



结束了数据结构基本功的学习，接下来在真正开始撸真题之前，大家还需要具备**评价算法的能力**。

平时我们定义一个人是否“懂行”，一个重要的依据就是看这个人对于某一个事物是否具备正确的评价能力。

举个例子，同样是买手机，外行进到手机店，他关注的可能是手机有没有跑马灯、有没有皮套护体、有没有“八心八箭”——这些东西，任何一部手机随便包装一下就都有了，根本没法反映出这台手机的本质问题。但如果是一个相对懂手机的人，他可能就会去关注这台手机的芯片、内存、屏幕材质及分辨率等等，从而对手机的整体性能和质量作出一个合理的判断，这样他买到好手机的概率就更大。

回到做算法题上，也是一样的道理。在面试时，自己给出的算法到底过不过得去，这一点在面试官给出评语之前，自己就应该有所感知。做到这一点，你才会掌握改进算法的主动权。

本节我们要学习的就是评价算法的两个重要依据——时间复杂度和空间复杂度。

很多同学算法入门直接就跪在复杂度理解这一环。时间复杂度、空间复杂度，直接读概念确实太无聊，我们本节从代码入手，大家的理解会更直观一点。

时间复杂度

大家先来看这样一个问题：下面这段代码，一共会执行多少次？

```
function traverse(arr) {  
  var len = arr.length  
  for(var i=0;i<len;i++) {  
    console.log(arr[i])  
  }  
}
```

js

首先，最没有悬念的是函数里的第一行代码，它只会被执行1次：



其次没有悬念的是循环体：

```
console.log(arr[i])
```

js

for 循环跑了 n 次，因此这条语句就会被执行 n 次。

循环体上面的几个部分我们拆开来看，首先是 i 的初始化语句：

```
var i = 0
```

js

初始化只有1次，因此它也只会被执行1次。

接着是 $i < \text{len}$ 这个判断。这里有个规律大家可以记下：在所有的 for 循环里，判断语句都会比递增语句多执行一次。在这里，判断语句执行的次数就是 $n+1$ 。

再往下就是递增语句 $i++$ 了，它跟随整个循环体，毫无疑问会被执行 n 次。

假如把总的执行次数记为 $T(n)$ ，下面咱们就可以来做个简单的加法：

$$T(n) = 1 + n + 1 + (n+1) + n = 3n + 3$$

接下来我们看看规模为 $n*n$ 的二维数组的遍历，一共需要执行多少次代码：

```
function traverse(arr) {  
  var outLen = arr.length  
  
  for(var i=0;i<outLen;i++) {  
    var inLen = arr[i].length  
  
    for(var j=0;j<inLen;j++) {  
      console.log(arr[i][j])  
    }  
  }  
}
```

js

首先仍然是没有悬念的第一行代码，它只会被执行一次：

```
var outLen = arr.length
```

js



```
console.log(arr[i][j])
```

js

因为咱们是两层循环，所以这货会被执行 $n*n = n^2$ 次。

其它语句的计算思路和咱们第一个👉区别不大，这里我就不重复讲了，直接给大家答案：

```
function traverse(arr) {  
    const outLen = arr.length 1  
    1    n+1    n  
    for(let i=0;i<outLen;i++) {  
        const inLen = arr[i].length n  
        n    n*(n+1) n*n  
        for(let j=0;j<inLen;j++) {  
            console.log(arr[i][j]) n*n  
        }  
    }  
}
```

@稀土掘金技术社区

继续来做个求总执行次数 $T(n)$ 的加法看看：

$$T(n) = 1 + 1 + (n+1) + n + n + n + n*(n+1) + n*n + n*n = 3n^2 + 5n + 3$$

js

代码的执行次数，可以反映出代码的执行时间。但是如果每次都逐行去计算 $T(n)$ ，事情会变得非常麻烦。算法的时间复杂度，它反映的不是算法的逻辑代码到底被执行了多少次，而是随着输入规模的增大，算法对应的执行总次数的一个**变化趋势**。要想反映趋势，那就简单多了，直接抓主要矛盾就行。我们可以尝试对 $T(n)$ 做如下处理：

- 若 $T(n)$ 是常数，那么无脑简化为1



经过这么一波操作， $T(n)$ 就被简化为了 $O(n)$ ：

$$\begin{aligned} T(n) &= 10 \\ O(n) &= 1 \end{aligned}$$

$$\begin{aligned} T(n) &= 3n^2 + 5n + 3 \\ O(n) &= n^2 \end{aligned}$$

到这里，我们思路仍然是 **计算 $T(n)$ -> 推导 $O(n)$** 。这么讲是为了方便大家理解 $O(n)$ 的简化过程，实际操作中， $O(n)$ 基本可以目测，比如咱们上面的两个遍历函数：

```
function traverse1(arr) {  
  var len = arr.length  
  for(var i=0;i<len;i++) {  
    console.log(arr[i])  
  }  
}  
  
function traverse2(arr) {  
  var outLen = arr.length  
  
  for(var i=0;i<outLen;i++) {  
    var inLen = arr[i].length  
  
    for(var j=0;j<inLen;j++) {  
      console.log(arr[i][j])  
    }  
  }  
}
```

js

遍历 N 维数组，需要 N 层循环，我们只需要关心其最内层那个循环体被执行多少次就行了。

我们可以看出，规模为 n 的一维数组遍历时，最内层的循环会执行 n 次，其对应的时间复杂度是 $O(n)$ ；规模为 $n*n$ 的二维数组遍历时，最内层的循环会执行 $n*n$ 次，其对应的时间复杂度是 $O(n^2)$ 。



应的时间复杂度就表示为 $O(n^3)$ 。

常见的时间复杂度表达，除了多项式以外，还有 $\log n$ 。我们一起来看另一个算法：

```
function fn(arr) {  
    var len = arr.length  
  
    for(var i=1;i<len;i=i*2) {  
        console.log(arr[i])  
    }  
}
```

js

这个算法读取一个一维数组作为入参，然后对其中的元素进行跳跃式的输出。这个跳跃的规则，就是数组下标从1开始，每次会乘以二。

如何计算这个函数的时间复杂度呢？在有循环的地方，我们关心的永远是最内层的循环体。这个算法中，我们关心的就是 `console.log(arr[i])` 到底被执行了几次，换句话说，也就是要知道 $i < n$ ($len === n$) 这个条件是在 i 递增多少次后才不成立的。

假设 i 在以 $i=i*2$ 的规则递增了 x 次之后， $i < n$ 开始不成立（反过来说也就是 $i \geq n$ 成立）。那么此时我们要计算的其实就是这样一个数学方程：

$$2^x \geq n$$
js

x 解出来，就是要大于等于以 2 为底数的 n 的对数：

$$x \geq \log_2 n$$

@掘金技术社区

也就是说，只有当 x 小于 $\log_2 n$ 的时候，循环才是成立的、循环体才能执行。注意涉及到对数的时间复杂度，底数和系数都是要被简化掉的。那么这里的 $O(n)$ 就可以表示为：



没错，这时的主要矛盾，就变成了一个对数表达式。

关于常见的时间复杂度，我们会在后面讲到具体知识点（尤其是排序算法）时，结合实例来给大家做分析。这里大家首先要认识一下常见时间复杂度有哪些，并且对这些常见时间复杂度之间的大小关系做个把握。

常见的时间复杂度按照从小到大的顺序排列，有以下几种：

常数时间 对数时间 线性时间 线性对数时间 二次时间 三次时间 指数时间

$O(1)$ $O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$ $O(n^3)$ $O(2^n)$

@稀土掘金技术社区

空间复杂度

空间复杂度是对一个算法在运行过程中临时占用存储空间大小的量度。和时间复杂度相似，它是内存增长的**趋势**。

常见的空间复杂度有 $O(1)$ 、 $O(n)$ 和 $O(n^2)$ 。

理解空间复杂度，我们照样来看一个🍌：

```
function traverse(arr) {  
  var len = arr.length  
  for(var i=0;i<len;i++) {  
    console.log(arr[i])  
  }  
}
```

js

在 `traverse` 中，占用空间的有以下变量：

arr
len
i

js



有开辟新的内存空间。因此，整个 `traverse` 函数对内存的占用量是恒定的，它对应的空间复杂度就是 $O(1)$ 。

下面我们来看另一个🍌，此时我想要初始化一个规模为 n 的数组，并且要求这个数组的每个元素的值与其索引始终是相等关系，我可以这样写：

```
function init(n) {  
    var arr = []  
    for(var i=0;i<n;i++) {  
        arr[i] = i  
    }  
    return arr  
}
```

js

在这个 `init` 中，涉及到的占用内存的变量有以下几个：

```
n  
arr  
i
```

js

注意这里这个 `arr`，它并不是一个一成不变的数组。`arr` 最终的大小是由输入的 n 的大小决定的，它会随着 n 的增大而增大、呈一个线性关系。因此这个算法的空间复杂度就是 $O(n)$ 。

由此我们不难想象，假如需要初始化的是一个规模为 $n*n$ 的数组，那么它的空间复杂度就是 $O(n^2)$ 啦。

小结

结束了本节的学习，相信各位对时间复杂度和空间复杂度都有了一个感性的认知和初步的了解。在后续的学习中，我们会在必要的时候继续为大家提点真题中的时间复杂度和空间复杂度，带领大家在实战中强化对理论概念的认知。

(阅读过程中有任何想法或疑问，或者单纯希望和笔者交个朋友啥的，欢迎大家添加我的微信xyalinode与我交流哈~)



输入评论 (Enter换行, ⌘ + Enter发送)

发表评论

全部评论 (54)



敲代码的小提琴手 Lv2

前端开发练习生 @ JDT 13天前

22/1/30 day1 明日二刷第七节 数组 顺带把三道题刷一遍, 把hot100和接下来要复习的题总结下 过一遍! 🙌

👍 点赞 🗨 1

console_man 4天前



👍 点赞 🗨 回复

江涛学编程 Lv1

前端开发 1月前

打开 2022.01.12

👍 点赞 🗨 回复



yanwuhc

前端白菜工程师 1月前

打卡

👍 点赞 🗨 回复

Running_C

前端攻城狮 @ 小公司不... 2月前

能理解这玩意是啥, 但是不会计算, 感觉像是回到了高中数学的排列组合 😞

👍 1 🗨 回复

小啾 Lv2

前端工程师 @ 公众号 ... 2月前

2^n 是怎么得到的

👍 点赞 🗨 1

用户507119... 1月前

$1 \times 2 \times 2 \times 2 \times 2 \times 2 \dots \times 2 = 2^n$



小啾 Lv2 前端工程师 @ 公众号 ... 2月前

指数时间复杂度是怎么得到的呢

点赞 回复



敲代码的小提琴手 Lv2 前端开发练习生 @ JDT 4月前

讲得不戳！复习下时间复杂度空间复杂度~~

点赞 回复

wangdian123 5月前

666 仰望大佬

点赞 回复



Wailen Lv2 ctrlc + ctrlv + 翻译中文... 6月前

终于有一篇关于复杂度的文章能看得懂了 😭

1 回复



月圆之夜 7月前

数学学渣流下了没文化的眼泪

点赞 回复



Mr.HoTwo 7月前

冲鸭~~

点赞 回复



懒惰的大螃蟹 cv工程师 8月前

通俗易懂

1 回复



MAT Lv2 前端 9月前

打卡

点赞 回复



三只萌新 Lv3 前端 9月前

打卡

点赞 回复



保密 前端开发 10月前

看了两遍，终于看懂了



打卡

👍 点赞 💬 回复



yangjay23 Lv2 互联网搬砖🧱 @ 杭州某... 11月前

打卡

👍 点赞 💬 回复



BarrySong4Real Lv1 一个路过的前端 @ 浑水... 1年前

对数复杂度那个，我总是学了忘，忘了学，这里倒是看明白了。

👍 点赞 💬 回复



已注销 1年前

大佬你讲的太好了，爱了爱了

👍 点赞 💬 回复



yz_ Lv2 前端开发 1年前

这个T和O到底是什么单词的缩写？

👍 点赞 💬 1



yz_ Lv2 1年前

Operational complexity

👍 点赞 💬 回复

查看全部 54 条回复 ▾

