椭圆曲线编程练习报告

姓名：陆皓喆　　学号：2211044　　班级：信息安全

一、源码部分

```
#include<bits/stdc++.h>
using namespace std;
class Point {//定义一个类，用于表示我们在椭圆曲线上的点
public:
    int x;//给出我们的横坐标
    int y;//给出我们的纵坐标
    bool is_Infinity_Point;//判断是否为无穷远点
    Point(int x = 0, int y = 0, bool is_Infinity_Point = false);
    friend ostream& operator<< (ostream& out, const Point& p);
    bool operator ==(const Point& p);
    void Output(ostream& out) const;

};

class Elliptic_Curve {//定义椭圆曲线的类，用于实现我们的各项功能
private:
    int p;
    int a;
    int 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

};

//开始编写我们的各项功能
Point::Point(int x, int y, bool is_Infinity_Point)
{
    this->x = x;
    this->y = y;
    this->is_Infinity_Point = is_Infinity_Point;
}

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

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

```
{
    p.Output(out);
    return out;
}

void Point::Output(ostream& out) const
{
    if (is_Infinity_Point) {
        cout << 'O';//输出无穷远点
    }
    else {
        cout << '(' << x << ',' << y << ')';//输出我们正常的坐标
    }
}

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;
}

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::Is_Inverse(const Point& p1, const Point& p2) //判断两个点是否互
逆
```

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

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) //判断p点是否在椭圆曲线上
{
    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)  //进行点加运算
{
    if (p1.is_Infinity_Point)
    {
        return p2;
    }
    else if (p2.is_Infinity_Point)
    {
        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 p, int k) //对点p进行k倍加
{
    Point ret(0, 0, true);
    while (k)
    {
        if (k & 1)
```

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

int Elliptic_Curve::Ord_Of_Point(const Point& pt) //计算点p的阶
{
    int ret = 1;
    Point tmp = pt;
    while (!tmp.is_Infinity_Point)
    {
        tmp = Add(tmp, pt);
        ++ret;
    }
    return ret;
}

int Elliptic_Curve::Ord_Of_Elliptic_Curve() //计算此椭圆曲线的阶#E
{
    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() //展示出椭圆曲线上的所有点
{
    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;
}

//开始测试
#define Elliptic_Curve_EC "E_" << p << "(" << a << ',' << b << ")"
#define Point_P "P(" << x << "," << y << ")"

int main()
{
    cout << "1.judge the p,a,b whether they can be the Elliptic Curve" << endl;
    int p, a, b;
    cout << "p=";
    cin >> p;
    cout << "a=";
    cin >> a;
    cout << "b=";
    cin >> b;

    Elliptic_Curve ec(p, a, b);
    int x, y;
    cout << endl;
    cout << Elliptic_Curve_EC << " is ";
    if (!ec.Test_Is_Elliptic_Curve())
    {
        cout << "not ";

    }
    cout << "Elliptic_Curve" << endl;

    cout << endl;
    cout << "2.judge the Point whether it is on the Elliptic Curve" << endl;
    cout << "x=";
    cin >> x;
    cout << "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.get the two Points' add result " << endl;
    int x1, y1, x2, y2;
    cout << "x1=";
    cin >> x1;
    cout << "y1=";
```

```
    cin >> y1;
    cout << "x2=";
    cin >> x2;
    cout << "y2=";
    cin >> y2;
    cout << "the add result is:" << ec.Add({ x1, y1 }, { x2, y2 }) << endl;

    cout << endl;
    cout << "4.get the Point's K times add result" << endl;
    cout << "x= ";
    cin >> x;
    cout << "y=";
    cin >> y;
    int times;
    cout << "time=";
    cin >> times;
    cout << "the add result is:" << ec.Add_K_Times({ x, y }, times) << endl;

    cout << endl;
    cout << "5.get the ord of the Point" << endl;
    cout << "x=";
    cin >> x;
    cout << "y=";
    cin >> y;
    cout << Point_P << "the ord of the Point is:" << ec.Ord_Of_Point({ x, y }) <<
endl;

    cout << endl;
    cout << "6.get the Elliptic Curve's ord " << endl;
    cout << Elliptic_Curve_EC << "the ord of the Elliptic Curve is:" <<
ec.Ord_Of_Elliptic_Curve() << endl;

    cout << endl;
    cout << "7.list all of the Points on the Elliptic Curve" << endl;
    cout << ec.Show_All_Points();

    return 0;
    system("pause");
}
```

二、说明部分

首先，我们定义了两个类，分别是$Point$和$Elliptic_Curve$，第一个是我们规定的点，第二个是我们定义的椭圆曲线，在其中，我们重载了一些输出的运算符，定义了基本的横纵坐标和模数等。

在$Point$类中，我们还定义了一个布尔类型的值$is_Infinity_Point$，用于判断这个点是不是在椭圆曲线上；指定特殊情况：若该点的坐标为$(0,0)$，就直接输出不是无穷远点，防止在后面的误判。

在$Elliptic_Curve$类中，我们首先定义了横纵坐标和我们的模数a，b，p，然后定义了以下的一些函数：

- $Is\_Inverse$：判断两个点是否互逆
- $Test\_Is\_Elliptic\_Curve$：检查当前参数是否能构成椭圆曲线

- $Is\_On\_Elliptic\_Curve$：检查当前的点是否在椭圆曲线上
- Add：实现点与点的点加操作
- Add_K_Times：实现点的倍加操作
- Ord_Of_Point：计算点p的阶
- $Ord\_Of\_Elliptic\_Curve$：计算此椭圆曲线的阶#E
- $Show\_All\_Points$：展示出椭圆曲线上的所有点

在我们的$main$函数中，我们对我们的功能分别进行了测试：

- $judge\ the\ p,a,b\ whether\ they\ can\ be\ the\ Elliptic\ Curve$：我们通过输入我们的三个参数来判断是否构成一个椭圆曲线
- $judge\ the\ Point\ whether\ it\ is\ on\ the\ Elliptic\ Curve$：通过输入点的横纵坐标来判断是否在我们的椭圆曲线上
- $get\ the\ two\ Points'\ add\ result$：通过课本上给出的公式来计算我们的点加结果
- $get\ the\ Point's\ K\ times\ add\ result$：通过课本上给出的公式来计算我们的倍加操作的结果
- $get\ the\ ord\ of\ the\ Point$：通过使用我们的课本上的算法来计算我们的点的阶
- $get\ the\ Elliptic\ Curve's\ ord$：通过二次剩余的知识来求解遍历我们的结果，再加上一个无穷远点就可以得到我们的最后的数量
- $list\ all\ of\ the\ Points\ on\ the\ Elliptic\ Curve$：根据上面一个问题的结果，来进行我们所有的点的输出

其中有许多的函数，我直接调用了前几次编程练习中编写的函数，这边就不再展示内容了。

三、运行实例

```
1.judge the p,a,b whether they can be the Elliptic Curve
p=19
a=3
b=7

E_19(3,7) is Elliptic_Curve

2.judge the Point whether it is on the Elliptic Curve
x=3
y=9
P(3,9) is on E_19(3,7)

3.get the two Points' add result
x1=1
y1=7
x2=3
y2=9
the add result is:(16,16)

4.get the Point's K times add result
x= 1
y=7
time=7
the add result is:(15,11)

5.get the ord of the Point
x=1
y=7
P(1,7)the ord of the Point is:11

6.get the Elliptic Curve's ord
E_19(3,7)the ord of the Elliptic Curve is:22

7.list all of the Points on the Elliptic Curve
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)  (15,11)  (16,3)  (16,16)
22
E:\学学学\本科\大二下\信息安全数学基础\homework\C++实现\homework8\x64\Debug\homework8.exe (进程 24600)已退出，代码为 0。
按任意键关闭此窗口. . .
```

如上所示，是我们本次的实验运行实例，我们测试了我们的功能，可以看到，我们很好地完成了所有的任务，实验成功！