椭圆曲线编程练习报告

姓名：武桐西　　学号：2112515　　班级：信安一班

编程练习——椭圆曲线

➢ 源码部分：

本程序实现了Z_p上椭圆曲线$E_p(a,b)$的相关计算（$E_p(a,b)$椭圆曲线计算器）。

本程序按照面向对象的编程思想，封装类，调用共有接口实现。源码部分共包含 5 个文件：

① main.cpp（主函数，调用接口实现Z_p上椭圆曲线$E_p(a,b)$上的相关计算）

② EC.h（椭圆曲线类 EC 的定义及相关成员函数的声明）

③ EC.cpp（椭圆曲线类 EC 的成员函数的定义和实现）

④ Euclid.h（椭圆曲线类 EC 中需要用到的初等数论的相关函数的声明）

⑤ Euclid.cpp（椭圆曲线类 EC 中需要用到的初等数论的相关函数的定义和实现）。

① main.cpp:

```
#include<iostream>
#include<cstring>
#include"EC.h"
using namespace std;
// 有限域F_p上的椭圆曲线 E_P(a, b)
```

```cpp
int main() {
    EC E;
    cout << "Elliptic Curve Calculator\n";

    int p, a, b;
    while (true) {
        cout << "Please input p, a, b for E_p(a, b):\n";
        cout << "p = ";
        cin >> p;
        cout << "a = ";
        cin >> a;
        cout << "b = ";
        cin >> b;

        if (E.Init(p, a, b)) {
            E.PrintEC();
            break;
        }
        else {
            E.PrintEC();
            cout << "Please input again!\n";
        }
    }

    // 帮助信息
    E.help();
    string s;

    cout << "Please input command:\n";
    cin >> s;
    while (s != "exit") {
        if (s == "help") {
            E.help();
        }
        if (s == "getEC") {
            E.getEC();
            cout << endl; // 换行
        }
        if (s == "onEC") {
            int x, y;
            cout << "x = ";
            cin >> x;
            cout << "y = ";
```

```
        cin >> y;
        if (E.onEC(x, y)) {
            cout << "P(" << x << ", " << y << ") is on ";
            E.getEC();
            cout << "!\n";
        }
        else {
            cout << "P(" << x << ", " << y << ") is NOT on ";
            E.getEC();
            cout << "!\n";
        }
    }
    if (s == "P+Q") {
        int x, y, m, n;
        cout << "x1 = ";
        cin >> x;
        cout << "y1 = ";
        cin >> y;
        cout << "x2 = ";
        cin >> m;
        cout << "y2 = ";
        cin >> n;

        if (!E.onEC(x, y) || !E.onEC(m, n)) {
            cout << "P或Q不是";
            E.getEC();
            cout << "上的点! \n";
        }
        else {
            int* R = E.P_plus_Q(x, y, m, n);
            if (R) {
                cout << "P + Q = ("
                    << x << ", " << y << ") + ("
                    << m << ", " << n << ")\n"
                    << " = ("
                    << R[0] << ", " << R[1] << ")\n";
                delete[]R;
            }
            else {
                cout << "P + Q = ("
                    << x << ", " << y << ") + ("
                    << m << ", " << n << ") = 0 (无穷远点)\n";
            }
        }
```

```
    }
    if (s == "mP") {
        int m, x, y;
        cout << "x = ";
        cin >> x;
        cout << "y = ";
        cin >> y;
        cout << "m = ";
        cin >> m;

        if (!E.onEC(x, y)) {
            cout << "P不是";
            E.getEC();
            cout << "上的点! \n";
        }
        else {
            int* R = E.mP(x, y, m);
            if (R) {
                cout << "mP = " << m << "("
                    << x << ", " << y << ")\n"
                    << " = ("
                    << R[0] << ", " << R[1] << ")\n";
                delete[]R;
            }
            else {
                cout << "mP = (" << m << "("
                    << x << ", " << y << ") = 0 (无穷远点)\n";
            }
        }
    }
    if (s == "OrdP") {
        int x, y;
        cout << "x = ";
        cin >> x;
        cout << "y = ";
        cin >> y;

        if (!E.onEC(x, y)) {
            cout << "P不是";
            E.getEC();
            cout << "上的点! \n";
        }
        else {
            cout << "ord(P) = " << E.OrdP(x, y) << endl;
```

```
            }
        }
        if (s == "OrdE") {
            cout << "#E = " << E.OrdE() << endl;
        }
        if (s == "allPoints") {
            cout << "All points on ";
            E.getEC();
            cout << ":\n";
            E.allPoints();
        }

        cout << "Please input command:\n";
        cin >> s;
    }

    system("pause"); // 暂停界面
    return 0;
}
```

② EC.h

```
#pragma once
#include<iostream>
#include<stack>
#include"Euclid.h"
using namespace std;

class EC {
    // 有限域F_p上的椭圆曲线E_p(a, b)
    int p;
    int a, b;
    bool isValid; // E_p(a, b)是椭圆曲线
    int ord; // #E
public:
    EC() :isValid(false) {};
    void help() const; // 帮助信息
    bool Init(int P, int A, int B); // 初始化
    void PrintEC() const; // 打印当前EC的信息
    bool onEC(int x, int y); // 判断P(x, y)是否在EC上
    int* P_plus_Q(int x, int y, int m, int n); //计算 P+Q
    int* mP(int x, int y, int m); // 计算 mP
    int OrdP(int x, int y); // 计算点P(x, y)的阶
```

```
    int OrdE(); // 计算EC的阶
    void allPoints(bool isPrint = true); // 计算EC上的所有点
    void getEC() const; // 获取当前EC参数
};
```

③ EC.cpp

```
#include "EC.h"
#include<math.h>

void EC::help() const {
    cout << "This is Help Info:\n";
    cout << "- Input 'help' for Help Info.\n";
    cout << "- Input 'exit' to exit the program.\n";
    cout << "- Input 'getEC' to get the parameters of current E_p(a, b).\n";
    cout << "- Input 'onEC' to check if P(x, y) is on EC.\n";
    cout << "- Input 'P+Q' to calculate P + Q.\n";
    cout << "- Input 'mP' to calculate mP.\n";
    cout << "- Input 'OrdP' to calculate ord(P).\n";
    cout << "- Input 'OrdE' to calculate #E.\n";
    cout << "- Input 'allPoints' to calculate all points on EC.\n";
}

bool EC::Init(int P, int A, int B) {
    // 检查P
    if (P <= 3) {
        isValid = false;
        // 打印信息
        // PrintEC();
        return false;
    }
    isValid = true;
    for (int i = 2; i <= sqrt(1.0 * P); i++) {
        if (P % i == 0) {
            isValid = false;
            // 打印信息
            // PrintEC();
            return false;
        }
    }

    // 赋值
    this->p = P;
```

```cpp
    this->a = A % P; // 限定于Z_p
    this->b = B % P; // 限定于Z_p

    if (a < 0)
        a += p; // 转为非负数
    if (b < 0)
        b += p; // 转为非负数

    if ((4 * a * a * a + 27 * b * b) % p == 0) {
        isValid = false; // 不光滑
    }

    return isValid;
}

void EC::PrintEC() const {
    if (isValid) {
        /*
        cout << "p = " << p
            << " a = " << a
            << " b = " << b << endl;
        */
        cout << "E_" << p << "(" << a << ", " << b << ") 是椭圆曲线！\n";
    }
    else {
        // 当前不是EC
        cout << "当前参数无效，不是椭圆曲线！\n";
    }
}

bool EC::onEC(int x, int y) {// 判断当前点P(x, y)是否在EC上
    // 限定于Z_p
    x %= p;
    y %= p;
    if (x < 0)
        x += p;
    if (y < 0)
        y += p;

    // 计算y^2
    int y2 = (x * x * x) % p;
    y2 = (y2 + a * x) % p;
    y2 = (y2 + b) % p;
```

```
    // 验证
    int Y2 = (y * y) % p;
    if (y2 == Y2)
        return true;
    return false;
}

int* EC::P_plus_Q(int x, int y, int m, int n) {
    // 计算P(x, y) + Q(m, n)
    // 若返回值为 nullptr，则表示无穷远点O；否则 R(int[0], int[1])

    // 限定于Z_p
    x %= p;
    y %= p;
    if (x < 0)
        x += p;
    if (y < 0)
        y += p;
    m %= p;
    n %= p;
    if (m < 0)
        m += p;
    if (n < 0)
        n += p;

    int k; // 斜率

    if (x == m) {
        // 横坐标相等
        if (n == ((p - y) % p)) {
            return nullptr; // 相加为无穷远点
        }
        // 倍加
        k = (3 * x * x) % p;
        k = (k + a) % p;
        int tmp = (2 * y) % p;
        tmp = Euclid(tmp, p);
        k = (k * tmp) % p;
    }
    else {
        // 点加
        k = (n - y) % p;
        if (k < 0)
            k += p;
```

```
        int tmp = (m - x) % p;
        if (tmp < 0)
            tmp += p;
        // 求逆元
        tmp = Euclid(tmp, p);
        k = (k * tmp) % p;
    }

    int* R = new int[2]; // R = P + Q
    R[0] = (k * k) % p;
    R[0] = (R[0] - x - m) % p;
    if (R[0] < 0)
        R[0] += p;
    R[1] = (k * (x - R[0]) - y) % p;
    if (R[1] < 0)
        R[1] += p;

    return R;
}

int* EC::mP(int x, int y, int m) {
    /*
    * 计算 mP
    * 倍加-和算法
    * 若返回值为 nullptr，则表示无穷远点O；否则 R(int[0], int[1])
    */

    if (m == 0)
        return nullptr; // 0P = O

    bool isNeg = false; // 判断m是否为负数
    if (m < 0) {
        isNeg = true;
        m = -m; // 将m置为正数
    }

    // 提取m的二进制
    stack<bool> S; // 保存m的二进制位
    while (m) {
        S.push(m & 1); // 按位与运算
        m >>= 1; // 采用移位运算，加快运算速度
    }
```

```
    int len = S.size();
    S.pop();
    // 初始化
    int* R = new int[2];
    R[0] = x;
    R[1] = y;

    // 倍加-和算法
    int* tmp;
    while (!S.empty()) {
        tmp = R;
        if (R) { // 当前点不为无穷远点
            R = P_plus_Q(R[0], R[1], R[0], R[1]);
            delete[]tmp;
        }
        // 若P为无穷远点，倍加之后仍未无穷远点
        if (S.top()) {
            tmp = R;
            if (R) { // 当前点不为无穷远点
                R = P_plus_Q(R[0], R[1], x, y);
                delete[]tmp;
            }
            else { // 当前点为无穷远点
                R[0] = x;
                R[1] = y;
            }
        }
        S.pop();
    }

    // 若m为负数，则求|m|P的椭圆曲线加法群的加法逆元
    if (isNeg)
        R[1] = (p - R[1]) % p;

    return R;
}

int EC::OrdP(int x, int y) {
    // 限定于Z_p
    x %= p;
    y %= p;
    if (x < 0)
        x += p;
    if (y < 0)
```

```
        y += p;

    // 易知，ord(P) > 1

    // 求#E的因子
    int E = OrdE();
    vector<int> F;
    Factor(E, F);
    int len = F.size();

    int* R = nullptr;
    for (int i = 0; i < len; i++) {
        // 因子从小到大排列
        R = mP(x, y, F[i]);
        if (!R) {
            // R为无穷远点
            return F[i];
        }
        else {
            delete[]R;
        }
    }
}

int EC::OrdE() {
    allPoints(false); // 设置this->ord
    return this->ord;
}

void EC::allPoints(bool isPrint) {
    this->ord = 1; // 考虑无穷远点O
    if (isPrint)
        cout << "O(无穷远点)\n";
    int y2;
    for (int x = 0; x < p; x++) { // 遍历Z_p
        // 计算y^2
        y2 = (x * x * x) % p;
        y2 = (y2 + a * x) % p;
        y2 = (y2 + b) % p;

        // 计算y2对p的Legendre符号
        int L = Legendre(y2, p);
        if (L == -1)
            continue;
```

```
        if (L == 0) {
            // 仅有一解 (x, 0)
            this->ord++;
            if (isPrint) {
                cout << "(" << x << ", 0)\n";
            }
            continue;
        }
        if (L == 1) {
            // 存在2解
            this->ord += 2;
            if (isPrint) {
                // 遍历Z_p
                int y = 0;
                for (; y < p; y++) {
                    if (((y * y) % p) == y2) {
                        break; // 找到一个解，即可终止
                    }
                }
                cout << "(" << x << ", " << y << ") ";
                cout << "(" << x << ", " << ((p - y) % p) << ")\n";
            }
        }
    }

    if (isPrint)
        cout << "Total: " << this->ord << endl;
}

void EC::getEC() const {
    // 无换行
    if (isValid)
        cout << "E_" << p << "(" << a << ", " << b << ")";
}
```

④ Euclid.h

```
#pragma once
#include<iostream>
#include<vector>
#include<math.h>
using namespace std;

```

```
// 扩展欧几里得算法求乘法逆元
int Euclid(int a, int b);

// 求正整数的非1的因子
void Factor(int m, vector<int>& A);

// 求a对p的Legendre符号
int Legendre(int a, int p);

// Eratosthenes筛法求n以内的所有素数
void Eratosthenes(int n, bool*& A);

// 算术基本定理：求n的素因子分解
void PrimeFactor(int n, vector<int>& F, vector<int>& C);
```

⑤ Euclid.cpp

```
#include "Euclid.h"

int Euclid(int a, int b) {
    /*
    * 求a模b的乘法逆元（小者模大者的乘法逆元）
    * return: a^(-1) (mod b)
    */
    vector<int> r;//余数序列
    r.push_back(a > b ? a : b);//a, b中的大者
    r.push_back(a < b ? a : b);//a, b中的小者
    vector<int> q;//商序列
    q.push_back(-1);//q[0]中的值无效
    vector<int> s;
    s.push_back(1);
    s.push_back(0);
    vector<int> t;
    t.push_back(0);
    t.push_back(1);

    int x = 0;//索引
    while (r[x] % r[x + 1]) {//余数非零，则循环
        r.push_back(r[x] % r[x + 1]);
        q.push_back(r[x] / r[x + 1]);
        s.push_back(s[x] - s[x + 1] * q[x + 1]);
        t.push_back(t[x] - t[x + 1] * q[x + 1]);
        x++;
```

```
    }

    int l = r.size() - 1;//序列的末尾元素下标
    if (r[l] == 1) {
        //可用扩展欧几里得算法求逆元(乘法逆元存在)
        if (a > b) {//根据a, b的大小讨论
            //转为最小正缩系中
            if (s[l] < 0)
                s[l] = b + s[l];
            if (t[l] < 0)
                t[l] = a + t[l];
            return t[l];
        }
        else {
            //转为最小正缩系中
            if (s[l] < 0)
                s[l] = a + s[l];
            if (t[l] < 0)
                t[l] = b + t[l];
            return t[l];
        }
    }
    return 0; // 实则不需要
}

void Factor(int m, vector<int>& A) {
    // m >= 1
    for (int i = 2; i <= m; i++) {
        if (m % i == 0) {
            A.push_back(i); // m的因子从小到大排列
        }
    }
}

int Legendre(int a, int p) {
    // a \in Z_p
    if (a == 0)
        return 0; // Legendre符号为0，表示倍数
    if (a == 1)
        return 1; // Legendre符号为1
    if (a == 2) {
        if ((p % 8 == 3) || ((p % 8) == 5))
            return -1;
        return 1;
```

```
    }

    vector<int> Factor;
    vector<int> Count;
    PrimeFactor(a, Factor, Count); // 素因子分解

    int Ans = 1; // 保存结果

    int len = Factor.size();
    for (int i = 0; i < len; i++) {
        Count[i] %= 2;
        if (Count[i]) { // 非零
            if (Factor[i] == 2) {
                // (2/p)单独处理
                if (p % 8 == 3 || p % 8 == 5)
                    Ans *= -1;
            }
            else {
                // 利用二次互反律
                int t = ((p - 1) / 2) * ((Factor[i] - 1) / 2);
                if (t % 2)
                    t = -1;
                else
                    t = 1;
                t *= Legendre((p % Factor[i]), Factor[i]);
                Ans *= t;
            }
        }
    }
    return Ans;
}

void Eratosthenes(int n, bool*& A) {
    //n为代求范围[2,n]，A保存整数表
    A = new bool[n + 1]{ false };
    for (int i = 2; i <= n; i++)
        A[i] = true;//初始化为true
    if (n < 4)
        return;//递归结束
    int n_ = (int)(sqrt(1.0 * n));
    bool* A_ = nullptr;
    Eratosthenes(n_, A_);
    for (int i = 2; i <= n_; i++) {
        if (A_[i]) {
```

```
            for (int j = 2; j <= n; j++) {
                if (j % i == 0 && j != i)
                    A[j] = false;
            }
        }
    }
}

void PrimeFactor(int n, vector<int>& F, vector<int>& C) {
    //先用Eratosthenes筛法求[2,n]内所有素数
    bool* A = nullptr;
    Eratosthenes(n, A);
    int a = n;
    for (int i = 2; i <= n; i++) {
        if (A[i]) {//i为素数
            int count = 0;//记录素因子指数
            while (a % i == 0) {
                count++;
                a /= i;
            }
            if (count) {//count非零
                F.push_back(i);
                C.push_back(count);
            }
        }
    }
}
```

➢ 说明部分：

(一) 算法说明

1. 给定参数p、a、b，判断$E_p(a,b)$是否为椭圆曲线。

相关函数：`bool EC::Init(int P, int A, int B);`

① 检验p是否是大于3的素数。

② 检测光滑性：$4a^3+27b^2\not\equiv 0(\mathrm{mod}\,p)$。

2. 判断给定点$P(x,y)$是否在$E_p(a,b)$上。

相关函数：`bool EC::onEC(int x, int y);`

将$P(x,y)$带入$E_p(a,b)$的方程，验证是否成立。

3. 计算$P+Q$

相关函数：

```
int* EC::P_plus_Q(int x, int y, int m, int n);
int Euclid(int a, int b);
```

利用点加或倍加公式（依据P、Q的横坐标是否相等进行分类讨论）进行计算。

其中，需要求解模p的乘法逆元，调用扩展欧几里得算法。

注意事项：

① 需要注意$P+Q=O\big($无穷远点$\big)$的情况。

② 利用扩展欧几里得算法求模p的乘法逆元。

4. 计算mP

相关函数：

```
int* EC::mP(int x, int y, int m);
int* EC::P_plus_Q(int x, int y, int m, int n);
```

采用倍加-和算法求解。

注意事项：

① 需要注意运算过程中$kP=O\big($无穷远点$\big)$的情况，需要针对是否

为无穷远点进行讨论。

② 注意m为非正数（零和负数）的情况。

③ 提取m的二进制时，可以采用移位运算，增加程序运行速度。

④ 可以在 Double 和 Add 的过程中调用计算$P+Q$的函数。

5. 计算$ord(P)$

相关函数：

```
int EC::OrdP(int x, int y);
void Factor(int m, vector<int>& A);
```

先求$\#E$，由于P不是无穷远点，因而$\mathrm{ord(P)} > 1$，而由 Lagrange 定理可知，$\mathrm{ord(P)} \mid \#\mathrm{E}$，因此$\mathrm{ord(P)}$必为$\#\mathrm{E}$的非一的因子。

因此只需遍历$\#E$的非一的因子k，按照因子从小到大的顺序，依次计算kP，判断其是否为无穷远点即可，第一个等于无穷远点的因子，即为所求。

6. 计算$\#E$

相关函数：

```
int EC::OrdE();
void EC::allPoints(bool isPrint);
```

调用求椭圆曲线$E_p(a,b)$上所有点的函数，所求得的点的个数即为$\#E$。

注意事项：

不要遗漏无穷远点O。

7. 给出$E_p(a,b)$上的所有点

相关函数：

```
void EC::allPoints(bool isPrint);
int Legendre(int a, int p);
void Eratosthenes(int n, bool*& A);
void PrimeFactor(int n, vector<int>& F, vector<int>& C);
```

x遍历模p的最小非负完全剩余系，依次求得$y^2(mod\ p)$的值，然后计算 Legendre 符号$\left(\frac{y^2}{p}\right)$，若$\left(\frac{y^2}{p}\right)=0$（表示$y^2$是$p$的倍数），则只有一解；若$\left(\frac{y^2}{p}\right)=1$，则有两个解；若$\left(\frac{y^2}{p}\right)=-1$，则无解。

在求解 Legendre 符号时，需要用 Eratosthenes 筛法求$[2,a]$的所有素数，用算术基本定理对a进行素因子分解，利用二次互反律简化 Legendre 符号的计算（递归调用函数 `int Legendre(int a, int p)`）。

(二) 编程技巧：

① 为便于程序的编写以及计算处理的方便性，同时为了防止溢出，可以将如a、b等首先转换到$Z_p=\{0,1,\cdots,p-1\}$上，然后再进行运算。

② 在计算过程中，可以对每一步求解的结果先进行模p运算，然后再继续运算，这样可以有效防止溢出，提升运算速度。

➢ 运行示例：

```
D:\软件\VS project\信息安全数学基础编程作业\Elliptic Curve\Debug\Elliptic Curve.exe
Elliptic Curve Calculator
Please input p, a, b for E_p(a, b):
p = 11
a = 0
b = 0
当前参数无效，不是椭圆曲线！
Please input again!
Please input p, a, b for E_p(a, b):
p = 19
a = 3
b = 7
E_19(3, 7) 是椭圆曲线！
This is Help Info:
- Input 'help' for Help Info.
- Input 'exit' to exit the program.
- Input 'getEC' to get the parameters of current E_p(a, b).
- Input 'onEC' to check if P(x, y) is on EC.
- Input 'P+Q' to calculate P + Q.
- Input 'mP' to calculate mP.
- Input 'OrdP' to calculate ord(P).
- Input 'OrdE' to calculate #E.
- Input 'allPoints' to calculate all points on EC.
Please input command:
onEC
x = 1
y = 2
P(1, 2) is NOT on E_19(3, 7)!
Please input command:
onEC
x = 1
y = 7
P(1, 7) is on E_19(3, 7)!
Please input command:
onEC
x = 3
y = 9
P(3, 9) is on E_19(3, 7)!
Please input command:
P+Q
x1 = 1
```

```
D:\软件\VS project\信息安全数学基础编程作业\Elliptic Curve\Debug\Elliptic Curve.exe
y = 9
P(3, 9) is on E_19(3, 7)!
Please input command:
P+Q
x1 = 1
y1 = 7
x2 = 3
y2 = 9
P + Q = (1, 7) + (3, 9)
  = (16, 16)
Please input command:
mP
x = 1
y = 7
m = 7
mP = 7(1, 7)
  = (15, 11)
Please input command:
OrdP
x = 1
y = 7
ord(P) = 11
Please input command:
OrdE
#E = 22
Please input command:
allPoints
All points on E_19(3, 7):
O(无穷远点)
(0, 8) (0, 11)
(1, 7) (1, 12)
(3, 9) (3, 10)
(4, 8) (4, 11)
(8, 7) (8, 12)
(10, 7) (10, 12)
(12, 2) (12, 17)
(13, 1) (13, 18)
(14, 0)
(15, 8) (15, 11)
(16, 3) (16, 16)
Total: 22
Please input command:
exit
请按任意键继续. . .
```