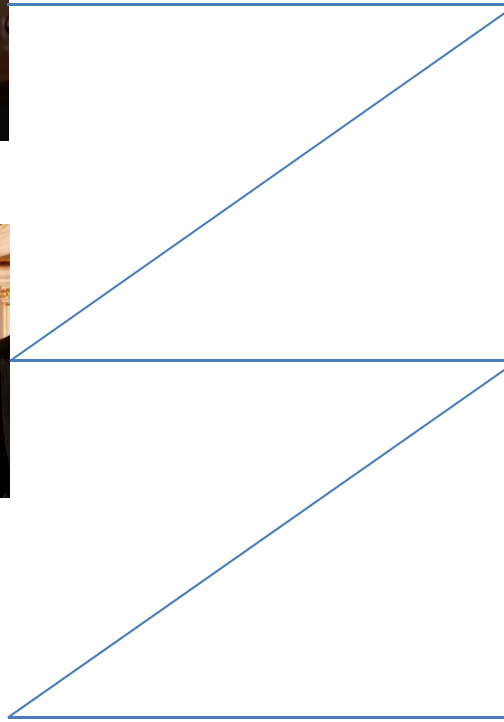


最大匹配算法

程龚 (gcheng@nju.edu.cn)

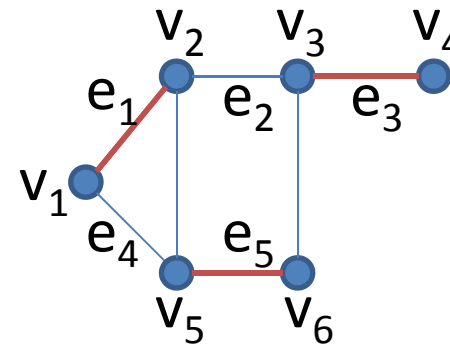
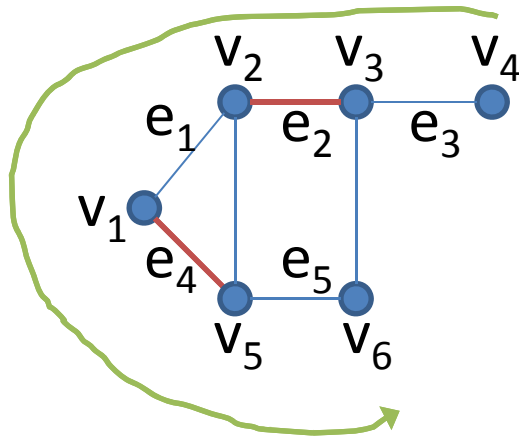
本节课的主要内容

- 面向二部图的增广路算法
- 面向二部图的Hopcroft-Karp算法
- 面向一般图的Edmonds算法



最大匹配的充要条件 (复习)

- 图 G 的一个匹配 M 是最大匹配的充分必要条件是 G 中不存在 M 增广路。
 - 假设存在 M 增广路 $P \Rightarrow$ 将 M 中在 P 上的边替换为 P 上的其它边 \Rightarrow 得到另一个匹配且势更大



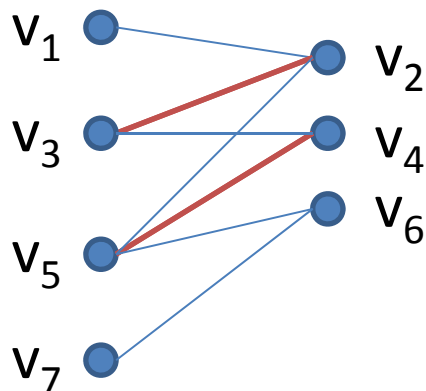
基于这个充要条件，
你能不能给出一种求最大匹配的方法？

面向二部图的增广路算法*

- 基本思路
 1. 搜索一条增广路
 2. 如果找到了：替换得到更大的匹配，回到第1步
 3. 否则：结束

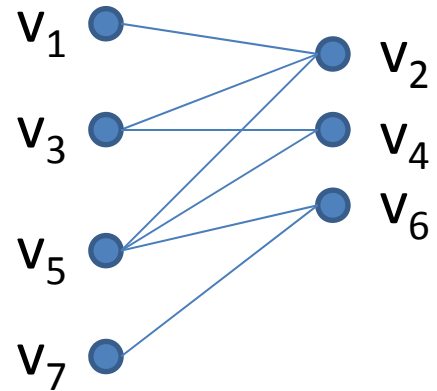
面向二部图的增广路算法 (续)

- 每轮搜索仅需从二部图特定的一侧顶点中开始（为什么不会遗漏？）
 - 增广路的长度是奇数 \Rightarrow 起点和终点位于二部图的两侧 \Rightarrow 仅从任何一侧开始搜索都能确保找到
- 每轮搜索的起点？
 - 从一侧中（不妨称作左侧）每一个未被当前匹配饱和的顶点开始
- 搜索哪些边？
 - 左侧到右侧：不在当前匹配中的边
 - 右侧到左侧：当前匹配中的边
- 搜索的终止条件（哪些情况下搜索结束？）
 - 找到一个未被当前匹配饱和的顶点（起点除外） \Rightarrow 找到一条增广路，替换得到更大的匹配 \Rightarrow 进入下一轮搜索
 - 或：已搜索所有的点和边，仍未找到 \Rightarrow 无增广路 \Rightarrow 找到最大匹配



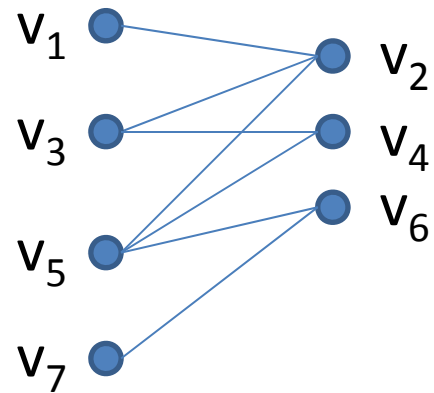
面向二部图的增广路算法 (续)

- 举例



面向二部图的增广路算法 (续)

- 举例

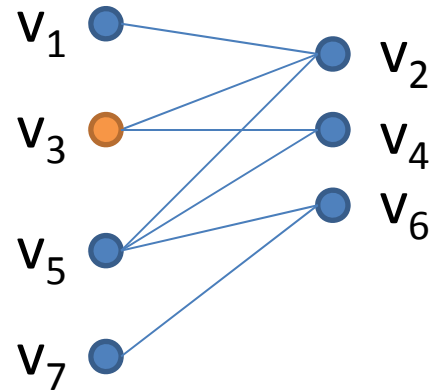


第一轮搜索开始

- 当前匹配: $\{\}$
- 未饱和的左侧顶点: $\{v_1, v_3, v_5, v_7\}$

面向二部图的增广路算法 (续)

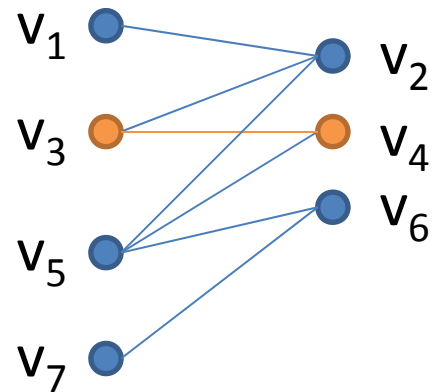
- 举例



从 $\{v_1, v_3, v_5, v_7\}$ 中未搜索过的 v_3 开始搜索

面向二部图的增广路算法 (续)

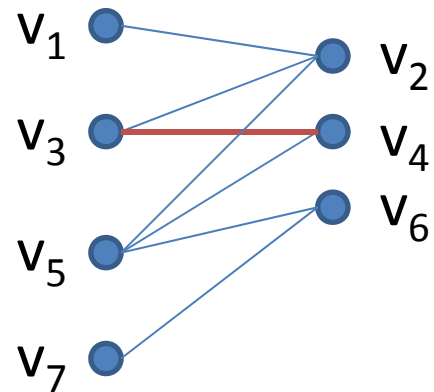
- 举例



沿不在当前匹配中且本轮未搜索过的(v_3, v_4)到达 $v_4 \Rightarrow v_4$ 未饱和

面向二部图的增广路算法 (续)

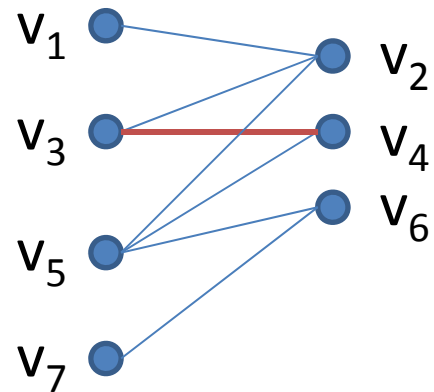
- 举例



找到增广路 $v_3v_4 \Rightarrow$ 替换进当前匹配中 \Rightarrow
本轮搜索结束

面向二部图的增广路算法 (续)

- 举例

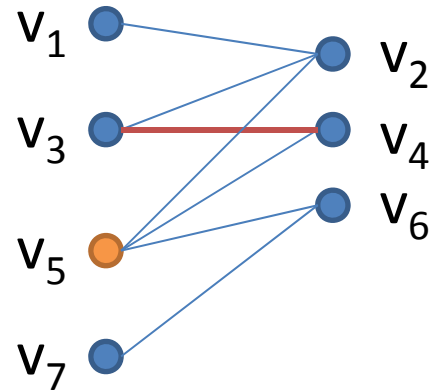


第二轮搜索开始

- 当前匹配: $\{(v_3, v_4)\}$
- 未饱和的左侧顶点: $\{v_1, v_5, v_7\}$

面向二部图的增广路算法 (续)

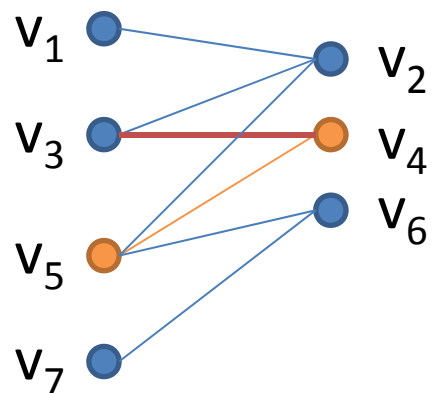
- 举例



从 $\{v_1, v_5, v_7\}$ 中未搜索过的 v_5 开始搜索

面向二部图的增广路算法 (续)

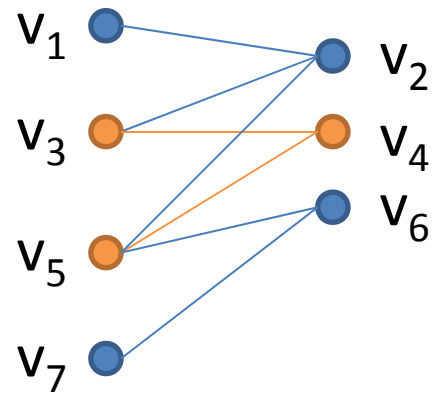
- 举例



沿不在当前匹配中且本轮未搜索过的(v_5, v_4)到达 $v_4 \Rightarrow v_4$ 已被当前匹配中的(v_3, v_4)饱和和

面向二部图的增广路算法 (续)

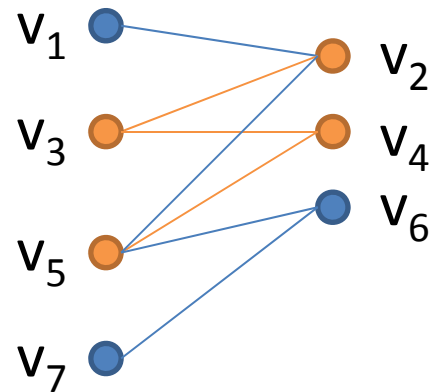
- 举例



沿当前匹配中的 (v_3, v_4) 到达 v_3

面向二部图的增广路算法 (续)

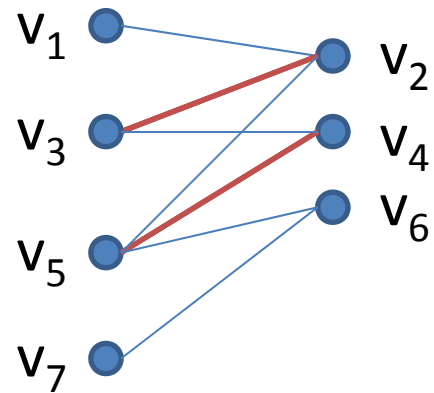
- 举例



沿不在当前匹配中且本轮未搜索过的(v_3, v_2)到达 $v_2 \Rightarrow v_2$ 未饱和

面向二部图的增广路算法 (续)

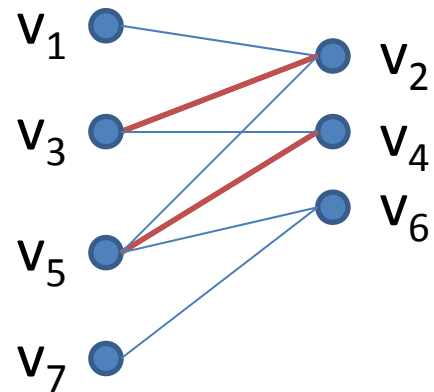
- 举例



找到增广路 $v_5v_4v_3v_2 \Rightarrow$ 替换进当前匹配中
 \Rightarrow 本轮搜索结束

面向二部图的增广路算法 (续)

- 举例

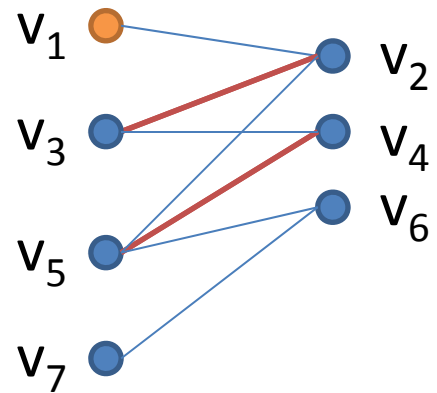


第三轮搜索开始

- 当前匹配: $\{(v_3, v_2), (v_5, v_4)\}$
- 未饱和的左侧顶点: $\{v_1, v_7\}$

面向二部图的增广路算法 (续)

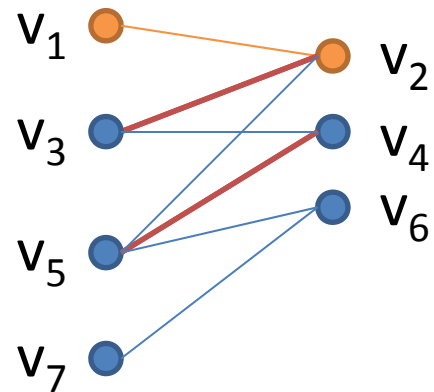
- 举例



从 $\{v_1, v_7\}$ 中未搜索过的 v_1 开始搜索

面向二部图的增广路算法 (续)

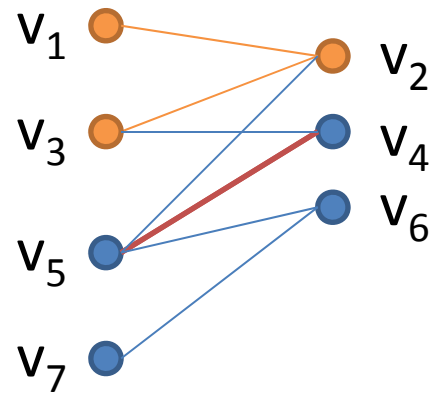
- 举例



沿不在当前匹配中且本轮未搜索过的(v_1, v_2)到达 $v_2 \Rightarrow v_2$ 已被当前匹配中的(v_3, v_2)饱和和

面向二部图的增广路算法 (续)

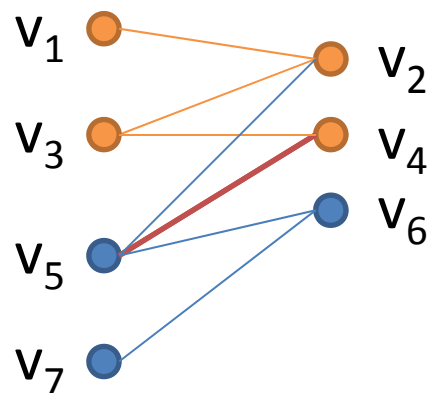
- 举例



沿当前匹配中的 (v_3, v_2) 到达 v_3

面向二部图的增广路算法 (续)

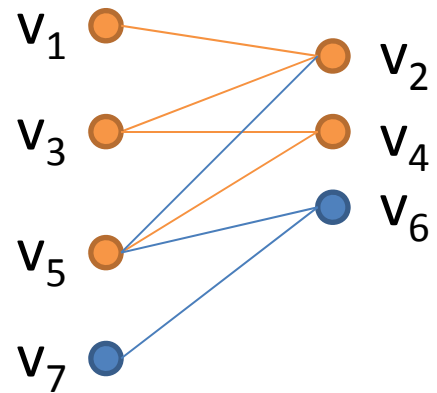
- 举例



沿不在当前匹配中且本轮未搜索过的(v_3, v_4)到达 $v_4 \Rightarrow v_4$ 已被当前匹配中的(v_5, v_4)饱和和

面向二部图的增广路算法 (续)

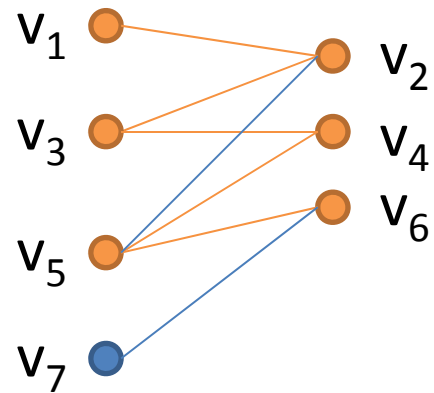
- 举例



沿当前匹配中的 (v_5, v_4) 到达 v_5

面向二部图的增广路算法 (续)

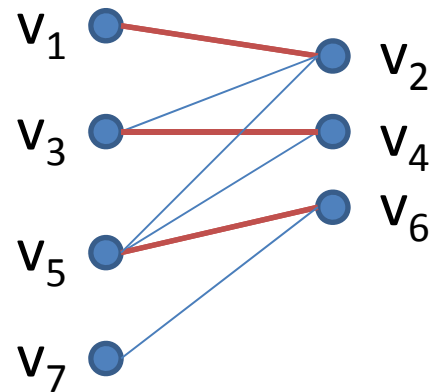
- 举例



沿不在当前匹配中且本轮未搜索过的(v_5, v_6)到达 $v_6 \Rightarrow v_6$ 未饱和

面向二部图的增广路算法 (续)

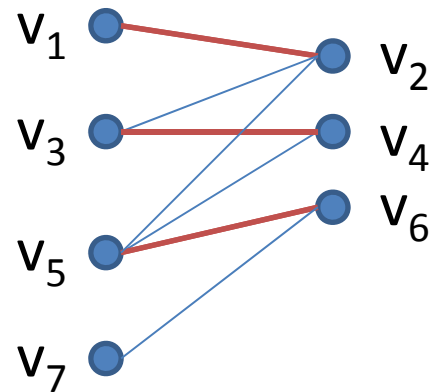
- 举例



找到增广路 $v_1v_2v_3v_4v_5v_6 \Rightarrow$ 替换进当前匹配中 \Rightarrow 本轮搜索结束

面向二部图的增广路算法 (续)

- 举例

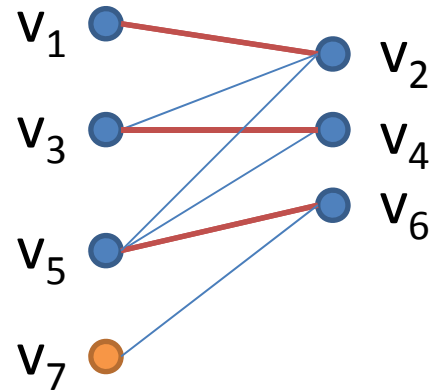


第四轮搜索开始

- 当前匹配: $\{(v_1, v_2), (v_3, v_4), (v_5, v_6)\}$
- 未饱和的左侧顶点: $\{v_7\}$

面向二部图的增广路算法 (续)

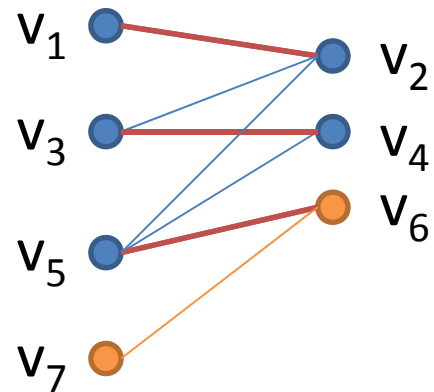
- 举例



从 $\{v_7\}$ 中未搜索过的 v_7 开始搜索

面向二部图的增广路算法 (续)

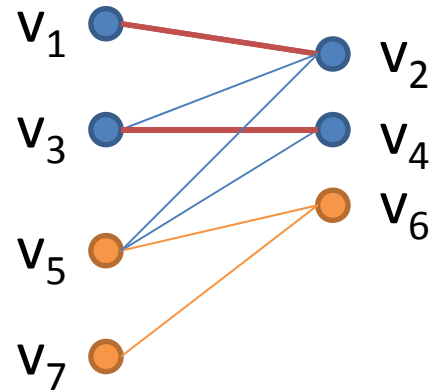
- 举例



沿不在当前匹配中且本轮未搜索过的(v_7, v_6)到达 $v_6 \Rightarrow v_6$ 已被当前匹配中的(v_5, v_6)饱和和

面向二部图的增广路算法 (续)

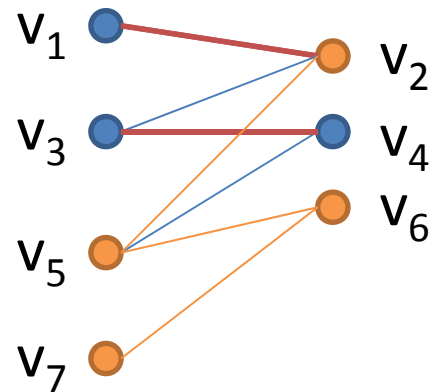
- 举例



沿当前匹配中的 (v_5, v_6) 到达 v_5

面向二部图的增广路算法 (续)

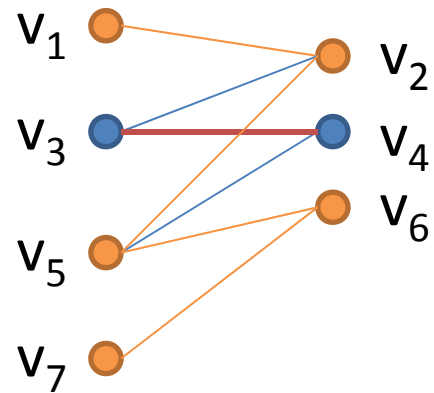
- 举例



沿不在当前匹配中且本轮未搜索过的 (v_5, v_2) 到达 $v_2 \Rightarrow v_2$ 已被当前匹配中的 (v_1, v_2) 饱和和

面向二部图的增广路算法 (续)

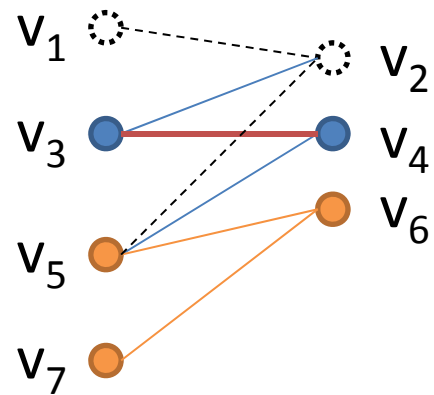
- 举例



沿当前匹配中的 (v_1, v_2) 到达 v_1

面向二部图的增广路算法 (续)

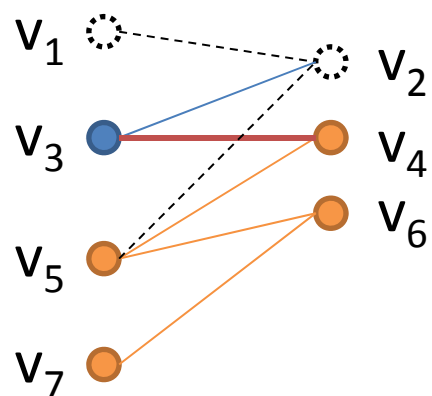
- 举例



v_1 没有关联到任何不在当前匹配中且本轮未搜索过的边 \Rightarrow 回溯到 v_5 (为什么不是 v_2)

面向二部图的增广路算法 (续)

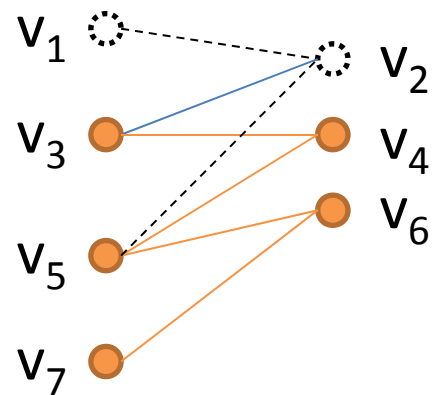
- 举例



沿不在当前匹配中且本轮未搜索过的(v_5 , v_4)到达 $v_4 \Rightarrow v_4$ 已被当前匹配中的(v_3 , v_4)饱和和

面向二部图的增广路算法 (续)

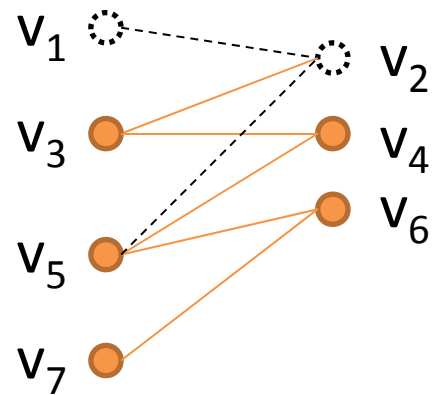
- 举例



沿当前匹配中的 (v_3, v_4) 到达 v_3

面向二部图的增广路算法 (续)

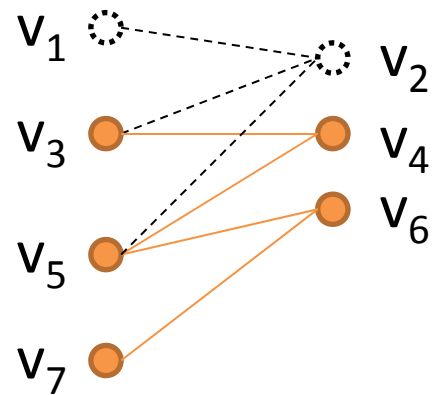
- 举例



沿不在当前匹配中且本轮未搜索过的(v_3, v_2)到达 v_2

面向二部图的增广路算法 (续)

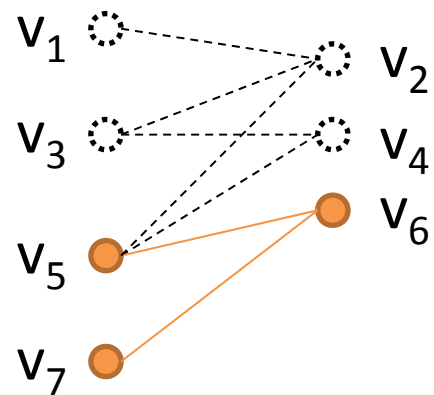
- 举例



v_2 本轮已搜索过 \Rightarrow 回溯到 v_3

面向二部图的增广路算法 (续)

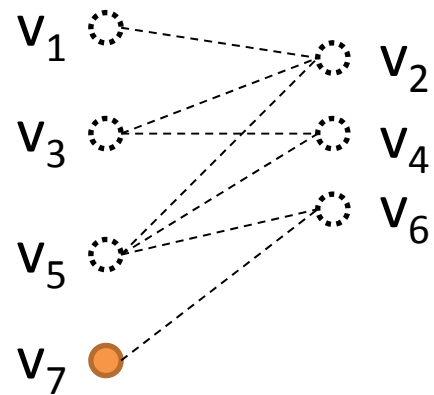
- 举例



v_3 没有关联到任何不在当前匹配中且本轮未搜索过的边 \Rightarrow 回溯到 v_5

面向二部图的增广路算法 (续)

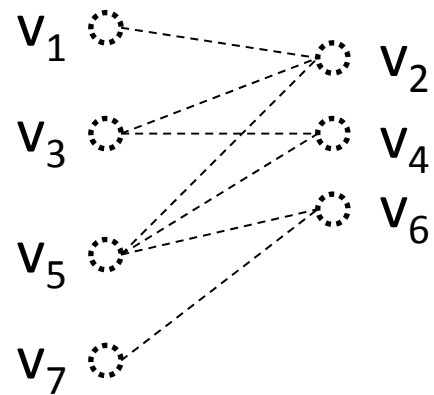
- 举例



v_5 没有关联到任何不在当前匹配中且本轮未搜索过的边 \Rightarrow 回溯到 v_7

面向二部图的增广路算法 (续)

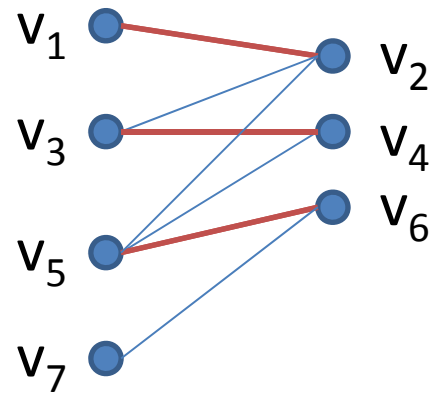
- 举例



v_7 没有关联到任何不在当前匹配中且本轮未搜索过的边 \Rightarrow 回溯到头了!

面向二部图的增广路算法 (续)

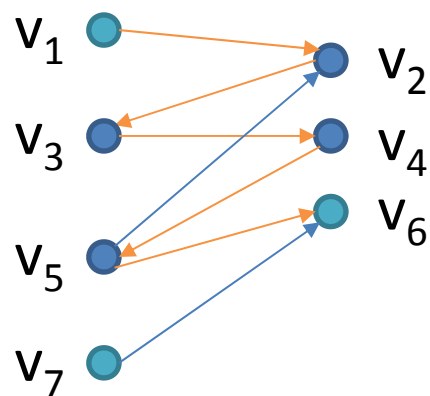
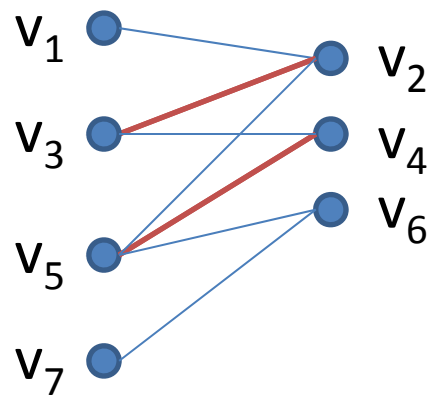
- 举例



已走过所有的点和边 \Rightarrow 无增广路 \Rightarrow 找到最大匹配

面向二部图的增广路算法 (续)

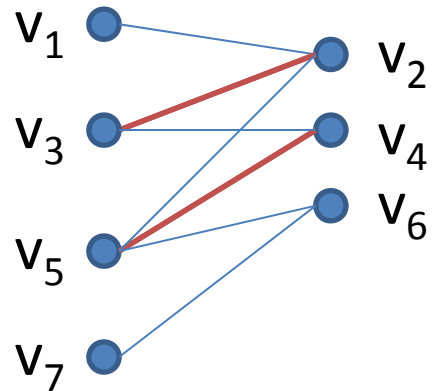
- 算法的正确性（要证明的结论是什么？）
 - 算法一定会终止（为什么？）
 - 每轮搜索一定会结束（为什么？）
 - 轮数一定是有限的（为什么？）
 - 算法一旦终止，找到的一定是最大匹配（为什么？）
 - 存在增广路 \Rightarrow 一定能找到
 - 找不到增广路 \Rightarrow 无增广路 \Rightarrow 找到最大匹配



实现技巧

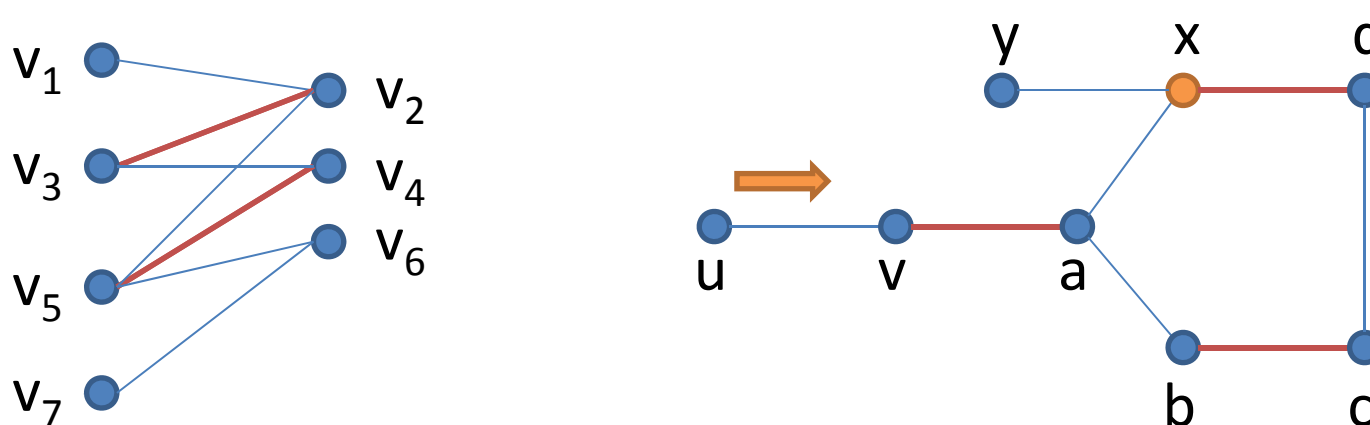
面向二部图的增广路算法 (续)

- 算法的运行时间
 - 搜索的最大轮数?
 - $O(v)$
 - 每轮搜索的最大步骤数?
 - $O(v+\epsilon)$



面向二部图的增广路算法 (续)

- 为什么增广路算法只能用于二部图？
 - 直观上
 - 二部图中，可以基于左右两侧为边定向
 - 一般图中，没有两侧的概念，不好定向
 - 本质上
 - 二部图中，每个左侧顶点在每一轮搜索中只需搜索一次（作为起点，或当前匹配中一条边的终点）
 - 一般图中，可能有奇圈，第一次到达其中的“左侧”顶点可能是错误地作为不在当前匹配中的一条边的终点，即错误地将其当作“右侧”顶点了

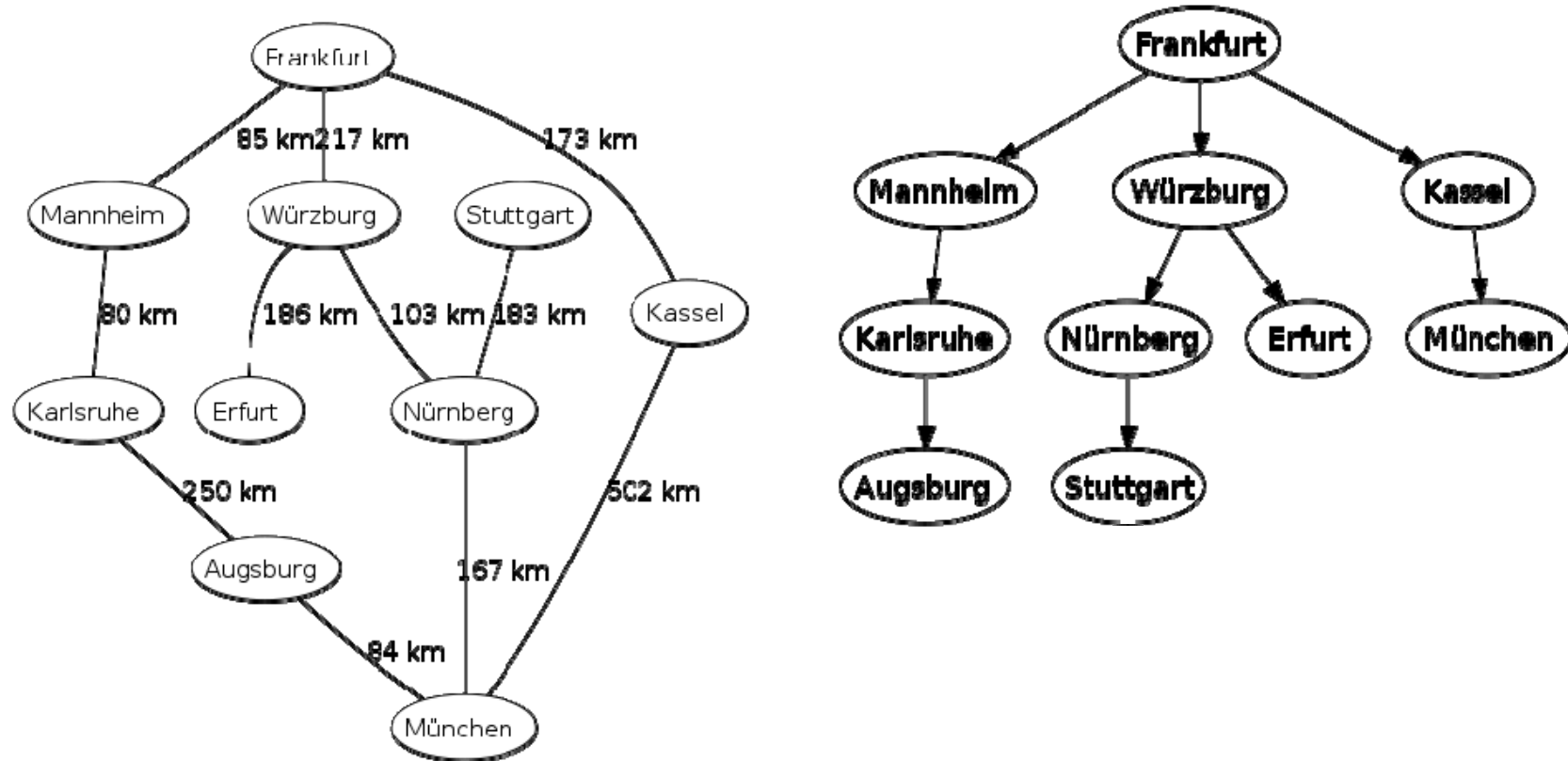


面向二部图的Hopcroft-Karp算法*

- 基本思路（在增广路算法的基础上）
 - 总是搜索最短的增广路
 - 每轮搜索多条无公共顶点的增广路，全部替换

面向二部图的Hopcroft-Karp算法 (续)

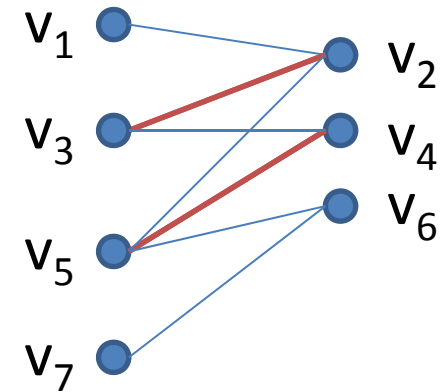
- 广度优先搜索 (复习)



面向二部图的Hopcroft-Karp算法 (续)

每轮搜索的步骤

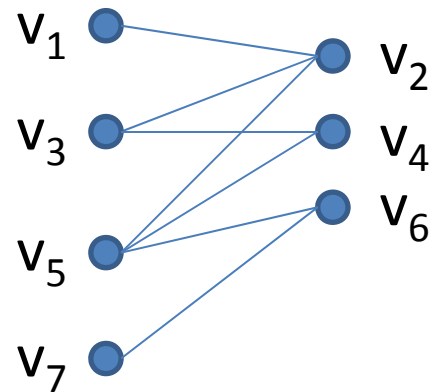
1. 确定最短增广路的潜在起点
 - 所有未饱和的左侧顶点
2. 确定最短增广路的长度与潜在终点
 - 利用广度优先搜索，对所有顶点进行分层
 - 上述所有潜在起点构成第0层
 - 第i层相邻的所有未分层顶点构成第i+1层
 - i为偶数时，只能通过不在当前匹配中的边关联
 - i为奇数时，只能通过当前匹配中的边关联
 - 搜索终止于第k层的条件：第k层包含未饱和的右侧顶点，或已搜完所有顶点
3. 搜索极多的无公共顶点的最短增广路，全部替换得到更大的匹配
 - 在分层信息的引导下，利用反向的深度优先搜索降回第0层
 - 每找到一条增广路，将其经过的顶点及其关联的边从图中临时删除



算法的终止条件：找不到增广路 \Rightarrow 无增广路 \Rightarrow 找到最大匹配

面向二部图的Hopcroft-Karp算法 (续)

- 举例

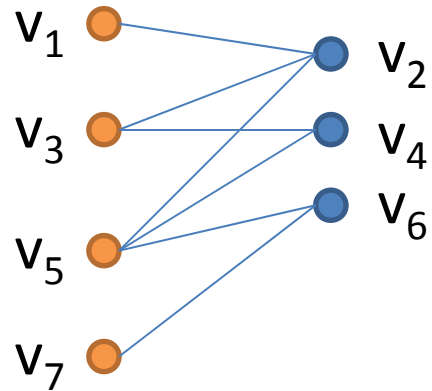


第一轮搜索开始

- 当前匹配: $\{\}$
- 未饱和的左侧顶点: $\{v_1, v_3, v_5, v_7\}$

面向二部图的Hopcroft-Karp算法 (续)

- 举例

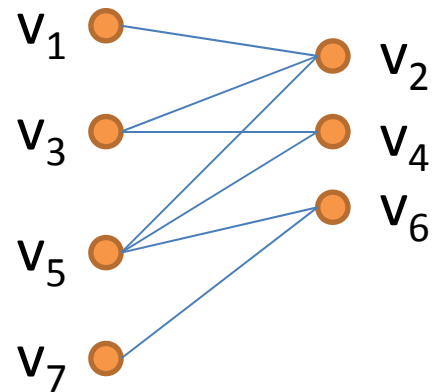


利用广度优先搜索分层

- 第0层 $\{v_1, v_3, v_5, v_7\}$

面向二部图的Hopcroft-Karp算法 (续)

- 举例

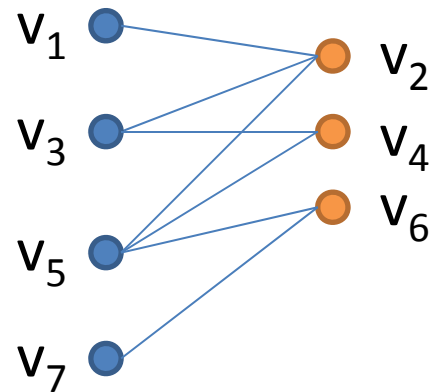


利用广度优先搜索分层

- 第0层 $\{v_1, v_3, v_5, v_7\}$
- 沿不在当前匹配中的边到达第1层 $\{v_2, v_4, v_6\}$

面向二部图的Hopcroft-Karp算法 (续)

- 举例

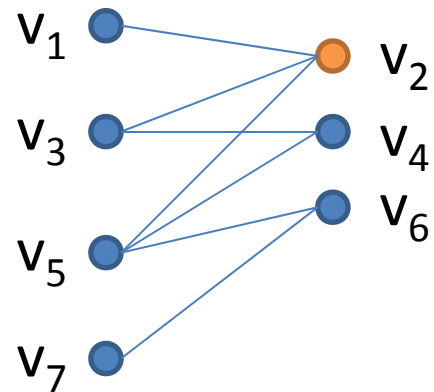


利用广度优先搜索分层

- 第0层{v₁, v₃, v₅, v₇}
- 沿不在当前匹配中的边到达第1层{v₂, v₄, v₆}
- 发现未饱和的右侧顶点{v₂, v₄, v₆}

面向二部图的Hopcroft-Karp算法 (续)

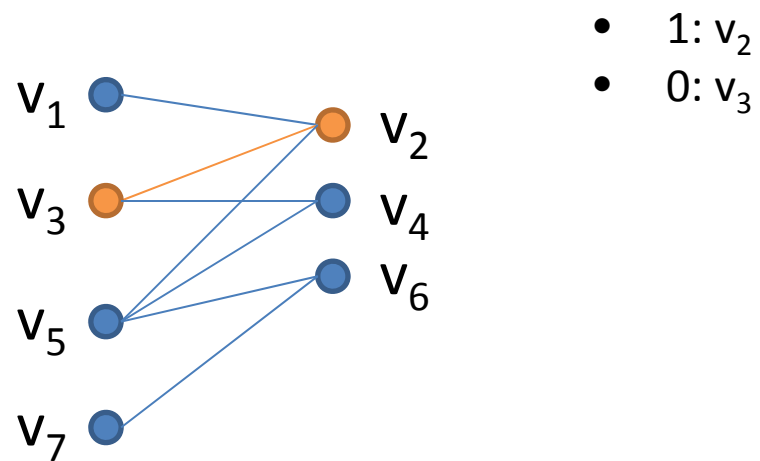
- 举例



从未饱和的右侧顶点 v_2 （位于第1层）开始反向降层的深度优先搜索

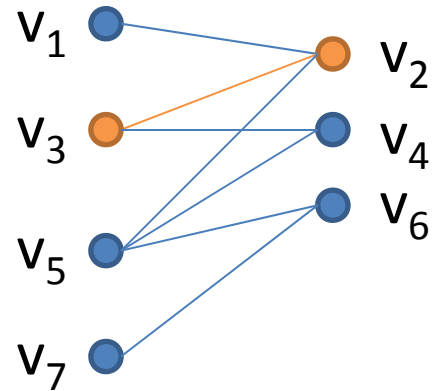
面向二部图的Hopcroft-Karp算法 (续)

- 举例



面向二部图的Hopcroft-Karp算法 (续)

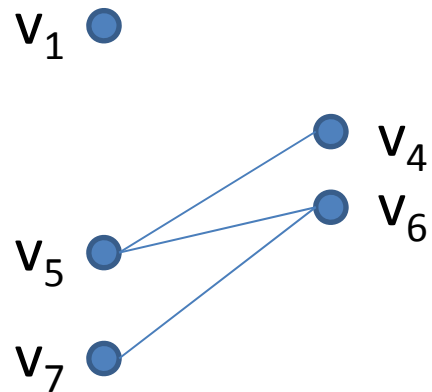
- 举例



找到增广路 $v_2v_3 \Rightarrow$ 替换进当前匹配中

面向二部图的Hopcroft-Karp算法 (续)

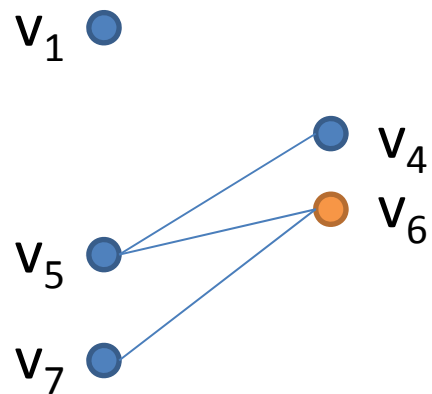
- 举例



临时删除 v_2 、 v_3 及其关联的边

面向二部图的Hopcroft-Karp算法 (续)

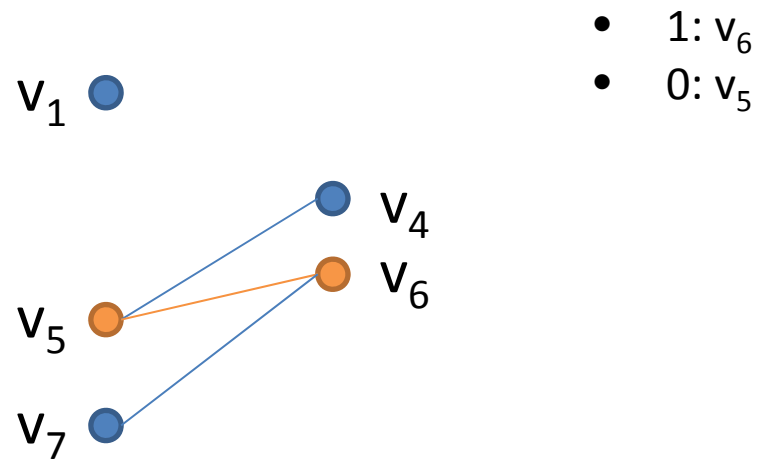
- 举例



从未饱和的右侧顶点 v_6 （位于第1层）开始反向降层的深度优先搜索

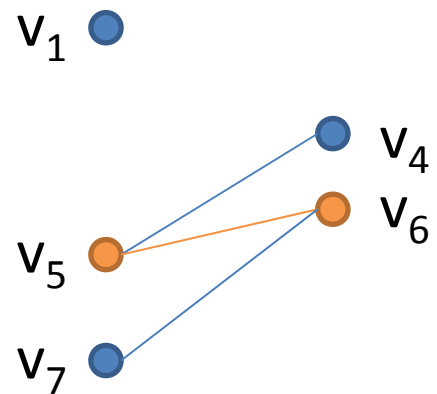
面向二部图的Hopcroft-Karp算法 (续)

- 举例



面向二部图的Hopcroft-Karp算法 (续)

- 举例



找到增广路 $v_6v_5 \Rightarrow$ 替换进当前匹配中

面向二部图的Hopcroft-Karp算法 (续)

- 举例

v_1 ●

● v_4

v_7 ●

临时删除 v_6 、 v_5 及其关联的边

面向二部图的Hopcroft-Karp算法 (续)

- 举例

v_1 ●

● v_4

v_7 ●

从未饱和的右侧顶点 v_4 （位于第1层）开始反向降层的深度优先搜索

面向二部图的Hopcroft-Karp算法 (续)

- 举例

v_1 ●

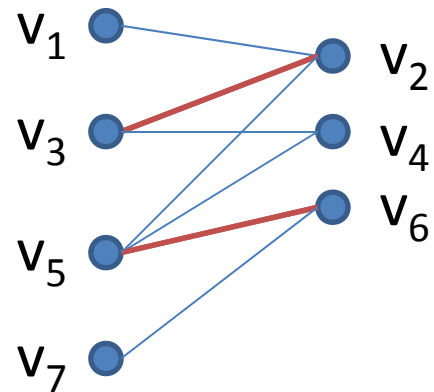
● v_4

v_7 ●

无法下降到第0层，放弃 v_4

面向二部图的Hopcroft-Karp算法 (续)

- 举例

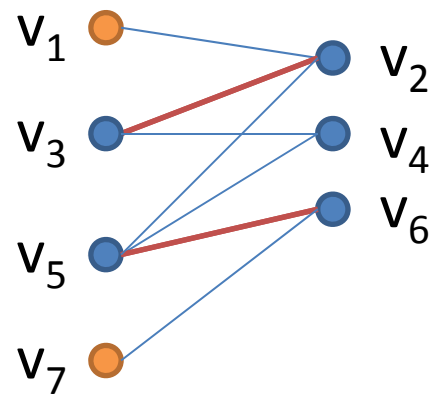


第二轮搜索开始

- 当前匹配: $\{(v_3, v_2), (v_5, v_6)\}$
- 未饱和的左侧顶点: $\{v_1, v_7\}$

面向二部图的Hopcroft-Karp算法 (续)

- 举例

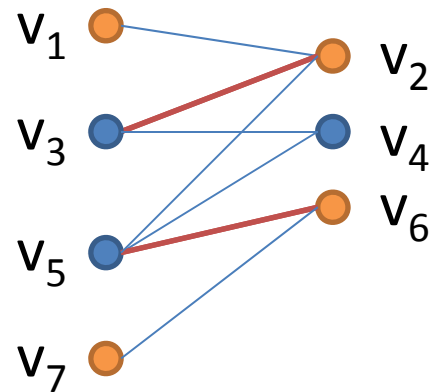


利用广度优先搜索分层

- 第0层 $\{v_1, v_7\}$

面向二部图的Hopcroft-Karp算法 (续)

- 举例

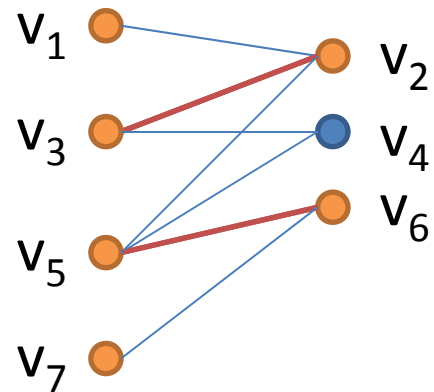


利用广度优先搜索分层

- 第0层 $\{v_1, v_7\}$
- 沿不在当前匹配中的边到达第1层 $\{v_2, v_6\}$

面向二部图的Hopcroft-Karp算法 (续)

- 举例

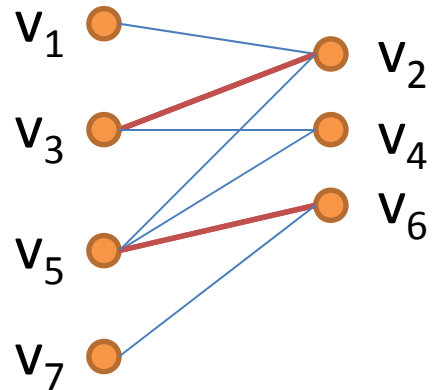


利用广度优先搜索分层

- 第0层 $\{v_1, v_7\}$
- 沿不在当前匹配中的边到达第1层 $\{v_2, v_6\}$
- 沿当前匹配中的边到达第2层 $\{v_3, v_5\}$

面向二部图的Hopcroft-Karp算法 (续)

- 举例

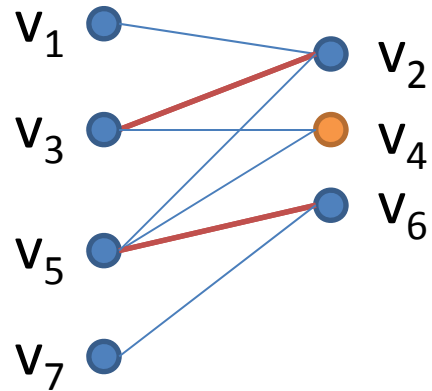


利用广度优先搜索分层

- 第0层 $\{v_1, v_7\}$
- 沿不在当前匹配中的边到达第1层 $\{v_2, v_6\}$
- 沿当前匹配中的边到达第2层 $\{v_3, v_5\}$
- 沿不在当前匹配中的边到达第3层 $\{v_4\}$

面向二部图的Hopcroft-Karp算法 (续)

- 举例

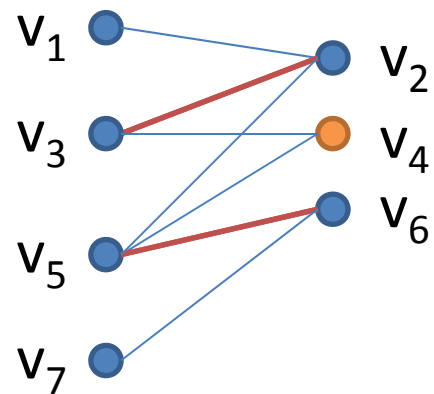


利用广度优先搜索分层

- 第0层 $\{v_1, v_7\}$
- 沿不在当前匹配中的边到达第1层 $\{v_2, v_6\}$
- 沿当前匹配中的边到达第2层 $\{v_3, v_5\}$
- 沿不在当前匹配中的边到达第3层 $\{v_4\}$
- 发现未饱和的右侧顶点 $\{v_4\}$

面向二部图的Hopcroft-Karp算法 (续)

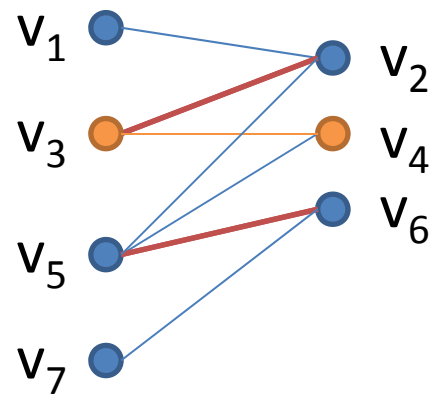
- 举例



从未饱和的右侧顶点 v_4 （位于第3层）开始反向降层的深度优先搜索

面向二部图的Hopcroft-Karp算法 (续)

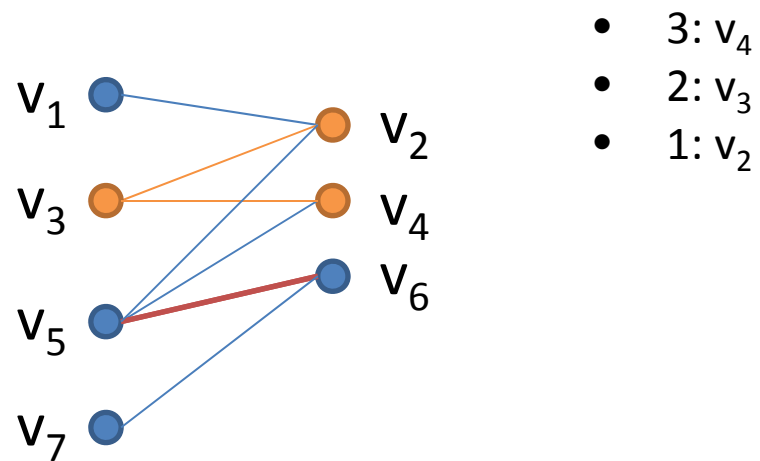
- 举例



- 3: v_4
- 2: v_3

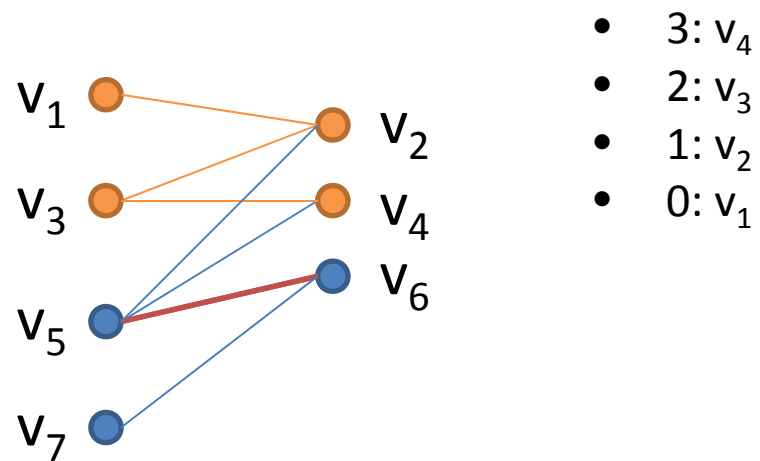
面向二部图的Hopcroft-Karp算法 (续)

- 举例



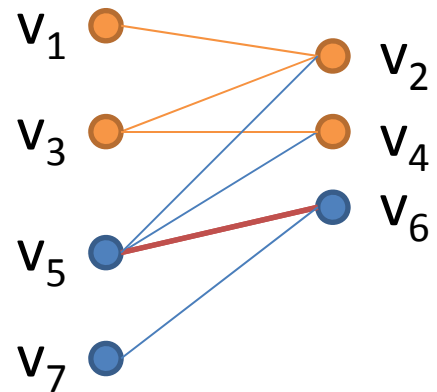
面向二部图的Hopcroft-Karp算法 (续)

- 举例



面向二部图的Hopcroft-Karp算法 (续)

- 举例

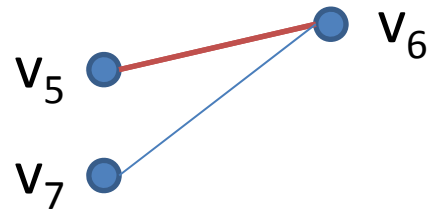


找到增广路 $v_4v_3v_2v_1 \Rightarrow$ 替换进当前匹配中

面向二部图的Hopcroft-Karp算法 (续)

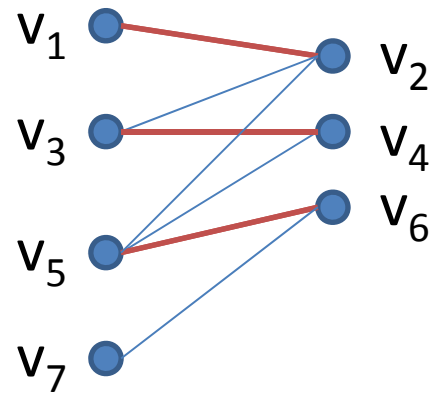
- 举例

临时删除 v_4 、 v_3 、 v_2 、 v_1 及其关联的边



面向二部图的Hopcroft-Karp算法 (续)

- 举例

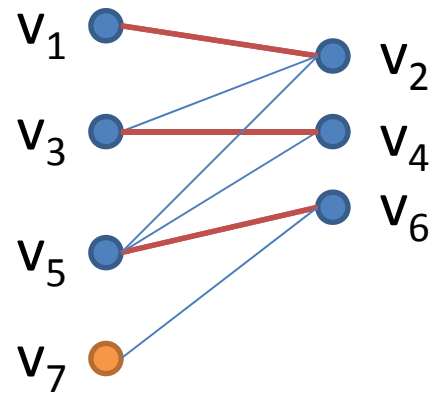


第三轮搜索开始

- 当前匹配: $\{(v_1, v_2), (v_3, v_4), (v_5, v_6)\}$
- 未饱和的左侧顶点: $\{v_7\}$

面向二部图的Hopcroft-Karp算法 (续)

- 举例

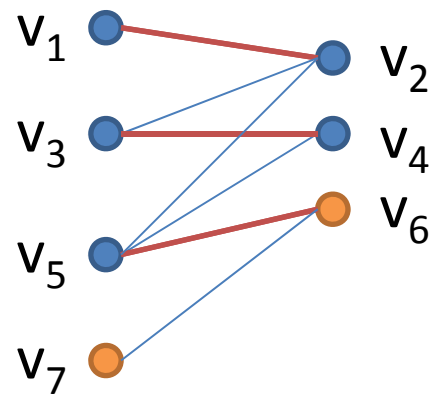


利用广度优先搜索分层

- 第0层 $\{v_7\}$

面向二部图的Hopcroft-Karp算法 (续)

- 举例

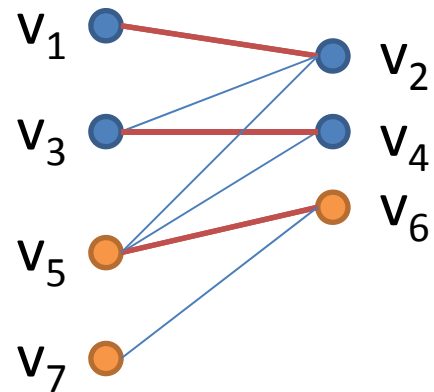


利用广度优先搜索分层

- 第0层 $\{v_7\}$
- 沿不在当前匹配中的边到达第1层 $\{v_6\}$

面向二部图的Hopcroft-Karp算法 (续)

- 举例

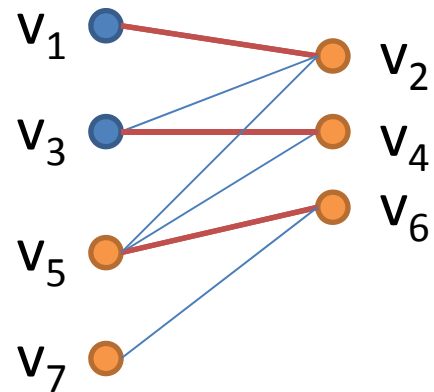


利用广度优先搜索分层

- 第0层 $\{v_7\}$
- 沿不在当前匹配中的边到达第1层 $\{v_6\}$
- 沿当前匹配中的边到达第2层 $\{v_5\}$

面向二部图的Hopcroft-Karp算法 (续)

- 举例

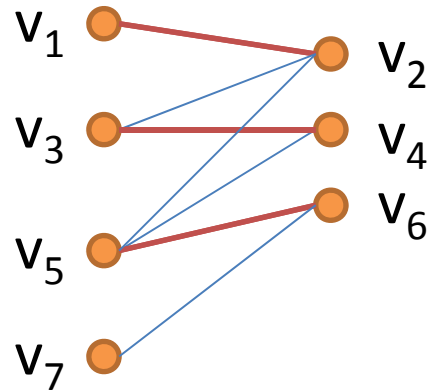


利用广度优先搜索分层

- 第0层{v7}
- 沿不在当前匹配中的边到达第1层{v6}
- 沿当前匹配中的边到达第2层{v5}
- 沿不在当前匹配中的边到达第3层{v2、v4}

面向二部图的Hopcroft-Karp算法 (续)

- 举例

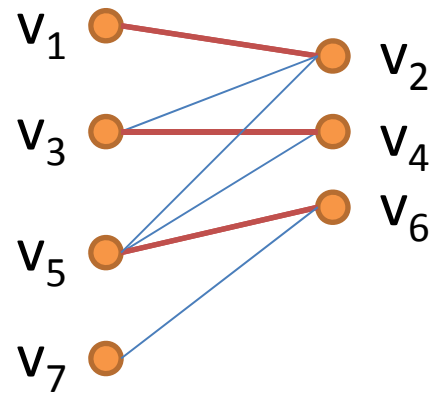


利用广度优先搜索分层

- 第0层 $\{v_7\}$
- 沿不在当前匹配中的边到达第1层 $\{v_6\}$
- 沿当前匹配中的边到达第2层 $\{v_5\}$
- 沿不在当前匹配中的边到达第3层 $\{v_2, v_4\}$
- 沿当前匹配中的边到达第4层 $\{v_1, v_3\}$

面向二部图的Hopcroft-Karp算法 (续)

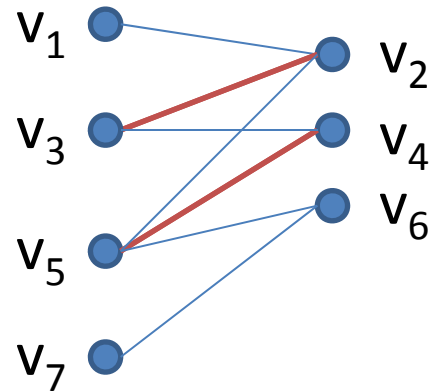
- 举例



已搜完所有顶点，未发现未饱和的右侧顶点
⇒ 无增广路 ⇒ 找到最大匹配

面向二部图的Hopcroft-Karp算法 (续)

- 算法的正确性
 - 算法一定会终止
 - 算法一旦终止，找到的一定是最大匹配



面向二部图的Hopcroft-Karp算法 (续)

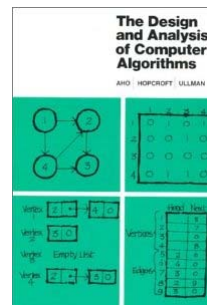
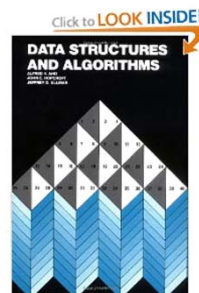
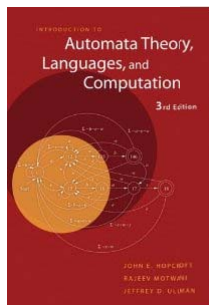
- 算法的运行时间*
 - 引理1: 如果每次都选取最短的增广路替换进当前匹配中, 那么这些增广路的长度是非严格单调增的。
 - 引理2: 上述过程中, 相同长度的增广路无公共顶点。
 - \Rightarrow 算法每轮找到的最短增广路都比上一轮找到的长
 - 引理3: 上述过程中, 增广路最多只有 $2 \lfloor \sqrt{\frac{v}{2}} \rfloor + 2$ 种不同的长度
 - \Rightarrow 算法最多搜索 $O(\sqrt{v})$ 轮
 - 结论
 - 搜索的最大轮数: $O(\sqrt{v})$
 - 每轮搜索的最大步骤数: $O(v + \epsilon)$



John Edward Hopcroft, 美国, 1939--



Richard Manning Karp, 美国, 1935--



Karp's 21 NP-complete problems

<http://upload.wikimedia.org/wikipedia/commons/thumb/6/6a/Hopcrofg.jpg/150px-Hopcrofg.jpg>

http://upload.wikimedia.org/wikipedia/commons/thumb/3/3e/Karp_mg_7725-b.cr2.jpg/220px-Karp_mg_7725-b.cr2.jpg



John Edward Hopcroft, 美国, 1939--



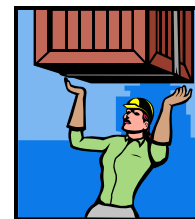
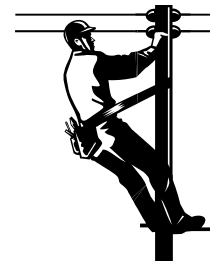
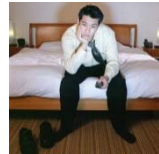
Richard Manning Karp, 美国, 1935--

他们至今仍工作在科研的第一线

<http://upload.wikimedia.org/wikipedia/commons/thumb/6/6a/Hopcrofg.jpg/150px-Hopcrofg.jpg>

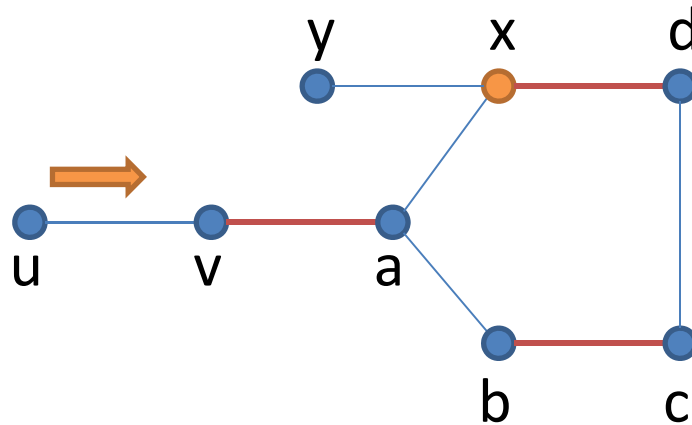
http://upload.wikimedia.org/wikipedia/commons/thumb/3/3e/Karp_mg_7725-b.cr2.jpg/220px-Karp_mg_7725-b.cr2.jpg

二部图的最大权匹配（分配/指派问题）



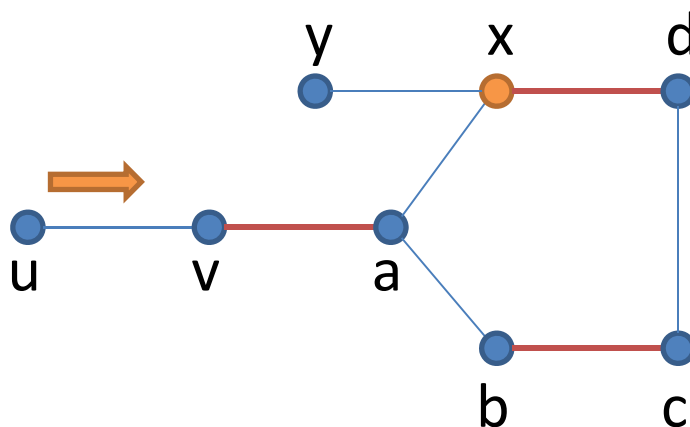
面向一般图的Edmonds算法*

- 目标：解决在一般图上使用增广路算法时，由奇圈带来的问题



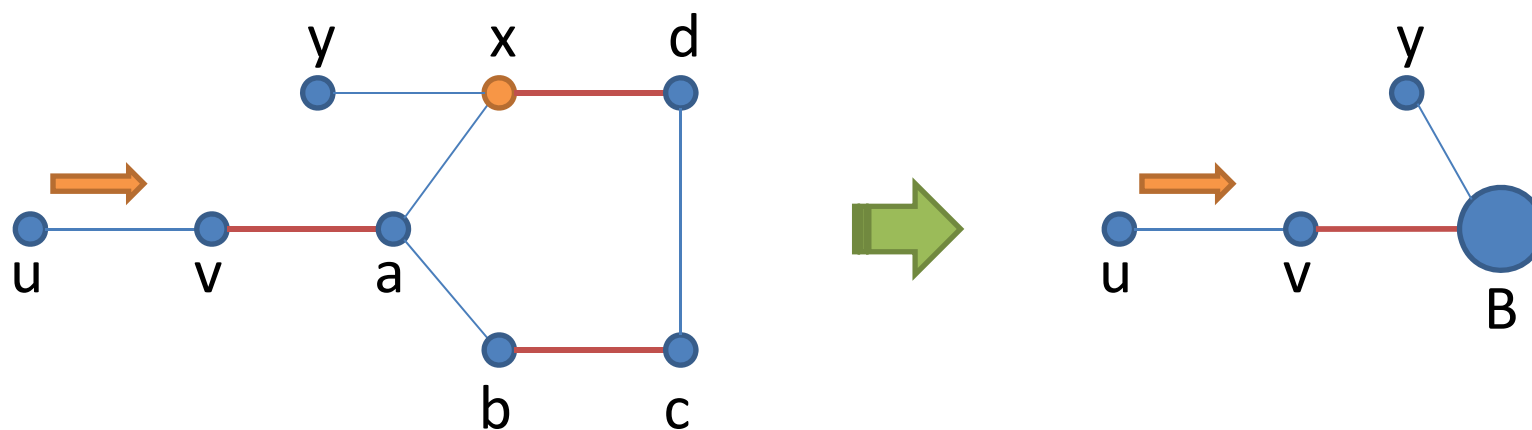
面向一般图的Edmonds算法 (续)

- 基本思路（在增广路算法的基础上）
 - 在一轮搜索中，如果经过长为奇数的交错路到达的一个顶点在本轮搜索中已经经过长为偶数的交错路到达过，那么就发现了一个奇圈，这两条路的并称作**flower**，这个奇圈称作**blossom**
 - 两条交错路的最长公共子路称作**stem**，它的终点称作**base**



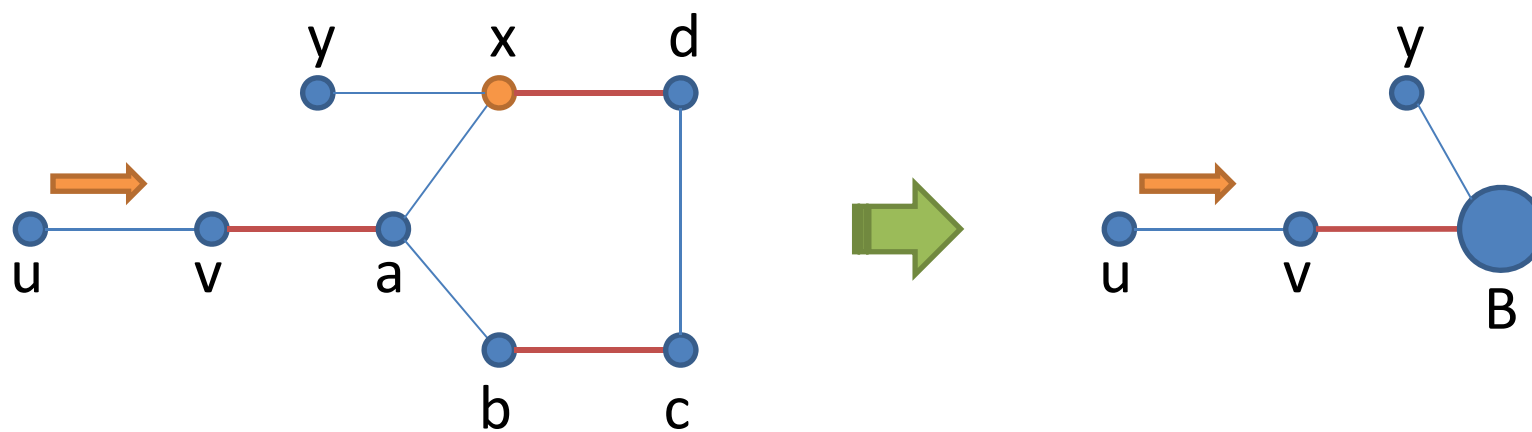
面向一般图的Edmonds算法 (续)

- 基本思路（在增广路算法的基础上）
 - stem的最后一条边是饱和的（为什么？）
blossom中的顶点关联的边中，除blossom中的边和stem的最后一条边以外，其它都未饱和（为什么？）
 - 将blossom收缩为一个顶点：顶点合并，内部边删除、外部边保留（保持饱和/未饱和状态）
 - 如果新图中有增广路，那么原图中一定有增广路（为什么？）



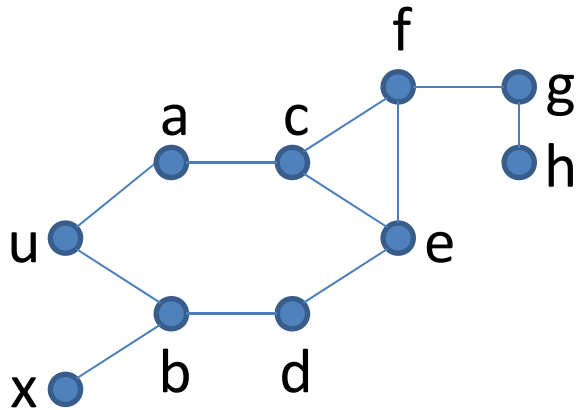
面向一般图的Edmonds算法 (续)

- 关键步骤（在增广路算法的基础上）
 - 在搜索的过程中，一旦发现奇圈，就将其收缩为一个顶点，再继续搜索
 - 如果新图中的增广路经过收缩后的顶点，那么利用奇圈中两条交错路之一将其还原到原图中的增广路



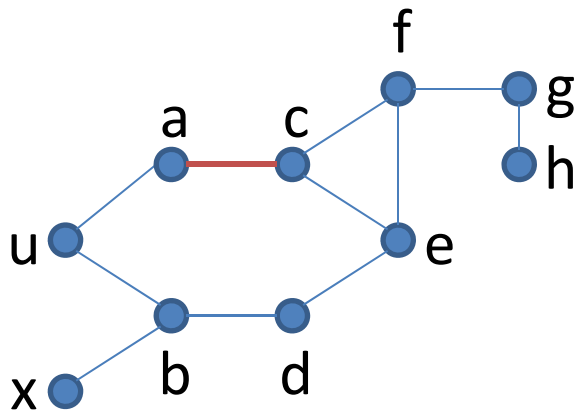
面向一般图的Edmonds算法 (续)

- 举例



面向一般图的Edmonds算法 (续)

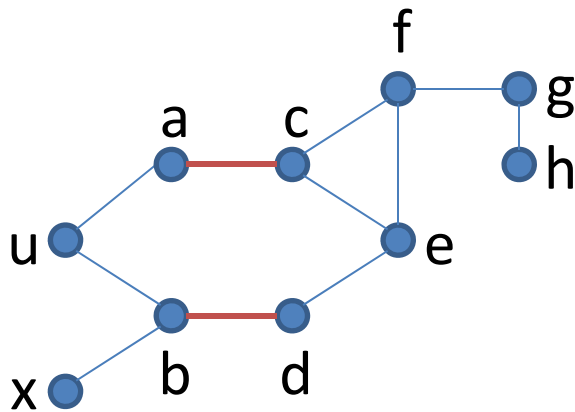
- 举例



从未饱和的a开始搜索，找到增广路ac

面向一般图的Edmonds算法 (续)

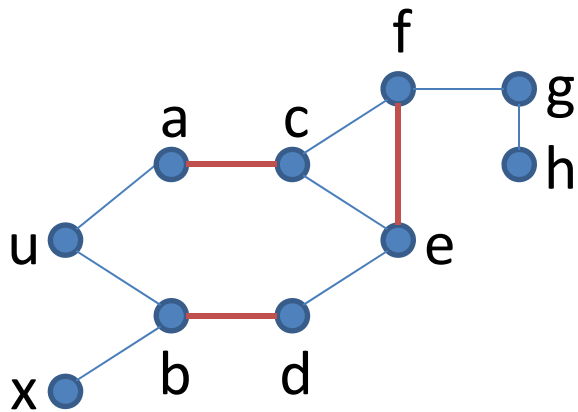
- 举例



从未饱和的b开始搜索，找到增广路bd

面向一般图的Edmonds算法 (续)

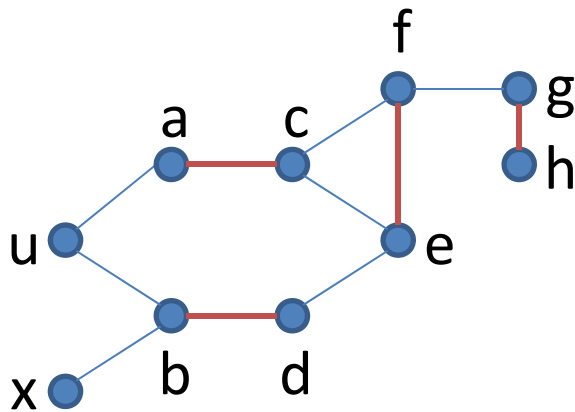
- 举例



从未饱和的e开始搜索，找到增广路ef

面向一般图的Edmonds算法 (续)

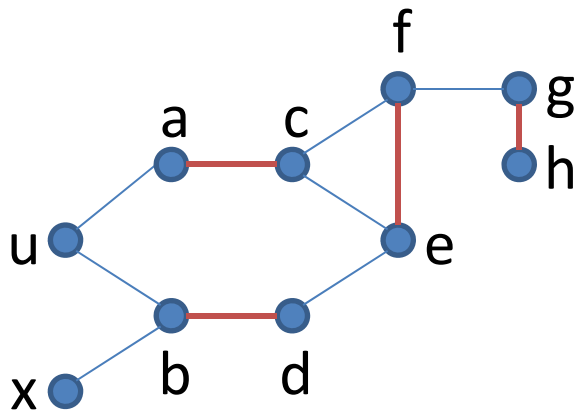
- 举例



从未饱和的g开始搜索，找到增广路gh

面向一般图的Edmonds算法 (续)

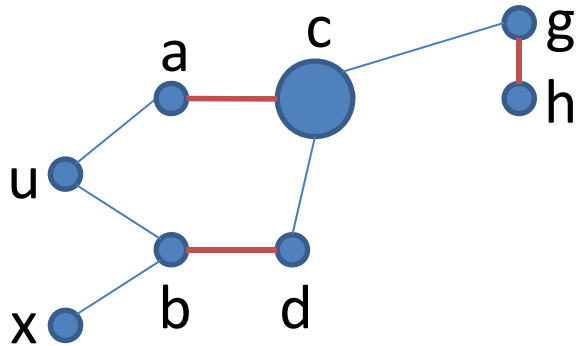
- 举例



从未饱和的u开始搜索，沿交错路uacfec，此时，长为5的交错路到达了之前长为2的交错路到达过的顶点c，发现flower，其中：blossom为cfe，stem为uac

面向一般图的Edmonds算法 (续)

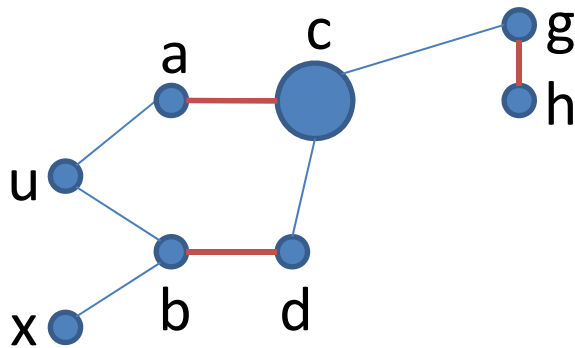
- 举例



收缩cfe为C，继续搜索

面向一般图的Edmonds算法 (续)

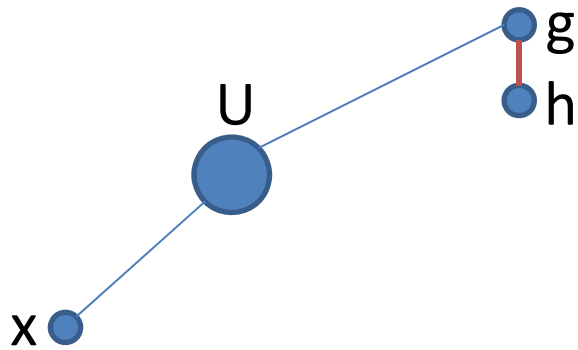
- 举例



沿交错路 $uaCdbu$ ，此时，长为5的交错路到达了之前长为0的交错路到达过的顶点 u ，发现flower，其中：blossom为 $uaCdb$ ，stem为空

面向一般图的Edmonds算法 (续)

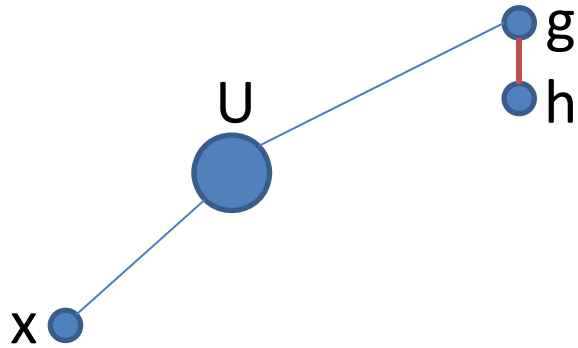
- 举例



收缩 $uaCdb$ 为 U ，继续搜索

面向一般图的Edmonds算法 (续)

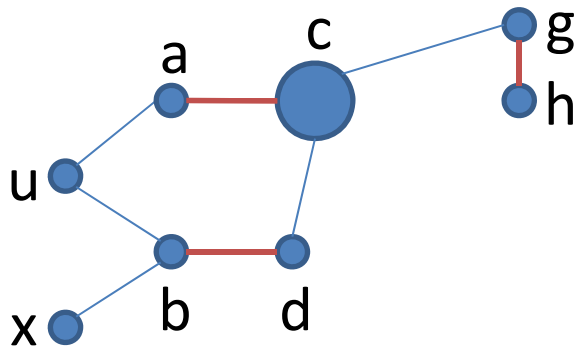
- 举例



找到增广路 Ux ，涉及收缩顶点 U

面向一般图的Edmonds算法 (续)

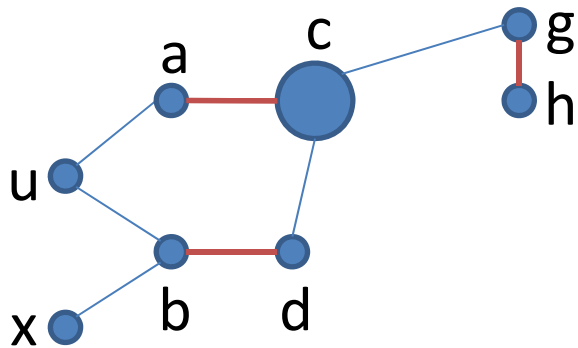
- 举例



将U还原，从关联到x的不在当前匹配中的边bx起，延交错路uaCdbx退回到base u

面向一般图的Edmonds算法 (续)

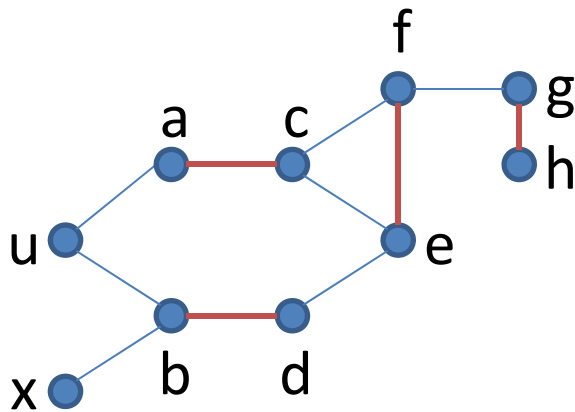
- 举例



找到增广路 $uaCdbx$ ，涉及收缩顶点C

面向一般图的Edmonds算法 (续)

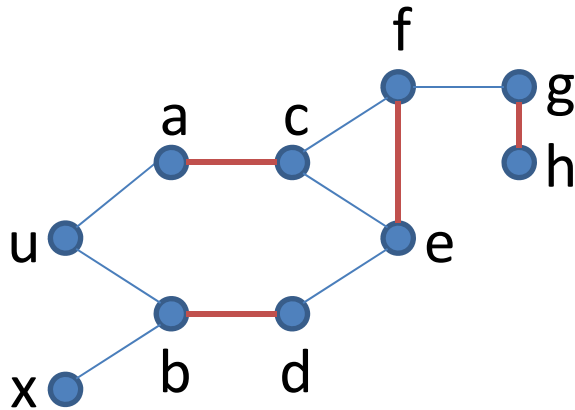
- 举例



将c还原，从关联到d的不在当前匹配中的边ed起，延交错路cfe退回到base c

面向一般图的Edmonds算法 (续)

- 举例



找到增广路uacfedbx，不涉及收缩顶点，
替换进当前匹配中，本轮搜索结束

面向一般图的Edmonds算法 (续)

- 算法的运行时间
 - 朴素的实现: $O(v^4)$
 - 用合适的数据结构表示blossom和处理收缩: $O(v^3)$
- 一般图最大匹配的其它算法: $O(\sqrt{v}\epsilon)$
 - 与面向二部图的Hopcroft-Karp算法相当



Jack R. Edmonds, 美国, 1934--

作业：编程实现增广路算法

- 采用标准输入/输出
- 输入格式：
 - 第一行：整数 n ，表示顶点个数
 - 后续每行：两个整数 a, b ($0 \leq a, b \leq n-1$)，表示顶点 v_a 与 v_b 相邻
 - $a=b=-1$ 表示输入结束
 - 输入可以保证是二部图
- 输出格式：
 - 第一行：整数 m ，表示匹配 M 中包含的边数
 - 后续 m 行： M 中的每条边 e_i ，按照以下格式：
 - 每行为 e_i 的两个端点标号 $e_i.a, e_i.b$ ($e_i.a < e_i.b$)
 - 各行顺序：按 $e_i.a$ 升序
- 源代码发到：gcheng@nju.edu.cn
- Deadline：5月7日中午12:00
- 在总成绩中占10分
 - 雷同的得-10分（包括：与其他同学代码雷同、与往届同学代码雷同、与网上代码雷同）
 - 迟交的分数减半（最高得5分）