基本概念/翻译

- closed trail：闭迹
- circuit：回路（本书定义：长度不小于3的闭迹）
- cycle：圈（除头尾外，顶点不重复的回路）

- loop：环

- 没有“循环”这种翻译

- 作业讲解
  - GC第5.1节练习3、4、6
  - GC第5.2节练习10、11、14
  - GC第5.3节练习20、22、30
  - GC第6.1节练习4、5、6
  - GC第6.2节练习13、16、21

GC第5.2节练习10

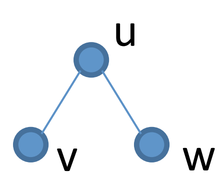
5.10 Prove that a connected graph G of size at least 2 is nonseparable if and only if any two adjacent edges of G lie on a common cycle of G.

$\Rightarrow$

- u不是割点 → G-u中有v-w路P → wu, uv, P形成圈

$\Leftarrow$

- 假设存在割点u，使v, w在G-u中不连通
- uv, uw共圈 → G-u中有v-w路 → 矛盾



GC第5.2节练习14

- block一定没有割边吗？
- 看清题意：need not be a block

GC第5.2节练习20

5.20 Let G be a connected graph of order $n = 4$ and let k be an integer with $2 \leq k \leq n - 2$.

(a) Prove that if G is not k-connected, then G contains a vertex-cut U with $|U| = k - 1$.

- G不是k连通 → $\kappa(G) \leq k-1$ → G有大小为k-1的点割集
  这样对吗？

GC第5.2节练习20

5.20 Let G be a connected graph of order $n = 4$ and let k be an integer with $2 \le \underline{k \le n-2}$.

(a) Prove that if G is not k-connected, then G contains a vertex-cut U with $|U| = k-1$.

- G不是k连通 → κ(G)≤k-1 → G有大小为k-1的点割集
  这样对吗？

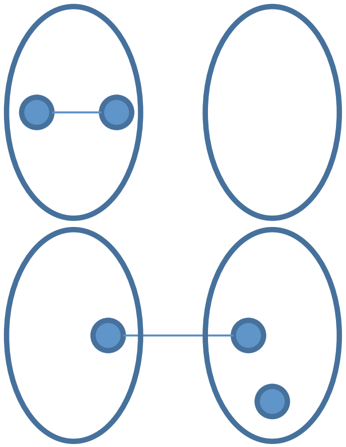
- G不是k连通 → κ(G)≤k-1 → G有大小不超过k-1的点割集 → 如果该点割集大小不到k-1，可以将图中剩余点加入直至大小恰为k-1（为什么一定能做到这一点？怎么做？）

GC第5.2节练习22

5.22 (a) Prove that if G is a k-connected graph and e is an edge of G, then $G-e$ is $(k-1)$-connected.

- 一些同学的证明，读起来很不顺畅。
- 实际上应该：要证什么，就说什么。

- 假设G-e不是k-1连通的 → G-e有一个大小为k-2的点割集 →
  - 如果e的某个端点在其中，那么这个点割集也是G的点割集，矛盾
  - 否则
    - 如果e的两个端点在G-e拿掉点割集后的同一个连通分支中
      那么将e的一个端点加入点割集中
      得到G-e（也是G）的一个大小为k-1的点割集，矛盾
    - 否则，在两个不同的连通分支中，必有一个包含不止一个顶点
      将这个连通分支中的那个e的端点加入点割集中
      得到G-e（也是G）的一个大小为k-1的点割集，矛盾



GC第5.2节练习30

5.30 For a graph G, define $\overline{\kappa}(G) = \max\{\kappa(H)\}$ and $\overline{\lambda}(G) = \max\{\lambda(H)\}$, each maximized over all subgraphs H of G. How are $\overline{\kappa}(G)$ and $\overline{\lambda}(G)$ related to $\kappa(G)$ and $\lambda(G)$, respectively, and to each other?

- \`κ\`≥κ
- \`λ\`≥λ
- \`κ\`≤\`λ\`：定理5.11 $\kappa(G) \leq \lambda(G) \leq \delta(G)$

GC第6.2节练习21

6.21 Let G be a graph of order $n \geq 3$ such that $\deg u + \deg v \geq n - 1$ for every two nonadjacent vertices u and v. Prove that G must contain a Hamiltonian path.

- 新增一个点v，连接到图中其它所有点，由定理：

Theorem 6.6 Let G be a graph of order $n \geq 3$. *If*

$$\deg u + \deg v \geq n$$

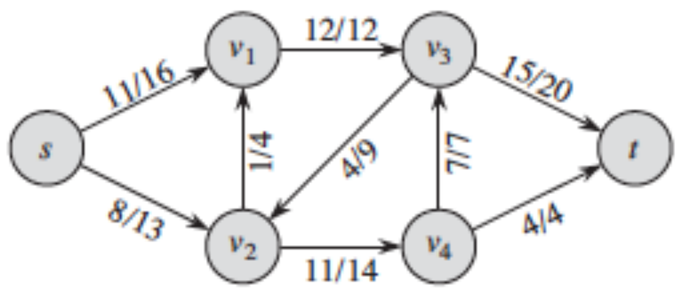
for each pair u, v of nonadjacent vertices of G, then G is Hamiltonian.

- 新图包含H圈（注意，n变为n+1）
- 再将v去掉，H圈变为H路

- 教材讨论
  - TC第26章

问题1：流网络

- 流网络包括哪些组成元素？
- 什么是最大流问题？



问题1：流网络 (续)

- 你会将实际问题建模为最大流问题来求解吗？

Survey design

We would like to design a survey of products used by consumers (i.e., "Consumer *i*: what did you think of product *j*?"). The *i*-th consumer agreed in advance to answer not more than c_i questions. For each product *j* we would like to have at least p_j opinions about it. Each consumer can be asked about a subset of the products which they consumed. We assume that we know in advance all the products each consumer used, and the above constraints. The question is how to assign questions to consumers, so that we get all the information we want to get, and every consumer is being asked a valid number of questions.

问题1：流网络 (续)

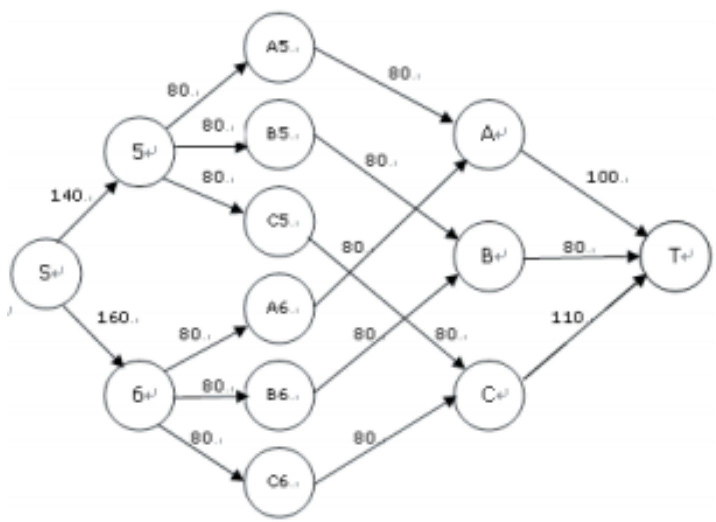
- 你会将实际问题建模为最大流问题来求解吗？

人员调配是人力资源配置决策的重要组成部分，为了确保经营活动能顺利进行，企业应该不断改善劳动组织，在空间和时间上合理安排劳动力.假设某市政工程公司准备在5、6月份完成3项工程：修建一条地下通道、修建一座人行天桥和进行道路维修.三项工程所需劳动力分别为100人、80人和110人.该公司预计5、6月份可以招聘到的劳动力分别为140人和160人，任一工程在一个月内的劳动力投入不能超过80人，问公司应如何分配劳动力以完成所有工程？

问题1：流网络 (续)

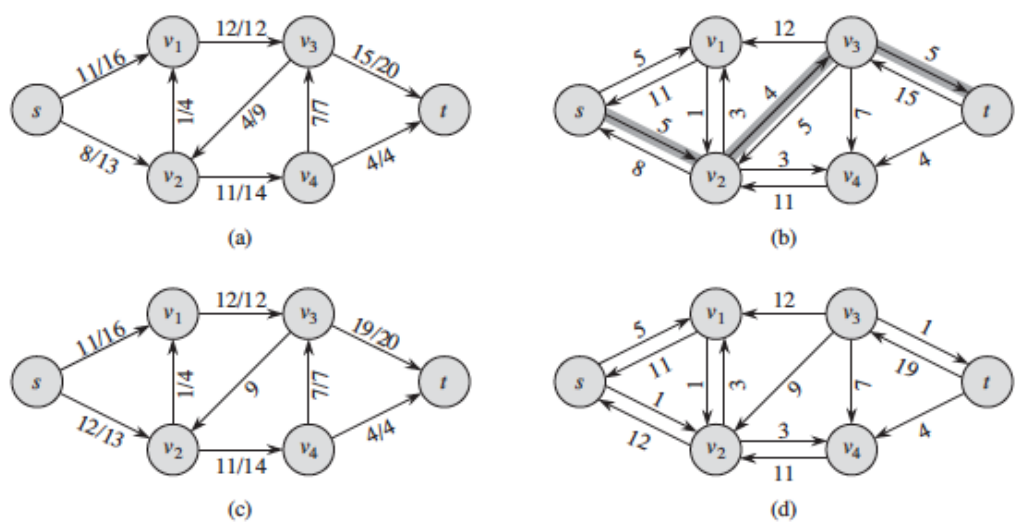
- 你会将实际问题建模为最大流问题来求解吗？

人员调配是人力资源配置决策的重要组成部分，为了确保经营活动能顺利进行，企业应该不断改善劳动组织，在空间和时间上合理安排劳动力.假设某市政工程公司准备在5、6月份完成3项工程：修建一条地下通道、修建一座人行天桥和进行道路维修.三项工程所需劳动力分别为100人、80人和110人.该公司预计5、6月份可以招聘到的劳动力分别为140人和160人，任一工程在一个月内的劳动力投入不能超过80人，问公司应如何分配劳动力以完成所有工程？



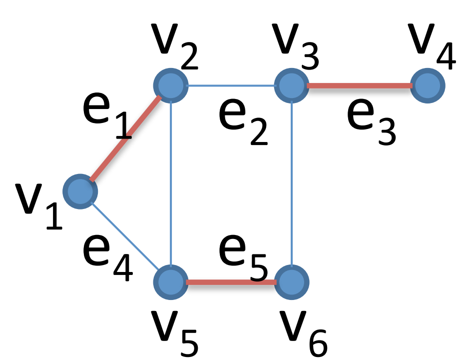
问题2：Ford-Fulkerson方法与最大匹配

- 你能结合这个例子，解释这些概念吗？
  - residual capacity、residual network
  - augmentation、augmenting path
  - Ford-Fulkerson method



问题2：Ford-Fulkerson方法与最大匹配 (续)

- 什么是最大匹配？

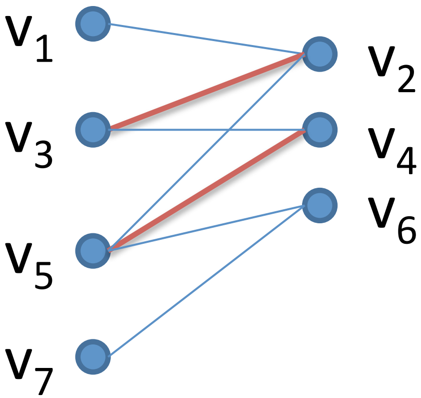


问题2：Ford-Fulkerson方法与最大匹配 (续)

- 求解二部图最大匹配的“可扩路”算法（注意：此可扩路非彼可扩路）
  1. 搜索一条可扩路
  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

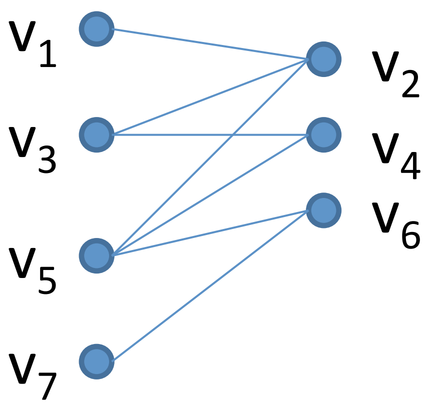
这里的可扩路：
首尾顶点未被当前匹配饱和的交错路

交错路：
边间隔地在/不在当前匹配中



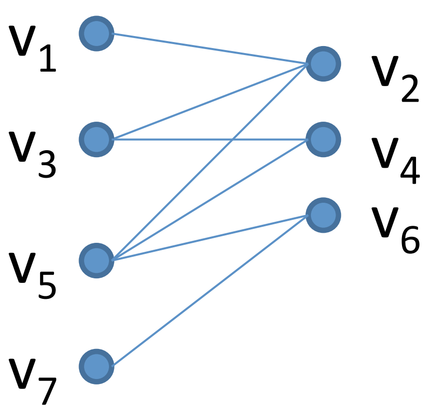
问题2：Ford-Fulkerson方法与最大匹配 (续)

- 举例



问题2：Ford-Fulkerson方法与最大匹配 (续)

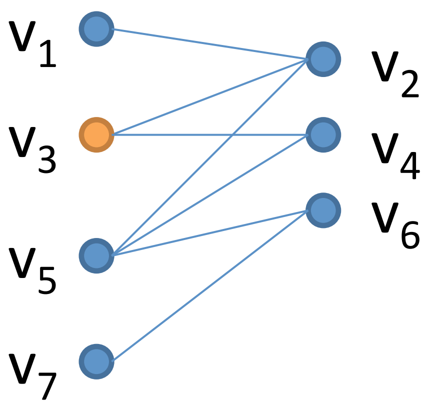
- 举例



第一轮搜索开始
- 当前匹配：{}
- 未饱和的左侧顶点：$\{v_1, v_3, v_5, v_7\}$

问题2：Ford-Fulkerson方法与最大匹配 (续)

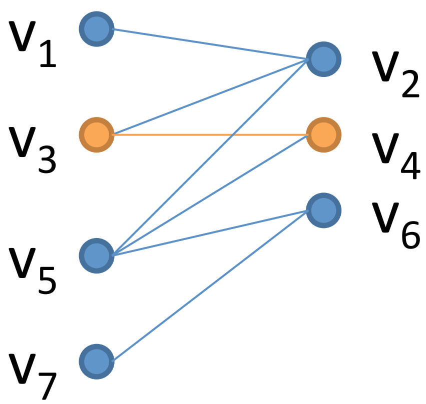
- 举例



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

问题2：Ford-Fulkerson方法与最大匹配 (续)

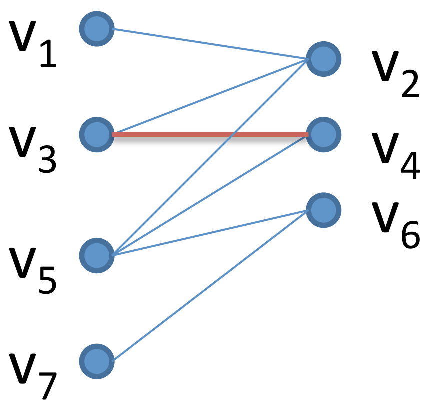
- 举例



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

问题2：Ford-Fulkerson方法与最大匹配 (续)

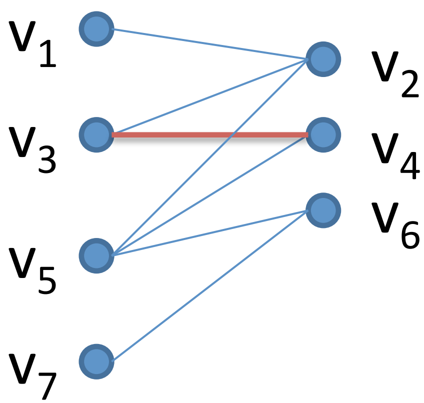
- 举例



找到可扩路$v_3v_4 \Rightarrow$ 替换进当前匹配中 $\Rightarrow$ 本轮搜索结束

问题2：Ford-Fulkerson方法与最大匹配 (续)

- 举例

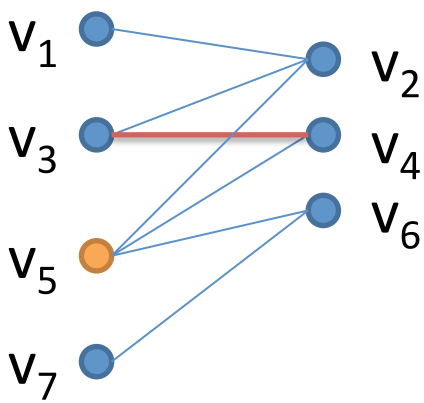


第二轮搜索开始

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

问题2：Ford-Fulkerson方法与最大匹配 (续)

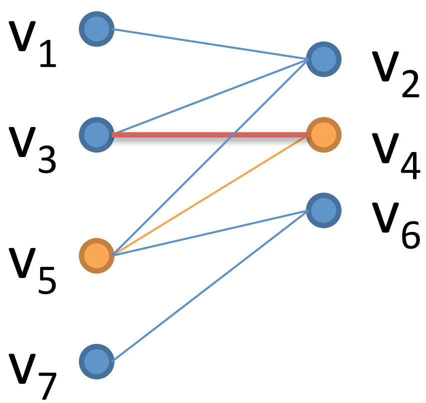
- 举例



从$\{v_1, v_5, v_7\}$中未搜索过的v_5开始搜索

问题2：Ford-Fulkerson方法与最大匹配 (续)

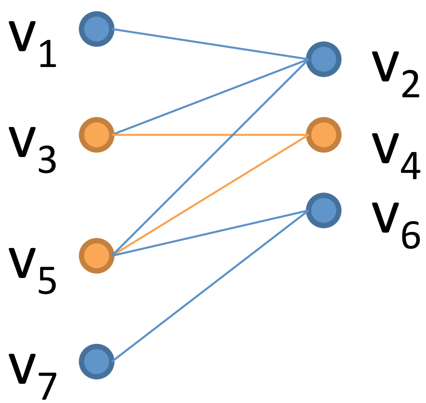
- 举例



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

问题2：Ford-Fulkerson方法与最大匹配 (续)

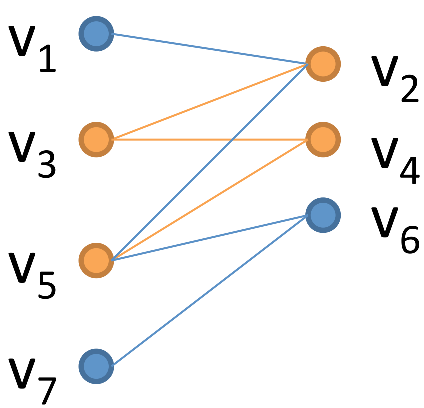
- 举例



沿当前匹配中的(v_3, v_4)到达v_3

问题2：Ford-Fulkerson方法与最大匹配 (续)

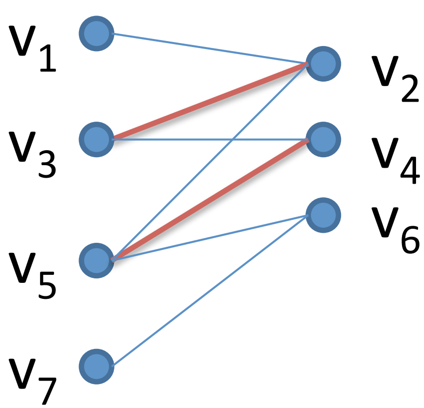
- 举例



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

问题2：Ford-Fulkerson方法与最大匹配 (续)

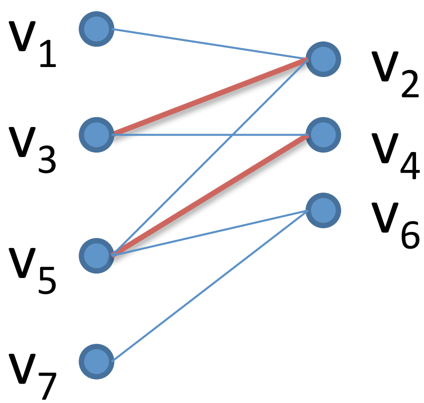
- 举例



找到可扩路$v_5v_4v_3v_2 \Rightarrow$ 替换进当前匹配中
$\Rightarrow$ 本轮搜索结束

问题2：Ford-Fulkerson方法与最大匹配 (续)

- 举例

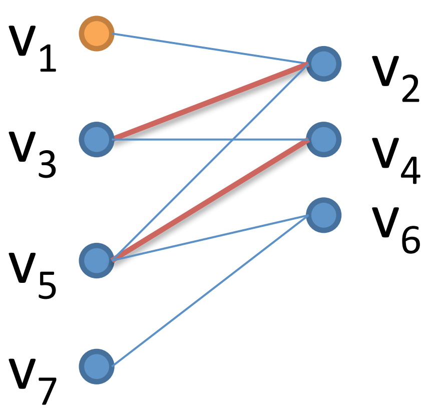


第三轮搜索开始

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

问题2：Ford-Fulkerson方法与最大匹配 (续)

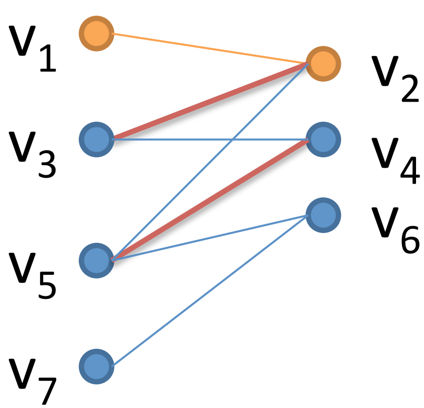
- 举例



从$\{v_1, v_7\}$中未搜索过的v_1开始搜索

问题2：Ford-Fulkerson方法与最大匹配 (续)

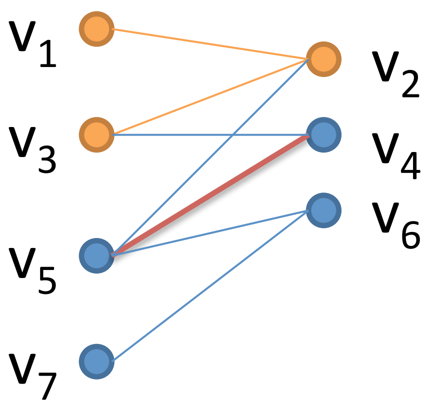
- 举例



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

问题2：Ford-Fulkerson方法与最大匹配 (续)

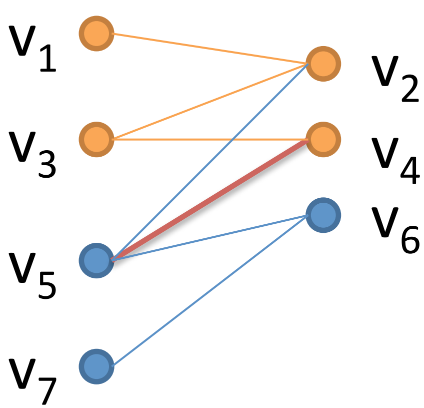
- 举例



沿当前匹配中的(v_3, v_2)到达v_3

问题2：Ford-Fulkerson方法与最大匹配 (续)

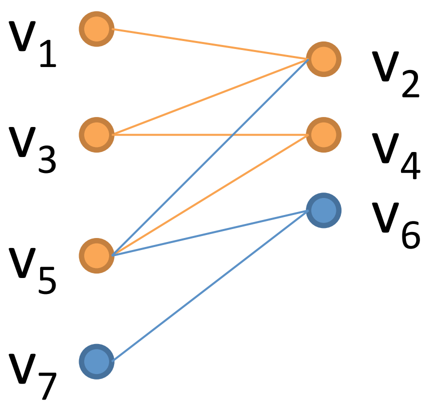
- 举例



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

问题2：Ford-Fulkerson方法与最大匹配 (续)

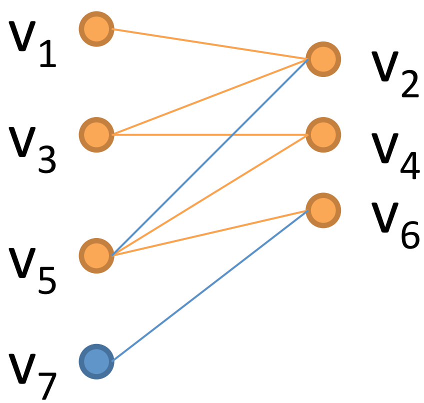
- 举例



沿当前匹配中的(v_5, v_4)到达v_5

问题2：Ford-Fulkerson方法与最大匹配 (续)

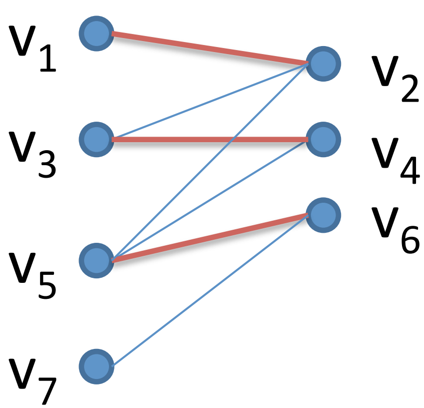
- 举例



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

问题2：Ford-Fulkerson方法与最大匹配 (续)

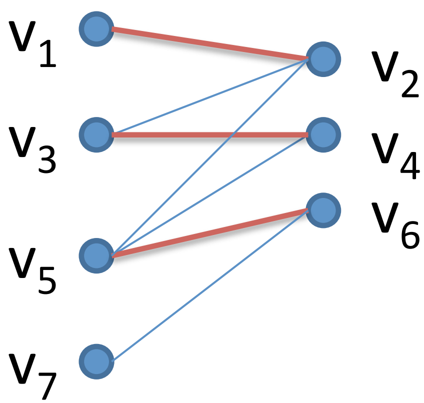
- 举例



找到可扩路$v_1v_2v_3v_4v_5v_6$ ⇒ 替换进当前匹配中 ⇒ 本轮搜索结束

问题2：Ford-Fulkerson方法与最大匹配 (续)

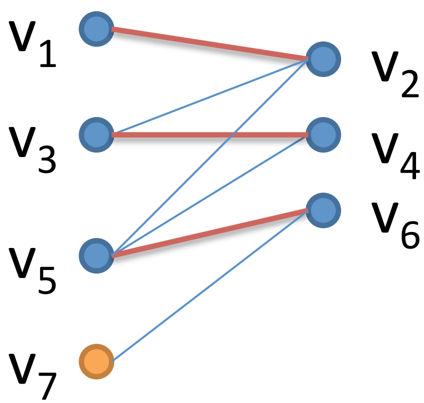
- 举例



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}

问题2：Ford-Fulkerson方法与最大匹配 (续)

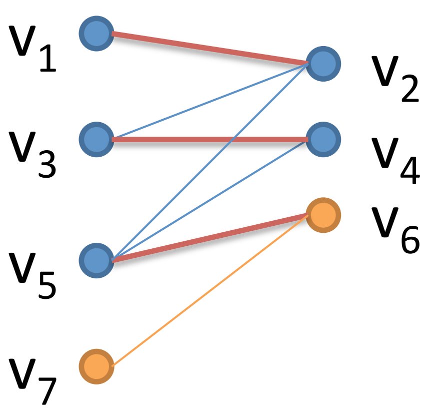
- 举例



从{v_7}中未搜索过的v_7开始搜索

问题2：Ford-Fulkerson方法与最大匹配 (续)

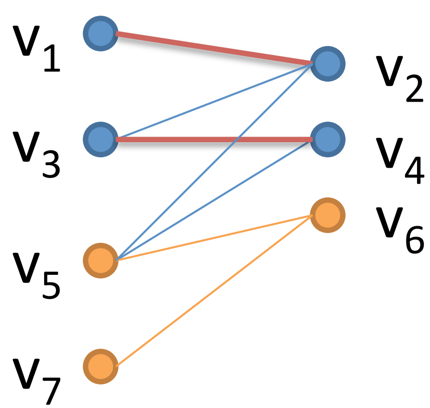
- 举例



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

问题2：Ford-Fulkerson方法与最大匹配 (续)

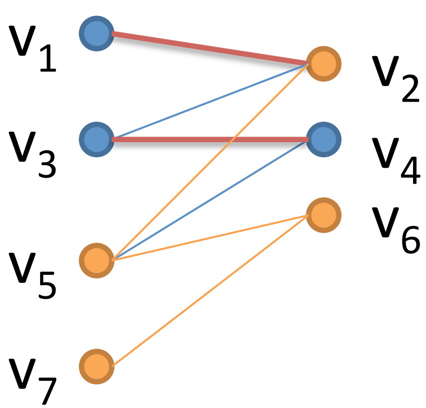
- 举例



沿当前匹配中的(v_5, v_6)到达v_5

问题2：Ford-Fulkerson方法与最大匹配 (续)

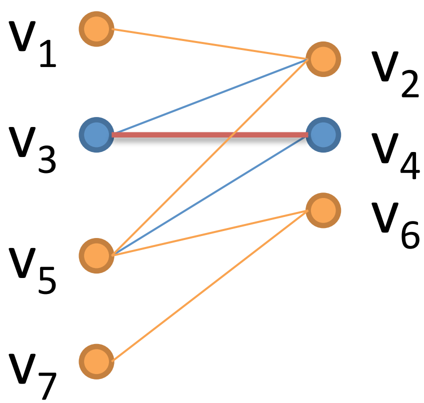
- 举例



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

问题2：Ford-Fulkerson方法与最大匹配 (续)

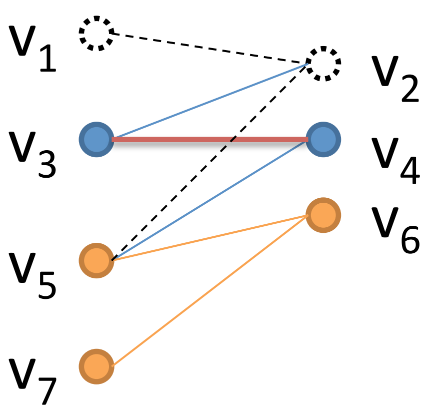
- 举例



沿当前匹配中的(v_1, v_2)到达v_1

问题2：Ford-Fulkerson方法与最大匹配 (续)

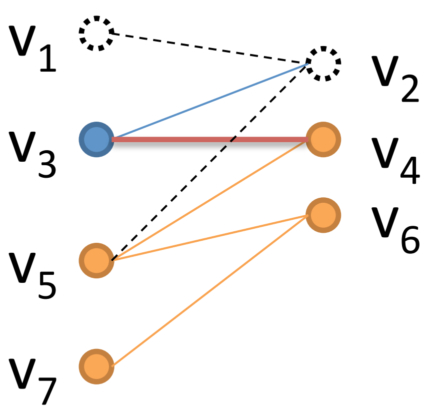
- 举例



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)

问题2：Ford-Fulkerson方法与最大匹配 (续)

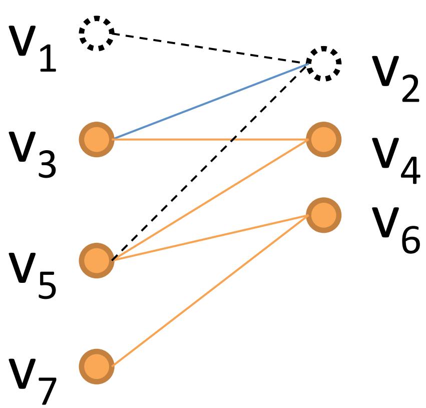
- 举例



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

问题2：Ford-Fulkerson方法与最大匹配 (续)

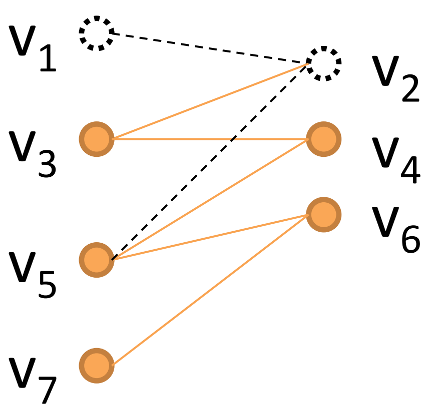
- 举例



沿当前匹配中的(v_3, v_4)到达v_3

问题2：Ford-Fulkerson方法与最大匹配 (续)

- 举例

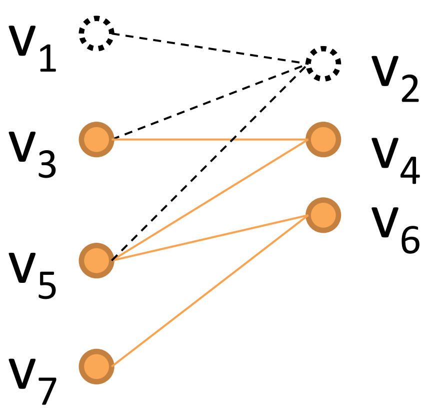


沿不在当前匹配中且本轮未搜索过的(v_3, v_2)到达v_2

问题2：Ford-Fulkerson方法与最大匹配 (续)

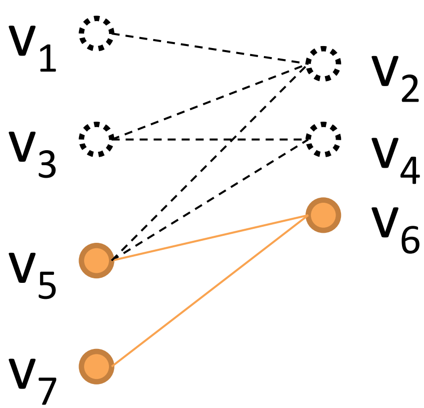
- 举例

v_2本轮已搜索过 $\Rightarrow$ 回溯到v_3



问题2：Ford-Fulkerson方法与最大匹配 (续)

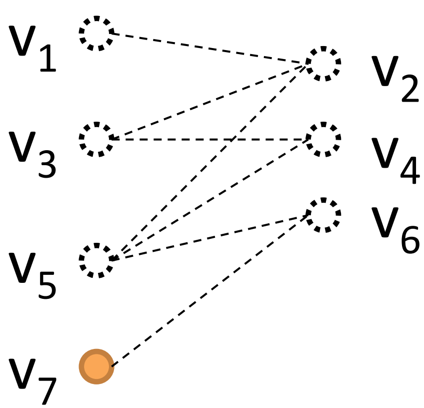
- 举例



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

问题2：Ford-Fulkerson方法与最大匹配 (续)

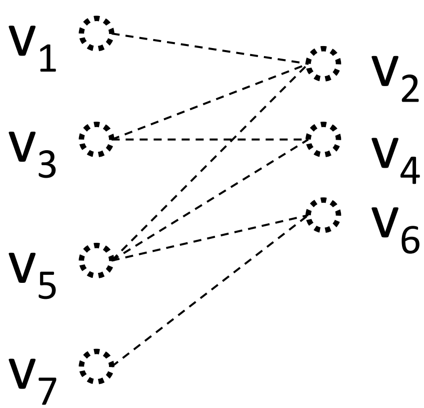
- 举例



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

问题2：Ford-Fulkerson方法与最大匹配 (续)

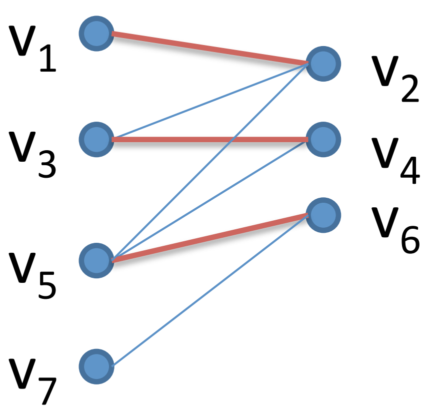
- 举例



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！

问题2：Ford-Fulkerson方法与最大匹配 (续)

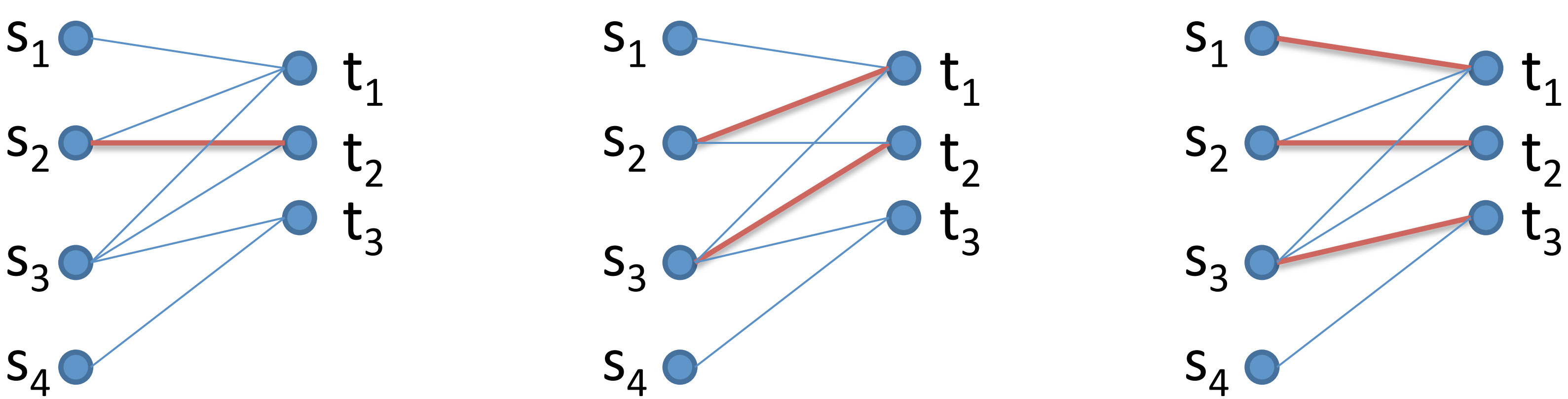
- 举例



已走过所有的点和边 $\Rightarrow$ 无可扩路 $\Rightarrow$ 找到最大匹配

问题2：Ford-Fulkerson方法与最大匹配 (续)

- 你能不能结合这个例子，讨论这两个问题的解法之间的关系？
  – 求解最大流的Ford-Fulkerson方法
  – 求解二部图最大匹配的“可扩路”算法

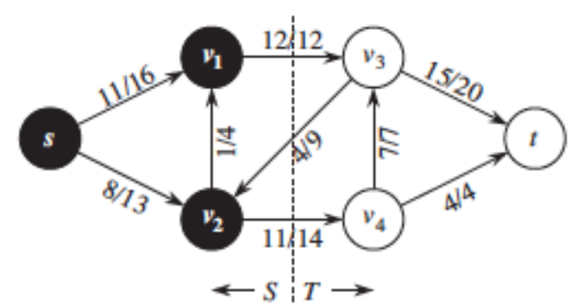


问题3：最大流最小割定理

- 什么是cut、net flow、capacity、minimum cut？
- 你理解最大流最小割定理了吗？

Theorem 26.6 (Max-flow min-cut theorem)
If f is a flow in a flow network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent:

1. f is a maximum flow in G.
2. The residual network G_f contains no augmenting paths.
3. $|f| = c(S, T)$ for some cut (S, T) of G.



问题3：最大流最小割定理 (续)

- 你能给出一种求边连通度的算法吗？

问题3：最大流最小割定理 (续)

- 你能给出一种求边连通度的算法吗？
  - 任取一个点，求到其它每个点的最大流
    即全局最小割
    即边连通度

问题4：更快的算法

- Ford-Fulkerson：找任意一条可扩路
- Edmonds-Karp：做出了怎样的改进？
- 你还能做出进一步的改进吗？
（针对最大流或者二部图最大匹配）

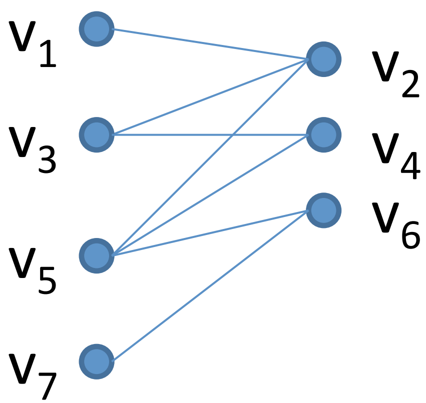
问题4：更快的算法

- Ford-Fulkerson：找任意一条可扩路
- Edmonds-Karp：做出了怎样的改进？
- 你还能做出进一步的改进吗？（针对最大流或者二部图最大匹配）

- 基本思想：同时找多条最短可扩路
  - 最大流：Dinic算法
  - 二部图最大匹配：Hopcroft-Karp算法

问题4：更快的算法 (续)

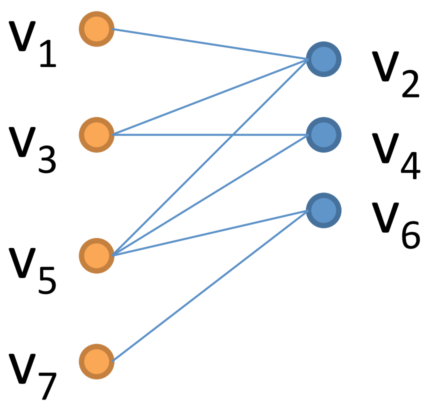
- 举例



第一轮搜索开始
- 当前匹配：{}
- 未饱和的左侧顶点：$\{v_1, v_3, v_5, v_7\}$

问题4：更快的算法 (续)

- 举例

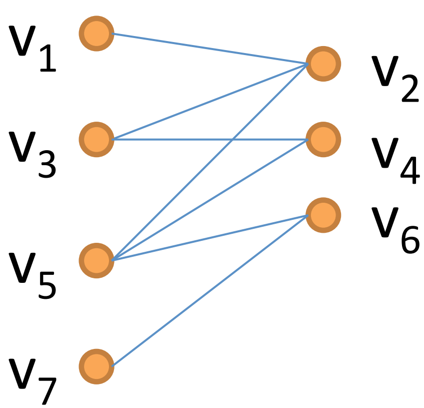


利用广度优先搜索分层

- 第0层$\{v_1, v_3, v_5, v_7\}$

问题4：更快的算法 (续)

- 举例

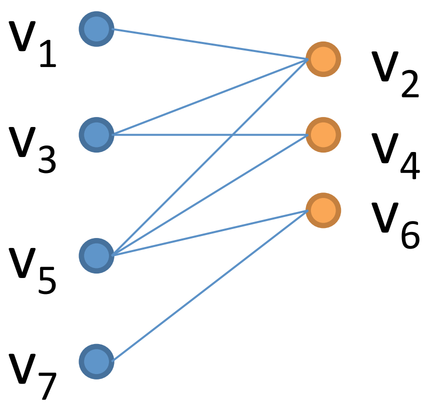


利用广度优先搜索分层

- 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$\{v_2, v_4, v_6\}$

问题4：更快的算法 (续)

- 举例

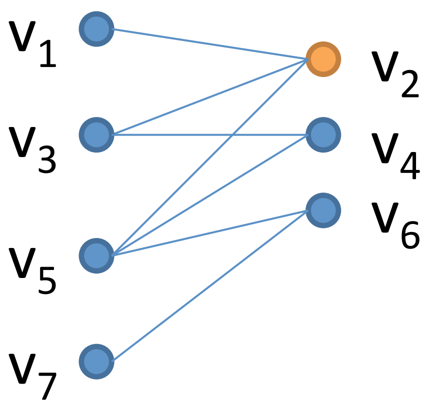


利用广度优先搜索分层

- 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$\{v_2, v_4, v_6\}$
- 发现未饱和的右侧顶点$\{v_2, v_4, v_6\}$

问题4：更快的算法 (续)

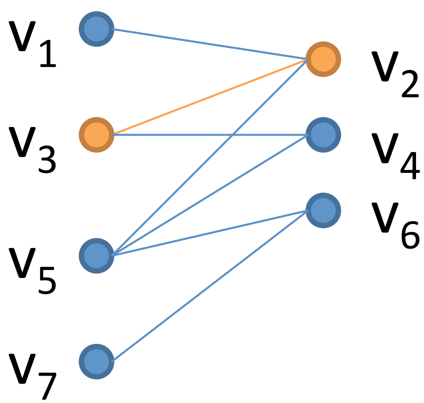
- 举例



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

问题4：更快的算法 (续)

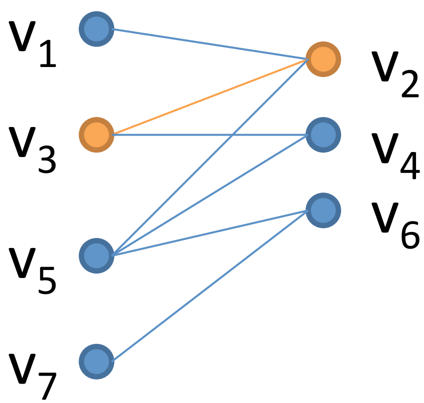
- 举例



- 1: v_2
- 0: v_3

问题4：更快的算法 (续)

- 举例

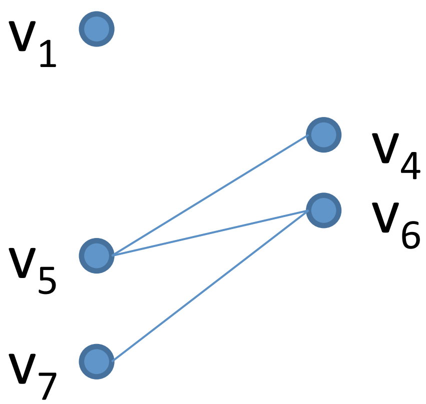


找到可扩路$v_2v_3 \Rightarrow$ 替换进当前匹配中

问题4：更快的算法 (续)

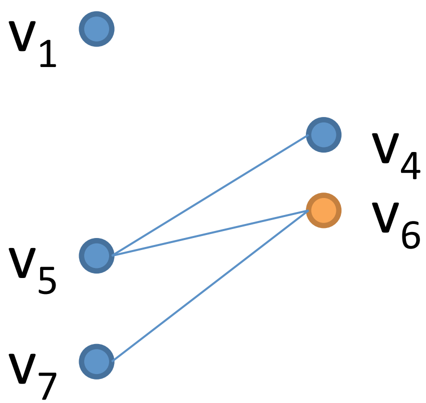
- 举例

临时删除v_2、v_3及其关联的边



问题4：更快的算法 (续)

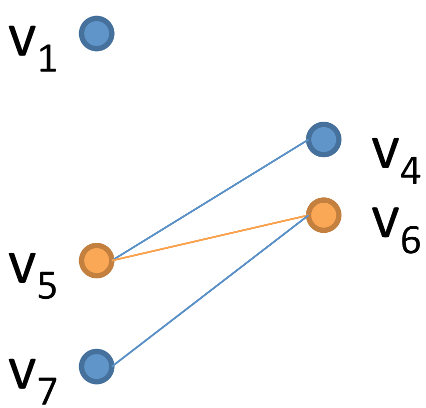
- 举例



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

问题4：更快的算法 (续)

- 举例

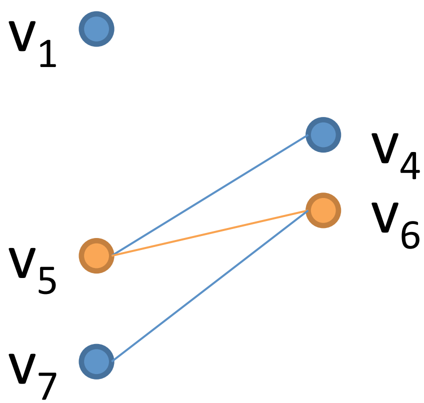


- 1: v_6
- 0: v_5

问题4：更快的算法 (续)

- 举例

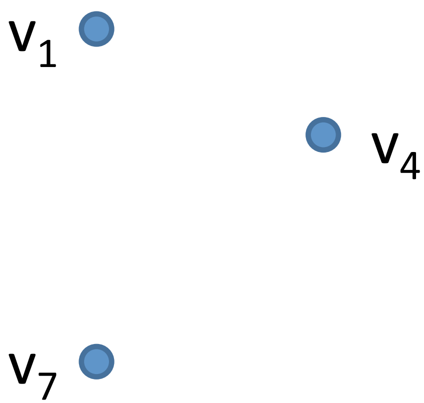
找到可扩路$v_6v_5 \Rightarrow$ 替换进当前匹配中



问题4：更快的算法 (续)

- 举例

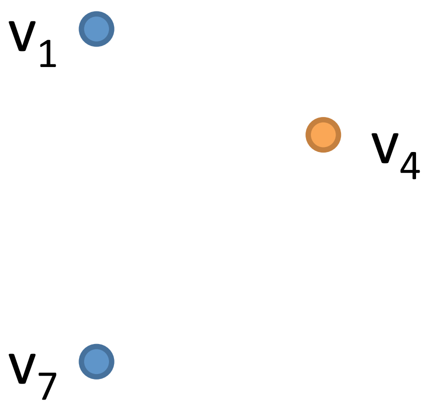
临时删除v_6、v_5及其关联的边



问题4：更快的算法 (续)

- 举例

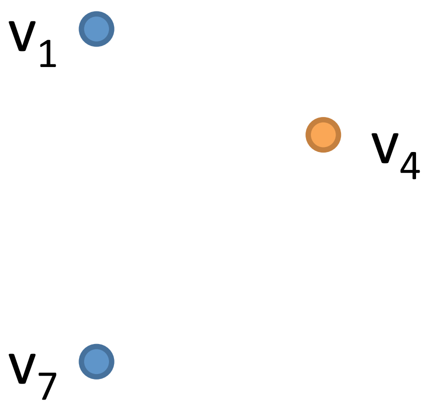
从未饱和的右侧顶点v_4（位于第1层）开始反向降层的深度优先搜索



问题4：更快的算法 (续)

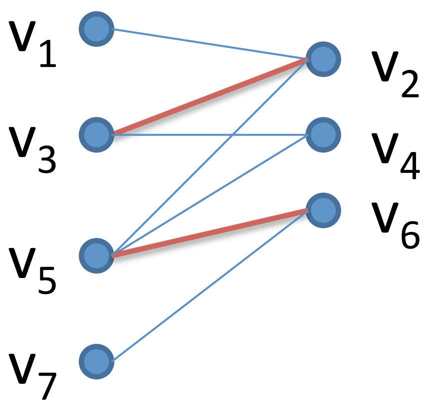
- 举例

无法下降到第0层，放弃v_4



问题4：更快的算法 (续)

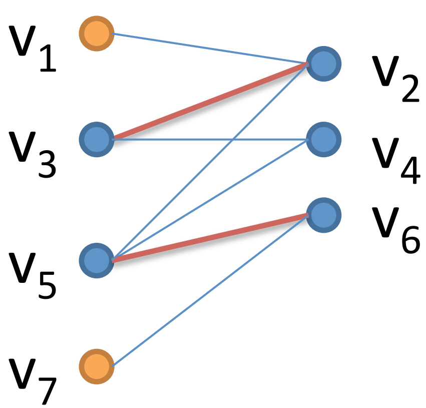
- 举例



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：$\{v_1, v_7\}$

问题4：更快的算法 (续)

- 举例

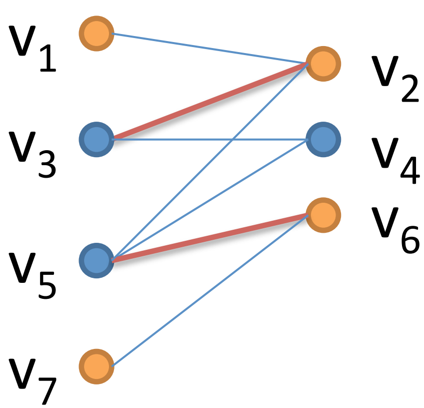


利用广度优先搜索分层

- 第0层$\{v_1, v_7\}$

问题4：更快的算法 (续)

- 举例

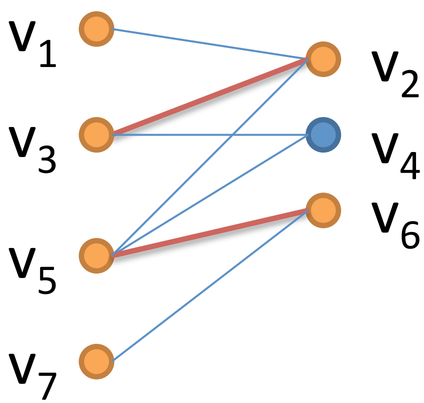


利用广度优先搜索分层

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

问题4：更快的算法 (续)

- 举例

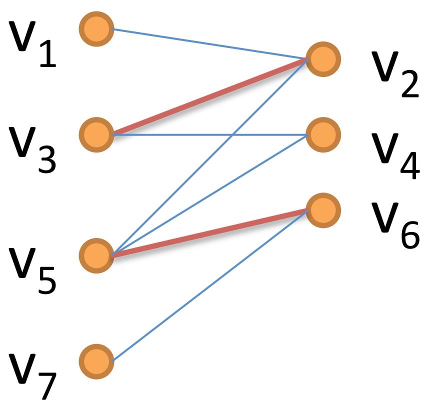


利用广度优先搜索分层

- 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$\{v_3, v_5\}$

问题4：更快的算法 (续)

- 举例

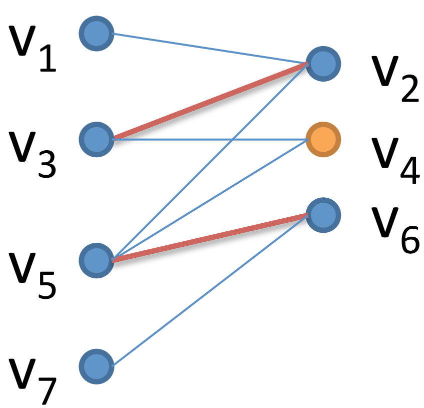


利用广度优先搜索分层

- 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}

问题4：更快的算法 (续)

- 举例

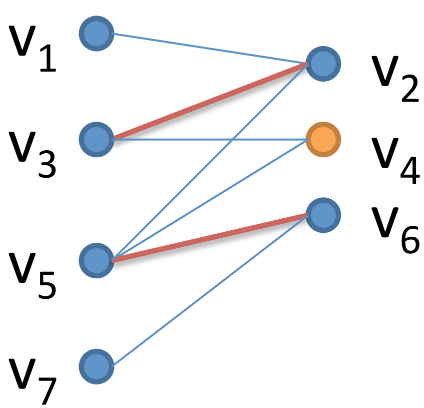


利用广度优先搜索分层

- 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$\{v_4\}$

问题4：更快的算法 (续)

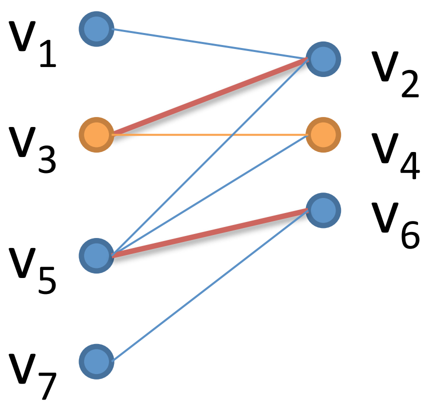
- 举例



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

问题4：更快的算法 (续)

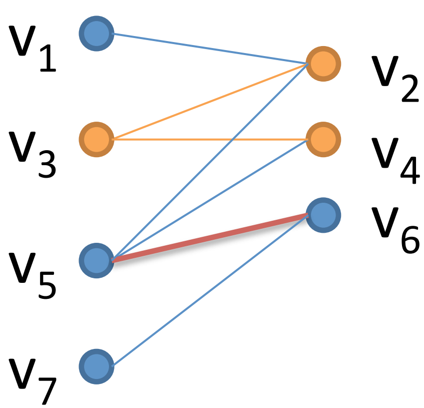
- 举例



- 3: v_4
- 2: v_3

问题4：更快的算法 (续)

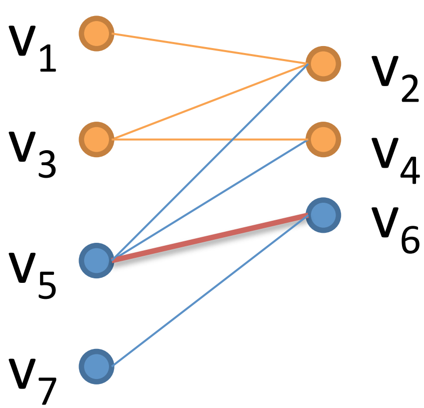
- 举例



- 3: v_4
- 2: v_3
- 1: v_2

问题4：更快的算法 (续)

- 举例

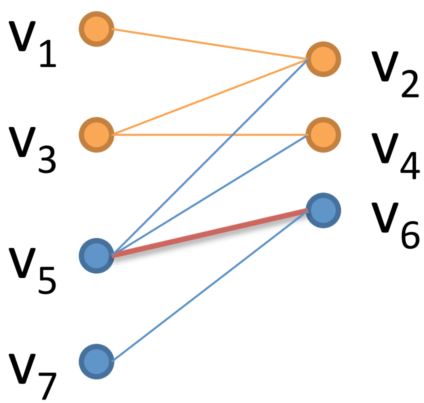


- 3: v_4
- 2: v_3
- 1: v_2
- 0: v_1

问题4：更快的算法 (续)

- 举例

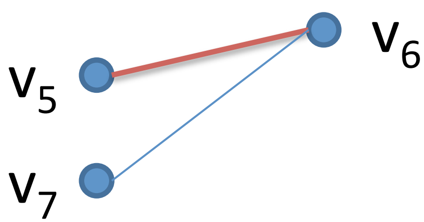
找到可扩路$v_4v_3v_2v_1 \Rightarrow$ 替换进当前匹配中



问题4：更快的算法 (续)

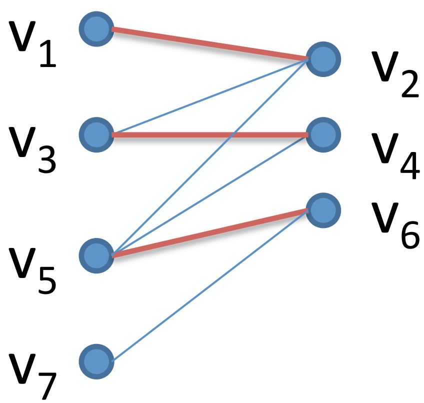
- 举例

临时删除v_4、v_3、v_2、v_1及其关联的边



问题4：更快的算法 (续)

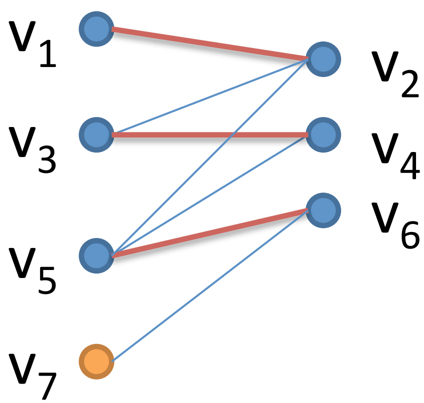
- 举例



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：$\{v_7\}$

问题4：更快的算法 (续)

- 举例

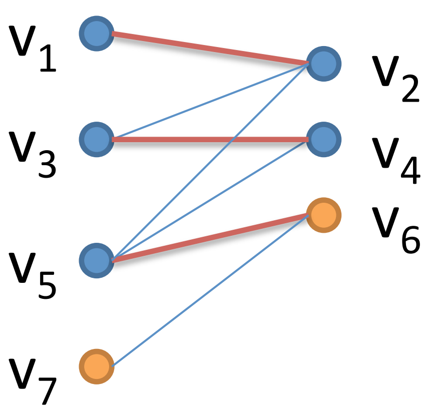


利用广度优先搜索分层

- 第0层{v_7}

问题4：更快的算法 (续)

- 举例

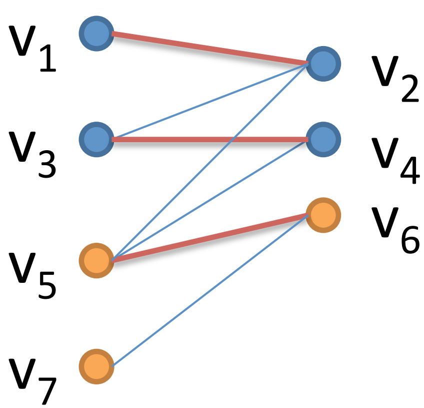


利用广度优先搜索分层

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

问题4：更快的算法 (续)

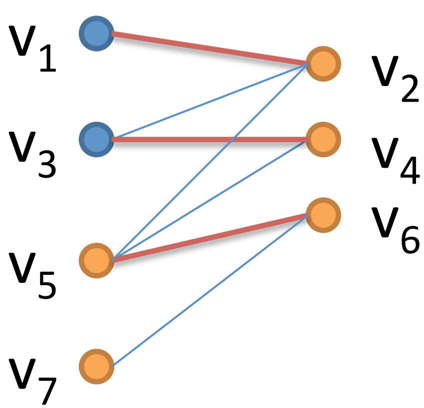
- 举例



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$\{v_5\}$

问题4：更快的算法 (续)

- 举例

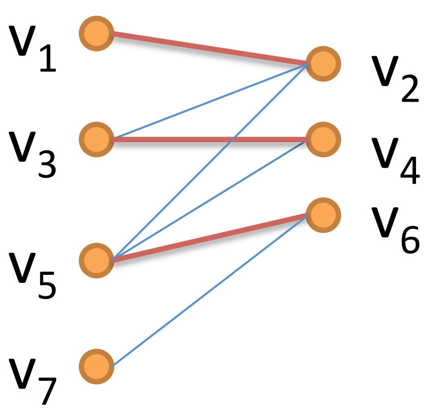


利用广度优先搜索分层

- 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}

问题4：更快的算法 (续)

- 举例

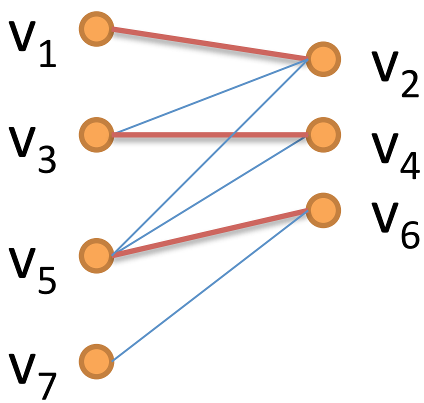


利用广度优先搜索分层

- 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}

问题4：更快的算法 (续)

- 举例



已搜完所有顶点，未发现未饱和的右侧顶点
⇒ 无可扩路 ⇒ 找到最大匹配