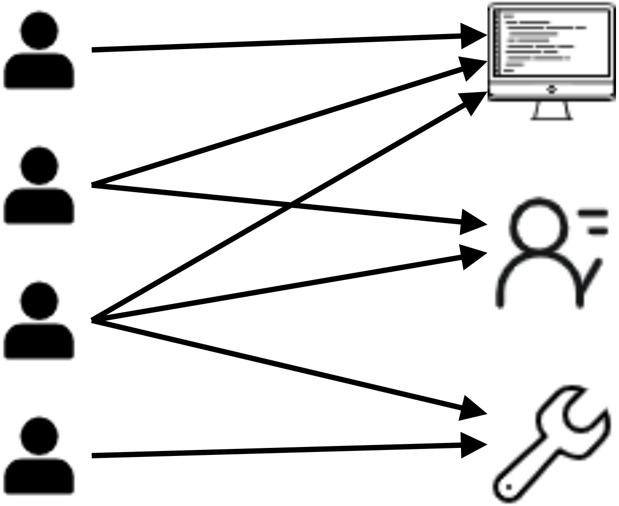
NANJING UNIVERSITY

Websoft

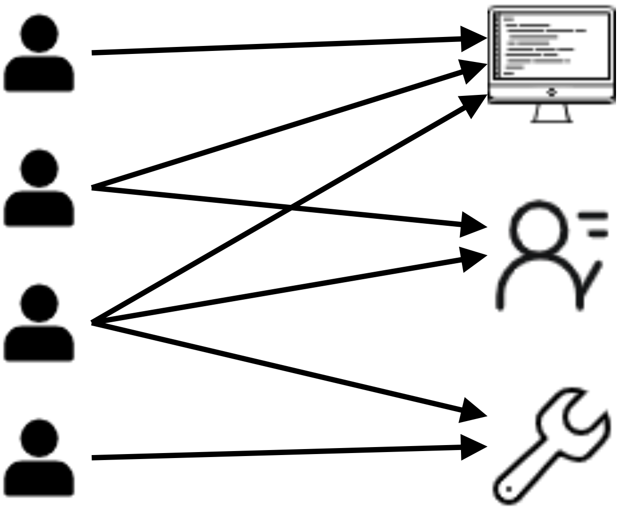
# 第5章 匹配

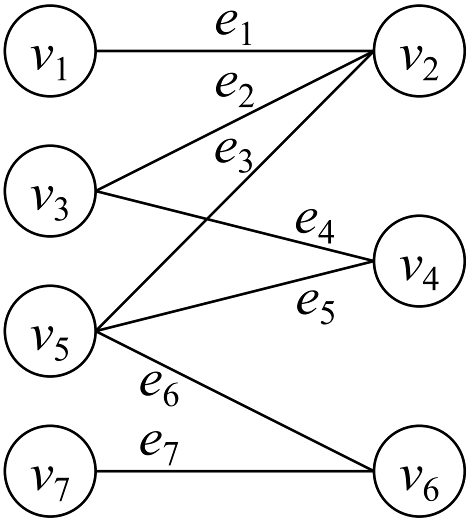
程龚

# 顶点的配对

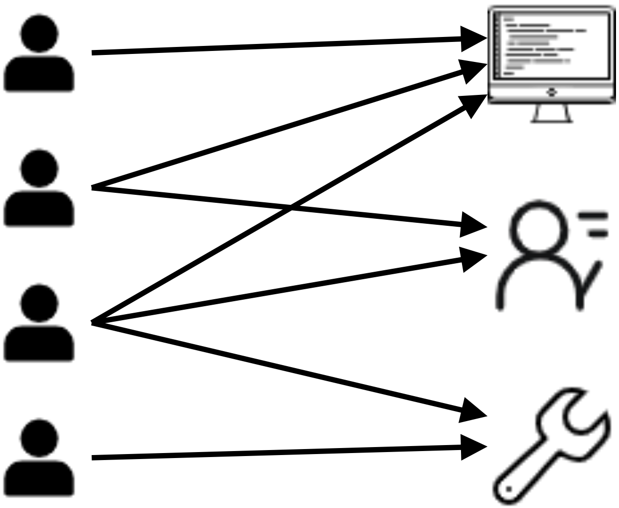


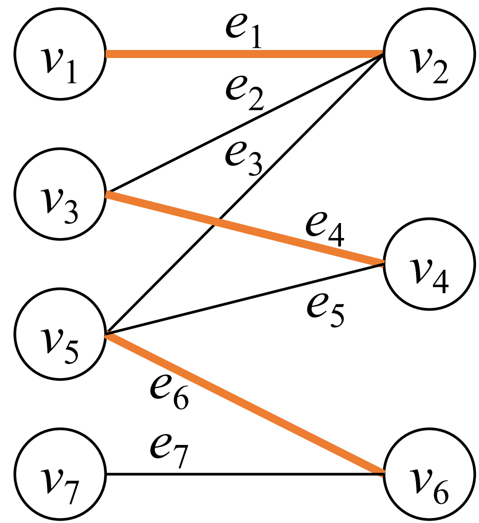
# 顶点的配对



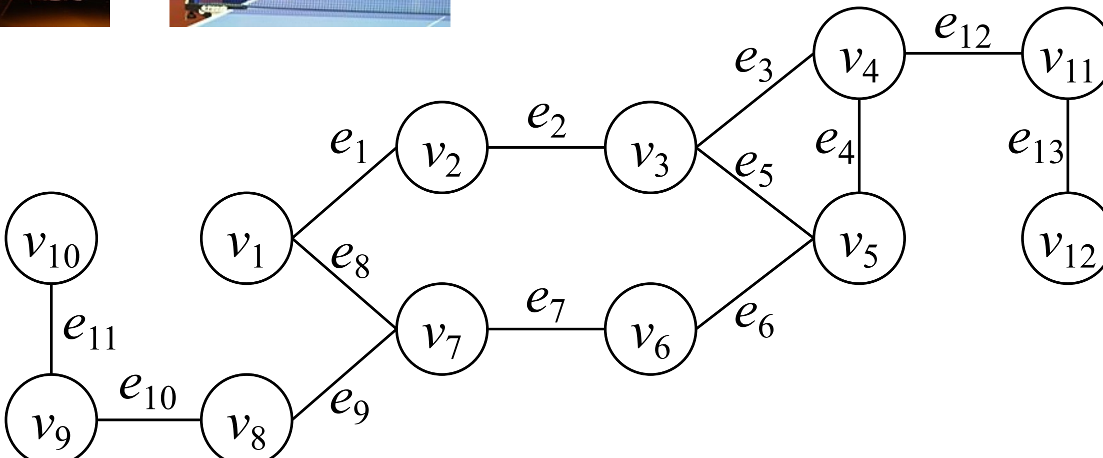


# 顶点的配对



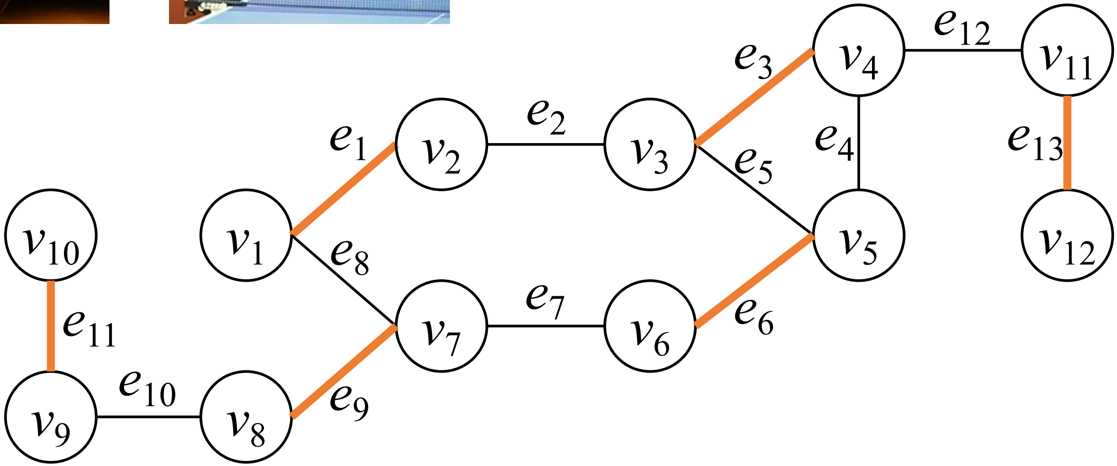


# 顶点的配对




https://zh.wikipedia.org/wiki/%E9%A9%AC%E9%BE%99_(%E4%B9%92%E4%B9%93%E7%90%83%E8%BF%90%E5%8A%A8%E5%91%98)
https://zh.wikipedia.org/wiki/%E8%AE%B8%E6%98%95

# 顶点的配对




https://zh.wikipedia.org/wiki/%E9%A9%AC%E9%BE%99_(%E4%B9%92%E4%B9%93%E7%90%83%E8%BF%90%E5%8A%A8%E5%91%98)
https://zh.wikipedia.org/wiki/%E8%AE%B8%E6%98%95

# 本次课的主要内容

5.1 匹配和最大匹配

5.2 完美匹配

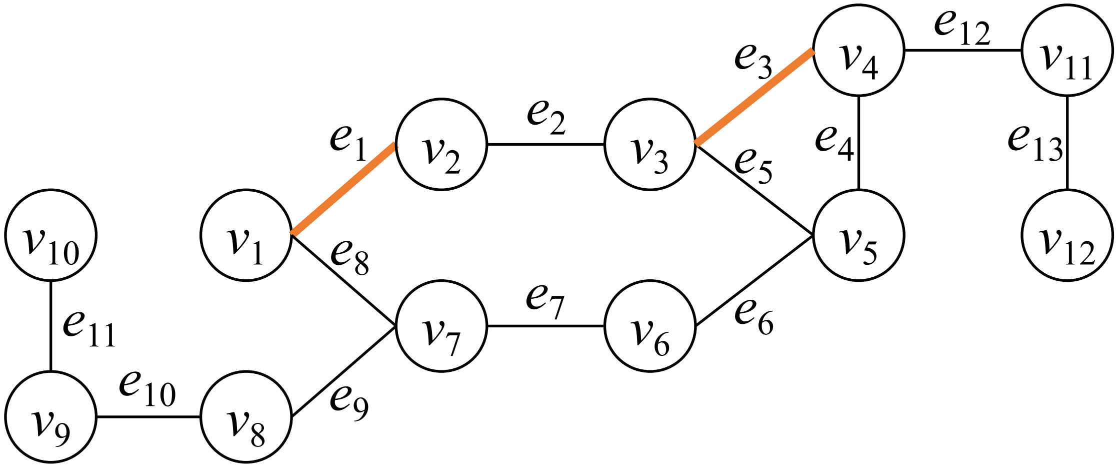
# 本次课的主要内容

5.1 匹配和最大匹配

5.2 完美匹配

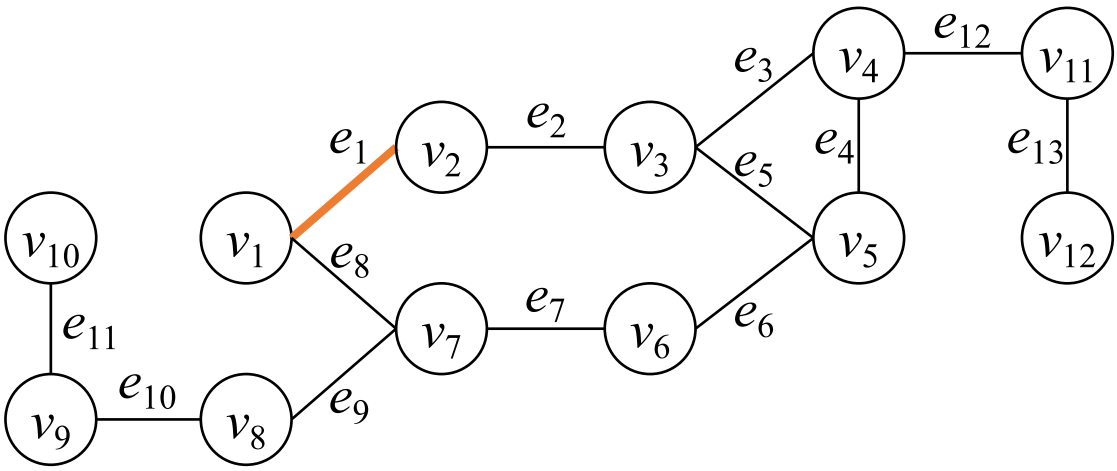
# 匹配和最大匹配

- **匹配**：两两不相邻的边的子集



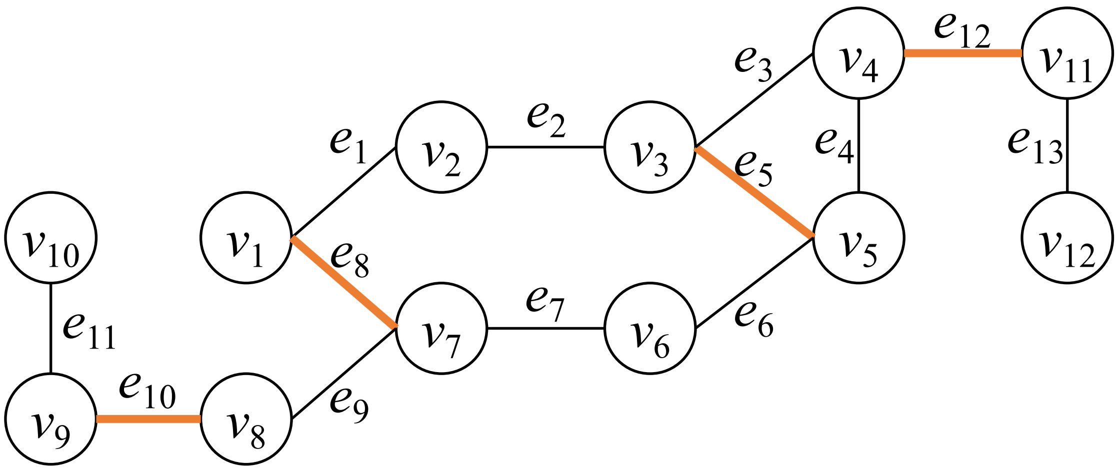
# 匹配和最大匹配

- 匹配：两两不相邻的边的子集
- **饱和**（已匹配）：匹配中边的端点被匹配饱和



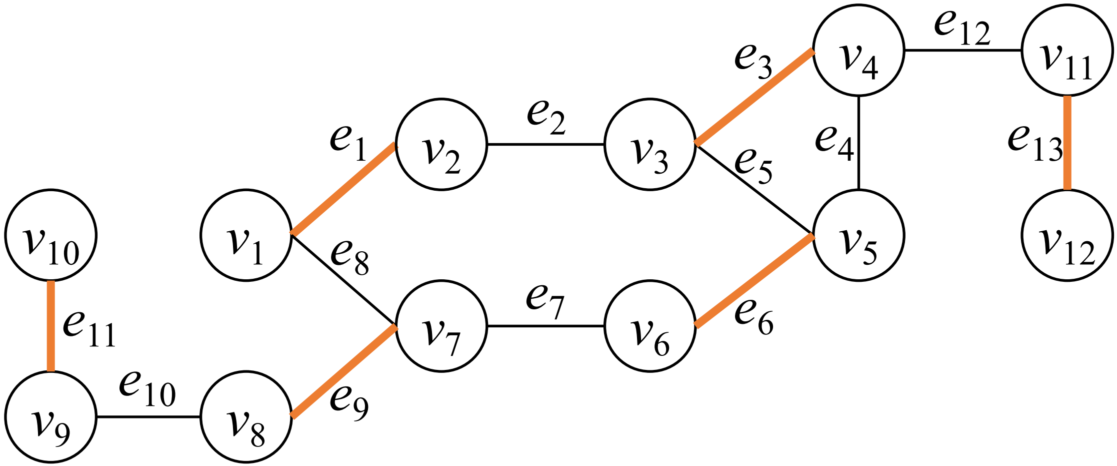
# 匹配和最大匹配

- 匹配：两两不相邻的边的子集
- 饱和（已匹配）：匹配中边的端点被匹配饱和
- **极大匹配**：匹配，且不是任何匹配的真子集



# 匹配和最大匹配

- 匹配：两两不相邻的边的子集
- 饱和（已匹配）：匹配中边的端点被匹配饱和
- 极大匹配：匹配，且不是任何匹配的真子集
- **最大匹配**：边的数量最多的匹配



# 匹配和最大匹配

- 每个图都有匹配吗？

# 匹配和最大匹配

- 每个图都有匹配吗？
- 阶为$n$的图的最大匹配至多有多少条边？

# 匹配和最大匹配

- 每个图都有匹配吗？
- 阶为$n$的图的最大匹配至多有多少条边？
- 完全图$K_n$的最大匹配有多少条边？

# 匹配和最大匹配

- 每个图都有匹配吗？
- 阶为$n$的图的最大匹配至多有多少条边？
- 完全图$K_n$的最大匹配有多少条边？
- 完全二分图$K_{m,n}$的最大匹配有多少条边？

# 匹配和最大匹配

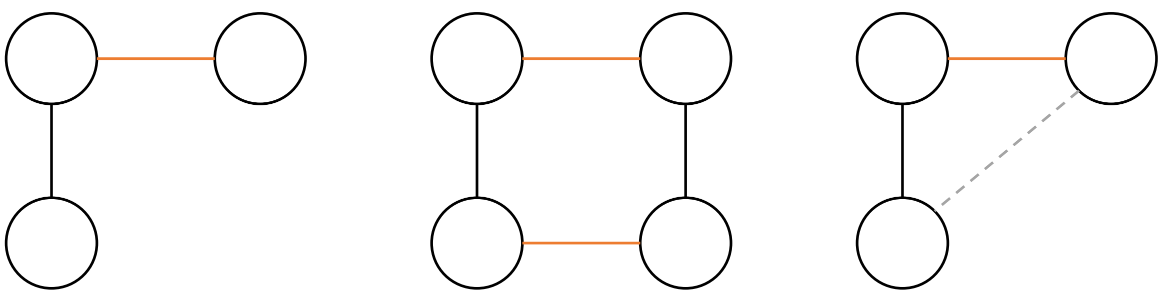
- 图的两个匹配的
  - 并集的边导出子图的每个连通分支的结构有什么特征？

**随堂小测**

# 匹配和最大匹配

- 图的两个匹配的
  - 并集的边导出子图的每个连通分支的结构有什么特征？

**随堂小测**

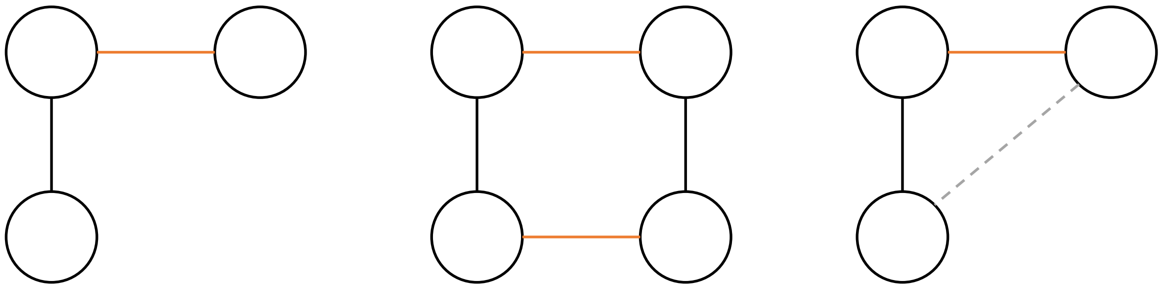


# 匹配和最大匹配

- 图的两个匹配的
  - 并集的边导出子图的每个连通分支的结构有什么特征？

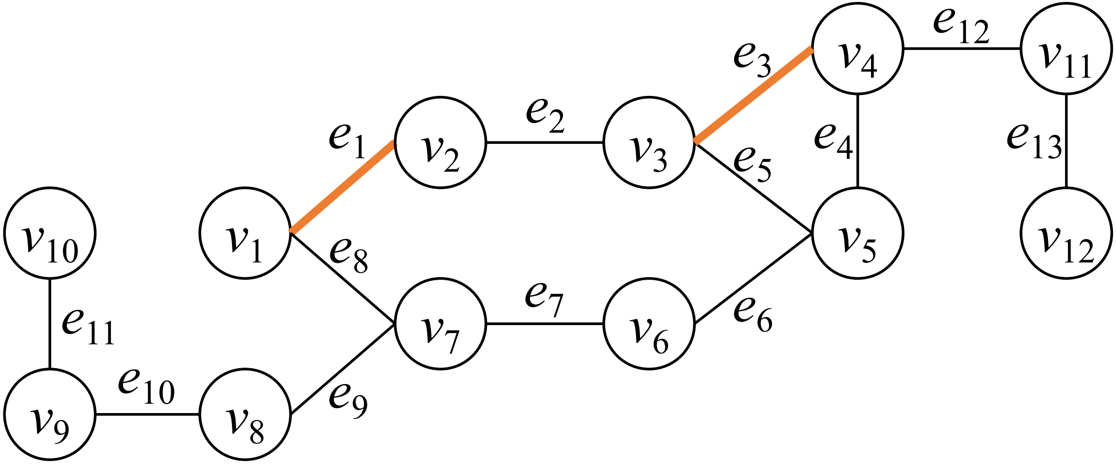
**随堂小测**

  - 对称差的边导出子图的每个连通分支的结构有什么特征？



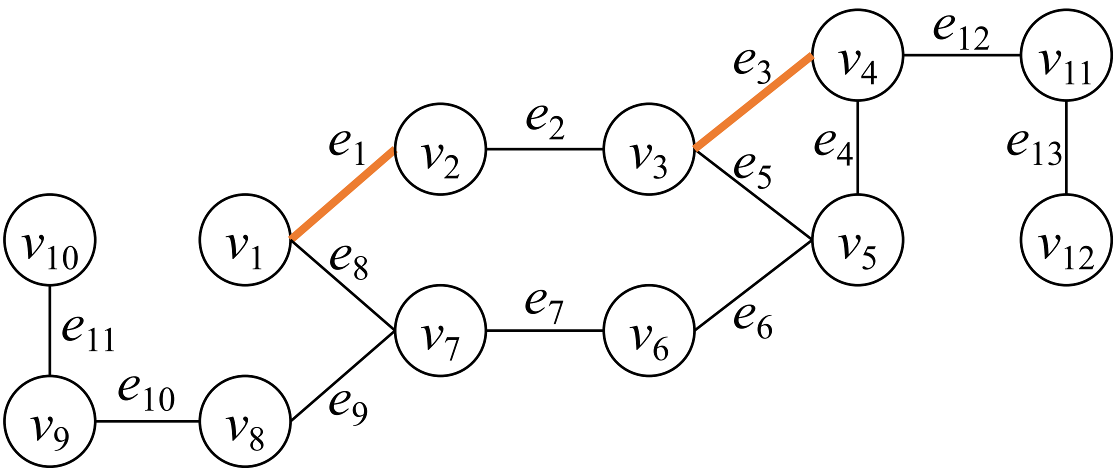
# 匹配和最大匹配

- 对于图$G = <V, E>$和匹配$M \subseteq E$
  - ***M*-交错路**：交错经过$M$和$E \setminus M$中的边



# 匹配和最大匹配

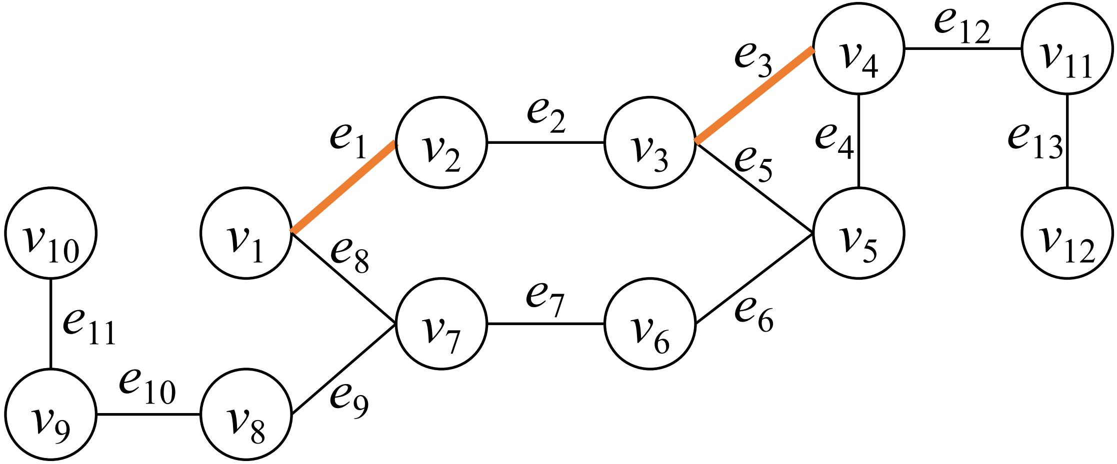
- 对于图$G = <V, E>$和匹配$M \subseteq E$
  - $M$-交错路：交错经过$M$和$E \setminus M$中的边
  - **$M$-增广路**：$M$-交错路，非平凡路，且起点和终点未被$M$饱和



# 匹配和最大匹配

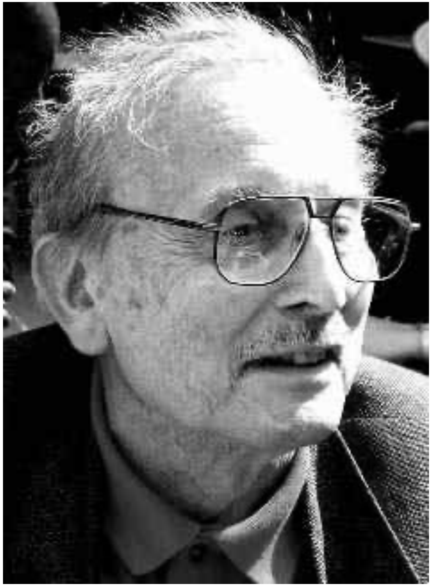
- 对于图$G = <V, E>$和匹配$M \subseteq E$
  - $M$-交错路：交错经过$M$和$E \setminus M$中的边
  - $M$-增广路：$M$-交错路，非平凡路，且起点和终点未被$M$饱和

- 每个匹配都有交错路和增广路吗？若有，唯一吗？



# 匹配和最大匹配

- Claude Berge， 1926-2002， 出生于法国



https://mathshistory.st-andrews.ac.uk/Biographies/Berge/Berge.jpeg

# 匹配和最大匹配

- 贝尔热定理：对于图$G = <V, E>$和匹配$M \subseteq E$，$M$是最大匹配当且仅当$G$不含$M$-增广路。

# 匹配和最大匹配

- 贝尔热定理：对于图$G = <V, E>$和匹配$M \subseteq E$，$M$是最大匹配当且仅当$G$不含$M$-增广路。

- 必要性：你能自己证明吗？

# 匹配和最大匹配

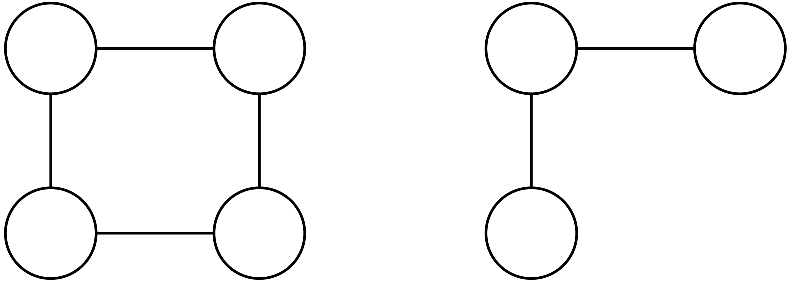
- 贝尔热定理：对于图$G = <V, E>$和匹配$M \subseteq E$，$M$是最大匹配当且仅当$G$不含$M$-增广路。

- 充分性：采用反证法，假设最大匹配$|M'| > |M|$

# 匹配和最大匹配

- 贝尔热定理：对于图$G = <V, E>$和匹配$M \subseteq E$，$M$是最大匹配当且仅当$G$不含$M$-增广路。

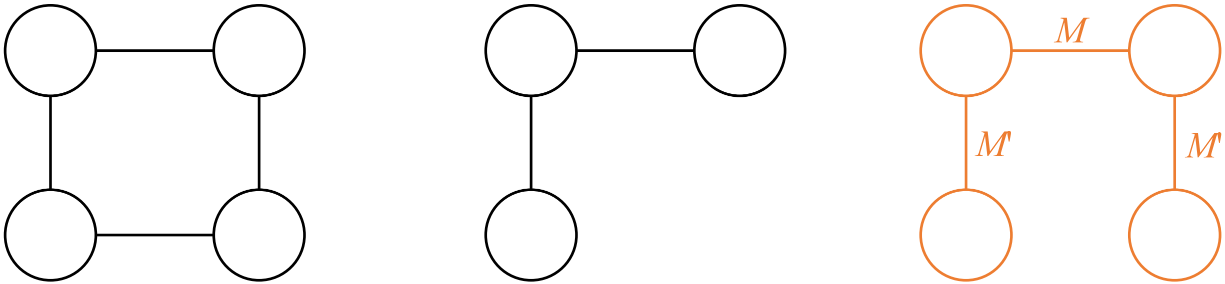
- 充分性：采用反证法，假设最大匹配$|M'| > |M|$
  - $M'$和$M$的对称差的边导出子图的连通分支有什么特征？



# 匹配和最大匹配

- 贝尔热定理：对于图$G = <V, E>$和匹配$M \subseteq E$，$M$是最大匹配当且仅当$G$不含$M$-增广路。

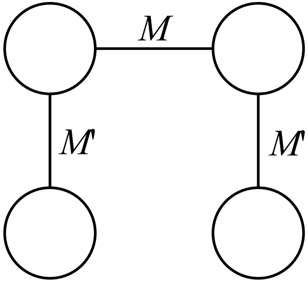
- 充分性：采用反证法，假设最大匹配$|M'| > |M|$
  - $M'$和$M$的对称差的边导出子图的连通分支有什么特征？
  - 有一条路$p$的长度为奇数且经过的$M'$中的边多于经过的$M$中的边



# 匹配和最大匹配

- 贝尔热定理：对于图$G = <V, E>$和匹配$M \subseteq E$，$M$是最大匹配当且仅当$G$不含$M$-增广路。

- 充分性：采用反证法，假设最大匹配$|M'| > |M|$
  - $M'$和$M$的对称差的边导出子图的连通分支有什么特征？
  - 有一条路$p$的长度为奇数且经过的$M'$中的边多于经过的$M$中的边
  - 且$p$是$M$-增广路（为什么？），矛盾



# 匹配和最大匹配

- 如何找出图中的最大匹配？

# 匹配和最大匹配

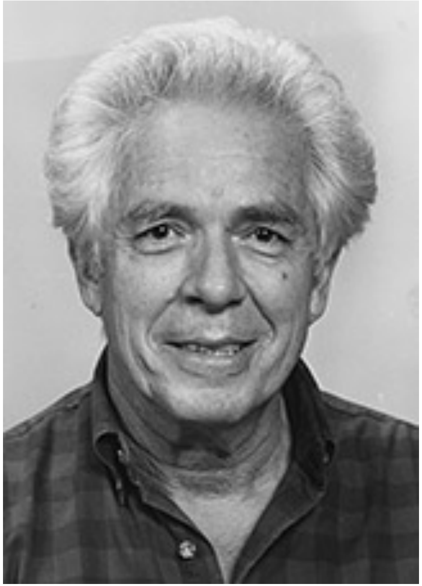
- 二分图
  - 匈牙利算法
  - 霍普克罗夫特-卡普算法
- 非二分图
  - 花算法

# 匹配和最大匹配

- 二分图
  - 匈牙利算法：找增广路
  - 霍普克罗夫特-卡普算法
- 非二分图
  - 花算法

# 匹配和最大匹配

- Harold Kuhn， 1925-2014， 出生于美国



基于两位匈牙利数学家的早期工作

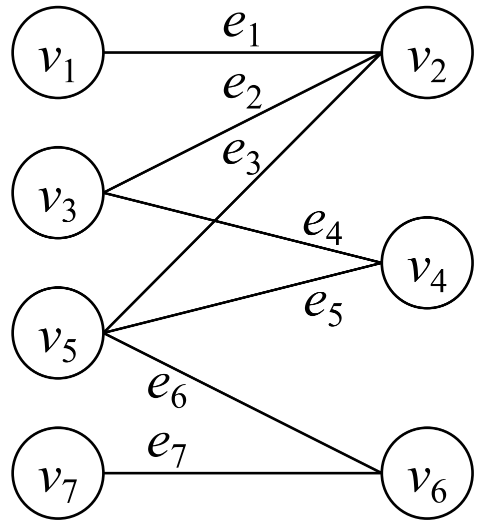
https://en.wikipedia.org/wiki/Harold_W._Kuhn

# 匹配和最大匹配

- 从初值为空集的匹配$M$开始

**算法 8:** 匈牙利算法

```
输入: 二分图 G = ⟨X ∪ Y, E⟩
初值: M 初值为 ∅
do
    foreach u ∈ (X ∪ Y) do
        u.visited ← false;
    foreach r ∈ X do
        if r.visited = false 且 r 未被 M 饱和 then
            p ← DFSAP(G, r, M);
            if p ≠ null then
                M ← Edges(p) △ M;
                中止 ForEach 循环;
while p ≠ null;
输出 (M);
```

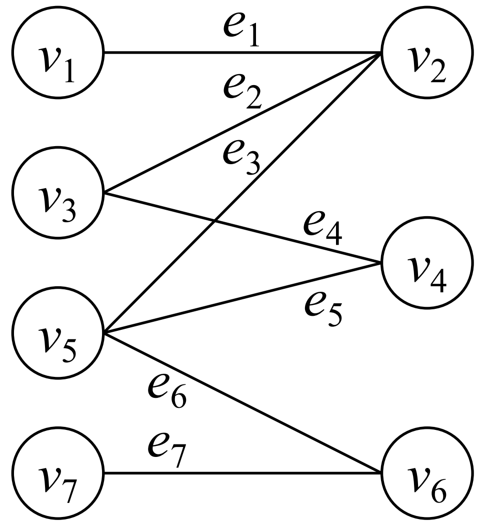


# 匹配和最大匹配

- 每轮do-while循环尝试找一条$M$-增广路$p$
  - 调用DFSAP算法
  - 从$X$中每个在本轮do-while循环中未被DFSAP算法访问过且未被$M$饱和的顶点$r$出发

**算法 8:** 匈牙利算法

```
输入: 二分图 G = ⟨X ∪ Y, E⟩
初值: M 初值为 ∅
do
    foreach u ∈ (X ∪ Y) do
        u.visited ← false;
    foreach r ∈ X do
        if r.visited = false 且 r 未被 M 饱和 then
            p ← DFSAP(G, r, M);
            if p ≠ null then
                M ← Edges(p) △ M;
                中止 ForEach 循环;
while p ≠ null;
输出 (M);
```



# 匹配和最大匹配

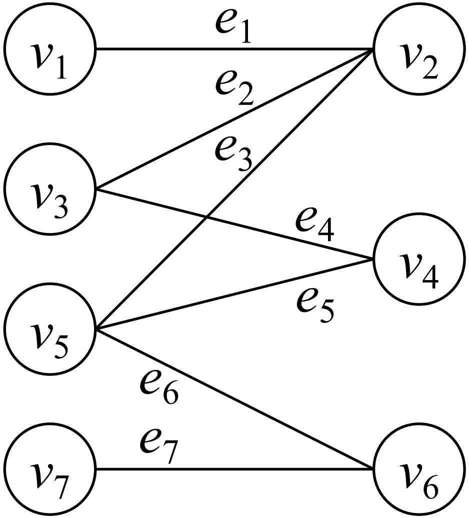
- 每轮do-while循环尝试找一条$M$-增广路$p$
  - 调用DFSAP算法
  - 从$X$中每个在本轮do-while循环中未被DFSAP算法访问过且未被$M$饱和的顶点$r$出发
    - 只从顶点子集$X$中的顶点出发运行DFSAP算法，会遗漏$M$-增广路吗？

**算法 8:** 匈牙利算法

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

**初值:** $M$ 初值为 $\emptyset$

```
do
    foreach u ∈ (X ∪ Y) do
        u.visited ← false;
    foreach r ∈ X do
        if r.visited = false 且 r 未被 M 饱和 then
            p ← DFSAP(G, r, M);
            if p ≠ null then
                M ← Edges(p) △ M;
                中止 ForEach 循环;
while p ≠ null;
输出 (M);
```



# 匹配和最大匹配

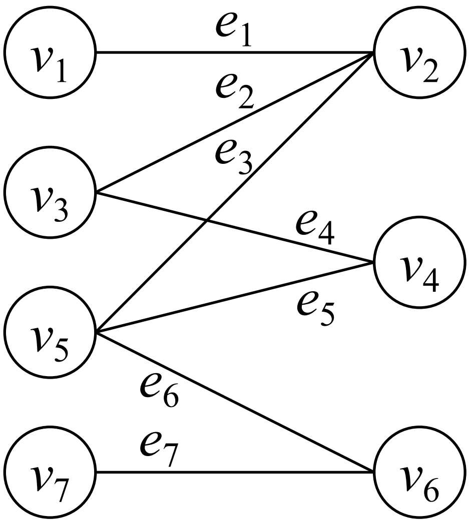
- 若能找到，则计算$p$经过的边集和$M$的对称差，得到一个包含边的数量更多的匹配

**算法 8:** 匈牙利算法

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

**初值:** $M$ 初值为 $\emptyset$

```
do
    foreach u ∈ (X ∪ Y) do
        u.visited ← false;
    foreach r ∈ X do
        if r.visited = false 且 r 未被 M 饱和 then
            p ← DFSAP(G, r, M);
            if p ≠ null then
                M ← Edges(p) △ M;
                中止 ForEach 循环;
while p ≠ null;
输出 (M);
```



# 匹配和最大匹配

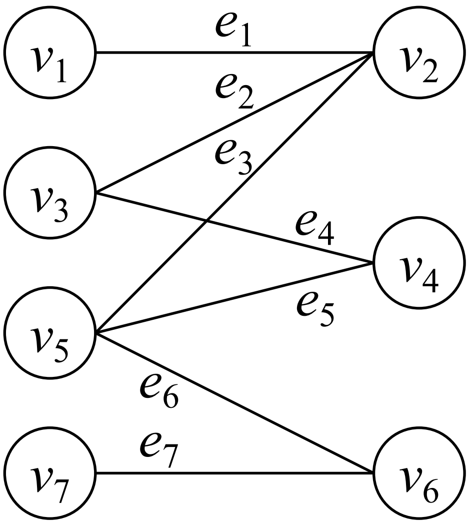
- 否则，不再进行下一轮尝试，输出最大匹配$M$

**算法 8:** 匈牙利算法

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

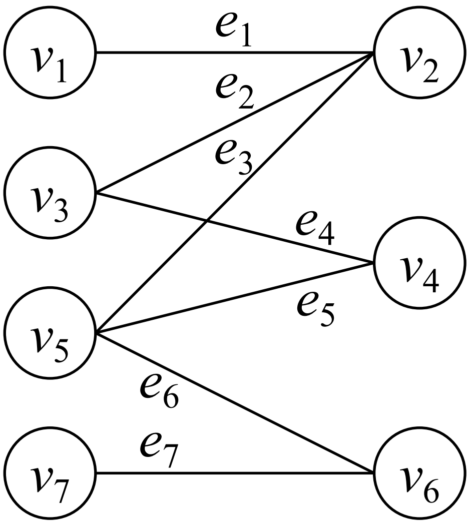
**初值:** $M$ 初值为 $\emptyset$

```
do
    foreach u ∈ (X ∪ Y) do
        u.visited ← false;
    foreach r ∈ X do
        if r.visited = false 且 r 未被 M 饱和 then
            p ← DFSAP(G, r, M);
            if p ≠ null then
                M ← Edges(p) △ M;
                中止 ForEach 循环;
while p ≠ null;
输出 (M);
```



# 匹配和最大匹配

$M = \{\}$




**算法 8:** 匈牙利算法

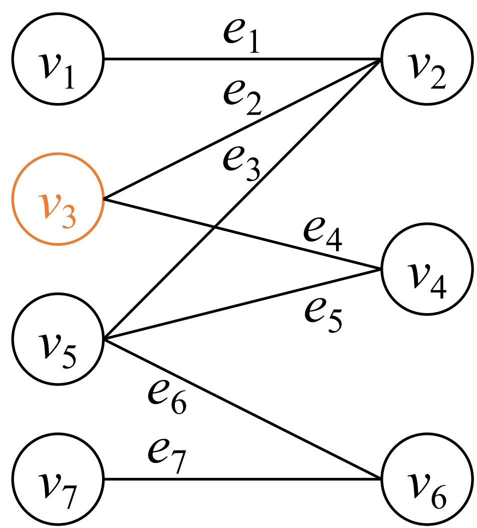
**输入:** 二分图 $G = \langle X \cup Y, E \rangle$

**初值:** $M$ 初值为 $\emptyset$

**do**
    **foreach** $u \in (X \cup Y)$ **do**
        $u$.visited $\leftarrow$ false;
    **foreach** $r \in X$ **do**
        **if** $r.visited = false$ 且 $r$ 未被 $M$ 饱和 **then**
            $p \leftarrow$ DFSAP($G$, $r$, $M$);
            **if** $p \neq null$ **then**
                $M \leftarrow$ Edges($p$) $\triangle$ $M$;
                中止 ForEach 循环;
**while** $p \neq null$;
输出 （$M$）;

# 匹配和最大匹配

$M = \{\}$




**算法 8:** 匈牙利算法

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

**初值:** $M$ 初值为 $\emptyset$

**do**
    **foreach** $u \in (X \cup Y)$ **do**
        $u$.visited $\leftarrow$ false;
    **foreach** $r \in X$ **do**
        **if** $r.visited = false$ 且 $r$ 未被 $M$ 饱和 **then**
            $p \leftarrow$ `DFSAP`$(G, r, M)$;
            **if** $p \neq null$ **then**
                $M \leftarrow$ `Edges`$(p) \triangle M$;
                中止 ForEach 循环;
**while** $p \neq null$;
输出 $(M)$;

# 匹配和最大匹配

---

**算法 8:** 匈牙利算法

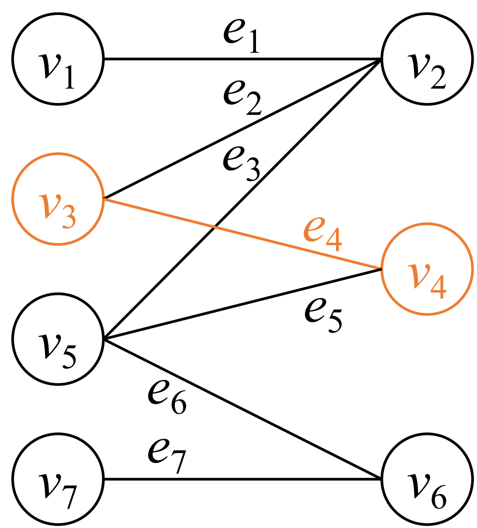
---

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

**初值:** $M$ 初值为 $\emptyset$

```
do
    foreach u ∈ (X ∪ Y) do
        u.visited ← false;
    foreach r ∈ X do
        if r.visited = false 且 r 未被 M 饱和 then
            p ← DFSAP(G, r, M);
            if p ≠ null then
                M ← Edges(p) △ M;
                中止 ForEach 循环;
while p ≠ null;
输出 (M);
```

$M = \{\}$



# 匹配和最大匹配

**算法 8:** 匈牙利算法

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

**初值:** $M$ 初值为 $\emptyset$

**do**

  **foreach** $u \in (X \cup Y)$ **do**

    $u$.visited $\leftarrow$ false;

  **foreach** $r \in X$ **do**

    **if** $r.visited = false$ 且 $r$ 未被 $M$ 饱和 **then**

      $p \leftarrow \texttt{DFSAP}(G, r, M)$;

      **if** $p \neq null$ **then**

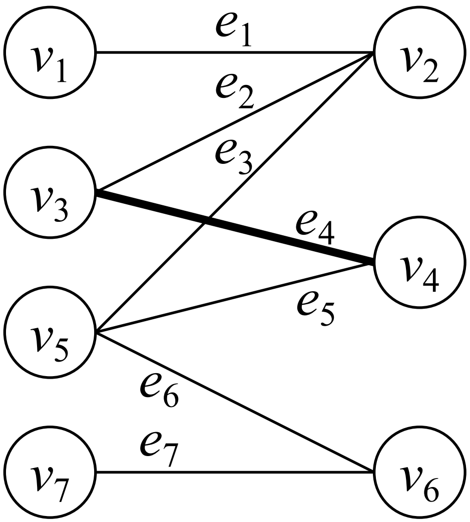
        $M \leftarrow \texttt{Edges}(p) \triangle M$;

        中止 ForEach 循环;

**while** $p \neq null$;

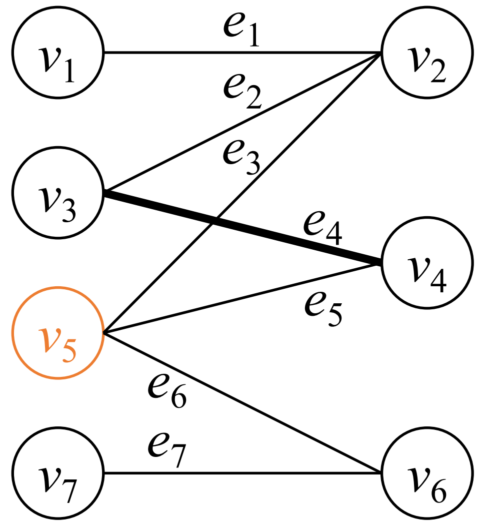
输出 $(M)$;

$M = \{e_4\}$



# 匹配和最大匹配

$M = \{e_4\}$




**算法 8:** 匈牙利算法

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

**初值:** $M$ 初值为 $\emptyset$

**do**
    **foreach** $u \in (X \cup Y)$ **do**
        $u$.visited $\leftarrow$ false;
    **foreach** $r \in X$ **do**
        **if** $r.visited = false$ 且 $r$ 未被 $M$ 饱和 **then**
            $p \leftarrow$ DFSAP$(G, r, M)$;
            **if** $p \neq null$ **then**
                $M \leftarrow$ Edges$(p) \triangle M$;
                中止 ForEach 循环;
**while** $p \neq null$;
输出 $(M)$;

# 匹配和最大匹配

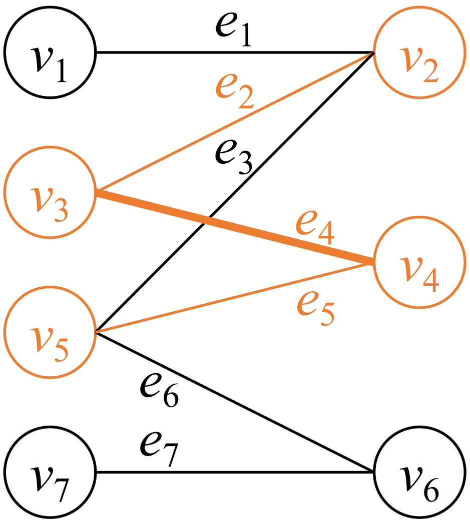
**算法 8:** 匈牙利算法

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

**初值:** $M$ 初值为 $\emptyset$

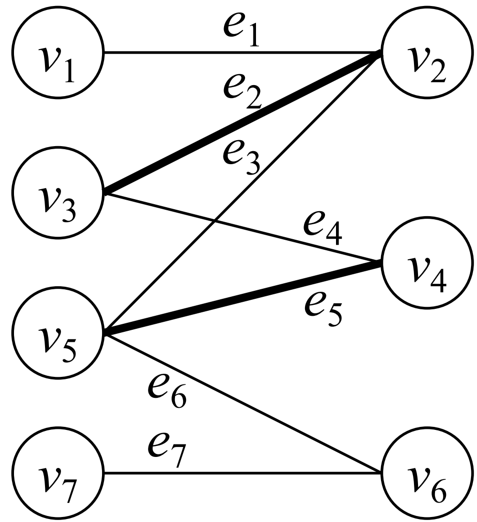
```
do
    foreach u ∈ (X ∪ Y) do
        u.visited ← false;
    foreach r ∈ X do
        if r.visited = false 且 r 未被 M 饱和 then
            p ← DFSAP(G, r, M);
            if p ≠ null then
                M ← Edges(p) △ M;
                中止 ForEach 循环;
while p ≠ null;
输出 (M);
```

$M = \{e_4\}$



# 匹配和最大匹配

$M = \{e_5, e_2\}$




**算法 8:** 匈牙利算法

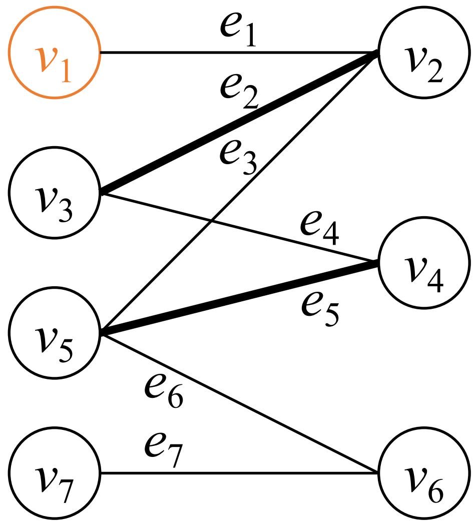
**输入:** 二分图 $G = \langle X \cup Y, E \rangle$

**初值:** $M$ 初值为 $\emptyset$

**do**
    **foreach** $u \in (X \cup Y)$ **do**
        $u$.visited $\leftarrow$ false;
    **foreach** $r \in X$ **do**
        **if** $r.visited = false$ 且 $r$ 未被 $M$ 饱和 **then**
            $p \leftarrow$ `DFSAP`$(G, r, M)$;
            **if** $p \neq null$ **then**
                $M \leftarrow$ `Edges`$(p) \triangle M$;
                中止 ForEach 循环;
**while** $p \neq null$;
输出 $(M)$;

# 匹配和最大匹配

$M = \{e_5, e_2\}$




---

**算法 8:** 匈牙利算法

---

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

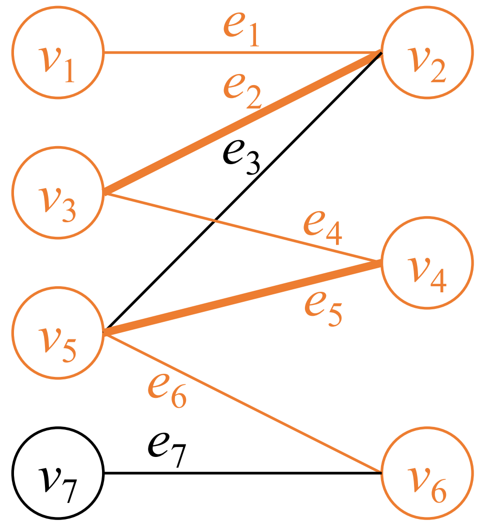
**初值:** $M$ 初值为 $\emptyset$

**do**
    **foreach** $u \in (X \cup Y)$ **do**
        $u$.visited $\leftarrow$ false;
    **foreach** $r \in X$ **do**
        **if** $r.visited = false$ 且 $r$ 未被 $M$ 饱和 **then**
            $p \leftarrow$ `DFSAP`$(G, r, M)$;
            **if** $p \neq null$ **then**
                $M \leftarrow$ `Edges`$(p) \triangle M$;
                中止 ForEach 循环;
**while** $p \neq null$;
输出 $(M)$;

---

# 匹配和最大匹配

$M = \{e_5, e_2\}$




**算法 8:** 匈牙利算法

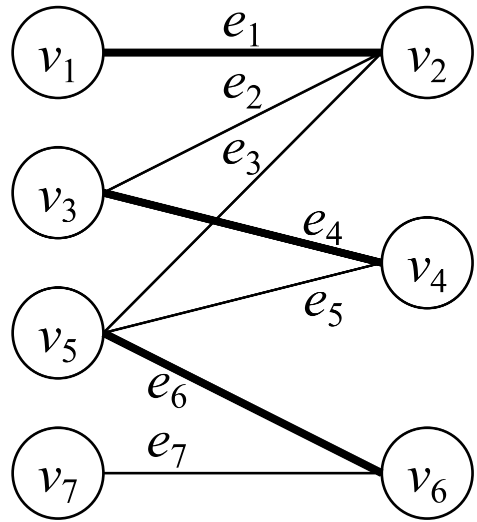
**输入:** 二分图 $G = \langle X \cup Y, E \rangle$

**初值:** $M$ 初值为 $\emptyset$

```
do
    foreach u ∈ (X ∪ Y) do
        u.visited ← false;
    foreach r ∈ X do
        if r.visited = false 且 r 未被 M 饱和 then
            p ← DFSAP(G, r, M);
            if p ≠ null then
                M ← Edges(p) △ M;
                中止 ForEach 循环;
while p ≠ null;
输出 (M);
```

# 匹配和最大匹配

$M = \{e_1, e_4, e_6\}$




**算法 8:** 匈牙利算法

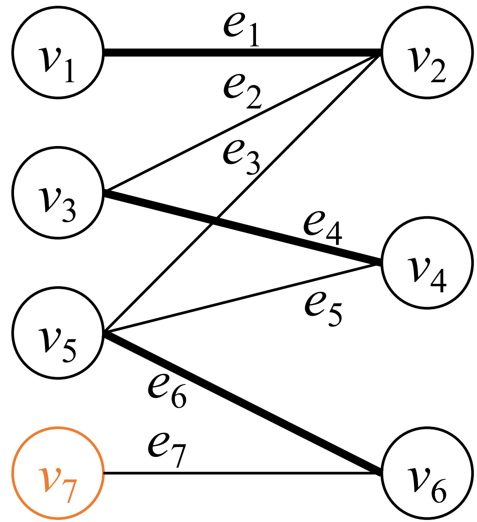
**输入:** 二分图 $G = \langle X \cup Y, E \rangle$

**初值:** $M$ 初值为 $\emptyset$

**do**
    **foreach** $u \in (X \cup Y)$ **do**
        $u$.visited $\leftarrow$ false;
    **foreach** $r \in X$ **do**
        **if** $r.visited = false$ 且 $r$ 未被 $M$ 饱和 **then**
            $p \leftarrow$ DFSAP($G$, $r$, $M$);
            **if** $p \neq null$ **then**
                $M \leftarrow$ Edges($p$) $\triangle$ $M$;
                中止 ForEach 循环;
**while** $p \neq null$;
输出 ($M$);

# 匹配和最大匹配

$M = \{e_1, e_4, e_6\}$




**算法 8:** 匈牙利算法

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

**初值:** $M$ 初值为 $\emptyset$

**do**
  **foreach** $u \in (X \cup Y)$ **do**
    $u$.visited $\leftarrow$ false;
  **foreach** $r \in X$ **do**
    **if** $r.visited = false$ 且 $r$ 未被 $M$ 饱和 **then**
      $p \leftarrow$ DFSAP$(G, r, M)$;
      **if** $p \neq null$ **then**
        $M \leftarrow$ Edges$(p) \triangle M$;
        中止 ForEach 循环;
**while** $p \neq null$;
输出 $(M)$;

# 匹配和最大匹配

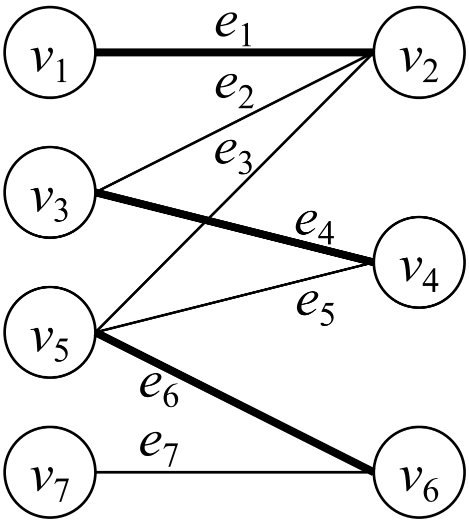
**算法 8:** 匈牙利算法

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

**初值:** $M$ 初值为 $\emptyset$

```
do
    foreach u ∈ (X ∪ Y) do
        u.visited ← false;
    foreach r ∈ X do
        if r.visited = false 且 r 未被 M 饱和 then
            p ← DFSAP(G, r, M);
            if p ≠ null then
                M ← Edges(p) △ M;
                中止 ForEach 循环;
while p ≠ null;
输出 (M);
```

$M = \{e_1, e_4, e_6\}$



# 匹配和最大匹配

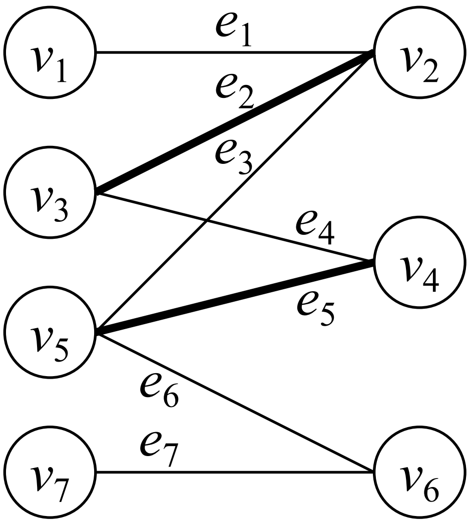
- 扩展DFS算法

**算法 9:** DFSAP

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

$u$.visited $\leftarrow$ true;
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**
  **return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
  **foreach** $(u, v)\in E$ **do**
    **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
      $p_v \leftarrow$ DFSAP$(G, v, M)$;
      **if** $p_v \neq null$ **then**
        **return** $p_v$;
  **return** *null*;

$M=\{e_5, e_2\}$



# 匹配和最大匹配

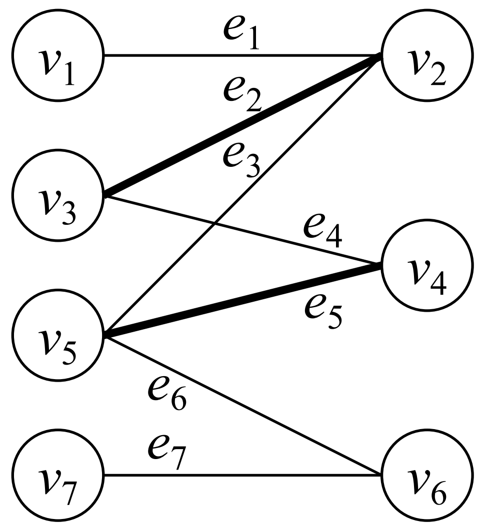
- 扩展DFS算法：限制了可访问的邻点，使DFS树中以根顶点（即匈牙利算法中的顶点$r$）为起点的每条路都是$M$-交错路

**算法 9:** DFSAP

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

$u$.visited $\leftarrow$ true;
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**
  **return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
  **foreach** $(u, v) \in E$ **do**
    **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
      $p_v \leftarrow$ DFSAP($G$, $v$, $M$);
      **if** $p_v \neq null$ **then**
        **return** $p_v$;
  **return** *null*;

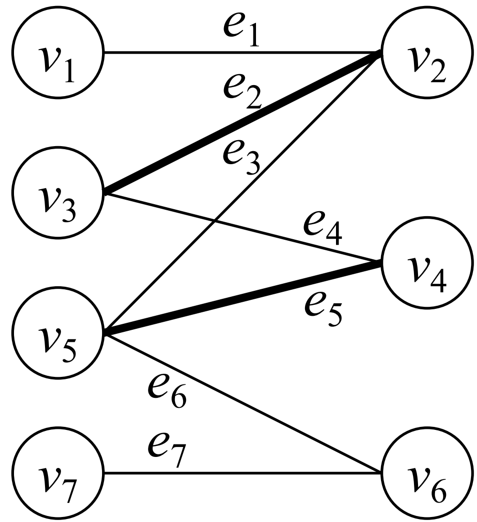
$M=\{e_5, e_2\}$



# 匹配和最大匹配

- 扩展DFS算法：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
  - 如何高效地判定DFS树中从根顶点到$v$的路是$M$-交错路？

$M = \{e_5, e_2\}$



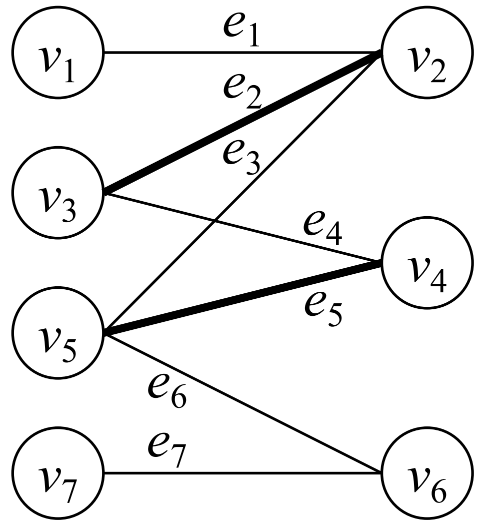

**算法 9:** DFSAP

```
输入: 图 G = ⟨X ∪ Y, E⟩, 顶点 u, 匹配 M
u.visited ← true;
if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
    return DFS 树中从根顶点到 u 的路;
else
    foreach (u, v) ∈ E do
        if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M-交错路 then
            p_v ← DFSAP(G, v, M);
            if p_v ≠ null then
                return p_v;
    return null;
```

# 匹配和最大匹配

- 扩展DFS算法：判定并返回增广路

$M = \{e_5, e_2\}$




**算法 9:** DFSAP

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

$u$.visited ← true;
**if** $u$ 未被 $M$ 饱和且 $u \neq DFS$ 树的根顶点 **then**
    **return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
    **foreach** $(u, v) \in E$ **do**
        **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
            $p_v$ ← DFSAP($G$, $v$, $M$);
            **if** $p_v \neq null$ **then**
                **return** $p_v$;
    **return** *null*;

# 匹配和最大匹配

- 扩展DFS算法：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

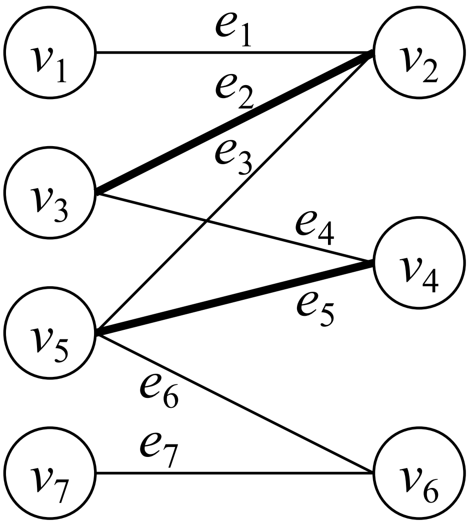
**算法 9:** DFSAP

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

$u$.visited ← true;
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**
　　**return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
　　**foreach** $(u, v) \in E$ **do**
　　　　**if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
　　　　　　$p_v \leftarrow$ DFSAP($G$, $v$, $M$);
　　　　　　**if** $p_v \neq null$ **then**
　　　　　　　　**return** $p_v$;
　　**return** $null$;

$u$是$M$-增广路的终点

$M = \{e_5, e_2\}$



# 匹配和最大匹配

- 扩展DFS算法：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$p_v$

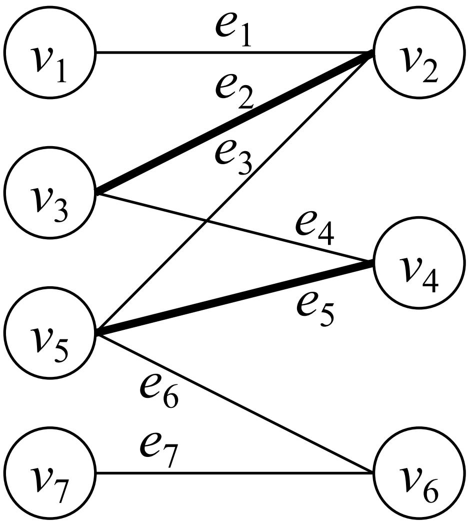
**算法 9:** DFSAP

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

$u$.visited ← true;
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**
  **return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
  **foreach** $(u, v) \in E$ **do**
    **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
      $p_v \leftarrow$ DFSAP($G$, $v$, $M$);
      **if** $p_v \neq null$ **then**
        **return** $p_v$;
  **return** *null*;

$u$是$M$-增广路的起点或内顶点

$M=\{e_5, e_2\}$



# 匹配和最大匹配

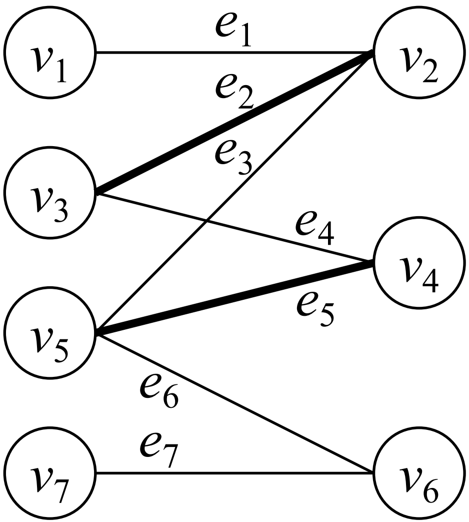
## 扩展DFS算法：判定并返回增广路

- 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

$$M = \{e_5, e_2\}$$

**算法 9:** DFSAP

```
  输入: 图 G = ⟨X ∪ Y, E⟩，顶点 u，匹配 M
  u.visited ← true;
  if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
  │ return DFS 树中从根顶点到 u 的路;
  else
  │ foreach (u, v) ∈ E do
  │ │ if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M-交错路 then
  │ │ │ p_v ← DFSAP(G, v, M);
  │ │ │ if p_v ≠ null then
  │ │ │ │ return p_v;
  │ return null;
```



# 匹配和最大匹配

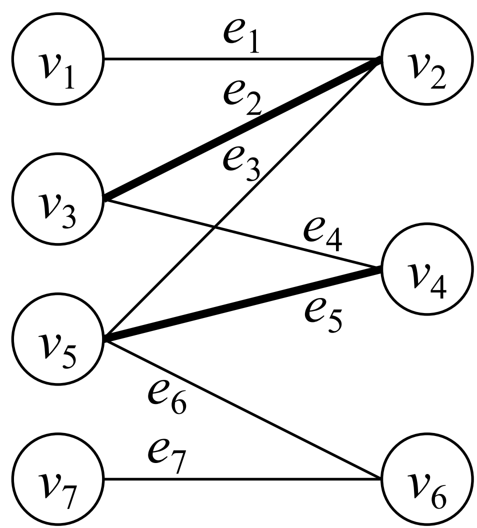
- 扩展DFS算法：判定并返回增广路
  - 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
  - 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
  - 否则，未找到$M$-增广路，算法返回null

**算法 9:** DFSAP

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

$u$.visited $\leftarrow$ true;
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**
  **return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
  **foreach** $(u, v) \in E$ **do**
    **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
      $p_v \leftarrow$ DFSAP($G$, $v$, $M$);
      **if** $p_v \neq null$ **then**
        **return** $p_v$;
  **return** $null$;

$M = \{e_5, e_2\}$



# 匹配和最大匹配

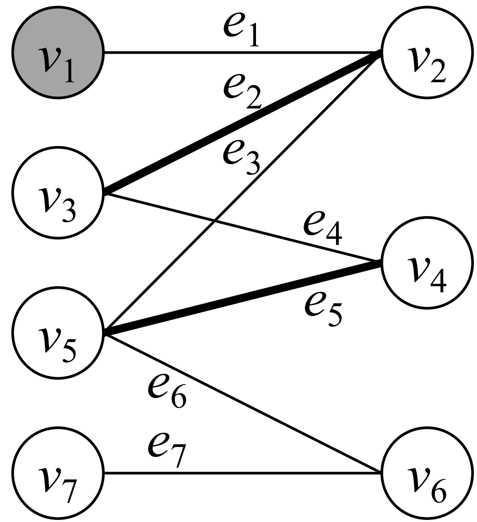
- 扩展DFS算法：判定并返回增广路
  - 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
  - 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
  - 否则，未找到$M$-增广路，算法返回null

**算法 9:** DFSAP

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

$u$.visited $\leftarrow$ true;
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**
  **return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
  **foreach** $(u, v) \in E$ **do**
    **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
      $p_v \leftarrow$ DFSAP($G$, $v$, $M$);
      **if** $p_v \neq null$ **then**
        **return** $p_v$;
  **return** $null$;

$$M = \{e_5, e_2\}$$



# 匹配和最大匹配

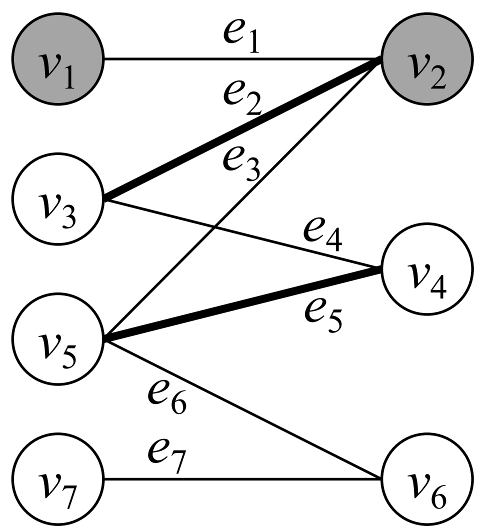
- 扩展DFS算法：判定并返回增广路
  - 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
  - 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
  - 否则，未找到$M$-增广路，算法返回null

$M = \{e_5, e_2\}$

**算法 9:** DFSAP

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

$u$.visited $\leftarrow$ true;
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**
  **return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
  **foreach** $(u, v) \in E$ **do**
    **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
      $p_v \leftarrow$ DFSAP($G$, $v$, $M$);
      **if** $p_v \neq null$ **then**
        **return** $p_v$;
  **return** $null$;



# 匹配和最大匹配

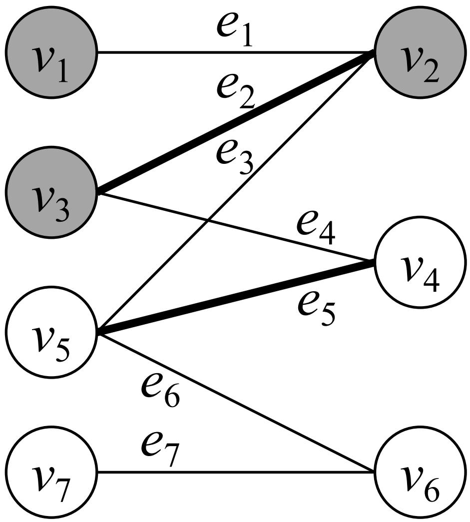
- 扩展DFS算法：判定并返回增广路
  - 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
  - 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
  - 否则，未找到$M$-增广路，算法返回null

$M = \{e_5, e_2\}$

**算法 9:** DFSAP

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

```
u.visited ← true;
if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
    return DFS 树中从根顶点到 u 的路;
else
    foreach (u, v) ∈ E do
        if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M-交错路 then
            p_v ← DFSAP(G, v, M);
            if p_v ≠ null then
                return p_v;
    return null;
```

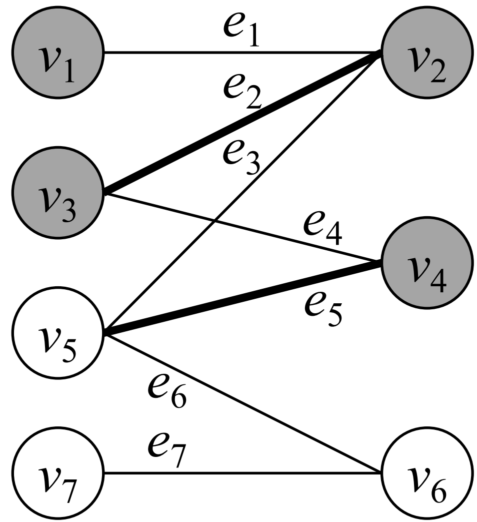


# 匹配和最大匹配

## 扩展DFS算法：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

$$M = \{e_5, e_2\}$$




**算法 9:** DFSAP

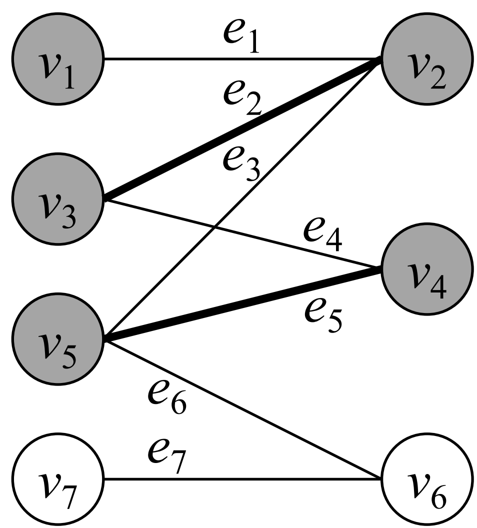
**输入:** 图 $G = \langle X \cup Y, E \rangle$，顶点 $u$，匹配 $M$

$u$.visited $\leftarrow$ true;
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**
　**return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
　**foreach** $(u, v) \in E$ **do**
　　**if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
　　　$p_v \leftarrow$ DFSAP($G$, $v$, $M$);
　　　**if** $p_v \neq null$ **then**
　　　　**return** $p_v$;
　**return** *null*;

# 匹配和最大匹配

- 扩展DFS算法：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

$M = \{e_5, e_2\}$




**算法 9:** DFSAP

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

$u$.visited ← true;<br>
**if** $u$ 未被 $M$ 饱和且 $u \neq DFS$ 树的根顶点 **then**<br>
    **return** *DFS* 树中从根顶点到 $u$ 的路;<br>
**else**<br>
    **foreach** $(u, v) \in E$ **do**<br>
        **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**<br>
            $p_v \leftarrow$ DFSAP($G$, $v$, $M$);<br>
            **if** $p_v \neq null$ **then**<br>
                **return** $p_v$;<br>
    **return** *null*;

# 匹配和最大匹配

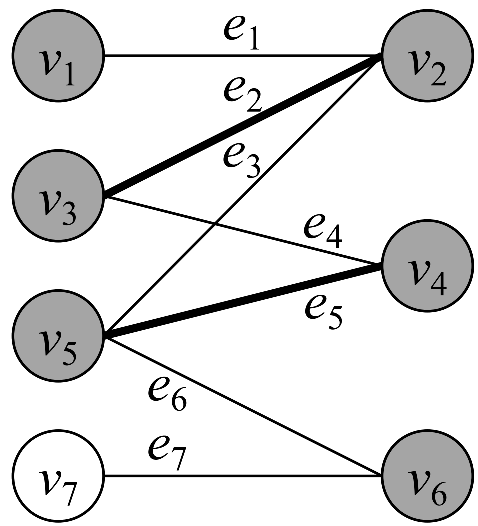
- 扩展DFS算法：判定并返回增广路
  - 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
  - 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
  - 否则，未找到$M$-增广路，算法返回null

$M = \{e_5, e_2\}$

**算法 9:** DFSAP

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

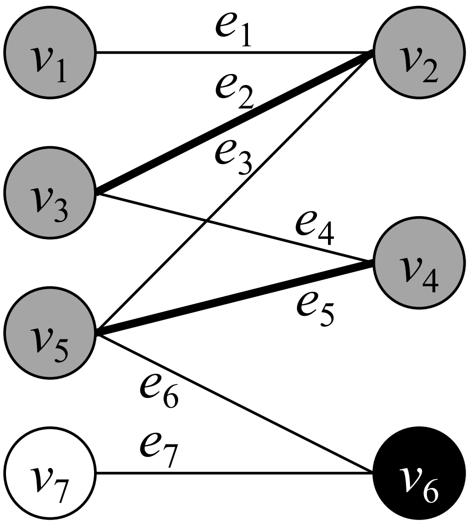
$u$.visited $\leftarrow$ true;<br>
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**<br>
    **return** *DFS* 树中从根顶点到 $u$ 的路;<br>
**else**<br>
    **foreach** $(u, v) \in E$ **do**<br>
        **if** $v$.*visited* = *false* 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**<br>
            $p_v \leftarrow$ DFSAP($G$, $v$, $M$);<br>
            **if** $p_v \neq$ *null* **then**<br>
                **return** $p_v$;<br>
    **return** *null*;



# 匹配和最大匹配

- 扩展DFS算法：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

$M = \{e_5, e_2\}$




**算法 9:** DFSAP

```
    输入: 图 G = ⟨X ∪ Y, E⟩，顶点 u，匹配 M
    u.visited ← true;
    if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
    |   return DFS 树中从根顶点到 u 的路;
    else
    |   foreach (u, v) ∈ E do
    |   |   if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M-交
    |   |     错路 then
    |   |   |   p_v ← DFSAP(G, v, M);
    |   |   |   if p_v ≠ null then
    |   |   |   |   return p_v;
    |   return null;
```

# 匹配和最大匹配

## ■ 扩展DFS算法：判定并返回增广路

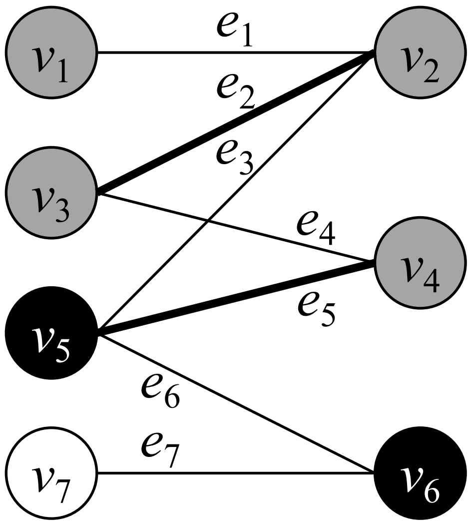
- 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
- 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
- 否则，未找到$M$-增广路，算法返回null

$M = \{e_5, e_2\}$

**算法 9:** DFSAP

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

$u$.visited ← true;
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**
　**return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
　**foreach** $(u, v) \in E$ **do**
　　**if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
　　　$p_v \leftarrow$ DFSAP($G$, $v$, $M$);
　　　**if** $p_v \neq null$ **then**
　　　　**return** $p_v$;
　**return** $null$;



# 匹配和最大匹配

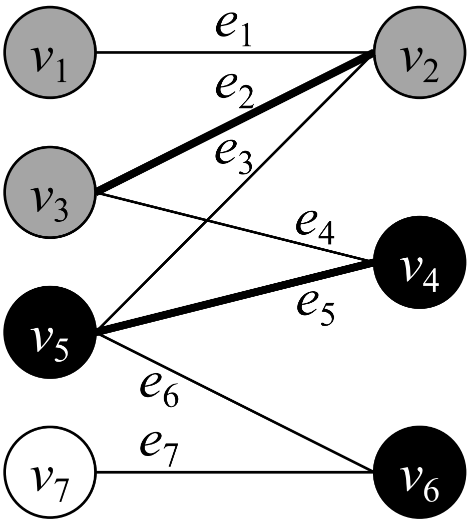
- 扩展DFS算法：判定并返回增广路
  - 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
  - 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
  - 否则，未找到$M$-增广路，算法返回null

$M = \{e_5, e_2\}$

**算法 9:** DFSAP

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

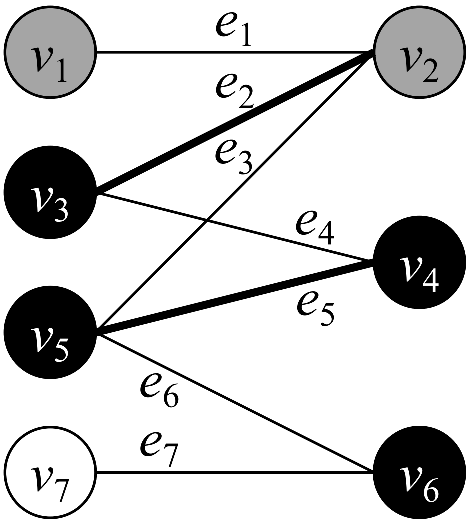
$u$.visited ← true;
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**
    **return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
    **foreach** $(u, v) \in E$ **do**
        **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
            $p_v$ ← DFSAP($G$, $v$, $M$);
            **if** $p_v \neq null$ **then**
                **return** $p_v$;
    **return** *null*;



# 匹配和最大匹配

- 扩展DFS算法：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

$M = \{e_5, e_2\}$




**算法 9:** DFSAP

```
输入: 图 G = ⟨X ∪ Y, E⟩，顶点 u，匹配 M
u.visited ← true;
if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
    return DFS 树中从根顶点到 u 的路;
else
    foreach (u, v) ∈ E do
        if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M-交错路 then
            p_v ← DFSAP(G, v, M);
            if p_v ≠ null then
                return p_v;
    return null;
```

# 匹配和最大匹配

## ■ 扩展DFS算法：判定并返回增广路

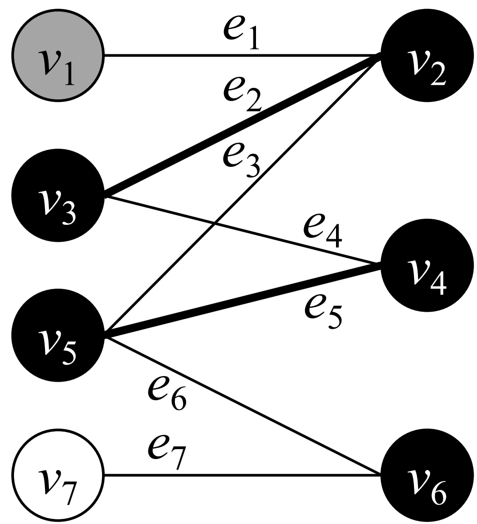
- 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
- 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
- 否则，未找到$M$-增广路，算法返回null

$M = \{e_5, e_2\}$

**算法 9:** DFSAP

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

```
u.visited ← true;
if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
│  return DFS 树中从根顶点到 u 的路;
else
│  foreach (u, v) ∈ E do
│  │  if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M-交
│  │   错路 then
│  │  │  p_v ← DFSAP(G, v, M);
│  │  │  if p_v ≠ null then
│  │  │  │  return p_v;
│  return null;
```



# 匹配和最大匹配

## 扩展DFS算法：判定并返回增广路

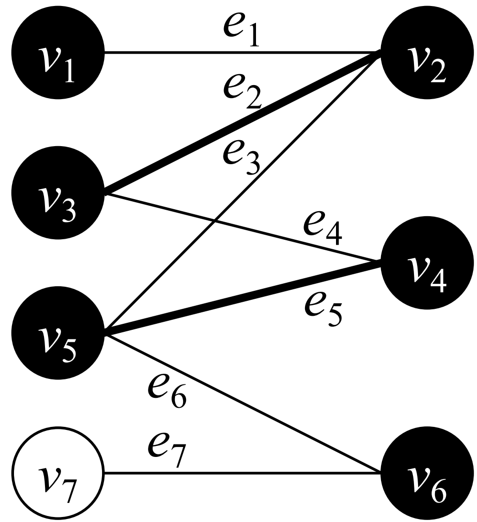
- 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
- 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
- 否则，未找到$M$-增广路，算法返回null

**算法 9:** DFSAP

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

```
u.visited ← true;
if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
    return DFS 树中从根顶点到 u 的路;
else
    foreach (u, v) ∈ E do
        if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M-交错路 then
            p_v ← DFSAP(G, v, M);
            if p_v ≠ null then
                return p_v;
    return null;
```

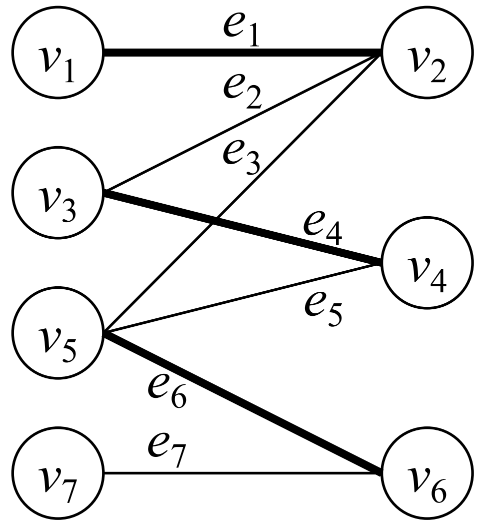
$M=\{e_5, e_2\}$



# 匹配和最大匹配

- 扩展DFS算法：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

$M = \{e_1, e_4, e_6\}$




**算法 9:** DFSAP

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

$u$.visited $\leftarrow$ true;
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**
  **return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
  **foreach** $(u, v) \in E$ **do**
    **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
      $p_v \leftarrow$ DFSAP($G$, $v$, $M$);
      **if** $p_v \neq null$ **then**
        **return** $p_v$;
  **return** *null*;

# 匹配和最大匹配

## 扩展DFS算法：判定并返回增广路

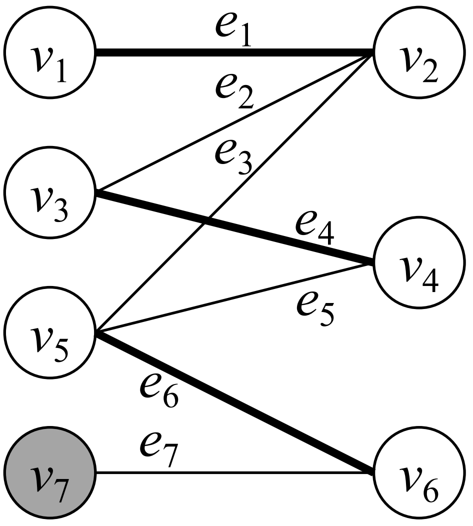
- 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
- 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
- 否则，未找到$M$-增广路，算法返回null

$M = \{e_1, e_4, e_6\}$

**算法 9:** DFSAP

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

$u$.visited ← true;
**if** $u$ 未被 $M$ 饱和且 $u \neq DFS$ 树的根顶点 **then**
  **return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
  **foreach** $(u, v) \in E$ **do**
    **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
      $p_v \leftarrow$ DFSAP($G$, $v$, $M$);
      **if** $p_v \neq null$ **then**
        **return** $p_v$;
  **return** *null*;



# 匹配和最大匹配

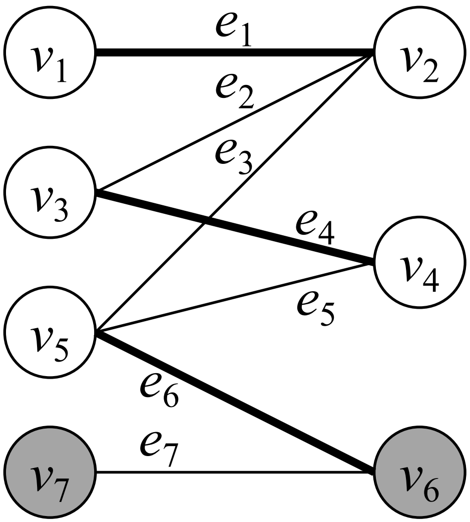
- 扩展DFS算法：判定并返回增广路
  - 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
  - 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
  - 否则，未找到$M$-增广路，算法返回null

$$M = \{e_1, e_4, e_6\}$$

**算法 9:** DFSAP

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

$u$.visited ← true;
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**
  **return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
  **foreach** $(u, v) \in E$ **do**
    **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
      $p_v \leftarrow$ DFSAP($G$, $v$, $M$);
      **if** $p_v \neq null$ **then**
        **return** $p_v$;
  **return** $null$;

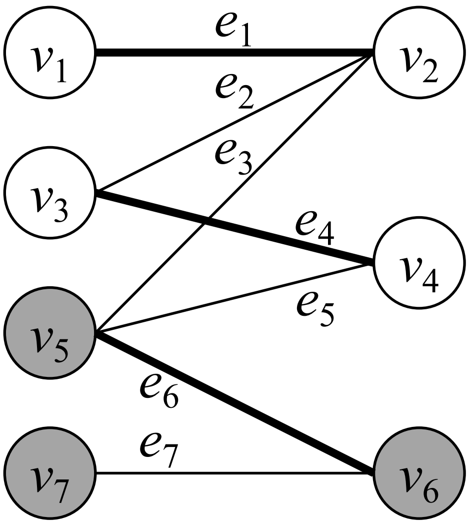


# 匹配和最大匹配

## 扩展DFS算法：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

$M = \{e_1, e_4, e_6\}$




**算法 9:** DFSAP

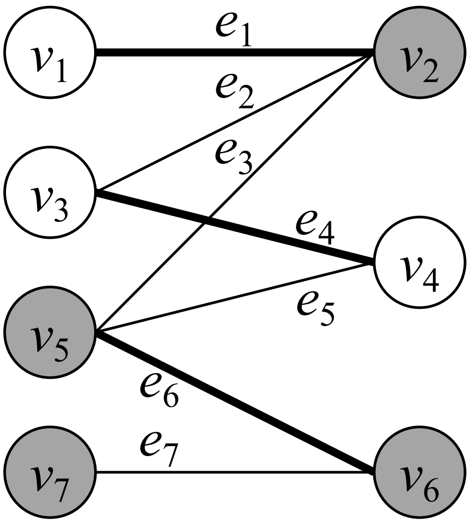
**输入:** 图 $G = \langle X \cup Y, E \rangle$，顶点 $u$，匹配 $M$

$u$.visited $\leftarrow$ true;
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**
  **return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
  **foreach** $(u, v) \in E$ **do**
    **if** $v$.*visited* = *false* 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
      $p_v \leftarrow$ DFSAP($G$, $v$, $M$);
      **if** $p_v \neq$ *null* **then**
        **return** $p_v$;
  **return** *null*;

# 匹配和最大匹配

- 扩展DFS算法：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

$$M = \{e_1, e_4, e_6\}$$




**算法 9:** DFSAP

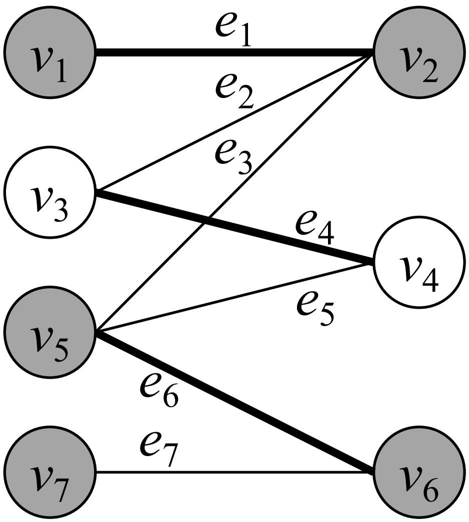
**输入:** 图 $G = \langle X \cup Y, E \rangle$，顶点 $u$，匹配 $M$

```
u.visited ← true;
if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
│ return DFS 树中从根顶点到 u 的路;
else
│ foreach (u, v) ∈ E do
│ │ if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M-交
│ │   错路 then
│ │ │ p_v ← DFSAP(G, v, M);
│ │ │ if p_v ≠ null then
│ │ │ │ return p_v;
│ return null;
```

# 匹配和最大匹配

- 扩展DFS算法：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

$M = \{e_1, e_4, e_6\}$




**算法 9:** DFSAP

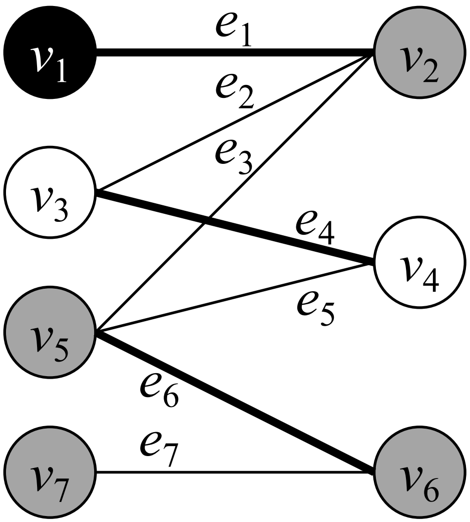
**输入:** 图 $G = \langle X \cup Y, E\rangle$，顶点 $u$，匹配 $M$

```
u.visited ← true;
if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
    return DFS 树中从根顶点到 u 的路;
else
    foreach (u, v) ∈ E do
        if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M-交
          错路 then
            p_v ← DFSAP(G, v, M);
            if p_v ≠ null then
                return p_v;
    return null;
```

# 匹配和最大匹配

- 扩展DFS算法：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

$$M = \{e_1, e_4, e_6\}$$




**算法 9:** DFSAP

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

$u$.visited ← true;
**if** $u$ 未被 $M$ 饱和且 $u \neq DFS$ 树的根顶点 **then**
  **return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
  **foreach** $(u, v) \in E$ **do**
    **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
      $p_v$ ← DFSAP($G$, $v$, $M$);
      **if** $p_v \neq null$ **then**
        **return** $p_v$;
  **return** *null*;

# 匹配和最大匹配

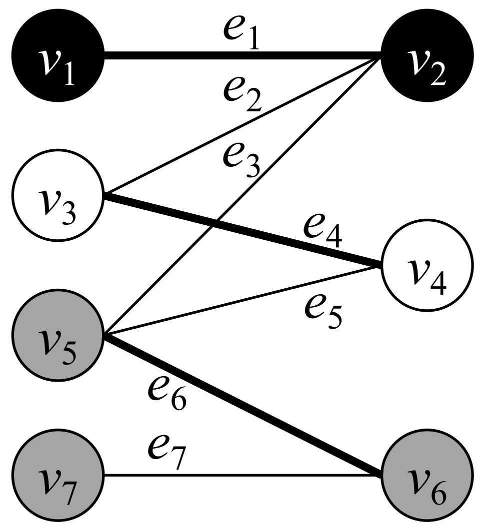
- 扩展DFS算法：判定并返回增广路
  - 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
  - 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
  - 否则，未找到$M$-增广路，算法返回null

$$M = \{e_1, e_4, e_6\}$$

**算法 9:** DFSAP

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

$u$.visited ← true;<br>
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**<br>
    **return** *DFS* 树中从根顶点到 $u$ 的路;<br>
**else**<br>
    **foreach** $(u, v) \in E$ **do**<br>
        **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**<br>
            $p_v$ ← DFSAP($G$, $v$, $M$);<br>
            **if** $p_v \neq null$ **then**<br>
                **return** $p_v$;<br>
    **return** $null$;



# 匹配和最大匹配

## 扩展DFS算法：判定并返回增广路

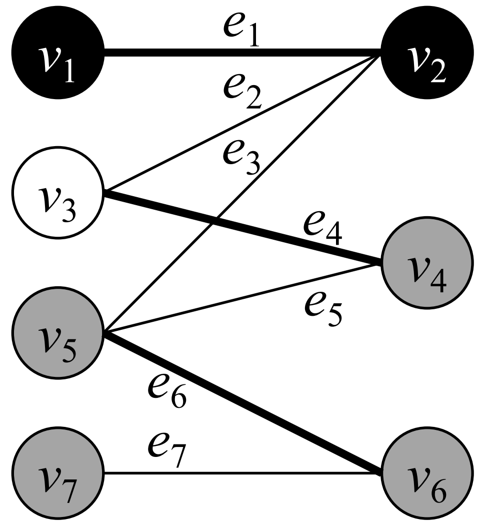
- 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
- 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
- 否则，未找到$M$-增广路，算法返回null

$$M = \{e_1, e_4, e_6\}$$

**算法 9:** DFSAP

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

```
u.visited ← true;
if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
    return DFS 树中从根顶点到 u 的路;
else
    foreach (u, v) ∈ E do
        if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M-交错路 then
            p_v ← DFSAP(G, v, M);
            if p_v ≠ null then
                return p_v;
    return null;
```



# 匹配和最大匹配

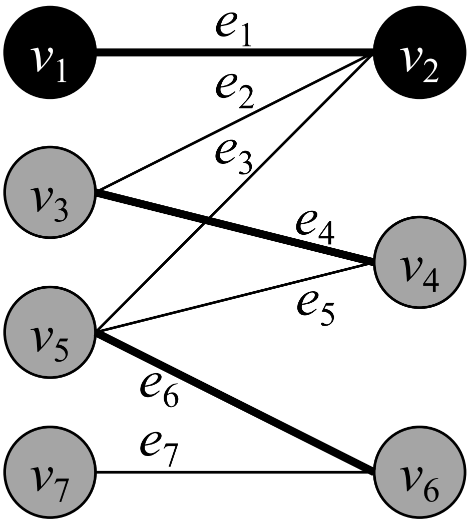
- 扩展DFS算法：判定并返回增广路
  - 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
  - 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
  - 否则，未找到$M$-增广路，算法返回null

$M = \{e_1, e_4, e_6\}$

**算法 9:** DFSAP

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

$u$.visited ← true;<br>
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**<br>
    **return** *DFS* 树中从根顶点到 $u$ 的路;<br>
**else**<br>
    **foreach** $(u, v) \in E$ **do**<br>
        **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**<br>
            $p_v$ ← DFSAP($G$, $v$, $M$);<br>
            **if** $p_v \neq null$ **then**<br>
                **return** $p_v$;<br>
    **return** *null*;



# 匹配和最大匹配

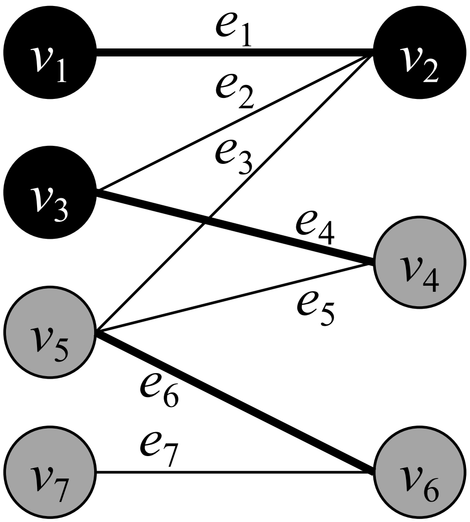
- 扩展DFS算法：判定并返回增广路
  - 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
  - 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
  - 否则，未找到$M$-增广路，算法返回null

$$M = \{e_1, e_4, e_6\}$$

**算法 9:** DFSAP

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

```
u.visited ← true;
if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
    return DFS 树中从根顶点到 u 的路;
else
    foreach (u, v) ∈ E do
        if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M-交
          错路 then
            p_v ← DFSAP(G, v, M);
            if p_v ≠ null then
                return p_v;
    return null;
```



# 匹配和最大匹配

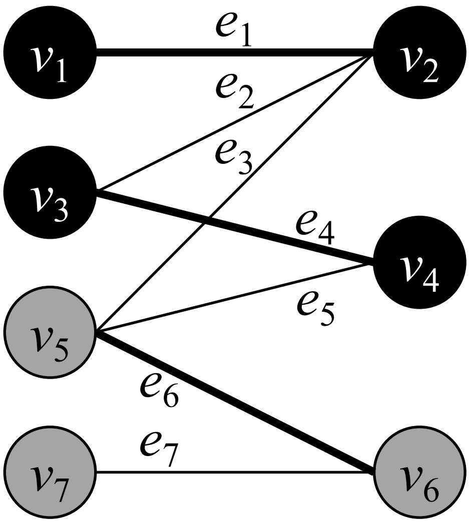
- 扩展DFS算法：判定并返回增广路
  - 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
  - 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
  - 否则，未找到$M$-增广路，算法返回null

$$M = \{e_1, e_4, e_6\}$$

**算法 9:** DFSAP

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

$u$.visited $\leftarrow$ true;<br>
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**<br>
    **return** *DFS* 树中从根顶点到 $u$ 的路;<br>
**else**<br>
    **foreach** $(u, v) \in E$ **do**<br>
        **if** $v$.*visited* $=$ *false* 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**<br>
            $p_v \leftarrow$ DFSAP($G$, $v$, $M$);<br>
            **if** $p_v \neq$ *null* **then**<br>
                **return** $p_v$;<br>
    **return** *null*;



# 匹配和最大匹配

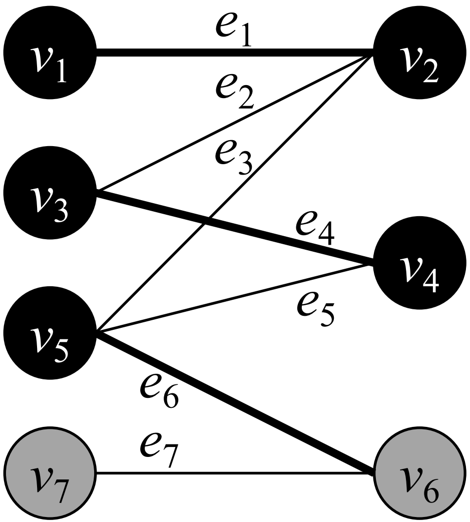
- 扩展DFS算法：判定并返回增广路
  - 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
  - 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
  - 否则，未找到$M$-增广路，算法返回null

$$M = \{e_1, e_4, e_6\}$$

**算法 9:** DFSAP

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

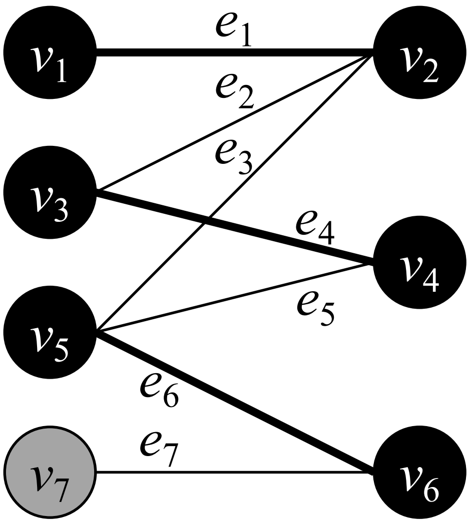
```
u.visited ← true;
if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
    return DFS 树中从根顶点到 u 的路;
else
    foreach (u, v) ∈ E do
        if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M-交错路 then
            p_v ← DFSAP(G, v, M);
            if p_v ≠ null then
                return p_v;
    return null;
```



# 匹配和最大匹配

- 扩展DFS算法：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

$M = \{e_1, e_4, e_6\}$




**算法 9:** DFSAP

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

$u$.visited $\leftarrow$ true;
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**
  **return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
  **foreach** $(u, v) \in E$ **do**
    **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
      $p_v \leftarrow$ DFSAP($G$, $v$, $M$);
      **if** $p_v \neq null$ **then**
        **return** $p_v$;
  **return** $null$;

# 匹配和最大匹配

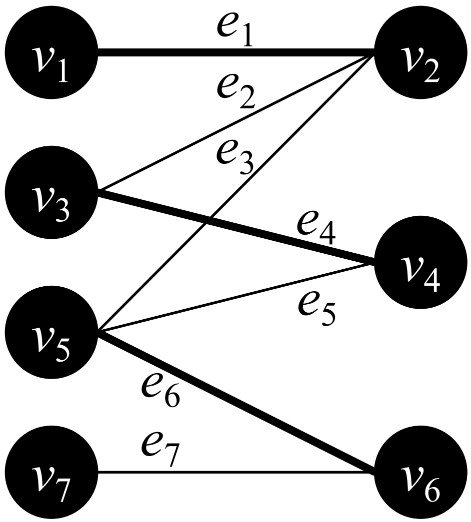
- 扩展DFS算法：判定并返回增广路
  - 若非根顶点$u$未被$M$饱和，则DFS树中从根顶点到$u$的$M$-交错路是$M$-增广路，算法返回这条路
  - 否则，若从$u$对邻点$v$的递归调用找到$M$-增广路$p_v$，则算法返回$p_v$
  - 否则，未找到$M$-增广路，算法返回null

$M = \{e_1, e_4, e_6\}$

**算法 9:** DFSAP

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

$u$.visited ← true;
**if** $u$ 未被 $M$ 饱和且 $u \neq DFS$ 树的根顶点 **then**
  **return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
  **foreach** $(u, v) \in E$ **do**
    **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
      $p_v$ ← DFSAP($G$, $v$, $M$);
      **if** $p_v \neq null$ **then**
        **return** $p_v$;
  **return** *null*;

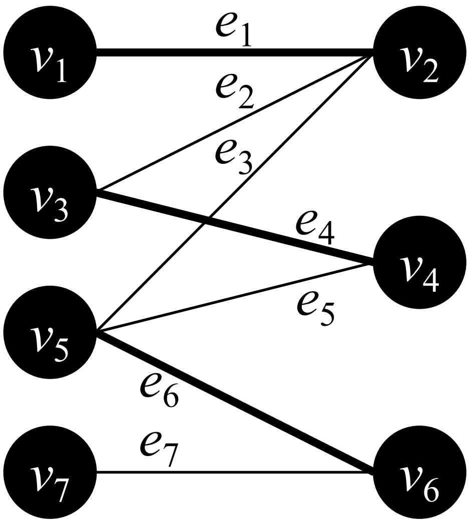


# 匹配和最大匹配

## 扩展DFS算法：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
- 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？

$M = \{e_1, e_4, e_6\}$




**算法 9:** DFSAP

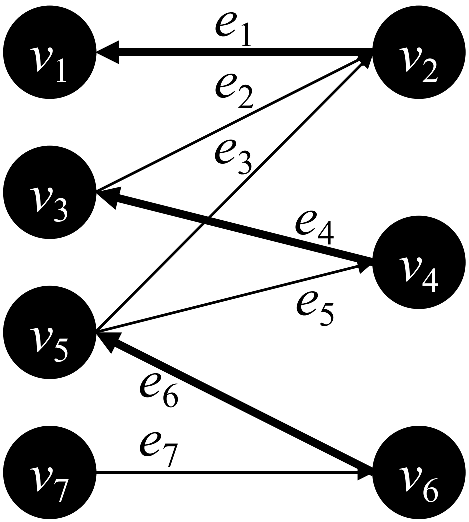
**输入:** 图 $G = \langle X \cup Y, E \rangle$，顶点 $u$，匹配 $M$

```
u.visited ← true;
if u 未被 M 饱和且 u ≠ DFS 树的根顶点 then
    return DFS 树中从根顶点到 u 的路;
else
    foreach (u, v) ∈ E do
        if v.visited = false 且 DFS 树中从根顶点到 v 的路是 M-交
          错路 then
            p_v ← DFSAP(G, v, M);
            if p_v ≠ null then
                return p_v;
    return null;
```

# 匹配和最大匹配

- 扩展DFS算法：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
  - 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？

$M = \{e_1, e_4, e_6\}$




**算法 9:** DFSAP

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

$u$.visited $\leftarrow$ true;
**if** $u$ 未被 $M$ 饱和且 $u \neq$ *DFS* 树的根顶点 **then**
  **return** *DFS* 树中从根顶点到 $u$ 的路;
**else**
  **foreach** $(u, v) \in E$ **do**
    **if** $v.visited = false$ 且 *DFS* 树中从根顶点到 $v$ 的路是 $M$-交错路 **then**
      $p_v \leftarrow$ DFSAP($G$, $v$, $M$);
      **if** $p_v \neq null$ **then**
        **return** $p_v$;
  **return** *null*;

# 匹配和最大匹配

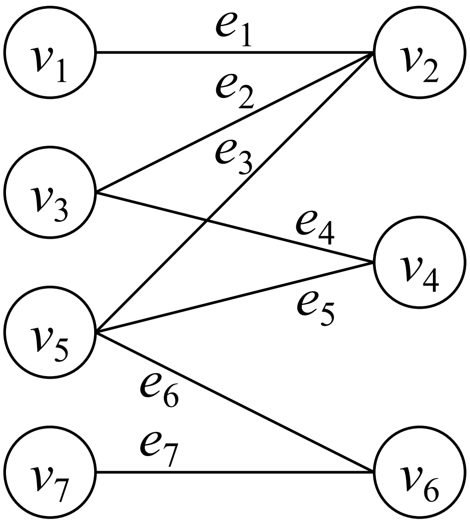
- 时间复杂度：$O(n(n+m))$
  - do-while循环运行多少轮？

**算法 8:** 匈牙利算法

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

**初值:** $M$ 初值为 $\emptyset$

```
do
    foreach u ∈ (X ∪ Y) do
        u.visited ← false;
    foreach r ∈ X do
        if r.visited = false 且 r 未被 M 饱和 then
            p ← DFSAP(G, r, M);
            if p ≠ null then
                M ← Edges(p) △ M;
                中止 ForEach 循环;
while p ≠ null;
输出 (M);
```



# 匹配和最大匹配

- 二分图
  - 匈牙利算法
  - 霍普克罗夫特-卡普算法：同时找多条最短增广路
- 非二分图
  - 花算法

# 匹配和最大匹配

- John Hopcroft，1939-，出生于美国，1986年获图灵奖
- Richard M. Karp，1935-，出生于美国，1985年获图灵奖

多项式规约与21个NPC问题

https://en.wikipedia.org/wiki/John_Hopcroft
https://en.wikipedia.org/wiki/Richard_M._Karp

# 匹配和最大匹配

- 改进1：通过改进BFS算法来找增广路，比通过改进DFS算法找到的增广路更短，从而提高单轮do-while循环的搜索效率
- 改进2：每轮do-while循环尝试同时找多条增广路，匹配的规模增幅更大，从而减少do-while循环的轮数

**算法 10:** 霍普克罗夫特-卡普算法

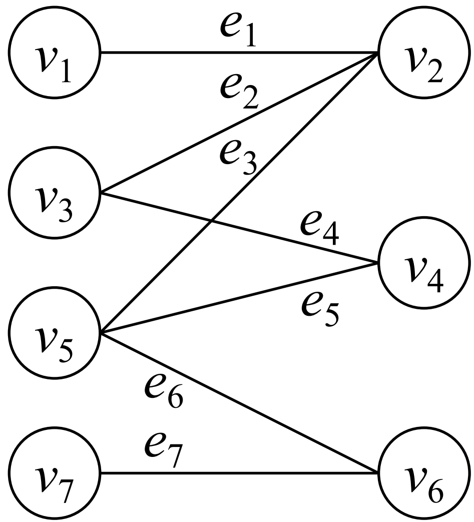
**输入:** 二分图 $G = \langle X \cup Y, E \rangle$

**初值:** $M$ 初值为 $\emptyset$

**do**
- $Q \leftarrow$ `HKInit`$(G)$;
- $Y' \leftarrow$ `HKBFS`$(G, Q)$;
- $P \leftarrow$ `HKPaths`$(G, Y')$;
- **foreach** $p \in P$ **do**
  - $M \leftarrow$ `Edges`$(p) \triangle M$;

**while** $P \neq \emptyset$;

输出 $(M)$;



# 匹配和最大匹配

- $Q$：$X$中所有未被匹配$M$饱和的顶点子集
- $Y'$：通过$M$-交错路找到的$Y$中未被$M$饱和的顶点子集$Y'$
- $P$：以$Y'$中的顶点为终点的一组不经过相同顶点的$M$-增广路

**算法 10:** 霍普克罗夫特-卡普算法

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

**初值:** $M$ 初值为 $\emptyset$

**do**

  $Q \leftarrow \texttt{HKInit}(G)$;

  $Y' \leftarrow \texttt{HKBFS}(G, Q)$;

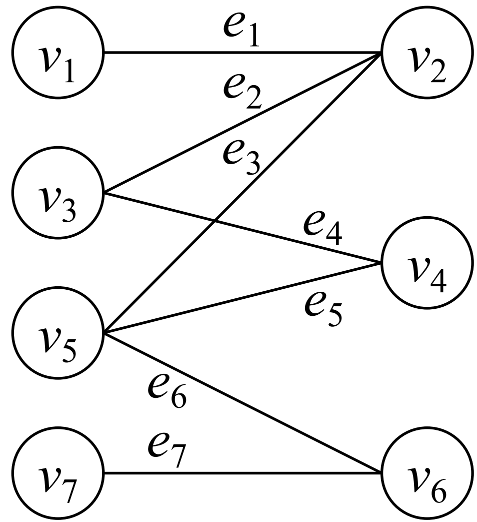
  $P \leftarrow \texttt{HKPaths}(G, Y')$;

  **foreach** $p \in P$ **do**

    $M \leftarrow \texttt{Edges}(p) \triangle M$;

**while** $P \neq \emptyset$;

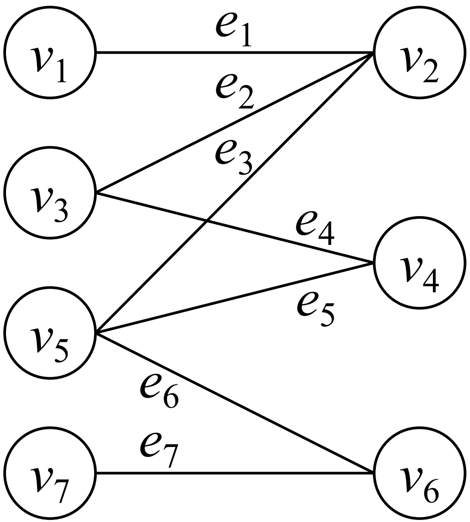
输出 $(M)$;



# 匹配和最大匹配

- $Q$：$X$中所有未被匹配$M$饱和的顶点子集

```
算法 11: HKInit
  输入: 二分图 G = ⟨X ∪ Y, E⟩
  初值: Q 初值为空队列
  foreach u ∈ (X ∪ Y) do
  |  if u ∈ X 且 u 未被 M 饱和 then
  |  |  u.visited ← true;
  |  |  u.d ← 0;
  |  |  入队列 (Q, u);
  |  else
  |  |  u.visited ← false;
  |  |  u.d ← ∞;
  return Q;
```



# 匹配和最大匹配

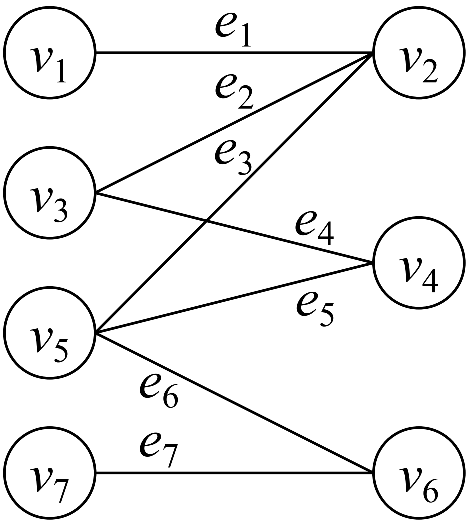
- $Q$：$X$中所有未被匹配$M$饱和的顶点子集
- 顶点的d属性：该顶点和$Q$中所有顶点间的最短距离

**算法 11:** HKInit

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

**初值:** $Q$ 初值为空队列

**foreach** $u \in (X \cup Y)$ **do**
    **if** $u \in X$ 且 $u$ 未被 $M$ 饱和 **then**
        $u$.visited $\leftarrow$ true;
        $u$.d $\leftarrow 0$;
        入队列（$Q$, $u$）;
    **else**
        $u$.visited $\leftarrow$ false;
        $u$.d $\leftarrow \infty$;
**return** $Q$;



# 匹配和最大匹配

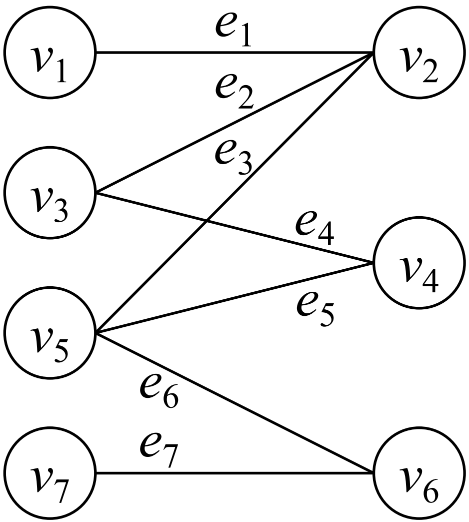
- $Y'$：通过$M$-交错路找到的$Y$中未被$M$饱和的顶点子集$Y'$
  - 扩展BFS算法

**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**<br>
    $v \leftarrow$ 出队列（$Q$）;<br>
    **if** $v.d > d'$ **then**<br>
        中止 while 循环;<br>
    **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**<br>
        $Y' \leftarrow Y' \cup \{v\}$;<br>
        $d' \leftarrow v.\mathrm{d}$;<br>
    **else**<br>
        **foreach** $(v, w) \in E$ **do**<br>
            **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**<br>
                $w.\mathrm{visited} \leftarrow \mathrm{true}$;<br>
                $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;<br>
                入队列（$Q$, $w$）;<br>
**return** $Y'$;

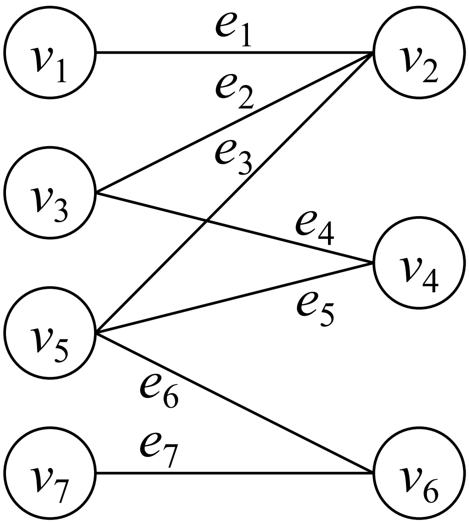


# 匹配和最大匹配

- $Y'$：通过$M$-交错路找到的$Y$中未被$M$饱和的顶点子集$Y'$
  - 扩展BFS算法：限制了可访问的邻点

**算法 12:** HKBFS

```
输入: 二分图 G = ⟨X ∪ Y, E⟩, 队列 Q
初值: Y′ 初值为 ∅; d′ 初值为 ∞
while Q 非空 do
    v ← 出队列 (Q);
    if v.d > d′ then
        中止 while 循环;
    else if v 未被 M 饱和且 v.d > 0 then
        Y′ ← Y′ ∪ {v};
        d′ ← v.d;
    else
        foreach (v, w) ∈ E do
            if w.visited = false 且 BFS 树中从根顶点到 w 的路是
             M-交错路 then
                w.visited ← true;
                w.d ← v.d + 1;
                入队列 (Q, w);
return Y′;
```



# 匹配和最大匹配

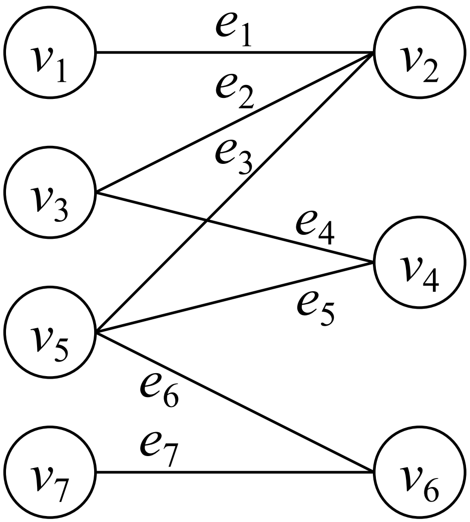
- $Y'$：通过$M$-交错路找到的$Y$中未被$M$饱和的顶点子集$Y'$
  - 扩展BFS算法：将未被$M$饱和的非根顶点$v$作为$M$-增广路的终点

**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
    $v \leftarrow$ 出队列（$Q$）;
    **if** $v.d > d'$ **then**
        中止 while 循环;
    **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
        $Y' \leftarrow Y' \cup \{v\}$;
        $d' \leftarrow v.\mathrm{d}$;
    **else**
        **foreach** $(v, w) \in E$ **do**
            **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
                $w.\mathrm{visited} \leftarrow \mathrm{true}$;
                $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;
                入队列（$Q$, $w$）;
**return** $Y'$;



# 匹配和最大匹配

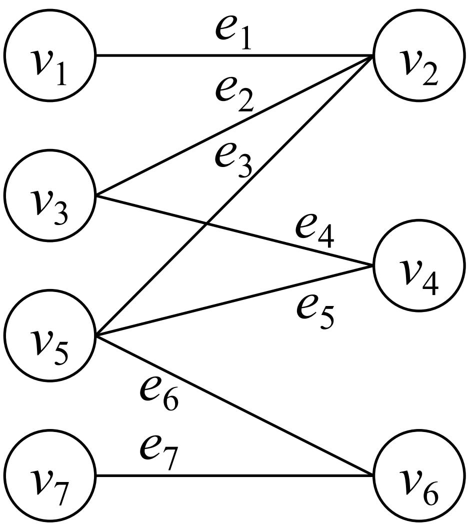
- $Y'$：通过$M$-交错路找到的$Y$中未被$M$饱和的顶点子集$Y'$
  - 扩展BFS算法：将未被$M$饱和的非根顶点$v$作为$M$-增广路的终点
    - $d'$：被访问的首个未被$M$饱和的非根顶点的d属性值
    - 在访问了d属性值不超过$d'$的所有顶点后，不再访问d属性值更大的顶点

**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
  $v \leftarrow$ 出队列（$Q$）；
  **if** $v.d > d'$ **then**
    中止 while 循环；
  **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
    $Y' \leftarrow Y' \cup \{v\}$；
    $d' \leftarrow v.\mathrm{d}$；
  **else**
    **foreach** $(v, w) \in E$ **do**
      **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
        $w.\mathrm{visited} \leftarrow \mathrm{true}$；
        $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$；
        入队列（$Q$，$w$）；
**return** $Y'$；



# 匹配和最大匹配

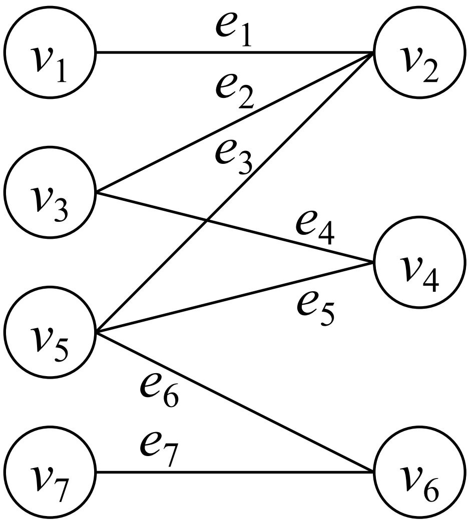
- $Y'$：所有d属性值为$d'$的未被$M$饱和的非根顶点
  - 扩展BFS算法：将未被$M$饱和的非根顶点$v$作为$M$-增广路的终点
    - $d'$：被访问的首个未被$M$饱和的非根顶点的d属性值
    - 在访问了d属性值不超过$d'$的所有顶点后，不再访问d属性值更大的顶点

**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

```
while Q 非空 do
    v ← 出队列 (Q);
    if v.d > d' then
        中止 while 循环;
    else if v 未被 M 饱和且 v.d > 0 then
        Y' ← Y' ∪ {v};
        d' ← v.d;
    else
        foreach (v, w) ∈ E do
            if w.visited = false 且 BFS 树中从根顶点到 w 的路是
             M-交错路 then
                w.visited ← true;
                w.d ← v.d + 1;
                入队列 (Q, w);
return Y';
```



# 匹配和最大匹配

- $P$：以$Y'$中的顶点为终点的一组不经过相同顶点的$M$-增广路
  - 从$Y'$出发，沿顶点d属性值递减的方向，找出并返回极多的一组$M$-增广路$P$，并要求这些增广路不经过相同顶点

**算法 10:** 霍普克罗夫特-卡普算法

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

**初值:** $M$ 初值为 $\emptyset$

**do**

  $Q \leftarrow$ `HKInit`$(G)$;

  $Y' \leftarrow$ `HKBFS`$(G, Q)$;

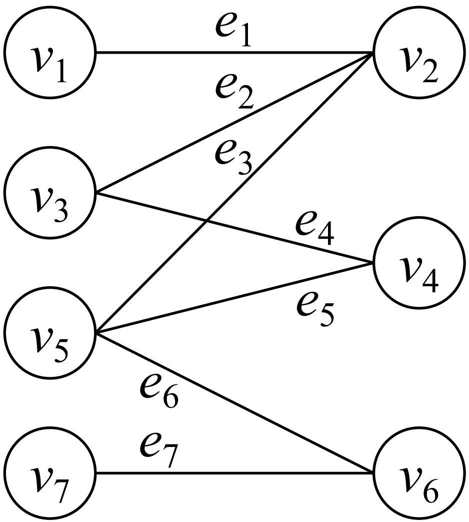
  $P \leftarrow$ `HKPaths`$(G, Y')$;

  **foreach** $p \in P$ **do**

    $M \leftarrow$ `Edges`$(p) \triangle M$;

**while** $P \neq \emptyset$;

输出 （$M$）;



# 匹配和最大匹配

- $P$：以$Y$中的顶点为终点的一组不经过相同顶点的$M$-增广路
  - 从$Y$出发，沿顶点d属性值递减的方向，找出并返回极多的一组$M$-增广路$P$，并要求这些增广路不经过相同顶点
- 它们经过的边集可以互不干扰地和$M$计算对称差

**算法 10:** 霍普克罗夫特-卡普算法

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

**初值:** $M$ 初值为 $\emptyset$

**do**

  $Q \leftarrow$ HKInit($G$);

  $Y' \leftarrow$ HKBFS($G$, $Q$);

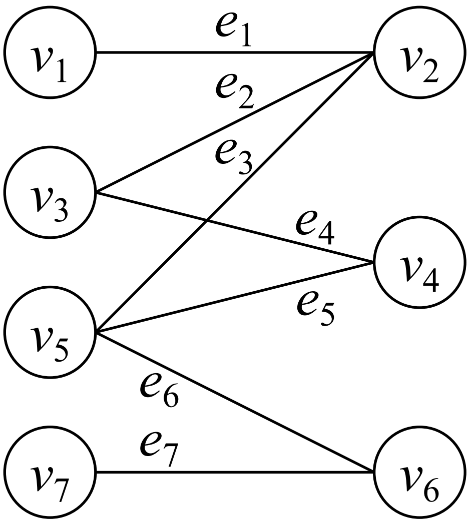
  $P \leftarrow$ HKPaths($G$, $Y'$);

  **foreach** $p \in P$ **do**

    $M \leftarrow$ Edges($p$) $\triangle$ $M$;

**while** $P \neq \emptyset$;

输出 ($M$);



# 匹配和最大匹配

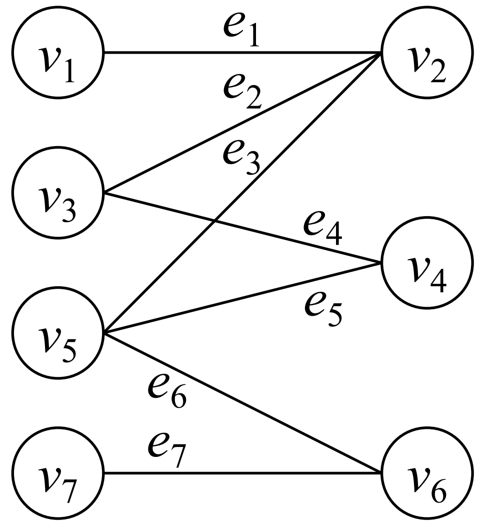
- 第1轮do-while循环开始

**算法 10:** 霍普克罗夫特-卡普算法

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

**初值:** $M$ 初值为 $\emptyset$

**do**
  $Q \leftarrow \texttt{HKInit}(G)$;
  $Y' \leftarrow \texttt{HKBFS}(G, Q)$;
  $P \leftarrow \texttt{HKPaths}(G, Y')$;
  **foreach** $p \in P$ **do**
    $M \leftarrow \texttt{Edges}(p) \triangle M$;
**while** $P \neq \emptyset$;
输出 $(M)$;

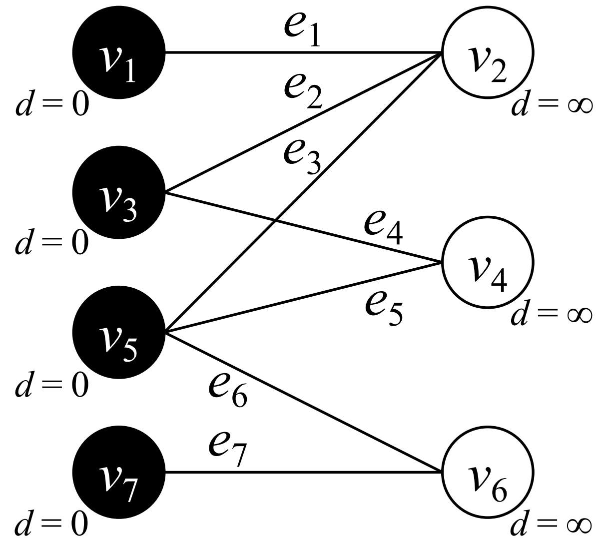


# 匹配和最大匹配

| $Q$ | $v_1$ | $v_3$ | $v_5$ | $v_7$ |
| --- | --- | --- | --- | --- |

**算法 11:** HKInit

```
输入: 二分图 G = ⟨X ∪ Y, E⟩
初值: Q 初值为空队列
foreach u ∈ (X ∪ Y) do
    if u ∈ X 且 u 未被 M 饱和 then
        u.visited ← true;
        u.d ← 0;
        入队列 (Q, u);
    else
        u.visited ← false;
        u.d ← ∞;
return Q;
```



# 匹配和最大匹配

| $Q$ | $v_1$ | $v_3$ | $v_5$ | $v_7$ |
|---|---|---|---|---|

$Y' = \{\}$

$d' = \infty$

---

**算法 12:** HKBFS

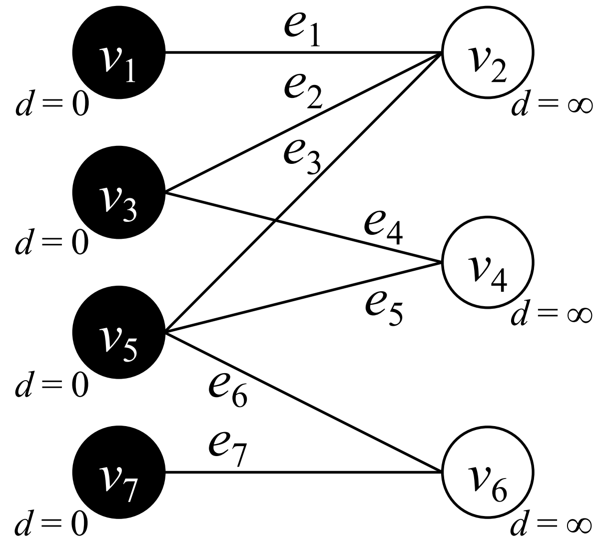
---

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
　　$v \leftarrow$ 出队列（$Q$）;
　　**if** $v.d > d'$ **then**
　　　　中止 while 循环;
　　**else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
　　　　$Y' \leftarrow Y' \cup \{v\}$;
　　　　$d' \leftarrow v.\mathrm{d}$;
　　**else**
　　　　**foreach** $(v, w) \in E$ **do**
　　　　　　**if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
　　　　　　　　$w.\mathrm{visited} \leftarrow \mathrm{true}$;
　　　　　　　　$w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;
　　　　　　　　入队列（$Q$, $w$）;
**return** $Y'$;

---



# 匹配和最大匹配

| $Q$ | $v_3$ | $v_5$ | $v_7$ | $v_2$ |
|---|---|---|---|---|

$Y' = \{\}$

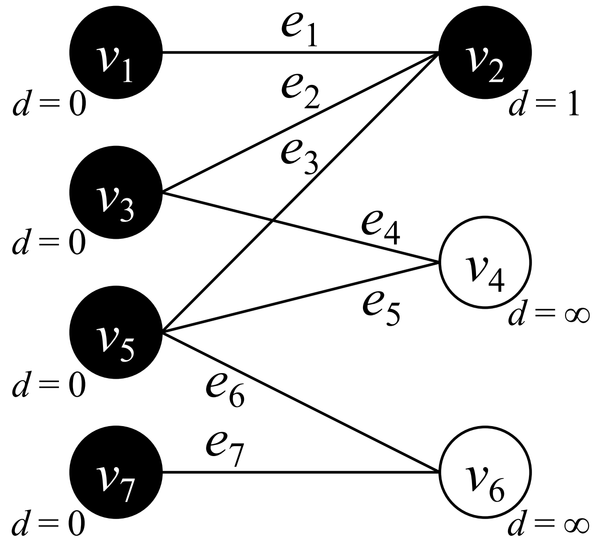
$d' = \infty$

**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
    $v \leftarrow$ 出队列（$Q$）;
    **if** $v.d > d'$ **then**
        中止 while 循环;
    **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
        $Y' \leftarrow Y' \cup \{v\}$;
        $d' \leftarrow v.\mathrm{d}$;
    **else**
        **foreach** $(v, w) \in E$ **do**
            **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
                $w.\mathrm{visited} \leftarrow \mathrm{true}$;
                $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;
                入队列（$Q$, $w$）;
**return** $Y'$;



# 匹配和最大匹配

**算法 12:** HKBFS

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

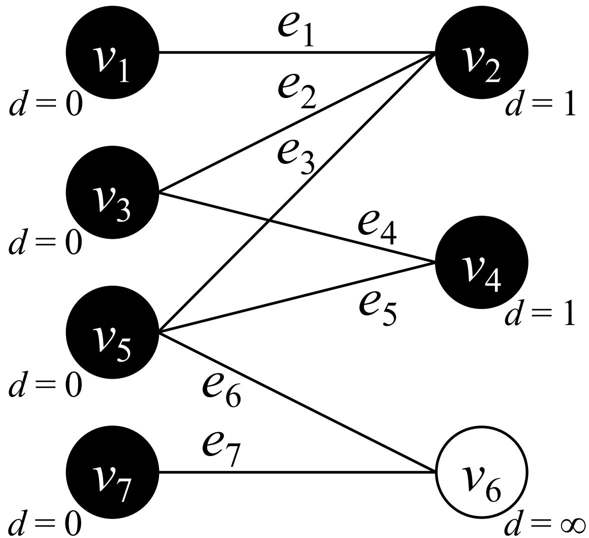
**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
    $v \leftarrow$ 出队列（$Q$）;
    **if** $v.d > d'$ **then**
        中止 while 循环;
    **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
        $Y' \leftarrow Y' \cup \{v\}$;
        $d' \leftarrow v.\mathrm{d}$;
    **else**
        **foreach** $(v, w) \in E$ **do**
            **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
                $w.\mathrm{visited} \leftarrow \mathrm{true}$;
                $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;
                入队列（$Q$, $w$）;
**return** $Y'$;

$Q$: $v_5$ $v_7$ $v_2$ $v_4$

$Y' = \{\}$

$d' = \infty$



# 匹配和最大匹配

$Q$ | $v_7$ $v_2$ $v_4$ $v_6$

$Y' = \{\}$

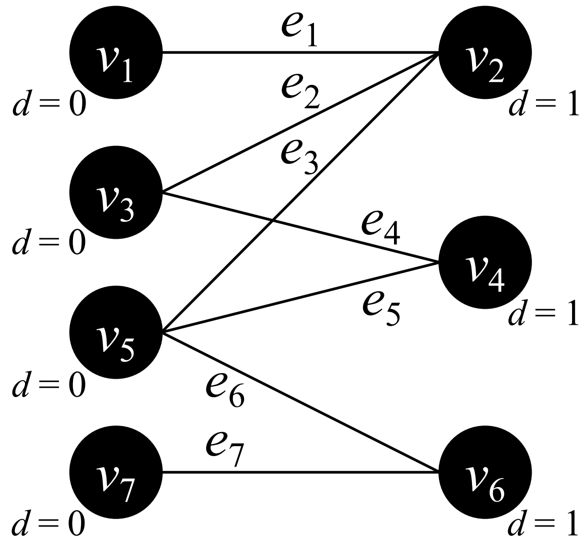
$d' = \infty$

**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
  $v \leftarrow$ 出队列（$Q$）;
  **if** $v.d > d'$ **then**
    中止 while 循环;
  **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
    $Y' \leftarrow Y' \cup \{v\}$;
    $d' \leftarrow v.\mathrm{d}$;
  **else**
    **foreach** $(v, w) \in E$ **do**
      **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
        $w.\mathrm{visited} \leftarrow \mathrm{true}$;
        $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;
        入队列（$Q$, $w$）;
**return** $Y'$;



# 匹配和最大匹配

| $Q$ | $v_2$ | $v_4$ | $v_6$ |
|---|---|---|---|

$Y' = \{\}$

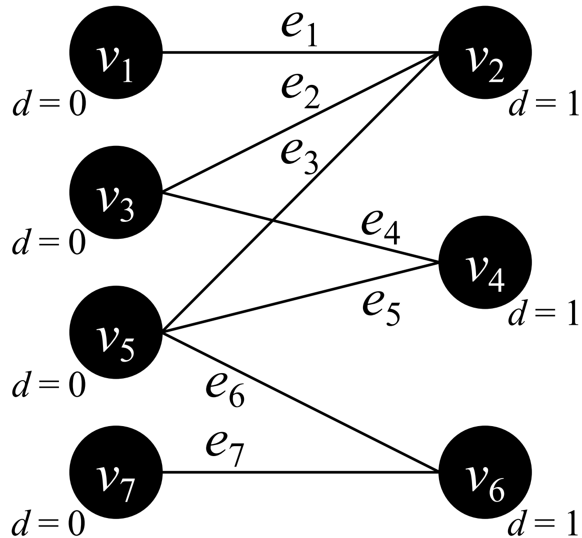
$d' = \infty$

**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
  $v \leftarrow$ 出队列（$Q$）;
  **if** $v.d > d'$ **then**
    中止 while 循环;
  **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
    $Y' \leftarrow Y' \cup \{v\}$;
    $d' \leftarrow v$.d;
  **else**
    **foreach** $(v, w) \in E$ **do**
      **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
        $w$.visited $\leftarrow$ true;
        $w$.d $\leftarrow v$.d + 1;
        入队列（$Q$, $w$）;
**return** $Y'$;



# 匹配和最大匹配

**算法 12:** HKBFS

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

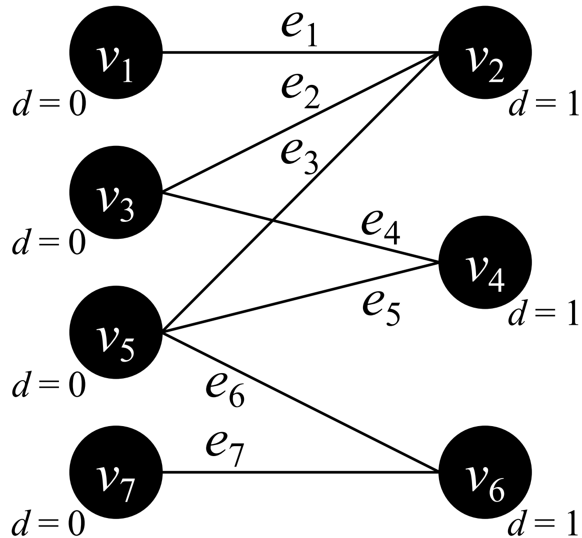
**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
  $v \leftarrow$ 出队列（$Q$）;
  **if** $v.d > d'$ **then**
    中止 while 循环;
  **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
    $Y' \leftarrow Y' \cup \{v\}$;
    $d' \leftarrow v.\mathrm{d}$;
  **else**
    **foreach** $(v, w) \in E$ **do**
      **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
        $w.\mathrm{visited} \leftarrow \mathrm{true}$;
        $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;
        入队列（$Q$, $w$）;
**return** $Y'$;

$Q$ | $v_4$ $v_6$

$Y' = \{v_2\}$

$d' = 1$



# 匹配和最大匹配

**算法 12:** HKBFS

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

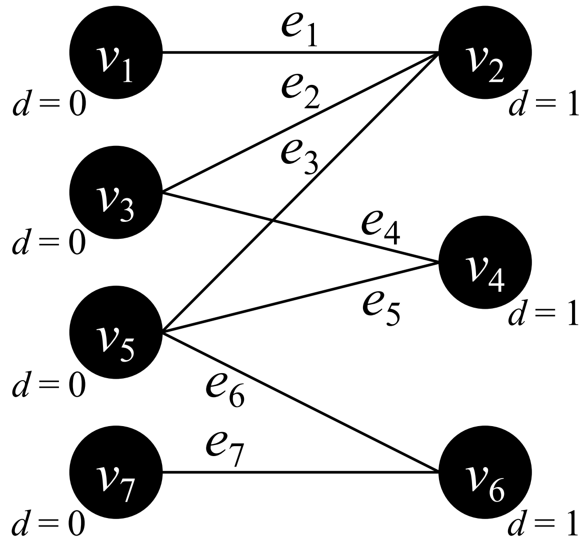
**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**<br>
 $v \leftarrow$ 出队列（$Q$）;<br>
 **if** $v.d > d'$ **then**<br>
  中止 while 循环;<br>
 **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**<br>
  $Y' \leftarrow Y' \cup \{v\}$;<br>
  $d' \leftarrow v.\mathrm{d}$;<br>
 **else**<br>
  **foreach** $(v, w) \in E$ **do**<br>
   **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**<br>
    $w.\mathrm{visited} \leftarrow \mathrm{true}$;<br>
    $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;<br>
    入队列（$Q$, $w$）;<br>
**return** $Y'$;

$Q$ $v_6$

$Y' = \{v_2, v_4\}$

$d' = 1$



# 匹配和最大匹配

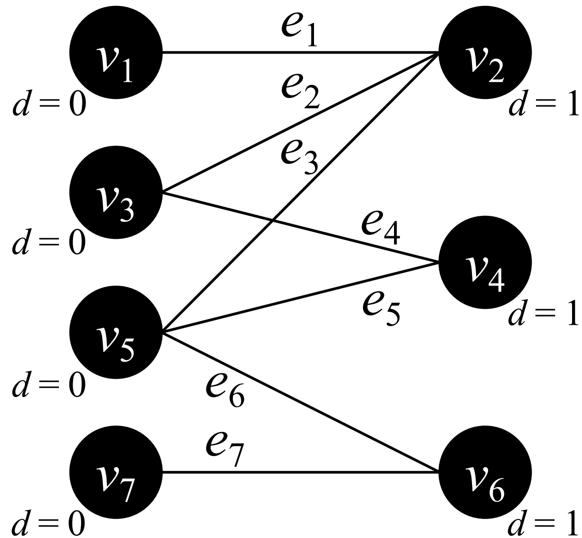
**算法 12:** HKBFS

```
输入: 二分图 G = ⟨X ∪ Y, E⟩, 队列 Q
初值: Y′ 初值为 ∅; d′ 初值为 ∞
while Q 非空 do
    v ← 出队列 (Q);
    if v.d > d′ then
        中止 while 循环;
    else if v 未被 M 饱和且 v.d > 0 then
        Y′ ← Y′ ∪ {v};
        d′ ← v.d;
    else
        foreach (v, w) ∈ E do
            if w.visited = false 且 BFS 树中从根顶点到 w 的路是
             M-交错路 then
                w.visited ← true;
                w.d ← v.d + 1;
                入队列 (Q, w);
return Y′;
```

$Q$

$Y' = \{v_2, v_4, v_6\}$

$d' = 1$



# 匹配和最大匹配

$$Y' = \{v_2, v_4, v_6\}$$

**算法 10:** 霍普克罗夫特-卡普算法

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

**初值:** $M$ 初值为 $\emptyset$

**do**

  $Q \leftarrow$ `HKInit`$(G)$;

  $Y' \leftarrow$ `HKBFS`$(G, Q)$;

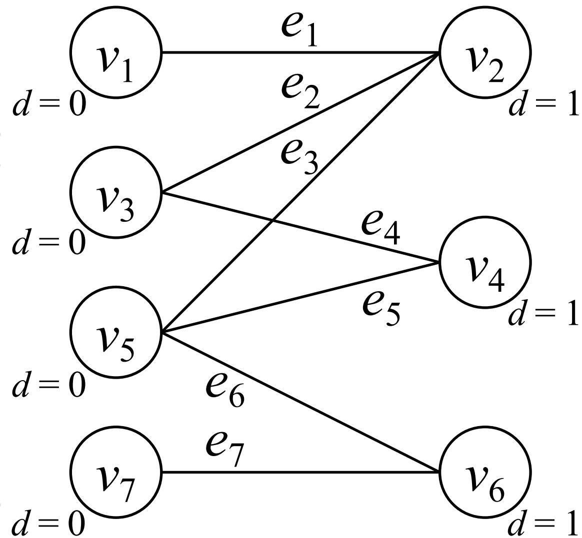
  $P \leftarrow$ `HKPaths`$(G, Y')$;

  **foreach** $p \in P$ **do**

    $M \leftarrow$ `Edges`$(p) \triangle M$;

**while** $P \neq \emptyset$;

输出 $(M)$;



# 匹配和最大匹配

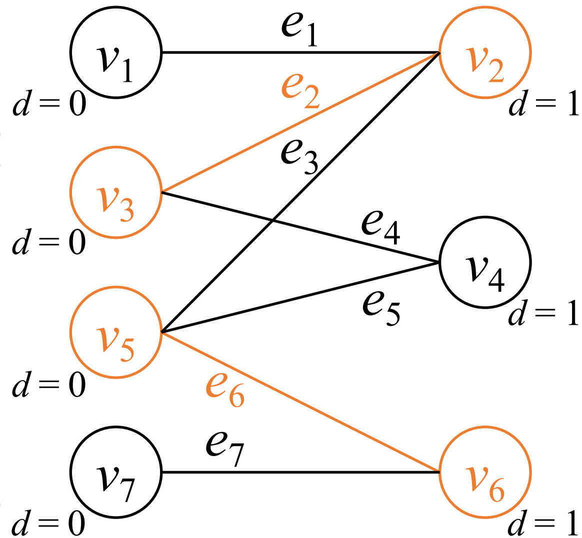
$$Y' = \{v_2,\ v_4,\ v_6\}$$

**算法 10:** 霍普克罗夫特-卡普算法

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

**初值:** $M$ 初值为 $\emptyset$

```
do
    Q ← HKInit(G);
    Y′ ← HKBFS(G, Q);
    P ← HKPaths(G, Y′);
    foreach p ∈ P do
        M ← Edges(p) △ M;
while P ≠ ∅;
输出 (M);
```



# 匹配和最大匹配

**算法 10:** 霍普克罗夫特-卡普算法

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

**初值:** $M$ 初值为 $\emptyset$

**do**

  $Q \leftarrow$ `HKInit`$(G)$;

  $Y' \leftarrow$ `HKBFS`$(G, Q)$;

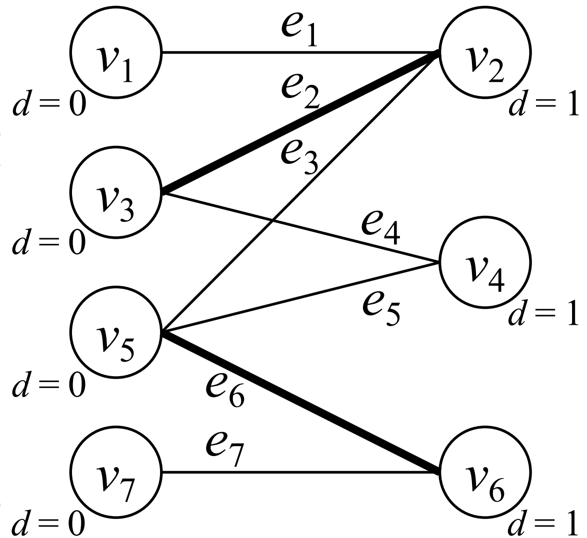
  $P \leftarrow$ `HKPaths`$(G, Y')$;

  **foreach** $p \in P$ **do**

    $M \leftarrow$ `Edges`$(p) \triangle M$;

**while** $P \neq \emptyset$;

输出 $(M)$;



# 匹配和最大匹配

- 第2轮do-while循环开始

**算法 10:** 霍普克罗夫特-卡普算法

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

**初值:** $M$ 初值为 $\emptyset$

**do**

  $Q \leftarrow \texttt{HKInit}(G)$;

  $Y' \leftarrow \texttt{HKBFS}(G, Q)$;

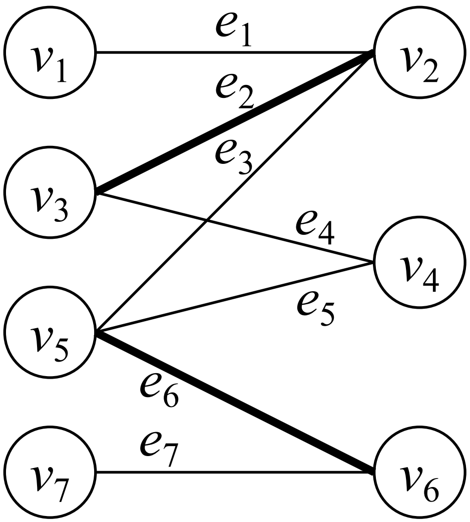
  $P \leftarrow \texttt{HKPaths}(G, Y')$;

  **foreach** $p \in P$ **do**

    $M \leftarrow \texttt{Edges}(p) \triangle M$;

**while** $P \neq \emptyset$;

输出 $(M)$;



# 匹配和最大匹配

$Q$ | $v_1$ $v_7$

---

**算法 11:** HKInit

---

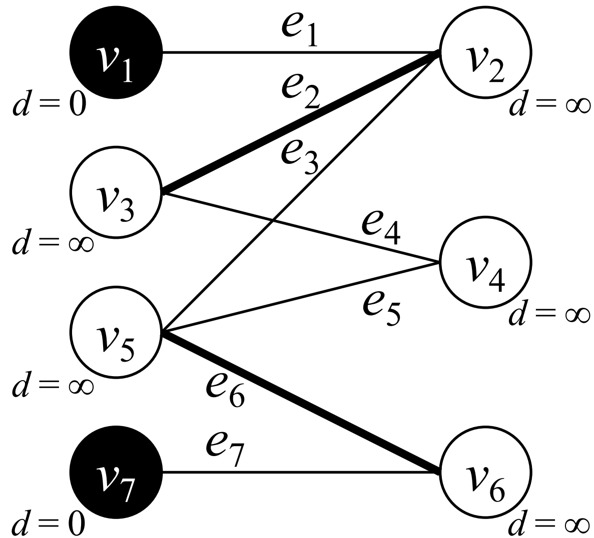
**输入:** 二分图 $G = \langle X \cup Y, E \rangle$

**初值:** $Q$ 初值为空队列

**foreach** $u \in (X \cup Y)$ **do**
- **if** $u \in X$ 且 $u$ 未被 $M$ 饱和 **then**
  - $u$.visited $\leftarrow$ true;
  - $u$.d $\leftarrow 0$;
  - 入队列 ($Q$, $u$);
- **else**
  - $u$.visited $\leftarrow$ false;
  - $u$.d $\leftarrow \infty$;

**return** $Q$;

---



# 匹配和最大匹配

$Q$　$v_1$　$v_7$

$Y' = \{\}$

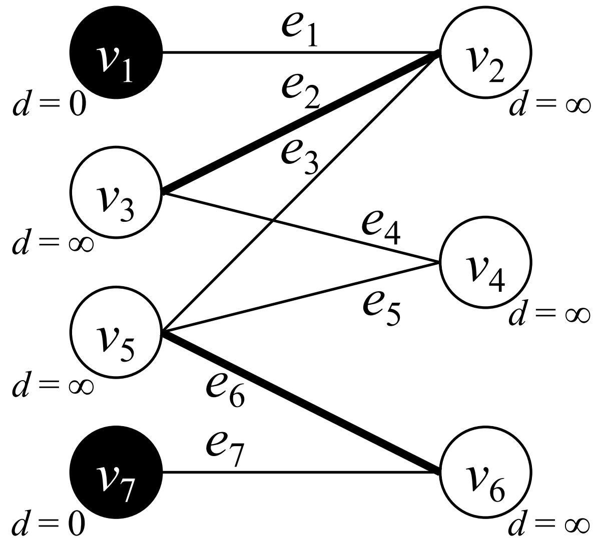
$d' = \infty$

**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
　$v \leftarrow$ 出队列（$Q$）;
　**if** $v.d > d'$ **then**
　　中止 while 循环;
　**else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
　　$Y' \leftarrow Y' \cup \{v\}$;
　　$d' \leftarrow v.\mathrm{d}$;
　**else**
　　**foreach** $(v, w) \in E$ **do**
　　　**if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
　　　　$w.\mathrm{visited} \leftarrow \mathrm{true}$;
　　　　$w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;
　　　　入队列（$Q$, $w$）;
**return** $Y'$;



# 匹配和最大匹配

$Q$ | $v_7$ $v_2$

$Y' = \{\}$

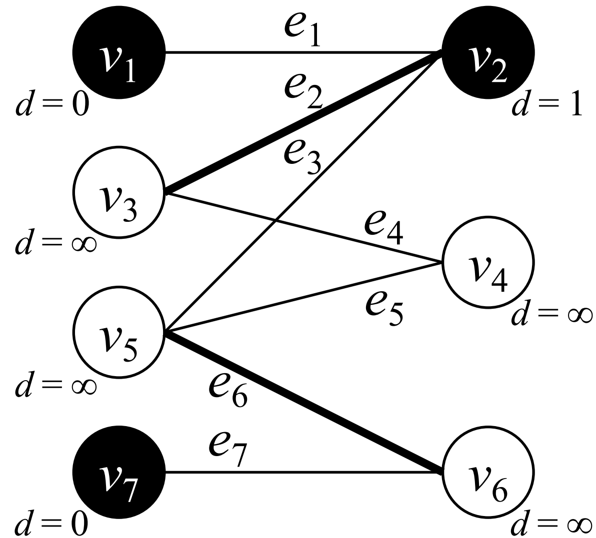
$d' = \infty$

**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
　$v \leftarrow$ 出队列（$Q$）;
　**if** $v.d > d'$ **then**
　　中止 while 循环;
　**else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
　　$Y' \leftarrow Y' \cup \{v\}$;
　　$d' \leftarrow v$.d;
　**else**
　　**foreach** $(v, w) \in E$ **do**
　　　**if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
　　　　$w$.visited $\leftarrow$ true;
　　　　$w$.d $\leftarrow v$.d + 1;
　　　　入队列（$Q$, $w$）;
**return** $Y'$;

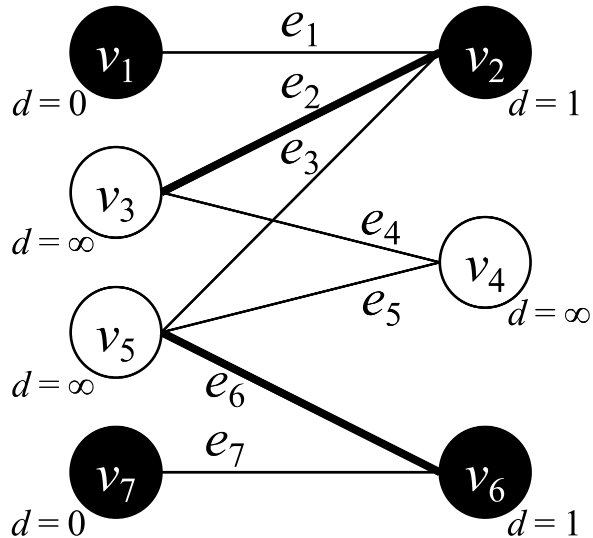


# 匹配和最大匹配

$$Q \quad v_2 \quad v_6$$

$$Y' = \{\}$$

$$d' = \infty$$




**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
  $v \leftarrow$ 出队列（$Q$）;
  **if** $v.d > d'$ **then**
    中止 while 循环;
  **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
    $Y' \leftarrow Y' \cup \{v\}$;
    $d' \leftarrow v.\mathrm{d}$;
  **else**
    **foreach** $(v, w) \in E$ **do**
      **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
        $w.\mathrm{visited} \leftarrow \mathrm{true}$;
        $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;
        入队列（$Q$, $w$）;
**return** $Y'$;

# 匹配和最大匹配

$Q$: $v_6$ $v_3$

$Y' = \{\}$

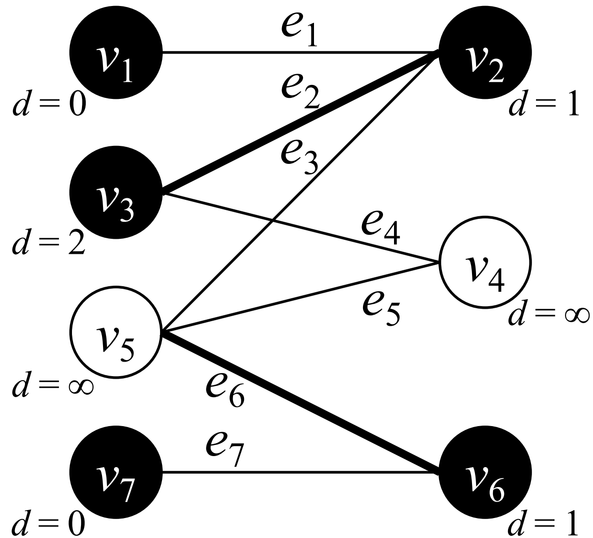
$d' = \infty$

**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

```
while Q 非空 do
    v ← 出队列 (Q);
    if v.d > d' then
        中止 while 循环;
    else if v 未被 M 饱和且 v.d > 0 then
        Y' ← Y' ∪ {v};
        d' ← v.d;
    else
        foreach (v, w) ∈ E do
            if w.visited = false 且 BFS 树中从根顶点到 w 的路是
             M-交错路 then
                w.visited ← true;
                w.d ← v.d + 1;
                入队列 (Q, w);
return Y';
```



# 匹配和最大匹配

| $Q$ | $v_3$ | $v_5$ |
|---|---|---|

$Y' = \{\}$

$d' = \infty$

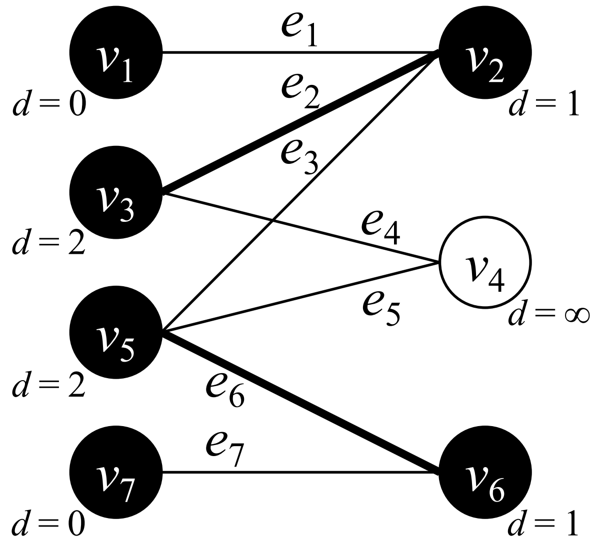
---

**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

```
while Q 非空 do
    v ← 出队列 (Q);
    if v.d > d' then
        中止 while 循环;
    else if v 未被 M 饱和且 v.d > 0 then
        Y' ← Y' ∪ {v};
        d' ← v.d;
    else
        foreach (v, w) ∈ E do
            if w.visited = false 且 BFS 树中从根顶点到 w 的路是
             M-交错路 then
                w.visited ← true;
                w.d ← v.d + 1;
                入队列 (Q, w);
return Y';
```



# 匹配和最大匹配

**算法 12:** HKBFS

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

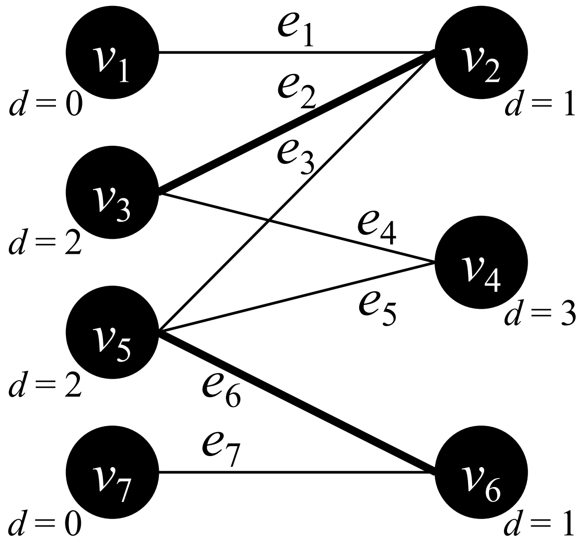
**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
    $v \leftarrow$ 出队列（$Q$）;
    **if** $v.d > d'$ **then**
        中止 while 循环;
    **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
        $Y' \leftarrow Y' \cup \{v\}$;
        $d' \leftarrow v.\mathrm{d}$;
    **else**
        **foreach** $(v, w) \in E$ **do**
            **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
                $w.\mathrm{visited} \leftarrow \mathrm{true}$;
                $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;
                入队列（$Q$, $w$）;
**return** $Y'$;

$Q$ | $v_5$ $v_4$

$Y' = \{\}$

$d' = \infty$



# 匹配和最大匹配

**算法 12:** HKBFS

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

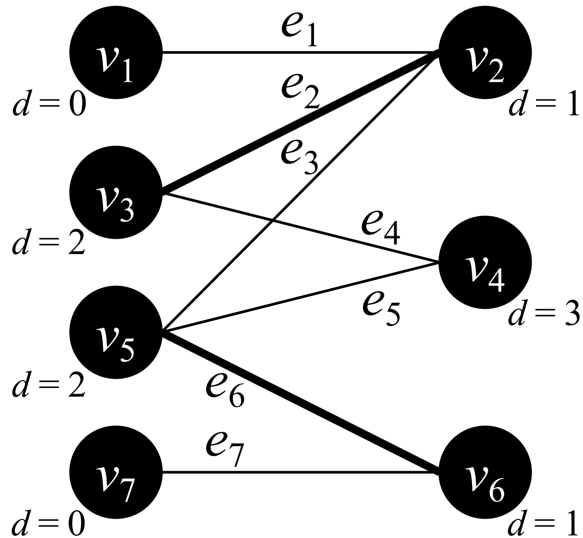
**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
    $v \leftarrow$ 出队列（$Q$）;
    **if** $v.d > d'$ **then**
        中止 while 循环;
    **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
        $Y' \leftarrow Y' \cup \{v\}$;
        $d' \leftarrow v.\mathrm{d}$;
    **else**
        **foreach** $(v, w) \in E$ **do**
            **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
                $w.\mathrm{visited} \leftarrow \mathrm{true}$;
                $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;
                入队列（$Q$, $w$）;
**return** $Y'$;

$Q$ | $v_4$

$Y' = \{\}$

$d' = \infty$

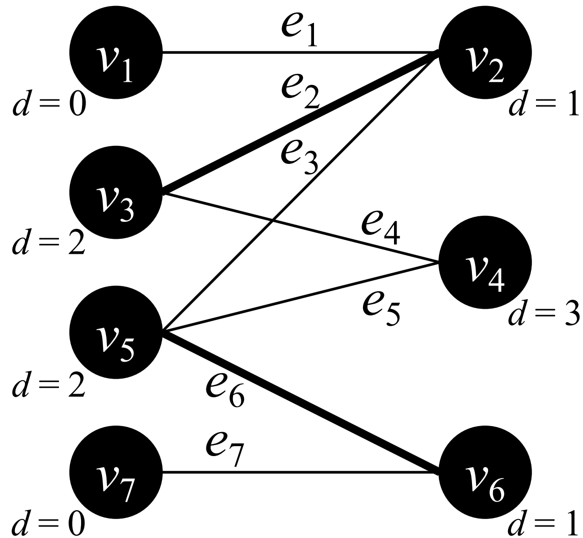


# 匹配和最大匹配

$Q$

$Y' = \{v_4\}$

$d' = 3$




---

**算法 12:** HKBFS

---

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
- $v \leftarrow$ 出队列（$Q$）;
- **if** $v.d > d'$ **then**
  - 中止 while 循环;
- **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
  - $Y' \leftarrow Y' \cup \{v\}$;
  - $d' \leftarrow v.\mathrm{d}$;
- **else**
  - **foreach** $(v, w) \in E$ **do**
    - **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
      - $w.\mathrm{visited} \leftarrow \mathrm{true}$;
      - $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;
      - 入队列（$Q$, $w$）;

**return** $Y'$;

# 匹配和最大匹配

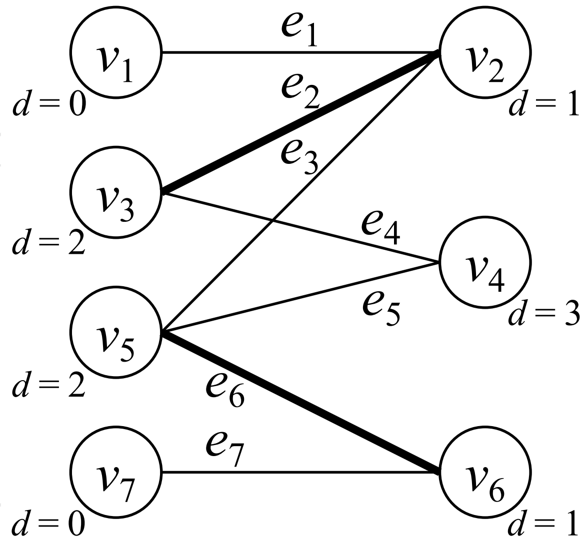
$$Y' = \{v_4\}$$

**算法 10:** 霍普克罗夫特-卡普算法

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

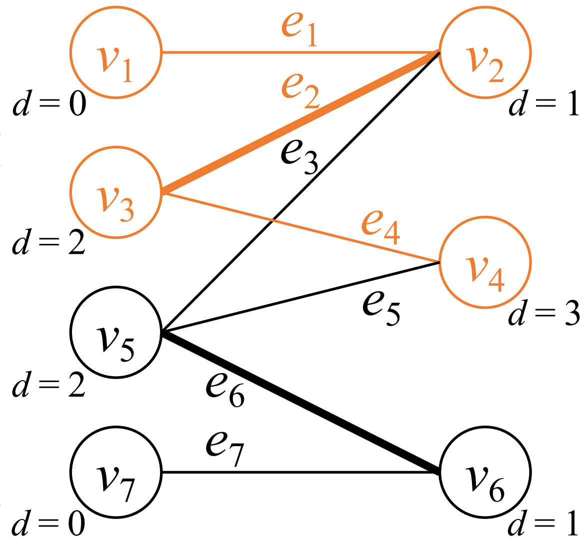
**初值:** $M$ 初值为 $\emptyset$

```
do
    Q ← HKInit(G);
    Y′ ← HKBFS(G, Q);
    P ← HKPaths(G, Y′);
    foreach p ∈ P do
        M ← Edges(p) △ M;
while P ≠ ∅;
输出 (M);
```



# 匹配和最大匹配

$$Y' = \{v_4\}$$




**算法 10:** 霍普克罗夫特-卡普算法

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

**初值:** $M$ 初值为 $\emptyset$

**do**

  $Q \leftarrow$ `HKInit`$(G)$;

  $Y' \leftarrow$ `HKBFS`$(G, Q)$;

  $P \leftarrow$ `HKPaths`$(G, Y')$;

  **foreach** $p \in P$ **do**

    $M \leftarrow$ `Edges`$(p) \triangle M$;

**while** $P \neq \emptyset$;

输出 $(M)$;

# 匹配和最大匹配

**算法 10:** 霍普克罗夫特-卡普算法

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

**初值:** $M$ 初值为 $\emptyset$

**do**

  $Q \leftarrow$ `HKInit`$(G)$;

  $Y' \leftarrow$ `HKBFS`$(G,\ Q)$;

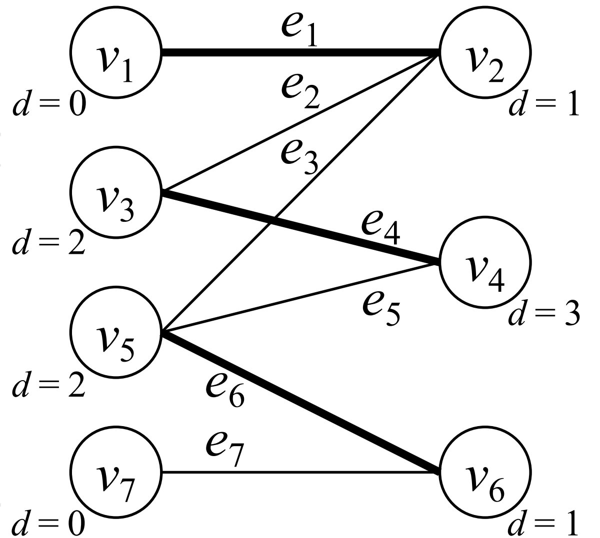
  $P \leftarrow$ `HKPaths`$(G,\ Y')$;

  **foreach** $p \in P$ **do**

    $M \leftarrow$ `Edges`$(p) \triangle M$;

**while** $P \neq \emptyset$;

输出 $(M)$;



# 匹配和最大匹配

- 第3轮do-while循环开始

**算法 10:** 霍普克罗夫特-卡普算法

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

**初值:** $M$ 初值为 $\emptyset$

**do**

  $Q \leftarrow$ `HKInit`$(G)$;

  $Y' \leftarrow$ `HKBFS`$(G,\ Q)$;

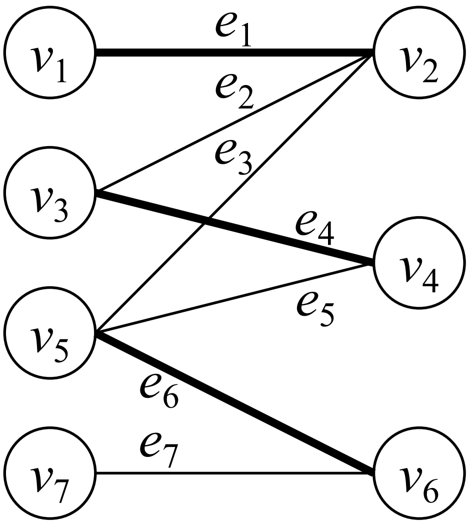
  $P \leftarrow$ `HKPaths`$(G,\ Y')$;

  **foreach** $p \in P$ **do**

    $M \leftarrow$ `Edges`$(p) \triangle M$;

**while** $P \neq \emptyset$;

输出 $(M)$;



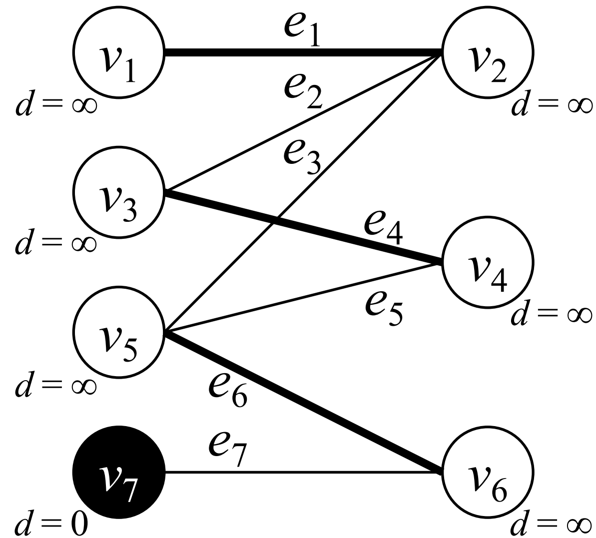
# 匹配和最大匹配


**算法 11:** HKInit

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

**初值:** $Q$ 初值为空队列

**foreach** $u \in (X \cup Y)$ **do**
　　**if** $u \in X$ 且 $u$ 未被 $M$ 饱和 **then**
　　　　$u$.visited $\leftarrow$ true;
　　　　$u$.d $\leftarrow$ 0;
　　　　入队列（$Q$, $u$);
　　**else**
　　　　$u$.visited $\leftarrow$ false;
　　　　$u$.d $\leftarrow \infty$;
**return** $Q$;



# 匹配和最大匹配

**算法 12:** HKBFS

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

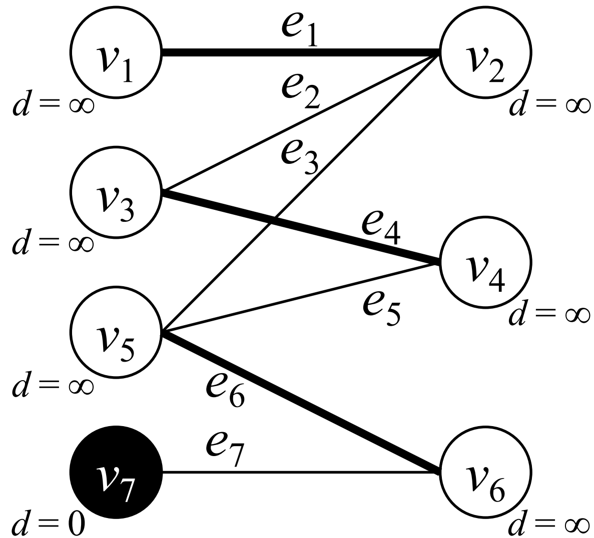
**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

```
while Q 非空 do
    v ← 出队列 (Q);
    if v.d > d' then
        中止 while 循环;
    else if v 未被 M 饱和且 v.d > 0 then
        Y' ← Y' ∪ {v};
        d' ← v.d;
    else
        foreach (v, w) ∈ E do
            if w.visited = false 且 BFS 树中从根顶点到 w 的路是
             M-交错路 then
                w.visited ← true;
                w.d ← v.d + 1;
                入队列 (Q, w);
return Y';
```

$Q$ | $v_7$

$Y' = \{\}$

$d' = \infty$



# 匹配和最大匹配

Q: $v_6$

$Y' = \{\}$

$d' = \infty$

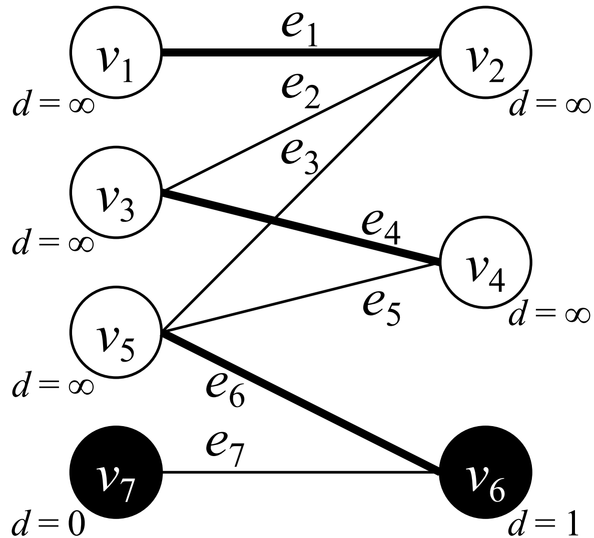
**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
- $v \leftarrow$ 出队列（$Q$）;
- **if** $v.d > d'$ **then**
  - 中止 while 循环;
- **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
  - $Y' \leftarrow Y' \cup \{v\}$;
  - $d' \leftarrow v.\mathrm{d}$;
- **else**
  - **foreach** $(v, w) \in E$ **do**
    - **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
      - $w.\mathrm{visited} \leftarrow \mathrm{true}$;
      - $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;
      - 入队列（$Q$, $w$）;

**return** $Y'$;



# 匹配和最大匹配

**算法 12:** HKBFS

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

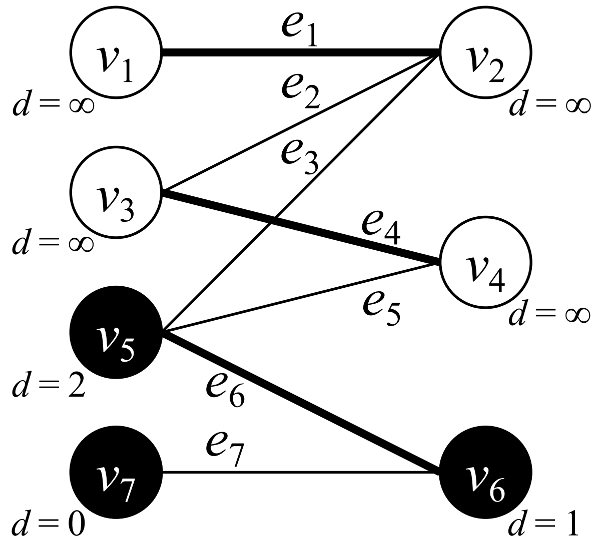
**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

```
while Q 非空 do
    v ← 出队列 (Q);
    if v.d > d' then
        中止 while 循环;
    else if v 未被 M 饱和且 v.d > 0 then
        Y' ← Y' ∪ {v};
        d' ← v.d;
    else
        foreach (v, w) ∈ E do
            if w.visited = false 且 BFS 树中从根顶点到 w 的路是
             M-交错路 then
                w.visited ← true;
                w.d ← v.d + 1;
                入队列 (Q, w);
return Y';
```

$Q$ $v_5$

$Y' = \{\}$

$d' = \infty$



# 匹配和最大匹配

**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

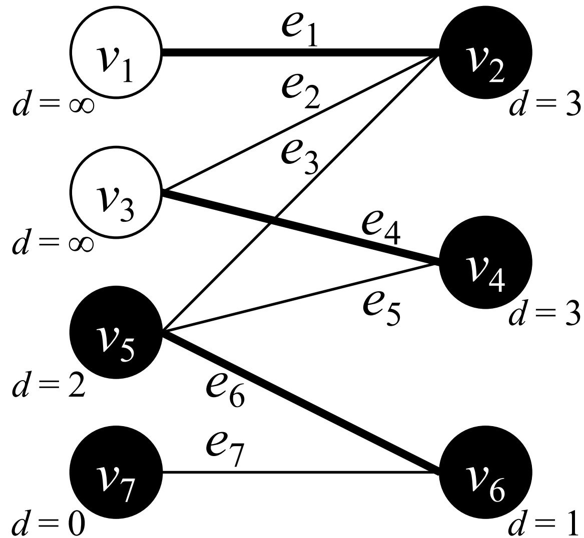
**while** $Q$ 非空 **do**
- $v \leftarrow$ 出队列（$Q$）;
- **if** $v.d > d'$ **then**
  - 中止 while 循环;
- **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
  - $Y' \leftarrow Y' \cup \{v\}$;
  - $d' \leftarrow v.\mathrm{d}$;
- **else**
  - **foreach** $(v, w) \in E$ **do**
    - **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
      - $w.\mathrm{visited} \leftarrow \mathrm{true}$;
      - $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;
      - 入队列（$Q$, $w$）;

**return** $Y'$;

$Q$: $v_2$ $v_4$

$Y' = \{\}$

$d' = \infty$



# 匹配和最大匹配

$Q$ $v_4$ $v_1$

$Y' = \{\}$

$d' = \infty$

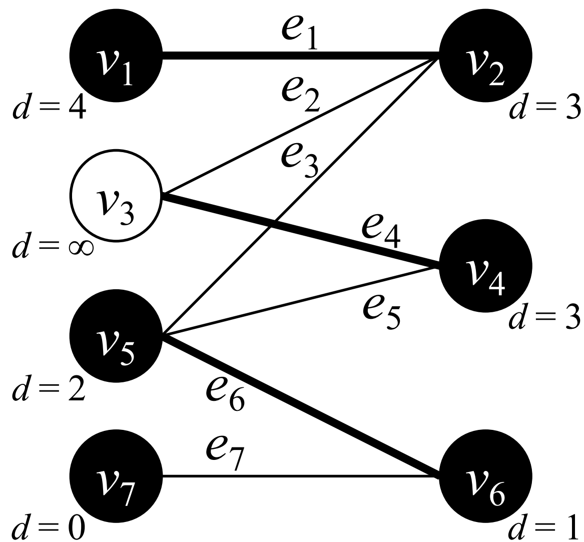
---

**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
    $v \leftarrow$ 出队列（$Q$）;
    **if** $v.d > d'$ **then**
        中止 while 循环;
    **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
        $Y' \leftarrow Y' \cup \{v\}$;
        $d' \leftarrow v.\mathrm{d}$;
    **else**
        **foreach** $(v, w) \in E$ **do**
            **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
                $w.\mathrm{visited} \leftarrow \mathrm{true}$;
                $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;
                入队列（$Q$, $w$）;
**return** $Y'$;



# 匹配和最大匹配

**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

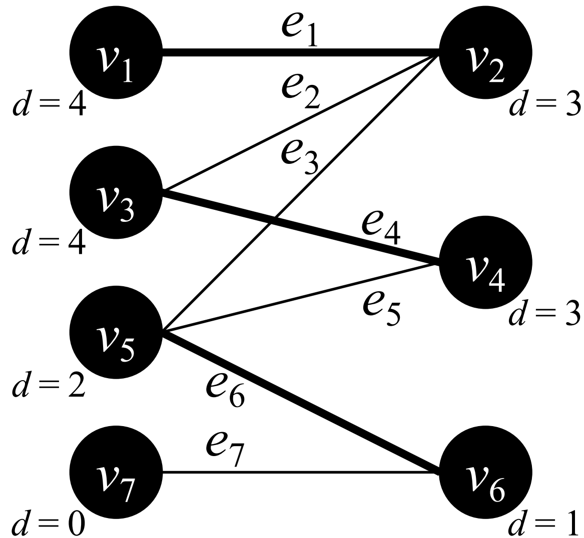
**while** $Q$ 非空 **do**
- $v \leftarrow$ 出队列（$Q$）;
- **if** $v.d > d'$ **then**
  - 中止 while 循环;
- **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
  - $Y' \leftarrow Y' \cup \{v\}$;
  - $d' \leftarrow v$.d;
- **else**
  - **foreach** $(v, w) \in E$ **do**
    - **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
      - $w$.visited $\leftarrow$ true;
      - $w$.d $\leftarrow v$.d + 1;
      - 入队列（$Q$, $w$）;

**return** $Y'$;

$Q$ | $v_1$ $v_3$

$Y' = \{\}$

$d' = \infty$



# 匹配和最大匹配

$Q$ $v_3$

$Y' = \{\}$

$d' = \infty$

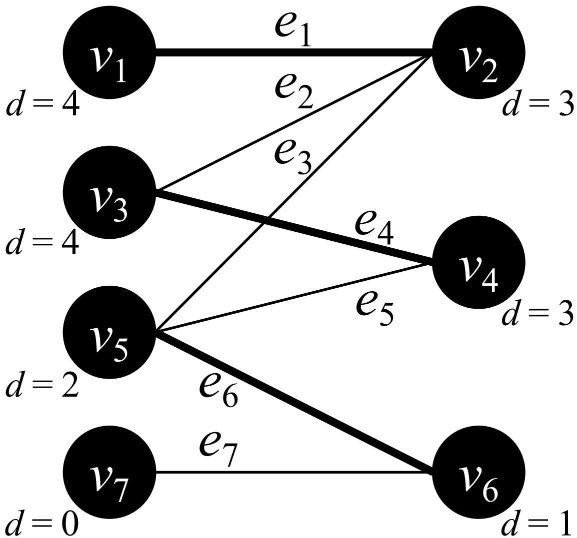
**算法 12:** HKBFS

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

**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
- $v \leftarrow$ 出队列（$Q$）;
- **if** $v.d > d'$ **then**
  - 中止 while 循环;
- **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
  - $Y' \leftarrow Y' \cup \{v\}$;
  - $d' \leftarrow v.\mathrm{d}$;
- **else**
  - **foreach** $(v, w) \in E$ **do**
    - **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
      - $w.\mathrm{visited} \leftarrow \mathrm{true}$;
      - $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;
      - 入队列（$Q$，$w$）;

**return** $Y'$;



# 匹配和最大匹配

$Q$

$Y' = \{\}$

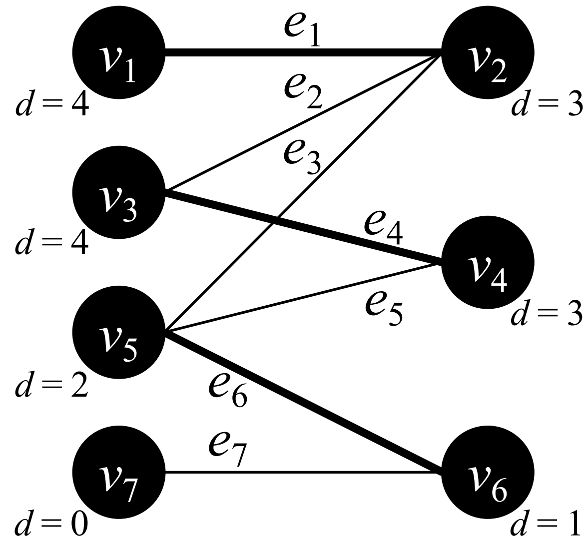
$d' = \infty$

**算法 12:** HKBFS

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

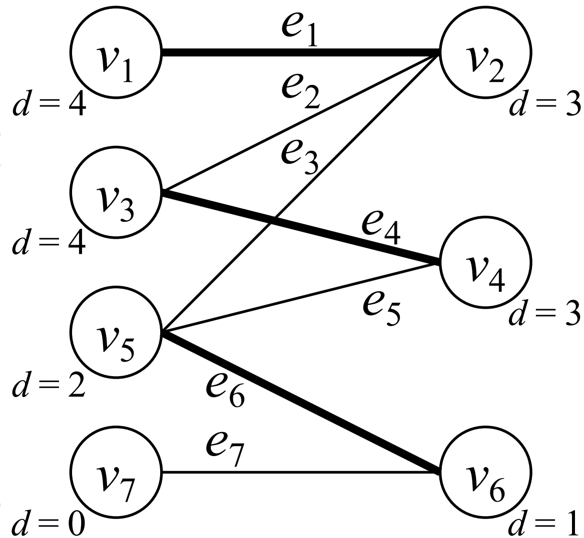
**初值:** $Y'$ 初值为 $\emptyset$；$d'$ 初值为 $\infty$

**while** $Q$ 非空 **do**
    $v \leftarrow$ 出队列（$Q$）;
    **if** $v.d > d'$ **then**
        中止 while 循环;
    **else if** $v$ 未被 $M$ 饱和且 $v.d > 0$ **then**
        $Y' \leftarrow Y' \cup \{v\}$;
        $d' \leftarrow v.\mathrm{d}$;
    **else**
        **foreach** $(v, w) \in E$ **do**
            **if** $w.visited = false$ 且 $BFS$ 树中从根顶点到 $w$ 的路是 $M$-交错路 **then**
                $w.\mathrm{visited} \leftarrow \mathrm{true}$;
                $w.\mathrm{d} \leftarrow v.\mathrm{d} + 1$;
                入队列（$Q$, $w$）;
**return** $Y'$;



# 匹配和最大匹配

$$Y' = \{\}$$




**算法 10:** 霍普克罗夫特-卡普算法

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

**初值:** $M$ 初值为 $\emptyset$

```
do
    Q ← HKInit(G);
    Y′ ← HKBFS(G, Q);
    P ← HKPaths(G, Y′);
    foreach p ∈ P do
        M ← Edges(p) △ M;
while P ≠ ∅;
输出 (M);
```

# 匹配和最大匹配

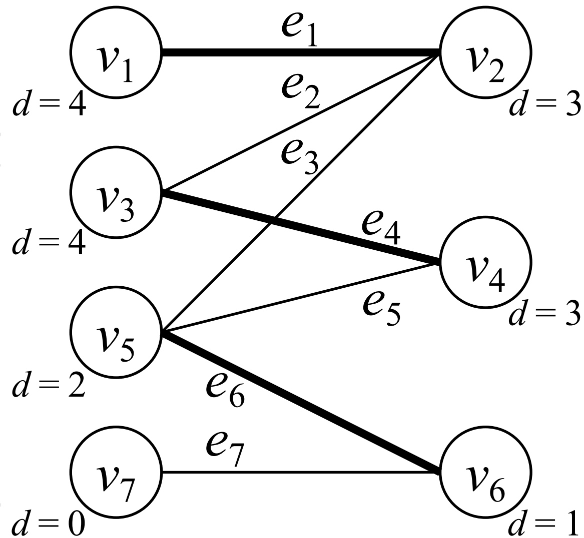
$Y' = \{\}$

**算法 10:** 霍普克罗夫特-卡普算法

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

**初值:** $M$ 初值为 $\emptyset$

```
do
    Q ← HKInit(G);
    Y′ ← HKBFS(G, Q);
    P ← HKPaths(G, Y′);
    foreach p ∈ P do
        M ← Edges(p) △ M;
while P ≠ ∅;
输出 (M);
```



# 匹配和最大匹配

- 时间复杂度 $O(\sqrt{n}(n+m))$
  - 随着do-while循环轮数的增加，$d'$严格单调增。
  - 对于阶为$n$的图，$d'$有至多 $2\lfloor\sqrt{\frac{n}{2}}\rfloor+2$种取值。

**算法 10:** 霍普克罗夫特-卡普算法

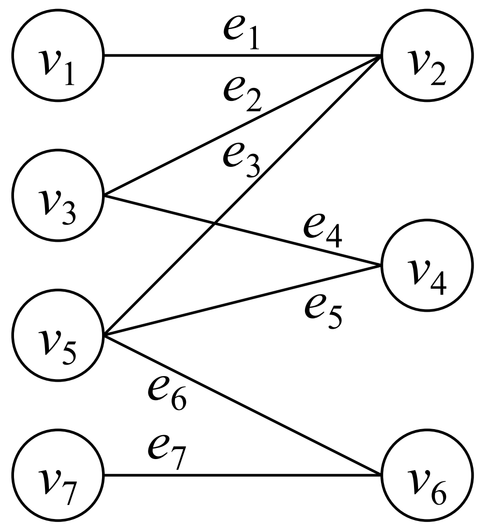
**输入:** 二分图 $G=\langle X\cup Y, E\rangle$

**初值:** $M$ 初值为 $\emptyset$

**do**
- $Q \leftarrow$ `HKInit`$(G)$;
- $Y' \leftarrow$ `HKBFS`$(G, Q)$;
- $P \leftarrow$ `HKPaths`$(G, Y')$;
- **foreach** $p \in P$ **do**
  - $M \leftarrow$ `Edges`$(p) \triangle M$;

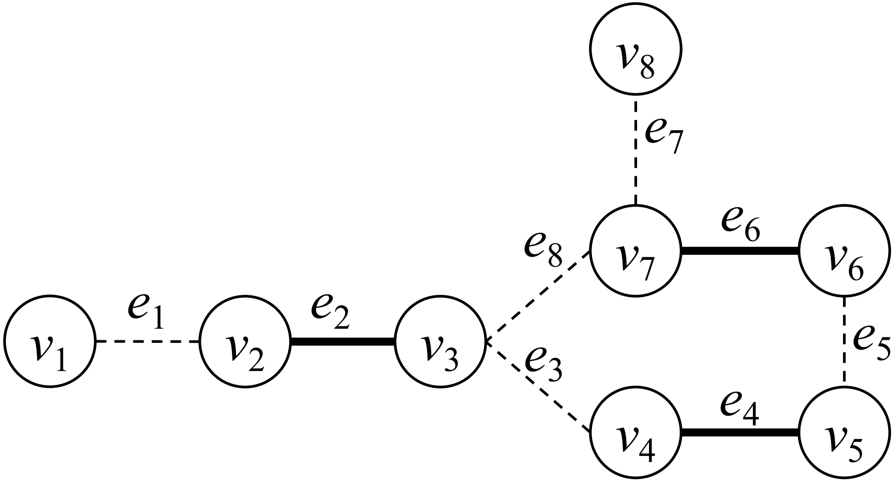
**while** $P \neq \emptyset$;

输出 $(M)$;



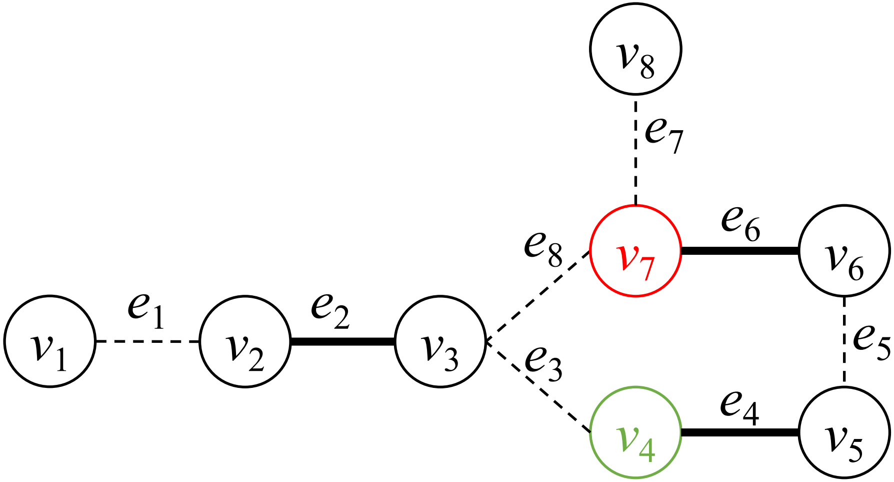
# 匹配和最大匹配

- 为什么匈牙利算法和霍普克罗夫特-卡普算法只适用于二分图？



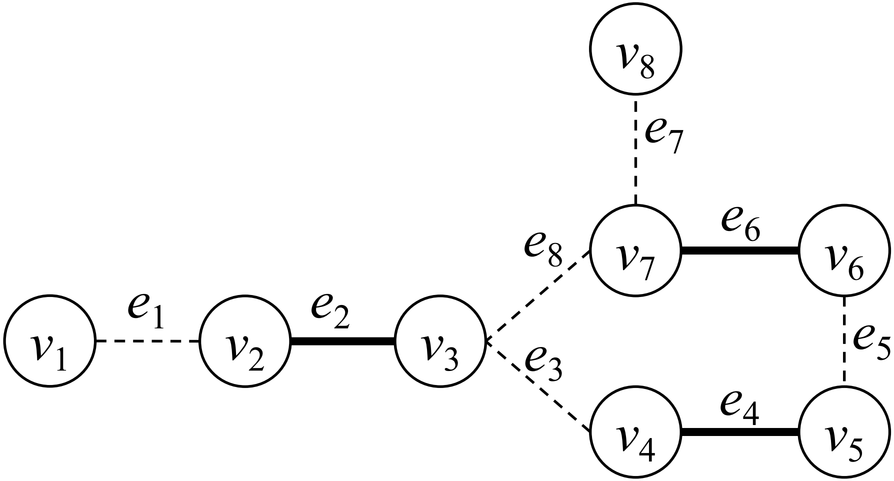
# 匹配和最大匹配

- 为什么匈牙利算法和霍普克罗夫特-卡普算法只适用于二分图？
  - 误认为找不到增广路



# 匹配和最大匹配

- 为什么匈牙利算法和霍普克罗夫特-卡普算法只适用于二分图？
  - 误认为找不到增广路
- 为什么二分图不存在上述问题？



# 匹配和最大匹配

- 二分图
  - 匈牙利算法
  - 霍普克罗夫特-卡普算法
- 非二分图
  - 花算法：收缩奇圈

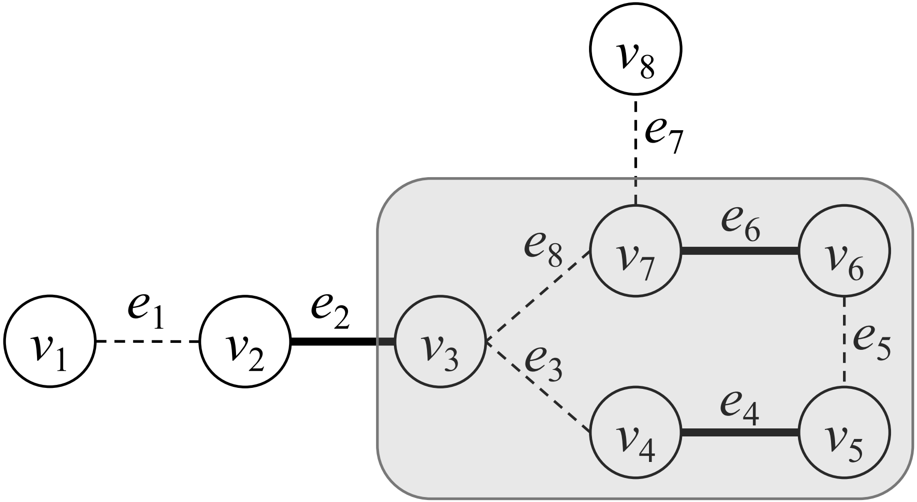
# 匹配和最大匹配

- Jack Edmonds，1934-，出生于美国


https://en.wikipedia.org/wiki/Jack_Edmonds

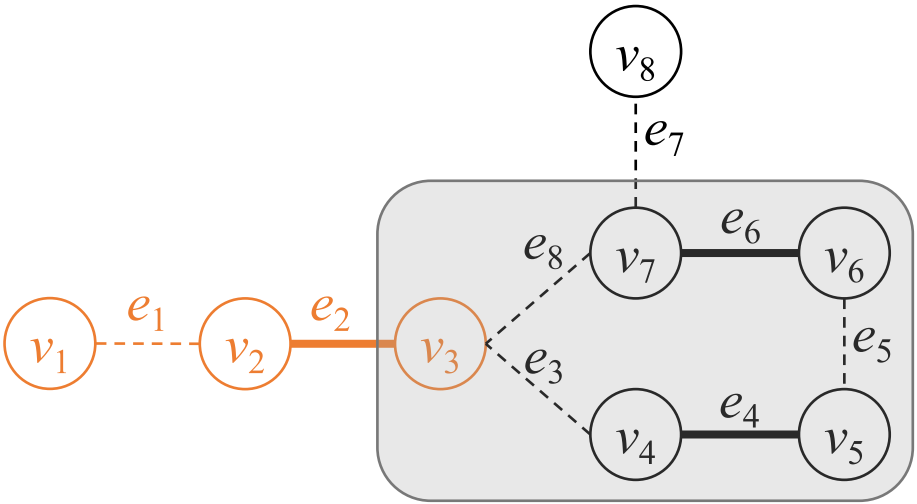
# 匹配和最大匹配

- **花**：两条交错路形成的奇圈



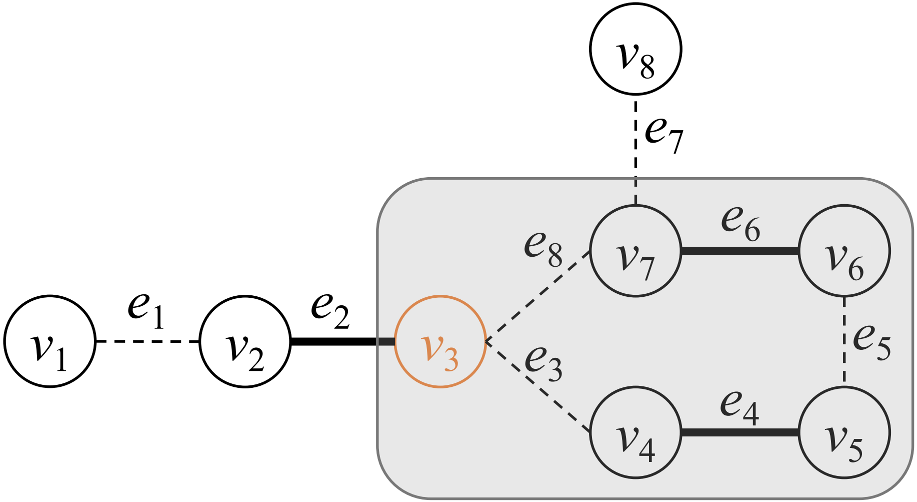
# 匹配和最大匹配

- 花：两条交错路形成的奇圈
- **花梗**：两条交错路的公共子路



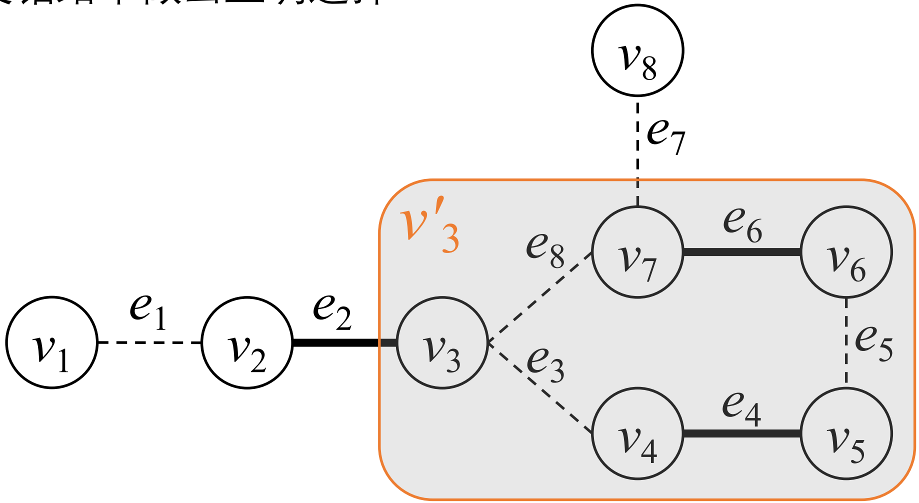
# 匹配和最大匹配

- 花：两条交错路形成的奇圈
- 花梗：两条交错路的公共子路
- **花托**：花梗的终点



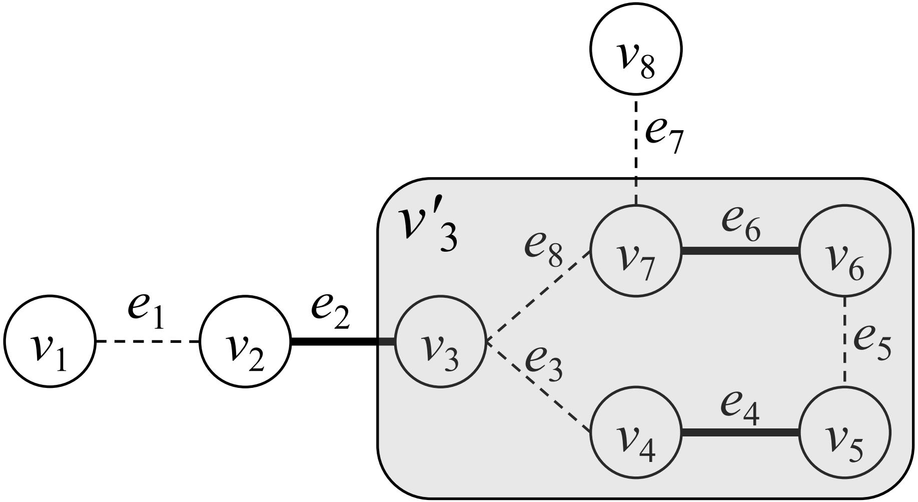
# 匹配和最大匹配

- 经过花梗从花托进入花时，花算法先不做任何选择，而是将花中的所有顶点收缩为一个新的顶点
- 在收缩后的新图上，继续
- 得到增广路，再将其经过的新顶点还原为花，并从花中的两条交错路中做出正确选择



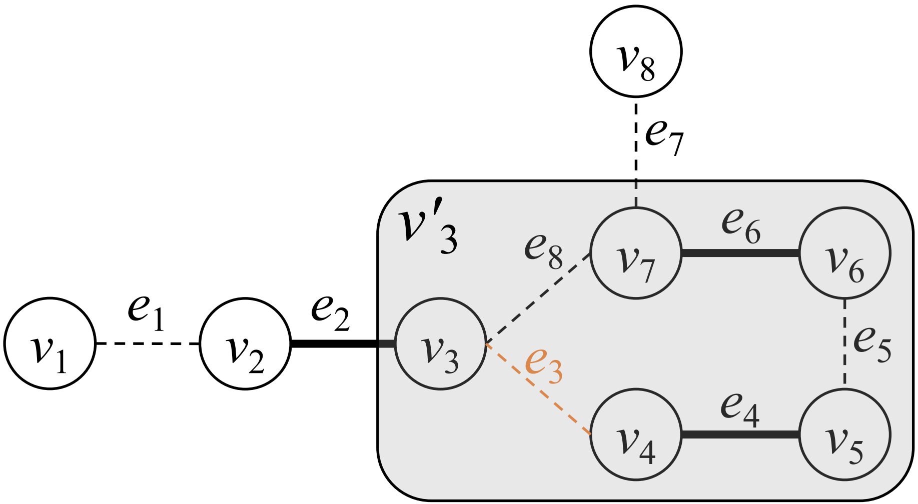
# 匹配和最大匹配

- 在花算法中，如何识别花？



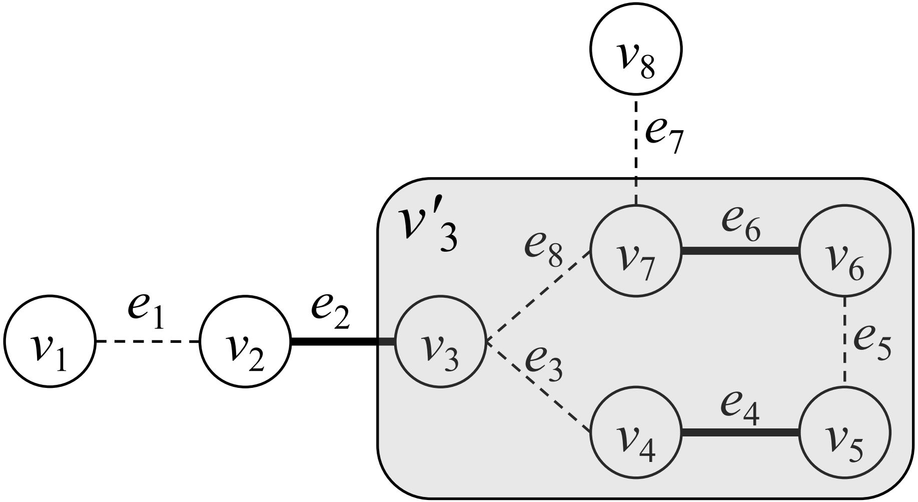
# 匹配和最大匹配

- 在花算法中，如何识别花？



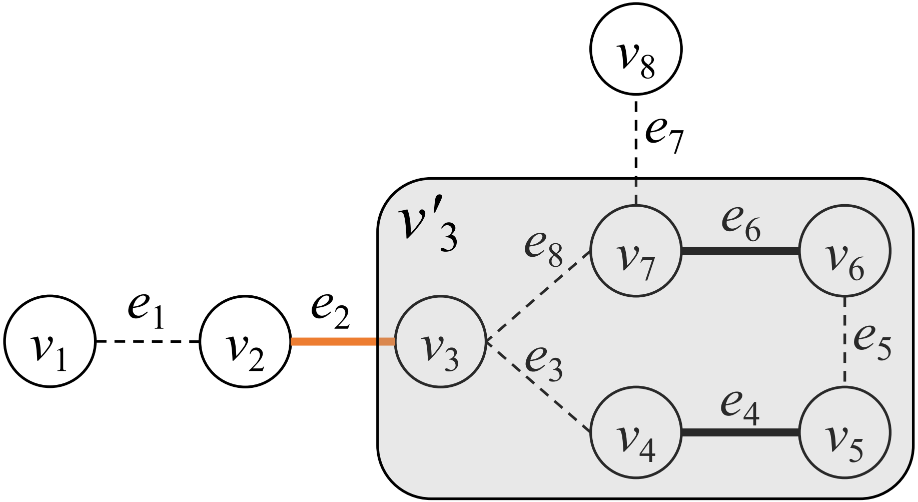
# 匹配和最大匹配

- 在花算法中，如何识别花？
- 在花算法运行过程中，所有花都会被收缩吗？



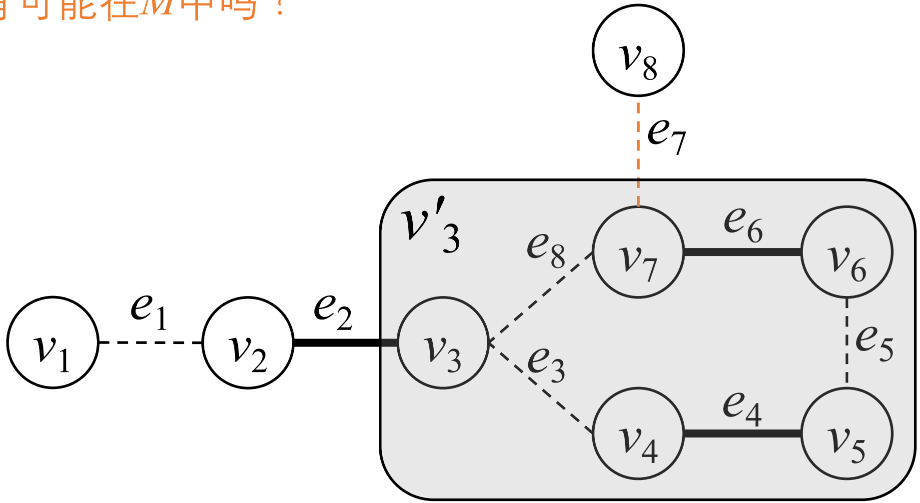
# 匹配和最大匹配

- 在花算法中，如何识别花？
- 在花算法运行过程中，所有花都会被收缩吗？
- 花梗的最后一条边有可能不在$M$中吗？

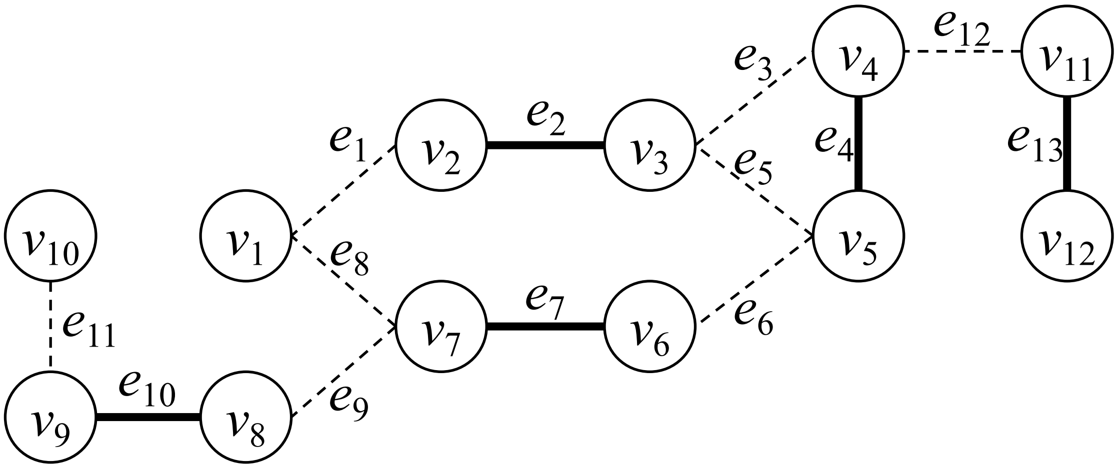


# 匹配和最大匹配

- 在花算法中，如何识别花？
- 在花算法运行过程中，所有花都会被收缩吗？
- 花梗的最后一条边有可能不在$M$中吗？
- 花中顶点与花外顶点间的边（除花梗的最后一条边以外），有可能在$M$中吗？

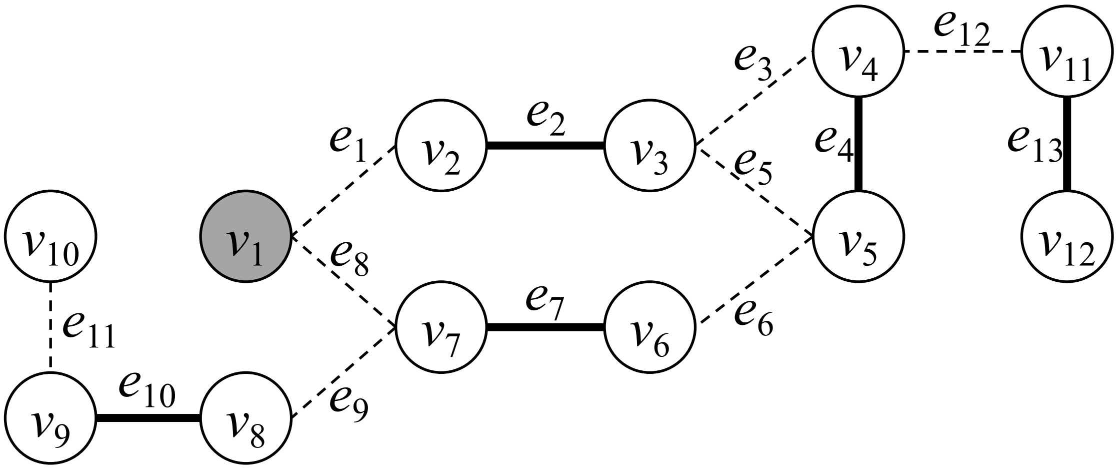


# 匹配和最大匹配



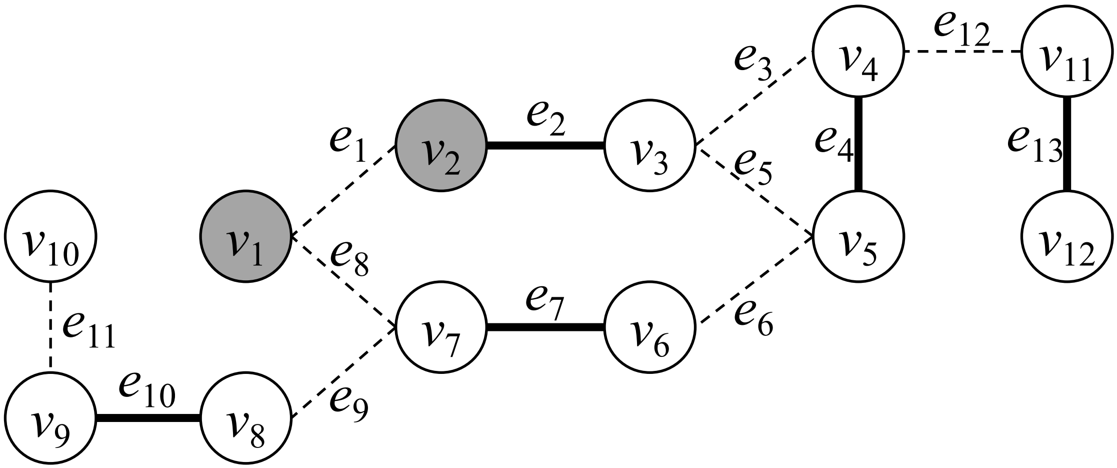
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$

# 匹配和最大匹配



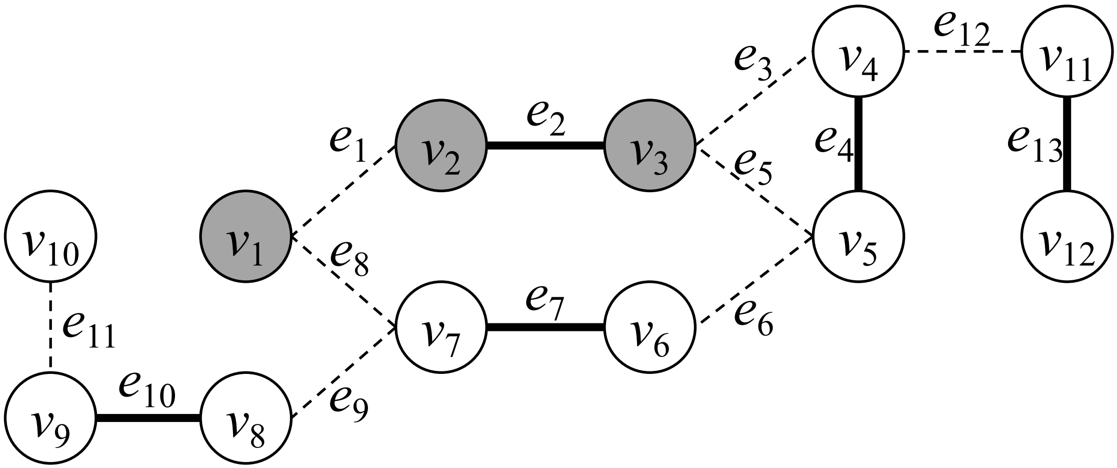
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$

# 匹配和最大匹配

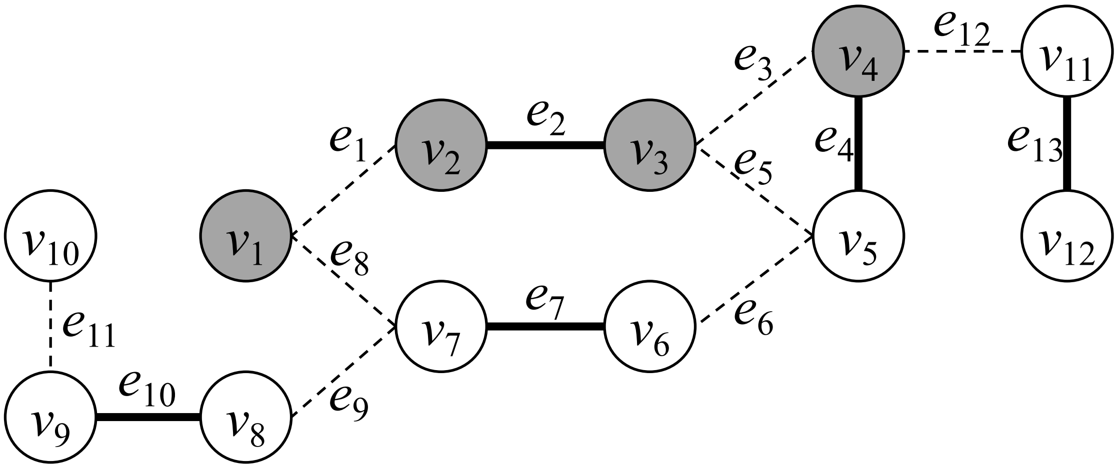


$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$

# 匹配和最大匹配

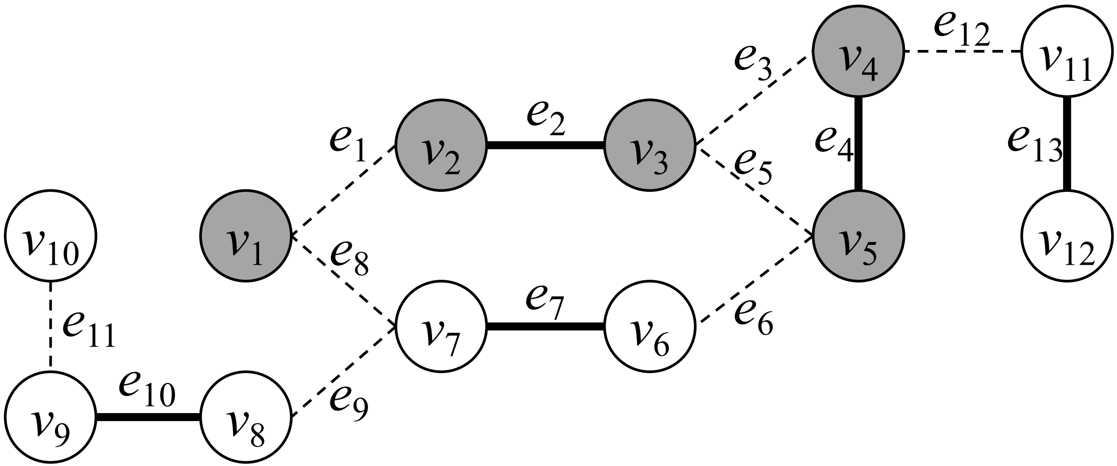

$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$

# 匹配和最大匹配

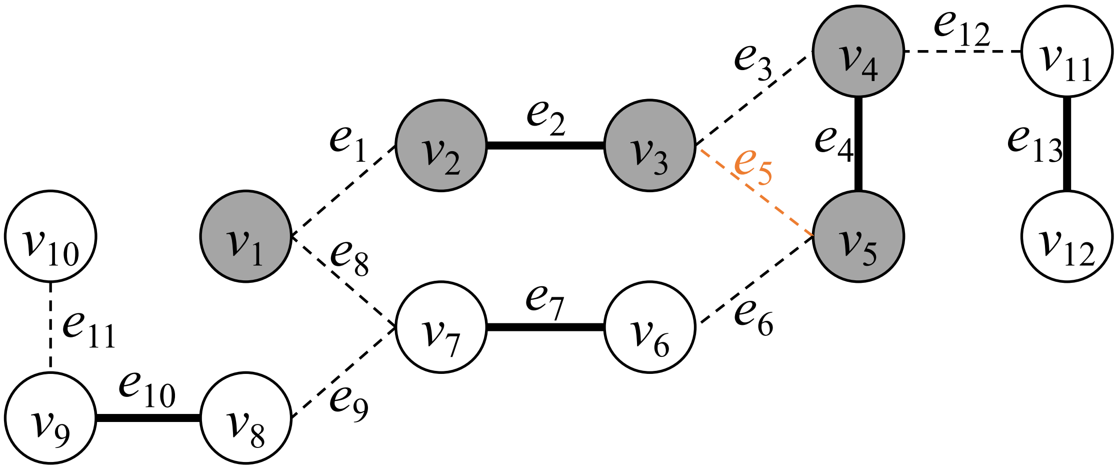


$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$

# 匹配和最大匹配

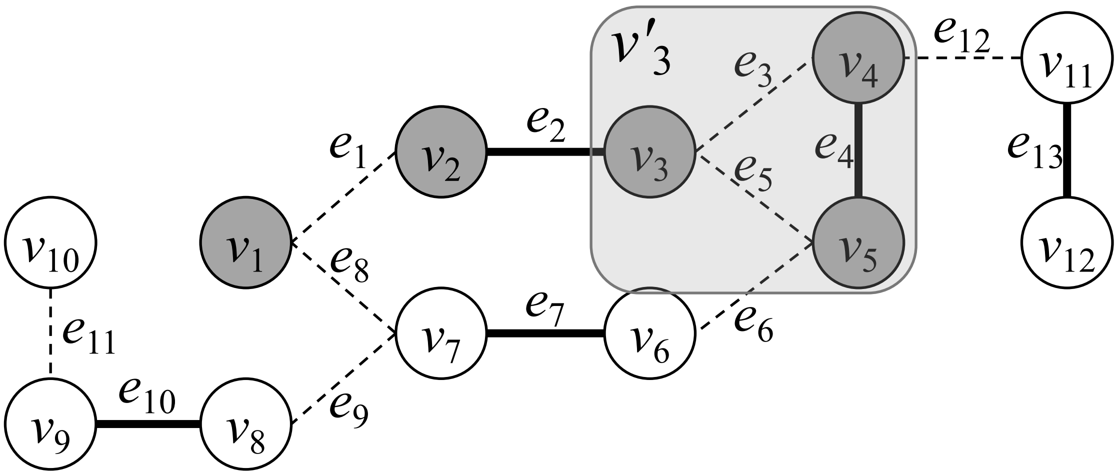

$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$

# 匹配和最大匹配

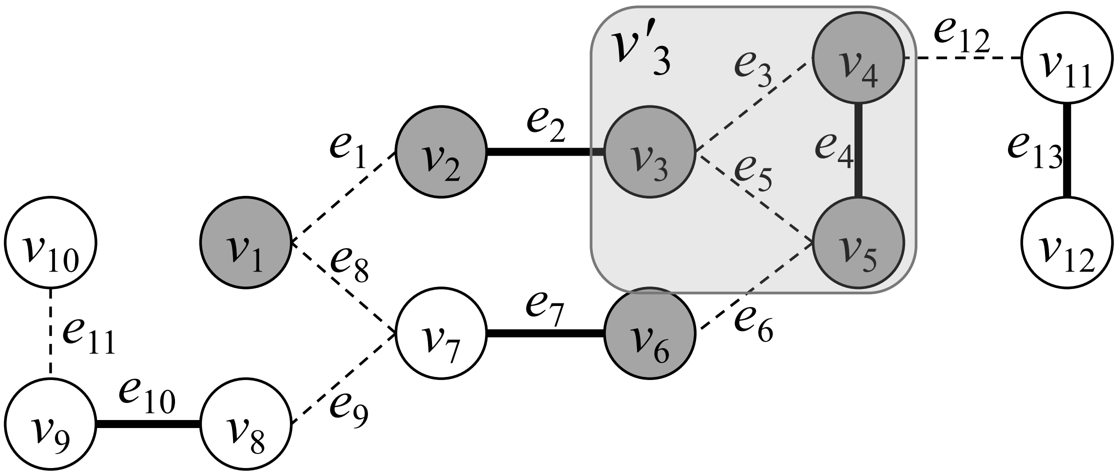


$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$

# 匹配和最大匹配

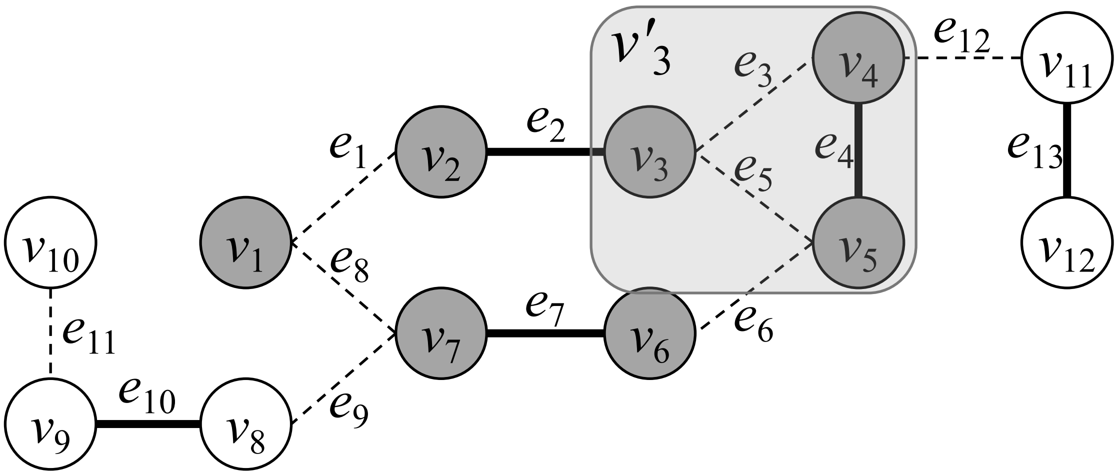

$v'_3$
$e_{12}$
$v_4$
$v_{11}$
$e_3$
$e_2$
$e_1$
$v_2$
$v_3$
$e_4$
$e_{13}$
$e_5$
$v_{10}$
$v_1$
$e_8$
$v_5$
$v_{12}$
$e_7$
$e_{11}$
$v_7$
$v_6$
$e_6$
$e_{10}$
$v_9$
$v_8$
$e_9$

# 匹配和最大匹配



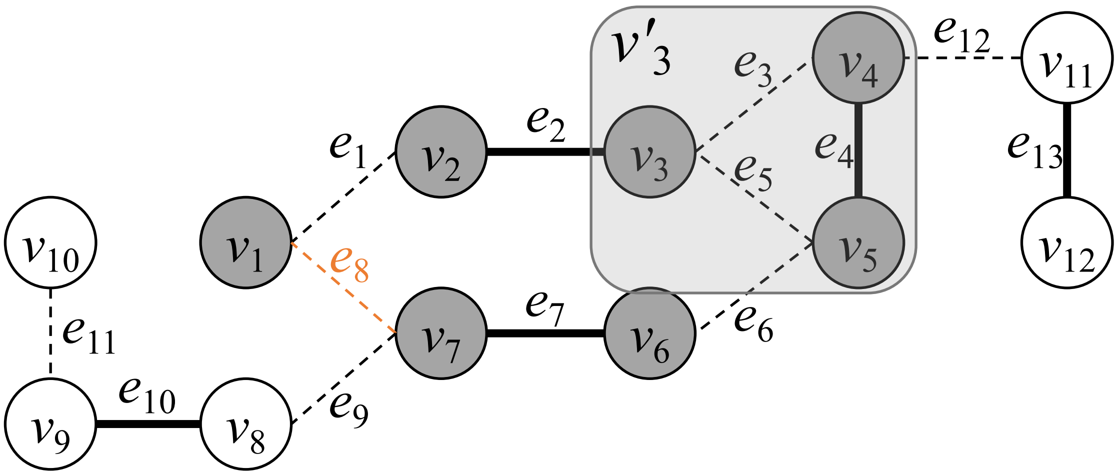
$v'_3$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$

# 匹配和最大匹配



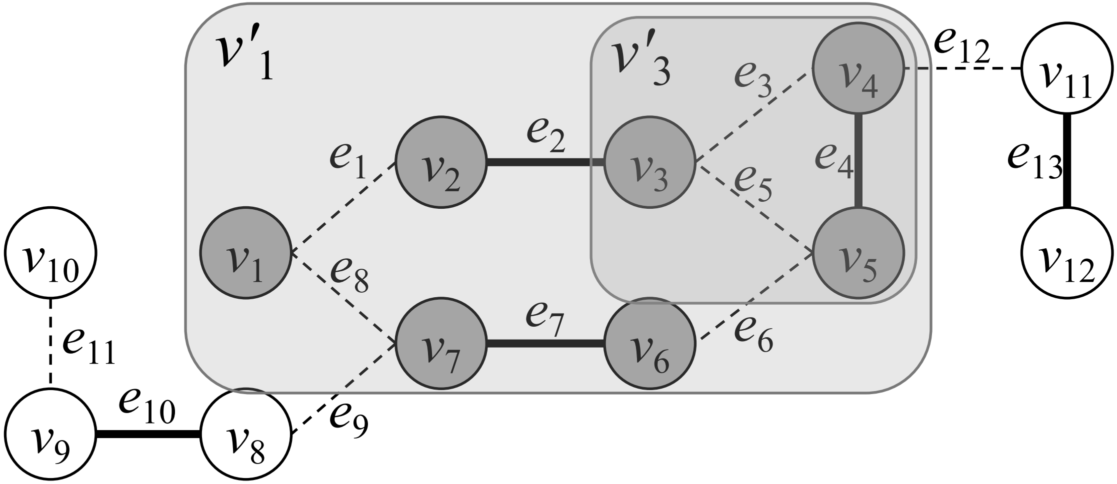
$v'_3$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$

# 匹配和最大匹配



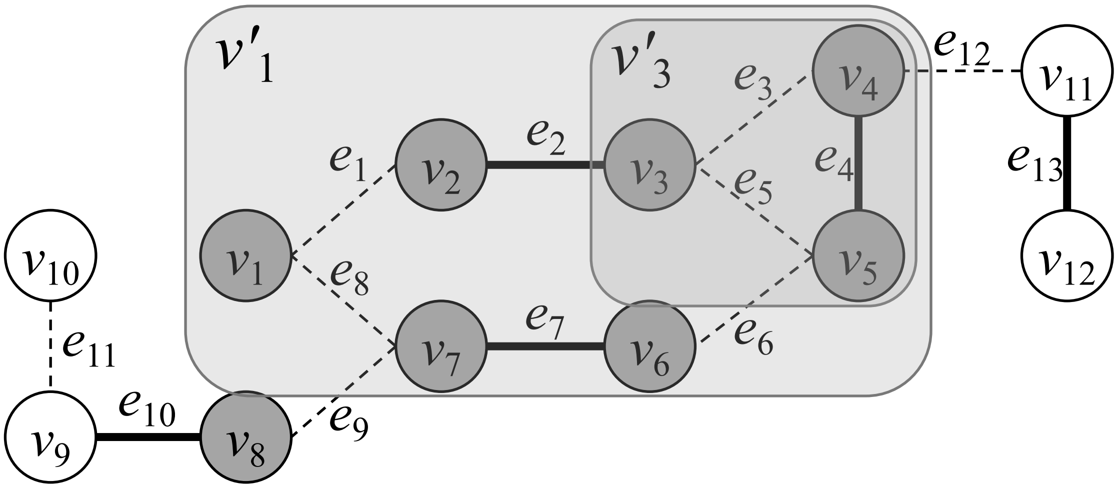
$v'_3$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$

# 匹配和最大匹配



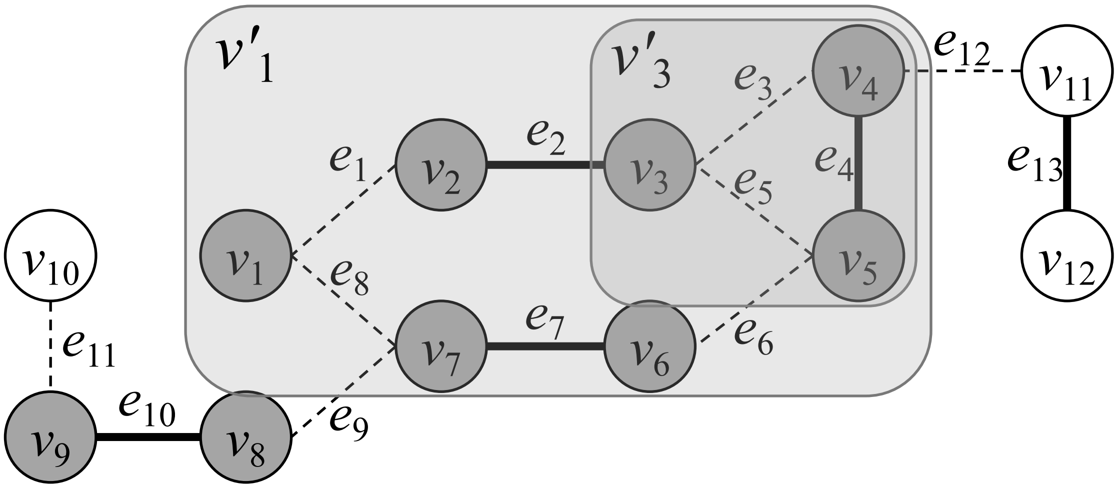
$v'_1$
$v'_3$
$e_{12}$
$e_3$
$v_4$
$v_{11}$
$e_2$
$e_1$
$v_2$
$v_3$
$e_4$
$e_{13}$
$e_5$
$v_{10}$
$v_1$
$e_8$
$v_5$
$v_{12}$
$e_7$
$e_{11}$
$v_7$
$v_6$
$e_6$
$e_{10}$
$v_9$
$v_8$
$e_9$

# 匹配和最大匹配



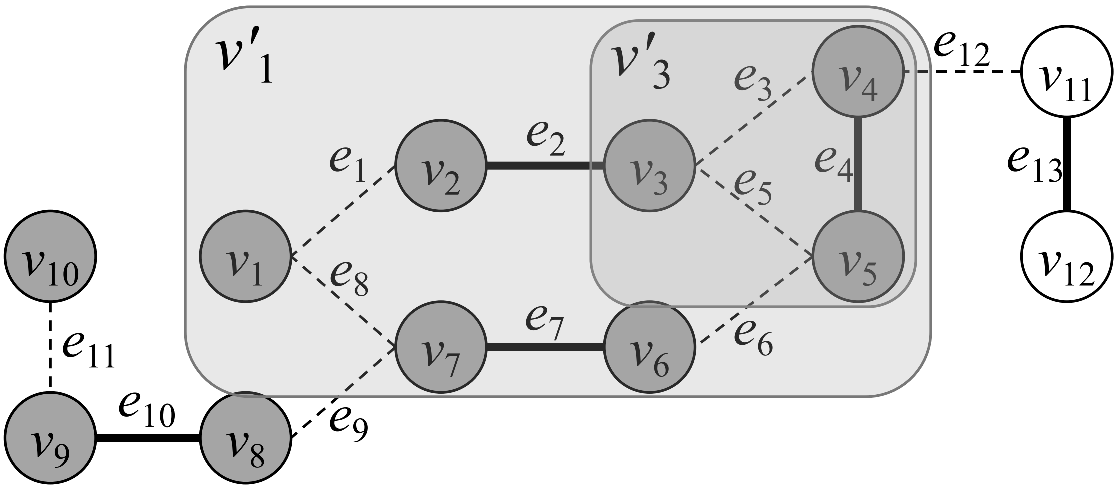
$v'_1$
$v'_3$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$

# 匹配和最大匹配



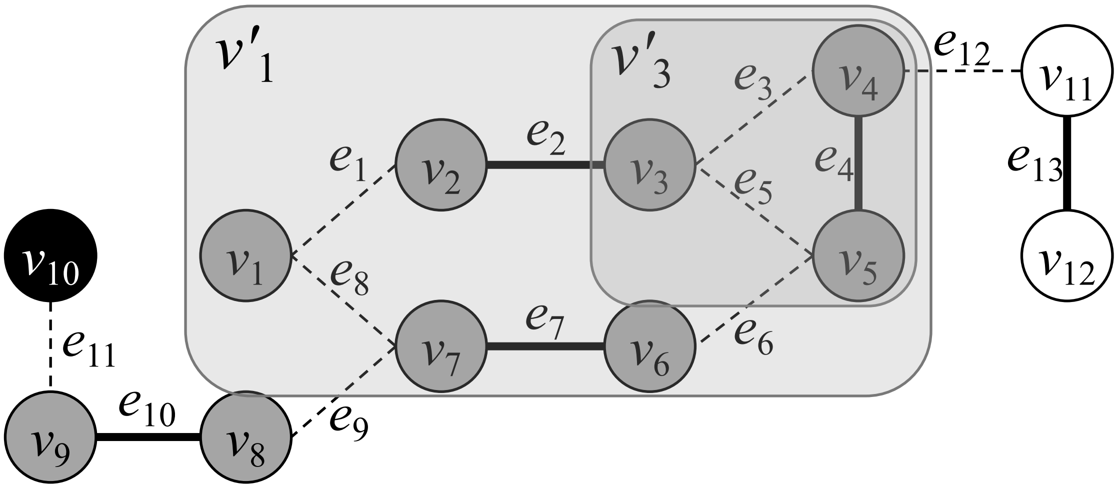
$v'_1$
$v'_3$
$e_{12}$
$e_3$
$v_4$
$v_{11}$
$e_2$
$e_1$
$v_2$
$v_3$
$e_4$
$e_{13}$
$e_5$
$v_{10}$
$v_1$
$e_8$
$v_5$
$v_{12}$
$e_7$
$e_6$
$e_{11}$
$v_7$
$v_6$
$e_{10}$
$e_9$
$v_9$
$v_8$

# 匹配和最大匹配



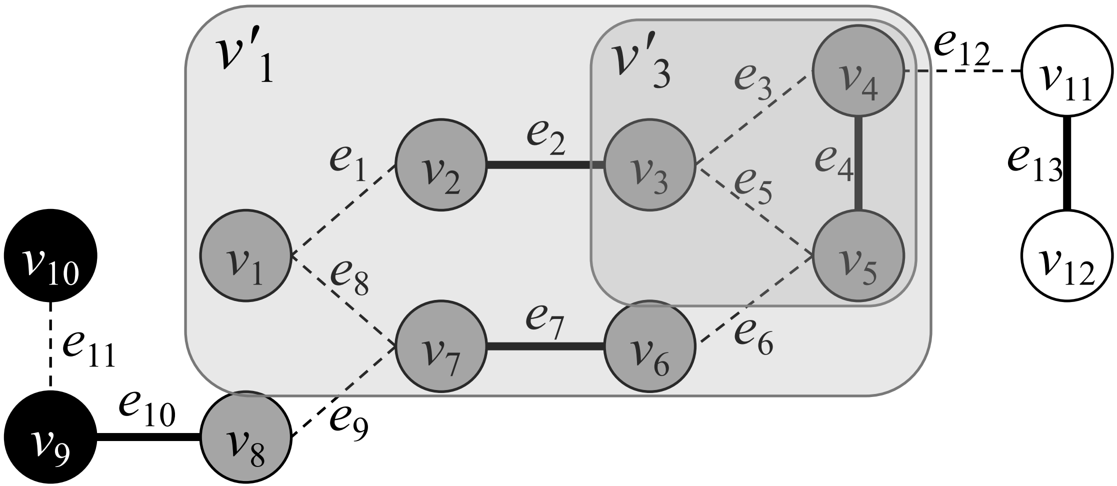
$v'_1$
$v'_3$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$

# 匹配和最大匹配



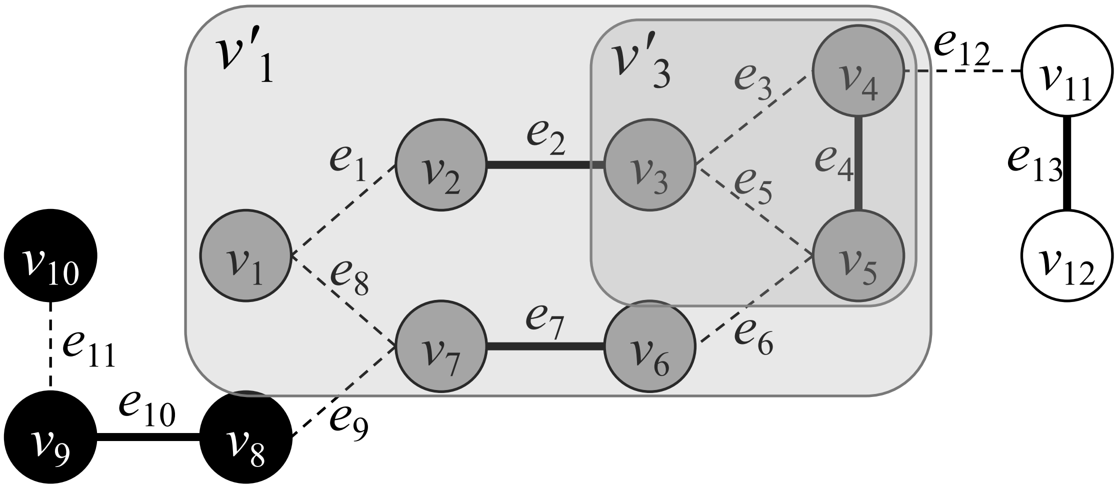
$v'_1$
$v'_3$
$e_{12}$
$e_3$
$v_4$
$v_{11}$
$e_2$
$e_1$
$v_2$
$v_3$
$e_5$
$e_4$
$e_{13}$
$v_{10}$
$v_1$
$e_8$
$v_5$
$v_{12}$
$e_7$
$e_6$
$v_7$
$v_6$
$e_{11}$
$e_{10}$
$e_9$
$v_9$
$v_8$

# 匹配和最大匹配



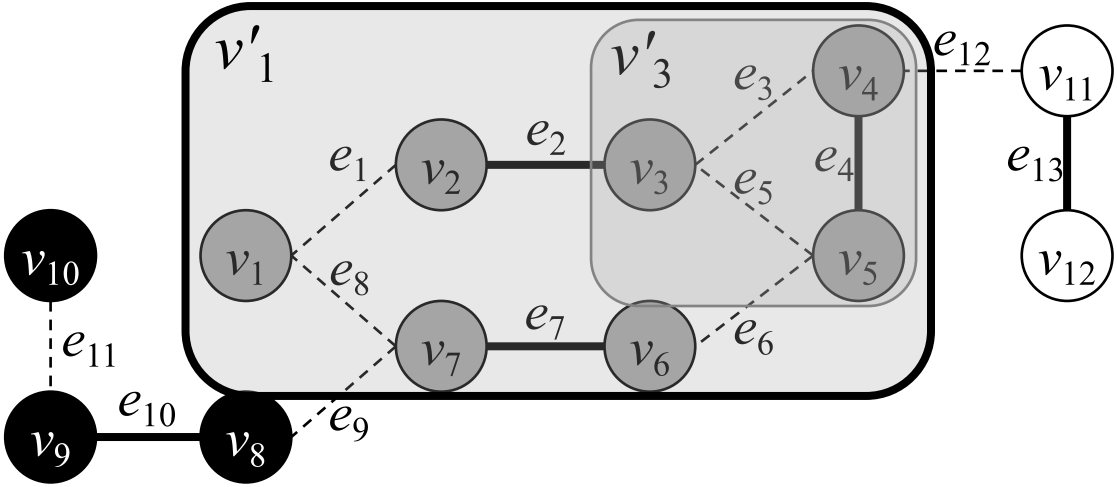
$v'_1$
$v'_3$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$

# 匹配和最大匹配



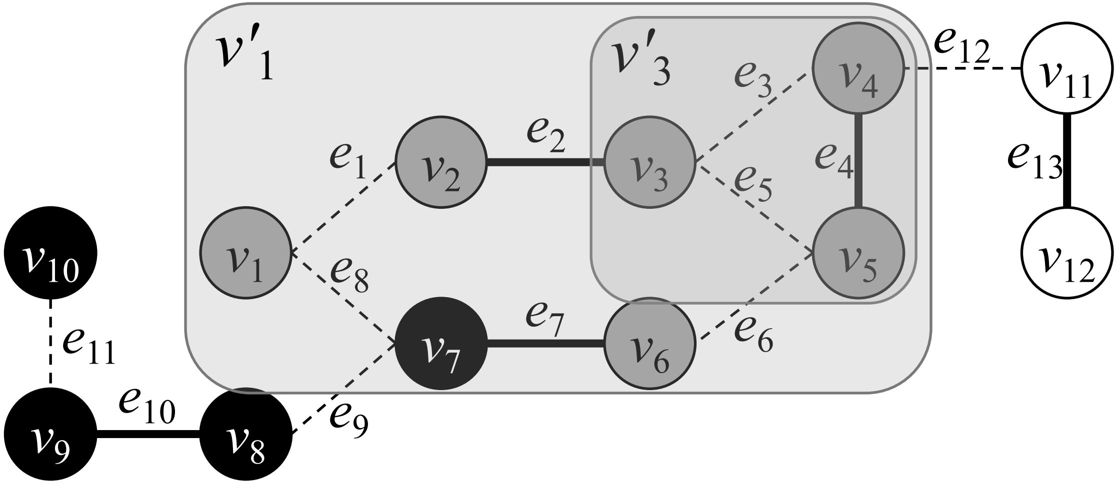
$v'_1$
$v'_3$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$

# 匹配和最大匹配



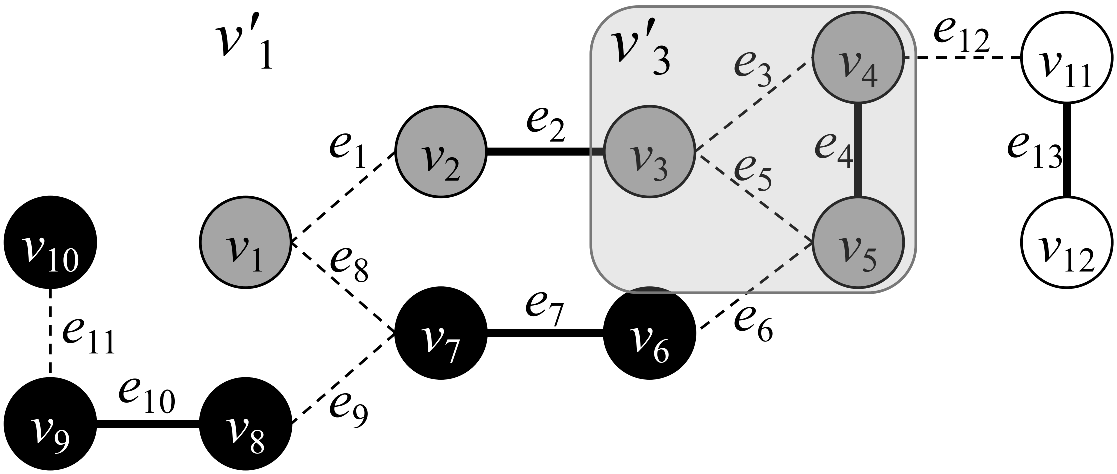
$v'_1$
$v'_3$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$

# 匹配和最大匹配

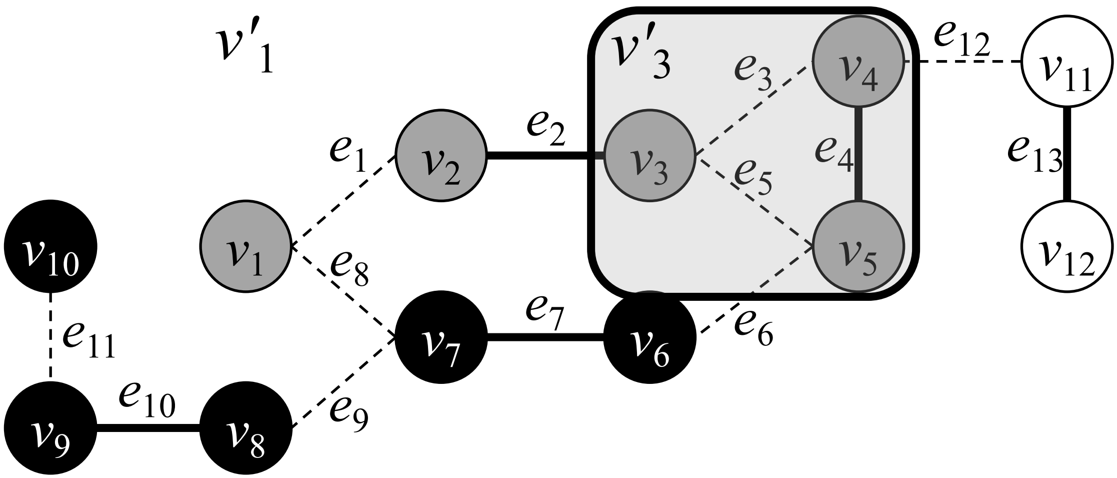


$v'_1$
$v'_3$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$

# 匹配和最大匹配

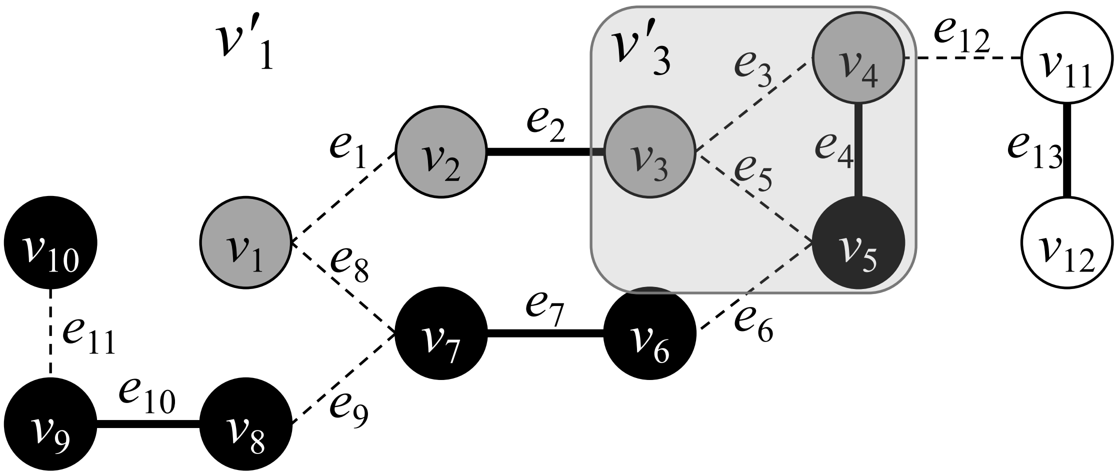

$v'_1$
$v'_3$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$

# 匹配和最大匹配



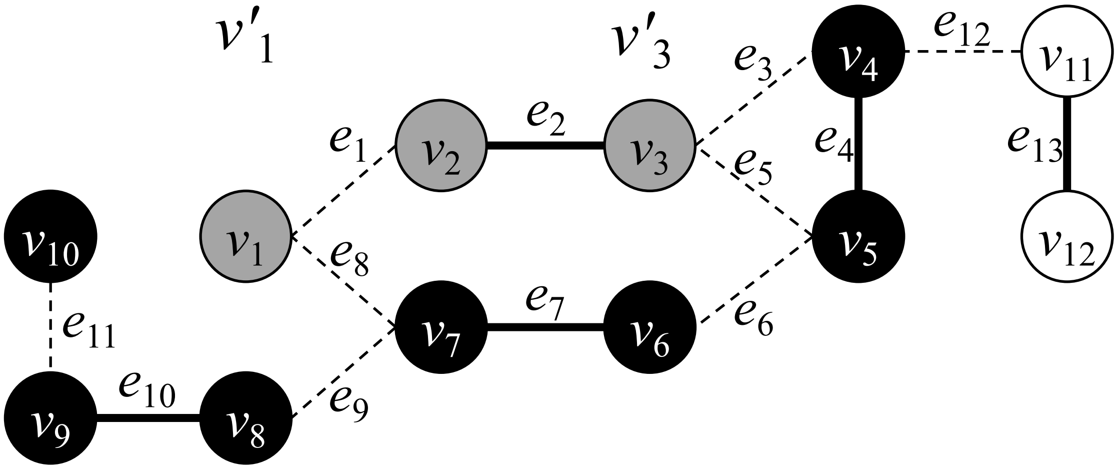
$v'_1$
$v'_3$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$

# 匹配和最大匹配



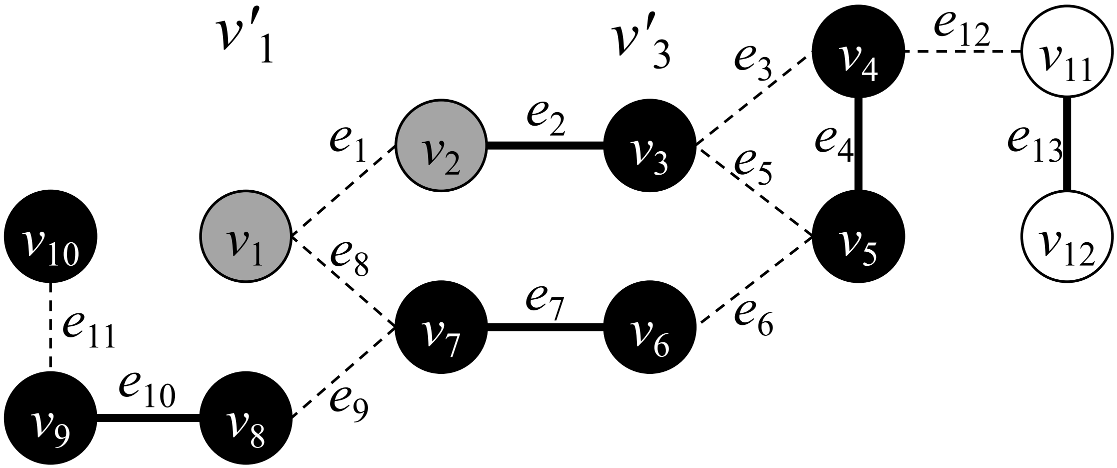
$v'_1$
$v'_3$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$

# 匹配和最大匹配



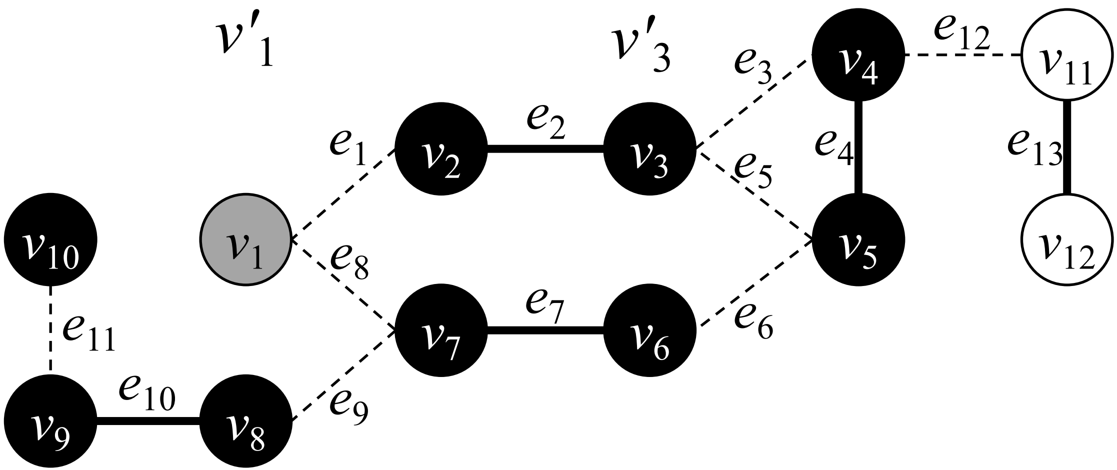
$v'_1$
$v'_3$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$

# 匹配和最大匹配



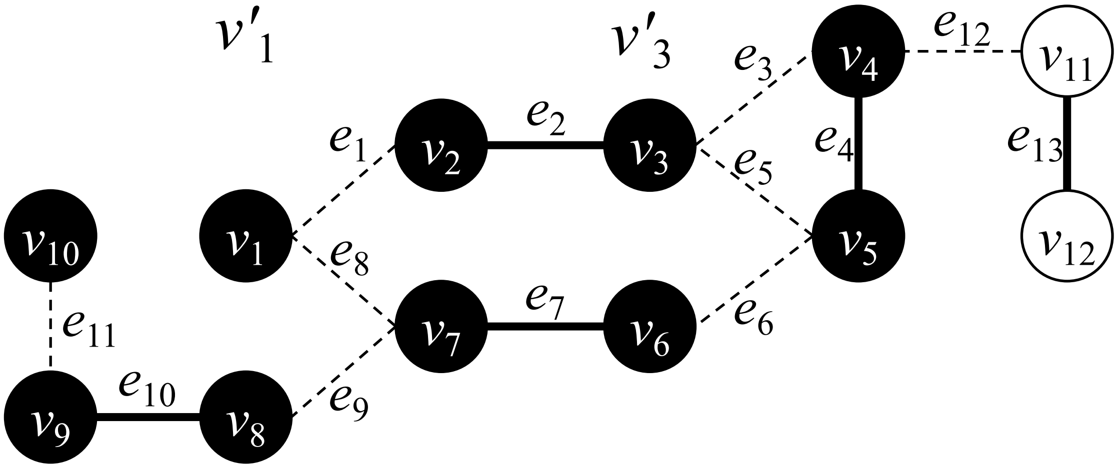
$v'_1$
$v'_3$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$

# 匹配和最大匹配



$v'_1$
$v'_3$
$v_1$
$v_2$
$v_3$
$v_4$
$v_5$
$v_6$
$v_7$
$v_8$
$v_9$
$v_{10}$
$v_{11}$
$v_{12}$
$e_1$
$e_2$
$e_3$
$e_4$
$e_5$
$e_6$
$e_7$
$e_8$
$e_9$
$e_{10}$
$e_{11}$
$e_{12}$
$e_{13}$

# 匹配和最大匹配



v'1
v'3
v1
v2
v3
v4
v5
v6
v7
v8
v9
v10
v11
v12
e1
e2
e3
e4
e5
e6
e7
e8
e9
e10
e11
e12
e13

# 匹配和最大匹配

- 时间复杂度$O(n^2(n+m))$
  - 每轮do-while循环中花的收缩和还原可发生$O(n)$ 次
  - 因此，花算法每轮do-while循环的时间复杂度为$O(n(n+m))$
- 若采用合适的数据结构处理花的收缩，时间复杂度可降为$O(n^3)$

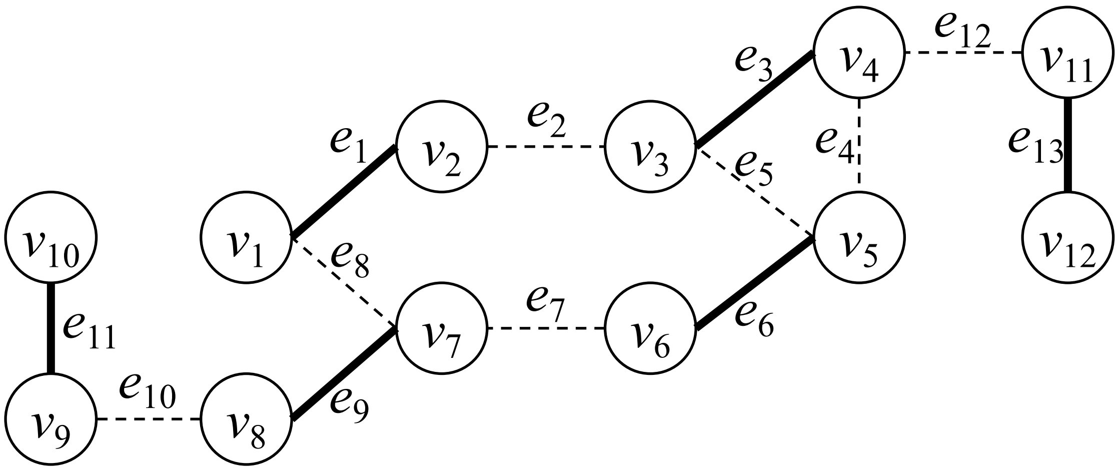
# 本次课的主要内容

5.1 匹配和最大匹配

5.2 完美匹配

# 完美匹配

- **完美匹配**：饱和所有顶点的匹配



# 完美匹配

- 每个图都有完美匹配吗？完美匹配存在的必要条件有哪些？

# 完美匹配

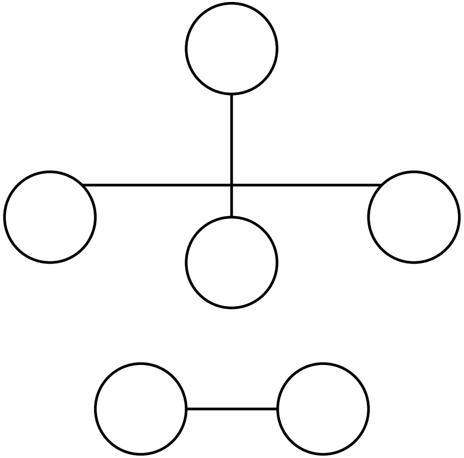
- 每个图都有完美匹配吗？完美匹配存在的必要条件有哪些？

- 完全图$K_{2n}$一定有完美匹配吗？
  若有，最多有多少个两两不相交的完美匹配？

# 完美匹配

- 每个图都有完美匹配吗？完美匹配存在的必要条件有哪些？

- 完全图$K_{2n}$一定有完美匹配吗？
  若有，最多有多少个两两不相交的完美匹配？



# 完美匹配

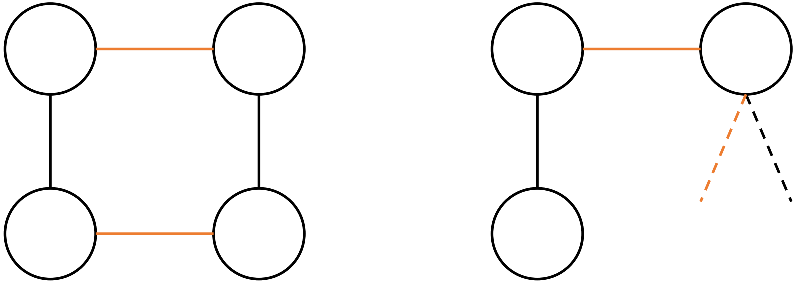
- 每个图都有完美匹配吗？完美匹配存在的必要条件有哪些？
- 完全图$K_{2n}$一定有完美匹配吗？
若有，最多有多少个两两不相交的完美匹配？
- 图的两个完美匹配的对称差的边导出子图的每个连通分支有什么特征？

# 完美匹配

- 每个图都有完美匹配吗？完美匹配存在的必要条件有哪些？

- 完全图$K_{2n}$一定有完美匹配吗？
  若有，最多有多少个两两不相交的完美匹配？

- 图的两个完美匹配的对称差的边导出子图的每个连通分支有什么特征？

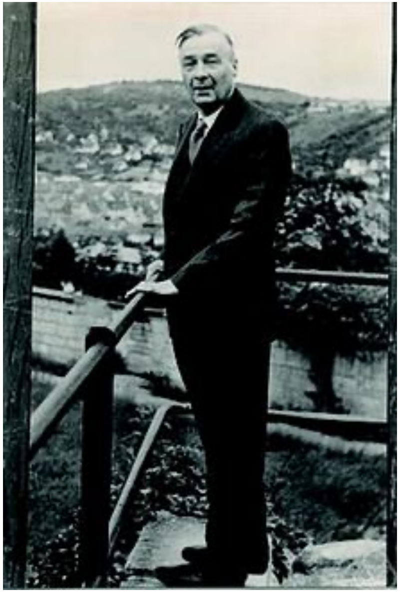


# 完美匹配

- 每个图都有完美匹配吗？完美匹配存在的必要条件有哪些？
- 完全图$K_{2n}$一定有完美匹配吗？
若有，最多有多少个两两不相交的完美匹配？
- 图的两个完美匹配的对称差的边导出子图的每个连通分支有什么特征？
- 树一定有完美匹配吗？若有，最多有多少个完美匹配？

# 完美匹配

- Philip Hall，1904-1982，出生于英国



https://en.wikipedia.org/wiki/Philip_Hall

# 完美匹配

- 霍尔定理：对于二分图$G = \langle X \cup Y, E \rangle$，$G$有饱和$X$中所有顶点的匹配当且仅当对于任意顶点子集$S \subseteq X$，$|\mathrm{N}(S)| \geq |S|$。
  - $\mathrm{N}(S)$：$S$中所有顶点的所有邻点形成的集合

# 完美匹配

- W. T. Tutte，1917-2002，出生于英国

在二战中解密了纳粹德国最高级别通讯……

https://en.wikipedia.org/wiki/W._T._Tutte

# 完美匹配

- 塔特定理：对于图$G = <V, E>$， $G$有完美匹配当且仅当对于任意顶点子集$S \subseteq V$， $\mathrm{o}(G - S) \leq |S|$。

# 书面作业

- 练习5.1
- 练习5.5、5.6

**练习 5.1.** 某个任务分配问题既要求某几位特定职员必须承担任务，又要求某几项特定任务必须有人承担。若分别满足上述两个要求之一的分配方式均存在，则同时满足上述两个要求的分配方式是否一定存在？

**练习 5.5.** “勇往直前”是图上的一种游戏，两位玩家参加，先手玩家任选一个顶点为起点，然后两位玩家交替在图上延长这条路，每次将路延长到终点的一个邻点，若某位玩家无法延长则败北。若两位玩家都足够聪明，先手和后手谁将获胜？

**练习 5.6.** “缺一不可”是一种扑克游戏，两位玩家参加，给定一副不含大小王的扑克，先手玩家将 *52* 张牌平均放入 *13* 个抽屉中，后手玩家若能从每个抽屉中选 *1* 张并恰凑齐从 $A$ 到 $K$ 的 *13* 种点数则获胜，否则失败。若两位玩家都足够聪明，先手和后手谁将获胜？