第2次编程报告

姓名：王雨萌　　学号：2311819　　专业：信息安全

编程练习一 ——实现平方乘算法

- **源代码部分**

```cpp
#include <iostream>

int yushu = 1;
int base = 1;

long long binfastpow(int a, int n, int m)
{
    long long yushu = 1;
    long long base = a % m;
    while (n > 0)
    {
        if (n % 2 == 1)
        {                                   // 如果 n 是奇数
            yushu = (yushu * base) % m; // 结果乘以当前的 base
        }
        base = (base * base) % m; // 更新 base 为 base 的平方
        n /= 2;
    }
    return yushu;
}

using namespace std;

int main()
{
    cout << "Calculate a^n(mod m)..." << endl
         << "Please input:" << endl;
    int a = 0, n = 0, m = 0;
    cout << "  a = ";
    cin >> a;
    cout << "  n = ";
    cin >> n;
    cout << "  m = ";
    cin >> m;

```

```
    base = (base * base) % m;
    cout << a << "^" << n << "(mod" << m << ") = " << binfastpow(a, n, m);
}
```

- **算法的数学与逻辑原理**

该算法适用于解决大整数模幂的问题。

$$a^n(mod m) \quad (1)$$

我们先将指数 n 展开为二进制的表达式。

$$n = b_k \cdot 2^k + b_{k-1} \cdot 2^{k-1} + \ldots + b_1 \cdot 2^1 + b_0 \cdot 2^0$$

结合同余的性质：

$$a \ \equiv \ b(mod\, m) \ \Rightarrow \ ac \ \equiv bc\ (mod \ \ m)\ \&\ a^n \ = \ (a^{\frac{n}{2}})^2(mod\, m) \quad (2)$$

然后我们带回原式（1）：

$$a^n \ \equiv \ a^{b_k \cdot 2^k + b_{k-1} \cdot 2^{k-1} + \ldots + b_1 \cdot 2^1 + b_0 \cdot 2^0} \ \equiv (((((a^{b_n})^2 a^{b_{n-1}})^2) a^{b_{n-2}})^2 \ldots a^{b_0})^2\ (modm)$$

而我们知道在二进制中， b_k 的取值 {0,1}

那么在编程实现中，我们可以分类讨论实现算法：

1. 我们先通过连续整除和对2取模运算，得到每一个 b_k
2. 如果 b_k 是1，那么说明在模幂运算中，根据（2）式，我们用 `yushu = (yushu * base) % m` 直接更新所求解的余数，如果 b_k 是0,我们直接跳过，因为不需要更新。
3. 截止条件，我们取完所有的 b_k

- **运行示例**

我们设计了简单、复杂的测试用例：运行结果如下

```
1 arguments:
argv[0] = '/Users/wangyumeng/000 - 学习/050 - 竞赛/算法code文件/.vscode/平方乘法
'
Calculate a^n(mod m)...
Please input:
  a = 3
  n = 125
  m = 73
3^125(mod73) = 24Process exited with status 0
```

复杂的测试用例

```
argv[0] = '/Users/wangyumeng/000 - 学习/050 - 竞赛/算法code文件/.vscode/平方乘法
'
Calculate a^n(mod m)...
Please input:
  a = 3
  n = 3
  m = 8
3^3(mod8) = 3Process exited with status 0
```

简单的测试用例

编程练习二——实现扩展欧几里得算法

- **源代码部分**

```
#include <iostream>

```

```
long long Format(int m, long long num)
{
    if (num < 0)
    {
        return num + m;
    }
    else
    {
        return num;
    }
}

long long Gcd(long long a, long long b)
{
    if (a < b) // 令a一定大于等于b
    {
        int temp = a;
        a = b;
        b = temp;
    }
    int r = a % b;
    while (r != 0)
    {
        a = b;
        b = r;
        r = a % b;
    }
    return b;
}

long long Lcm(long long a, long long b)
{
    return (a * b) / Gcd(a, b);
}

int s0 = 1, s1 = 0, s;
int t0 = 0, t1 = 1, t;

long long Ep(int a, int m) // 详细推导一下这个整个过程
{
    if (a > m)
    {
        long long q = 1; // 初始化，使之进入

        while (q != 0) // 为什么在这里截止：当r_{n+1} = 0时候，q_{n} = r_{n-1}/rn, 正好计算完成
        {
```

```
            q = a / m; // q_i = r_{i-1}/r_i
            long long temp = m;
            m = a % temp;
            a = temp;

            s = s0 - s1 * q; // q -> q_i
            s0 = s1;
            s1 = s;
        }
        return s;
    }
    else
    {
        int temp = a;
        a = m;
        m = temp;

        long long q = 1; // 初始化，使之进入

        while (q != 0) // 为什么在这里截止：当r_{n+1} = 0时候，q_{n} = r_{n-1}/rn，正好计算完成
        {
            q = a / m; // q_i = r_{i-1}/r_i
            long long temp = m;
            m = a % temp;
            a = temp;

            t = t0 - t1 * q; // q -> q_i
            t0 = t1;
            t1 = t;
        }
        return t;
    }
}

using namespace std;

int main()
{
    int a = 0, b = 0;
    cout << "a = ";
    cin >> a;
    cout << "b = ";
    cin >> b;
    cout << "Gcd(a,b) = " << Gcd(a, b) << endl;
    cout << "Lcm(a,b) = " << Lcm(a, b) << endl;
    cout << "a^(-1) = " << Format(b, Ep(a, b)) << "(mod " << b << ")" << endl;
```

```
    cout << "b^(-1) = " << Format(a, Ep(b, a)) << "(mod " << a << ")" << endl;
}
```

- **算法的数学以及逻辑**

最小公因数和最大公倍数

我们使用欧几里得算法计算最小公因数

$$a \ = bq \ + r_0$$

$$b \ = q_1 r_0 \ + \ r_1$$

$$.......$$

$$r_{n-1} \ = q_{n+1} r_n , r_{n+1} \ = 0$$

结论是:

$$(a,b) \ = \ (b,r_0) \ = \ (r_0,r_1) \ = \ ... \ = \ (r_n,0) \ = \ r_n$$

我们再根据下面的式子计算最小公倍数

$$[a,b] \ = \ \frac{ab}{(a,b)}$$

扩展欧几里得算法部分

我参考了《信息安全数学基础》关于扩展欧几里得算法的二级结论

设有 $(r_0,r_1) \ = \ s_n r_0 \ + \ t_n r_1$ ，并且已经知道 $s_0 \ = \ 1, s_1 \ = \ 0; t_0 \ = \ 0, t_1 \ = \ 1;$

我们可以得到以下递推关系式:

$$s_i \ = \ s_{i-2} \ - \ q_{i-1} s_{i-1} \ \ , \ \ t_i \ = \ t_{i-2} \ - \ q_{i-1} t_{i-1} \ \ , \ \ q_i \ = \ r_{i-1}/r_i$$

然后我们有结论，

$$s_n \ = \ r_0{}^{-1}(mod \ \ r_1), \ \ t_n \ = \ r_1{}^{-1}(mod \ r_0)$$

- 运行结果

```
1 arguments:
argv[0] = '/Users/wangyumeng/000 - 学习/050 - 竞赛/算法code文件/.vscode/扩展欧几里得算法_副本'
a = 12345
b = 65432
Gcd(a,b) = 1
Lcm(a,b) = 807758040
a^(-1) = 63561(mod 65432)
b^(-1) = 353(mod 12345)
Process exited with status 0
```

测试结果