学习情况表

|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | 王一博 | **学号** | 2021902655 |
| **学院** | 信息工程学院 | **专业** | 计算机科学与技术 |

（请在下面表格本周学习情况）

|  |
| --- |
| 学习情况简述 |
| 根据网站ACWing和书籍《大话数据结构》进行学习，学习内容包括：质数，约数，欧拉函数，快速幂，扩展欧几里得算法，高斯消元法等 |
| 本周练习过的代码（例） |
| （本周练习写过的代码，如实填写，疑似抄袭会单独测试）  *代码1：*  *#include<stdio.h>*  *int main()*  *{*  *printf("HelloWorld\n");*  *return 0;*  *}*  *代码2：*  …….. |
| 本周练习过的代码 |

代码1：

#include <iostream>

#include <algorithm>

using namespace std;

bool prime(int x)

{

if (x <= 1) return false;

for(int i = 2; i <= x / i; i ++)

if(x % i == 0) return false;

return true;

}

int main()

{

int n;

scanf("%d", &n);

while (n -- )

{

int a;

scanf("%d", &a);

if(prime(a)) puts("Yes");

else puts("No");

}

}

代码2：

#include <iostream>

using namespace std;

void divide(int x)

{

for(int i = 2; i <= x / i; i ++)

{

if (x % i == 0) //当i是x的因数时，求i的次数

{

int s = 0;

while(x % i == 0)

{

x /= i;

s ++;

}

cout << i << " " << s << endl;

}

}

//一个数的最小质因数（除1以外）一定是质数

//若经上述过程后没有整除完，则输出最后一个因数，且一定是1次的

if(x >= 2) cout << x << " " << 1 << endl;

cout << endl;

}

int main()

{

int n;

scanf("%d", &n);

while (n -- )

{

int x;

scanf("%d", &x);

divide(x);

}

}

代码3：

###方法一：朴素筛法：

#include <iostream>

using namespace std;

const int N = 1e6 + 10;

int primes[N], cnt; //primes存具体的质数的值

bool st[N]; //记录是否是合数（true则是，false则不是）

void find\_primes(int n)

{

for(int i = 2; i <= n; i ++)

{

if(!st[i]) primes[cnt ++] = i; //若i是质数，则放入primrs[]数组中

for(int j = 2 \* i; j <= n; j += i) //当i是质数时，排除1~n中所有i的倍数（合数）

st[j] = true;

}

}

int main()

{

int n;

cin >> n;

find\_primes(n);

cout << cnt;

}

###方法二：线性筛法

#include <iostream>

using namespace std;

const int N = 1e6 + 10;

int cnt, primes[N];

bool st[N];

void find\_primes(int n)

{

for(int i = 2; i <= n; i ++)

{

if(!st[i]) primes[cnt ++] = i;

for(int j = 0; primes[j] <= n / i; j ++) //从小到大枚举现存的所有质数

{

st[primes[j] \* i] = true;

if(i % primes[j] == 0) break; //primes[j]一定是i的最小质因子

}

}

}

int main()

{

int n;

cin >> n;

find\_primes(n);

cout << cnt;

}

代码4：

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

vector<int> get\_divisors(int n)

{

vector<int> res;

for(int i = 1; i <= n / i; i ++)

{

if(n % i == 0)

{

res.push\_back(i);

if(n != i \* i) res.push\_back(n / i);

}

}

sort(res.begin(), res.end());

return res;

}

int main()

{

int n;

cin >> n;

while (n -- )

{

int x;

cin >> x;

auto res = get\_divisors(x);

for(int i = 0; i < res.size(); i ++) cout << res[i] << ' '; //或写成： for(auto t : res) cout << t << ' ';

cout << endl;

}

}

代码5：

#include <iostream>

#include <cstring>

#include <algorithm>

#include <unordered\_map>

using namespace std;

const int MOD = 1e9 + 7;

unordered\_map<int, int> primes; //map中存 质因子-指数 键值对

void divisors\_num(int x)

{

for(int i = 2; i <= x / i; i ++)

{

while(x % i == 0) //当某一个数i为x的因数时

{

x /= i;

primes[i] ++; //确定i作为因数时的指数

}

}

if(x >= 2) primes[x] ++; //最后剩下一个小质数时，将其放入map中

}

int main()

{

int n;

cin >> n;

while (n -- )

{

int x;

cin >> x;

divisors\_num(x); //先单算每个数的质因数都有哪些，通过循环得出每个质因数的指数和

}

long long res = 1; //存最后的答案，约数的个数

for(auto p : primes) res = res \* (p.second + 1) % MOD; //先取模可以保证中间结果不会溢出

cout << res ;

}

代码6：

#include <iostream>

#include <cstring>

#include <algorithm>

#include <unordered\_map>

using namespace std;

typedef long long LL;

const int MOD = 1e9 + 7;

unordered\_map <int, int> primes;

void divisors(int x)

{

for(int i = 2; i <= x / i; i ++)

{

while(x % i == 0)

{

x /= i;

primes[i] ++;

}

}

if(x >= 2) primes[x] ++;

}

int main()

{

int n;

cin >> n;

while (n -- )

{

int x;

cin >> x;

divisors(x);

}

LL res = 1;

for(auto p : primes)

{

LL a = p.first, b = p.second;

LL t = 1; // t存公式中的各个因式

while(b --)

t = (t \* a + 1) % MOD;

res = res \* t % MOD;

}

cout << res;

}

代码7：

#include <iostream>

using namespace std;

int gcd(int a, int b)

{

return b ? gcd(b, a % b) : a;

}

int main()

{

int n;

cin >> n;

while (n -- )

{

int a, b;

cin >> a >> b;

cout << gcd(a, b) << endl;

}

}

代码8：

#include <iostream>

using namespace std;

//求欧拉函数的公式中无关质因数的指数

int phi(int x)

{

int res = x;

for(int i = 2; i <= x / i; i ++)

{

if(x % i == 0)

{

res = res / i \* (i - 1);

while(x % i == 0) x /= i; //直至x中无i这个质因子

}

}

if(x >= 2) res = res / x \* (x - 1);

return res;

}

int main()

{

int n;

scanf("%d", &n);

while (n -- )

{

int x;

cin >> x;

cout << phi(x) << endl;

}

}

代码9：

#include <iostream>

using namespace std;

typedef long long LL;

const int N = 1e6 + 10;

int primes[N], cnt; //primes[]存质数的值

int euler[N]; //ehler[i]存i的欧拉函数值

bool st[N]; //st[i]存数i是否是质数

void get\_eulers(int n)

{

euler[1] = 1; // 1的欧拉函数值为1

for(int i = 2; i <= n; i ++) //线性筛i

{

if(!st[i])

{

primes[cnt ++] = i;

euler[i] = i - 1; //当i是质数时，1 ~ i-1全是与i互质的数（共 i-1 个）

}

for(int j = 0; primes[j] <= n / i; j ++)

{

int t = primes[j] \* i;

st[t] = true;

if(i % primes[j] == 0)

{

euler[t] = euler[i] \* primes[j];

break; //保证了每个数只被其最小的质因数筛一次

}

euler[t] = euler[i] \* (primes[j] - 1);

}

}

}

int main()

{

int n;

cin >> n;

get\_eulers(n);

LL res = 0;

for (int i = 1; i <= n; i ++ ) res += euler[i];

cout << res;

}

代码10：

#include <iostream>

using namespace std;

typedef long long LL;

LL fast\_exp(int a, int b, int p)

{

int res = 1;

while(b)

{

if (b & 1) res = (LL)res \* a % p; //当b的二进制表示的个位为1时

//a的最大值为2e9，a\*a会爆int

a = (LL)a \* a % p;

b >>= 1;

}

return res;

}

int main()

{

int n;

scanf("%d", &n);

while (n -- )

{

int a, b, p;

scanf("%d%d%d", &a, &b, &p);

cout << fast\_exp(a, b, p) << endl;

}

}

代码11：

#include <iostream>

using namespace std;

typedef long long LL;

LL fast\_exp(int a, int b, int p)

{

LL res = 1;

while(b)

{

if(b & 1) res = (LL)res \* a % p;

a = (LL)a \* a % p;

b >>= 1;

}

return res;

}

int main()

{

int n;

scanf("%d", &n);

while (n -- )

{

int a, p;

scanf("%d%d", &a, &p);

if(a % p == 0) puts("impossible"); //当p是a的倍数时，不存在逆元

else cout << fast\_exp(a, p - 2, p) << endl; //求a模p的乘法逆元（费马定理）

}

}

代码12：

#include <iostream>

using namespace std;

int exgcd(int a, int b, int &x, int &y)

{

if(b == 0)

{

x = 1, y = 0;

return a;

}

int d = exgcd(b, a % b, y, x);

y -= a / b \* x;

return d;

}

int main()

{

int n;

scanf("%d", &n);

while (n -- )

{

int a, b, x, y;

scanf("%d%d", &a, &b);

exgcd(a, b, x, y);

printf("%d %d \n", x, y);

}

}

代码13：

#include <iostream>

using namespace std;

int exgcd(int a, int b, int &x, int &y)

{

if(b == 0)

{

x = 1, y = 0;

return a;

}

int d = exgcd(b, a % b, y, x);

y -= a / b \* x;

return d;

}

int main()

{

int n;

scanf("%d", &n);

while (n -- )

{

int a, b, m;

scanf("%d%d%d", &a, &b, &m);

int x, y;

int d = exgcd(a, m, x, y);

if(b % d != 0) puts("impossible");

else printf("%d \n", (long long)b / d \* x % m);

}

}

代码14：

#include <iostream>

#include <cmath>

using namespace std;

const int N = 105;

const double eps = 1e-8; //极小的一个数

int n;

double a[N][N];

int gauss() //若有唯一解，则答案存于最后一列（a[i][n]）中

{

int c, r;

for(c = 0, r = 0; c < n; c ++) //遍历每一列

{

int t = r;

for(int i = r; i < n; i ++) //找该列中的绝对值最大数所在的行

{

if (abs(a[i][c]) > abs(a[t][c]) )

t = i;

}

if (abs(a[t][c]) < eps) continue; //若当前列的所有数都为0，直接进行下一列的遍历

for(int i = c; i <= n; i ++ ) swap(a[r][i], a[t][i]); //将该行换到最上面（第r行） (此次循环t的使命结束)

for(int i = n; i >= c; i -- ) a[r][i] /= a[r][c]; //该行所有数都要把第一个数(a[r][c])作为除数，因此最后更改第一个数的值

for(int i = r + 1; i < n; i ++ ) //将该列中下面的数消为0（所在行的数相应变化）

{

if(abs(a[i][c]) > eps)

for(int j = n; j >= c; j -- )

a[i][j] -= a[r][j] \* a[i][c];

}

r ++;

}

if (r < n) //若是不完美阶梯形，即存在没有未知数的行

{

for(int i = r; i < n; i ++ )

if (abs(a[i][n]) > eps)

return 2; // 0 = 0 型，无穷多组解

return 1; // 0 = 非零 型，无解

}

for (int i = n - 1; i >= 0; i -- )

for (int j = i + 1; j < n; j ++ )

a[i][n] -= a[j][n] \* a[i][j];

return 0; //有唯一解

}

int main()

{

scanf("%d", &n);

for (int i = 0; i < n; i ++ )

for (int j = 0; j <= n; j ++ )

scanf("%lf", &a[i][j]);

int t = gauss();

if (t == 1) puts("Infinite group solutions");

else if (t == 2) puts("No solution");

else if(t == 0)

{

for (int i = 0; i < n; i ++ )

{

if (abs(a[i][n]) < eps) a[i][n] = 0; //避免输出0.00

printf("%.2lf \n", a[i][n]);

}

}

}