贪心算法

陈长建

计算机科学系

回顾 – 最优装载

- 问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。
- [数学模型] 输入：$(x_1, x_2, \ldots, x_n)$, $x_i=0$表示集装箱i不装船；$x_i=1$表示集装箱i装船
  – 可行解：满足约束条件 $\sum_{i=1}^{n} w_i x_i \le c$的输入
  – 优化函数：$\sum_{i=1}^{n} x_i$

例3 – 最优装载

- **[算法思路]** 将装船过程划为多步选择，每步装一个货箱，每次从剩下的货箱中选择重量最轻的货箱. 如此下去直到所有货箱均装上船或船上不能再容纳其他任何一个货箱。
- [例]设n=8，[w_1，…, w_8]=[100，200，50，90，150，50 ， 20 ， 80]，c=400.
  - 所考察货箱的次序为:7，3，6，8，4，1，5 ， 2。货箱7，3，6，8，4，1的总重量为390个单位且已被装载，剩下的装载能力为10,小于任意货箱. 所以得到解

    [1，0，1，1，0，1，1，1]

例子4 - 哈夫曼编码

- 1951年，哈夫曼和他在MIT信息论的同学需要选择是完成学期报告还是期末考试。导师Robert M. Fano给他们的学期报告的题目是，寻找最有效的二进制编码。由于无法证明哪个已有编码是最有效的，哈夫曼放弃对已有编码的研究，转向新的探索，最终发现了基于有序频率二叉树编码的想法，并很快证明了这个方法是最有效的。

1. 前缀码

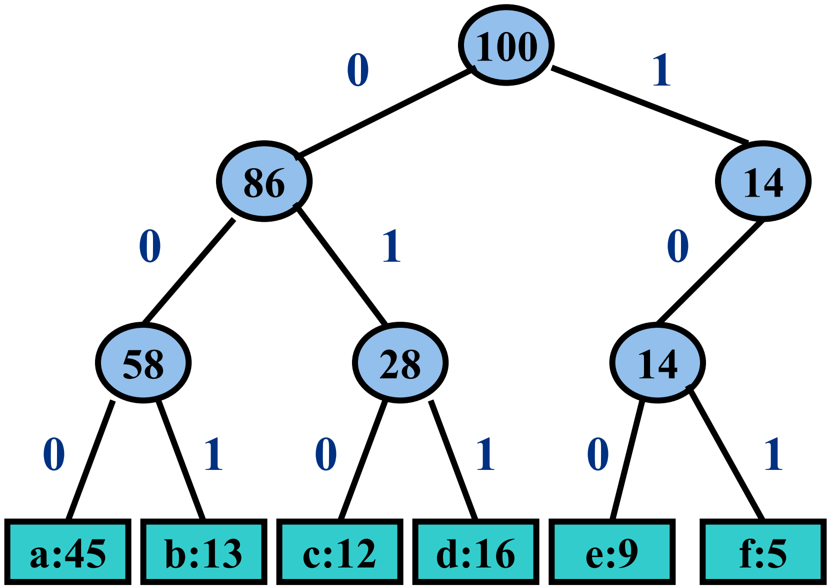
- 定义：对每一个字符规定一个0,1串作为其代码，并要求任一字符的代码都不是其他字符代码的前缀。这种编码称为**前缀码**。
- 编码的前缀性质可以使译码方法非常简单。由于任一字符的代码都不是其他字符代码的前缀，从编码文件中不断取出代表某一字符的前缀码，转换为原字符，即可逐个译出文件中的所有字符。

解码示例

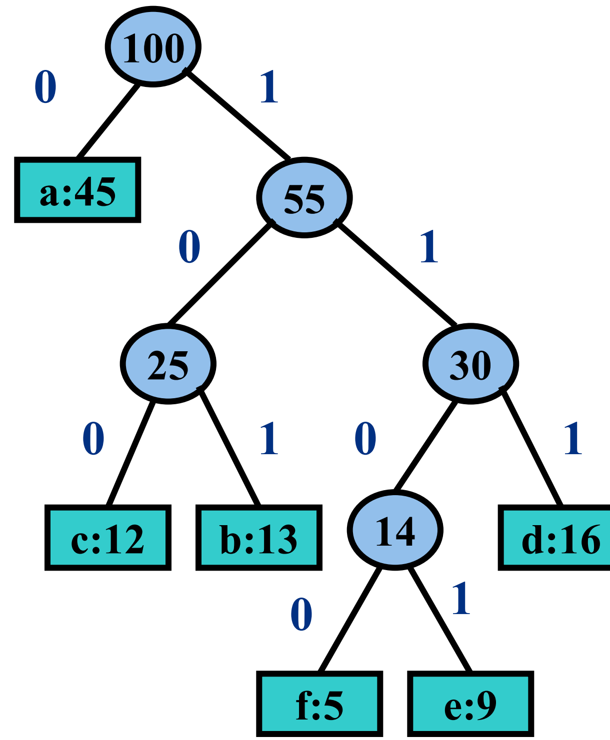
- 010110011111011100101

	a	b	c	d	e	f
频率(千次)	45	13	12	16	9	5
定长码	000	001	010	011	100	101
变长码	0	101	100	111	1101	1100

前缀码的二叉树表示



定长码



变长码

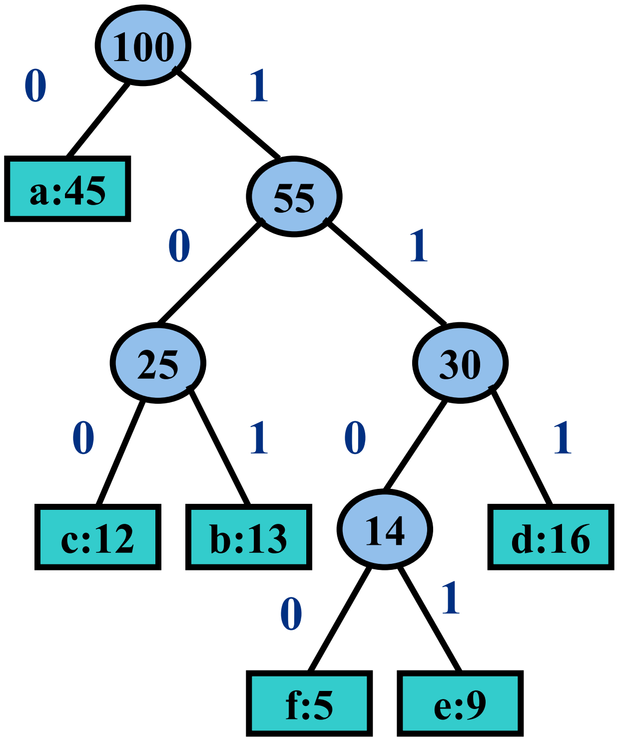
该编码方案的平均码长定义为： $B(T)=\sum_{c\in C} f(c)d_T(c)$

2. 构造哈夫曼编码

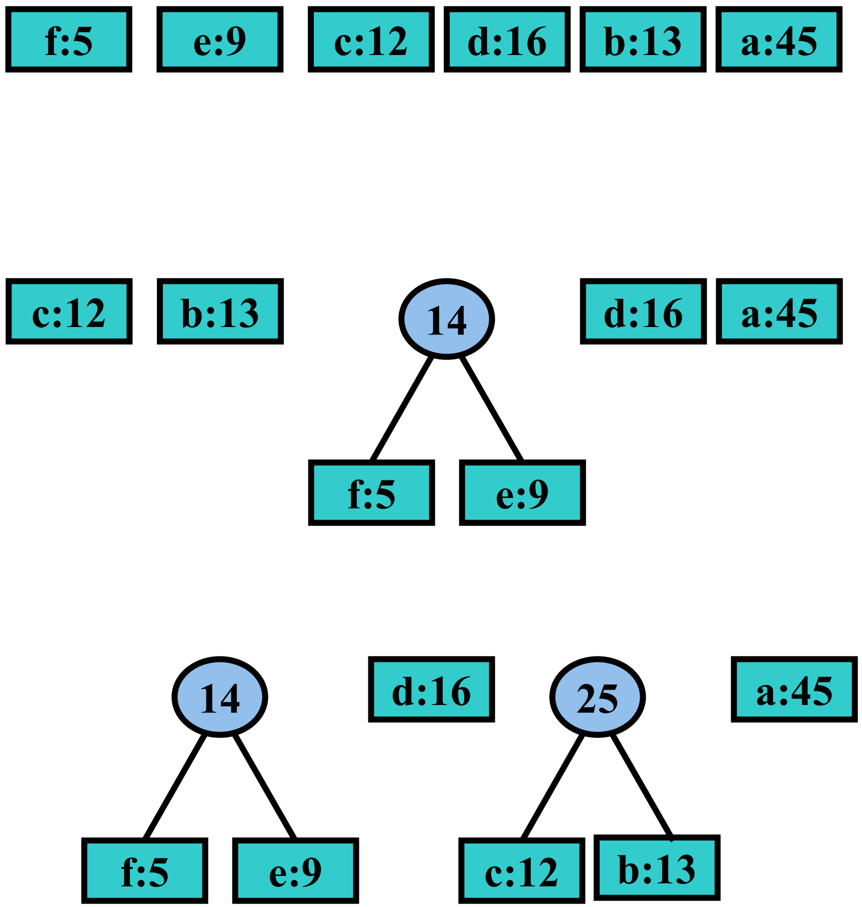
- 哈夫曼算法以自底向上的方式构造表示最优前缀码的二叉树T。
- 编码字符集中每一字符c的频率是$f(c)$。以f为键值的优先队列Q用以在作贪心选择时有效地确定算法当前要合并的两棵具有最小频率的树。一旦两棵具有最小频率的树合并后，产生一棵新的树，其频率为合并的两棵树的频率之和，并将新树插入优先队列Q中，再进行新的合并。

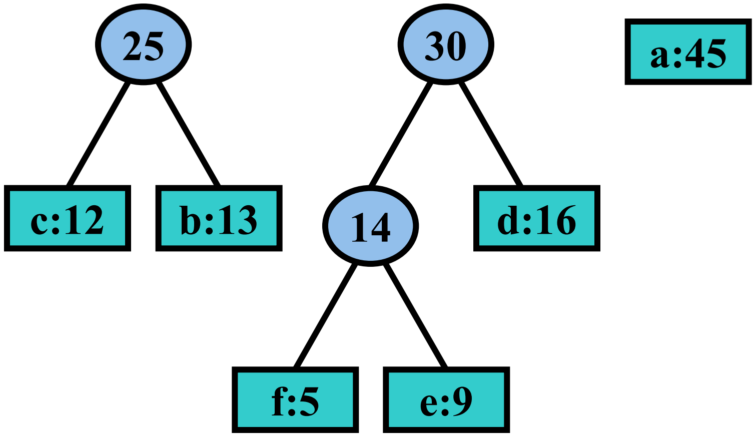
举例

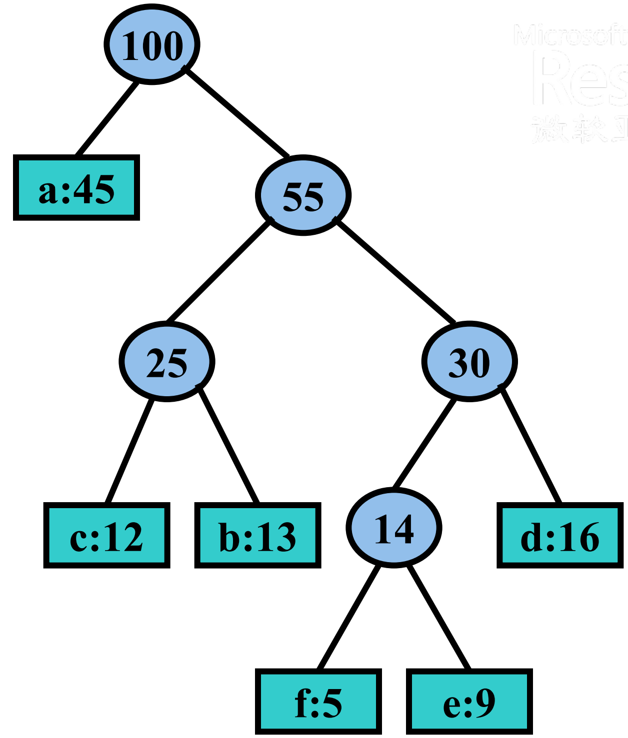
	a	b	c	d	e	f
频率(千次)	45	13	12	16	9	5

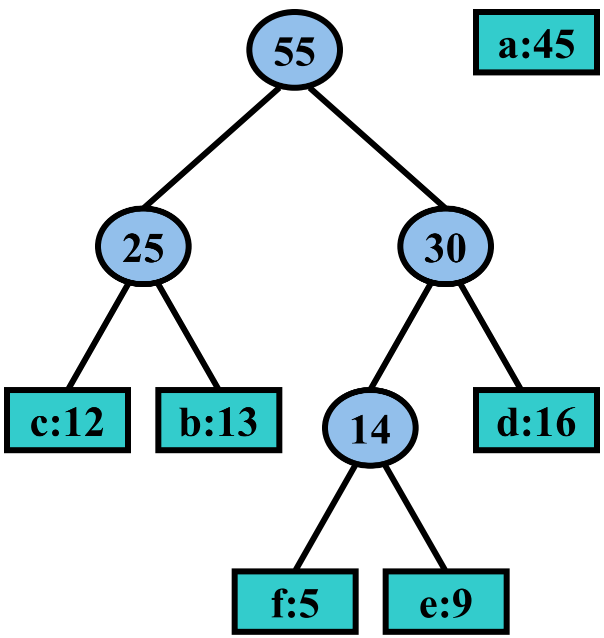


- 哈夫曼算法的执行过程示例：









- 由于字符集中有6个字符，优先队列的大小初始为6，总共用5次合并得到最终的编码树T。
- 每次合并使Q的大小减1，最终得到的树就是**最优前缀编码：哈夫曼编码树**，每个字符的编码由树T的根到该字符的路径上各边的标号所组成。

- 算法首先用字符集C中每一个字符c的频率$f(c)$初始化优先队列Q。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。
- 然后不断地从优先队列Q中取出具有最小频率的两棵树x和y，将它们合并为一棵新树z。z的频率是x和y的频率之和。
- 新树z以x为其左儿子，y为其右儿子（也可以y为其左儿子，x为其右儿子。不同的次序将产生不同的编码方案，但平均码长是相同的）。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。

3. 哈夫曼编码的正确性

- 要证明哈夫曼算法的正确性，就要证明最优前缀码问题具有贪心选择性质和最优子结构性质。

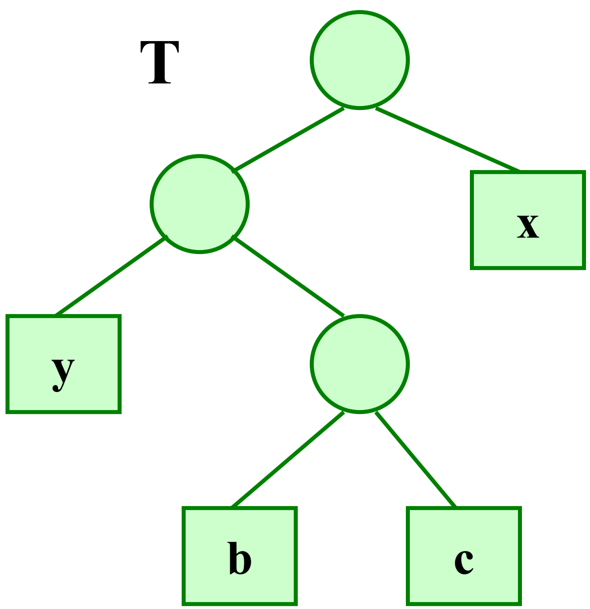
(1) 贪心选择性质

引理：设C为一字母表，其中每个字符c具有频度f[c]。设x和y为C中具有最低频度的两个字符，则存在C的一种最优前缀编码，其中x和y的编码长度相同但最后一位不同。

证明：贪心选择性质

- 设b和c是二叉树T的最深叶子且为兄弟。
- 不失一般性，可设： $f(b) \le f(c), \quad f(x) \le f(y)$
  - **由于x和y是C中具有最小频率的两个字符，故：**

$$f(x) \le f(b), \quad f(y) \le f(c)$$

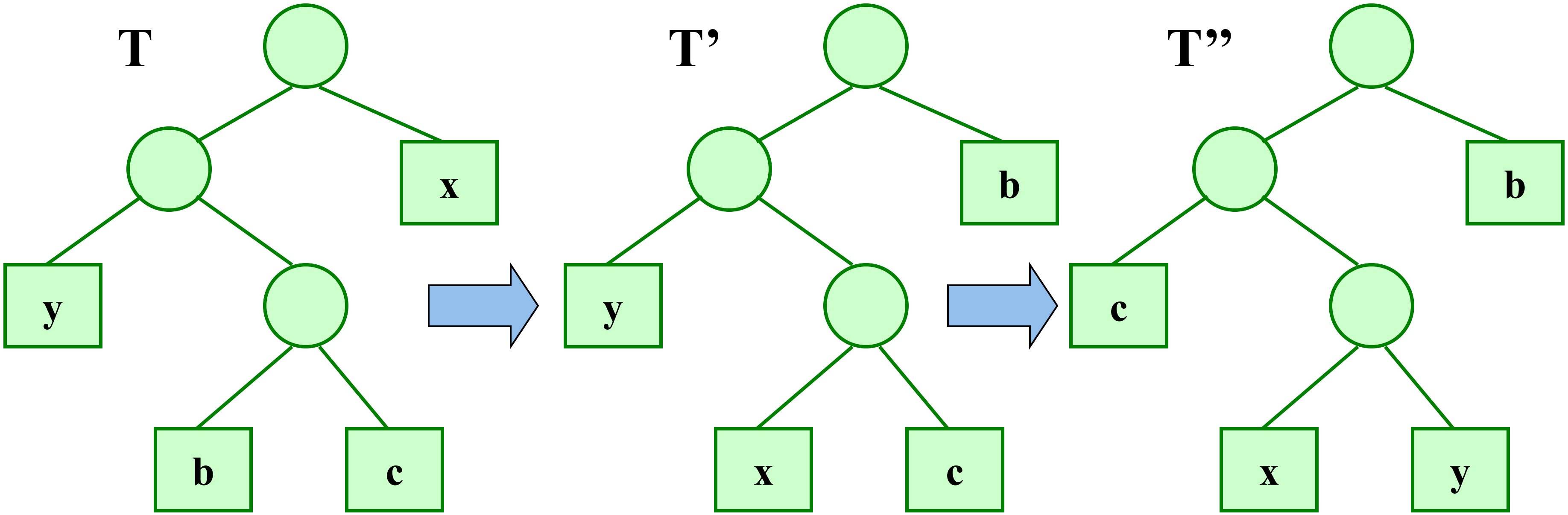


设，二叉树T表示C的任意一个最优前缀码。

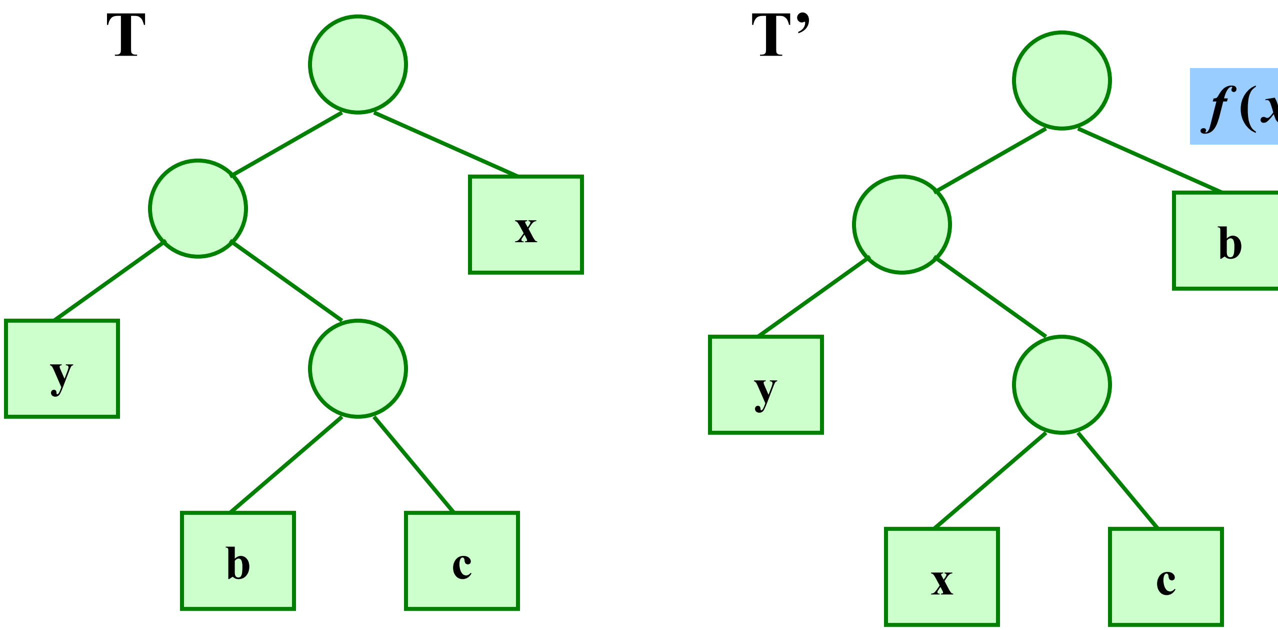
需证明，可以对T做适当修改后得到一棵新的二叉树T”，使在新树中，x和y是最深叶子且为兄弟。同时新树T”表示的前缀码也是C的最优前缀码。

$f(x) \le f(b), \quad f(y) \le f(c)$

- 首先在树T中交换叶子b和x的位置得到树T'：



- **然后在树T'中交换叶子c和y的位置得到树T''。**

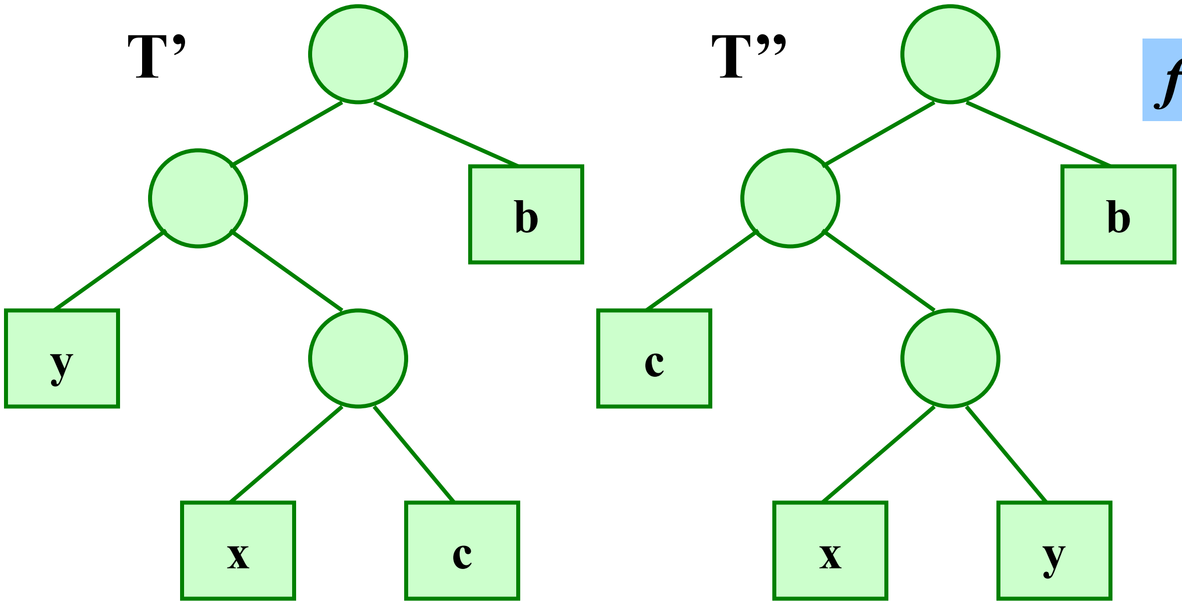


$f(x) \le f(b), \quad f(y) \le f(c)$

树T’的平均码长不会长于树T。

- 由此可知，树T和T’表示的前缀码的平均码长之差为：

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



$f(x) \le f(b), \quad f(y) \le f(c)$

- 类似地，可以证明在T’中交换y与c的位置也不增加平均码长，即：

$$B(T') - B(T'') \ge 0$$

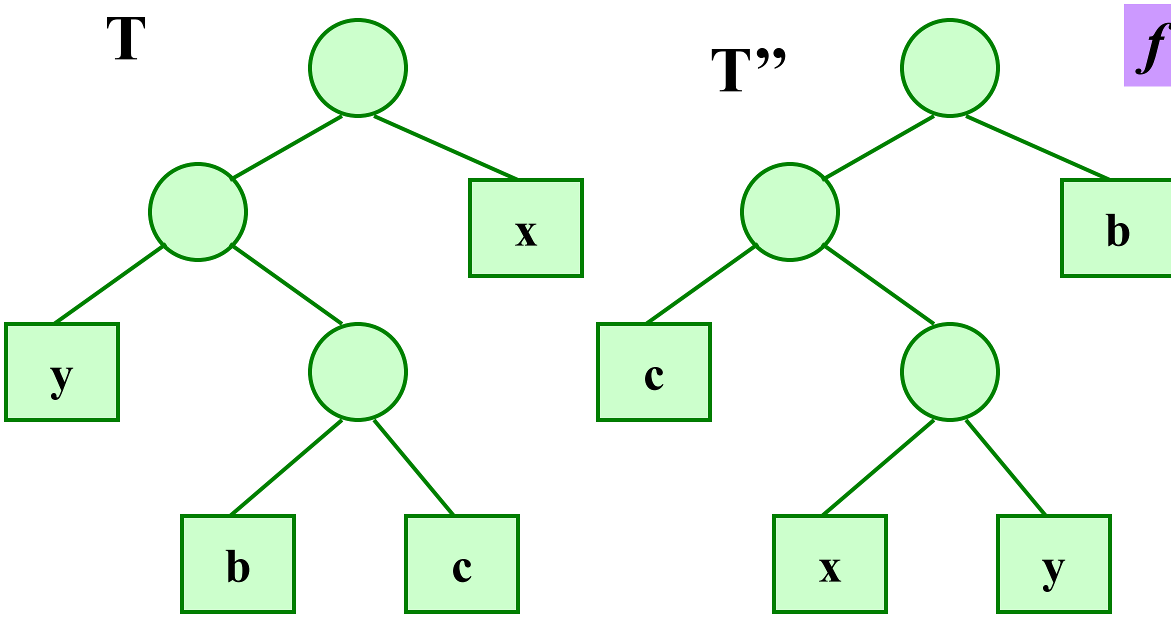
- **由此可知：**

$$B(T) \ge B(T') \ge B(T'')$$

$B(T) \geq B(T') \geq B(T'')$

问题：什么时候取到等号？

- 由于T所表示的前缀码是最优的，故 $B(T) \leq B(T'')$
- 因此： $B(T) = B(T'')$
- 结论：T''表示的前缀码也是最优前缀码，且x和y具有最长的码长，同时仅仅最后一位编码不同。



$f(x) \leq f(b), \quad f(y) \leq f(c)$

(2) 最优子结构性质

- 引理：设T为表示字母表C上一种最优前缀代码的二叉树。对C中每个字符定义有频度f[c]。考虑T中任意两个为兄弟叶节点的字符x和y，并设z为它们的父节点。那么，若认为z是一个频度为f[z]=f[x]+f[y]的字符的话，树T' = T - {x,y}就表示了字母表C' = C - {x,y} ∪ {z}上的一种最优前缀编码。

设去掉x,y后计算各部分代价为B(T')：

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

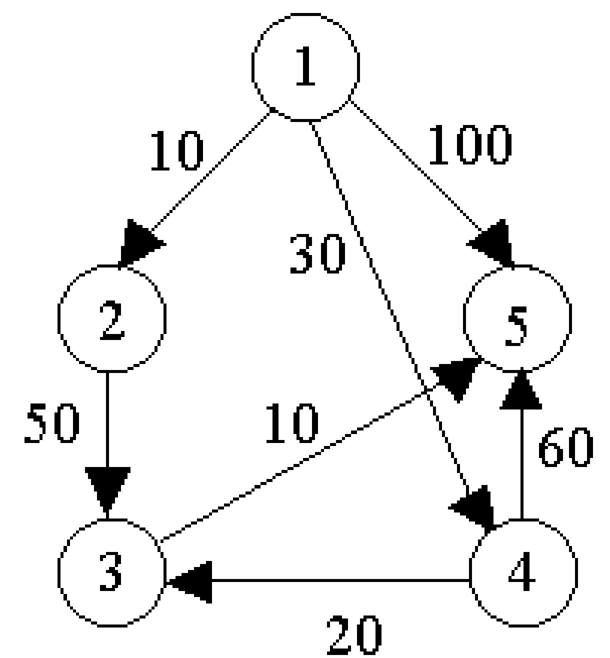
所以根据此式：B(T)=B(T') + f[x] + f[y]

如果T'代表C'上一种非最优前缀代码，则存在叶节点z为C'中的字符的树T''，将x和y插入T''中使它们成为z的子结点，可以得到C的一种前缀代码，使得B(T'') + f[x] + f[y] < B(T)，和T的最优性矛盾。

例5 - 单源最短路径

- 给定带权有向图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。
- 现在要计算从源到其他所有各顶点的最短路径长度。这里的路径长度是指路径上各边权之和，这个问题通常称为单源最短路径问题。

例如：右图中的有向图，计算从源顶点1到其他顶点的最短路径。



算法基本思想

- Dijkstra算法是求解单源最短路径问题的一个贪心算法。
- 基本思想：设置一个顶点集合S，并不断地作贪心选择来扩充这个集合。一个顶点属于集合 S 当且仅当从源到该顶点的最短路径长度已知。
- Dijkstra算法通过分步方法求出最短路径。
  - 每一步产生一个到达新的目的顶点的最短路径。
  - 下一步所能达到的目的顶点通过这样的贪心准则选取：在还未产生最短路径的顶点中，选择路径长度最短的目的顶点。
  - 也就是说，Dijkstra算法按路径长度顺序产生最短路径。

Dijkstra算法的执行

- 设置一个顶点集合S。一个顶点属于集合 S 当且仅当从源到该顶点的最短路径长度已知。
- 初始时，S中仅含有源。
- 设u是G的某一个顶点，把从源到u且中间只有经过S中顶点的路称为从源到u的特殊路径，并且用数组dist来记录当前每个顶点所对应的最短特殊路径长度。
- Dijkstra算法每次从V-S中取出具有最短特殊路径长度的顶点u，将u添加到 S 中，同时对数组dist作必要的修改。
- 一旦S包含了所有V中顶点，dist就记录了从源到所有其他顶点之间的最短路径长度。

举例

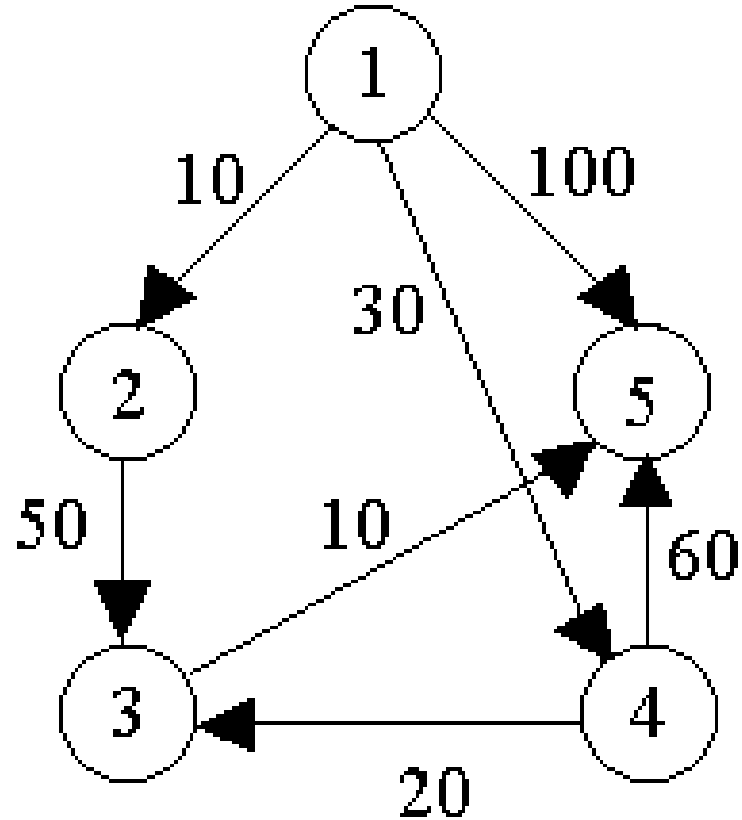
- 已知：带权有向图

$V = \{ v_1, v_2, v_3, v_4, v_5 \}$

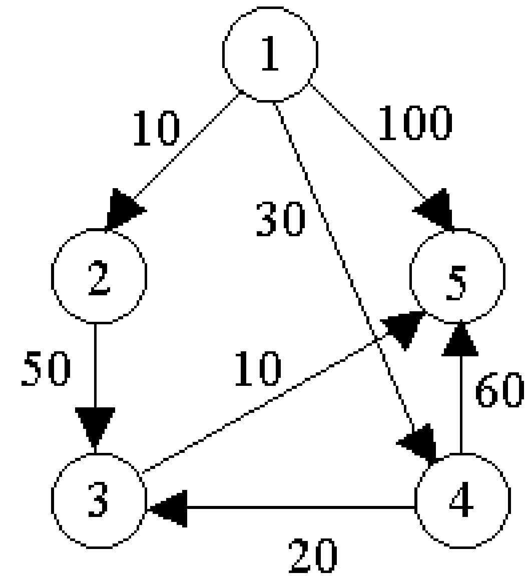
$E = \{ < v_1, v_2 >, < v_1, v_4 >, < v_1,$

$>, < v_4, v_3 >, < v_4, v_5 > \}$

- 设为v_1源点，求其到其余顶

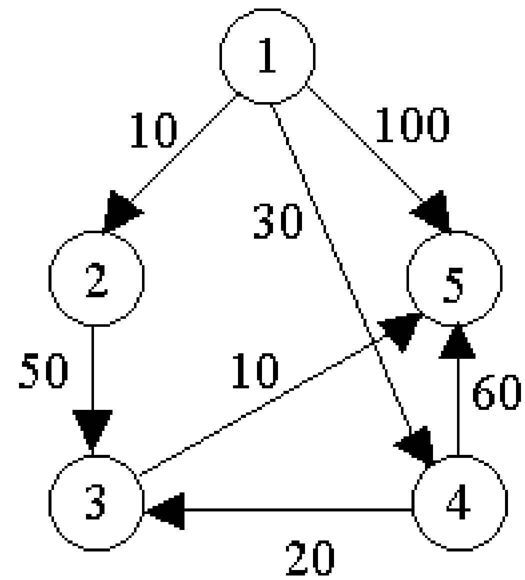


- 初始时，集合S只有源点v_1，即加入集合S中的顶点u为v_1。
- 从源点v_1到其它顶点的**最短特殊路径（中间只有来自于集合S中的顶点）长度**分别为:
  - dist[2]=10; dist[3]=maxint; dist[4]=30; dist[5]=100.
  - 其中，没有特殊路径的顶点v_3用maxint表示其最短特殊路径长度。



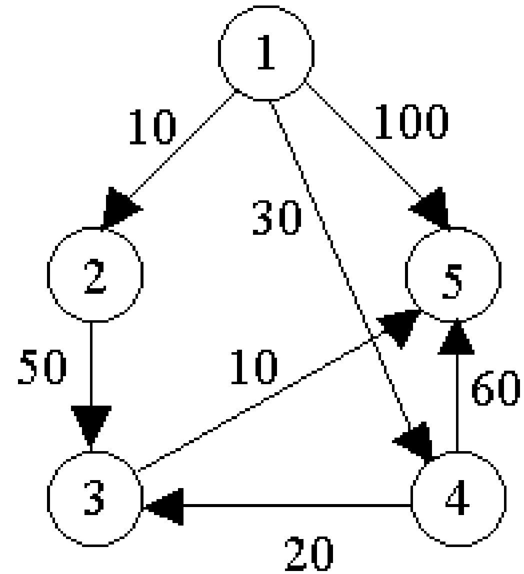
迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100

- 集合S为{v_1}，其余顶点的最短特殊路径长度已确定。
- 由于dist[2]的值最小，为10，所以将顶点v_2加入集合S中。
- 由于集合S为{v_1,v_2}，需要修改剩余的三个顶点的最短特殊路径值。
- **例如，** v3的最短特殊路径为<v_1,v_2> ,<v_2,v_3>；长度为60。



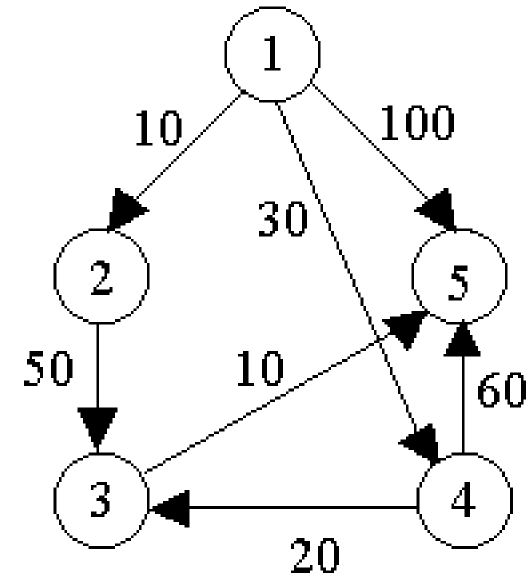
迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1,2}	2	10	60	30	100

- 集合S为$\{v_1,v_2\}$，其余顶点的最短特殊路径长度已确定。
- 其中dist[4]的值最小，为30，所以将顶点v_4加入集合S中。
- 由于集合S为$\{v_1,v_2,v_4\}$，需要修改剩余的两个顶点的最短特殊路径值。
- 例如，v_3的最短特殊路径为$<v_1,v_4>$,$<v_4,v_3>$；长度为50。



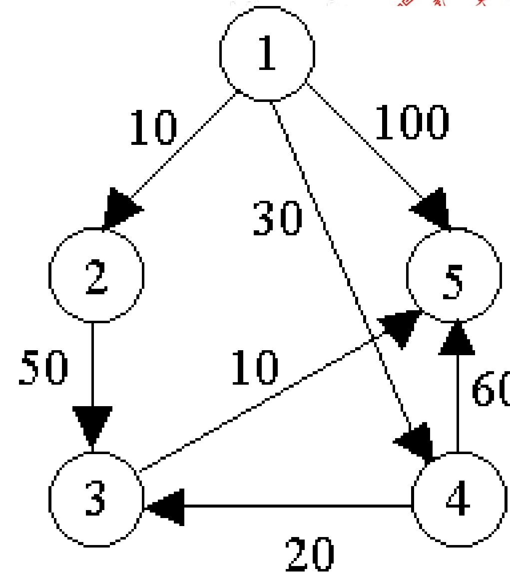
迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90

- 集合S为{v_1,v_2,v_4}，其余顶点的最短特殊路径长度已确定。
- 其中dist[3]的值最小，为50，所以将顶点v_3加入集合S中。
- 由于集合S为{v_1,v_2,v_4,v_3}，需要修改剩余的一个顶点的最短特殊路径值。
- 例如，v_5的最短特殊路径为 <v_1,v_2> ,<v_2,v_3> ,<v_3,v_5>；长度为60。



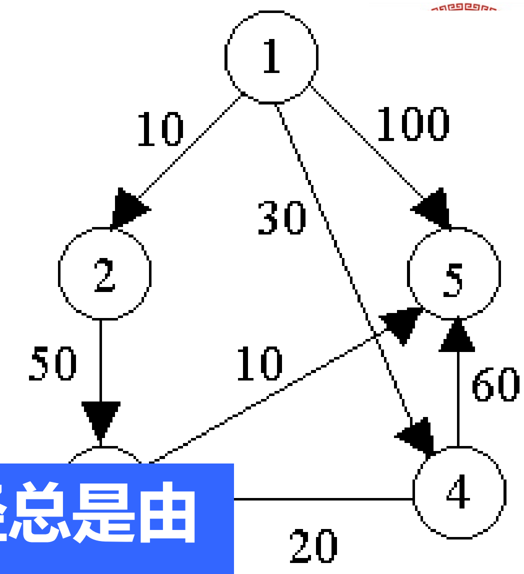
迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60

- 集合S为$\{v_1,v_2,v_4,v_3\}$，其余顶点的最短特殊路径长度已确定。
- 由于只剩余一个顶点v_5不在集合中，所以应该把它加入集合。
- 此时集合S为$\{v_1,v_2,v_4,v_3,v_5\}=V$，完成。dist值为源点到对应顶点的最短路径长度。



迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

- 第2条路径是第1条路径扩充一条边形成的；
- 第3条路径则是第2条路径扩充一条边；
- 第4条路径是第1条路径扩充一条边；
- 第5条路径是第3条路径扩充一条边。

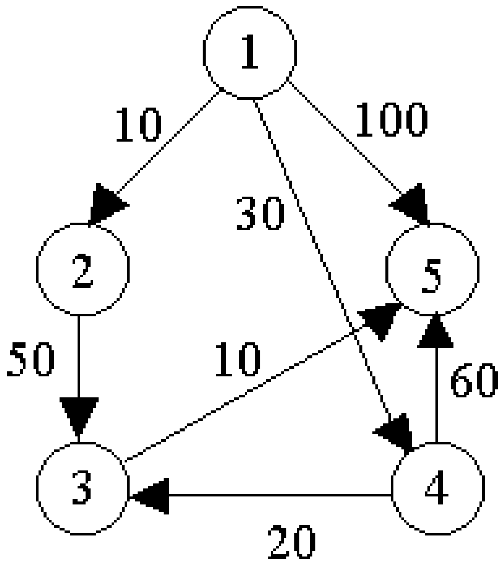


按长度顺序产生最短路径时，下一条最短路径总是由一条已产生的最短路径加上一条边形成。

迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

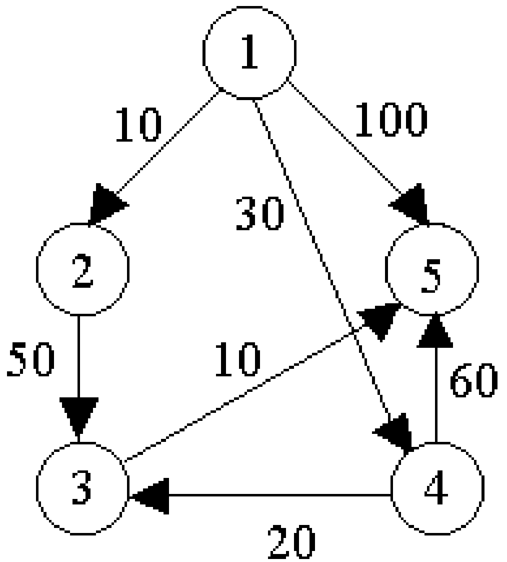
迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

1. 用Dist[v]记录任一顶点v到源点的最短路径，建立一S集合且为空（开始只有源点），用以记录已找出最短路径的点。



迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

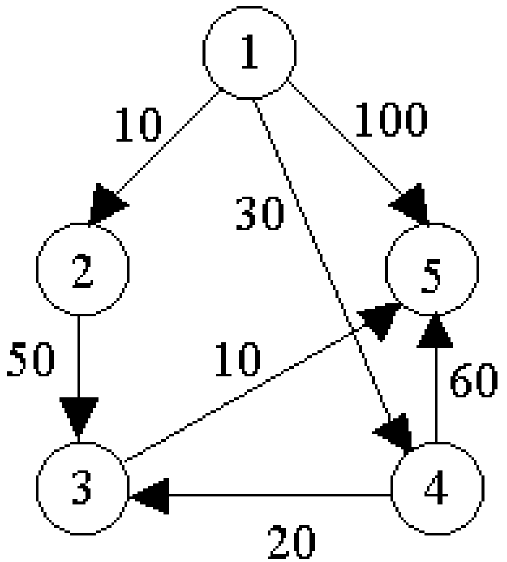
2. 扫描非S集中Dist[]值最小的节点Dist[u]，也就是找出下一条最短路径，把节点u加入S集中。



迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

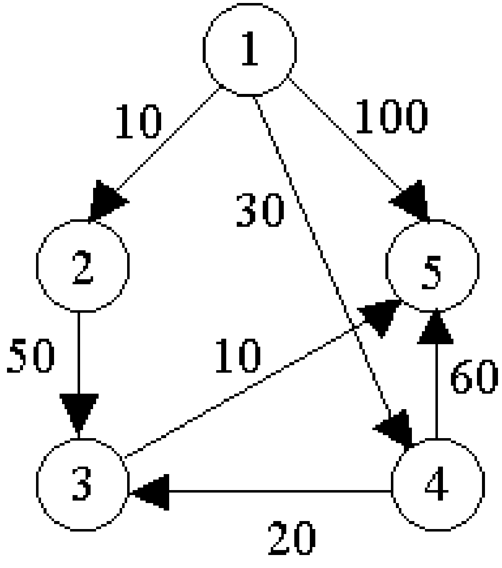
3. 更新所有非S集中的Dist[]值，看看是否可通过新加入的u点让其路径更短：

```
if ( Dist[u]+(u,v)<Dist[v] ) then
    Dist[v]=Dist[u]+(u,v);
```



迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

4. 跳转到2操作，循环（顶点数-1）次，依次找出所有顶点的最短路径。



算法的正确性

1. 贪心选择性质

- 存在一个最优解，在开始时，Dist[u]中最小的节点是源头到这个点的最短路径

算法的正确性

2. 最优子结构性质

- Dijkstra算法所作的贪心选择是从V-S中选择具有最短特殊路径的顶点u，从而确定从源到u的最短路径长度dist[u]。
- 为什么从源到u没有更短的其他路径呢?

•假设Dijkstra算法确定到u的路径为v -> .. -> t -> u

• 如果存在一条从源到u且长度比dist[u]更短的路，设这条路最后经过S内的点是x

$d(v,x)+d(x,u) \le d(v,t) + d(t,u)$ 这与dist[u]的计算方式矛盾

算法的正确性

2. 最优子结构性质

- Dijkstra算法所作的贪心选择是从V-S中选择具有最短特殊路径的顶点u，从而确定从源到u的最短路径长度dist[u]。
- 为什么从源到u没有更短的其他路径呢?

• 如果存在一条从源到u且长度比dist[u]更短的路，设这条路初次走出S之外到达的顶点为x∈V-S，然后徘徊于S内外若干次，最后离开S到达u。

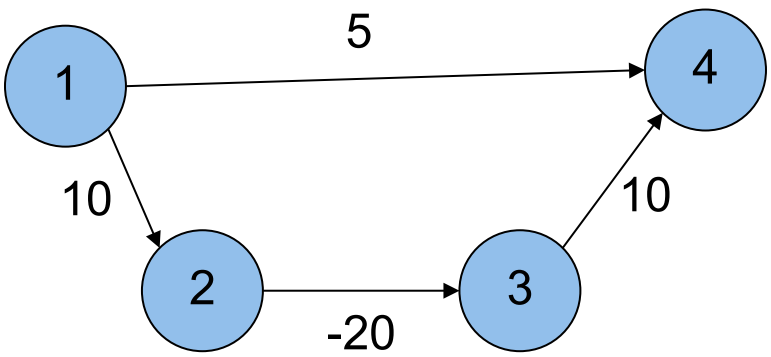
• 在这条路径上，分别记d(v,x)，d(x,u)和d(v,u)为顶点v到顶点x，顶点x到顶点u和顶点v到顶点u的路径长，那么，有

$$\mathrm{dist}[x] \le d(v,x) \quad d(v,x)+d(x,u)=d(v,u)<\mathrm{dist}[u]$$

利用边权的非负性，可知d(x,u)≥0从而推得dist[x]≤dist[u]，产生矛盾。证明dist[u]是源到顶点u的最短路径长度。

思考

- 例子



例6 - 最小生成树

- 设G=(V,E)是无向带权连通图，即一个网络。
- E中每条边(v,w)的权为c[v][w]。如果G的子图G’是一棵包含G的所有顶点的树，则称G’为G的生成树。
- 生成树上各边权的总和称为该生成树的耗费。在G的所有生成树中，耗费最小的生成树称为G的最小生成树。

应用

- 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。

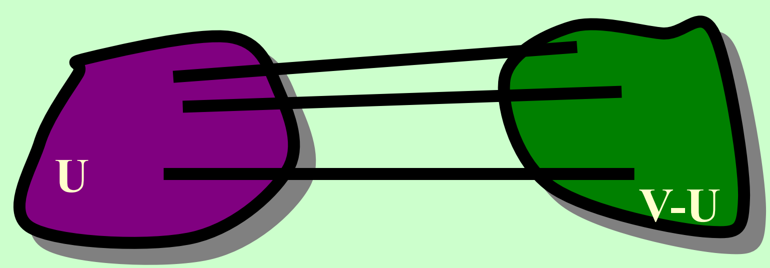
贪心法求解准则

- 将贪心策略用于求解无向连通图的最小代价生成树时，核心问题是需要确定贪心准则。
- 根据最优量度标准，算法的每一步从图中选择一条符合准则的边，共选择n-1条边，构成无向连通图的一棵生成树。
- 贪心法求解的关键：该量度标准必须足够好。它应当保证依据此准则选出n-1条边构成原图的一棵生成树，必定是最小代价生成树。

算法步骤分析

- 设G=(V,E)是带权的连通图，T=(V,S)是图G的最小代价生成树。

```
ESetType SpanningTree(ESetType E,int n)
{ //G=(V,E)为无向图，E是图G的边集，n是图中结点数
    ESetType TE=Ø;    //TE为生成树上边的集合
    int u,v,k=0; EType e;      //e=(u,v)为一条边
    while(k<n-1 && E中尚有未检查的边)
    {   //选择生成树的n-1条边
        e=select(E);   //按最优量度标准选择一条边
        if(TE∪e 不包含回路)  //判定可行性
        {    TE=TE∪e; k++;   }//在生成树边集TE中添加一条边
    }
    return S;
}
```



普里姆（Prim）算法
克鲁斯卡尔（Kruskal）算法

- Kruskal算法的贪心准则：按边代价的非减次序考察E中的边，从中选择一条代价最小的边e=(u,v)。
  - 这种做法使得算法在构造生成树的过程中，当前子图不一定是连通的。
- Prim算法的贪心准则：在保证S所代表的子图是一棵树的前提下选择一条最小代价的边e=(u,v)。

Prim算法的基本步骤

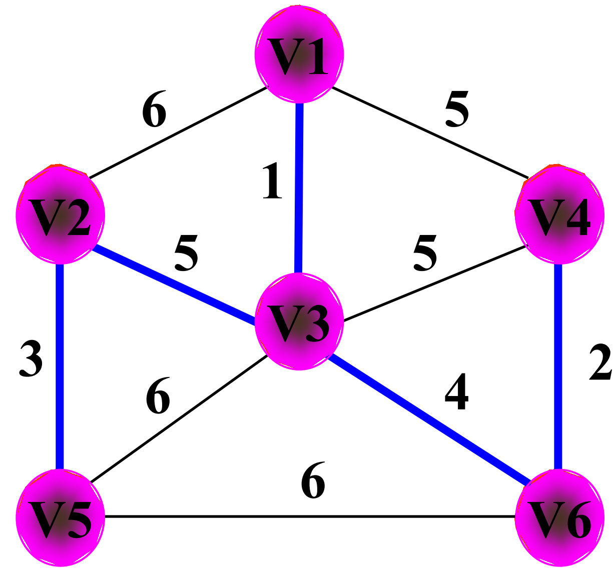
1. 在图G=(V, E)（V表示顶点集合，E表示边集合）中，从集合V中任取一个顶点（例如取顶点v_1）放入集合U中，这时U={v_1}，生成树边集合T(E)为空。
2. 寻找与S中顶点相邻（另一顶点在V-U中）权值最小的边的另一顶点v_2，并使v_2加入S。即U={v_1,v_2}，同时将该边加入集合T(E)中。
3. 重复2，直到U=V为止。

- 这时T(E)中有n-1条边，T=(U,T(E))就是一棵最小生成树。

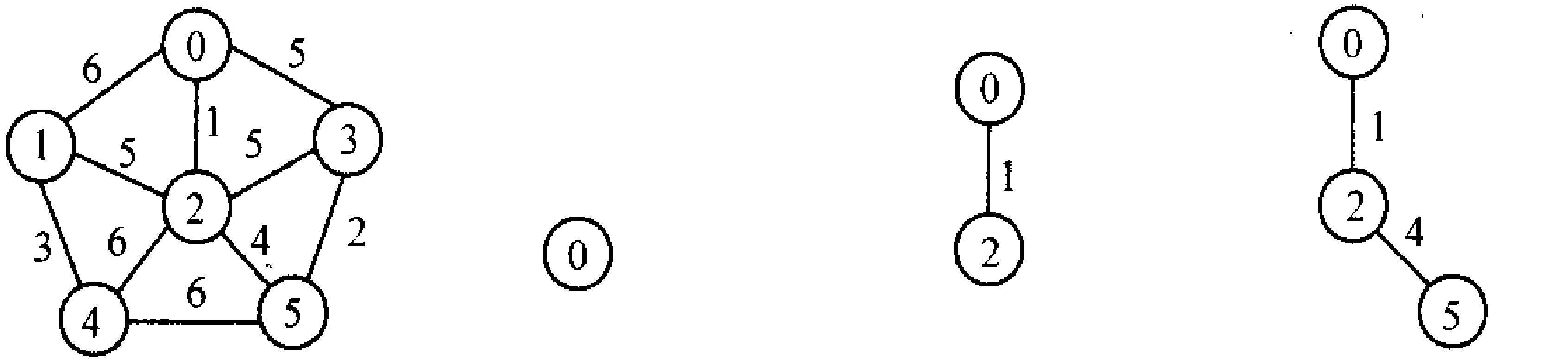
prim算法

算法思想：

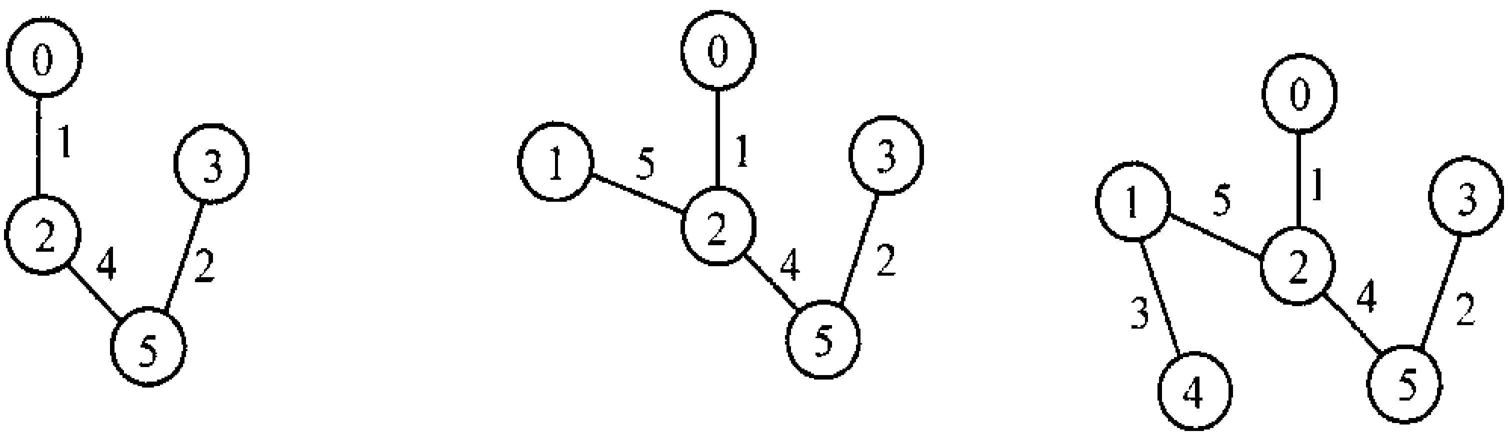
- 设 $N=(V, E)$ 是连通网，TE 是 N 上最小生成树中**边的集合**。
- 初始令 $U=\{u_0\}, (u_0 \in V), TE=\{\ \}$。
- 在所有 $u \in U, v \in V-U$ **的边** $(u, v) \in E$ 中，找一条**代价最小**的边 (u_0, v_0)。
- 将 (u_0, v_0) 并入集合 TE，同时 v_0 并入 U。
- 重复上述操作直至 $U=V$ 为止，则 $T=(V, TE)$ 为 N 的最小生成树。



Prim算法举例



（a）无向图G　（b）只有源点　（c）加入第1条边　（d）加入第2条边

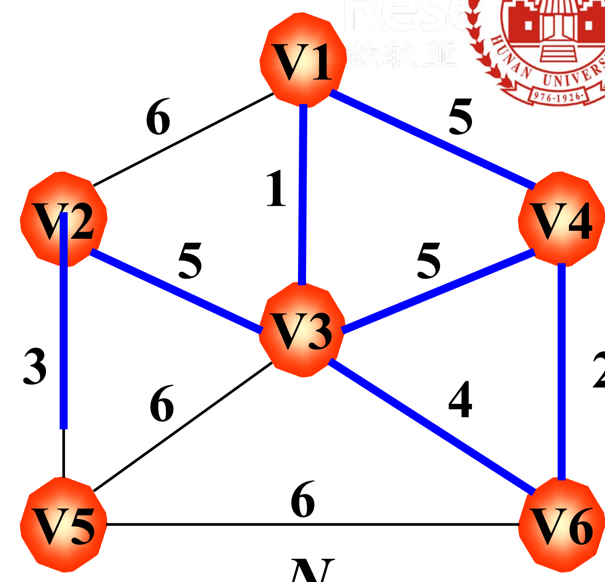


（e）加入第3条边　（f）加入第4条边　（g）图G的最小代价生成树

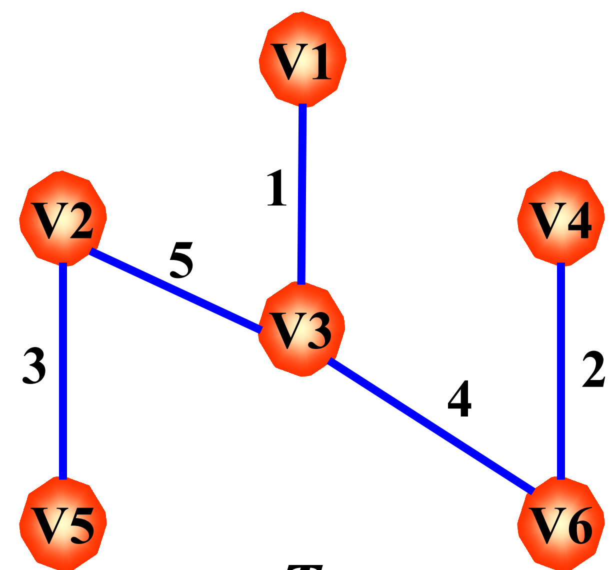
Kruskal算法

算法思想：

- 设连通网 $N=(V,E)$，令最小生成树初始状态为只有 n 个顶点而无边的非连通图 $T=(V,\{\ \})$，每个顶点自成一个连通分量。
- 在 E 中选取代价最小的边，若该边依附的顶点落在 T 中不同的连通分量上（即：不能形成环），则将此边加入到 T 中；否则，舍去此边，选取下一条代价最小的边。
- 依此类推，直至 T 中所有顶点都在同一连通分量上为止。

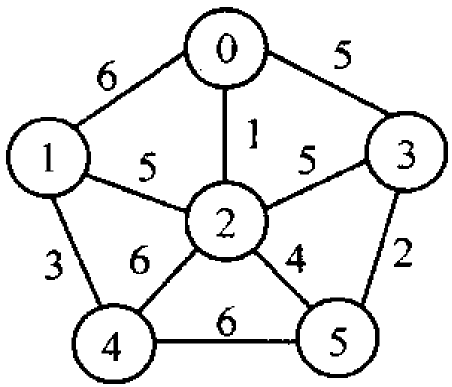


N

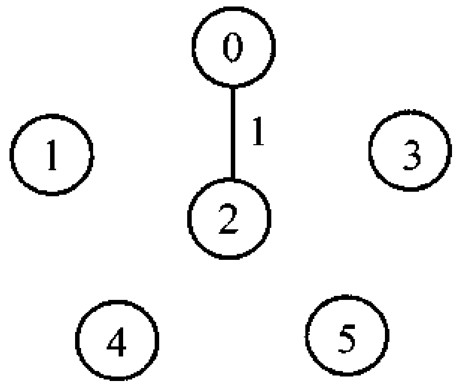


T

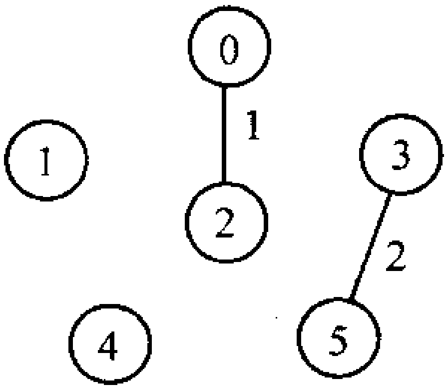
Kruskal算法举例



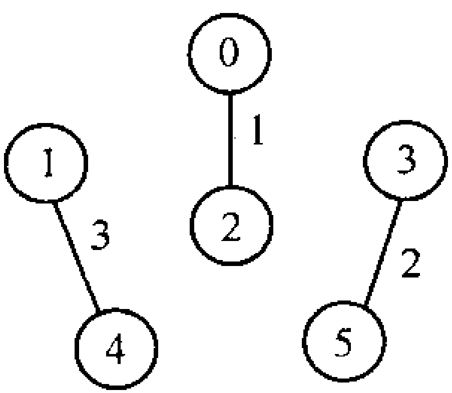
（a）无向图G



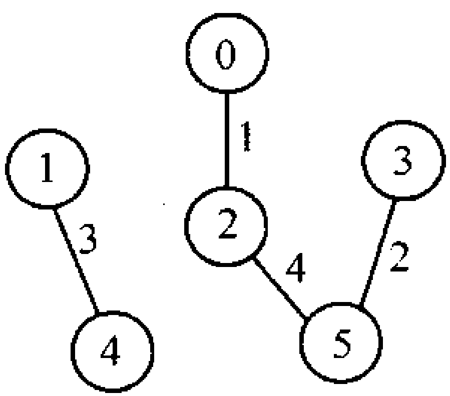
（b）加入第1条边



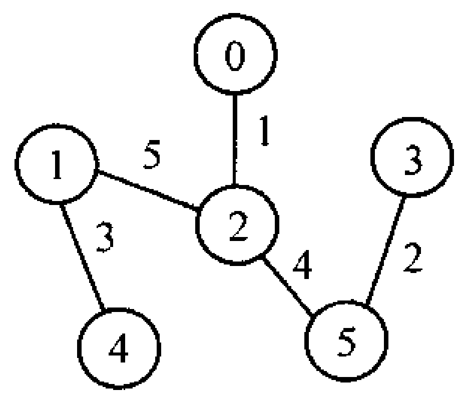
（c）加入第2条边



（d）加入第3条边



（e）加入第4条边



（f）图G的最小代价生成树

时间复杂度

- 如何定义两种算法的时间复杂度?
- Kruskal算法的时间复杂度为O(eloge)
- Prim算法的时间复杂度为$O(n^2)$
- 分别适合怎样的应用场合?

算法正确性

- 设图G=(V,E)是一个带权连通图，U是V的一个真子集。若边(u,v) ∈E是所有u∈U，v∈V-U的边中权值最小者，那么一定存在G的一棵最小代价生成树T=(V,TE)，(u,v) ∈TE。
- 这一性质称为MST（minimum spanning tree）性质。

证明：可以用反证法证明。

如果图G的任何一棵最小代价生成树都不包括(u,v)。将(u,v)加到图G的一棵最小代价生成树T中，将形成一条包含边(u,v)的回路，并且在此回路上必定存在另一条不同的边(u',v,)，使得u'∈U， v' ∈V-U。删除边(u',v,)，便可消除回路，并同时得到另一棵生成树T'。

算法正确性

- 设图G=(V,E)是一个带权连通图，U是V的一个真子集。若边(u,v) ∈E是所有u∈U，v∈V-U的边中权值最小者，那么一定存在G的一棵最小代价生成树T=(V,S)，(u,v) ∈S。
- 这一性质称为MST（minimum spanning tree）性质。

因为(u,v)的权值不高于(u',v)，则T'的代价亦不高于T，且T'包含(u,v)，故与假设矛盾。

这一结论是Prim算法和Kruskal算法的理论基础。

无论Prim算法和还是Kruskal算法，每一步选择的边均符合MST，因此必定存在一棵最小代价生成树包含每一步上已经形成的生成树（或者森林），并包含新添加的边。