# 第 8 次编程练习报告

姓名：申宗尚　　学号：2213924　　班级：信安班

## 一、编程练习 1——实现基本的 Zp 上的椭圆曲线 Ep(a,b)上的计算，具体要求如下：

1. 功能要求：
   - 给定参数 $p,a,b$，判断 $E_p(a,b)$ 是否为椭圆曲线；
   - 判断给定的点 $P,Q$ 是否在椭圆曲线 $E_p(a,b)$ 上；
   - 对在椭圆曲线 $E_p(a,b)$ 上的两点 $P,Q$，计算 $P+Q$；
   - 对在椭圆曲线 $E_p(a,b)$ 上的点 $P$，使用倍加-和算法计算 $mP$；
   - 对在椭圆曲线 $E_p(a,b)$ 上的点 $P$，计算阶$\mathrm{ord}(P)$；
   - 对在椭圆曲线 $E_p(a,b)$，计算阶$\#E$；
   - 对在椭圆曲线 $E_p(a,b)$，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 << ")"

   class Point{
   public:
       int x,y;
       bool isINF;//是否为无穷远点
       Point(int x = 0, int y = 0, bool isINF = 0);
       friend ostream& operator<< (ostream& out, const Point& P);//重
   定义输出
       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

};

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;
}

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 (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::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;
}
```

```cpp
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.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)
        {
            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{
            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.isINF)
{
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;
    }

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 ;
    if (!ec.Test_Is_Elliptic_Curve())
    {
        cout << "不构成 ";
        return 0; }
    cout << "构成 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：**

代码中主要实现了两个类：点 Point 类和椭圆曲线 Elliptic_Curve 类。

在点类中，记录三个值 x，y 和 isINF，分别表示点的 x 坐标，y 坐标和是否为无穷远点，然后多态定义了点类的输出、相等判定。

在椭圆曲线类中，通过不同的函数，实现了种种功能，下面挨个介绍：

```
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( a: -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)
            {
```

首先，通过二次互反律进行 Legendre 符号的计算，这可以计算是否为二次剩余，后面阶的计算等等函数都可以在此利用。

```
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; }
```

然后，写一个次方 pow 函数

```
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;
}
```

通过之前的编程作业代码，拿出来使用扩展欧几里得定理的求逆元部分，实现 get_Inverse 函数。

接下来，进行正式的功能函数介绍：

1、判断是否互逆，只需要判断如下通过公式判断。

```
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;
}
```

2、3 判断是否能构成椭圆曲线、是否在曲线上，也是类似的，使用公式

```
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( x: a,  n: 3) * 4 + pow( x: b,  n: 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( x: pt.y,  n: 2) - (pow(pt.x,  n: 3) + a * pt.x + b);
return tmp % p == 0;
}
```

4、点加，先进行无穷远点的讨论，如果一个是无穷远点，最后的值直接等于另一个，如果两个点是互逆的点，则最后的值就是无穷远点。如果都不是特殊情况，就通过点加\倍加的两种公式进行讨论，然后用公式获得最后的结果。

```
Point Elliptic_Curve::Add(const Point& p1, const Point& p2){ //进行点加运算
    if (p1.isINF){
        return p2;
    }
    else if (p2.isINF){
        return p1;
    }
    else if (Is_Inverse(p1, p2)){
        return { x: 0,  y: 0,  isINF: true };
    }
    else
    {
        if((p1.x-p2.x) % p == 0)
        {
            int k = ((3 * p1.x * p1.x + a) * Get_Inverse( a: 2 * p1.y,  m: 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 { x: x3,  y: y3 };
        }
        else{
            int k = ((p2.y - p1.y) * Get_Inverse( a: p2.x - p1.x,  m: 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 { x: x3,  y: y3 };
        }
    }
    }
```

5、k 倍加，使用循环，Add k 次。

```
Point Elliptic_Curve::Add_K_Times(Point p, int k){ //对点p进行k倍加
    Point ret( x: 0,  y: 0,  isINF: true);
    while (k)
    {
    if (k & 1) {
        ret = Add( p1: ret,  p2: p);
    }
    p = Add( p1: p,  p2: p);
    k >>= 1; }
    return ret;
}
```

6、计算点 p 的阶：对 p 一直进行倍加，知道结果为 INF，返回加法次数即为 p 点的阶：

```
int Elliptic_Curve::Ord_Of_Point(const Point& pt){ //计算点p的阶
    int ret = 1;
Point tmp = pt;
while (!tmp.isINF)
{
tmp = Add( p1: tmp,  p2: pt);
++ret; }
return ret;
}
```

7、计算椭圆曲线的阶：初始化变量 ret 为 1，存储椭圆曲线的阶，遍历所有可能的 x 值，对于每个 x，计算椭圆曲线上的值，如果是 0，则 ret+1，不是的话判断是不是 p 的二次剩余（通过 Legendre 符号），如果是的话，说明 y^2 = tmp 有解，因此有两个点，ret+2，从而最终计算出阶。

```
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( a: tmp, p) == 1){
    ret += 2; }
    }
    return ret;
}
```

8、展示所有点：遍历找解。

```cpp
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( a: 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;
    }
```

➢ 运行示例：

```
/Users/kkkai/CLionProjects/xinanshuji8/cmake-build-debug/xinanshuji8
请输入椭圆曲线的参数 p: 11
请输入椭圆曲线的参数 a: 0
请输入椭圆曲线的参数 b: 0

1.判断所给参数是否能构成一个椭圆曲线
E_11(0,0) is not
进程已结束，退出代码为 0
```

```
/Users/kkkai/CLionProjects/xinanshuji8/cmake-build-debug/xinanshuji8
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

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

2.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)

3.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)

4.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)
```

```
5.计算给出的点的阶
输入 x: 1
输入 y: 7
P(1,7)的阶是11

6.计算给出的椭圆曲线的阶
E_19(3,7)的阶是22

7.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) (15,11) (16,3) (
22
进程已结束，退出代码为 0
```