

椭圆曲线编程练习报告

一、源码部分

- Elliptic_Curve_Cryptography.h

```
1  #pragma once
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  class Point
6  {
7  public:
8      int x, y;
9      bool isINF; //是否是无穷远点
10     Point(int x = 0, int y = 0, bool isINF = false);
11     friend ostream& operator<< (ostream& out, const Point& p);
12     bool operator ==(const Point& p);
13     void Output(ostream& out) const;
14 };
15
16 class Elliptic_Curve
17 {
18 private:
19     int p;
20     int a, b;
21 public:
22     Elliptic_Curve(int p, int a, int b);
23     bool Is_Inverse(const Point& p1, const Point& p2); //判断两个点是否互逆
24     bool Test_Is_Elliptic_Curve(); //检查当前参数是否能构成椭圆曲线
25     bool Is_On_Elliptic_Curve(const Point& p); //判断p点是否在椭圆曲线上
26     Point Add(const Point& p1, const Point& p2); //进行点加运算
27     Point Add_K_Times(Point p, int k); //对点p进行k倍加
28     int Ord_Of_Point(const Point& p); //计算点p的阶
29     int Ord_Of_Elliptic_Curve(); //计算此椭圆曲线的阶#E
30     int Show_All_Points(); //展示出椭圆曲线上的所有点
31 };
```

- Elliptic_Curve_Cryptography.cpp

```
1  #include "Elliptic_Curve_Cryptography.h"
2
3  int Legendre(int a, int p) //p是奇素数, (a, p) = 1
4  {
5      if (a < 0)
6      {
7          if (a == -1)
8          {
9              return p % 4 == 1 ? 1 : -1;
```

```

10     }
11     return Legendre(-1, p) * Legendre(-a, p);
12 }
13 a %= p;
14 if (a == 1)
15 {
16     return 1;
17 }
18 else if (a == 2)
19 {
20     if (p % 8 == 1 || p % 8 == 7) return 1;
21     else return -1;
22 }
23 // 下面将a进行素数分解
24 int prime = 2;
25 int ret = 1;
26 while (a > 1)
27 {
28     int power = 0;
29     while (a % prime == 0)
30     {
31         power++;
32         a /= prime;
33     }
34     if (power % 2 == 1)
35     {
36         if (prime <= 2)
37         {
38             return Legendre(prime, p);
39         }
40         else
41         {
42             if (((prime - 1) * (p - 1) / 4) % 2 == 1)
43             {
44                 ret = -ret;
45             }
46             ret *= Legendre(p, prime);
47         }
48     }
49     prime++;
50 }
51 return ret;
52 }
53
54 int pow(int x, int n) //x的n次方
55 {
56     int ret = 1;
57     while (n)
58     {
59         if (n & 1)
60         {
61             ret *= x;

```

```

62     }
63     x *= x;
64     n >>= 1;
65 }
66 return ret;
67 }
68
69 int Get_Inverse(int a, int m) //在 (a, m) = 1 的条件下, 求a模m的乘法逆元
70 {
71     a = (a + m) % m;
72     int s0 = 1, s1 = 0;
73     int r0 = a, r1 = m;
74     while (1)
75     {
76         int q = r0 / r1;
77         int tmp = r1;
78         r1 = r0 % r1;
79         r0 = tmp;
80         if (r1 == 0)
81         {
82             break;
83         }
84         tmp = s1;
85         s1 = s0 - s1 * q;
86         s0 = tmp;
87     }
88     return (s1 + m) % m;
89 }
90
91 Point::Point(int x, int y, bool isINF)
92 {
93     this->x = x;
94     this->y = y;
95     this->isINF = isINF;
96 }
97
98 bool Point::operator == (const Point& p)
99 {
100     return x == p.x && y == p.y;
101 }
102
103 ostream& operator<< (ostream& out, const Point& p)
104 {
105     p.Output(out);
106     return out;
107 }
108
109 void Point::Output(ostream& out) const
110 {
111     if (isINF) cout << '0';
112     else cout << '(' << x << ',' << y << ')';
113 }

```

```

114
115 Elliptic_Curve::Elliptic_Curve(int p, int a, int b) //椭圆曲线构造函数
116 {
117     this->p = p;
118     this->a = a;
119     this->b = b;
120 }
121
122 bool Elliptic_Curve::Is_Inverse(const Point& p1, const Point& p2) //判断两个点是否互逆
123 {
124     return (p1.x - p2.x) % p == 0 && (p1.y + p2.y) % p == 0;
125 }
126
127 bool Elliptic_Curve::Test_Is_Elliptic_Curve() //检查当前参数是否能构成椭圆曲线
128 {
129     int tmp = pow(a, 3) * 4 + pow(b, 2) * 27;
130     return tmp % p != 0;
131 }
132
133 bool Elliptic_Curve::Is_On_Elliptic_Curve(const Point& p) //判断p点是否在椭圆曲线上
134 {
135     int tmp = pow(pt.y, 2) - (pow(pt.x, 3) + a * pt.x + b);
136     return tmp % p == 0;
137 }
138
139 Point Elliptic_Curve::Add(const Point& p1, const Point& p2) //进行点加运算
140 {
141     if (p1.isINF)
142     {
143         return p2;
144     }
145     else if (p2.isINF)
146     {
147         return p1;
148     }
149     else if (Is_Inverse(p1, p2))
150     {
151         return { 0, 0, true };
152     }
153     else
154     {
155         if ((p1.x - p2.x) % p == 0) //倍加公式
156         {
157             int k = ((3 * p1.x * p1.x + a) * Get_Inverse(2 * p1.y, p) % p + p) % p;
158             int x3 = ((k * k - 2 * p1.x) % p + p) % p;
159             int y3 = ((k * (p1.x - x3) - p1.y) % p + p) % p;
160             return { x3, y3 };
161         }
162         else //点加公式
163         {
164             int k = ((p2.y - p1.y) * Get_Inverse(p2.x - p1.x, p) % p + p) % p;
165             int x3 = ((k * k - p1.x - p2.x) % p + p) % p;

```

```

166         int y3 = ((k * (p1.x - x3) - p1.y) % p + p) % p;
167         return { x3, y3 };
168     }
169 }
170 }
171
172 Point Elliptic_Curve::Add_K_Times(Point p, int k) //对点p进行k倍加
173 {
174     Point ret(0, 0, true);
175     while (k)
176     {
177         if (k & 1)
178         {
179             ret = Add(ret, p);
180         }
181         p = Add(p, p);
182         k >>= 1;
183     }
184     return ret;
185 }
186
187 int Elliptic_Curve::Ord_Of_Point(const Point& p) //计算点p的阶
188 {
189     int ret = 1;
190     Point tmp = pt;
191     while (!tmp.isINF)
192     {
193         tmp = Add(tmp, pt);
194         ++ret;
195     }
196     return ret;
197 }
198
199 int Elliptic_Curve::Ord_Of_Elliptic_Curve() //计算此椭圆曲线的阶#E
200 {
201     int ret = 1;
202     for (int x = 0; x < p; ++x)
203     {
204         int tmp = (x * x * x + a * x + b + p) % p;
205         if (tmp == 0)
206         {
207             ret += 1;
208         }
209         else if (Legendre(tmp, p) == 1)
210         {
211             ret += 2;
212         }
213     }
214     return ret;
215 }
216
217 int Elliptic_Curve::Show_All_Points() //展示出椭圆曲线上的所有点

```

```

218 {
219     cout << "0 ";
220     int sum = 1;
221     for (int x = 0; x < p; ++x)
222     {
223         int tmp = (x * x * x + a * x + b + p) % p;
224         if (tmp == 0)
225         {
226             cout << " (" << x << ', ' << "0) ";
227             sum++;
228         }
229         else if (Legendre(tmp, p) == 1) //贡献两个点
230         {
231             for (int y = 1; y < p; ++y) //从1遍历到p-1, 寻找解
232             {
233                 if ((y * y - tmp) % p == 0)
234                 {
235                     cout << " (" << x << ', ' << y << ") ";
236                     sum++;
237                     cout << " (" << x << ', ' << p - y << ") ";
238                     sum++;
239                     break;
240                 }
241             }
242         }
243     }
244     cout << endl;
245     return sum;
246 }
247
248

```

- Experiment.cpp

```

1  #include "Elliptic_Curve_Cryptography.h"
2  #define Elliptic_Curve_EC "E_" << p << "(" << a << ', ' << b << ")"
3  #define Point_P "P(" << x << ", " << y << ")"
4
5  int main()
6  {
7      int p, a, b;
8      cout << "请输入椭圆曲线的参数 p: ";
9      cin >> p;
10     cout << "请输入椭圆曲线的参数 a: ";
11     cin >> a;
12     cout << "请输入椭圆曲线的参数 b: ";
13     cin >> b;
14
15     Elliptic_Curve ec(p, a, b);
16     int x, y;
17     cout << endl;

```

```

18     cout << "1.判断所给参数是否能构成一个椭圆曲线" << endl;
19     cout << Elliptic_Curve_EC << " is ";
20     if (!ec.Test_Is_Elliptic_Curve())
21     {
22         cout << "not ";
23         return 0;
24     }
25     cout << "Elliptic_Curve" << endl;
26
27     cout << endl;
28     cout << "2.判断给出的点是否在给定的椭圆曲线上" << endl;
29     cout << "输入 x: ";
30     cin >> x;
31     cout << "输入 y: ";
32     cin >> y;
33     cout << Point_P " is ";
34     if (!ec.Is_On_Elliptic_Curve(Point(x, y))) cout << "not ";
35     cout << "on " << Elliptic_Curve_EC << endl;
36
37     cout << endl;
38     cout << "3.计算给定的两点相加" << endl;
39     int x1, y1, x2, y2;
40     cout << "输入 x1: ";
41     cin >> x1;
42     cout << "输入 y1: ";
43     cin >> y1;
44     cout << "输入 x2: ";
45     cin >> x2;
46     cout << "输入 y2: ";
47     cin >> y2;
48     cout << "结果是" << ec.Add({ x1, y1 }, { x2, y2 }) << endl;
49
50     cout << endl;
51     cout << "4.计算给出的点的倍加" << endl;
52     cout << "输入 x: ";
53     cin >> x;
54     cout << "输入 y: ";
55     cin >> y;
56     int times;
57     cout << "输入倍数: ";
58     cin >> times;
59     cout << "结果是" << ec.Add_K_Times({ x, y }, times) << endl;
60
61     cout << endl;
62     cout << "5.计算给出的点的阶" << endl;
63     cout << "输入 x: ";
64     cin >> x;
65     cout << "输入 y: ";
66     cin >> y;
67     cout << Point_P << "的阶是" << ec.Ord_Of_Point({ x, y }) << endl;
68
69     cout << endl;

```

```

70     cout << "6.计算给出的椭圆曲线的阶" << endl;
71     cout << Elliptic_Curve_EC << "的阶是" << ec.Ord_Of_Elliptic_Curve() << endl;
72
73     cout << endl;
74     cout << "7.列出给出的椭圆曲线上的所有点" << endl;
75     cout << ec.Show_All_Points();
76
77     return 0;
78 }

```

二、说明部分

定义了两个类，`Point` 和 `Elliptic_Curve`，并包含了必要的标准库和 `ostream` 类的头文件。

`Point` 类表示椭圆曲线上的一个点。它具有成员变量 `x` 和 `y`，表示点的坐标，以及一个布尔变量 `isINF`，表示该点是否为无穷远点。该类还包括一个友元类 `elliptic_curve` 和重载的 `operator<<`，用于输出点的信息。

`Elliptic_Curve` 类表示一个椭圆曲线。它具有私有成员变量 `p`、`a` 和 `b`，表示椭圆曲线方程的参数。该类包括各种成员函数，用于在椭圆曲线上执行操作，例如测试参数是否构成有效的椭圆曲线，检查点是否在曲线上，点加法、点倍加法，计算点的阶和曲线的阶，以及显示曲线上的所有点。该类还包括辅助函数，如 `Legendre` 和 `Get_Inverse`，用于计算Legendre符号和模素数的乘法逆元。

`main` 函数提供了一个命令行界面，用于与椭圆曲线交互并执行各种操作，例如检查给定的参数是否构成有效的椭圆曲线，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。

代码使用了一些特定于椭圆曲线密码学的数学概念和操作，如Legendre符号和点加法公式。

三、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

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: 3
输入 y: 9
P(3,9) 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
0 (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

```