# 椭圆曲线编程练习报告

姓名：胡博浩 学号：2212998 班级：信息安全

##### 编程要求

实现基本的Zp上的椭圆曲线Ep(a,b)的计算,平台可以是Windows/Linux/macOS,具体如下:

1.功能要求:

■ 给定参数p,a,b,判断 Ep(a,b)是否为椭圆曲线;

■ 判断给定的点P,Q是否在椭圆曲线 Ep(a,b)上;

■ 对在椭圆曲线 Ep(a,b)上的两点P,Q,计算P+Q;

■ 对在椭圆曲线 Ep(a,b)上的点P,使用倍加-和算法计算mP;

■ 对在椭圆曲线 Ep(a,b)上的点P,计算阶ord(P);

■ 对在椭圆曲线 Ep(a,b),计算阶#E;

■ 对在椭圆曲线 E2(a,b),计算所有点;

■ 其他功能的进一步扩展 ..…

2. 编程要求:

■ 不允许使用第三方的库;

■ 按照面向对象的编程思想,封装类,调用公有接口实现;

■ 符合一定的编程规范;

■ 利用之前的知识模块解耦实现:如扩展Euclid算法求逆、二次互反律求Legendre符号、群的一些基础知识等;

■ 在实现功能的基础上,尽可能提高计算的效率等.

##### 源码部分

#include<iostream>

using namespace std;

#define Elliptic\_Curve\_EC "E\_" << p << "(" << a << ',' << b << ")"

#define Point\_P "P(" << x << "," << y << ")"

// 类声明

class Point

{

public:

int x, y;

bool isINF; //是否是无穷远点

Point(int x = 0, int y = 0, bool isINF = false); // 构造函数

friend ostream& operator<< (ostream& out, const Point& p); // 输出运算符重载

void output(ostream& out) const; // 输出函数

};

// 类定义

Point::Point(int x, int y, bool isINF)

: x(x), y(y), isINF(isINF) {}

void Point::output(ostream& out) const

{

if (isINF) out << 'O'; // 无穷远点

else out << '(' << x << ',' << y << ')'; // 非无穷远点

}

ostream& operator<< (ostream& out, const Point& p)

{

p.output(out);

return out;

}

// 椭圆曲线类声明

class Elliptic\_Curve

{

private:

int p, a, b;

int powMod(int x, int n, int mod); // 快速幂算法

int get\_Inverse(int a, int m); // 在 (a, m) = 1 的条件下，求a模m的乘法逆元

int Legendre(int a, int p); // 计算勒让德符号

public:

Elliptic\_Curve(int p, int a, int b);

bool is\_inverse(const Point& p1, const Point& p2); //判断两个点是否互逆

bool is\_Elliptic\_Curve(); //检查当前参数是否能构成椭圆曲线

bool is\_on\_Elliptic\_Curve(const Point& p); //判断p点是否在椭圆曲线上

Point addPoints(const Point& p1, const Point& p2); //进行点加运算

Point multiplyPoint(Point p, int k); //对点p进行倍加

Point inversePoint(const Point& pt); // 添加计算点的逆元的方法

int ord\_of\_Point(const Point& p); //计算点p的阶

int ord\_of\_Elliptic\_Curve(); //计算此椭圆曲线的阶#E

int show\_all\_Points(); //展示出椭圆曲线上的所有点

};

// 椭圆曲线类定义

int Elliptic\_Curve::powMod(int x, int n, int mod) // 快速幂算法

{

int c = 1;

while (n) {

if (n & 1) {

c = (c \* x) % mod;

}

x = (x \* x) % mod;

n >>= 1;

}

return c;

}

int Elliptic\_Curve::get\_Inverse(int a, int m) //在 (a, m) = 1 的条件下，求a模m的乘法逆元

{

a = (a + m) % m;

int s0 = 1, s1 = 0;

int r0 = a, r1 = m;

while (r1 != 0)

{

int q = r0 / r1;

int tmp = r1;

r1 = r0 % r1;

r0 = tmp;

tmp = s1;

s1 = s0 - s1 \* q;

s0 = tmp;

}

return (s0 + m) % m;

}

int Elliptic\_Curve::Legendre(int a, int p) //p是奇素数, (a, p) = 1

{

if (a < 0)

{

if (a == -1)

{

return p % 4 == 1 ? 1 : -1;

}

return Legendre(-1, p) \* Legendre(-a, p);

}

a %= p;

if (a == 1)

{

return 1;

}

else if (a == 2)

{

if (p % 8 == 1 || p % 8 == 7) return 1;

else return -1;

}

// 下面将a进行素数分解

int ret = 1;

while (a > 1)

{

int prime = 2;

while (a % prime == 0)

{

a /= prime;

if (prime == 2)

{

if (p % 8 == 3 || p % 8 == 5) ret = -ret;

}

else

{

if (((prime - 1) \* (p - 1) / 4) % 2 == 1) ret = -ret;

ret \*= Legendre(p % prime, prime);

}

}

if (a == 1) break;

if (a % 4 == 3 && p % 4 == 3) ret = -ret;

swap(a, p);

a %= p;

}

return ret;

}

Elliptic\_Curve::Elliptic\_Curve(int p, int a, int b) //椭圆曲线构造函数

: p(p), a(a), b(b) {}

bool Elliptic\_Curve::is\_inverse(const Point& p1, const Point& p2)

{

return (p1.x - p2.x) % p == 0 && (p1.y + p2.y) % p == 0;

}

bool Elliptic\_Curve::is\_Elliptic\_Curve() //检查当前参数是否能构成椭圆曲线

{

return (4 \* powMod(a, 3, p) + 27 \* powMod(b, 2, p)) % p != 0; // 椭圆曲线条件判断

}

bool Elliptic\_Curve::is\_on\_Elliptic\_Curve(const Point& pt)// 判断点是否在曲线上

{

if (pt.isINF) return true;

return (powMod(pt.y, 2, p) - powMod(pt.x, 3, p) - a \* pt.x - b) % p == 0;

}

Point Elliptic\_Curve::addPoints(const Point& p1, const Point& p2)// 计算两点之和

{

if (p1.isINF) return p2;

if (p2.isINF) return p1;

if (is\_inverse(p1, p2)) return { 0, 0, true };

if ((p1.x - p2.x) % p == 0) //倍加公式

{

int k = ((3 \* powMod(p1.x, 2, p) + a) \* get\_Inverse(2 \* p1.y, p) % p + p) % p;

int x3 = ((powMod(k, 2, p) - 2 \* p1.x) % p + p) % p;

int y3 = ((k \* (p1.x - x3) - p1.y) % p + p) % p;

return { x3, y3 };

}

else //点加公式

{

int k = ((p2.y - p1.y) \* get\_Inverse(p2.x - p1.x, p) % p + p) % p;

int x3 = ((powMod(k, 2, p) - p1.x - p2.x) % p + p) % p;

int y3 = ((k \* (p1.x - x3) - p1.y) % p + p) % p;

return { x3, y3 };

}

}

Point Elliptic\_Curve::multiplyPoint(Point pt, int k)// 使用倍加-和算法计算kP

{

Point ret(0, 0, true);

while (k)

{

if (k & 1)

{

ret = addPoints(ret, pt);

}

pt = addPoints(pt, pt);

k >>= 1;

}

return ret;

}

Point Elliptic\_Curve::inversePoint(const Point& pt) {

if (pt.isINF) return pt;

return Point(pt.x, (p - pt.y) % p);

}

int Elliptic\_Curve::ord\_of\_Point(const Point& pt)// 计算点的阶

{

if (!is\_on\_Elliptic\_Curve(pt)) return -1;// 点不在椭圆曲线上，返回-1

int ord = 1;

Point tmp = pt;

while (!tmp.isINF)

{

tmp = addPoints(tmp, pt);

++ord;

}

return ord;

}

int Elliptic\_Curve::ord\_of\_Elliptic\_Curve()

{

int ord = 1;

for (int x = 0; x < p; ++x)

{

int tmp = (x \* x \* x + a \* x + b + p) % p;

if (tmp == 0)

{

ord += 1;

}

else if (Legendre(tmp, p) == 1)

{

ord += 2;

}

}

return ord;

}

int Elliptic\_Curve::show\_all\_Points()

{

cout << "O ";

int sum = 1;

for (int x = 0; x < p; ++x)

{

int tmp = (x \* x \* x + a \* x + b + p) % p;

if (tmp == 0)

{

cout << " (" << x << ',' << "0) ";

sum++;

}

else if (Legendre(tmp, p) == 1) //贡献两个点

{

for (int y = 1; y < p; ++y) //从1遍历到p-1，寻找解

{

if ((y \* y - tmp) % p == 0)

{

cout << " (" << x << ',' << y << ") ";

sum++;

cout << " (" << x << ',' << p - y << ") ";

sum++;

break;

}

}

}

}

cout << endl;

return sum;

}

int main()

{

cout << "Z\_P上的椭圆曲线E\_p(a, b)的计算" << endl;

cout << "————————————————————————————————————————" << endl;

cout << endl;

//功能一：给定参数p，a，b，判断E\_p(a,b)是否为椭圆曲线

cout << "1.判断所给参数是否能构成一个椭圆曲线" << endl;

int p, a, b;

cout << "请输入椭圆曲线的参数 p: ";

cin >> p;

cout << "请输入椭圆曲线的参数 a: ";

cin >> a;

cout << "请输入椭圆曲线的参数 b: ";

cin >> b;

Elliptic\_Curve ec(p, a, b);

cout << Elliptic\_Curve\_EC << " is ";

if (!ec.is\_Elliptic\_Curve())

{

cout << "not ";

system("pause");

return 0;

}

cout << "Elliptic\_Curve" << endl;

cout << endl;

//功能二：判断给定的点P，Q是否在椭圆曲线E\_p(a,b)上

cout << "2.判断给出的点是否在给定的椭圆曲线上" << endl;

int x, y;

cout << "输入 x: ";

cin >> x;

cout << "输入 y: ";

cin >> y;

cout << Point\_P " is ";

if (!ec.is\_on\_Elliptic\_Curve(Point(x, y))) cout << "not ";

cout << "on " << Elliptic\_Curve\_EC << endl;

cout << endl;

//功能三：对于椭圆曲线E\_p(a,b)上的两点P,Q，计算P+Q

cout << "3.计算给定的两点相加" << endl;

int x1, y1, x2, y2;

cout << "输入 x1: ";

cin >> x1;

cout << "输入 y1: ";

cin >> y1;

cout << "输入 x2: ";

cin >> x2;

cout << "输入 y2: ";

cin >> y2;

cout << "(" << x1 << "," << y1 << ")" << " + " << "(" << x2 << "," << y2 << ") = " << ec.addPoints({ x1, y1 }, { x2, y2 }) << endl;

cout << endl;

//功能四：对于椭圆曲线E\_p(a,b)上的点P，使用倍加-和算法计算mP

cout << "4.计算给出的点的倍加" << endl;

cout << "输入 x: ";

cin >> x;

cout << "输入 y: ";

cin >> y;

int m;

cout << "输入倍数: ";

cin >> m;

cout << m << Point\_P << " = " << ec.multiplyPoint({ x, y }, m) << endl;

cout << endl;

//功能五：对于椭圆曲线E\_p(a,b)上的点P，计算阶ord（P）

cout << "5.计算给出的点的阶" << endl;

cout << "输入 x: ";

cin >> x;

cout << "输入 y: ";

cin >> y;

int ord = ec.ord\_of\_Point({ x, y });

if (ord != -1)

{

cout << Point\_P << "的阶是" << ord << endl;

}

else {

cout << Point\_P << " is not on " << Elliptic\_Curve\_EC << endl;

}

cout << endl;

//功能六：对于椭圆曲线E\_p(a,b)，计算阶#E

cout << "6.计算给出的椭圆曲线的阶" << endl;

cout << Elliptic\_Curve\_EC << "的阶是" << ec.ord\_of\_Elliptic\_Curve() << endl;

cout << endl;

//功能七：对于椭圆曲线E\_p(a,b)，计算所有点

cout << "7.列出给出的椭圆曲线上的所有点" << endl;

ec.show\_all\_Points();

cout << endl;

// 新增功能：计算椭圆曲线上的点的逆元

cout << "8. 计算给定点的逆元" << endl;

cout << "输入 x: ";

cin >> x;

cout << "输入 y: ";

cin >> y;

cout << Point\_P << " 的逆元是 " << ec.inversePoint({ x,y }) << endl;

system("pause");

return 0;

}

##### 说明部分

利用之前的知识模块解耦实现了椭圆曲线密码学中的基本操作，包括定义点、定义椭圆曲线、点的加法、点的倍加、计算点的阶、计算曲线的阶以及列出曲线上的所有点等功能。

1. 定义点类Point：

定义了一个名为Point的类来表示椭圆曲线上的点。每个点由其 x 和 y 坐标表示，并包含一个布尔值isINF，用于标识是否是无穷远点。该类还重载了输出运算符<<以便于输出点的信息。

2. 定义椭圆曲线类Elliptic\_Curve：

Elliptic\_Curve的类表示椭圆曲线，包含椭圆曲线方程的参数p、a和b。该类提供了各种成员函数用于椭圆曲线上的操作，如判断参数是否构成椭圆曲线、判断点是否在曲线上、点加法、点倍加、计算点的阶、计算曲线的阶以及列出曲线上的所有点等。辅助函数powMod、get\_Inverse和Legendre用于数学计算，分别实现了快速幂算法、扩展Euclid算法求逆、二次互反律求Legendre符号。

3. 椭圆曲线基本操作：

实现了题目所要求的以下八种功能：（七种基础功能+一种拓展功能）

1）is\_Elliptic\_Curve函数检查当前参数是否能构成椭圆曲线。

2）is\_on\_Elliptic\_Curve函数判断点是否在椭圆曲线上。

3）addPoints函数用于计算两点之和。

4）multiplyPoint函数用于对点进行倍加。

5）ord\_of\_Point函数用于计算点的阶。

6）ord\_of\_Elliptic\_Curve函数用于计算椭圆曲线的阶。

7）show\_all\_Points函数用于展示出椭圆曲线上的所有点。

**8）扩展功能：inversePoint函数用于计算点的逆元。**

4. 主函数main：

进行检查给定参数是否构成椭圆曲线、检查点是否在曲线上、点相加、点倍加、计算点的阶和曲线的阶以及列出曲线上的所有点等操作。

该代码实现了椭圆曲线密码学中的基本功能，并在多个方面展现了良好的编程实践和面向对象的设计思想：

1. 不使用第三方库：

代码没有依赖于任何第三方库，完全使用了C++标准库和语言特性。

2. 面向对象的封装：

使用了面向对象的编程思想，将椭圆曲线和点封装成了类（Elliptic\_Curve和 Point），通过公有接口提供了对椭圆曲线的操作。

3. 符合编程规范：

 代码采用了清晰的命名规范，变量和函数名字具有描述性，易于理解和维护。

 使用了适当的注释来解释代码功能和实现细节，提高了代码的可读性。

 代码缩进和排版整齐，符合常见的编程风格，易于阅读和理解。

4. 利用之前的知识模块解耦实现：

使用了快速幂算法、扩展的欧几里得算法求逆元、二次互反律求勒让德符号等数学知识，使代码更加灵活和高效。

5. 提高计算效率：

使用了快速幂算法和模素数的性质来提高计算效率，尤其是在计算点的倍加和阶等操作上。并且通过合理的算法设计和优化，避免了不必要的重复计算和循环，提高了代码的性能。

##### 运行示例

