**算法Algorithm：**规则的有限集合，为解决特定问题规定的一系列操作

**算法的特性：**

1. 有限性：在有限步骤内正常结束，不能形成无穷循环
2. 确定性：算法中每一个步骤必须有确定含义
3. 可行性：可以被执行，即操作可以通过已经实现的基本运算执行有限的次数完成
4. 无二义性：算法中不能拥有二义性
5. 输入输出：至少有一个或多个输入输出

**算法设计要求：**

正确性（Correctness）算法的正确性是评价一个算法优劣的最重要的标准。

1. 程序不含语法错误，可正确编译执行
2. 程序对于几组输入数据能够得到满足规格说明要求的结果
3. 程序对于非法的输入数据能够得到满足规格说明要求的结果
4. 程序对于精心挑选的几组严苛的输入数据得到满足规格说明要求的结果

可读性（Readability）

算法的可读性是指一个算法可供人们阅读的容易程度。 （算法是可以阅读，理解和交流的）

健壮性（Robustness）

算法的健壮性是指一个算法对不合理数据输入的反应能力和处理能力,也称为容错性。

（当输入数据不合法时，算法也能做出相关的处理，而不是产生不可预计的效果）

高效率、低存储

**算法评价：**

一个算法的评价主要从时间复杂度和空间复杂度来考虑

1. 时间复杂度

算法的时间复杂度是指执行算法所需要的时间。

一般来说，计算机算法是问题规模n的函数f(n)，算法的时间复杂度也因此记做T(n)=Ο(f(n))

1. 空间复杂度

算法的空间复杂度是指算法需要消耗的内存空间。

一般来说，计算机算法是问题规模n的函数f(n)，算法的空间复杂度也因此记做S(n)=Ο(f(n))

**影响算法执行时间的因素：**

1. 算法选用的策略
2. 问题的规模N
3. 编写程序的语言
4. 编译程序产生的机器代码的质量
5. 计算机执行指令的速度

（后面三个因素与具体的机器、程序设计语言有关，当语言、机器确定后算法的执行时间主要和前两个因素有关）

**算法的存储量：**（算法所占空间）

输入数据所占空间

程序本身所占空间

辅助变量所占空间

**度量时间复杂度的两种方法：**

事后统计法（在程序运行结束之后直接查看运行时间的方式进行时间复杂度的统计）

缺点：必须执行程序，其他因素会掩盖算法本质

事前估算法（在计算机编译程序前对算法进行预估的方式时间复杂度的估算）

**算法执行时间：**一个算法的执行时间大致等于其所有语句执行时间的总和

（语句的执行时间是该条语句的执行次数和执行一次所需时间的乘积）

分析：不是针对实际执行时间精确地算出算法执行的具体时间，而是针对算法中基本操作语句（源语句）的执行次数进行估计，从而得到算法执行时间。

基本操作语句（源语句）指的是在程序里嵌套层次最深的，一般是在循环的内部嵌套层次最深、运行次数最多的这样一个操作或者一个语句，估算它执行了多少次，用它的执行时间来作为整个时间复杂度估算的一个依据。

语句频度是指该语句在一个算法中重复执行的次数

问题规模N：算法运行时所占空间S和所耗费的时间T给出一个数量关系的评价

问题规模N——对于不同的问题其含义不同

矩阵的问题规模是阶数

多项式运算的问题规模是多项式项数

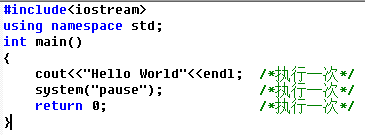
图的问题规模是顶点个数

集合运算的问题规模是集合中元素个数

**算法的时间复杂度，即算法的时间量度记作：T(n)=Ο(f(n))**

**f(n)是关于问题规模n的一个函数，表示算法的时间随着函数f(n)的增长而增长**

1. O(1) / O(C) C代表常数（常量阶时间复杂度）

#include<iostream>

using namespace std;

int main()

{

cout<<"Hello World"<<endl; /\*执行一次\*/

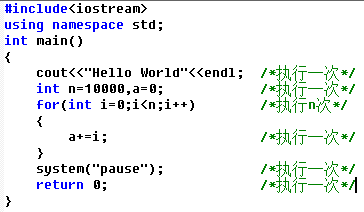
system("pause"); /\*执行一次\*/

return 0; /\*执行一次\*/

}

对于如上代码，我们一共执行了3次，即O(3)，其复杂度为O(1)，或称之为常数级时间复杂度。

1. O()（线性阶时间复杂度）

#include<iostream>

using namespace std;

int main()

{

cout<<"Hello World"<<endl; /\*执行一次\*/

int n=10000,a=0; /\*执行一次\*/

for(int i=0;i<n;i++) /\*执行n次\*/

{

a+=i; /\*执行一次\*/

}

system("pause"); /\*执行一次\*/

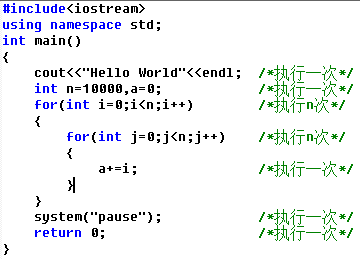
return 0; /\*执行一次\*/

}

对于如上代码，我们一共执行了n\*1+4次，即O(n\*1+4)，其复杂度为O(n)，或称之为线性阶时间复杂度。

1. O()（平方阶时间复杂度）

#include<iostream>

using namespace std;

int main()

{

cout<<"Hello World"<<endl; /\*执行一次\*/

int n=10000,a=0; /\*执行一次\*/

for(int i=0;i<n;i++) /\*执行n次\*/

{

for(int j=0;j<n;j++) /\*执行n次\*/

{

a+=i; /\*执行一次\*/

}

}

system("pause"); /\*执行一次\*/

return 0; /\*执行一次\*/

}

对于如上代码，我们一共执行了n\*n\*1+4次，即O(n\*n\*1+4)，其复杂度为O(n^2)，或称之为平方阶时间复杂度。

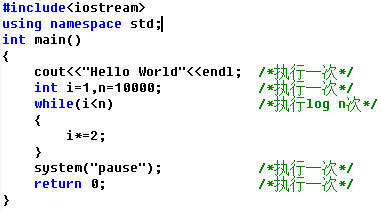
此外还有三层循环结构嵌套组成的O(n^3)级别的时间复杂度，称之为立方阶时间复杂度。

随着嵌套的增多，甚至还有O(n!)级，称之为阶层级时间复杂度，但是这种级别复杂度极高，程序运行极其缓慢。

1. O(log n)（对数阶时间复杂度）

#include<iostream>

using namespace std;

int main()

{

cout<<"Hello World"<<endl; /\*执行一次\*/

int i=1,n=10000; /\*执行一次\*/

while(i<n) /\*执行log n次\*/

{

i\*=2;

}

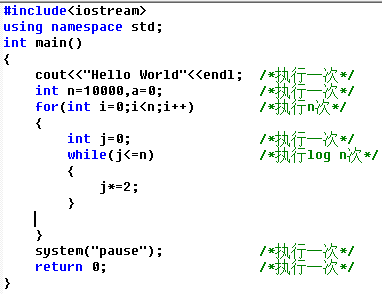
system("pause"); /\*执行一次\*/

return 0; /\*执行一次\*/

}

对于如上代码，与上文的线性增长不同，其i的增长是倍增的形式，也就是说i会随着运行次数的增加变大的趋势变更大，这样会比那些简单的用加法上涨的变量更快到达循环结构的边界，这样的代码时间复杂度一般为log级别，对于本样例，有O(log n+4)=O(log n)，称之为对数阶时间复杂度

1. O(n\*log n)

#include<iostream>

using namespace std;

int main()

{

cout<<"Hello World"<<endl; /\*执行一次\*/

int n=10000,a=0; /\*执行一次\*/

for(int i=0;i<n;i++) /\*执行n次\*/

{

int j=0; /\*执行一次\*/

while(j<=n) /\*执行log n次\*/

{

j\*=2;

}

}

system("pause"); /\*执行一次\*/

return 0; /\*执行一次\*/

}

对于如上代码，我们一共执行了n\*(log n+1)+4次，即O(n\*(log n+1)+4)=O(n\*log n)

**数据结构中常用的时间复杂度频率计数有：**

|  |  |  |  |
| --- | --- | --- | --- |
| O(1) | 常数型 | O() | 线性型 |
| O() | 平方型 | O() | 立方型 |
| O() | 指数型 | O() | 对数型 |
| O() | 二维型 |  |  |

各个复杂度的比较：O(1) **<** O() **<** O() **<** O() **<** O() **<** O() **<** O()

即：常数型**<**对数型**<**线性型**<**二维型**<**平方型**<**立方型**<**指数型

例：编写程序实现用一元人民币换成一分、两分、五分的硬币共50枚。

分析：设一分、两分、五分的硬币各为x、y、z枚，则

方法一：三重循环

#include<iostream>

using namespace std;

int main()

{

int i,j,k;

for(i=0;i<50;i++)

{

for(j=0;j<50;j++)

{

for(k=0;k<50;k++)

{

if(i+j+k==50 && i+2\*j+5\*k==100)

cout<<"一分："<<i<<"二分："<<j<<"五分："<<k<<endl;

}

}

}

system("pause");

return 0;

}

循环次数为51\*51\*51=132651，时间复杂度为O()

方法二：二重循环

#include<iostream>

using namespace std;

int main()

{

int i,j,k;

for(i=0;i<50;i++)

{

for(j=0;j<50;j++)

{

k=50-i-j;

if(i+2\*j+5\*k==100)

{

cout<<"一分："<<i<<"二分："<<j<<"五分："<<k<<endl;

}

}

}

system("pause");

return 0;

}

循环次数为51\*51=2601，时间复杂度为O()

方法三：二重循环（改进）

#include<iostream>

using namespace std;

int main()

{

int i,j,k;

for(k=0;k<20;k++)

{

for(j=0;j<50;j++)

{

i=50-j-k;

if(i+2\*j+5\*k==100)

{

cout<<"一分："<<i<<"二分："<<j<<"五分："<<k<<endl;

}

}

}

system("pause");

return 0;

}

循环次数为21\*51=1027，时间复杂度为O()

方法四：单重循环

则z<13 (50/4)

#include<iostream>

using namespace std;

int main()

{

int i,j,k;

for(k=0;k<13;k++)

{

j=50-4\*k;

i=50-j-k;

cout<<"一分："<<i<<"二分："<<j<<"五分："<<k<<endl;

}

system("pause");

return 0;

}

循环次数为13，时间复杂度为O(1)，常数阶