# 第k次编程练习报告

姓名：周钰宸 学号：2111408 班级：信安一班

##### **编程练习1——编程实现求解最小原根并基于最小原根构造指数表**

* **源码部分：**

#include<iostream>

#include<cmath>

#include<stack>

#include<vector>

#include<iomanip>

using namespace std;

stack<int> DecimalToBinary(unsigned long long n) {//计算十进制转二进制，用一个栈s来保存，用来辅助使用平方乘算法

stack<int> s;

while (n != 0) {

int r = n % 2;

s.push(r);

n = n / 2;

}

return s;

}

unsigned long long SquareAndMultiplyAlgorithm(unsigned long long a, unsigned long long n, unsigned long long m) {//平方乘算法，用来辅助求解最小原根与生成指数表

unsigned long long result = 1;//result定义为64位的整型,防止中间结果超过32位而发生溢出

stack<int> s = DecimalToBinary(n);

while (!s.empty()) {//由于算法的特性，先要处理的是二进制的头位，即在二进制转为十进制方法中最后得到的余数

//符合FILO的特性，故采用栈的形式存储并操作

result = (result \* result) % m;

if (s.top() == 1) {

result = (result \* a) % m;

}

s.pop();

}

return result;

}

bool if\_vegetable(int n) {//判断是否为素数,辅助使用计算欧拉函数

if (n <= 1)

return false;

else {

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

if (n % i == 0) {

return false;

}

}

return true;

}

}

int gcd(int a, int b) {//采用辗转相除法,计算最大公因数，用来辅助判断两个数是否互素

int max, min;

if (a > b)

max = a, min = b;

else

max = b, min = a;

while (max % min != 0) {//只要余数不为0，就继续

int r = max % min;

max = min;

min = r;

}

return min;//此时的min（除数）是上一步的余数r，即最后一个不为0的余数，即最大公因数

}

bool if\_Coprime(int a, int b) {//判断两个数是否互素，用来辅助求出n的欧拉函数，即n的缩系大小

return gcd(a, b) == 1;

}

int EulerTotientFunction(int n) {//计算欧拉函数，用来辅助求解最小原根

int CoPrimeCount = 0;

for (int i = 1; i < n; i++) {//判断从1-n-1中与n互素的数，其个数即为n的缩系大小

if (if\_Coprime(i, n)) {

CoPrimeCount++;

}

}

return CoPrimeCount;

}

void ReducedResidueSystem(int n, vector<int>& CoPrimes) {//用来求出n的所有缩系的元素，用来辅助求解最小原根

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

if (if\_Coprime(i, n)) {//判断从1-n-1中与n互素的数，若其与之互素，将其压入vector

CoPrimes.push\_back(i);

}

}

}

void FundamentalTheoremOfArithmetic(int n,vector<int>& PrimeFactor) {//算术基本定理，用来辅助求解最小原根

int\* factor = new int[n];//存放因数

int NumOfFactor = 0;

int\* exponent = new int[n];//存放指数

int i = 2;//从最小的素数2开始找起

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

factor[i] = exponent[i] = -1;

}

if (if\_vegetable(n)) {//若n本身就为素数，则其唯一一个素因子为其自己

factor[NumOfFactor] = n;

NumOfFactor++;

for (int k = 0; k < NumOfFactor; k++) {//将之前保存的素因子放入PrimeFactor中

PrimeFactor.push\_back(factor[k]);

}

return;

}

int temp = n;

while (!if\_vegetable(temp)&&temp!=1) {//只要此时的n（temp）没有被除的只剩下一个素数，就继续除

if (temp % i == 0) {//若能被这个素数整除，则该素数为其一个素数幂

factor[NumOfFactor] = i;//保存该素数幂

int current\_exponent = 1;//指数为1

temp = temp / i;

while (temp % i == 0) {//只要还能被i整除，就继续循环，增加指数

current\_exponent++;

temp = temp / i;

}

exponent[NumOfFactor] = current\_exponent;//保存指数

NumOfFactor++;

}

for (int j = i + 1;; j++) {//若i不能整除n，则证明i不是其素数幂，去寻找大于i的最小素数，重复此过程

if (if\_vegetable(j))

{

i = j;

break;

}

}

}

//接下来处理一种特殊情况，比如若n=8，则其temp=2时就已经退出循环，但是此时2也为其素因子，需要记录；

bool flag = false;//是否最后剩下的temp是之前出现过的素因子

for (int t = 0; t < NumOfFactor; t++) {

if (factor[t] == temp) {//若之前出现过

exponent[t]++;

flag = true;

break;

}

}

if (flag == false&&temp!=1) {//若之前所剩下的temp是未出现过的素因子，则将其保存

factor[NumOfFactor] = temp;

exponent[NumOfFactor] = 1;

NumOfFactor++;

}

for (int k = 0; k < NumOfFactor; k++) {//将之前保存的素因子放入PrimeFactor中

PrimeFactor.push\_back(factor[k]);

}

return;

}

int TheMinPrimitiveRoot(int n) {//求解最小原根

int ETF\_n = EulerTotientFunction(n);//获取n的欧拉函ETF\_n

vector<int> PrimeFactor;//寻找并记录所有ETF\_n的素因子

FundamentalTheoremOfArithmetic(ETF\_n, PrimeFactor);

vector<int> RequiredExponent;//用来记录所有用于判断的指数

for (int i = 0; i < PrimeFactor.size(); i++) {

RequiredExponent.push\_back(ETF\_n/PrimeFactor[i]);

}

vector<int> ReducedResidueSystem\_n;//用来记录n的缩系

ReducedResidueSystem(n, ReducedResidueSystem\_n);

for (int i = 1; i < ETF\_n; i++) {//从除去1的第二个元素开始依次判断n的缩系中的元素是否符合条件

bool flag = true;//用来记录是否判断成功

for (int j = 0; j < RequiredExponent.size(); j++) {//对所有要求的指数都检查一遍

if (SquareAndMultiplyAlgorithm(ReducedResidueSystem\_n[i], RequiredExponent[j], n) == 1) {

flag = false;//失败

break;

}

}

if (flag)//若成功，则此时第i个即为最小原根

return ReducedResidueSystem\_n[i];

}

return -1;

}

int DiscreteLogarithm(int g, int a, int n) {//计算离散对数，用来辅助构造指数表

int ETF\_n = EulerTotientFunction(n);//因为g的0-ETF\_n-1次方构成模n的一个缩系

for (int i = 0; i < ETF\_n; i++) {

if (SquareAndMultiplyAlgorithm(g, i, n) == a) {

return i;

}

}

}

void CreateInd\_Table(int n, int g) {//构造指数表

int ETF\_n = EulerTotientFunction(n);

vector<int> ReducedResidueSystem\_n;//用来记录n的缩系

ReducedResidueSystem(n, ReducedResidueSystem\_n);

cout << setw(5) << " ";

cout.fill(' ');//数据宽度不够时，自动填充空格，fill成员函数可以设置填充的字符

for (int i = 0; i < 10; i++) {//先输出指数表的第一行，即个位数字

cout << setw(5) << i;

cout.fill(' ');//数据宽度不够时，自动填充空格，fill成员函数可以设置填充的字符

}

cout << endl;

int currentNum = 0;//记录现在遍历到n的缩系的第几位了

for (int i = 0; i <=n/10; i++) {//外层循环即每行，为指数表的十位，用n/10向下取整正好得到外层的行数

cout << setw(5) << i;//输出十位数字先

for (int j = 0; j < 10; j++) {//内层循环即每列，为指数表的个位

if (i == 0 && j == 0) {//00的位置永远不输出

cout << setw(5) << "-";

cout.fill(' ');//数据宽度不够时，自动填充空格，fill成员函数可以设置填充的字符

continue;

}

int a = 10 \* i + j;//计算对应的indga的a

if (a<n) {//若此时的a比n小，正常输出

if (ReducedResidueSystem\_n[currentNum] == a) {//因为储存n的所有素因子的n的缩系也是从小到大，这一点与a相同

//故先判断a是不是n的缩系中的元素，若是，调用离散对数函数直接计算

cout << setw(5) << DiscreteLogarithm(g, a, n);

cout.fill(' ');//数据宽度不够时，自动填充空格，fill成员函数可以设置填充的字符

currentNum++;

continue;

}

else {//若a不为n的缩系中的元素，直接输出

cout << setw(5) << "-";

cout.fill(' ');//数据宽度不够时，自动填充空格，fill成员函数可以设置填充的字符

continue;

}

}

else {//若此时指数比n大，直接输出“-”

cout << setw(5) << "-";

cout.fill(' ');//数据宽度不够时，自动填充空格，fill成员函数可以设置填充的字符

continue;

}

}

cout << endl;

}

return;

}

int main() {

int n;

cout << "Please input n(n>0): ";

cin >> n;

int g = TheMinPrimitiveRoot(n);

if (g == -1) {

cout <<n<<" has no primitive root!" << endl;

return 0;

}

cout << "The min primitive root of " << n << ": g=" << g << endl;

cout << "The ind\_table of " << n << " based on g=" << g << " is:" << endl;

CreateInd\_Table(n, g);

return 0;

}

* **说明部分：**

1. **预备函数：（为了求解最小原根以及指数表的构建，直接使用或稍加修改前几次编程作业完成的函数）**

（1）stack<int> DecimalToBinary(unsigned long long n) ——计算十进制转二进制，用一个栈s来保存，用来辅助使用平方乘算法

（2）unsigned long long SquareAndMultiplyAlgorithm(unsigned long long a, unsigned long long n, unsigned long long m) ——平方乘算法，用来辅助求解最小原根与生成指数表

（3）bool if\_vegetable(int n)——判断是否为素数,辅助使用计算欧拉函数,以及用来各种地方

（4）int gcd(int a, int b)——采用辗转相除法,计算最大公因数，用来辅助判断两个数是否互素

（5）bool if\_Coprime(int a, int b)——判断两个数是否互素，用来辅助求出n的欧拉函数，即n的缩系大小

（6）int EulerTotientFunction(int n)——计算欧拉函数，用来辅助求解最小原根

（7）void ReducedResidueSystem(int n, vector<int>& CoPrimes)——用来求出n的所有缩系的元素，用来辅助求解最小原根

（8）void FundamentalTheoremOfArithmetic(int n,vector<int>& PrimeFactor)——算术基本定理，用来辅助求解最小原根

（9）int DiscreteLogarithm(int g, int a, int n)——计算离散对数，用来辅助构造指数表

**2.求解最小原根**int TheMinPrimitiveRoot(int n)

（1）先利用1.（6）计算n的欧拉函数，保存为**ETF\_n**。

（2）利用1.（8）求出的所有素因子，保存在**vector类型变量PrimeFactor**中。

（3）利用ETF\_n和PrimeFactor求出所有需要使用定理判断的指数，保存在**vector类型变量RequiredExponent**中。

（4）利用1.（7）求出n的所有缩系元素，保存在**vector类型变量ReducedResidueSystem\_n**中。

（5）从ReducedResidueSystem\_n[1]即n的**第二个缩系元素（除去1）**开始，并利用RequiredExponent和平方乘算法1.（2）依次判断是否满足定理所需要的高次模运算不等于1的条件。**若已经有一个运算成功，则输出该n的缩系元素，即n的最小原根**；若都未成功，直接跳出循环还没运行函数结束，**函数返回-1，标志未找到最小原根，即不存在原根，**此时在主函数中输出相应提示后结束程序。

**3.构造指数表**void CreateInd\_Table(int n, int g)

（1）同样先利用1.（6）计算n的欧拉函数，保存为**ETF\_n**；再利用1.（7）求出n的所有缩系元素，保存在**vector类型变量ReducedResidueSystem\_n**中。

（2）为了指数表格式的整齐，使用**iomanip**头文件中的**setw(5)**和**cout.fill(' ')**默认右对齐格式输出，并且用空格填满。

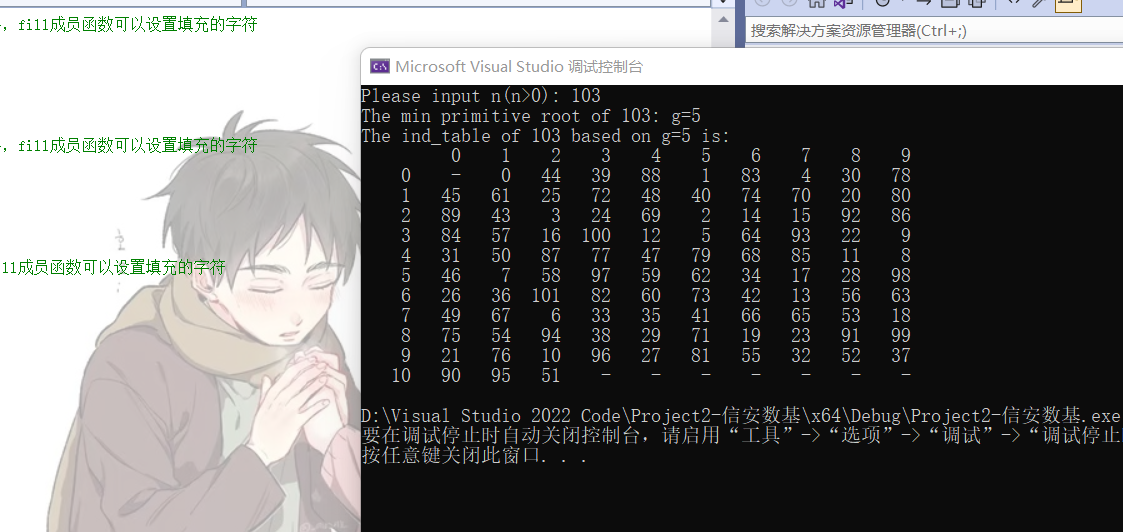
（3）外层循环即每行，为指数表的十位，用n/10向下取整正好得到外层的行数；内层循环即每列，为指数表的个位（最大为9）。

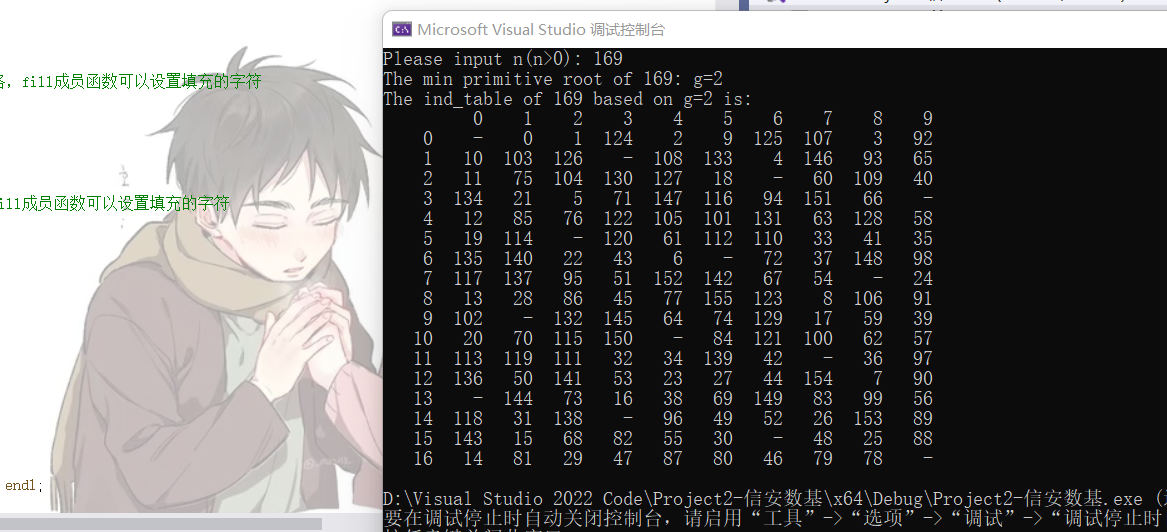
（4）内层循环内部，利用i和j算出此时对应的指数a，然后**判断a是否在n的缩系之中（若不在，即a不与n互素，则不存在n的指数等于a，利用“-”输出）**。由于n的缩系保存在的vector类型变量ReducedResidueSystem\_n是由小到大获取的，因此不需要遍历，可以与同样从小变大的a依次进行判断。利用1.（9）计算离散对数。

使用**currentNum记录此时遍历到了n的缩系的第几个元素了。**

（5）若a大于等于n，则直接输出**“-”填满。**

* **运行示例：（分别选取了n=103,169,19以及没有原根的20）**









* **其他：**

1. 为了实现求解最小原根以及构造指数表，多次使用了之前已经编程得到的定理来应用，**实现了函数编程。**
2. 整体来看，代码略冗长，虽然可读性较强，但应该存在更简单实现功能的方式，**代码有待改进，空间复杂度和时间复杂度都有待提高。**