算法

1. 基本概念

1. 算法：
    i. 非形式定义：是若干指令的有穷序列
        a. 输入
        b. 输出
        c. 确定性
        d. 有限性
    ii. 形式定义：
        a. 对所有的有效输入停机的Turing机
        b. 算法A解问题P即把问题P的任何实例作为算法A的输入，A能够在有限步停机，并输出该实例的正确的解
2. 程序，例如操作系统，不是算法，不满足有限性
3. 最优算法：问题的计算时间下界为$\Omega(f(n))$，则计算时间复杂性为$O(f(n))$的算法是最优算法。例如基于数值比较的排序问题的计算时间下界为$\Omega(n\log n)$，计算时间复杂性为$O(\log n)$的排序算法是最优算法，例如堆排序算法
4. 主定理
5. 多项式时间的算法即时间复杂度为n的多项式的算法
6. 不是多项式时间的算法即不存在多项式$p(n)$，使得该算法的时间复杂度为$O(p(n))$，包含指数时间甚至更高阶的算法
7. 多项式时间可解的问题P即存在着解P的多项式时间的算法
8. 难解的问题P即不存在解P的多项式时间的算法
9. 实际上可计算的问题即多项式时间可解的问题
10. 对数多形式时间的算法是高度并行可解的
11. 可计算理论即研究上面问题是可计算的理论，核心论题是$Church-Turing$论题：如果一个函数在某个合理的计算模型上可计算，那么它在Turing机上也是可计算的
12. 可计算性是不依赖于计算模型的客观性质
13. 合理的计算模型有：递归函数、图灵机、λ演算、Post系统等
    i. 计算一个函数只要有限条指令
    ii. 每条指令可以由模型中的有限个计算步骤完成
    iii. 指令执行的过程是确定的
14. 问题：需要回答的一般性提问，通常有若干参数
    i. 问题的实例即对问题的参数的一组赋值

    ii. 问题描述所包含的内容包括对问题参数的一般性描述、解满足的条件
    iii. 一个问题是由它的全体实例构成的集合
15. P类问题即所有可以在多项式时间内求解的判定问题。
16. 判定问题：判断是否有一种能够解决某一类问题的可能行算法的研究课题。
17. NP类问题：所有的非确定性多项式时间可解的判定问题
18. NP问题：可以在P时间内验证给定解是否正确的一类问题

2. 递归与分治

1. 设计思想：将一个难以解决的大问题分解为一些规模较小的相同问题，以便各个击破，分而治之。
2. 直接或间接地调用自身的算法称为递归算法
3. 用函数自身给出定义的函数称为递归函数
    i. 递归函数的两个要素
        a. 边界条件
        b. 递归方程
4. 递归的优点：
    i. 结构清晰，可读性强
    ii. 容易用数学归纳法证明正确性
5. 递归的缺点：
    i. 运行效率低
    ii. 占用存储空间比非递归算法多
6. 分治法能解决的问题所有的特征：
    i. 问题的规模缩小到一定程度后就可以容易地解决
    ii. 问题可以分解为若干个规模较小的相同问题
    iii. 利用该问题分解出来的子问题的解可以合并为该问题的解
    iv. 该问题分解出来的子问题是相互独立的，即子问题之间不包含公共的子问题（不强求，但是如果不满足，会大大影响效率）
7. 理想情况下，分解得到的子问题的规模应该相同
8. 优化分治法的途径：
    i. 通过代数变换减少子问题的个数
    ii. 通过预处理（预先排序等）减少递归内部的计算量

3. 动态规划

3.1. 基本特点

1. 与分治法的类似之处：

    i. 基本思想是将待求解问题分解为若干子问题
2. 与分治法的不同之处：
    i. 动态规划分解得到的子问题往往不相互独立（重叠子问题）。不同子问题的个数常常只有多项式量级
    ii. 动态规划主要利用问题的最优子结构性质来从子问题的最优解逐步构造出整个问题的最优解
3. 基本步骤：
    i. 找出最优解性质，刻画结构特征（最优子结构，全局最优解中包含局部最优解）
    ii. 递归定义最优解
    iii. 计算出最优值，通常采用自底向上的方法
    iv. 根据计算最优值时得到的信息，构造最优解
4. 两种方法：
    i. 自顶向下：
        a. 将问题分解成多个子问题
        b. 得到子问题时，先查看memo中是否记录了对应的解
        c. 如果有，则直接跳过
        d. 否则，解决该子问题，并将答案记录在memo中
    ii. 自底向上：
        a. 先从最小的子问题开始解决
        b. 用子问题拼凑成更大的问题的解
        c. 最终得到原问题的解
5. 最优子结构证明方法：
    i. 设某个解是最优解
    ii. 证明从中得到的子问题也是最优解，常用反证法
6. 动态规划要素：
    i. 最优子结构
    ii. 重叠子问题

3.2. 典型题目

1. 矩阵连乘（$A_1A_2\cdots A_n$）
    i. 特征：若干个连续的最小子问题才能够组成更大的问题
    ii. 方法：将$dp[i,j]$定义为由$A_i\cdots A_j$组成的子问题的最优解
    iii. 延申：
        a. 有些时候，可能要至少两个最小子问题才有意义，例如连成的线，这个时候可以将$dp[i,j]$定义为$A_{i-1}\cdots A_j$
        b. 如果是个圈，则将$m[i,j]$设为由$A_i\cdots A_{i+j-1\mod i}$构成的子问题的解

4. 贪心算法

4.1. 基本特点

1. 适合组合优化问题
2. 求解过程是多步判断的过程，最终的判断序列对应问题的最优解
3. 必须进行正确性证明，利用的关键性质是局部最优性
4. 优势：
    i. 算法简单
    ii. 时空复杂度低
5. 能用贪心算法解决的问题所拥有的特点
    i. 贪心选择性质——整体最优解可以由一系列局部最优的选择达到
    ii. 最优子结构性质
6. 与动态规划的区别：
    i. 贪心算法常以自顶向下的方式进行，将所求问题化为规模更小的子问题，而动态规划常以自底向上的问题解决子问题
    ii. 贪心算法作出的每一步贪心策略都无法改变，每一步的最优解***一定包含上一步***的最优解，而动态规划算法中，全局最优解***一定包含某个***局部最优解，但是***不一定包含前一个***局部最优解，因此要记录之前的所有最优解
7. 与动态规划的类似之处：
    i. 都是一种递推算法，均有最优子结构性质，通过局部最优解来推导全局最优解

4.2. 证明方法

1. 方法一：
    i. 根据贪心选择策略每次得到的是和原问题相同的子问题
    ii. 证明存在最优子结构性质
    iii. 利用数学归纳法证明贪心选择性质
    iv. 第一步需要单独证明
    v. 证明$k+1$步时，存在包含前k步的最优解$\{x_1,\cdots,x_k\}\cup B$，同时由于最优子结构性质，剩余部分B是由前k步得到的子问题的最优解，而由于第一步已经得到证明，所以根据贪心选择策略，做出来的第$k+1$步，即子问题的第一步，存在子问题的一个最优解B'包含它，该最优解与前k步合在一起仍是原问题的最优解，因为$|B|=|B'|$
2. 方法二：
    i. 对问题规模进行归纳假设，命题是根据贪心策略能够对于规模为n的问题得到最优解
    ii. 归纳基础通常十分显然
    iii. 证明规模为$k+1$步时，将规模为$k+1$的问题视为由贪心策略作出第一步后得到的规模为k的子问题，从而利用归纳假设，得到一个根据贪心策略对规模为$k+1$的问题得到的解

    iv. 然后证明该解是最优解，通常利用反证法，矛盾点在于，如果不是最优解，可以将最优解转换为包含由贪心选择作出来的第一步的解，剩余部分也是之前规模为k的子问题的解，并且比由贪心策略得到的解更优，从而矛盾
3. 方法二不需要直接证明最优子结构，适用于一些最优解结构较为复杂的问题，例如哈夫曼编码
4. 有些特殊问题需要特殊处理
    i. 单源最短路径：
        a. 利用归纳假设证明的命题是：按照贪心策略作出的每一步都能确定到一个顶点的最短路径
    ii. Prim算法：
        a. 利用归纳假设证明的命题是：按照贪心策略作出的每一步后已经得到的边都在某颗最小生成树中
        b. 利用MST性质直接证明$k+1$步得到的边包含在第k对应的最小生成树中

5. 回溯法

5.1. 基本特点

1. 用于找出问题的所有解，或者满足某些约束条件的最优解（本质上还是找出所有解再比较）
2. 本质上是深度优先搜索，但是必须要具有限界函数
3. 不断利用限界函数处死不可能产生所需解的活结点
4. 可以利用递归函数，也可以用非递归迭代函数
5. 当所给问题是从n个元素的集合S中找出满足某种性质的子集时
    i. 解空间为子集树
    ii. 分量x_i表示第i个元素是否在要找的子集中
    iii. 遍历需要$\Omega(2^n)$的计算时间
    iv. 用循环的方式来取遍每个分量上的所有可能值
6. 当所给问题是从n个元素的集合S中找出满足某种性质的排列时
    i. 解空间为排列树
    ii. 分量x_i表示第i个位置上的元素是x_i
    iii. 遍历需要$\Omega(n!)$的计算时间
    iv. 利用Swap来取遍所有的全排列

5.2. 求解基本方法

1. 解向量：n元组$(x_1,\cdots,x_n)$
2. 显约束：x_i的取值范围
3. 隐约束：不同分量之间产生的约束
4. 解空间：满足***显式约束条件***的***所有***多元组构成的解空间
    i. 树（绝大多数），在边上标注对应的分量值

    ii. 图
    iii. 第一个结点为空，即使指定了第一个分量的值，例如TSP问题中的其实地点，也是空
5. 基本思路：
    i. 从根开始深度优先搜索
    ii. 到达活结点
        a. 判断是否为（最优）解，是则输出
        b. 得到一个儿子结点（根据显约束生成）
        c. 判断是否满足隐约束
            a. 确定是否能够导致可行解——约束条件
            b. 确定能否能够导致最优解——限界条件
            c. 通常，对于子集树，如果选择加入当前元素，只需要判断约束条件是否成立，如果选择不加入当前元素，只需要判断限界条件是否成立即可
        d. 满足则返回到2（因此扩展结点一定是满足隐约束的结点）
        e. 不满足则继续生成儿子结点
        f. 若没有儿子结点，回溯到父节点
    iii. 找到问题的解或者根节点变为死结点
6. 常用限界函数：
    i. 直接用当前代价与当前已找到最优解进行比较
    ii. 当前部分解代价的上界（一个估计）与当前已找到最优解进行比较
7. 时间复杂度：限界函数时间复杂度×限界函数调用次数＋约束函数时间复杂度×约束函数调用次数

6. 分支限界法

6.1. 基本特点

1. 求解目标是满足约束条件的一个解，或者是在满足约束条件的解中找到某种意义下的最优解（不是通过枚举所有解得到）
2. 与回溯法的区别：
    i. 回溯法的求解目标是找出解空间树种满足约束条件的所有解
    ii. 分支限界法的求解目标是找出满足约束条件的一个解，或是在满足约束条件的解种找出某种意义下的最优解
    iii. 回溯法以深度优先的方式搜索解空间树；分支限界法以广度优点或以最小耗费优先的方式搜索解空间树
    iv. 分支限界法种，每个活结点只有一次机会成为扩展结点，一旦成为扩展结点，就一次性产生其所有的儿子节点；而在回溯法中，活结点不一次性产生其所有的儿子结点

6.2. 求解基本方法

1. 定义解空间：
    i. 解向量
    ii. 显约束
    iii. 隐约束
2. 确定数据结构：
    i. 可能需要增加指向父节点的指针来还原解序列
3. 确定约束条件和限界条件
4. 确定活结点组织方式（队列/优先队列（优先级函数））
5. 确定识别答案结点的方式（在回溯法中是通过部分解的长度来判断的）
6. 一般队列式分支限界法使用限界函数剪枝，而优先队列式分支限界函数则是将限界函数转化为优先级
7. 常用限界函数
    i. 上界与当前得到的部分解最优解比较（因为是BFS，算法到最后才能够得到完整解，所以必须实时更新最优解）

7. 线性规划

7.1. 基本概念

1. 线性规划数学模型的三要素：
    i. 决策变量
    ii. 目标函数
    iii. 约束条件
2. 可行解即满足约束的一组值
3. 假设等式约束和非负约束的个数分别为m,n，则至少有n个约束以等号满足的可行解成为基本可行解，基本可行解中最多有m个分量非零
4. 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
5. 单纯形一般不经过大于m或n次迭代就可以求得最优解
6. 约束标准型线性规划问题：
    i. 目标函数求最大值
    ii. 约束条件为等式方程，右端常数项非负
    iii. 决策变量非负