算法基础（二十）：数学基础 - 数论5 - 扩展欧几里得算法

裴蜀定理

对任意的一对正整数`a, b`，那么一定存在非零整数`x, y`，使得`ax + by = gcd(a, b)`，其中`gcd(a, b)`表示`a, b`的最大公约数

且`gcd(a, b)`是`a, b`可以凑出来的最小的正整数，即若`ax+ by > 0`，则`gcd(a, b) = min(ax + by)`

证明：

若`ax + by = d`，则`d`一定是`gcd(a, b)`的倍数，这是显然的，所以`ax + by`的最小的正整数也就是最大公约数的一倍，就是`gcd(a, b)`

下证：存在非零整数`x, y`，使得`ax + by = gcd(a, b)`

证明存在性的问题，我们一般使用构造法，对任意的正整数`a, b`，构造`x, y`使得，都有`ax + by = gcd(a, b)`

而这个构造的方法就是扩展欧几里得算法

基本思想

存在`x,y` 使得 `ax + by = gcd(a,b)`，扩展欧几里得算法就是为了求出`x, y`，我们用递归去求，基本思想就是用递归下一层得到的`x', y'`，来求得本层的`x, y`

因为欧几里得算法：`ax + by = gcd(a,b) = gcd(b,amodb)`

递归进入下一层的时候，我们传递参数`exgcd(a = b, b = amodb, x = y, y = x),`，于是在下一层的递归中有：`by + (amodb)x = gcd(b,amodb)`

我们记作：`a'x' + b'y' = gcd(a', b')`，在下一层的递归中`a' = b, b' = amodb`

在下一层中也就是：`bx' + amodby' = gcd(b, amodb)`

由模运算的定义，有：`bx′ + (a − ⌊a/b⌋b)y′ = gcd(a, b)`

我们将左边展开有：`ay′ + b(x′ − ⌊a/b⌋y′) = gcd(a, b)`
而`gcd(a, b)`在上一层的递归中有：`gcd(a, b) = ax + by`

所以有：`ay′ + b(x′ − ⌊a/b⌋y′) = ax + by`

故`x = y′,y = x′ − ⌊a/b⌋y′`，于是我们得到了欧几里得算法求解中两层递归的参数之间的联系，我们就可以将下一层计算完成后计算出本层的参数，于是这样一直往上返回，计算出最开始的`x, y`

经过传参，在下一层的递归中我们计算出来了`x', y'`，由于传参`x' = y, y' = x`也就是计算出来了上一层的`y, x`的引用值，但此时的`y, x`的值是下一层的`a'x' + b'y' = gcd(a, b)`对应的变量，而在上一层中，根据上面得到的两层之间的传递关系等式，我们需要再进行一次计算，即：

`x = x, y = y - ⌊a/b⌋x`

于是就得到了上一层的`x, y`的正确的值

当我们递归到最底层的时候，也就是`b' = 0`的时候，由于等式`a'x' + b'y' = gcd(a', b') = a'`，所以`x' = 1, y' = 任意数`，我们取`x' = 1, y' = 0`作为最底层的一组解，然后返回到上一层的递归，就这样一步步求出来了最开始的`ax + by = gcd(a, b)`的`x, y`

代码实现

给定 n 对正整数 a_i, b_i，对于每对数，求出一组 x_i, y_i，使其满足 $a_i \times x_i + b_i \times y_i = gcd(a_i, b_i)$。

输入格式

第一行包含整数 n。

接下来 n 行，每行包含两个整数 a_i, b_i。

输出格式

输出共 n 行，对于每组 a_i, b_i，求出一组满足条件的 x_i, y_i，每组结果占一行。

本题答案不唯一，输出任意满足条件的 x_i, y_i 均可。

数据范围

$1 \le n \le 10^5$,
$1 \le a_i, b_i \le 2 \times 10^9$

输入样例：

```
2
4 6
8 18
```

输出样例：

```
-1 1
-2 1
```

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

//扩展欧几里得求系数x和y
int exgcd(int a, int b, int &x, int &y)
{
    //首先是边界情况，当b = 0的时候
    //gcd(a, 0) = a, 那么ax + 0y = a
    //那么x = 1,  y = 0就是一组解
    if(!b)
    {
        x = 1, y = 0;
        return a;
    }
    //记录最大公约数
    //反转一下，y, x也需要反转，便于计算
    int d = exgcd(b, a % b, y, x);
    //更新本层的x, y的值
```

```
    y -= a / b * x;
    //返回最大公约数到上一层
    return d;
}

int main()
{
    int n;
    scanf("%d", &n);

    while(n --)
    {
        int a, b, x, y;
        scanf("%d%d", &a, &b);

        //首先传入系数和a, b
        exgcd(a, b, x, y);
        cout << x << " " << y << endl;
    }

    return 0;

}
```

应用：线性同余方程

求一个整数`x`，使得$ax \equiv b(mod m)$，随便返回一个解即可，也就等价于：存在一个整数`y`，使得，$ax = my + b$，即$ax + my` = b$

也就等价于找x, y，使得上面那个方程成立

这个方程有解的条件是`gcd(a, m) | b`，假如`b`可以整除`gcd(a, m)`，那么根据扩展欧几里得算法一定可以求出$ax + my` = gcd(a, m)$，于是我们只需要将`x, y'`翻倍即可

假设`b`不能整除`gcd(a, m)`，则由于裴蜀定理，解不存在

代码实现

给定 n 组数据 a_i, b_i, m_i，对于每组数求出一个 x_i，使其满足 $a_i \times x_i \equiv b_i \pmod{m_i}$，如果无解则输出 `impossible`。

输入格式

第一行包含整数 n。

接下来 n 行，每行包含一组数据 a_i, b_i, m_i。

输出格式

输出共 n 行，每组数据输出一个整数表示一个满足条件的 x_i，如果无解则输出 `impossible`。

每组数据结果占一行，结果可能不唯一，输出任意一个满足条件的结果均可。

输出答案必须在 int 范围之内。

数据范围

$1 \le n \le 10^5$,
$1 \le a_i, b_i, m_i \le 2 \times 10^9$

输入样例：

```
2
2 3 6
4 3 5
```

输出样例：

```
impossible
-3
```

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

typedef long long LL;
int exgcd(int a, int b, int &x, int &y)
{
    if(!b)
    {
        x = 1, y = 0;
        return a;
    }

    int d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}

int main()
{
    int n;
    scanf("%d", &n);

    while(n --)
    {
```

```
        int a, b, m;
        scanf("%d%d%d", &a, &b, &m);
        int x, y;
        int d = exgcd(a, m, x, y);
        if(b % d) puts("impossible");
        //最后x乘相应的倍数得到正确的方程的解
        else printf("%d\n", (LL)x * (b / d) % m);
    }

    return 0;
}
```

需要注意的几个地方：

1. **为什么用`(LL)`**

有可能是会爆`int`的，比如$ax + my` = b$中，`a, m, y'`足够小，`b`足够大，这时`x`足够大，若是再乘上一个倍数的话就可能会爆`int`

2. **为什么要取余**

对于一个特解：$ax_0 + my_0 = b$

假设通解的形式是：$ax + my = b$

两个式子相减得到：$a(x_0 - x) = m(y - y_0)$

由于`a, m`互质，所以`x0 - x`必然是`m`的倍数，所以`x = x0 + km`，我的得到真实的`x`之后再对`m`取模可以得到最小的解，这样可以即保证结果是正确的，又保证结果在`int`内