算法基础（十七）：数学基础 - 数论2 - 约数

求一个数的所有的约数（式除法）

约数一定是成双出现的，如果一个约数是`d`，那么快一个约数就必然是`r / d`，我们枚举较小的那一个，也就是满足`d <= r / d`的约数，另外一个我们直接用`r / d`算出来

```
vector<int> get_divisors(int n)
{
    vector<int> res;
    //约数是成对出现的，我们只需要枚举小的约数
    for(int i = 1; i <= n / i; i ++)
    {
        if(n % i == 0)
        {
            res.push_back(i);
            //有可能n是i的平方，比如9 = 3 * 3
            //但是我们加入进去的约数不能一样所以这里需要判断一下
            if(i != n / i) push_back(n / i);
        }

    }
    sort(res.begin(), res.end());
}
```

时间复杂度妥妥`O(sqrt(n))`

记住做数论的题目一定要计算时间复杂度，很容易超时，我们还需要计算一下排序的复杂度

首先，我们来算一下`1 ~ n`这些数中所有约数的个数，也就是任意一个数`k`它的约数有`m`个，计算所有数的约数个数都加起来的个数

对于`1`来讲，他是`1`的倍数的约数，对于`2`来讲，他是`2`的倍数的约数，对于`3`来讲，他是`3`的倍数的约数，一个数`k`，其倍数为`m1 m2 m3`，有三个，那么`k`作为约数，也要被加上三次

所以`1 ~ n`中总共约数的个数与`1 ~ n`中总共倍数的个数相同，所以我们只需要统计`1 ~ n`中所有的倍数即可

`1`的倍数：`n`个

`2`的倍数：`n / 2`个

`3`的倍数：`n / 3`个

`n`的倍数：`n / n`个

所以倍数的总数是`n + n/2 + n/3 + n/4 + ... + n/n`一共`n * logn`个，平均下来每个数的约数个数就是`logn`个，所以排序的复杂度平均就是`O(logn * log(logn))`小于`O(sqrt(n))`

约数个数与约数之和

约数个数

由算术的基本定理，任何一个正整数都可以写成素数幂之积的形式：

$N = p_1^{\alpha_1} p_2^{\alpha_2} p_3^{\alpha_3} p_4^{\alpha_4} \ldots$

那么对于`N`的一个约数`d`也可以写成这样的形式：

$d = p_1^{\beta_1} p_2^{\beta_2} p_3^{\beta_3} p_4^{\beta_4} \ldots$，其中$0 \leq \beta_i \leq \alpha_i$ **（1）**

注意上面`N`和`d`的`p1，p2，p4...`是相等的，因为`d`是`N`的约数，如果`d`含有除了`N`之外的质数，那么显然`d`不再是`N`的约数了

于是那么对于`d`来讲，只要β的取值发生变化，那么约数也就发生了变化，根据算术基本定理，对于一个`d`来讲，**其$\beta_1 - \beta_k$的取值是唯一的，取值发生变化，`d`也就不同**，不可能出现取值不同的两组对应的`d`相同这样的状态**（2）**

而对于`N`的每一个约数，都对应了`d`中$\beta_1 - \beta_k$的一种取法，对于`d`的每一个`pi`，其指数的取值都在$0 - \alpha_i$之间，一共有$\alpha_i + 1$种取法

所以根据排列组合的知识，所有的约数个数也就是$\beta_1 - \beta_k$的所有取法就是乘起来
$(\alpha_1 + 1) * (\alpha_2 + 1) * (\alpha_3 + 1) \ldots (\alpha_k + 1)$ **（3）**

其中（1）保证一定是约数，也就是约数一定是这种形式（2）保证不重复（3）保证这种形式的所有的取值

另外，`int`返范围的整数，一个数约数最多在`1500`左右

代码实现：

> 给定 n 个正整数 a_i，请你输出这些数的乘积的约数个数，答案对 $10^9 + 7$ 取模。
>
> #### 输入格式
>
> 第一行包含整数 n。
>
> 接下来 n 行，每行包含一个整数 a_i。
>
> #### 输出格式
>
> 输出一个整数，表示所给正整数的乘积的约数个数，答案需对 $10^9 + 7$ 取模。
>
> #### 数据范围
>
> $1 \leq n \leq 100$,
> $1 \leq a_i \leq 2 \times 10^9$
>
> #### 输入样例：
>
> ```
> 3
> 2
> 6
> 8
> ```
>
> #### 输出样例：
>
> ```
> 12
> ```

求`a1 * a2 * a3...`的积的质因子的指数，也就是分别求`a1,a2,a3...`的质因子的指数，然后将这些指数累加起来，就得到了乘积对应的质因子的指数，质数之间始终是互素的，这些数的所有的质因子乘起来之后不会形成新的质因子，也不会减少质因子，恰好就是这些数的乘积的质因子

数的范围：

`int`：四个字节，`2147483647 ~ -2147483648`，最大的数不超过`2 ^ 31 - 1`，大约是`2 * 10^9`（比这个数大一点）

`long`：四个字节，同`int`

`float`：四个字节

`double`：八个字节

`long long (int)`：最大值大于`10 ^ 18`

```
#include<iostream>
#include<algorithm>
#include<unordered_map>

using namespace std;
//求a1 * a2 * a3...的积的质因子的指数
//也就是分别求a1,a2,a3...的质因子的指数，然后将这些指数加起来

typedef long long LL;

const int mod = 1e9 + 7;

int main()
{
    int n;
    cin >> n;

    //定义一个哈希表来记录所有的底数和指数
    unordered_map<int, int> primes;

    //分别分解每一个数
    while(n --)
    {
        int x;
        //输入一个数x，找到他的所有的质因数
        cin >> x;

        for(int i = 2; i <= x / i; i ++)
        {
            while(x % i == 0)
            {
                x /= i;
                //i这个质因数对应的指数加一
                primes[i] ++;
            }
        }
```

```

        //如果最后的这个结果大于1，说明x中存在大于sqrt(x)的质因数
        if(x > 1) primes[x] ++;
    }
    //跳出while循环之后，primes里面存的就是乘积的所有的质因数的指数
    //接下来就是把所有的指数加一再相乘

    //因为结果可能很大，所以用LL存储
    LL res = 1;

    for(auto prime : primes)
    {
        //取模公式(a*b)mod = (a mod * b mod)mod
        //long long的最大值是大于10^18的
        //res的值不会大于10^9 + 7
        //res的值不会超过long long的范围，不用考虑溢出，结果计算符合上面的取模公式，不会出现10^9 * 10^9这样的情况
        res = res * (prime.second + 1) % mod;
    }

    cout << res <<endl;
    return 0;
}
```

约数之和

一个数表示为：

$N = p_1^{\alpha_1}p_2^{\alpha_2}p_3^{\alpha_3}p_4^{\alpha_4}...$

那么它的约数之和为：

$\Pi_1^k(\Sigma_0^\alpha p_k)$

也就是：

$(p_1^0 + p_1^1 + p_1^2 +..p_1^{\alpha_1})*...(p_k^0 + p_k^1 + p_k^2 +..p_k^{\alpha_k})$

其实将这个式子展开的每一项都是$d = p_1^{\beta_1}p_2^{\beta_2}p_3^{\beta_3}p_4^{\beta_4}...$的形式其中$0 \le \beta_i \le \alpha_i$，**也就是`N`的约数**

这些乘积和`() + () + ()... + ()`**一共有**$(\alpha_1 + 1) * (\alpha_2 + 1) * (\alpha_3 + 1)...(\alpha_k + 1)$**项**

因为`p1，p2...pk`都是素数，当两项指数不同的时候，两项互相做除法，必然有因子互素，也就是不可能互约相等，**所以每一项都不同**

所以这些数必然就是`N`的所有的约数

给定 n 个正整数 a_i，请你输出这些数的乘积的约数之和，答案对 10^9+7 取模。

输入格式

第一行包含整数 n。

接下来 n 行，每行包含一个整数 a_i。

输出格式

输出一个整数，表示所给正整数的乘积的约数之和，答案需对 10^9+7 取模。

数据范围

$1 \le n \le 100$,
$1 \le a_i \le 2 \times 10^9$

输入样例：

```
3
2
6
8
```

输出样例：

```
252
```

当我们求出每个数的质因数，并把指数累加之后其实就是乘积的质因数以及对应的指数

这时我们只需要套公式即可，约数之和为$(p_1^0+p_1^1+p_1^2+..p_1^{\alpha_1})*...(p_k^0+p_k^1+p_k^2+..p_k^{\alpha_1})$

在求$(p_1^0+p_1^1+p_1^2+..p_1^{\alpha_1})$的时候令`t = 1`然后循环$\alpha_1$次`t * p1 + 1`，循环一次的结果是`p1 + 1`，两次就是`p1^2 + p1 + 1`，三次的结果就是`p1 ^ 3 + p1 ^ 2 + p1 + 1`，这是秦九韶算法

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

typedef long long LL;

int mod = 1e9 + 7;

int main()
{
    //同前面的代码，先求出所有数的质因数以及对应的指数
    int n;
    cin >> n;
    unordered_map<int, int> primes;
    while(n --)
    {
        int x;
        cin >> x;
```

```
            for(int i = 2; i <= x / i; i ++)
            {
                while(x % i == 0)
                {
                    x /= i;
                    primes[i] ++;
                }
            }

            if(x > 1) primes[x] ++;
        }

        //此时primes里面就是乘积的质因数以及对应的指数
        LL res = 1;
        //按公式求所有约数的和
        for(auto prime : primes)
        {
            int p = prime.first;
            int a = prime.second;
            //用公式
            LL t = 1;
            //为例防止中间结果溢出，每一步都要取余
            while(a --) t = (t * p + 1) % mod;
            res = res * t % mod;

        }

        cout << res << endl;

        return 0;
}
```

注意最后的结果要求对某个数取余的时候，最好是在中间的过程中取余然后计算，而且中间结果的数据类型尽量开大一点，以免数据溢出导致出错

需要注意的是，在用秦九韶算法求和的时候时间复杂度是`O(a)`,`a`是上面代码中的指数，若用快速幂，时间复杂度是`aloga`其实是大于秦九韶算法的

对于本题来讲，整个的算法的时间复杂度最多是`O(a1 + a2 + ...)`，`ai`是最后乘积结果`N`某个质因数的指数

对于一个给定的乘积结果`N`，只有其质因数足够小，其指数的所有的和加起来才足够大

所以对于本题来讲，其质因子全是`2`的时候，指数才会最大

那么对于全体乘积结果`N`，假设`100`个数（这一百个数都是`2`的倍数）中最大的一个数是`ak`，那么最后乘积的指数最大估算一下就是`100 * log(ak)`，进入循环计算和的时候复杂度也就是`O(100 * log(ak))`，这个复杂度可以接受，不用再改进

最大公约数（欧几里得算法）

基本思想

又叫辗转相除法

基本性质：若`d | a`且`d | b`，则`d | a + b`且`d | ax + by`

最核心的一点： `gcd(a, b) = gcd(b, a mod b)`， `a`与`b`的最大公约数也就是`b`与`a mod b`的最大公约数

证明：

`a mod b = a - [a/b](取整) * b`

简单记成`a mod b = a - c * b`

于是我们需要证明的就变成了`gcd(a, b) = acd(b, a - c * b)`

对于左边的任何一个公约数有`d | a`且`d | b`，根据性质可以得到`d | b`且`d | (a - c * b)`，也就是说左边的公约数也一定是右边的公约数

反过来看

`d | b, d | a - c * b`根据性质就有`d | a - c * b + b * c = d | a`，所以右边的公约数也一定是左边的公约数

所以左右两边的公约数是完全相同的，所以`gcd(a, b) = gcd(b, a mod b)`

代码实现

给定 n 对正整数 a_i, b_i，请你求出每对数的最大公约数。

输入格式

第一行包含整数 n。

接下来 n 行，每行包含一个整数对 a_i, b_i。

输出格式

输出共 n 行，每行输出一个整数对的最大公约数。

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
3 6
4 6
```

输出样例：

```
3
2
```

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


int gcd(int a, int b)
{
    //如果b不为零，则计算b和a % b的最大公约数
    //如果b为零，说明调用本次函数的上一次函数中a 可以被b整除
    //说明上一次函数中b是a的约数，一个数是另一个数的约数也就是最大公约数
    //那么在本次中b传递给了a
    //所以将a返回
    //gcd(21, 14) = gcd(14, 7) = gcd(7, 0) = 7,一步步返回
    return b ? gcd(b, a % b) : a;
}


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


    while(n --)
    {
        int a, b;
        cin >> a >> b;
        cout << gcd(a, b) << endl;
    }

    return 0;
}
```

欧几里得算法的时间复杂度是`O(logn)`