二、递归与分治:

(1)将一个规模为 n 的问题分解为 k 个规模较小的子问题，这些子问题相互独立且与原问题相同。递归地解这些问题，然后将各子问题的解合并得到原问题的解。

(2)分治法所能解决的问题一般具有以下几个特征：问题的规模缩小到一定的程度就可以容易地解决；该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质；该问题分解出的子问题的解可以合并为该问题的解；该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。

在用分治法设计算法时，最好使子问题的规模大致相同。即将一个问题分成大小相等的 k 个子问题的处理方法是行之有效的。这种使子问题规模大致相等的做法是出自一种平衡子问题的思想，它几乎总是比子问题规模不等的做法要好。

$$\mathbf{T(n)}=\begin{cases}\boldsymbol{O(1)} & \boldsymbol{n=1}\\ \boldsymbol{kT(n/m)+f(n)} & \boldsymbol{n>1}\end{cases}$$

$$\mathbf{T(n)}=\boldsymbol{n^{log_mk}}+\sum_{j=0}^{log_mn-1}\boldsymbol{k^j f(n/m^j)}$$

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

大整数的乘法 **T(n)=O(n^{log3})=O($n^{1.59}$)**

X=A$2^{n/2}$+B Y=C$2^{n/2}$+D $\mathbf{T(n)}=\begin{cases}\boldsymbol{O(1)} & \boldsymbol{n=1}\\ \boldsymbol{kT(n/m)+f(n)} & \boldsymbol{n>1}\end{cases}$

XY=ac 2^n+(ad+bc) $2^{n/2}$+bd **T(n)=O(n^2)**

XY=ac 2^n+((a-c)(b-d)+ac+bd)$2^{n/2}$+bd

$$\mathbf{T(n)}=\begin{cases}\boldsymbol{O(1)} & \boldsymbol{n=1}\\ \boldsymbol{3T(n/2)+O(n)} & \boldsymbol{n>1}\end{cases}$$

Strassen 矩阵乘法:

T(n)= O(n^3) T(n)= O(n^{log7}) $\mathbf{T(n)}=\begin{cases}\boldsymbol{O(1)} & \boldsymbol{n=2}\\ \boldsymbol{7T(n/2)+O(n^2)} & \boldsymbol{n>2}\end{cases}$

棋盘覆盖 **T(n)=O(4^k)**

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，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。

$$\mathbf{T(k)}=\begin{cases}\boldsymbol{O(1)} & \boldsymbol{k=0}\\ \boldsymbol{4T(k-1)+O(1)} & \boldsymbol{k>0}\end{cases}$$

合并排序 **T(n)=O(nlogn)**

辅助空间：O(n)将待排序元素分成大小大致相同的 2 个子集合，分别对 2 个子集合进行排序，最终将排好序的子集合合并成为所要求的排好序的集合。

快速排序 **最坏时间复杂度：O(n2) 平均时间复杂度：O(nlogn) 辅助空间：O(n)或 O(logn)**

步骤：a[p:r]（1）**分解**：以 a[q]为基准元素将 a[p:r]划分为 3 段 a[p:q-1],a[q]和 a[q+1:r]使 a[p:q-1]中元素<= a[q],而 a[p+1:r]中任和一个元素大于等于 a[q]。下标 q 在划分过程中确定。（2）**递归求解**：通过递归调用快速排序方法分别对 a[p:q-1]和 a[q+1:r]排序;(3)**合并**：由于对 a[p:q-1]和 a[p+1:r]的排序是就地进行的，所以在 a[p:q-1]和 a[p+1:r]都以排好的序后，不需要执行任何计算，a[p:r]已排好序

线性时间选择 **O(n)**

将 n 个输入元素划分成⌈n/5⌉个组，每组 5 个元素，只可能有一个组不是 5 个元素。用任意一种排序算法，将每组中的元素排好序，并取出每组的中位数，共⌈n/5⌉个。递归调用 select 来找出这⌈n/5⌉个元素的中位数。如果⌈n/5⌉是偶数，就找它的 2 个中位数中较大的一个，以这个元素作为划分基准。设所有元素互不相同。在这种情况下，找出的基准 x 至少比 3(n-5)/10 个元素大，因为在每一组中有 2 个元素小于本组的中位数，而 n/5 个中位数中又有(n-5)/10 个小于基准 x。同理，基准 x 也至少比 3(n-5)/10 个元素小。而当 n≥75 时，3(n-5)/10≥n/4 所以按此基准划分所得的 2 个子数组的长度都至少缩短 1/4。

$$\mathbf{T(n)}\le\begin{cases}\boldsymbol{C_1} & \boldsymbol{n<75}\\ \boldsymbol{C_2+T(n/5)+T(3n/4)} & \boldsymbol{n\ge 75}\end{cases}$$

上述算法将每一组的大小定为 5，并选取 75 作为是否作递归调用的分界点。这 2 点保证了 T(n)的递归式中 2 个自变量之和 n/5+3n/4=19n/20=εn

最接近点对 **O(nlogn)**

循环赛日程表 **O(nlogn)**

将所有的选手分为两半，n 个选手的比赛日程表就可以通过为 n/2 个选手设计的比赛日程表来决定。递归地用对选手进行分割，直到只剩下 2 个选手时，比赛日程表的制定就变得很简单。这时只要让这 2 个选手进行比赛就可以了。

三、动态规划的基本要素:

1.**最优子结构**：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。

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。

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)**

(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**

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]=A_i，因此，m[i,i]=0，i=1,2,···,n

当 i<j 时，m[i,j]=m[i,k]+m[k+1,j]+$p_{i-1}p_kp_j$ Ai 的维数为 $p_{i-1}*p_i$ k 的位置有 j-i 种可能

$$\mathbf{m[i,j]}=\begin{cases}\mathbf{0} & \boldsymbol{i=j}\\ \min\limits_{i\le k<j}\{\boldsymbol{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j}\} & \boldsymbol{i<j}\end{cases}$$

(3)**计算最优值**

依据递归式自底向下的方式计算，输入参数{p_0,p_1···p_n}存储于数组 p 中，除了输出最优值 m 外，还输出最有断开位置的数组 s。首先计算出 m[i][i]=0,i=1,2···m-1;根据递归式按矩阵链长递增的方式计算 m[i][i+1],m[i][i+2]，只用到计算过得值

(4)**构造最优解**

最长公共子序列 **O(mn)**

(1) **最优子结构性质**：设序列 X={x1,x2,···,xm} 和 Y={y1,y2,···,yn}的最长公共子序列为 Z={z_1,z_2,···,z_k}，则 1)若 x_m=y_n，则 z_k=x_m=y_n，且 z_{k-1}是 x_{m-1}和 y_{n-1}的最长公共子序列；2)若 x_m≠y_n 且 z_k≠x_m，则 Z 是 x_{m-1}和 Y 的最长公共子序列；3)若 x_m≠y_n 且 z_k≠y_n，则 Z 是 X 和 y_{n-1}的最长公共子序列。

$$c[i][j]=\begin{cases}0 & i=0,j=0\\ c[i-1][j-1]+1 & i,j>0;x_i=y_j\\ \max\{c[i][j-1],c[i-1][j]\} & i,j>0;x_i\ne y_j\end{cases}$$

(2)**子问题的递归结构**：(3)计算最优值 c[i][j]存储长度，b[i][j]记录 c[i][j]的值由哪个子问题得到 (4)构造最长公共子序列

证明:(1)用反证法。若$z_k\ne x_m$,则{z1,z2,···,z_k,x_m}是 X 和 Y 的长度为 k+1 的公共子序列。这与 Z 是 X 和 Y 的一个最长公共子序列矛盾。因此,必有$z_k=x_m=y_n$。由此可知Z_{k-1}是X_{m-1}和Y_{n-1}的一个长度为 k-1 的公共子序列。若X_{m-1}和Y_{n-1}有一个长度大于 k-1 的公共子序列 W, ,则将x_m加在其尾部产生 X 和 Y 的一个长度大于 K 的公共子序列，此为矛盾。故Z_{k-1}是X_{m-1}和Y_{n-1}的一个最长公共子序列。

(2)由于$z_k\ne x_m$,Z 是X_{m-1}和 Y 的一个公共子序列。若X_{m-1}和 Y 有一个长度大于 k 的公共子序列 W,则 W 也是 X 和 Y 的一个长度大于 K 的公共子序列。这与 Z 是 X 和 Y 的一个最长公共子序列矛盾。由此即知,Z 是X_{m-1}和 Y 的一个最长公共子序列。

(3)证明与(2)类似。

上述性质告诉我们,两个序列的最长公共子序列包含了这两个序列的前缀的最长公共子序列。因此,最长公共子序列问题具有最优子结构性质。

最大子段和 **分治法 O(nlogn) 动规：O(n)空间和时间**

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{v_{i-1},v_i,···,vj}的最优三角剖分所对应的权函数值

$$\mathbf{t[i][j]}=\begin{cases}\mathbf{0} & \boldsymbol{i=j}\\ \min\limits_{i\le k<j}\{\boldsymbol{t[i][k]+t[k+1][j]+w(v_{i-1}v_kv_j)}\} & \boldsymbol{i<j}\end{cases}$$

(3)计算最优值 (4)构造最优值

多边形游戏时间 O(n3) 图像压缩 O(n) 电路布线 O(n2) 流水作业调度时间 O(nlogn)空间 O(n)

0-1 背包问题 **O(nc) 改进 O(min{nc,2n})**

(1)给定 n 种物品和一背包。物品 i 的重量是 wi，其价值为 vi，背包的容量为 C。问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?

$$max\sum_{i=1}^{n}v_ix_i\quad\begin{cases}\sum_{i=1}^{n}w_ix_i\le C\\ x_i\in\{0,1\}\ 1\le i\le n\end{cases}$$

(2)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,y3,···,yn)是下面相应子问题的一个最优解：

$$max\sum_{i=2}^{n}v_ix_i\quad\begin{cases}\sum_{i=2}^{n}w_ix_i\le C-w_iy_1\\ x_i\in\{0,1\}\ 2\le i\le n\end{cases}$$

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

(3)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。

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

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

最优二叉搜索树:

四、贪心算法（贪心选择性质、最优子结构性质）:

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，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。

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。

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。

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，（x_1,x_2,…… x_n,）是最优装载问题的一个最优解。又设 k=min{ i | x_i=1}{ 1≤i≤n}.易知，如果给定的最优装载问题有解，则 1≤k≤n;证明：

当 k=1 时，（x_1,x_2,···x_n,）是满足贪心选择性质的最优解。

当 k>1 时，取$y_1=1,y_k=0,y_i=x_i$,1<i≤n,i≠k,则 $\sum_{i=1}^{n}w_iy_i=w_1-w_k+\sum_{i=1}^{n}w_ix_i\le\sum_{i=1}^{n}w_ix_i\le c$

因此（y_1,y_2,···y_n,）是所给最优装载问题的可行解，又因为$\sum_{i=1}^{n}y_i=\sum_{i=1}^{n}x_i$知，（$y_1,y_2$,···$y_n$,）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（x_1,x_2,…… x_n,）是最优装载的满足贪心选择性质的最优解，易知，x_1=1，（x_2,x_3,…… x_n,）是轮船载重量为 c-w_1，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：**时间复杂度：O(n2)**

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。

2.**贪心选择性质**：Dijkstra 算法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。

3.**最优子结构性质**：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。

最小生成树 Prim 算法 **(点)**

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。

证明：设 G 的任何一棵最小生成树都不含边(u,v),将边(u,v)添加到 G 的一棵最小生成树 T 上，将产生给含有边(u,v)的圈，并且在这个圈上有一条不同于(u,v)的边(u',v')使 u'∈U，v'∈V-U，将边(u',v')删除，得到 G 的另一棵生成树 T'，由于 c[u][v]<=c[u'][v']，所以 T'的耗费<=T 的耗费。于是 T'是一棵包含边(u,v)的最小生成树，与假设矛盾。

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，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（边）

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

五、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。常用剪枝函数：用约束函数在扩展结点处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:舍开始时 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

七、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$a_0=d;a_n$=(b$a_{n-1}+c$)mod m

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。这是随机性理论研究的内容，已超出本书讨论的范围。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:

基本思想：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$t_A(x)$。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$\bar{t}_A(n)$=$\sum_{x\in X_n}\frac{t_A(x)}{|X_n|}$,这显然不能排除存在 x∈Xn 使得$t_{A(x)}\gg\bar{t}_A(n)$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$t_B(x)=\bar{t}_A(n)+s(n)$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2^k-1，2^{k-1}-1，···，2^0-1 个中间结点。第 i 个 k 级结点安排在跳跃表的位置 i2^k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；···；(100/2^(k+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 1/2^(k+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2^i-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:

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

```
void obstinate(Object x, Object y)
  {// 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
     bool success= false; while (!success) success=lv(x,y);
  }
```

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)(1-p(x))/(p(x))

N 后问题：在棋盘上相继的各行中随机地放置皇后，并注意使新放置的皇后与已放置的皇后互不攻击，直至 n 个皇后均已相容地放置好，或已没有下一个皇后的可放置位置时为止；如果将上述随机放置策略与回溯法相结合，可能会获得更好的效果。可以先在棋盘的若干行中随机地放置皇后，然后在后继行中用回溯法继续放置，直至找到一个解或宣告失败。随机放置的皇后越多，后继回溯搜索所需的时间就越少，但失败的概率也就越大。

整数因子分解：设 n>1 是一个整数。关于整数 n 的因子分解问题是找出 n 的如下形式的唯一分解式:n=$p_1^{m_1}p_2^{m_2}\dots p_k^{m_k}$,其中，p1<p2<···<pk 是 k 个素数，m1,m2,···,mk 是 k 个正整数。如果 n 是一个合数，则 n 必有一个非平凡因子 x，1<x<n，使得 x 可以整除 n。给定一个合数 n，求 n 的一个非平凡因子的问题称为整数 n 的因子分割问题。设位数 m=⌈log10(1+n)⌉，算法 Split(n)是关于 m 的指数时间算法。

Pollard 算法：在开始时选取 0 ~ n-1 范围内的随机数，然后递归地由$x_i=(x_{i-1}{}^2-1)mod\ n$,产生无穷序列：$x_1x_2\dots x_k$...对于 i=$2^k$，以及 2^k<j≤2^{k+1}，算法计算出 x_j-x_i 与 n 的最大公因子 d=gcd(x_j-x_i，n)。如果 d 是 n 的非平凡因子，则实现对 n 的一次分割，算法输出 n 的因子 d。对 Pollard 算法更深入的分析可知，执行算法的 while 循环约 次后，Pollard 算法会输出 n 的一个因子 p。由于 n 的最小素因子 p≤$\sqrt{n}$，故 Pollard 算法可在 O($n^{1/4}$)时间内找到 n 的一个素因子。

蒙特卡洛算法：在实际应用中常会遇到一些问题，不论采用确定性算法或概率算法都无法保证每次都能得到正确的解答。蒙特卡罗算法则在一般情况下可以保证对问题的所有实例都以高概率给出正确解，但是通常无法判定一个具体解是否正确。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。如果重复调用一个一致的(1/2+ε)正确的蒙特卡罗算法 2m-1 次，得到正确解的概率至少为 1-δ，其中

$$\delta=\frac{1}{2}-\varepsilon\sum_{i=0}^{m-1}\binom{2i}{i}\left(\frac{1}{4}-\varepsilon^2\right)^i\le\frac{(1-4\varepsilon^2)^m}{4\varepsilon\sqrt{\pi m}}$$

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

主元素问题：设 T[1:n]是一个含有 n 个元素的数组。当|{i|T[i]=x}|>n/2 时，称元素 x 是数组 T 的主元素。 对于任何给定的ε>0，算法 majorityMC 重复调用⌈log(1/ε)⌉次算法 majority。它是一个偏真蒙特卡罗算法，且其错误概率小于ε。算法 majorityMC 所需的计算时间显然是 O(nlog(1/ ε))

素数测试:

Wilson 定理：对于给定的正整数 n，判定 n 是一个素数的充要条件是(n-1)!≡ -1(mod n)。

费尔马小定理：如果 p 是一个素数，且 0<a<p，则a^{p-1}(mod p)。

二次探测定理：如果 p 是一个素数，且 0<x<p，则方程

x^2≡1(mod p)的解为 x=1，p-1。

八、现行规划问题和单纯性算法:

单纯形算法:

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。

$$flow(v,w)=\begin{cases}flow(v,w)+d & (v,w)\in P^{+}\\ flow(v,w)-d & (v,w)\in P^{-}\\ flow(v,w) & (v,w)\notin P\end{cases}$$

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。

2.**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。

算法复杂性：(1)整个算法找可增广路的次数；(2)每次找可增广路所需时间

设给定的网络中有 n 个顶点和 m 条边，且每条边的容量不超过 M。可以证明，在一般情况下，增广路算法中找增广路的次数不超过 nM 次。

最短增广路算法在最坏情况下，找可增广路的次数不超过 nm/2 次。找 1 次可增广路最多 O(m)。最坏情况下最短增广路算法所需的计算时间为 O(nm2)。当给定的网络是稀疏网络，即 m=O(n)时，最短增广路算法所需的计算时间为 O(n3)。 最大容量增广路算在最坏情况下找可增广路的次数不超过 2mlogM 次。由于使用堆来存储优先队列，找 1 次增广路最多需要 O(nlogn)计算时间。在最坏情况下，最大容量增广路算法所需的计算时间为 O(mnlognlogM)。 当给定的网络是稀疏网络时，最大容量增广路算法所需的计算时间为 O(n2lognlogM)。

九、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。有限状态就是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

p 类与 NP 类问题:

1.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

2.多项式时间验证:

定理 1：VP=NP 证明：先证明 VP⊆NP。对于任意 L∈VP，设 p 是一个多项式，A 是一个多项式时间验证算法，则下面的非确定性算法接受语言 L：(1)对于输入 X，非确定性地产生一字符串 Y∈Σ*；(2)当 A(X,Y)=1 时接受 X。

该算法的步骤(1)与团问题的第 2 阶段的非确定性算法一样，至多在 O(|X|)时间完成。步骤(2)的计算时间是|X|和|Y|的多项式，而|Y|<=p(|X|)。因此，它也是|X|的多项式。整个算法可在多项式时间内完成。因此，L∈NP，VP⊆NP。

反之，设 L∈NP，L∈Σ*，且非确定性图灵机 M 在多项式时间 p 内接受语言 L。设 M 在任何情况下只有不超过 d 个的下一个动作选择，则对于输入串 X，M 的任一动作序列可用{0，1,…d-1}的长度不超过 p(|X|)的字符串编码。不失一般性，设|Σ|≥d.验证算法 A(X,Y)用于验证“Y 是 M 上关于输入 X 的一条接受计算路径的编码”。即当 Y 是这样一个编码时,A(X,Y)=1。A(X,Y)显然可在多项式时间内确定性地进行验证，且 L={X|存在 Y 使得|Y|≤p(|X|)且 A(X,Y)=1}。因此 L∈VP，VP⊇NP。 综上，VP=NP。

3. NP 完全问题:

(1)多项式时间变换:设 L1⊆Σ1*,L2⊆Σ2*是两个语言。所谓语言 L1 能在多项式时间内变换为语言 L2(简记为 L1∝pL2)是指存在映射 f: Σ1* -> Σ2*,且 f 满足:

(1)有一个计算 f 的多项式时间确定性图灵机；(2)对于任意的 x∈Σ1*，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'∝pL。

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

证明:

(1)若 P=NP,则显然 L∈P。反之，设 L∈P,而$\mathbf{L_1}$∈NP,则 L 可在多项式时间$\mathbf{p_1}$内被确定性图灵机 M 所接受。又由 L 的 NP 完全性知$\mathbf{L_1 \propto pL}$,即存在映射 f,使 L=f($\mathbf{L_1}$)。

设 N 是在多项式时间$\mathbf{p_2}$内计算 f 的确定性图灵机。用图灵机 M 和 N 构造识别语言$\mathbf{L_1}$的算法 A 如下:

1)对于输入 x,用 N 在$\mathbf{p_2}$(|x|)时间内计算出 f(x);

2)在时间|f(x)|内将读写头移到 f(x)的第一个符号处;

3)用 M 在时间 p(f|xl)内判定 f(x)∈L。若 f(x)∈L,则接受 x,否则拒绝 x。

上述算法显然可接受语言 L ,其计算时间为$\mathbf{p_2}$(|x|) + |f(x)|+ $\mathbf{p_1}$(f|x|)。由于图灵机一次只能在一个方格中写入一个符号,故 If(x)|≤|xl 十$\mathbf{p_2}$(|x|)。因此,存在多项式 r 使得$\mathbf{p_2}$(|x|)+If(x)| + $\mathbf{p_1}$(f|x|)≤r(x)。因此,$\mathbf{L_1}$∈P。由$\mathbf{L_1}$的任意性即知 P=NP。

(2)只要证明对任意的 L'∈NP,有$\mathrm{L}' \propto \mathrm{p}L_1$。由于 L 是 NP 完全的,故存在一个多项式时间变换 f 使 L=f(L')。又由于$\mathrm{L} \propto \mathrm{p}L_1$,故存在一个多项式时间变换 g 使L_1=g(L)。因此，若取 f 和 g 的和复合函数 h=g(f),则L_1 =h(L')。易知 h 为一多项式。因此$\mathrm{L}' \propto \mathrm{p}L_1$。由 L'的任意性即知$L_1$∈NPC。

从定理 9-2 的(1)可知,如果任一 NP 完全问题可在多项式时间内求解,则所有 NP 中的问题都可在多项式时间内求解。反之,若 P≠NP,则所有 NP 完全问题都不可能在多项式时间内求解。

定理 9-2 的(2)实际上是证明问题的 NP 完全性的有力工具。一旦建立了问题 L 的 NP 完全性后,对于L_1∈NP,只要证明问题 L 可在多项式时间内变换为L_1,即$\mathrm{L} \propto \mathrm{p}L_1$,就可证明$L_1$也是 NP 完全的。

典型的 NP 完全问题:

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 η=max{c/c^* ,c^*/c} 。在通常情况下，该性能比是问题输入规模 n 的一个函数ρ(n),
即 max{c/c^* ,c^*/c} ≤ρ(n).

近似算法的相对误差：定义为 γ= |(c-c^*)/c^* | 。若对问题的输入规模 n，有一函数ε(n)使得 |(c-c^*)/c^* |≤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:

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。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=$\cup_{S\in F} S$。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=$\cup_{S\in C} S$ ，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。