

5 贪心策略

Greedy Strategy

引例

- ▶ 假设有4种硬币，面值分别为2角5分、1角、5分和1分。现在要求以最少的硬币个数找给顾客6角3分。

基于贪心策略：面值越大，需要的个数越少。通常的做法是：先拿两个2角5分+1个1角+3个1分。

贪心策略总是做出在**当前**看来**最好的选择**，并且不从整体上考虑最优，所做出的每一步选择只是**局部意义上**最优选择(因而效率往往较高)，逐步扩大解的规模。当然，我们希望贪心策略得到的最终结果也是整体最优的(上面的解法得到的恰好是最优解)，但不能保证必定得到最优解。



例如：面值有1角1分、5分和1分，要找给顾客1角5分。如果还是使用上述贪心策略，得到的解是：1个1角1分+ 4个1分。而实际上最优解是3个5分。

虽然贪心策略不能保证对所有问题都能得到最优解，但是对很多问题(包括很多著名的问题)都能产生整体最优解。例如，我们今天要介绍的图的单源最短路径问题、最小生成树问题。在有些情况下，虽然不能得到整体最优解，但是结果却是最优解的很好近似。

使用贪心策略的难点在于：证明所设计的算法确实能够正确解决所求解的问题。(正确性证明)



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，这是两类算法的一个共同点。但是，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？下面研究2个经典的组合优化问题，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
 - ▶ 背包问题

0-1背包问题与背包问题

▶ 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？
- ▶ 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，也不能只装入部分的物品 i 。

▶ 背包问题:

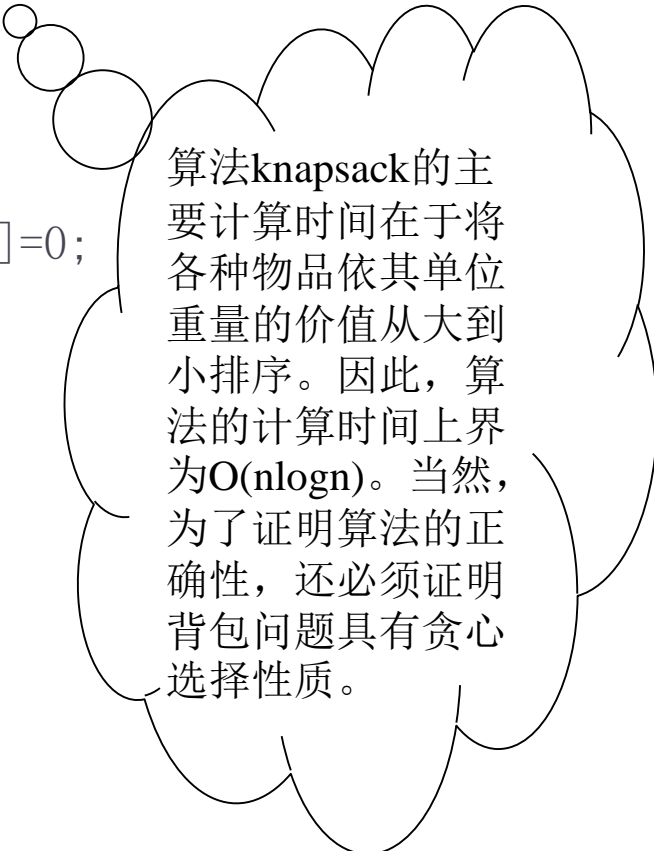
- ▶ 与0-1背包问题类似，所不同的是在选择物品 i 装入背包时，可以选择物品 i 的一部分，而不一定要全部装入背包， $1 \leq i \leq n$ 。
- ▶ 这两类问题都具有最优子结构性质，极为相似，但背包问题可以用贪心算法求解得到最优解，而0-1背包问题却用贪心算法求解未必能得到最优解。



用贪心算法解背包问题的基本步骤

- ▶ 首先计算每种物品单位重量的价值 V_i/W_i , 然后, 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。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。依此策略一直地进行下去, 直到背包装满为止。
- ▶ 具体算法可描述如下:

```
void Knapsack(int n, float M, float v[], float w[], float x[])  
{  
    Sort(n, v, w);  
    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];  
}
```



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 $O(n\log n)$ 。当然, 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。

单源最短路径问题

- ▶ $G = (V, E)$ 是一个有向图，每条边上有一个非负整数表示长度值，其中有一个节点称为源节点。所谓的单源最短路径问题就是：求解该源节点到所有其它节点的最短路径值。
- ▶ 不失一般性，假设 $V = \{1, 2, 3, \dots, n\}$ 并且源节点 $s = 1$ 。那么该问题可以使用Dijkstra算法来求解，该算法是一种贪心算法，并且能求得最优解。

埃德斯加.狄克斯特拉(Edsger Dijkstra) 1930-2002

- ▶ 1956年，成功设计并实现了在有障碍物的两个地点之间找出一条**最短路径**的高效算法，这个算法被命名为“狄克斯特拉算法”，解决了机器人学中的一个十分关键的问题，即运动路径规划问题，至今仍被广泛应用。
- ▶ 1968年3月，Communications of ACM登出了狄克斯特拉的一封信，在信中他根据自己编程的实际经验和大量观察，得出如下结论：一个程序的易读性和易理解性同其中所包含的无条件转移控制的个数成反比关系，也就是说，转向语句的个数愈多，程序就愈难读、难懂。因此他认为“goto语句是有害的”，并从而启发了结构化程序设计的思想。
- ▶ 1972年的图灵奖授予荷兰的计算机科学家埃德斯加.狄克斯特拉。
- ▶ 1983年，ACM为纪念Communications of ACM 创刊25周年，评选出从1958-1982的四分之一个世纪中在该杂志上发表的25篇有里程碑意义的论文，每年一篇，狄克斯特拉一人就有两篇入选，是仅有的这样的两位学者之一。
- ▶ 在程序设计技术、算法和算法理论、编译器、操作系统等诸多方面，狄克斯特拉都有许多创造，作出了杰出贡献。

- ▶ 开始时，我们将所有的节点划分为两个集合 $X = \{1\}$, $Y = \{2, 3, 4, \dots, n\}$ 。所有已经计算好的节点存放在 X 中， Y 中表示还没有计算好的。 Y 中的每个节点 y 有一个对应的量 $\lambda[y]$ ，该值是从源节点到 y （并且只经由 X 中的节点）的最短路径值。

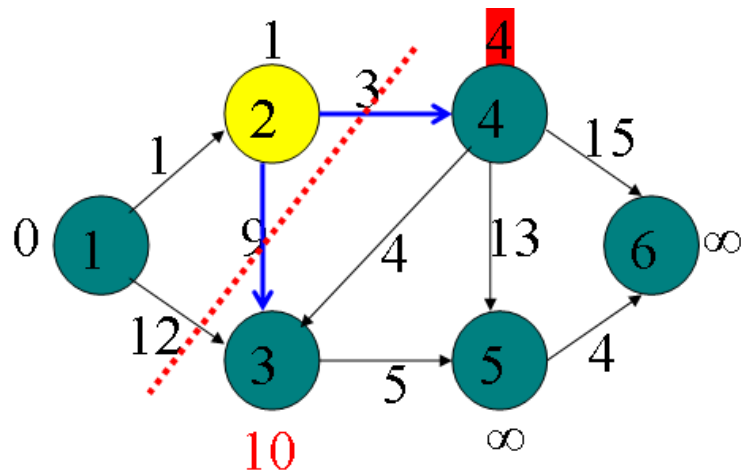
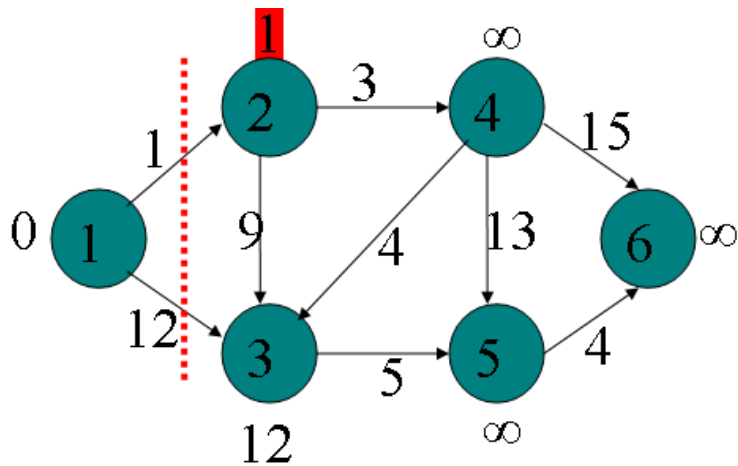
▶ Dijkstra算法

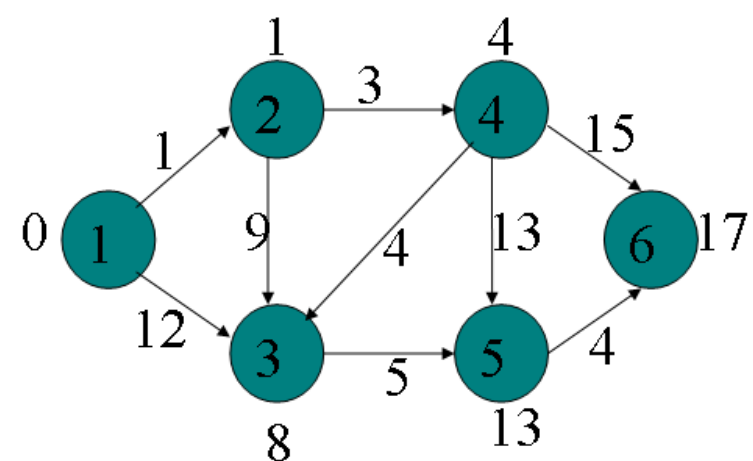
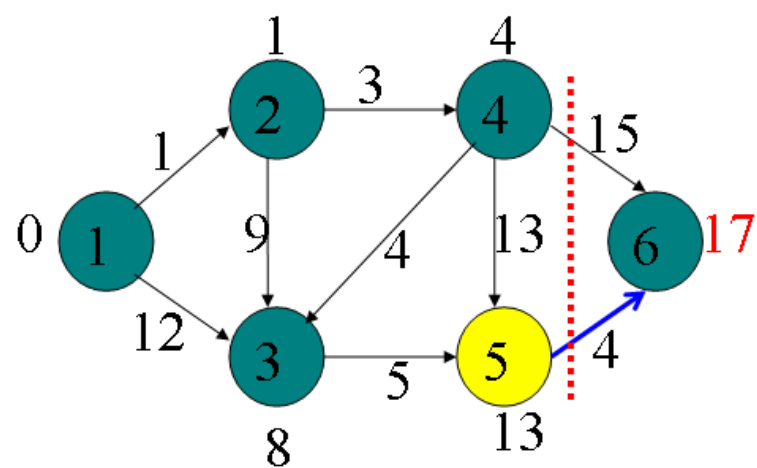
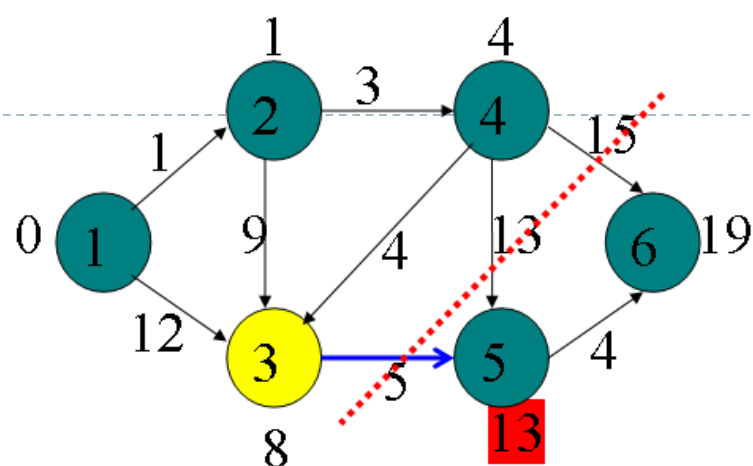
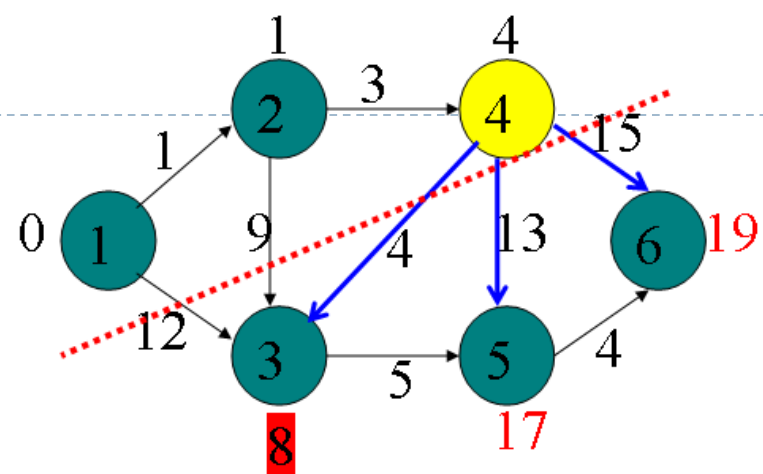
- ▶ 假设 $V = \{1, 2, 3, \dots, n\}$ 并且 $s = 1$ 。
- ▶ 选择一个 $\lambda[y]$ 最小顶点 $y \in Y$ ，并将其移动到 X 中。
- ▶ 若 y 被从 Y 移动到 X 中， Y 中每个和 y 相邻的顶点 w 的 $\lambda[w]$ 都要更新(表示经由 y 到 w 的一条更短的路径被发现)。

算法框架：

1. $X \leftarrow \{1\}; Y \leftarrow V - \{1\}$
2. 对于任意一个 $v \in V$ ，如果存在一条边从 1 到 v ，那么 $\lambda[v] = \text{该边的长度}$ 。否则 $\lambda[v] = \infty$ ；并设定 $\lambda[0] = 0$ 。
3. while $Y \neq \Phi$
 - 对于 $y \in Y$ ，找到最小 $\lambda[y]$
 - 将 y 从 Y 移动到 X 中
 - 更新 Y 中和 y 相邻顶点的 λ 值end while

if $w \in Y$ and $(y, w) \in E$
and $\lambda[y] + \text{length}[y, w] < \lambda[w]$
then $\lambda[w] \leftarrow \lambda[y] + \text{length}[y, w]$





Dijkstra算法

```
1.  $X = \{1\}; Y \leftarrow V - \{1\}; \lambda[1] \leftarrow 0 \quad // \Theta(n)$ 
2. for  $y \leftarrow 2$  to  $n$ 
3.   if  $y$  相邻于  $1$  then  $\lambda[y] \leftarrow \text{length}[1, y] \quad // \Theta(n)$ 
4.   else  $\lambda[y] \leftarrow \infty \quad // O(n)$ 
5.   end if
6. end for
7. for  $j \leftarrow 1$  to  $n-1$ 
8.   令  $y \in Y$ , 使得  $\lambda[y]$  为最小的  $// \sum_{i=1}^{n-1} (n-i) = \Theta(n^2)$ 
9.    $X \leftarrow X \cup \{y\} \quad // \Theta(n)$ 
10.   $Y \leftarrow Y - \{y\} \quad // \Theta(n)$ 
11.   for 每条边  $(y, w) \quad // \text{每条边恰好检查一次, } \Theta(m), m = |E|$ 
12.     if  $w \in Y$  and  $\lambda[y] + \text{length}[y, w] < \lambda[w]$  then
13.        $\lambda[w] \leftarrow \lambda[y] + \text{length}[y, w] \quad // \Theta(m)$ 
14.     end if
15.   end for
16. end for
```

算法的时间复杂度:

$$T(n) = \Theta(n) + O(n) + \Theta(n^2) + \Theta(n) + \Theta(n) + \Theta(m) + \Theta(m) = \Theta(m + n^2)$$



算法结束后，每个节点的 λ 值就是从源节点到该节点的最短路径值。

回忆：Y中的每个节点y有一个对应的量 $\lambda[y]$ ，该值是从源节点到y (并且只经由X中的节点) 的最短路径值。

正确性证明：对于任意一个节点 $y \in V$, 使用 $\delta[y]$ 表示源节点到节点y的真正最短路径值，下面我们证明, Dijkstra算法结束后有 $\lambda[y] = \delta[y]$ 。

证明:(数学归纳法)

- 1) 显然 $\lambda[1] = \delta[1] = 0$ 。
- 2) 假设当前将y从Y中移动到X中，并且在y之前移动到X中的任何一个顶点c，都有 $\lambda[c] = \delta[c]$ 。
- 3) 下面证明 $\lambda[y] = \delta[y]$ 。

我们知道：必定存在一条从源节点1到节点y的**真正最短路径**，该路径长度值用 $\delta[y]$ 表示，并且这条最短路径总可以用以下节点序列表示：

$1 \rightarrow [] \rightarrow [] \rightarrow \dots \rightarrow [] \rightarrow [] \rightarrow [] \rightarrow y$



沿此方向(逆向)找到第一个属于X的节点，不妨称之为x。

分析：[]中的节点，不属于X，就属于Y，必居其一。

$1 \rightarrow [] \rightarrow [] \rightarrow \dots \rightarrow [] \rightarrow [x] \rightarrow y$ (A)

$1 \rightarrow [] \rightarrow [] \rightarrow \dots \rightarrow [x] \rightarrow [w] \dots \rightarrow y$ (B)

(A)

(B)

1个或1个以上属于Y的节点。



$$1 \rightarrow [\] \rightarrow [\] \rightarrow \dots \rightarrow [\] \rightarrow [x] \rightarrow y \quad (A)$$

$$\begin{aligned} \lambda[y] &\leq \lambda[x] + \text{length}[x, y] && // \text{算法要求} \\ &= \delta[x] + \text{length}[x, y] && // \text{归纳假设} \\ &= \delta[y] && // (A) \text{是最短路径} \end{aligned}$$

$$1 \rightarrow [\] \rightarrow [\] \rightarrow \dots \rightarrow [x] \rightarrow [w] \dots \rightarrow y \quad (B)$$

$$\begin{aligned} \lambda[y] &\leq \lambda[w] && // \text{由于} y \text{在} w \text{之前离开} Y \\ &\leq \lambda[x] + \text{length}[x, w] && // \text{算法要求, 原因同}(A) \\ &= \delta[x] + \text{length}[x, w] && // \text{归纳假设} \\ &= \delta[w] && // \text{因为}(B) \text{是最短路径} \\ &\leq \delta[y] && // \text{因为}(B) \text{是最短路径} \end{aligned}$$

$$\left. \begin{array}{l} \\ \\ \\ \end{array} \right\} \Rightarrow \lambda[y] = \delta[y]$$



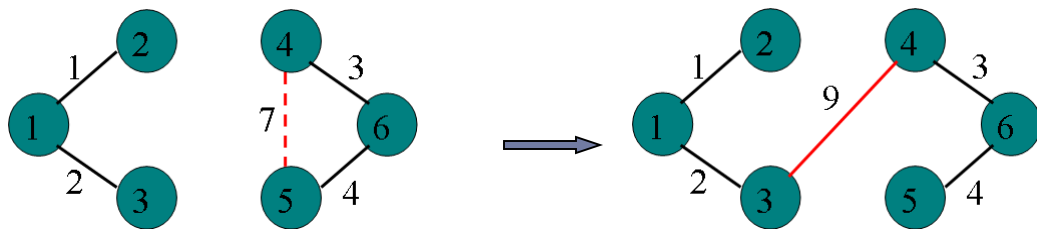
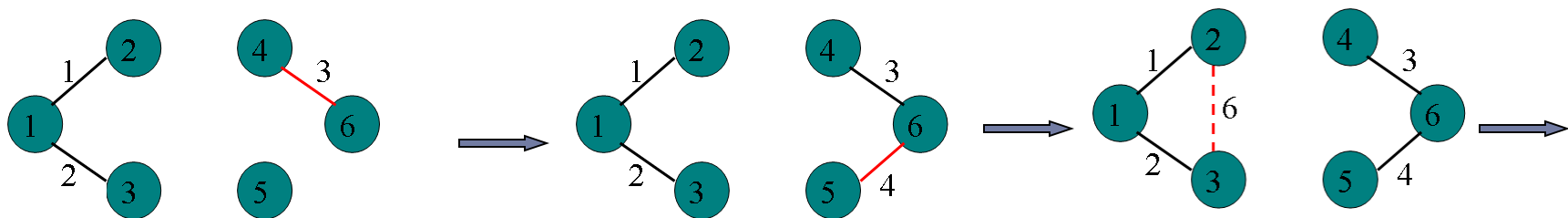
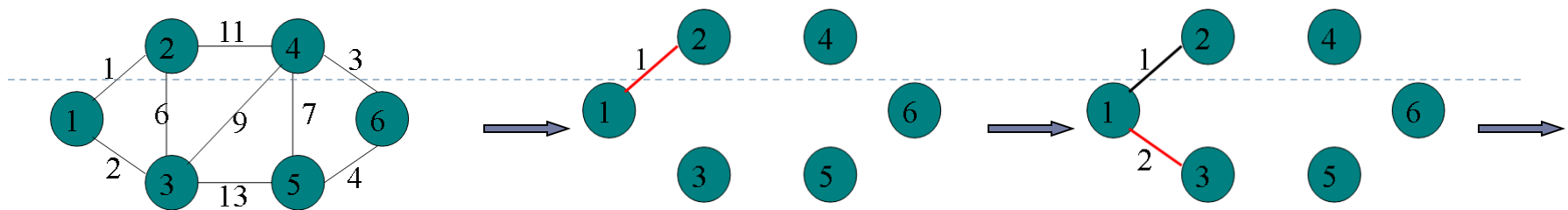
最小生成树

- ▶ 设 $G=(V,E)$ 是连通无向带权图, (V,T) 是 G 的一个子图, 并且 T 是一颗树, 那么称 (V,T) 是 G 的生成树。如果 T 的权之和是所有生成树中最小的, 那么则称之为最小生成树。
- ▶ 网络的最小生成树在实际中有广泛应用。例如, 在设计通信网络时, 用图的顶点表示城市, 用边 (v,w) 的权 $c[v][w]$ 表示建立城市 v 和城市 w 之间的通信线路所需的费用, 则最小生成树就给出了建立通信网络的最经济的方案。
- ▶ 假定 G 是连通的, 如果 G 是非连通的, 那么, 可以对 G 的每个子图应用求解最小生成树的算法。

Kruskal算法

1. 对 G 的边 E 按权重以非降序排列。
2. 初始时输出树 $T=\{\}$ ；依次取排序表中的每条边，若加入 T 不形成回路，则加入 T ；否则将其丢弃；
3. 不断重复步骤2，直到树 T 包含 $n-1$ 条边，算法结束。





Kruskal算法

1. 对E中的边按权重以非降序排列 //O(mlogm)
2. for 每个顶点 $v \in V$ // $\Theta(n)$
3. MAKESET($\{v\}$)
4. end for
5. $T = \{\}$
6. while $|T| < n-1$
7. 令(x,y)为E中的下一条边 //取每条边试探, 最多m次
8. if FIND(x) \neq FIND(y) then //最多2m次查找, 原因如上
9. 将(x,y)加入T //刚好n-1次, $\Theta(n)$
10. UNION(x,y) //如上, n-1次
11. end if
12. end while

$$O(m \log m) + \Theta(n) + O(m) + O(m) + \Theta(n) + \Theta(n) = O(m \log m + n)$$



正确性证明(课后阅读)

证明：我们只要证明，使用Kruskal 算法过程中，每次循环所得到的 T (从空集增至最小生成树)总是图 G 的最小生成树的子集即可。证明使用归纳法 + 反证法。

- (1) G 总是具有一个最小生成树，不妨记为 T^*
- (2) 当前要加入的边为 $e = (x, y)$
- (3) 包含 x 的那棵子树的所有顶点用 X 表示，有 $x \in X, y \in V - X$
- (4) 假设在 $e = (x, y)$ 加入之前得到的 T 均满足 $T \subset T^*$



令 $T' = T \cup \{e\}$ ，下面我们要证明 T' 也是图 G 的最小生成树的子集。

依据归纳假设，有 $T \subset T^*$ 。

i) 如果 $e \in T^*$ ，显然有 $T' \subset T^*$

ii) 如果 $e \notin T^*$ ，我们知道 T^* 中必定包含这样一条边 $e' = (w, z)$ ，且 $w \in X, z \in V - X$ ， $\text{cost}(e') \geq \text{cost}(e)$ 。（否则若 $\text{cost}(e') < \text{cost}(e)$ ，则 e' 将被选择加入(而不是 e)）

下面我们来研究：如果 $e' \in T^*$ 会带来什么结果。

定义 $T^{**} = T^* - \{e'\} \cup \{e\}$ ，那么 T^{**} 也是图的一个生成树，并且 $\text{cost}(T^{**}) = \text{cost}(T^*) - \text{cost}(e') + \text{cost}(e) < \text{cost}(T^*)$ 。也就是说， T^* 不是最小生成树，矛盾。所以 e 必定是属于 T^* 的。也就必定有 $T' \subset T^*$ ，证毕。

Prim算法

- 设 $G=(V,E)$ 是连通无向带权图， $V=\{1,2,\dots,n\}$ 。构造 G 的最小生成树的Prim算法的基本思想是：
 - 首先置 $X=\{1\}$,
 - 然后，只要 X 是 V 的真子集，就作如下的贪心选择：
 - 选取权重最小的边 (x,y) , 其中 $x\in X$, $y\in Y$,
 - 将边 (x,y) 加入当前的最小生成树,
 - 将顶点 y 从 Y 移到 X 中,
 - 这个过程一直进行到 $X=V$ 时为止。
- 在这个过程中选取到的所有边恰好构成 G 的一棵最小生成树。



算法设计要点:

寻找 (x,y) , 使得 $x \in X, y \in Y$, 且 $\text{cost}(x,y)$ 最小.

候选边 (x,y) 对应的顶点 $y \in Y$, 可考察 y 在 X 中的邻居.

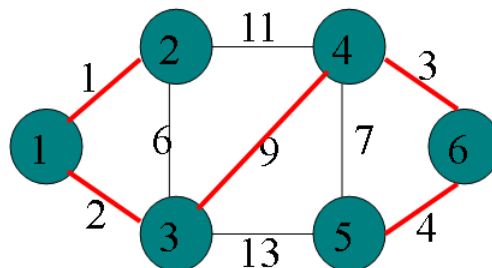
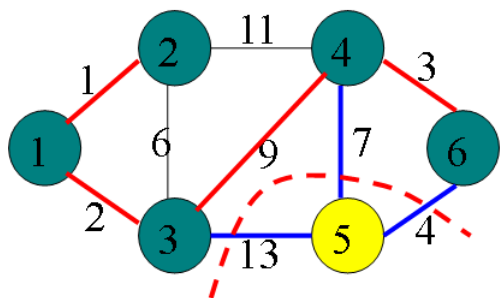
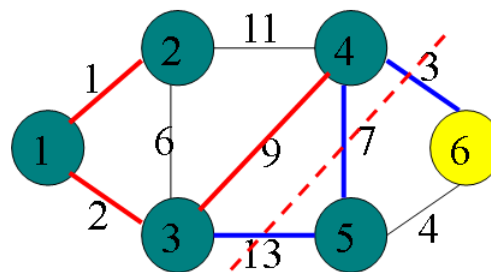
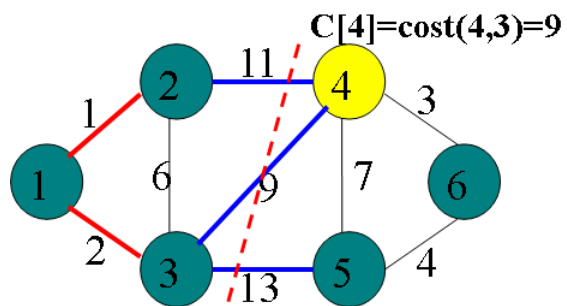
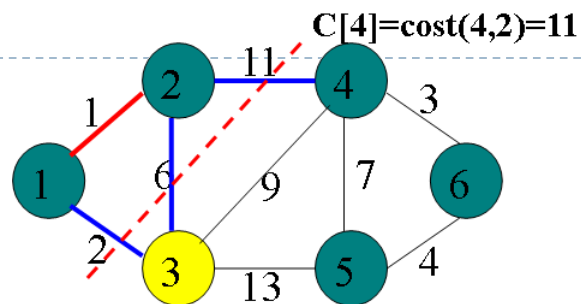
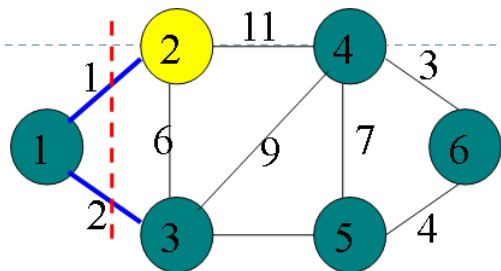
邻居就是 X 中离 y 最近的那个点 x^* , 即

$\text{cost}(x^*, y) = \min\{\text{cost}(x, y) \mid x \in X \text{ 且 } (x,y) \text{ 存在}\}$

记 $x^* = N[y]$, 定义 $C[y] = \text{cost}(x^*, y) = \text{cost}(y, N[y])$.

X, Y 可用 n 维 $\{0,1\}$ 向量表示, 例 $X[i]=1$ 表示顶点 $i \in X$.






```

1.  $T \leftarrow \{\}; X \leftarrow \{1\}; Y \leftarrow V - \{1\}$  //  $\Theta(n)$ 
2. for  $y \leftarrow 2$  to  $n$  //  $\Theta(n)$ 
3.   if  $y$  邻接于  $1$  then // 第3~6步,  $O(n)$ 
4.      $N[y] \leftarrow 1$ 
5.      $C[y] \leftarrow \text{cost}[1,y]$ 
6.   else  $C[y] \leftarrow \infty$ 
7.   end if
8. end for
9. for  $j \leftarrow 1$  to  $n-1$  // 寻找MST的 $n-1$ 条边
10.  令  $y \in Y$ , 使得  $C[y]$  最小 // 每次  $\Theta(n)$ , 共  $n-1$  次, 计  $\Theta(n^2)$ 
11.   $T \leftarrow T \cup (y, N[y])$  //  $\Theta(1) \times (n-1) = \Theta(n)$ 
12.   $X \leftarrow X \cup \{y\}$  //  $\Theta(1) \times (n-1) = \Theta(n)$ 
13.   $Y \leftarrow Y - \{y\}$  // ...
14.  for 每个邻接于  $y$  的顶点  $w \in Y$  //  $\Theta(m)$ 
15.    if  $\text{cost}[y,w] < C[w]$  then // 每条边执行1次, 共  $m$  次
16.       $N[w] \leftarrow y$  // 最多  $m$  次,  $\Theta(m)$ 
17.       $C[w] \leftarrow \text{cost}[y,w]$  //  $\Theta(m)$ 
18.    end if
19.  end for
20. end for

```



$$\Theta(n) + \Theta(n^2) + \Theta(m) = \Theta(m + n^2)$$

哈夫曼编码

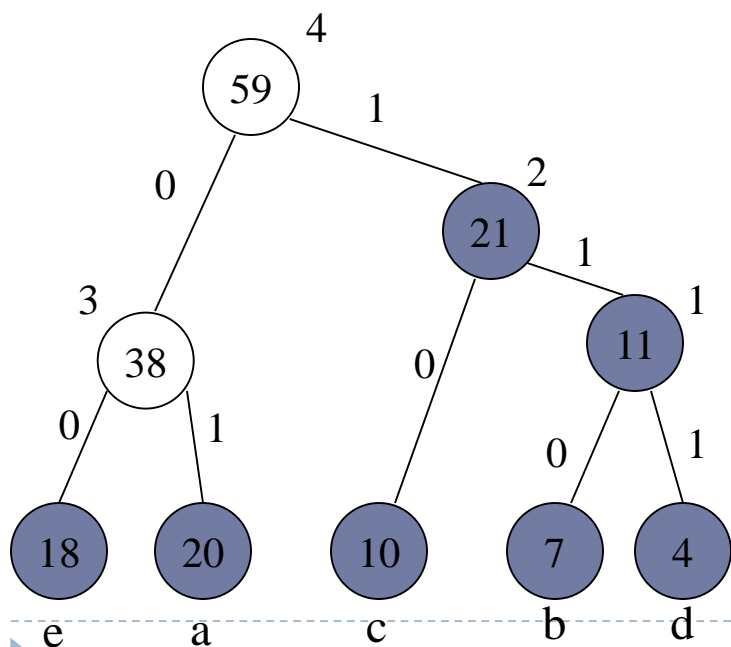
- ▶ **哈夫曼编码**是广泛地用于数据文件压缩的十分有效的编码方法。其压缩率通常在20%~90%之间。哈夫曼编码算法用字符在文件中出现的频率表来建立一个用0,1串表示各字符的最优表示方式。
- ▶ 给出现频率高的字符较短的编码,出现频率较低的字符以较长的编码,可以大大缩短总码长。
- ▶ **前缀码**
 - ▶ 对每一个字符规定一个0,1串作为其代码,并要求任一字符的代码都**不是**其他字符代码的**前缀**。这种编码称为**前缀码**。
 - ▶ 编码的前缀性质可以使译码方法非常简单

- ▶ 哈夫曼提出构造最优前缀码的贪心算法，由此产生的编码方案称为**哈夫曼编码**。
- ▶ 哈夫曼算法以自底向上的方式构造表示最优前缀码的二叉树T。
- ▶ 算法以 $|C|$ 个叶结点开始，执行 $|C|-1$ 次的“合并”运算后产生最终所要求的树T。
- ▶ 在书上给出的算法huffmanTree中，编码字符集中每一字符c的频率是 $f(c)$ 。以f为键值的优先队列Q用在贪心选择时有效地确定算法当前要合并的2棵具有最小频率的树。一旦2棵具有最小频率的树合并后，产生一棵新的树，其频率为合并的2棵树的频率之和，并将新树插入优先队列Q。经过 $n-1$ 次的合并后，优先队列中只剩下一棵树，即所要求的树T。

一个实例

- 假设一个文件由字符a,b,c,d,e组成。每个字符的出现频率为

$$f(a)=20, f(b)=7, f(c)=10, f(d)=4, f(e)=18.$$



使用定长编码: $3 \times (20+7+10+4+18)=177$

使用哈夫曼编码:

a: 01

b: 110

c: 10

d: 111

e: 00

$$2 \times 20 + 3 \times 7 + 2 \times 10 + 3 \times 4 + 2 \times 18 = 129$$

$$(177-129)/177=27\%$$

算法

输入：n个字符的集合 $C=\{c_1, \dots, c_n\}$ 及其频度 $\{f(c_1), \dots, f(c_n)\}$

输出：C的Huffman树(V,T)

1.根据频度将所有字符插入最小堆 H

2. $V \leftarrow C; T = \{\}$

3. for $j \leftarrow 1$ to $n-1$ //n个节点经n-1次合并，每次少1个节点

4. $c \leftarrow \text{Delete_Min}(H)$ //O(logn)

5. $c' \leftarrow \text{Delete_Min}(H)$ //O(logn)

6. $f(v) \leftarrow f(c) + f(c')$

7. $\text{INSERT}(H, v)$ //O(logn)

8. $V = V \cup \{v\}$

9. $T = T \cup \{(v, c), (v, c')\}$

10. end for

$T(n) = O(n \log n)$

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。

► 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如下：

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



输入的活动以其完成时间的**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。

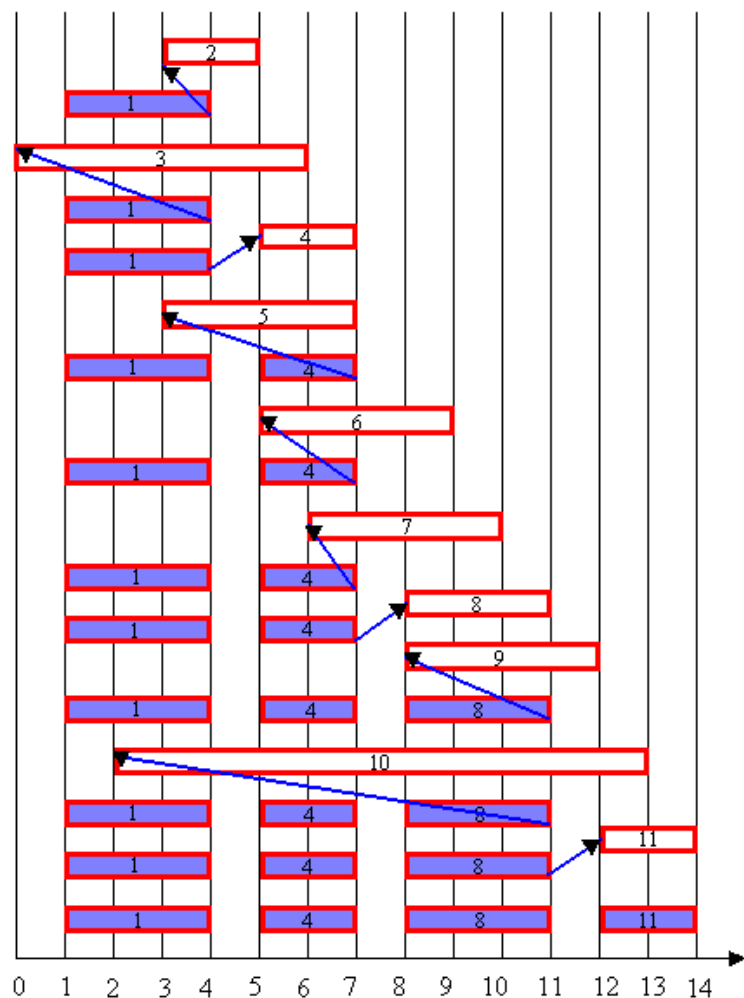
当输入的活动已按结束时间的非减序排列，算法只需 **$O(n)$** 的时间安排n个活动，使最多的活动能相容地使用公共资源。如果所给出的活动未按非减序排列，可以用 **$O(n \log n)$** 的时间重排。



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





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。

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。
- ▶ 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 $\{2, 14, 4, 16, 6, 5, 3\}$

- ▶ 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 $O(n \log n)$ 。

例:

- ▶ 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。

