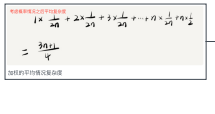
1.复杂度分析（渐进复杂度）

5. 最好情况、最坏情况、平均情况、均摊时间复杂度




```
// 在数组array的长度
int Find(int[] array, int n, int x) {
  int i = 0;
  int pos = -1;
  for (; i < n; ++i) {
    if (array[i] == x) {
      pos = i;
      break;
    }
  }
  return pos;
}
```

单例代码

- 最坏情况时间复杂度=即时间复杂度
  - 最好：第一个元素就是要找的，复杂度是O(1)
  - 最坏：最后一个元素是要找的，复杂度是O(n)
  - 平均：需要考虑所有情况的加权平均值，加权平均值

首要：
当代码的计算存在不确定性时(可能随时遇到条件语句进入执行)，例如 无序的数组（array）中，查找变量 x 出现的位置，找了就不再执行，这时，复杂度有多种情况

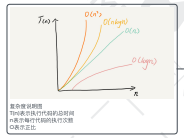
最好情况时间复杂度：以上面的例子来说，第一个元素就是想要找的，那么复杂度就是O(1)

最坏情况时间复杂度：想要的元素在最后一个位置，那么复杂度就是O(n)

平均情况复杂度：当要查找的变量 x 在数组中的位置，有 n+1 种情况：在数组的 0～n-1 位置中(n) 和 不在数组中(1)这两种情况；概率很诠都是1/2，计算过程如右图所示

均摊时间复杂度：均摊时间复杂度就是一种特殊的平均时间复杂度，例如 每一次 O(n) 的插入操作，都会跟着 n-1 次 O(1) 的插入操作，所以把这时间多的那次操作均摊到接下来的 n-1 次时间的平均操作上，可摊下来，这一组连续的操作的时间顺利时复杂度就是 O(1)

4.复杂度常见阶数



复杂度曲线图
T(n)表示执行代码的总时间
n表示每行代码的执行次数
O表示正比

复杂度也叫做渐进复杂度，包括时间复杂度和空间复杂度，用来分析算法执行效率与数据规模之间的增长关系，可以粗略地表示，越高阶复杂度的算法，执行效率越低。常见的复杂度并不多，从低阶到高阶有：O(1)、O(logn)、O(n)、O(nlogn)、O(n^2)

3.空间复杂度

空间复杂度全称就是渐近空间复杂度（asymptotic space complexity）

概念：表示算法的存储空间与数据规模之间的增长关系

常见的空间复杂度就是O(1)、O(n)、O(n^2)，像O(logn)、O(nlogn) 这样的对数阶复杂度平时都用不到

1. 事后分析法

用监控的方式计算程序运行的时间

局限性
1. 测试结果和原始种子硬件 2.测试结果受数据规模和影响很大

2. 时间复杂度

常说的时间复杂度，是时间复杂度

概念：算法的执行时间与数据规模之间的增长关系，用以不管低阶，系数，常量的执行时间多大，我们都可以忽略掉，因为它本身对增长趋势并没有影响

a.确定：所有代码的执行时间 T(n) 与每行代码的执行次数n 成正比
b.公式：$T(n) = O(f(n))$，f(n)表示代码执行次数的总和，O表示代码执行时间 随数据规模增长的 变化趋势 同成正比
c.表示方法：当n很大时一般计算时间即，常量、系数不左右增长趋势，所以可以表示为$T(n) = O(n)$、$T(n) = O(n^2)$

时间复杂度分析

注意：复杂度计算中，循环常量无论多大都是可以忽略的，复杂度关注的是

1.分析原则
a.只关注循环执行次数最多的一段代码

b.加法法则：总的时间复杂度就等于最级最大的那段代码的时间复杂度
公式：$T1(n)=O(f(n))$，$T2(n)=O(g(n))$；
则 $T(n)=T1(n)+T2(n)=max(O(f(n)), O(g(n)))=O(max(f(n), g(n)))$

c.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
公式：$T1(n)=O(f(n))$，$T2(n)=O(g(n))$；
则 $T(n)=T1(n)*T2(n)=O(f(n)*O(g(n)))=O(f(n)*g(n))$

2.复杂度量级：总体分为多项式量级和非多项式量级
a.多项式：
只有两个 O(2^n)和 O(n!) 剩余都为多项式量级的算法问题叫作 NP（Non-Deterministic Polynomial，中国定多项式）问题，当数据很大 n 越来越大时，非多项式量级算法的执行时间会急剧增加，求解问题的执行时间会无限增长

b.多项式：
(1) O(1)：常数级，代码的执行时间不随n 的增大而增长，只要算法中不存在循环语句，递归语句，即使有成千上万行代码，其时间复杂度也是O(1)
(2) O(logn)、O(nlogn)：对于n的数量都用比来之的，可以用算法为的形式表示其排序的值，例如等比数列，最终会有一个k=g = n ---> g = logan(n) 计算复杂度时 对数系的复杂度大论就算是多少可以通过换底公式进变换排恐略掉，所以对数系的复杂度都是logn，对于可以算出上面那个k值 例如 以并排序，快速排序时间复杂度是 O(nlogn)
(3) O(m+n)、O(m*n) 对于有两个数据规模的代码，例如 两个不同的for循环，那么就是O(m+n),数量小的是O(m*n)

分析原则1-1

```
int call(int n) {
  int sum = 0;
  int i = 1;
  for (; i < n; ++i) {
    sum = sum + i;
  }
  return sum;
}
```

- sum = 0 ← 1unit time(即常数，可忽略)
- for (; i < n; ++i) ← n unit time，执行次数最多的代码就是总的时间复杂度

分析原则1-2

```
int cal(int n) {
  int sum_1 = 0;
  int p = 1;
  for (; p < 100; ++p) {
    sum_1 = sum_1 + p;
  }

  int sum_2 = 0;
  int q = 1;
  for (; q < n; ++q) {
    sum_2 = sum_2 + q;
  }

  int sum_3 = 0;
  int i = 1;
  int j = 1;
  for (; i < n; ++i) {
    j = 1;
    for (; j < n; ++j) {
      sum_3 = sum_3 + i * j;
    }
  }

  return sum_1 + sum_2 + sum_3;
}
```

- 100 unit time 常数级
- n*1unit time
- n*n*1 unit time

标准说明：这段复杂度等于max(O1,O2,O3)
O1和O2表示每个部分的复杂度

分析原则1-3

```
int cal(int n) {
  int ret = 0;
  int i = 1;
  for (; i < n; ++i) {
    ret = ret + f(i);
  }
}

int f(int n) {
  int sum = 0;
  int i = 1;
  for (; i < n; ++i) {
    sum = sum + i;
  }
  return sum;
}
```

乘法法则：
这段代码的复杂度为O(n)，但是还嵌套着一层f(i)，f(i)的复杂度为O(n)，所以总体的复杂度就是O(n^2)

复杂度量级

- 多项式
  - 常数 O(1)
  - 对数阶 O(logn)
  - 线性阶 O(n)
  - 线性对数阶 O(nlogn)
  - 平方阶 O(n^2)、立方阶 O(n^3)、k次阶 O(n^k)
- 非多项式
  - O(2^n)
  - O(n!)

- 数据量小，或者是多项式量级的算法
- 数据量很大，非多项式量级的算法
- 数据量很大，多项式量级的算法
- 数据量很大，多项式量级的算法

- 数据量的两个数量级，小的忽略

XMind:ZEN | 试用模式