第5章 匹配

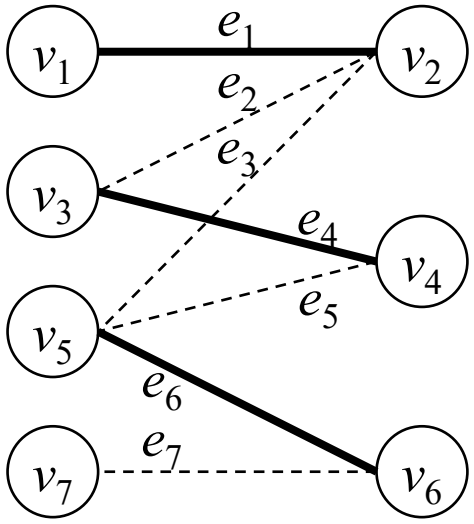
程龚
南京大学 计算机学院
gcheng@nju.edu.cn
http://ws.nju.edu.cn/~gcheng

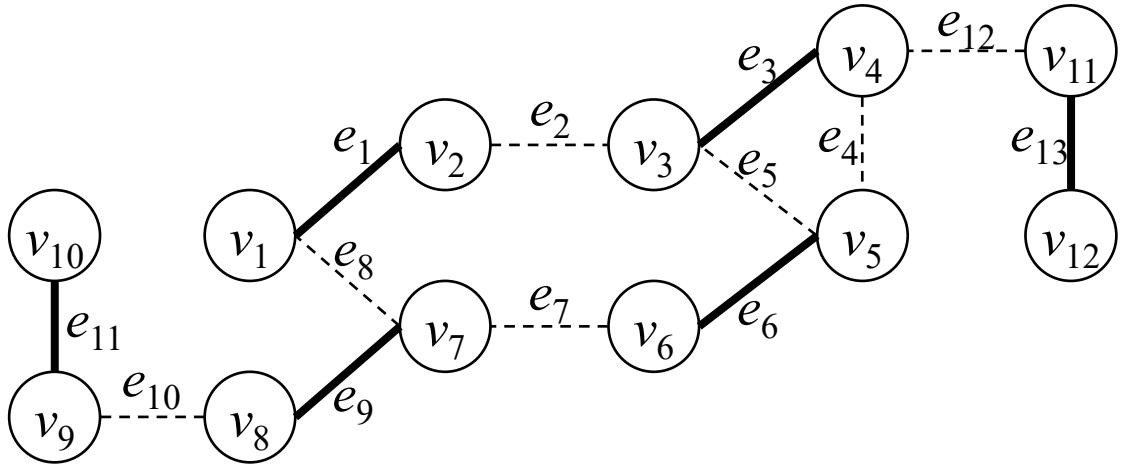
本章内容

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

如何找出图中的最大匹配?

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





霍普克罗夫特-卡普算法

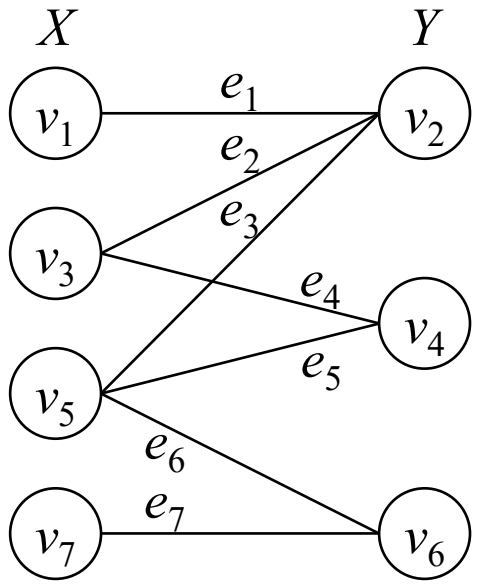
- 基本思路：改进匈牙利算法
  - 通过扩展BFS算法找增广路，比通过扩展DFS算法找到的增广路更短 → 减少单轮do-while循环的时间
  - 每轮do-while循环尝试同时找多条增广路，匹配的规模增幅更大 → 减少do-while循环的轮数

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y′ ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y′);
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



霍普克罗夫特-卡普算法

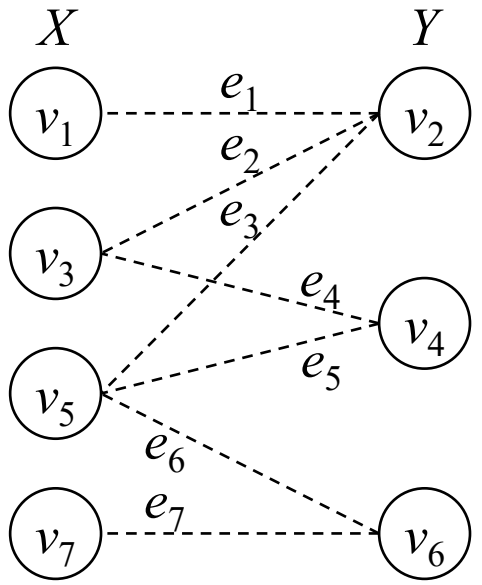
- 从初值为空集的匹配M开始

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



霍普克罗夫特-卡普算法

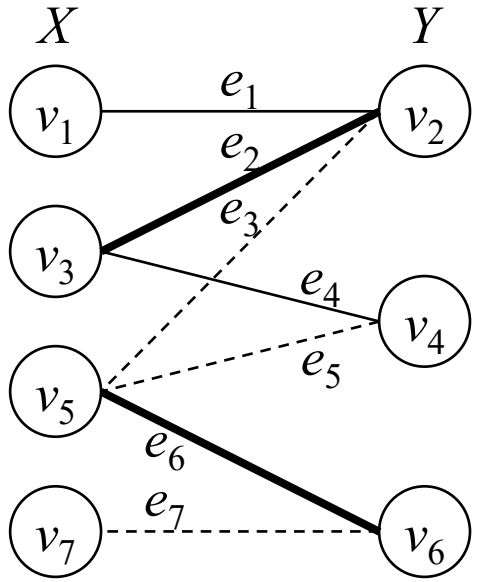
- 每轮do-while循环尝试找一个M增广路的集合$\mathcal{P}$，直至G中不存在M增广路，即$\mathcal{P}$为空：

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



霍普克罗夫特-卡普算法

- 每轮do-while循环尝试找一个M增广路的集合$\mathcal{P}$，直至G中不存在M增广路，即$\mathcal{P}$为空：
  - HKInit算法初始化BFS并返回队列Q，存储顶点子集X中所有未被M饱和的顶点子集作为BFS的出发点；

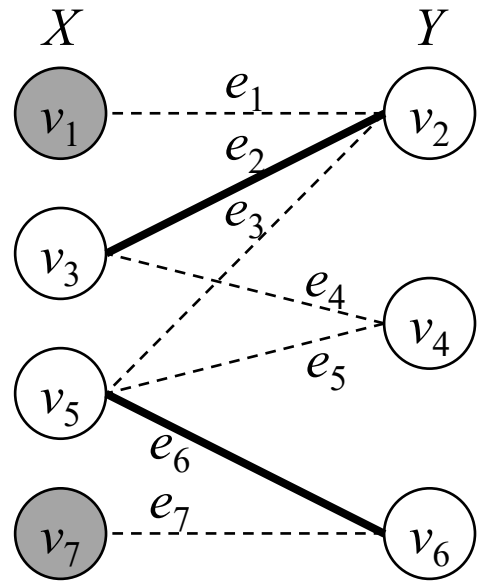
算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```

灰色顶点：Q



霍普克罗夫特-卡普算法

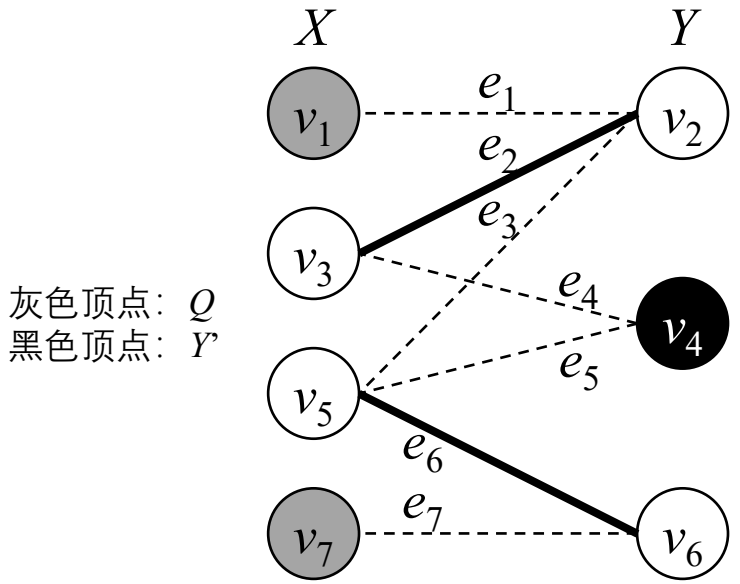
- 每轮do-while循环尝试找一个M增广路的集合$\mathcal{P}$，直至G中不存在M增广路，即$\mathcal{P}$为空：
  - HKInit算法初始化BFS并返回队列Q，存储顶点子集X中所有未被M饱和的顶点子集作为BFS的出发点；
  - HKBFS算法运行扩展的BFS算法并返回通过M交错路找到的顶点子集Y中未被M饱和的顶点子集Y'；

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



霍普克罗夫特-卡普算法

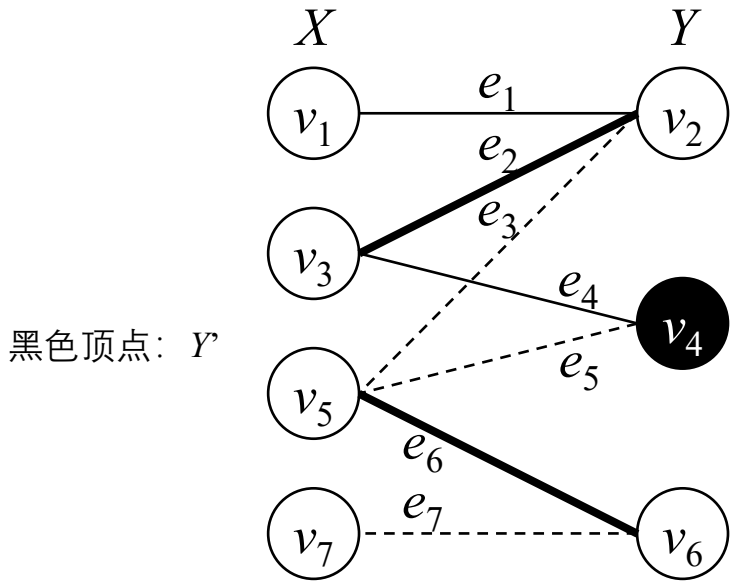
- 每轮do-while循环尝试找一个M增广路的集合$\mathcal{P}$，直至G中不存在M增广路，即$\mathcal{P}$为空：
  - HKInit算法初始化BFS并返回队列Q，存储顶点子集X中所有未被M饱和的顶点子集作为BFS的出发点；
  - HKBFS算法运行扩展的BFS算法并返回通过M交错路找到的顶点子集Y中未被M饱和的顶点子集Y'；
  - HKPaths算法找出以Y'中的顶点为终点的一组不经过相同顶点的M增广路$\mathcal{P}$；

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



霍普克罗夫特-卡普算法

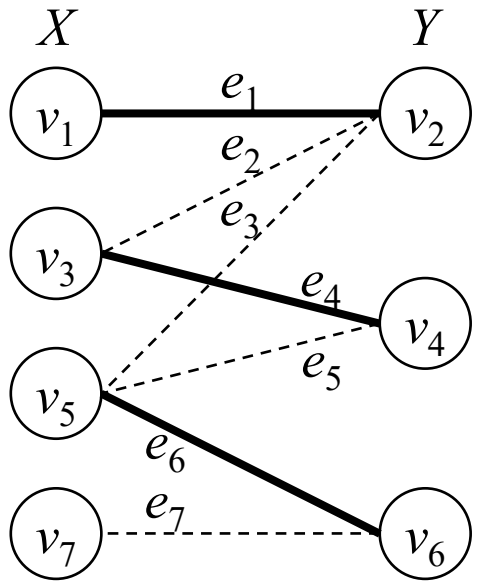
- 每轮do-while循环尝试找一个M增广路的集合$\mathcal{P}$，直至G中不存在M增广路，即$\mathcal{P}$为空：
  - HKInit算法初始化BFS并返回队列Q，存储顶点子集X中所有未被M饱和的顶点子集作为BFS的出发点；
  - HKBFS算法运行扩展的BFS算法并返回通过M交错路找到的顶点子集Y中未被M饱和的顶点子集Y'；
  - HKPaths算法找出以Y'中的顶点为终点的一组不经过相同顶点的M增广路$\mathcal{P}$；
  - 相继计算$\mathcal{P}$中每条路P经过的边的集合和M的对称差，得到包含边的数量更多的匹配。

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



霍普克罗夫特-卡普算法

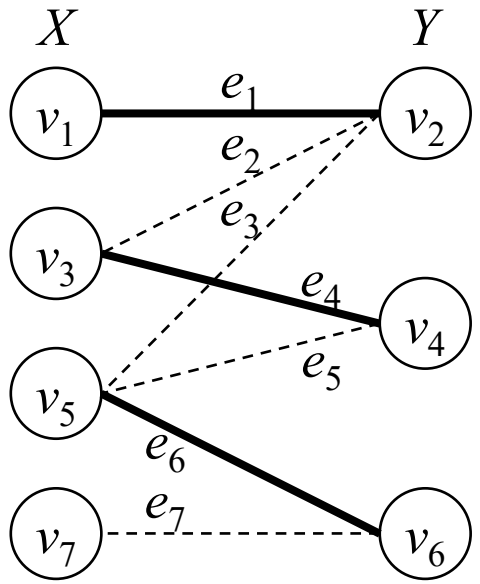
- 算法运行结束时，输出最大匹配M。

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



HKInit算法

- 初始化BFS并返回队列Q，存储顶点子集X中所有未被M饱和的顶点子集作为BFS的出发点：

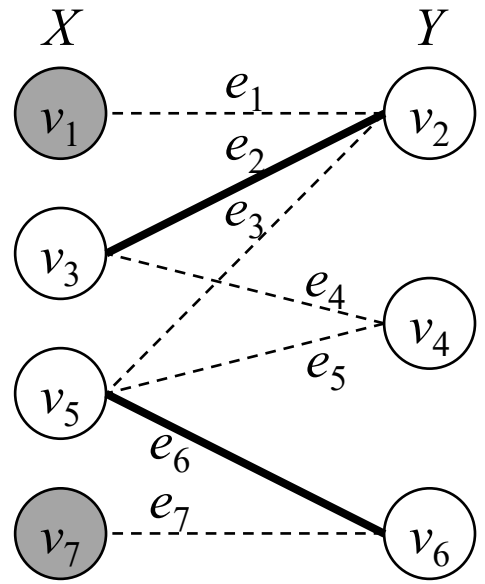
算法 5.4: HKInit 算法

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

初值: 队列 Q 初值为空

```
1 foreach u ∈ (X ∪ Y) do
2     if u ∈ X 且 u 未被 M 饱和 then
3         u.visited ← true;
4         u.d ← 0;
5         入队列 (Q, u);
6     else
7         u.visited ← false;
8         u.d ← ∞;
9 return Q;
```

灰色顶点：Q



HKInit算法

- 初始化BFS并返回队列Q，存储顶点子集X中所有未被M饱和的顶点子集作为BFS的出发点：
  - 同时从顶点子集X中所有未被匹配M饱和的顶点u出发（扩展BFS算法）

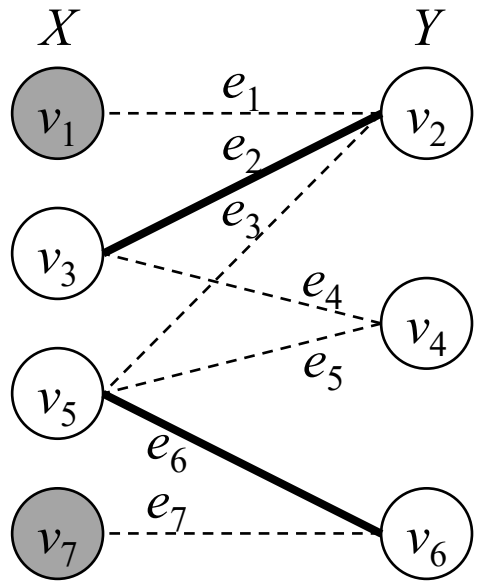
算法 5.4: HKInit 算法

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

初值: 队列 Q 初值为空

```
1 foreach u ∈ (X ∪ Y) do
2     if u ∈ X 且 u 未被 M 饱和 then
3         u.visited ← true;
4         u.d ← 0;
5         入队列 (Q, u);
6     else
7         u.visited ← false;
8         u.d ← ∞;
9 return Q;
```

灰色顶点：Q



HKInit算法

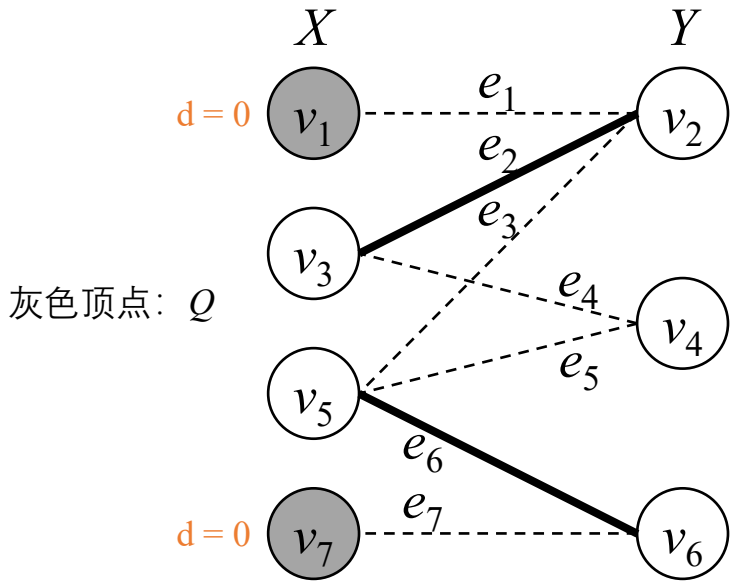
- 初始化BFS并返回队列Q，存储顶点子集X中所有未被M饱和的顶点子集作为BFS的出发点：
  - 同时从顶点子集X中所有未被匹配M饱和的顶点u出发（扩展BFS算法）
  - 顶点集$X \cup Y$中每个顶点的d属性：该顶点和所有出发点u间的最短距离（扩展BFS算法）

算法 5.4: HKInit 算法

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

初值: 队列 Q 初值为空

```
1 foreach u ∈ (X ∪ Y) do
2     if u ∈ X 且 u 未被 M 饱和 then
3         u.visited ← true;
4         u.d ← 0;
5         入队列 (Q, u);
6     else
7         u.visited ← false;
8         u.d ← ∞;
9 return Q;
```



HKBFS算法

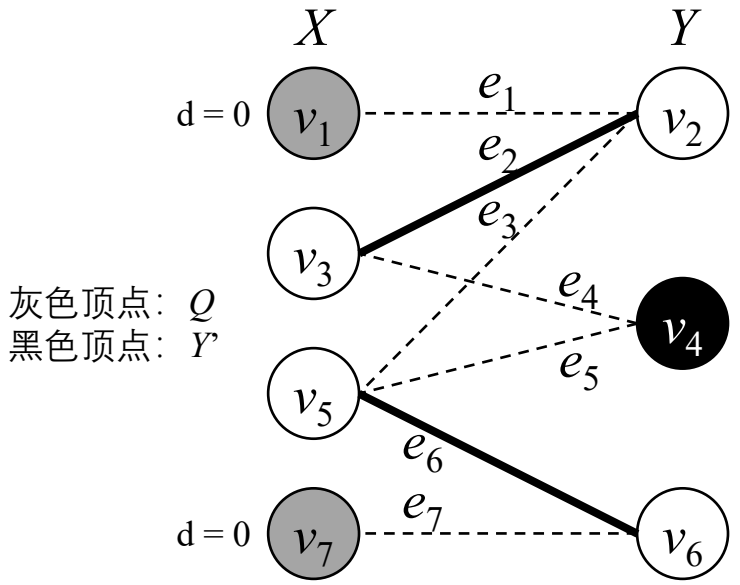
- 运行扩展的BFS算法并返回通过M交错路找到的顶点子集Y中未被M饱和的顶点子集Y’:

算法 5.5: HKBFS 算法

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

初值: 顶点子集 Y' 初值为 $\emptyset$；变量 d' 初值为 ∞

```
1  while Q 非空 do
2      v ← 出队列 (Q);
3      if v.d > d' then
4          中止 while 循环;
5      else if v 未被 M 饱和且 v.d > 0 then
6          Y' ← Y' ∪ {v};
7          d' ← v.d;
8      else
9          foreach (v, w) ∈ E do
10             if w.visited = false 且 BFS 树中从根顶点到 w 的路是
                M 交错路 then
11                 w.visited ← true;
12                 w.d ← v.d + 1;
13                 入队列 (Q, w);
14 return Y';
```



HKBFS算法

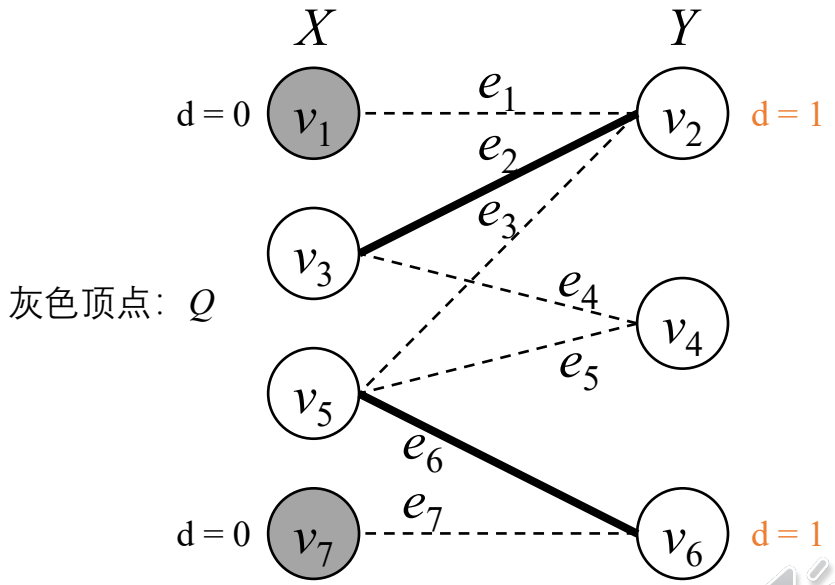
- 运行扩展的BFS算法并返回通过M交错路找到的顶点子集Y中未被M饱和的顶点子集Y'：
  - 基于BFS算法

算法 5.5: HKBFS 算法

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

初值: 顶点子集 Y' 初值为 $\emptyset$

```
1  while Q 非空 do
2      v ← 出队列 (Q);
3
4
5
6
7
8
9      foreach (v, w) ∈ E do
10         if w.visited = false
                       then
11             w.visited ← true;
12             w.d ← v.d + 1;
13             入队列 (Q, w);
14 return Y';
```



HKBFS算法

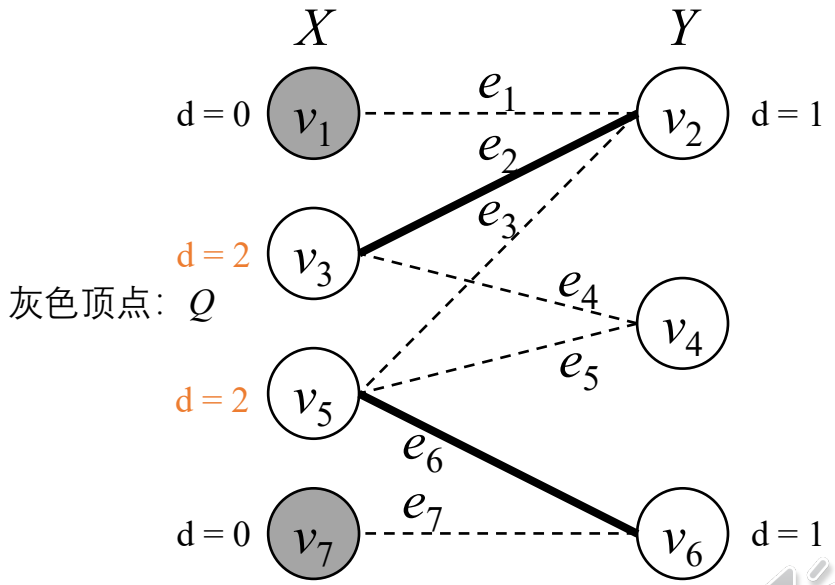
- 运行扩展的BFS算法并返回通过M交错路找到的顶点子集Y中未被M饱和的顶点子集Y'：
  - 第1项扩展：限制了可访问的邻点（与DFSAP算法类似）

算法 5.5: HKBFS 算法

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

初值: 顶点子集 Y' 初值为 $\emptyset$

```
1  while Q 非空 do
2      v ← 出队列 (Q);
3
4
5
6
7
8
9      foreach (v, w) ∈ E do
10         if w.visited = false 且 BFS 树中从根顶点到 w 的路是
           M 交错路 then
11             w.visited ← true;
12             w.d ← v.d + 1;
13             入队列 (Q, w);
14 return Y';
```



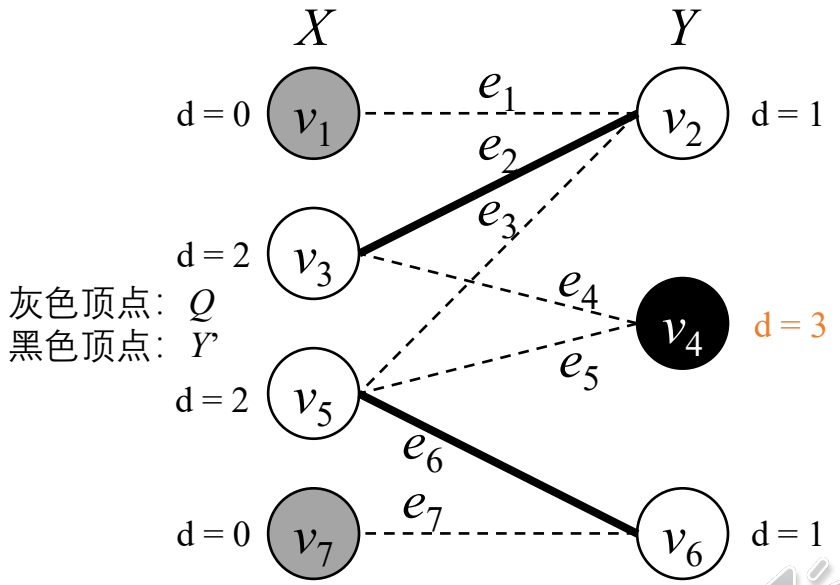
HKBFS算法

- 运行扩展的BFS算法并返回通过M交错路找到的顶点子集Y中未被M饱和的顶点子集Y'：
  - 第2项扩展：将未被匹配M饱和的非根顶点v作为M增广路的终点（与DFSAP算法类似）
    并不立刻返回BFS树中从根顶点到v的M增广路，而是先用集合Y'存储这些终点v（与DFSAP算法不同）

算法 5.5: HKBFS 算法

输入: 二分图 $G = \langle X \cup Y, E \rangle$，匹配 M，队列 Q
初值: 顶点子集 Y' 初值为 $\emptyset$

```
1  while Q 非空 do
2      v ← 出队列 (Q);
3
4
5          if v 未被 M 饱和且 v.d > 0 then
6          Y' ← Y' ∪ {v};
7
8      else
9          foreach (v, w) ∈ E do
10             if w.visited = false 且 BFS 树中从根顶点到 w 的路是
                M 交错路 then
11                 w.visited ← true;
12                 w.d ← v.d + 1;
13                 入队列 (Q, w);
14 return Y';
```



HKBFS算法

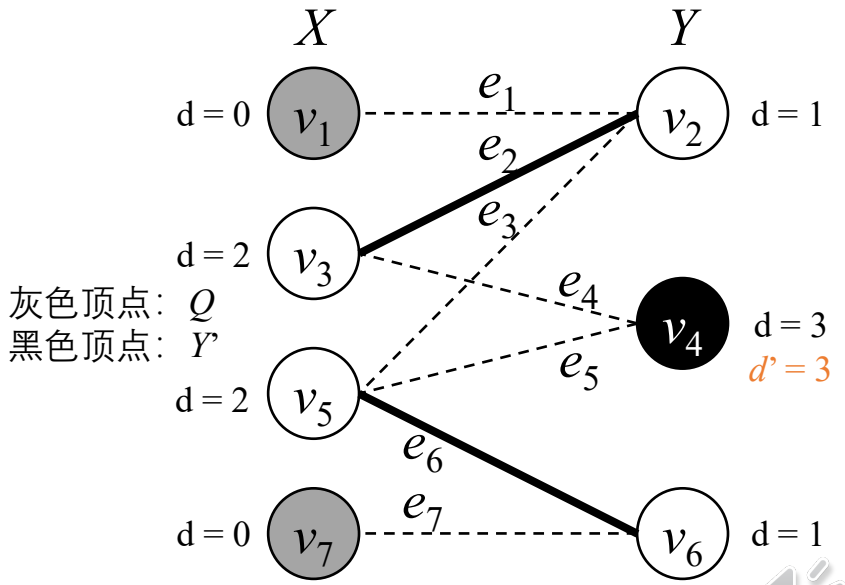
- 运行扩展的BFS算法并返回通过M交错路找到的顶点子集Y中未被M饱和的顶点子集Y’：
  - 第3项扩展：有可能提前中止BFS
    - d’：被访问的首个未被M饱和的非根顶点的d属性值，即最短的M增广路的长度；
    - 在访问了d属性值不超过d’的所有顶点后，不再访问d属性值更大的顶点，算法提前中止。
    - 最终，Y’存储所有d属性值为d’的未被M饱和的非根顶点。

算法 5.5: HKBFS 算法

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

初值: 顶点子集 Y' 初值为 $\emptyset$；变量 d' 初值为 ∞

```
1  while Q 非空 do
2      v ← 出队列 (Q);
3      if v.d > d' then
4          中止 while 循环;
5      else if v 未被 M 饱和且 v.d > 0 then
6          Y' ← Y' ∪ {v};
7          d' ← v.d;
8      else
9          foreach (v, w) ∈ E do
10             if w.visited = false 且 BFS 树中从根顶点到 w 的路是
                M 交错路 then
11                 w.visited ← true;
12                 w.d ← v.d + 1;
13                 入队列 (Q, w);
14 return Y';
```



HKPaths算法

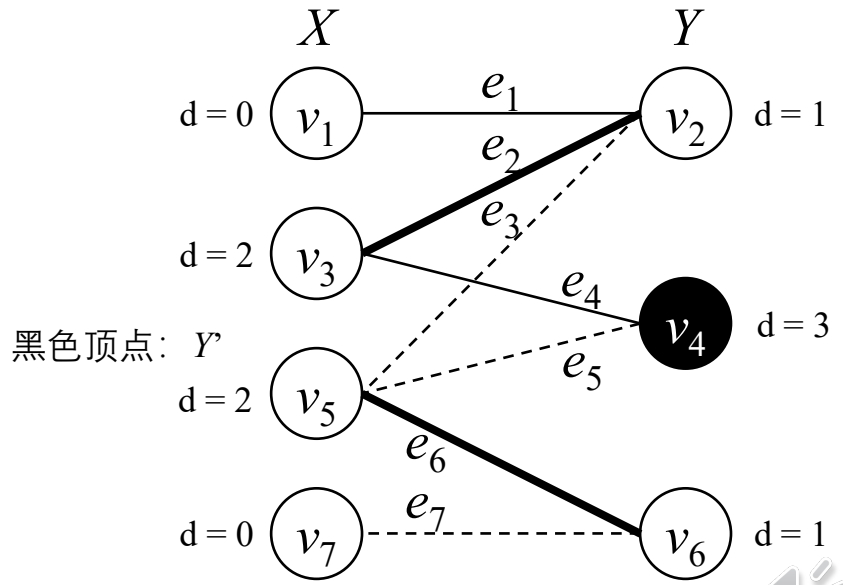
- 找出以Y'中的顶点为终点的一组不经过相同顶点的M增广路$\mathcal{P}$:

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



HKPaths算法

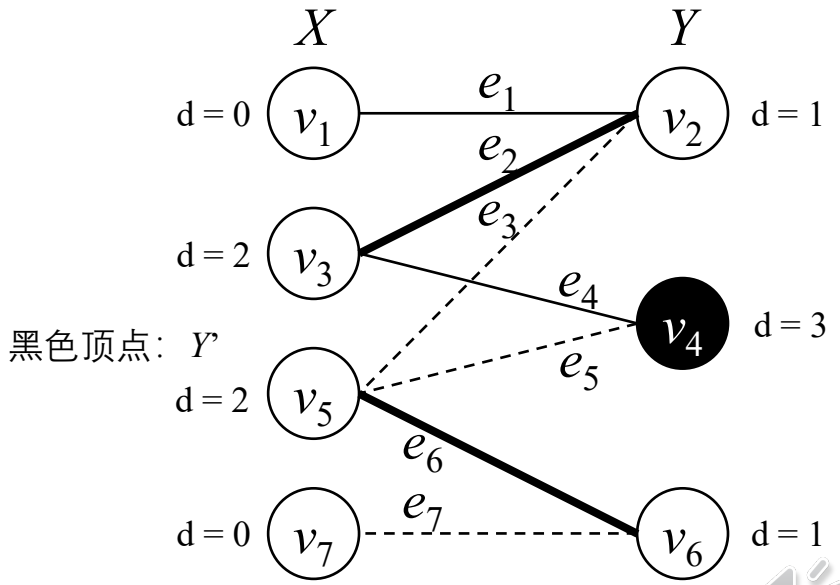
- 找出以Y'中的顶点为终点的一组不经过相同顶点的M增广路$\mathcal{P}$:
  - 相继从集合Y'中的每个顶点出发，沿顶点d属性值递减的方向，找出并返回极多的一组M增广路P，满足这些增广路不经过相同顶点，即后找到的增广路不允许经过先找到的增广路经过的顶点。

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



HKPaths算法

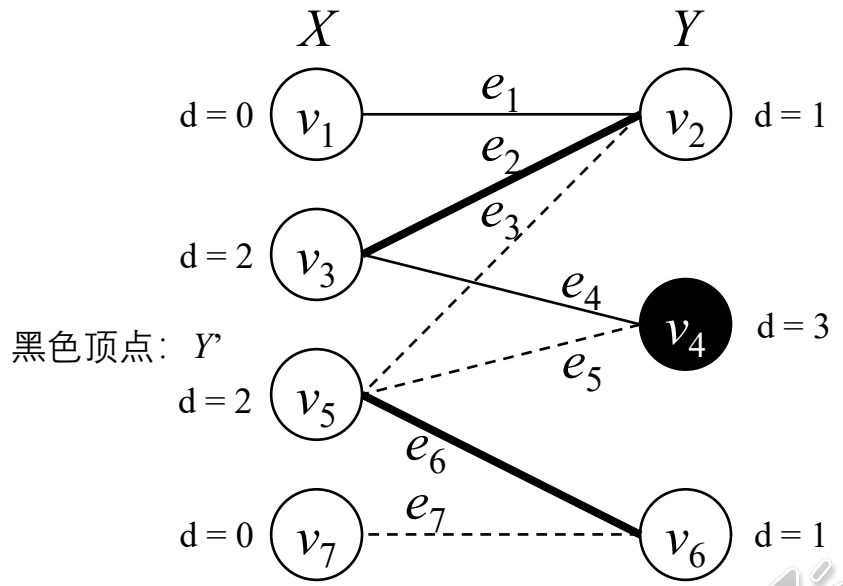
- 找出以Y'中的顶点为终点的一组不经过相同顶点的M增广路$\mathcal{P}$:
  - 相继从集合Y'中的每个顶点出发，沿顶点d属性值递减的方向，找出并返回极多的一组M增广路P，满足这些增广路不经过相同顶点，即后找到的增广路不允许经过先找到的增广路经过的顶点。
  - 由于这组M增广路不经过相同顶点，它们经过的边的集合可以互不干扰地和匹配M计算对称差，因此，霍普克罗夫特-卡普算法每轮do-while循环中M的规模增幅有可能超过1。

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



霍普克罗夫特-卡普算法

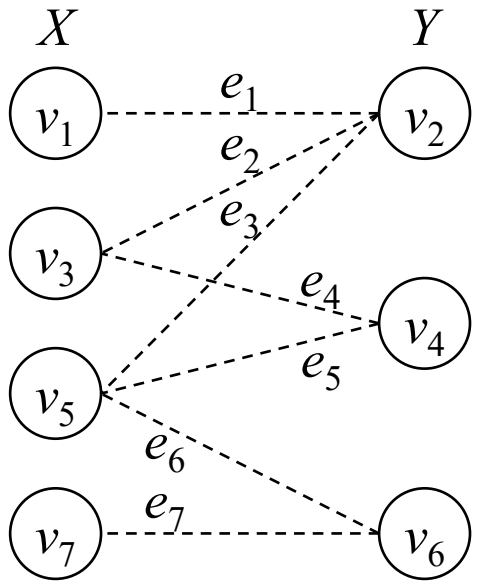
- 例如：第1轮do-while循环
  - 开始前：$M \leftarrow \emptyset$

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y′ ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y′);
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



霍普克罗夫特-卡普算法

- 例如：第1轮do-while循环
  - 开始前：$M \leftarrow \emptyset$
  - 开始后：
    - $Q \leftarrow \{v_1, v_3, v_5, v_7\}$

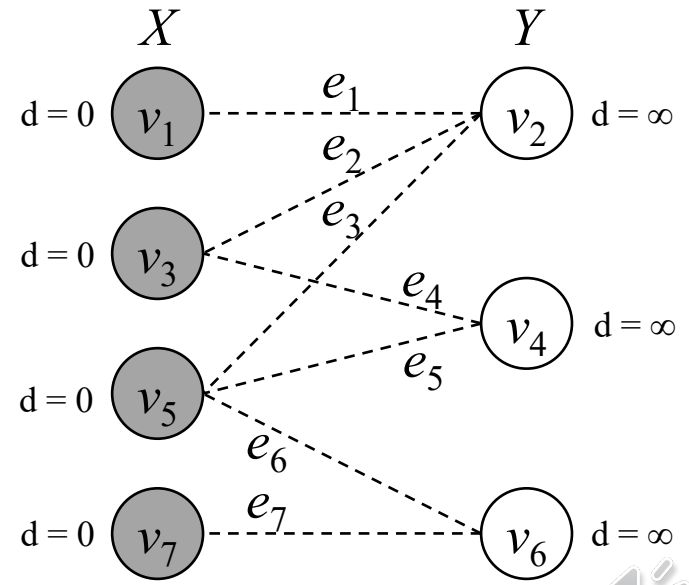
算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y′ ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y′);
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```

灰色顶点：Q



霍普克罗夫特-卡普算法

- 例如：第1轮do-while循环
  - 开始前：$M \leftarrow \emptyset$
  - 开始后：
    - $Q \leftarrow \{v_1, v_3, v_5, v_7\}$
    - $Y' \leftarrow \{v_2, v_4, v_6\}$

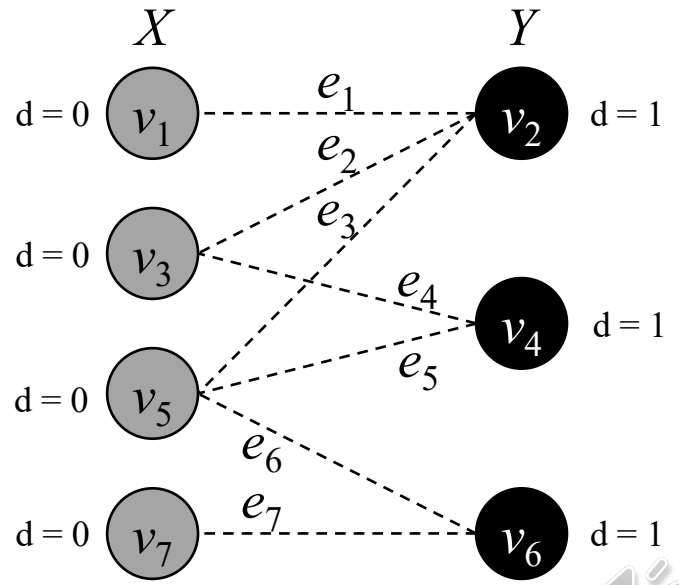
算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y′ ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y′);
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```

灰色顶点：Q
黑色顶点：Y'



霍普克罗夫特-卡普算法

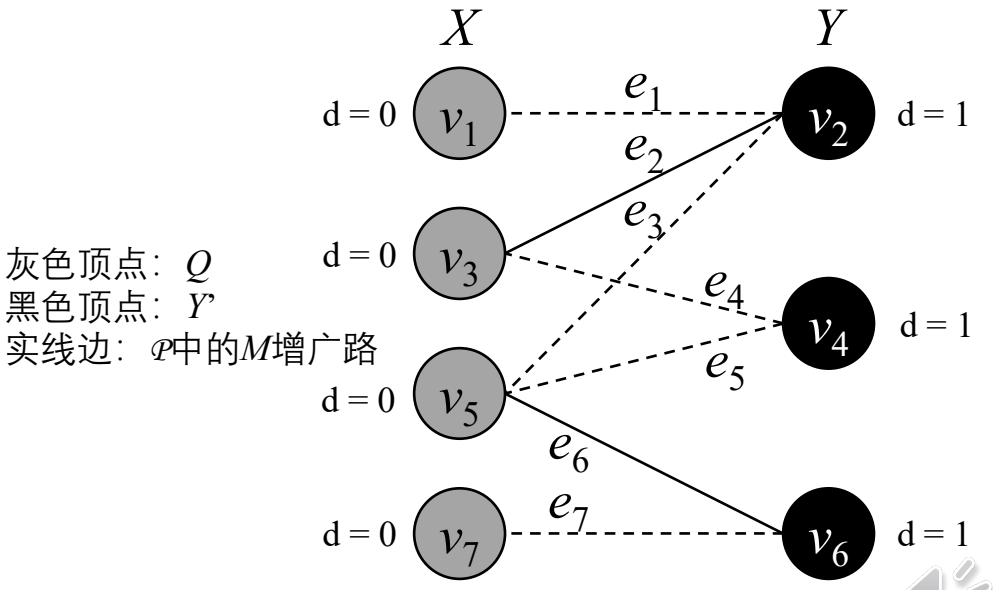
- 例如：第1轮do-while循环
  - 开始前：$M \leftarrow \emptyset$
  - 开始后：
    - $Q \leftarrow \{v_1, v_3, v_5, v_7\}$
    - $Y' \leftarrow \{v_2, v_4, v_6\}$
    - $\mathcal{P} \leftarrow$ 2条M增广路

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y′ ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y′);
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



霍普克罗夫特-卡普算法

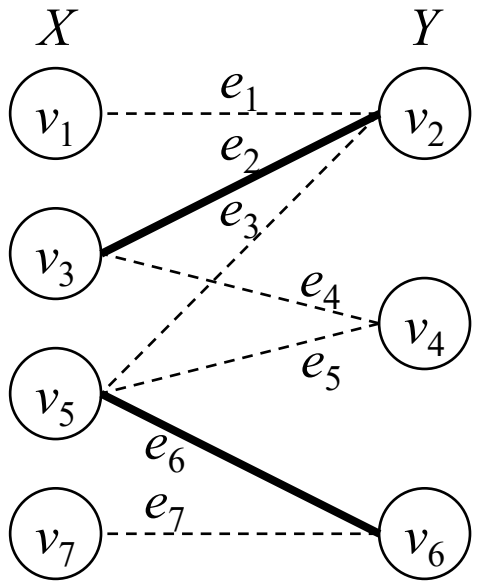
- 第2轮do-while循环
  - 开始前：$M \leftarrow \{e_2, e_6\}$

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



霍普克罗夫特-卡普算法

- 第2轮do-while循环
  - 开始前：$M \leftarrow \{e_2, e_6\}$
  - 开始后：
    - $Q \leftarrow \{v_1, v_7\}$

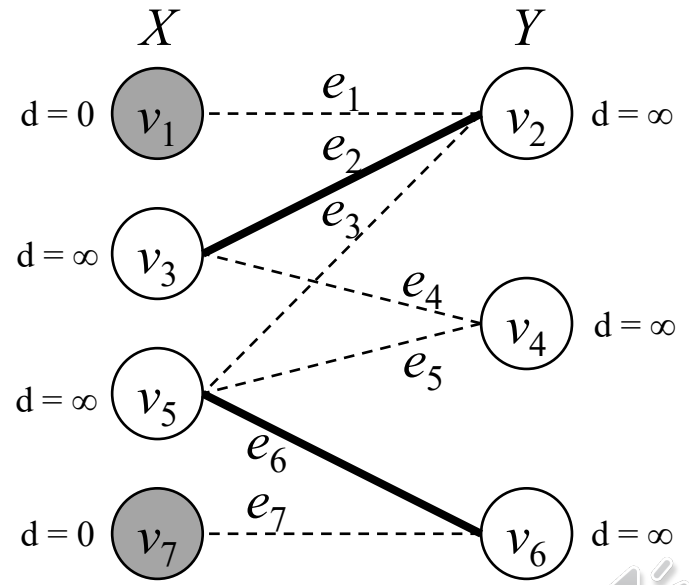
算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```

灰色顶点：Q



霍普克罗夫特-卡普算法

- 第2轮do-while循环
  - 开始前：$M \leftarrow \{e_2, e_6\}$
  - 开始后：
    - $Q \leftarrow \{v_1, v_7\}$
    - $Y' \leftarrow \{v_4\}$

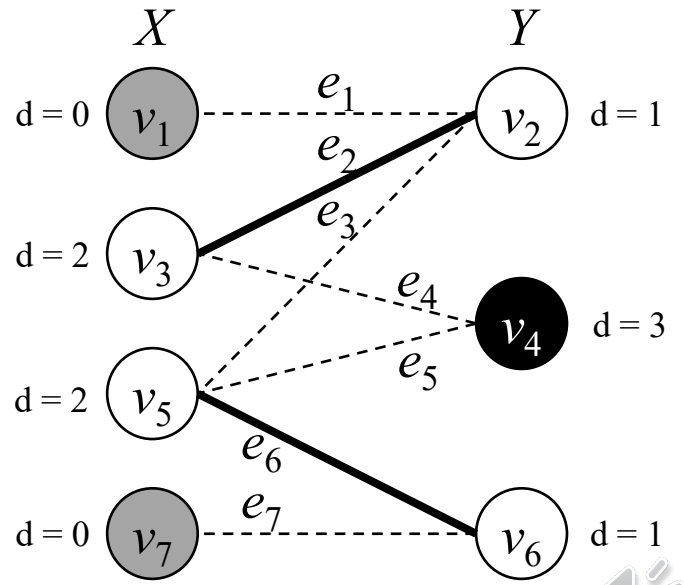
算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```

灰色顶点：Q
黑色顶点：Y'



霍普克罗夫特-卡普算法

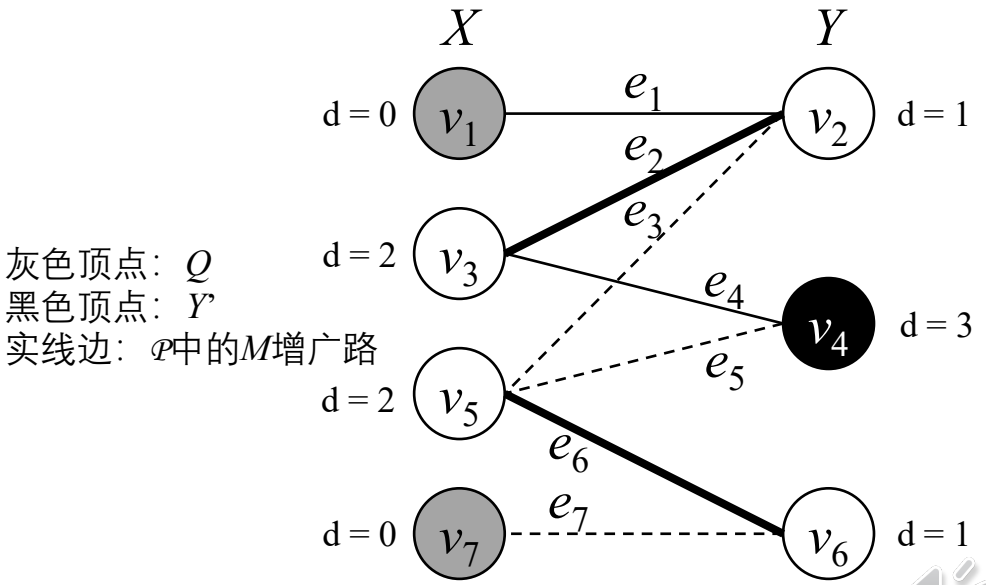
- 第2轮do-while循环
  - 开始前：$M \leftarrow \{e_2, e_6\}$
  - 开始后：
    - $Q \leftarrow \{v_1, v_7\}$
    - $Y' \leftarrow \{v_4\}$
    - $\mathcal{P} \leftarrow$ 1条M增广路

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



霍普克罗夫特-卡普算法

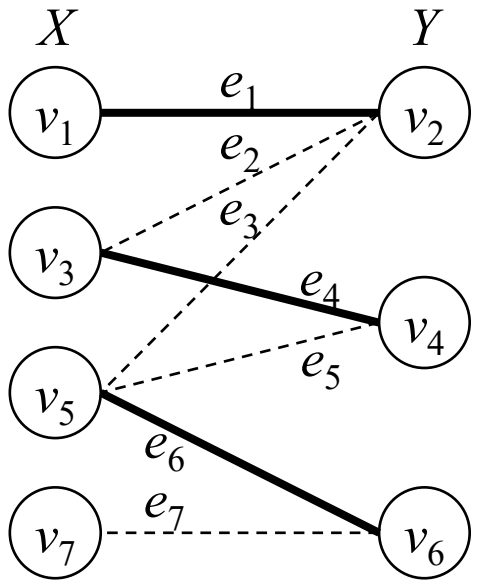
- 第3轮do-while循环
  - 开始前：$M \leftarrow \{e_1, e_4, e_6\}$

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y′ ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y′);
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



霍普克罗夫特-卡普算法

- 第3轮do-while循环
  - 开始前：$M \leftarrow \{e_1, e_4, e_6\}$
  - 开始后：
    - $Q \leftarrow \{v_7\}$

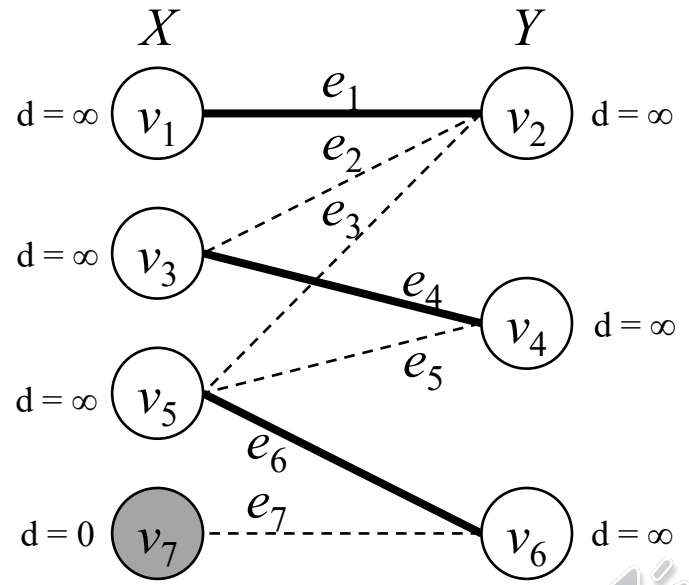
算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y′ ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y′);
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```

灰色顶点：Q



霍普克罗夫特-卡普算法

- 第3轮do-while循环
  - 开始前：$M \leftarrow \{e_1, e_4, e_6\}$
  - 开始后：
    - $Q \leftarrow \{v_7\}$
    - $Y' \leftarrow \emptyset$

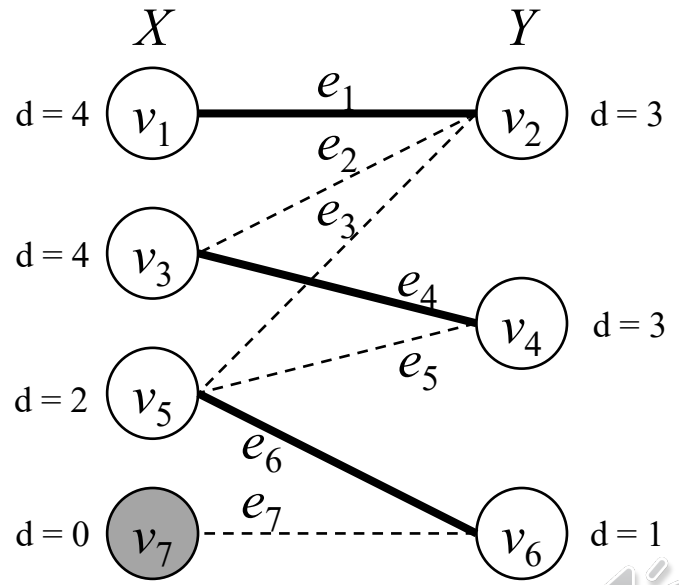
算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```

灰色顶点：Q
黑色顶点：Y'



霍普克罗夫特-卡普算法

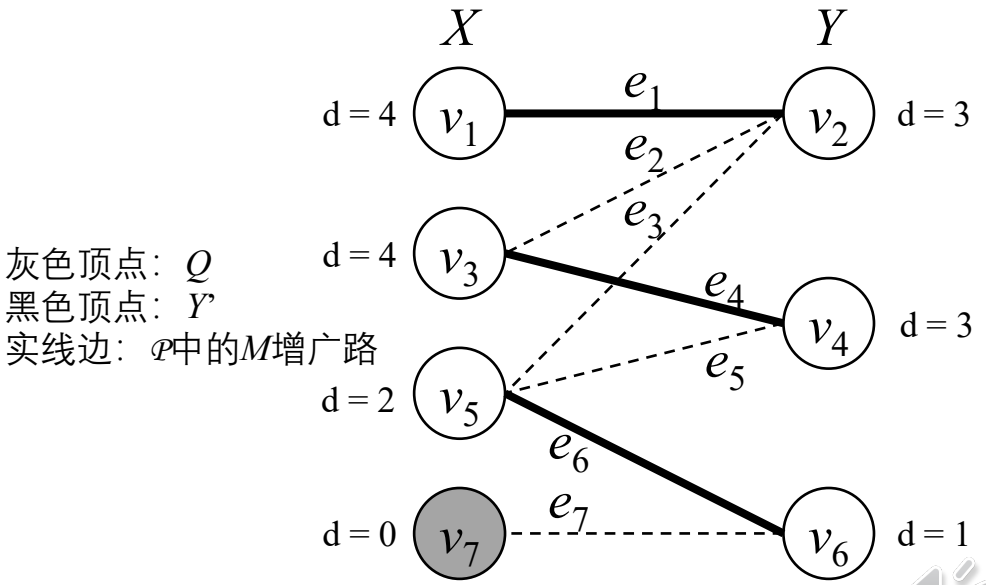
- 第3轮do-while循环
  - 开始前：$M \leftarrow \{e_1, e_4, e_6\}$
  - 开始后：
    - $Q \leftarrow \{v_7\}$
    - $Y' \leftarrow \emptyset$
    - $\mathcal{P} \leftarrow \emptyset$

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



霍普克罗夫特-卡普算法

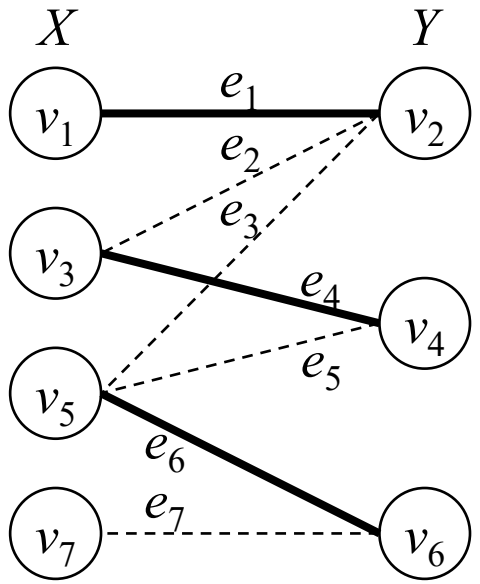
- 第3轮do-while循环
  - 开始前：$M \leftarrow \{e_1, e_4, e_6\}$
  - 开始后：
    - $Q \leftarrow \{v_7\}$
    - $Y' \leftarrow \emptyset$
    - $\mathcal{P} \leftarrow \emptyset$
- 算法运行结束，输出$M = \{e_1, e_4, e_6\}$

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y′ ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y′);
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



霍普克罗夫特-卡普算法

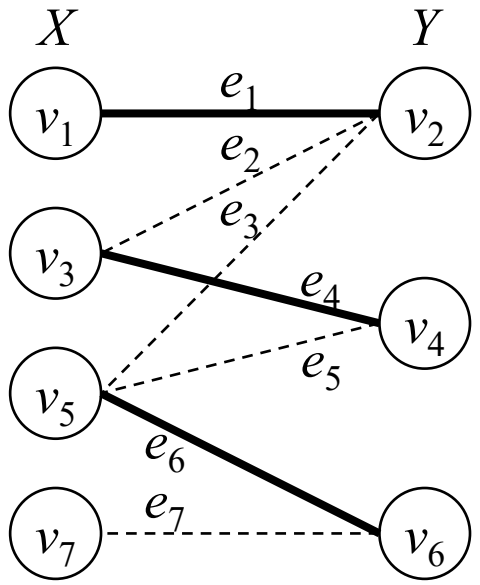
- 正确性：核心原理和匈牙利算法相似，
  只是将从单个顶点出发的DFS改为了从多个顶点同时出发的BFS。

算法 5.3: 霍普克罗夫特-卡普算法

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

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y′ ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y′);
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



霍普克罗夫特-卡普算法

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

> 定理5.3 随着do-while循环轮数的增加，变量d'的值严格单调增。

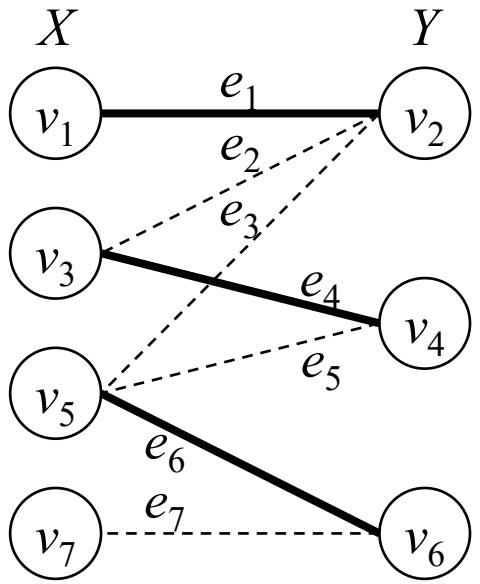
> 定理5.4 对于阶为n的图，所有do-while循环的变量d'有至多 $2\left\lfloor\sqrt{\frac{n}{2}}\right\rfloor+2$ 种值。

算法 5.3: 霍普克罗夫特-卡普算法

输入: 二分图 $G=\langle X\cup Y,E\rangle$

初值: 集合 M 初值为 $\emptyset$

```
1 do
2     Q ← HKInit(G, M);
3     Y' ← HKBFS(G, M, Q);
4     P ← HKPaths(G, Y');
5     foreach P ∈ P do
6         M ← P经过的边的集合 △ M;
7 while P ≠ ∅;
8 输出 (M);
```



接下来进入其它算法部分