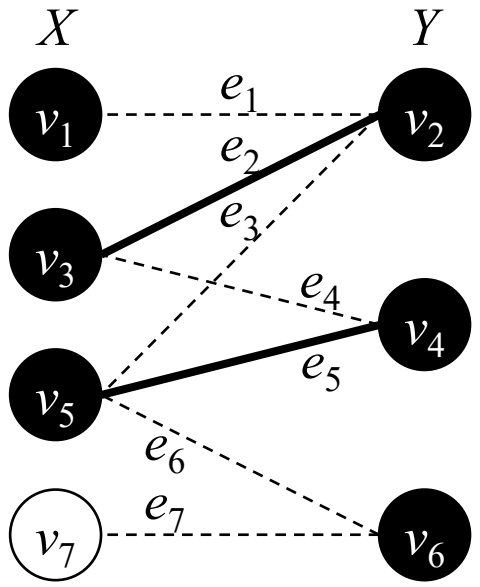
DFSAP算法

- DFSAP(G, v_1, M)结束

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



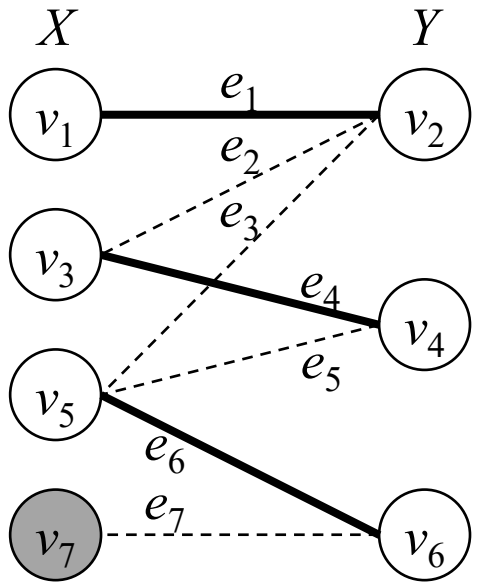
DFSAP算法

- 从v_7出发，调用DFSAP(G, v_7, M)
  - 判断v_6.visited为false 且 从v_7到v_6的路是M交错路
  - 递归调用DFSAP(G, v_6, M)

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |    错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



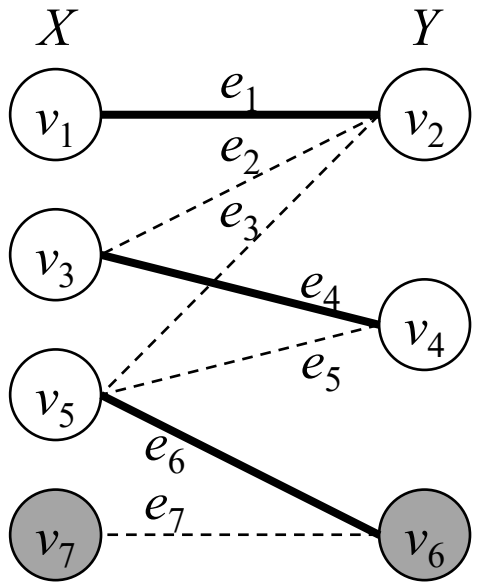
DFSAP算法

- 递归调用DFSAP(G, v_6, M)
  - 判断v_5.visited为false 且 从v_7到v_5的路是M交错路
  - 递归调用DFSAP(G, v_5, M)

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |    错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



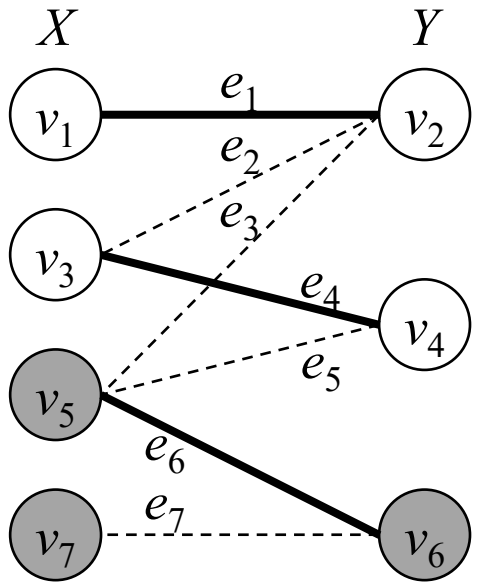
DFSAP算法

- 递归调用DFSAP(G, v_5, M)
  - 判断v_2.visited为false 且 从v_7到v_2的路是M交错路
  - 递归调用DFSAP(G, v_2, M)

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |    错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



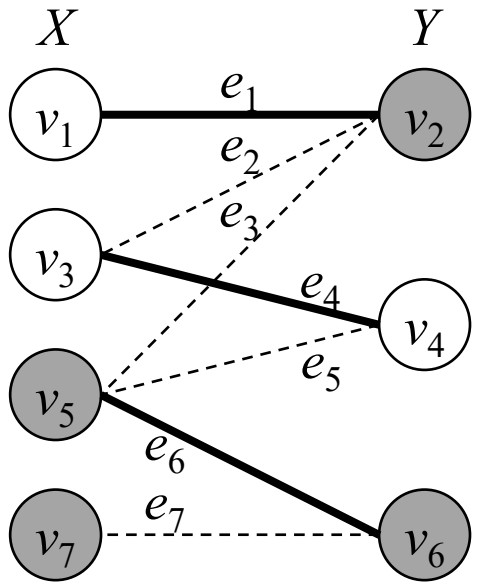
DFSAP算法

- 递归调用DFSAP(G, v_2, M)
  - 判断v_1.visited为false 且 从v_7到v_1的路是M交错路
  - 递归调用DFSAP(G, v_1, M)

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |    错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



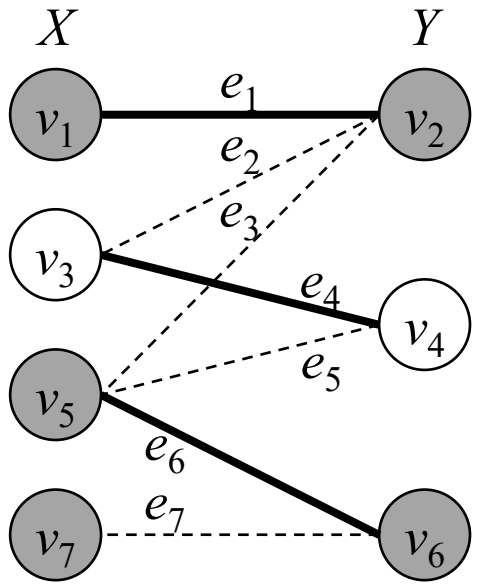
DFSAP算法

- 递归调用DFSAP(G, v_1, M)
  - 返回null

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



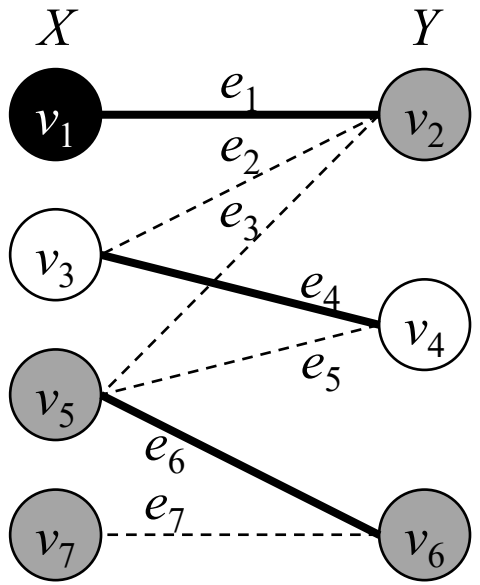
DFSAP算法

- DFSAP(G, v_1, M)结束
- DFSAP(G, v_2, M)尚未结束
  - 返回null

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



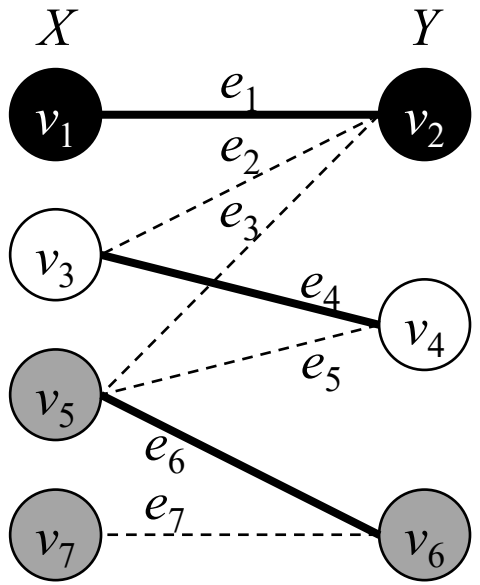
DFSAP算法

- DFSAP(G, v_2, M)结束
- DFSAP(G, v_5, M)尚未结束
  - 判断v_4.visited为false 且 从v_7到v_4的路是M交错路
  - 递归调用DFSAP(G, v_4, M)

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |    错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



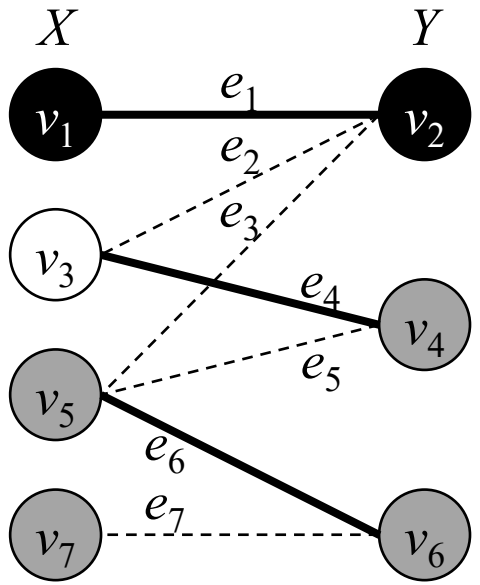
DFSAP算法

- 递归调用DFSAP(G, v_4, M)
  - 判断v_3.visited为false 且 从v_7到v_3的路是M交错路
  - 递归调用DFSAP(G, v_3, M)

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |    错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



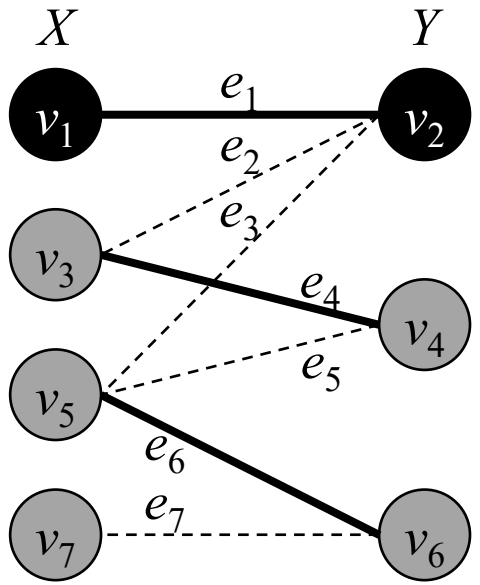
DFSAP算法

- 递归调用DFSAP(G, v_3, M)
  - 返回null

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



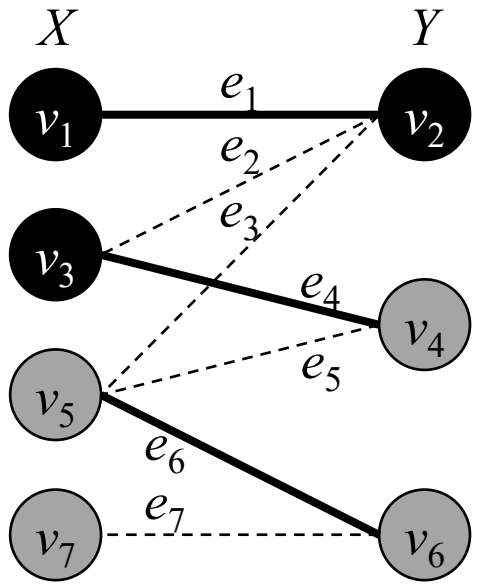
DFSAP算法

- DFSAP(G, v_3, M)结束
- DFSAP(G, v_4, M)尚未结束
  - 返回null

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



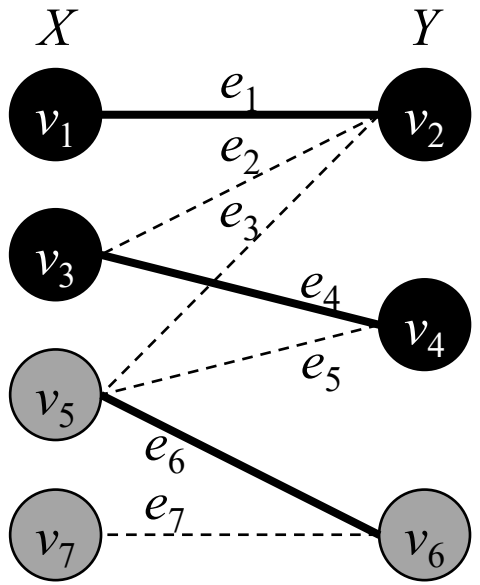
DFSAP算法

- DFSAP(G, v_4, M)结束
- DFSAP(G, v_5, M)尚未结束
  - 返回null

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



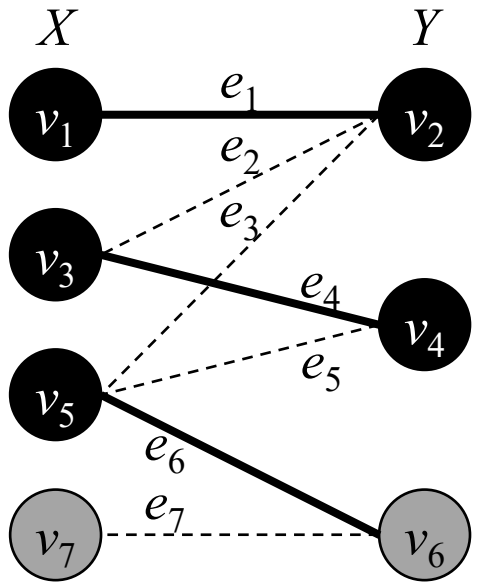
DFSAP算法

- DFSAP(G, v_5, M)结束
- DFSAP(G, v_6, M)尚未结束
  - 返回null

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



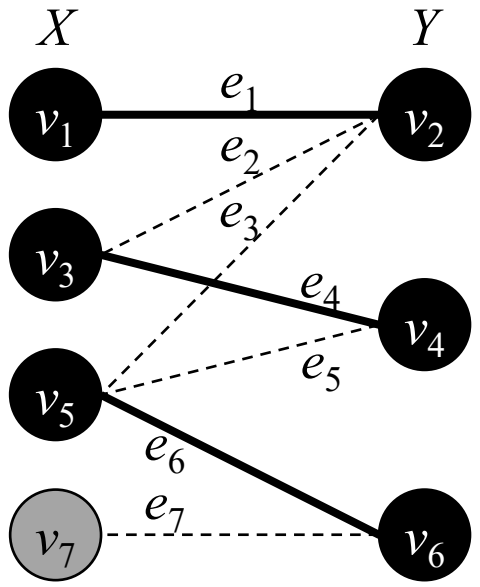
DFSAP算法

- DFSAP(G, v_6, M)结束
- DFSAP(G, v_7, M)尚未结束
  - 返回null

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



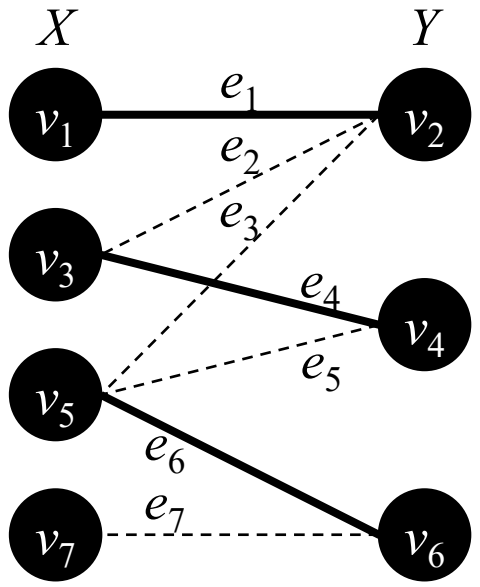
DFSAP算法

- DFSAP(G, v_7, M)结束

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



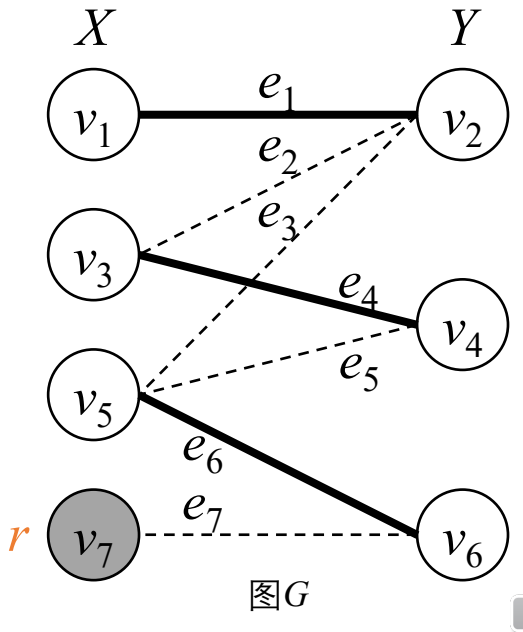
思考题5.13

- 对顶点r调用DFSAP算法返回null时，是否已尝试所有以r为起点的M交错路？会遗漏M增广路吗？

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



图G

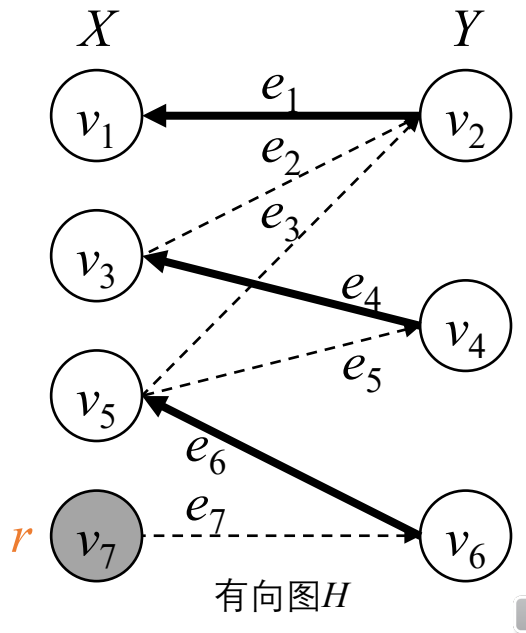
思考题5.13

- 对顶点r调用DFSAP算法返回null时，是否已尝试所有以r为起点的M交错路？会遗漏M增广路吗？
  - 考虑图G的定向H:
    - 对于集合$E \setminus M$中的每条边，以其在顶点子集X中的端点为尾、在顶点子集Y中的端点为头形成一条弧；
    - 对于匹配M中的每条边，以其在Y中的端点为尾、在X中的端点为头形成一条弧。
  - 因此，
    - G中的交错路 对应 H中的有向路，
      在G中对顶点r调用DFSAP算法 对应 在H中对r调用适用于有向图的DFS算法，
      DFSAP算法返回null当且仅当未被M饱和的所有顶点（除r外）在H中从r都不可达，对应 G不含以r为起点的M增广路。

算法 5.2: DFSAP

输入: 图 $G = \langle X \cup Y, E \rangle$，顶点 u，匹配 M

```
1  u.visited ← true;
2  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
3  |  return DFS 树中从根顶点到 u 的路;
4  else
5  |  foreach (u, v) ∈ E do
6  |  |  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M 交
   |  |   错路 then
7  |  |  |  P_v ← DFSAP(G, v, M);
8  |  |  |  if P_v ≠ null then
9  |  |  |  |  return P_v;
10 |  return null;
```



有向图H

思考题5.14

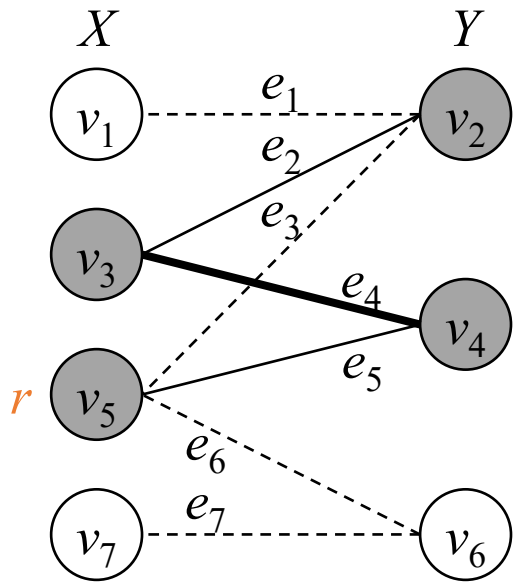
- 只从顶点子集X中的顶点出发运行DFSAP算法，会遗漏M增广路吗？

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



思考题5.14

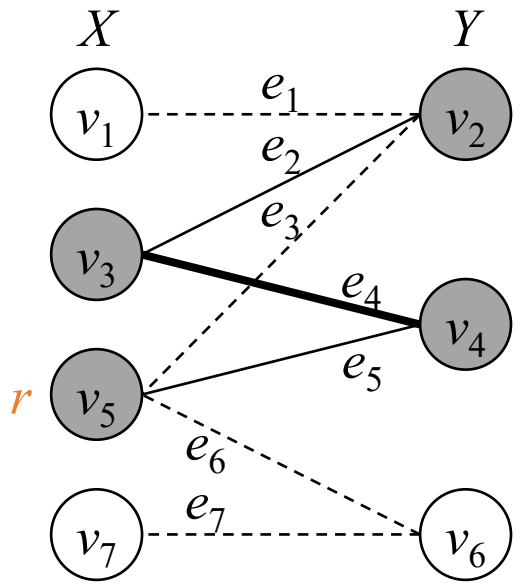
- 只从顶点子集X中的顶点出发运行DFSAP算法，会遗漏M增广路吗？
  - M增广路的长度是奇数，起点和终点分属于顶点子集X和Y，
    因此，若只从X中的顶点出发运行DFSAP算法找不到M增广路，
    则从Y中的顶点出发运行DFSAP算法也找不到M增广路。

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



定理5.2

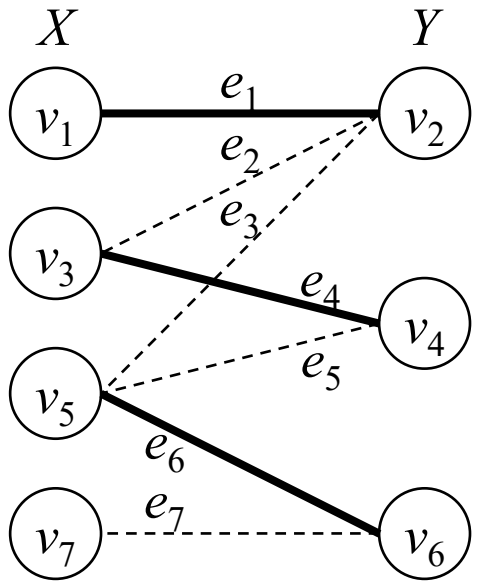
- 匈牙利算法输出的集合M是图G的最大匹配。

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



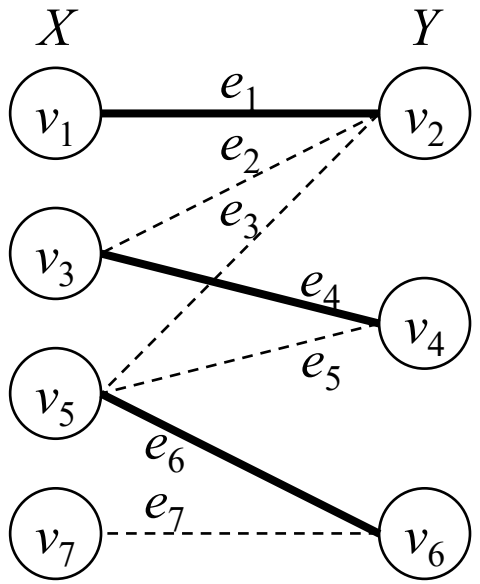
定理5.2

- 匈牙利算法输出的集合M是图G的最大匹配。
  - 先证集合M是G的匹配:
  - 再证匹配M是G的最大匹配:

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$
初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



定理5.2

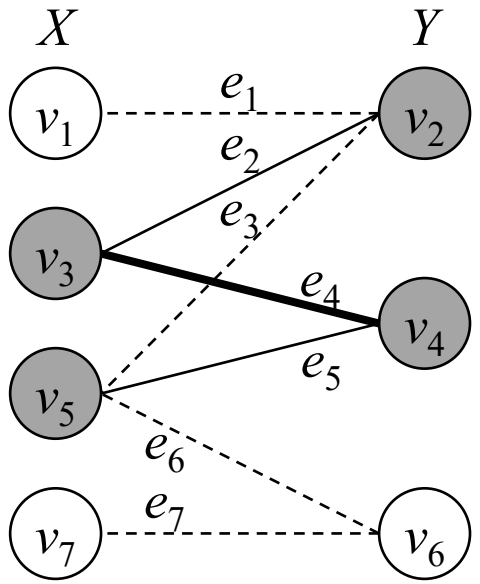
- 匈牙利算法输出的集合M是图G的最大匹配。
  - 先证集合M是G的匹配：
    - 第1轮do-while循环开始前，成立。
    - 若本轮do-while循环开始前成立，则本轮do-while循环计算M增广路P经过的边的集合和M的对称差仍是匹配，因此，下轮do-while循环条件判定前仍成立。
  - 再证匹配M是G的最大匹配：

思考题5.9 如何利用增广路得到一个更大的匹配？

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$
初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```



定理5.2

- 匈牙利算法输出的集合M是图G的最大匹配。
  - 先证集合M是G的匹配;
  - 再证匹配M是G的最大匹配:
    - 采用反证法，假设匈牙利算法运行结束时，M不是最大匹配，则由贝尔热定理，G含M增广路，因此，最后一轮do-while循环开始前，G含M增广路。
    - 由思考题5.13和思考题5.14，最后一轮do-while循环结束时，路P不为null，矛盾。

算法 5.1: 匈牙利算法

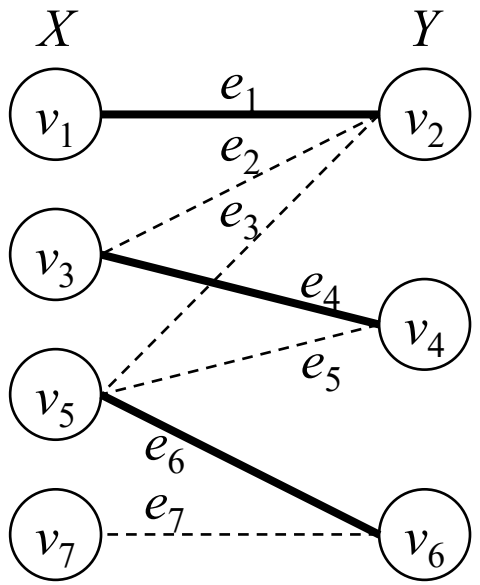
输入: 二分图 $G = \langle X \cup Y, E \rangle$

初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```

思考题5.13 对顶点r调用DFSAP算法返回null时，是否已尝试所有以r为起点的M交错路？会遗漏M增广路吗？

思考题5.14 只从顶点子集X中的顶点出发运行DFSAP算法，会遗漏M增广路吗？



匈牙利算法

- 时间复杂度：$O(m(n+m))$
  - 每轮do-while循环：$O(n+m)$
  - 循环的轮数：$O(m)$

思考题5.11 do-while循环运行多少轮？
最大匹配的大小 + 1

算法 5.1: 匈牙利算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$
初值: 集合 M 初值为 $\emptyset$

```
1  do
2      foreach u ∈ (X ∪ Y) do
3          u.visited ← false;
4      foreach r ∈ X do
5          if r.visited = false 且 r 未被 M 饱和 then
6              P ← DFSAP(G, r, M);
7              if P ≠ null then
8                  M ← P经过的边的集合 △ M;
9                  中止 foreach 循环;
10 while P ≠ null;
11 输出 (M);
```

接下来进入其它算法部分