第一题的主要部分在于把转换为二进制形式，然后就可以利用n的二进制形式进行扩展欧几里得算法的各项运算，直到达到最终的结果。

#include<iostream>

using namespace std;

int Square\_Multi(int a, int n, int m) {

int c = 1;

int binary[100];

int k = 0; //length of the number n

while (n) {

binary[k] = n % 2;

n /= 2;

k++;

}

while (k) {

k--; //current index of binary\_array

c = (c \* c) % m;

if (binary[k])

c = (c \* a) % m;

}

return c;

}

int main() {

cout << "Calulate a^n(mod m)...\nPlease input: ";

cout << "\n a=";

int a; cin >> a;

cout << " n=";

int n; cin >> n;

cout << " m=";

int m; cin >> m;

int r = Square\_Multi(a, n, m);

cout << a << "^" << n << "(mod " << m << ")=" << r;

}

第二题的主要部分是递归计算每一层的r,q,s,t的值，然后把当r=0的时候的s和t的值输出即可

#include<iostream>

using namespace std;

int Square\_Multi(int a, int n, int m) {

int c = 1;

int binary[100];

int k = 0; //length of the number n

while (n) {

binary[k] = n % 2;

n /= 2;

k++;

}

while (k) {

k--; //current index of binary\_array

c = (c \* c) % m;

if (binary[k])

c = (c \* a) % m;

}

return c;

}

void Get\_Inverse(int a, int b) {

int r0 = a, r1 = b;

//si = si−2 − qi−1si−1, ti = ti−2 − qi−1ti−1, 其中 qi = ri−1/r

int s\_2 = 1, s\_1 = 0, t\_2 = 0, t\_1 = 1;

int s, q, t, r;

while (r0 % r1) {

q = r0 / r1;

s = s\_2 - q \* s\_1;

t = t\_2 - q \* t\_1;

s\_2 = s\_1;

s\_1 = s;

t\_2 = t\_1;

t\_1 = t;

r = r0 % r1;

r0 = r1;

r1 = r;

}

cout << "gcb(a,b)=" << r1 << endl;

cout << "lcm(a,b)=" << a \* b / r1 << endl;

cout << "a^(-1)=" << s << "(mod " << b << ")\n";

cout << "b^(-1)=" << t << "(mod " << a << ")\n";

}

int main()

{

cout << "a=";

int x; cin >> x;

cout << "b=";

int y; cin >> y;

Get\_Inverse(x, y);

}