没有数学就没有算法 新知一下

逻辑航线

数论初步

N O I 基 础 算 法 系 列 课 程

版本：2.0.0

讲师：孙伟航

01

高精度运算

高精度运算

在已知的数据类型中，unsigned long long 所能容纳的数值为最大，即$2^{64}-1$，为 18446744073709551615。但是，当我们遇到了一个位数超过了2^{64}位的数字时，例如2^{65}，应该怎么处理呢？

这时，就需要我们通过技术手段，来对这些高精度的数字进行模拟计算了。

本节内容，我们将学习高精度加法，高精度减法，高精度乘法和高精度除法。

高精度的读入

逻辑航线

我们可以使用一个数组来保存整个高精度整数，并记录高精度整数的长度len。为了方便后面的计算，我们常常把整个数字进行倒置。即a[0]存储个位，a[1]存储十位……

```
#include <iostream>
#include <string>
using namespace std;
string num;
int a[105];//105位数字，是一个极大的数字
int main(int argc, char** argv)
{
	cin>>num;
	int len = num.size();
	for(int i=0; i<len; i++)
	{
		a[i] = num[len - 1 - i] - '0'; //存储其对应的真实数值
	}
	return 0;
}
```

01-1 高精度加法

高精度加法-模拟过程

在学习高精度加法之前，我们先来回忆一道小学数学题，看看加法是如何进位的。

对于给定式子999 + 111，我们列出竖式：

```
     9 9 9
 +   1 1 1
 ---------
   1 1 1 0
```

我们仔细想想他的进位过程，是否可以写成如下形式：

1、每一位对齐后，独立运算

```
     9   9   9
 +   1   1   1
 -------------
    10  10  10
```

2、判断低位是否应该进行进位，如果应该进位则高位进行加1。即使用低位除10，将商进位，将余数保留。

```
     9   9   9
 +   1   1   1
 -------------
    10  11   0
```

3、以此类推

```
     9   9   9
 +   1   1   1
 -------------
    11   1   0
```

高精度加法-核心代码

逻辑航线

```
a[i] += x; //按位进行加法运算
for(int i=0; i<len; i++){
    a[i+1] = a[i]/10; //高位加1
    a[i] %= 10; //低位保留余数
}

while(a[len]){ //循环判断最高位是不是需要进行进位
    a[len + 1] += a[len] / 10;
    a[len] %= 10;
    len++;
}
```

高精度加法-完整代码

逻辑航线

```cpp
#include <iostream>
#include <string>
using namespace std;
string numA,numB;
int a[105],b[105];//105位数字，是一个极大的数字
int lenA,lenB;
void cinNum(){
	cin>>numA>>numB;
	lenA = numA.size();
	lenB = numB.size();
	for(int i=0; i<lenA; i++){
		a[i] = numA[lenA - 1 - i] - '0';
	}
	for(int i=0; i<lenB; i++){
		b[i] = numB[lenB - 1 - i] - '0';
	}
}
```

```cpp
void countSum(){
	lenA = max(lenA,lenB); //保证后续的计算按照位数最高的进行循环。
	for(int i=0; i<lenA; i++){
		a[i] += b[i]; //按位进行加法运算
	}
	for(int i=0; i<lenA; i++){
		a[i+1] += a[i]/10; //高位进位
		a[i] %= 10; //低位保留余数
	}
	while(a[lenA]){ //循环判断最高位是不是需要进行进位
		a[lenA + 1] += a[lenA] / 10;
		a[lenA] %= 10;
		lenA++;
	}
}


int main(int argc, char** argv){
	cinNum();
	countSum();
	for(int i=lenA-1; i>=0; i--){ //逆序输出最后的结果;
		cout << a[i];
	}
	return 0;
}
```

01-2 高精度减法

高精度减法-模拟过程

首先，我们来回忆一下减法的运算。注意，在这个过程中，我们必须保证被减数大于减数，即结果一定是正值。

如果题目中给出了被减数小于减数的情况，我们则需要将两个数字调换位置，依然按照如下模拟过程计算，只不过在最后输出结果时，在数字前添加一个“-”号即可。

对于给定式子555 - 465，我们列出竖式：

$$\begin{array}{r} 5\ 5\ 5 \\ -\quad 4\ 6\ 5 \\ \hline 9\ 0 \end{array}$$

我们仔细想想他的进位过程，是否可以写成如下形式：

1、每一位对齐后，独立运算

$$\begin{array}{rrrr} & 5 & 5 & 5 \\ - & 4 & 6 & 5 \\ \hline & 1 & -1 & 0 \end{array}$$

2、判断低位是否小于0，如果是，则低位加10，高位减1。

$$\begin{array}{rrrr} & 5 & 5 & 5 \\ - & 4 & 6 & 5 \\ \hline & 0 & 9 & 0 \end{array}$$

高精度减法-核心代码


```
a[i] -= x; //按位进行加法运算

for(int i=0; i<len; i++){
     while(a[i]<0){
          a[i+1]--; //高位减1
          a[i] += 10; //低位加10
     }
}

while(len>1 && a[len-1] ==0){ //处理最高位的退位
          len--;
}
```

高精度减法-完整代码

逻辑航线

```cpp
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
string numA,numB;
int lenA,lenB,a[105],b[105];//105位数字，是一个极大的数字
bool sign = false; //标记是否答案为负数
bool cmp(string a,string b){ //比较a是否小于b
	if(a.size() != b.size()){
		return a.size() < b.size();
	}
	return a < b;
}

void cinNum(){
	cin>>numA>>numB;
	if(cmp(numA,numB)){ //a小于b，结果为负
		sign = true;
		swap(numA,numB); //交换两数的位置，保证大数在前
	}
	lenA = numA.size();
	lenB = numB.size();
	for(int i=0; i<lenA; i++){
		a[i] = numA[lenA - 1 - i] - '0';
	}
	for(int i=0; i<lenB; i++){
		b[i] = numB[lenB - 1 - i] - '0';
	}
}

void countSub(){
	lenA = max(lenA,lenB); //保证后续的计算按照位数最高的进行循环。
	for(int i=0; i<lenA; i++){
		a[i] -= b[i]; //按位进行加法运算
	}
	for(int i=0; i<lenA; i++){
		while(a[i]<0){
			a[i+1]--; //高位减1
			a[i] += 10; //低位加10
		}
	}
	while(lenA>1 && a[lenA-1] ==0){ //处理最高位的退位
		lenA--;
	}
}

int main(int argc, char** argv){
	cinNum();
	countSub();
	if(sign){ //当前值为负数
		cout << "-";
	}
	for(int i=lenA-1; i>=0; i--){ //逆序输出最后的结果；
		cout << a[i];
	}
	return 0;
}
```

01-3 高精度乘法

高精度乘法-模拟过程1

同样，我们来模拟一下乘法的过程。

对于给定式子253 x 67，我们列出竖式：

```
     2 5 3
 x     6 7
   1 7 7 1
 1 5 1 8
 1 6 9 5 1
```

拆分运算过程

1、每一位对齐后，独立相乘

```
       2    5    3
 x          6    7
      14   35   21
 12   30   18
```

2、计算乘积和。

```
       2    5    3
 x          6    7
      14   35   21
 12   30   18
 12   44   53   21
```

3、依次处理进位和长度增加

```
          2    5    3
   x           6    7
         14   35   21
     12  30   18
  1   6   9    5    1
```

注意：两个非零因数相乘，积的位数有两种可能：
1、等于两个数的位数之和
2、等于两个数的位数之和减1

所以，在处理进位的时候，根据你设定的循环长度的不同，处理方式也略有不同。详见实现代码

高精度乘法-模拟过程2

高精度乘法的关键是如何计算进位之前每一位上的乘积。

我们再看一个例子：

	3	2	1
x		5	4
	3×4	2×4	1×4
3×5	2×5	1×5	
[3]	[2]	[1]	[0]

假设我们使用一个全新的数组来存储各个位上的乘积，仔细观察这个等式，我们可以发现两个下标分别为i，j的数字，他们的乘积所处的位置正好是i+j之和。

对于上面这个示例，我们可以将321按照逆序存入数组A，即int A = {1,2,3}。同理，int B = {4,5}。注意，我们都是按照逆序进行存储的。

那么上式中的1 × 4就可以表示为：A[0] × B[0] = C[0+0]；同理， 2 × 4可以表示为A[1] × B[0] = C[0+1]。

核心代码

```
//计算乘积
for(int i=0; i<len1; i++){
    for(int j=0; j<len2; j++){
            a[i+j] += a1[i] * a2[j];
    }
}
```

高精度乘法-实现代码1

```cpp
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
string num1,num2; //输入的两个乘数
int a1[105],a2[105],len1,len2,a[205],len;
int main(int argc, char** argv){
     //分别读入两个数字，并逆序存储
     cin >> num1 >> num2;
     len1 = num1.size();
     for(int i=0; i<len1; i++){
          a1[i] = num1[len1-1-i] - '0'; //存储真实数字
     }
     len2 = num2.size();
     for(int i=0; i<len2; i++){
          a2[i] = num2[len2-1-i] - '0';
     }
     //计算乘积
     for(int i=0; i<len1; i++){
          for(int j=0; j<len2; j++){
                    a[i+j] += a1[i] * a2[j];
          }
     }
```

```cpp
     //长度算法1：设定积的长度为两数的位数-1
     len = len1 + len2 -1;
     //处理进位
     for(int i=0; i<len; i++){
          a[i+1] += a[i] /10;
          a[i] %= 10;
     }
     //处理最高位进位
     while(a[len]){
          a[len+1] += a[len] /10;
          a[len] %=10;
          len++;
     }
     for(int i=len-1; i>=0; i--){
          cout << a[i];
     }
     return 0;
}
```

高精度乘法-思考

在处理进位的时候，当我们设定积的位数为两个数的和的时候，该如何实现这个代码呢？

高精度乘法-实现代码2


```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
string num1,num2; //输入的两个乘数
int a1[105],a2[105],len1,len2,a[205],len;
int main(int argc, char** argv){
    //分别读入两个数字，并逆序存储
    cin >> num1 >> num2;
    len1 = num1.size();
    for(int i=0; i<len1; i++){
        a1[i] = num1[len1-1-i] - '0'; //存储真实数字
    }
    len2 = num2.size();
    for(int i=0; i<len2; i++){
        a2[i] = num2[len2-1-i] - '0';
    }
    //计算乘积
    for(int i=0; i<len1; i++){
        for(int j=0; j<len2; j++){
            a[i+j] += a1[i] * a2[j];
        }
    }
    //长度算法1: 设定积的长度为两数的位数和
    len = len1 + len2;
    //处理进位
    for(int i=0; i<len; i++){
        a[i+1] += a[i] /10;
        a[i] %= 10;
    }
    //处理高位为0的情况
    while (a[len] == 0 && len > 0){
        len--;
    }
    for(int i=len; i>=0; i--){
        cout << a[i];
    }
    return 0;
}
```

01-4 高精度除法

高精度除法-模拟过程

模拟计算 128 / 2

竖式过程：

步骤1：最高对2进行试除，得出商0，余128

```
        0
     ________
2  | 1  2  8
     0
     ________
     1  2  8
```

步骤2：十位对2进行试除，得出商6，余8

```
        0  6
     ________
2  | 1  2  8
     1  2
     ________
           8
```

步骤3：最后，各位再对2进行试除，得出商4

```
        0  6  4
     ________
2  | 1  2  8
     1  2
     ________
           8
           8
     ________
           0
```

最终，我们将高位的0舍去，得出最终结果为64。

高精度除法-核心代码

逻辑航线

通过上述模拟，我们可以将整个的计算模拟过程拆分如下：

1、从高位起，用每一位的数字试出除数，计算出商和余数
2、将余数乘10并加上低位数字，再次对除数进行试除
3、循环执行，直到将被除数的每一位数字都被进行过计算

实现代码

```
int x=0;
for(i=0; i<lena; i++){
     c[i] = (x*10 + a[i]) / b;
     x = (x*10 + a[i]) % b;
}
```

高精度除法-思考

前面的例子其实是计算高精度除以低精度，即除数只有一位。那么，如果我们想计算除数存在多位的情况，该如何处理呢？

高精度除法-实现代码

逻辑航线

```
#include <iostream>
#include <cstring>
#include <cstdio>
using namespace std;
string num1,num2; //输入的两个乘数
int a1[105],a2[105],len1,len2,a[205],len;
int main(int argc, char** argv){
	char al[100];
	int a[100],c[100],lena,i,x=0,lenc,b;
	memset(a,0,sizeof(a));
	memset(c,0,sizeof(c));
	gets(al);
	cin>>b;
	lena=strlen(al);
	for(i=0; i<lena; i++){
		a[i]=al[i] - '0'; //记录真实的数字
	}
	for(i=0; i<lena; i++){//除法运算
		c[i] = (x*10 + a[i]) / b;
		x = (x*10 + a[i]) % b;
	}
	lenc = 0; //从最高位开始判断是否为0
	while(c[lenc] == 0 && lenc<lena)
		lenc++;
	//从第一个非0数字开始进行输出
	for(i=lenc; i<lena; i++){
		cout << c[i];
	}
	return 0;
}
```

02

素数的N种求法

素数的N种求法

素数也叫做质数，即只有1和自身两个因数的自然数。

在各类比赛中，求出指定范围内的素数是非常常见的一类问题，但是对于不同的范围，求解的方式也各不相同，下面，我们来讲解一下求解素数的几个常用手段。

02-1 试除法

试除法

给定一个数字n，如果它不能整除[2，$\sqrt{n}$]内的所有数，那么它就是素数。

复杂度为O$\sqrt{n}$，适用范围n <= 10^{12}。

模板代码

```
bool IsPrime(int n){
     if(n <= 1)
          return false;
     if(n == 2)
          return true;
     for(int i = 2; i * i <= n; i++)
          if(n%i==0)
              return false;
          return true;
}
```

02-2

埃式筛法

埃式筛法

试除法最大的问题是枚举数量过多，运行时间太长。

在这里，我们推荐一种古老而简单的算法，可以快速求解[2，n]以内的所有素数。

具体算法如下:

对于给定的一个队列{2,3,4,5,6,7……}

A[N] = {2,3,4,5,6,7,8,9,10,11,12,13,14}
B[N] = {}　//桶　，索引为上面的数字　值：0,1区分是否为质数

2,3,4,5,6,7,8,9,10,11,12,13,14

1　1　1　1　1　1

首先，我们先输出最小的素数2，然后删掉队列中所有2的倍数。
之后，输出当前队列中最小的素数3，然后删掉队列中所有3的倍数。
再次，输出当前队列中最小的素数5，然后删掉队列中所有5的倍数。
重复执行以上步骤，直到队列为空。

该方法简单易懂，很容易实现。但是，由于受到空间复杂度的限制，使用该方法时n一般不能超过10^7。

埃式筛法-模板代码

逻辑航线

```
int E_sieve(int n){
    for(int i=0; i<=n; i++){
        visit[i] = false; //将所有数字设置为素数
    }
    //0和1不是素数
    for(int i=2; i*i<n; i++){
        if(!visit[i]){
            //将素数的倍数标记为合数，使用j=i*i能够尽量避免一些重复筛选
            //比如10这个数字就只会被2筛选一次，而不会被5进行筛选
            //但是，依然会存在重复筛选的情况，例如数字12，会被2和3进行筛选
            for(int j=i*i; j<n; j+=i){
                visit[j] = true;
            }
        }
    }
    int k=0;
    for(int i=2; i<=n; i++){
        if(!visit[i]){
                k++; //素数的个数}
    }
    return k;
}
```

02-3 欧拉筛法

欧拉筛法

通过观察，我们会发现，在埃式筛法中，每个合数会被筛选多次，例如数字12，它同时是2和3的倍数，所以它会被2和3两个质数筛选，因此，该方法不够快捷。于是，聪明的工程师们又发明了欧拉筛法。

欧拉筛法的基本思想

在埃氏筛法的基础上，保证每个合数只被它的最小质因子筛选一次，以达到不重复的目的。

从算法来说，N范围内的每个数都只会被筛一次，所以算法复杂度是O(n)。

s	0	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
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i = 2					1																										
i = 3							1			1																					
i = 4									1				1																		
i = 5											1					1										1					
i = 6													1																		
i = 7															1							1									
i = 8																	1														
i = 9																			1									1			
i = 10																					1										
i = 11																							1								
i = 12																									1						
i = 13																											1				
i = 14																													1		
i = 15																															1
i = 16																															
i = 17																															
i = 18																															
i = 19																															
i = 20																															
prime	0	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
	0	2	3	5	7	11	13	17	19	23	29																				

欧拉筛法-核心代码

```
void Is_Prime(){  //核心 欧拉筛代码
    memset(sf,true,sizeof(sf));
    for(int i=2; i<=limit; i++){ //外层枚举1~maxn
        if(sf[i])
        prime[++num]=i; //如果是质数就加入素数表,注意，从1开始进行存储
        for(int j=1; j<=num; j++){ //内层枚举num以内的质数
            if(i*prime[j]>maxn)
                break; //筛完结束
            sf[i*prime[j]]=false; //筛掉...

            if(i%prime[j]==0)
                break; //关键代码，避免重复筛
        }
    }
    sf[1]=false;
    sf[0]=false; //1 0 特判
}
```

红色代码便是欧拉筛法的核心代码，举例说明：

当i=8，j=1时，此刻prime[j]的值为2（第一个素数）。在正常运行下，因为8对2取余恒等于0，所以跳出循环，使i的值继续增长，即进入i=9。

但是，如果我们此刻不去终止代码，而是继续后面的循环，则会使j进行增长，即进入prime[j]的值为3的状态（第二个素数为3）。这时，8 * prime[j] = 24，于是，我们将24标记为非素数，即代码 sf[i*prime[j]]=false，此刻，我们认为是它的质因数3将它筛选出来的。

我们继续执行代码，当执行到i=12，j=1时，我们发现，此刻i * prime[j]依然等于24，即24又被筛选了一遍。而在这一次，将它筛选出来的，则是它的最小质因数2。

也就是说，当一个合数（例子中的8），能够被一个最小质数整除时（例子中的2），那么这个合数的倍数也一定能够被这个最小的质数整除，因此，我们无需再去遍历下一个质数（例子中的3），来标记当前合数的倍数（例子中的24），而是应该直接跳出循环。

这个合数的倍数（例子中的24，将会在后续的循环中，被一个更大的i（例子中的12）与当前最小的质数（例子中的2）来进行标记。这样，便保证了每个合数尽能够被自己最小的质因数所筛选。

欧拉筛法-模板代码

逻辑航线

```cpp
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
const int maxn=100000001;
int limit,num=0; //num 用来记筛到第几个质数
int prime[maxn]; //就是个素数表
bool sf[maxn]; //判断这个数是不是素数, sf[i]中的i是从1到maxn的数
void Is_Prime() //核心 欧拉筛代码
{
	memset(sf,true,sizeof(sf));
	for(int i=2; i<=limit; i++) //外层枚举1~maxn
	{
		if(sf[i])
		prime[++num]=i; //如果是质数就加入素数表,注意, 从1开始进行存储
		for(int j=1; j<=num; j++) //内层枚举num以内的质数
		{
			if(i*prime[j]>maxn)
				break; //筛完结束
			sf[i*prime[j]]=false; //筛掉...
			if(i%prime[j]==0)
				break; //关键代码, 避免重复筛
		}
	}
	sf[1]=false;
	sf[0]=false; //1 0 特判
}
```

```cpp
int main()
{
	cin >> limit;
	Is_Prime();
	for(int i=1; i<=num; i++)
	{
		cout << prime[i] << " ";
	}
	return 0;
}
```

03

分解质因数

分解质因数

根据因数的定义，只要是可以整除该数的正整数，均是该数的因数。因为可能为合数，也可能为质数。

从数学定义上可以得知，最小因数为1，最大因数为其本身。

把一个数拆成若干质因数的乘积，就叫做分解质因数。例如：6 = 2 * 3，12 = 2 * 2 * 3。

我们可以将12的分解质因数写成这样的形式：$12 = 2^2 * 3$ 。这样的分解结果是唯一的。

那我们如何在程序中来实现呢？其实很简单，与比较质数一样，我们从最小的质数开始判断，是否能整除当前数字，如果能，就把它记下来，并且把这个数除干净即可。

核心代码

```
for(int i=2; i<=n; i++) //n本身可能是个质数
{
	if(n%i == 0 && IsPrime(i)) //如果i是n的因数，且为质数
	{
		cnt = 0;
		while(n%i==0) //把i除尽，并统计次数
		{
			n /= i;
			cnt++;
		}
		cout << i << " " << cnt << endl;
	}
}
```

分解质因数-模板代码

```cpp
#include <iostream>
#include <cstdio>
using namespace std;
int cnt; //因数出现的次数
bool IsPrime(int n){
	if(n <= 1)
		return false;
	if(n == 2)
		return true;
	for(int i = 2; i * i <= n; i++)
		if(n%i==0)
				return false;
	return true;
}
int main(){
	int n;
	cin >> n;
	for(int i=2; i<=n; i++){ //n本身可能是个质数
		if(n%i == 0 && IsPrime(i)){ //如果i是n的因数，且为质数
			cnt = 0;
			while(n%i==0){ //把i除尽，并统计次数
				n /= i;
				cnt++;
			}
			cout << i << " " << cnt << endl;
		}
	}
	return 0;
}
```

04

因数个数

因数个数-公式

一个非i零自然数的因数个数公式，用一句话概括为：将质因数的指数加1后，依次连乘。

举个例子：

8可以分解为 2^3，所以8的因数数量一共有3+1，共4个，即1,2,4,8。

12可以分解为2^2 x 3，所以12的因数数量共有（2+1） x （1+1），共6个，即1,2,3,4,6,12。

那么，这个公式的原理是什么呢？道理其实很简单，这里隐藏着一个组合的概念。

我们来看12 = 2^2 x 3，我们可以换一个角度来理解，即12中的某一个因数包含2的情况共有2+1种：0个2,1个2,2个2；同理，包含3的情况共有2种：0个3,1个3，所以这个因数的可能情况共有3 x 2 = 6种。

第一种：0个2,0个3为1。
第二种：0个2,1个3为3。
第三种：1个2,0个3为2。
第四种：1个2,1个3为6。
第五种：2个2,0个3为4。
第六种：2个2,1个3为12。

因数个数-模板代码


```cpp
#include <iostream>

#define ll long long    //将longlong进行定义，便于后面的书写

using namespace std;

ll count(ll n){
    ll s = 1;
    for(ll i = 2 ; i * i <= n ; i++){
        if(n % i == 0){      //找到最小的质数，并将其除尽
            ll a = 0 ;
            while(n % i == 0){
                n /= i;
                a++;
            }
            s = s * (a+1) ; //套用因数个数公式
        }
    }
    if(n > 1)
        s = s * 2; //套用因数个数公式
    return s;
}
```

补充：虽然这段代码改变了n的数值，但是并不会对最终结果产生影响

05

最大公因数

最大公因数-欧几里得算法

逻辑航线

最大公因数，也称最大公约数、最大公因子，指两个或多个整数共有约数中最大的一个。a，b的最大公约数记为（a，b），同样的，a，b，c的最大公约数记为（a，b，c），多个整数的最大公约数也有同样的记号。求最大公约数有多种方法，常见的有质因数分解法、短除法、辗转相除法、更相减损法。

本节课我们介绍欧几里得算法，即辗转相除法。

公式：

gcd(a,b) = gcd(b,a mod b)

核心代码：

```
int gcd(int a, int b)
{
    if(b==0)
    {
        return a;
    }
    return gcd(b,a%b);
}
```

最大公因数-内置函数

C++的algorithm库内置了最大公因数的模板函数__gcd(a,b)

使用方法如下：

```
#include <iostream>
#include <algorithm>

using namespace std;

int main(int argc, char** argv)
{
	int a,b;
	while(cin>>a>>b)
	{
		cout << "最大公因数:" << __gcd(a,b)<<endl;
	}
	return 0;
}
```

06

最小公倍数

最小公倍数

定义：

几个数共有的倍数叫做这几个数的公倍数，其中除0以外最小的一个公倍数，叫做这几个数的最小公倍数。我们在欧几里得的基础上进行求解。 a，b的最小公倍数记为[a，b]，

公式：

lcm(a,b) = ab / gcd(a,b)

模板：

```
#include <iostream>
#include <algorithm>

using namespace std;

int main(int argc, char** argv)
{
	int a,b;
	while(cin>>a>>b)
	{
		cout << "最小公倍数:" << a*b/__gcd(a,b)<<endl;
	}
	return 0;
}
```

性质

1、对任意整数m，m(a1,a2……,ak) = (ma1,ma2,……,mak)，即整数同时成倍放大，最大公约数也放大相同倍数。

示例：9 = （9,18,27） = （3 x 3, 3 x 6, 3 x 9） = 3 x (3,6,9) = 3 x 3 = 9

本性质适用于最小公倍数。

2、对任意整数x，(a1，a2) = (a1，a2 + a1x)，即一个整数加上另一个整数的任意倍数，它们的最大公约数不变。

示例：（16，10) = (16-10，10) = （6，10） = （6,10-6） = （6,4） = （6-4,4） = （2,4） = （2,4-2 x 2） = （2,0） = 2

注意： (a，0) = a。

此条性质即为欧几米德公式的原理。

本性质不适用于最小公倍数。

3、（a1,a2,a3……,ak） = （(a1,a2),a3……ak）以及一个显然的推论(a1,a2,a3……ak+r) = （(a1……ak),(ak+1……ak)）

这是计算多元最大公约数的主要手段

例如求(12,18,21)，我们可以先求出(12,18) = 6，再把6带入，（6,21） = 3。即(12,18,21) = ((12,18),21) = (6,21) = 3。

本性质适用于最小公倍数。

4、[a1,a2] (a1,a2) = a1 x a2。即最大公约数 x 最小公倍数 = 原来两个数的乘积。

(16,10) x [16,10] = 2 x 80 = 160 = 16 x 10

此条性质即为最小公倍数原理。

07

阶乘取模

模运算与阶乘

在比赛中，我们常常会遇到对计算结果取余的试题，而给出的模数常常是10^9+7。这是因为，我们在计算过程中有可能产生过大的数字，所以，出题人为了方便大家，特此要求对结果取模。

基本性质：

(a + b) % p = (a%p + b%p) % p
(a - b) % p = (a%p - b%p + p) % p **注意，取模相减后一定要再加一个P，保证非负**
(a * b) % p = (a%p * b%p) % p

利用取模运算，能够保证每次计算得到的结果都不会超过P，能够有效的避免溢出问题。建议，对于这样的题目，我们算一步取一次模。

阶乘

阶乘是一个非负整数的概念，意思是从1开始连乘，N的阶乘表示为N！，其结果为1 x 2 x 3 x ...... x N。

计算方法

普通方式：循环从1开始累乘

递推方式：N！ = (N-1)! x N

由于阶乘的计算结果数值较大，所以我们一般不会直接求阶乘的计算结果，而是改为求其对某个数值取余后的结果。

阶乘取模的实现

逻辑航线

```
#include <iostream>
#include <algorithm>

using namespace std;

const int mod = 1e9 + 7;
int fac[1000005];

int main(int argc, char** argv)
{
	int n;
	while(cin>>n)
	{
		fac[0] = 1;
		for(int i=1; i<=n; i++)
		{
				fac[i] = 1LL * fac[i-1] * i % mod;
		}
		cout << "阶乘值" << fac[n] << endl;
	}
	return 0;
}
```

08

取数位

取数位

我们常常会需要知道某一个数字上的每一位是多少。这里最常用的方法就是除10取余法，示例代码如下：

```
#include <iostream>

using namespace std;

int main(int argc, char** argv)
{
	int n;
	cin >> n;
	while(n)
	{
		int num = n % 10;
		n /= 10; //去掉最后一位
		cout << "num:" << num << endl;
	}
	return 0;
}
```

取数位的题目类型较为多变，但是我们只要掌握了其取余算法的核心本质，就能够解决全部问题。

09

进制转换

十进制转K进制

我们回想一下上面的取数位的算法，思考一下为什么最后是除以10?

其实原因很简单，那就是因为我们平时使用的数字是10进制的。

在上面的算法中，我们先对K取余，得出当前位上的数字。然后，又用当前数字除以10得到了去除最低位后的数字。重复执行以上步骤，便能拆出每一位上的数字。

假设当前有个数字是K进制的，那么这个数字上的每一位能够出现的数字范围就是0到K-1，每一位对K取余，则结果的范围同样是也0到K-1。我们再将这个数字除以K，则会得到一个去除了最低位后的数字，同十进制拆数一样。

根据这个原理，我们可以很容易的将拆数函数改成进制转换的函数。

模板代码：

```cpp
#include <iostream>
using namespace std;
int digit[25];
int main(int argc, char** argv){
	int n,k,cnt; //n为要转换的数字，k为进制
	cin >> n>>k;
	printf("%d的%d进制数字：",n,k);
	while(n){
		int num = n % k;
		digit[++cnt] = num;
		n /= k; //去掉最后一位
	}
	for(int i=cnt; i>0; i--){
		cout << digit[i];
	}
	return 0;
}
```

K进制转十进制

当我们已知一个K进制数字的每一位之后，我们可以使用十进制转K进制的逆运算来求出这个K进制在十进制下的表示。

模板代码：

```
#include <iostream>
using namespace std;
int main(int argc, char** argv)
{
	int k,now;
	string n;
	cin >>n>>k;
	int len = n.size();
	for(int i=0; i<len; i++)
	{
		now = now * k + n[i] - '0';
	}
	cout << now;
	return 0;
}
```

补充：对于给定的A,B两个进制的转换，我们一般都是将其中一个转换为10进制，然后再转成另一个进制。

十进制转负K进制-负数除法

逻辑航线

我们想要做十进制转负K进制，首先需要弄明白负数除法。

想要弄清楚负数除法，我们需要弄明白负数中“负号”的意义，”负号”的意义在于“方向”。

例如：我们定义 “给我”为+； “给你”为-。

那么下列除法用语言描述就是：

1、12 / 3 = 4：给我12个苹果（+12），每次给我3个（+3），需要4次

2、12 / 4 = 3：给我12个苹果（+12），分4次给，每次给我3个（+3）

3、13 / 3 = 4...1：给我13个苹果（+13），每次给我3个（+3），需要4次，还得再给我1个（+1）

4、-12 / -3 = 4：给你12个苹果（-12），每次给你3个（-3），需要4次

5、-12 / 4 = -3：给你12个苹果（-12），分4次给，每次给你3个（-3）

6、-13 / -3 = 4...-1：给你13个苹果（-13），每次给你3个（-3），需要4次，还得再给你1个（-1）

7、-15 / 4 = -3…-3：给你15个苹果（-15），分4次给，，需要给你3次（-3）,还得再给你3个（-3）

8、12 / -4 = -3：给我12个苹果（+12），每次给你4个（-4），需要给你3次（-3）

9、15 / -2 = -7…1：给我15个苹果（+15），每次给你2个（-2），需要给你7次（-7），我手里1个（+1）。

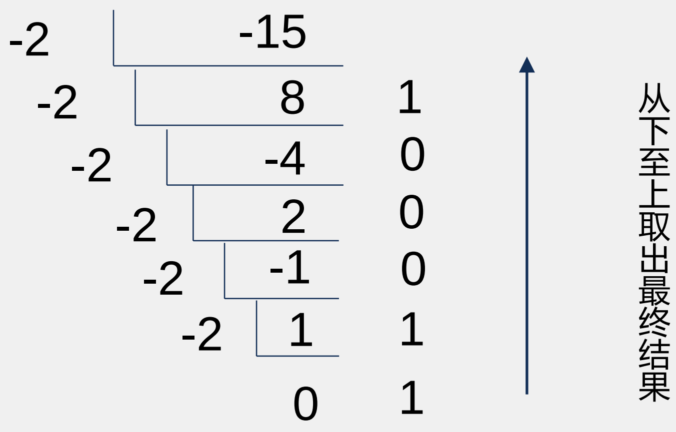
十进制转负K进制-短除法

理解了负数除法后，我们可以继续进行。

我们都知道十进制转正数N进制，只要短除法即可。但是对于负进制，我们该如何处理呢?

答案其实很简答!

关键思想：保证余数为正。计算过程如下：



所以，-15(10) = 11001(-2)

十进制转负K进制-余数转正

短除法的精要在于余数为正。那么，如何才能保证余数为正呢？我们来看一下下面的例子：

7 = (-3) * (-3) + (-2)

7 = (-2) * (-3) + 1

-7 = 2 * (-3) + (-1)

-7 = 3 * (-3) + 2

上面这些式子都是以(-3)作为除数，亦可以看为被转换的进制基数，即-3进制。

那么，我们怎么把余数转换成正数呢？

通过观察，我们可以发现，负余数减去基数，同时商再加1，就完成了变换。

代码为：

```
void work(int num,int base){
     if (num==0)return;
     int rest = num % base;   //余数
     num /= base;
     if(rest < 0){
          rest -= base;        //减去基数
          num += 1;            //商加1
     }
     work(num,base);   //回归了，我们取余除K
     realans[cnt++]=rest;
}
```

十进制转负K进制-模板代码

逻辑航线

```cpp
#include <iostream>
#include <cstdio>

using namespace std;
int realans[100];
int cnt = 0;

void work(int num,int base){
	if (num==0)return;
	int rest = num % base;
	num /= base;
	if(rest < 0){
		rest -= base;
		num += 1;
	}
	work(num,base);
	realans[cnt++]=rest;
}
```

```cpp
int main(){
	int num,base;
	cin>>num>>base;
	work(num,base);
	for(int i=0; i<cnt; i++){
		cout << realans[i];
	}
	return 0;
}
```

10

快速幂

快速幂-方法一

逻辑航线

快速幂以及扩展的矩阵快速幂，由于应用行经比较常见，也是竞赛中常见的题型。

幂运算

A^n 即n个A相乘。快速幂就是指高效的算出A^n 。当n很大时，例如n= 10^9 时，计算A^n 这样的大数会得出一个巨大的数字，同时也会消耗非常多的计算时间。

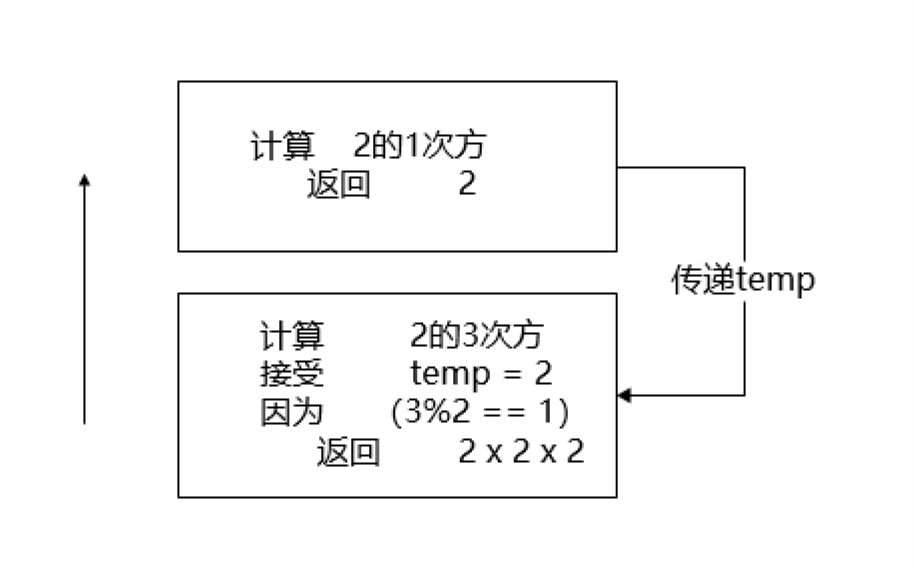
由此，聪明的程序员想出了一种通过递归的方式来实现快速幂。

快速幂模板一：

```
int fastPow(int a,int n){
    if(n==0){ //特殊处理0的情况
        return 1;
    }
    if(n==1){ //1次方，返回其本身
        return a;
    }
    int temp = fastPow(a,n/2);
    if(n%2 == 1){ //处理奇数次
        return temp * temp * a;
    }
    else { //处理偶数次
        return temp * temp;
    }
}
```

示例：

计算2的三次方，堆栈调用过程



快速幂-方法二

前面的分治算法，已经是一种优秀的算法了。但是，在这里我们还有一种更好的算法。

举例：

我们现在要来计算A^{11}，该怎么做呢？我们将面对以下2个问题。

1、如何能够减少计算次数呢？

答案是把 A^{11}拆分成A^8 A^2 A^1的乘积。那么，我们需要分别去求A^8、A^2、A^1的数值吗？答案是否定的，我们可以观察出$A^2 = A^1 \times A^1$、$A^4 = A^2 \times A^2$、$A^8 = A^4 \times A^4$，所以，我们可以使用模板方法Base *= Base来实现。

2、如何把A^{11}拆分成A^8 A^2 A^1这样的形式呢？

答案是使用二进制的形式。因为 11(10) = 1011(2)，同时，还能够通过判断二进制位上的数字是否为0，来跳过不需要的数字。

我们可以使用与计算来实现这步操作。

例如：1011 & 1 的结果是大于0的，这意味着A^1是存在的。然后，我们可以通过右移数字，依次判断其他数位是否存在。

快速幂-模板代码二

```
int fastPow(int a,int n)  //a为底数，n为指数
//              2     3
{
     int base = a;
     int res = 1;
     while(n)
     {
          if(n & 1)
          {
                   res *= base;
          }
          base *= base;
          n >> 1;
     }
}
```

11

专项练习

专项练习1-最简真分数

逻辑航线

题目地址： https://nanti.jisuanke.com/t/T1923

解题思路： 枚举分子与分母。
须满足条件——1、分子小于分母；2、最大公因数为1

专项练习1-最简真分数AC代码

逻辑航线

```
#include <iostream>
#include <algorithm>
using namespace std;

int son[1000000],mother[1000000];
long long cnt=0;

int main(int argc, char** argv)
{
	int n;
	cin >> n;
	for(int i=1; i<=n; i++){ //枚举分子
		for(int j=i+1; j<=n; j++){ //枚举分母
			if(__gcd(j,i) == 1){
				son[cnt] = i;
				mother[cnt]=j;
				cnt+=1;
			}
		}
	}


	for(int i=0; i<cnt; i++){
		printf("%d/%d\n",son[i],mother[i]);
	}
	return 0;
}
```

专项练习2-高精度计算

题目地址： https://www.luogu.com.cn/problem/P1601

解题思路： 加法高精度

题目地址： https://www.luogu.com.cn/problem/P1303

解题思路： 乘法高精度

专项练习3-阶乘数码

题目地址： https://www.luogu.com.cn/problem/P1591

解题思路： 单精度乘法

专项练习3-阶乘数码AC代码

逻辑航线

```
#include <bits/stdc++.h>
using namespace std;
int c[100000];
int main(){
	int t,n,a;
	cin>>t;
	for(int i=0; i<t; i++){
		cin>>n>>a;
		memset(c,0,sizeof(c));
		c[0]=1;
		int l=1;
		for(int j=2; j<=n; j++){//开始阶乘计算
			int w=0;
			for(int k=0; k<l; k++){//高精度乘单精度
				c[k]=c[k]*j+w;
				w=c[k]/10;
				c[k]%=10;
			}
			while(w>0){//处理多进位
				c[l]=w%10;
				l++;
				w/=10;
			}
		}
		int sum=0;
		for(int j=0; j<l; j++)
				if(c[j]==a) sum++;//统计个数
		cout<<sum<<endl;
	}
	return 0;
}
```

专项练习4-进制转换

题目地址： https://www.luogu.com.cn/problem/P1143

解题思路： 注意超过10进制以上的数字需要用字母代替

专项练习4-进制转换AC代码

```cpp
#include <stdio.h>
#include <string.h>
#include <iostream>
using namespace std;
int n,m,top;
char stack[33],a[33];//a存储读入数据, stack存储转换后的数据
int c2i(char c){ //字符转数字
	if('0'<=c&&c<='9')return c-'0';
	if('A'<=c&&c<='F')return c-'A'+10;
}
int s2i(char *s){ //字符串转数字
	int i,len,ret=0;
	char c;
	len=strlen(s);
	for(i=0; i<len; i++){
		ret*=n;  //把任意进制转化成十进制
		ret+=c2i(s[i]);
	}
	return ret;
}
char i2c(int x){ //数字转字符
	if(0<=x&&x<=9)return '0'+x;
	if(10<=x&&x<=15)return x-10+'A';
}
void i2i(int b){ //数字转数字
	top=0;
	while(b){
		stack[++top]=i2c(b%m);
		b/=m;
	}
}
int main(){
	scanf("%d%s%d",&n,a,&m);
	int ret = s2i(a); //转化成原数字在10进制下的数字
	i2i(ret); //转化成对应的进制
	while(top)
		printf("%c",stack[top--]);//打印
	return 0;
}
```

专项练习5-负数的进制转换

题目地址： https://www.luogu.com.cn/problem/P1017

解题思路： 十进制转负进制

专项练习5-负数的进制转换AC代码

```cpp
#include <iostream>
#include <cstdio>
#include <stack>
#include <string>


using namespace std;
string str="0123456789ABCDEFG";
string realans="";
//注意我使用了传址调用，直接修改num的值
char getachar(int key,int &num,const int &base)
{
     if (key<0)
          key-=base,num+=base;
     return str[key];//返回应该有的值
}

void work(int num,int base)
{
     if (num==0)return;
     char ch=getachar(num%base,num,base);
     //先记录一下当前的字符
     //然后再直接下一层调用
     work(num/base,base);
     //短除法的精髓所在
     realans=realans+ch;
}
```

```cpp
int main()
{
     int num,base;
     cin>>num>>base;
     work(num,base);
     cout<<num<<'='<<realans<<"(base"<<base<<')'<<endl;
     return 0;
}
```

专项练习6-拍头游戏

题目地址： https://www.luogu.com.cn/problem/P2926

解题思路： 简单的倍数统计

专项练习6-拍头游戏AC代码

逻辑航线

```
#include<cstdio>
#include<iostream>
#include<algorithm>
using namespace std;
int n,maxn;
int a[100010];
int ma[1000010],ans[1000100];
int main(){
	scanf("%d",&n);
	for(int i=1; i<=n; i++){
		scanf("%d",&a[i]); //记录每一个奶牛的号码
		ma[a[i]]++; //记录奶牛手中数字出现的次数
		maxn=max(a[i],maxn); //记录奶牛手中最大的数字，减少遍历次数
	}
	for(int i=1; i<=maxn; i++){
		if(!ma[i]) //没有奶牛持有这个数字
				continue;
		for(int j=i; j<=maxn; j+=i){
				ans[j]+=ma[i]; //所有i的倍数都进行一次标记
		}
	}
	for(int i=1; i<=n; i++)
		cout<<ans[a[i]]-1<<endl; //减掉自己的次数
}
```

专项练习7-线性筛素数

题目地址： https://www.luogu.com.cn/problem/P3383

解题思路： 埃式筛，欧拉筛

专项练习8-素数密度（经典例题）

逻辑航线

题目地址： https://www.luogu.com.cn/problem/P1835

解题思路：

1、使用埃式筛或欧拉筛计算2147483647平方根以内，即<=50000的质数，约6000个。
2、利用这些质数标记L-R以内的所有合数。
3、统计所有未被标记的数。

专项练习8-素数密度（AC代码）

```cpp
#include <bits/stdc++.h>
using namespace std;
#define re register
#define ll long long
const int maxn=1e6+1;
ll l,r;
int prime[maxn],cnt,ans;
bool vis[maxn];
inline void Gprime(){//线性筛素数,预处理根号R内的素数
	for(re int i=2; i<=50000; ++i){
		if(!vis[i])
			prime[++cnt]=i;
		for(re int j=1; i*prime[j]<=50000; j++){
			vis[i*prime[j]]=1;//标记合数
			if(i%prime[j]==0) break;
		}
	}
}//50000的范围很小即使不用线性筛用根号N的筛法也能过
```

```cpp
int main(){
	cin>>l>>r;
	l=l==1?2:l;//特判L=1的情况
	Gprime();//筛出根号R内的所有质数以及剩下的合数
	memset(vis,0,sizeof(vis));//懒得多开一个数组,沿用函数中的数组时记得清空
	for(int i=1; i<=cnt; i++){
		ll p=prime[i];
		for(int j=l/p; j<=r/p; j++){ //取出两边的边界
			if(j>1){  //将L=素数本身的情况排除出去
				int current = p*j;
				int index = current -l+1;
				if(index>0){ //index<0则表示小于L
					vis[index]=1;
				}
			}
		}
	}
	for(int i=1; i<=r-l+1; ++i){
		if(!vis[i]){
				ans++;//r-l+1即为区间长度,
				//遍历区间找没有被标记的元素并累加答案
		}
	}
	cout<<ans;
}
```

专项练习9-最大公约数和最小公倍数问题

题目地址： https://www.luogu.com.cn/problem/P1029

解题思路：

1、根据最大公约数和最小公倍数的性质4：两个数(P,Q)的最小公倍数y与最大公约数x的积即为两个数的乘积，我们可以得出PQ = xy
2、从1可以得出，P必然是y的约数，所以，我们可以通过枚举y的所有约数k，求出P,Q的组合。
3、进一步分析，我们可以得出P存在两种可能性
当P = k时，　Q = xy / k
当P = y/k时，Q = x k

小提示：

1、枚举因数时，注意枚举的上限为 k * k <= y，避免重复枚举
2、当k==y/k的时候，二者是同一种情况，所以，请注意排重。

专项练习9-最大公约数和最小公倍数问题（AC代码）

逻辑航线

```
#include <iostream>
#include <algorithm>

using namespace std;

int x,y,P,Q,cnt;

int main(int argc, char** argv){
	scanf("%d%d",&x,&y);
	for(int k=1; k *k<=y; k++){
		if(y%k==0){
			if(__gcd(k,y/k*x)==x){
				cnt++;
			}
			if(y/k != k){
				if(__gcd(y/k,k*x)==x){
					cnt++;
				}
			}
		}
	}
	cout << cnt;
	return 0;
}
```

专项练习10- Hankson 的趣味题

题目地址： https://www.luogu.com.cn/problem/P1072

解题思路：

同例9，枚举最小公倍数的约数后，判断符合条件的数据

专项练习10- Hankson 的趣味题（AC代码）

逻辑航线

```
#include <iostream>
#include <algorithm>

using namespace std;

int lcm(int x,int y){
        return x / __gcd(x,y) * y;
}
int T,a0,a1,b0,b1;
int main(int argc, char** argv){
     scanf("%d",&T);
     for(; T--;){
          int cnt = 0;
          scanf("%d%d%d%d",&a0,&a1,&b0,&b1);
          for(int x = 1; x<=b1 / x; x++){
               if(b1%x == 0){
                    if(__gcd(x,a0)==a1 && lcm(x,b0)==b1){
                              cnt++;
                    }
                    if(b1 / x != x){
                         if(__gcd(b1/x,a0) == a1 && lcm(b1/x,b0)==b1){
                              cnt++;
                         }
                    }
               }
          }
          cout << cnt << endl;
     }
     return 0;
}
```

专项练习11-计算分数

题目地址： https://www.luogu.com.cn/problem/P1572

解题思路：

1、scanf的带符号读入用法

2、用加法替代减法

专项练习11-计算分数（AC代码）

逻辑航线

```
#include<cstdio>
#include<iostream>

using namespace std;
int a,b,c,d;
int main(){
	scanf("%d/%d",&a,&b);
	//注意，此处读入的c是带有符号的
	while(scanf("%d/%d",&c,&d)!=EOF){
		//通分并计算
		int m=__gcd(b,d);
		a*=d/m;
		c*=b/m;
		a+=c;
		b*=d/m;
		//约分结果
		m=__gcd(a,b);
		a/=m;
		b/=m;
	}
	//当前计算值为负值，同时变号
	if(b<0){
		a=-a;
		b=-b;
	}
	if(b==1)
		printf("%d\n",a);
	else
		printf("%d/%d\n",a,b);
	return 0;
}
```

专项练习12-晨跑

题目地址： https://www.luogu.com.cn/problem/P4057

解题思路：

1、多元最小公倍数
2、数值范围

专项练习12-晨跑（AC代码）

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

long long lcm(long long a,long long b){
          return a*b / __gcd(a,b);
}

int main(int argc, char** argv){
     long long a,b,c;
     cin >> a >> b >>c;
     long long value = lcm(a,b);
     value = lcm(value,c);
     cout << value;
     return 0;
}
```

感　谢　观　看

联系地址：河北省廊坊市大厂县孔雀英国宫二期