算法分析与问题的计算复杂度

计算复杂度的意义

对于每个问题，可以设计不同的算法来解决。不同的算法有不同的时间复杂度，如何找到复杂度最优的算法？这首先依赖于确定解决这个问题的算法类可能达到的最低复杂度是多少。这个下界是由问题本身确定的，与具体的算法无关，称为问题的计算复杂度。
对问题的计算复杂度进行分析，可以得到解决该问题的算法能达到的复杂度的一个下界。而一个正确的算法可以给出解决该问题的复杂度上界。通过改进分析方式，可以逐渐提高下界；通过改进算法，可以不断降低上界。当下界和上界不断逼近，直到在阶上相等时，我们得到了解决该问题的最优的算法和一个紧的下界。

确定算法类的时间复杂度的具体方式

1. 设计一个算法 A，给出 A 在最坏情况下的时间复杂度 $W(n)$，从而得到了该算法类在最坏情况下的时间复杂度的一个上界。

2. 寻找函数 $F(n)$，使得对任何算法都存在一个规模为 n 的输入并且该算法在这个输入下至少要做 $F(n)$ 次基本运算。即找到该问题的算法类在最坏情况下的时间复杂度的一个下界。

3. 如果 $W(n)=F(n)$ 或 $W(n)=\Theta(F(n))$，则 $F(n)$ 就是一个紧的下界，而 $W(n)$ 是解决该问题的最优的算法。

4. 如果 $W(n)>F(n)$，可能 A 不是最优算法，或者这个下界太低，因此有一下两种改进方式：
   - 改进 A 或设计新算法，降低 $W(n)$
   - 找出更高的新下界 $F'(n)$

分析算法类计算复杂度的具体方式

1. 对求解这个问题的所有算法建立执行过程的统一模型（比如决策树），从而对算法在平均情况和最坏情况下的工作量进行估计。

2. 给出针对任意求解算法设计“最坏”实例的方式。对于这个具体输入实例，该算法至少要做的工作量作为问题计算复杂度的下界。

3. 通过归约的方式。假设想知道算法 A 的复杂度下界 $W_A(n)$，已知的是算法 B 的复杂度下界 $W_B(n)$。通过构造一个调用 A 来解决 B 的算法，来证明 A 所需要的计算量不会低于 B 所需要的最低计算量，即 $W_A(n)=\Omega(W_B(n))$。

   NOTE： 在确定了算法类的一个复杂度下界后，一般通过直接构造一个与该复杂度同阶的算法来说明该下界是紧的。

直接估计最少的操作次数

实例

1. Findmax 算法：
   每次比较淘汰一个元素，至少比较 $n-1$ 次。故估计计算复杂度下界为 $n-1$。事实上可以设计出顺序比较的算法证明该下界是紧的。

   NOTE: 直接估计操作次数的方式往往需要说明估计里的每步操作都是必须的。通常采用构造特定输入的方式来说明如果没有某一步操作，当出现该特定输入时算法必然出现错误结果来证明。

决策树

- 二叉树的性质：
  1. 在二叉树的 t 层至多有 2^t 个节点。

  2. 深度为 d 的二叉树至多有 $2^{d+1}-1$ 个节点。

  3. n 个节点的二叉树深度至少为 $\lfloor \log n \rfloor$

4. 若树的每个节点都有两个儿子，则树叶个数 t 和 树的深度 d 满足 $t \leq 2^d$

实例

以元素的比较作为基本运算的问题，比如搜索和排序问题，都适合用决策树分析下界。一般的使用方式就是给出决策树的构造方式，即每个节点如何定义，以及左右子节点应该如何定义，然后根据叶节点的数量 n 和树深 d 满足：$d \leq \log n$ 来确定最坏情况下的操作次数（一般等于树深）。

1. 二分检索算法：
   若当前步骤是将 $A[i]$ 与目标值 x_0 进行比较，则，将当前节点标记为 i，若 $A[i] < x_0$ 且下一步探索目标是 $A[j]$，则将当前节点的左儿子标记为 j；若 $A[i] > x_0$ 且下一步探索目标是 $A[k]$，则将当前节点的左儿子标记为 k。最后根据有 n 个节点的二叉树的深度至少为 $\lfloor \log n \rfloor$ 得到最坏情况下的复杂度 $O(\log n)$。

构造“最坏”的输入实例

实例

1. 找中位数问题：
   设中位数为 z，且有 $x < z < y$，将 x 和 y 的比较定义为无效比较（因为这种比较不能得到任何与中位数相关的信息），但是如果 $x > z \ \wedge \ y > z$，则 x 和 y 的比较定义为有效比较。我们试图构造出输入样例使得无效比较的次数最多。方法：首先给中位数任意赋值为 z，其他元素的值待定。设算法在当前这一步比较 x 和 y
   - 当 x 和 y 均未赋值时：给 x 和 y 赋值满足 $x > z > y$；
   - 当 $x > z$ 且 y 未赋值时：给 y 赋值满足 $y < z$；
   - 当 $x < z$ 且 y 未赋值时：给 y 赋值满足 $y > z$；
   - 当 x 和 y 均已经赋值时，保持原赋值不变。

   这种定义下，当出现未赋值元素时，算法所作的必然是无效比较。因为算法至少需要 $(n-1)/2$ 次无效比较来给所有元素赋值，然后至少需要 $n-1$ 次有效比较来确定所有元素的顺序，故算法的复杂度下界为 $3n/2 - 3/2$。而 Select 算法在阶上等于该下界，故该下界是紧的。

通过归约确定计算复杂度的下界

实例

1. P 为正整数的素因子分解问题，Q 是素数测试问题。现在想知道 P 的复杂度下界。因为对于 Q 可以设计算法如下：
   1. 调用 P 对 n 进行素因子分解
   2. 如果素因子个数大于等于 2，则不是素数。

这说明 P 的复杂度不会低于 Q 的复杂度下界。