**递归与分治:**

**棋盘覆盖**T(n)=O(4k) 当k>0时，将2k×2k棋盘分割为4个2k-1×2k-1 子棋盘。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，这3个子棋盘上被L型骨牌覆盖的方格就成为该棋盘上的特殊方格，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。T(k)满足以下递归方程,解此递归方程T(k)= O(4k)，

**二分搜索技术** O(logn) 将n个元素分成大致相同的两半，取a[n/2]与x比较，如果a[n/2]=x，则找到，算法终止；如果a[n/2]>x,则只在数组的左半部继续搜索x；如果a[n/2]<x，则只在数组a的右半部继续搜索x。算法复杂度分析：每执行一次算法的while循环， 待搜索数组的大小减少一半。因此，在最坏情况下，while循环被执行了O(logn) 次。循环体内运算需要O(1) 时间，因此整个算法在最坏情况下的计算时间复杂性为O(logn) 。

**大整数的乘法** O(nlog3)=O(n1.59) 设X和Y都是n位二进制整数，计算XY的乘积，将n位二进制整数X和Y都分为2段，每段的长为n/2位，由X=A2n/2+B，Y=C2n/2+D，这样X和Y的乘积XY = AC2n + (AD+BC) 2n/2 + BD，设T(n)是2个n位整数相乘所需的运算总数，则有

解得T(n)=O(n2)。为改进算法的计算复杂性,必须减少乘法次数，故有XY = AC2n + ((A-B)(D-C)+AC+BD) 2n/2 + BD，则有 解得T(n)=O(nlog3)

**最接近点对**O(nlogn)

**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

**回溯法解空间：**问题的解向量：回溯法希望一个问题的解能够表示成一个n元式(x1,x2,…,xn)的形式。显约束：对分量xi的取值限定。隐约束：为满足问题的解而对不同分量之间施加的约束。解空间：对于问题的一个实例，解向量满足显式约束条件的所有多元组，构成了该实例的一个解空间。注意：同一个问题可以有多种表示，有些表示方法更简单，所需表示的状态空间更小（存储量少，搜索方法简单）

**回溯法基本思想:** (1)针对所给问题，定义问题的解空间（排列树和子集树）；(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。**解空间**：{问题的解向量：回溯法希望一个问题的解能够表示成一个n元式(x1,x2,…,xn)的形式。显约束：对分量xi的取值限定。隐约束：为满足问题的解而对不同分量之间施加的约束。解空间：对于问题的一个实例，解向量满足显式约束条件的所有多元组，构成了该实例的一个解空间。}

**旅行商问题：**旅行商问题的解空间是一个排列树，则对于该排列树求解最小代价。遍历整棵树，在n层时，代表所有的节点都已经访问，则形成解的条件是：当前节点是否与1（也就是起点）连通，如果连通则构成一个解，在所有解中取最小值则为最终解。不在n层时，则考虑当前路径的消耗是否小于当前最优解，如果大于当前最优解则剪枝。

**动态规划的基本要素 (自底向上依赖子问题的解):**

**矩阵连乘问题** 穷举法：P(n)=Ω(4n/n3/2) 动规：时间O(n3) 空间 O(n2) 设矩阵连乘乘积AiAi+1…Aj简记为A[i:j].考察A[1:n]的最优计算次序，设这个计算次序在矩阵Ak 和Ak+1 之间将矩阵链断开,1≤k<n，对应完全加括号方式为((A1…Ak)(Ak+1…An))则(1)**最优子结构性质:** 总计算量为A[1:k]的计算量加上A[k+1:n]的计算量，再加上A[1:k]和A[k+1:n]相乘的计算量。计算A[1:n]的最优次序包含计算矩阵子链A[1:k]和A[k+1:n]的次序也是最优的。事实上，如果有一个计算A[1:k]的次序需要的计算量更少，则用此次序替代原来计算A[1:k]的次序，得到计算A[1:n]的计算量将比最优次序所需计算量更少，这是一个矛盾。同理可知，计算A[1:n]的最优次序所包含的计算矩阵子链A[k+1:n]的次序也是最优的。因此，矩阵连乘乘积计算次序问题的最优解包含着其子问题的最优解。 (2)**建立递归关系:** 设计算A[i:j]，1≤i≤j≤n，所需要的最少数乘次数m[i,j]，则原问题的最优值为m[1,n] 。当i=j时，A[i:j]=Ai为单一矩阵，无需计算，因此，m[i,i]=0，i=1,2,…,n；当i<j时，m[i,j]=m[i,k]+m[k+1,j]+pi-1pkpj Ai的维数为pi-1\*pi ,k的位置有j-i种可能： (3)**计算最优值:** 依据递归式自底向上的方式计算，输入参数{p0,p1…pn}存储于数组p中，除了输出最优值数组m外，还输出记录最优断开位置的数组s。首先计算出m[i][i]=0,i=1,2…n;根据递归式按矩阵链长递增的方式计算m[i][i+1], i=1,2…n-1（矩阵链长度为2）；m[i][i+2]，i=1,2…n-2（矩阵链长度为3）;在计算m[i][j]时，只用到已计算出的m[i][k]和m[k+1][j]。 (4**)构造最优解:** s[i][j]中的数表明，计算矩阵链A[i:j]的最佳方式应在矩阵Ak和Ak+1之间断开，即最优的加括号方式应为(A[i:k])(A[k+1:j])。因此，从s[1][n]记录的信息可知计算A[1:n]的最优加括号方式为(A[1:s[1][n]])(A[s[1][n]+1:n])。而A[1:s[1][n]]的最优加括号方式为(A[1:s[1][s[1][n]]])(A[s[1][s[1][n]]+1:s[1][s[1][n]]])。同理可以确定A[s[1][n]+1:n]的最优加括号方式在s[s[1][n]+1][n]处断开……照此递推下去，最终可以确定A[1:n]的最优完全加括号方式，即构造出问题的一个最优解。

**凸多边形的最优三角剖分**(与矩阵连乘问题类似) 时间O(n3) 空间 O(n2)

(1)**最优子结构性质**：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。可以断言，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**：t[i][j]，1≤i<j≤n为凸子多边形{vi-1,vi,…,vj}的最优三角剖分所对应的权函数值，即其为最优值(t[i][j]记录的是vi-1 - vj 的权值和)，t[i][j]可递归的定义为： 其中，i≤k≤j-1,k的位置有j-i个。 (3)**计算最优值** (4)**构造最优值**，在计算每一个凸子多边形{vi-1,vi,…,vj}的最优值时，用数组s记录了最优三角剖分中所有三角形信息。s[i][j]记录了与vi-1和vj一起构成三角形的第三个顶点位置，据此，用O(n)时间就可构造出最优三角剖分中的所有三角形。

**0-1背包问题** O(nc) 改进O(min{nc,2n})

**(1)最优子结构性质** 0-1背包问题具有最优子结构性质。设(y1,y2,..,yn)是所给0-1背包问题的一个最优解，则(y2,..,yn)是下面相应子问题的一个最优解： 。因若不然，设(z2,..,zn)是上述子问题的一个最优解，而(y2,..,yn)不是它的最优解。由此可知，且。因此 ，,这说明(y1,z2,..,zn)是所给0-1背包问题的一个更优解，从而(y1,y2,..,yn)不是所给0-1背包问题的最优解。此为矛盾。 **(2)递归关系** 设所给0-1背包问题的子问题

的最优值为m(i，j)，即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。 ， **(3)算法描述** 当wi(1<=i<=n)为正整数时，用二维数组m[][]来存储m(i,j)的相应值，m[1][c]给出所要求的0-1背包问题的最优值。若m[1][c]=m[2][c]，则x1 =0，否则x1=1.当x1=0时，由m[2][c]继续构造最优解。当x1=1时，由m[2][c-w1]继续构造最优解。以此类推，可构造出相应的最优解(x1,x2,…,xn). **(4)算法复杂度分析** 从m(i,j)的递归式容易看出，算法需要O(nc)计算时间。当背包容量c很大时，算法需要的计算时间较多。改进后算法的计算时间复杂性为O(2n)。当所给物品的重量wi(1≤i≤n)是整数时，|p[i]|≤c+1，(1≤i≤n)。在这种情况下，改进后算法的计算时间复杂性为O(min{nc,2n})。

**贪心算法(自顶向下):**贪心算法是指，在对问题求解时，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。贪心算法不是对所有问题都能得到整体最优解，即使贪心算法不能得到整体最优解，但其最终结果却是最优解的很好的近似解。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。

**贪心算法基本要素：**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达到）和最优子结构性质（问题的最优解包含其子问题的最优解，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）

**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小到大排序，（,,.......,）是最优装载问题的一个最优解。又设k=min{i|=1}{1in}.易知，如果给定的最优装载问题有解，则1kn; **证明**：当k=1时，(,,…,)是满足贪心选择性质的最优解。当k1时，取,,,1in,ik,则=+ (x1=0,xk=1),因此（,,…,）是所给最优装载问题的可行解，又因为知，,,…,）是一个满足贪心性质的最优解，所以最优装载问题具有贪心选择性质.其次，证明该问题具备最优子结构性质：设(,,.......是最优装载的满足贪心选择性质的最优解，易知，=1,（,,.......是轮船载重量为c-，待装船集装箱为{2,3，.....n}时相应的最优装载问题的最优解。

**单源最短路径Dijkstra算法: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。**贪心选择性质：**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。

**最小生成树Prim算法(点)**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iS，jV-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(n2)。

**最小生成树Kruskal算法(边)**设G=(V,E)是无向连通带权图，V={1,2,…,n},该算法构造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

**回溯法解空间：**问题的解向量：回溯法希望一个问题的解能够表示成一个n元式(x1,x2,…,xn)的形式。显约束：对分量xi的取值限定。隐约束：为满足问题的解而对不同分量之间施加的约束。解空间：对于问题的一个实例，解向量满足显式约束条件的所有多元组，构成了该实例的一个解空间。注意：同一个问题可以有多种表示，有些表示方法更简单，所需表示的状态空间更小（存储量少，搜索方法简单）

**回溯法基本思想:** (1)针对所给问题，定义问题的解空间（排列树和子集树）；(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。**解空间**：{问题的解向量：回溯法希望一个问题的解能够表示成一个n元式(x1,x2,…,xn)的形式。显约束：对分量xi的取值限定。隐约束：为满足问题的解而对不同分量之间施加的约束。解空间：对于问题的一个实例，解向量满足显式约束条件的所有多元组，构成了该实例的一个解空间。}

**旅行商问题：** **(1)算法描述：**旅行售货员问题的解空间是一棵排列树。 对于排列树的回溯搜索与生成1,2, ... n的所有排列的递归算法Perm类似。开始时x=[1, 2, ... n]，则相应的排列树由x[1:n]的所有排列构成。

在递归算法Backtrack中，当i=n时，当前扩展结点是排列树的叶结点的父结点。此时算法检测图G是否存在一条从顶点x[n-1]到顶点x[n]的边和一条从顶点x[n]到顶点1的边。如果这两条边都存在，则找到一条旅行售货员回路。算法还需判断这条回路的费用是否优于已找到的当前最优回路的费用bestc。如果是，则必须更新当前最优值bestc和当前最优解bestx。当i<n时，当前扩展结点位于排列树的第i-1层。图G中存在从顶点x[i-1]到顶点x[i]的边时，x[1:i]构成图G的一条路径，且当x[1:i]的费用小于当前最优值时算法进入排列树的第i层，否则将剪去相应的子树。算法中用变量cc记录当前路径x[1:i]的费用。**(2)算法效率：**如果不考虑更新bestx所需的计算时间，则Backtrack 需要O((n- 1)!)计算时间。由于算法Backtrack在最坏情况下可能需要更新当前最优解0((n- 1)!)次，每次更新bestx需O(n)计算时间，从而整个算法的计算时间复杂性为O(n!)。

**分支限界法定义：**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。

**最大团问题：**上界函数：用变量表示与该结点相应的团的顶点数；level表示结点在子集空间树中所处的层次；用作为顶点数上界的值。 在此优先队列式分支限界法中，实际上也是优先队列中元素的优先级。算法总是从活结点优先队列中抽取具有最大值的元素作为下一个扩展元素。算法思想:子集树的根结点是初始扩展结点，对于这个特殊的扩展结点，其的值为0。算法在扩展内部结点时，首先考察其左儿子结点。在左儿子结点处，将顶点加入到当前团中，并检查该顶点与当前团中其它顶点之间是否有边相连。当顶点与当前团中所有顶点之间都有边相连，则相应的左儿子结点是可行结点，将它加入到子集树中并插入活结点优先队列，否则就不是可行结点。接着继续考察当前扩展结点的右儿子结点。当时，右子树中可能含有最优解，此时将右儿子结点加入到子集树中并插入到活结点优先队列中。

**装载问题：**队列式分支限界法：在算法的while循环中，首先检测当前扩展结点的左儿子结点是否为可行结点。如果是则将其加入到活结点队列中。然后将其右儿子结点加入到活结点队列中(右儿子结点一定是可行结点)。2个儿子结点都产生后，当前扩展结点被舍弃。活结点队列中的队首元素被取出作为当前扩展结点，由于队列中每一层结点之后都有一个尾部标记-1，故在取队首元素时，活结点队列一定不空。当取出的元素是-1时，再判断当前队列是否为空。如果队列非空，则将尾部标记-1加入活结点队列，算法开始处理下一层的活结点。算法改进：节点的左子树表示将此集装箱装上船，右子树表示不将此集装箱装上船。设bestw是当前最优解；ew是当前扩展结点所相应的重量；r是剩余集装箱的重量。则当ew+r≤bestw时，可将其右子树剪去，因为此时若要船装最多集装箱，就应该把此箱装上船。另外，为了确保右子树成功剪枝，应该在算法每一次进入左子树的时候更新bestw的值。

解装载问题的优先队列式分支限界法用最大优先队列存储活结点表。活结点x在优先队列中的优先级定义为从根结点到结点x的路径所相应的载重量再加上剩余集装箱的重量之和。优先队列中优先级最大的活结点成为下一个扩展结点。以结点x为根的子树中所有结点相应的路径的载重量不超过它的优先级。子集树中叶结点所相应的载重量与其优先级相同。在优先队列式分支限界法中，一旦有一个叶结点成为当前扩展结点，则可以断言该叶结点所相应的解即为最优解。此时可终止算法。

**01背包：**算法思想：首先，要对输入数据进行预处理，将各物品依其单位重量价值从大到小进行排列。在下面描述的优先队列分支限界法中，节点的优先级由已装袋的物品价值加上剩下的最大单位重量价值的物品装满剩余容量的价值和。算法首先检查当前扩展结点的左儿子结点的可行性。如果该左儿子结点是可行结点，则将它加入到子集树和活结点优先队列中。当前扩展结点的右儿子结点一定是可行结点，仅当右儿子结点满足上界约束时才将它加入子集树和活结点优先队列。当扩展到叶节点时为问题的最优值。

**第七章**

**随机投点求pai：**设有一半径为r的圆及其外切四边形。向该正方形随机的投掷n个点。设落入园内的点数为k。由于所投入的点在正方形上均匀分布，一二所投入的点落入园内的概率为。所以当n足够大时，k与n 的比就逼近这一概率。从而

**随机投点求定积分:**在0-1的正方形中均匀的作投点实验，则随机落在曲线下面的概率为假设向单位正方形内随机的投入n个点。随机点落入G内，则。如果有m个点落入G内，则近似等于随机点落入G内的概率，即

**随机点投法求解非线性方程组：**在指定求根区域D内，选定一个随机点作为随机搜索的出发点。在算法的搜索过程中，假设第j步随机搜索得到的随机搜索 点为。在第j+1步，计算出下一步的随机搜索增量。从当前点依得到第j+1步的随机搜索点。当时，取 为所求非线性方程组的近似解。否则进行下一步新的随机搜索过程。

**拉斯维加斯算法（需要反复调用直到给出一个正确解）:**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。拉斯维加斯算法不会得到不正确的解。一旦用拉斯维加斯算法找到一个解，这个解就一定是正确解。但有时用拉斯维加斯算法找不到解。与蒙特卡罗算法类似，拉斯维加斯算法找到正确解的概率随着它所用的计算时间的增加而提高。对于所求解问题的任一实例，用同一拉斯维加斯算法反复对该实例求解足够多次，可使求解失败的概率任意小。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：t(x)=p(x)\*s(x)+(1-p(x))(e(x)+t(x));解此方程可得： t(x)=s(x)+e(x)

   {// 反复调用拉斯维加斯算法LV(x,y)，直到找到问题的一个解y

void **obstinate**(Object x, Object y)

      bool success= false; while (!success) success=lv(x,y);  }