运筹学通论I

胡晓东

应用数学研究所

中国科学院数学与系统科学研究院

http://www.amt.ac.cn/member/huxiaodong/index.html

5. 组合优化-算法设计技巧

精确算法

分而治之 (搜索、排大小序、旅行商)
动态规划 (最短路、三角剖分、背包)
分支定界 (整数线性规划、旅行商、工件排序)

近似算法或启发式算法

贪婪策略 (最小生成树、指派、最大可满足、背包、
顶点覆盖、独立集、旅行商)
局部搜索(最大匹配、旅行商、最大割)
序贯法 (工件排序、装箱、顶点着色)
整数规划法 (顶点覆盖、最大可满足、最大割)
随机方法 (最小割、最大可满足、顶点覆盖)
在线算法 (页面调度、k-服务器、工件排序、装箱)
不可近似 (最大团、背包、旅行商、装箱、连通控制集等)

5.4 贪婪策略（续一）

Someone reminded me that I once said,

“Greed is good.”

Now it seems that it’s legal.

—Gordon Gekko

(in *Wall Street: Money Never Sleeps*)

I think greed is healthy.

You can be greedy

and still feel good about yourself.

—Ivan Boesky

5.4 贪婪策略（续一）

当要求解一个优化问题时（找到它的一个最优解或者一个比较好的解），我们通常最先想到的方法就是使用**贪婪策略**； 基于贪婪策略的算法，在求解的过程中总是采用当前看起来最优的选择或者步骤：追求**局部最优**的效果，以期望这样最后可以达到全局或者**整体最优**。

在求解如下问题时人们更喜欢用贪婪策略：问题的实例由由一组元件构成，问题的目标是要确定这些元件的一个子集使得费用或者效益函数达到最小值或者最大值。 如果要采用贪婪策略，那么问题的可行解需要满足某种单调性，亦即，若一组元件 S 构成了一个可行解，则它的任意一个子集 $S' \subset S$ 也构成问题的一个可行解。

5.4 贪婪策略（续二）

基于贪婪策略的算法的一般步骤如下：首先将所给的元件依照某种指标排个顺序，置可行解为空集；然后，逐步地将所考虑的元件放入可行解，直到构造出一个可行解。在运行算法这个过程中，

- 每一次仅考虑一个元件，自始至终保持解的可行结构。
- 当每次考虑一个元件时，如果放入可行解可以保持解的可行结构，那么就将其放入；否则就不再考虑这个元件（再也不可能放入可行解中）。
- 一旦将一个元件放入可行解，那么就再也不会将其从可行解中移除了。

5.4 贪婪策略（续三）

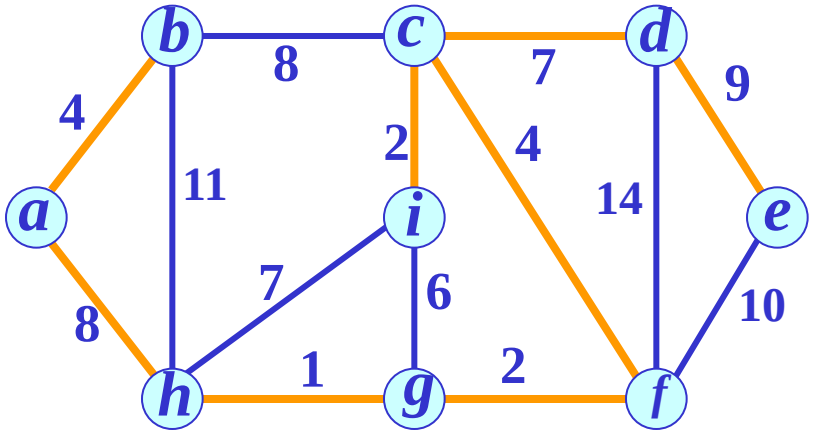
基于贪婪策略的算法的运行时间包括，将 ***n*** 个元件排序所需要的时间***O*(*n* log *n*)**，考虑 ***n*** 个元件时所进行的保持可行结构的测试（复杂程度依问题而异）。

此外，基于贪婪策略的算法所得到的解的好坏与考虑所有元素的最初顺序有关；显然，对于所考虑问题的每一个实例，总存在一个（最优）顺序，它可使得基于贪婪策略的算法找到最优解。然而，对于一个**NP-**难问题来讲，我们不能期望可以在多项式时间内求得这样一个（最优）顺序。

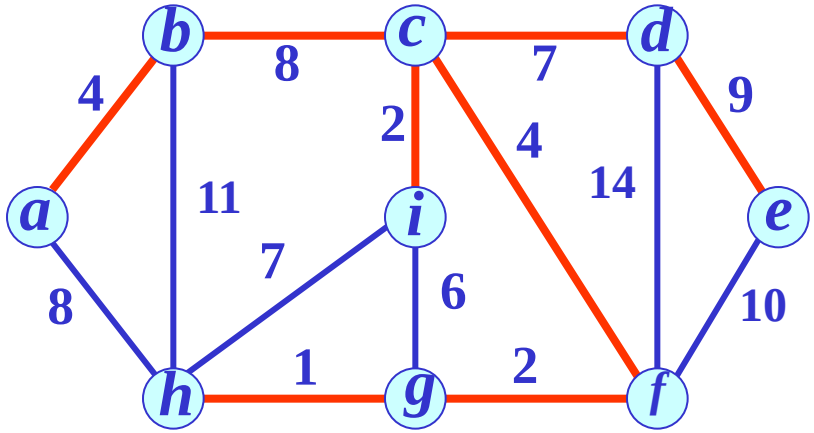
5.4 最小生成树

首先讨论**最小生成树问题**。任意给定一个边赋权图，找出该图的一棵生成树（图中的每一个节点都在树上），且树上所有边的权和最小。

问　题: 图上最小生成树问题
实　例: 图 $G(V, E)$，每一条边 $(u, v) \in E$ 有一个权值 $w(u, v)$
可行解: 图 G 的生成树 T
目　标: 最小化树 T 的权值, $c(T) = \sum_{(u, v) \in T} w(u, v)$



权值为37的最小生成树



另一棵最小生成树

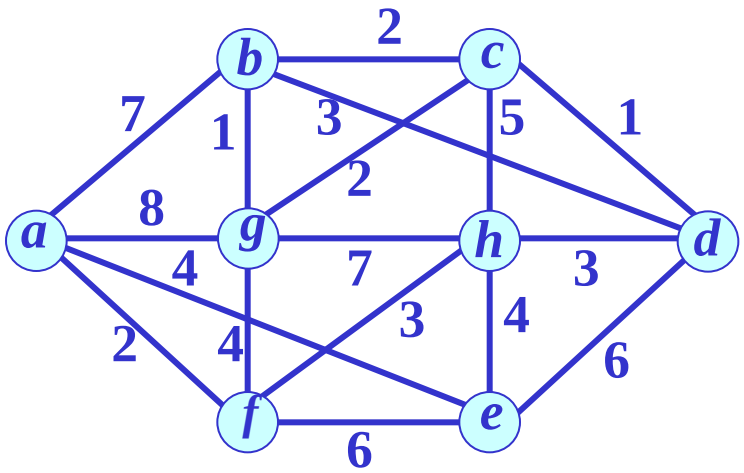
5.4 最小生成树（续一）

Kruskal 于1956年提出了以下算法：

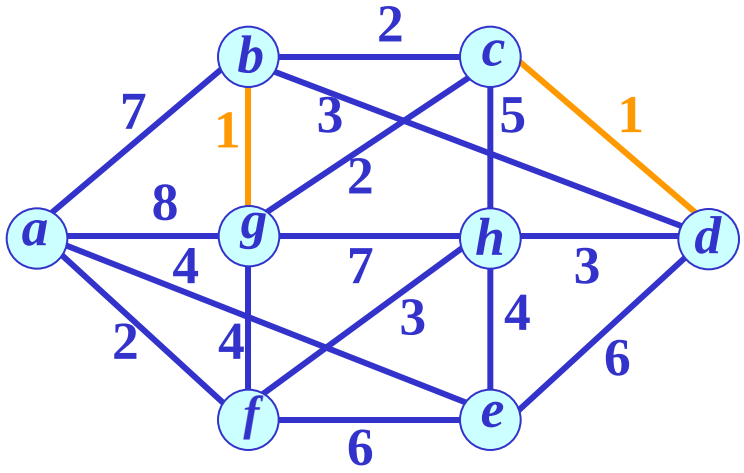
- 开始, 取集合 ***R*** 为空集，并造|***V***| 棵树, 每一棵树还有一个节点。另将边集 ***E*** 中所有的边依照它们权值由小到大排序。
- 然后，依照顺序考虑每一条边 (***u***, ***v***)，检查边的端点 ***u*** 和 ***v*** 是否属于同一棵树。如果“是”，那么这条边 (***u***, ***v***)不能添加进现有的森林（否则就会产生一个圈）；
  因此，可将此边去除不再考虑。
- 如果“不是”，即两个端点属于不同的树，那么将边(***u***, ***v***) 添加到集合 ***R*** 中。（两棵树合成一棵树了）

Kruskal 提出的上述算法可以视为一个贪婪算法，因为它每一次都是在可添加的边中选择一条权值最小的边。

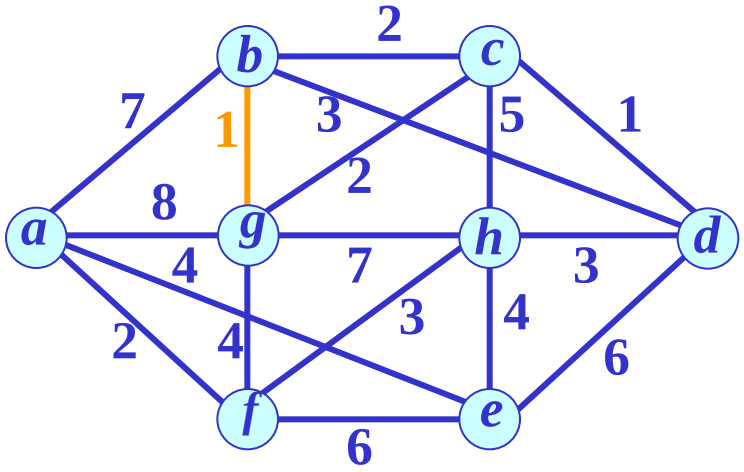
5.4 最小生成树（续二）



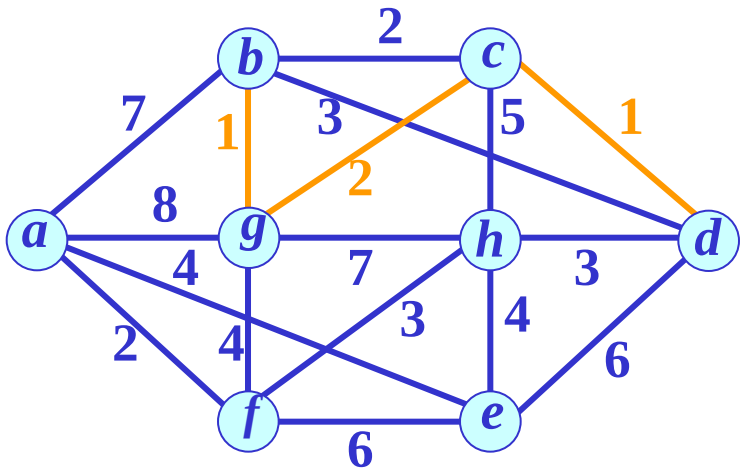
(a)



(c)

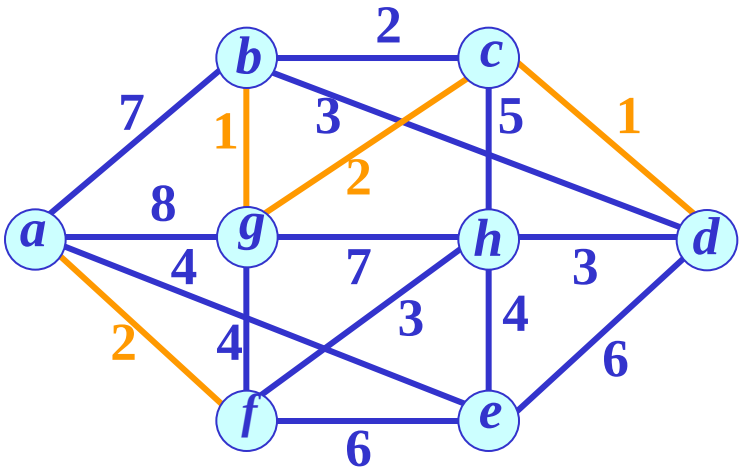


(b)

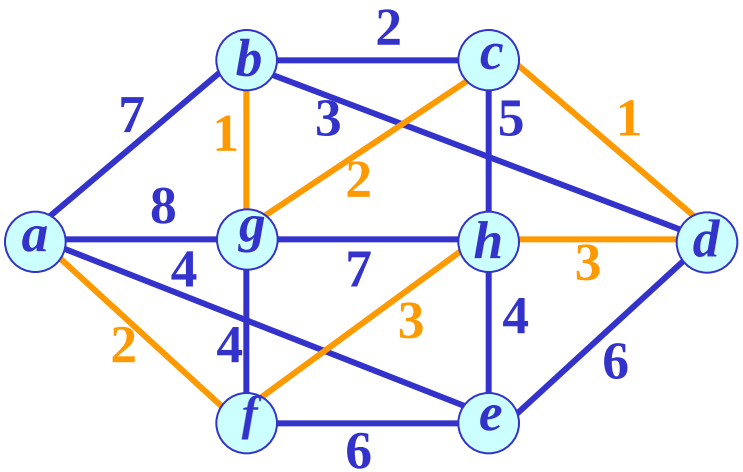


(d)

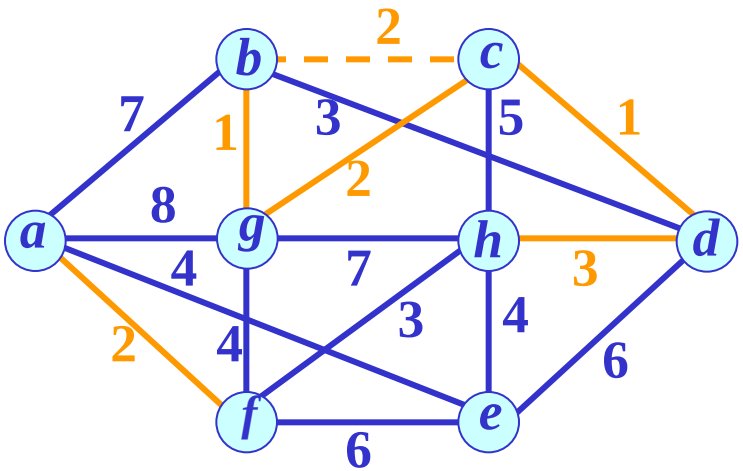
5.4 最小生成树（续三）



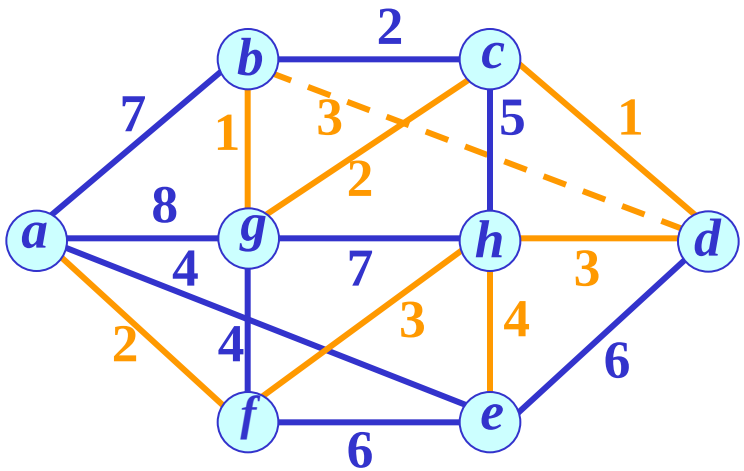
(e)



(g)



(f)

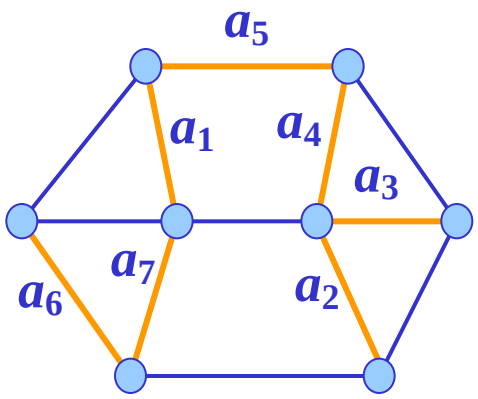


(h)

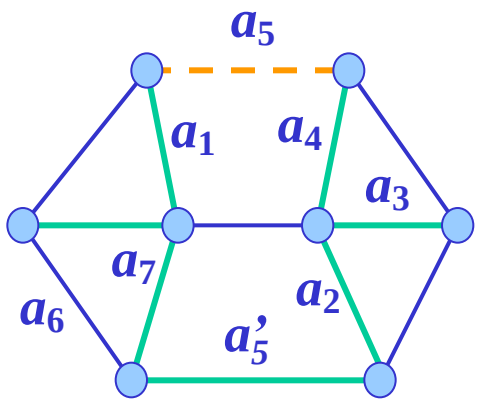
5.4 最小生成树（续四）

定理 1 任意给定最小生成树的一个实例，**Kruskal** 算法都可以在$O(|E|\log|E|+|V|^2)$时间内找到一棵最小生成树。

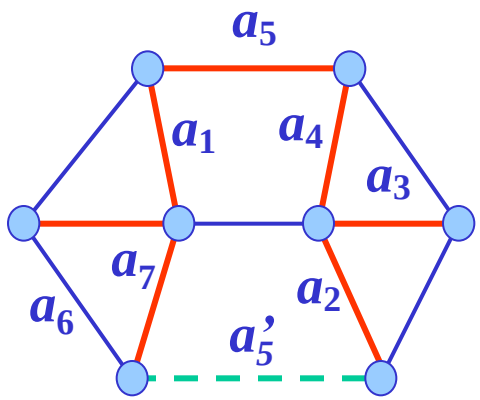
证明：用反证法。假设**Kruskal** 算法求得的解为 T_K，而它与最小生成树 T 不同。我们进一步假设，$a_1, a_2, \ldots, a_i \in T_K \cap T$，而$a_{i+1} \in T_K$ 但是$a_{i+1} \notin T$。若将a_{i+1} 添加进T，则会产生一个圈C，其中有一条边 $a'_{i+1} \notin T_K$。注意根据贪婪原则，a'_{i+1}的权不小于 a_{i+1} 的权值。因此替换掉它后产生的生成树 T' 仍是最小的。



Kruskal 算法求得解 T_K



最小生成树 T



另一棵最小生成树 T'

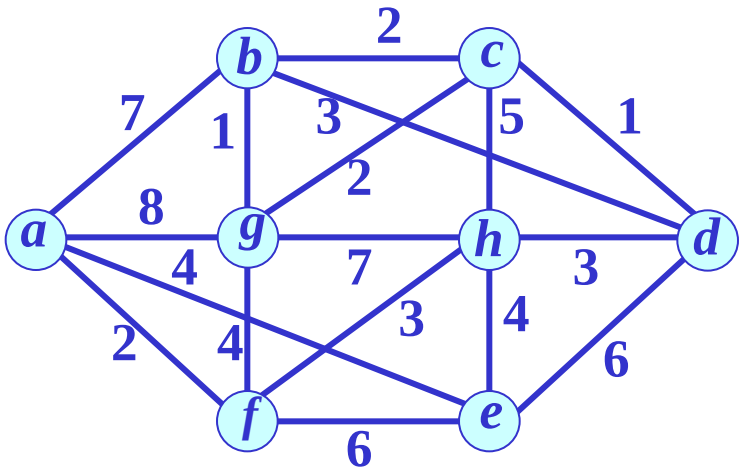
5.4 最小生成树（续五）

Prim于1957年提出了另外一个算法:

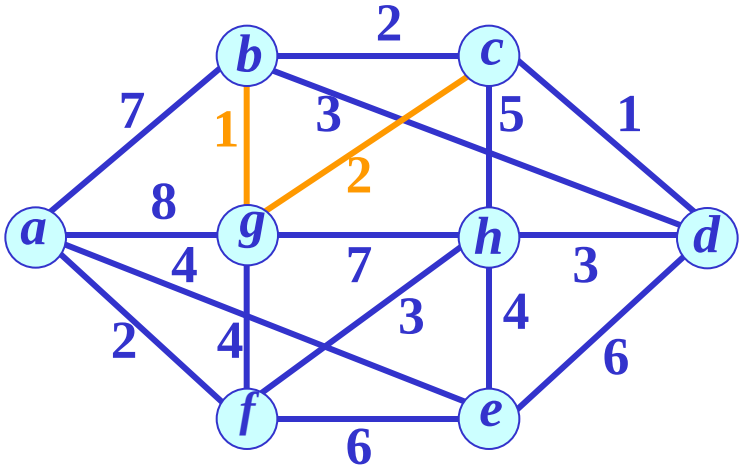
- 开始, 在节点集合中人选一个节点$r \in V$，令 R 是包含一个节点 r 的一棵树。
- 然后, 逐步添加点进 R，将它培养成一棵生成树：确定节点 $u \in V \setminus R$ 它与R 中的一个节点有边(u, v) 相连，而且这条边的权值是最小的。将 u 放到集合 R 中。
- 重复上面的步骤直到 $R = V$, 此时就构造出一棵生成树。

很显然，**Prim算法**同样可以看成是一个贪婪算法：在树的每一次生长过程中， 增加权值最小的边添加到树中了。

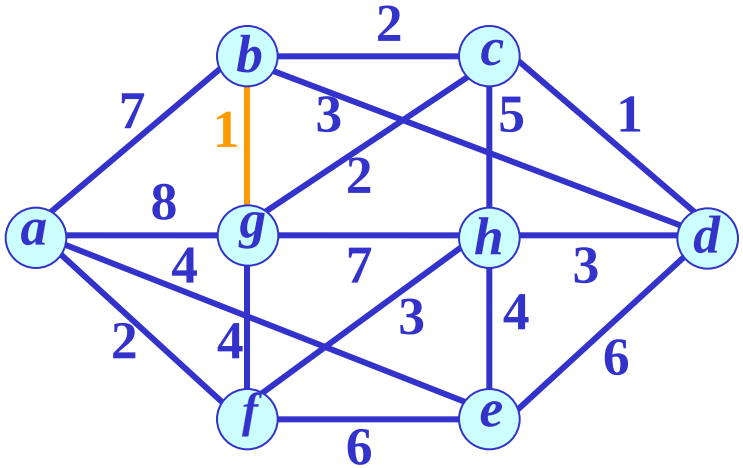
5.4 最小生成树（续六）



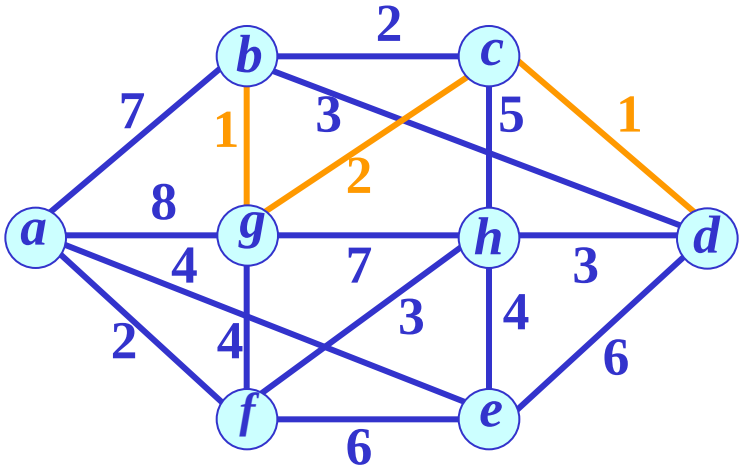
(a)



(c)

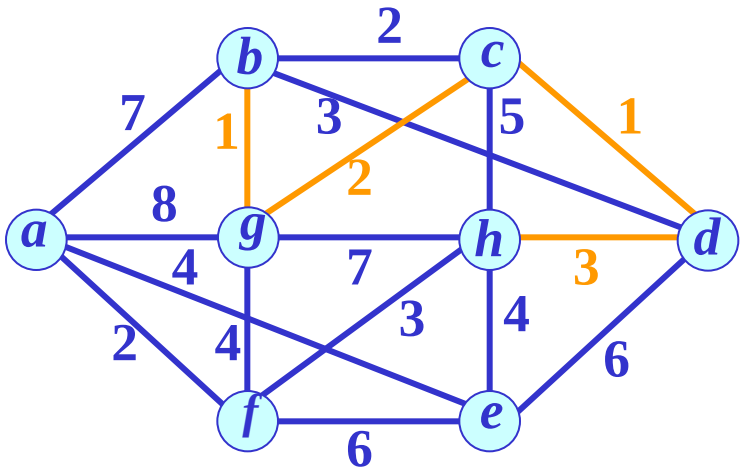


(b)

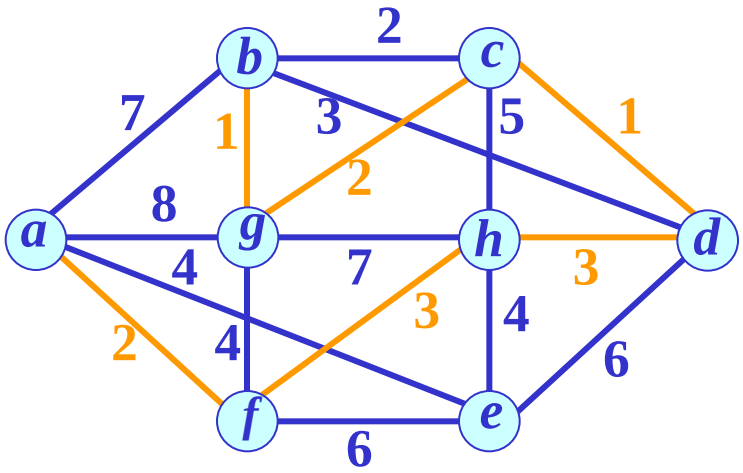


(d)

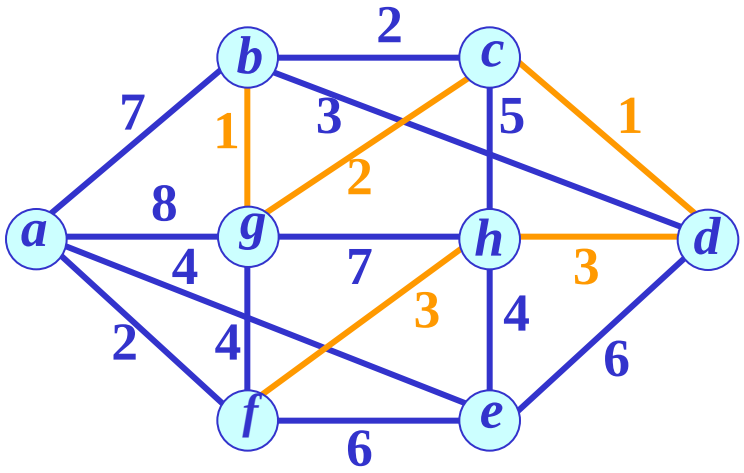
5.4 最小生成树（续七）



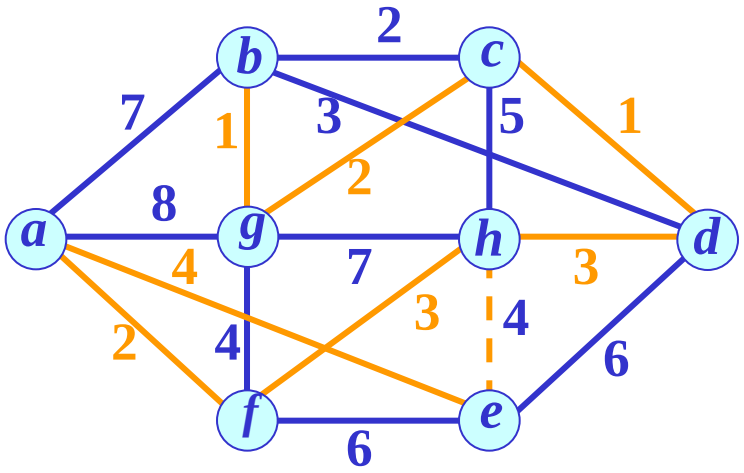
(e)



(g)



(f)

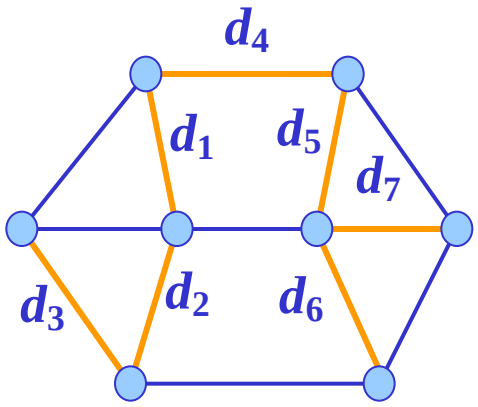


(h)

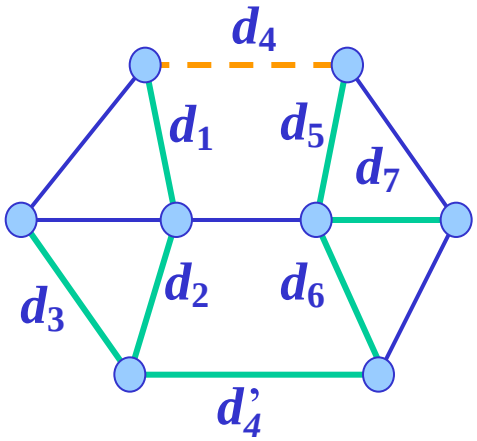
5.4 最小生成树（续八）

定理 2 任意给定最小生成树的一个实例，**Prim** 算法都可以在 $O(|V|^2 \log|V|)$时间内找到一棵最小生成树。

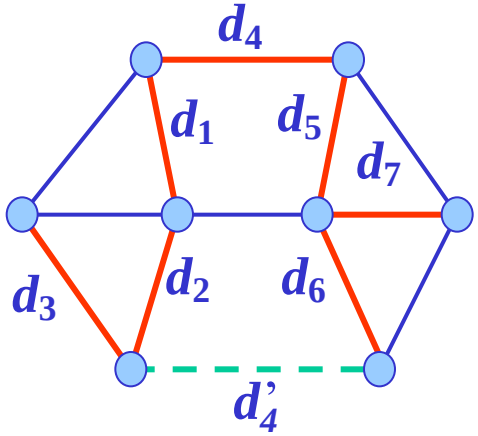
证明： 与**定理1**的证明思想一样，也是用反证法。



Prim算法求得解T_P



一棵最小生成树 T



另一棵最小生成树 T'

练习 求出前面实例中所给出的边赋权连通图的一棵最大生成树。

5.4 斯坦纳最小树

现在我们考虑**斯坦纳最小树问题**，它看起来与刚讨论过的最小生成树问题，但是是一种实质性的推广形式。该问题不是要将给定连通图中的所有顶点都连接起来，而是将事先给定的一个顶点子集中的所有点连接起来。

问　题: 图上斯坦纳最小树问题

实　例: 连通图 $G(V, E)$，每条边 $(u, v) \in E$ 都赋予权值 $w(u, v)$ 和一个顶点子集 $S \subset V$。

可行解: 图 G 的一棵将 S 中所有顶点都连接起来的树 T。

目　标: 使得树 T 的权值最小，$c(T) = \sum_{(u, v) \in T} w(u, v)$

定理 3 (M. R. Garey et al, 1979)

图上斯坦纳最小树问题是**NP-难**解的。

5.4 斯坦纳最小树（续一）

利用构造最小生成树的算法，我们可以设计一个求解斯坦纳最小树问题的 **2-**近似算法。

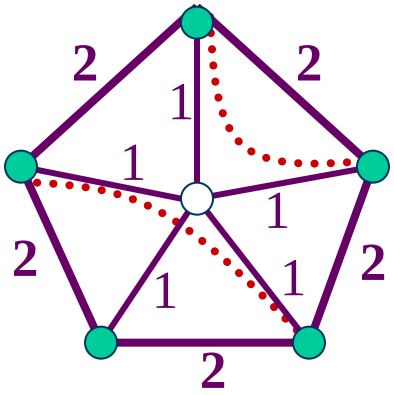
考虑一个求最小值的组合优化问题 $\prod$，设算法 **A** 是求解该问题的一个多项式时间算法。称算法 **A** 是一个**α-绝对近似算法** (α-Absolute Approximation)，如果存在一个常数α，对于每一个实例 **I**，都可求出一个解

$$| c_A(I) - c_{opt}(I) | \le \alpha$$

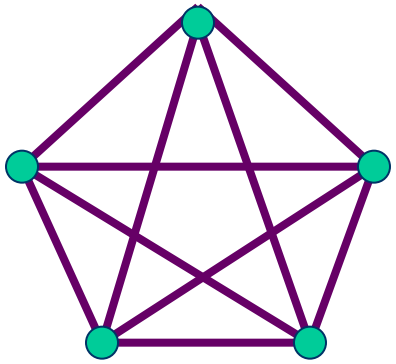
称算法 **A** 是一个 **α-近似算法** (α-Approximation)，如果存在一个常数α，对于每一个实例 **I**，都可求出一个解

$$1 \le \frac{c_A(I)}{c_{opt}(I)} \le \alpha$$

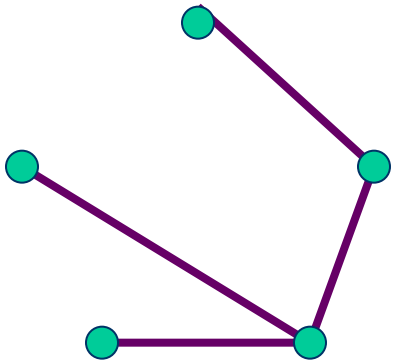
5.4 斯坦纳最小树（续二）



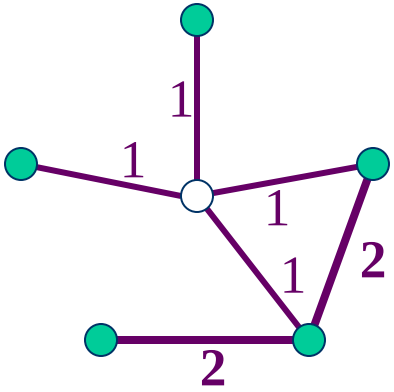
给定的原始图 ***G***



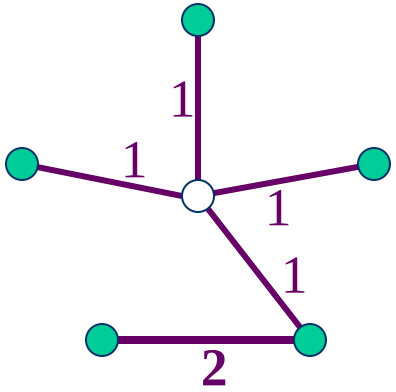
构造辅助图 ***G'***



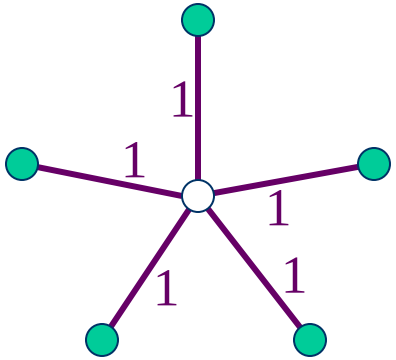
构造图 ***G'*** 的最小生成树



将树中的边用 ***G*** 中相应的路替换



破圈得到一棵 ***G*** 中的树



最优解

5.4 斯坦纳最小树（续三）

定理 4 基于最小生成树的算法是求解斯坦纳最小树问题的一个 **2-**近似算法。

证明. 设 T_{opt} 是一棵斯坦纳最小树，我们可以使用其中的每条边两次，构造一个有向闭圈C。再沿着 C 中的所给的顶点集 S 巡游一圈，得到一个圈 C'。因为C' 中每条边的长度是图 G 中两个端点的最短距离，所以$w(C') \le w(C)$。现将 C' 中最长的边 e 去掉，即得辅助图G' 的一棵生成树 T'。注意 $w(e)$ 至少是 $w(C')$ 的$1/|S|$。由此可得

$$w(T') \le \left(1-\frac{1}{|S|}\right) w(C') \le \frac{|S|-1}{|S|} w(C') \le \frac{2(|S|-1)}{|S|} w(T_{opt})$$

因为基于最小生成树算法构造的树 T 是辅助图G' 的一棵最小生成树，所以其长度 $w(T) \le w(T')$。

5.4 最优指派

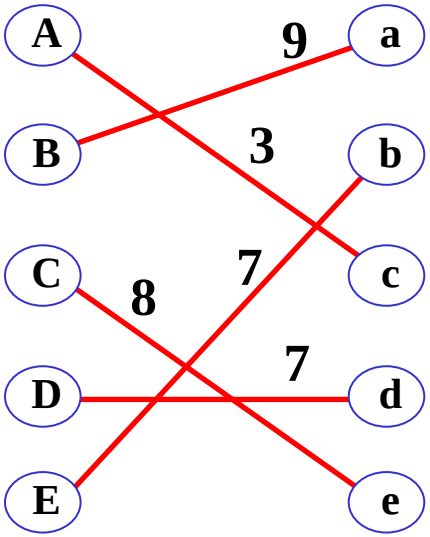
考虑一个**酋长嫁女问题**：古时一个部落酋长有5个女儿，他想将她们分别嫁给5户人家。按照习俗，每家都会给酋长一份彩礼，彩礼的多少与男孩儿的家境和对其女儿的满意程度有关。酋长想找到一种婚嫁方案，使其可收到尽可能多的彩礼。

酋长嫁女		男孩				
		a	b	c	d	d
女儿	A	6	5	3	7	4
	B	9	7	5	2	6
	C	8	4	5	3	8
	D	7	5	4	7	4
	E	1	7	6	7	6

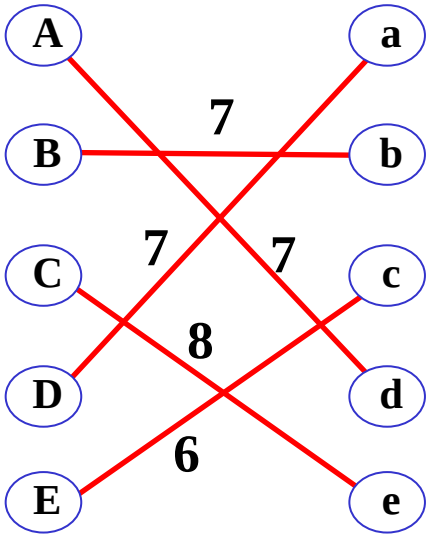
酋长采取了贪婪策略：每一次考虑一个女儿，将某个女儿嫁个能得到最多彩礼的男孩。

5.4 最优指派（续一）

酋长嫁女		男 孩				
		a	b	c	d	e
女儿	A	6	5	3	7	4
	B	9	7	5	2	6
	C	8	4	5	3	8
	D	7	5	4	7	4
	E	1	7	6	7	6



用贪婪策略得到彩礼34



最优指派得到彩礼35

5.4 最优指派（续二）

酋长嫁女问题可以很容易化成**指派问题**：假定有m 个工人和 m 项任务。若指派工人 i 去完成任务 j，其产生的费用为（工资）c_{ij}。我们希望付给这些工人尽可能少的工资以完成这些任务。若一个基础可行解 $x_{ij}=1$，则表示指派工人 i 去完成任务 j，$x_{ij}=0$ 表示不指派工人 i 去完成任务 j。通过松弛变量 x_{ij} 的整数性约束条件，该问题可以化成**线性规划问题**的标准型。

$$\begin{aligned}
&\textbf{Min} \sum_i \sum_j c_{ij} x_{ij} \\
&\textbf{s.t.} \sum_j x_{ij} = 1, \qquad i=1,\ldots, m; \\
&\qquad \sum_i x_{ij} = 1, \qquad j=1,\ldots, m; \\
&\qquad x_{ij} = 1 \text{ 或 } 0,\ i, j=1,\ldots, m.
\end{aligned}$$

若c_{ij}表示第i个女儿嫁给第j个男孩所得到的彩礼，酋长嫁女的目标函数 **Max** $\sum_i \sum_j c_{ij} x_{ij}$ 可以表示为 **Min** $\sum_i \sum_j (M-c_{ij}) x_{ij}$ 其中M是一个足够大的数。指派问题可以用**匈牙利算法**求解。

5.4 最大可满足

我们下面考虑第一个**NP-C**问题，可满足问题，的优化形式，**最大可满足问题**。

问　题: 最大可满足
实　例: n 布尔变量的一个合取范式, $C = C_1 \times C_2 \times \ldots \times C_k$
可行解: n 布尔变量的一个0-1赋值
目　标: 使得赋值 F 满足尽可能多的子句

求解最大可满足问题的一个非常简单的**贪婪算法**如下：选取这样一个布尔变量，将其赋值 **0** 或者 **1** 可以满足最多的子句；然后给其赋相应的布尔值，并将它的“字”从所有为被满足的子句中移除；重复此过程直到所有的子句都被满足或者没有子句可以考虑了。

5.4 最大可满足（续一）

ALGORITHM 7.3 *Greedy Algorithm*

for all $x \in X$ **do** $f(x) := true$.
while $C \neq \emptyset$ **do**
    $l :=$ the literal that occurs in the maximum number of clauses in C;
    $C_l :=$ the set of clauses in which l occurs;
    $C_{\overline{l}} :=$ the set of clauses in which $\overline{l}$ occurs;
    $x_l :=$ the variable corresponding to l.
    if l is positive **then**
        $C := C \setminus C_l$;
        remove $\overline{l}$ from all clauses in $C_{\overline{l}}$;
        remove all empty clauses in C.
end-while
return f.

5.4 最大可满足（续二）

定理 5 任给最大可满足问题的一个实例，贪婪算法可以在 $O(\max\{|C|\,|X|^2, |C|\,|X|\})$ 时间内给出一个赋值 F，它所满足的子句个数至少是最优赋值所能满足的子句个数的 **1/2** 倍。

证明. 假设给定的实例含有 $|C|$ 个子句。我们针对布尔变量的个数进行归纳证明，贪婪算法总能至少满足 $|C|/2$ 个子句。这样就可以证明定理了，因为任意一个赋值方法最多也就能满足 $|C|$ 个子句。

显然，当只有一个布尔变量时，定理结论成立。现在假定定理结论对 $n-1$ 个变量情形成立，并考虑 n 个变量的情形。假设变量 x_i 的“字”出现在最多的子句中。此外，再不妨假设，将 x_i 赋值“真”可以满足 c_t 个子句，将 x_i 赋值“伪”可以满足 c_f 个子句，且 $c_t \geq c_f$。

5.4 最大可满足（续三）

根据贪婪算法所采取的策略，应给变量 x_i 赋值“真”。这样至少还有含个 $n-1$ 变量的 $|C|-c_t-c_f$ 个子句需要考虑。依据归纳假设，这些子句中至少有 $(|C|-c_t-c_f)/2$ 个子句可以被贪婪算法的随后赋值所满足。因而，贪婪算法所满足的子句个数至少是

$$c_t+(|C|-c_t-c_f)/2 \geq |C|/2。$$

定理结论成立。

练习 考虑求解最大可满足问题的一个更加简单的算法：首先将所有变量赋值为“真”；然后，再将所有变量赋值为“伪”；最后，比较这两种赋值方法所能满足的子句的个数，采用其中比较好的赋值方法。证明这个算法的近似比同样是 2。

5.4 背包问题

我们现在考虑**背包问题**：我们一家三口明天要出去郊游，今天要准备带的东西。我只有一个背包，可是有许多物品都想要带，每个物品的重要性/价值和它的体积/重量都不同。假如我的包装不下所有的物品，那么我应该装哪些物品呢？

问　题：背包

实　例：n 件物品$X=\{x_1, \ldots, x_n\}$，物品 x_i 可产生效益 $p_i \in \mathbf{Z}^+$，但占用空间 $s_i \in \mathbf{Z}^+$，容积上限 $c \in \mathbf{Z}^+$。

可行解：若干件物品 $Y \subseteq X$ 使得 $\sum_{x_i \in Y} s_i \leq c$。

目　标：最大化 Y 中物品的效益之和 $\sum_{x_i \in Y} p_i$。

处理背包问题的一个非常自然的想法就是**贪婪策略**： 首先按照每个物品的单位体积的价值，p_i / s_i，从大到小排列，然后一个一个考虑是否可以放入背包（直到背包装不下了或者所有物品都考虑了）。

5.4 背包问题（续一）

n 个物品

$p_1=1$ $s_1=1$	$p_2=1$ $s_2=1$	$p_3=1$ $s_3=1$	$p_i=1$ $s_i=1$

$p_n = kn - 1$
$s_n = kn$

一个背包

$c = kn$

贪婪装包方法
装$n-1$个物品

$p_1=1$ $s_1=1$	$p_2=1$ $s_2=1$	$p_3=1$ $s_3=1$	$p_i=1$ $s_i=1$

$p = n - 1$
$s = n - 1$

最优装包方法
仅装一个物品

$p_n = kn - 1$
$s_n = kn$

5.4 背包问题（续二）

上面这个例子说明，贪婪装法所产生的价值与最优装法所产生的价值相比可以任意小。原因尽管贪婪装法有它的合理性（一般情况可以带来很好的价值），但是它可能“挑了芝麻，丢了西瓜”。这就促使我们**改进贪婪装法**：先用贪婪算法装一次，再用“先挑西瓜，再拣芝麻”方法装一次，最后，选择其中比较好的一种装法。

定理 6 改进的贪婪装法所产生的价值不会少于最优装法所产生的价值的**1/2**；算法所用时间不超过 $O(n \log(n\, c\, p_{\max}))$。

证明 假定给定的物品集合是 X，记贪婪装法所产生的价值为 $c_{IG}(X)$，而最优装法所产生的价值为 $c_{opt}(X)$。假设 x_j 是第一个用贪婪算法装不进背包的物品，则有

5.4 背包问题（续三）

$$\bar{p}_j \equiv p_1 + p_2 + \ldots + p_{j-1} \le c_{IG}(X),$$

且有

$$\bar{s}_j \equiv s_1 + s_2 + \ldots + s_{j-1} \le c.$$

我们来证明 $c_{opt}(X) \le p_j + \bar{p}_j$.

因为每一个物品 x_i 都是根据其单位体积所能产生的价值 p_i/s_i，由大到小排列和考虑的，因此将已装入背包的物品 $x_1, \ldots, x_{j-1}$ 中的任何物品取出，放入未装入背包的物品 $x_j, \ldots, x_n$ 中的若干物品（只要它们装入后包内所占体积不超过 $\bar{s}_j$），价值不会增加。也就是说最优装填所产生的价值不会超过 $\bar{p}_j$ 加上剩余空间（都装满）所可能产生的价值。另外注意到，$s_j + \bar{s}_j > c$。这样我们可以得到以下不等式

$$c_{opt}(X) \le \bar{p}_j + (c - \bar{s}_j)\frac{p_j}{s_j} \le \bar{p}_j + p_j.$$

5.4 背包问题（续四）

再注意到，若 $p_j \le \bar{p}_j$，则有

$$c_{\text{opt}}(X) \le 2\bar{p}_j \le 2c_{\text{G}}(X) \le 2c_{\text{IG}}(X)。$$

而若 $p_j > \bar{p}_j$，则有 $p_{\max} \ge \bar{p}_j$，因此可得

$$c_{\text{opt}}(X) \le \bar{p}_j + p_j \le \bar{p}_j + p_{\max} \le 2p_{\max} \le 2c_{\text{IG}}(X)。$$

以上分析说明，不论两种情况出现那一种，最优装法所产生的价值都不会超过改进贪婪装法所产生的价值的 2 倍。

最后，给 n 个数由大到小排序所用时间不超过 $O(n \log n)$，问题的输入不超过 $O(n \log p_{\max}) + O(n \log c)$。

下面我们说明如何将上述的贪婪算法进行推广，使其可以得到一个更好的近似解，尽管它的时间复杂度增加了，但是它仍然是一个多项式时间的算法。

5.4 背包问题（续五）

这个算法的基本思想如下：将所有的物品分成两组，

第一组$= \{x_i \mid p_i \le p\}$，

第二组$= \{x_j \mid p_j > p\}$，

这里 p 是一个预先设定的常数，算法中令 $p := \varepsilon c_G$。

注意：对任意一个可行解，价值超过 p 的物品最多有 $c_{\text{opt}}/p \le 2c_G/p$ 个。因此，可以在第二组中考虑物品数不超过 $2c_G/p$ (亦即 $2/\varepsilon$) 的所有子集合 $S \subseteq \{1, 2, \ldots, n\}$。对每个这样的子集合 S，在剩余的空间应用贪婪算法在第一组中求一个近似解。这样便可得到不超过 $n^{2c_G/p}$ 个近似解。最后，在这些近似解中找一个最好的近似解。

5.4 背包问题（续六）

定理 7 推广的贪婪装法可在 $O(n^{1+2/\varepsilon})$ 时间内求得一个近似解，其产生的价值 c_{GG} 不少于最优装法所产生价值的 $1/(1+\varepsilon)$ 倍。

证明. 设最优解中所有物品的指标集合是 S^*，

$$\sum_{i\in S^*} p_i = c_{opt} \quad 和 \quad \sum_{i\in S^*} s_i \le c$$

令 $S'=\{i \in S^* \mid p_i > p=\varepsilon c_G\}$。则有 $|S'| \le \frac{c_{opt}}{p} \le 2\frac{c_G}{p} = \frac{2}{\varepsilon}$

因此，算法会取S'作为最终的子集合 S。再由**定理 6** 的证明可知，$c(S') \le c_{opt} \le c(S') + p$。又因为算法最终选定的解是所有解 $c(S)$ 中价值最大的。固有$c(S') \le c_{GG} \le c_{opt} \le c(S') + p \le c_{GG} + p$。令 $S_G=\{i \in c_G \mid p_i > p\}$，则有 $c(S_G) \le c_G$。同时 $|S_G| \le c_G/p \le 1/\varepsilon$。因此有 $c_G \le c_{GG}$。由此可得

$$c_{opt} \le c_{GG} + p = c_{GG} + \varepsilon c_G \le (1+\varepsilon)\, c_{GG}。$$

5.4 背包问题（续七）

最后，因为最多只有 $n^{2/\varepsilon}$ 个指标集 S 所含的物品个数 $|S| \le 2/\varepsilon$，所以算法的运行时间是 $O\left(n^{1+2/\varepsilon} \log (c\, n\, p_{\max})\right)$。

注意，尽管推广的贪婪算法的运行时间是 n 的一个多项式，但是当趋于 0 时，其运行时间则是 $1/\varepsilon$ 的一个指数函数。下面介绍一个权衡算法，它可以将关于 $1/\varepsilon$ 增长的运行时间降下来。

输入：背包问题的实例 I，c，p_i 和 s_i, $i = 1, 2, \ldots, n$

取 $h > 0$

令 $p_i' := \lfloor p_i\, n\, (h+1)/p_{\max} \rfloor$，$i = 1, 2, \ldots, n$。

求出背包问题新实例 I' 的最优解 x_i, $i = 1, 2, \ldots, n$

设最优解放入背包中的物品指标集为 T

输出：背包问题原实例的可行解 $\{x_i,\ i \in T\}$

5.4 背包问题（续八）

定理 8 推广的权衡算法求得的最优解的目标函数值 c_T不会少于最优装法所产生的价值的 $1/(1+1/h)$ 倍。

证明. 设原实例最优解 c_{opt} 中所有物品的指标集合是 S，即有

$$\sum_{i\in S} p_i = c_{opt} \quad 和 \quad \sum_{i\in S} s_i \le c。$$

另设新实例最优解 c'_{opt} 中所有物品的指标集合是 S'，即有

$$c_T = \sum_{i\in S'} p_i = \sum_{i\in S'} \frac{p_i n(h+1)}{p_{max}} \frac{p_{max}}{n(h+1)}$$

$$\ge \sum_{i\in S'} \left\lfloor \frac{p_i n(h+1)}{p_{max}} \right\rfloor \frac{p_{max}}{n(h+1)}$$

$$= \frac{p_{max}}{n(h+1)} \sum_{i\in S'} p'_i \ge \frac{p_{max}}{n(h+1)} \sum_{i\in S} p'_i$$

上面不等式成立是因为 S'为最优解 c'_{opt} 中所有物品的指标集合。

5.4 背包问题（续九）

$$= \frac{p_{\max}}{n(h+1)} \sum_{i \in S'} p_i' \geq \frac{p_{\max}}{n(h+1)} \sum_{i \in S} p_i'$$

$$\geq \frac{p_{\max}}{n(h+1)} \sum_{i \in S} \left(\frac{p_i n(h+1)}{p_{\max}} - 1 \right)$$

$$\geq c_{\text{opt}} - \frac{p_{\max}}{h+1} \geq c_{\text{opt}} \left(1 - \frac{1}{h+1}\right)$$

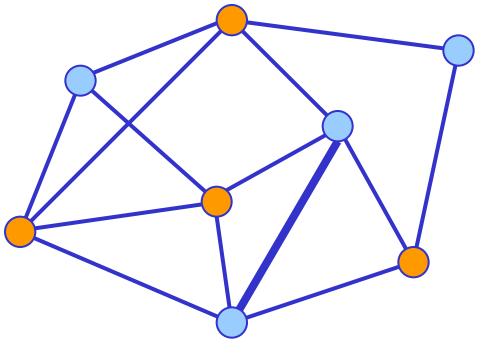
最后一个不等式成立是因为 $p_{\max} \leq c_{\text{opt}}$。

注意，在这个权衡算法中，求解背包问题的新实例需要的时间为 $O(n^3 p_{\max} \log(c\, p_{\max}))$，其中 $p_{\max} \leq n(h+1)$。因此，整个权衡算法所需要的运行时间为 $O(n^4 h \log(c\, n\, h))$，它是一个关于 n，$\log c$ 和 $h=1/\varepsilon$ 的多项式。

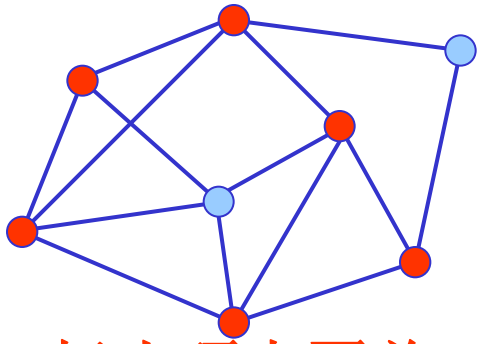
5.4 最小顶点覆盖

我们再来考虑**最少顶点覆盖问题:** 假设你是公安局治安处处长。你打算在城市的主要道路口都安装一个监视探头，用于监视每一条主要道路。由于经费有限，所以需要用尽可能少的探头。你应该安装在哪些路口呢？

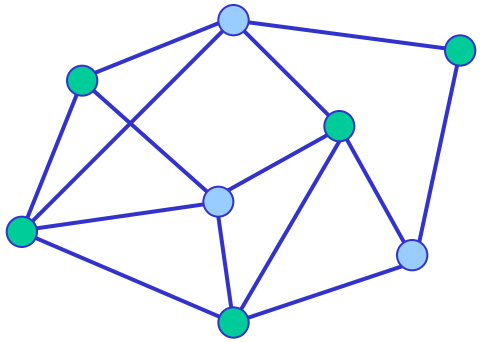
问　题: 最少顶点覆盖问题
实　例: 一个道路图 $G=(V, E)$
可行解: 顶点覆盖 $C \subseteq V$（每一条边至少有一个端点属于C）
目　标: 最小化顶点覆盖所包含的顶点个数 $|C|$



不是顶点覆盖



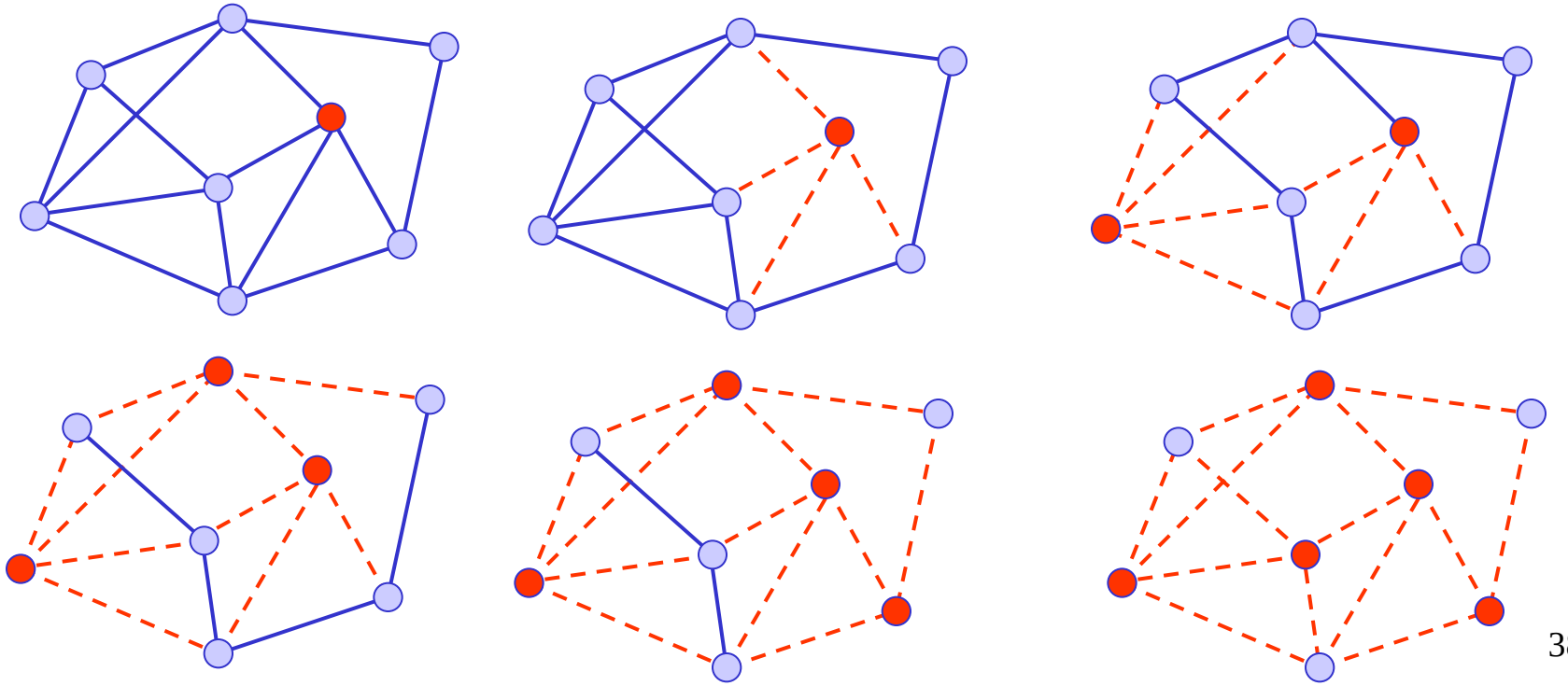
极小顶点覆盖



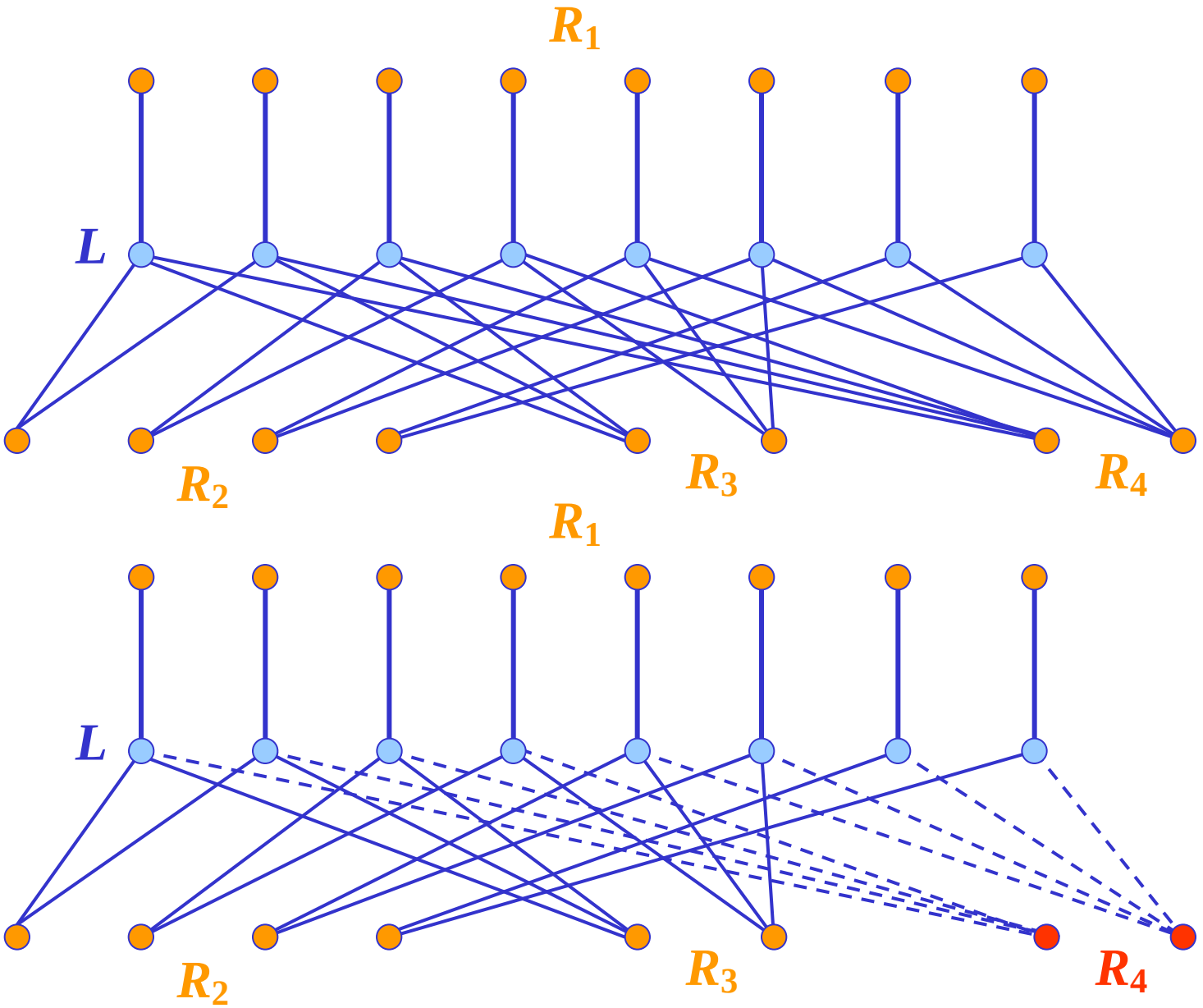
最小顶点覆盖

5.4 最小顶点覆盖（续一）

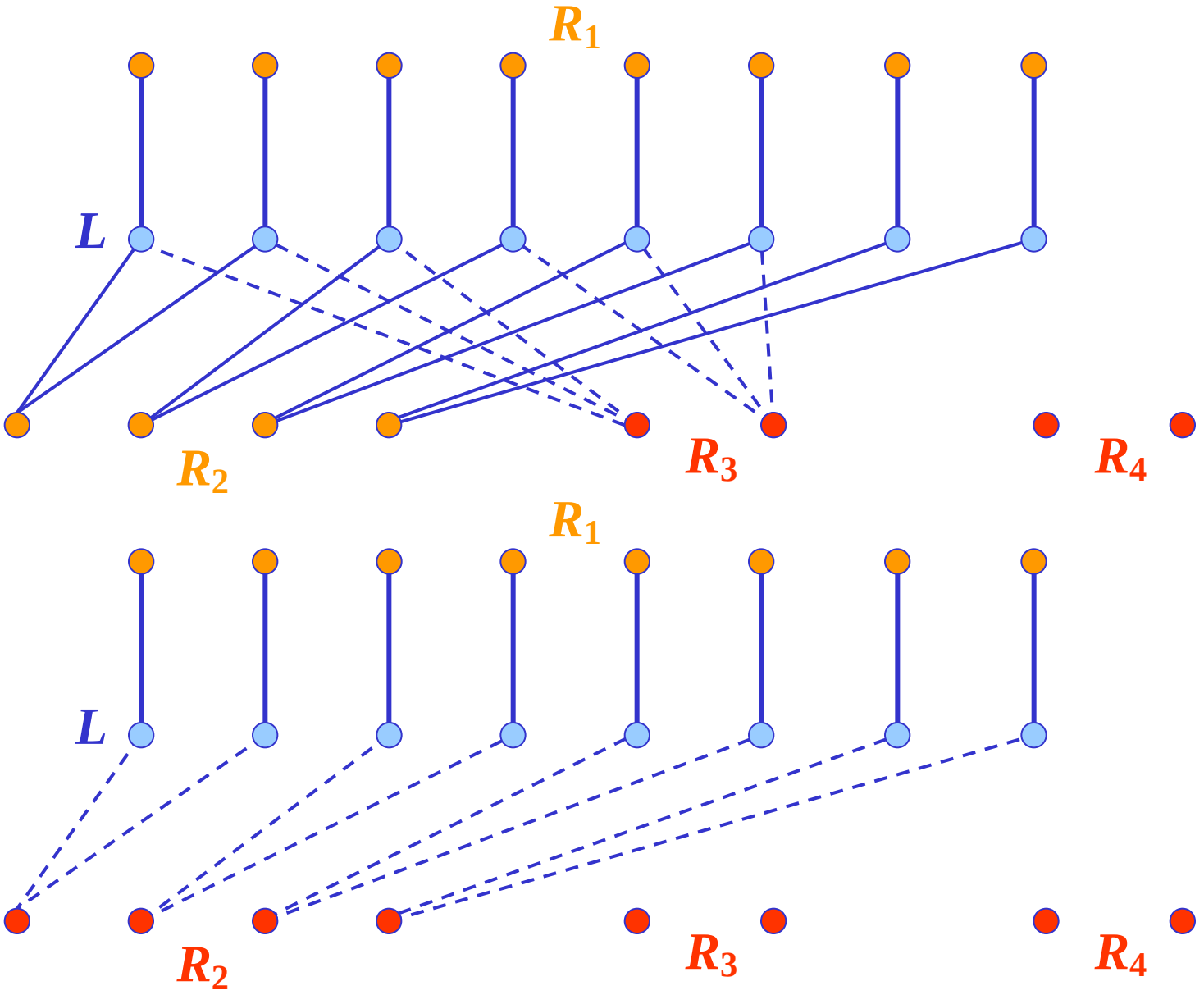
一个非常简单的**贪婪算法**是这样的：开始时，令 C 为空集。首先选取一个最大度数的顶点v (以它为端点的边的个数)，使得它可以覆盖最多的边，并把它放到 C 中。然后，去掉该顶点和以它为端点的所以的边(因为这些边都已经被覆盖了)。对剩下的图进行同样的运算，直到所有的边都被覆盖了为止。



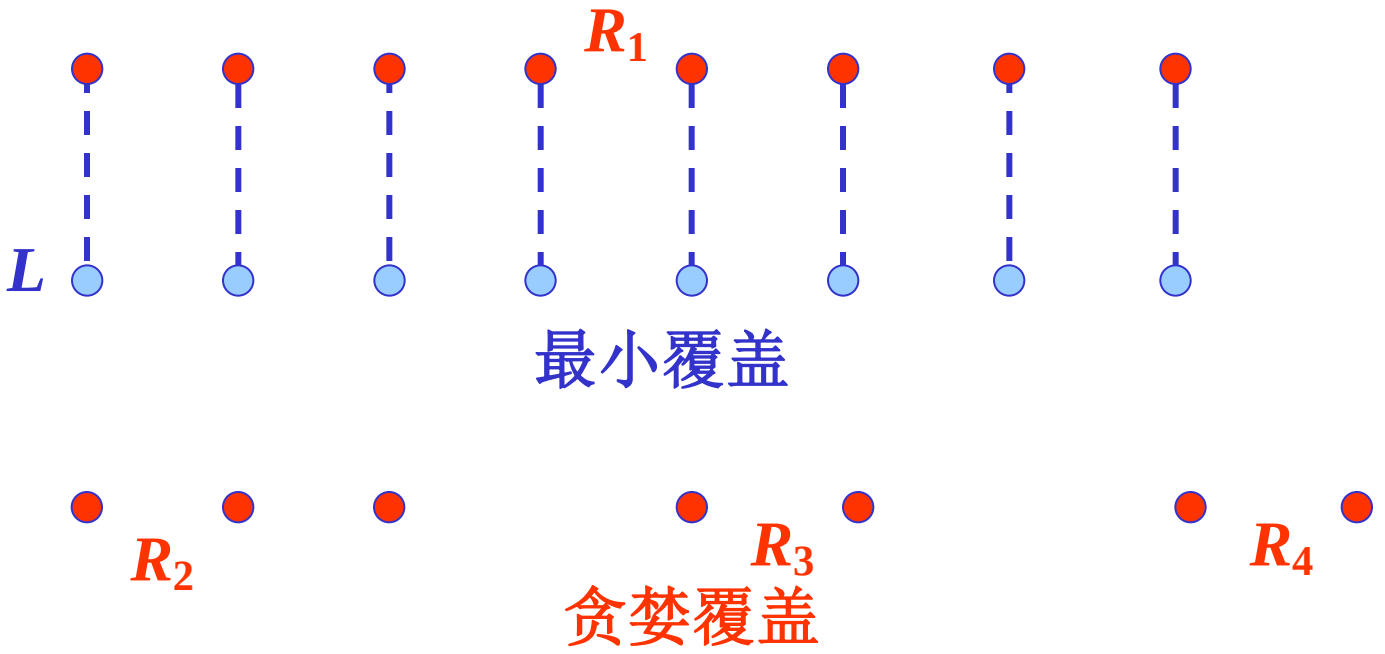
5.4 最小顶点覆盖（续二）



5.4 最小顶点覆盖（续三）



5.4 最小顶点覆盖（续四）

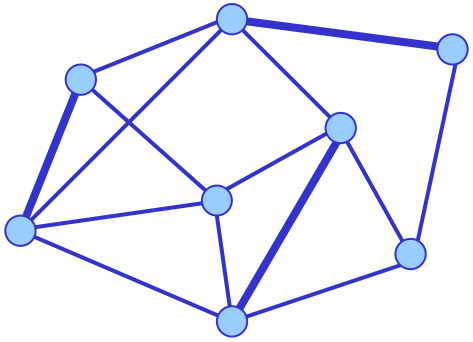


我们可以把这个例子进行推广，从而进一步说明，贪婪算法可能先后逐个选取 $R_k, R_{k-1}, \ldots, R_1$ 中的所有顶点，最后得到顶点覆盖 R。然而最小顶点覆盖却是 L。注意这个两个集合所含顶点个数之比是 $|R|/|L|=\theta(\log k)$。

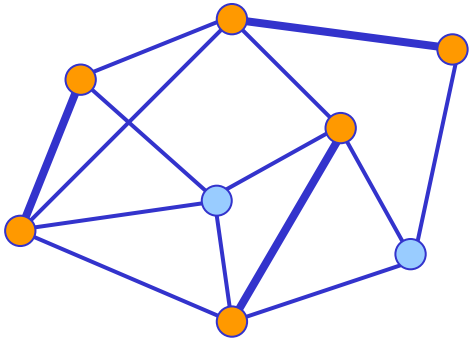
5.4 最小顶点覆盖（续五）

其实，有一个简单的基于**极大匹配的算法**：首先构造一个**极大匹配** M；然后把 M 中所包含的边的两个端点都放入 C 中。

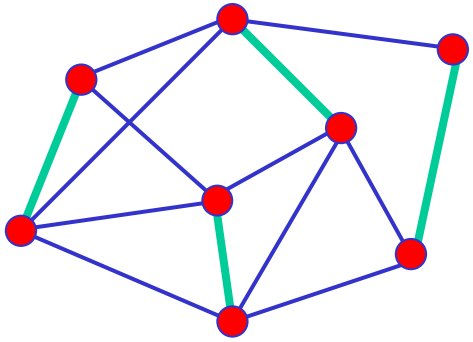
给定一个图 $G=(V, E)$，设 M 边集 E 的一个子集。如果 M 中任意两条边都没有公共的端点，则称 M 是一个**匹配**。若将 $E \backslash M$ 中任意一条边添加进 M，它都不再是匹配，则称 M 是极大匹配。显然，最大匹配一定是极大匹配。反之则不然。



极大匹配



基于**极大匹配**的**顶点覆盖**



基于**最大匹配**的**顶点覆盖**

5.4 最小顶点覆盖（续六）

定理 9 任意给定一个图$G=(V, E)$，基于**极大匹配的算法**所得到的顶点覆盖所含有的顶点个数 $|C_M|$ 不会超过**最小顶点覆盖**所含有的顶点个数 $|C_{opt}|$ 的**2**倍。

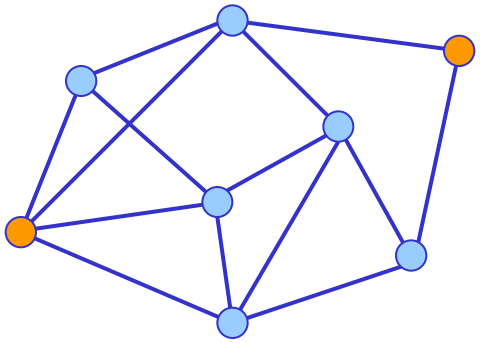
证明 注意边集E 中的每一条边 e 都至少有一个端点在 C_M 中，否则 M 就不是一个极大匹配 (因为还可以把边 e 添加进 M) 。另外，极大贪婪算法所得到的顶点覆盖含有顶点的个数 $|C_M|= 2|M|$。又因为，即使仅仅覆盖 M 中所有的边，任何一种方法都至少需要 $|M|$ 个顶点，因此有 $|C_{opt}| \geq |M|$。由这以上等式和不等式，即可推出定理结论。

思考题：能否找到一个图，使得该算法求出的顶点覆盖的个数恰好是最小顶点覆盖个数的2倍呢？

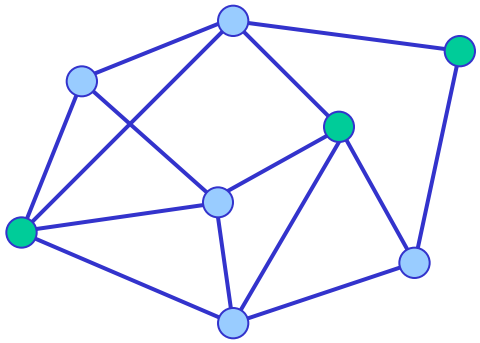
5.4 最大独立集

最后我们考虑**最大独立集问题**：假设你是一个研究所的所长助理，所长让你组织一个由尽可能多的部门负责人组成的评审委员会。你知道有些部门负责人之间存在“死党”关系，为了让这个委员会能得到尽可能公正的结果，你希望委员会包含的成员尽可能的多，且其中任意两个人都不存在“死党”。

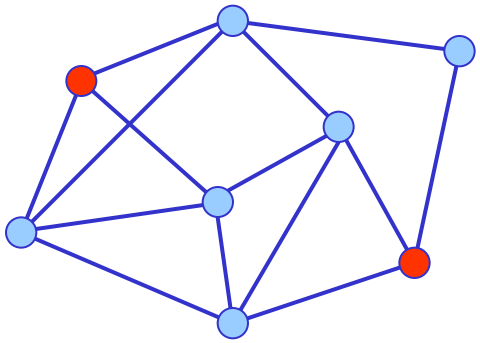
问　题: 最大独立集问题
实　例: 一个关系图 $G=(V, E)$
可行解: 独立集 $I \subseteq V$，即 I 中任意两点之间都没有边相连。
目　标: 最大化独立集中所含有的顶点个数 $|I|$。



一个独立集



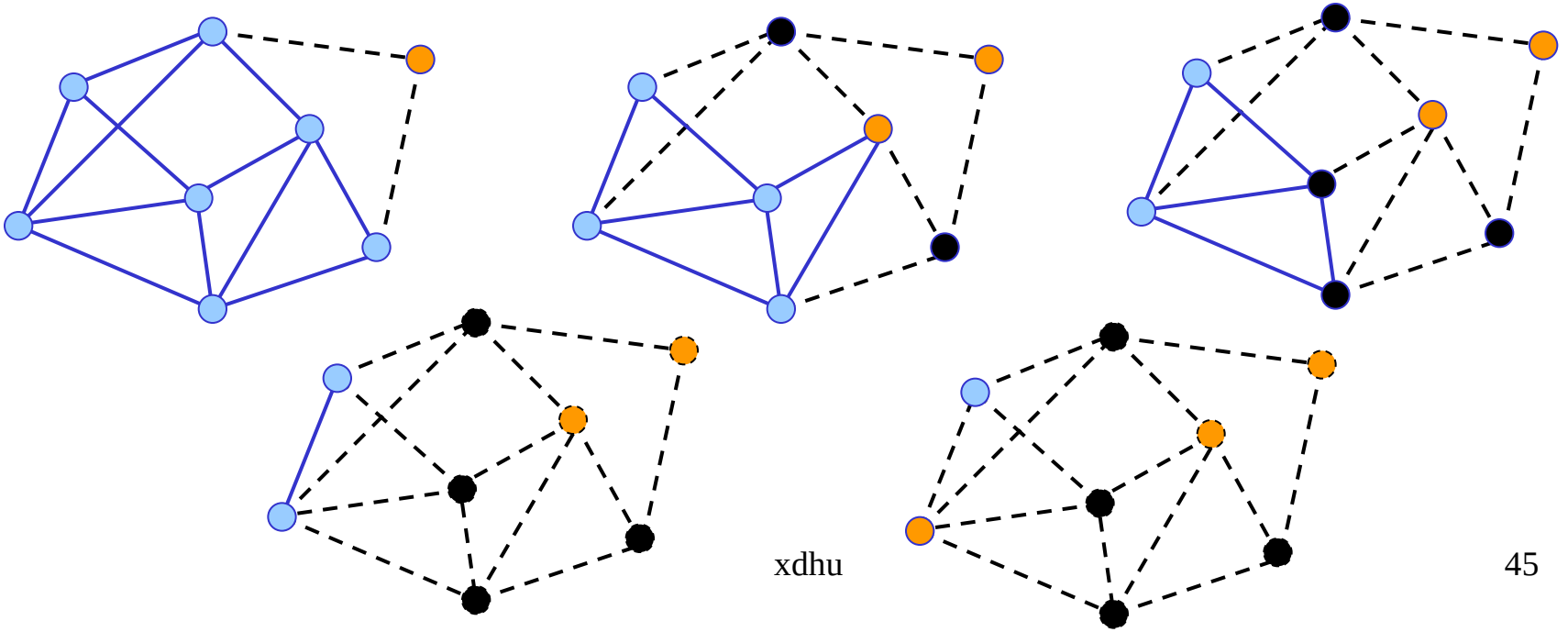
一个最大独立集



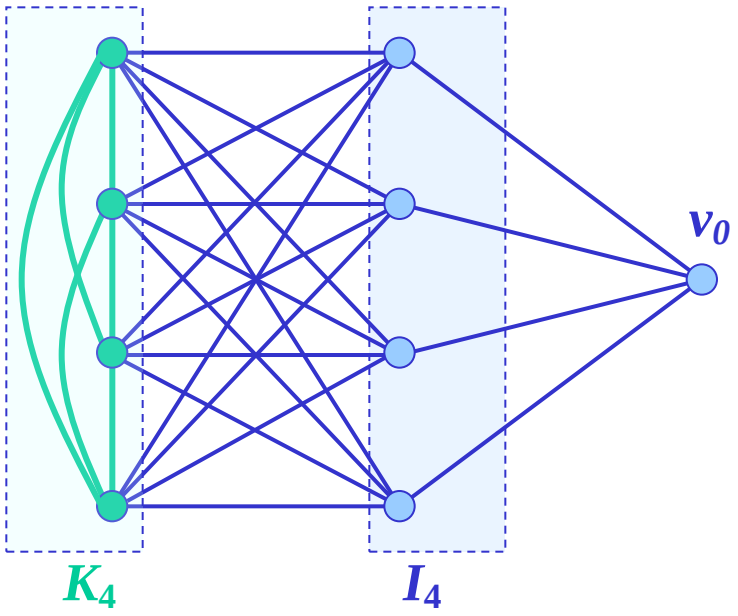
一个极大独立集

5.4 最大独立集（续一）

与处理最小顶点覆盖问题类似，最大独立集问题也有一个很简单的**贪婪算法**：初始时，令 ***I*** 为空集。首先选取一个度数最小的顶点***v***，使其尽可能与其它顶点没有关系，并把 ***v*** 放到 ***I*** 中（这样就有机会在 ***I*** 中加入更多的顶点）。然后，去掉 ***v*** 和与 ***v*** 有边相连的顶点（因为这些顶点不可能放入 ***I*** 中了）。最后，对剩下的图进行同样的运算，直到每一顶点可以选取了为止。



5.4 最大独立集（续二）



当我们用贪婪算法求左图的最大独立集时，开始令 I 为空集，首先将 v_0 放入 I 中；然后将集合 I_4 中所有顶点去掉，最后在集合 K_4 中任选一个顶点。所得独立集仅包含两个顶点，而显然最大独立集包含 I_4 中的四个顶点。很容易将这个例子进行推广。

然而，与最小顶点覆盖问题不同，不仅用贪婪算法不能得到一个独立集使其所含有的顶点个数不少于最大独立集所含顶点个数的一个常数倍，实际上，现在人们普遍认为根本就**不存在这样的算法**，换言之，贪婪算法对最大独立集问题无效，是因为这个问题**实在是太难了**。

5.4 总结

因为贪婪策略/方法/算法非常**直观**且**简单易行**，所以它们是人们在处理优化问题的时候最常用的一类方法。以上我们通过几个简单的例子，主要想说明以下四点：

- 有些优化问题可以用贪婪算法找到**最优解**。这样的优化问题都有特殊的结构，称作**拟阵**，或者目标函数是**次模函数**。
- 有些优化问题不可以用贪婪算法总找到**最优解**。
  然而，存在多项式时间算法总能求出这些问题的最优解。
- 有些优化问题可以用贪婪算法找到**很好的近似最优解**。
- 有些优化问题可以用贪婪算法找到一个**比较满意的解**。
- 有些优化问题可以用贪婪算法找到一个**比较好的初始解**，
  然后再通过其他方法或技巧（如迭代法、局部搜索法），
  找到更好的解。