

# 第11章 贪心算法



# 学习要点



- 理解贪心算法的概念。
- 掌握贪心算法的基本要素
  - (1) 最优子结构性质
  - (2) 贪心选择性质
- 理解贪心算法的一般理论
- 通过应用范例学习贪心算法策略。
  - (1) 活动安排问题；
  - (2) 最优装载问题；
  - (3) 哈夫曼编码；
  - (4) 最小生成树；
  - (5) 单源最短路径；
  - (6) 多机调度问题

# 前言



顾名思义，贪心算法总是作出在当前看来最好的选择。也就是说贪心算法并不从整体最优考虑，它所作出的选择只是在某种意义上的**局部最优选择**。

希望贪心算法得到的最终结果也是整体最优的。虽然贪心算法不能对所有问题都得到整体最优解，但对许多问题它能产生整体最优解。如单源最短路径问题，最小生成树问题等。在一些情况下，即使贪心算法不能得到整体最优解，其最终结果却是最优解的很好近似。

# 第一节 贪心算法的基本要素



# 前言



本节着重讨论可以用贪心算法求解的问题的一般特征。

对于一个具体的问题，怎么知道是否可用贪心算法解此问题，以及能否得到问题的最优解呢？

这个问题很难给予肯定的回答。

从许多可以用贪心算法求解的问题中看到这类问题一般具有2个重要的性质：**贪心选择性质**和**最优子结构性质**。

# 1.贪心选择性质



所谓**贪心选择性质**是指所求问题的整体最优解可以通过一系列**局部最优**的选择，即**贪心选择**来达到。这是贪心算法可行的**第一个基本要素**。

贪心算法则通常以自顶向下的方式进行，以迭代的方式作出相继的贪心选择，每作一次贪心选择就将所求问题**简化为规模更小的子问题**。

## 2.最优子结构性质



对于一个具体问题，要确定它是否具有贪心选择性质，必须证明每一步所作的贪心选择最终导致问题的**整体最优解**。

当一个问题的**最优解**包含其**子问题的最优解**时，称此问题具有**最优子结构性质**。

问题的最优子结构性质是该问题可用贪心算法求解的关键特征。

## 第二节 活动安排问题





# 活动安排问题



活动安排问题就是要在所给的活动集合中选出最大的相容活动子集合，是可以用贪心算法有效求解的很好例子。该问题要求高效地安排一系列争用某一公共资源的活动。

贪心算法提供了一个简单、漂亮的方法使得尽可能多的活动能兼容地使用公共资源。

# 活动安排问题



- 设有  $n$  个活动的集合  $E = \{1, 2, \dots, n\}$ ，其中每个活动都要求使用同一资源，如演讲会场等，而在同一时间内只有一个活动能使用这一资源。
- 每个活动  $i$  都有一个要求使用该资源的起始时间  $s_i$  和一个结束时间  $f_i$ ，且  $s_i < f_i$ 。
- 如果选择了活动  $i$ ，则它在半开时间区间  $[s_i, f_i)$  内占用资源。若区间  $[s_i, f_i)$  与区间  $[s_j, f_j)$  不相交，则称**活动  $i$  与活动  $j$  是相容的**。也就是说，当  $s_i \geq f_j$  或  $s_j \geq f_i$  时，活动  $i$  与活动  $j$  相容。

# 活动安排问题

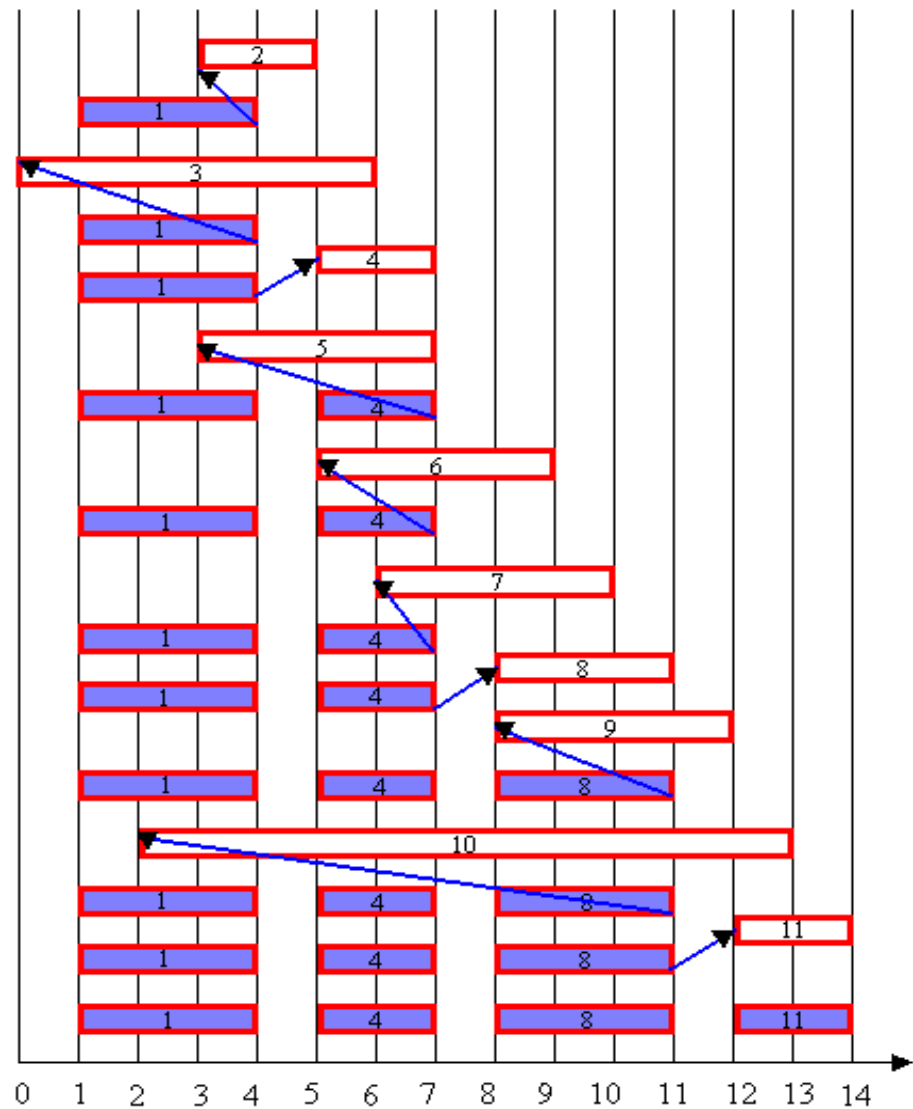


➤ 例：设待安排的11个活动的开始时间和结束时间按结束时间的非减序排列如下：

i	1	2	3	4	5	6	7	8	9	10	11
S[i]	1	3	0	5	3	5	6	8	8	2	12
f[i]	4	5	6	7	8	9	10	11	12	13	14

# 活动安排问题

算法greedySelector 的计算过程如左图所示。图中每行相应于算法的一次迭代。阴影长条表示的活动是已选入集合A的活动，而空白长条表示的活动是当前正在检查相容性的活动。



i	1	2	3	4	5	6	7	8	9	10	11
S[i]	1	3	0	5	3	5	6	8	8	2	12
f[i]	4	5	6	7	8	9	10	11	12	13	14

# 活动安排问题



由于输入的活动按照其完成时间的**非减序排列**，所以算法greedySelector每次总是**选择具有最早完成时间的相容活动加入集合A中**。直观上，**按这种方法选择相容活动为未安排活动留下尽可能多的时间**。也就是说，该算法的贪心选择的意义是**使剩余的可安排时间段极大化**，以便安排尽可能多的相容活动

算法greedySelector的效率极高。当输入的活动已按**结束时间的非减序排列**，算法只需 **$O(n)$** 的时间安排n个活动，使最多的活动能相容地使用公共资源。如果所给出的活动未按**非减序排列**，可以用 **$O(n\log n)$** 的时间重排。

# 活动安排问题



下面给出解活动安排问题的贪心算法

```
GreedySelector :  
template<class Type>  
void GreedySelector(int n, Type s[], Type f[], bool A[])  
{  
    A[1]=true;  
    int j=1;  
    for (int i=2;i<=n;i++) {  
        if (s[i]>=f[j]) { A[i]=true; j=i; }  
        else A[i]=false;  
    }  
}
```

各活动的起始时间和  
结束时间存储于数组  
s和f中且按结束时间  
的非减序排列。

# 活动安排问题贪心算法正确性



## (1) 贪心选择性质

首先证明活动安排问题有一个最优解以贪心选择开始，即该最优解中包含活动1。

设  $A \subseteq E$  是所给活动安排问题的一个最优解，且  $A$  中活动也按时间结束顺序时间非减序排列， $A$  中第一个活动是  $k$ ，若  $k=1$  则结论成立。

若  $k>1$ ，则设  $B=A-\{k\} \cup \{1\}$ 。由于  $f_1 \leq f_k$ ，且  $A$  中活动是相容的，故  $B$  中活动也是相容的。由于  $A$  和  $B$  活动数相同，故  $B$  和  $A$  一样也是最优活动安排。因此**总存在以贪心选择开始的最优解**，结论成立！

## (2) 最优子结构性质

需要证明：若  $A$  是原问题的最优解，则  $A' = A - \{1\}$  是活动安排问题  $E' = \{i \in E: s_i \geq f_1\}$  的最优解。

如果能够找到的一个解  $B'$ ，比  $A'$  包含更多的活动，则将1加入  $B'$  中将产生  $E$  的一个解  $B$ ，它包含比  $A$  更多的活动。**这与  $A$  最优产生矛盾。**

因此，每一步所作出的贪心选择都将问题简化为一个更小的与原问题具有相同形式的子问题。

对贪心选择次数采用**数学归纳法**即知：**贪心算法最终产生原问题的最优解。**

## 第三节 最优装载





# 最优装载



有一批集装箱要装上一艘载重量为 $c$ 的轮船。其中集装箱 $i$ 的重量为 $W_i$ 。最优装载问题要求**确定在装载体积不受限制的情况下，将尽可能多的集装箱装上轮船。**

# 算法描述



## 1. 算法描述

最优装载问题可用贪心算法求解。**采用重量最轻者先装的贪心选择策略**，可产生最优装载问题的最优解

```
template<class Type>
void Loading(int x[], Type w[], Type c, int n)
{
    int *t = new int [n+1];

    Sort(w, t, n); // t(i)=物品i的排序序号, 按重量的升序排序。

    for (int i = 1; i <= n; i++) x[i] = 0;

    for (int i = 1; i <= n && w[t[i]] <= c; i++) { x[t[i]] = 1; c -= w[t[i]]; }
}
```

# 最优装载



## 2.贪心选择性质

可以证明最优装载问题具有贪心选择性质。

## 3.最优子结构性质

最优装载问题具有最优子结构性质。

由最优装载问题的贪心选择性质和最优子结构性质，容易证明算法loading的正确性。

算法loading的主要计算量在于将集装箱依其重量从小到大排序，故算法所需的计算时间为  $O(n\log n)$ 。

# 最优装载问题贪心算法正确性



## (1) 贪心选择性质

设集装箱已按照重量从小到大排序,  $(x_1, x_2, \dots, x_n)$  是最优装载问题的一个最优解。又设  $k = \min_{1 \leq i \leq n} \{i | x_i = 1\}$ 。易知, 如果给定的最优装载问题有解, 则  $1 \leq k \leq n$ 。

(1) 当  $k=1$  时,  $(x_1, x_2, \dots, x_n)$  是一个满足贪心选择性质的最优解。

(2) 当  $k>1$  时, 取  $y_1 = 1, y_k = 0, y_i = x_i, 1 < i \leq n, i \neq k$ , 则

$$\sum_{i=1}^n w_i y_i = w_1 - w_k + \sum_{i=1}^n w_i x_i \leq \sum_{i=1}^n w_i x_i \leq c$$

因此,  $(y_1, y_2, \dots, y_n)$  是所给最优装载问题的可行解。

另外, 由  $\sum_{i=1}^n y_i = \sum_{i=1}^n x_i$ , 知道  $(y_1, y_2, \dots, y_n)$  是满足贪心选择的最优解。

## (2) 最优子结构性质

设  $(x_1, x_2, \dots, x_n)$  是最优装载问题的满足贪心选择的最优解, 则易知,  $x_1 = 1$ , 且  $(x_2, \dots, x_n)$  是轮船载重量为  $c - x_1$ , 待装船集装箱为  $\{2, 3, \dots, n\}$  时相应最优装载问题的最优解。也就是说, 最优装载问题具有最优子结构性质。

## 第五节 哈夫曼编码



# 1.哈夫曼编码基本原理



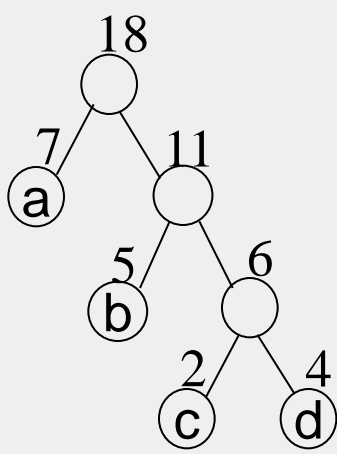
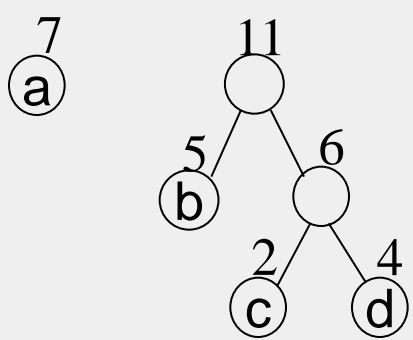
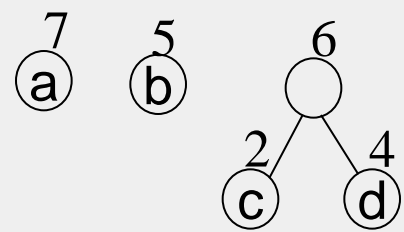
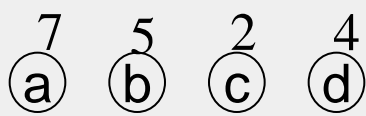
- 根据给定的 $n$ 个权值  $\{w_1, w_2, \dots, w_n\}$ ，构造 $n$ 棵只有根结点的二叉树，令其权值为 $w_j$
- 在森林中选取两棵根结点权值最小的树作左右子树，构造一棵新的二叉树，置新二叉树根结点权值为其左右子树根结点权值之和
- 在森林中删除这两棵树，同时将新得到的二叉树加入森林中
- 重复上述两步，直到只含一棵树为止，这棵树即**哈夫曼树**

$$WPL = \sum_{k=1}^n W_K L_K$$



有4个结点，权值分别为7，5，2，4，构造有4个叶子结点的二叉树

例



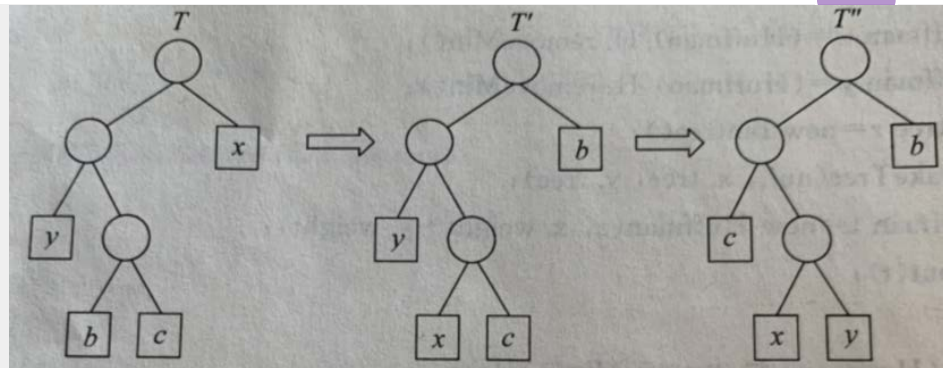
$$WPL = \sum_{k=1}^n W_K L_K$$

## 2. 哈夫曼贪心算法的正确性



### (1) 贪心选择性质

设 $C$ 是编码字符集， $C$ 中字符 $c$ 的频率为 $f(c)$ 。又设 $x$ 和 $y$ 是 $C$ 中具有最小频率的两个字符，存在 $C$ 的最优前缀码使 $x$ 和 $y$ 具有相同码长且仅最后一位不同



设二叉树 $T$ 表示 $C$ 的任意一个最优前缀码，如果能证明对 $T$ 作适当修改得到 $T''$ ，使得 $T''$ 表示的前缀码也是 $C$ 的最优前缀码。则结论成立。

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c)d_T(c) - \sum_{c \in C} f(c)d_{T'}(c) = f(x)d_T(x) + f(b)d_T(b) - f(x)d_{T'}(x) - f(b)d_{T'}(b) \\ &= f(x)d_T(x) + f(b)d_T(b) - f(x)d_T(b) - f(b)d_T(x) = (f(b) - f(x))(d_T(b) - d_T(x)) \geq 0 \end{aligned}$$

可类似证明在 $T'$ 交换 $y$ 与 $c$ 的位置也不增加平均码长，即 $B(T') - B(T'')$ 也是非负的。由此可知 $B(T'') \leq B(T') \leq B(T)$ 。另一方面，由于 $T$ 所表示的前缀码是最优的，所以 $B(T) \leq B(T'')$ 。因此 $B(T) = B(T'')$ ，即 $T''$ 表示的前缀码也是最优的，且 $x$ 和 $y$ 具有相同的码长，仅最后一位不同。



## 2. 哈夫曼贪心算法的正确性



### (2) 最优子结构性质

设 $T$ 是字符集 $C$ 一个最优前缀码。  $C$ 中字符 $c$ 的频率为 $f(c)$ ， 设 $x$ 和 $y$ 是 $T$ 中两个叶子且为兄弟，  $z$ 是它们的父亲。 若将 $z$ 看作是具有频率  $f(z)=f(x)+f(y)$  字符， 则树  $T' = T - \{x, y\}$  表示的字符集  $C' = C - \{x, y\} + \{z\}$  的一个最优前缀码。

首先证明 $T$ 的平均码长 $B(T)$ 可用 $T'$  的平均码长 $B(T')$ 表示。

对任意 $c \in C - \{x, y\}$ 有 $d_T(c) = d_{T'}(c)$ ， 故 $f(c)d_T(c) = f(c)d_{T'}(c)$ ， 且 $d_T(x) = d_T(y) = d_{T'}(z) + 1$

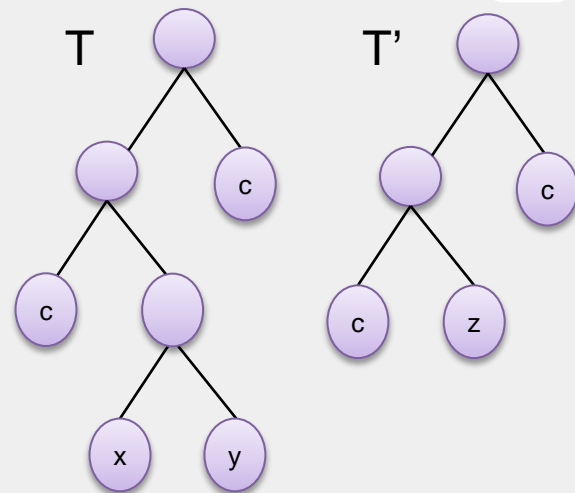
因此： $f(x)d_T(x) + f(y)d_T(y) = (f(x) + f(y))(d_{T'}(z) + 1) = f(x) + f(y) + f(z)d_{T'}(z)$

则： $B(T) = B(T') + f(x) + f(y)$

如果 $T'$ 所表示的字符集 $C'$ 的前缀码不是最优的， 则 $T''$ 表示的 $C'$ 的前缀码使得 $B(T'') < B(T')$ 。 由于 $z$ 被看作是 $C'$ 中一个字符， 故 $z$ 在 $T''$ 中是一颗叶子。 若将 $x$ 和 $y$ 加入树中作为 $z$ 的儿子， 则得到表示字符集 $C$ 的前缀码的二叉树 $T'''$ ， 且有：

$B(T''') = B(T'') + f(x) + f(y) < B(T') + f(x) + f(y) = B(T)$

产生矛盾， 因为 $T$ 是最优的



## 第六节 最小生成树



# 最小生成树



对于一个无向网络——无向加权连通图 $N=(V,E,C)$ ( $C$ 表示该图为权图)，其顶点个数为 $|V|=n$ ，图中边的个数为 $|E|$ ，我们可以从它的 $|E|$ 条边中选出 $n-1$ 条边，使之满足

(1) 这 $n-1$ 条边和图的 $n$ 个顶点构成一个连通图。

(2) 该连通图的代价是所有满足条件(1)的连通图的代价的最小值。

这样的连通图被称为网络的最小生成树( Minimum-cost Spanning Tree )。

# 1.Prim贪心算法基本步骤



## 普里姆(Prim)算法(逐点加入)

设 $N=(V,E,C)$ 为连通网,  $TE$ 是 $N$ 的最小支撑树MST的边的集合,  $U$ 为MST顶点集。

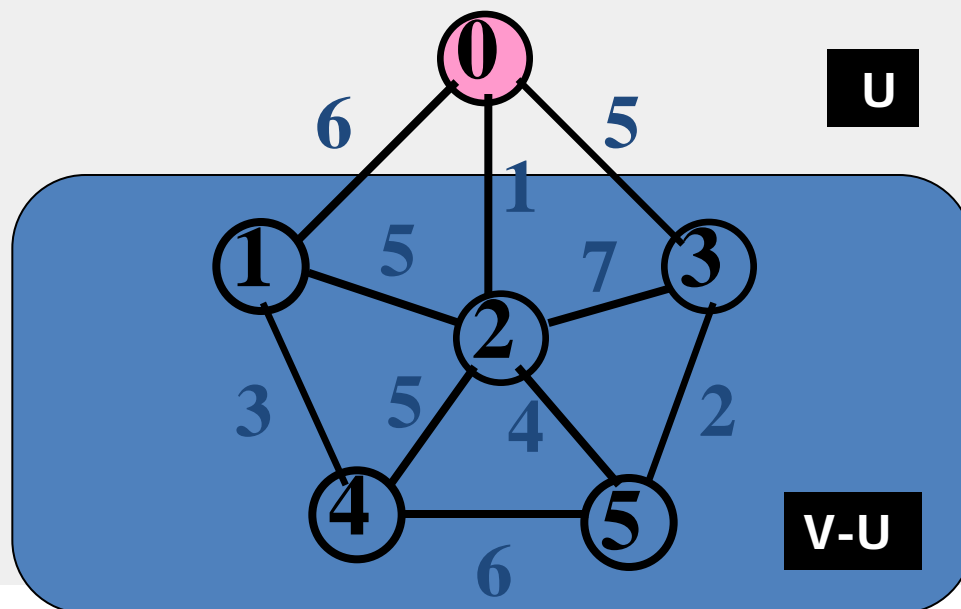
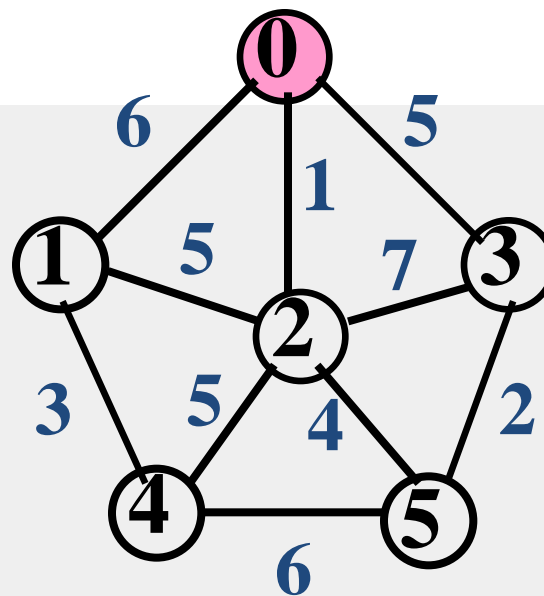
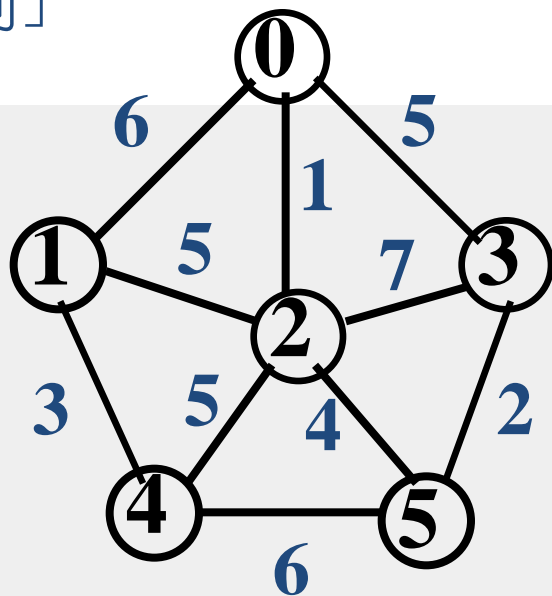
① 初始设 $U=\{u_0\}(u_0 \in V)$ ,  $TE= \Phi$  ;

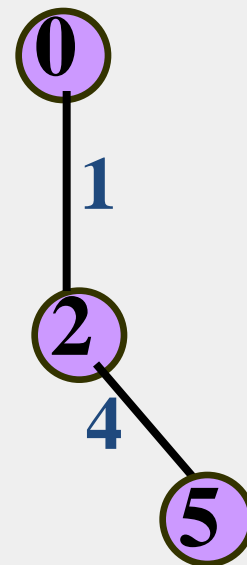
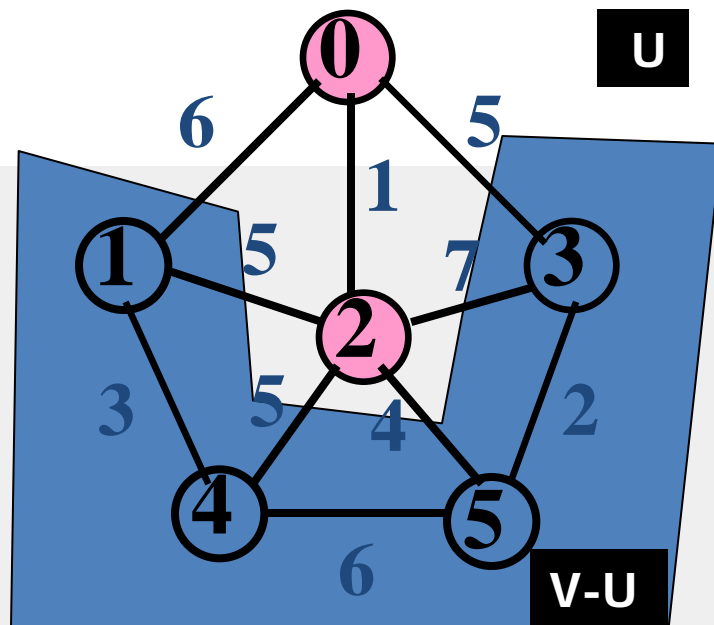
② 找到满足

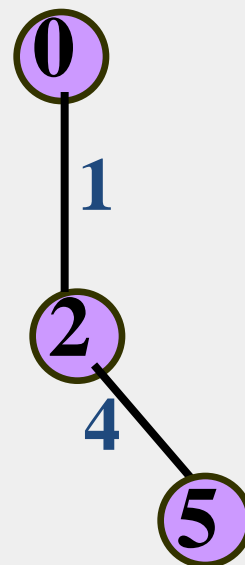
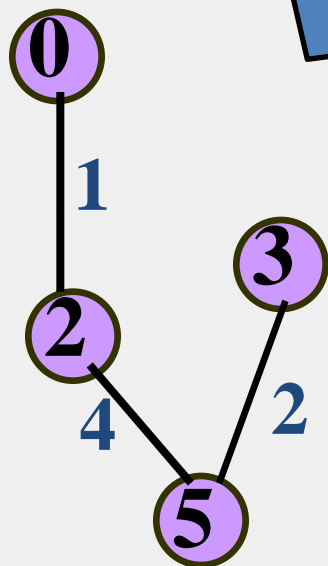
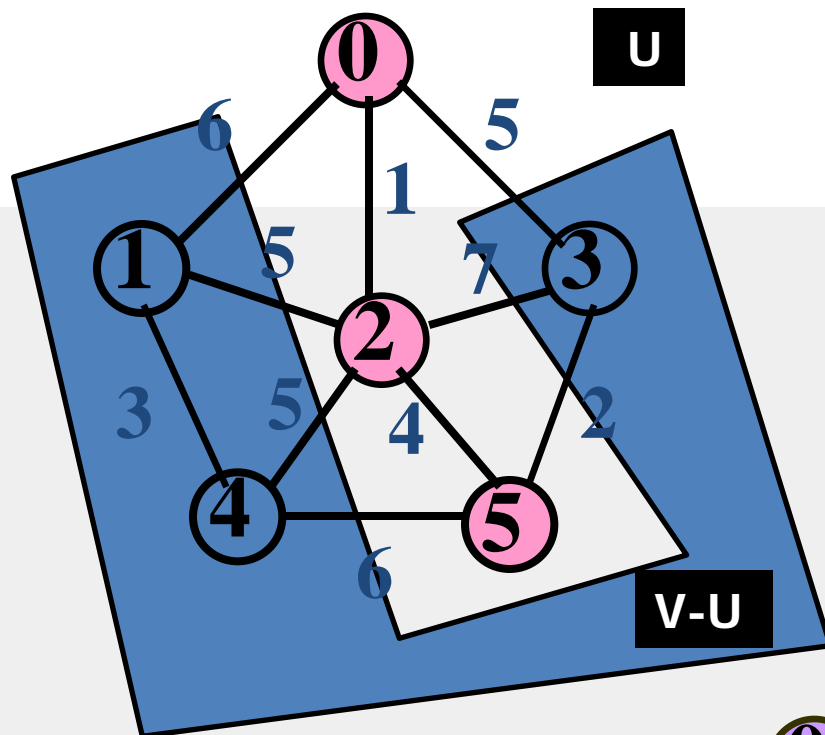
$weight(u,v)=\min\{weight(u_1,v_1)|u_1 \in U, v_1 \in V-U\}$ , 的边,  
把它并入 $TE$ , 同时 $v$ 并入 $U$ ;

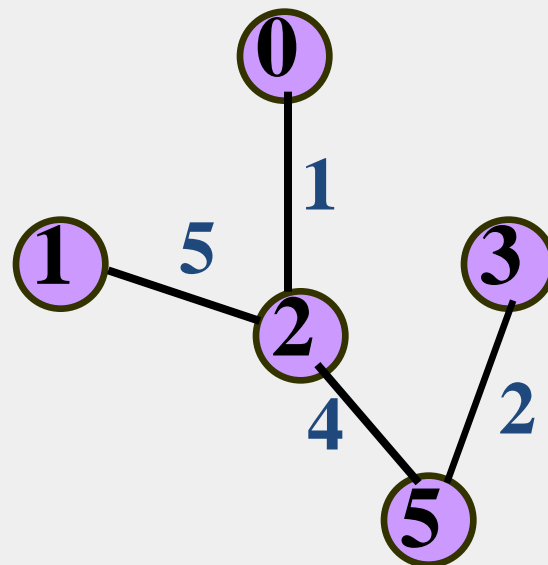
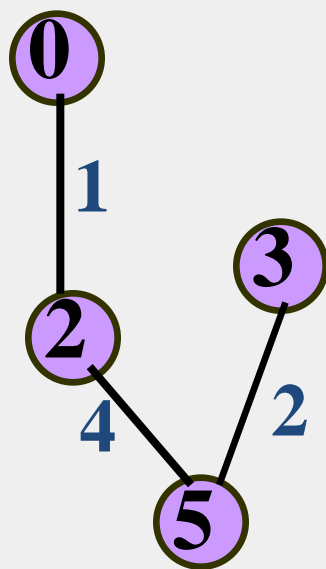
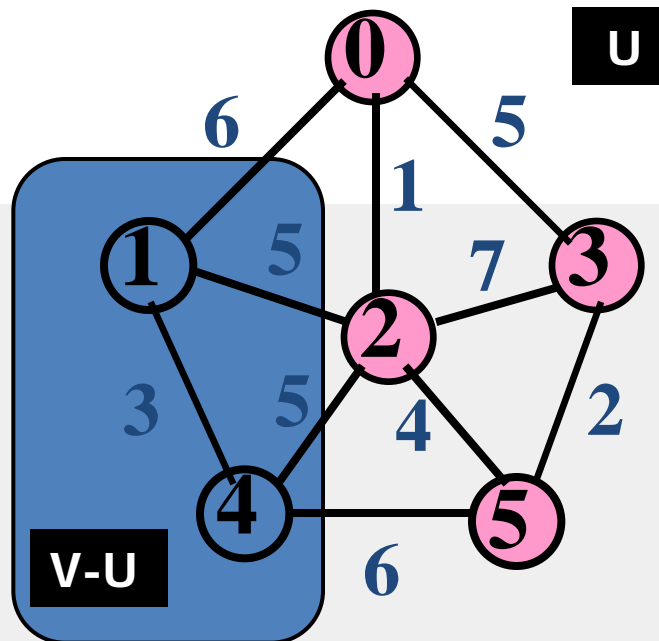
③ 反复执行②, 直至  $V=U$ , 则 $T=(V,\{TE\})$ 为 $N$ 的最小生成树, 算法结束。

[例]

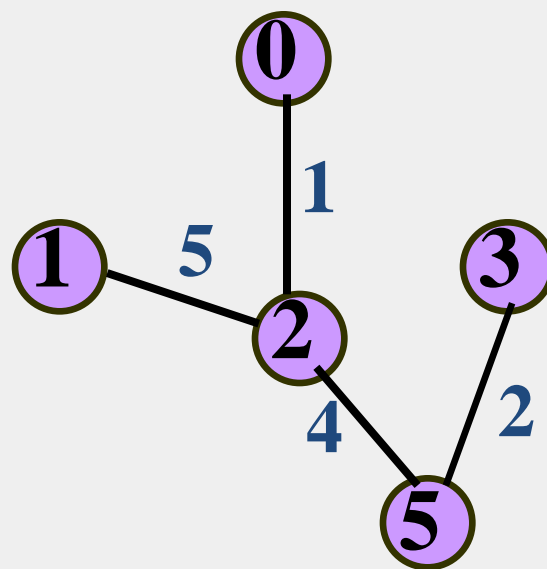
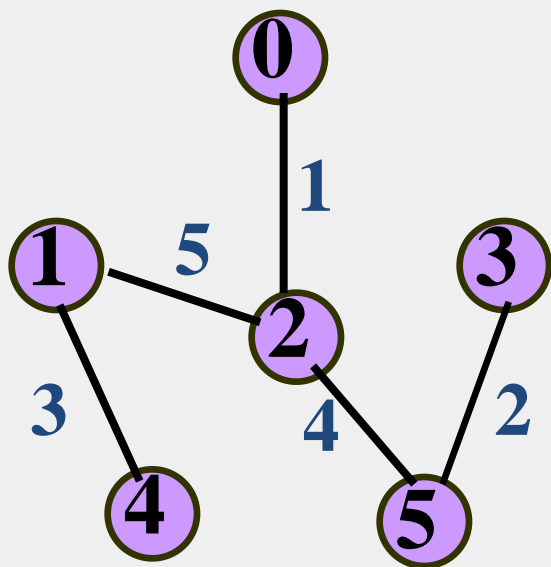
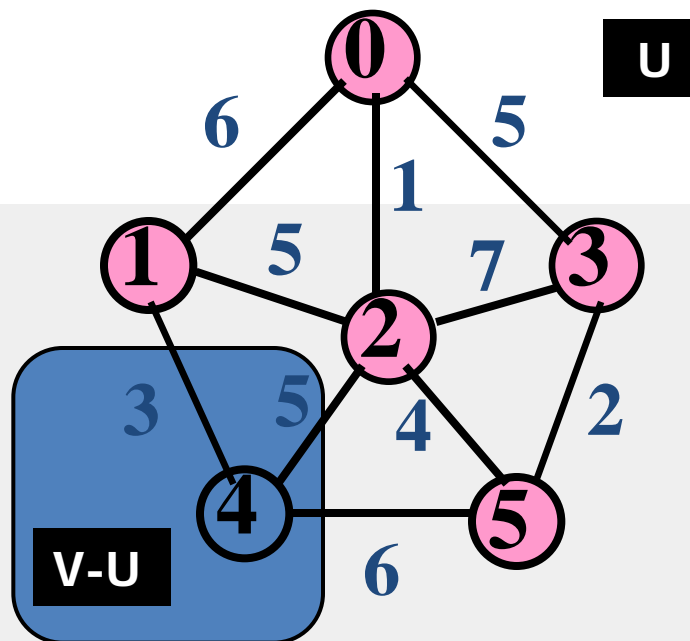




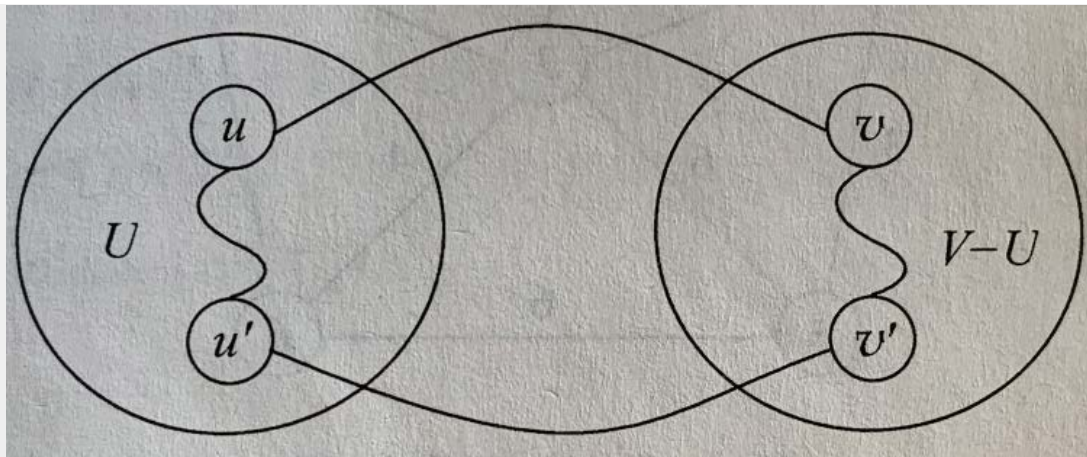








## 2. Prim贪心算法的正确性



### MST性质:

设  $G=(V,E)$  是连通带权图， $U$  是  $V$  的真子集。如果  $(u,v) \in E$ ，且  $u \in U, v \in V-U$ ，且在所有这样的边中， $(u,v)$  的权  $c[u][v]$  最小，那么一定存在  $G$  的一颗最小生成树，它以  $(u,v)$  为其中一条边。

由这一特性，可证明(1)贪心选择性质和 (2)最优子结构性质

# 第七节 单源最短路径



# 单源最短路径



设带权有向图 $G=(V,E)$ ，其中每条边的权是非负实数。**给定 $V$ 中的一个顶点，称为源。**

**计算从源到所有其它各顶点的最短路长度。**这里路的长度是指路上各边权之和。

这个问题通常称为单源最短路径问题。

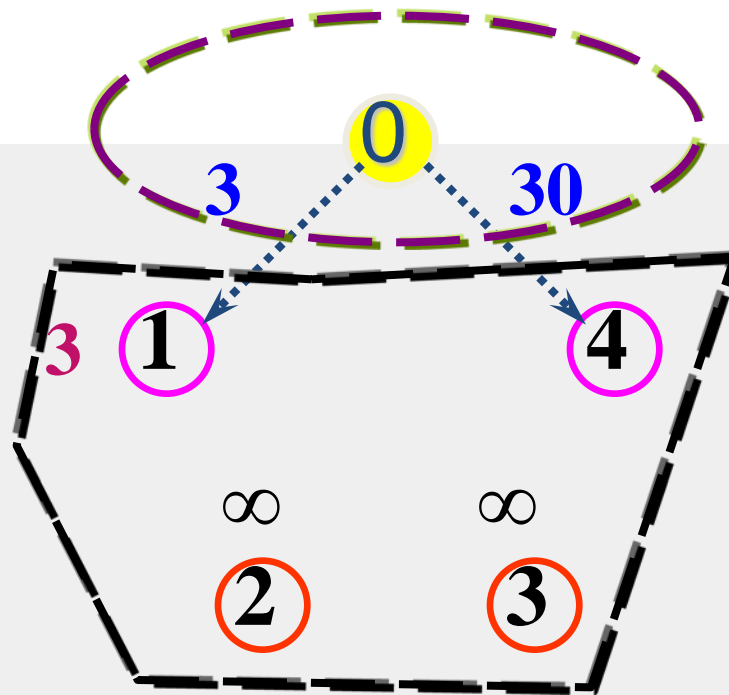
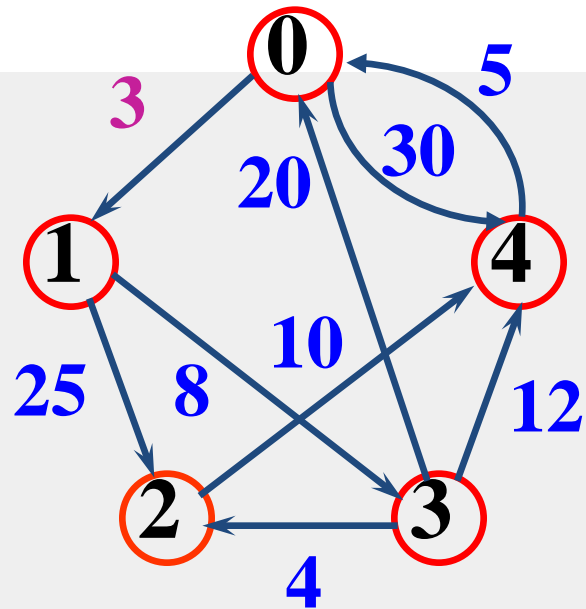
## 1. 算法基本思想

Dijkstra算法是解单源最短路径问题的贪心算法。

# 1. 算法基本思想和步骤

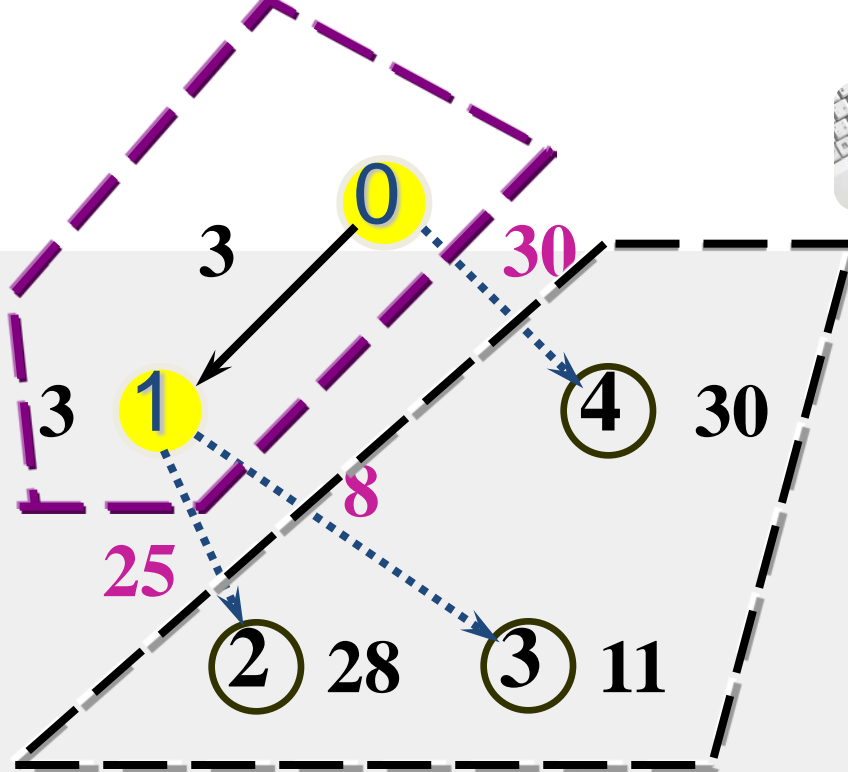
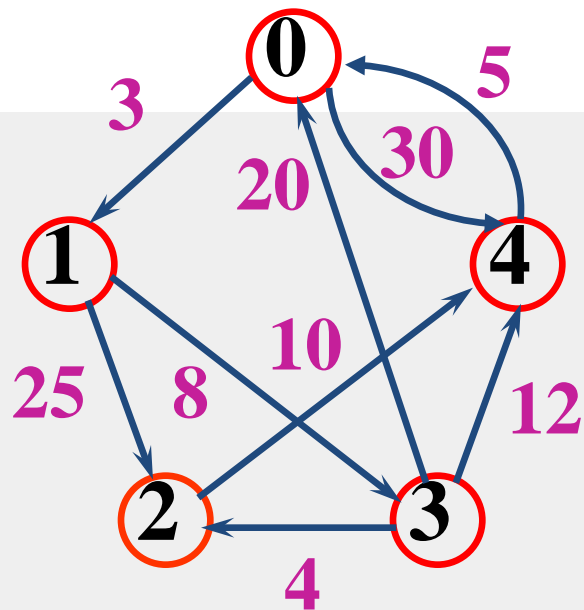


- ◆ 初使时令  $S = \{V_0\}$ ,  $T = \{\text{其余顶点}\}$ ,  $T$ 中顶点对应的距离值:
  - 若存在 $\langle V_0, V_i \rangle$ , 为 $\langle V_0, V_i \rangle$ 弧上的权值
  - 若不存在 $\langle V_0, V_i \rangle$ , 为 $\infty$
- ◆ 从 $T$ 中选取一个其距离值为最小的顶点 $W$ , 加入 $S$
- ◆ 对 $T$ 中顶点的距离值进行修改: 若加进 $W$ 作中间顶点, 从 $V_0$ 到 $V_i$ 的距离值比不加 $W$ 的路径要短, 则修改此距离值
- ◆ 重复上述步骤, 直到 $S$ 中包含所有顶点, 即 $S = V$ 为止



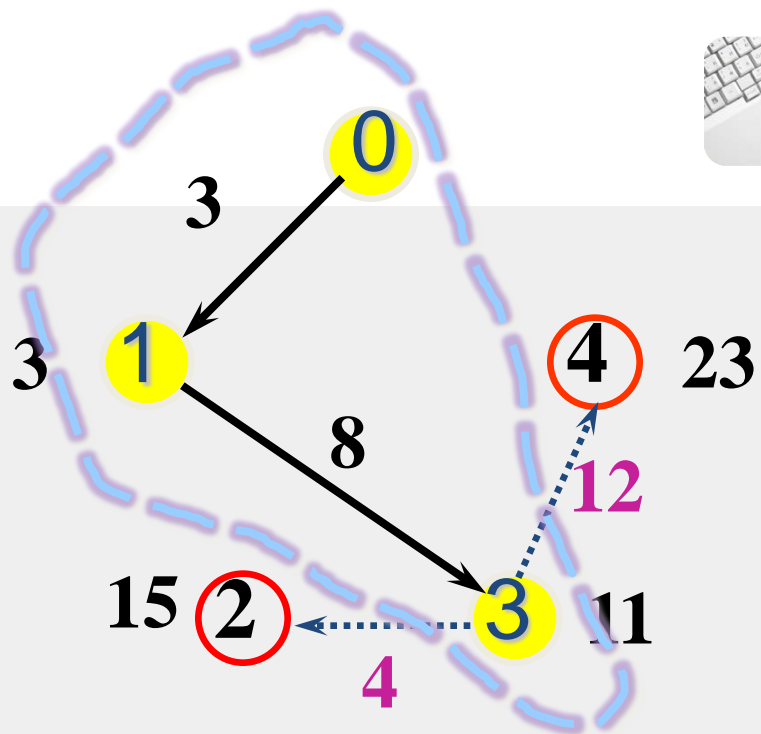
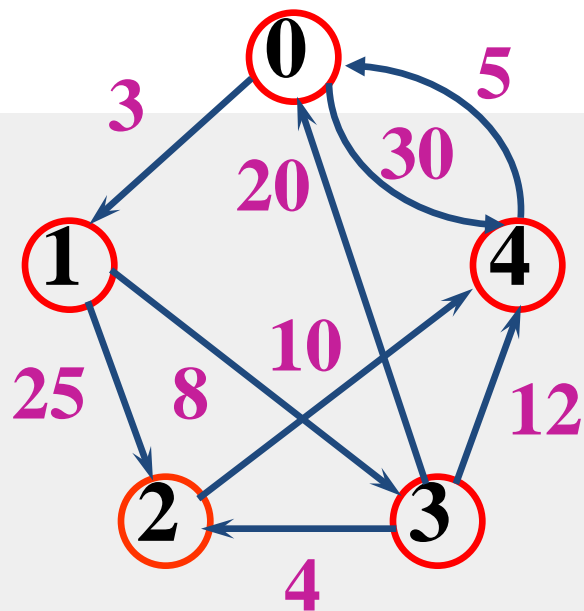
## 最小最短特殊路径

Dijkstra算法按**递增次序**依次得到各顶点的**最小路径长度**



## 次小最短特殊路径

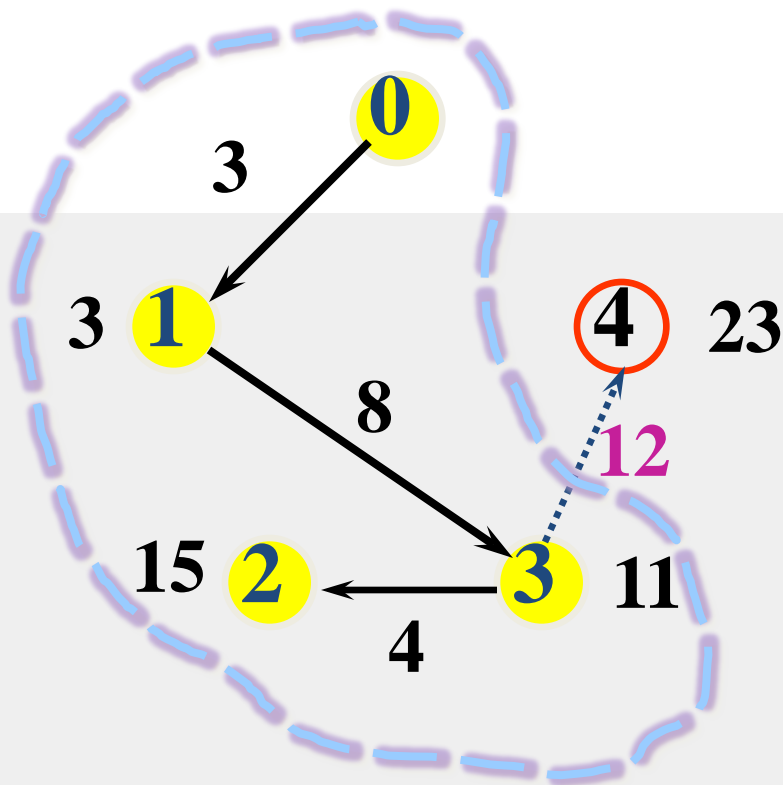
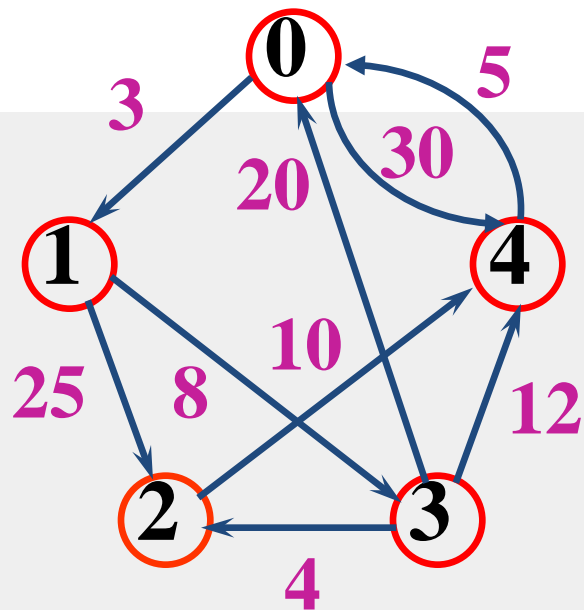
Dijkstra算法按**递增次序**依次得到各顶点的**最小路径长度**



## 第三小最短特殊路径

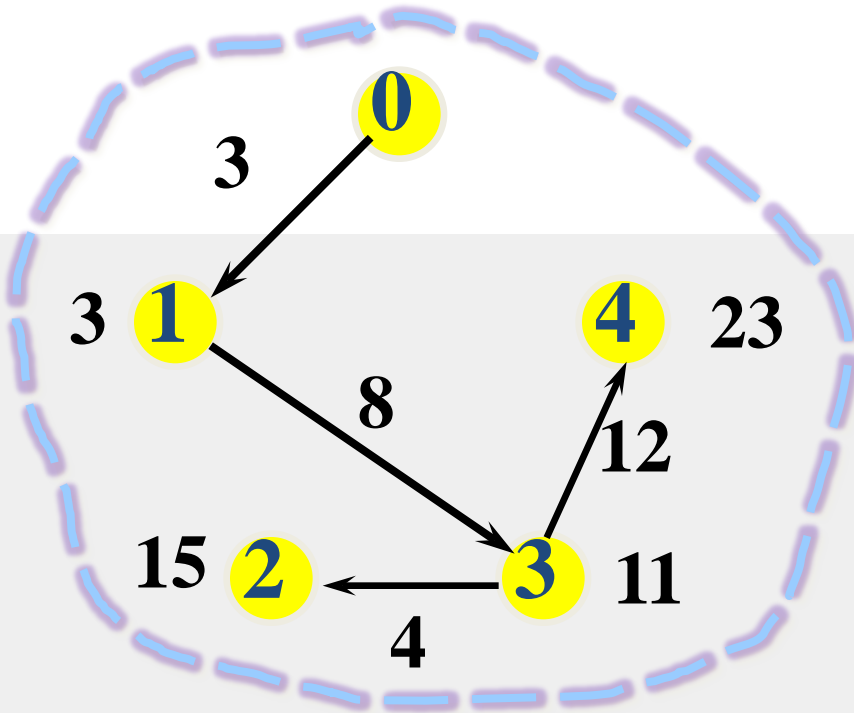
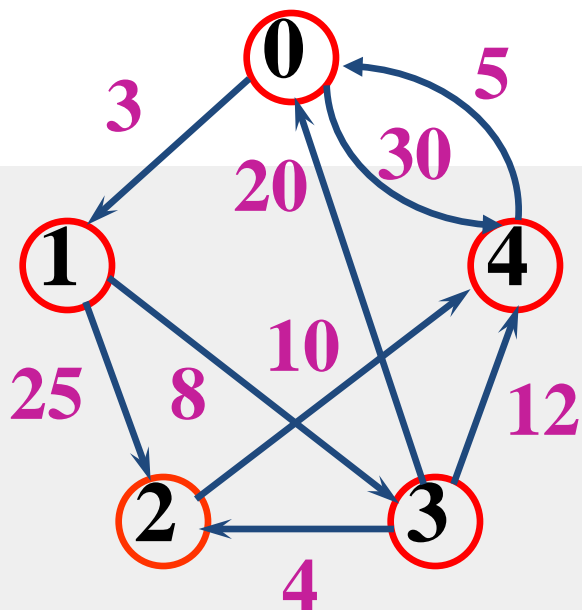
Dijkstra算法按**递增次序**依次得到各顶点的**最小路径长度**





## 第四小最短特殊路径

Dijkstra算法按**递增次序**依次得到各顶点的**最小路径长度**



## 依次得到所有最短路径

Dijkstra算法按**递增次序**依次得到各顶点的**最小路径长度**

## ■ Dijkstra算法描述



初始时 ( $S$ 为初始顶点),  $D_s=0$ 且  $\forall i \neq S$ , 有  $D_i = +\infty$ 。

①在未访问顶点中选择 $D_v$ 最小的顶点 $v$ , 访问 $v$ , 令  $S[v]=1$ 。

②依次考察 $v$ 的邻接顶点 $w$ , 若

$$D_v + \text{weight}(\langle v, w \rangle) < D_w,$$

则改变 $D_w$ 的值, 使  $D_w = D_v + \text{weight}(\langle v, w \rangle)$ 。

③重复① ②, 直至所有顶点被访问。

## ■ 说明

- 引入一个辅助数组  $dist$ 。它的每一个分量  $dist[i]$  表示当前找到的从源点  $s$  到顶点  $i$  的最短路径的长度。初始状态:  
 $dist[s]=0$ , 对其它节点  $i$  有  $dist[i]=+\infty$ 。
- 引入  $path[i]$  记录  $s$  到  $i$  最短路径中  $i$  的前驱节点编号

## 2. Dijkstra贪心算法的正确性



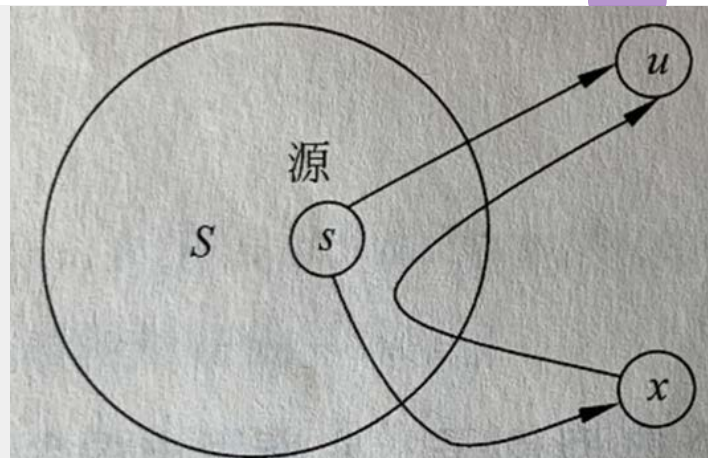
### (1) 贪心选择性质

贪心策略是从  $V-S$  中选择具有最短特殊路径的顶点  $u$ ，从而确定从源  $s$  到  $u$  的最短路径长度  $\text{dist}[u]$ ，为何能导致最优解呢？

假设存在一条从  $s$  到  $u$  且长度小于  $\text{dist}[u]$  的路径，该路径初次走出  $S$  到达  $x$ ，然后徘徊  $S$  内外若干次到达  $u$ 。则有：

$$\text{dist}[x] \leq d(s, x) \leq d(s, x) + d(x, u) = d(s, u) < \text{dist}[u]$$

矛盾，因为  $\text{dist}[u]$  是当前最短的特殊路径，否则节点  $x$  就应该先加入，而不是在  $S$  外面。



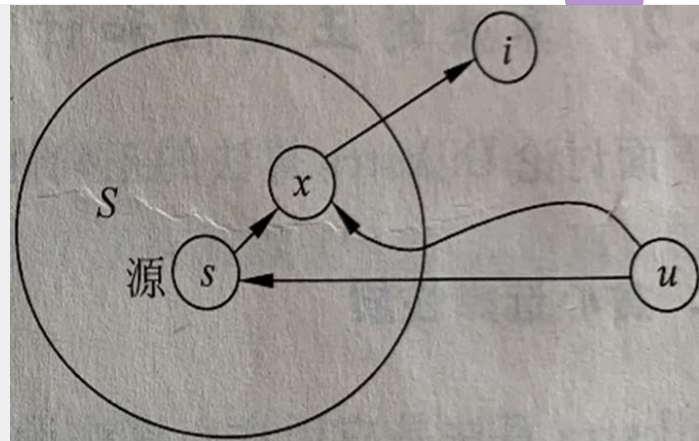
## 2. Dijkstra贪心算法的正确性



### (2) 最优子结构性质

顶点 $u$ 的**特殊路径**定义为**从源 $s$ 仅通过 $S$ 中顶点进行中转**的路径。

需要证明算法中确定的 $\text{dist}[u]$ 确实是当前从**源 $s$** 到**顶点 $u$** 的**最短特殊路径**长度。



考察算法在添加 $u$ 到 $S$ 中后 $\text{dist}[u]$ 的值所起的变化。添加 $u$ 到 $S$ 之后，对 $S$ 外的顶点 $i$ ，可能会出现一条经过 $u$ 中转的特殊路径 $\text{dist}[u] + a[u][i]$ ，如果比原来的 $\text{dist}[i]$ 小则更新。

需要确定顶点 $i$ 除了 $\text{dist}[u] + a[u][i]$ 是否还存在其它的特殊路径更小，即是经过 $u$ 之后不是直接通过边 $a[u][i]$ 到达 $i$ ，而是又回到 $S$ 中某个顶点 $x$ 中转才到达 $i$ 。

即用 $\text{dist}[u] + d(u, x) + a[x][i]$  更新。

由于 $x$ 比 $u$ 先加入 $S$ ，故 $\text{dist}[x] < \text{dist}[u]$ ，则 $\text{dist}[x] + a[x][i] < \text{dist}[u] + d(u, x) + a[x][i]$ ，

**矛盾**，不可能存在。

# 第七节 多机调度问题



# 多机调度问题



多机调度问题要求给出一种作业调度方案，使所给的 $n$ 个作业在尽可能短的时间内由 $m$ 台机器加工处理完成。

约定，每个作业均可在任何一台机器上加工处理，但未完工前不允许中断处理。作业不能拆分成更小的子作业。

这个问题是NP完全问题，到目前为止还没有有效的解法。对于这一类问题,用贪心选择策略有时可以设计出较好的近似算法。

# 多机调度问题



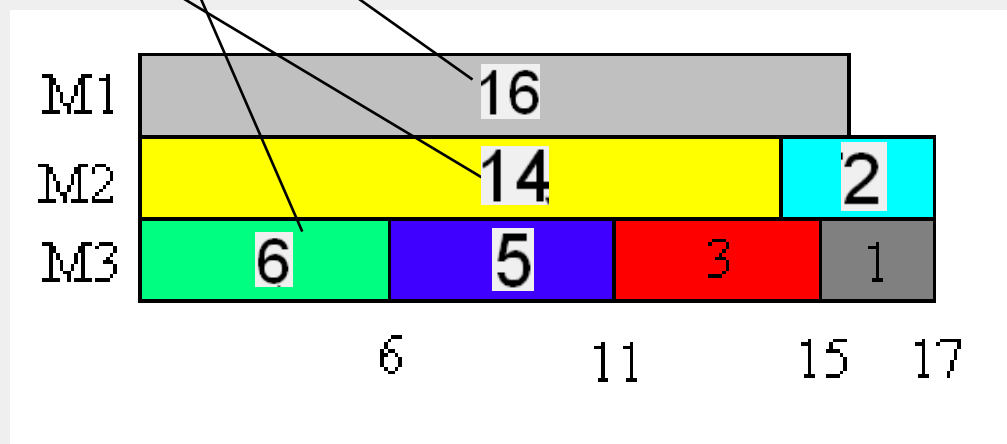
- 采用**最长处理时间作业优先的贪心选择策略**可以设计出**解多机调度问题的较好的近似算法**。
- 按此策略，当  $n \leq m$  时，只要将机器  $i$  的  $[0, t_i]$  时间区间分配给作业  $i$  即可，算法只需要  $O(1)$  时间。
- 当  $n > m$  时，**首先将  $n$  个作业依其所需的处理时间从大到小排序**。然后依此顺序将作业分配给空闲的处理机。算法所需的计算时间为  $O(n \log n)$ 。



# 多机调度问题



➤ 例如，设7个独立作业{1,2,3,4,5,6,7}由3台机器 $M_1$ ， $M_2$ 和 $M_3$ 加工处理。各作业所需的处理时间分别为{2,14, 4, 16, 6, 5, 3}。按算法greedy产生的作业调度如下图所示，所需的加工时间为17。



## 第八节 贪心算法与动态规划



# 1.贪心选择性质



所谓**贪心选择性质**是指所求问题的整体最优解可以通过一系列**局部最优的选择**，即**贪心选择来达到**。这是贪心算法可行的**第一个基本要素**，也是贪心算法与动态规划算法的**主要区别**。

**动态规划算法通常以自底向上的方式解各子问题**，而**贪心算法则通常以自顶向下的方式进行**，以迭代的方式作出相继的贪心选择，每作一次贪心选择就将所求问题**简化为规模更小的子问题**。

## 2.最优子结构性质



对于一个具体问题，要确定它是否具有贪心选择性，必须证明每一步所作的贪心选择最终导致问题的**整体最优解**。

**当一个问题的最优解包含其子问题的最优解时，称此问题具有最优子结构性质。**

问题的最优子结构性质是该问题可用动态规划算法或贪心算法求解的关键特征。

# 贪心算法与动态规划算法的差异



贪心算法和动态规划算法都要求问题具有最优子结构性质，这是2类算法的一个共同点。

对于具有最优子结构的问题应该选用贪心算法还是动态规划算法求解？是否能用动态规划算法求解的问题也能用贪心算法求解？

通过研究经典的背包和0-1背包组合优化问题，并以此说明贪心算法与动态规划算法的主要差别。

# 0-1背包问题



给定 $n$ 种物品和一个背包。物品 $i$ 的重量是 $W_i$ ，其价值为 $V_i$ ，背包的容量为 $C$ 。应如何选择装入背包的物品，使得装入背包中物品的总价值最大？

**限制：**在选择装入背包的物品时，对每种物品 $i$ 只有2种选择，即装入背包或不装入背包。**不能将物品 $i$ 装入背包多次，也不能只装入物品 $i$ 的一部分。**

# 背包问题



背包问题与0-1背包问题类似，所不同的是在选择物品  $i$  装入背包时，**可以选择物品  $i$  的一部分**，而不一定要全部装入背包， $1 \leq i \leq n$ 。

这2类问题都具有**最优子结构**性质，极为相似，但**背包问题可以用贪心算法求解，而0-1背包问题却不能用贪心算法求解。**

# 用贪心算法解背包问题的基本步骤



1. 计算每种物品**单位重量的价值**  $V_i / W_i$ ;
2. 依贪心选择策略，将尽可能多的**单位重量价值**最高的物品装入背包；
3. 若将这种物品全部装入背包后，背包内的物品总重量未超过C，则选择**单位重量价值次高**的物品并尽可能多地装入背包；
4. 依此策略一直地进行下去，直到背包装满为止。



# 用贪心算法解背包问题的算法



```
void Knapsack(int n, float M, float v[], float w[], float x[]) {  
    Sort(n, v, w); // 将各种物品依其单位重量的价值从大到小排序  
    int i;  
    for (i=1; i<=n; i++) x[i]=0;  
    float c=M;  
    for (i=1; i<=n; i++) {  
        if (w[i]>c) break;  
        x[i]=1;  
        c-=w[i];  
    }  
    if (i<=n) x[i]=c/w[i];  
    //如果有剩余空间, 就将i物品的部分装入。  
}
```

算法knapsack的主要计算时间在于**将各种物品依其单位重量的价值从大到小排序**。因此, 算法的计算时间上界为 $O(n \log n)$ 。  
为了证明算法的正确性, 还必须证明背包问题具有贪心选择性质。

# 背包问题



对于0-1背包问题，贪心选择之所以不能得到最优解是因为在这种情况下，**它无法保证最终能将背包装满**，部分闲置的背包空间使每公斤背包空间的价值降低了。

事实上，**在考虑0-1背包问题时，应比较选择该物品和不选择该物品所导致的最终方案，然后再作出最好选择**。由此就导出许多**互相重叠的子问题**。这正是该问题可用动态规划算法求解的另一重要特征。

实际上也是如此，动态规划算法的确可以有效地解0-1背包问题。