# 椭圆曲线编程练习报告

姓名：张洋　　学号：2111460　　班级：信安二班

## 编程练习——实现基本的$Z_p$上的椭圆曲线$E_p(a,b)$的计算

### Ellipse curve.h

```
#pragma once
#include <iostream>
using namespace std;

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);
    friend ostream& operator<< (ostream& out, const Point& p);
    bool operator ==(const Point& p);
    void Output(ostream& out) const;
};

class Elliptic_Curve
{
private:
    int p;
    int a, b;
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 Test_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 Add(const Point& p1, const Point& p2); //进行点加运算
    Point Add_K_Times(Point p, int k); //对点p进行k倍加
    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
};
```

### Ellipse curve.cpp

```
#include "ellipse curve.h"
#include <ostream>
```

```cpp
#include <iostream>
using namespace std;
int 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 prime = 2;
    int ret = 1;
    while (a > 1)
    {
        int power = 0;
        while (a % prime == 0)
        {
            power++;
            a /= prime;
        }
        if (power % 2 == 1)
        {
            if (prime == 2)
            {
                return Legendre(prime, p);
            }
            else
            {
                if (((prime - 1) * (p - 1) / 4) % 2 == 1)
                {
                    ret = -ret;
```

```
                }
                ret *= Legendre(p, prime);
            }
        }
        prime++;
    }
    return ret;
}

int pow(int x, int n) //x的n次方
{
    int ret = 1;
    while (n)
    {
        if (n & 1)
        {
            ret *= x;
        }
        x *= x;
        n >>= 1;
    }
    return ret;
}

int 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 (1)
    {
        int q = r0 / r1;
        int tmp = r1;
        r1 = r0 % r1;
        r0 = tmp;
        if (r1 == 0)
        {
            break;
        }
        tmp = s1;
        s1 = s0 - s1 * q;
        s0 = tmp;
    }
    return (s1 + m) % m;
```

```
}

Point::Point(int x, int y, bool isINF)
{
    this->x = x;
    this->y = y;
    this->isINF = isINF;
}

ostream& operator<< (ostream& out, const Point& p)
{
    p.Output(out);
    return out;
}

bool Point::operator ==(const Point& p)
{
    return x == p.x && y == p.y;
}

void Point::Output(ostream& out) const
{
    if (isINF)
        cout << 'O';
    else cout << '(' << x << ',' << y << ')';
}

Elliptic_Curve::Elliptic_Curve(int p, int a, int b) //椭圆曲线构造函数
{
    this->p = p;
    this->a = a;
    this->b = b;
}

bool Elliptic_Curve::Test_Is_Elliptic_Curve() //检查当前参数是否能构成椭圆曲线
{
    int tmp = pow(a, 3) * 4 + pow(b, 2) * 27;
    return tmp % p != 0;
}

bool Elliptic_Curve::Is_On_Elliptic_Curve(const Point& pt)
{
    int tmp = pow(pt.y, 2) - (pow(pt.x, 3) + a * pt.x + b);
    return tmp % p == 0;
```

```
}

Point Elliptic_Curve::Add(const Point& p1, const Point& p2)
{
    if (p1.isINF)
    {
        return p2;
    }
    else if (p2.isINF)
    {
        return p1;
    }
    else if (Is_Inverse(p1, p2))
    {
        return { 0, 0, true };
    }
    else
    {
        if ((p1.x - p2.x) % p == 0) //倍加公式
        {
            int k = ((3 * p1.x * p1.x + a) * Get_Inverse(2 * p1.y, p) % p + p) % p;
            int x3 = ((k * k - 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 = ((k * k - p1.x - p2.x) % p + p) % p;
            int y3 = ((k * (p1.x - x3) - p1.y) % p + p) % p;
            return { x3, y3 };
        }
    }
}

Point Elliptic_Curve::Add_K_Times(Point pt, int k)
{
    Point ret(0, 0, true);
    while (k)
    {
        if (k & 1)
        {
            ret = Add(ret, pt);
        }
```

```
        pt = Add(pt, pt);
        k >>= 1;
    }
    return ret;
}

int Elliptic_Curve::Ord_Of_Point(const Point& pt)
{
    int ret = 1;
    Point tmp = pt;
    while (!tmp.isINF)
    {
        tmp = Add(tmp, pt);
        ++ret;
    }
    return ret;
}

int Elliptic_Curve::Ord_Of_Elliptic_Curve()
{
    int ret = 1;
    for (int x = 0; x < p; ++x)
    {
        int tmp = (x * x * x + a * x + b + p) % p;
        if (tmp == 0)
        {
            ret += 1;
        }
        else if (Legendre(tmp, p) == 1)
        {
            ret += 2;
        }
    }
    return ret;
}

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;
}

bool Elliptic_Curve::Is_Inverse(const Point& p1, const Point& p2)
{
    return (p1.x - p2.x) % p == 0 && (p1.y + p2.y) % p == 0;
}
```

## test.cpp

```
#include "ellipse curve.h"
#define Elliptic_Curve_EC "E_" << p << "(" << a << ',' << b << ")"
#define Point_P "P(" << x << "," << y << ")"
#include <iostream>
using namespace std;
int main()
{
    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);
    int x, y;
    cout << endl;
    cout << "1.判断所给参数是否能构成一个椭圆曲线" << endl;
    cout << Elliptic_Curve_EC << " is ";
    if (!ec.Test_Is_Elliptic_Curve())
    {
        cout << "not ";
        return 0;
    }
    cout << "Elliptic_Curve" << endl;

    cout << endl;
    cout << "2.判断给出的点是否在给定的椭圆曲线上" << endl;
    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;
    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 << "结果是" << ec.Add({ x1, y1 }, { x2, y2 }) << endl;

    cout << endl;
    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 times;
```

```
    cout << "输入倍数: ";
    cin >> times;
    cout << "结果是" << ec.Add_K_Times({ x, y }, times) << endl;

    cout << endl;
    cout << "5.计算给出的点的阶" << endl;
    cout << "输入 x: ";
    cin >> x;
    cout << "输入 y: ";
    cin >> y;
    cout << Point_P << "的阶是" << ec.Ord_Of_Point({ x, y }) << endl;

    cout << endl;
    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;
    cout << "7.列出给出的椭圆曲线上的所有点" << endl;
    cout << ec.Show_All_Points();

    return 0;
}
```

**说明部分：**

### 1. 给定参数 p, a, b，判断$E_p(a,b)$是否为椭圆曲线

当 $p$ 为大于 3 的素数时，有限域 $\mathbb{Z}_p$ 上的 Weierstrass 方程

$$y^2 \equiv x^3 + ax + b(\mathrm{mod}p)(4a^3 + 27b^2 \neq 0(\mathrm{mod}p))$$

上的所有点，再加一个无穷远点 $O$ 构成的集合，称为有限域 $\mathbb{Z}_p$ 上的椭圆曲线。所以只需判断 $4a^3 + 27b^2 \neq 0(\mathrm{mod}p)$是否成立。

### 2. 判断给定的点 P, Q 是否在椭圆曲线$E_p(a,b)$上

根据上面的定义，计算$y^2 \equiv x^3 + ax + b(\mathrm{mod}p)$是否成立即可判断给定的点是否在椭圆曲线上。

### 3. 对在椭圆曲线$E_p(a,b)$上的两点 P, Q 计算 P+Q

（1） 首先，通过检查 p1 是否是无穷远点来判断是否存在一个加数是无穷远点。如果是，直接返回另一个点 p2 作为结果。

（2） 然后，通过检查 p2 是否是无穷远点来判断是否存在另一个加数是无穷远点。如果是，直接返回另一个点 p1 作为结果。

（3） 接着，检查 p1 和 p2 是否互为逆元。如果是，说明两点相加结果为无

穷远点，返回表示无穷远点的特殊点。

（4） 如果以上条件都不满足，则根据点的位置关系使用倍加公式或点加公式来计算两点相加的结果。

（5） 如果 p1 和 p2 的 x 坐标之差对模 p 取余为 0，说明两点在同一垂直线上，使用倍加公式进行计算。

计算公式如下：k= $\frac{3x_1^2+2a_2x_1+a_4-a_1y_1}{2y_1+a_1x_1+a_3} \equiv \frac{3x_1^2+a}{2y_1} (\mathrm{mod}\ p)$

$$\begin{cases} x_3 = x_1^2 + \frac{b}{x_1^2} \\ y_3 = x_1^2 + \left(x_1 + \frac{y_1}{x_1}\right)x_3 + x_3 \end{cases}$$

（6） 如果 p1 和 p2 的 x 坐标之差对模 p 取余不为 0，说明两点不在同一垂直线上，使用点加公式进行计算。

计算公式如下：k= $\frac{y_2-y_1}{x_2-x_1} (\mathrm{mod}\ p)$

$$\begin{cases} x_3 = \left(\frac{y_2+y_1}{x_2+x_1}\right)^2 + \frac{y_2+y_1}{x_2+x_1} + a + x_1 + x_2 \\ y_3 = \frac{(y_2+y_1)(x_1+x_3)}{x_2+x_1} + x_3 + y_1 \end{cases}$$

## 4. 求乘法逆元

求取整数 a 对于模 m 的乘法逆元，函数名为 Get_Inverse，接受两个参数 a 和 m。其中 a 和 m 需要满足条件(a, m) = 1，即 a 与 m 互质。函数返回一个整数值，表示 a 模 m 的乘法逆元。

（1） 首先，将 a 对 m 取模，确保 a 处于[0, m-1]的范围内。

（2） 初始化变量 s0 和 s1 分别为 1 和 0，用于存储计算过程中的系数。

（3） 初始化变量 r0 和 r1 分别为 a 和 m，用于存储计算过程中的余数。

（4） 进入循环，通过欧几里得算法进行迭代计算，直到 r1 为 0 为止。

（5） 在每次循环中，计算商 q 为 r0 除以 r1 的整数部分，并更新 r0 为 r1，r1 为 r0 除以 r1 的余数。

（6） 同时更新系数 s0 和 s1，通过计算 s0 减去 s1 乘以 q，并将原来的 s1 赋值给 s0。

（7） 循环结束后，r1 为 0，此时 r0 为 a 和 m 的最大公约数，s1 存储的是 a 模 m 的乘法逆元的系数。

（8） 最后，通过将 s1 加上 m 并对 m 取模，确保乘法逆元处于[0, m-1]的范围内，并返回计算得到的结果。

代码使用欧几里得算法来计算整数 a 对于模 m 的乘法逆元，通过迭代计算和更新系数来求解乘法逆元。

## 5. 对在椭圆曲线$E_p(a,b)$上的点 P，使用倍加-和算法计算 mP

倍加和算法
输入: 椭圆曲线 $E, E$ 上的点 $P$ 和正整数 $d, d$ 的二进制为 $d_t d_{t-1} \cdots d_0$;
输出: $T = dP$;
1. $T \leftarrow d_t$
2. FOR $i = t - 1$ DOWNTO 0
3. $T \leftarrow T + T$;
4. IF $d_i = 1$ THEN
5. $T \ll T + P$;
6. END IF
7. END FOR
8. RETURN $T$;

## 6. 对在椭圆曲线$E_p(a,b)$上的点 P，计算阶 ord(P)

通过连续相加某个点自身，直到得到无穷远点为止所需要的次数。
（1） 将阶的初始值设为 1，用于记录相加的次数。
（2） 创建一个临时变量 tmp，初始化为给定的点 pt。
（3） 进入循环，只要 tmp 不是无穷远点，就执行以下操作：
- 调用椭圆曲线的 Add 函数，将 tmp 和给定的点 pt 相加，得到一个新的点 tmp。
- 将阶的值加 1，表示进行了一次相加操作。
（4） 循环结束后，返回计算得到的阶的值。

## 7.对在椭圆曲线$E_p(a,b)$，计算阶#E

（1） 将阶的初始值设为 1，用于记录椭圆曲线的阶数。
（2） 使用一个循环遍历椭圆曲线上所有可能的 x 坐标值，从 0 到 p-1，其中 p 是模数。
（3） 对于每个 x 坐标，计算一个临时值 tmp，通过对应的椭圆曲线方程计算得到。公式为：tmp = (x * x * x + a * x + b + p) % p。
（4） 检查 tmp 的值，根据以下情况进行操作：
- 如果 tmp 等于 0，表示找到了一个点位于椭圆曲线上的特殊情况。将阶的值加 1，表示找到一个额外的点。
- 否则，如果 tmp 满足 Legendre 符号等于 1，表示 tmp 是一个模 p 的二次剩余。将阶的值加 2，表示找到两个相关的点。
（5） 循环结束后，返回计算得到的阶的值。
通过遍历椭圆曲线上所有可能的 x 坐标值，计算对应的 y 坐标，并根据结果判断点是否位于椭圆曲线上的特殊情况或满足 Legendre 符号等于 1 的情况。通过累加特殊点和满足条件的点的个数，计算得到椭圆曲线的阶。

## 8. legendre 符号

勒让德符号用于判断给定的整数是否是一个模为奇素数的二次剩余。
该函数接受两个参数：a 和 p，其中 p 是一个奇素数，而 a 是与 p 互质的整

数。函数返回一个整数值，表示勒让德符号的结果。

（1） 如果 a 小于 0，代码会通过递归调用 Legendre 函数来计算-a 的勒让德符号，并根据-1 的勒让德符号的性质来计算结果。

（2） 对于非负的 a，代码将 a 对 p 取模。

（3） 如果 a 等于 1，那么 a 是 p 的一个二次剩余，返回 1。

（4） 如果 a 等于 2，根据 p 除以 8 的余数来判断 a 是否是 p 的一个二次剩余，返回相应的值。

（5） 如果 a 不等于 1 或 2，则对 a 进行素数分解。

（6） 在素数分解的循环中，代码遍历从 2 开始的所有素数，对 a 进行素数分解，判断每个素因子的指数是否是奇数。

（7） 如果某个素因子的指数是奇数，代码将根据素因子的大小和 p 的值来计算素因子的勒让德符号，并根据勒让德符号的性质更新结果。

（8） 最后，代码返回计算得到的结果。

## 9. 对在椭圆曲线$E_p(a,b)$，计算所有点

（1） 初始化变量和输出：

- 定义一个整数变量 sum，用于记录点的数量。
- 输出字符串"O "，表示椭圆曲线上的无穷远点。

（2） 迭代 x 坐标：

- 使用一个 for 循环从 0 到 p-1 迭代变量 x。

（3） 计算临时变量 tmp：

- 计算临时变量 tmp，其值等于(x^3 + ax + b + p) % p。

（4） 处理特殊情况（tmp == 0）：

- 如果 tmp 等于 0，表示在椭圆曲线上找到了一个点。输出形式为" (x, 0)"。
- 增加 sum 的值。

（5） 处理一般情况（Legendre(tmp, p) == 1）：

- 如果 tmp 不等于 0，并且 tmp 的 Legendre 符号等于 1，则存在两个点满足椭圆曲线方程。
- 使用一个内部的 for 循环从 1 到 p-1 迭代变量 y，寻找满足方程的 y 值。

（6） 输出解和增加 sum：

- 如果(y^2 - tmp) % p 等于 0，表示找到了一个满足方程的 y 值。
- 输出形式为" (x, y)"，其中 x 是当前的 x 坐标，y 是满足方程的 y 坐标。
- 增加 sum 的值。
- 输出形式为" (x, p-y)"，表示另一个满足方程的点。
- 再次增加 sum 的值。

（7） 输出结果和返回：

- 返回 sum，表示椭圆曲线上的点的数量。

运行示例：

```
请输入椭圆曲线的参数 p: 19
请输入椭圆曲线的参数 a: 3
请输入椭圆曲线的参数 b: 7

判断所给参数是否能构成一个椭圆曲线
E_19(3,7) is Elliptic Curve

判断给出的点是否在给定的椭圆曲线上
输入 x: 1
输入 y: 7
P(1,7) is on E_19(3,7)

计算给定的两点相加
输入 x1: 1
输入 y1: 7
输入 x2: 3
输入 y2: 9
结果是(16,16)

计算给出的点的倍加
输入 x: 1
输入 y: 7
输入倍数: 7
结果是(15,11)

计算给出的点的阶
输入 x: 1
输入 y: 7
P(1,7)的阶是11

计算给出的椭圆曲线的阶
E_19(3,7)的阶是22

列出给出的椭圆曲线上的所有点
O (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) (15,1
1) (16,3) (16,16)
Total:22
```