“高级算法设计与分析”课程学习报告

21215122, 何峙，大数据与人工智能

本课程把算法原理跟实际问题相结合，将原本枯燥的理论通过一些实际例子阐述明白，十分有趣！不是讲解一个个算法的实现过程，而是深入具体分析一类类算法背后的设计思想，对算法的正确性和合理性进行论证，而且讲解的每类算法之间一环扣一环，既互相区别又互相联系，很具有启发性。！现对课程内容进行总结性回顾，以加深对算法分析设计思想的理解。

1. **算法复杂度**

这是衡量算法优劣的重要指标。我们不仅要用算法解决问题，而且要越来越快的解决问题，这也是我们学习算法设计最主要驱动力。算法复杂度通常用数量阶表示，如我们熟悉的时间复杂度从小到大的排列依次是：O(1) < O(logn) < O(n) < O(nlogn) < O(n2) < O(n3) < O(2n) < O(n!) < O(nk)。我们可以跑代码，通过统计监控得出算法的执行时间，但这种统计方式具有很大局限性：1）测试结果受硬件的影响较大；2）测试结果受数据规模的影响很大。于是，课程介绍了一些快速分析出算法的时间复杂度的方法，如：

1. 递推法：利用递推式不断的循环解嵌套；
2. 换元法：通过代换参数可将复杂的递推式转换为较为简单的递推式；
3. 生成函数法：构造数列的生成函数，再利用数列的性质求解N的表达式方法；
4. 特征方程法：通过求解特征方程的根从而求解递推方程的解的方法；

这对快速衡量各种算法的执行效率很有帮助。了解了算法的复杂度，不难认识到随着算法的改进，相同时间内能够处理的数据量也大大提升。

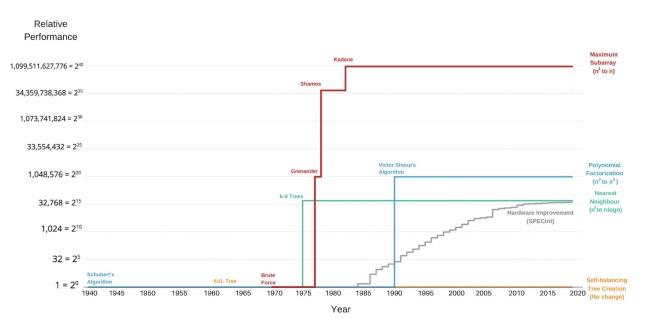


图1 随年份增长而获得改进的算法对于处理数据量的影响

（from MIT Computer Science & Artificial Intelligence Laboratory《How Fast Do Algorithms Improve?》）

图1列出的4个算法族，它们随着年份增长，获得了改进，处理的数据量呈直线式的上升。图中还标有根据摩尔定律的硬件（灰色线）随年份的改进趋势。对比算法的改进，硬件的改进对处理数据量的提升显得相当平滑。由此可得：

1. 算法改进带来的改变问题的可操作性，是硬件改进不能比拟的；
2. 随着数据量的大量提升，算法的改进比硬件的改进显得更加重要。

这些发现表明算法的改进在例如数据分析和机器学习等领域上相当重要，因为它们都依赖于大数据。

1. **分治与递归**

分治是一种算法思想，从字面意思即可以理解：当一个问题规模较大且不易求解的时候，就可以考虑将问题分成几个小的模块，逐一解决。而递归是一种算法实践方法，通常跟迭代法相对比：迭代使用的是循环结构，递归则使用选择结构。递归的优点：能使程序的结构更清晰、更简洁、更容易让人理解，从而减少读懂代码的时间。其缺点：但大量的递归调用会建立函数的副本，会消耗大量的时间和内存，而迭代则不需要此种付出。

分治通常跟递归搭配使用会比较让人容易理解，其思维及使用步骤一般为：

1. 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；
2. 解决：若子问题规模较小而容易被解决则直接解，否则递归地解各个子问题；
3. 合并：将各个子问题的解合并为原问题的解。

例如作业中的整数划分问题：将正整数划分成一系列小的正整数之和。其递归方程为：

q(n, m) =

其中q(n, m)表示最大加数n不大于m的划分个数，只要我们将正整数划分为小整数的4种情况梳理出来，即可以利用递归方法顺利求解。

1. **动态规划**

该方法的核心思想是：将待求解的问题分解为若干个子问题，先求解子问题，然后从这些子问题得到原问题的解。经动态规划法得到的子问题往往不是相互独立的，为了避免重复计算，我们可以用一个表来记录所有已解决的子问题的答案。它常用于求解具有最优子结构性质（问题的最优解包含了子问题的最优解）和子问题重叠性质（在用递归算法自顶向下的解决问题时，每次产生的子问题并不总是新问题，有些问题被反复计算多次）的问题。所以，动态规划不同于递归的自顶向下，它是自底向上的，一般有循环迭代完成。

本课程介绍了一些用动态规划解决问题的例子，如构建最优二叉树，矩阵的连乘问题等，其一般解题步骤是：

1. 定义子问题；
2. 写出子问题的递推关系；
3. 依次计算出这些子问题。这里通常会有一个子问题数组，数组的每一个元素对应一个子问题，然后确定这个数组的计算顺序，即可迭代求解；
4. **贪心策略**

在对问题求解时，总是采取在当前状态下最好或最优（即最有利）的选择，从而希望导致结果是最好或最优的算法。它不从整体最优上加以考虑，所做出的是某种意义上的局部最优解。该算法的一般求解步骤可归纳为：

1. 将问题分解为若干个子问题；
2. 找出适合的贪心策略；
3. 求解每一个子问题的最优解，即局部最优解；
4. 将局部最优解堆叠为全局最优解；

本人认为该算法的唯一难点是找到待求解问题的贪心策略，即贪心策略的选择是否能达到整个问题的最优解。譬如“0-背包问题”：有一个背包，背包最大承载总重量是已知且固定，另有有若干个物品，物品有自己的价值和重量，且不可以分割成任意大小。要求尽可能让装入背包中的物品总价值最大，但不能超过背包总重量。对此可以有如下几个贪心策略：

1. 每次挑选价值最大的物品装入背包？
2. 每次挑选所占重量最小的物品装入？
3. 每次选取单位重量价值最大的物品装入？

哪种贪心策略才解除整个问题的最优解，还需求证明后才能真正运用到解题中。一般来说，贪心算法的证明围绕着整个问题在贪心策略中存在的子问题的最优解得来的。

或许穷尽所有计算资源也找不到全局最优解，但“轻松”找到局部最优解在现有条件下也未尝不可。

1. **回溯与分枝界限策略**

不同于分治、动态规划、贪心策略等式基于规范的算法，回溯与分枝界限是基于搜索的算法。

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的树，当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（也叫剪枝），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。

分支界限法类似回溯法，也是在问题的解空间上搜索问题解的算法，其求解目标是找出满足约束条件的一个解(而回溯是找出所有的解)，或是在满足条件的解中找出最优解。

这两种算法的容易混淆，主要区别如下：

* 回溯法

1. 对于求解目标：回溯法的求解目标是找出解空间中满足约束条件的一个解或所有解。
2. 对于搜索方式：回溯法使用深度优先遍历搜索整个解空间，当不满条件时，丢弃，继续搜索下一个儿子结点，如果所有儿子结点都不满足，向上回溯到它的父节点。

* 分支限界法

1. 对于求解目标：分支限界法的目标一般是在满足约束条件的解中找出在某种意义下的最优解，也有找出满足约束条件的一个解。
2. 对于搜索方式：分支限界法以广度优先或以最小损耗优先的方式搜索解空间：

a.广度优先分支界限法：按照队列先进先出原则选取下一个结点为扩展结点

b.最小损坏分支限界法：按照优先队列规定的优先级选取优先级最高的结点成为当前扩展结点

1. **局部与随机搜索策略**

（提一下N皇后问题）

1. **遗传算法**