第4章　贪心算法

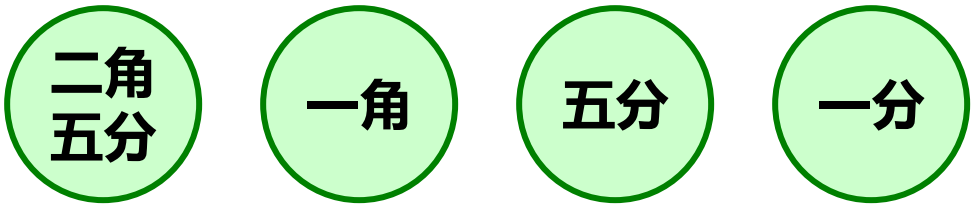
第4章 贪心算法

学习要点

- 理解贪心算法的概念。
- 掌握贪心算法的基本要素：
  - 最优子结构性质、贪心选择性质。
- 理解贪心算法与动态规划算法的差异。
- 通过应用范例学习动态规划算法设计策略：
  - 活动安排问题；背包问题；哈夫曼编码问题；单源最短路劲问题；最小生成树问题。

贪心算法应用举例——找硬币

- 假设有四种硬币，它们的面值分别为五角、一角、五分和一分。现找顾客六角三分钱，请给出硬币个数最少的找钱方案。



分析

- 找硬币问题本身具有最优子结构性质，它可以用动态规划算法来解。但显然贪心算法更简单，更直接且解题效率更高。
- 该算法利用了问题本身的一些特性，例如硬币面值的特性。
- 再例：如果有一分、五分和一角三种不同的硬币，要找给顾客一角五分钱。

结论

- 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。
- 当然，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。

贪心法的基本思路

- 从问题的某一个初始解出发逐步逼近给定的目标，以尽可能快地求得更好的解。当达到某算法中的某一步不能再继续前进时，算法停止。
- 实现该算法的过程

从问题的某一初

while 能朝给定

求出可行解

由所有解元素组合成问题的一个可行解。

该算法存在的问题：

1. 不能保证求得的最后解是最佳的；

2. 不能用来求最大或最小解问题；

3. 只能求满足某些约束条件的可行解的范围。

4.1 活动安排问题

- 活动安排问题：要求高效地安排一系列争用某一公共资源的活动。
- 举例：

活动序号	1	2	3	4	5	6	7	8	9	10	11
起始时间	1	3	0	5	3	5	6	8	8	2	12
结束时间	4	5	6	7	8	9	10	11	12	13	14

问题定义

- 设有n个活动的集合E={1,2,...,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。（临界资源）
- 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。即，当$s_i \geq f_j$或$s_j \geq f_i$时，活动i与活动j相容。
- 问题就是选择一个*由互相兼容的活动组成的最大集合*。

活动安排问题的贪心算法

```
template<class Type>
void GreedySelector(int n, Type s[], Type f[], bool A[])
{ //各活动的起始时间和结束时间存储在数组s和f中
  //且按结束时间的非递减排序：f1≤f2≤...≤fn排列。
  A[1]=true;  //用集合A存储所选择的活动
  int j=1;
  for(int i=2; i<=n; i++) {
      //将与j相容的具有最早完成时间的相容活动加入集合A
    if(s[i]>=f[j]) {A[i]=true; j=i; }
    else A[i]=false; }
}
```

算法分析

- 设集合a包含已被选择的活动，初始时为空。所有待选择的活动按结束时间的非递减顺序排列：

$$f_1 \le f_2 \le \cdots f_n$$

- 变量*j*指出**最近加入a的活动序号**。由于按结束时间非递减顺序来考虑各项活动的，所以f_j**总是a中所有活动的最大结束时间**

```
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;
  }
```

首先将活动1加入集合a；

初始化j为1；

然后从活动2出发逐一检查：

☆ 若待选活动i与a中的所有活动兼容，即活动i 的s_i不早于最近加入a中的j活动的f_j，则活动i加入a集合，并取代活动j的位置。

☆否则不选择该活动。

由于输入活动是以完成时间的非递减排列，所选择的下一个活动总是可被合法调度的活动中具有最早结束时间的那个，所以算法是一个“贪心的”选择，即使得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。

结论

- 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)的时间就可安排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(nlogn)的时间重排。

举例

- 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

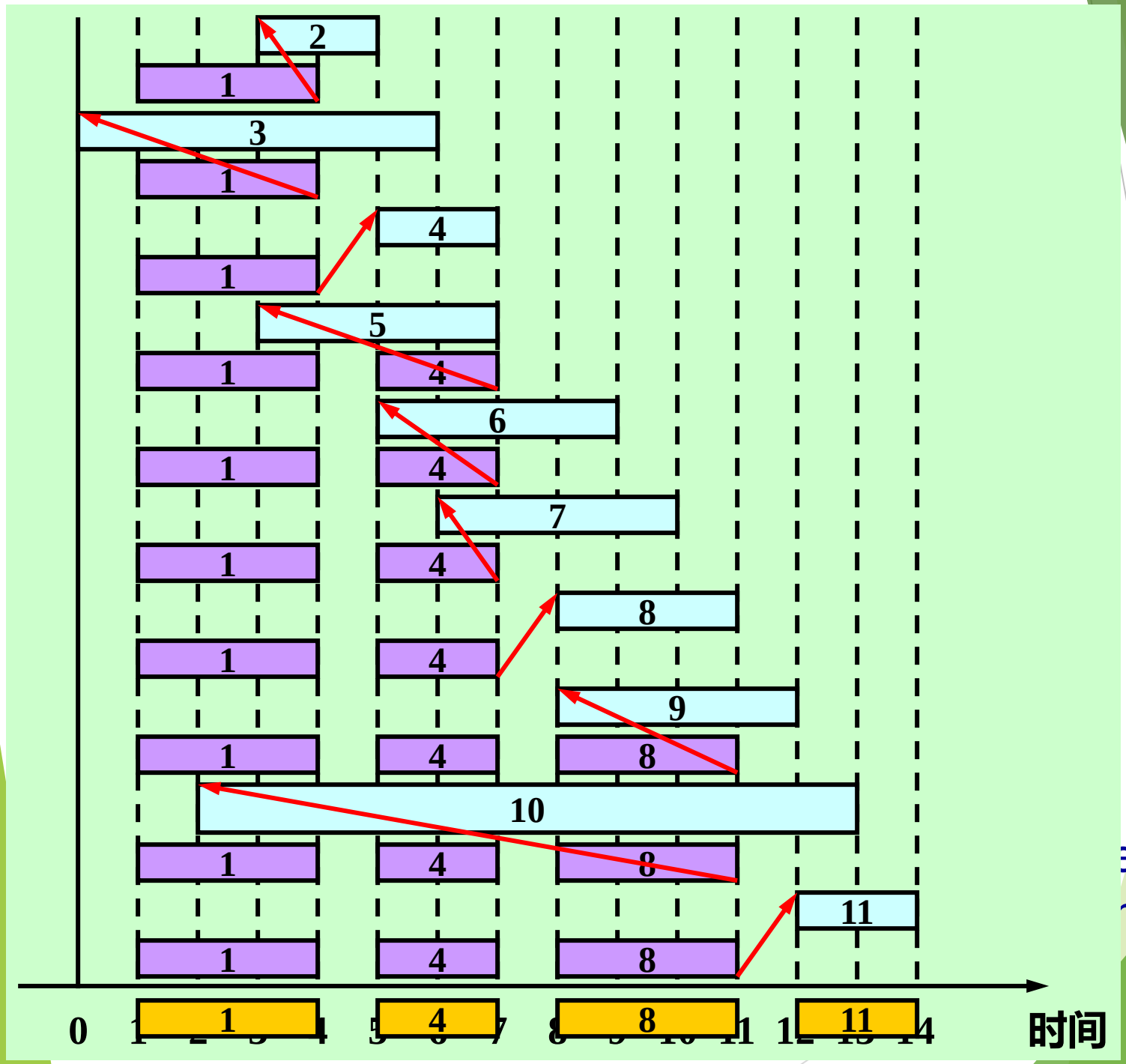


图4-1 算法
eedySelecto
r的计算过程

补充说明

- 贪心算法并不总能求得问题的整体最优解。
- 但对于活动安排问题，贪心算法GreedySelector却总能求得的整体最优解，即它最终所确定的相容活动集合a的规模最大。这个结论可以用数学归纳法证明。

用数学归纳法证明活动安排问题

- 设集合E={1，2，…，n}为所给的活动集合。由于E中活动按结束时间的非减序排列，故活动1有最早完成时间。
- 证明I：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。
- 证明II：对集合E中所有与活动1相容的活动进行活动安排求得最优解的子问题。

证明I：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$A \subseteq E$是所给的活动安排问题的一个最优解，

且A中活动也按结束时间递减序排列，A中第一个活动为活动k。

若k=1，则A就是一个以贪心选择开始的最优解。

若k>1，则设 $B = A - \{k\} \cup \{1\}$

由于$f_1 \leq f_k$，且A中的活动是相容的，故B中的活动也是相容的。

结论： 由于B中活动个数与A中活动个数相同，且A是最优的，故B也是最优的。**总存在以贪心选择开始的最优活动方案。**

贪心选择性质

证明II：对E中所有与*活动1相容的活动*进行活动安排求得最优解的子问题。

- 即需证明：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。
- 如果能找到E'的一个最优解B'，它包含比A'更多的活动，则将活动1加入到B'中将产生E的一个解B，它包含比A更多的活动。这与A的最优性矛盾。
- 结论：每一步所做的贪心选择问题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。

最优子结构性质

4.2 贪心算法的基本要素

- 对于一个具体的问题，如何知道是否可用贪心算法解决此问题，以及能否得到问题的最优解？这个问题很难给予肯定的回答。
- 但是，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两个重要的性质：贪心选择性质和最优子结构性质。

1. 贪心选择性质

- 贪心选择性质：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，每步所作的选择依赖于相关子问题的解。贪心算法仅在当前状态下作出局部最优选择，再去解作出这个选择后产生的相应的子问题。

分析

- 对于一个具体问题，要确定它是否具有贪心选择的性质，我们必须证明每一步所作的贪心选择最终能够导致问题的最优解。
- 通常可以首先证明问题的一个整体最优解是从贪心选择开始的，而且作了贪心选择后，原问题简化为一个规模更小的类似子问题。然后，用数学归纳法证明，通过每一步作贪心选择，最终可得到问题的一个整体最优解。
- 其中，证明贪心选择后的问题简化为规模更小的类似子问题的关键在于利用该问题的最优子结构性质。

2. 最优子结构性质

- 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。

3. 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？

回顾：0-1背包问题（动态规划）

- 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?
- 在选择装入背包中的物品时，对每种物品i只有两种选择：即装入或不装入背包。不能将物品i装入背包多次，也不能只装入部分的物品i。因此，该问题称为0-1背包问题。

问题的形式化描述

- 此问题的形式化描述为，给定c>0,w_i>0,v_i>0,1≤i≤n，要求找出一个n元0-1向量 (x_1,x_2, .. ,x_n)，其中x_i∈{0,1}，使得对w_ix_i求和小于等于c，并且对v_ix_i求和达到最大。
- 因此，0-1背包问题是一个特殊的整数规划问题。

$$\max \sum_{i=1}^{n} v_i x_i$$

$$\begin{cases} \sum_{i=1}^{n} w_i x_i \le c \\ x_i \in \{0,1\}, \quad 1 \le i \le n \end{cases}$$

0-1背包问题的子问题递归关系

- 设所给0-1背包问题的子问题：

$$\max \sum_{k=i}^{n} v_k x_k$$

$$\begin{cases} \sum_{k=1}^{n} w_k x_k \le j \\ x_k \in \{0,1\}, \quad i \le k \le n \end{cases}$$

m(i,j)是背包容量为j，可选择物品为i,i+1,…,n时0-1背包问题的最优值。由于0-1背包问题的最优子结构性质，可以建立计算m(i,j)的递归式如下：

$$m(i,j)=\begin{cases} \max\{m(i+1,j), m(i+1,j-w_i)+v_i\} & j \ge w_i \\ m(i+1,j) & 0 \le j \le w_i \end{cases}$$

$$m(n,j)=\begin{cases} v_n & j \ge w_n \\ 0 & 0 \le j \le w_n \end{cases}$$

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。
- 此问题的形式化描述为，给定$c>0, w_i>0, v_i>0, 1\le i\le n$，要求找出一个n元0-1向量$(x_1, x_2, \ldots, x_n)$，其中$0\le x_i\le 1$，$1\le i\le n$，使得对$w_i x_i$求和小于等于c，并且对$v_i x_i$求和达到最大。

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。

举例

- 有3种物品，背包的容量为50千克。物品1重10千克，价值60元；物品2重20千克，价值100元；物品3重30千克，价值120元。
- 用贪心算法求背包问题。

背包

有3种物品，背包的容量为50千克。物品1重10千克，价值60元；物品2重20千克，价值100元；物品3重30千克，价值120元。

- 贪心策略：物品1，6元/千克；物品2，5元/千克；物品3，4元/千克。

物品3，20kg	80	
	+	
物品2，20kg	100	=￥240
	+	
物品1，10kg	60	

具体算法

```
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];
}
```

该算法前提：

所有物品在集合中按其单位重量的价值从小到大排列。

- 算法Knapspack的主要计算时间在于将各种物品按其单位重量的价值从小到大排序，算法的时间复杂度O(nlogn)。
- 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。

- 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。

4.4 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。
- 哈夫曼编码算法是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。
- 编码目标：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。

举例

- 一个数据文件包含100000个字符，要用压缩的方式来存储它。该文件中各字符出现的频率如表所示。

	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

使用变长码要比使用定长码好得多。通过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。

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。

举例

可唯一地分解为0,0,101,1101，因而其译码为aabe。

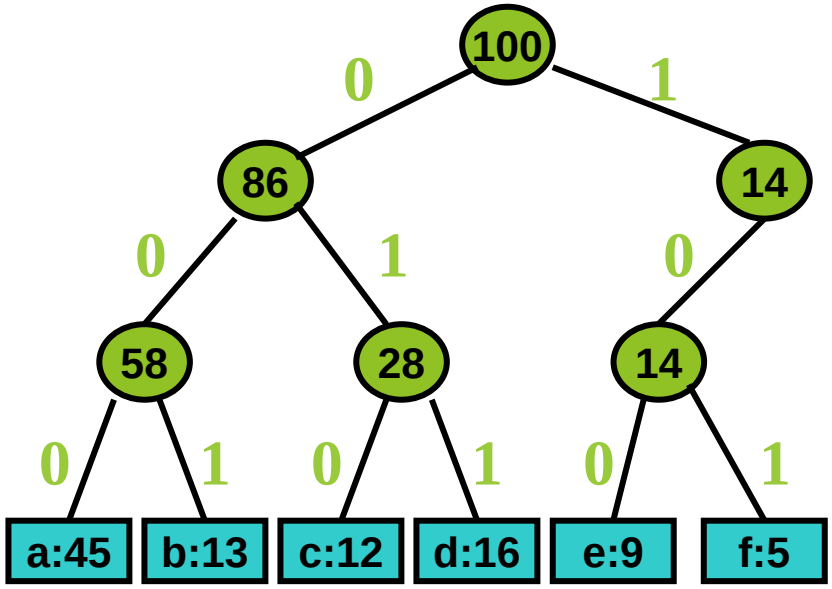
- 给定序列：001011101。

	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

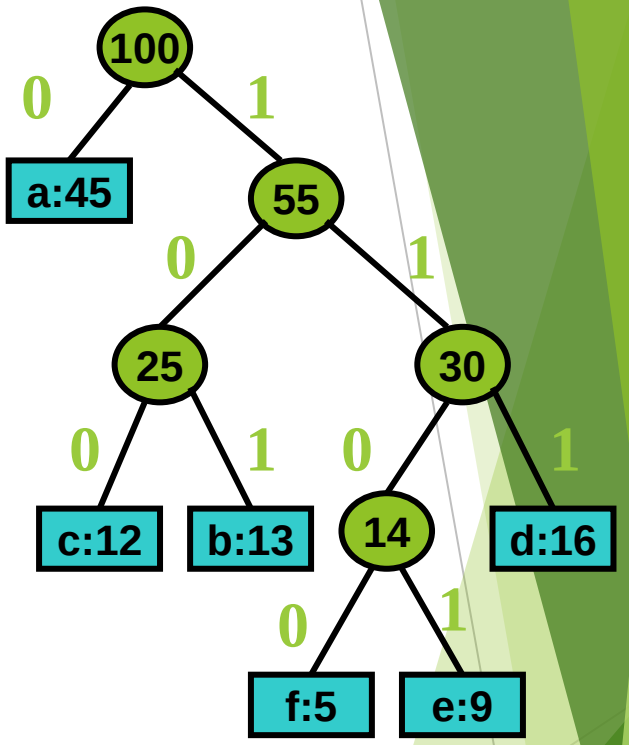
问题分析

- 译码过程需要方便地取出编码的前缀，因此需要一个表示前缀码的合适的数据结构。
- 用二叉树作为前缀编码的数据结构。在表示前缀码的二叉树中，树叶代表给定的字符，并将每个字符的前缀码看作是从树根到代表该字符的树叶的一条道路。代码中每一位的0或1分别作为指示某结点到其左儿子或右儿子的“路标”。

前缀码的二叉树表示



定长码



变长码

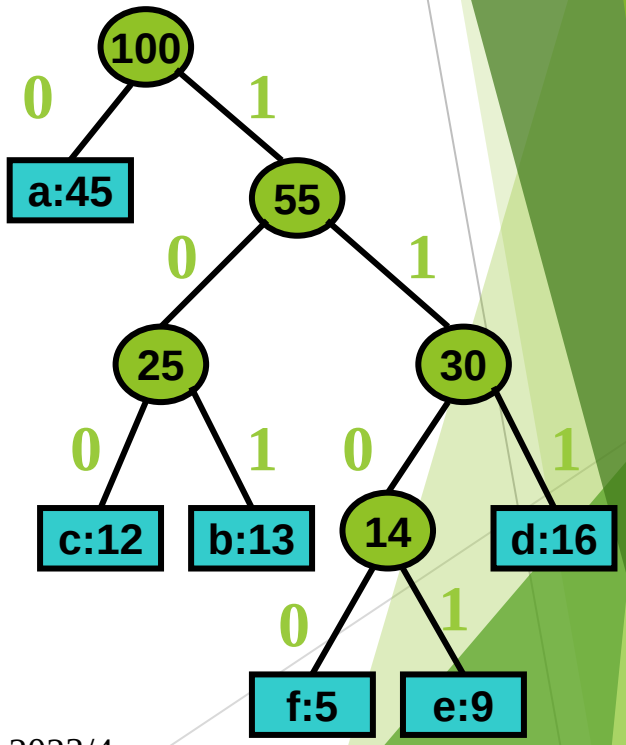
该编码方案的平均码长定义为： $B(T)=\sum_{c\in C} f(c)d_T(c)$

2. 构造哈夫曼编码

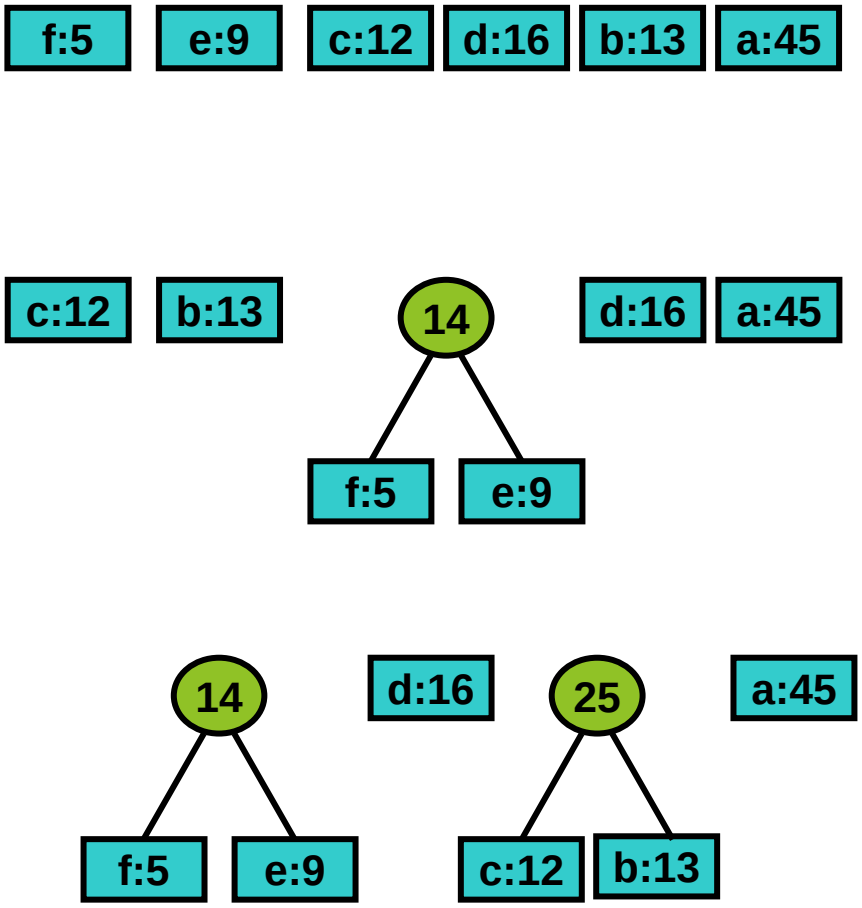
- 哈夫曼算法以自底向上的方式构造表示最优前缀码的二叉树T。
- 编码字符集中每一字符c的频率是$f(c)$。以f为键值的优先队列Q用以在作贪心选择时有效地确定算法当前要合并的两棵具有最小频率的树。一旦两棵具有最小频率的树合并后，产生一棵新的树，其频率为合并的两棵树的频率之和，并将新树插入优先队列Q中，再进行新的合并。

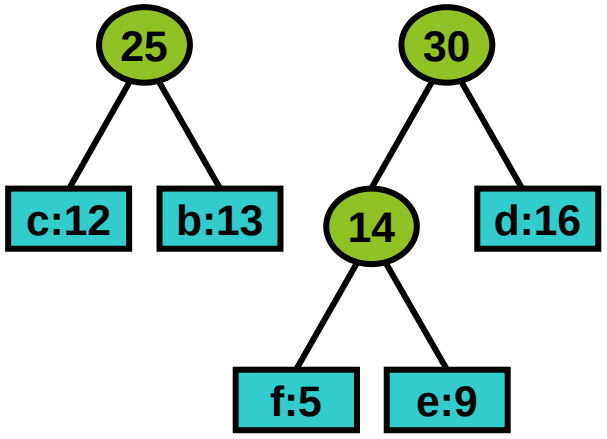
举例

	a	b	c	d	e	f
频率(千次)	45	13	12	16	9	5

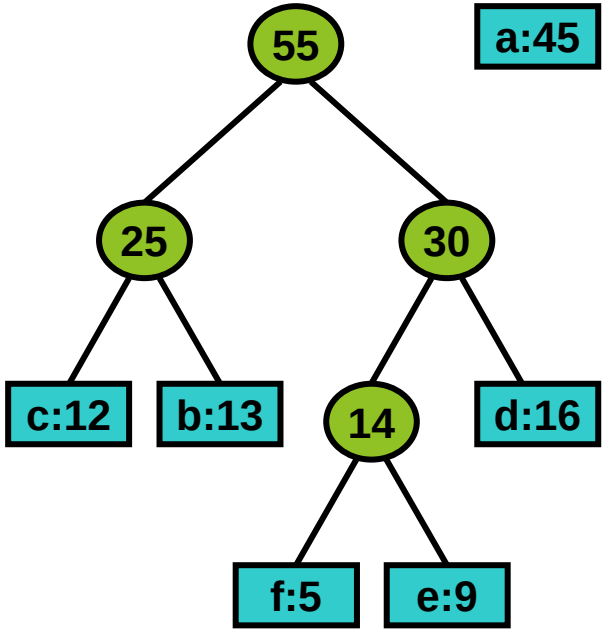


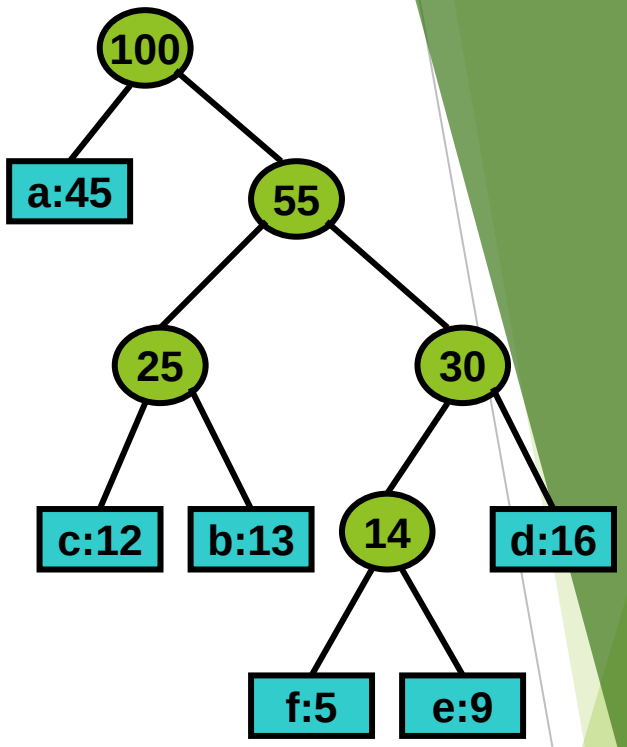
- 哈夫曼算法的执行过程示例：





a:45





- 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

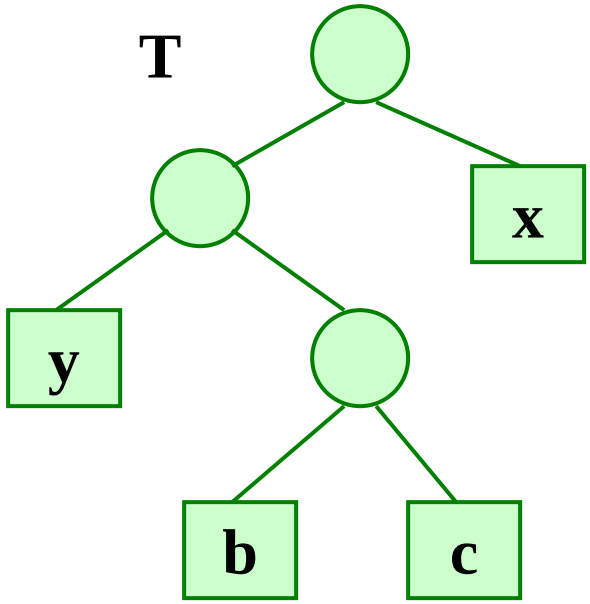
引理：设C为一字母表，其中每个字符c具有频度f[c]。设x和y为C中具有最低频度的两个字符，则存在C的一种最优前缀编码，其中x和y的编码长度相同但最后一位不同。

证明：根据构造过程证明，交换任意一个节点与叶子节点不增加平均码长，说明最优。

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：

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

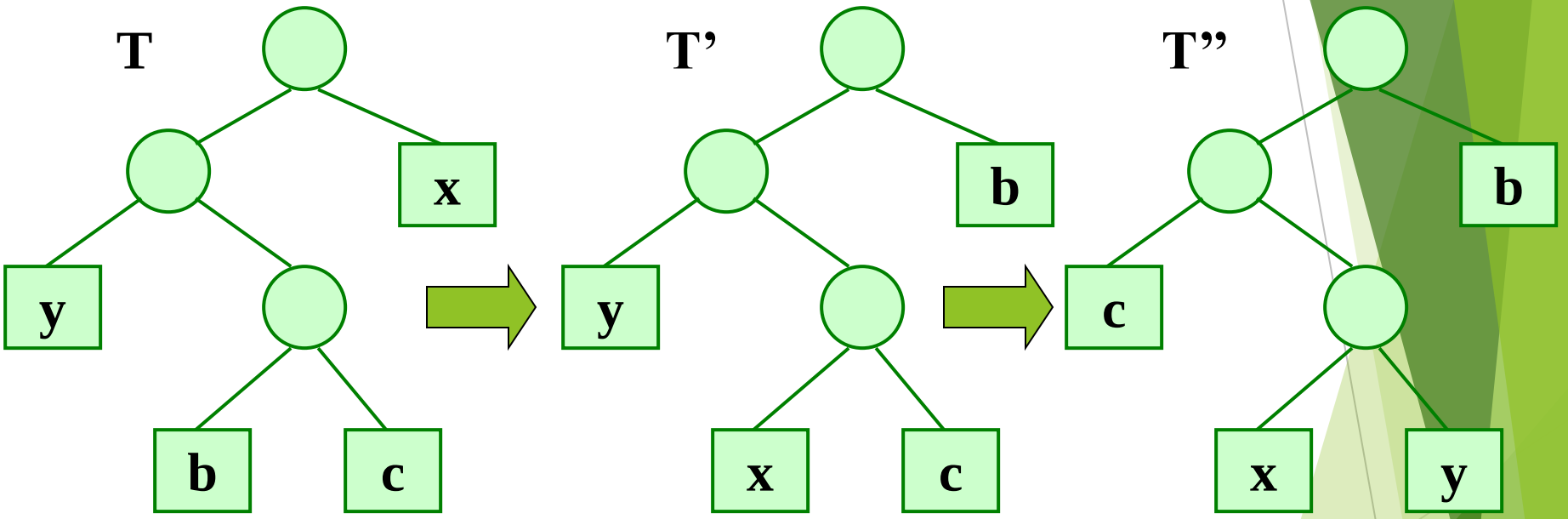


设，二叉树T表示C的任意一个最优前缀码。

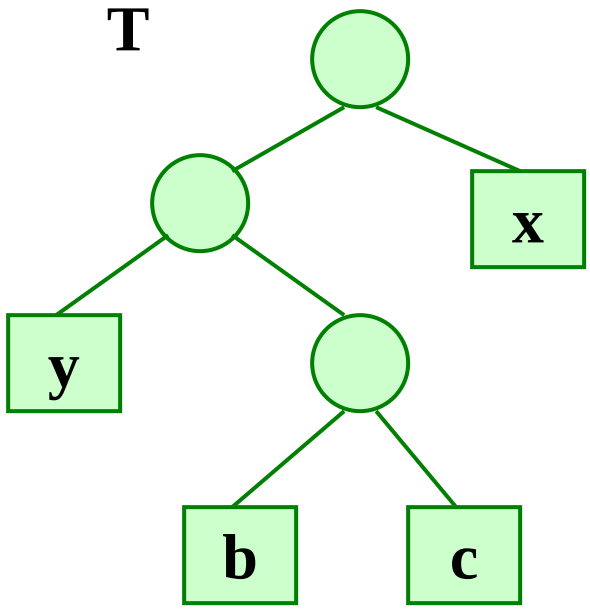
需证明，可以对T做适当修改后得到一棵新的二叉树T”，使在新树中，x和y是最深叶子且为兄弟。同时新树T”表示的前缀码也是C的最优前缀码。

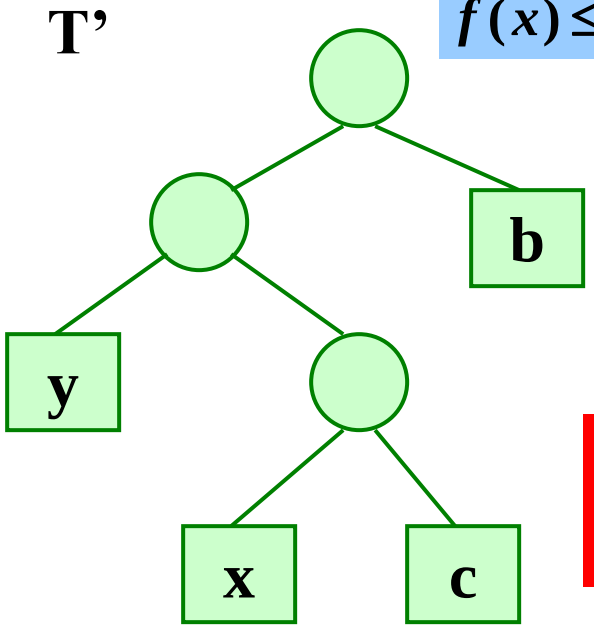
$$f(x) \le f(b), \quad f(y) \le f(c)$$

- 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”。**



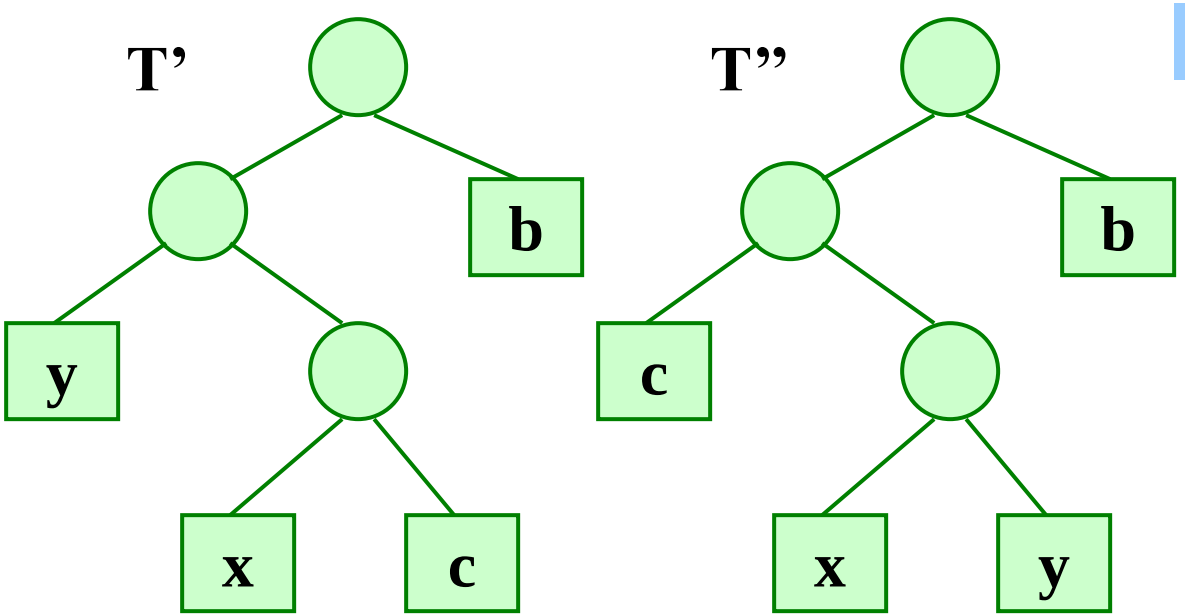


$f(x) \le f(b), \quad f(y) \le f(c)$

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：

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

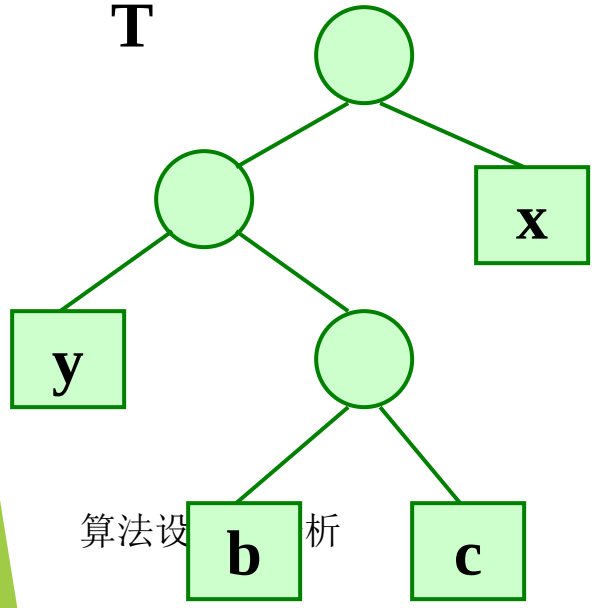
- 由此可知：

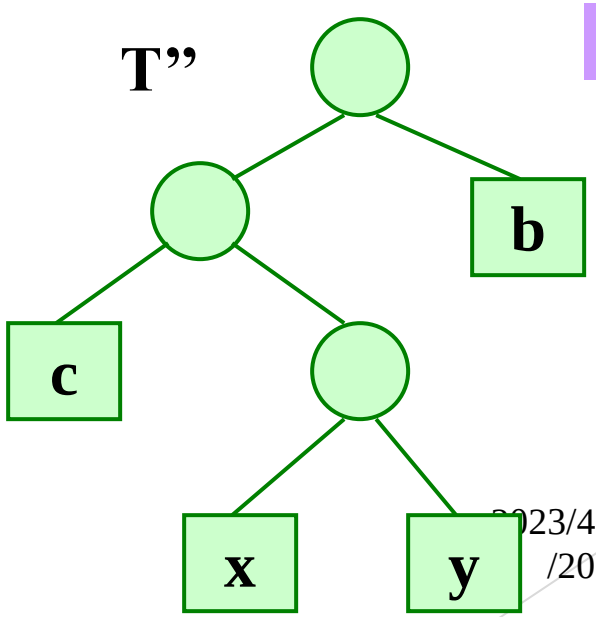
$$B(T) \ge B(T') \ge B(T'')$$

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

- 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’)：

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

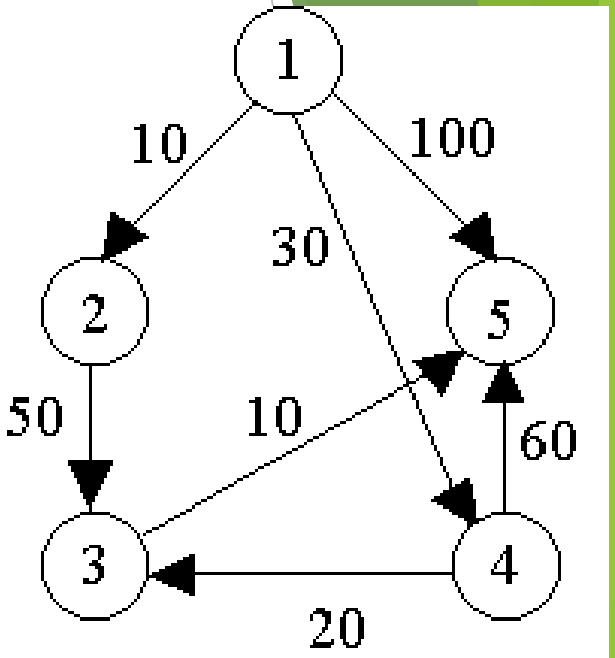
所以根据此式：B(T)=B(T’) + f[x] + f[y]

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。

4.5 单源最短路径

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。

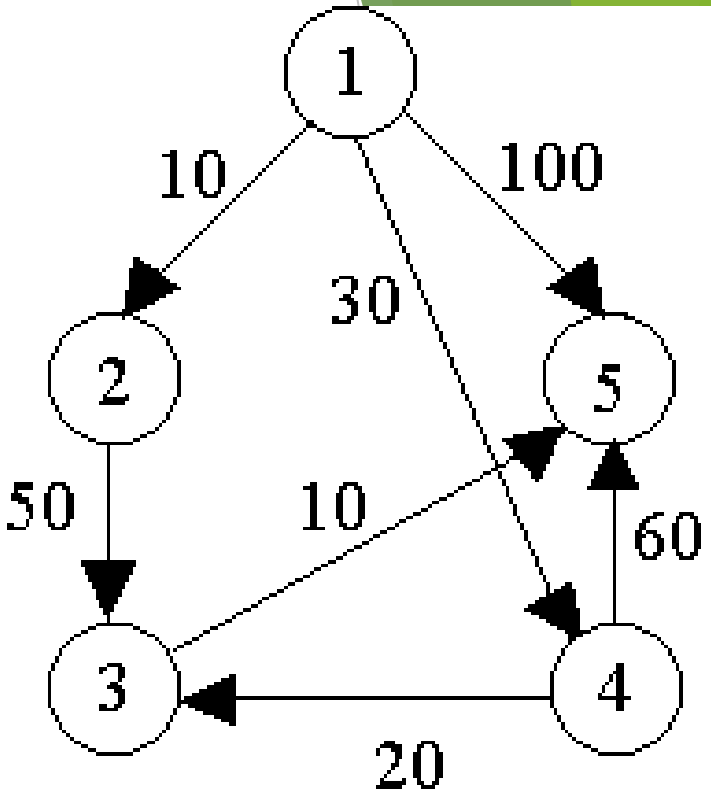
- 给定带权有向图G=(V,E)，其中每条边的权是非负实数。
- 给定V中的一个顶点，称为源。
- 现在要计算从源到其他所有各顶点的最短路径长度。这里的路径长度是指路径上各边权之和，这个问题通常称为单源最短路径问题。



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。

- 实际上，下一条最短路径总是由已产生的最短路径再扩充一条最短的边得到的，且这条路径所到达的顶点其最短路径还未产生。
- 例如：右图。



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。

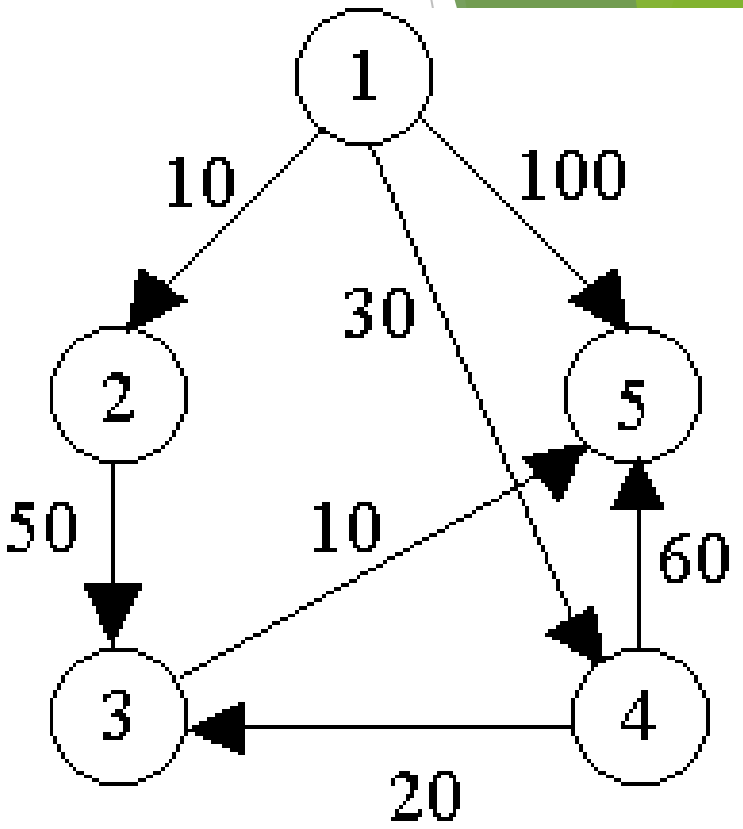
举例

- 已知：带权有向图

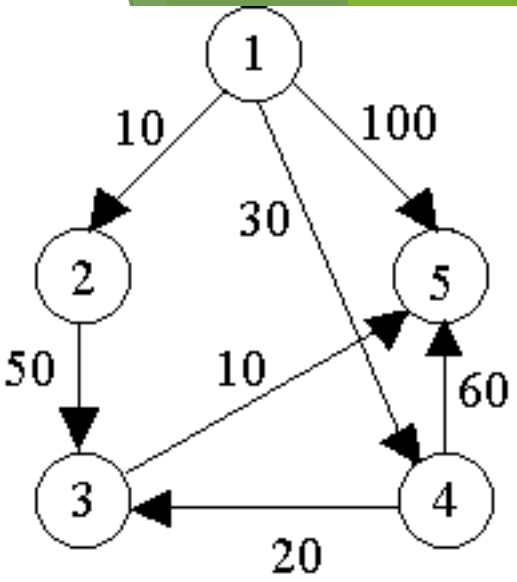
$V = \{ v_1, v_2, v_3, v_4, v_5 \}$

$E = \{ < v_1, v_2 >, < v_1, v_4 >, < v_1, v_5 >, < v_2, v_3 >, < v_3, v_5 >, < v_4, v_3 >, < v_4, v_5 > \}$

- 设为v_1源点，求其到其余顶点的最短路径。

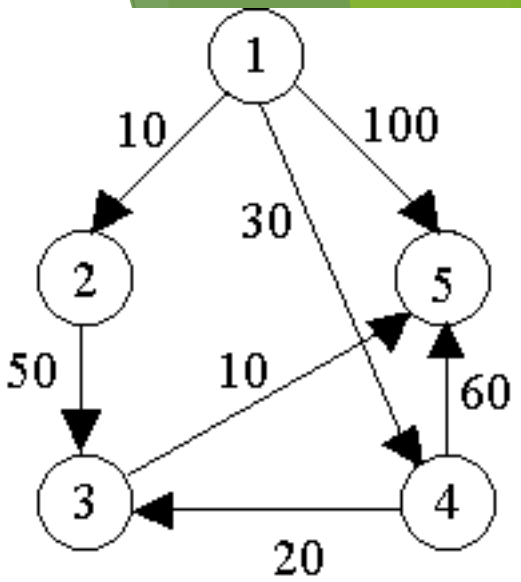


- 初始时，集合S只有源点v_1，即加入集合S中的顶点u为v_1。
- 从源点v_1到其它顶点的最短特殊路径（中间只有来自于集合S中的顶点）长度分别为：
  - dist[2]=10; dist[3]=maxint; dist[4]=30; dist[5]=100.
  - 其中，没有特殊路径的顶点v_3用maxint表示其最短特殊路径长度。



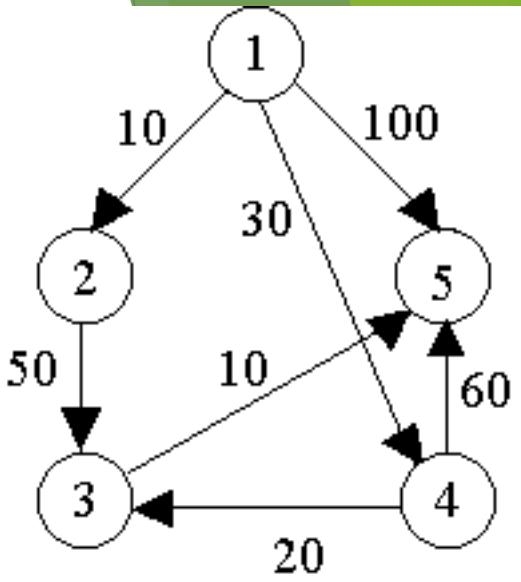
迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100

- 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。



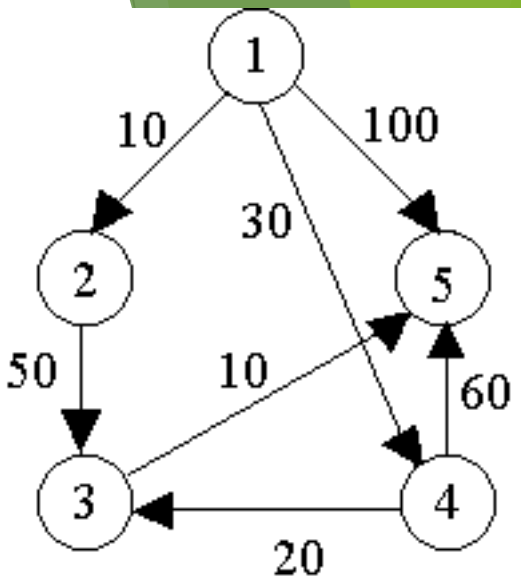
迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1,2}	2	10	60	30	100

- 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

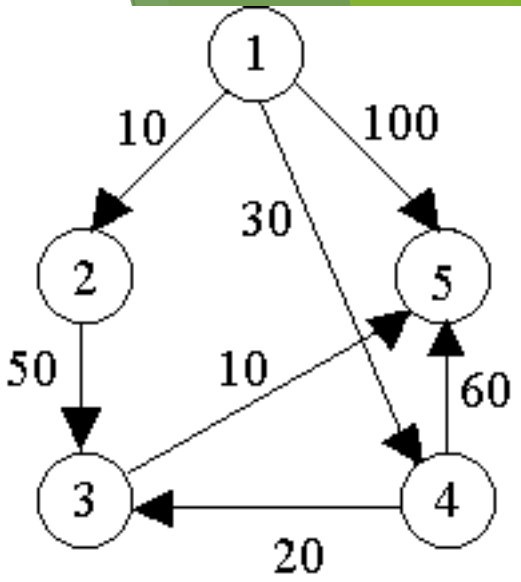
- 集合S为{v_1,v_2,v_4}，其余顶点的最短特殊路径长度已确定。
- 其中dist[3]的值最小，为50，所以将顶点v_3加入集合S中。
- 由于集合S为{v_1,v_2,v_4,v_3}，需要修改剩余的一个顶点的最短特殊路径值。
- 例如，v_5的最短特殊路径为 <v_1,v_2> ,<v_2,v_3> ,<v_3,v_5>；长度为60。



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

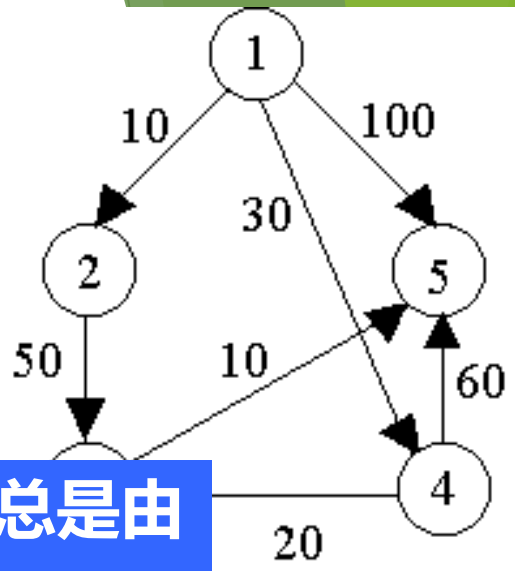
- 集合S为{v_1,v_2,v_4,v_3}，其余顶点的最短特殊路径长度已确定。
- 由于只剩余一个顶点v_5不在集合中，所以应该把它加入集合。
- 此时集合S为{v_1,v_2,v_4,v_3,v_5}=V，完成。

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。

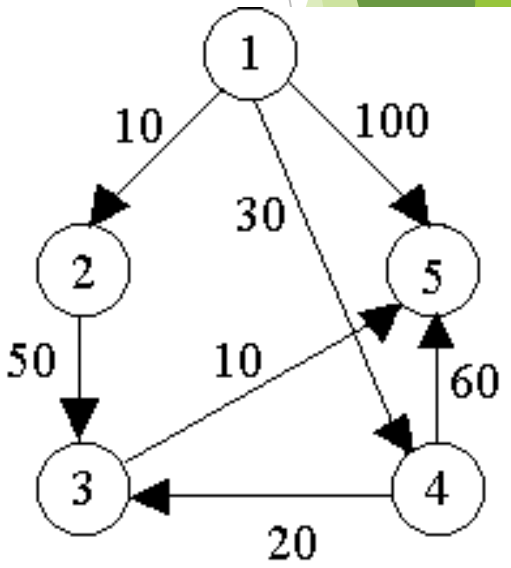


按长度顺序产生最短路径时，下一条最短路径总是由一条已产生的最短路径加上一条边形成。

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

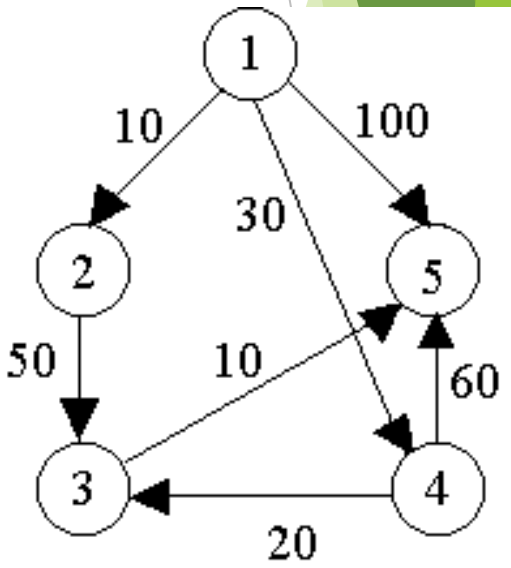
迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

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

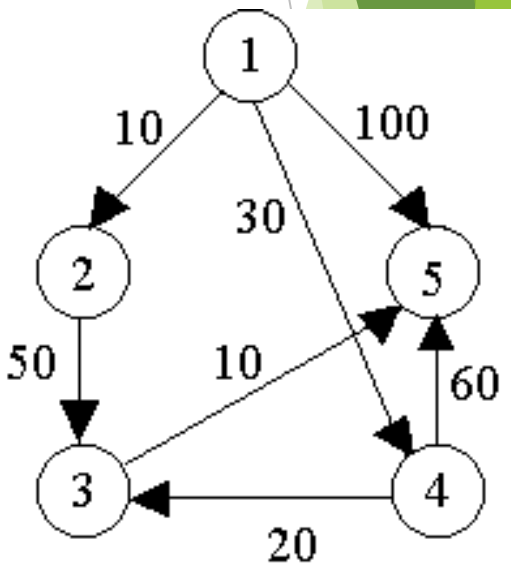
2. 扫描非S集中Dist[]值最小的节点Dist[u]，也就是找出下一条最短路径，把节点u加入S集中。



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

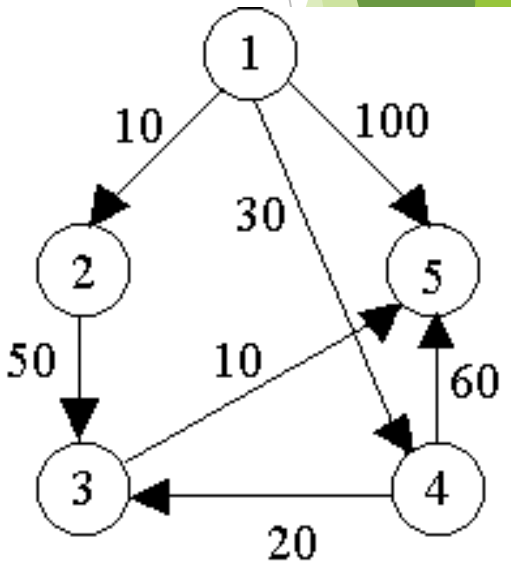
3. 更新所有非S集中的Dist[]值，看看是否可通过新加入的u点让其路径更短：

```
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

- 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

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

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

2. 最优子结构性质

- 证明最优子结构性质，即算法中确定的dist[u]确实是当前从源到顶点u的最短特殊路径长度。

• 为此，只要考察算法在添加u到S中后，dist[u]的值所起的变化就行了。

• 不论算法中dist[u]的值是否有变化，它总是关于当前顶点集S到顶点u的最短特殊路径长度。

4.6 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。

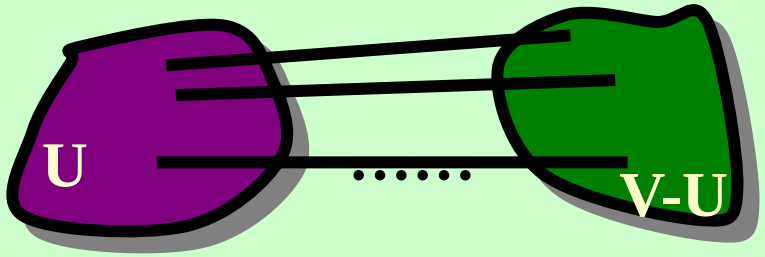
贪心法求解准则

- 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

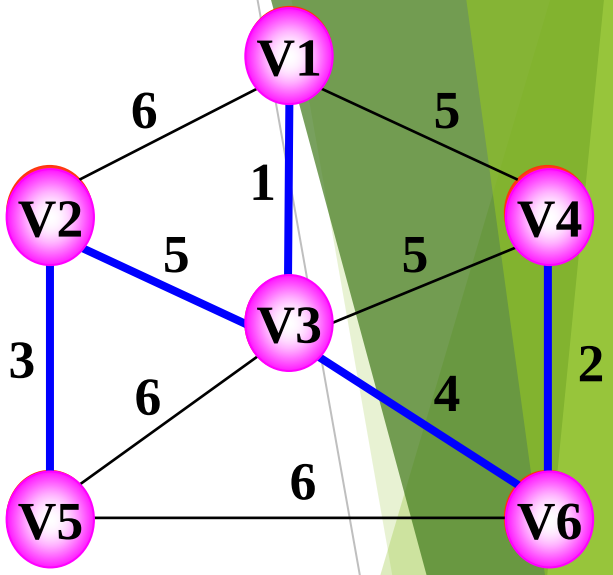
1. 在图G=(V, E)（V表示顶点集合，E表示边集合）中，从集合V中任取一个顶点（例如取顶点v_1）放入集合U中，这时U=$\{v_1\}$，生成树边集合T(E)为空。
2. 寻找与S中顶点相邻（另一顶点在V-U中）权值最小的边的另一顶点v_2，并使v_2加入S。即U=$\{v_1, v_2\}$，同时将该边加入集合T(E)中。
3. 重复2，直到U=V为止。

- 这时T(E)中有n-1条边，T=(U,T(E))就是一棵最小生成树。

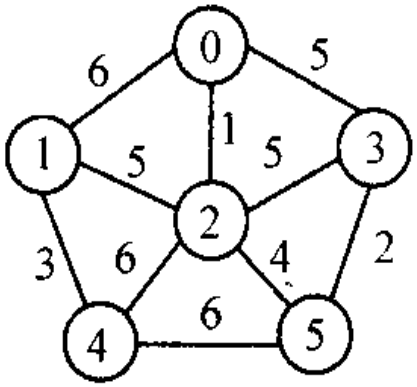
prim算法

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。



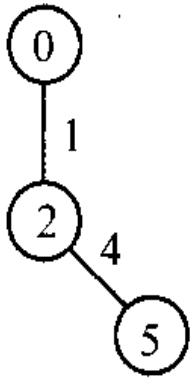
Prim算法举例



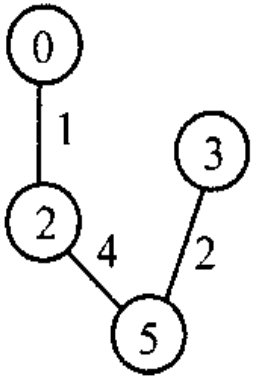
（a）无向图G

（b）只有源点

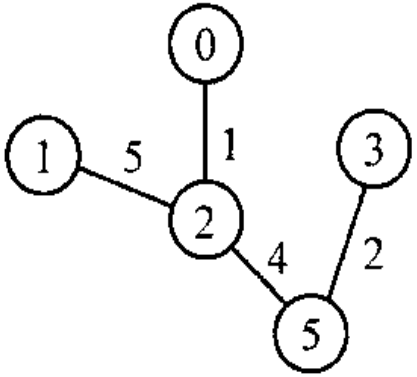
（c）加入第1条边



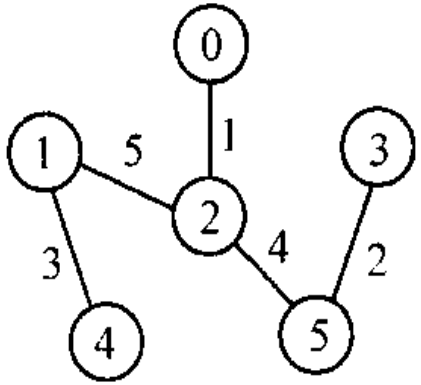
（d）加入第2条边



（e）加入第3条边



（f）加入第4条边

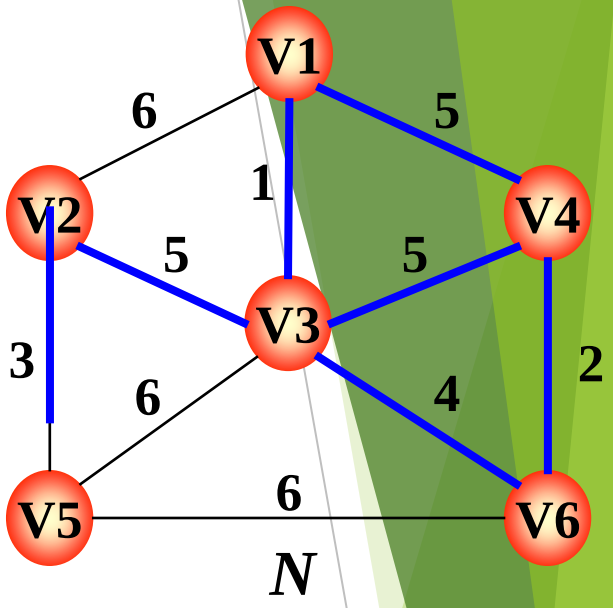


（g）图G的最小代价生成树

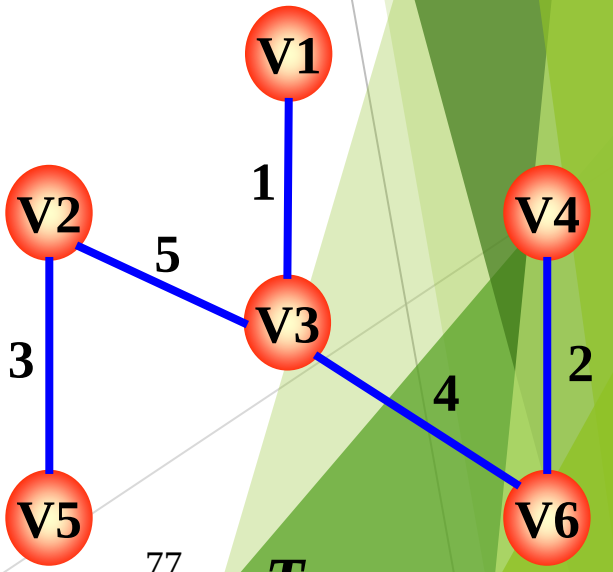
Kruskal算法

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。

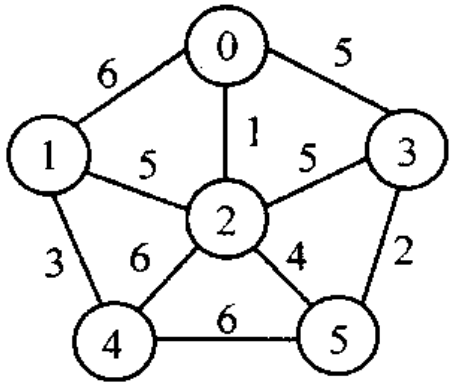


N

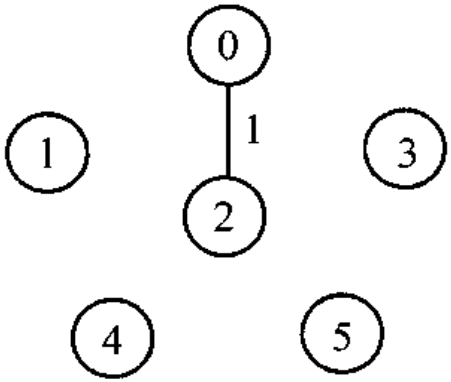


T

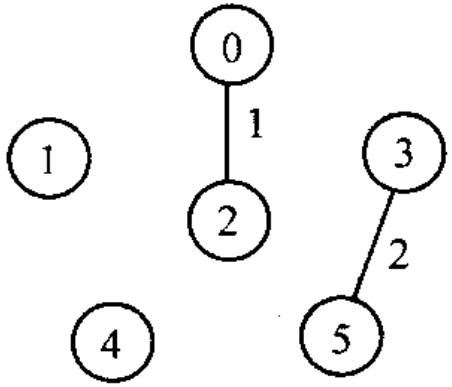
Kruskal算法举例



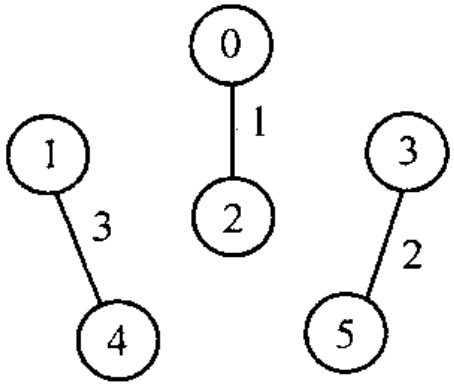
（a）无向图G



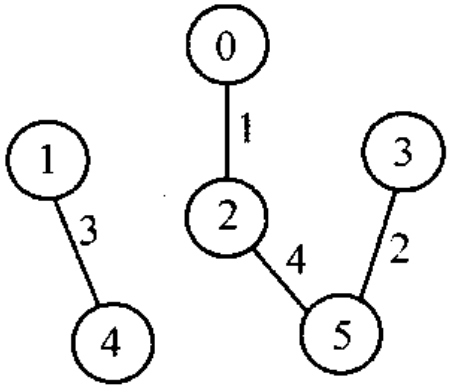
（b）加入第1条边



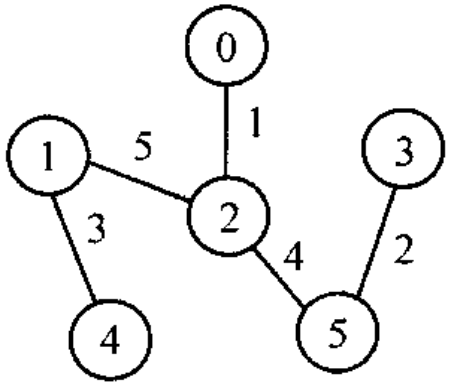
（c）加入第2条边



（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。

课后练习

- 算法分析题4-6（教材第128页）：字符a~h出现的频率恰好是前8个Fibonacci数，它们的哈夫曼编码是什么？
- 练习2：假设有25分、10分、5分和1分四种硬币，需要找给顾客2元5角钱，请问用贪心算法可以求出何种找零方案？该方案是最优的么？为什么？

4.7 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。

4.7 多机调度问题

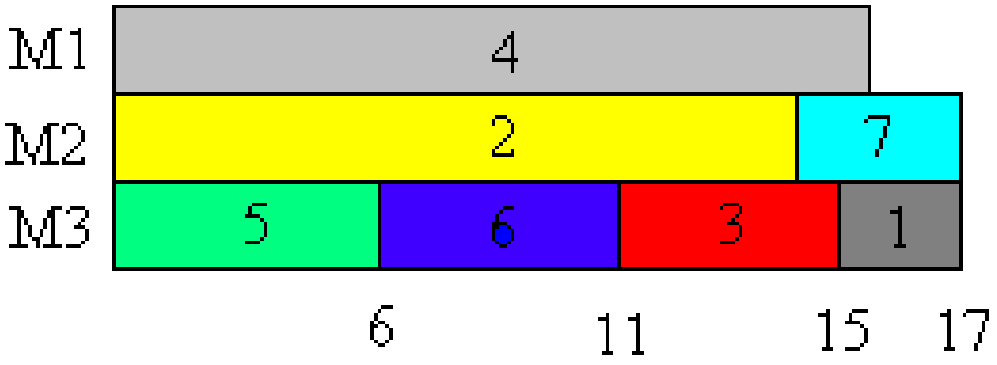
采用**最长处理时间作业优先**的贪心选择策略可以设计出解多机调度问题的较好的近似算法。

按此策略，当 $n \le m$ 时，只要将机器i的[0, ti]时间区间分配给作业i即可，算法只需要O(1)时间。

当 $m < n$ 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(nlogn)。

4.7 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M1，M2和M3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。



课后练习

- 练习3：活动安排问题。假设有9个活动申请使用1个会议室，每个活动的开始时间和终止时间如下。用贪心算法设计一个活动安排表，要求要尽可能的多安排活动。会议室不允许被多个活动同时占用。

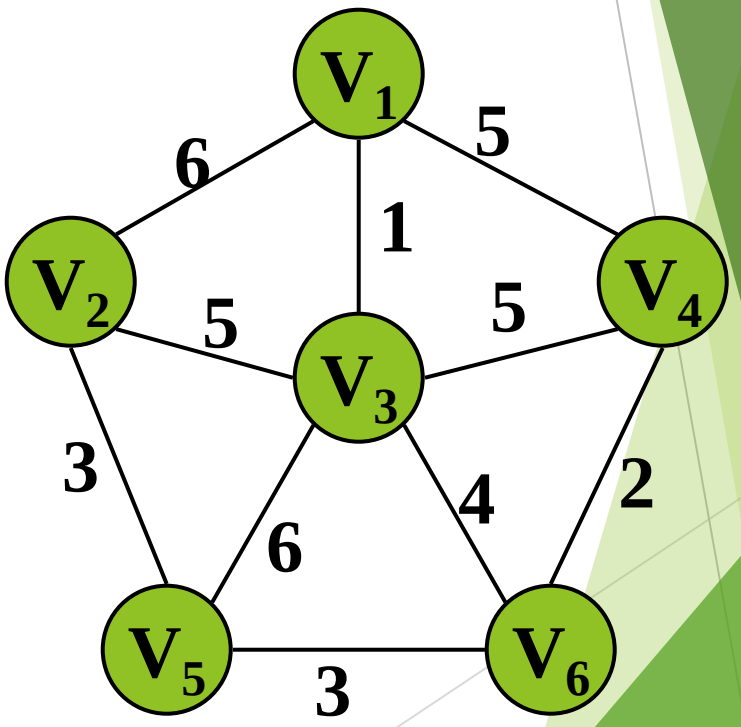
活动序号	1	2	3	4	5	6	7	8	9
起始时间	2	1	2	5	7	4	6	8	15
结束时间	5	5	8	10	11	13	15	22	24

课后练习

练习5：最小生成树问题。

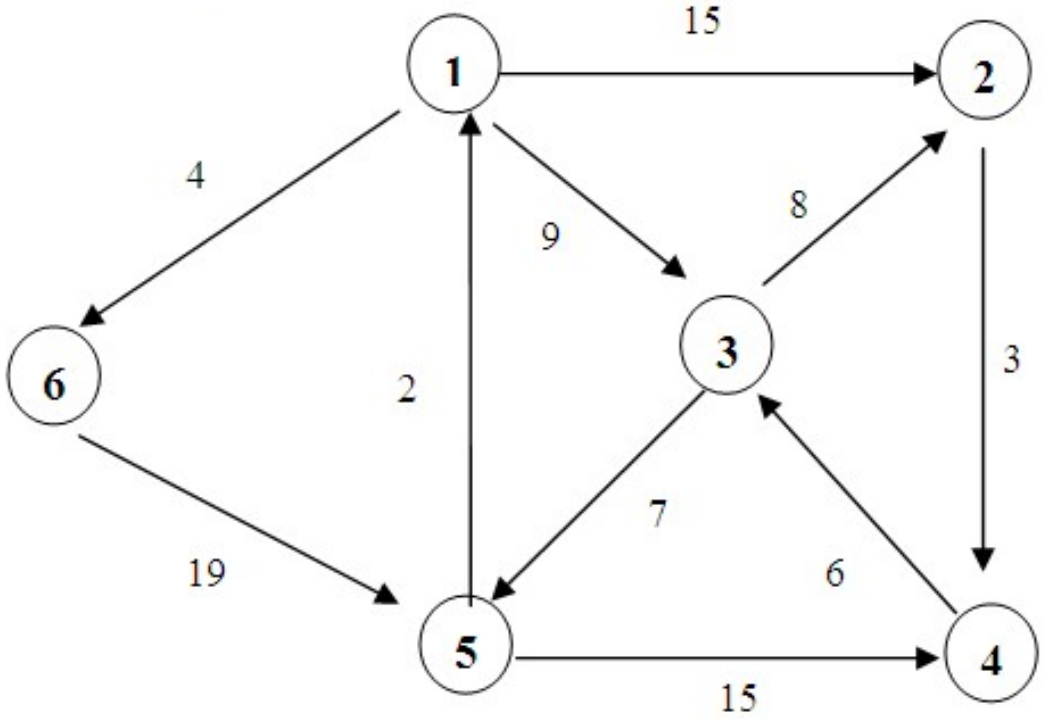
① 用Prim或者Kruskal算法求出下图中的最小生成树。

② 证明该算法的正确性。



- 练习6：一队徒步旅行者要从A地到B地，两地间距离为L公里。他们每天最多可以走d公里（与地形、天气条件等无关），中间需要就地宿营。
- 假定这些潜在的宿营地点位于起点A地的距离为x_1, x_2, ……, x_n的地方，显然它们之间的距离小于等于d，称这些地点（停止点）是有效的。
- 于是，一组停止点是有效的，如果人们只能在这些地方宿营并且仍旧能顺利完成旅行。则必须假设n个停止点所组成的集合都是有效的；否则就没法走完整个路程。
- 问题1：假设两地距离为100公里，潜在的宿营地点为{ 6, 14, 30, 37, 48, 65, 73, 76, 88, 90,94 }。旅行者每天最多走18公里。给出最优（最少）的有效宿营地组合（最少几天能走完）。
- 问题2：证明该算法的正确性（解的最优）。

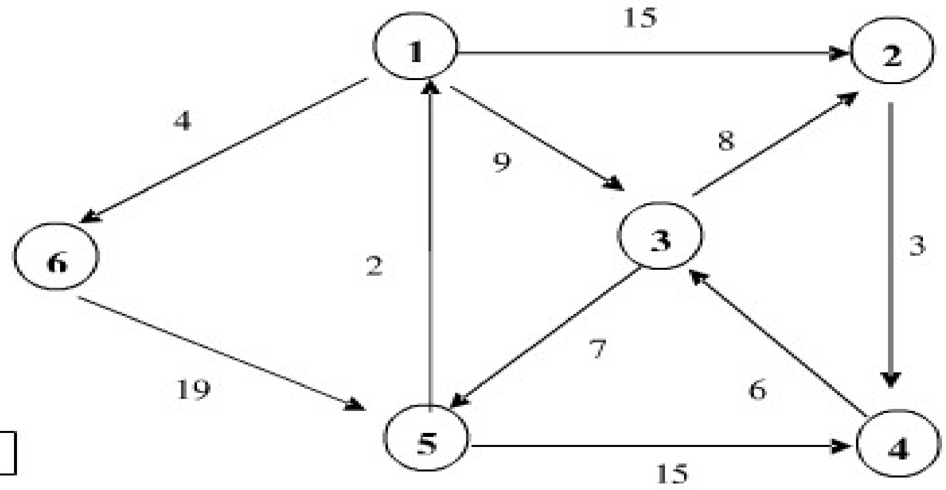
2．对下图所给的带权有向图执行 dijkstra 算法，求顶点 v1 到其余顶点的最短路径，试写出算法执行过程中辅助数组 dist 和 path 的变化情况，并写出最短路径结果。



带权有向图

1												
2												
3												
4												
5												
6												

2．对下图所给的带权有向图执行 dijkstra 算法，求顶点 v1 到其余顶点的最短路径，试写出算法执行过程中辅助数组 dist 和 path 的变化情况，并写出最短路径结果。



带权有向图

path dist

	path	dist	path	dist	path	dist	path	dist	path	dist	path	dist
1	1	0	1	0	1	0	1	0	1	0	1	0
2	1	15	1	15	1	15	1	15	1	15	1	15
3	1	9	1	9	1	9	1	9	1	9	1	9
4	1	∞	1	∞	1	∞	2	18	2	18	2	18
5	1	∞	6	23	3	16	3	16	3	16	3	16
6	1	4	1	4	1	4	1	4	1	4	1	4
	1 进入第 1 组		1 进入第 1 组		1 进入第 1 组		1 进入第 1 组		1 进入第 1 组		1 进入第 1 组	

最短路径：

2←1 ： 15

3←1 ： 19

4←2←1 ： 18

5←3←1 ： 16

6←1 ： 4