#### 优化算法★★

|  |  |
| --- | --- |
| man | 你这种方法每求得一个余数，都要到前面的余数序列中查找是否已存在该余数，其实还可以使用数学方法进行优化的。 |

小数的种类有三类：

第一类为有限小数，其分母的质因子分解式仅含2与5的两个质因子，即在所有的质数中，只有2和5才能相互被除尽；

第二类为无限循环小数，其分母，除9之外均为质因子，而9是3的倍数；

第三类为混合无限循环小数，其分母的质因子分解式含2与5及其他质因子的乘积，混合小数中含循环及不循环数字，不循环数字的个数称为延迟数，小数中重复的数字或循环的数字称为循环节。

分母质因子分解式2与5的最高次方，与延迟数的个数相同，例如 1/12＝ 0.08（3）的延迟数字为 2，而分母12的质因子分解式12 ＝ 22 × 3 中质因子2的最高次方亦为2。

那么循环节位数是多少呢？设分母为N，则10k－1能整除N的最小值k即为循环节位数。例如17，当k最小值为16即1016－1时能够被17整除，则循环节位数为16。又如14＝2×7，由于2的指数为1，即延迟数为1位，而剩下的7能整除106－1，则循环节位数为6。

参考程序如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70 | //高精度分数优化算法  #include <bits/stdc++.h>  using namespace std;  #define MAXN 100  int Delay(int x) //计算延迟数的位数  {  int n2=0,n5=0;  while(x%2==0)  {  n2++;  x/=2;  }  while(x%5==0)  {  n5++;  x/=5;  }  return n2>n5?n2:n5;  }  int Gcd(int a, int b) //欧几里得迭代求最大公约数  {  return a%b?Gcd(b,a%b):b;  }  int Repetend(int x) //计算循环节位数  {  int digit,k=9;  while(x%2==0) //质因数中除去所有2和5  x/=2;  while(x%5==0)  x/=5;  for(digit=1; digit<=MAXN; ++digit) //查找最小的k值，满足10^k－1被x整除  {  if(k%x==0)  break;  k=k%x\*10+9;  }  return digit;  }  void compute(int m,int n,int delay,int repetend)  {  int digit=delay+repetend;  for(int i=1; i<=digit; ++i)  {  m\*=10; //余数扩大10倍  cout<<m/n;  m%=n; //求下一个余数  if(m==0) //余数为0,则为有限小数  {  printf("\n");  return; //退出循环  }  }  cout<<"\nfrom "<<delay+1<<" to "<<digit<<"\n";  }  int main()  {  int m,n;  scanf("%d/%d",&m,&n);  printf("%d/%d=0.",m,n);  int gcd=Gcd(m,n); //求最大公约数  m/=gcd;  n/=gcd;  compute(m,n,Delay(n),Repetend(n));  return 0;  } |

此处用到的欧几里得算法又称辗转相除法，是用于计算两个整数a，b的最大公约数。其计算原理依赖于定理：gcd(a，b)＝gcd(b，a% b)，其中a＞b且a % b≠0。

欧几里得算法是计算两个整数的最大公约数的传统算法，但该算法在计算大素数时会稍有力不从心的感觉，此时，可以采用更具优势的Stein算法，有关Stein算法的详细介绍请自行查阅相关资料。