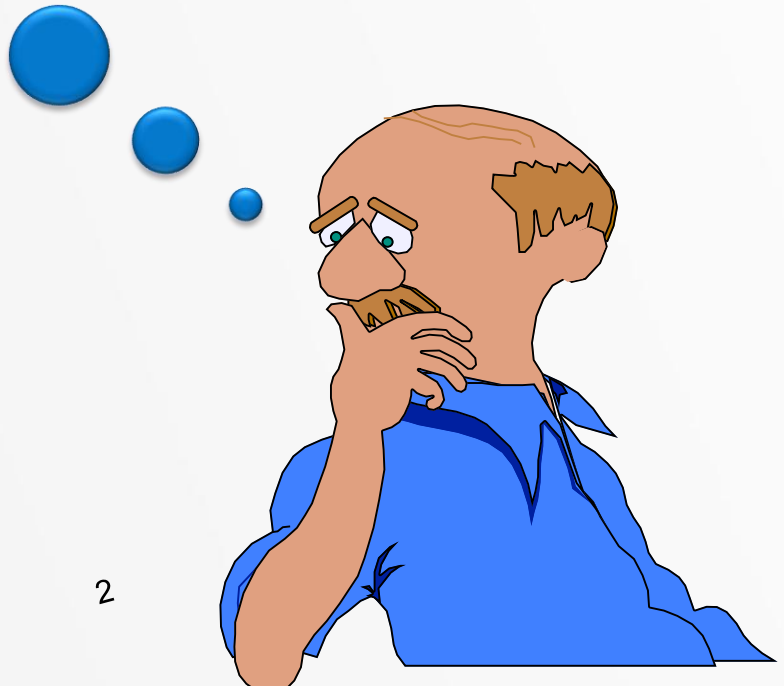
算法复杂度分析

一个项目有多种解决方案?
项目提交后老师如何
评判哪种方案最好?




问题的理解
数据结构设计
算法设计
算法分析
程序实现与测试

第1章 绪论 （算法分析）

好的算法应该具备以下特性：

正确性：正确性是对算法能否正确求解问题的评价，是首要和最基本的特性；

可读性：可读性是对算法描述的思路、层次的评价。好的算法应该是思路清晰、层次分明、阅读和修改容易；

健壮性：健壮性是对算法在异常情况下处理能力的评价。好的算法在出现异常或非法数据时，在操作不当时，算法都能作适当处理；

高效性：算法的效率是对求解同样问题的不同算法所占用的时间或空间的评价。好的算法应该是高效的，即求解问题所占用存储空间少，执行时间短；

算法高效性的评判

如何比较算法效率?

算法性能比较方法

- 编程后测试运行时间
- 编程前分析可能的运行时间

例如，让一台巨型机做插入排序，让另一台微型机做合并排序，它们的输入都是一个长度为 100 万的数组。假设巨型机每秒执行 1 亿条指令，微型机每秒仅仅做 100 万条指令。为了使差别更明显，假设世界上最优秀的程序员用机器代码在巨型机上实现插入排序，编出的程序需要执行 $2n^2$ 条巨型机指令来排序 n 个数。另一方面，让一个一般的程序员在微型机上用高级语言写合并排序，产生的代码要花 $50n \lg n$ 条微型机指令。

为排序 100 万个数，巨型机需费时间：

$$\frac{2 \cdot (10^6)^2 \text{条指令}}{10^8 \text{条指令} / \text{秒}} = 20000\text{秒} \approx 5.56\text{小时}$$

实际运行 5.55小时

微型机需费时间：

$$\frac{50 \cdot 10^6 \lg 10^6 \text{条指令}}{10^6 \text{条指令} / \text{秒}} \approx 1000\text{秒} \approx 16.67\text{分}$$

实际运行 18.25分

运行时间比较算法效率是很难做到准确的，只能做定性分析，要定量分析得靠估计出可能的运行时间

算法复杂性分析

算法复杂性是算法运行所需要的计算机资源的量，需要时间资源的量称为时间复杂性，需要的空间资源的量称为空间复杂性。这个量应该只依赖于算法要解的问题的规模、算法的输入和算法本身的函数。如果分别用N、I和A表示算法要解问题的规模、算法的输入和算法本身，而且用C表示复杂性，那么，应该有C=F(N,I,A)。一般把时间复杂性和空间复杂性分开，并分别用T和S来表示，则有： T=T(N,I)和S=S(N,I) 。

（通常，让A隐含在复杂性函数名当中）

1.4 算法复杂性分析

主要问题：如何将复杂性函数F具体化?

根据T（N,I)的概念，它应该是算法在一台抽象的计算机上运行所需要的时间。

◆设此抽象的计算机所提供的元运算有k种，分别记为O_1，O_2,...O_k

◆又设每次执行一次这些元运算的时间分别为t_1,..t_k

◆对于算法A，设统计用到元运算O_i的次数为e_i

$e_i=e_i(N,I)$　　因此　$T(N,I)=\sum_{i=1}^{k}t_ie_i(N,I)$

程序的算法分析方法

抛开软件和硬件因素，只和问题规模有关。编写程序前预先估计算法优劣，改进或者选择最佳算法编程实现

一个算法用程序设计语言表示后，算法就是由一组语句构成，算法的执行效率就由各语句执行的次数所决定

计算步

- 一个算法花费的时间与算法中语句的执行次数成正比例，哪个算法中语句执行次数多，它花费时间就多。一个算法中的语句执行次数(计算步)称为语句频度或时间频度。记为T(n)。
- 一般情况下，算法的基本操作重复执行的次数是模块n的某一个函数f（n），因此，算法的时间复杂度记做：T（n）=O（f（n））

- 分析：随着模块n的增大，算法执行的时间的增长率和f（n）的增长率成正比，所以f（n）越小，算法的时间复杂度越低，算法的效率越高。

〖例子1〗Iterative function for summing a list of numbers

```
float  sum ( float  list[ ],  int  n )
{  /* add a list of numbers */
   float  tempsum = 0;
   int  i ;
   for ( i = 0; i < n; i++ )

       tempsum  += list [ i ] ;

   return  tempsum;
}
```

循环n次

$$T_{sum}(n) = n$$

例二 选择排序

```
void select_sort(int& a[], int n)
{ // 将 a 中整数序列重新排列成自小至大有序的整数序列。
    for ( i = 0;  i< n-1;  ++i )
    { j = i;  // 选择第 i 个最小元素
      for ( k = i+1;  k < n;  ++k )
          if (a[k] < a[j] ) j = k;

      if ( j != i )  a[j] ←→ a[i]


    }
} // select_sort
```

$$T(n) = n^2$$

规定输入

最坏情况下的时间复杂性：

$$T_{\max}(N)=\max_{I\in D_N}T(N,I)=\max_{I\in D_N}\sum_{i=1}^{k}t_i e_i(N,I)=\sum_{i=1}^{k}t_i e_i(N,I^*)=T(N,I^*)$$

最好情况下的时间复杂性：

$$T_{\min}(N)=\min_{I\in D_N}T(N,I)=\min_{I\in D_N}\sum_{i=1}^{k}t_i e_i(N,I)=\sum_{i=1}^{k}t_i e_i(N,\tilde{I})=T(N,\tilde{I})$$

平均情况下的时间复杂性：

$$T_{\text{avg}}(N)=\sum_{I\in D_N}P(I)T(N,I)=\sum_{I\in D_N}P(I)\sum_{i=1}^{k}t_i e_i(N,I)$$

其中D_N是规模为N的合法输入的集合；I^*是D_N中使$T(N, I^*)$达到$T_{max}(N)$的合法输入；$\tilde{I}$是中使$T(N,\tilde{I})$达到$T_{min}(N)$的合法输入；而P(I)是在算法的应用中出现输入I的概率。