第八章 贪心算法

- 本章主要知识点（8）：
  - 8.1 活动安排问题
  - 8.2 贪心算法的基本要素
  - 8.3 最优装载
  - 8.4 哈夫曼编码
  - 8.5 单源最短路径
  - 8.6 最小生成树
  - 8.7 多机调度问题

引言

- 找零钱问题
- 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。如单源最短路经问题，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。

8.1 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。
- 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。每个活动i都有一个要求使用该资源的起始时间si和一个结束时间fi，且si <fi 。如果选择了活动i，则它在半开时间区间[si, fi)内占用资源。若区间[si, fi)与区间[sj, fj)不相交，则称活动i与活动j是相容的。也就是说，当si≥fj或sj≥fi时，活动i与活动j相容。

贪心算法描述

- 在下面所给出的解活动安排问题的贪心算法greedySelector：

```
public static int greedySelector(int [] s, int [] f, boolean a[])
{
    int n=s.length-1;
    a[1]=true;
    int j=1;
    int count=1;
    for (int i=2;i<=n;i++) {
        if (s[i]>=f[j]) {
            a[i]=true;
            j=i;
            count++;
        }
        else a[i]=false;
    }
    return count;
}
```

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

复杂性分析

- 由于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。
- 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(nlogn)的时间重排。

一个实例

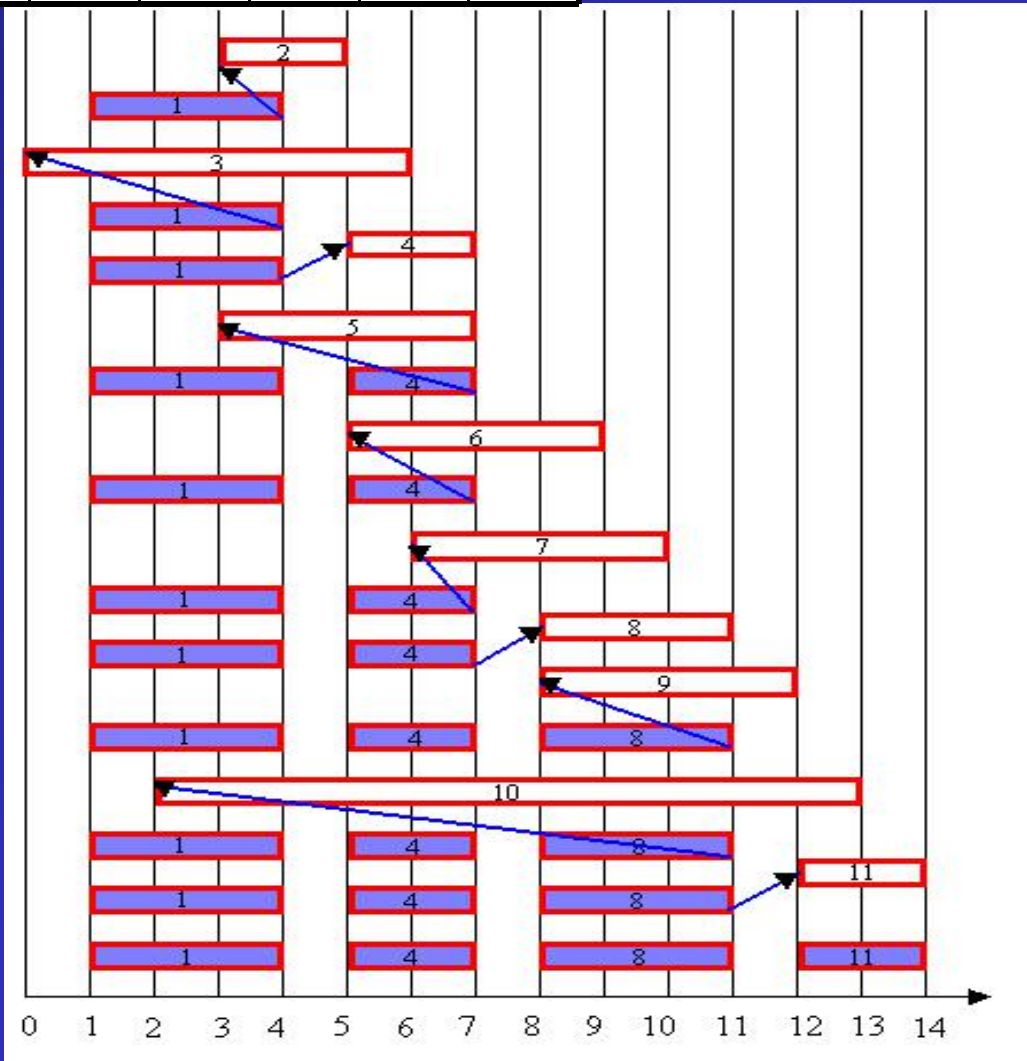
- 例：设待安排的11个活动的开始时间和结束时间按结束时间的非减序排列如下：

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

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

图示

- 算法greedySelector的计算过程如右图所示。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。



说明

- 若被检查的活动i的开始时间S_i小于最近选择的活动j的结束时间f_j，则不选择活动i，否则选择活动i加入集合A中。
- 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。

8.2 贪心算法的基本要素

- 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。
- 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

- 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。
- 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。

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*只有两种选择，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，$1 \le i \le n$。
- 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，而0-1背包问题却不能用贪心算法求解。

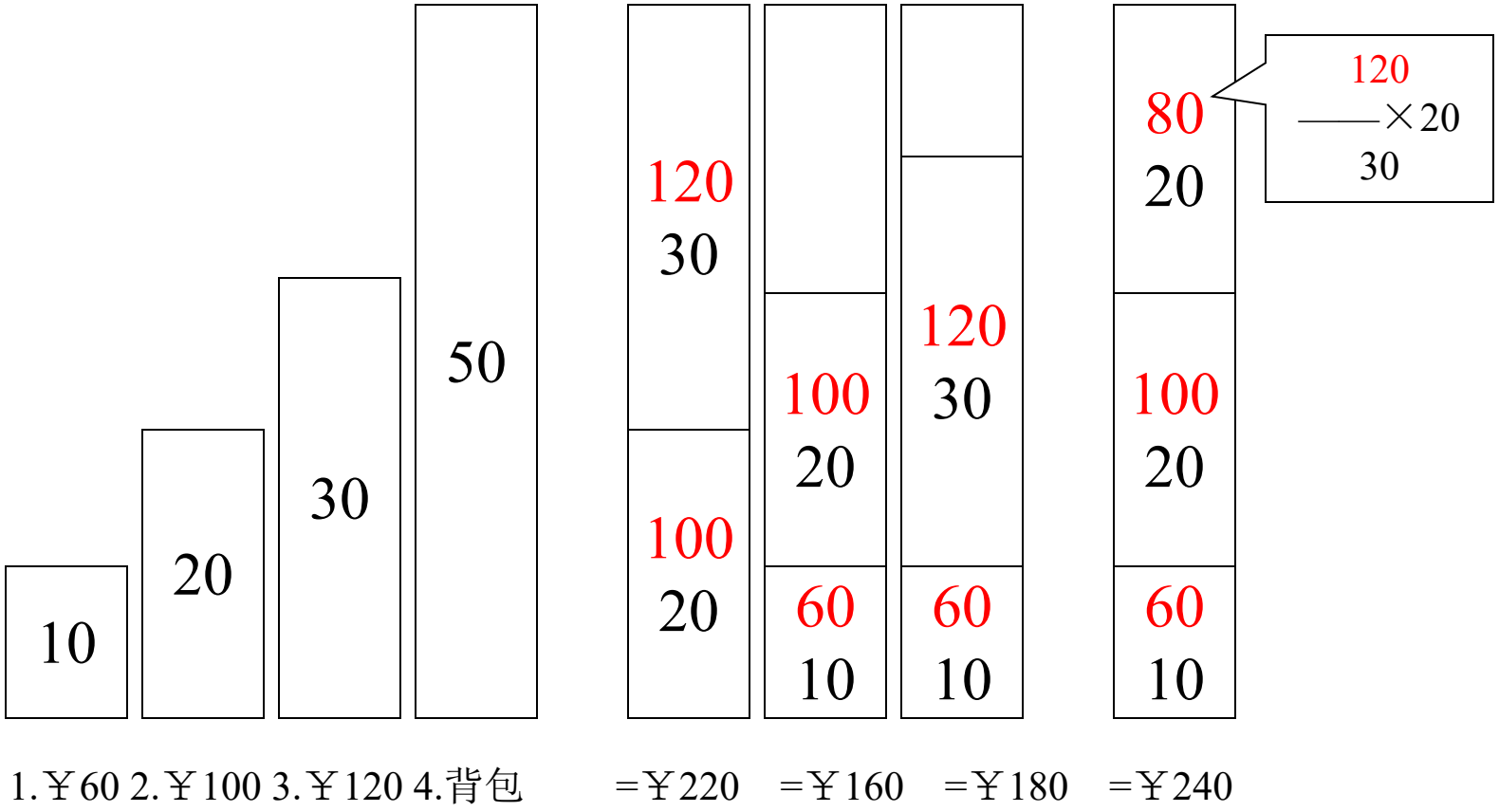
用贪心算法解背包问题的基本步骤

- 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(nlogn)。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。

说明



0-1背包问题的例子

8.3 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Wi。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。
- 改问题可形式化描述为

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

$$\sum_{i=1}^{n} w_i x_i \le c$$

$$x_i \in \{0,1\}, 1 \le i \le n$$

- 其中变量$x_i=0$表示不装入集装箱i，$x_i=1$表示装入集装箱i。

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。具体算法描述如下：

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

算法分析

- 贪心选择性质
  - 可以证明最优装载问题具有贪心选择性质。
- 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。
- 算法复杂性：
  - 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(nlogn)。

8.4 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。
- 例如一个包含100,000个字符的文件，各字符出现频率不同，如下表所示。定长变码需要300,000位，而按表中变长编码方案，文件的总码长为：
  （45×1+13×3+12×3+16×3+9×4+5×4）×1000=224,000。
- 比用定长码方案总码长较少约45%。

	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

1.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它字符代码的前缀。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。
- 表示最优前缀码的二叉树总是一棵完全二叉树，即树中任一结点都有2个孩子结点。
- 平均码长定义为：$$B(T)=\sum_{c\in C} f(c)d_T(c)$$
- 使平均码长达到最小的前缀码编码方案称为给定编码字符集C的最优前缀码。

2.构造哈夫曼编码

- 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。

算法说明

- huffmanTree算法中，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。
- 算法huffmanTree用最小堆实现优先队列Q。初始化优先队列需要O(n)计算时间，由于最小堆的removeMin和put运算均需O(logn)时间，n－1次的合并总共需要O(nlogn)计算时间。因此，关于n个字符的哈夫曼算法的计算时间为O(nlogn) 。

8.5 单源最短路径

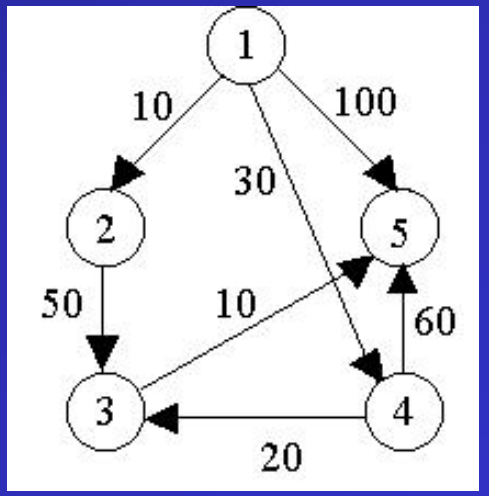
- 给定带权有向图G =(V,E)，其中每条边的权是非负实数。另外，还给定V中的一个顶点，称为源。现在要计算从源到所有其它各顶点的最短路长度。这里路的长度是指路上各边权之和。这个问题通常称为单源点最短路径问题。

1、算法基本思想

- Dijkstra算法是解单源点最短路径问题的贪心算法。
- 基本思想：设置顶点集合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。
- 初始时，S中仅含有源。设u是G的某一个顶点，把从源点到u且中间只经过S中顶点的路称为从源到u的特殊路径，并用数组dist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它顶点之间的最短路径长度。
- 算法描述

一个实例

- 例如，对右图中的有向图，应用Dijkstra算法计算从源顶点1到其它顶点间最短路径的过程列在下页的表中。
- Dijkstra算法的迭代过程：



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

2、算法的正确性和计算复杂性

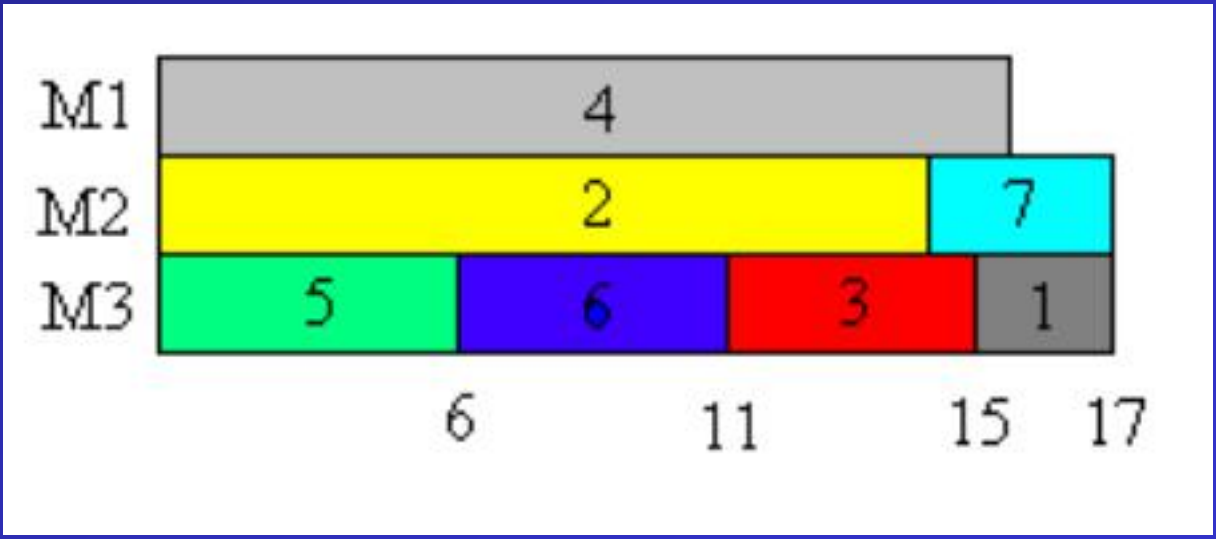
- 下面我们首先来讨论Dijkstra算法的正确性：
  - (1)贪心选择性质
  - (2)最优子结构性质
- 计算复杂性：
  - 对于具有n个顶点和e条边的带权有向图，如果用带权邻接矩阵表示这个图，那么Dijkstra算法的主循环体需要$O(n)$时间。这个循环需要执行n-1次，所以完成循环需要O(n^2)时间。算法的其余部分所需要时间不超过$O(n^2)$。

8.7 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。
- 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$n > m$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(*nlogn*)。

一个实例

- 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。



8.6 最小生成树

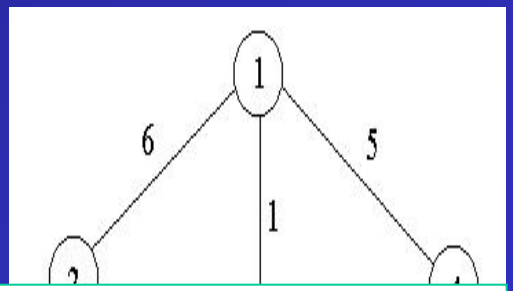
- 设G =(V,E)是无向连通带权图，即一个网络。E中每条边(v,w)的权为c[v][w]。如果G的子图G’是一棵包含G的所有顶点的树，则称G’为G的生成树。生成树上各边权的总和称为该生成树的耗费。在G的所有生成树中，耗费最小的生成树称为G的最小生成树。
- 网络的最小生成树在实际中有广泛应用。例如，在设计通信网络时，用图的顶点表示城市，用边(v,w)的权c[v][w]表示建立城市v和城市w之间的通信线路所需的费用，则最小生成树就给出了建立通信网络的最经济的方案。

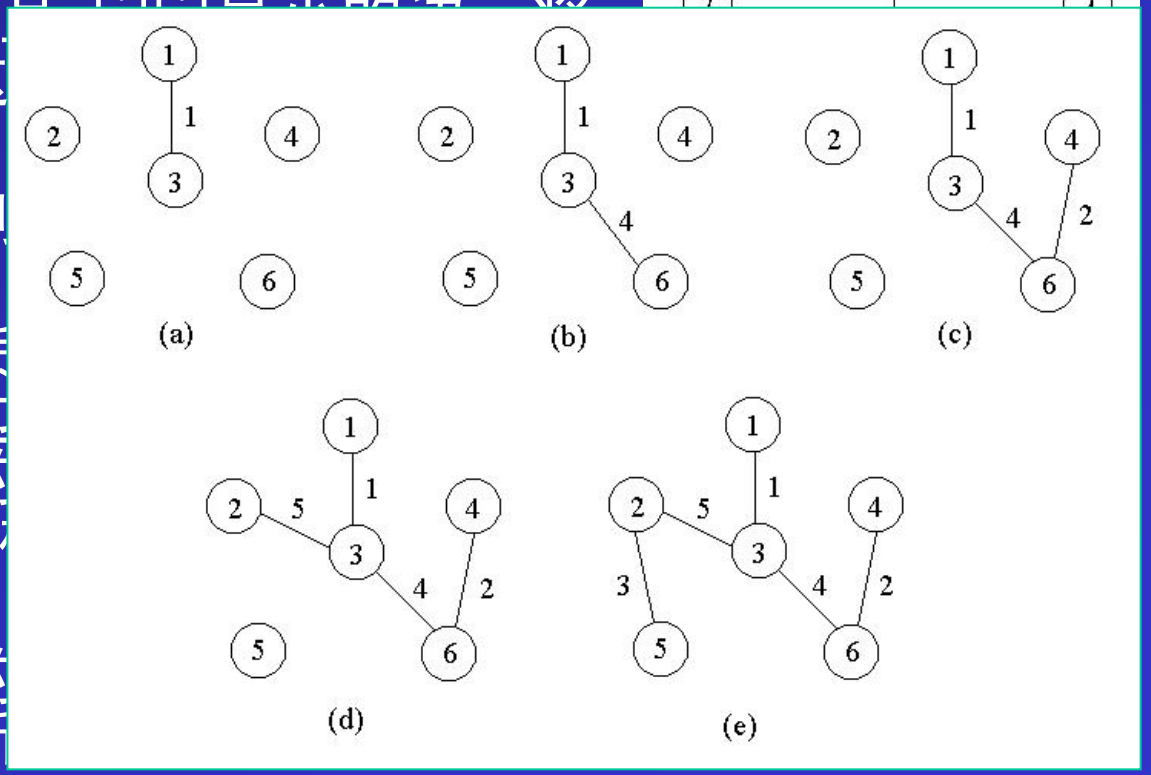
1、最小生成树性质

- 用贪心算法设计策略可以设计出构造最小生成树的有效算法。本节介绍的构造最小生成树的Prim算法和Kruskal算法都可以看作是应用贪心算法设计策略的例子。尽管这2个算法做贪心选择的方式不同，它们都利用了下面的最小生成树性质：
- 设G=(V,E)是连通带权图，U是V的真子集。如果(u,v)∈E，且u∈U，v ∈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的一棵最小生成树，它以(u,v)为其中一条边，这个性质有时也称为MST性质。

2、Prim算法

- 设G=(V,E)是连通带权图，V={1,2,…,n}。
- 构造G的最小生成树的Prim算法的基本思想是：首先置S={1}，然后，只要S是V的真子集，就作如下的贪心选择：选取满足条件i∈S，j∈V-S，
顶点j添加到S中。这
S=V时为止。.
- 在这个过程中选取到
的一棵最小生成树。
- 利用最小生成树性质
明，上述算法中的边
某棵最小生成树中的
束时，T中的所有边
成树。
- 例如，对于右图中的
选取边的过程如图所





3、Kruskal算法

- Kruskal算法构造G的最小生成树的基本思想是，首先将G的n个顶点看成n个孤立的连通分支。将所有的边按权从小到大排序，然后从第一条边开始，依边权递增的顺序
个不同的连通分
果端点v和w分
中的顶点时，就
分支，然后继续
当前的同一个连
边。这个过程一
止。
- 例如，对前面的
到的最小生成树