Thuật toán bài MINX

Ký hiệu gcd là ước số chung lớn nhất và lcm là bội số chung nhỏ nhất

$$ax \vdots b \Leftrightarrow x \vdots \frac{b}{\gcd(a,b)}$$
$$bx \vdots c \Leftrightarrow x \vdots \frac{c}{\gcd(b,c)}$$
$$cx \vdots a \Leftrightarrow x \vdots \frac{a}{\gcd(c,a)}$$

Vậy $x \vdots \operatorname{lcm}\left(\frac{b}{\gcd(a,b)}, \frac{c}{\gcd(b,c)}, \frac{a}{\gcd(c,a)}\right)$

Vì ta muốn x là số nguyên dương nhỏ nhất nên ta sẽ chọn $x = \operatorname{lcm}\left(\frac{b}{\gcd(a,b)}, \frac{c}{\gcd(b,c)}, \frac{a}{\gcd(c,a)}\right)$

Công thức này có thể tính bằng thuật toán Euclid hoặc bằng cách phân tích a, b, c ra thừa số nguyên tố. Nếu dùng cách phân tích thừa số nguyên tố, cần sàng trước để lưu một ước nguyên tố của tất cả các số nguyên dương $\leq 10^6$

Độ phức tạp $O(\log \max(a,b,c))$ cho mỗi test.

Thuật toán bài LAKE

Định lý Pytago (Pythagorean theorem):

Trong một tam giác vuông, bình phương độ dài cạnh huyền bằng tổng bình phương độ dài hai cạnh góc vuông.

Bộ ba số nguyên (a,b,c) được gọi là bộ ba Pytago (Pythagorean triples) nếu $a^2 + b^2 = c^2$. Ở đây a, b có thể hoán vị cho nhau, tức là bộ ba (a,b,c) cũng coi là bộ ba (b,a,c)

Subtask 1: Với mỗi test, ta chỉ cần thử mọi cặp số $1 < a < b < p$, đặt $c = p - a - b$ và kiểm chứng định lý Pytago. Độ phức tạp $O(p^2)$

Subtask 2: Tính trước `Result[p]` với $p = 1 \ldots 5000$ là kết quả với dữ liệu p. Khởi tạo mảng Result toàn 0. Sau đó ta thử mọi cặp số $1 < a < b \leq 5000$ và kiểm tra $c = \sqrt{a^2 + b^2}$ có phải số nguyên hay không. Nếu c nguyên, ta tính $p = a + b + c$ rồi cực đại hoá $Result[p]$ theo $\frac{ab}{2}$. Cuối cùng chỉ đọc từng dữ liệu p và in ra Result[p]

Subtask 3: Với mỗi dữ liệu p, ta thử mọi số nguyên dương $c \leq p$. Sau đó giải hệ phương trình $\begin{cases} a + b = p - c \\ a^2 + b^2 = c^2 \end{cases}$.

Hệ này quy về một phương trình bậc hai với ẩn a: $f(a) = a^2 + (p - c - a)^2 = c^2$. Có thể giải bằng công thức hoặc thuật toán tìm kiếm nhị phân (Trong khoảng $a = 1 \ldots \frac{p-c}{2}$, khi a tăng $f(a)$ sẽ giảm). Nếu hệ có nghiệm nguyên ta ghi nhận kết quả theo $\frac{ab}{2}$...

Subtask 4:

Một bộ ba Pytago (a,b,c) được gọi là nguyên thủy nếu ước số chung lớn nhất của a, b, c bằng 1: $\gcd(a,b,c) = 1$. Dễ thấy rằng trong bộ ba Pytago nguyên thủy (a,b,c) thì hai số bất kỳ nguyên tố cùng nhau, bởi nếu một số nguyên tố p là ước chung của hai trong số chúng thì từ hệ thức $a^2 + b^2 = c^2$, p cũng phải là ước của số còn lại.

Một bộ ba Pytago bất kỳ có thể thu được từ một bộ ba nguyên thủy bằng cách nhân cả ba số trong bộ ba nguyên thủy lên cùng một hằng số nguyên dương.

Trong bộ ba nguyên thủy (a,b,c) chắc chắn c là số lẻ, và trong hai số a, b có đúng một số chẵn và một số lẻ:

- Nếu a, b đều chẵn thì chúng không nguyên tố cùng nhau, trái giả thiết về bộ ba nguyên thủy
- Nếu a, b đều lẻ thì a^2 và b^2 đều chia 4 dư 1, vậy $c^2 = a^2 + b^2$ chia 4 dư 2, vô lý vì không có số chính phương chia 4 dư 2.

Vậy trong hai số a, b có một số chẵn và một số lẻ, từ $c^2 = a^2 + b^2$ là số lẻ suy ra c là số lẻ. Trong các bộ ba nguyên thủy sau này, ta giả thiết a là số chẵn còn b, c là số lẻ.

Với x, y là hai số nguyên dương, $x > y$, thì bộ ba $(a = 2xy, b = x^2 - y^2, c = x^2 + y^2)$ là một bộ ba Pytago. Điều này dễ dàng kiểm chứng.

Ngược lại với (a, b, c) là một bộ ba **nguyên thủy**, luôn tồn tại hai số nguyên dương x, y để $a = 2xy, b = x^2 - y^2$ và $c = x^2 + y^2$. Điều này chứng minh như sau

Với (a, b, c) là một bộ ba nguyên thủy, a chẵn, từ:

$$\begin{aligned} & a^2 + b^2 = c^2 \\ \Leftrightarrow \quad & \frac{c+b}{a} = \frac{a}{c-b} \end{aligned}$$

Gọi $\frac{x}{y}$ là phân số tối giản có giá trị bằng $\frac{c+b}{a}$ cũng như $\frac{a}{c-b}$, ta có:

$$\begin{cases} \frac{c+b}{a} = \frac{x}{y} \\ \frac{c-b}{a} = \frac{y}{x} \end{cases} \Rightarrow \begin{cases} \frac{c}{a} + \frac{b}{a} = \frac{x}{y} \\ \frac{c}{a} - \frac{b}{a} = \frac{y}{x} \end{cases} \Rightarrow \begin{cases} \frac{c}{a} = \frac{1}{2}\left(\frac{x}{y} + \frac{y}{x}\right) \\ \frac{b}{a} = \frac{1}{2}\left(\frac{x}{y} - \frac{y}{x}\right) \end{cases} \Rightarrow \begin{cases} \frac{c}{a} = \frac{x^2+y^2}{2xy} \\ \frac{b}{a} = \frac{x^2-y^2}{2xy} \end{cases}$$

Ta chỉ cần chứng minh hai phân số $\frac{x^2+y^2}{2xy}$ và $\frac{x^2-y^2}{2xy}$ tối giản nữa là xong.

Vì $\frac{x}{y}$ tối giản nên x và y không thể cùng chẵn. Nếu x, y cùng lẻ thì phân số $\frac{x^2-y^2}{2xy}$ có tử số chia hết cho 4 còn mẫu số chia hết cho 2 nhưng không chia hết cho 4. Sau khi tối giản phân số này thành $\frac{b}{a}$ thì chắc chắn b sẽ là số chẵn và a là số lẻ, trái với quy ước a chẵn và b lẻ.

Vậy x và y nguyên tố cùng nhau và trong đó có một số chẵn và một số lẻ. Xét hai phân số $\frac{x^2+y^2}{2xy}$ và $\frac{x^2-y^2}{2xy}$, nếu có một số nguyên tố p là ước của mẫu $2xy$ thì $p = 2$ hoặc $p|x$ hoặc $p|y$,

- Nếu $p = 2$ thì p không phải là ước số của $x^2 \pm y^2$ vì đây là số lẻ.
- Nếu $p|x$ thì $p \nmid y$ nên p cũng không phải là ước của $x^2 \pm y^2$
- Nếu $p|y$ thì $p \nmid x$ nên p cũng không phải là ước của $x^2 \pm y^2$

Từ $\frac{c}{a} = \frac{x^2+y^2}{2xy}$ và $\frac{b}{a} = \frac{x^2-y^2}{2xy}$, các phân số đều tối giản. Suy ra $a = 2xy, b = x^2 - y^2$ và $c = x^2 + y^2$

Đến đây ta thiết kế thuật toán như sau: Tính trước `Result[p]` với $p = 1 \dots 10^6$ là kết quả với dữ liệu p:

Khởi tạo mảng Result toàn 0. Duyệt mọi cặp (x, y) mà $x^2 + y^2 \leq 10^6$ (Có $\leq 10^6$ cặp như vậy). Đặt $a = 2xy, b = x^2 - y^2, c = x^2 + y^2, p = a + b + c$. Sau đó ghi nhận:

```
Result[p] = max(Result[p], a * b / 2);
```

Chú ý rằng việc này quét hết các bộ ba nguyên thủy và có thể xét thừa ra các bộ ba Pytago không nguyên thủy nhưng không vấn đề gì cả. Nếu không muốn xét thừa cần kiểm tra $\gcd(x, y) = 1$, điều này có thể làm tăng độ phức tạp tính toán.

Để cập nhật trên tất cả các bộ ba Pytago, ta chỉ cần một phép xử lý đơn giản.

```
for (p = 1; p <= 10^6; ++p)
    if (Result[p] ≠ 0)
        for (k = 2; k * p <= 10^6; ++k) //Nhân bộ ba lên k, diện tích nhân lên k^2
            Result[k * p] = max(Result[k * p], Result[p] * k^2);
```

Độ phức tạp $O(maxP \ln maxP)$ ở đây $maxP = 10^6$

Sau khi đã tính trước mảng kết quả thì mỗi truy vấn chỉ cần in ra output trong thời gian $O(1)$

Thuật toán bài COUNT

Sắp xếp tăng dần dãy A và bổ sung thêm phần tử $a_0 = 0$:

$$0 = a_0 < a_1 \le a_2 \le \cdots \le a_n$$

Với mỗi chỉ số i, tính $f[i]$ là số giá trị nguyên dương $< a_i$ mà không có mặt trong dãy A:

$f[0] = 0$;

$\forall i \ge 1$:

$$f[i] = \begin{cases} f[i-1], \text{nếu } a_i = a_{i-1} \\ f[i-1] + (a_i - a_{i-1} - 1), \text{nếu } a_i > a_{i-1} \end{cases}$$

Giải thích:

- Nếu $a_i = a_{i-1}$ thì những giá trị nguyên dương $< a_i$ cũng là những giá trị nguyên dương $< a_{i-1}$
- Nếu $a_i > a_{i-1}$ thì $f[i]$ được đếm bằng:
  - Những giá trị nguyên dương $< a_{i-1}$ mà không có mặt trong A: $f[i-1]$
  - Những giá trị nguyên dương nằm trong khoảng từ $a_{i-1} + 1$ tới $a_i - 1$ cũng không có mặt trong A: $a_i - a_{i-1} + 1$

Với mỗi truy vấn k, tìm a_i lớn nhất $< k$. Khi đó số giá trị nguyên dương $< k$ mà không có mặt trong dãy A sẽ được tính bằng:

- Số giá trị nguyên dương $< a_i$ mà không có mặt trong dãy A: $f[i]$
- Cộng với số giá trị nguyên dương nằm trong khoảng từ $a_i + 1$ tới $k - 1$: bằng $k - a_i - 1$

Độ phức tạp $O((m+n)\log n)$

Có những cách khác có thể chấp nhận: Thay vì sắp xếp dãy A ta có thể sắp xếp dãy k, hoặc sắp xếp cả hai dãy A và k rồi dùng 2 con trỏ….

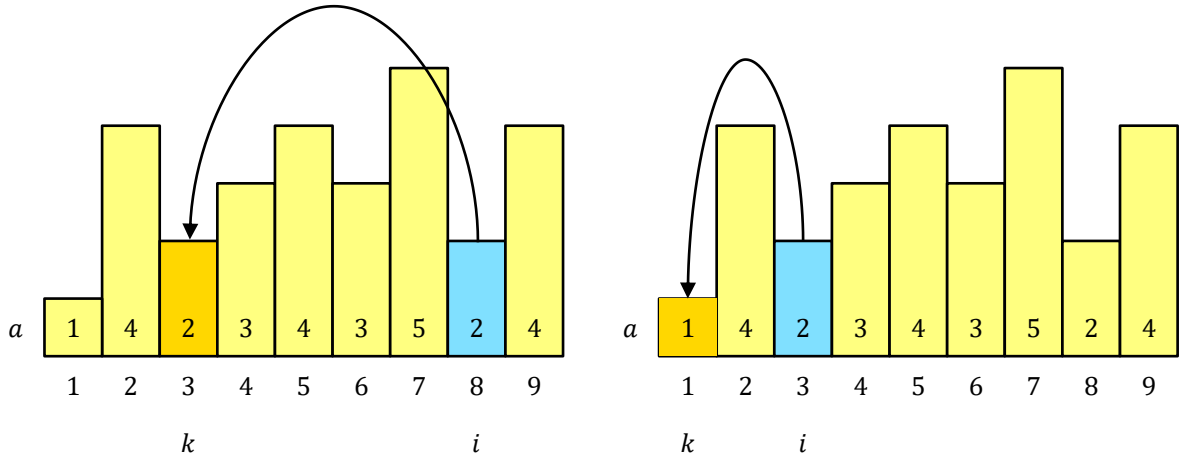
Thuật toán bài CELLFILL

Vì chỉ được ghi đè số lớn hơn lên số bé hơn, ta sẽ thực hiện các lệnh Fill(i, j, v) theo thứ tự từ v nhỏ tới v lớn. Bằng một quan sát đơn giản, ta thấy rằng chỉ cần xác định các vị trí i mà tại đó ta phải thực hiện lệnh $Fill(i, \dots, a_i)$ là xong.

Với một vị trí i có $a_i > 0$, tìm vị trí k đứng trước, gần i nhất mà $a_k \le a_i$.

Nếu tìm được a_k như vậy mà $a_k = a_i$, chắc chắn ta không cần thực hiện lệnh Fill bắt đầu từ vị trí i vì lệnh này có thể bỏ đi, thay thế bằng cách "kéo dài" lệnh Fill từ vị trí k cho tới qua vị trí i. Như hình bên trái dưới đây ta không cần thực hiện lệnh Fill$(8, \dots, 2)$ vì để điền số 2 vào vị trí 8, ta có thể điền từ vị trí 3 trở đi.

Nếu không tìm được a_k như vậy hoặc tìm thấy mà $a_k < a_i$, chắc chắn ta phải thực hiện lệnh Fill bắt đầu từ vị trí i. Lý do là $a[k+1 \dots i-1]$ đều $> a_i$, có điền giá trị a_i vào những ô này cũng vô nghĩa. Ngoài ra ta lại không được điền giá trị a_i vào vị trí k. Như hình bên phải dưới đây, bắt buộc ta phải thực hiện lệnh $Fill(3, \dots, 2)$ và $Fill(4, \dots, 3)$



Với $\forall i$, để xác định vị trí k đứng trước, gần i nhất mà $a_k \leq a_i$ ta có thể dùng một stack S chứa các chỉ số: Xét lần lượt các chỉ số $i = 1,2, \dots, n$: Loại bỏ các chỉ số ở đỉnh stack nếu chỉ số đó ứng với $a[.] > a_i$, chỉ số còn lại trong stack chính là chỉ số k đứng trước, gần i nhất mà $a_k \leq a_i$, cuối cùng ta đẩy chỉ số i vào stack.

```
stack<int> S;
Result = 0;
for (int i = 1; i <= n; ++i)
{
    while (!S.empty() && S.top() > a[i]) S.pop(); //Vứt bỏ các chỉ số của phần tử > a[i] ở đỉnh stack
    if (a[i] > 0 && (S.empty() || S.top() < a[i]))
        ++Result; //Phải thực hiện lệnh Fill từ vị trí i
    S.push(a[i]);
}
Output ← Result;
```

Độ phức tạp $O(n)$ có thể đánh giá qua số lần thực hiện lệnh $S.pop()$: Có tổng cộng n lần S.push(.) nên có tối đa n lần $S.pop()$. Đánh giá chặt hơn thì có tối đa $n - 1$ lần $S.pop()$ do chỉ số n không bao giờ bị lấy ra.