空间复杂度与时间复杂度

时间复杂度表示法

时间复杂度全称为**渐进时间复杂度，**表示的是算法执行时间与数据规模之间的增长关系，一般看程序中每行代码执行次数的总和，记录最大量级即可。

```
1  int cal(int n) {
2      int sum = 0;
3      int i = 1;
4      for (; i <= n; ++i) {
5          sum = sum + i;
6      }
7      return sum;
8  }
```

上述代码中，2、3行执行1次，4，5行执行n次，所以就是2n+2次，只关注最大的量级并且忽略系数，所以说上述代码的时间复杂度就是O(n)。

> **加法法则：总复杂度等于量级最大的那段代码的复杂度**
>
> **乘法法则：嵌套代码的复杂度等于嵌套内外代码复杂度的乘积**

常见的时间复杂度主要包括一下几种：

- 常量阶：O(1)
- 对数阶：O(log n)
- 线性阶：O(n)
- 线性对数阶：O(nlog n)
- 指数阶： $O(2^n)$
- 阶乘阶：O(n!)
- 平方阶$O(n^2)$、立方阶$O(n^3)$ .... k次方阶$O(n^k)$

上面列出的时间复杂度包括两类，**多项式量级**和**非多项式量级**。其中O(2n) 和 O(n!)属于非多项式量级。**非多项式量级的算法随着数据规模增大，执行时间会急剧增加。**

几类常见的时间复杂度及案例

1. 对数阶O(logn)、O(nlogn)

对数阶是非常常见也是最难分析的一种。

```
int i = 1;
while (i <= n) {
    i = i * 2;
}
```

在上述代码中，i的每次取值分别是2^0, 2^1, 2^2, 2^3 ... 2^x = n。所以只需要知道x的值，就知道代码执行了多少次。2^x = n --> x = $\log_2 n$。**归并排序和快速排序的时间复杂度都是$O(n\log_n)$。**

空间复杂度表示法

空间复杂度全称就是**渐进空间复杂度**（asymptotic space complexity），**表示算法的存储空间与数据规模之间的增长关系**

一般常见的空间复杂度主要是O(1), O(n), $O(n^2)$，比较简单。看代码申请了多少内存就好。

复杂度分析

主要包括四个概念：最好时间复杂度，最差时间复杂度，平均时间复杂度和均摊时间复杂度。

以一个代码举例：

```
1   // n 表示数组 array 的长度
2   int find(int[] array, int n, int x) {
3     int i = 0;
4     int pos = -1;
5     for (; i < n; ++i) {
6       if (array[i] == x) {
7          pos = i;
8          break;
9       }
10    }
11    return pos;
12  }
```

在上述代码中

- 如果第6行的判断条件在第1次就成立，那么时间复杂度就是O(1)，这就是最好时间复杂度。
- 如果数组中不存在x变量，那么时间复杂度就是O(n)，这就是最坏时间复杂度。

平均时间复杂度

因为最好时间复杂度和最坏时间复杂度对应的是极端情况，所以引入平均时间复杂度。

平均时间复杂度就是把每一种情况下需要遍历的元素个数累加起来，然后除以n+1，对于上述代码就是 $\frac{1+2+3+...+n+n}{n+1} = \frac{n(n+3)}{2(n+1)}$ 。因为时间复杂度可以省略系数、低阶、常量，所以上述公式结果简化后，平均时间复杂度就是 $\frac{n^2+3n}{2n+2}$ --> n 。所以时间复杂度仍然是O(n)。

但是上述计算忽略了每种情况出现的概率问题，比如说要查找的变量x在数组中和不在数组中的**假设**都是 $\frac{1}{2}$ （此处仅为假设概率）。那么计算时间复杂度的过程就是

$$1 * \frac{1}{2n} + 2 * \frac{1}{2n} + 3 * \frac{1}{2n} + ... + n * \frac{1}{2n} + n * \frac{1}{2} = \frac{3n+1}{4}$$

省略后依然是O(n)。这个值就是加权平均值，也叫做期望值。**只有在对于不同情况下时间复杂度有量级上的差异时，才会考虑区分三种复杂度。**

均摊时间复杂度

均摊时间复杂度的适用范围更小，只有在**大部分情况下复杂度都很低，个别情况下复杂度很高，而且这些操作存在前后连贯固定的时序关系时。**通过摊还分析法将较高复杂度的那次操作的时间平摊到其他操作上。