[**从零开始\_学\_数据结构（一）**](http://blog.csdn.net/qq20004604/article/details/50869632)**——算法**

**算法的定义：**

解决问题的方法。

对于同一个问题，一个好的算法比一个差的算法，效率更高，更节约资源。

For Computer：算法是解决特定问题的求解步骤的描述，在计算机中，表示指令的有限序列，每条指令表示一个或者多个操作。

简单来说，**算法就是输入代码，告诉计算机，你应该怎么解决这个问题**。

**算法的特性：**

（1）输入和输出。

光算出结果但不输出结果，跟没算没区别；要计算，总得有数据，不然没法计算。

（2）有穷性：

能出结果，并且在接受时间之内出结果。如果时间太长，这个算法往往就没什么意义了。

（3）确定性：

同样的数据，同一个算法，出同样的结果。

（4）可行性；

如果一个算法太复杂，都没办法变成代码给计算机，那肯定不行。

**一个好的算法的要求：**

（1）正确。

对于计算的数据，能得到预期的结果。

①首先得做到程序没问题，能运行；

②其次得对于计算的数据，能得到符合要求的结果；

③更好的算法，对于非法数据，能得出满足要求的结果（比如说提示有问题）；

④对于专门用于为难人的数据，也能有满足要求的输出结果；

至少要做到第②步

（2）可读性

代码和算法，起码让人能读懂，不然写代码的人都不明白自己写的代码，如果代码出问题，那么也没办法鉴别出来。

（3）健壮性

面对不符合要求（非法）的数据，能够适当的处理。

例如，我们正常处理的一组数据里，只有正数，比如说我们使用unsigned int类型。那么假如有负数输入，如果没有代码去鉴别这些，那么就容易出现出乎意料的结果。

（4）速度快，用内存小

假如计算一组1GB大小的数据，同样的处理器

①假如一个算法用1分钟，另一个算法用1小时，显然前者更好；

②假如一个算法用10MB内存，另一个算法用1GB内存，显然前者更好。

在实际中，有时候会出现对于较少的数据来说，某一种算法更好，对于较多的数据来说，另一种算法更好。因此，选择哪种算法，要根据实际情况而决定。

**算法效率的度量方法：**

（1）事后统计：

简单直接，运行一遍就知道。

他到底用1秒还是100秒，用10MB内存还是500MB内存，一目了然。

**缺点：**假如需要1天甚至10天难道也要这么做？肯定是不靠谱的。因此，一般是不会使用这种方法的。

（2）事前分析估算方法：

这个方法，简单来说，就是：

①预估数据量（比如100万个数据）

＋

②根据算法计算其运行次数（比如说３行代码，一般是运算３次的。但若涉及到循环，那么就是循环代码的行数＊循环次数）

＋

③机器执行代码的速度（计算求和与在一串字符串中插入一个字符，其执行速度自然是有区别的。而且，机器配置不同，运行同样指令其时间也是有区别的）

由于③是不可控的，因此一般不考虑。

而①是我们要为之服务的对象，因此是需要考虑，但无法选择的；

只有②，是我们最需要关心的内容。

以循环为例：

int total;

for(int i=0; i < n; i ++)

{

cin>>data[i];

data[i]\*=2;

data[i]+=5;

total += data[i];

}

cout<< total;

**//假如cin是读取某个文件的int类型的数据（实际上不能这么写）**

首先，循环范围内有4条指令。因此，当data数组里有n个元素时，**这个代码将执行4\*n + 2次**，4\*n是循环，1是int total，另一个1是cout<<total。

如果我们使用另外一个方法：

int total;

for(int i=0; i<n; i++)

{

cin>>data[i];

total+=data[i];

}  
total=total\*2+5\*n;

cout<<total;

**//注意，这里的cin只是用来意会，实际应用ifstream类变量来读取文件**

与上面相比，这个算法需要执行的次数是：2\*n+3次。

假如n=100万时，第一个算法需要执行的次数是400万+2次，第二个算法执行的次数是200万+3次，节约了几乎50%的时间。

这样的话，哪个算法好，通过简单的预估执行代码的次数，一目了然。

当然了，实际中，会因为代码不同，不同的代码的效率略有差异，但是显然执行次数更少的代码，更剩时间，效率也更高。

除了以上两种以外，还有双重for循环（即外层是一个for循环，执行n次；然后在外层的for循环内部还有一个for循环，执行m次，如代码）；

for (int i=0; i < n ; i++)

for(int j=0; j<m; j++)

total=total+m+n;

这样一段代码，其执行次数将为m\*n次（假如m=n的话，可以认为其运行次数为n的乘方）。

**总结一下：**

不同的算法（假设代码行数为m），针对不同的数据量（n），将执行不同的次数：

有的次数为n，有的次数为m，有的为m\*n，有的为n\*n，也有其他的。

一般来说，需要特别设计算法的，数据量都比较大（比如说几万甚至更多），代码行数相对就偏少（也许只有几行或者几十行）。

因此，m次最好，n次其次，m\*n次再次，n\*n次最次。

四个字：**避免乘方**

如下表：

|  |  |  |
| --- | --- | --- |
| 数据量n | 算法A执行代码（n2）次  记作f(A)= | 算法B执行代码（3n）次  f(B)= |
| 1 | 1 | 3 |
| 2 | 4 | 6 |
| 3 | 9 | 9 |
| 10 | 100 | 30 |
| 100 | 一万 | 三百 |
| 10000 | 一亿 | 三万 |

假如需要特别的算法，一般来说，原因是数据量都很大（几百MB，几GB，几十几百GB，甚至TB级别），只有一个好的算法，才能做到高效率的完成任务。

因此，需要尽量避免那种执行的代码次数，为数据量的乘方甚至n次方这样的情况。

以下概念有印象就行，反正知道怎么用最重要，单纯背概念毫无意义。

**概念：函数的渐进增长**

如上表，当n<3时，f(A)>f(B)；

当n=3时，f(A)=f(B)；

当n>3时，f(A)>f(B)；

假如对于给定的函数，存在一个整数N，使得对于所有n>N的情况，f(A)总是大于f(B)，那么我们就说，f(a)的渐进增长快于f(B)

假如两个函数，都存在幂的情况（比如说n的2次方或者3次方）。

那么主要关心其最高阶的部分（因为假如n=10，其2次方的值，仅仅只有3次方的十分之一，n越大，差距越大。即使一个是n的平方+一万，另一个是n的3次方，只需要n=100时，前者的效率便能轻松高于后者）。

**因此：避免使用需要执行n的m次幂 次代码的算法。**

除此之外，一个算法的需要执行代码的次数还有对数的情况、或者是指数的情况。

个人意见是，这里有个概念就可以，暂时不需要深究，毕竟这里学的是数据结构。

**算法需要考虑最坏的情况：**

例如我们设计一个算法，然后通过这个算法去试图去寻找一个文件里的数据。

那么①这个数据可能存在于文件的开始，也可能存在于数据的结尾；

②如果这个算法是逐个字节寻找，那么假如在开始，就会立刻找到，但若假如在结尾，那么可能需要找很久很久。而这个 最坏的情况（需要很多时间），是需要我们考虑的。

③特别是当这个文件很大很大时（遍历整个文件需要很多时间），那么考虑如何解决这个问题，是很重要的（因此可能需要优化算法）。比如说，假如这个数据存在的开始地址，是0xXXXX0这样，我们就可以考虑一次读取十六个字节，然后先对比第一个，符合后在继续对比，这样的话，就相对节约了很多时间。

④除了优化算法外，我们还应该考虑平均时间。例如，当算法无法优化时，那么最好的情况下，这个算法需要多久，最差的情况下，这个算法需要多久，二者平均起来，往往就是普通的情况下，需要的时间。

而这个 **平均时间**，是这个算法的期望运行时间，是需要我们关注的。

**算法的存储空间：**

当使用算法时，除了需要花时间外，可能还需要使用一定的存储空间。

例如：

①使用一个10个元素的int数组来存储计算的内容，由于一个int是4字节，因此需要使用40字节来存储这些内容；

②可能还要创建一个临时的变量（堆或者栈）用于存储算法中创建的临时对象，而在提高效率方面，创建的临时对象是有一定意义的；

③有时候，对于算法，要求其最多只能使用多少空间，算法在运行过程中，不能超出这个界限。

因此，对于存储空间的使用，也是有必要考虑的。