算法基础（二十一）：数学基础 - 数论6 - 中国剩余定理（扩展）

有这样一组数：`m1，m2，m3，m4....mk`两两互质，那么对于一个同余方程组：

$$\begin{cases} x \equiv a1(mod m1) \\ x \equiv a2(mod m2) \\ \dots \\ x \equiv ak(mod mk) \end{cases}$$

令$M = m1 * m2 * m3 * .... mk$，$M_i = \frac{M}{mi}$，$M_i^{-1}$表示$Mi(mod mi)$的逆也就是$M_i * M_i^{-1} = 1(mod mi)$

那么可以得到这个方程组的通解为：

$$x = a1 * M_1 * M_1^{-1} + a2 * M_2 * M_2^{-1} + \dots + ak * M_k * M_k^{-1}$$

将这个通解带入可以发现是正确的，这个就是中国剩余定理。

中国剩余定理的扩展

对于下面这个方程，我们求其最小正整数的解

$$\begin{cases} x \equiv m1(mod a1) \dots\dots 1 \\ x \equiv m2(mod a2) \dots\dots 2 \\ \dots \\ x \equiv mk(mod ak) \dots\dots k \end{cases}$$

我们如果单纯拿出前两个方程：

$$\begin{cases} x \equiv m1(mod a1) \\ x \equiv m2(mod a2) \end{cases}$$

也就是：

$$\begin{cases} x = k1 * a1 + m1 \\ x = k2 * a2 + m2 \end{cases}$$

将这两个方程联立可以得到：

$$k1 * a1 - k2 * a2 = m2 - m1 \dots\dots 3$$

在这里我们可以先根据扩展欧几里得算法就可以求得一组解：$k1$，$k2$，前提是$m2 - m1$是$gcd(a1, a2)$的倍数，令扩展欧几里得算法得到的结果为k_1^0, k_2^0，令$y = \frac{m2 - m1}{gcd(a1, a2)}$由此我们可以得到：

$$\begin{cases} k1 = y * k_1^0 \\ k2 = y * k_2^0 \end{cases}$$

于是对于3式来说，它的通解的形式可以写成：

$$\begin{cases} k1` = k1 + k * \frac{a2}{d} \\ k2` = k2 + k * \frac{a1}{d} \end{cases}$$

其中$d = gcd(a1, a2)$，$k1, k2$是任意一组解，这个式子带入3就可以得到证明，但是需要注意的是当我们根据扩展欧几里得算法求出来了$k1 = y * k_1^0, k2 = y * k_2^0$后，由于在题目中给的数据范围可能比较极限，中间结果可能会溢出，所以我们需要将$k1, k2$进行如下处理

$$\begin{cases} k1 = k1\% \frac{a2}{d} \\ k2 = k2\% \frac{a1}{d} \end{cases}$$

就可以将$k1, k2$变成最小正整数，哪怕原来的$k1, k2$是负数也不影响，因为得到的结果仍然满足上面的通解形式，即由于k的任意性：$k1' = k1 + k * \frac{a2}{d} = k1_{min} + k' * \frac{a2}{d}$，所以新的$k1_{min}, k2_{min}$也可以作为任意一组解来构成通解形式。注意扩展欧几里得算法求出的其实是$-k2$，但是$k1$的值与原来的一样，所以我们代入$k1$而不代入$k2$，当我们求出来了通解形式$k1` = k1 + k * \frac{a2}{d}$，将这个式子带入到1式中可以得到：

$$x = a1k1 + m1 + k\frac{a1a2}{d} = a1k1 + m1 + k[a1, a2]$$

令$a1k1 + m1 = x0, k[a1, a2] = ka$，$[a1, a2]$是$a1, a2$的最小公倍数，于是我们可以得到：

$$x = a1k1 + m1 + k[a1, a2] = x0 + ka......4$$

其中$x0, a$都是定值，k是任意的值与原来的两个方程无关，4式是由1，2推导出来的，所以满足$1, 2$的解一定满足4，那么我们解出了4，也就解出了$1, 2$，于是我们可以将其合并成4，由于4的形式与$1, 2$相同，所以我们可以用4与最上面的方程组中的其他方程进行合并，在代码中就是将`m1 = a1k1 + m1, a1a2/d = a1`，得到一个新的$x \equiv m1(moda1)$然后再与$x \equiv m3(moda3)$进行合并，一共合并$n-1$次可以最终得到`x = x0’ + k’a’`，也就是最后的$x \equiv m1(moda1) => x = k1 * a1 + m1$要求最小正整数解，直接用`x % a1`即可

代码实现

给定 $2n$ 个整数 $a_1, a_2, \ldots, a_n$ 和 $m_1, m_2, \ldots, m_n$，求一个最小的非负整数 x，满足 $\forall i \in [1, n], x \equiv m_i (mod\ a_i)$。

输入格式

第 1 行包含整数 n。

第 $2 \ldots n+1$ 行：每 $i+1$ 行包含两个整数 a_i 和 m_i，数之间用空格隔开。

输出格式

输出最小非负整数 x，如果 x 不存在，则输出 -1。
如果存在 x，则数据保证 x 一定在 64 位整数范围内。

数据范围

$1 \leq a_i \leq 2^{31} - 1$,
$0 \leq m_i < a_i$
$1 \leq n \leq 25$

输入样例：

```
2
8 7
11 9
```

输出样例：

```
31
```

```
#include<iostream>
#include<algorithm>
using namespace std;

typedef long long LL;

//扩展欧几里得算法计算a * x + b * y = gcd(a, b)中x, y的值
LL exgcd(LL a, LL b, LL &x, LL &y)
{
    if(!b)
    {
        //如果b为0，则返回一个x, y的值
        x = 1;
        y = 0;
        return a;
    }
    //递归到下一层计算本层的x,  y
    LL d = exgcd(b, a % b, y, x);
    //计算出了本层的x, y计算上一层递归的x, y
    y -= a / b * x;

    return d;
}

int main()
{
```

```cpp
    int n;
    cin >> n;
    bool has_ans = true;//用一个变量表示当前是否无解
    LL a1, m1;

    //先读入第一个方程
    cin >> a1 >> m1;
    //每次把一个新的方程合并到现有的方程中
    for(int i = 0; i < n - 1; i ++)
    {
        LL a2, m2;
        cin >> a2 >> m2;

        LL k1, k2;
        //先用扩展欧几里得算法求出两个方程k1, k2的值
        //以及得到a1, a2的 最小公约数，就是扩展欧几里得算法的返回值
        //扩展欧几里得算法是求k1* a1 - k2 * a2 = gcd(a1, a2)中k1, k2的值，一定可以得到解
        //但是这里是k1* a1 - k2 * a2 = m2 - m1，不一定有解
        LL d = exgcd(a1, a2, k1, k2);
        if((m2 - m1) % d)
        {
            //如果不为0，则说明无解
            has_ans = false;

            break;
        }
        //扩展欧几里得算法是求k1* a1 - k2 * a2 = gcd(a1, a2)的解，
        // 但是这里是k1* a1 - k2 * a2 = m2 - m1，所以需要翻(m2 - m1) / d倍
        //扩展欧几里得求的是k1 ，-k2这样一组解，这里不带入k2，而带入k1，k1就是原来的方程的一个解
        k1 *= (m2 - m1) / d;
        //写成通解形式：k1` = k1 + k * a2 / d
        LL t = a2 / d;
        //将k1变成最小正整数的一个解
        //c语言的取模公式是A % B = A - A / B * B，所以负数取模仍然是负数
        //这里可以将一个负数取模后映射到0 ~ t的一个正数
        k1 = (k1 % t + t) % t;
        //通解形式带入后得到一个新的方程x = a1 * k1 + m1 +k[a1, a2] => x = m1 + ka1
        //最小公倍数a，由于d可能为负数，所以最小公倍数可能为负数，因为k是任意的，所以变为正数不受影响，这里主要是为了保持与题中的方程形式一致
        //x = m1 + ka1与x = m1 - ka1是等价的
        m1 = a1 * k1 + m1;
        //这里先用除法再用乘法，以免数据溢出
        a1 = abs(a1 / d * a2);
        //继续循环，用新的方程与方程组中的下一个方程合并
    }

    if(has_ans)
    {
        //如果有解的话我们用当前的x的值取最小即可
        //跳出循环的时候我们合并了最后一次得到x = m1 + k*a1
```

```
        //这里用k = 0得到一个x的解，然后取模得到最小正整数的解即可
        //当然本题中最后的m1不可能为负数，所以计算m1 = a1 * k1 + m1也可以直接m1 % a1
        cout << (m1 % a1 + a1) % a1;
    }
    else puts("-1");

    return 0;
}
```

扩展欧几里得算法与欧几里得算法的时间复杂度相同都是`O(loga)`，所以本题的时间复杂度就是`O(nloga)`

这就是比赛中数论题的难度.