**Chapter 01：引言**

**-----------------------------------------------------------**

**算法：**

* 定义：一系列解决问题的明确指令
* 特性：
  + 确定性：每一步操作无二义性
  + 输入：无输入或有输入
  + 输出：一定有确定的输出
  + 有穷性：在有限步内得到结果
  + 可行性：问题本身有解
* 表示方法：自然语言、伪代码、计算机程序

**Chapter 02：算法效率分析**

**-----------------------------------------------------------**

**分析框架：**

* 测定输入规模
* 确定分析运行时间的单位
  + 直接使用运行时间评估，会导致计算机本身性能等影响变得非常复杂
  + 使用Basic Operation解决：在总的运行时间里占比最高的操作，一般是算法最内层循环的最耗时操作（如排序的值比较，或四则运算的乘除法）
* 增长阶数
  + Basic Operation次数随着输入规模的增长而增长的规模
* 分最坏、最好、平均分析
  + 和输入规模无关，仅和输入的特性有关
  + 平均是一种完全随机的情况，不一定是最坏和最好的平均

**渐进复杂度：**

设算法A的复杂度为T(n)，有函数t(n)满足：



则t(n)是T(n)的渐进态，是算法A的渐进复杂度

**比较：**

* 
  + 判定：若存在函数、正常数和非负整数，使得对于所有的，都有，则
  + 变种：，小写的，去掉等号
* 
  + 判定：若存在函数、正常数和非负整数，使得对于所有的，都有，则
  + 变种：，小写的，去掉等号
* 
  + 判定：若存在函数、正常数和非负整数，使得对于所有的，都有，则
* 用极限比较增长阶数：计算
  + 0，则T(n)<g(n)
  + c(c>0)，则T(n)=g(n)
  + ∞，则T(n)>g(n)
  + 较为复杂时可用洛必达法则
* 注意：
  + 所有对数复杂度都属于同一增长阶数
  + 各个指数复杂度因底数不同而属于不同增长阶数
  + 各个阶乘复杂度因数字不同而属于不同增长阶数

**时间复杂度计算：**

* 非递归算法：
  + 确定输入规模n
  + 确定Basic Operation
  + 检查运行次数是否与输入的特性有关，若有关则分最好、最坏和平均情况讨论
  + 用**函数**C(n)反映Basic Operation的执行总次数
  + 用标准的增长阶数表示C(n)的复杂度
* 递归算法：
  + 确定输入规模n
  + 确定Basic Operation
  + 检查运行次数是否与输入的特性有关，若有关则分最好、最坏和平均情况讨论
  + 建立**递推方程**C(n)及其**初始条件**，反映Basic Operation的执行总次数
  + 解出C(n)（用反向替换法或其他方法）

**Chapter 03：蛮力法**

**-----------------------------------------------------------**

**价值：**

* 简单直接，几乎可以解决所有问题
* 若问题规模限制在很小，而蛮力法能在可接受时间内解决，则无需再考虑更高效的算法
* 可作为衡量其他算法效率的准绳

**缺陷：**

大多效率低，有些甚至成为了NP-hard

**实例：**

* **选择排序**：前i个已有序，从第i+1到n个元素中选最小的交换到i+1位
* **冒泡排序**：每次从第一个开始往后比，比当前小则交换，比当前大则当前元素换到下一个，直到到达有序序列的第一个位置前（即倒数i个已有序）
* **顺序查找**：在一个序列中查找值为k的元素，从头找到尾
* **子串匹配**：在字符串中寻找某个字串的初始位置，每个位置都检查——
* **TSP**：穷举所有路径并找出长度最小的——
* **0-1背包**：穷举所有存放情况并找出价值最大的——
* **任务分配**（一人一任务，i任务给人j的开销为C[i, j]）：穷举所有分配情况并找出开销最小的——

**Chapter 05：减治法**

**-----------------------------------------------------------**

**概念：**

* 同一问题下，利用给定实例的解和较小实例的解之间的关系
* 其中有实例规模逐步变化的过程。可以自顶向下（递归）、自底向上（非递归）解决

**种类：**

* 减常量
* 减可变规模（一种情况：减常因子）

**实例：**

* **减常量**
  + **计算次幂**：
  + **插入排序**：前i个已有序，第i+1个元素在前i个有序队列中找到其位置并插入
    - **最好**：已升序有序——
    - **最坏**：已降序有序——
    - **平均**——
  + **拓扑排序**：每次减去一个入度为0的顶点——（还有一种方法是DFS遍历，出栈顺序的倒序即为排序结果，复杂度相同）
* **减可变规模**
  + **减常因子**
    - **计算次幂**（）
    - **折半搜索**
      * **最坏**：找到只剩最后一个元素，——
      * **最好**：带查找元素在“中间”——
      * **平均**——
    - **假币问题**：用一架天平在n个硬币中找1个较轻的假币
      * 分两份：分成两半（n为奇数时多分出1枚），若两半同样重则多出的1枚为假币；若不一样重则假币在轻的那一半，从那一半重复以上过程
        + **最坏**：——
      * 分三份：分成三份（不能整除时多出的硬币放第三份），比较前两份，若两半同样重则迭代至第三份；若两半不一样重则迭代至轻的那一份
        + **平均**：——
    - **俄式乘法**：正整数n乘m时，若n是偶数，可如此转化：。若n是奇数，可如此转化：。每次迭代至两个相乘的数，并记录n是奇数时要加上的值，当n=1时得出结果
      * **最好**：n=1——
      * **平均**：——
  + **其它**
    - 欧几里得辗转相除法

**复杂度分析：**

* **减常量**：
  + f(n)为将规模为n的问题转化为n-1的问题的用时
  + 用反向替换法得到：
* **减常因子**：
  + f(n)为按常因子减小问题规模，并将小实例的解拓展为大实例的解的用时
  + 基于平滑法则，用反向替换法得到：

**Chapter 06：分治法**

**-----------------------------------------------------------**

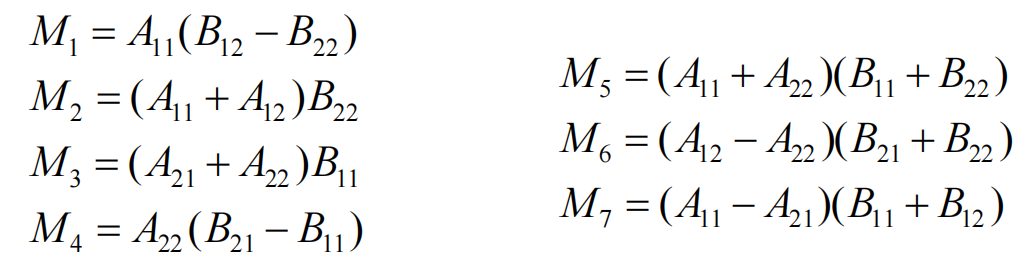
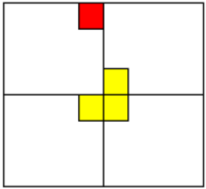
**基本步骤：**

* 将问题分成多个子问题（相同类型、相互独立、规模均衡）
* 递归分解子问题至可以直接解决
* 合并子问题直至原问题得到解决

**通用递推关系式：**

* 形式：
* 说明：
  + a：子问题个数
  + n/b：子问题规模
  + f(n)：划分子问题和合并子问题解的时间
* 主定理：
  + 可以根据通用关系式求增长次数
  + 由于初始状态未知，只能求增长次数，精确解无法解出。只有在得到初始状态后，求递推方程得到精确解
  + 
  + 内容：

**实例：**

* **归并排序**：，——
* **折半搜索**：，——
* **快速排序**：
  + **最好**：分裂点在子数组的中点——
  + **最坏**：数组已经有序——
  + **平均**：分裂点出现位置完全随机——
* **大数乘法**（两个n位数（10进制整数）相乘，**Basic Operation为乘法**）：
  + **蛮力法**：各个位两两相乘——
  + **分治法一**：
    - 按以下方法分解：，。即将两个数对半分
    - 则
    - 递归计算4个n/2位数的乘法，直到剩余1位
    - 复杂度：——**（未有提升）**
  + **分治法二**：
    - 按法一分解两数
    - 由，代入得
    - 递归计算3个n/2位数的乘法，直到剩余1位
    - 复杂度：——**（有提升）**
* **Strassen矩阵乘法**（两个n阶矩阵相乘，**Basic Operation为数的乘法**）：
  + **蛮力法**：逐行逐列逐个相乘——
  + **分治法一**：
    - 将两个矩阵划分成四个等大的子矩阵，再进行相乘
    - 
    - 复杂度：——**（未有提升）**
  + **分治法二**：
    - 按法一分解两矩阵
    - 
    - 其中：
    - 复杂度：——**（有提升）**
* **棋盘覆盖问题**（除一个特殊方格外，用L型骨牌覆盖的棋盘）：
  + 算法：将棋盘划为四个相等的部分，则特殊方格必定位于其中一个部分中，其余三个部分的会合处刚好可以用一个L形骨牌覆盖，就可以作为其余三个部分的特殊方格。就得到了四个完全相同的子棋盘问题
  + 
  + 复杂度：，——

**Chapter 07：变治法**

**-----------------------------------------------------------**

**基本步骤：**

* 把问题的实例变得更容易（变）
* 解决较简单的问题（治）

**变治类型：**

* 实例化简：同问题，简实例
* 改变表现：同实例，换表现形式
* 问题化简：同实例，换问题

**实例：**

* **实例化简**：
  + **预排序——检查所有元素是否唯一**：
    - **蛮力法**：两两比对序列中下标不同的元素——
    - **实例化简**：进行高效排序后检查临近元素是否相等——**（有提升）**
  + **预排序——统计序列中出现最多的元素**：
    - **蛮力法**：用一个辅助数组记录出现的值及其出现频率（创建数组+找最大值）
      * 最坏：——
    - **实例化简**：进行高效排序后查找最长的相同元素序列长度——**（有提升）**
  + **预排序——查找元素**：
    - **蛮力法（顺序查找）**：
      * **最好**：
      * **最坏**：
      * **平均**：——
    - **实例化简**：高效排序+折半搜索——**（未有提升）**
  + **高斯消去法解多元线性方程组**：
    - 前向消去法构造上三角——
    - 回溯替换求得各个解——
  + **堆排序**：
    - 堆构建：最坏情况是每个结点都要到达最底层——
    - 堆排序：每次选最大的从堆中删除——
* **改变表现**：
  + **霍纳法则**（求多项式在一点的值，Basic Operation为乘法）：
    - 按指数降序排列多项式，指数为n的系数为P[n]，得到数组P
    - result = P[n]
    - 从n-1到0，result = x \* result + P[i]
    - 复杂度：
* **问题化简**：
  + **判断三个点的位置关系**：转换为行列式的符号判断——
  + **线性规划**：0-1/分数背包问题、投资问题
  + **状态问题**：过河、下棋等（用状态图解决）

**Chapter 08：回溯法&分支限界法**

**-----------------------------------------------------------**

**盲目搜索：**

* 生成所有潜在解的列表
* 选出满足约束的解
* 逐个评估解，去掉不可行的（对于优化问题，跟踪目前找到的所有最优解）
* 搜索结束时得出解

**解的相关概念：**

* 解向量：将一个问题的解表示成n元式的形式
* 显约束：每个值的约束
* 隐约束：为满足问题而对不同值之间的约束
* 解空间：满足所有显约束的解向量

**回溯法：**

* 特性：
  + 系统性：在解空间树中按深度优先策略从根结点出发搜索
  + 跳跃性：搜索至一结点时，判断以该结点为根的子树是否包含解。若有则进入子树继续搜索；若无则跳过对子树的搜索，向其祖先回溯
* 基本思想：
  + 从解空间树根结点出发，按深度优先策略搜索
  + 开始结点变为活结点，也是扩展结点。向下搜索到达新结点时，新结点成为新的活结点，也是当前的扩展结点
  + 若在当前扩展结点处不能再向下搜索，且没有到达叶结点找到解，则扩展结点成为死结点
  + 此时应回溯到最近的活结点处继续搜索，直到搜索完全部可行区域为止
* 基本步骤：
  + 定义问题的解空间
  + 确定易于搜索的解空间结构（子集树、排列树）
  + 进行深度优先搜索（**可以**使用剪枝函数避免无效搜索）
* 剪枝函数：
  + 约束函数：剪去不满足约束的子树
  + 限界函数：剪去得不到最优解的子树
* 算法框架：
  + 子集树直接遍历扩展结点的所有孩子进行递归
  + 排列树由于事先生成有序排列，到第t层（t≥1）需确定前t-1个值，而第t-1位有t-n+1个取值，在x[t]到x[n]中。因此，t≤i≤n进行循环，每次将x[t]和x[i]交换确定第t-1位并递归，递归出来后再交换一次回到递归前状态

**分支限界法：**

* 基本思想：
  + 以广度优先或最值优先方式进行搜索
  + 裁剪不能得到最优解的子树
  + 在扩展结点处先将所有孩子加入活结点表，选择一个作为下一扩展结点。可以通过计算各活结点的某种函数值，通过比较得出“最有利”的一个活结点作为扩展结点，从而加速搜索
* 基本步骤：
  + 定义问题的解空间
  + 确定易于搜索的解空间结构
  + 进行广度优先搜索
    - 删除不可能产生最优解的活结点
    - 从活结点表中选择“最优”的活结点继续搜索

**实例：**

* **回溯法**：
  + **排列生成问题**（**排列树**）
  + **TSP问题**（**排列树**，**可限界**）
  + **n皇后问题**（**排列树**）
  + **0-1背包问题**（**子集树**，**可限界**）：
    - 限界函数：当前背包价值
      * 左子树：
      * 右子树
* **分支限界法**：
  + **0-1背包问题**（**优先队列式**）：
    - Bound计算：
      * 剩余部分用未放入的最大价值密度填补
      * 继续按顺序无条件装入，直到第m个放不进时，用第m个的价值密度填补剩余部分
  + **装载问题**（**队列式**）

**回溯与分支限界的区别：**

* 目标不同：回溯要找到所有解，分支限界要尽快找到一个解
* 搜索方法：回溯用深度优先，分支限界用广度优先或最值优先
* 扩展方式：分支限界的结点只有一次成为扩展结点的机会，因为一次就能产生所有分支；而回溯法由于一次产生一个分支，节点可能多次成为扩展结点
* 空间开销：分支限界由于存储所有的活结点，开销比回溯大

**回溯与穷举的区别联系：**

* 联系：都是基于试探的搜索方法
* 区别：穷举在解完全生成后才进行判断，若不满足条件会完全放弃；回溯法的解是逐步生成的，若不满足条件仅放弃最近一步的操作

**Chapter 09：动态规划**

**-----------------------------------------------------------**

**基本概念：**

* 动态规划：主要用于解决以时间划分阶段的动态过程的优化问题，既是一种考察问题的方式，也是一种解决问题的方法
* 阶段：问题求解过程中相互联系的部分
* 状态：每个阶段开始时，问题或系统处于的客观状况。其具有马尔可夫性（状态的无后效性）：某阶段状态给定后，该阶段以后的过程发展不受该阶段以前各阶段状态的影响
* 策略：各个阶段决策组成的序列（从某个阶段到最终阶段为子过程，对应的决策序列为子策略）

**思想实质：**

分治和解决冗余

**基本步骤：**

* 找出最优解性质，刻画结构特征
* 递归定义最优值（得出相关递推方程）
* 计算最优值（自顶向下或自底向上）
* 根据计算最优值时的记录信息，构造最优解
  + 使用动态规划往往只能求得最终结果，却无法记录下如何得到最终结果的
  + 因此需要最优解的构造方法时，需手动记录额外信息

**求解方法：**

* 自顶向下：备忘录法，将求过的子问题解记录下来，需要时启用
* 自底向上：可以进一步压缩空间，避免使用递归

**适用动态规划的条件：**

* 最优子结构：问题的最优解通过其子问题的最优解构造
* 重叠子问题：假如使用递归求解，会反复计算到相同的子问题，而不会生成新的子问题

**实例：**

* **计算组合数**：
  + **递归定义**：
  + **复杂度**：
* **计算最长离散子串**：
  + **递归定义**：
  + **复杂度**：
  + **信息记录**：
    - **记录方法**：创建辅助数组b[i, j]。若xi=yj，b[i, j]为左上箭头；否则，若C[i-1, j]>=C[i, j-1]，b[i, j]为上箭头，否则为左箭头
    - **回溯方法**：设最大离散子串为sub，初始为空。从b[i, j]右下角出发，沿箭头回到左上角，途中经过左上箭头时，将对应的X、Y中相等字母添到sub的首位，即可得到sub
* **动态矩阵相乘**：
  + **递归定义**：
  + **复杂度**：
  + **信息记录**：
    - **记录方法**：创建辅助数组s[i, j]，m[i, j]取得最小值的k就是s[i, j]的值
    - **回溯方法**：从s[1, j]开始，按s[1, j]的值进行第1次分割。分割成的两半再查询对应的s值进行递归分割，直到不可分割为止
* **0-1背包问题**：
  + **递归定义**：
  + **复杂度**：
  + **回溯方法**：从V[n, W]出发，设当前到达位置为(i, j)。若V[i, j]=V[i-1, j]，则往上一格且第i个元素不放入；否则到达(i-1, j-wi)且第i个元素放入。回到(0, 0)时所有物品的放入情况已经得到
* **多段图问题（单源单终点）**：
  + **递归定义**：
    - d[i][j]指第i段第k个结点到源点的最短路径长度
    - w[i][j][k]指第i段第j个结点到下一段第k个结点的边长）：
    - 
  + **复杂度**：
  + **信息记录**：
    - **记录方法**：创建辅助数组path[i][j]。当计算d[i][j]时，发现最短路径来自于上一段的第k个结点，则path[i][j]=k
    - **回溯方法**：设有N段，当前点是(i, j)。从(N-1, 0)出发，pre=path[i][j]，然后到达(i-1, pre)。重复以上过程回到原点，经过的路径就是最短路径
* **Warshall算法**：
  + **递归定义**：
    - R(k)：k=0时是原图的邻接矩阵。若对任意一个结点v，对于所有的边i→v和v→j，添上了边i→j，那么k就会+1。k=n时就是传递闭包的邻接矩阵
    - rij(k)：在R(k)中i行j列的元素（表示结点i能否到达j）
    - 
  + **复杂度**：
* **Floyd算法**：
  + **递归定义**：
    - D(k)：k=0时是原图的邻接矩阵（权重）。若对任意一个结点v，对于所有i→v和v→j的权重和小于i→j，将i→j的权重改为两者权重和，那么k就会+1。k=n时就能求出两两节点间的最短距离
    - dij(k)：在D(k)中i行j列的元素（表示结点i到j的最短距离）
    - wij：原图中i到j的边长
    - 
  + **复杂度**：

**最优性原理证明：**

* 证明符合：
  + 设有某个最优解S=s+r，但其中的一部分，即某个子问题的解s不是最优的
  + 设子问题的最优解为s’
  + 构造总问题新解S’=s’+r
  + 证明S优于S’
  + 用反证法得出，最优的S能带出最优的s
* 证明不符合：
  + 找出两个子问题的解s、r
  + 证明S=s+r不是总问题的最优解

**与分治法比较：**

* 相同：都是将问题分解成子问题求解
* 不同：子问题之间往往不独立

**Chapter 10：贪心算法**

**-----------------------------------------------------------**

**定义：**

每一步做出局部最优的决策，期望得到全局最优解的算法

**单步决策特点：**

可行、局部最优、不可撤销

**应用场景：**

* 查找最优解：
  + 找零钱问题（部分实例）
  + 最小生成树问题
  + 单源最短路径问题
  + 哈夫曼编码
* 多数情况下无法得到最优解，但可以快速查找近似解：
  + TSP问题
  + 0-1背包问题
  + （其余优化问题）

**实例：**

* **Dijkstra算法（单源最短路径）**：选择V-S中具有最短特殊路径的顶点u，从而确定源点到u的最短路径
* **Prim算法（最小生成树/选点法）**：选择T和V-T间连接的最短边
* **Kruskal算法（最小生成树/选边法）**：选择待选边集合中最短的边

**与动态规划的区别与联系：**

* 区别：
  + 主要区别：贪心通过局部最优达到全局最优
  + 子问题：动态规划要依靠子问题的解；贪心算法直接做出最优选择，然后解决决策做出后的子问题，不依赖子问题的解
  + 求解方式：动态规划一般自底向上；贪心算法一般自顶向下
* 联系：都具有最优子结构的性质