**三种求幂算法的用时比较**

\*\*\*

**时间：2021-10-6**

**第一章：介绍**

不同的算法在应对数据规模的增长时会呈现出不同的反应，它们用于计算的时间的增长与这些算法的时间复杂度（Time Complexities）有关。根据题目的要求，我通过两个循环一个递归三种不同的算法（分别命名为：Algorithm 1, Algorithm 2(iterative version), Algorithm 2(recursive version)）在控制变量的情况下计算同等增长的数据规模，探究算法用时增长与算法时间复杂度的关系。

我的任务是：‎

‎ （1） 实现Algorithm 1和Algorithm 2的‎‎迭次‎‎版本：‎

‎ （2） 分析两种算法的复杂性：‎

（3） 测量和比较Algorithm 1的性能和Algorithm 2的迭次和递归实现‎X=1.0001‎和‎N‎= 1000， 5000， 10000， 20000， 40000， 60000， 80000， 100000。‎

**第二章：算法详述**

算法要求实现给出double X和long int N后，求出X^N。由于单次计算时间过短没有统计意义，程序将得到重复计算long int K次后的总用时，通过后期数据处理来计算算法进行单次计算的用时。

主函数

在main()中，内置确定算法循环次数long int K，通过用户输入得到double X和long int N，再将用户输入1、2或3赋值到int type，通过三叉if选择到不同的算法进行计算。在每一个选择中都会在算法循环前开启计时器，结束后关闭计时器，再在最后进行结果、时间的输出。

奇偶判断函数

在程序中的函数名为IsEven，描述为int IsEven(long int N)

利用求余数函数求出N的余数，若为0即为偶数，函数返回1。

Algorithm 1

在程序中的函数名为POW1，描述为double POW1(double X, long int N)

遍历相乘。若N=0，则返回1，否则算法首先将X赋值给变量double out，再通过一个for循环，将out与X相乘N-1次，返回out，从而得出X^N。

Algorithm 2 (iterative version)

在程序中的函数名为POW2I，描述为double POW2I(double X, long int N)

N可以分解成 2^2k+···+2^3+2^2+2^1+2^0

则X^N可以分解成X^2k+···+X^8+X^4+X^2+X^1中若干项的乘积

相当于将N转换为二进制，若哪一位上为1就累积X^2k。

若N=0，则返回1。否则

用循环实现：double out初始值为1，用于储存累积的结果。while循环即为十进制转换为二进制“除2取余”方法的代码实现。N每步整除2，X每步自乘（相当于X^2k，k每步加一），当有余数为1时，即为N的二进制码上该位为1，将X^2k乘入out。最后函数返回out即可。

Algorithm 2 (recursive version)

在程序中的函数名为POW2R，描述为POW2R(double X, long int N)

若N=0，则返回1。否则

用递归方式实现以下思路：

如果N为偶数 X^N=X^(N/2)\*X^(N/2);

如果N为奇数 X^N=X^((N-1)/2)\* X^((N-1)/2)\*X;

即X进行log2(N)或者log2(N)+1次（N为奇数）平方后得到X^N。首先设置递归底点，当N=1时函数返回X。在剩余情况下（即“递”未结束）返回值改为POW2R(X\*X, N/2)，相当于对X进行一次平方以及将N整除2来计数（因为当N=1时就会结束）。由于N奇偶数的不确定，使用 if结构和IsEven函数来针对输入N的为奇与否选择POW2R(X\*X, N/2)\*X。

**第三章：测试结果**

根据任务（3）完成测试，测试将X定为1.0001，N‎= 1000， 5000， 10000， 20000， 40000， 60000， 80000， 100000。为了时间输出可观测，针对不同算法和N的大小将算法重复K次。记录计时器跳动次数、总时间以及单次时间。

结果如下：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | N | 1000 | 5000 | 10000 | 20000 | 40000 | 60000 | 80000 | 100000 |
| Algorithm 1 | Iterations(K) | 10000 | 10000 | 10000 | 10000 | 1000 | 1000 | 1000 | 1000 |
| Ticks | 80 | 396 | 800 | 1599 | 316 | 479 | 640 | 799 |
| Total Time(sec) | 0.080000 | 0.396000 | 0.800000 | 1.599000 | 0.316000 | 0.479000 | 0.640000 | 0.799000 |
| Duration(sec) | 0.000008 | 0.000040 | 0.000080 | 0.000160 | 0.000316 | 0.000479 | 0.000640 | 0.000799 |
| Algorithm 2 (tierative version) | Iterations(K) | 1000000 | 1000000 | 1000000 | 1000000 | 1000000 | 1000000 | 1000000 | 1000000 |
| Ticks | 27 | 34 | 38 | 39 | 41 | 43 | 45 | 47 |
| Total Time(sec) | 0.027000 | 0.034000 | 0.038000 | 0.039000 | 0.041000 | 0.043000 | 0.045000 | 0.047000 |
| Duration(sec) | 0.000000027 | 0.000000034 | 0.000000038 | 0.000000039 | 0.000000041 | 0.000000043 | 0.000000045 | 0.000000047 |
| Algorithm 2 (recursive version) | Iterations(K) | 1000000 | 1000000 | 1000000 | 1000000 | 1000000 | 1000000 | 1000000 | 1000000 |
| Ticks | 60 | 78 | 88 | 92 | 99 | 101 | 106 | 109 |
| Total Time(sec) | 0.060000 | 0.078000 | 0.088000 | 0.092000 | 99.000000 | 0.101000 | 0.106000 | 0.109000 |
| Duration(sec) | 0.000000060 | 0.000000078 | 0.000000088 | 0.000000092 | 0.000099000 | 0.000000101 | 0.000000106 | 0.000000109 |

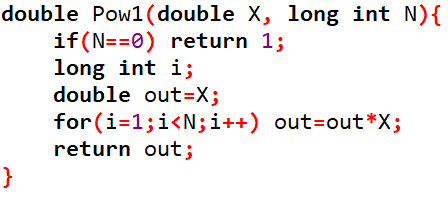
由表格可以得出：当N逐渐变大时算法二的迭次和递归实现比算法一用时更短且增长慢。算法二的迭次实现不论是用时长短还是增长速度都比递归实现高效。

进一步结论将结合时间复杂度的计算结果进行

**第四章：时间复杂度计算**

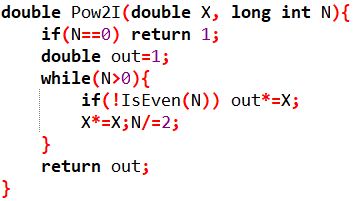
这里针对三个算法进行时间复杂度的计算，结合上表数据探究时间复杂度的概念和意义。

Algorithm 1 代码如下



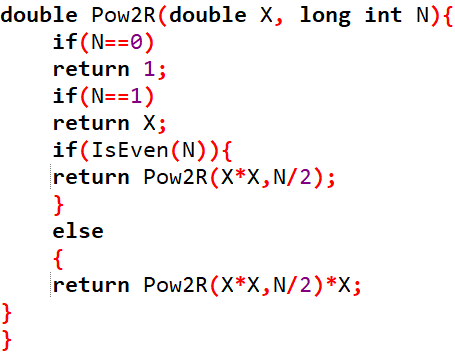
t(n)=1+1+1+N\*(1+1+1)=3+3\*N=O(N)

Algorithm 2 (iterative version) 代码如下



t(n)=1+1+log2(N)\*(1+1+2+2+2)=O(log2(N))

Algorithm 2 (recursive version) 代码如下



当N为偶数：t(n)=1+1+1+t(n/2)=3+t(n/2)

当N为奇数：t(n)=1+1+1+t(n/2)+1=4+t(n/2)

t(n)=4\*log2(N)+t(1)=O(log2(N))

分析与总结：

结合测试数据和时间复杂度的计算结果可知，算法的时间复杂度越低，算法在时间上的效率就越高。特别是遇到数据规模快速变大时依旧保持较平稳的时间表现。所以优秀的算法往往都有更低的时间复杂度。

**附录：代码（C语言）**

#include<stdio.h>

#include<time.h>

clock\_t start,stop;

//定义计时器

double duration,ticks,total;

//duration-算法单次运行所用时长; ticks-算法K次运行计时器打点次数； total-算法K次运行所用总时长

double X,result;

//X-底数; result-求幂的结果

long int N;

//N-指数

int IsEven(long int N);

/\*

用于判断输入是否为偶数。

输入long int N，若N为偶数，则返回 1，否则返回0

\*/

double Pow1(double X, long int N);

/\*

Algorithm 1在程序中的函数实现。

输入double X 与long int N,用遍历相乘的方法求得并返回X^N

若N=0，则返回1，否则算法首先将X赋值给变量double out，再通过一个for循环，将out与X相乘N-1次，返回out，从而得出X^N;

\*/

double Pow2I(double X, long int N);

/\*

Algorithm 2 (iterative version) 在程序中的函数实现

输入 double X 与 long int N， 返回 X^N

\*/

double Pow2R(double X, long int N);

/\*

Algorithm 2 (iterative version) 在程序中的函数实现

输入 double X 与 long int N ,返回 X^N

\*/

int main()

{

int type;

//判断选用算法类型的标记变量,1,2,3代表不同算法

//1-Algorithm 1; 2-Algorithm 2 (iterative version); 3-Algorithm 2 (recursive version)

double i;

long int K=1000000;

//算法重复的次数

printf("请间隔一空输入X和N\n");

scanf("%lf %ld",&X,&N);

//printf("%lf %ld\n",X,N);

printf("请输入将使用的算法编号:\n1-Algorithm1\n2-Algorithm2(iterative)\n3-Algorithm2(recursive)\n");

scanf("%d",&type);

start=clock();

if(type==1){

for(i=1;i<=K;i++) result=Pow1(X,N);

}

else if(type==2){

for(i=1;i<=K;i++) result=Pow2I(X,N);

}

else if(type==3){

for(i=1;i<=K;i++) result=Pow2R(X,N);

}

stop=clock();

ticks=((double)(stop-start));

total=((double)(stop-start))/CLK\_TCK;

//duration=((double)(stop-start))/(CLK\_TCK\*K);

//duration 由于太小，不予显示，转而在后期数据处理展现

printf("结果为%lf\n",result);

printf("运行%ld次打点数为%lf 用时为 %lfsec\n",K,ticks,total);

//printf("单次运行用时为%lfsec\n",duration);

return 0;

}

int IsEven(long int N){

if(N%2==0) return 1;

return 0;

}

double Pow1(double X, long int N){

if(N==0) return 1;

long int i;

double out=X;

for(i=1;i<N;i++) out=out\*X;

return out;

}

double Pow2I(double X, long int N){

if(N==0) return 1;

double out=1;

while(N>0){

if(!IsEven(N)) out\*=X;

X\*=X;N/=2;

}

return out;

}

/\*

函数名：POW2I

若N=0，则返回1。

否则

用循环实现：double out初始值为1，用于储存累积的结果。

while循环即为十进制转换为二进制"除2取余"方法的代码实现。

N每步整除2，X每步自乘（相当于X^2k，k每步加一），

当有余数为1时，即为N的二进制码上该位为1，将X^2k乘入out。

最后函数返回out。

\*/

double Pow2R(double X, long int N){

if(N==0)

return 1;

if(N==1)

return X;

if(IsEven(N)){

return Pow2R(X\*X,N/2);

}

else

{

return Pow2R(X\*X,N/2)\*X;

}

}

/\*

函数名：POW2R

X进行log2(N)或者log2(N)+1次（N为奇数）平方后得到X^N。首先设置递

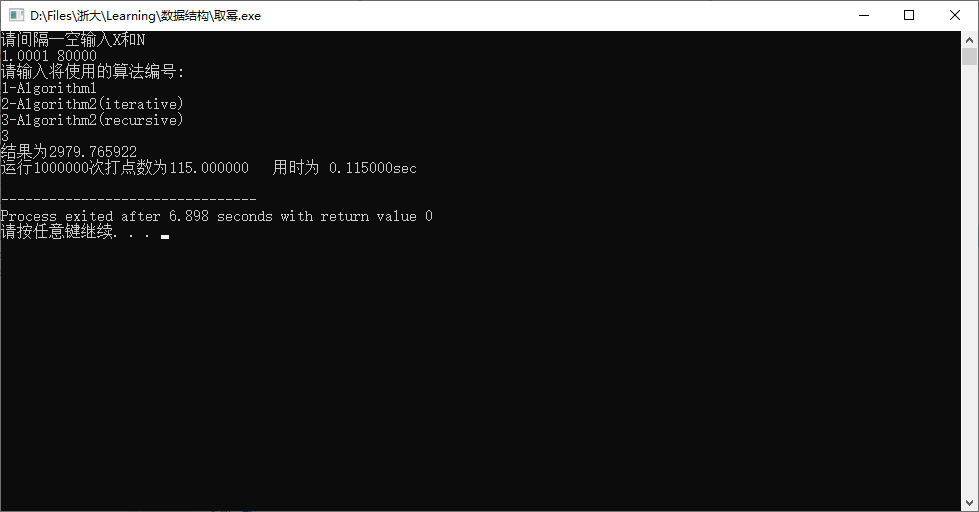
归底点，当N=1时函数返回X。在剩余情况下（即"递"未结束）返回值改为

POW2R(X\*X, N/2)，相当于对X进行一次平方以及将N整除2来计数（因为当

N=1时就会结束）。由于N奇偶数的不确定，使用 if结构和IsEven函数来针

对输入N的为奇与否选择POW2R(X\*X, N/2)\*X。

\*/



***I hereby declare that all the work done in this project titled***

***"三种求幂算法的用时比较" is of my independent effort.***

***我在此声明，在这个《三种求幂算法的用时比较》项目中所做的所有工作是我独立完成的。***