

数据结构与算法

算法的时间复杂度和空间复杂度分析

通常，对于一个给定的算法，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。而在证明算法是正确的基础上，第二部就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，掌握基本的算法时间复杂度分析方法是很有必要的。算法执行时间需通过依据该算法编制的程序在计算机上运行时所消耗的时间来度量。而度量一个程序的执行时间通常有两种方法。

一、事后统计的方法

这种方法可行，但不是一个好的方法。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依赖于计算机的硬件、软件等环境因素，有时容易掩盖算法本身的优势。

二、事前分析估算的方法

因事后统计方法更多的依赖于计算机的硬件、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。

在编写程序前，依据统计方法对算法进行估算。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：

(1). 算法采用的策略、方法；(2). 编译产生的代码质量；(3). 问题的输入规模；(4). 机器执行指令的速度。

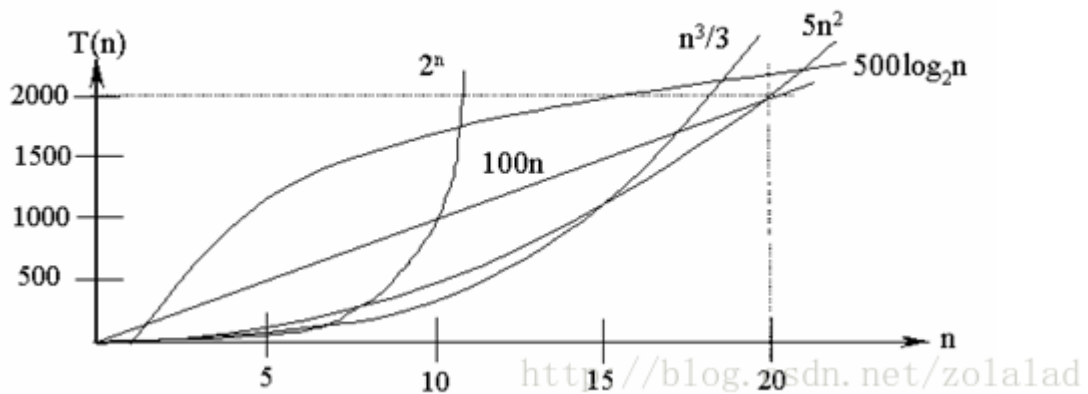
一个算法是由控制结构（顺序、分支和循环3种）和原操作（指固有数据类型的数据操作）构成的，则算法时间取决于两者的综合效果。为了便于比较同一个问题的不同算法，通常的做法是，从算法中选取一种对于所研究的问题（或算法类型）来说是基本操作的原操作，以该基本操作的重复执行的次数作为算法的时间度量。

1、时间复杂度

(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 $T(n)$ 。**(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 $f(n)$ ，使得当 n 趋近于无穷大时， $T(n)/f(n)$ 的极限值为不等于零的常数，则称 $f(n)$ 是 $T(n)$ 的同数量级函数。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。

另外，上面公式中用到的 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。**Landau 符号的作用在于用简单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。这里的 O ，最初是用大写希腊字母，但现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 。 $T(n) = O(f(n))$ 表示存在一个常数 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 $T(n)$ 的上界是 $C * f(n)$ 。其虽然对 $f(n)$ 没有规定，但是一般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 符号里隐藏着一个常数 C ，所以

$f(n)$ 里一般不加系数。如果把 $T(n)$ 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。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 $O(\log_2 n)$ ，线性阶 $O(n)$ ，线性对数阶 $O(n\log_2 n)$ ，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。



从图中可见，我们应该尽可能选用多项式阶 $O(n^k)$ 的算法，而不希望用指数阶的算法。

常见的算法时间复杂度由小到大依次为： $O(1) < O(\log_2 n) < O(n) < O(n\log_2 n) < O(n^2) < O(n^3) < \dots < O(2^n) < O(n!)$

一般情况下，对一个问题（或一类算法）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，以反映执行不同操作所需的相对时间，这种做法便于综合比较解决同一问题的两种完全不同的算法。

(3) 求解算法的时间复杂度的具体步骤是：

(1) 找出算法中的基本语句；

算法中执行次数最多的那条语句就是基本语句，通常是最内层循环的循环体。

(2) 计算基本语句的执行次数的数量级；

只需计算基本语句执行次数的数量级，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，并且使注意力集中在最重要的一点上：增长率。

(3) 用大O记号表示算法的时间性能。

将基本语句执行次数的数量级放入大O记号中。

如果算法中包含嵌套的循环，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++)
    x++;
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        x++;
```

第一个for循环的时间复杂度为 $O(n)$ ，第二个for循环的时间复杂度为 $O(n^2)$ ，则整个算法的时间复杂度为 $O(n+n^2)=O(n^2)$ 。

$O(1)$ 表示基本语句的执行次数是一个常数，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 $O(\log 2n)$ 、 $O(n)$ 、 $O(n \log 2n)$ 、 $O(n^2)$ 和 $O(n^3)$ 称为多项式时间，而 $O(2n)$ 和 $O(n!)$ 称为指数时间。计算机科学家普遍认为前者（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。

一般来说多项式级的复杂度是可以接受的，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。有些问题要复杂些，没有多项式时间的解，但是可以在多项式时间里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。还好欧拉告诉我们，这个数等于641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“解”是否正确，这类问题叫做NP问题。

(4) 在计算算法时间复杂度时有以下几个简单的程序分析法则:

(1).对于一些简单的输入输出语句或赋值语句,近似认为需要 $O(1)$ 时间

(2).对于顺序结构,需要依次执行一系列语句所用的时间可采用大O下"求和法则"

求和法则:是指若算法的2个部分时间复杂度分别为 $T_1(n)=O(f(n))$ 和 $T_2(n)=O(g(n))$,则 $T_1(n)+T_2(n)=O(\max(f(n), g(n)))$

特别地,若 $T_1(m)=O(f(m))$, $T_2(n)=O(g(n))$,则 $T_1(m)+T_2(n)=O(f(m) + g(n))$

(3).对于选择结构,如if语句,它的主要时间耗费是在执行then字句或else字句所用的时间,需注意的是检验条件也需要 $O(1)$ 时间

(4).对于循环结构,循环语句的运行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的时间耗费,一般可用大O下"乘法法则"

乘法法则: 是指若算法的2个部分时间复杂度分别为 $T_1(n)=O(f(n))$ 和 $T_2(n)=O(g(n))$,则 $T_1 T_2=O(f(n)g(n))$

(5).对于复杂的算法,可以将它分成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

另外还有以下2个运算法则:(1) 若 $g(n)=O(f(n))$,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

(5) 下面分别对几个常见的时间复杂度进行示例说明:

(1)、 $O(1)$

```
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 $O(1)$ 。**

(2)、 $O(n^2)$

2.1. 交换i和j的内容

```
sum=0;                (一次)
for(i=1;i<=n;i++)      (n+1次)
    for(j=1;j<=n;j++)  (n2次)
        sum++;         (n2次)
```

解：因为 $\Theta(2n^2+n+1)=n^2$ （ Θ 即：去低阶项，去掉常数项，去掉高阶项的常参得到），所以 $T(n)=O(n^2)$;

2.2.

```

for (i=1; i<n; i++)
{
    y=y+1;           ①
    for (j=0; j<=(2*n); j++)
        x++;         ②
}

```

解：语句1的频度是 $n-1$ 语句2的频度是 $(n-1)*(2n+1)=2n^2-n-1$ $f(n)=2n^2-n-1+(n-1)=2n^2-2$;

又 $\Theta(2n^2-2)=n^2$ 该程序的时间复杂度 $T(n)=O(n^2)$.

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，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。

(3)、 $O(n)$

```

a=0;
b=1;           ①
for (i=1; i<=n; i++) ②
{
    s=a+b;      ③
    b=a;        ④
    a=s;        ⑤
}

```

解：语句1的频度：2，

语句2的频度： n ,

语句3的频度： $n-1$,

语句4的频度： $n-1$,

语句5的频度： $n-1$,

$T(n)=2+n+3(n-1)=4n-1=O(n)$. (4)、 $O(\log_2 n)$

```

i=1;           ①
while (i<=n)
    i=i*2;     ②

```

解：语句1的频度是1, 设语句2的频度是 $f(n)$, 则： $2^{f(n)} \leq n; f(n) \leq \log_2 n$

取最大值 $f(n)=\log_2 n$, $T(n)=O(\log_2 n)$

(5)、 $O(n^3)$

```

for(i=0;i<n;i++)
{
    for(j=0;j<i;j++)
    {
        for(k=0;k<j;k++)
            x=x+2;
    }
}

```

解：当*i*=*m*, *j*=*k*的时候,内层循环的次数为*k*当*i*=*m*时, *j* 可以取 0,1,...,*m*-1 , 所以这里最内循环共进行了0+1+...+*m*-1=(*m*-1)*m*/2次所以,*i*从0取到*n*, 则循环共进行了: 0+(1-1)*1/2+...+(*n*-1)*n*/2=*n*(*n*+1)(*n*-1)/6所以时间复杂度为 $O(n^3)$.

(5) 常用的算法的时间复杂度和空间复杂度

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	$O(1)$	<i>n</i> 小时较好
交换	$O(n^2)$	$O(n^2)$	不稳定	$O(1)$	<i>n</i> 小时较好
选择	$O(n^2)$	$O(n^2)$	不稳定	$O(1)$	<i>n</i> 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	$O(n)$	<i>B</i> 是真数(0-9), <i>R</i> 是基数(个十百)
Shell	$O(n \log n)$	$O(n^s) \ 1 < s < 2$	不稳定	$O(1)$	<i>s</i> 是所选分组
快速	$O(n \log n)$	$O(n^2)$	不稳定	$O(n \log n)$	<i>n</i> 大时较好
归并	$O(n \log n)$	$O(n \log n)$	稳定	$O(1)$	<i>n</i> 大时较好
堆	$O(n \log n)$	$O(n \log n)$	不稳定	$O(1)$	<i>n</i> 大时较好

一个经验规则：其中*c*是一个常量，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 $2n$, $3n$, $n!$ ，那么稍微大一些的*n*就会令这个算法不能动了，居于中间的几个则差强人意。

算法时间复杂度分析是一个很重要的问题，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。

2、算法的空间复杂度

类似于时间复杂度的讨论，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。空间复杂度(Space Complexity)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。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和算法在运行过程中临时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，它不随本算法的不同而改变。存储算法本身所占用的存储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，有的算法只需要占用少量的临时工作单元，而且不随

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数与解决问题的规模 n 有关，它随着 n 的增大而增大，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。

如当一个算法的空间复杂度为一个常量，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 $O(\log_2 n)$ ；当一个算法的空间复杂度与 n 成线性比例关系时，可表示为 $O(n)$ 。若形参为数组，则只需要为它分配一个存储由实参传送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一个地址的空间，用它来存储对应实参变量的地址，以便由系统自动引用实参变量。

##

数组

1. 如何在一个1到100的整数数组中找到丢失的数字? google, Amazon, tencent

<https://leetcode-cn.com/problems/que-shi-de-shu-zi-lcof/>

2. 如何在给定的整数数组中找到重复的数字? 小米

<https://leetcode-cn.com/problems/shu-zu-zhong-zhong-fu-de-shu-zi-lcof/>

3. 给定一个存放整数的数组，重新排列数组使得数组左边为奇数，右边为偶数。要求：空间复杂度 $O(1)$ ，时间复杂度为 $O(n)$ 。百度

https://blog.csdn.net/dahai_881222/article/details/7824958>

<https://leetcode-cn.com/problems/diao-zheng-shu-zu-shun-xu-shi-qi-shu-wei-yu-ou-shu-qian-mian-lcof/>

4. 如何在未排序整数数组中找到最大值和最小值? 字节跳动

考点说明：这道题太过于简单，用两个变量同时保存最大和最小值，然后用一层for循环遍历整个数组，在循环过程中对数据进行对比，找出最大和最小值即可。由于题目简单，示例代码就不提供了，请大家对此感兴趣的同学自行编码实现。

5. 一个巨大无序数组中，查找一个不连续的自然数的节点，例如1, 2, 3, 5, 6, 7...第一个不连续自然数的节点是4。喜马拉雅FM

算法方向：插入排序，找到第一个不连续自然数就结束

6. 给定一个二进制数据位数，输出所有2进制数所对应的所有的自然数，要求时间复杂度优先。喜马拉雅

https://ask.csdn.net/questions/382489?sort=votes_count

7. 在Java中如何从给定排序数组中删除重复项? 小米

<https://leetcode-cn.com/problems/remove-duplicates-from-sorted-array/>>

<https://leetcode-cn.com/problems/remove-duplicates-from-sorted-array-ii/>

链表

1. 一个数组插入删除查找和链表的效率对比？如果一个数组要反复插入删除怎么优化降低时间复杂度？（一开始没想到，面试官提示其实就是垃圾回收的算法 原理就是“标记-查找”。每次删除的时候元素不是真的被删除了，而是先标记，最后统一移动数组元素，减少移动次数）腾讯

2. ArrayList查询第一个跟最后一个复杂度一样么？（我说一样）那LinkedList查询第一个跟最后一个复杂度一样么？（我说不一样，因为链表是从头往后查，只有前一个才能记录后一个的位置）你确定么？（想了下，LinkedList是双向的，查询第一个跟最后一个是一样的）那查询第一个跟倒数第二个呢？（这就不一样了，第一个直接给了头结点，倒数第二个需要从倒数第一个开始查询，走两步） 腾讯
3. arrayList底层原理 滴滴 字节跳动
4. 如何得到单链表的长度? 360
https://blog.csdn.net/weixin_41924879/article/details/100065838
5. 如何在不使用递归的情况下逆转单链表? 小米/美团/快手
<https://blog.csdn.net/shine0312/article/details/87938865>
6. 如何在一次遍历中找到单个链表的中间节点的值? 中国平安
<https://blog.csdn.net/skyejy/article/details/100112481>
7. 如何证明给定的链表是否包含循环?如何找到循环的头节点? 优酷 腾讯 滴滴
<https://blog.csdn.net/yangruxi/article/details/80333000>
8. 两个有交叉的单链表，求交叉点 华为
<https://blog.csdn.net/u011046509/article/details/80918952>>
<https://leetcode-cn.com/problems/intersection-of-two-linked-lists/>
9. 合并单链表 字节跳动/美团
10. 如何找链表倒数第n个元素？快慢指针 腾讯
<https://leetcode-cn.com/problems/lian-biao-zhong-dao-shu-di-ke-jie-dian-lcof/>
11. 求矩阵中连续1的个数 Number of Islands 三星
https://blog.csdn.net/weixin_33941350/article/details/91521633>
<https://leetcode-cn.com/problems/number-of-closed-islands/>
12. 大数相加(今日头条，美团)
<https://leetcode-cn.com/problems/add-two-numbers-ii/>

队列&堆栈

1. 对比一下队列和栈，以及它们底部实现 腾讯
2. 一个送礼的场景，礼物有权重属性，怎么根据权重进行对礼物进行处理，然后再排队分发，每次取一个礼物，怎么设计数据结构。喜马拉雅
3. 如何使用栈实现队列的功能? (广州荔枝FM)
https://blog.csdn.net/Jocker_coding/article/details/86655565>
<https://leetcode-cn.com/problems/implement-stack-using-queues/>
4. 两个栈实现一个队列 蘑菇街 小米
<https://leetcode-cn.com/problems/yong-liang-ge-zhan-shi-xian-dui-lie-lcof/>
https://blog.csdn.net/Jocker_coding/article/details/86655565
5. 两个队列实现一个栈 腾讯

<https://leetcode-cn.com/problems/implement-queue-using-stacks/>

二叉树

1. 已知前序遍历为{1,2,4,7,3,5,6,8}，中序遍历为{4,7,2,1,5,3,8,6}，它的二叉树是怎么样？ 58

二叉树的概念

2. 什么是平衡二叉树，它有什么特征 美团
3. 平衡二叉树和红黑树的区别？ 字节跳动
4. B 树，B+树 是什么区别？ 字节跳动
5. 如何在给定的二叉树中执行先序遍历？百度

<https://leetcode-cn.com/problems/binary-tree-preorder-traversal/>

6. 如何实现后序遍历算法？ 百度

<https://leetcode-cn.com/problems/binary-tree-postorder-traversal/>

7. 输入两棵二叉树 A 和 B，判断 B 是不是 A 的子结构。 爱奇艺

<https://leetcode-cn.com/problems/check-subtree-lcci/>

8. 请实现两个函数，分别用来序列化二叉树和反序列化二叉树。 YY

<https://leetcode-cn.com/problems/xu-lie-hua-er-cha-shu-lcof/>

9. 任意一颗二叉树，求最大节点距离 字节跳动 百度

<https://blog.csdn.net/liuyi1207164339/article/details/50898902>

HashMap

1. HashMap的底层原理是什么？线程安全么？ 百度 美团
2. HashMap中put是如何实现的？ 滴滴
3. 谈一下HashMap中什么时候需要进行扩容，扩容resize()又是如何实现的？
4. 什么是哈希碰撞？怎么解决？ 滴滴 美团
5. HashMap和HashTable的区别 小米
6. HashMap中什么时候需要进行扩容，扩容resize()是如何实现的？ 滴滴
7. hashmap concurrenthashmap原理 美团
8. arraylist和hashmap的区别，为什么取数快？ 字节跳动



1. 旋转输出矩阵 美团

<https://leetcode-cn.com/problems/rotate-matrix-lcci/>

2. 给定一个矩阵 `int matrixA[[m]][[n]]`, 每行每列都是增序的, 实现一个算法去寻找矩阵中的某个元素 `element`. 搜狗

https://blog.csdn.net/v_july_v/article/details/7085669

排序算法

1. top-k排序(堆排序, 位图法) 美团
2. 冒泡排序的手写 华捷艾米
3. 堆排序算法的手写 华捷艾米
4. 椭圆形场地有两个赛道, 可以同时提供两匹马比赛, 两匹马比赛后, 可以获知两匹马中跑的快的那匹马, 但是没有计时工具。问题, 如何最优的算法(比赛次数最少), 获知10匹马中速度最快的三匹马 阿里
5. 输入一个整型无序数组, 对堆排序的方法使得数组有序 阿里
6. 快速排序 手写实现 小米 CVTE
可以参考下面题目以快速排序实现
7. 直接插入排序手写实现 小米
可以使用这个思想实现下面的题目

<https://leetcode-cn.com/problems/average-salary-excluding-the-minimum-and-maximum-salary/>

<https://leetcode-cn.com/problems/relative-sort-array/>

查找算法

1. 有序数组的二分查找算法 百度 苏宁
<https://leetcode-cn.com/problems/binary-search/>
2. 如何在给定数组中执行二分法搜索? 苏宁
3. 设计一个算法, 已知某年某月某日是星期几, 求另外年月日对应的星期几 华为
4. n个人排成一圈报数, 报到3的退出, 循环直至最后一个, 问, 最后一个的原来号码是多少, 头条 https://blog.csdn.net/flyonedream/article/details/74942710?utm_source=blogxgwz8

串

1. 给定一个字符串, 请你找出其中不含有重复字符的 最长子串的长度。 字节跳动
<https://leetcode-cn.com/problems/longest-substring-without-repeating-characters/>
2. 给定一个字符串 `s`, 找到 `s` 中最长的回文子串。你可以假设 `s` 的最大长度为 1000。
<https://leetcode-cn.com/problems/longest-palindromic-substring/>
3. 给定一个只包括 '('', ')', '{', '}', '[', ']' 的字符串, 判断字符串是否有效。
有效字符串需满足:
左括号必须用相同类型的右括号闭合。 左括号必须以正确的顺序闭合。
<https://leetcode-cn.com/problems/valid-parentheses/>
4. 逆转一个句子中的单词?

<https://leetcode-cn.com/problems/fan-zhuan-dan-ci-shun-xu-lcof/>

5. 编写一个函数来查找字符串数组中的最长公共前缀。

<https://leetcode-cn.com/problems/longest-common-prefix/>

6. 字符串匹配, t1 [a,b,c], t2 :'abcdeadda' 滑动窗口 字节跳动
7. 将两个字符串表示二进制数相加, 输出对应和的二进制数 字节

<https://leetcode-cn.com/problems/add-binary/>

8. url按"."分组原地反转 头条

例如: www.toutiao.com www.oaituot.moc

<https://leetcode-cn.com/problems/reverse-words-in-a-string-iii/>

9. 大数相加(今日头条, 美团)

<<https://leetcode-cn.com/problems/add-two-numbers/>

其他算法

1. 常用的对称加密算法, 有什么同? 字节跳动
2. 如何在无序 (有负数) 的数组中查找是否存在和为target的两个数组合, twoSum(); 字节