算法

1 递归算法

递归算法是一种直接或者间接调用自身函数或者方法的算法。递归算法的实质是把问题分解成规模缩小的同类问题的子问题，然后递归调用方法来表示问题的解。递归算法对解决一大类问题很有效，它可以使算法简洁和易于理解。

1.2 适用场景

递归算法一般用于解决三类问题：

数据的定义是按递归定义的。（斐波那契数列）

问题解法按递归算法实现。（回溯）

数据的结构形式是按递归定义的。（树的遍历，图的搜索）

递归的解题策略：

第一步：明确你这个函数的输入输出，先不管函数里面的代码什么，而是要先明白，你这个函数的输入是什么，输出为何什么，功能是什么，要完成什么样的一件事。

第二步：寻找递归结束条件，我们需要找出什么时候递归结束，之后直接把结果返回

第三步：明确递归关系式，怎么通过各种递归调用来组合解决当前问题

2 分治算法

2.1 算法策略

在计算机科学中，分治算法是一个很重要的算法，快速排序、归并排序等都是基于分治策略进行实现的，所以，建议理解掌握它。

分治，顾名思义，就是 分而治之 ，将一个复杂的问题，分成两个或多个相似的子问题，在把子问题分成更小的子问题，直到更小的子问题可以简单求解，求解子问题，则原问题的解则为阿子问题解的合并。

2.2 适用场景

当出现满足以下条件的问题，可以尝试只用分治策略进行求解：

原始问题可以分成多个相似的子问题

子问题可以很简单的求解

原始问题的解是子问题解的合并

各个子问题是相互独立的，不包含相同的子问题

分治的解题策略：

第一步：分解，将原问题分解为若干个规模较小，相互独立，与原问题形式相同的子问题

第二步：解决，解决各个子问题

第三步：合并，将各个子问题的解合并为原问题的解

3 贪心算法

3.1 算法策略

贪心算法，故名思义，总是做出当前的最优选择，即期望通过局部的最优选择获得整体的最优选择。

某种意义上说，贪心算法是很贪婪、很目光短浅的，它不从整体考虑，仅仅只关注当前的最大利益，所以说它做出的选择仅仅是某种意义上的局部最优，但是贪心算法在很多问题上还是能够拿到最优解或较优解，所以它的存在还是有意义的。

3.2 适用场景

在日常生活中，我们使用到贪心算法的时候还是挺多的，例如：

从100章面值不等的钞票中，抽出 10 张，怎样才能获得最多的价值？

我们只需要每次都选择剩下的钞票中最大的面值，最后一定拿到的就是最优解，这就是使用的贪心算法，并且最后得到了整体最优解。

但是，我们任然需要明确的是，期望通过局部的最优选择获得整体的最优选择，仅仅是期望而已，也可能最终得到的结果并不一定不能是整体最优解。

那么一般在什么时候可以尝试选择使用贪心算法喃？

当满足以下条件时，可以使用：

原问题复杂度过高

求全局最优解的数学模型难以建立或计算量过大

没有太大必要一定要求出全局最优解，“比较优”就可以

如果使用贪心算法求最优解，可以按照以下 步骤求解 ：

首先，我们需要明确什么是最优解（期望）

然后，把问题分成多个步骤，每一步都需要满足：

可行性：每一步都满足问题的约束

局部最优：每一步都做出一个局部最优的选择

- 不可取消：选择一旦做出，在后面遇到任何情况都不可取消

最后，叠加所有步骤的最优解，就是全局最优解

4 回溯算法

4.1 算法策略

回溯算法是一种搜索法，试探法，它会在每一步做出选择，一旦发现这个选择无法得到期望结果，就回溯回去，重新做出选择。深度优先搜索利用的就是回溯算法思想。

4.2 适用场景

回溯算法很简单，它就是不断的尝试，直到拿到解。它的这种算法思想，使它通常用于解决广度的搜索问题，即从一组可能的解中，选择一个满足要求的解。

4.3 使用回溯算法的经典案例

深度优先搜索

0-1背包问题

正则表达式匹配

八皇后

数独

全排列

5 动态规划

5.1 算法策略

动态规划也是将复杂问题分解成小问题求解的策略，与分治算法不同的是，分治算法要求各子问题是相互独立的，而动态规划各子问题是相互关联的。

所以，动态规划适用于子问题重叠的情况，即不同的子问题具有公共的子子问题，在这种情况下，分治策略会做出很多不必要的工作，它会反复求解那些公共子子问题，而动态规划会对每个子子问题求解一次，然后保存在表格中，如果遇到一致的问题，从表格中获取既可，所以它无需求解每一个子子问题，避免了大量的不必要操作。

5.2 适用场景

动态规划适用于求解最优解问题，比如，从面额不定的100个硬币中任意选取多个凑成10元，求怎样选取硬币才可以使最后选取的硬币数最少又刚好凑够了10元。这就是一个典型的动态规划问题。它可以分成一个个子问题（每次选取硬币），每个子问题又有公共的子子问题（选取硬币），子问题之间相互关联（已选取的硬币总金额不能超过10元），边界条件就是最终选取的硬币总金额为 10 元。

针对上例，也许你也可以说，我们可以使用回溯算法，不断的去试探，但回溯算法是使用与求解广度的解（满足要求的解），如果是用回溯算法，我们需要尝试去找所有满足条件的解，然后找到最优解，时间复杂度为 O(2^n^) ，这性能是相当差的。大多数适用于动态规划的问题，都可以使用回溯算法，只是使用回溯算法的时间复杂度比较高而已。

最后，总结一下，我们使用动态规划求解问题时，需要遵循以下几个重要步骤：

定义子问题

实现需要反复执行解决的子子问题部分

识别并求解出边界条件

5.3 使用动态规划求解的一些经典问题

爬楼梯问题：假设你正在爬楼梯。需要 n 阶你才能到达楼顶。每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？

背包问题：给出一些资源（有总量及价值），给一个背包（有总容量），往背包里装资源，目标是在背包不超过总容量的情况下，装入更多的价值

硬币找零：给出面额不定的一定数量的零钱，以及需要找零的钱数，找出有多少种找零方案

图的全源最短路径：一个图中包含 u、v 顶点，找出从顶点 u 到顶点 v 的最短路径

最长公共子序列：找出一组序列的最长公共子序列（可由另一序列删除元素但不改变剩下元素的顺序实现）

6 枚举算法

6.1 算法策略

枚举算法的思想是：将问题的所有可能的答案一一列举，然后根据条件判断此答案是否合适，保留合适的，丢弃不合适的。

6.2 解题思路

确定枚举对象、枚举范围和判定条件。

逐一列举可能的解，验证每个解是否是问题的解。