**算法：有输入输出的、有限的、确定的、有效的过程**

**程序：实现算法的一种方式**

**递归分治，动态规划，贪心，回溯分支（数学推导）**

算法设计：抽象为数学模型、写求解过程

随机算法：按照一定概率进行选择

描述：自然语言、流程图、程序语言、Pseudocode

算法评估：Empirical（实验结果->理论推导），Theoretical（算法效率：时间/空间）

W(n)：最坏情况时间复杂度

A(n)=∑P()\*t：平均时间复杂度

**基本运算的执行次数**（加减、比较）

时间复杂度：时间的函数表示基本运算 （I：输入 N：问题规模）

复杂性的**渐近性态**：舍弃低阶项，不必考虑常数因子（**输入规模趋近于无穷**）

**渐近上，下界记号**：

**O**（存在c与n0，使n>=n0都有f(n)<=c\*g(n),则f(n)=O(g(n))（<=）

**/=最坏情况下时间复杂度（输入I相关）**

**Ω**（存在c与n0，使n>=n0都有f(n)>=c\*g(n),则f(n)= Ω(g(n))（>=）

紧渐近界限记号：**Θ**（c1，c2，n0被两个相似函数夹住 **O且Ω**）（=）

非紧上，下界限：**o**（f(n)/g(n)->0），**ω**（f(n)/g(n)->∞）（<，>）

Determinism（确定性算法）：每一步有确定选择

Polynomial（P类问题）：多项式时间内**求解**

Non- Determinism（非确定性算法）：穷举并用确定性算法验证

Non- Deterministic Polynomial（**NP**）：多项式时间内**可验证**

Non- Deterministic Polynomial Complete Problem（NPC NP完全问题）：NP问题可以归约至NPC问题

Non- Deterministic Polynomial Hard Problem NP难问题：**不一定为NP问题**，NPC问题可归约到NP难问题（不一定在多项式时间内**可验证**）

**问题复杂度**不会超过**解决其的算法**的复杂度

判定问题（decision problem）：证明易于求解

判定形式的NP完全问题的**最优化**为NP难问题

TSP：时间复杂度n! 最优化：min{∑d（Ck,Ck+1）+d（Ck,C1）} （d（Ck,Ck+1）存在）

**递归分治**：

分治：分解问题，解决相同子问题 递归：直/间接调用自身（终止，通项条件）

递归：

双递归函数：Ackerman函数

整数划分：q(n, m) m为最大加数

1. q(n, 1)=1 （n>=1）
2. q(n, m)=q(n, n) （m>=n）
3. q(n, n)=1+q(n,n-1)
4. q(n, m)=q(n-m, m)+q(n,m-1) (n>m>1)

**Hanoi塔**：Hanoi(n, A, B, C)（从A利用B移到C）

1. n=1, move(A, C)
2. Hanoi(n-1, A, C, B)
3. Move(A, C)
4. Hanoi(n-1, B, A, C)

分治：

问题具有**最优子结构性质**（二分搜索）

分解问题，解决小问题，合并，**子问题相互独立**（非独立则使用动态规划）

分治：

大整数乘法：

X=a(n/2位)b(n/2位) Y=c(n/2位)d(n/2位)

XY=ac2^n+(ad + bc) \*2^(n/2)+bd

**快速傅里叶变换**

Strassen矩阵乘法：

将矩阵分为大小相等的子矩阵 ，使用分治法降阶求出子矩阵

T(n)=8T(n/2)+O(n^2)

排序问题：

二分归并排序：划分，求解子问题，合并（时间复杂度O(n log n)，稳定）

快速排序：选择轴值与右指针，从右开始扫描，遇小交换，从左扫描（不稳定 平均情况**O(n log n)** **差消法**）

最近点对问题：分为左右两个点集，再考虑跨越边界的点对

**递推方程：T（n）=2T（n-1）+1**（终止条件：T(1)=1）

**迭代法**求解（找出T(n)通项公式，**数学归纳法验证**正确性）

**换元迭代法**：将n转换为变元K的递推，

**T(n)=4T(n/2)+O(n)**

T(n)=2T(n/2)+n-1，令n=2^k

**T(n)=a\*T(n/b)+f(n)**

**公式：T(n)=n^(log b a)+∑a^ i f(n/b^ i )**

**递归树**验证换元迭代法：

T(N)等于树上所有节点的值（非函数项）

W(m)=（函数项）W(m1)+…+W(m k)+（节点值 非函数项）f(m)+…+g(m)

**主定理Master：**T(n)=a\*T(n/b)+f(n)

1. 若f(n)**阶小**（<O(n^(log b^ a)-e)），则T（n）=Θ(n^(log b^ a))
2. 若与f(n)**同阶 (Θ)**，则T（n）=Θ(n^(log b^ a) \* log n)
3. 若f(n)**阶大**(>Ω(n^(log b^ a)+e))，且**a\*f(n/b)≤c\*f(n)（c<1）**，则T（n）=Θ(f(n))

主定理三的两个条件需同时满足才能使用

**动态规划：**

**子问题及其间依赖关系确定（填表+追踪解）**

**设计备忘录**

最优化问题：解空间中满足约束条件的可行解中可使目标函数取得极值的解

最优化原理：**多段决策过程**，子问题为原始问题的一个子集

**最优子结构**+**重叠子问题性质**：最大子问题的最优解包含其他**子问题最优解**

**D**ynamic **P**rogramming：重叠子问题，最优子结构性质,通过表记录已知结果，避免重复计算 **选择向量**记录选择结果

1. 建模，寻找**目标函数**与**约束条件**
2. 分段，将问题**分解为子问题**，确定子问题**边界**
3. 分析，原/子问题间**依赖关系**
4. 判断，是否满足**最优子结构性质**
5. 确定最小子问题初值，**自底向上**，利用已知结果简化计算**选择向量**

最短路径问题：

子问题界定：后边界不变，前边界前移

设计要素：目标函数，约束条件，边界，递推方程，最优化原则，最小子问题界定

**矩阵连乘：加括号划分至只有两两连乘**（简化矩阵连乘运算）

A<i,j>，B<j,k> AB相乘共需ijk次运算

迭代：时间复杂度低，空间消耗多

**保存子问题的解**结果并**标记决策**（m 子问题最少次数，s 划分方式k的值）

1. 确定链长r 1~n
2. 左右边界i，j
3. 遍历确定划分位置k
4. 更新解 m[i, j]

时间复杂度：（三次O(n)循环，个备忘录一个O(n)）

动态规划时间复杂度：备忘录计算+各项合并时间

**最长公共子序列LCS：**

子序列：从给定序列中按序任意选出一段

**L**ongest **C**ommon **S**ubsequence：序列X，Y的最长的相同子序列

穷举：个子序列，

DP：

构造解O(m\*n)，追踪解O(m+n)

空间复杂度O(m\*n)

1. **i=0/j=0，C[i, j]=0(设置初值**C[i, j]=0**)**
2. 若Xn=Yn， Zk为Xn-1与Yn-1的LCS+1
3. 若Xn≠Yn，Zk=max{ C[i, j-1]，C[i-1, j] }

标记函数B[i, j]，左上 ↖（LCS+1）/左 ← (C[i, j-1])/上↑(C[i-1, j])

**背包问题KP：**

填表O(n\*b)（**伪多项式时间**算法b为常数：输入规模 **，**）

约束条件∑wi\*xi≤b（最大重量）

物品(vi, wi) 使∑vi\*xi最大

线性规划：目标函数与约束条件为线性函数

K=1时，Fk(y)初值为向下取整

**贪心算法：**

最优子结构，**贪心选择**性质

**G**reedy **A**lgorithm：每步选择均为**局部最优选择**（最优解的近似解）

选择性质：整体最优可通过局部最优得到，**自顶向下**

**背包问题**：

一般背包（物品数可为小数）：每次均装入性价比最大的

0-1背包（物品为整数）：贪心无法保证背包装满

活动选择问题：

N个活动集合， 开始/结束时间

1. 截止时间排序，选取第一个
2. 遍历比较，若选取

证明：

K=1，假设存在j≠1的活动为A的第一个活动，用1替换必定满足

K=nK=n+1，假设对前K=n为真，证明K+1为真

最优装载问题：

装载能力C，无体积限制（0-1背包子问题）

策略：轻者优先

正确性证明：

**数学归纳法**：

**第一数归：**归纳基础（P(1)为真），归纳步骤（假设P(n)为真，证明P(n+1)真）

**第二数归：**归纳基础（P(1)为真），归纳步骤（假设P(1,2…n)为真，证明P(n+1)真）

描述**与自然数n有关命题**，证明第一个命题成立然后利用数归证明贪心正确

二元前缀码：

0-1代码表示字符，字符代码不能为其它前缀

Huffman哈夫曼树：选取频率最小字符并形成子根，构建向上构建树

最小生成树：n阶联通图G的生成树T无环且含n-1条边

Prim算法：选取结点S{1}，选取到集合S距离最小的顶点j加入S (更新维护堆) （运行n-1次，每次执行）（点少适用）

Kruskal算法：排序并选择最小边，判断不形成回路则加入（并查集：更新父结点，若两结点根节点相同，则不能将它们连接）（边少适用）

单源最短路径：带权有向图，选取原点到其它所有点的最短路径

**Dijkstra算法**：S=s{u}，S=V时结束，维护节点到S最短路径，选取中最小的j加入S并修改（用堆维护，）

多机调度问题NPC：使用贪心**近似**求解

近似比：（贪心策略/最优策略，取及差下限）

**回溯Backtracking：**

**组织解空间，跳跃性穷举**（找出所有解，比较得出最优解）

条件：多米诺性质**（）**进行减枝

深度/宽度优先搜索，满足（扩张解向量，继续搜索），不满足（回溯至父节点）

显约束（解空间），隐约束（约束条件）

4皇后问题：解向量（使用**n叉树**表示解空间）

0-1背包问题： 表示是否选择物品，**子集树**含片树叶

旅行商问题：<k1,k2…,ki>城市排列，个节点的**排列树**

递归：

1. 确定Xi初始取值
2. 计算满足约束的分支Sk
3. 遍历解空间Sk，取出最大节点
4. If(k==n) 为解，否则goto 2

迭代：

1. If(k>n) 为解
2. 遍历Sk，计算Sk+1并调用ReBack(k+1)（跳跃性：递归调用，保存现场）

ReBack（沿右分支遍历向左，遇右遍历右分支，遇根停止）

轮船载重问题：装入第一艘船，使其剩余空间尽量少

1. 遍历wi，若满足x[i]置1，否则置0
2. 记录当前best
3. 回溯（**沿着右分支向上回溯，直到发现为左分支再向右回溯**（至根停止））

图着色问题：解空间（m叉树，一个节点有m种可能着色方案）

每个节点选取可能颜色，若不满足则回溯

**分支限界**

以**广度/最小耗费优先**方式搜索解空间（根据**代价函数**调整搜索，细粒度的回溯）

队列式，栈式优先节点

代价函数：子树区域内所有可行解的上界（极大化 初值为0）（极小化相反）

界：已知最大**可行解**

背包问题：代价函数（已装入价值+（剩余背包\*可行最大性价比））

0-1背包：代价函数高方向搜索（广度优先 使最先找出的界最优）

非对称TSP：优先访问当前最低+未选城市的最低出发票价之和（最小化问题）

**分布式**计算不同分支

最大团问题：

最大团：无向图顶点最多的完全子图

点独立集：U是G的最大团当且仅当U是（G的补图）的最大点独立集

选择树，代价函数

约束条件：与当前被选择所有节点均有边

**随机化算法**

函数难以化简，多次随机产生值代入

随机数值算法：算法输出近似解，精确度与运行时间正比（随机投点，计算比例）

Sherwood算法：，将输入实例排序，**消除最坏行为影响**

Las Vegas算法：随机产生解并带入验证（随机+回溯 设置合适的随机数）

**M**onte **C**arlo算法：获得精确解的概率与算法执行时间正比（反复运行）

主元素：数组中一半以上均为x，寻找是否有主元素

**线性规划与网络流：**

线性规划：目标函数与约束条件均为线性

单纯形法：持续寻找基本可行解（约束条件内）直到找到最优解

网络流：网络G=(V,E)，有源点s与汇点t，边e有最大容量cap(e)

兔子匹配**FF**：

1. 找出一条s->t
2. 构建残余网络，反向路径（边容量=流大小）
3. 于残余网络中重复寻找直至无法找到

从汇点返回到源点提取流