

和空间复杂度-总结

阅读数：142081

124

27

1月

...

<

>

算法的时间复杂度和空间复杂度-总结

我们要做 两项分析。**第一是从数学上证明算法的正确性**，这一步主要用到形式化证明的方法及相关推理模式，如循环不变式、数学归纳法等。而**第二是分析算法的时间复杂度**。算法的时间复杂度反映了程序执行时间随输入规模增长而增长的量级，在很大程度上能很好反映出算法的优劣与否。因此，作为程序员，必须掌握估算、分析方法，以设计高效的算法，达到更好的时间复杂度和空间复杂度的控制。

算法编制的时间复杂度是指算法编制的时间复杂度。而度量一个程序的执行时间通常有两种方法。

好的方法。该方法有两个缺陷：一是要想对设计的算法的运行性能进行评测，必须先依据算法编制相应的程序并实际运行；二是所得时间的统计量太大，难于测试和度量，从而不能掩盖算法本身的优劣。

硬件、软件等环境因素，有时容易掩盖算法本身的优劣。**因此人们常常采用事前分析估算的方法。**

算。一个用高级语言编写的程序在计算机上运行时所消耗的时间取决于下列因素：

生的代码质量；(3). 问题的输入规模；(4). 机器执行指令的速度。

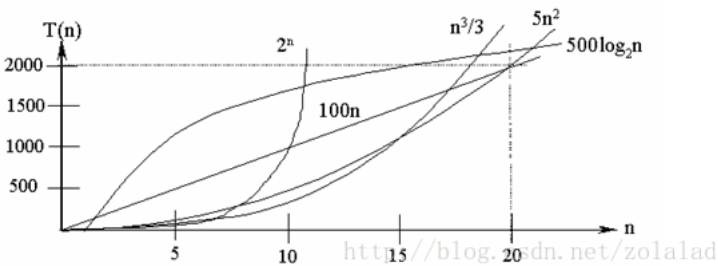
环3种）和原操作（指固有数据类型的操作）构成的，则算法时间取决于两者的综合效果。为了便于比较同一个问题的不同算法，通常的做法是，从算法中选取一种对于所研究的问题（或算法类型）来说是耗时最多的操作来度量算法的时间复杂度。

花费的时间，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，哪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。记作T(n)。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。通常大写字母O称为“大O记号”，它是指一个函数f(n)相对于另一个函数g(n)当n趋近于无穷大时的渐近时间复杂度，简称时间复杂度。

Landau符号其实是由德国数论学家保罗·巴赫曼（Paul Bachmann）在其1892年的著作《解析数论》首先引入，由另一位德国数论学家艾德蒙·朗道（Edmund Landau）推广的函数来描述复杂函数行为，给出一个上或下（确）界。在计算算法复杂度时一般只用到大O符号，Landau符号体系中的小o符号、Θ符号等等比较不常用，但在都用大写英语字母O；小o符号也是用小写英语字母o，Θ符号则维持大写希腊字母Θ。

个常数C，使得在当n趋于正无穷时总有 $T(n) \leq C * f(n)$ 。简单来说，就是T(n)在n趋于正无穷时最大也就跟f(n)差不多大。也就是说当n趋于正无穷时T(n)的增长速度与f(n)的增长速度是一样的。一般我们都是取尽可能简单的函数。例如， $O(2n^2+n+1) = O(3n^2+n+3) = O(7n^2+n) = O(n^2)$ ，一般都只用 $O(n^2)$ 表示就可以了。注意到大O符号里隐藏着常数项，T(n)当做一棵树，那么O(f(n))所表达的就是树干，只关心其中的主干，其他的细枝末节全都抛弃不管。

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，但时间复杂度都是 $O(n^2)$ 。时间复杂度不同的算法，其时间复杂度随问题规模n的递增排列，常见的时间复杂度有：常数阶O(1),对数阶O(log₂n),线性阶O(n),线性对数阶O(nlog₂n),平方阶O(n²),立方阶O(n³),..., k次方阶O(n^k),指数阶O(2ⁿ),阶乘阶O(n!)。



复杂度不断增大，算法的执行效率越低。

能用多项式阶O(n^k)的算法，而不希望用指数阶的算法。

大依次为： $O(1) < O(\log_2 n) < O(n) < O(n \log_2 n) < O(n^2) < O(n^3) < \dots < O(2^n) < O(n!)$

选一类算法）只需选择一种基本操作来讨论算法的时间复杂度即可，有时也需要同时考虑几种基本操作，甚至可以对不同的操作赋予不同的权值，以反映执行时间。比较解决同一问题的两种完全不同的算法。

语句就是基本语句，通常是最内层循环的循环体。

的数量级；

的数量级，这就意味着只要保证基本语句执行次数的函数中的最高次幂正确即可，可以忽略所有低次幂和最高次幂的系数。这样能够简化算法分析，使

的性能。

及放入大O记号中。

则基本语句通常是最内层的循环体，如果算法中包含并列的循环，则将并列循环的时间复杂度相加。例如：

```
for (i=1; i<=n; i++)
{
    for (j=1; j<=n; j++)
    {
        // 基本语句
    }
}
```

为O(n)，第二个for循环的时间复杂度为O(n²)，则整个算法的时间复杂度为O(n+n²)=O(n²)。

数是一个常数，一般来说，只要算法中不存在循环语句，其时间复杂度就是O(1)。其中O(log₂n)、O(n)、O(nlog₂n)、O(n²)和O(n³)称为多项式时间，而O(2ⁿ)、O(n!)为前者（即多项式时间复杂度的算法）是有效算法，把这类问题称为**P（Polynomial,多项式）类问题**，而把后者（即指数时间复杂度的算法）称为**NP（Non-deterministic Polynomial,非确定多项式）问题**。

是可以接受的，很多问题都有多项式级的解——也就是说，这样的问题，对于一个规模是n的输入，在n^k的时间内得到结果，称为P问题。有些问题要复杂一点，我们验证某个猜测是不是正确。比如问4294967297是不是质数？如果要直接入手的话，那么要把小于4294967297的平方根的所有素数都拿出来，看看能不能整除41和6700417的乘积，不是素数，很好验证的，顺便麻烦转告费马他的猜想不成立。大数分解、Hamilton回路之类的问题，都是可以多项式时间内验证一个

简单的程序分析法则：

或赋值语句,近似认为需要O(1)时间

-系列语句所用的时间可采用大O下"求和法则"

时间复杂度分别为 T1(n)=O(f(n))和 T2(n)=O(g(n)),则 T1(n)+T2(n)=O(max(f(n), g(n)))

O(g(n)),则 T1(m)+T2(n)=O(f(m) + g(n))

要时间耗费是在执行then语句或else语句所用的时间,需注意的是检验条件也需要O(1)时间

时间主要体现在多次迭代中执行循环体以及检验循环条件的时间耗费,一般可用大O下"乘法法则"

时间复杂度分别为 T1(n)=O(f(n))和 T2(n)=O(g(n)),则 T1*T2=O(f(n)*g(n))

几个容易估算的部分,然后利用求和法则和乘法法则技术整个算法的时间复杂度

g(n)=O(f(n)),则O(f(n))+ O(g(n))= O(f(n)); (2) O(Cf(n)) = O(f(n)),其中C是一个正常数

复杂度进行示例说明：

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**
算法的时间复杂度是O(1)。

（一次）

2019人工智能薪资

Python资料免费领

会员任意学

Java薪资多少

怎样才能不被裁员

人脸识别算法

小项目外包

去低阶项，去掉常数项，去掉高阶项的常参得到），所以 $T(n) = O(n^2)$ ；

①
: * n) ; j ++)
②

) = 2n² - n - 1

$O(n^2)$.

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，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。


①
②


n).


$2^{f(n)} \leq n; f(n) \leq \log_2 n$


·)


k++)


124

27









空间复杂度

排序法	平均时间	最差情形	稳定度	额外空间	备注
冒泡	$O(n^2)$	$O(n^2)$	稳定	$O(1)$	n小时较好
交换	$O(n^2)$	$O(n^2)$	不稳定	$O(1)$	n小时较好
选择	$O(n^2)$	$O(n^2)$	不稳定	$O(1)$	n小时较好
插入	$O(n^2)$	$O(n^2)$	稳定	$O(1)$	大部分已排序时较好
基数	$O(\log_R B)$	$O(\log_R B)$	稳定	$O(n)$	B是真数(0-9), R是基数(个十百)
Shell	$O(n\log n)$	$O(n^s)$ $1 < s < 2$	不稳定	$O(1)$	s是所选分组
快速	$O(n\log n)$	$O(n^2)$	不稳定	$O(n\log n)$	n大时较好
归并	$O(n\log n)$	$O(n\log n)$	稳定	$O(1)$	n大时较好
堆	$O(n\log n)$	$O(n\log n)$	不稳定	$O(1)$	n大时较好

，如果一个算法的复杂度为c、 $\log_2 n$ 、n、 $n \cdot \log_2 n$,那么这个算法时间效率比较高，如果是 2^n 、 3^n 、 $n!$ ，那么稍微大一些的n就会令这个算法不能动了，后

很重要的问题，任何一个程序员都应该熟练掌握其概念和基本方法，而且要善于从数学层面上探寻其本质，才能准确理解其内涵。

一个算法的空间复杂度(Space Complexity)S(n)定义为该算法所耗费的存储空间，它也是问题规模n的函数。渐近空间复杂度也常常简称为空间复杂度。
对一个算法在运行过程中临时占用存储空间大小的量度。一个算法在计算机存储器上所占用的存储空间，包括存储算法本身所占用的存储空间，算法的输入输出数据所占用的存储空间这三个方面。算法的输入输出数据所占用的存储空间是由要解决的问题决定的，是通过参数表由调用函数传递而来的，它不随本算法空间与算法书写的长短成正比，要压缩这方面的存储空间，就必须编写出较短的算法。算法在运行过程中临时占用的存储空间随算法的不同而异，有的算法问题规模的大小而改变，我们称这种算法是“就地”进行的，是节省存储的算法，如这一节介绍过的几个算法都是如此；有的算法需要占用的临时工作单元曾大，当n较大时，将占用较多的存储单元，例如将在第九章介绍的快速排序和归并排序算法就属于这种情况。

个常量，即不随被处理数据量n的大小而改变时，可表示为O(1)；当一个算法的空间复杂度与以2为底的n的对数成正比时，可表示为O(log₂n)；当一个算法的O(n).若形参为数组，则只需要为它分配一个存储由实参传来的一个地址指针的空间，即一个机器字长空间；若形参为引用方式，则也只需要为其分配存储地址，以便由系统自动引用实参变量。

[songQQ/archive/2009/10/20/1587122.html](#)
[/85940806/archive/2011/03/12/141672.html](#)

经常用到，所以很多同学早就将算法打了个大礼包送还给了老师了，况且很多同学并没有学习过算法。这个系列就让对算法头疼的同...

是到现在还是没有结果！ (8个月前 #21楼) [查看回复\(1\)](#)

转载。。 (9个月前 #20楼)

#19楼)

写得很棒 (11个月前 #18楼)

也的猜想不成立。。。这一句还是再斟酌一下，虽然文章写于2013年，但是费马猜想在95年已被证明>2时成立 (1年前 #17楼)

1) 中语句3的频率为什么是：n-1, 不应该是n么？ (1年前 #16楼) [查看回复\(1\)](#)

年前 #15楼)

11年前 #14楼)

2019人工智能薪资

Python资料免费领取

会员任意学

Java薪资多少

怎样才能不被裁员

人脸识别算法

小项目外包