第 0 节 算法入门

> 普及一下，计算机分为内存（RAM：Read Access Memory）和外存（ROM：Read Only Memory）。
>
> 内存比较小，现在的个人计算机大都是4 / 8 / 16G内存。程序的运行是在内存中进行的，也就是说，所有正在运行的程序所需要的内存之和不能超过计算机内存（这里的内存是虚拟内存，比物理内存要大一些，但是肯定没有外存大，这里可以先不深究虚拟内存的相关）的容量。

1. 评价算法的标准

- **准确性**：对于一个问题的算法来说，之所以称之为算法，首先它必须能够解决这个问题
- **健壮性**：通过这个算法编写的程序要求在任何情况下不能崩溃
- **可读性**：一个算法供人们阅读的容易程度（这个性质通常可以忽略，往往一个优秀的算法都会极致的追求效率，相比之下，可读性都不重要。eg: Linux内核源码，STL源码……
- **鲁棒性**：指一个算法处理处理异常值的能力

  比方说一个下面这段代码：

  ```
  int Division(int a, int b){
      return a / b;
  }
  ```

  它是一个除法函数，所谓鲁棒性就是指它能否处理`b`等于0的情况，显然上面这个除法算法就不具备鲁棒性，经过下面这种改动，它才具备了**鲁棒性**：

  ```
  int Divsion(int a, int b){
      assert(b);
      return a / b;
  }
  ```

  PS:

  - assert(exp)函数是C语言中的**断言函数**，如果参数exp为0，则程序会报错
  - 算法的表现形式往往以函数的形式呈现，一个具备指定功能的函数就是一个处理某个问题的算法
- **时间复杂度**：算法的运行时间
- **空间复杂度**：运行这个算法需要的内存空间

2. 时间复杂度

> 一个算法语句总的执行次数是关于问题规模N的某个函数，记为f(N)，N 称为问题的规模。语句总的执行次数记为T(N)，当N不断变化时，T(N)也 在变化，算法执行次数的增长速率和f(N)的增长速率相同。则有T(N) = O(f(N))，称O(f(n))为时间复杂度的O渐进表示法。

具体计算方法：

- 修改运行次数函数，去掉所有的常数项
- 修改运行次数函数，只保留最高次项
- 把剩下这一项的系数变为1

示例：

计算下面这个函数中每个循环都要执行多少次：

```
int Func(int n){
    int result = 0;
    for(int i = 0; i < n; ++i){
        for(int j = 0; j < n; ++j){
            result++;
        }
    }
    for(int i = 0; i < 2*n; ++i){
        result++;
    }
    int count = 10;
    while(count--){
        result++;
    }
    return result;
}
```

答案：Func(n) = n^2 + 2 *n + 10

----------时间复杂度计算----------

第零步，去掉常数项，剩下`n^2 + 2*n`

第一步，只保留最高次项，剩下了`n^2`

第二步，将剩下这一项的系数变为1，`n^2`的系数就是1，所以不需要再改变

结果：上面这个Func()函数的时间复杂度就是`O(n^2)`

> 递归类算法的时间复杂度计算：
>
> 时间复杂度等于：递归总次数 * 每次递归的次数

常见时间复杂度以及比较：

执行次数函数	阶	非正式术语
12	O(1)	常数阶
2n+3	O(n)	线性阶
$3n^2+2n+1$	$O(n^2)$	平方阶
$5\log_2 n+20$	O(logn)	对数阶
$2n+3n\log_2 n+19$	O(nlogn)	nlogn 阶
$6n^3+2n^2+3n+4$	$O(n^3)$	立方阶
2^n	$O(2^n)$	指数阶

速度依次减慢：

O(1) < O(lgn) < O(n) < O(nlgn) < O(n^2)< O(n^3)<O(2^n) < O(n!) < O(n^n)

3. 空间复杂度

> 空间复杂度是指：函数中创建对象的个数关于问题规模函数表达式。

看下面这段代码，这个函数在执行开始到结束一共需要多少内存空间?

```
void Func(int n, int size){
    void* p = NULL;
    for(int i = 0; i < n; ++i){
        p = malloc(sizeof(size));
    }
}
```

答案是：n*size + [4 | 8]

> PS:一个指针的大小在32为平台下占4个字节大小，在64位平台下占8个字节大小。

这个函数的空间复杂度是：O(n)。即去掉了常数项和系数(即size)，保留了含有变量的最高此项和此项次方数。

再看下面这段代码，问题同上?

```
void Func(){
    int arr[100] = {0};
}
```

答案是：100

按照上面的计算方式，如果此式只有常数项，那么将常数项变为1，即时间复杂度为：O(1)

下面是扩展内容：

算法存在最好、平均和最坏情况：

- 最坏情况：任意输入规模的最大运行次数(上界)
- 平均情况：任意输入规模的期望运行次数
- 最好情况：任意输入规模的最小运行次数，通常最好情况不会出现 (下界)

例如：在一个长度为N的线性表中搜索一个数据x

- 最好情况：1次比较
- 最坏情况：N次比较
- 平均情况：N/2次比较

在实际中通常关注的是算法的最坏运行情况，即：任意输入规模N，算法 的最长运行时间。

理由如下：

- 一个算法的最坏情况的运行时间是在任意输入下的运行时间上界
- 对于某些算法，最坏的情况出现的较为频繁 大体上看，平均情况与最坏情况一样