图算法篇：近似算法

北京航空航天大学
计算机学院

提纲

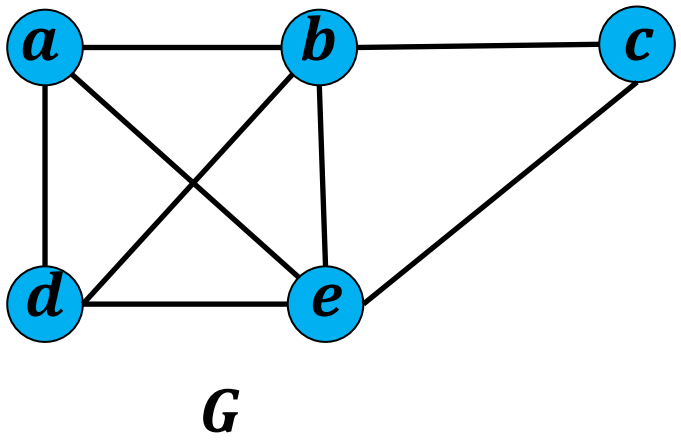
- 优化问题与判定问题
- 基本概念
- 顶点覆盖问题
- 旅行商问题
- 集合覆盖问题

判定问题

- 判定问题： 仅有两种答案： “是”或“否”（yes or no, 1 或 0）
  - DMST
  - DCLIQUE
  - DVC
  - DIS

判定问题

- 判定问题： 仅有两种答案： “是”或“否”（yes or no, 1 或 0）
  - DMST
  - DCLIQUE
  - DVC
  - DIS

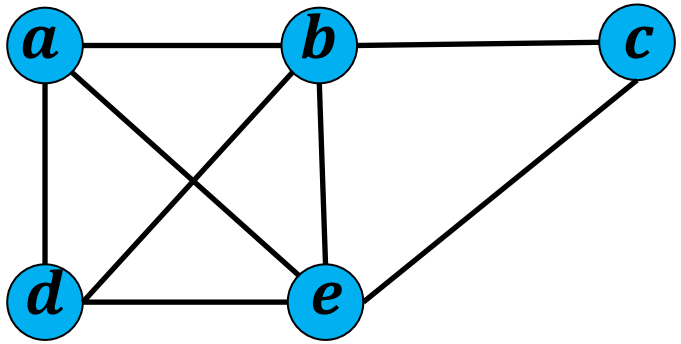


G

✓ $\{a, b, d, e\}$是图 G 的一个规模为 4 的团

判定问题

- 判定问题： 仅有两种答案： “是”或“否”（yes or no, 1 或 0）
  - DMST
  - DCLIQUE
  - DVC
  - DIS

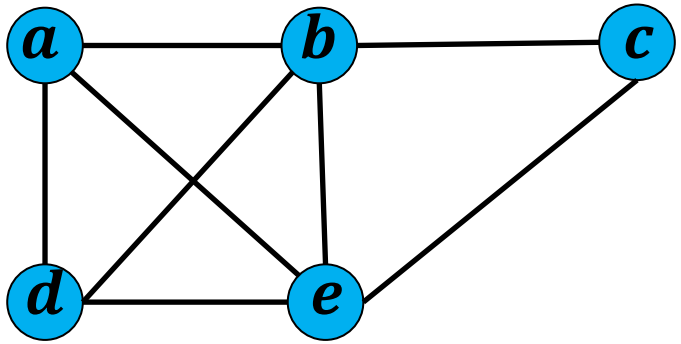


G

- ✓ $\{a, b, d, e\}$是图 G 的一个规模为 4 的团
- ✓ $\{b, d, e\}$是图 G 的一个规模为 3 的顶点覆盖

判定问题

- 判定问题： 仅有两种答案： “是”或“否”（yes or no, 1 或 0）
  - DMST
  - DCLIQUE
  - DVC
  - DIS



G

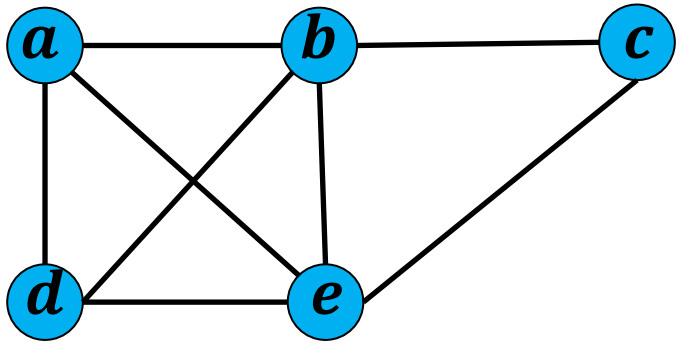
- ✓ $\{a, b, d, e\}$是图 G 的一个规模为 4 的团
- ✓ $\{b, d, e\}$是图 G 的一个规模为 3 的顶点覆盖
- ✓ $\{a, c\}$ 是图 G 的 一个规模为 2 的独立集

判定问题

- 判定问题： 仅有两种答案： “是”或“否”（yes or no, 1 或 0）
  - DMST
  - DCLIQUE
  - DVC
  - DIS

→ 最小生成树问题
→ 最大团问题
→ 最小顶点覆盖问题
→ 最大独立集问题

优化问题



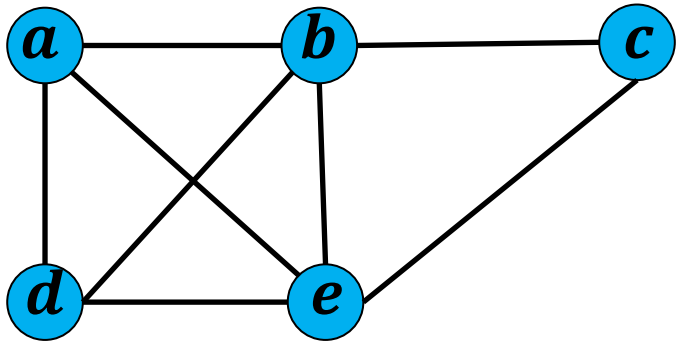
G

- ✓ $\{a, b, d, e\}$是图 G 的一个规模为 4 的团
- ✓ $\{b, d, e\}$是图 G 的一个规模为 3 的顶点覆盖
- ✓ $\{a, c\}$ 是图 G 的 一个规模为 2 的独立集

判定问题

- 判定问题： 仅有两种答案： “是”或“否”（yes or no, 1 或 0）
  - DMST → 最小生成树问题
  - DCLIQUE → 最大团问题
  - DVC → 最小顶点覆盖问题
  - DIS → 最大独立集问题

优化问题



G

- ✓ $\{a, b, d, e\}$是图 G 的一个最大团
- ✓ $\{b, d, e\}$是图 G 的一个最小顶点覆盖
- ✓ $\{a, c\}$ 是图 G 的 一个最大独立集

优化问题（Optimization problems）

- 优化问题
  - 每个可行解（feasible solution）都有一个相关的值
  - 优化问题的目标是找出一个具有最佳值的可行解。

优化问题（Optimization problems）

- 优化问题
  - 每个可行解（feasible solution）都有一个相关的值
  - 优化问题的目标是找出一个具有最佳值的可行解。

例：最小生成树问题（MST）

- 输入：加权无向图 G
- 输出： G的最小生成树 T

可行解： G的生成树 T

可行解的值：生成树 T 中所有边的权重的和

优化问题（Optimization problems）

- 优化问题
  - 每个可行解（feasible solution）都有一个相关的值
  - 优化问题的目标是找出一个具有最佳值的可行解。

例：0-1背包问题（0-1 knapsack）

- 输入：n 个物品的集合 $X = \{1,2,\dots,n\}$ 和背包容量 W，其中，每个物品 i 具有权重 w_i和价值v_i $(i = 1,2,\dots,n)$
- 输出：能装入背包的价值和最大的物品子集 $T \subseteq X$，即 $\sum_{i\in T} w_i \le W$ 且 $\sum_{i\in T} v_i = max\{\sum_{i\in T'} v_i \,|T' \subseteq X,\ \sum_{i\in T'} w_i \le W\}$。

优化问题（Optimization problems）

- 优化问题
  - 每个可行解（feasible solution）都有一个相关的值
  - 优化问题的目标是找出一个具有最佳值的可行解。

例：0-1背包问题（0-1 knapsack）

- 输入：n 个物品的集合 $X = \{1,2,\dots,n\}$ 和背包容量 W，其中，每个物品 i 具有权重 w_i和价值v_i $(i = 1,2,\dots,n)$
- 输出：能装入背包的价值和最大的物品子集 $T \subseteq X$，即 $\sum_{i\in T} w_i \le W$ 且 $\sum_{i\in T} v_i = max\{\sum_{i\in T'} v_i \,|T' \subseteq X,\ \sum_{i\in T'} w_i \le W\}$。

可行解：能装入背包的物品子集 $T \subseteq X$，即 $\sum_{i\in T} w_i \le W$。

可行解的值：物品子集的价值和

优化问题与对应的判定问题

- 一个优化问题通常有对应的判定问题

优化问题与对应的判定问题

- 一个优化问题通常有对应的判定问题

例：最小生成树问题（MST）

- 输入：加权无向图 G
- 输出： G 的最小生成树 T

优化问题与对应的判定问题

- 一个优化问题通常有对应的判定问题

例：最小生成树问题（MST）

- 输入：加权无向图 G
- 输出： G 的最小生成树 T

生成树判定问题（DMST）：

- 输入：加权无向图 G，正整数 k
- 问题：G 是否存在边权和不超过 k 的生成树 ？

优化问题与对应的判定问题

- 一个优化问题通常有对应的判定问题

例：0-1背包问题（0-1 knapsack）

- 输入：n 个物品的集合 $X = \{1,2,\dots,n\}$ 和背包容量 W，其中每个物品 i 具有重量w_i和价值v_i $(i = 1,2,\dots,n)$
- 输出：能装入背包的价值和最大的物品子集 $T \subseteq X$，即 $\sum_{i\in T} w_i \le W$ 且 $\sum_{i\in T} v_i = max\{\sum_{i\in T'} v_i \,|T' \subseteq X\}$。

优化问题与对应的判定问题

- 一个优化问题通常有对应的判定问题

例：0-1背包问题（0-1 knapsack）

- 输入：n 个物品的集合 $X = \{1,2,\dots,n\}$ 和背包容量 W，其中每个物品 i 具有重量w_i和价值v_i $(i = 1,2,\dots,n)$
- 输出：能装入背包的价值和最大的物品子集 $T \subseteq X$，即 $\sum_{i\in T} w_i \le W$ 且 $\sum_{i\in T} v_i = max\{\sum_{i\in T'} v_i \mid T' \subseteq X\}$。

例：0-1背包判定问题(0-1 Dknapsack)

- 输入：n 个物品的集合$X = \{1,2,\dots,n\}$，其中每个物品 i 具有重量w_i和价值 $v_i, i = 1,2,\dots,n$，和背包容量 W，正整数 V
- 问题：是否存在能装入背包的价值和至少为 V 的物品子集？

优化问题与对应的判定问题

- 一个优化问题的算法通常可用来求解对应的判定问题

优化问题与对应的判定问题

- 一个优化问题的算法通常可用来求解对应的判定问题

例：用最小生成树问题的Kruskal算法解决生成树判定问题

生成树判定问题（DMST）：

- 输入：加权无向图 G，正整数 k
- 问题：G 是否存在边权和不超过 k 的生成树 ？

优化问题与对应的判定问题

- 一个优化问题的算法通常可用来求解对应的判定问题

例：用最小生成树问题的Kruskal算法解决生成树判定问题

1. 用 Kruskal算法求出图 G 的最小生成树 T；
2. 计算最小生成树的边权和 $w(T)$；

生成树判定问题（DMST）：

- 输入：加权无向图 G，正整数 k
- 问题：G 是否存在边权和不超过 k 的生成树 ？

优化问题与对应的判定问题

- 一个优化问题的算法通常可用来求解对应的判定问题

例：用最小生成树问题的Kruskal算法解决生成树判定问题

1. 用 Kruskal算法求出图 G 的最小生成树 T；
2. 计算最小生成树的边权和 $w(T)$；
3. 若 $w(T) \leq k$，则返回“是”；
4. 否则，返回“否”。

生成树判定问题（DMST）：

- 输入：加权无向图 G，正整数 k
- 问题：G 是否存在边权和不超过 k 的生成树？

优化问题与对应的判定问题

- 一个优化问题的算法通常可用来求解对应的判定问题

例：用最小生成树问题的Kruskal算法解决生成树判定问题

1. 用 Kruskal算法求出图 G 的最小生成树 T；
2. 计算最小生成树的边权和 $w(T)$；
3. 若 $w(T) \leq k$，则返回“是”；
4. 否则，返回“否”。

算法是正确：

- 若 $w(T) \leq k$，显然 T 为边权和不超过 k 的生成树
- 若 $w(T) > k$，则 G 的所有生成树的边权和均大于 k

优化问题与对应的判定问题

- 一个优化问题的算法通常可用来求解对应的判定问题

例：用最小生成树问题的Kruskal算法解决生成树判定问题

1. 用 Kruskal算法求出图 G 的最小生成树 T；
2. 计算最小生成树的边权和 $w(T)$；
3. 若 $w(T) \leq k$，则返回“是”；
4. 否则，返回“否”。

算法是正确：

- 若 $w(T) \leq k$，显然 T 为边权和不超过 k 的生成树
- 若 $w(T) > k$，则 G 的所有生成树的边权和均大于 k

判定问题不会比对应的优化问题更难

优化问题的难度

- 最小生成树问题存在多项式时间算法
  - Prim算法
  - Kruskal算法

优化问题的难度

- 最小生成树问题存在多项式时间算法
  - Prim算法
  - Kruskal算法
- 以下问题是否存在多项式时间算法？
  - 最大团问题
  - 最小顶点覆盖问题
  - 最大独立集问题

优化问题的难度

- 最小生成树问题存在多项式时间算法
  - Prim算法
  - Kruskal算法
- 以下问题是否存在多项式时间算法？
  - 最大团问题
  - 最小顶点覆盖问题
  - 最大独立集问题
- 当一个优化问题对应的判定问题是NP难问题时，该优化问题是否有多项式时间算法？

优化问题的难度

- 最小生成树问题存在多项式时间算法
  - Prim算法
  - Kruskal算法
- 以下问题是否存在多项式时间算法？
  - 最大团问题
  - 最小顶点覆盖问题
  - 最大独立集问题
- 当一个优化问题对应的判定问题是NP难问题时，该优化问题是否有多项式时间算法？
- 近似算法是解决一类NP难优化问题的一种途径
  - 这里未给出NP难优化问题的严谨的定义

近似算法

提纲

- 优化问题与判定问题
- 基本概念
- 顶点覆盖问题
- 旅行商问题
- 集合覆盖问题

近似比（ Approximation ratio ）

- 给定优化问题 Q、近似算法 A、及 Q的任意一个规模为 n的实例 x
  - C：近似算法 A 在实例 x 返回的解的值
  - C^*：实例 x 的最优解

近似比（ Approximation ratio ）

- 给定优化问题 Q、近似算法 A、及 Q的任意一个规模为 n的实例 x
  - C：近似算法 A 在实例 x 返回的解的值
  - C^*：实例 x 的最优解

近似算法 A 的近似比 $r(n)$定义为：

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq r(n)$$

$r(n) \geq 1$

近似比（ Approximation ratio ）

- 给定优化问题 Q、近似算法 A、及 Q的任意一个规模为 n的实例 x
  - C：近似算法 A 在实例 x 返回的解的值
  - C^*：实例 x 的最优解

近似算法 A 的近似比 $r(n)$定义为：

$$\max(\frac{C}{C^*}, \frac{C^*}{C}) \leq r(n)$$

$r(n) \geq 1$

- 最大优化问题：$1 \leq \frac{C^*}{C} \leq r(n)$
- 最小优化问题：$1 \leq \frac{C}{C^*} \leq r(n)$

近似比（ Approximation ratio ）

- 给定优化问题 Q、近似算法 A、及 Q的任意一个规模为 n的实例 x
  - C：近似算法 A 在实例 x 返回的解的值
  - C^*：实例 x 的最优解

  近似算法 A 的近似比 $r(n)$定义为：

$$\max(\frac{C}{C^*}, \frac{C^*}{C}) \leq r(n)$$

$r(n) \geq 1$

  - 最大优化问题： $1 \leq \frac{C^*}{C} \leq r(n)$
  - 最小优化问题： $1 \leq \frac{C}{C^*} \leq r(n)$

- 若 $r(n)$ 不依赖于 n，则 $r(n)$ 为一个常数，写为 r 或 $1+\varepsilon$

提纲

- 背景
- 基本概念
- **最小顶点覆盖问题**
- 集合覆盖问题
- 旅行商问题

顶点覆盖问题

- 定义（顶点覆盖）给定无向图 $G=(V,E)$， G的一个顶点覆盖为 G的顶点子集$V'\subseteq V$，使得

  对于G中任意一条边 $e=(u,v)\in E$， $u\in V'$ 或$v\in V'$。

顶点覆盖问题

- 定义（顶点覆盖）给定无向图 $G=(V,E)$， G的一个顶点覆盖为 G的顶点子集$V'\subseteq V$，使得

  对于G中任意一条边 $e=(u,v)\in E$， $u\in V'$ 或$v\in V'$。

  - V 是 G 的一个顶点覆盖

顶点覆盖问题

- 定义（顶点覆盖）给定无向图 $G=(V,E)$，G的一个顶点覆盖为 G的顶点子集$V'\subseteq V$，使得

  对于G中任意一条边 $e=(u,v)\in E$，$u\in V'$ 或$v\in V'$。

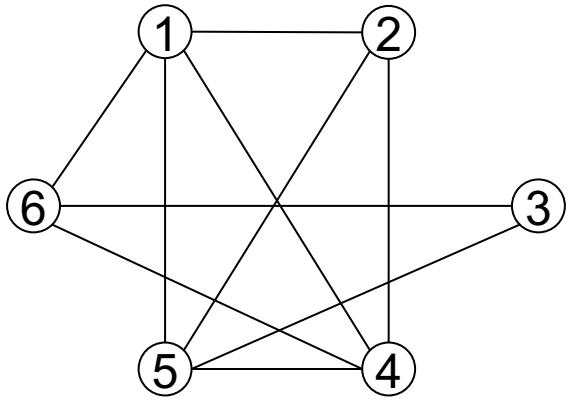
  - V 是 G 的一个顶点覆盖
- 顶点覆盖的规模：包含的顶点的个数

顶点覆盖问题

- 定义（顶点覆盖）给定无向图 $G=(V,E)$， G的一个顶点覆盖为 G的顶点子集$V'\subseteq V$，使得

  对于G中任意一条边 $e=(u,v)\in E$， $u\in V'$ 或$v\in V'$。

  - V 是 G 的一个顶点覆盖
- 顶点覆盖的规模：包含的顶点的个数



G

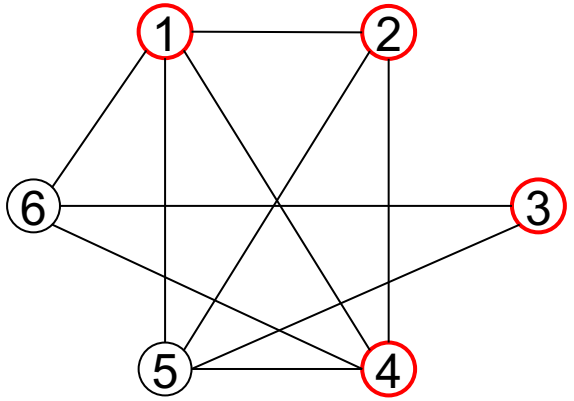
$\{1,2,3,4,5,6\}$ 是 G 的一个顶点覆盖。

顶点覆盖问题

- 定义（顶点覆盖）给定无向图 $G=(V,E)$， G的一个顶点覆盖为 G的顶点子集$V'\subseteq V$，使得

  对于G中任意一条边 $e=(u,v)\in E$， $u\in V'$ 或$v\in V'$。

  - V 是 G 的一个顶点覆盖
- 顶点覆盖的规模：包含的顶点的个数



G

$\{1,2,3,4,5,6\}$ 是 G 的一个顶点覆盖。

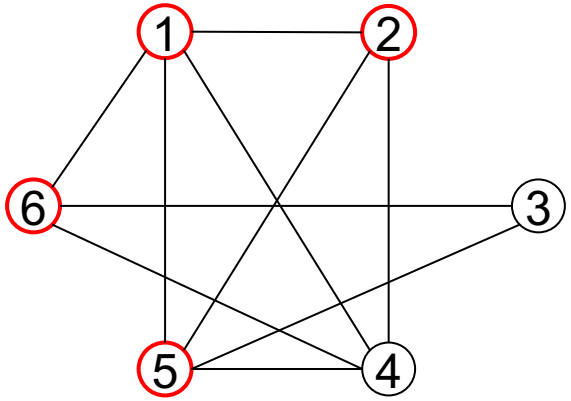
$\{1,2,3,4\}$ 是 G 的一个顶点覆盖。

顶点覆盖问题

- 定义（顶点覆盖）给定无向图 $G=(V,E)$， G的一个顶点覆盖为 G的顶点子集$V'\subseteq V$，使得

  对于G中任意一条边 $e=(u,v)\in E$， $u\in V'$ 或$v\in V'$。

  - V 是 G 的一个顶点覆盖
- 顶点覆盖的规模：包含的顶点的个数



G

$\{1,2,3,4,5,6\}$ 是 G 的一个顶点覆盖。

$\{1,2,3,4\}$ 是 G 的一个顶点覆盖。

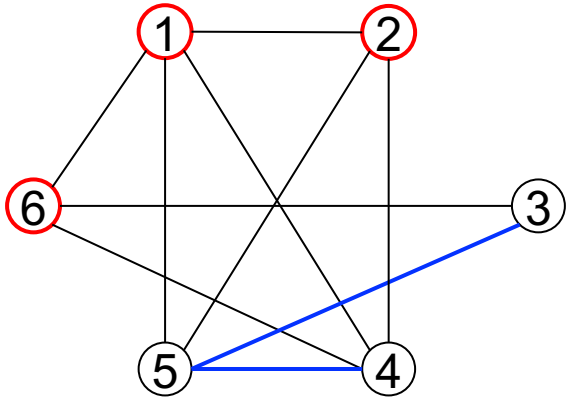
$\{1,2,5,6\}$ 是 G 的一个顶点覆盖。

顶点覆盖问题

- 定义（顶点覆盖）给定无向图 $G=(V,E)$， G的一个顶点覆盖为 G的顶点子集$V'\subseteq V$，使得

  对于G中任意一条边 $e=(u,v)\in E$， $u\in V'$ 或$v\in V'$。

  - V 是 G 的一个顶点覆盖
- 顶点覆盖的规模：包含的顶点的个数



G

$\{1,2,3,4,5,6\}$ 是 G 的一个顶点覆盖。

$\{1,2,3,4\}$ 是 G 的一个顶点覆盖。

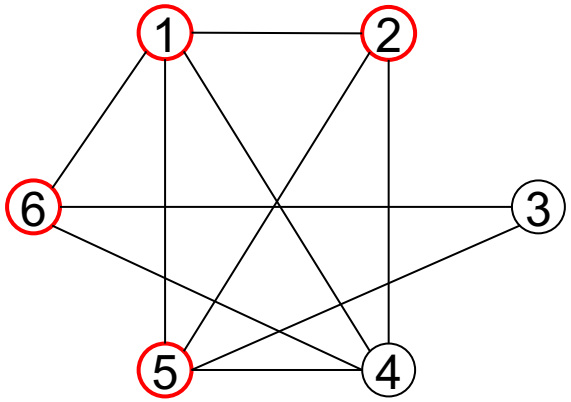
$\{1,2,5,6\}$ 是 G 的一个顶点覆盖。

$\{1,2,6\}$ 不是 G 的顶点覆盖。

最小顶点覆盖问题

- 最小顶点覆盖问题
  - 输入：无向图 G
  - 输出： G的最小规模的顶点覆盖

最小顶点覆盖也称为最优顶点覆盖



G

$\{1,2,3,4,5,6\}$ 是 G 的一个顶点覆盖。

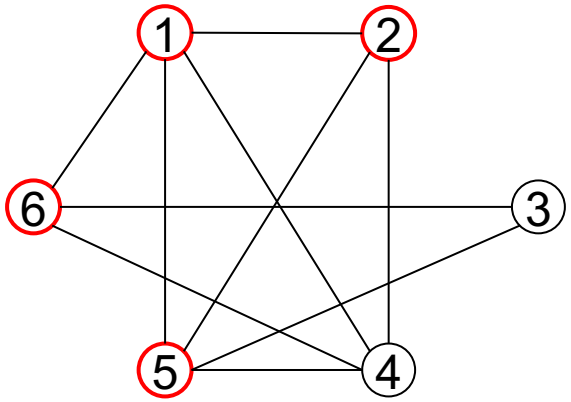
$\{1,2,3,4\}$ 是 G 的一个顶点覆盖。

$\{1,2,5,6\}$ 是 G 的一个顶点覆盖。

最小顶点覆盖问题

- 最小顶点覆盖问题
  - 输入：无向图 G
  - 输出： G的最小规模的顶点覆盖

最小顶点覆盖也称为最优顶点覆盖



G

$\{1,2,3,4,5,6\}$ 是 G 的一个顶点覆盖。

$\{1,2,3,4\}$ 是 G 的一个顶点覆盖。

$\{1,2,5,6\}$ 是 G 的一个顶点覆盖。

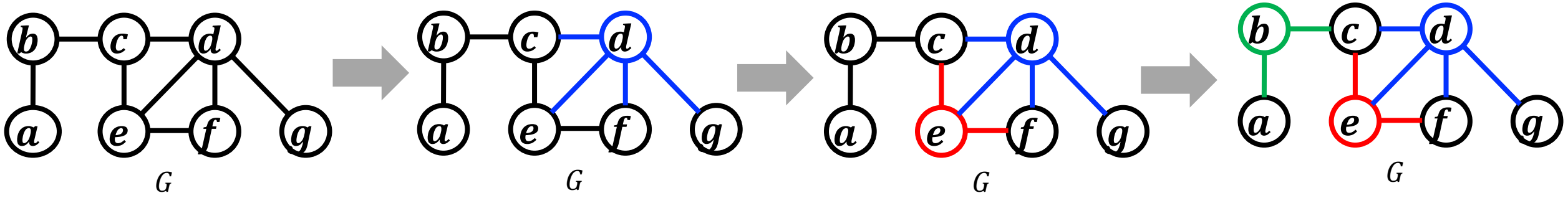
$\{1,2,5,6\}$、 $\{1,2,3,4\}$均是 是 G 的最小顶点覆盖。

贪心算法

- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。

运行实例

- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。

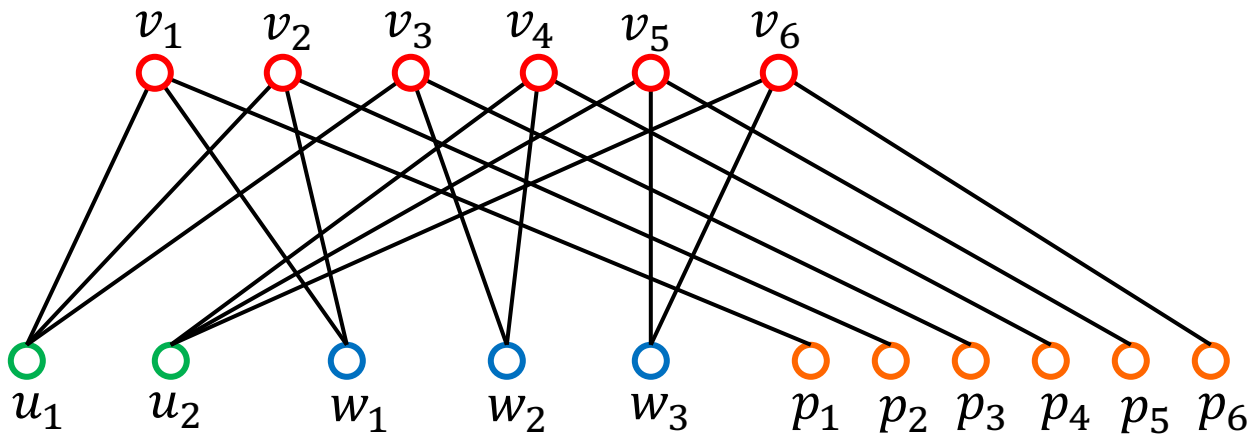


- 算法输出顶点覆盖 $C = \{d, e, b\}$
- C 是最优解：任意两个顶点的子集都不是顶点覆盖
- 问题：近似比 $r(n) = 1$？

  该算法甚至没有常数的近似比。

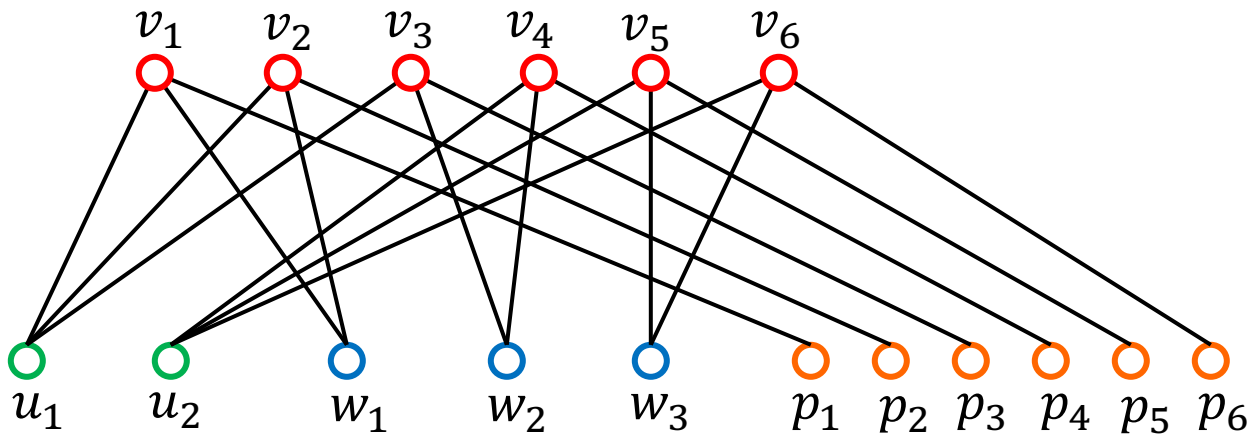
例子

- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。



例子

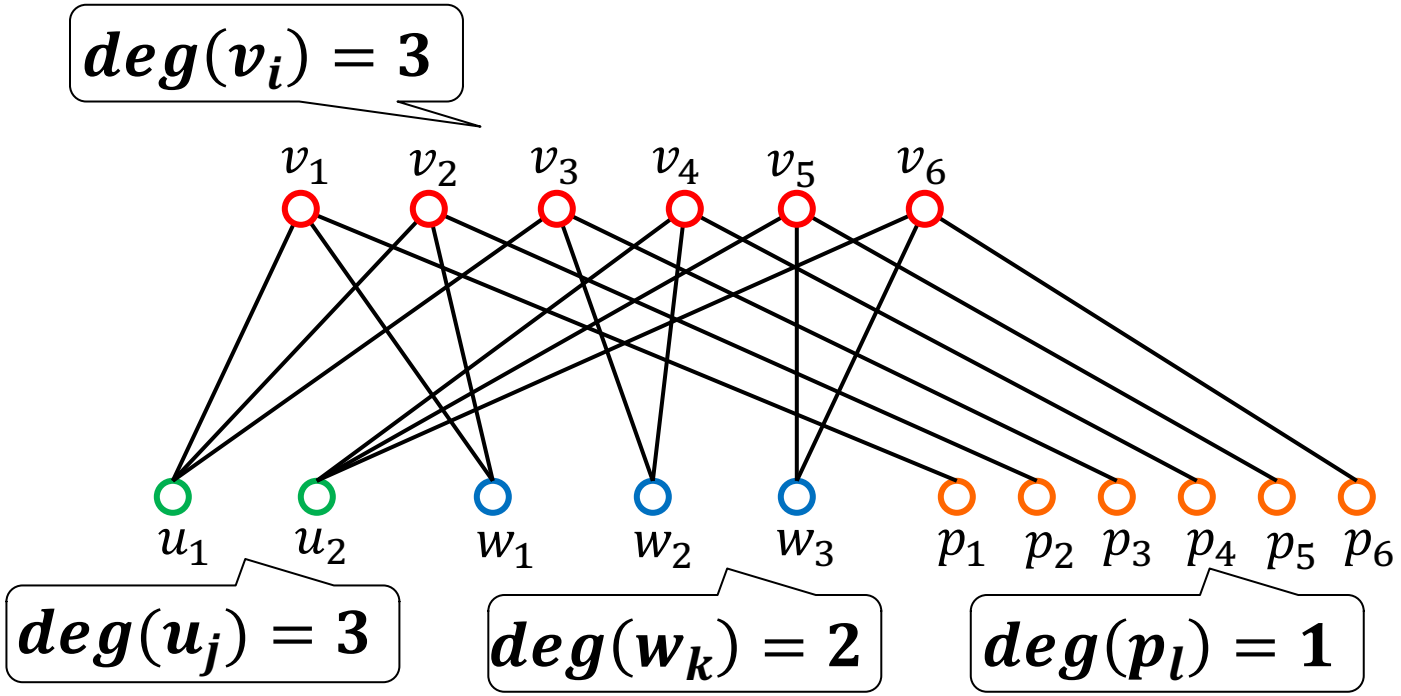
- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。



- 最优解 $=\{v_1, v_2, v_3, v_4, v_5, v_6\}$， OPT=6

例子

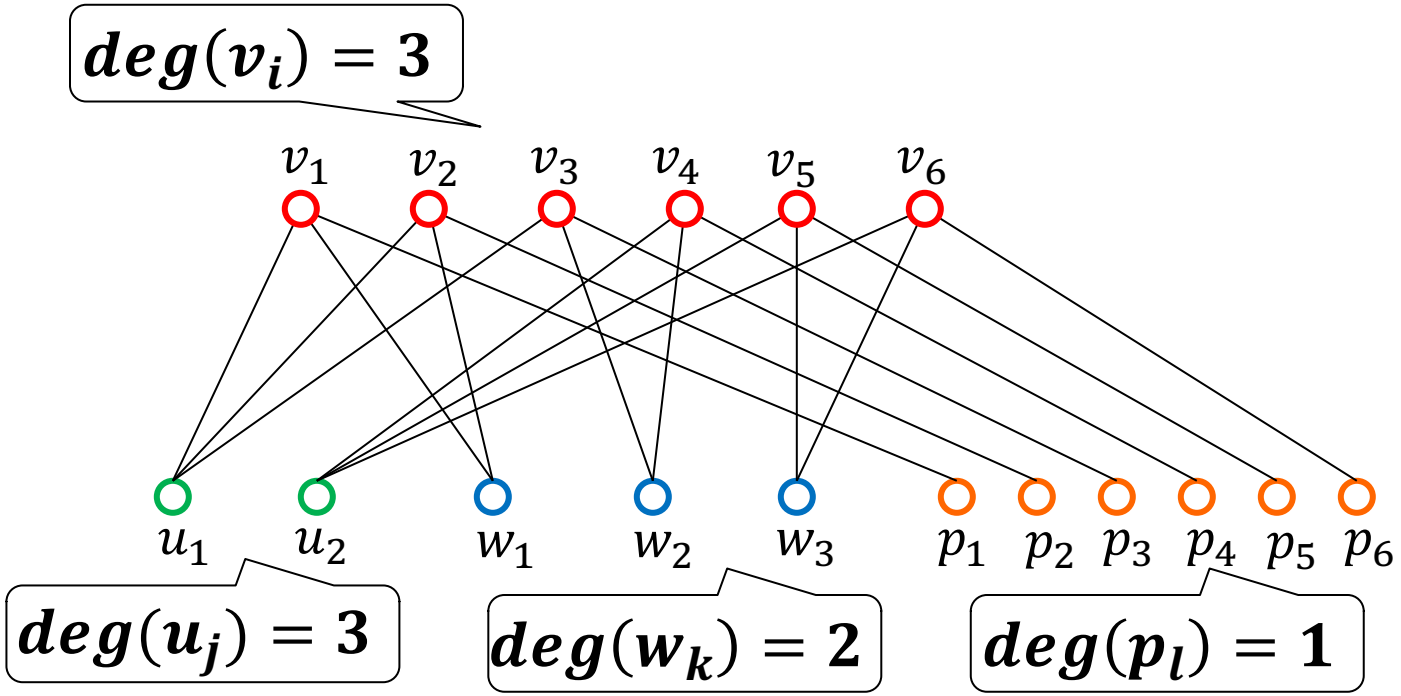
- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。



- 最优解 $=\{v_1, v_2, v_3, v_4, v_5, v_6\}$， OPT=6

例子

- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。



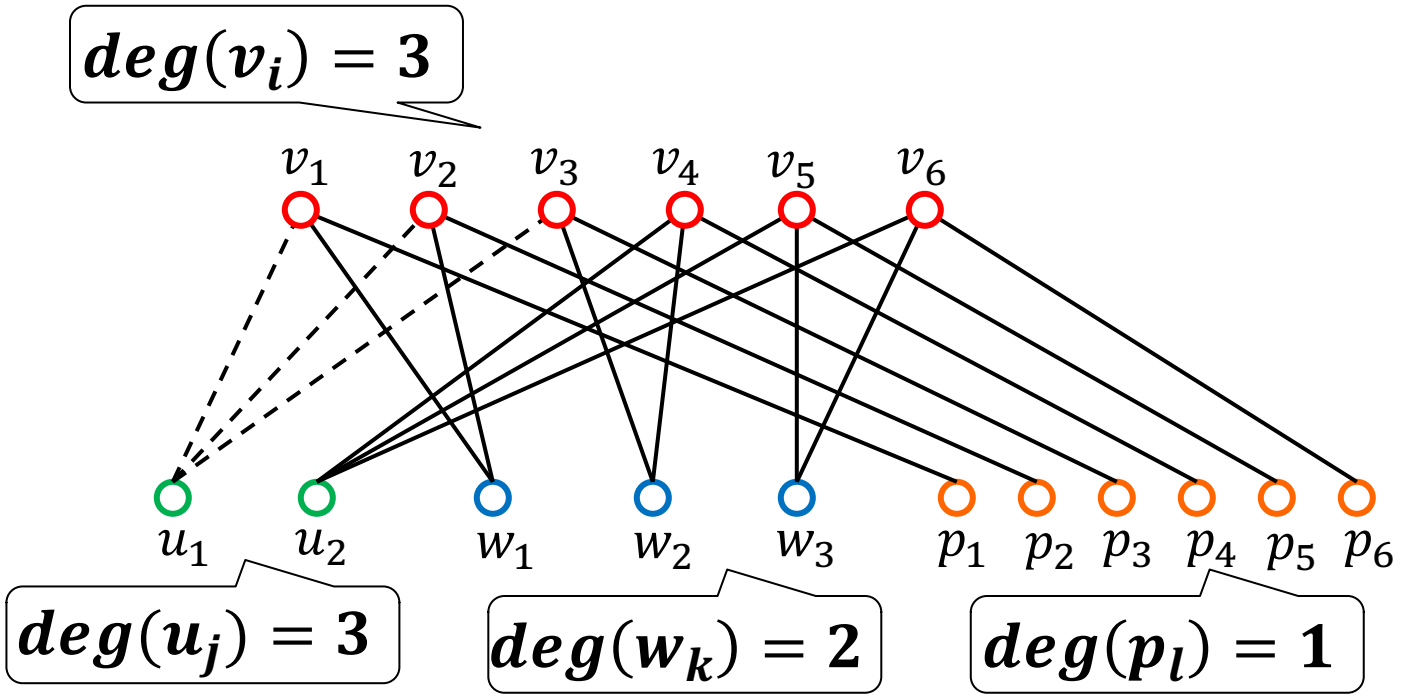
算法运行过程：

1. 选择 u_1

- 最优解 $=\{v_1, v_2, v_3, v_4, v_5, v_6\}$， OPT=6

例子

- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。



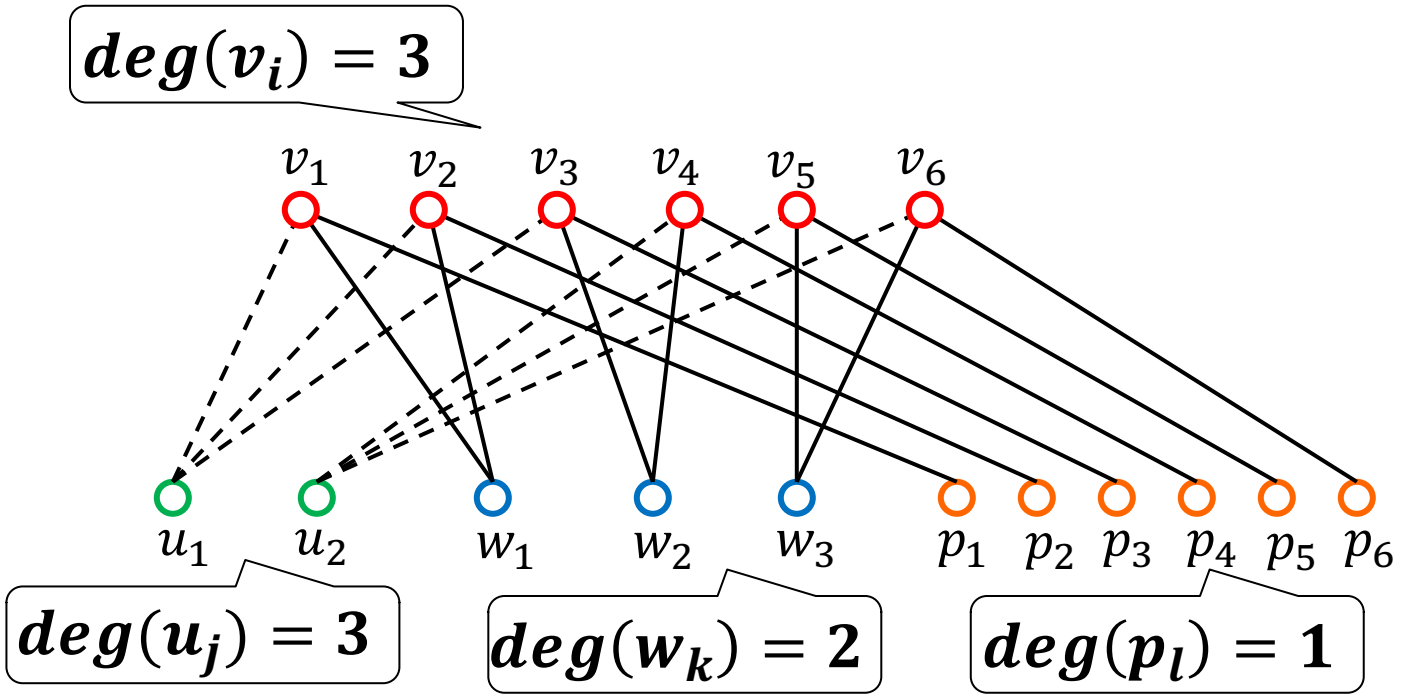
算法运行过程：

1. 选择 u_1
2. 选择 u_2

- 最优解 $=\{v_1, v_2, v_3, v_4, v_5, v_6\}$， OPT=6

例子

- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。



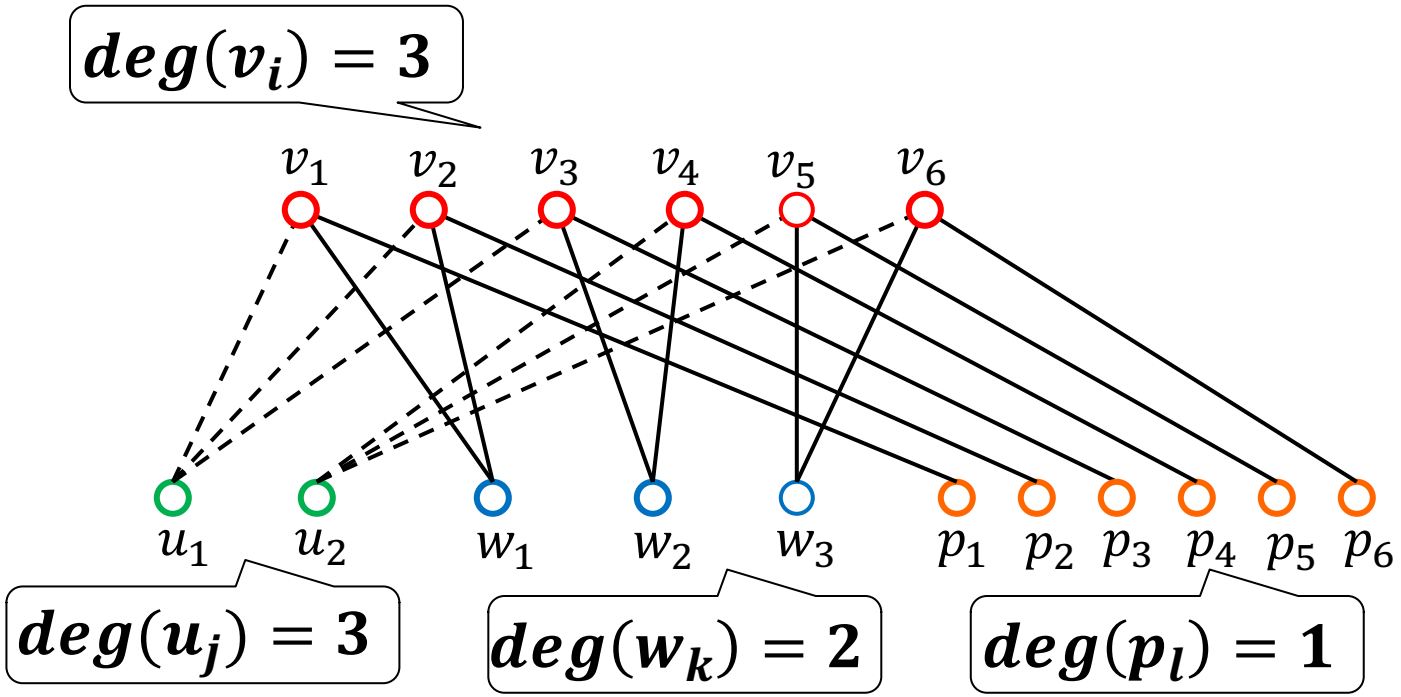
算法运行过程：

1. 选择 u_1
2. 选择 u_2

- 最优解 $=\{v_1, v_2, v_3, v_4, v_5, v_6\}$， OPT=6

例子

- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。



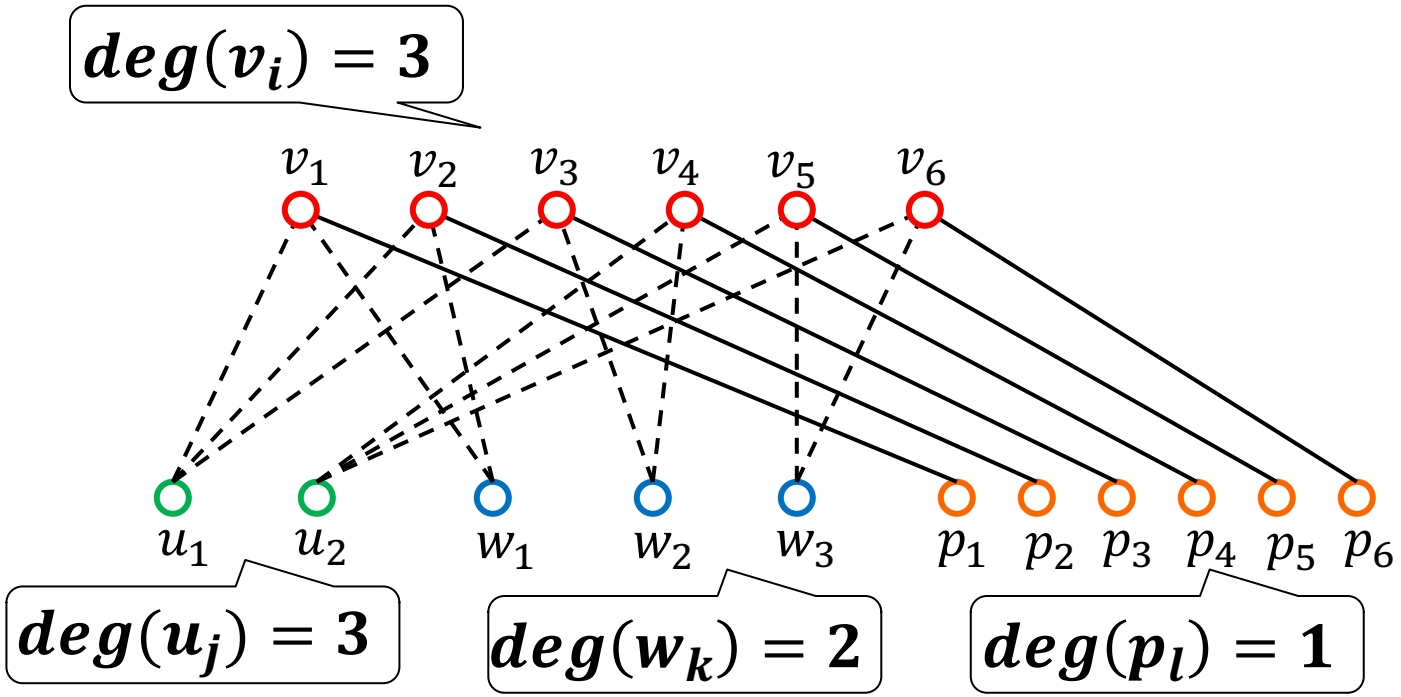
算法运行过程：

1. 选择 u_1
2. 选择 u_2
3. 依次选择w_1, w_2, w_3

- 最优解 $=\{v_1, v_2, v_3, v_4, v_5, v_6\}$， OPT=6

例子

- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。



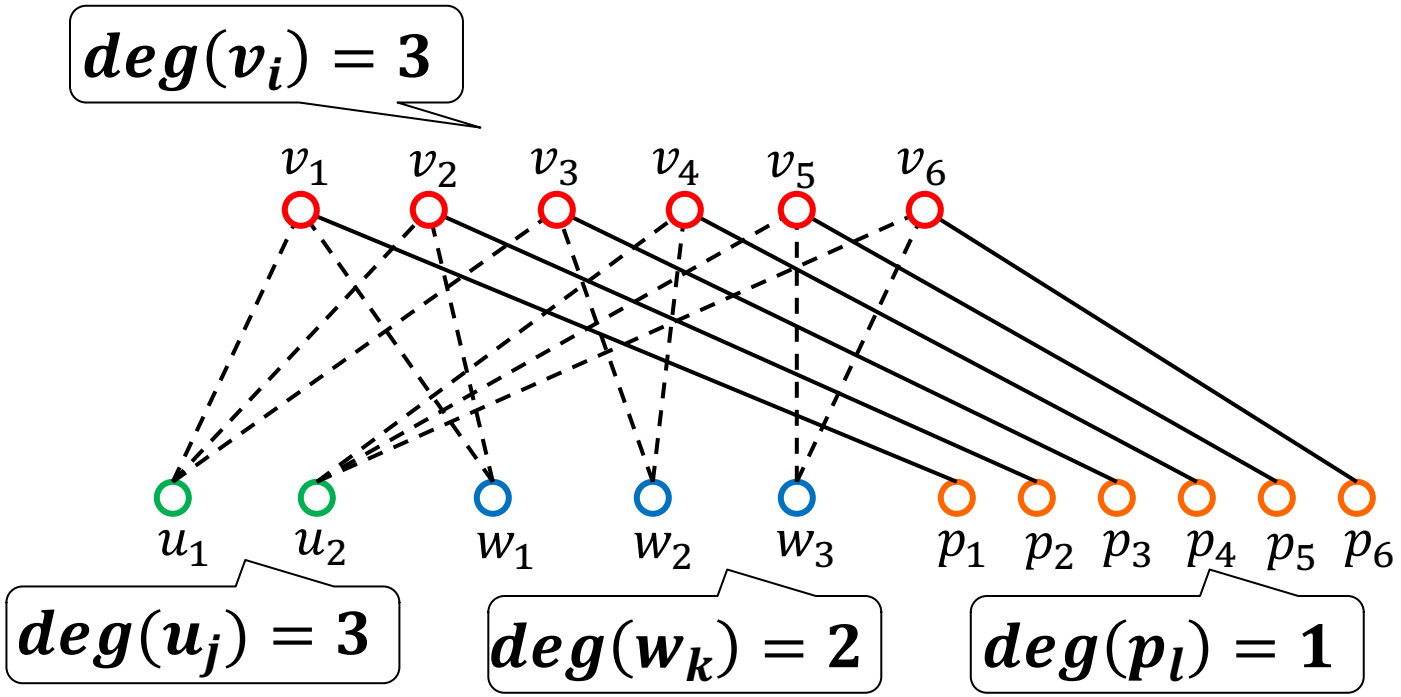
算法运行过程：

1. 选择 u_1
2. 选择 u_2
3. 依次选择w_1, w_2, w_3

- 最优解 $=\{v_1, v_2, v_3, v_4, v_5, v_6\}$， OPT=6

例子

- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。



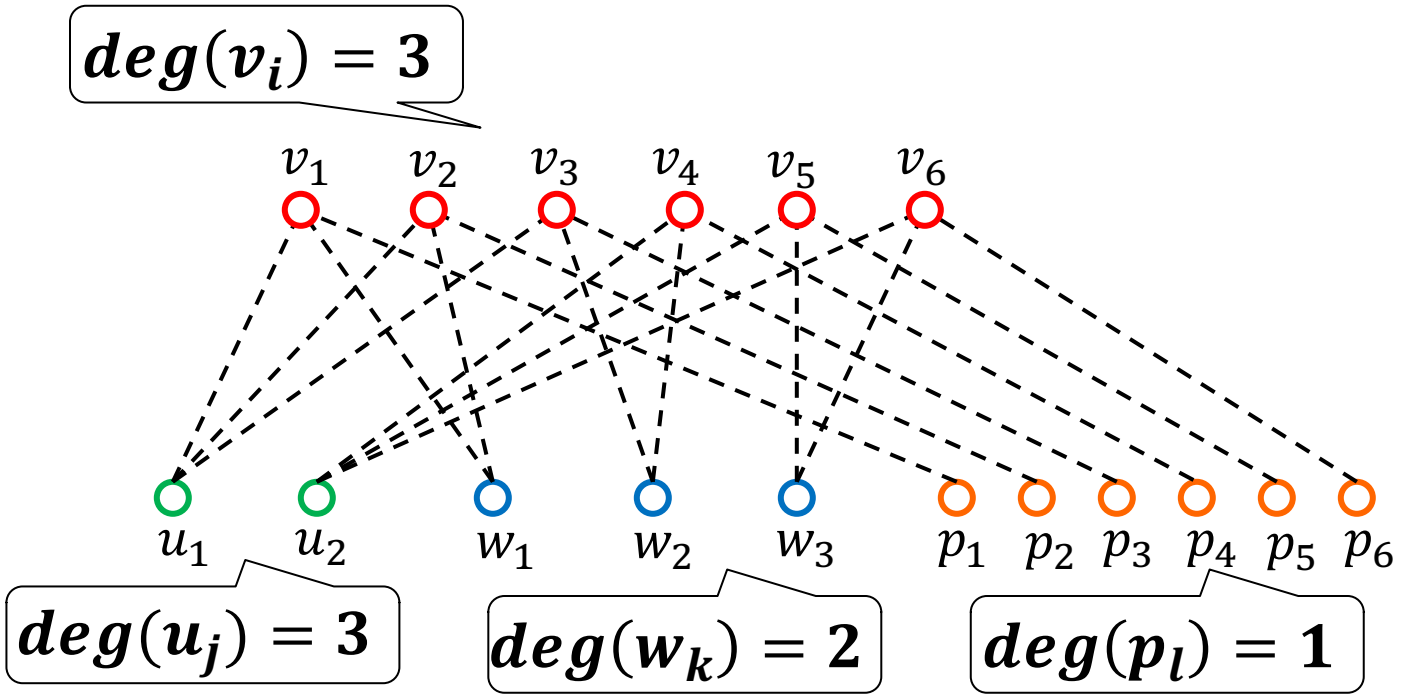
算法运行过程：

1. 选择 u_1
2. 选择 u_2
3. 依次选择w_1, w_2, w_3
4. 依次选择$p_1, \ldots, p_6$

- 最优解 $=\{v_1, v_2, v_3, v_4, v_5, v_6\}$， OPT=6

例子

- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。



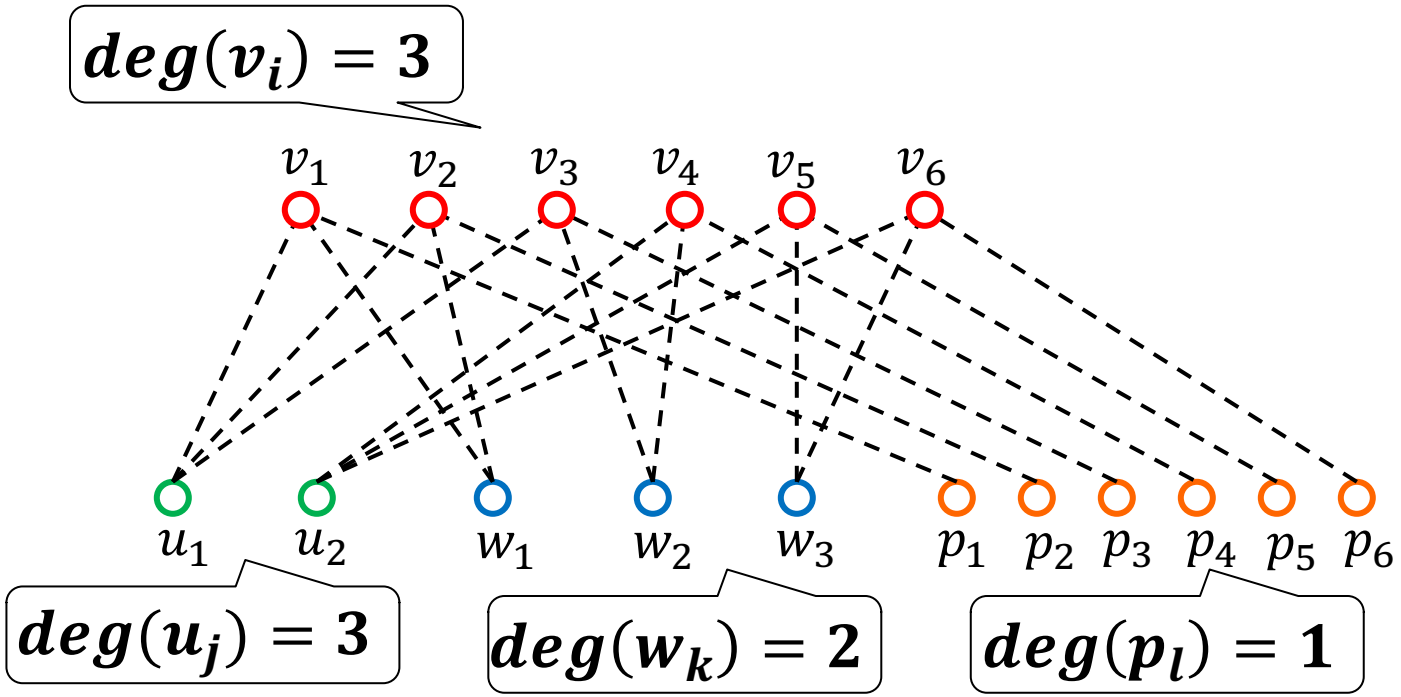
算法运行过程：

1. 选择 u_1
2. 选择 u_2
3. 依次选择w_1, w_2, w_3
4. 依次选择$p_1, \ldots, p_6$

- 最优解 $=\{v_1, v_2, v_3, v_4, v_5, v_6\}$， OPT=6

例子

- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。



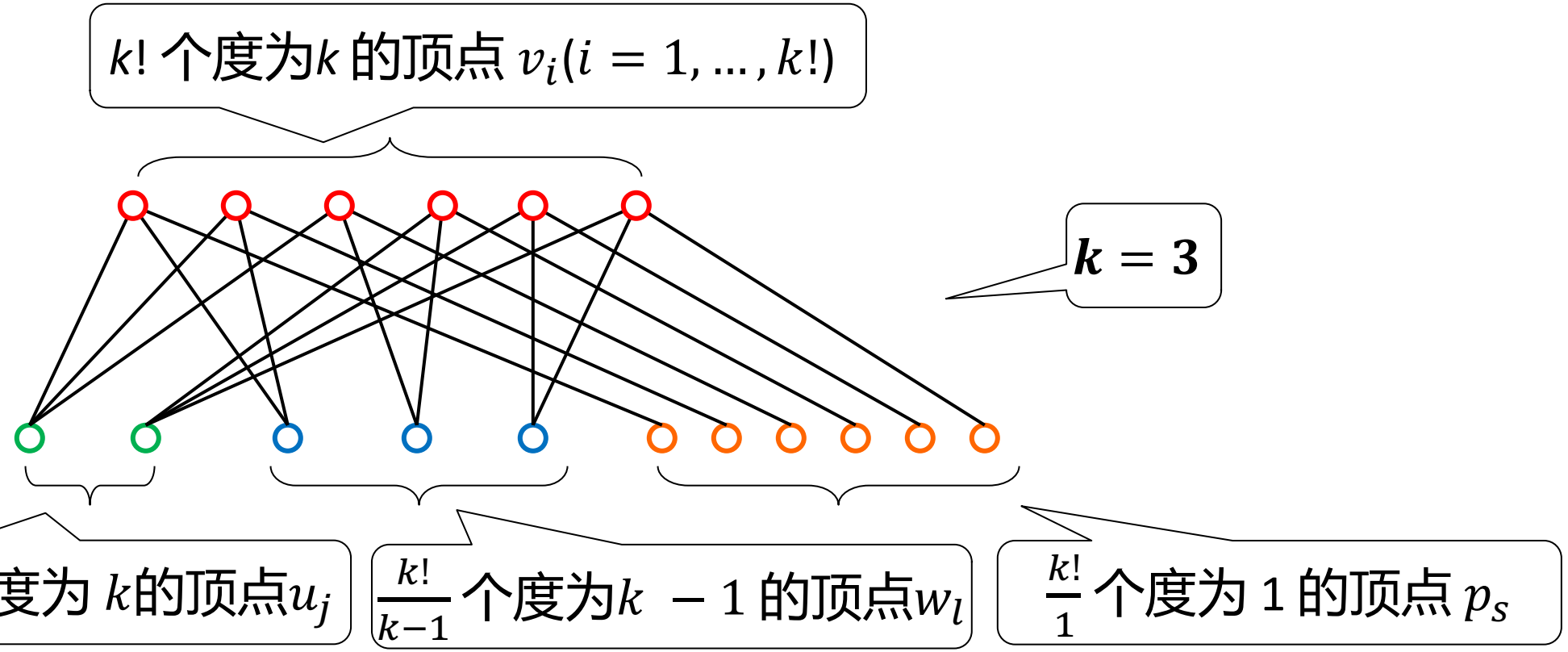
算法运行过程：

1. 选择 u_1
2. 选择 u_2
3. 依次选择w_1, w_2, w_3
4. 依次选择$p_1, \ldots, p_6$

- 最优解 $=\{v_1, v_2, v_3, v_4, v_5, v_6\}$， OPT=6
- 算法输出 $\{u_1, u_2, w_1, w_2, w_3, p_1, p_2, p_3, p_4, p_5, p_6\}$， SOL=11

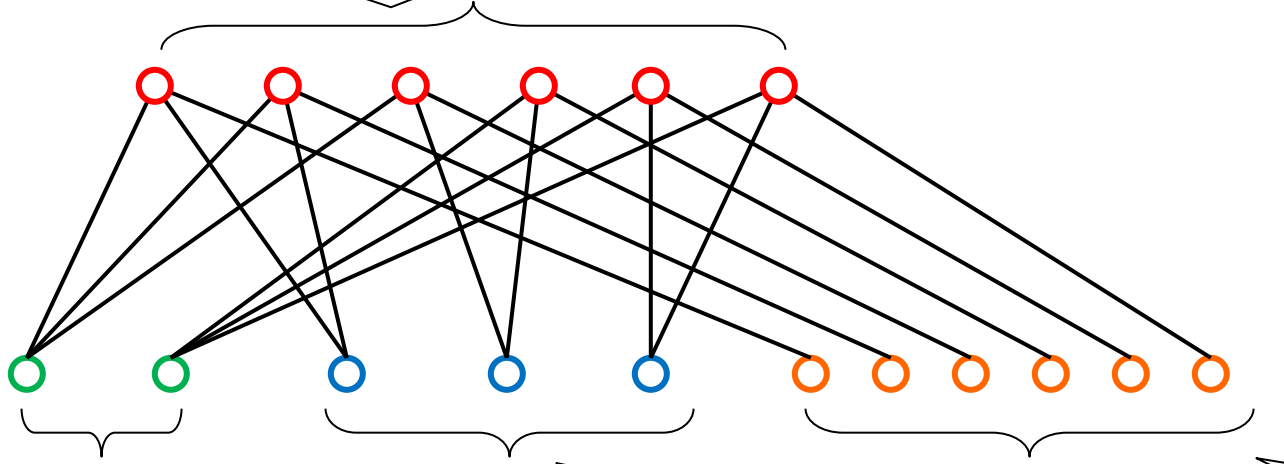
例子推广至一般形式

- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。



例子推广至一般形式

- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。



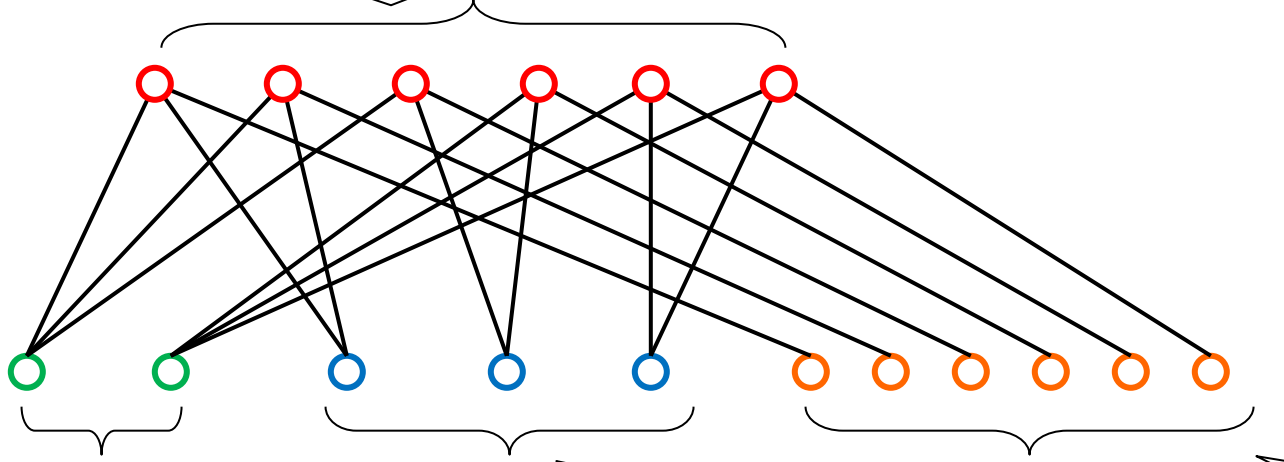
算法运行过程：

1. 依次选择 $u_1, \dots, u_{\frac{k!}{k}}$
2. 依次选择$w_1, \dots, w_{\frac{k!}{k-1}}$
3. 依次选择$p_1, \dots p_{\frac{k!}{1}}$

- 最优解 $=\{v_1, v_2, \dots, v_{k!}\}$， OPT$= k!$

例子推广至一般形式

- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。



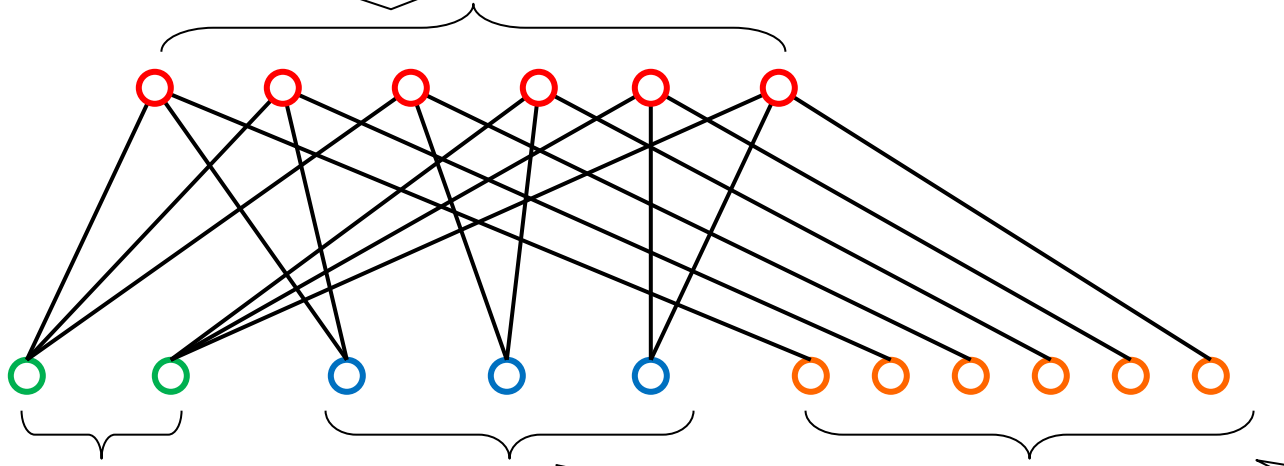
算法运行过程：

1. 依次选择 $u_1, \dots, u_{\frac{k!}{k}}$
2. 依次选择$w_1, \dots, w_{\frac{k!}{k-1}}$
3. 依次选择$p_1, \dots p_{\frac{k!}{1}}$

- 最优解 $=\{v_1, v_2, \dots, v_{k!}\}$， OPT$= k!$
- 算法输出 $\{u_1, \dots, u_{\frac{k!}{k}}, w_1, \dots, w_{\frac{k!}{k-1}}, p_1, \dots p_{\frac{k!}{1}}\}$

例子推广至一般形式

- 算法思想：每次找到一个顶点，使得它覆盖最大数目的未覆盖边。



算法运行过程：

1. 依次选择 $u_1, \dots, u_{\frac{k!}{k}}$
2. 依次选择$w_1, \dots, w_{\frac{k!}{k-1}}$
3. 依次选择$p_1, \dots p_{\frac{k!}{1}}$

- 最优解 $=\{v_1, v_2, \dots, v_{k!}\}$， OPT$= k!$
- 算法输出 $\{u_1, \dots, u_{\frac{k!}{k}}, w_1, \dots, w_{\frac{k!}{k-1}}, p_1, \dots p_{\frac{k!}{1}}\}$， SOL$= k!\left(\frac{1}{k} + \frac{1}{k-1} + \frac{1}{k-2} + \cdots + 1\right)$

$\approx k!\,(\log(k))$

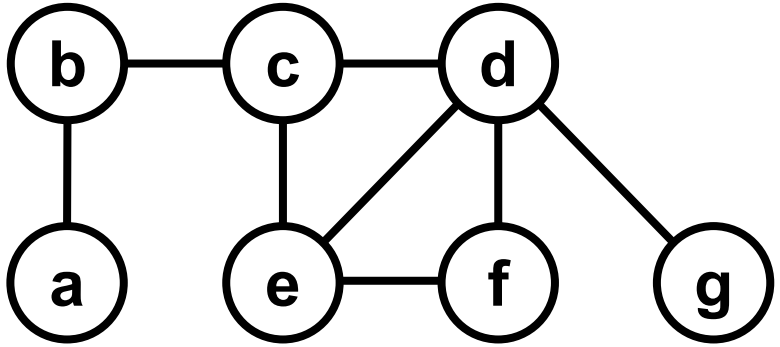
常数近似比算法

Approx-Vertex-Cover(G)

Input: A graph $G = (V, E)$
Output: A set of vertices C
$C \leftarrow \emptyset$
$E' \leftarrow E$
while $E' \neq \emptyset$
    let (u, v) be an arbitrary edge of E'
    $C \leftarrow C \cup \{u, v\}$
    remove from E' every edge incident on either u or v
return C

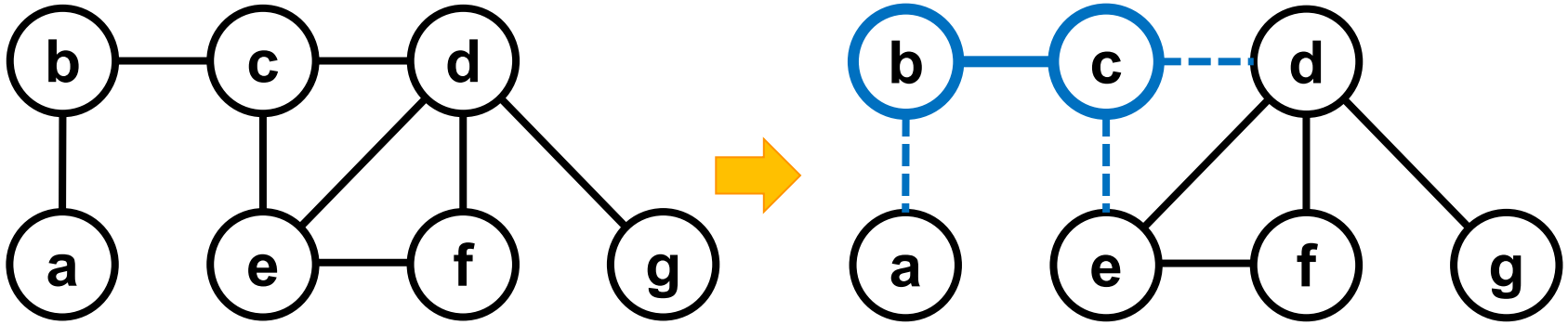
- 时间复杂度：$O(|V| + |E|)$

运行实例



(1)

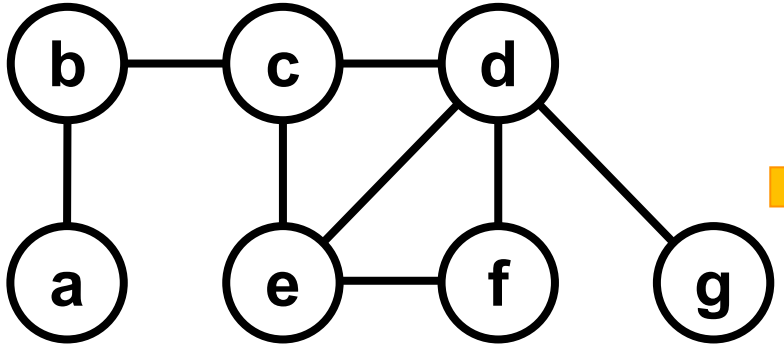
运行实例



(1)

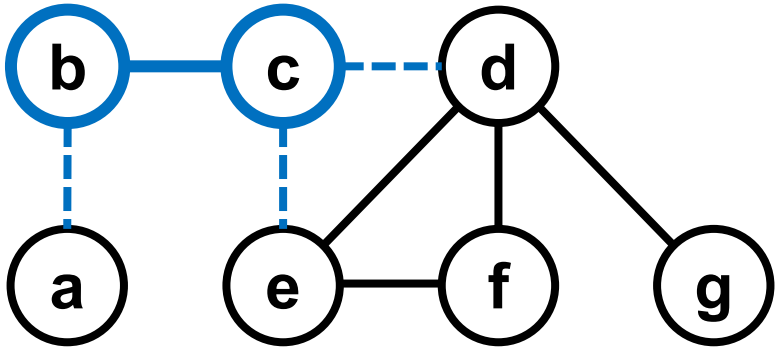
(2)

运行实例



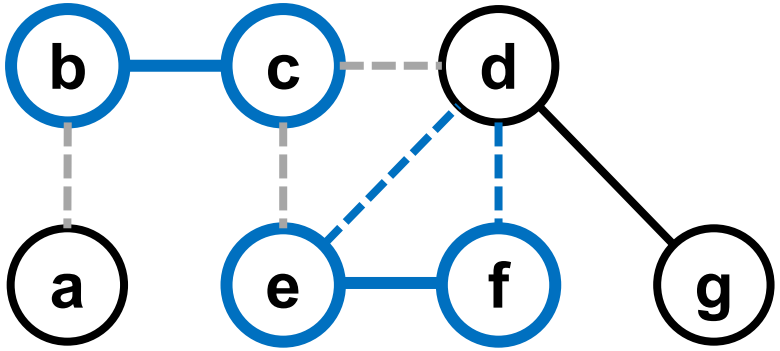
(1)

$C = \{b, c\}$



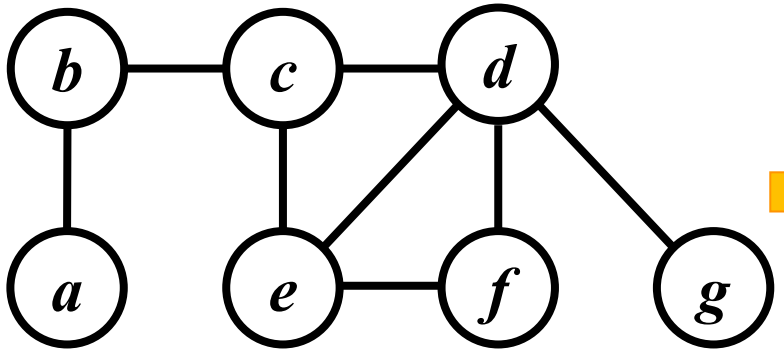
(2)

$C = \{b, c, e, f\}$



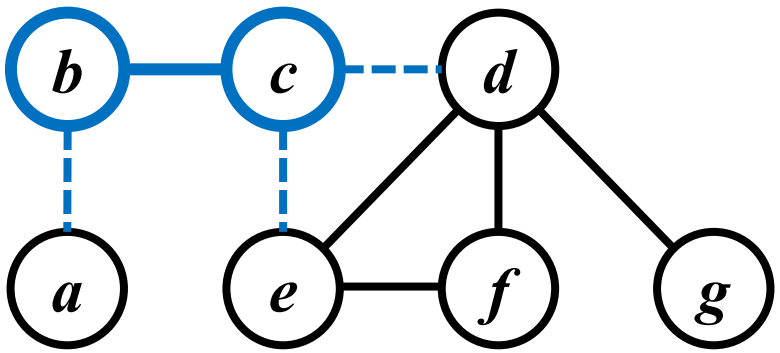
(3)

运行实例



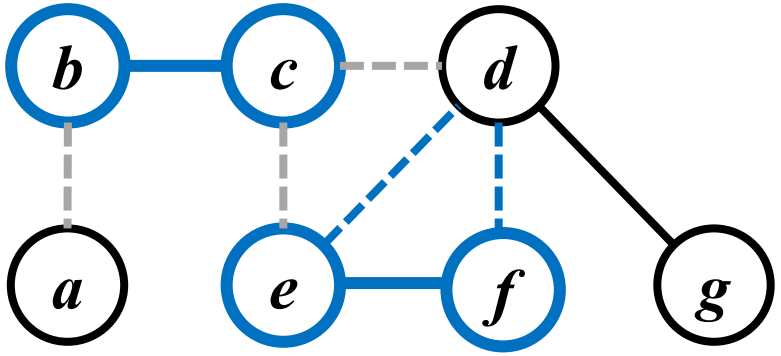
(1)

$C = \{b, c\}$



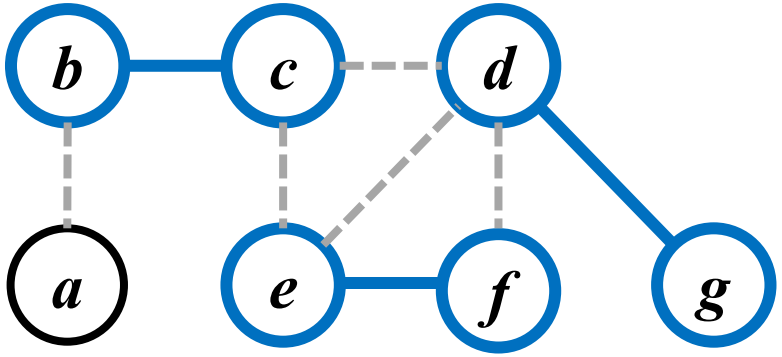
(2)

$C = \{b, c, e, f\}$



(3)

$C = \{b, c, e, f, d, g\}$



(4)

常数近似比算法

Approx-Vertex-Cover(G)

Input: A graph $G = (V, E)$
Output: A set of vertices C
$C \leftarrow \emptyset$
$E' \leftarrow E$
while $E' \neq \emptyset$
    let (u, v) be an arbitrary edge of E'
    $C \leftarrow C \cup \{u, v\}$
    remove from E' every edge incident on either u or v
return C

- 时间复杂度：$O(|V| + |E|)$
- 2-近似算法

近似比分析

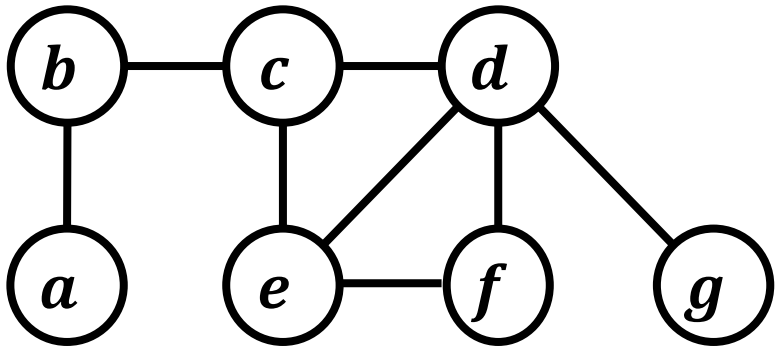
- 匹配（Matching）

给定无向图 $G=(V,E)$，G的一个匹配为G的一个边的子集 $M \subseteq E$，使得 V中每个顶点最多是M的一条边的端点。

近似比分析

- 匹配（Matching）

给定无向图 $G=(V,E)$，G的一个匹配为G的一个边的子集 $M \subseteq E$，使得 V中每个顶点最多是M的一条边的端点。



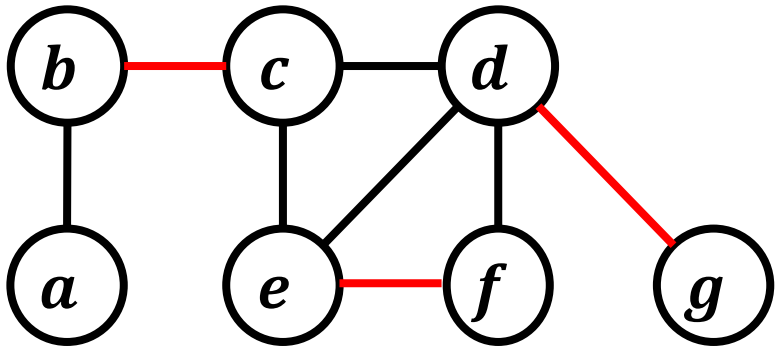
- 每条边构成一个匹配

如：$\{(a,b)\}, \{(b,c)\}$

近似比分析

- 匹配（Matching）

给定无向图 $G=(V,E)$，G的一个匹配为G的一个边的子集 $M \subseteq E$，使得 V中每个顶点最多是M的一条边的端点。



- 每条边构成一个匹配

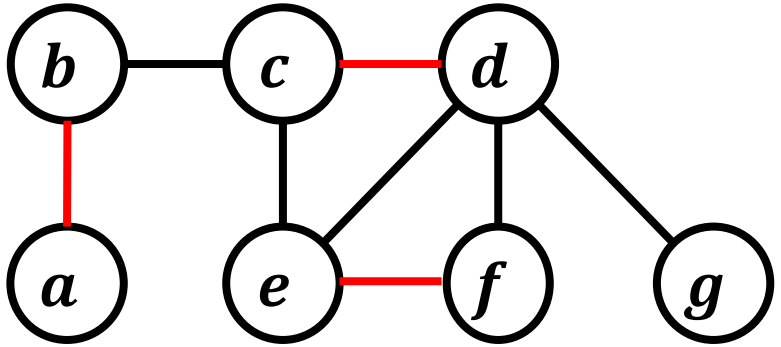
如：$\{(a,b)\}, \{(b,c)\}$

- $\{(b,c),(e,f),(d,g)\}$是一个匹配

近似比分析

- 匹配（Matching）

给定无向图 $G=(V,E)$，G的一个匹配为G的一个边的子集 $M \subseteq E$，使得 V中每个顶点最多是M的一条边的端点。



- 每条边构成一个匹配

如：$\{(a,b)\}, \{(b,c)\}$

- $\{(b,c),(e,f),(d,g)\}$是一个匹配
- $\{(a,b),(c,d),(e,f)\}$是一个匹配

近似比分析

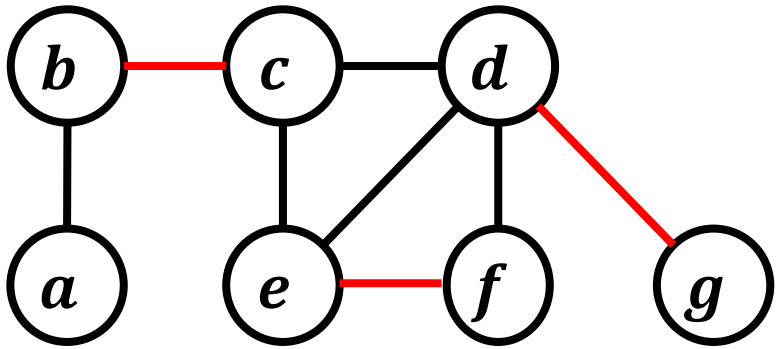
- 匹配（Matching）

  给定无向图 $G=(V,E)$，G的一个匹配为G的一个边的子集 $M \subseteq E$，使得 V中每个顶点最多是M的一条边的端点。

- 最大匹配（Maximum matching）问题

  输入：无向图 $G=(V,E)$

  输出：G 的最大匹配，即G 不是任何匹配的真子集。



- $\{(b,c),(e,f),(d,g)\}$ 是一个最大匹配

近似比分析

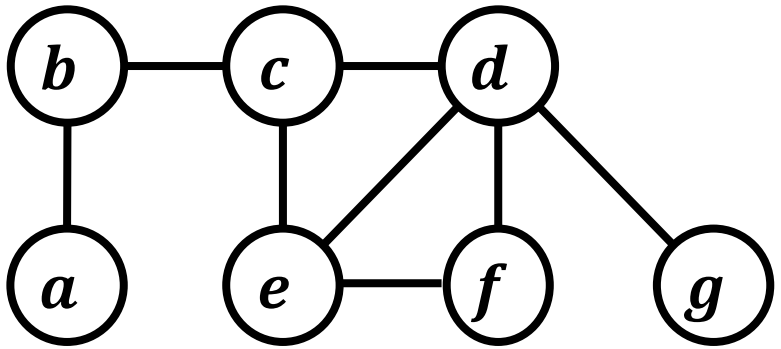
- 匹配（Matching）

  给定无向图 $G=(V,E)$，G的一个匹配为G的一个边的子集 $M \subseteq E$，使得 V中每个顶点最多是M的一条边的端点。

- 最大匹配（Maximum matching）问题

  输入：无向图 $G=(V,E)$

  输出：G 的最大匹配，即包含边的数目最大的匹配。



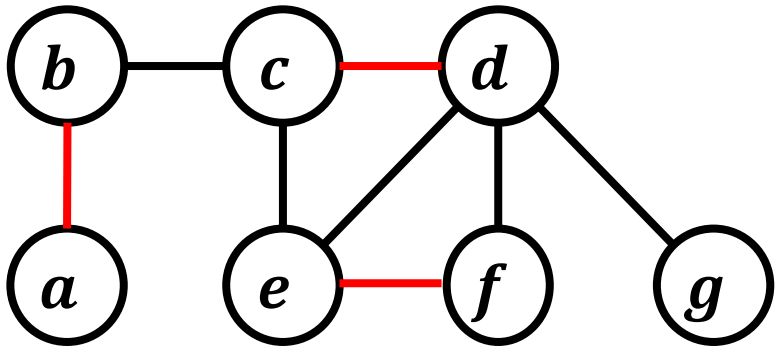
- $\{(b,c),(e,f),(d,g)\}$ 是一个最大匹配
- $\{(a,b),(c,d),(e,f)\}$ 是一个最大匹配

近似比分析

- 假设 M^*是图 G 的最大匹配， C^*是G的最小顶点覆盖， C是算法输出

近似比分析

- 假设 M^*是图 G 的最大匹配， C^*是G的最小顶点覆盖， C是算法输出

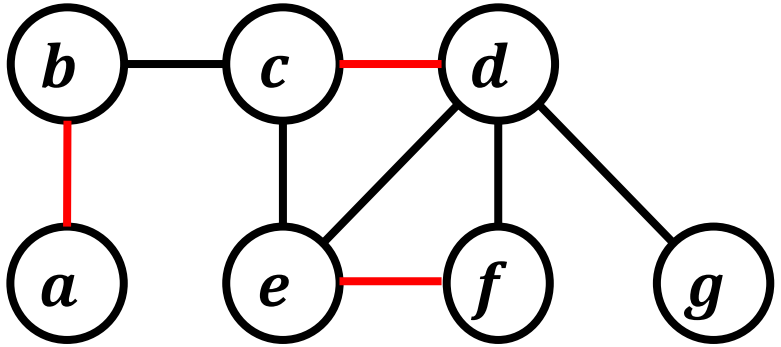


- $\{(a,b),(c,d),(e,f)\}$是一个最大匹配
- $\{a,b,c,d,e,f\}$是一个顶点覆盖

近似比分析

- 假设 M^*是图 G 的最大匹配， C^*是G的最小顶点覆盖， C是算法输出.

由于 M^*为最大匹配， M^* 中所有边的端点构成 G 的一个顶点覆盖。



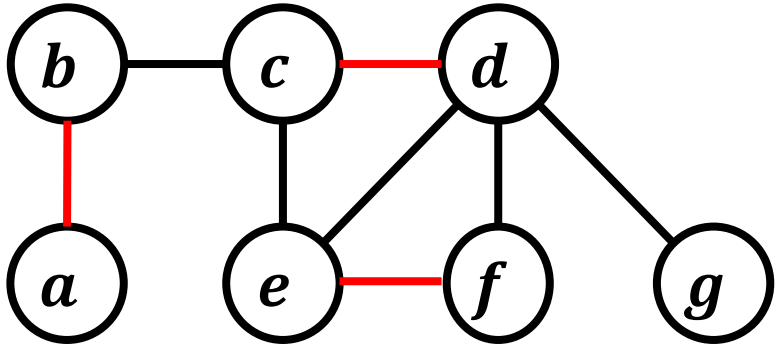
- $\{(a,b),(c,d),(e,f)\}$是一个最大匹配
- $\{a,b,c,d,e,f\}$是一个顶点覆盖

近似比分析

- 假设 M^*是图 G 的最大匹配， C^*是G的最小顶点覆盖， C是算法输出.

由于 M^*为最大匹配， M^* 中所有边的端点构成 G 的一个顶点覆盖。

（反证：若不构成G 的一个顶点覆盖，则存在一条边(u,v)使得u,v均不在顶点覆盖中，则可把$M^* \cup \{(u,v)\}$ 是一个匹配，与M^* 是最大匹配矛盾。）

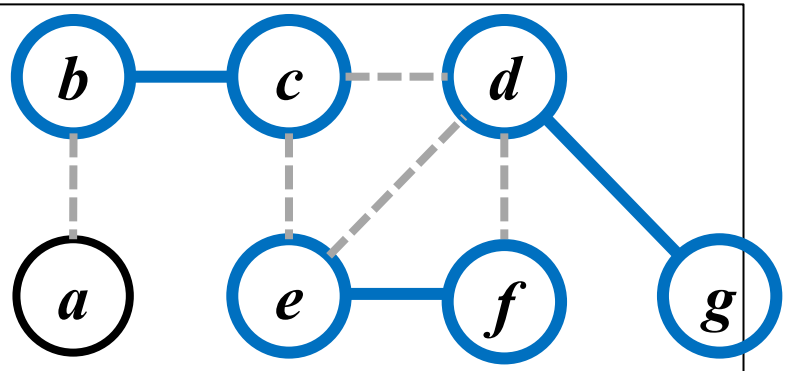


- $\{(a,b),(c,d),(e,f)\}$是一个最大匹配
- $\{a,b,c,d,e,f\}$是一个顶点覆盖

常数近似比算法

Approx-Vertex-Cover(G)

$C = \{b, c, e, f, d, g\}$



Input: A graph $G = (V, E)$
Output: A set of vertices C
$C \leftarrow \emptyset$
$E' \leftarrow E$
while $E' \neq \emptyset$
    let (u, v) be an arbitrary edge of E'
    $C \leftarrow C \cup \{u, v\}$
    remove from E' every edge incident on either u or v
return C

记算法第4行取出的边构成集合A，A是一个匹配

- 时间复杂度：$O(|V| + |E|)$
- 2-近似算法

近似比分析

- 假设 M^*是图 G 的最大匹配， C^*是G的最小顶点覆盖， C是算法输出.

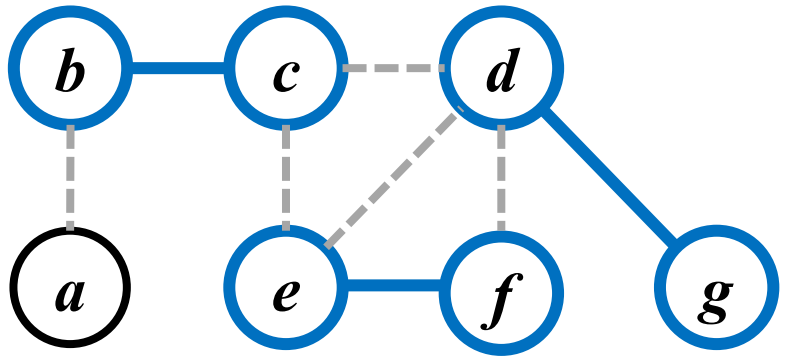
由于 M^*为最大匹配， M^* 中所有边的端点构成 G 的一个顶点覆盖。

（反证：若不构成G 的一个顶点覆盖，则存在一条边(u,v)使得u,v均不在顶点覆盖中，则可把$M^* \cup \{(u,v)\}$ 是一个匹配，与M^* 是最大匹配矛盾。）

记算法第4行取出的边构成集合 A，显然 A 是一个匹配，且$|C| = 2|A|$.

因此， $|C| \leq 2|M^*|$。

$$C = \{b, c, e, f, d, g\}$$



近似比分析

- 假设 M^*是图 G 的最大匹配， C^*是G的最小顶点覆盖， C是算法输出.

由于 M^*为最大匹配， M^* 中所有边的端点构成 G 的一个顶点覆盖。

（反证：若不构成G 的一个顶点覆盖，则存在一条边(u,v)使得u,v均不在顶点覆盖中，则可把$M^* \cup \{(u,v)\}$ 是一个匹配，与M^* 是最大匹配矛盾。）

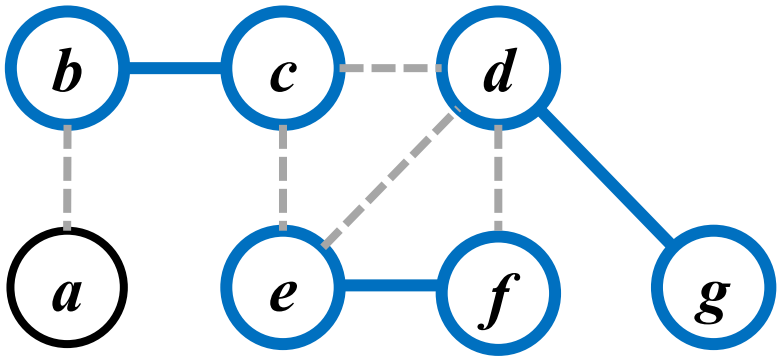
记算法第4行取出的边构成集合 A，显然 A 是一个匹配，且$|C| = 2|A|$.

因此， $|C| \leq 2|M^*|$。

由于 A 中任意两条边没有公共点，即A 中每条边被不同的顶点覆盖，

因此， $|C^*| \geq |A| = |M^*|$

$\boldsymbol{C = \{b, c, e, f, d, g\}}$



近似比分析

- 假设 M^*是图 G 的最大匹配， C^*是G的最小顶点覆盖， C是算法输出.

由于 M^*为最大匹配， M^* 中所有边的端点构成 G 的一个顶点覆盖。

（反证：若不构成G 的一个顶点覆盖，则存在一条边(u,v)使得u,v均不在顶点覆盖中，则可把$M^* \cup \{(u,v)\}$ 是一个匹配，与M^* 是最大匹配矛盾。）

记算法第4行取出的边构成集合 A，显然 A 是一个匹配，且$|C| = 2|A|$.

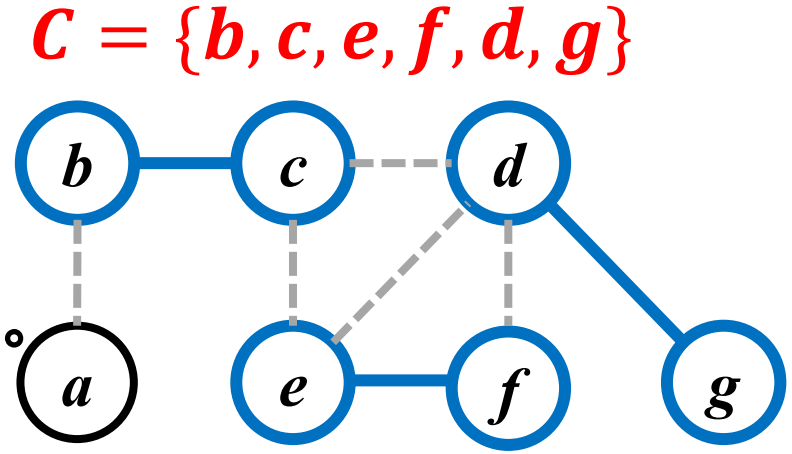
因此， $|C| \leq 2|M^*|$。

由于 A 中任意两条边没有公共点，即A 中每条边被不同的顶点覆盖，

因此， $|C^*| \geq |A| = |M^*|$

综上，得 $\frac{|C|}{|C^*|} \leq \frac{2|M^*|}{|M^*|} = 2$。

因此，算法Approx-Vertex-Cover(G) 为2-近似算法。



提纲

- 背景
- 基本概念
- 顶点覆盖问题
- **旅行商问题**
- 集合覆盖问题

哈密顿回路问题

- 哈密顿回路问题
  - 输入：无向图 G
  - 问题： G 是否存在一条哈密顿回路，即通过图中每个节点一次且仅一次的回路

NP完全问题

- 欧拉回路问题
  - 输入：无向图
  - 问题： G 是否存在一条欧拉回路，即通过图中每条边一次且仅一次的回路

- 多项式时间可解
- G有欧拉回路当且仅当G中每个点的度为2

- 定理：如果 P≠NP，则对任意常数 $r \geq 1$，旅行商问题不存在具有近似比 r 的多项式时间近似算法。

反证：假设 A 是旅行商问题的近似比为常数 r 的多项式时间近似算法。

基于算法A 构造哈密顿回路的多项式时间算法 A_H：给定任意哈密顿回实例，即无向图 G_{HC}，且顶点数为 n

1. 将 G_{HC} 映射为 G_{TSP}；
2. 用算法 A 求 G_{TSP} 的旅行商问题近似解，即旅程 ρ；
3. 如果 $|\rho| = n$ ，则 G_{HC} 存在哈密顿回路；
4. 否则， G_{HC} 不存在Hamilton 回路。

下面证明算法A_H的正确性，即证明：

G_{HC}有哈密顿回路 当且仅当 $|\rho| = n$ 。

下面证明算法A_H的正确性，即证明：

G_{HC}有哈密顿回路 当且仅当 $|\rho| = n$ 。

（$\Leftarrow$）若 $|\rho| = n$ ，则 ρ 中每条边都在 G_{HC} 中，

因此，ρ 为 G_{HC} 的哈密顿回路。

（$\Rightarrow$）若 G_{HC} 有哈密顿回路，则也为图G_{TSP}上旅行商问题的最优解，其值为n。假设近似解 ρ 的长度 $|\rho| > n$ ，

则 ρ 中至少包含一条权重为 $nr + 1$ 的边，得

$$|\rho| \geq n - 1 + nr + 1 = n(r + 1) \ ,$$

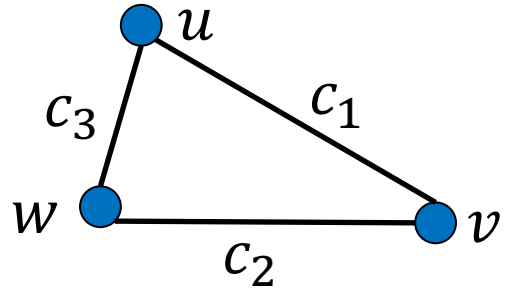
与 A_H 是 r -近似算法矛盾，则假设不成立，即 $|\rho| = n$ 。

综上，算法 A_H 是正确的，即 A_H 是哈密顿回路问题的多项式时间算法，与其是NP完全问题矛盾。

故假设不成立，即旅行商问题不存在 r -近似算法。

旅行商问题的一个特例

- 满足三角不等式的无向完全图 G
  - 对G中任意三条边： $(u,v),(v,w),(w,u)$ ，边上代价分别为c_1 ， c_2 ， c_3 均有： $c_1 + c_2 \geq c_3$



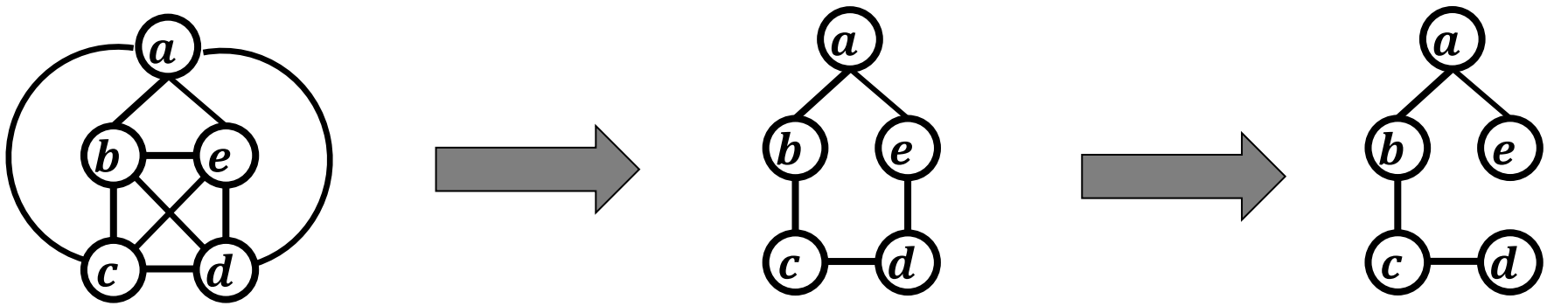
- 度量旅行商问题（ Metric TSP ）
  - 输入：满足三角不等式的无向完全图$G = (\mathrm{V}, E)$，其中每条边 (u,v)都有一个非负整数代价 $c(u,v)$
  - 输出：具有最小代价的哈密顿回路A，其中

$$c(A) = \sum_{(u,v)\in a} c(u,v)$$

- 度量旅行商问题有常数近似比近似算法
  - 2-近似算法
  - 1.5 近似算法

基于最小生成树的近似算法

- 观察：图 G的一个哈密顿回路去掉一条边，得到图 G的一棵生成树



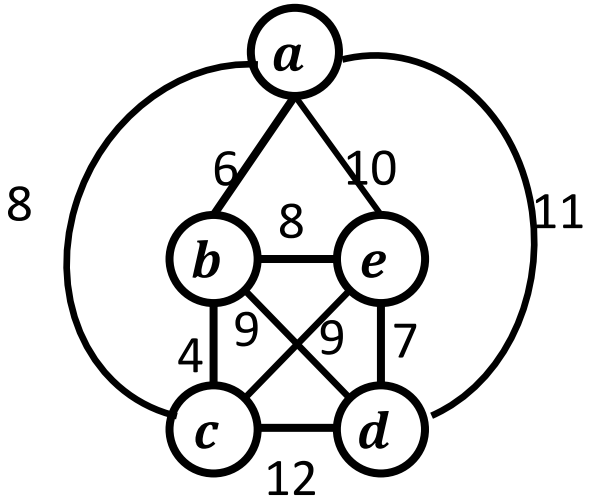
G

G的一条
哈密顿回路

G的一个
生成树

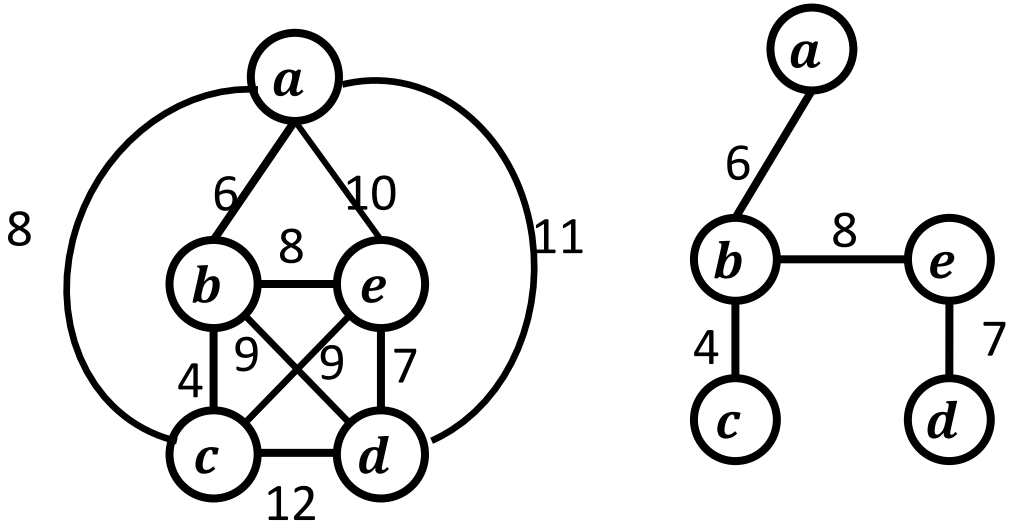
基于最小生成树的近似算法

- 观察：图 G的一个哈密顿回路去掉一条边，得到图 G的一棵生成树
- 算法Metric-TSP步骤：



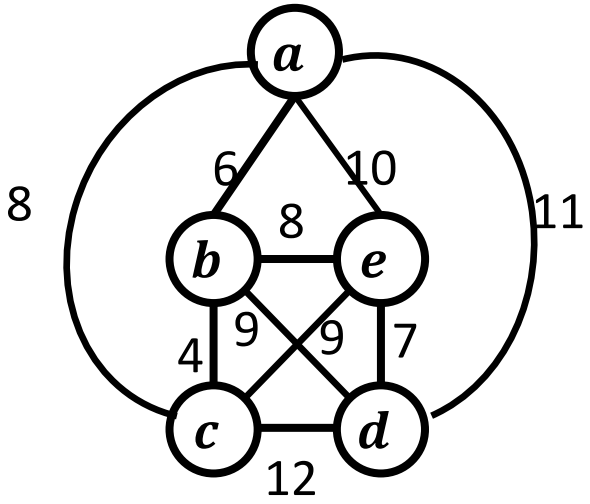
基于最小生成树的近似算法

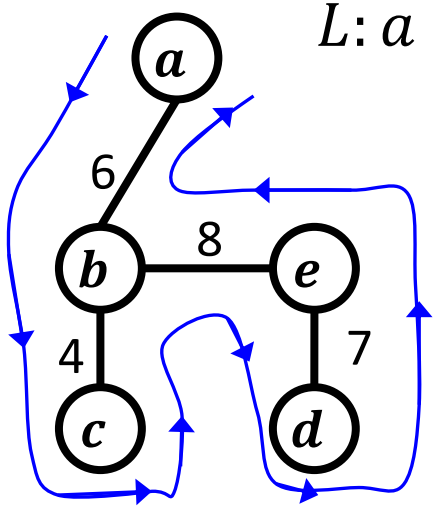
- 观察：图 G的一个哈密顿回路去掉一条边，得到图 G的一棵生成树
- 算法Metric-TSP步骤：
  1. 构造图G的最小生成树 T^*；



基于最小生成树的近似算法

- 观察：图 G的一个哈密顿回路去掉一条边，得到图 G的一棵生成树
- 算法Metric-TSP步骤：
  1. 构造图G的最小生成树 T^*；
  2. 从T^*的任意一个顶点开始，绕着这棵最小生成树散步一周，并记录下经过的顶点，构成顶点列表 L；

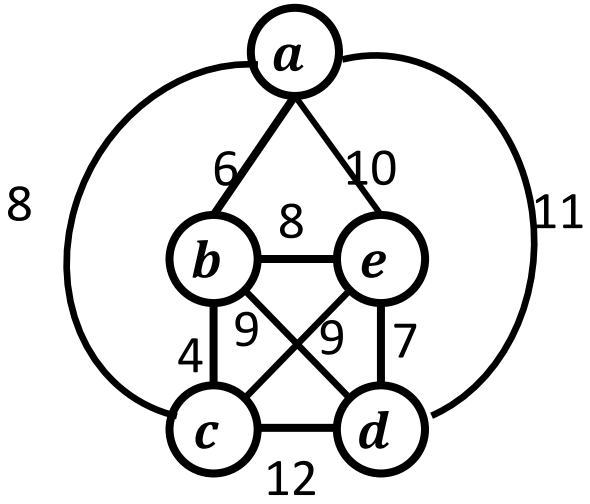


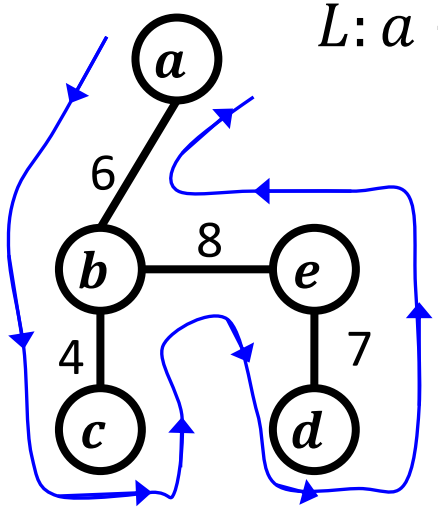


$L: a \to b \to c \to b \to e \to d \to e \to b \to a$

基于最小生成树的近似算法

- 观察：图 G的一个哈密顿回路去掉一条边，得到图 G的一棵生成树
- 算法Metric-TSP步骤：
  1. 构造图G的最小生成树 T^*；
  2. 从T^*的任意一个顶点开始，绕着这棵最小生成树散步一周，并记录下经过的顶点，构成顶点列表 L；
  3. 扫描顶点列表 L ，删除所有重复出现的顶点，但留下L 尾部的起始顶点；





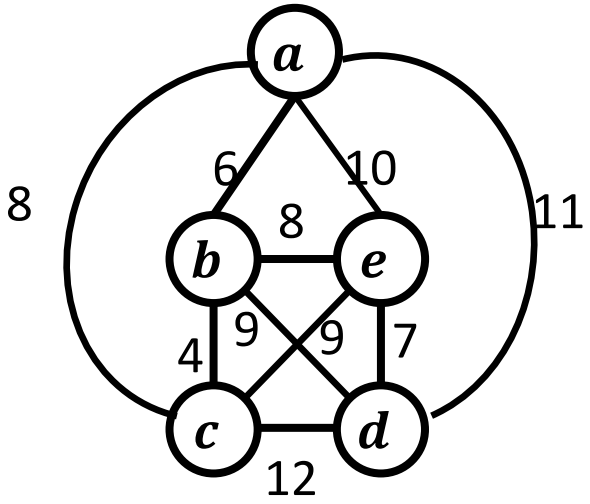
$L: a \to b \to c \to b \to e \to d \to e \to b \to a$

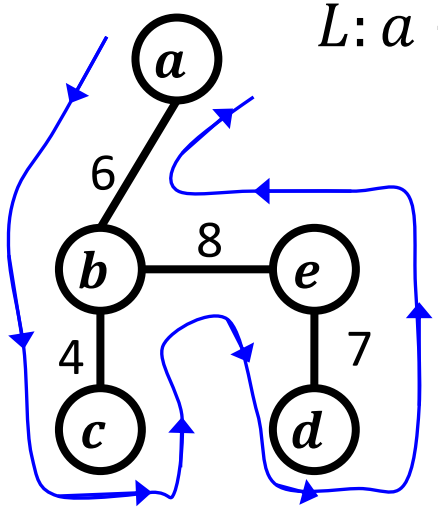
$L: a \to b \to c \to e \to d \to a$

走捷径

基于最小生成树的近似算法

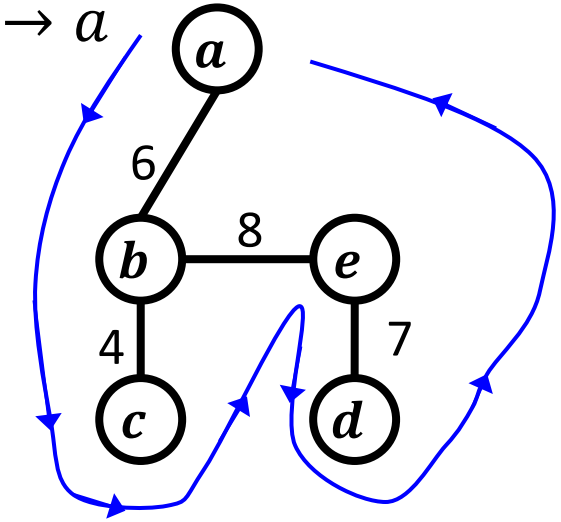
- 观察：图 G的一个哈密顿回路去掉一条边，得到图 G的一棵生成树
- 算法Metric-TSP步骤：
  1. 构造图G的最小生成树 T^*；
  2. 从T^*的任意一个顶点开始，绕着这棵最小生成树散步一周，并记录下经过的顶点，构成顶点列表 L；
  3. 扫描顶点列表 L ，删除所有重复出现的顶点，但留下L 尾部的起始顶点；
  4. 列表中余下的顶点就构成了一条哈密顿回路，为算法输出。





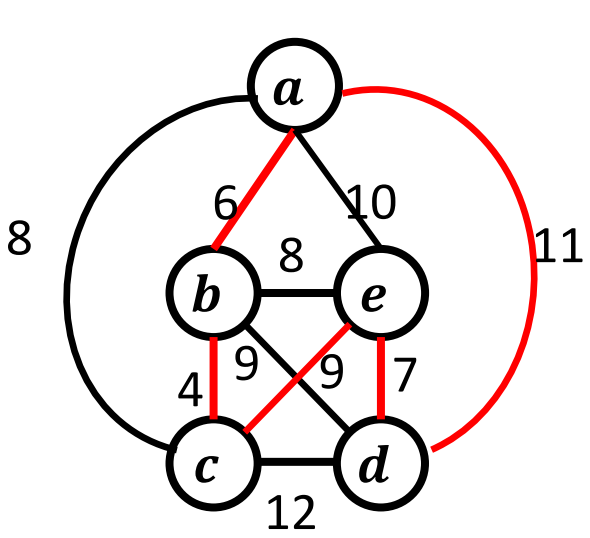
$L: a \to b \to c \to b \to e \to d \to e \to b \to a$

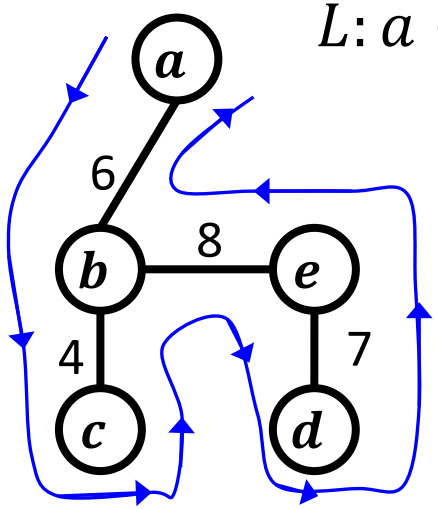
$L: a \to b \to c \to e \to d \to a$



基于最小生成树的近似算法

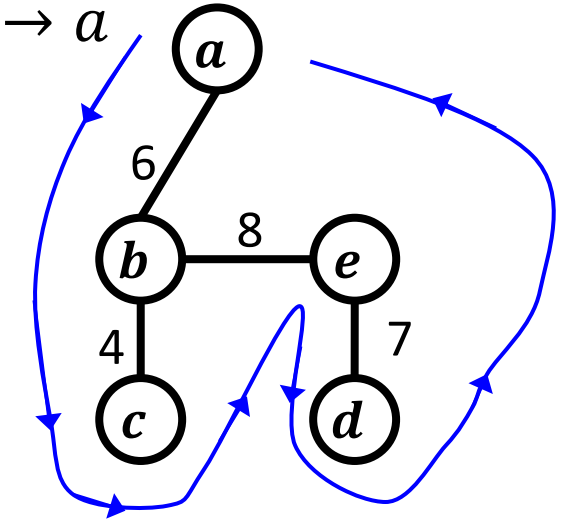
- 观察：图 G的一个哈密顿回路去掉一条边，得到图 G的一棵生成树
- 算法Metric-TSP步骤：
  1. 构造图G的最小生成树 T^*；
  2. 从T^*的任意一个顶点开始，绕着这棵最小生成树散步一周，并记录下经过的顶点，构成顶点列表 L；
  3. 扫描顶点列表 L ，删除所有重复出现的顶点，但留下L 尾部的起始顶点；
  4. 列表中余下的顶点就构成了一条哈密顿回路，为算法输出。





$L: a \to b \to c \to b \to e \to d \to e \to b \to a$

$L: a \to b \to c \to e \to d \to a$



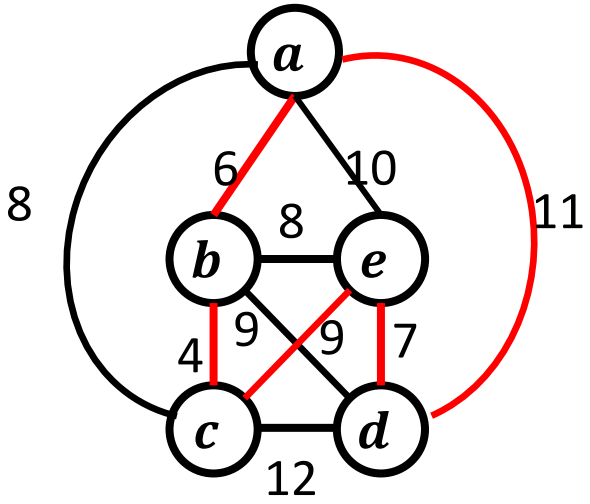
近似比分析

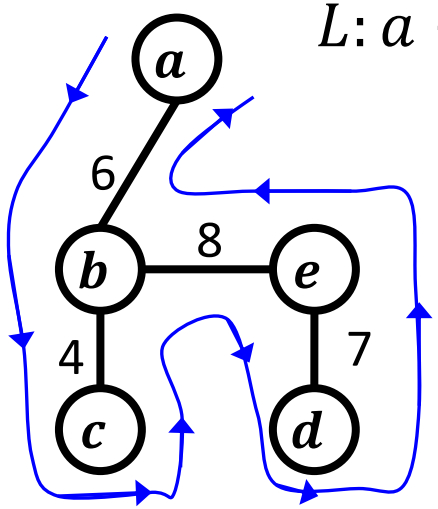
- 定理：算法Metric-TSP是 2-近似算法。

近似比分析

- 定理：算法Metric-TSP是 2-近似算法。

证明：给定无向加权图 G，最优解 为 C^*，C^* 去掉一条边后为G的一个生成树 T， G的最小生成树为 T^*，算法输出 C。

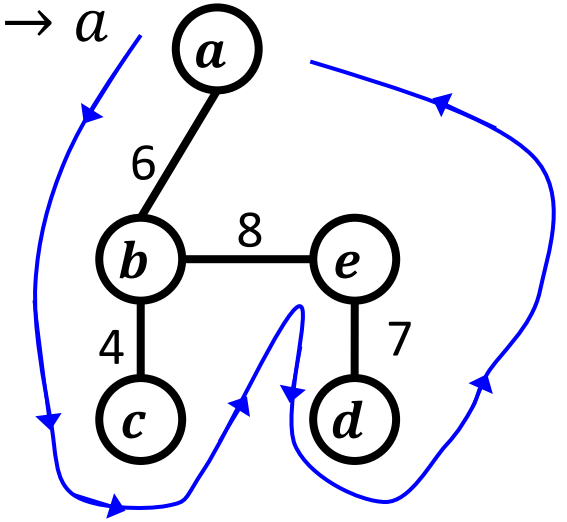




$L: a \to b \to c \to b \to e \to d \to e \to b \to a$

$L: a \to b \to c \to e \to d \to a$

C



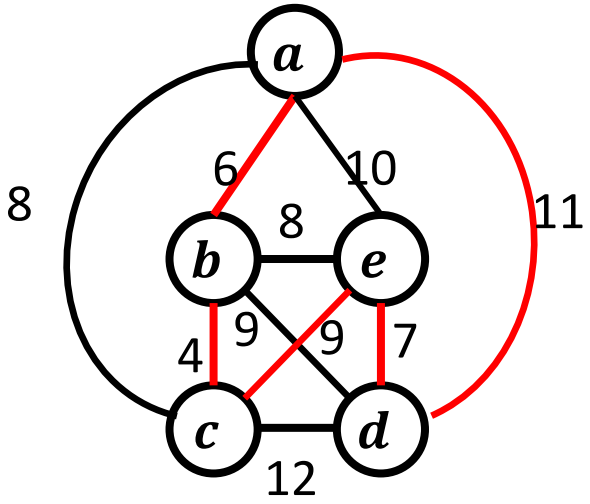
近似比分析

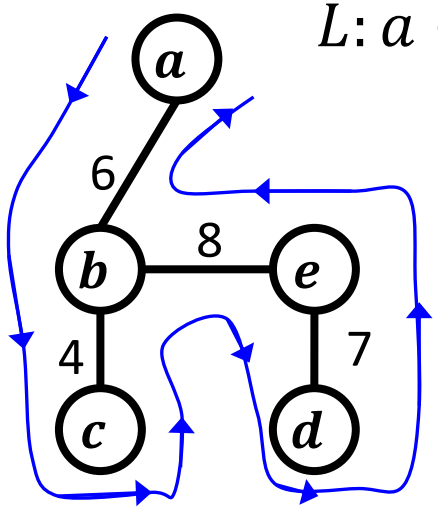
- 定理：算法Metric-TSP是 2-近似算法。

证明：给定无向加权图 G，最优解 为 C^*，C^* 去掉一条边后为G的一个生成树 T， G的最小生成树为 T^*，算法输出 C。有以下结论：

1) $c(C^*) > c(T) \geq c(T^*)$

2) $2\ c(T^*) > c(C)$

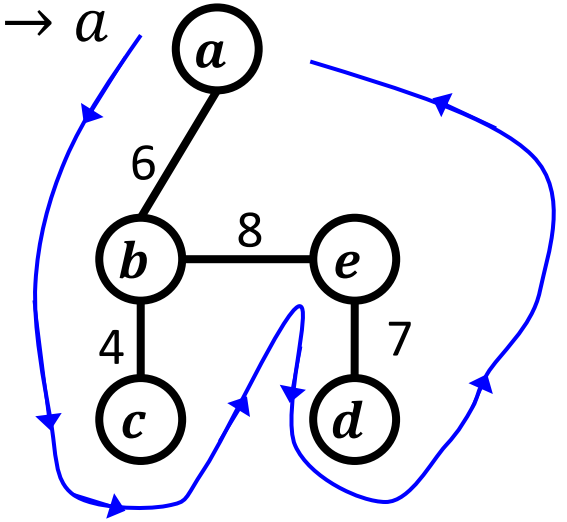




$L: a \to b \to c \to b \to e \to d \to e \to b \to a$

$L: a \to b \to c \to e \to d \to a$

C



近似比分析

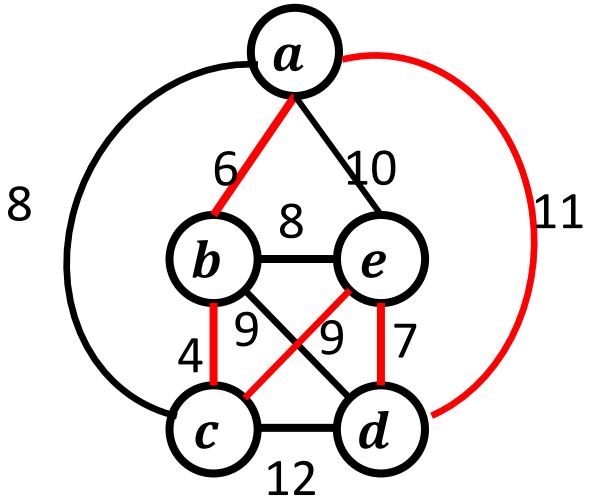
- 定理：算法Metric-TSP是 2-近似算法。

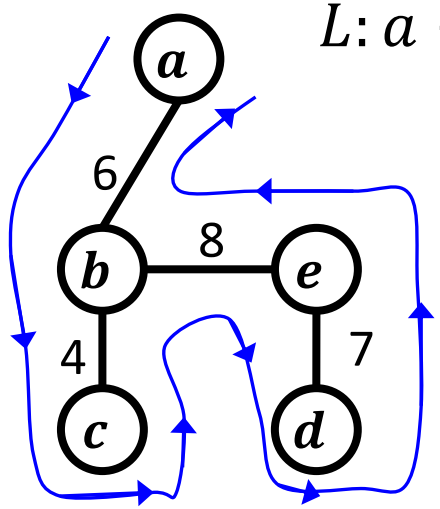
证明：给定无向加权图 G，最优解 为 C^*，C^* 去掉一条边后为G的一个生成树 T， G的最小生成树为 T^*，算法输出 C。有以下结论：

1) $c(C^*) > c(T) \geq c(T^*)$

2) $2\ c(T^*) > c(C)$

得： $c(C) < 2c(C^*)$。因此， Metric-TSP是 2-近似算法。

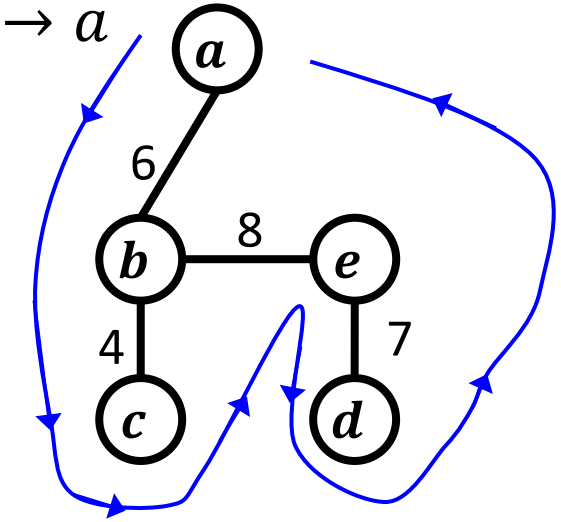




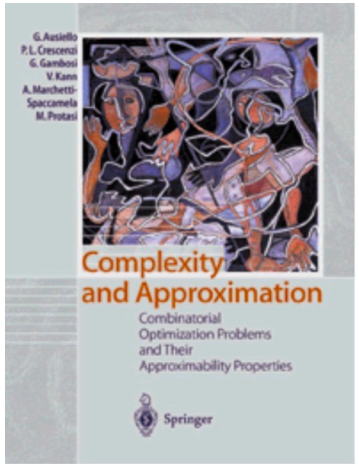
$L: a \to b \to c \to b \to e \to d \to e \to b \to a$

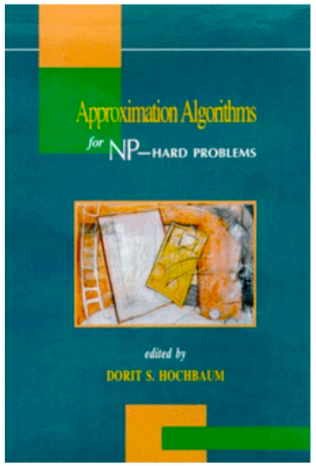
$L: a \to b \to c \to e \to d \to a$

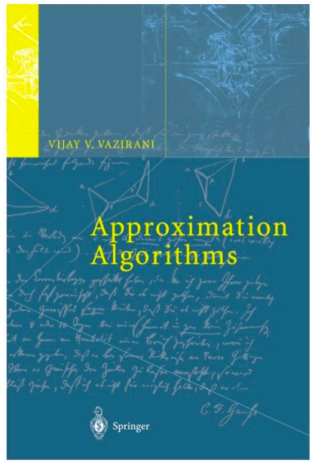
C

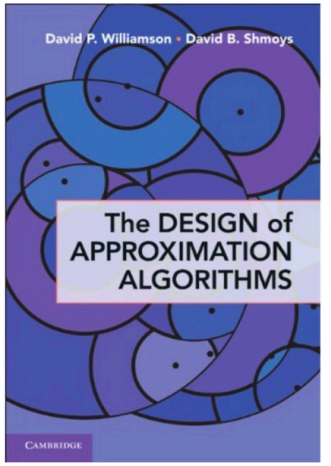


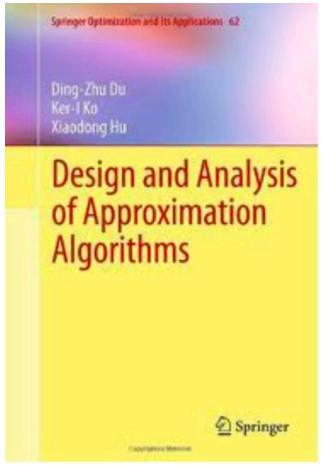
相关书籍











[1] Ausiello, Crescenzi, Gambosi, etc. Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. 1999

[2] Hochbaum (Editor) . Approximation Algorithms for NP-Hard Problems. 1997

[3] Vijay V. Vazirani. Approximation Algorithms. 2001

[4] D.P. Williamson & D.B. Shmoys. The Design of Approximation Algorithms. 2010

[5] D.Z Du, K-I. Ko & X.D. Hu. Design and Analysis of Approximation Algorithms. 2012

考试须知

北京航空航天大学
计算机学院

北航《算法设计与分析》

考试安排

- 考试时间
  - 正式考试时间：2022年12月23日13：20-15：20
  - 模拟时间：2022年12月17日19：00-20：00

- 考试方式
  - 线上开卷考试
  - 只允许使用课件（打印/电子版），不允许上网查询，不可使用键盘
  - 考试过程全程进行视频录制，请遵守考试纪律规范

- 平台选用
  - 监考平台：腾讯会议
  - 试卷发放平台：北航云盘
  - 试卷回收平台：北航云盘

考题类型

- 填空题 ×4
- 判断题 ×4
- 算法运行实例题 ×2
- 算法设计题 ×4