算法的复杂度

参考 ： https://baike.baidu.com/item/%E7%AE%97%E6%B3%95%E5%A4%8D%E6%9D%82%E5%BA%A6/210801?fr=aladdin
https://www.cnblogs.com/zknublx/p/5885840.html

简介

编辑

同一问题可用不同算法解决，而一个算法的质量优劣将影响到算法乃至程序的效率。算法分析的目的在于选择合适算法和改进算法。一个算法的评价主要从时间复杂度和空间复杂度来考虑。

时间复杂度

编辑

（1）时间频度

一个算法执行所耗费的时间，从理论上是不能算出来的，必须上机运行测试才能知道。但我们不可能也没有必要对每个算法都上机测试，只需知道哪个算法花费的时间多，哪个算法花费的时间少就可以了。并且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称为语句频度或时间频度。记为T(n)。算法的时间复杂度是指执行算法所需要的计算工作量。

（2）时间复杂度

在刚才提到的时间频度中，n称为问题的规模，当n不断变化时，时间频度T(n)也会不断变化。但有时我们想知道它变化时呈现什么规律。为此，我们引入时间复杂度概念。

一般情况下，算法中基本操作重复执行的次数是问题规模n的某个函数，用T(n)表示，若有某个辅助函数f(n),存在一个正常数c使得fn*c>=T(n)恒成立。记作T(n)=O(f(n)),称O(f(n)) 为算法的渐进时间复杂度，简称时间复杂度。

在各种不同算法中，若算法中语句执行次数为一个常数，则时间复杂度为O(1),另外，在时间频度不相同时，时间复杂度有可能相同，如T(n)=n^2+3n+4与T(n)=4n^2+2n+1它们的频度不同，但时间复杂度相同，都为O(n^2)。

按数量级递增排列，常见的时间复杂度有：

常数阶O(1),对数阶O(log2n)(以2为底n的对数，下同),线性阶O(n),

线性对数阶O(nlog2n),平方阶O(n^2)，立方阶O(n^3),...,

k次方阶O(n^k),指数阶O(2^n)。随着问题规模n的不断增大，上述时间复杂度不断增大，算法的执行效率越低。

算法的时间性能分析

（1）算法耗费的时间和语句频度

一个算法所耗费的时间=算法中每条语句的执行时间之和

每条语句的执行时间=语句的执行次数(即频度(Frequency Count))×语句执行一次所需时间

算法转换为程序后，每条语句执行一次所需的时间取决于机器的指令性能、速度以及编译所产生的代码质量等难以确定的因素。

若要独立于机器的软、硬件系统来分析算法的时间耗费，则**设每条语句执行一次所需的时间均是单位时间，一个算法的时间耗费就是该算法中所有语句的频度之和。**

求两个n阶方阵的乘积 C=A×B,其算法如下:

```
1  # define n 100 // n 可根据需要定义,这里假定为100
2  void MatrixMultiply(int A[n][n], int B [n][n], int C[n][n])
3  { //右边列为各语句的频度
4      int i ,j ,k;
5      for(i=0; i<n;i++) //n
6      for (j=0;j<n;j++) { //n*n
7          C[i][j]=0; //n²
8          for (k=0; k<n; k++) //n²*n
9          C[i][j]=C[i][j]+A[i][k]*B[k][j];//n³
10     }
11 }
```

该算法中所有语句的频度之和(即算法的时间耗费)为:

T(n)=2n3+3n2+2n （1.1）

分析:

语句(1)的循环控制变量i要增加到n，测试到i=n成立才会终止。故它的频度是n+1。但是它的循环体却只能执行n次。语句(2)作为语句(1)循环体内的语句应该执行n次，但语句(2)本身要执行n+1次，所以语句(2)的频度是n(n+1)。同理可得语句(3)，(4)和(5)的频度分别是n2，n2(n+1)和n3。

算法MatrixMultiply的时间耗费T(n)是矩阵阶数n的函数。

（2）问题规模和算法的时间复杂度

算法求解问题的输入量称为问题的规模(Size),一般用一个整数表示。

矩阵乘积问题的规模是矩阵的阶数。

一个图论问题的规模则是图中的顶点数或边数。

一个算法的时间复杂度(Time Complexity，也称时间复杂性)T(n)是该算法的时间耗费，是该算法所求解问题规模n的函数。当问题的规模n趋向无穷

大时，时间复杂度T(n)的数量级(阶)称为算法的渐进时间复杂度。
算法MatrixMultiply的时间复杂度T(n)如(1.1)式所示，当n趋向无穷大时，显然有T(n)~O(n^3)；
这表明，当n充分大时，T(n)和n^3之比是一个不等于零的常数。即T(n)和n^3是同阶的，或者说T(n)和n^3的数量级相同。记作T(n)=O(n^3)是算法MatrixMultiply的渐近时间复杂度。
（3）渐进时间复杂度评价算法时间性能
主要用算法时间复杂度的数量级(即算法的渐近时间复杂度)评价一个算法的时间性能。
算法MatrixMultiply的时间复杂度一般为T(n)=O(n^3)，f(n)=n^3是该算法中语句(5)的频度。下面再举例说明如何求算法的时间复杂度。
交换i和j的内容。

```
Temp=i;
i=j;
j=temp;
```

以上三条单个语句的频度均为1，该程序段的执行时间是一个与问题规模n无关的常数。算法的时间复杂度为常数阶，记作T(n)=O(1)。
注意：如果算法的执行时间不随着问题规模n的增加而增长，即使算法中有上千条语句，其执行时间也不过是一个较大的常数。此类算法的时间复杂度是O(1)。
变量计数之一：

```
(1) x=0;y=0;
(2) for(k=1;k<=n;k++)
(3) x++;
(4) for(i=1;i<=n;i++)
(5) for(j=1;j<=n;j++)
(6) y++;
```

一般情况下，对步进循环语句只需考虑循环体中语句的执行次数，忽略该语句中步长加1、终值判别、控制转移等成分。因此，以上程序段中频度最大的语句是(6)，其频度为f(n)=n^2，所以该程序段的时间复杂度为T(n)=O(n^2)。
当有若干个循环语句时，算法的时间复杂度是由嵌套层数最多的循环语句中最内层语句的频度f(n)决定的。
变量计数之二：

```
(1) x=1;
(2) for(i=1;i<=n;i++)
(3) for(j=1;j<=i;j++)
(4) for(k=1;k<=j;k++)
(5) x++;
```

该程序段中频度最大的语句是(5)，内循环的执行次数虽然与问题规模n没有直接关系，但是却与外层循环的变量取值有关，而最外层循环的次数直接与n有关，因此可以从内层循环向外层分析语句(5)的执行次数：
则该程序段的时间复杂度为T(n)=O(n^3/6+低次项)=O(n^3)。
（4）算法的时间复杂度不仅仅依赖于问题的规模，还与输入实例的初始状态有关。
在数值A[0..n-1]中查找给定值K的算法大致如下：

```
(1)i=n-1;
(2)while(i>=0&&(A[i]!=k))
(3) i--;
(4)return i;
```

此算法中的语句(3)的频度不仅与问题规模n有关，还与输入实例中A的各元素取值及K的取值有关:
①若A中没有与K相等的元素，则语句(3)的频度f(n)=n；
②若A的最后一个元素等于K,则语句(3)的频度f(n)是常数0。

空间复杂度

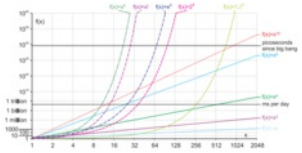
编辑
与时间复杂度类似，空间复杂度是指算法在计算机内执行时所需存储空间的度量。记作:
S(n)=O(f(n))
算法执行期间所需要的存储空间包括3个部分：

- 算法程序所占的空间；
- 输入的初始数据所占的存储空间；
- 算法执行过程中所需要的额外空间。

在许多实际问题中，为了减少算法所占的存储空间，通常采用压缩存储技术。

复杂度分析

编辑
通常一个算法的复杂度是由其输入量决定的，随着输入的增加，



复杂度

复杂度

不同算法的复杂度增长速度如右图所示：

为了降低算法复杂度，应当同时考虑到输入量，设计较好的算法。