第5章 匹配

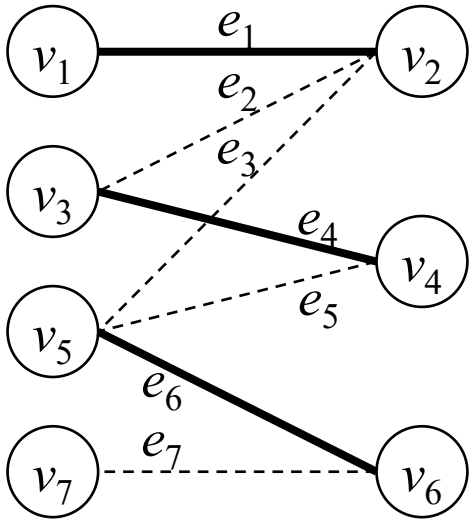
程龚
南京大学 计算机学院
gcheng@nju.edu.cn
http://ws.nju.edu.cn/~gcheng

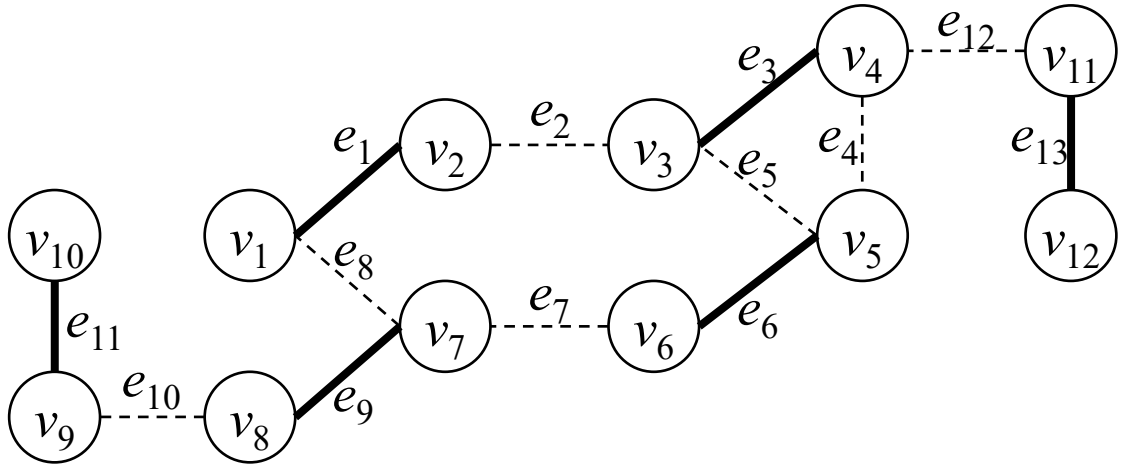
本章内容

- 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?

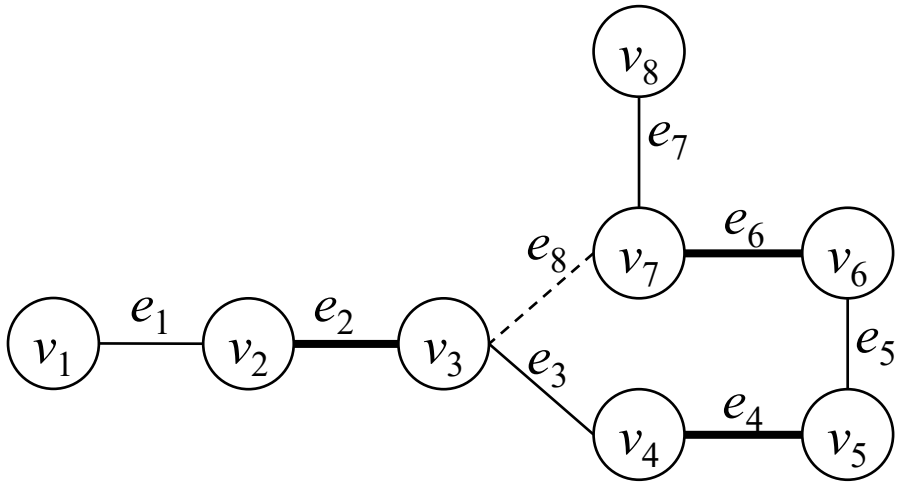
- 对于二分图
  1. 匈牙利算法
  2. 霍普克罗夫特-卡普算法
- 对于非二分图
  3. **花算法**





匈牙利算法和霍普克罗夫特-卡普算法只适用于二分图

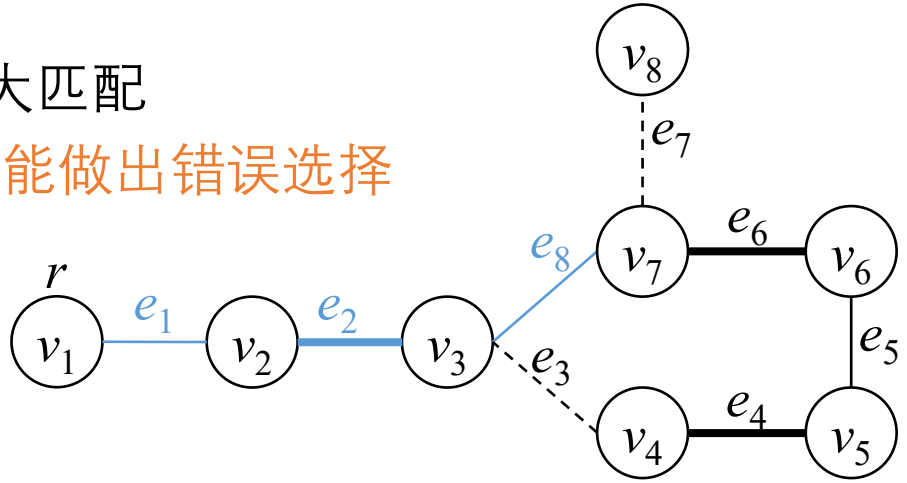
- 对于非二分图，这些算法未必能找到增广路和最大匹配



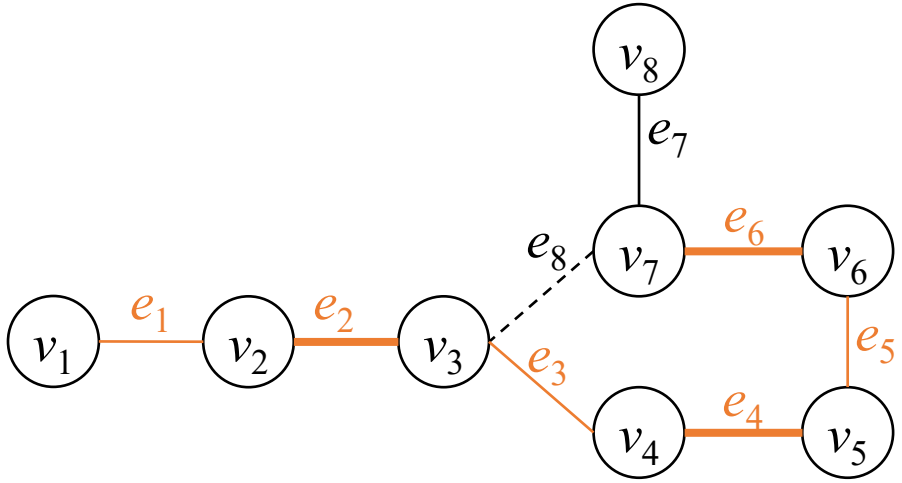
存在M增广路

匈牙利算法和霍普克罗夫特-卡普算法只适用于二分图

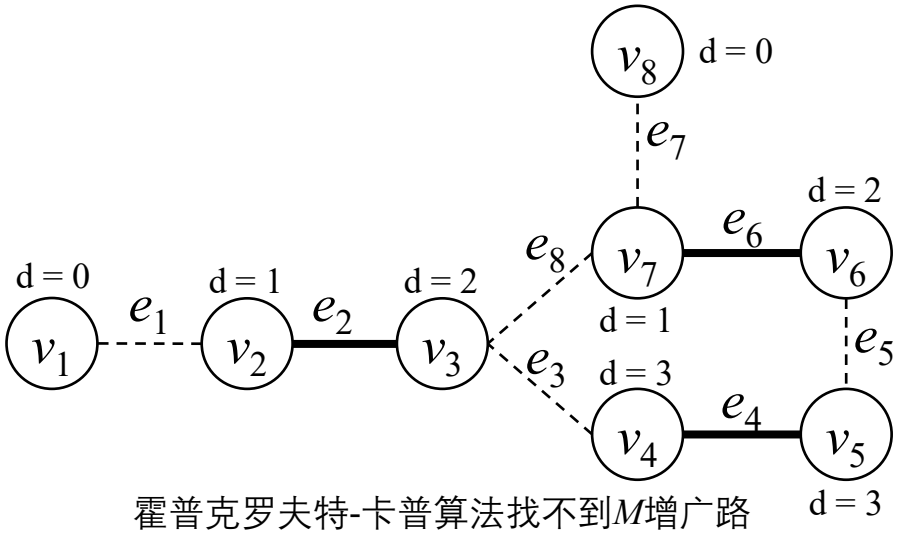
- 对于非二分图，这些算法未必能找到增广路和最大匹配
- 成因：从顶点v_1到v_7存在两条M交错路，算法有可能做出错误选择
  - 若选择$v_1, v_2, v_3, v_4, v_5, v_6, v_7$，则可找到$M$增广路
  - 若选择v_1, v_2, v_3, v_7，则找不到M增广路



匈牙利算法找不到M增广路



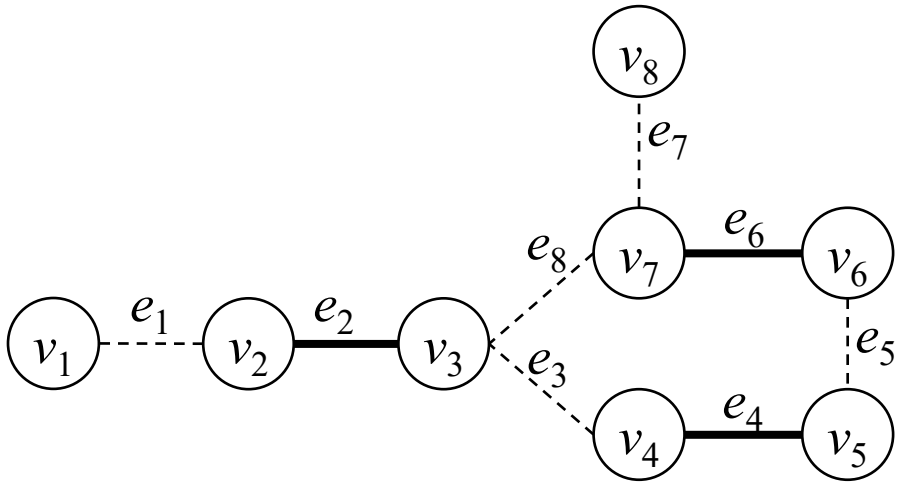
存在M增广路



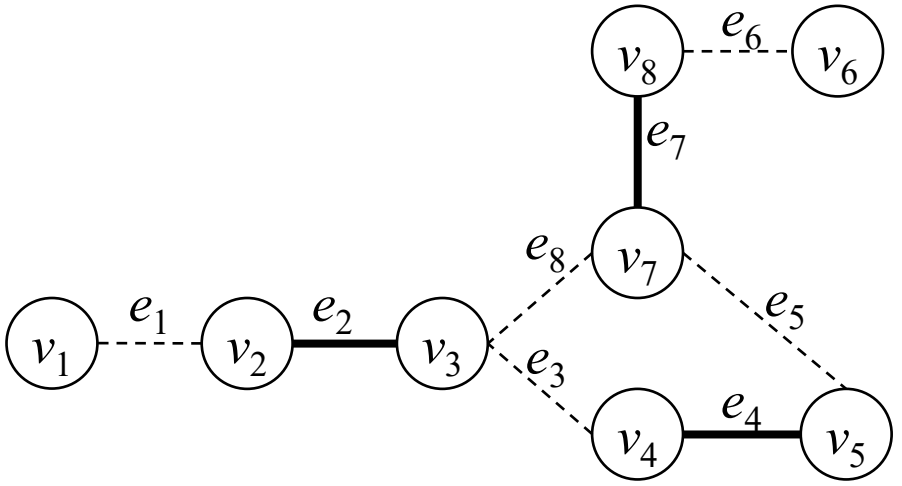
霍普克罗夫特-卡普算法找不到M增广路

思考题5.16

- 为什么二分图不存在上述问题?



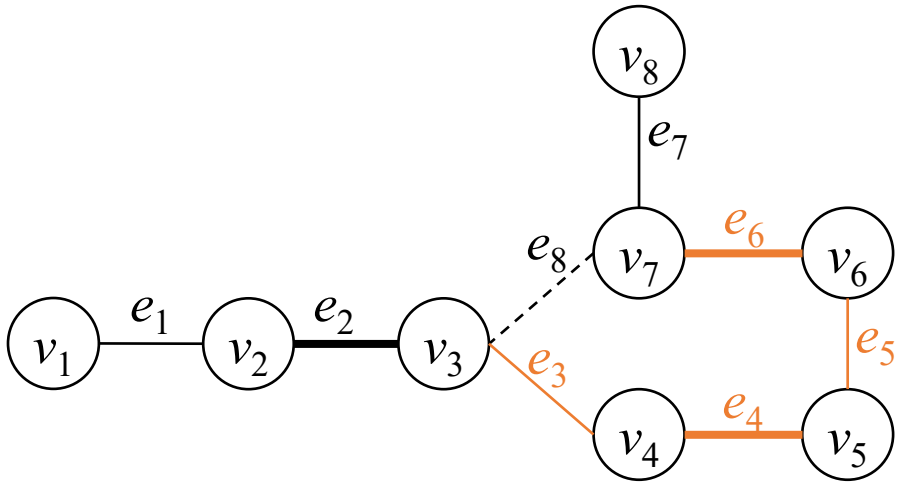
非二分图



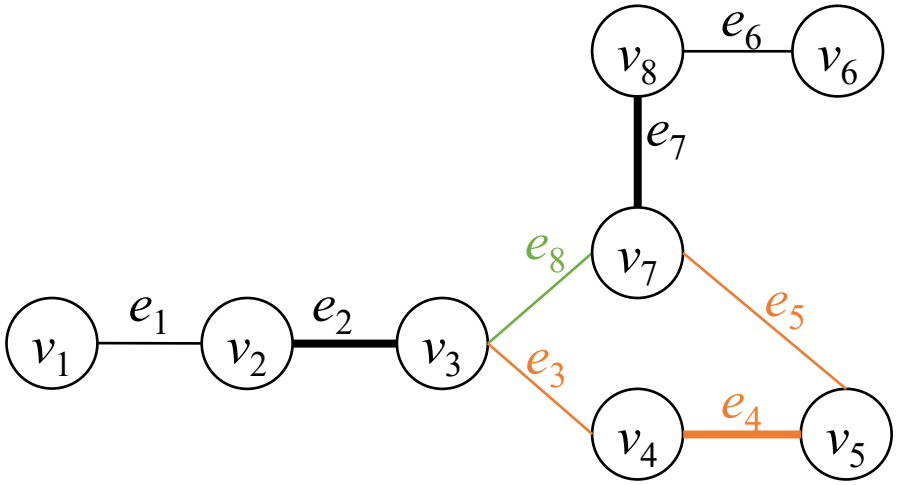
二分图

思考题5.16

- 为什么二分图不存在上述问题?
  - 偶圈对应的两条M交错路是等效的



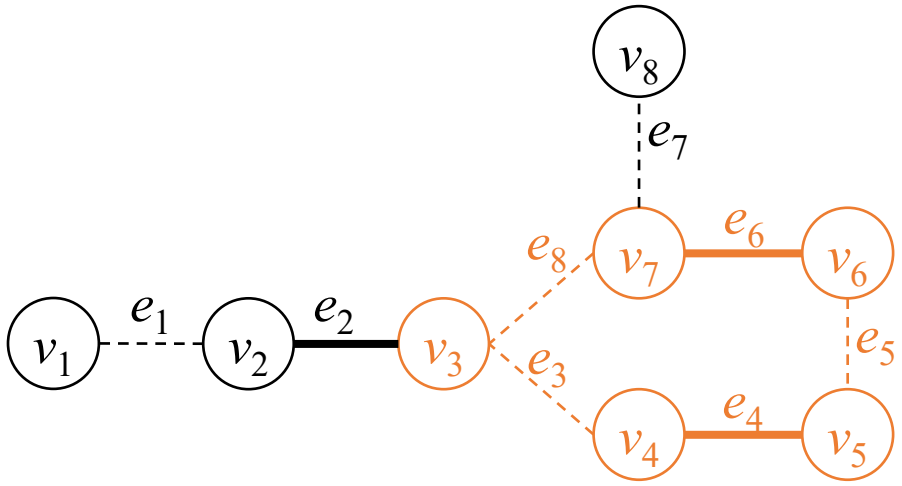
非二分图



二分图

花、花梗、花托

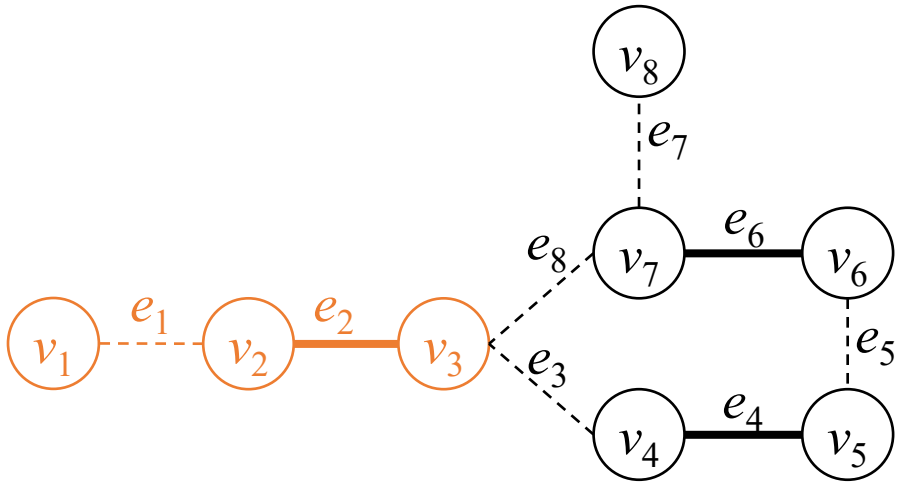
- 上述两条M交错路形成的奇圈称作**花**
  - 两条M交错路的差异部分形成一个长度为$2k+1$的奇圈，恰经过k条在M中的边



花

花、花梗、花托

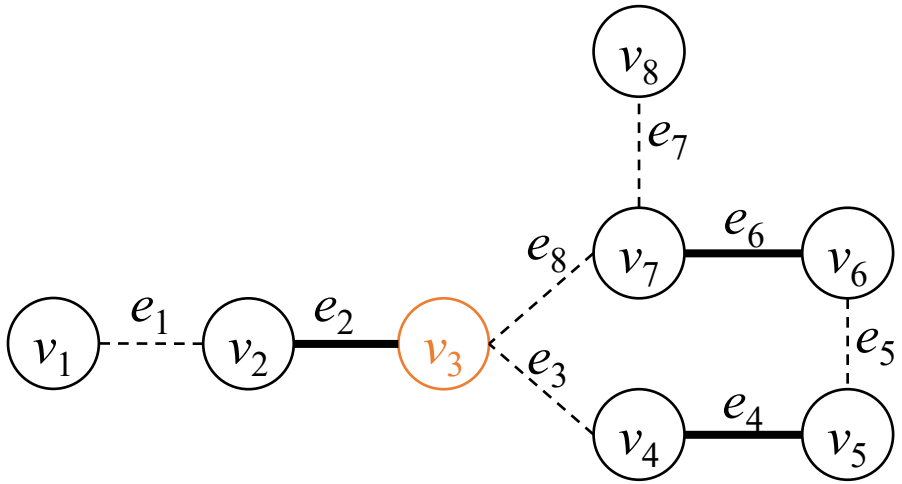
- 上述两条M交错路形成的奇圈称作花
  - 两条M交错路的差异部分形成一个长度为$2k+1$的奇圈，恰经过k条在M中的边
- 两条M交错路的公共子路称作**花梗**



花梗

花、花梗、花托

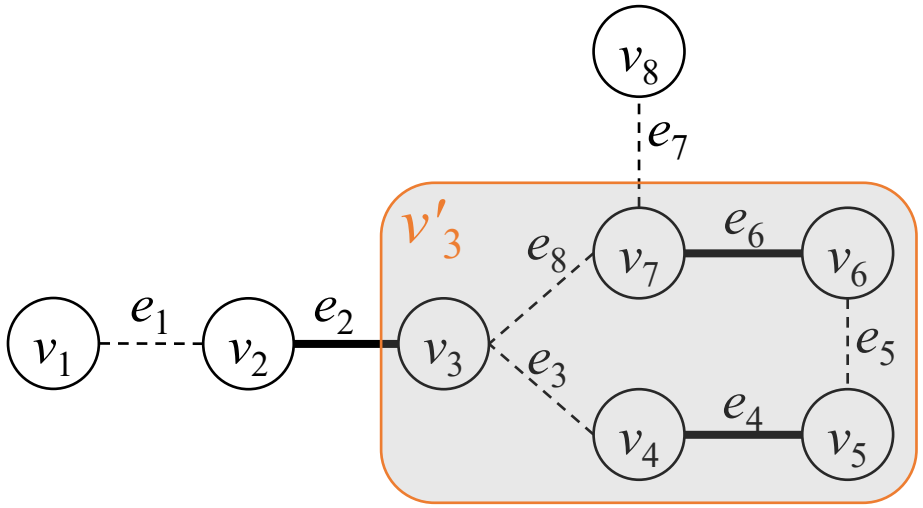
- 上述两条M交错路形成的奇圈称作花
  - 两条M交错路的差异部分形成一个长度为$2k+1$的奇圈，恰经过k条在M中的边
- 两条M交错路的公共子路称作花梗
- 花梗的终点称作**花托**



花托

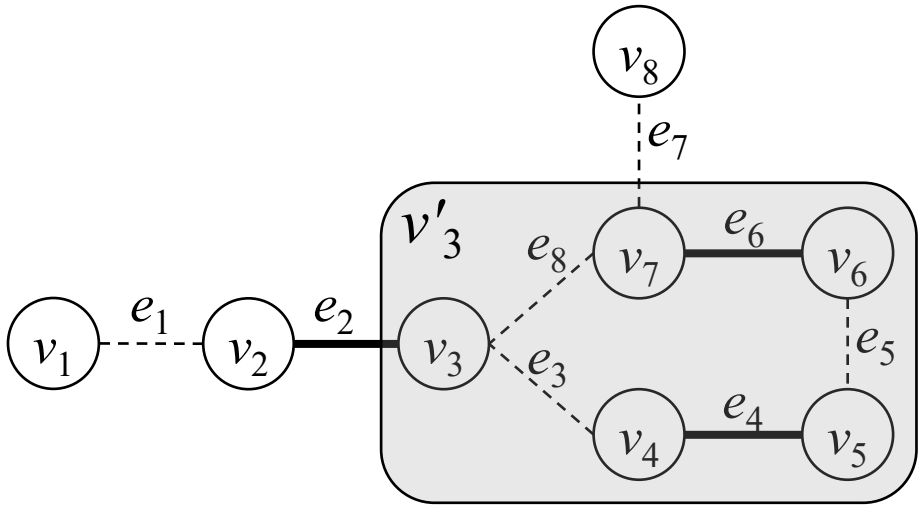
花算法

- 主要思想：扩展匈牙利算法的do-while循环
  - 经过花梗从花托进入花时，算法先不做任何选择，而是将花中所有顶点收缩为一个新的顶点。
  - 在收缩后的新图上，继续访问。
  - 当访问到未被匹配M饱和的顶点时，得到M增广路，但这并非原图中的路，需要再将新顶点还原为花。需要从花中的两条路中做出选择。此时，只有一种选择才能组成M增广路，从而做出正确选择。



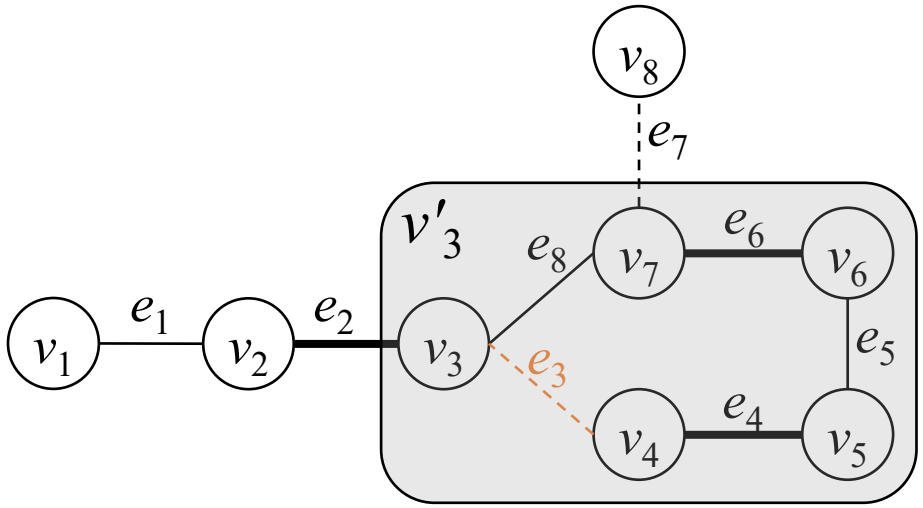
思考题5.17

- 在花算法中，如何识别花?



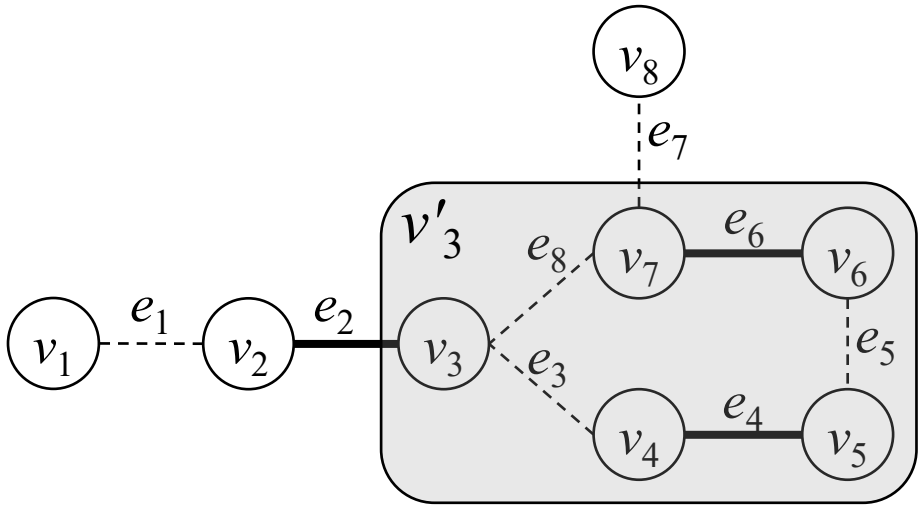
思考题5.17

- 在花算法中，如何识别花？
  - 经过长度为偶数的M交错路访问的顶点关联的一条后向边的另一个端点也是经过长度为偶数的M交错路访问的



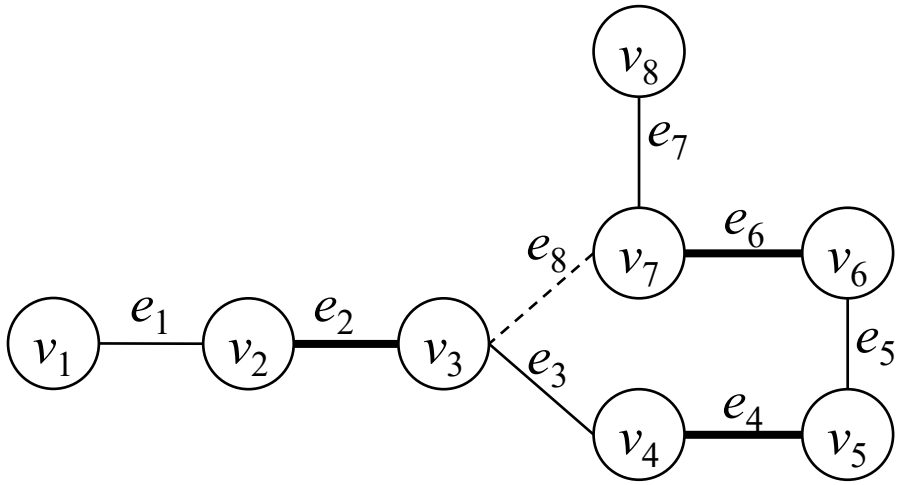
思考题5.18

- 在花算法运行过程中，所有花都会被收缩吗？



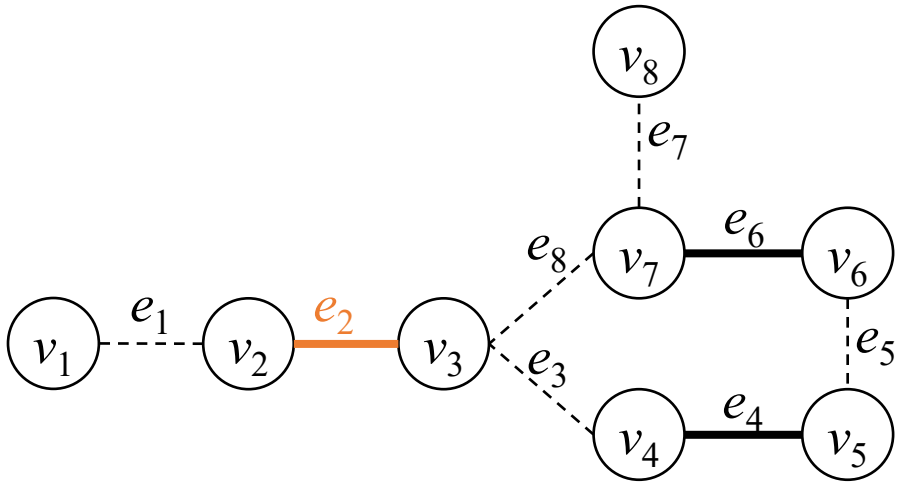
思考题5.18

- 在花算法运行过程中，所有花都会被收缩吗？
  - 不一定会



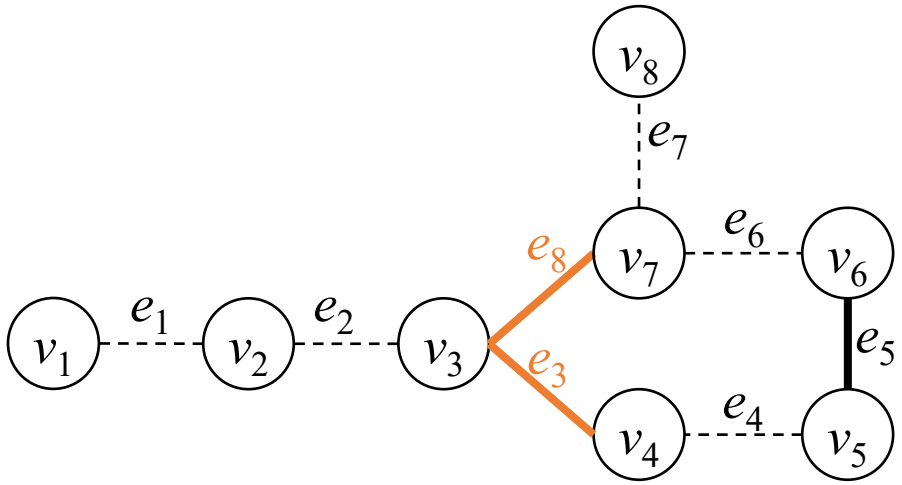
思考题5.19

- 花梗的最后一条边有可能不在匹配M中吗?



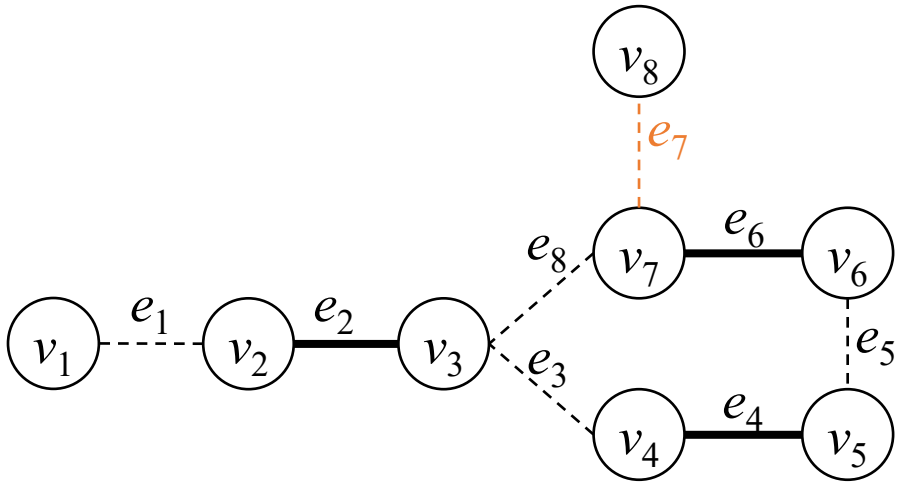
思考题5.19

- 花梗的最后一条边有可能不在匹配M中吗？
  - 采用反证法，假设花梗的最后一条边不在匹配M中，而花是由两条M交错路形成的奇圈，则花托关联的花中的两条边都在M中，与M是匹配矛盾。



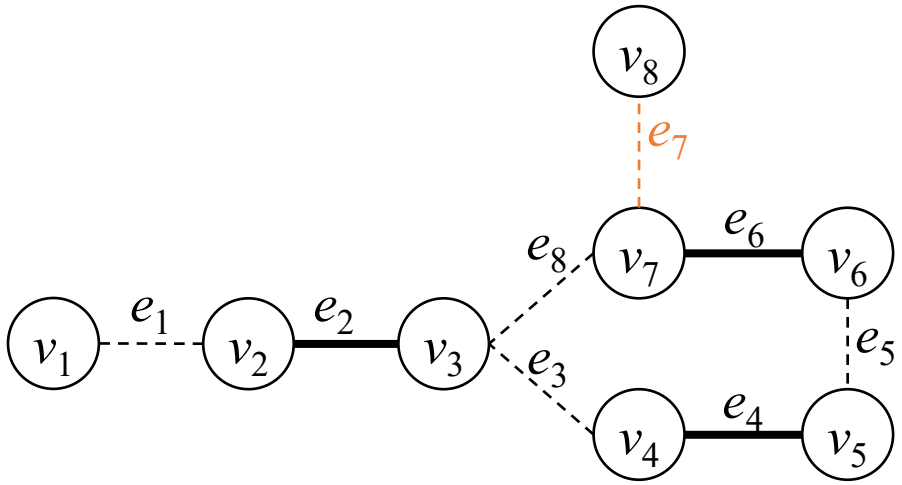
思考题5.20

- 花中顶点与花外顶点间的边（除花梗的最后一条边外）有可能在匹配M中吗？



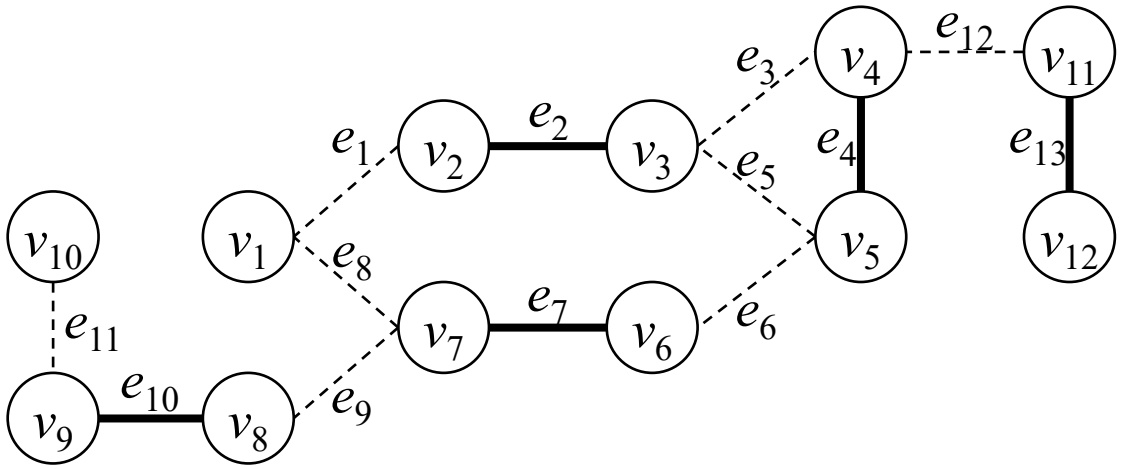
思考题5.20

- 花中顶点与花外顶点间的边（除花梗的最后一条边外）有可能在匹配M中吗？
  - 花中顶点都被匹配M饱和，因此，除花梗的最后一条边外，花中顶点与花外顶点间的边都不在M中。



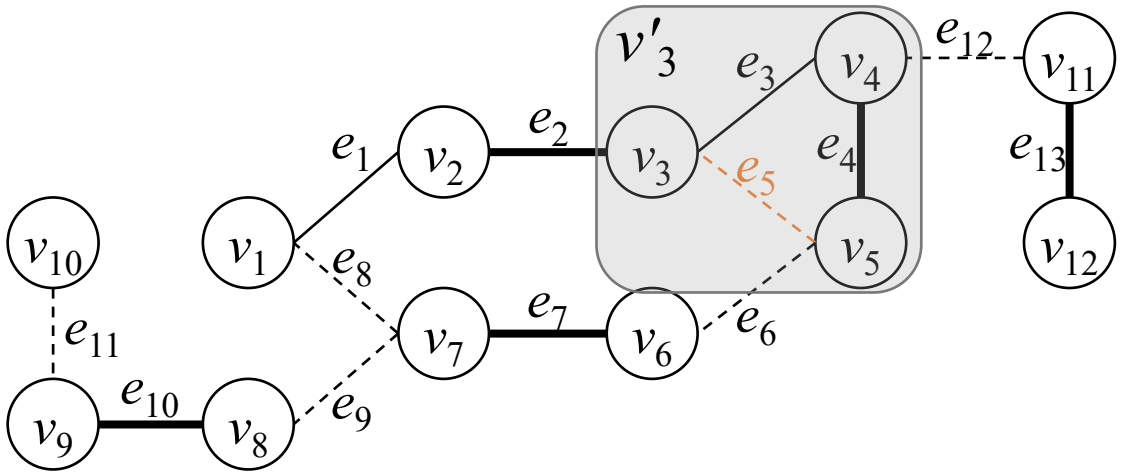
花算法

- 例如：第5轮do-while循环后，$M \leftarrow \{e_2, e_4, e_7, e_{10}, e_{13}\}$



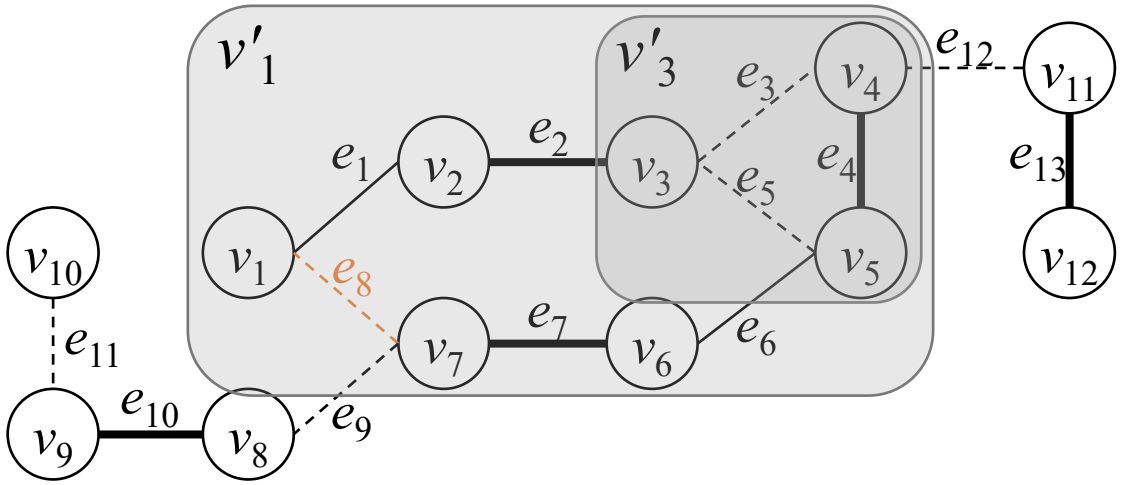
花算法

- 第6轮do-while循环：
  - 从v_1出发，识别出花梗v_1, v_2, v_3和花v_3, v_4, v_5, v_3，花收缩为v'_3



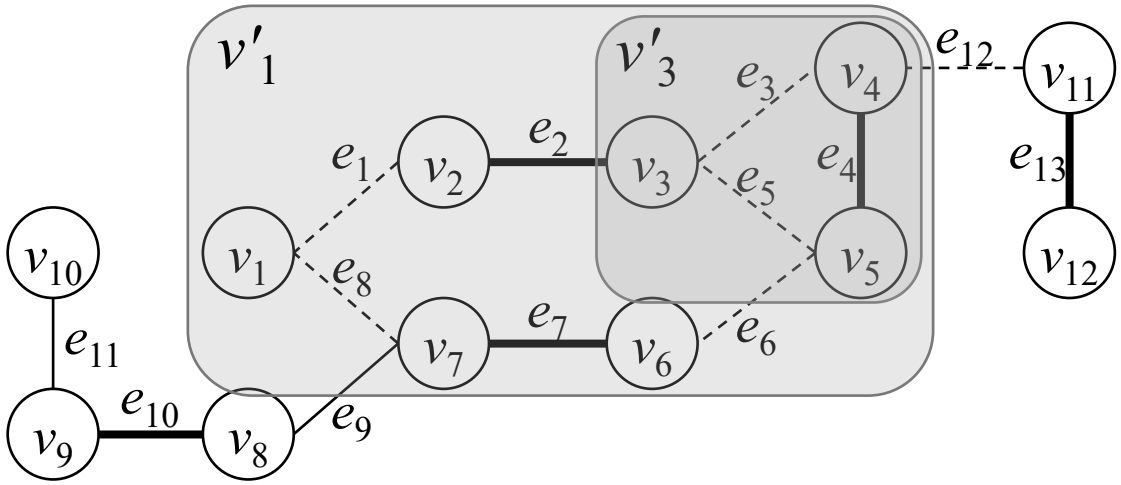
花算法

- 第6轮do-while循环：
  - 从v_1出发，识别出花梗v_1, v_2, v_3和花v_3, v_4, v_5, v_3，花收缩为v'_3
  - 继续访问v'_3的邻点，识别出花梗v_1和花$v_1, v_2, v'_3, v_6, v_7, v_1$，花收缩为$v'_1$



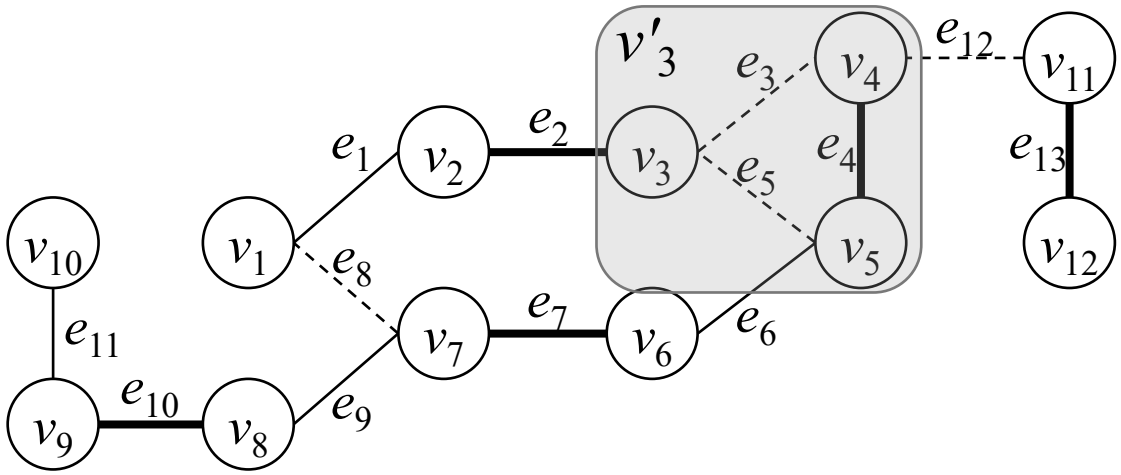
花算法

- 第6轮do-while循环：
  - 从v_1出发，识别出花梗v_1, v_2, v_3和花v_3, v_4, v_5, v_3，花收缩为v'_3
  - 继续访问v'_3的邻点，识别出花梗v_1和花$v_1, v_2, v'_3, v_6, v_7, v_1$，花收缩为$v'_1$
  - 继续访问v'_1的邻点，找到M增广路v'_1, v_8, v_9, v_{10}



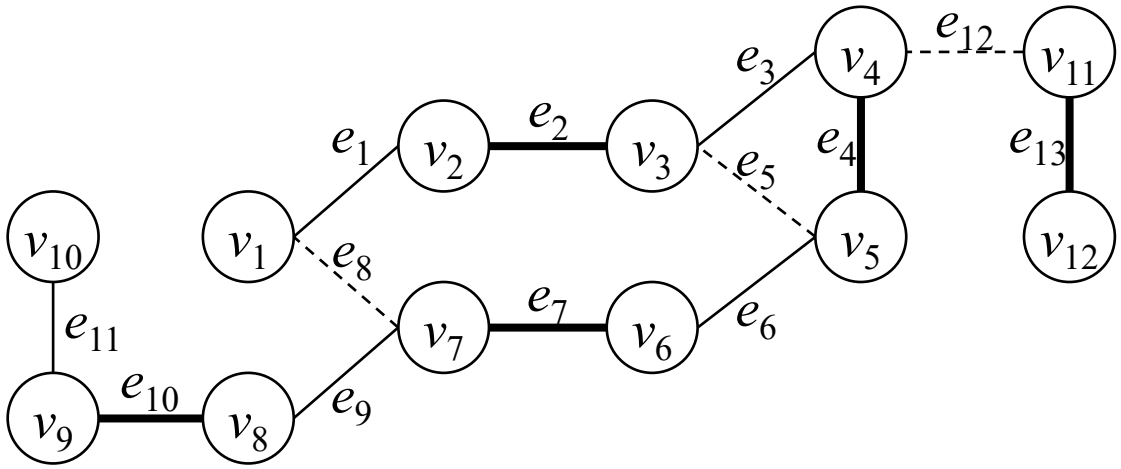
花算法

- 第6轮do-while循环：
  - 从v_1出发，识别出花梗v_1, v_2, v_3和花v_3, v_4, v_5, v_3，花收缩为v'_3
  - 继续访问v'_3的邻点，识别出花梗v_1和花$v_1, v_2, v'_3, v_6, v_7, v_1$，花收缩为$v'_1$
  - 继续访问v'_1的邻点，找到M增广路v'_1, v_8, v_9, v_{10}
  - 将v'_1还原为花，从中选择v_1-v_7路v_1, v_2, v'_3, v_6, v_7，组成M增广路$v_1, v_2, v'_3, v_6, v_7, v_8, v_9, v_{10}$



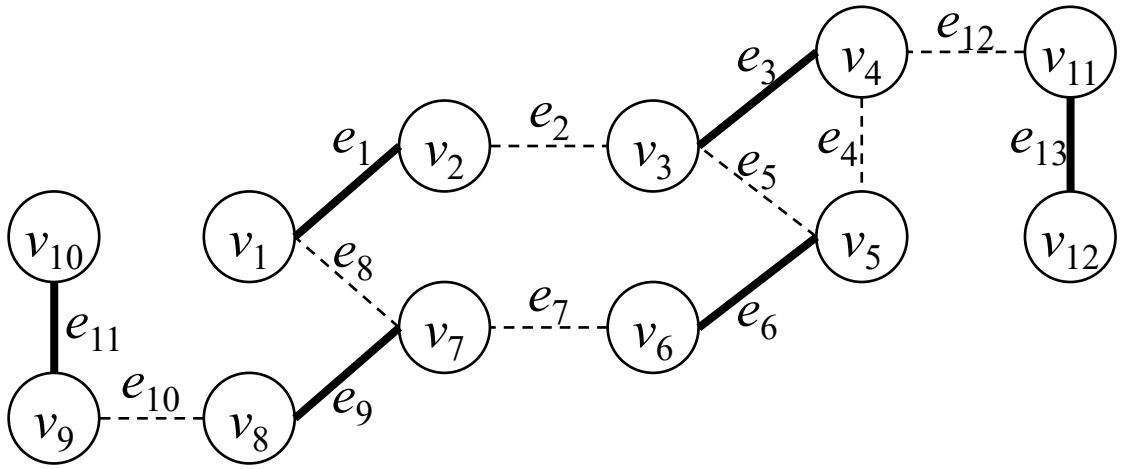
花算法

- 第6轮do-while循环：
  - 从v_1出发，识别出花梗v_1, v_2, v_3和花v_3, v_4, v_5, v_3，花收缩为v'_3
  - 继续访问v'_3的邻点，识别出花梗v_1和花$v_1, v_2, v'_3, v_6, v_7, v_1$，花收缩为$v'_1$
  - 继续访问v'_1的邻点，找到M增广路v'_1, v_8, v_9, v_{10}
  - 将v'_1还原为花，从中选择v_1-v_7路v_1, v_2, v'_3, v_6, v_7，组成M增广路$v_1, v_2, v'_3, v_6, v_7, v_8, v_9, v_{10}$
  - 将v'_3还原为花，从中选择v_3-v_5路v_3, v_4, v_5，组成M增广路$v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}$



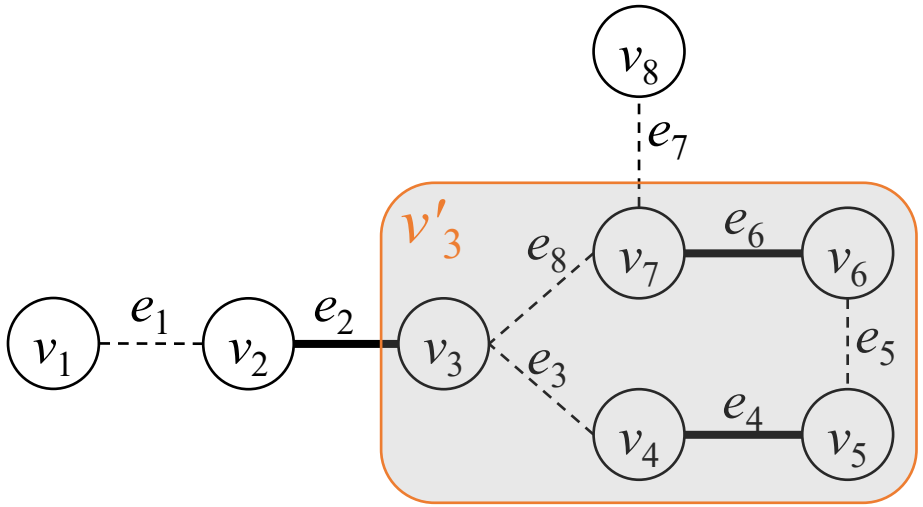
花算法

- 第6轮do-while循环：
  - 从v_1出发，识别出花梗v_1, v_2, v_3和花v_3, v_4, v_5, v_3，花收缩为v'_3
  - 继续访问v'_3的邻点，识别出花梗v_1和花$v_1, v_2, v'_3, v_6, v_7, v_1$，花收缩为$v'_1$
  - 继续访问v'_1的邻点，找到M增广路v'_1, v_8, v_9, v_{10}
  - 将v'_1还原为花，从中选择v_1-v_7路v_1, v_2, v'_3, v_6, v_7，组成M增广路$v_1, v_2, v'_3, v_6, v_7, v_8, v_9, v_{10}$
  - 将v'_3还原为花，从中选择v_3-v_5路v_3, v_4, v_5，组成M增广路$v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}$
  - 得到匹配$M \leftarrow \{e_1, e_3, e_6, e_9, e_{11}, e_{13}\}$



花算法

- 正确性：
  - 只有花影响匈牙利算法的正确性（可分情况讨论证明）
  - 花的收缩不影响增广路的存在性（可分情况讨论证明）
  - 新图中的增广路可还原为原图中的增广路



花算法

- 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：$O(n(n+m))$
  - 循环的轮数：$O(n)$
- 若采用合适的数据结构处理花的收缩，则花算法的时间复杂度可降为$O(n^3)$

请认真完成课后练习