KẾT HỢP NGUYÊN LÝ BAO HÀM LOẠI TRỪ VÀ ƯỚC CHUNG LỚN NHẤT

A. Lời nói đầu

Trong các kỳ thi Tin học, kiến thức liên quan đến đại số và số học là yếu tố quan trọng trong nhiều thuật toán thông dụng. Việc áp dụng các công thức toán học một cách thông minh có thể giảm thiểu thời gian tính toán một cách đáng kể, giúp một số thuật toán vượt trội hơn so với những thuật toán khác.

Trong bài viết này, tôi sẽ trình bày một phương pháp ngắn gọn nhưng cực kỳ hiệu quả, đặc biệt là đối với dạng bài toán liên quan đến Ước Chung Lớn Nhất (GCD), đó là kỹ thuật sử dụng nguyên lý bao hàm và loại trừ qua các bội số. Đây là một chiến lược tuyệt vời giúp giải quyết những vấn đề khó khăn liên quan đến GCD mà không cần đến những kiến thức phức tạp ở cấp độ đại học như hàm Mobius đảo ngược.

Nhận thấy rằng phương pháp này vẫn còn khá mới mẻ nhưng lại mang lại hiệu quả cao và dễ dàng tiếp cận đối với học sinh ở cấp Trung học phổ thông, tôi sẽ phân tích sâu hơn về lý thuyết và các bài tập áp dụng phương pháp này.

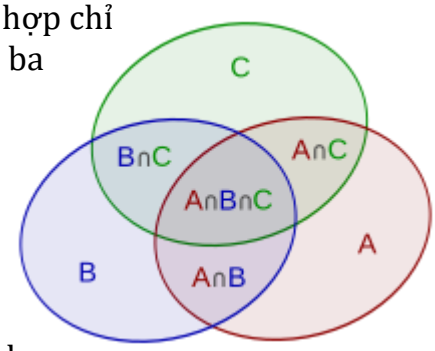
Các thầy cô có thể tải bộ test và code tại link sau:

B. Lý thuyết

1. Nguyên lý bao hàm-loại trừ

- **Định nghĩa:** Nguyên lý bao hàm-loại trừ là kỹ thuật đếm tổng quát, giúp tìm số phần tử của hợp của các tập hữu hạn bằng cách cộng số lượng phần tử của từng tập, sau đó trừ đi số lượng phần tử chung giữa chúng. Trong đó A và B là hai tập hữu hạn và $|S|$ là số lực lượng của tập hợp S (có thể coi là số phần tử trong tập hợp nếu tập hợp đó hữu hạn). Công thức trên nói rằng khi cộng kích thước của hai tập hợp với nhau, giá trị cho có thể quá lớn bởi có thể sẽ có một số phần tử bị đếm hai lần. Các phần tử bị đếm hai lần nằm trong phần giao của hai tập hợp đó và do đó để tìm ra đúng giá trị, ta trừ đi kích thước của phần giao.

- Nguyên lý bao hàm-loại trừ là dạng tổng quát của trường hợp chỉ xét hai tập hợp, nên để bắt đầu ví dụ, ta xét công thức cho ba tập hợp A, B và C:



Ta có thể kiểm chứng công thức này bằng cách đếm số lần mỗi vùng trong biểu đồ Venn nằm trong vế phải của công thức.

Minh hoạ nguyên lý bao hàm-loại trừ bằng biểu đồ Venn cho ba tập hợp

- **Công thức Tổng quát**: Đối với n tập hợp hữu hạn A1, …, An, công thức tính số phần tử của hợp của chúng được biểu diễn qua các phép cộng và trừ số lượng phần tử của các tập hợp và giao của chúng

Tổng quát hoá các ví dụ này sẽ dẫn tới nguyên lý bao hàm-loại trừ. Để tìm lực lượng của hợp của n tập hợp:

1. Thêm lực lượng của các tập hợp.
2. Trừ lực lượng của các phần giao của 2 tập hợp.
3. Thêm lực lượng của các phần giao của 3 tập hợp.
4. Trừ lực lượng của các phần giao của 4 tập hợp.
5. Thêm lực lượng của các phần giao của 5 tập hợp..
6. Tiếp tục cho đến khi xét hết phần giao của n tập hợp. Bước cuối sẽ là thêm vào nếu n lẻ và trừ đi nếu n chẵn}}.

2. Các ví dụ

- Đếm số nguyên

Đây là ví dụ đơn giản trong áp dụng nguyên lý bao hàm-loại trừ, xét câu hỏi sau:[6]

Có bao nhiêu số nguyên trong tập {1, ..., 100} không chia hết cho 2, 3 hoặc 5?

Gọi $S = \{1,...,100\}$ và P_1 là tính chất số nguyên chia hết cho 2, P_2 là tính chất số nguyên chia hết cho 3 và P_3 là tính chất số nguyên chia hết cho 5. Gọi A_i là tập con của S chứa các phần tử có tính chất P_i, ta đếm được rằng: $|A_1| = 50$, $|A_2| = 33$, và $|A_3| = 20$. Có 16 số nguyên chia hết cho 6, 10 số chia hết cho 10, và 6 số chia hết cho 15. Và cuối cùng, chỉ có 3 số chia hết cho 30, nên số các số nguyên không chia hết cho bất kỳ 2, 3 hoặc 5 được tính bằng:

$100 - (50 + 33 + 20) + (16 + 10 + 6) - 3 = 26.$

- Đếm số mất thứ tự

Một câu hỏi phức tạp hơn trên được phát biểu như sau: Giả sử có bộ *n* lá bài được đánh số từ 1 đến *n*. Ta gọi lá số *m* nằm đúng vị trí nếu nó là lá thứ *m* trong bộ bài. Hỏi có bao nhiêu cách, (ký hiệu là *W*), để xáo bộ bài sao cho trong bộ có ít nhất 1 lá nằm đúng vị trí?

Ta bắt đầu bằng cách định nghĩa A_m, là số các thứ tự bài mà trong đó lá thứ *m* nằm đúng vị trí. Khi đó, số các thứ tự *W*, với *ít nhất* một lá nằm đúng vị trí được tính bởi

$$w = \left|\bigcup_{m=1}^{n} A_m\right|$$

Áp dụng nguyên lý bao hàm-loại trừ ta có:

$$w = \sum_{m_1}^{n}\left|A_{m_1}\right| - \sum_{1\le m_1 <m_2 \le n}\left|A_{m_1} \cap A_{m_2}\right| + \ldots + (-1)^{p-1} \sum_{1\le m_1 <m_p \le n}\left|A_{m_1} \cap \ldots \cap A_{m_p}\right| + \ldots$$

Mỗi giá trị $A_{m_1} \cap \ldots \cap A_{m_p}$ biểu diễn tập các phép xáo trong đó có ít nhất *p* giá trị $m_1, \ldots, m_p$ đang nằm đúng vị trí Lưu ý rằng số các xáo trộn với ít nhất *p* giá trị nằm đúng chỉ phụ thuộc vào *p*, chứ không trên giá trị cụ thể của m. Ví dụ chẳng hạn, số các phép xáo có vị trí thứ 1, thứ ba và thứ 17 nằm đúng bằng với số các phép xáo có vị trí thứ 2, thứ 5 và thứ 14 nằm đúng. Ta chỉ quan tâm rằng trong *n* lá đó và trong trường hợp xét ba lá, có 3 vị trí được chọn là vị trí đúng. Do đó có $\binom{n}{p}$phần tử bằng nhau trong tổng thứ *p*

3. **Ứng dụng:** Nguyên lý này được sử dụng trong lý thuyết số, tổ hợp, và xác suất rời rạc. Nó cũng có thể áp dụng trong việc tính xác suất và trong lý thuyết độ đo.

- **Đếm số phần giao**

Nguyên lý bao hàm-loại trừ khi kết hợp với luật De Morgan, có thể dùng để đếm số lực lượng của phần giao của các tập hợp. Gọi $\overline{A_k}$ là phần bù của A_k tương ứng với tập phổ dụng *A* nào đó sao cho *Ak*€*A* với mỗi *k*. Khi đó ta có

$$\cap_{i=1}^{n} A_i = \overline{\cup_{i=1}^{n} \overline{A_i}}$$

Do đó chuyển bài toán từ đếm phần giao sang đếm phần hợp.

- **Tô màu đồ thị**: Nguyên lý bao hàm-loại trừ lập thành cơ sở của các thuật toán giải các bài phân hoạch đồ thị thuộc lớp NP-hard, ví dụ chẳng hạn như tô màu đồ thị

- So khớp hoàn hảo trong đồ thị hai phía: Số các so khớp hoàn hảo của một đồ thị hai phía có thể tính bằng nguyên lý này
- Đếm số toàn ánh
- Đếm các hoán vị cấm vị trí

4. **Ý nghĩa**

Nguyên lý **bao hàm-loại trừ** có mối liên hệ mật thiết với thuật toán trong nhiều khía cạnh. Trong lập trình và phân tích thuật toán, nguyên lý này giúp chúng ta:

1. **Tối ưu hóa độ phức tạp**: Khi phân tích độ phức tạp của một thuật toán, chúng ta cần xác định số lượng các bước tính toán cần thiết để thực hiện một nhiệm vụ. Nguyên lý bao hàm-loại trừ giúp loại bỏ sự trùng lặp trong quá trình đếm, từ đó cung cấp một ước lượng chính xác hơn về độ phức tạp thời gian.
2. **Phân tích tổ hợp**: Trong các thuật toán liên quan đến tổ hợp, như thuật toán tìm đường đi trong đồ thị hoặc các bài toán tối ưu hóa tổ hợp, nguyên lý này giúp tính toán số lượng các trường hợp có thể xảy ra mà không bỏ sót hoặc đếm trùng.
3. **Xử lý dữ liệu lớn**: Khi làm việc với dữ liệu lớn, việc đếm số lượng các sự kiện duy nhất là quan trọng. Nguyên lý bao hàm-loại trừ giúp xác định số lượng sự kiện duy nhất mà không cần phải lưu trữ toàn bộ dữ liệu.
4. **Thiết kế và phân tích thuật toán**: Nguyên lý này cũng được sử dụng trong việc thiết kế các thuật toán mới, giúp nhà phát triển thuật toán xác định các trường hợp cần xử lý và loại trừ các trường hợp không cần thiết, từ đó tối ưu hóa hiệu suất của thuật toán.

5. Kết hợp giữa bao hàm loại trừ và ước chung lớn nhất

- Nguyên lý **bao hàm-loại trừ** và khái niệm **ước chung lớn nhất (ƯCLN)** có thể kết hợp với nhau trong một số bài toán toán học và lập trình. Ví dụ, khi giải quyết vấn đề liên quan đến việc đếm số lượng các số nguyên tố cùng nhau trong một phạm vi nhất định, chúng ta có thể sử dụng nguyên lý bao hàm-loại trừ để loại bỏ sự trùng lặp trong quá trình đếm, đồng thời sử dụng ƯCLN để xác định tính nguyên tố cùng nhau của các số[12].

Một ví dụ cụ thể là khi chúng ta muốn đếm số lượng các số nguyên tố cùng nhau với một số (n) trong khoảng từ 1 đến (m). Chúng ta có thể sử dụng nguyên lý bao hàm-loại trừ để tính số lượng các số không nguyên tố cùng nhau với (n), sau đó trừ đi từ tổng số lượng các số trong khoảng. Trong quá trình này, ƯCLN giúp xác định các số có ước chung với (n), và từ đó chúng ta có thể loại bỏ chúng khỏi quá trình đếm.

- **Ứng dụng kết hợp**

Một ứng dụng điển hình của việc kết hợp nguyên lý Bao hàm loại trừ và GCD là trong bài toán đếm số các số nguyên trong một khoảng nhất định chia hết cho ít nhất một trong các số đã cho.

Ví dụ: Đếm số các số nguyên từ 1 đến N chia hết cho ít nhất một trong các số

$$a_1, a_2, \dots, a_k$$

Để giải bài toán này, ta có thể sử dụng nguyên lý Bao hàm loại trừ:

1. Tập hợp các số chia hết cho a_i:

+ Gọi A_i *là tập hợp các số từ* 1 *đến* N *chia hết cho* a_i.

+ Số phần tử của A_i *là* $\left\lfloor \frac{N}{a_i} \right\rfloor$, *trong* đó $\lfloor x \rfloor$ *là hàm lấy phần nguyên của* x

2. Giao của các tập hợp

+ *Gọi* $A_{i,j}$ *là tập hợp các số chia hết cho cả* a_i *và* a_j, *số phần tử của* $A_{i,j}$ *là* $\left\lfloor \frac{N}{LCM(a_i, a_j)} \right\rfloor$,

trong đó $LCM(a_i, a_j)$ là bội chung nhỏ nhất của a_i *và* a_j

$$+ Tương\ tự\ A_{i,j,k}\ là\ tập\ hợp\ các\ số\ chia\ hết\ cho\ cả\ a_i, a_j\ và\ a_k,$$

$$số\ phần\ tử\ của\ nó\ là\ \left\lfloor \frac{N}{LCM(a_i, a_j, a_k)} \right\rfloor$$

3. Sử dụng bao hàm loại trừ

Số các số từ 1 đến N chia hết cho ít nhất một trong các số $a_1, a_2, \dots, a_k$ là:

$$\left| \bigcup_{i=1}^{k} A_i \right| = \sum_{i=1}^{k} \left\lfloor \frac{N}{a_i} \right\rfloor - \sum_{1 \le i \le j \le k} \left\lfloor \frac{N}{LCM(a_i, a_j)} \right\rfloor + \sum_{1 \le i \le j \le l \le k} \left\lfloor \frac{N}{LCM(a_i, a_{j,} a_l)} \right\rfloor$$

C. Bài tập vận dụng

Bài số 1: Cho N số tự nhiên a_1, a_2, …, a_N. Hãy tính giá trị của biểu thức sau:

$$\sum_{1 \le i < j \le N} GCD(a_i, a_j)$$

Giới hạn:

- $1 \le N \le 2 * 10^5$
- $1 \le a_i \le 2 * 10^5$
- Thời gian chạy: 1.5 s
- Bộ nhớ: Tiêu chuẩn

Ví dụ:

Dữ liệu vào	Dữ liệu ra
5 2 7 8 9 7	17

Subtask:

- Subtask 1: $1 \le N \le 1000$
- Subtask 2: $1 \le N \le 8000, 1 \le a_i \le 1000$
- Subtask 3: Không có thêm ràng buộc nào cả

Lời giải: Trước hết, ta đặt hằng số $T = 2 * 10^5$ để thuận cho việc trình bày.

Không khó để thấy việc chạy bruteforce tất cả các cặp số trong bài rồi cộng lại là kém hiệu quả khi thời gian chạy của giải thuật Euclid cho mỗi cặp số là $O\left(log\left(min(a_i, a_j)\right)\right)$, vậy thời gian chạy của thuật toán bruteforce sẽ là $O(N^2 \log T)$.

Ngoài ra, với subtask 2, chúng ta có thể tạo mảng tính trước GCD cho các cặp giá trị trong khoảng 1000, và từ đó khi bruteforce sẽ loại bớt được độ phức tạp trong phần tính GCD. Tuy nhiên, đó vẫn là chưa đủ để giải quyết bài toán của chúng ta.

Do đó, sẽ hiệu quả hơn nếu chúng ta có thể tìm ra được một thuật toán hướng đến nội dung chính của bài toán, hàm ước chung lớn nhất. Do đó, ta hướng đến việc tìm $g(x)$ là số cặp có ước chung lớn nhất bằng chính xác x.

Hãy định nghĩa $f(x)$ là số các cặp sao cho ước chung lớn nhất của chúng là một bội của x. Vậy, nếu đặt d_x là số lượng bội của x trong các số đã cho, ta suy ra được:

$$f(x) = \frac{d_x(d_x - 1)}{2}$$

Và theo nguyên lý bao hàm loại trừ, ta thấy $f(x)$ không chỉ chứa $g(x)$ mà còn bao gồm $g(y)$ với y là các bội của x trong khoảng N. Vậy ta có thể suy ra được công thức sau:

$$g(x) = f(x) - \sum_{x|y;\, x<y\leq T} g(y)$$

Và sau khi đã tính xong $g(x)$, ta có thể tính được đáp án bài toán bằng công thức sau:

$$\sum_{i=1}^{T} i * g(i)$$

Tuy nhiên, bài toán có 3 vấn đề cần giải quyết:

1. Làm sao để tính mảng d_x ?

Chúng ta sẽ xây dựng vector $divisor[x]$ chứa tất cả các ước của x bằng thuật toán như sau:

```
1. const int T = 200000;
2. vector<int> divisor[T + 1];
3. for (int i = 1; i <= T; i ++) {
4.   for (int j = i; j <= T; j += i) {
5.     divisor[j].push_back(i);
6.   }
7. }
```

Trong code trên, chúng ta chạy qua mọi biến i có thể và push nó vào các vector chứa ước của j là bội của i. Và mỗi vòng for sẽ chạy $\left\lfloor \frac{T}{i} \right\rfloor$ lần, dẫn đến độ phức tạp tổng của chúng ta là:

$$O\left(\sum_{i=1}^{T} \left\lfloor \frac{T}{i} \right\rfloor\right) \leq O\left(\sum_{i=1}^{T} \frac{T}{i}\right) = O(T * (\sum_{i=1}^{T} \frac{1}{i}))$$

Để ý $\sum_{i=1}^{\infty} \frac{1}{i}$ chính là chuỗi điều hòa. Khi đó, tổng riêng thứ T của chuỗi điều hòa, hay còn được gọi là số điều hòa thứ T là:

$$H_T = \sum_{i=1}^{T} \frac{1}{i}$$

Suy ra được:

$$O\left(\sum_{i=1}^{T} \left\lfloor \frac{T}{i} \right\rfloor\right) \leq O(T * H_T)$$

Chúng ta sẽ đề cập kết quả trên sau. Và khi ta duyệt qua mỗi số đã cho, ta cộng 1 vào d_u với u là ước của số đã cho trong vector divisor đã tính trước. Không khó để tìm ra trong khoảng T đổ lại, số có số ước lớn nhất là 196560 với 160 ước. Và điều này là ổn với thời gian chạy.

2. Làm sao để tính $g(x)$ cho hiệu quả ?

Để ý kết quả của $g(x)$ luôn chịu ảnh hưởng của các $g(y)$ với y là bội của x và lớn hơn nó, chúng ta có thể for ngược để đảm bảo việc tính toán được thuận lợi. Khi đó, mọi kết quả sẽ được cập nhật theo thứ tự tính toán. Ta có thể giải bằng phần code sau:

```
1. for (int i = T; i >= 1; i --) {
2.   f[i] = d[i] * (d[i] - 1) / 2;
3.   g[i] = f[i];
4.   for (int j = i + i; j <= T; j += i)
5.     g[i] -= g[j];
6. }
```

Và phần này cũng sử dụng hai vòng for như phần trước, cũng có độ phức tạp là:

$$O\left(\sum_{i=1}^{T} \left\lfloor \frac{T}{i} \right\rfloor\right) \leq O\left(\sum_{i=1}^{T} \frac{T}{i}\right) = O\left(T * \left(\sum_{i=1}^{T} \frac{1}{i}\right)\right) = O(T * H_T)$$

3. Xác định H_T *thế nào* ?

Công thức này đã được chứng minh là có độ phức tạp xấp xỉ sau[1]:

$H_T \approx \ln T + \gamma \approx \ln T$
Từ đó, ta có:

[1] Donald E. Knuth, "The Art of Computer Programming," Volume 1 (Fundamental Algorithms), 1968, 1.2.11.2

$$O\left(\sum_{i=1}^{T}\left\lfloor\frac{T}{i}\right\rfloor\right) \le O\left(T * \left(\sum_{i=1}^{T}\frac{1}{i}\right)\right) = O(T * H_T) \approx O(T * \ln T)$$

Kết quả này sẽ còn sử dụng tiếp nhiều lần và là một trong những mấu chốt của kỹ thuật này.

Kết luận: Độ phức tạp xấp xỉ của lời giải là $O(T * \ln T) + O(160 * T)$.

Code mẫu:

```
#include "bits/stdc++.h"
using namespace std;
const int T = 200000;
vector<int> divisors[T + 1];
long long g[T + 1], d[T + 1];
signed main() {
  freopen(".\\input\\123.txt", "r", stdin);
  int n;
  cin >> n;
vector<int> a(n);
for (int &u : a) cin >> u;
for (int i = 1; i <= T; i ++)
  for (int j = i; j <= T; j += i)
    divisors[j].push_back(i);
for (int &u : a)
  for (int &v : divisors[u])
    d[v] ++;
long long ans = 0;
for (int i = T; i >= 1; i --) {
  g[i] = d[i] * (d[i] - 1) / 2;
  for (int j = i + i; j <= T; j += i)
    g[i] -= g[j];
  ans += g[i] * i;
}
cout << ans;
}
```

Bình luận: Mặc dù nhìn code rất ngắn và đơn giản, ý tưởng của code này lại được sử dụng rất nhiều trong các bài tập liên quan đến tính toán và tổ hợp có liên quan đến GCD và LCM. Điểm đặc biệt của nó là sự dễ cài đặt và không cần kiến thức quá cao siêu. Bên cạnh đó, nó còn được sử dụng như một giải pháp thay thế so với hàm Mobius.

Các bài tập có liên quan sẽ xoay quanh câu chuyện chúng ta định nghĩa và tính toán hàm $f(x)$ và $g(x)$ ra sao.

Bài số 2

Đề bài: Hôm nay là ngày V tháng O tại hành tinh I, và ở đây diễn ra kỳ thi VOI dành cho các học sinh trung học cơ sở để thử sức. Tại kỳ thi VOI năm nay, sẽ có Y học sinh được tranh tài với nhau. Ban tổ chức đang cân nhắc không biết nên dùng bao phòng thi và mỗi phòng phân bao người.

Hiện tại, hãy coi như sẽ chọn n phòng làm phòng thi, mỗi phòng sẽ có a_i thí sinh dự thi tham dự. Các thí sinh sẽ vào phòng thi theo số báo danh nên chỉ số lượng thí sinh trong các phòng thi được đánh số sẽ quan trọng. Vì năm nay đang bị dịch bệnh COVID-X hoành hành, ban tổ chức muốn đảm bảo an toàn cho thí sinh và do đó, kỳ thi sẽ không êm xuôi nếu cả 2 điều kiện sau không diễn ra cùng lúc:

- $GCD(a_1, a_2, \dots, a_n) = X$
- $a_1 + a_2 + \cdots + a_n = Y$

Khi này, ban tổ chức băn khoăn không biết có bao cách để chọn số phòng thi cũng như số thí sinh được phát cho mỗi phòng. Biết hai cách chia là riêng biệt nếu hai cách chia dùng số phòng khác nhau hoặc ở phòng bất kỳ được đánh cùng số có số học sinh khác nhau.

Bạn đang hi vọng một ngày sẽ được tham gia vào ban tổ chức và nhận thấy đây là cơ hội để chứng tỏ bản thân. Hãy giúp ban tổ chức tính ra đáp án dưới modulo $10^9 + 7$.

Tóm lược đề bài: Hãy đếm số dãy phân biệt gồm các số nguyên dương $a_1, a_2, \dots, a_n$ sao cho $GCD(a_1, a_2, \dots, a_n) = X$ và $a_1 + a_2 + \cdots + a_n = Y$. In ra đáp án ở modulo $10^9 + 7$.

Giới hạn:

- $1 \leq X, Y \leq 10^9$
- Thời gian chạy: 1s
- Bộ nhớ tiêu chuẩn

Ví dụ:

Dữ liệu vào	Dữ liệu ra
3 9	3

Subtask:

- Subtask 1: $Y \leq 20$
- Subtask 2: $X, Y \leq 60$
- Subtasl 3: Không có ràng buộc nào thêm

Lời giải: Trước hết, ta đặt $T = 10^9$ để thuận cho trình bày.

Nhìn vào bài toán, với subtask 1 chúng ta có thể dùng backtrack để sinh ra tất cả các dãy có thể và kiểm tra.

Khi tiếp cận subtask 2, hoàn toàn chúng ta có thể xây dựng một thuật toán quy hoạch động như sau:

$$DP[i][sum][g] = Số\ dãy\ có\ i\ số, tổng\ bằng\ sum\ và\ GCD\ bằng\ g.$$

Tuy nhiên, không khó để thấy một thuật toán như vậy chạy rất không hiệu quả với một giới hạn lớn như T. Do đó, ta hướng đến một cách giải mang tính số học hơn.

Trước hết, không khó để thấy $X|a_i$ dẫn đến $X|Y$, do đó, nếu Y không chia hết cho X, bài toán lập tức trả về đáp án bằng 0.

Với ý tưởng như trước, ta sẽ cố gắng xác định một hàm $f(x)$ và $g(x)$ để sử dụng lại ý tưởng như trên.

Một cách tương tự, ta tính $g(x)$ $là\ đáp\ án\ bài\ toán\ nếu\ X = x$ và $f(x)$ sẽ là số các dãy sao cho GCD toàn dãy chia hết cho x.

Không khó để nhận ra sự tương đương với bài toán trước, ta có công thức sau:

$$g(x) = f(x) - \sum_{x|y;\ x<y\le T} g(y)$$

Và khi đó đáp án bài toán sẽ là $g(X)$.

Vấn đề ở đây là chúng ta sẽ giải quyết $f(x)$ như thế nào. Nếu Y không chia hết cho x, $f(x)$ lập tức bằng 0. Nếu không, ta đặt $Y = xZ$ và $a_i = xb_i$. Khi đó bài toán trở về tìm số dãy $b_1, b_2, \dots, b_n$ sao cho có tổng bằng Z. Tới đây, chúng ta áp dụng một ý tưởng có trong bài toán chia một số vật cho một số người.

Tưởng tượng có Z vật, ta có thể nhét hoặc không nhét $Z - 1$ tấm ngăn cách giữa 2 vật. Khi đó, dãy các vật liền nhau mà không có lớp ngăn sẽ hợp thành một số, và dãy số đó ứng với một dãy khả thi của chúng ta. Vì có $Z - 1$ tấm ngăn, chúng ta có 2^{Z-1} cách để xây dựng dãy. Do đó

$$f(x) = 2^{Z-1} = 2^{\binom{Y}{x}-1}$$

Với Z lớn, việc tính số mũ có thể được cải thiện lên thành $log2(Z)$ bằng phương pháp mũ lũy thừa nhanh.

Tuy nhiên, có quá nhiều giá trị của x có thể chọn. Do đó, ta giới hạn số các giá trị có thể của x xuống bằng cách chỉ chọn các ước của Y. Sử dụng phương pháp lấy ước trong thời gian $\sqrt{T}$. Bên cạnh đó, chúng ta biết được số có nhiều ước số nhất trong khoảng 10^9 có 1344, nên 2 vòng for của chúng ta hoàn toàn vừa vặn thời gian chạy.

Code mẫu:

```
#include "bits/stdc++.h"
using namespace std;
#define int long long
const int mod = 1000000007;
inline int power(int x, int y) {
  int r = 1;
  while (y) {
    if (y & 1) r = (r * x) % mod;
    x = (x * x) % mod;
    y >>= 1;
  }
  return r;
}
signed main() {
  int x, y;
  cin >> x >> y;
  if (y % x) {
    cout << 0;
    return 0;
  }
  vector<int> d;
  for (int i = 1; i * i <= y; i ++) {
    if (y % i == 0) {
      d.push_back(i);
      if (i * i < y) {
        d.push_back(y / i);
      }
    }
  }
  sort(d.rbegin(), d.rend());
  while (d.size() && d.back() < x) d.pop_back();
  int D = d.size();
  vector<int> f(D);
  for (int i = 0; i < D; i ++) {
    f[i] = power(2, y / d[i] - 1);
    for (int j = 0; j < i; j ++)
      if (d[j] % d[i] == 0) {
        f[i] = (f[i] - f[j] + mod * mod) % mod;
      }
  }
  cout << f.back();
}
```

Kết luận: Bài toán có độ phức tạp $O\left(1344^2 + 1344 * log2(T) + \sqrt{T}\right)$.
Bình luận: Đây là một khía cạnh mới mẻ của bài toán bằng cách vận dụng các kiến thức liên quan trong số học.

Bài số 3
Đề bài: Đã một năm trôi qua kể từ đại dịch COVID-X xuất hiện. Rất may kỳ thi năm đó ban tổ chức đã giải quyết được vấn đề chia phòng nhờ bạn. Nhưng năm nay đại dịch COVID-X đã có biến thể mới là COVID-LCM còn mạnh hơn trước.

Ở khu bạn sống, có n người sống cùng nhau. Biết người thứ i đã tiêm tổng cộng là a_i liều vaccine theo chỉ đạo của chính phủ. Bạn được cho biết rằng khi hai người i và j tiếp xúc thì độ an toàn của cặp đó là $LCM(a_i, a_j)$ vì càng tiêm nhiều vaccine thì càng an toàn. Và độ an toàn của một khu được tính bằng tổng của tất cả độ an toàn của tất cả các cặp người sống ở đó.

Băn khoăn không biết khu mình sống đã đủ an toàn chưa, bạn muốn tính độ an toàn của khu vực mình. Hãy dùng kỹ năng của mình để tính nó ra ở modulo 998244353.
Tóm lược đề bài:

Cho N số tự nhiên a_1, a_2, …, a_N. Hãy tính giá trị của biểu thức sau ở modulo 998244353:

$$\sum_{1 \le i < j \le N} LCM(a_i, a_j)$$

Giới hạn:

- $1 \le N \le 2 * 10^5$
- $1 \le a_i \le 10^6$
- Thời gian chạy: 2s
- Bộ nhớ tiêu chuẩn

Ví dụ:

Dữ liệu vào	Dữ liệu ra
5 10 7 4 3 3	73

Subtask:
- Subtask 1: $1 \le N \le 1000$
- Subtask 2: $1 \le N \le 8000, 1 \le a_i \le 1000$
- Subtask 3: Không có thêm ràng buộc nào cả

Lời giải: Đặt $T = 10^6$ để thuận tiện trình bày.
Đây có vẻ là một bài tương tự với bài 1, tuy chỉ khác ở hàm LCM thay vì GCD. Một lần nữa, bruteforce hoàn toàn không khả thi và chúng ta phải hướng đến một lựa chọn khác.

Các subtask 1 và 2 có thể giải bằng ý tưởng đã nêu.

Tiếp tục theo ý tưởng cũ, ta cố gắng xuất hiện hàm $f(x)$ và $g(x)$ hợp lý.

Một cách tự nhiên, ta có thể nghĩ đến tính chất $LCM(A,B) = \frac{A*B}{GCD(A,B)}$.

Vậy, ta sẽ cố gắng đưa nó về giải với GCD. Ta xuất hiện hàm $f(x)$ và $g(x)$ như sau:

$$g(x) = \sum_{1 \le i < j \le N, x = GCD(a_i, a_j)} a_i * a_j$$

$$f(x) = \sum_{1 \le i < j \le N, x | GCD(a_i, a_j)} a_i * a_j$$

Vậy ta có thể áp dụng nguyên ý tưởng cũ vào và có được mối tương quan $f(x)$ và $g(x)$ như sau:

$$g(x) = f(x) - \sum_{x|y;\, x<y \le T} g(y)$$

Khi đó, đáp án bài toán sẽ có công thức là:

$$\sum_{i=1}^{T} \frac{g(i)}{i}$$

Tuy nhiên, mấu chốt của bài toán nằm ở việc ta xử lý $f(x)$ ra sao. Để ý phép biến đổi sau:

$$f(x) = \sum_{1 \le i < j \le N, x | GCD(a_i, a_j)} a_i * a_j = \frac{\left(\sum_{x|a_i} a_i\right)^2 - \sum_{x|a_i} a_i^2}{2}$$

Tới đây, nhận ra hai tổng $\left(\sum_{x|a_i} a_i\right)^2$, $\sum_{x|a_i} a_i^2$ có thể tính toán trong vòng for lấy ước, ta có thể giải $f(x)$ và từ đó giải được toàn bộ bài toán.
Code mẫu:

```
#include "bits/stdc++.h"
using namespace std;
#define int long long
const int N = 200005;
const int mod = 998244353;
const int T = 1000001;
int power(int x, int y) {
  int r = 1;
  while (y) {
   if (y & 1) r = (r * x) % mod;
   x = (x * x) % mod;
   y >>= 1;
 }
```

```
14.  return r;
15.}
16.signed main() {
17.  int n;
18.  cin >> n;
19.  vector<int> a(n);
20.  vector<int> g(T), s(T);
21.  int i2 = (mod + 1) / 2;
22.  for (int &u : a) {
23.    cin >> u;
24.    for (int v = 1, v2; v * v <= u; v ++) {
25.      if (u % v == 0) {
26.        g[v] += u;
27.        g[v] %= mod;
28.        s[v] += u * u;
29.        s[v] %= mod;
30.        if (v * v < u) {
31.          v2 = u / v;
32.          g[v2] += u;
33.          g[v2] %= mod;
34.          s[v2] += u * u;
35.          s[v2] %= mod;
36.        }
37.      }
38.    }
39.  }
40.  int ans = 0;
41.  for (int i = T - 1; i >= 1; i --) {
42.    g[i] = (g[i] * g[i] - s[i]) % mod;
43.    g[i] *= i2;
44.    g[i] %= mod;
45.    if (g[i] < 0) g[i] += mod;
46.    for (int j = i + i; j < T; j += i) {
47.      g[i] -= g[j];
48.      if (g[i] < 0) g[i] += mod;
49.    }
50.    ans += g[i] * power(i, mod - 2);
51.    ans %= mod;
52.  }
53.  cout << ans;
54.}
```

Kết luận: Lời giải chúng ta có độ phức tạp $O(T \ln T + N\sqrt{T})$.
Bình luận: Đây là một bài tập có ý tưởng khá giống với bài 1, nhưng yêu cầu thêm kiến thức số học cũng như các kỹ năng đại số trong biến đổi.

Bài số 4:
Đề bài: Rất may mắn trong đại dịch, độ an toàn của khu bạn sống là rất tốt và được gọi là vùng xanh. Tuy nhiên vẫn còn một số vùng chưa xác định được. Chính phủ biết bạn là một lập trình viên giỏi và nhờ bạn xác định xem còn bao vùng bị gọi là nguy hiểm ở một số thành phố.

Ở trong thành phố N có $N-1$ vùng với chỉ số lần lượt là $2, 3, \dots, N$. Gọi chỉ số của một vùng là x. Biết khi phân tích thành các thừa số nguyên tố, ta sẽ có $x = p_1^{k_1} * p_2^{k_2} * \dots * p_z^{k_z}$. Số x được gọi là số bất khả căn nếu $GCD(k_1, k_2, \dots, k_n) = 1$. Và các vùng có chỉ số là số bất khả căn sẽ bị coi là nguy hiểm và lập tức cần tăng cường hỗ trợ.

Để có thể chứng minh năng lực của mình, bạn hãy tính số lượng vùng nguy hiểm của M thành phố được giao với mỗi thành phố có một N khác nhau.

Tóm tắt đề bài:

Cho một số nguyên dương x. Biết khi phân tích thành các thừa số nguyên tố, ta sẽ có $x = p_1^{k_1} * p_2^{k_2} * \dots * p_z^{k_z}$. Số x được gọi là số bất khả căn nếu $GCD(k_1, k_2, \dots, k_n) = 1$. Hãy đếm tất cả các số bất khả căng từ 2 đến N. Bài có M truy vấn, và phải giải riêng biệt với mỗi N.

Giới hạn:

- $1 \le N \le 10^{18}$
- $1 \le M \le 10^4$
- Thời gian chạy: 2s
- Bộ nhớ tiêu chuẩn

Ví dụ:

Dữ liệu vào	Dữ liệu ra
1 100000	99633

Subtask:
- Subtask 1: $M = 1, N \le 10^4$
- Subtask 2: $M \le 10^4, N \le 10^6$
- Subtask 3: Không có ràng buộc nào khác

Lời giải: Ta đặt $T = 10^{18}$ để thuận tiện cho việc giải bài toán.
Không khó để nhìn ra subtask 1 yêu cầu chúng ta for tất cả các số và kiểm tra bằng cách chạy phương pháp phân tích thừa số nguyên tố trong thời gian $O(\sqrt{N})$, dẫn đến độ phức tạp là $O(N\sqrt{N})$.

Để giải quyết subtask 2, chúng ta cần cách để phân tích thừa số nguyên tố nhanh hơn nữa. Ở đây, chúng ta có thể sử dụng kỹ thuật phân tích thừa số nguyên tố trong thời gian tuyến tính[2] và sau đó lưu trước đáp án vào một mảng hằng số để trả lời truy vấn trong $O(1)$. Dẫn đến độ phức tạp là $O(N \ln N)$.

Với subtask cuối, không khó để nhận ra chúng ta không thể lưu trước đáp án, và cần có một cách giải trong thời gian đủ nhanh vì số lượng truy vấn quá lớn. Nhận thấy có yếu tố GCD trong đề bài, ta một lần nữa tìm cách định nghĩa hai hàm $f(x)$ và $g(x)$ để có thể tái sử dụng ý tưởng cũ.

Hãy suy xét lại đề bài. Xét với một số dương x, ta đặt $GCD(k_1, k_2, \dots, k_z) = y = d * y'$. Vậy $d|k_i$ và dẫn đến $k_i = k_i' * d$.

Khi đó:

$$x = \prod p_i^{k_i} = \prod p_i^{k_i' * d} = (\prod p_i^{k_i'})^d = x'^d$$

Mà theo đề, ta hướng đến các $x \leq n$, dẫn đến $x' \leq \sqrt[d]{n}$. Và với mọi x' thỏa mãn điều kiện, chúng ta có thể tạo ra một x có chắc chắn GCD các số mũ của cơ số nguyên tố chia hết cho d. Phát biểu theo cách khác, đó là số cách số khai căn bậc d vẫn cho ta một số nguyên, tạm gọi là số d-phương. Và có $\lfloor \sqrt[d]{n} \rfloor$ số nguyên như vậy. Tuy nhiên ta sẽ bỏ qua số 1 vì đề yêu cầu tính các đáp án từ 2, do đó, ta có cách định nghĩa hàm $f(d)$ và $g(d)$ như sau. $f(d) =$ số các số d-phương bé hơn n và lớn hơn 1, $g(d)$ = số các số x có $y = d$. Do đó, ta có:

$$f(d) = \lfloor \sqrt[d]{n} \rfloor - 1$$

Đến đây, ta áp dụng lại ý tưởng cũ và có thể tính được $g(d)$ như sau:

$$g(d) = f(d) - \sum_{d|k;\, d<k} g(k)$$

Tuy nhiên, ở đây chúng ta vẫn chưa giới hạn được d để giải quyết bài toán. Ta để ý rằng cơ số nguyên tố bé nhất có thể là 2, dẫn đến:

$$d_{max} \leq k_{max} \leq \lfloor \log_2 10^{18} \rfloor = 60$$

Vậy ta đã giới hạn được d_{max}, mỗi truy vấn giờ đã có thể được giải quyết trong độ phức tạp $O(d_{max} \ln d_{max})$.

Bên cạnh đó, thao tác tìm căn bậc d của n cũng quan trọng không kém, có nhiều cách để xử lý. Ở đây, chúng ta có thể sử dụng hàm pow của c++ cùng kiểu dữ liệu long double. Việc còn lại là khéo léo trong kiểm soát epsilon của kiểu dữ liệu double để đưa ra kết quả chính xác.

[2] https://cp-algorithms.web.app/algebra/prime-sieve-linear.html

Code mẫu:

```
#include "bits/stdc++.h"
using namespace std;
#define int long long
inline int power(int x, int y) {
  if (!y) return 1;
  int r = x;
  y --;
  while (y) {
    if (y & 1) r *= x;
  x *= x;
  y >>= 1;
 }
 return r;
}
int kthroot(int n, int k) {
 int r = pow((long double) n, ((long double) (1.0)) / k) + 3;
 while (r && (pow((long double) r, (long double) k) > 2e18 || power(r, k) > n)) r --;
 return r;
}
signed main() {
 ios_base::sync_with_stdio(0); cin.tie(0);
 int tc = 1;
 cin >> tc;
 while (tc --) {
  int n;
  cin >> n;
  int ans = 0;
  vector<int> f(61);
  for (int i = 60; i >= 1; i --) {
   f[i] = kthroot(n, i) - 1;
   for (int j = i + i; j <= 60; j += i) {
    f[i] -= f[j];
   }
  }
  cout << f[1] << endl;
 }
}
```

Kết luận: Độ phức tạp của bài toán là: $O(M * 60 * \ln 60)$.

Bình luận: Đây là một bài tập cần có những quan sát số học kỹ lưỡng so với những bài trước, đồng thời một lần nữa áp dụng tư tưởng của toàn chuyên đề rất hợp lý.

Bài số 5

Đề bài: Thật đáng tiếc dịch bệnh đang bùng phát quá mạnh ở thành phố X là một trung tâm kinh tế của nước bạn. Các thành phố lân cận đang gửi các bác sĩ giỏi nhất đến tiếp viện để giúp thành phố X dập dịch. Và thành phố bạn cũng không phải ngoại lệ.
Thành phố bạn có n bác sĩ với chuyên môn được biểu diễn qua các số $a_1, a_2, \dots, a_n$. Một nhóm bác sĩ sẽ được gọi là hoàn hảo để đi hỗ trợ dập dịch nếu như GCD của toàn bộ chuyên môn của họ bằng chính xác 1.
Các nhà điều hành ở thành phố bạn đang muốn gửi bác sĩ để hỗ trợ thành phố X và không biết liệu có thể tìm ra được một đội hình bác sĩ phù hợp để gửi vào không. Nếu gửi được thì số lượng bác sĩ tối thiểu có thể gửi sao cho đội hình vẫn an toàn là bao nhiêu để hạn chế rủi ro.
Là một lập trình viên uy tín, bạn hãy giải quyết bài toán kia để giúp các bác sĩ có thể nhanh chóng chuẩn bị đi hỗ trợ.
Tóm tắt đề bài:

Cho n số nguyên dương $a_1, a_2, \dots, a_n$. Hãy kiểm tra xem có tồn tại cách chọn một số các phần tử của dãy sao cho chúng có GCD bằng 1 không. Nếu có, in ra số phần tử nhỏ nhất cần chọn. Nếu không, in ra -1.
Giới hạn:

- $1 \leq n \leq 3 * 10^5$
- $1 \leq a_i \leq 3 * 10^5$
- Thời gian chạy: 3s
- Bộ nhớ tiêu chuẩn

Ví dụ:

Dữ liệu vào	Dữ liệu ra
5 26 28 30 18 17	2

Subtask:

- Subtask 1: $n \leq 20$
- Subtask 2: $n \leq 100$
- Subtask 3: Không còn giới hạn nào.

Lời giải: Chúng ta đặt $T = 3 * 10^5$ để tiện cho việc trình bày.
Không khó để thấy bằng bruteforce sinh ra tất cả các tập con có thể, chúng ta có thể giải quyết subtask 1 rất nhanh gọn.
Sang đến subtask 2, để ý giới hạn rất vừa phải, ta có thể nghĩ tới công thức quy hoạch động như sau: $DP[i][j] =$ số phần tử tối thiểu phải chọn từ i phần tử đầu tiên sao cho GCD của chúng bằng chính xác j. Chúng ta sẽ có độ phức tạp của giải thuật này là $O(n * T * \log_2 T)$.
Tuy nhiên, để có trọn vẹn điểm bài toán, ta cần có những quan sát kỹ càng hơn.

Đầu tiên, ta nhận định sẽ không giải được nếu GCD của toàn bộ các phần tử lớn hơn 1. Khi đó sẽ không có một tập con nào có GCD thỏa mãn thứ ta cần tìm.
Thứ hai, nếu có một tập con thỏa mãn, thì độ lớn của nó sẽ tối đa là 7. Nguyên nhân là do số có nhiều cơ số nguyên tố nhất nhỏ hơn T có 7 cơ số nguyên tố trong thành phần của nó. Để có GCD các phần tử đều bằng một, mỗi phần tử sẽ phải góp phần triệt giảm đi một cơ số so với các số khác cùng tập, do đó để triệt hạ tối đa 7 cơ số ta cần tối đa 7 số.
Khi ta đã giới hạn được độ lớn của tập, ta sẽ thử nghĩ xem với tập có độ lớn là sz, liệu ta có thể tìm được một tập thỏa mãn yêu cầu đề bài hay không. Bằng cách áp dụng ý tưởng sẵn có trong lớp bài tập này, ta sẽ cố gắng định nghĩa hàm $f(x)$ và $g(x)$ để có được thông tin ta cần. Để ý ta đang tìm kiếm xem liệu có tồn tại tập thỏa mãn, ta sẽ cố đếm số tập có GCD là một số cụ thể.

Một cách tự nhiên, ta có thể nghĩ tới như sau:

$$g(x) = số\ tập\ độ\ lớn\ sz\ có\ GCD\ bằng\ chính\ xác\ x$$

$$f(x) = số\ tập\ độ\ lớn\ sz\ có\ GCD\ là\ bội\ của\ x$$

Và khi đó, bằng cách áp dụng ý tưởng sẵn có, ta giải được $g(x)$ như sau:

$$g(x) = f(x) - \sum_{x|y;\ x<y\le T} g(y)$$

Tuy nhiên vấn đề ở đây là ta tính $f(x)$ ra sao. Gọi d_x là số phần tử là bội của x trong các số đã cho. Vậy nếu ta chọn đúng sz phần tử từ d_x phần tử là bội của x, ta sẽ tạo được một tập có GCD là bội của x. Do đó, ta có:

$$f(x) = \binom{d_x}{sz}$$

Từ đó, ta có thể giải quyết được $g(x)$. Vì phép tính tổ hợp rất lớn, chúng ta có thể sử dụng phép tổ hợp lấy mod, và cuối cùng kiểm tra xem $g(1)$ có bằng 0 hay không. Thao tác chuẩn bị lũy thừa mod có thể giải quyết trong $O(n)$.

Code mẫu:

```
#include "bits/stdc++.h"
using namespace std;
#define int long long
const int T = 300001;
const int mod = 999999937;
int f[T], rf[T], n, a[T], g[T], d[T];
int mul(int x, int y) {
  return x * y % mod;
}
void add(int &x, int y) {
 x += y; if (x >= mod) x -= mod;
```

```
12.}
13.int power(int x, int y) {
14.  int r = 1;
15.  while (y) {
16.    if (y & 1) r = mul(r, x);
17.    y >>= 1;
18.    x = mul(x, x);
19.  }
20.  return r;
21.}
22.int C(int N, int K) {
23.  if (K > N) return 0;
24.  return mul(f[N], mul(rf[K], rf[N - K]));
25.}
26.signed main() {
27.  ios_base::sync_with_stdio(0); cin.tie(0);
28.  cin >> n;
29.  int G = 0;
30.  for (int i = 1; i <= n; i ++) {
31.    cin >> a[i];
32.    G = __gcd(G, a[i]);
33.    for (int j = 1; j * j <= a[i]; j ++) {
34.      if (a[i] % j == 0) {
35.        d[j] ++;
36.        if (j * j < a[i]) d[a[i] / j] ++;
37.      }
38.    }
39.  }
40.  if (G > 1) {
41.    cout << -1;
42.    return 0;
43.  }
44.  f[0] = 1;
45.  for (int i = 1; i < T; i ++)
46.    f[i] = mul(f[i - 1], i);
47.  rf[T - 1] = power(f[T - 1], mod - 2);
48.  for (int i = T - 2; i >= 0; i --)
49.    rf[i] = mul(rf[i + 1], i + 1);
50.  for (int sz = 1; sz <= 7; sz ++) {
51.    for (int i = T - 1; i; i --) {
52.      g[i] = C(d[i], sz);
53.      for (int j = i + i; j < T; j += i)
54.        add(g[i], mod - g[j]);
55.    }
```

```
56.   if (g[1] != 0) {
57.     cout << sz;
58.     return 0;
59.   }
60. }
61.}
```

Kết luận: Độ phức tạp của giải thuật: $O(7 * T * \ln T)$
Bình luận: Đây là một bài tập khá hay để luyện thêm tư duy kỹ thuật này. Bên cạnh đó, còn một số lời giải sử dụng kỹ thuật các thuật toán ngẫu nhiên randomization heuristics đáng để tham khảo.

Bài số 6:
Đề bài: Có vẻ dịch bệnh lại tiếp tục quay trở lại thành phố của bạn với biến thế mạnh hơn, và lần này mọi thứ trở nên đáng quan ngại hơn trước. Lần này, bạn quyết định tham mưu cho các lãnh đạo về tình hình dịch bệnh dựa trên khả năng lập trình của bạn. Thành phố của bạn có n huyện, và số người nhiễm bệnh tại từng huyện lần lượt là $a_1, a_2, \dots, a_n$. Trong các tình huống có thể xảy ra, chỉ các tình huống đảm bảo cả 3 điều kiện sau là an toàn và có thể kiểm soát được:

- Với mọi i $(1 \le i \le n)$ ta có $l_i \le a_i \le r_i$
- $\sum a_i \le m$
- $GCD(a_1, a_2, \dots, a_n) = 1$

Để có thể triển khai các phương án chống dịch hiệu quả, chính phủ cần biết có bao nhiêu trường hợp là sẽ an toàn. Bằng khả năng tính toán của mình, hãy tìm ra đáp án dưới modulo 998244353.
Tóm lược đề bài:

Cho dãy n phần tử $a_1, a_2, \dots, a_n$. Có bao nhiêu dãy thỏa mãn 3 điều kiện sau:

- Với mọi i $(1 \le i \le n)$ ta có $l_i \le a_i \le r_i$
- $\sum a_i \le m$
- $GCD(a_1, a_2, \dots, a_n) = 1$

In ra đáp án dưới modulo 998244353.

Giới hạn:

- $1 \le n \le 50$
- $1 \le m \le 10^5$
- $0 \le l_i \le r_i \le m$
- Thời gian chạy: 2s
- Bộ nhớ: Tiêu chuẩn

Ví dụ:

Dữ liệu vào	Dữ liệu ra
5 20 1 10 1 20 1 9 1 1 1 3	1770

Subtask:

- Subtask 1: $m \leq 50$
- Subtask 2: Không có thêm ràng buộc nào

Lời giải:

Để giải quyết subtask 1, chúng ta có thể nghĩ ngay tới trạng thái quy hoạch động như sau: $DP[i][j][k] =$ số cách để tạo được dãy với i phần tử đầu tiên, tổng là j và GCD của chúng là k. Đây là lời giải khá dễ suy ra và có độ phức tạp vừa phải.

Để giải quyết subtask 2, ta lại áp dụng ý tưởng cũ vào, cố gắng làm xuất hiện hai hàm $f(x)$ và $g(x)$ như sau:

$$g(x) = số\ dãy\ thỏa\ mãn\ điều\ kiện\ 1, 2\ và\ có\ GCD\ là\ x\ ở\ điều\ kiện\ 3$$

$$f(x) = \ số\ dãy\ thỏa\ mãn\ điều\ kiện\ 1, 2\ và\ có\ GCD\ là\ bội\ x\ ở\ điều\ kiện\ 3$$

Từ đó không khó để thấy ta có thể giải ra $g(x)$ bằng ý tưởng cũ:

$$g(x) = f(x) - \sum_{x|y;\ x<y\leq m} g(y)$$

Và một lần nữa, vấn đề là ta sẽ giải quyết $f(x)$ ra sao. Ta có $x|a_i$ nên:
$a_i = x * a_i'$.

Thay vào điều kiện 1 của bài toán, ta có:

$$l_i \leq x *\ a_i' \leq r_i$$

Suy ra được:

$$\left\lceil \frac{l_i}{x} \right\rceil \leq a_i' \leq \left\lfloor \frac{r_i}{x} \right\rfloor$$

Thay vào điều kiện 2 của bài toán, ta suy ra được:

$$\sum a_i' \leq \left\lfloor \frac{m}{x} \right\rfloor$$

Vậy ta đã có bài toán mới là đếm số dãy a'_i sao cho $\left\lceil \frac{l_i}{x} \right\rceil \le a'_i \le \left\lfloor \frac{r_i}{x} \right\rfloor$ và $\sum a'_i \le \left\lfloor \frac{m}{x} \right\rfloor$ cùng xảy ra. Đến đây, điều kiện GCD ràng buộc không còn hiện hữu, ta sẽ giải bài toán mới bằng quy hoạch động.

Ta có $DP[i][j]$ bằng số cách tạo dãy a'_i được i phần tử đầu tiên và đã có tổng bằng j. Khi đó, công thức của trạng thái quy hoạch động là:

$$DP[i][j] = \sum_{t=l[i]}^{r[i]} DP[i-1][j-t]$$

Cuối cùng, $f(x) = \sum_{i=1}^{\left\lfloor \frac{m}{x} \right\rfloor} DP[n][i]$.

Và khi đó, độ phức tạp của phần quy hoạch động sẽ là $O(n \left\lfloor \frac{m}{x} \right\rfloor^2)$.

Để ý công thức quy hoạch động có lấy tổng trên một mảng liên tiếp, ta có thể dùng tổng tiền tố để tối ưu sự chuyển tiếp trạng thái về $O(1)$ và khi đó độ phức tạp khi tính $f(x)$ sẽ còn $O(n \left\lfloor \frac{m}{x} \right\rfloor)$.

Vậy khi ta tính toàn bộ các giá trị từ $f(1)$ đến $f(m)$, độ phức tạp là:

$$O\left(\sum_{i=1}^{m} n \left\lfloor \frac{m}{x} \right\rfloor\right) \approx O\left(nm \sum_{i=1}^{m} \frac{1}{i}\right) = O(nmH_m) = O(nm \ln m)$$

Sau khi đã có $f(x)$, ta có thể tính nốt $g(x)$ và hoàn thiện bài toán.

Code mẫu:

```
#include "bits/stdc++.h"
using namespace std;
const int mod = 998244353;
int add(int x, int y) {
  x += y; if (x >= mod) x -= mod;
  return x;
}
int p[2][100001];
int get(int i, int l, int r) {
if (r < 0) return 0;
l = max(l, 0);
return add(p[i][r], mod - (l == 0 ? 0 : p[i][l - 1]));
}
int main() {
int n, m;
cin >> n >> m;
```

```
vector<int> f(m + 1);
vector<int> l(n), r(n);
for (int i = 0; i < n; i ++)
  cin >> l[i] >> r[i];
for (int d = 1; d <= m; d ++) {
  vector<int> L(n), R(n);
  bool fail = false;
  for (int i = 0; i < n; i ++) {
    L[i] = (l[i] + d - 1) / d;
    R[i] = r[i] / d;
    if (R[i] < L[i]) fail = true;
  }
  if (fail) continue;
  int bound = m / d;
  for (int i = 0; i <= 1; i ++)
    for (int j = 0; j <= bound; j ++)
      p[i][j] = 0;
  p[0][0] = 1;
  for (int i = 1; i <= bound; i ++) p[0][i] = 1;
  int cur = 0, nxt = 1;
  for (int i = 0; i < n; i ++) {
    for (int j = 0; j <= bound; j ++)
      p[nxt][j] = add(p[nxt][j], get(cur, j - R[i], j - L[i]));
    for (int j = 1; j <= bound; j ++)
      p[nxt][j] = add(p[nxt][j], p[nxt][j - 1]);
    for (int j = 0; j <= bound; j ++)
      p[cur][j] = 0;
    swap(cur, nxt);
  }
  f[d] = p[cur][bound];
}
for (int i = m; i >= 1; i --) {
  for (int j = i + i; j <= m; j += i)
    f[i] = add(f[i], mod - f[j]);
}
cout << f[1];
```

Kết luận: Độ phức tạp cuối cùng của bài là $O(nm \ln m)$.

Bình luận: Ở bài toán này, chúng ta đã vận dụng ý tưởng cũ nhưng với những kỹ thuật liên quan như quy hoạch động, cho thấy sự đa dạng trong chiến thuật giải các bài tập liên quan.

Bài số 7

Đề bài: Lần này, các biến chủng virus đang đột biến với tốc độ chóng mặt. Điều này khiến cho xã hội vô cùng lo ngại và phải tìm ra giải pháp để ngăn chặn nó. Và bạn được giao cho nhiệm vụ phân tích độ đột biến của virus.

Chuỗi gene của virus có N đoạn và mỗi đoạn thứ i được cho một chỉ số là a_i. Một phiên bản virus sẽ là một chuỗi n chỉ số a_i , và sẽ được cho là nguy hiểm nếu $GCD(a_1, a_2, \dots, a_n) = 1$. Chỉ số lớp virus thứ j là $g(j)$, với giá trị là số các chuỗi sao cho $1 \leq a_i \leq j$ với mọi $1 \leq i \leq n$ của chuỗi khi lấy $mod\ 10^9 + 7$.
Độ đột biến của virus sẽ là $\sum_{i=1}^{k}(g(i)\ xor\ i)\ (mod\ 10^9 + 7)$.

Để tìm phương án ngăn chặn dịch bệnh, trước hết ta phải nắm bắt được nó. Bằng kỹ năng của mình, bạn hãy giúp các nhà khoa học tìm ra độ đột biến của virus để đưa ra giải pháp phù hợp.

Tóm lược đề bài:

Cho $g(i)$ là số chuỗi có n phần tử và mọi phần tử đều thuộc khoảng $[1, n]$ và GCD toàn bộ các phần tử đều bằng 1 khi lấy $mod\ 10^9 + 7$. Hãy tính $\sum_{i=1}^{k} g(i)\ xor\ i\ (mod\ 10^9 + 7)$.

Giới hạn:

- $1 \leq n, k \leq 10^6$

Ví dụ:

Dữ liệu vào	Dữ liệu ra
3 4	82

Subtask:

- Subtask 1: $k = 2$
- Subtask 2: $1 \leq n, k \leq 100$
- Subtask 3: Không có giới hạn nào nữa

Lời giải:
Với subtask 1, chúng ta có thể dễ dàng thấy $g(1) = 1, g(2) = 2^n - 1$ và từ đó có thể suy ra đáp án nhanh chóng.

Với subtask 2, ta có thể giải quyết bằng cách quy hoạch động như sau:

$$DP[i][j][k] = số\ chuỗi\ với\ i\ phần\ tử, phần\ tử\ lớn\ nhất\ là\ j\ và\ GCD = k$$

Tuy nhiên, chỉ ngần đó là hoàn toàn không đủ để có thể giải quyết triệt để bài toán. Do đó chúng ta cần một quan sát giúp giải quyết bài toán hiệu quả hơn.

Gọi $f(x, y)$ là số chuỗi có n phần tử dương với phần tử lớn nhất bằng x và có $GCD = y$. Ta nhận thấy $f(xk, k) = f(x, 1) = g(x)$.

Đồng thời ta dùng bao hàm loại trừ, thấy được số các chuỗi có phần tử lớn nhất bằng x là $x^n - (x-1)^n$, dẫn đến:

$$\sum_{i=1}^{x} f(x,i) = x^n - (x-1)^n$$

Mà với mọi y mà x không chia hết cho y, $f(x,y) = 0$, ta suy được:

$$\sum_{i|x} f(x,i) = x^n - (x-1)^n$$

Dẫn đến:

$$\sum_{i|x} f(i,1) = x^n - (x-1)^n$$

Và suy ra được:

$$f(x,1) = x^n - (x-1)^n - \sum_{i|x, i<x} f(i,1)$$

Bên cạnh đó, ta có $f(1,1) = 1$, là điều kiện gốc, dẫn đến ta sử dụng công thức trên liên tục và có được mọi $f(x,i)$.
Khi này, công thức $g(x)$ có thể được tính bằng:

$$g(x) = \sum_{i=1}^{x} f(x,i) = g(x-1) + f(x,i)$$

Sau khi lần lượt tính các $g(x)$, ta có thể dễ dàng tính được đáp án.

Code mẫu:

```
#include "bits/stdc++.h"
using namespace std;
#define int long long
const int mod = 1000000007;
inline int power(int x, int y) {
  int r = 1;
  while (y) {
    if (y & 1) r = (r * x) % mod;
    x = (x * x) % mod;
    y >>= 1;
  }
  return r;
}
```

```
signed main() {
  int n, k;
  cin >> n >> k;
  vector<int> f(k + 1);
  int ans = 0;
  for (int i = 1; i <= k; i ++) {
    f[i] = (f[i] + power(i, n) - power(i - 1, n) + f[i - 1] + mod * mod) % mod;
    ans = (ans + (f[i] ^ i)) % mod;
    for (int j = i + i; j <= k; j += i)
      f[j] = (f[j] + mod - f[i] + f[i - 1]) % mod;
  }
  cout << ans;
}
```

Kết luận: Độ phức tạp của thuật toán: $O(n * \log(n))$

Bài số 8

Đề bài: Dịch bệnh ngày càng diễn biến phức tạp, và vaccine đang dần là một vấn đề mấu chốt để có thể khống chế nó. Lần này, chính phủ đang tổ chức tiêm vaccine cho các vùng, và cần bạn phân tích độ hiệu quả của chiến dịch.

Có n vùng được tiêm chủng, vùng thứ i sẽ chọn a_i $(1 \leq a_i \leq k)$ người để tiêm. Khi ấy, độ hiệu quả của chiến dịch sẽ là $GCD(a_1, a_2, \dots, a_n)$.

Với toàn bộ khả năng tiêm chủng của chiến dịch, tổng của độ hiệu quả toàn bộ các khả năng sẽ là một thông số quan trọng để cân nhắc trước khi bắt đầu chiến dịch.

Là một cố vấn đắc lực, bạn hãy tính thông số đó dưới $mod\ 10^9 + 7$.

Tóm lược đề bài:
Hãy tính tổng của GCD toàn bộ phần tử của chuỗi a có n phần tử và các phần tử là số nguyên thuộc khoảng $[1, k]$.

Giới hạn:

- $1 \leq k \leq 10^5$
- $2 \leq n \leq 10^5$

Ví dụ:

Dữ liệu vào	Dữ liệu ra
3 2	9

Subtask:

- Subtask 1: $n, k \leq 5$
- Subtask 2: $n, k \leq 80$
- Subtask 3: Không có giới hạn nào khác

Lời giải:
Với subtask 1, chúng ta chỉ việc for toàn bộ khả năng, và như thế có thể giải quyết bài toán với giới hạn rất bé.
Với subtask 2, tương tự với các bài trước, quy hoạch động có thể giúp chung ta giải với giới hạn vừa phải.
Tuy nhiên để giải quyết trọn vẹn bài toán, ta sẽ cố gắng áp dụng các ý tưởng cũ vào bài bằng cách định nghĩa hàm $f(x)$ và $g(x)$.

Đặt $T = 10^5$.

Ta có:

$$g(x) = số\ chuỗi\ có\ GCD\ toàn\ chuỗi\ bằng\ x$$

$$f(x) = số\ chuỗi\ có\ GCD\ toàn\ chuỗi\ là\ bội\ của\ x$$

Vậy không khó để thấy:

$$g(x) = f(x) - \sum_{x|y;\ x<y\le T} g(y)$$

Vấn đề sẽ nằm ở việc ta tính $f(x)$ ra sao.

Để chuỗi a có GCD toàn chuỗi là bội của x, bản thân mọi phần tử trong a cũng phải là bội của x. Do đó, ta có thể suy ra được:

$$f(x) = \left\lfloor \frac{T}{x} \right\rfloor^n$$

Tới đây, ta có thể dễ dàng tính được $f(x)$ nhờ phép tính mũ mod nhanh và từ đó tính được $g(x)$. Và cuối cùng, đáp án của bài toán sẽ là:

$$\sum_{i=1}^{t} i * g(i)$$

Code mẫu:

```
#include "bits/stdc++.h"
using namespace std;
#define int long long
const int mod = 1000000007;
inline int power(int x, int y) {
  int r = 1;
  while (y) {
    if (y & 1) r = (r * x) % mod;
    x = (x * x) % mod;
    y >>= 1;
  }
```

```
12.  return r;
13.}
14.const int T = 100001;
15.signed main() {
16.  int n, k;
17.  cin >> n >> k;
18.  vector<int> f(T);
19.  int ans = 0;
20.  for (int i = T - 1; i >= 1; i --) {
21.    f[i] = power(k / i, n);
22.    for (int j = i + i; j < T; j += i)
23.      f[i] = (f[i] + mod - f[j]) % mod;
24.    ans = (ans + f[i] * i) % mod;
25.  }
26.  cout << ans;
27.}
```

Kết luận: Độ phức tạp của thuật toán là $O(T * log2(T))$

Bài số 9

Đề bài: Dịch bệnh lần này bùng phát quá mạnh, và có vẻ đã ngoài tầm kiểm soát. Tới lúc này, chính phủ quyết định chuyển sang test ngẫu nhiên một số cá nhân xem có mắc bệnh không. Để đảm bảo tính hiệu quả, bạn được giao việc cố vấn xem có những tổ hợp cá nhân nào nên được xét nghiệm.

Ở khu vực cần xét nghiệm lần này có n người. Người thứ i khai báo rằng trong khoảng thời gian gần đây họ đã đi ra ngoài a_i lần. Trong lần xét nghiệm lần này, sẽ có một số người (chắc chắn không thể không có ai) phải đi xét nghiệm. Tổ hợp lựa chọn người đi xét nghiệm sẽ được gọi là cần thiết nếu như GCD của a của toàn bộ những người được đi xét nghiệm là 1.

Với tư cách cố vấn, bạn hãy vận dụng khả năng của mình để tính xem có bao cách chọn người đi xét nghiệm mà cần thiết. In ra đáp án $ở\ mod\ 10^9 + 7$.

Tóm lược đề bài:

Cho dãy $a_1, a_2, \dots, a_n$, tính xem có bao dãy con không liên tiếp không rỗng sao cho GCD của các phần tử trong dãy con bằng 1.

Giới hạn:

- $1 \leq n, a_i \leq 10^5$

Ví dụ:

Dữ liệu vào	Dữ liệu ra
3 1 2 3	5

Subtask:

- Subtask 1: $n, a_i \leq 5$
- Subtask 2: $n, a_i \leq 80$
- Subtask 3: Không có ràng buộc nào khác

Lời giải:

Không khó để thấy các subtask 1 và 2 có thể giải quyết bằng cách vét cạn trường hợp hoặc quy hoạch động như đã đề cập trong các bài trước. Ở bài này, một lần nữa ta tập trung vào cách định nghĩa hàm $f(x)$ và $g(x)$ sao cho giải quyết bài toán thuận tiện nhất.

Đặt $T = 10^5$.

Bằng cách vận dụng ý tưởng đã có, ta có thể đặt $f(x)$ và $g(x)$ như sau:

$$g(x) = Số\ dãy\ con\ có\ GCD\ đúng\ bằng\ x$$

$$f(x) = Số\ dãy\ con\ có\ GCD\ là\ bội\ của\ x$$

Và ta có thể suy ra được mối tương quan như sau:

$$g(x) = f(x) - \sum_{x|y;\ x<y\leq T} g(y)$$

Vấn đề giờ nằm ở việc ta tính $f(x)$ ra sao.

Bằng kỹ thuật đã đề cập trước đó, ta có thể tính d_i là số lượng số là bội của i trong thời gian $O(\mathrm{N} * \log(\mathrm{T}))$. Và bằng nhận xét rằng số lượng dãy con không rỗng của của một dãy có độ dài L là $2^L - 1$, ta suy ra được công thức sau:

$$f(x) = 2^{d_x} - 1$$

Tới đây, ta có thể tính $f(x)$ và tính ngược lại $g(x)$. Cuối cùng, đáp án là $g(1)$.

Code mẫu:

```
#include "bits/stdc++.h"
using namespace std;
#define int long long
const int mod = 1000000007;
inline int power(int x, int y) {
  int r = 1;
  while (y) {
    if (y & 1) r = (r * x) % mod;
    x = (x * x) % mod;
    y >>= 1;
  }
  return r;
```

```
}
const int T = 100001;
signed main() {
  int n;
  cin >> n;
  vector<int> f(T);
  vector<int> d(T);
  for (int i = 0; i < n; i ++) {
    int u;
    cin >> u;
    d[u] ++;
  }
  for (int i = 1; i < T; i ++)
    for (int j = i + i; j < T; j += i)
      d[i] += d[j];
  for (int i = T - 1; i >= 1; i --) {
    f[i] = power(2, d[i]) - 1;
    for (int j = i + i; j < T; j += i)
      f[i] = (f[i] + mod - f[j]) % mod;
  }
  f[1] = (f[1] + mod) % mod;
  cout << f[1];
}
```

Kết luận: Độ phức tạp của thuật toán là $O(Tlog2(T))$

Bài số 10

Đề bài: Việc dịch bệnh bùng phát dẫn đến giãn cách xã hội cũng khiến tinh thần của người dân nhiều nơi không được tích cực. Và lần này, chính phủ cần bạn hỗ trợ trong việc tính toán sự tách biệt của một khu dân cư.

Khu dân cư bạn sắp tính toán có n hộ. Hộ thứ i có chỉ số nguy hiểm là a_i. Biết có $n-1$ con đường giữa 2 hộ dân cư, và từ mỗi hộ dân có thể đến mọi hộ dân khác qua các con đường. Đồng thời giữa 2 hộ dân chỉ có duy nhất 1 cách đi.

Biết một cặp hộ dân thứ x và y với $x < y$ được gọi là tách rời cấp độ i nếu GCD của chỉ số của các hộ trên đường đi giữa 2 hộ đó bằng chính xác i.

Với mỗi i sao cho tồn tại ít nhất một cặp hộ dân có độ tách rời cấp độ i, hãy in ra i và số cặp dân đang ở độ tách rời cấp i trên mỗi dòng với i tăng dần.

Là một lập trình viên uy tín và được ủy thác cho nhiệm vụ này, bạn hãy cố gắng giúp chính phủ tính toán các thông số trên.

Tóm lược đề bài:

Cho một cây có n đỉnh, đỉnh thứ i có hệ số a_i. Với mỗi i mà có tối thiểu một cặp đỉnh thỏa mãn điều kiện sau, hãy in ra i và số cặp trên một dòng:

- GCD của hệ số của tất cả các đỉnh trên đường đi đơn giản giữa 2 đỉnh đó bằng chính xác i.

Giới hạn:

- $1 \leq n, a_i \leq 10^5$
- Thời gian: 3s

Ví dụ:

Dữ liệu vào	Dữ liệu ra
3 1 2 3	5

Subtask:

- Subtask 1: $n \leq 100$
- Subtask 2: $n, a_i \leq 5000$
- Subtask 3: Không còn giới hạn nào khác

Lời giải:

Để giải quyết subtask số 1, chúng ta có thể for mọi cặp đỉnh và lấy GCD đường đi trên cây của chúng một cách bình thường không cần tối ưu gì cả.
Tuy nhiên để giải quyết subtask 2 của bài toán, chúng ta cần dựng trước mảng GCD để tính nhanh trong $O(1)$. Sau đó, chúng ta cần dựng mảng thưa để tính tổ tiên chung nhỏ nhất trên cây cùng với GCD đoạn nối tổ tiên ngay cùng mảng thưa. Từ đó, ta có thể tính được GCD trên đường đi giữa 2 đỉnh trong thời gian $O(\log(n))$ và từ đó giải quyết bài toán trong thời gian $O(n^2 \log(n))$.

Tuy nhiên, chỉ ngần đó là chưa đủ để giải quyết bài toán. Chúng ta sẽ phải tìm cách định nghĩa hàm $f(x)$ và $g(x)$ sao cho ta áp dụng được các ý tưởng đã có vào bài.

Một cách tự nhiên, ta có thể suy ra được:

$$g(x) = số\ cặp\ đường\ đi\ có\ GCD\ bằng\ chính\ xác\ x$$

$$f(x) = số\ cặp\ đường\ đi\ có\ GCD\ là\ bội\ của\ x$$

Vậy ta có thể dễ dàng suy ra:

$$g(x) = f(x) - \sum_{x|y;\ x<y\leq T} g(y)$$

Tuy nhiên, một lần nữa vấn đề lại nằm ở việc ta giải quyết $f(x)$ ra sao khi điều kiên đề cho quá nan giải.

Ở đây, chúng ta có 2 phương án để giải.

Phương án giải 1:

Ta xét cạnh giữa 2 đỉnh x và y, đặt cho nó trọng số là $GCD(a_x, a_y)$. Vậy khi đó, nếu ta xét một đồ thị với toàn bộ các cạnh là bội của i, thì nó sẽ tạo thành một rừng cây con. Và số cặp đường đi của đồ thị sẽ chính bằng tổng số cặp đường đi ở các cây con, và bằng số cặp có thể chọn ở các cây con. Đó chính là giá trị $f(i)$ ta cần tìm.

Vậy với từng i, ta sẽ dựng một đồ thị với các cạnh có trọng số là bội của i đã được lưu từ trước. Để ý nếu ta DFS thử tất cả các điểm như ta thường làm, độ phức tạp của mỗi lần sẽ lên tới $O(n)$ và hoàn toàn vượt quá giới hạn. Do đó, ta sẽ chỉ DFS từ từng đầu mút của cạnh một trong các cạnh đã cho.

Ta để ý số có nhiều ước số nhất trong khoảng 10^5 có 128 ước số, vậy mỗi cạnh sẽ xuất hiện trong tối đa 128 thao tác dựng đồ thị, và đảm bảo thời gian chạy cho chúng ta ở mức an toàn.

Code mẫu phương án giải 1:

```
#include "bits/stdc++.h"
using namespace std;
const int T = 100001;
int cnt = 0, x[T], y[T], a[T], sz[T];
int* g[T];
vector<int> z[T];
long long f[T];
bool vis[T];
void dfs(int u) {
 vis[u] = true;
 cnt ++;
 for (int i = 0; i < sz[u]; i ++) {
  if (!vis[g[u][i]]) {
   dfs(g[u][i]);
  }
 }
}
int main() {
 ios::sync_with_stdio(0); cin.tie(0);
 int n;
 cin >> n;
 for (int i = 0; i < n; i ++) cin >> a[i];
 for (int i = 0; i < n - 1; i ++) {
  cin >> x[i] >> y[i];
  x[i] --;
  y[i] --;
  sz[x[i]] ++;
  sz[y[i]] ++;
```

```
    int w = __gcd(a[x[i]], a[y[i]]);
    z[w].push_back(i);
  }
  for (int i = 0; i < n; i ++) {
    g[i] = new int[sz[i]];
    sz[i] = 0;
  }
  for (int i = T - 1; i >= 1; i --) {
    for (int j = i; j < T; j += i) {
      for (int id : z[j]) {
        g[x[id]][sz[x[id]] ++] = y[id];
        g[y[id]][sz[y[id]] ++] = x[id];
      }
    }
    for (int j = i; j < T; j += i) {
      for (int id : z[j]) {
        cnt = 0;
        if (!vis[x[id]]) dfs(x[id]);
        f[i] += 1LL * cnt * (cnt - 1) / 2;
      }
    }
    for (int j = i; j < T; j += i) {
      for (int id : z[j]) {
        sz[x[id]] = 0;
        sz[y[id]] = 0;
        vis[x[id]] = vis[y[id]] = false;
      }
      if (j > i) f[i] -= f[j];
    }
  }
  for (int i = 1; i < T; i ++) {
    if (f[i]) {
      cout << i << ' ' << f[i] << endl;
    }
  }
}
```

Phương án giải 2:

Ở đây ta sẽ sử dụng kỹ thuật chia nhỏ cây centroid.

Một số tài liệu có thể tìm thấy ở đây: https://laptrinhthidau.wordpress.com/2016/08/23/centroid-decomposition-phan-ra-trong-tam-tren-cay/

Centroid là một kỹ thuật phân rã một cây thành nhiều cây con, mỗi cây con sẽ có một đỉnh là centroid. Đặc tính của centroid là nếu tháo nó ra, cây sẽ chia thành các cây con với độ lớn không quá một nửa cây gốc. Do đó, mỗi đỉnh sẽ là một centroid với một cây con của riêng nó, và tổng độ lớn của tất cả các cây centroid cộng lại không vượt quá $Nlog2(N)$.
Xét cây con có centroid là u. Ta sẽ tính số cặp mà chắc chắn đi qua u, đồng thời có GCD trên đường đi là bội của các ước của a_u. Nếu ta xét lại đặc tính của centroid, ta có thể thấy nó phân đồ thị thành nhiều cây con. Và mỗi cặp muốn đi qua u thì chúng không được thuộc cùng cây con.
Từ đỉnh u, ta tạo được các giá trị $val[v]$ chính là GCD từ đỉnh v đến u. Nhận thấy GCD từ u đến v cũng chính là $GCD(val[u], val[v])$, nếu ta lấy tích các số có val là bội x trong cây con 1 nhân với cây con 2, ta sẽ ra được số cặp từ 2 cây con mà đi qua u, đồng thời có GCD là bội của x. Vậy, nếu t đặt sẵn thứ tự của các cây con, lần lượt đi qua từng cây và duy trì một mảng $cnt[x]$ đếm số các đỉnh có val là bội của x, ta có thể cập nhật lần lượt từng cây con một vào f. Đồng thời ban đầu ta cũng cập nhật vào cnt bằng các ước của $a[u]$. Sau khi cập nhật xong, ta chỉ việc xóa hết các cnt đi và đi xuống các centroid sâu hơn để giải.

Để ý mỗi tầng centroid chạy mất O(độ $lớn\ centroid * \log(a)$), vậy do tổng độ lớn tất cả các centroid bằng $Nlog2(N)$, dẫn đến độ phức tạp cuối cùng của thuật toán là $O(N \log_2(N)^2)$.

Code mẫu phương án giải 2:

```
#include "bits/stdc++.h"
using namespace std;
const int T = 100001;
vector<int> dv[T], g[T];
int a[T], val[T], cnt[T], child[T];
long long f[T];
bool vis[T];
void dfs(int u, int p) {
  child[u] = 1;
 for (int v : g[u]) if (v != p && !vis[v]) {
  val[v] = __gcd(val[u], a[v]);
  dfs(v, u);
  child[u] += child[v];
 }
}
int find(int u, int p, int r) {
 for (int v : g[u]) if (v != p && !vis[v]) {
  if (child[v] * 2 >= child[r])
   return find(v, u, r);
 }
 return u;
}
void get(int u, int p) {
```

```
  for (int v : dv[val[u]]) f[v] += cnt[v];
  for (int v : g[u]) if (v != p && !vis[v])
    get(v, u);
}
void update(int u, int p, int x) {
  for (int v : dv[val[u]]) cnt[v] += x;
  for (int v : g[u]) if (v != p && !vis[v])
    update(v, u, x);
}
void centroid(int u) {
  dfs(u, u);
  u = find(u, u, u);
  vis[u] = true;
  val[u] = a[u];
  dfs(u, u);
  for (int v : dv[a[u]]) cnt[v] ++;
  for (int v : g[u]) if (!vis[v]) {
    get(v, u);
    update(v, u, 1);
  }
  for (int v : dv[a[u]]) cnt[v] --;
  for (int v : g[u]) if (!vis[v])
    update(v, u, -1);
  for (int v : g[u]) if (!vis[v])
    centroid(v);
}
int main() {
  ios::sync_with_stdio(0); cin.tie(0);
  int n;
  cin >> n;
  for (int i = 1; i < T; i ++)
    for (int j = i; j < T; j += i)
      dv[j].push_back(i);
  for (int i = 0; i < n; i ++) cin >> a[i];
  for (int i = 0; i < n - 1; i ++) {
    int u, v;
    cin >> u >> v;
    u --;
    v --;
    g[u].push_back(v);
    g[v].push_back(u);
  }
  centroid(0);
  for (int i = T - 1; i >= 1; i --)
```

```
68.    for (int j = i + i; j < T; j += i)
69.      f[i] -= f[j];
70.  for (int i = 1; i < T; i ++) {
71.    if (f[i]) cout << i << ' ' << f[i] << endl;
72.  }
73.}
```

D. Tài liệu tham khảo

[1] Nguyên lý bao hàm-loại trừ – Wikipedia tiếng Việt
[2] Số học 7 - Bao hàm - Loại trừ (Inclusion-Exclusion) | VNOI Wiki
[3] Công thức Bao hàm - Loại trừ - Viblo

Các bài tập khác cũng giải được bằng kỹ thuật này:

https://atcoder.jp/contests/abc191/tasks/abc191_f
https://codeforces.com/contest/1436/problem/F
https://www.codechef.com/problems/CHEFSUMS
https://codeforces.com/problemset/problem/839/D
https://www.codechef.com/problems/COPRIME3
https://www.spoj.com/problems/GCDEX
https://codeforces.com/contest/1554/problem/E
https://www.codechef.com/NOV15/problems/SMPLSUM
https://www.codechef.com/MAY21C/problems/ISS