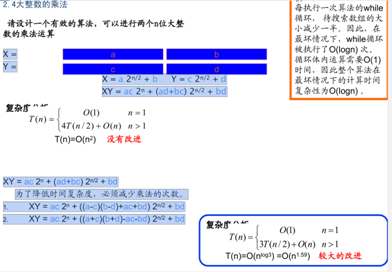
**分治法基本思想：**将一个规模为n的问题分解为k个规模较小的子问题，这些子问题互相独立且与原问题相同。递归的解这些子问题，然后将各个子问题的解合并成原问题的解。

**二分搜索**：将n个数据大致分为相同的两半，取a【n/2】与x比较。如果x= a【n/2】。结束，如果小于，在左边找。大于，右边找。最坏O（log2n）

**2.4大整数的乘法**

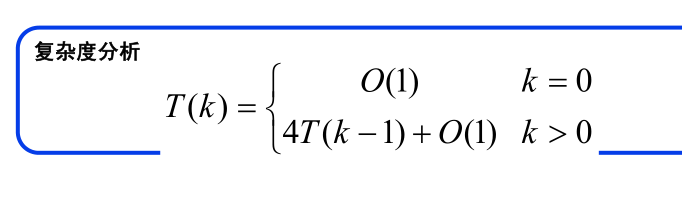


**2.6棋盘覆盖**

当k>0时，将2 k ×2 k 棋盘分割为4个2 k-1 ×2 k-1 子棋盘(a)所示。

特殊方格必位于4个较小子棋盘之一中，其余3个子棋盘中无特殊方格。为了将这3个无特殊方格的子棋盘转化为特殊棋盘，可以用一个L型骨牌覆盖这3个较小棋盘的会合处，如 (b)所示，从而将原问题转化为4个较小规模的棋盘覆盖问题。递归地使用这种分割，直至棋盘简化为棋盘

1×1。

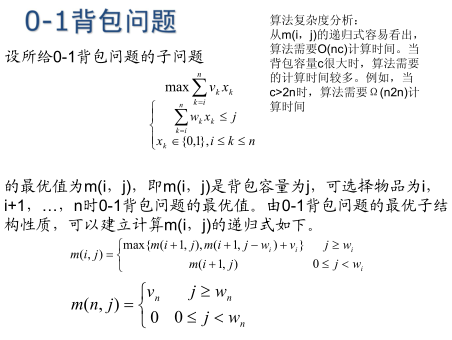
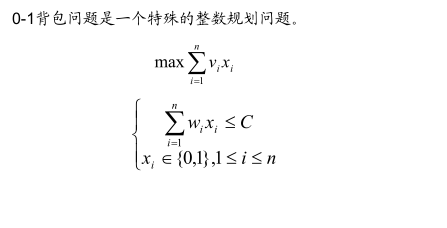


* **矩阵连乘**：1.分析最优解的结构特征：计算A[i:j]的最优次序所包含的计算矩阵子链A[i:k]和A[k+1:j]的次序也是最优的。矩阵连乘计算次序问题的最优解包含着其子问题的最优解。这种性质称为最优子结构性质。问题的最优子结构性质是该问题可用动态规划算法求解的显著特征2.建立递归关系:的位置只有种可能
* 
* 3.计算最优值：对于1≤i≤j≤n不同的有序对(i,j)对应于不同的子问题。因此，不同子问题的个数最多只有由此可见，在递归计算时，许多子问题被重复计算多次。这也是该问题可用动态规划算法求解的又一显著特征
* 4.构造最优解
* **动态规划的基本要素**:1.最优子结构：问题的最优解包含着其子问题的最优解。这种性质称为最优子结构性质。
* 2.子问题重叠性：递归算法求解问题时，每次产生的子问题并不总是新问题，有些子问题被反复计算多次。这种性质称为子问题的重叠性质。动态规划算法，对每一个子问题只解一次，而后将其解保存在一个表格中，当再次需要解此子问题时，只是简单地用常数时间查看一下结果。
* **凸多边形最优三角划分**：
* 1**.最优子结构划**分:若凸(n+1)边形P={v0,v1,…,vn-1}的最优三角剖分T包含三角形v0vkvn，1≤k≤n-1，则T的权为3个部分权的和：三角形v0vkvn的权，子多边形{v0,v1,…,vk}和{vk,vk+1,…,vn}的权之和。由T所确定的这2个子多边形的三角剖分也是最优的。因为若有{v0,v1,…,vk}或{vk,vk+1,…,vn}的更小权的三角剖分将导致T不是最优三角剖分的矛盾
* **2.递归结构**
* 

3.计算最优值

4.构造最优三角划分

**0-1背包问题**



**贪心算法：**

贪心算法是指，在对[问题求解](http://baike.baidu.com/view/1099373.htm)时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的是在某种意义上的局部[最优解](http://baike.baidu.com/view/1009692.htm)。贪心算法不是对所有问题都能得到整体最优解，关键是贪心策略的选择，选择的贪心策略必须具备的特点是：某个状态以前的过程不会影响以后的状态，只与当前状态有关。

贪心算法的要素：

• 1、贪心选择性质：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，在贪心算法中，仅在当前状态下做出最好选择，即局部最优选择，然后再去解做出这个选择后产生的相应的子问题。贪心算法所做的贪心选择可以依赖以往所做过的选择，但决不依赖于将来所做的选择，也不依赖于子问题的解。正是由于这种差别，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的进行，以迭代的方式做出相应的贪心选择，每做一次贪心选择就将所求的问题简化为规模更小的子问题。

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。

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的进行，以迭代的方式做出相应的贪心选择，每做一次贪心选择就将所求的问题简化为规模更小的子问题。

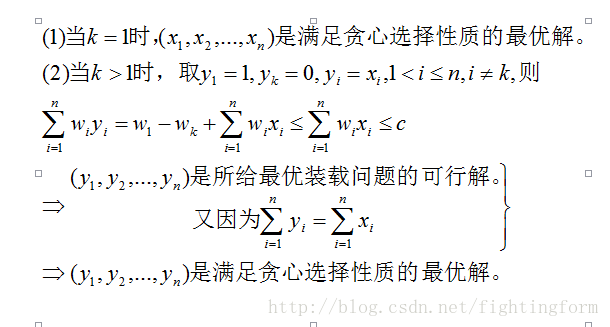
**最优装载：**

要使用贪心算法解决问题，我们必须先证明：

（1）该问题具备贪心选择性质；

（2）该问题具备最优子结构性质。

首先先证明贪心选择性质：设集装箱已依其重量从大到小排序，（x1,x2.......xn）是最优装载问题的一个最优解。又设k=min{ i |xi=1}{ 1<=i<=n}.易知，如果给定的最优装载问题有解，则1<=k<=n;



其次，证明该问题具备最优子结构性质：设(x1,x2....xn)是最优装载的满足贪心选择性质的最优解，易知，x1=1，(x2,x3.....xn)是轮船载重量为c-w1，待装船集装箱为{2,3，.....n}时相应的最优装载问题的最优解

最差o（nlogn）

**单源最短路径：**

1. 算法思想：

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。初始时，S中仅含有源。设u是G的某个顶点，从源到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来记录当前每个顶点所对应的最短特殊路径长度。该算法每次从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。S包含所有V中顶点时，dist就记录了从源到所有其他顶点的最短路径。

1. 贪心选择性质：

Dijkstra算法是应用贪心算法设计策略的一个典型例子。它所做的贪心选择是从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]。这种贪心选择能导致最优解，是因为，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的路长，那么，dist[x]≤d（v，x），d（v，x）+ d（x，u）=d（v，u）<dist[u]。

利用边权的非负性，可知d（x，u）≥0，从而推得dist[x]<dist[u]。此为矛盾。这就证明

了dist[u]是从源到顶点u的最短路径长度。

1. 最优子结构性质：

该性质描述为：如果P(i,j)={Vi....Vk..Vs...Vj}是从顶点i到j的最短路径，k和s

是这条路径上的一个中间顶点，那么P(k,s)必定是从k到s的最短路径。下面证明该性质的正确性。

证明：假设P(i,j)={Vi....Vk..Vs...Vj}是从顶点i到j的最短路径，则有P(i,j)=P(i,k)+P(k,s)+P(s,j)。而P(k,s)不是从k到s的最短距离，那么必定存在另一条从k到s的最短路径P'(k,s)，那么P'(i,j)=P(i,k)+P'(k,s)+P(s,j)<P(i,j)。则与P(i,j)是从i到j的最短路径相矛盾。因此该性质得证。

1. 时间复杂度：

O(n2)

**最小生成树**

Prim[**算法**](http://lib.csdn.net/base/datastructure)

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，且c[i][j]最小的边，将顶点j添加到S中。这个过程一直进行到S=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。 利用最小生成树性质和数学归纳法容易证明，上述算法中的边集合T始终包含G的某棵最小生成树中的边。因此，在算法结束时，T中的所有边构成G的一棵最小生成树，Prim算法所需要的计算时间为O(n^2)。

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，依边权递增的顺序查看每一条边，并按下述方法连接2个不同的连通分支：当查看到第k条边(v,w)时，如果端点v和w分别是当前2个不同的连通分支T1和T2中的顶点时，就用边(v,w)将T1和T2连接成一个连通分支，然后继续查看第k+1条边；如果端点v和w在当前的同一个连通分支中，就直接再查看第k+1条边。这个过程一直进行到只剩下一个连通分支时为止

**回溯算法基本概念**:回溯法在问题的解空间树中，按深度优先策略，从根结点出发搜索解空间树。算法搜索至解空间树的任意一点时，先判断该结点是否包含问题的解。如果肯定不包含，则跳过对该结点为根的子树的搜索，逐层向其祖先结点回溯；否则，进入该子树，继续按深度优先策略搜索, 直至找到所要求的解或解空间中已无活结点时为止。

**回溯法基本思想**:(1)针对所给问题，定义问题的解空间；

(2)确定易于搜索的解空间结构；

(3)以深度优先方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。用约束函数在扩展结点出剪去不满足约束的子树，用限界函数剪去得不到最优解的子树。

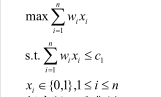
**装载问题:**有一批共n个集装箱要装上2艘载重量分别为c1和c2的轮船，其中集箱i的重量为wi，且装载问题要求确定是否有一个合理的装载方案可将这个集装箱装上这2艘轮船。如果有，找出一种装载方案。

容易证明，如果一个给定装载问题有解，则采用下面的策略可得到最优装载方案。

(1)首先将第一艘轮船尽可能装满；

(2)将剩余的集装箱装上第二艘轮船。

将第一艘轮船尽可能装满等价于选取全体集装箱的一个子集，使该子集中集装箱重量之和最接近。由此可知，装载问题等价于以下特殊的0-1背包问题



* 解空间：子集树
* 可行性约束函数(选择当前元素)：
* **上界函数**(不选择当前元素)：

当前载重量cw+剩余集装箱的重量r≤当前最优载重量bestw

**批处理作业调度:**给定n个作业的集合{J1,J2,…,Jn}。每个作业必须先由机器1处理，然后由机器2处理。作业Ji需要机器j的处理时间为tji。对于一个确定的作业调度，设Fji是作业i在机器j上完成处理的时间。所有作业在机器2上完成处理的时间和称为该作业调度的完成时间和。批处理作业调度问题要求对于给定的n个作业，制定最佳作业调度方案，使其完成时间和达到最小。

**算法设计**:设开始时x=[1,2…,n]是所给的n个作业，则相应的排列数x[1,n]的所有排列构成。i>n时算法搜索至叶节点，得到一个新的作业调度方案，此时算法适时更新当前最优值和最佳的作业调度。I<n时，当前扩展结点位于排列树的n-1层，此时算法选择下一个要安排的作业，以深度优先的方式递归的对相应子树进行搜索，对于不满足上界约束的结点则减去相应的子树。

**分支限界的基本思想**:分支限界法常以广度优先或以最小耗费（最大效益）优先的方式搜索问题的解空间树.在分支限界法中，每一个活结点只有一次机会成为扩展结点。活结点一旦成为扩展结点，就一次性产生其所有儿子结点。在这些儿子结点中，导致不可行解或导致非最优解的儿子结点被舍弃，其余儿子结点被加入活结点表中此后，从活结点表中取下一结点成为当前扩展结点，并重复上述结点扩展过程。这个过程一直持续到找到所需的解或活结点表为空时为止

**随机化算法**：

**线性同余法(产生随机数的算法**)：是产生伪随机数的最常用的方法。由线性同余法产生的随机序列a0,a1,…,an满足

其中b≥0，c≥0，d≤m。d称为该随机序列的种子。如何选取该方法中的常数b、c和m直接关系到所产生的随机序列的随机性能。从直观上看，m应取得充分大，因此可取m为机器大数，另外应取gcd(m,b)=1，因此可取b为一素数

**舍伍德算法:**

设A是一个确定性算法，当它的输入实例为x时所需的计算时间记为tA(x)。设Xn是算法A的输入规模为n的实例的全体，则当问题的输入规模为n时，算法A所需的平均时间为

这显然不能排除存在x∈Xn使得的可能性。希望获得一个概率算法B，使得对问题的输入规模为n的每一个实例均有

这就是舍伍德算法设计的基本思想。当s(n)与tA(n)相比可忽略时，舍伍德算法可获得很好的平均性能。

**跳跃表：**提高有序链表效率的一个技巧是在有序链表的部分结点处增设附加指针以提高其搜索性能。在增设附加指针的有序链表中搜索一个元素时，可借助于附加指针跳过链表中若干结点，加快搜索速度。这种增加了向前附加指针的有序链表称为跳跃表。

在一般情况下，给定一个含有n个元素的有序链表，可以将它改造成一个完全跳跃表，使得每一个k级结点含有k+1个指针，分别跳过2k-1，2k-1-1，…，20-1个中间结点。第i个k级结点安排在跳跃表的位置i2k处，i≥0。这样就可以在时间O(logn)内完成集合成员的搜索运算。在一个完全跳跃表中，最高级的结点是⎡logn⎤级结点。

为了在动态变化中维持跳跃表中附加指针的平衡性，必须使跳跃表中k级结点数维持在总结点数的一定比例范围内。注意到在一个完全跳跃表中，50%的指针是0级指针；25%的指针是1级指针；…；(100/2k+1)%的指针是k级指针。因此，在插入一个元素时，以概率1/2引入一个0级结点，以概率1/4引入一个1级结点，…，以概率1/2k+1引入一个k级结点。另一方面，一个i级结点指向下一个同级或更高级的结点，它所跳过的结点数不再准确地维持在2i-1。经过这样的修改，就可以在插入或删除一个元素时，通过对跳跃表的局部修改来维持其平衡性。

**拉斯维加斯算法:**

舍伍德算法优点是其计算时间复杂性对所有实例相对均匀。但与其相应的确定性算法

相比，其平均时间复杂性没有改进。拉斯维加斯算法则不然，他能显著地改进算法的

有效性，甚至对某些迄今为止找不到有效算法的问题，也能得到满意结果

拉斯维加斯算法的一个显著特征是它所作的随机性决策有可能导致算法找不到所需的解

拉斯维加斯算法的典型调用形式为：bool success=LV(X,Y)；x是输入参数。当success为true时，y返回问题的解。否则，未能找到解。

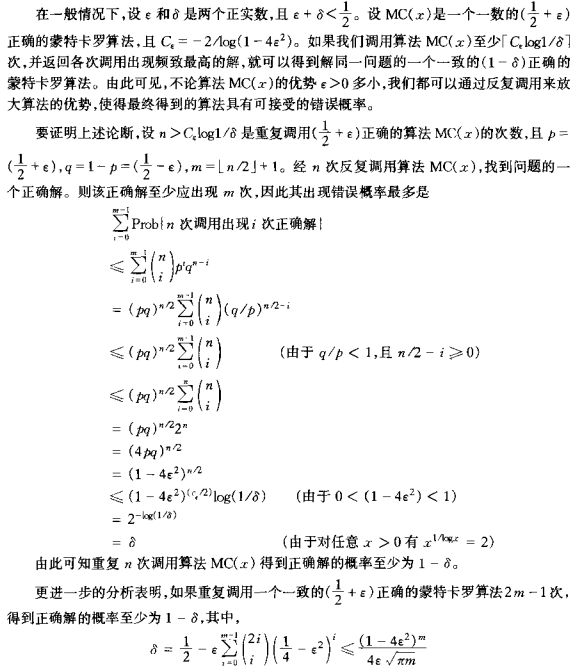
设p(x)是对输入x调用拉斯维加斯算法获得问题的一个解的概率。一个正确的拉斯维加斯算法应该对所有输入x均有p(x)>0。

设t(x)是算法obstinate找到具体实例x的一个解所需的平均时间,s(x)和e(x)分别是算法对于具体实例x求解成功或求解失败所需的平均时间，则有：

解此方程可得：



**蒙特卡洛算法基本思想:**设p是一个实数，且1/2<p<1。如果一个蒙特卡罗算法对于问题的任一实例得到正确解的概率不小于p，则称该蒙特卡罗算法是p正确的，且称p-1/2是该算法的优势。如果对于同一实例，蒙特卡罗算法不会给出2个不同的正确解答，则称该蒙特卡罗算法是一致的。有些蒙特卡罗算法除了具有描述问题实例的输入参数外，还具有描述错误解可接受概率的参数。这类算法的计算时间复杂性通常由问题的实例规模以及错误解可接受概率的函数来描述.对于一个一致的p正确蒙特卡罗算法，要提高获得正确解的概率，只要执行该算法若干次，并选择出现频次最高的解即可。

对于一个解所给问题的蒙特卡罗算法MC(x)，如果存在问题实例的子集X使得：

(1)当x∉X时，MC(x)返回的解是正确的；

(2)当x∈X时，正确解是y0，但MC(x)返回的解未必是y0。

称上述算法MC(x)是偏y0的算法

**线形规划问题和单纯形算法**：

**线性规划基本定理：**

如果线性规划问题有最优解，则必有一基本可行最优解。只要对不同的组合进行测试，并比较每种情况下的目标函数值就能找到

**单纯形算法特点：**

• (1)只对约束条件的若干组合进行测试，测试的每一步都使目标函数的值增加；

• (2)一般经过不大于m或n次迭代就可求得最优解

**最优解单纯形算法的第1步：**选出使目标函数增加的非基本变量作为**入基变量**

**单纯形算法的第2步：选**取离基变量。在单纯形表中考察由第1步选出的入基变量所相应的列。在一个基本变量变为负值之前，入基变量可以增到多大。如果入基变量所在的列与基本变量所在行交叉处的表元素为负数，那么该元素将不受任何限制，相应的基本变量只会越变越大。如果入基变量所在列的所有元素都是负值，则目标函数无界，已经得到了问题的无界解。

如果选出的列中有一个或多个元素为正数，要弄清是哪个数限制了入基变量值的增加。

受限的增加量可以用入基变量所在列的元素（称为主元素）来除主元素所在行的“常数列”（最左边的列）中元素而得到。所得到数值越小说明受到限制越多。应该选取受到限制最多的基本变量作为离基变量，才能保证将入基变量与离基变量互调位置后，仍满足约束条件。

**单纯形算法的第3步：**转轴变换。转轴变换的目的是将入基变量与离基变量互调位置。给入基变量一个增值，使之成为基本变量；修改离基变量，让入基变量所在列中，离基变量所在行的元素值减为零，而使之成为非基本变量

**单纯形算法的第4步：**转回并重复第1步，进一步改进目标函数值。不断重复上述过程，直到z行的所有非基本变量系数都变成负值为止。这表明目标函数不可能再增加了

**增广路算法:1 算法基本思想**设P是网络G中联结源s和汇t的一条路。定义路的方向是从s到t。

将路P上的边分成2类：一类边的方向与路的方向一致，称为**向前边**。向前边的全体记为P+。另一类边的方向与路的方向相反，称为**向后边**。向后边的全体记为P-。

设flow是一个可行流，P是从s到t的一条路，若P满足下列条件：

（1）在P的所有向前边(v,w)上，flow(v,w)<cap(v,w)，即P+中的每一条边都是非饱和边；

（2）在P的所有向后边(v,w)上，flow(v,w)>0，即P-中的每一条边都是非零流边。

则称P为关于可行流flow的一条可增广路。

可增广路是残流网络中一条容量大于0的路。

将具有上述特征的路P称为可增广路是因为可以通过修正路P上所有边流量flow(v,w）将当前可行流改进成一个流值更大的可行流。增流的具体做法是：

（1）不属于可增广路P的边(v,w)上的流量保持不变；

（2）可增广路P上的所有边(v,w)上的流量按下述规则变化：

在向前边(v,w)上，flow(v,w)+d；

在向后边(v,w)上，flow(v,w)-d。

按下面的公式修改当前的流。

其中d称为可增广量，可按下述原则确定：d取得尽量大，又要使变化后的流仍为可行流。

按照这个原则，d既不能超过每条向前边(v,w)的cap(v,w)-flow(v,w)，也不能超过每向后边(v,w)的flow(v,w)。

因此d应该等于向前边上的cap(v,w)-flow(v,w)与向后边上的flow(v,w)的最小值。就是残流网络中P的最大容量。

**增广路定理：**设flow是网络G的一个可行流，如果不存在从s到t关于flow的可增广路P，则flow是G的一个最大流。



**图灵机:**一台图灵机由一个有限状态控制器和K条读写带组成，这些读写带的右端无限，每条带从左到右划分为一个方格，每个方格可以存放一个带符号，带符号的总数是有限的。每条带上都有一个由有限状态控制器操纵的读写头（带头），它可以对这K条带头进行读写操作，有限状态就控制器在某一时刻处于某种状态。且状态总数是有限的。

根据有限状态控制器的当前状态及每个读写头读到的带符号，图灵机的一个计算步可实现下面3个操作之一或全部。

(1)改变有限状态控制器中的状态。

(2)清除当前读写头下的方格中原有带符号并写上新的带符号。

(3)独立地将任何一个或所有读写头，向左移动一个方格(L)或向右移动一个方格(R)或停在当前单元不动(S)

k带图灵机可形式化地描述为一个7元组(Q，T，I，δ，b，q0，qf)，其中:

(1)Q是有限个状态的集合。 (2)T是有限个带符号的集合。

(3)I是输入符号的集合，I⊆T。(4)b是唯一的空白符，b∈T-I。

(5)q0是初始状态。 (6)qf是终止(或接受)状态。

(7)δ是移动函数。它是从Q×Tk的某一子集映射到Q× (T×{L，R，S})k的函数

**N类与NP类问题：**

**P类和NP类语言的定义：**

P={L|L是一个能在**多项式时间内**被一台**DTM**所接受的语言}

NP={L|L是一个能在**多项式时间**内被一台**NDTM**所接受的语言}

由于一台确定性图灵机可看作是非确定性图灵机的特例，所以可在多项式时间内被确定性图灵机接受的语言也可在多项式时间内被非确定性图灵机接受。故**P ⊆ NP**

**多项式时间变换:**设,是2个语言。所谓语言L1能在多项式时间内变换为语言L2(简记为L1 ∝pL2)是指存在映身f:.且f满足:

(1)有一个计算f的多项式时间确定性图灵机；

(2)对于所有x∈，x∈L1，当且仅当f(x)∈L2。

定义:语言L是NP完全的当且仅当(

1)L∈NP；

(2)对于所有L’∈NP有L’∝p L。

如果有一个语言L满足上述性质(2)，但不一定满足性质(1)，则称该语言是NP难的。所有NP完全语言构成的语言类称为NP完全语言类，记为NPC。

**定理2**：设L是NP完全的，则

(1)L∈P当且仅当P＝NP；

(2)若L∝p ，且∈NP，则是NP完全的

**近似算法的性能：**

若一个最优化问题的最优值为c\*，求解该问题的一个近似算法求得的近似最优解相应的目标函数值为c，则将该近似算法的性能比定义为=,该性能比是问题输入规模n的一个函数ρ(n)，即≤ρ(n).该近似算法的相对误差定义为*"*= 。若对问题的输入规模n，有一函数ε(n)使得≤ε(n)，则称ε(n)为该近似算法的相对误差界。

近似算法的性能比ρ(n)与相对误差界ε(n)之间显然有如下关系：**ε(n)≤ρ(n)-1**。

**旅行商售货问题近似算法**:问题描述：给定一个完全无向图G=(V,E)，其每一边(u,v)∈E有一非负整数费用c(u,v)。要找出G的最小费用哈密顿回路

比如，费用函数c往往具有**三角不等式性质**，即对任意的3个顶点u,v,w∈V，有：c(u,w)≤c(u,v)+c(v,w)。当图G中的顶点就是平面上的点，任意2顶点间的费用就是这2点间的欧氏距离时，费用函数c就具有三角不等式性质。

对于给定的无向图G，可以利用找**图G的最小生成树**的算法设计找近似最优的旅行售货员回路的算法。void **approxTSP** (Graph g)

{

(1)选择g的任一顶点r；

(2)用Prim算法找出带权图g的一棵以r为根的最小生成树T；

(3)前序遍历树T得到的顶点表L；

(4)将r加到表L的末尾，按表L中顶点次序组成回路H，作为计 算结果返回；

}

当费用函数满足三角不等式时，算法找出的旅行售货员回路的费用不会超过最优旅行售货员回路费用的2倍

**一般旅行售货问题**：在费用函数不一定满足三角不等式的一般情况下，不存在具有常数性能比的解TSP问题的多项式时间近似算法，除非**P=NP**。换句话说，若P≠NP，则对任意常数ρ>1，不存在性能比为ρ的解旅行售货员问题的多项式时间近似算法

**集合覆盖问题的近似算法**：

集合覆盖问题的一个实例〈X,F〉由一个有限集X及X的一个子集族F组成。子集族F覆盖了有限集X。也就是说X中每一元素至少属于F中的一个子集，即X= 。对于F中的一个子集C⊆F，若C中的X的子集覆盖了X，即X= ，则称C覆盖了X。集合覆盖问题就是要找出F中覆盖X的最小子集C\*，使得 |C\*|=min{|C||C⊆F且C覆盖X}

集合覆盖问题近似算法——贪心算法

Set **greedySetCover** (X,F)

{

U=X；

C=∅；

while (U !=∅) {

选择F中使|S∩U|最大的子集S；

U=U-S；

C=C∪{S}；

}

return C； }

算法的循环体最多执行min{|X|，|F|}次。而循环体内的计算显然可在O(|X||F|)时间内完成。因此，算法的计算时间为O(|X||F|min{|X|，|F|})。由此即知，该算法是一个多项式时间算法。