Báo cáo Nghiên cứu Chuyên sâu: Phân tích Thuật toán Số học và Ứng dụng Sàng Eratosthenes trong Lập trình Thi đấu (Codeforces Rating 800-1000)

1. Tổng quan về Số học trong Lập trình Thi đấu

Trong lĩnh vực lập trình thi đấu (Competitive Programming - CP), Số học (Number Theory) đóng vai trò là một trong những trụ cột kiến thức nền tảng quan trọng nhất, đặc biệt đối với những người mới bắt đầu (Newbie) đang nỗ lực vượt qua ngưỡng xếp hạng (rating) 800-1000 trên các nền tảng như Codeforces. Khả năng xử lý các bài toán liên quan đến số nguyên tố, ước chung lớn nhất (GCD), và các tính chất chia hết không chỉ đòi hỏi tư duy toán học nhạy bén mà còn yêu cầu sự hiểu biết sâu sắc về độ phức tạp thuật toán để tránh các lỗi vượt quá thời gian (Time Limit Exceeded - TLE).[1]

Báo cáo này được xây dựng nhằm cung cấp một cái nhìn toàn diện và tường tận về hai kỹ thuật cốt lõi: Kiểm tra tính nguyên tố (Primality Test) với độ phức tạp $O(\sqrt{N})$ và thuật toán Sàng Eratosthenes (Sieve of Eratosthenes) để liệt kê số nguyên tố trong phạm vi lớn (10^6 đến 10^7). Đồng thời, báo cáo sẽ phân tích chi tiết 20 bài tập điển hình trên Codeforces nằm trong dải rating 800-1000, giúp người học chuyển hóa lý thuyết thành kỹ năng thực chiến.[2]

1.1 Tầm quan trọng của Số nguyên tố

Số nguyên tố là các số tự nhiên lớn hơn 1 chỉ có hai ước số dương phân biệt là 1 và chính nó. Chuỗi các số nguyên tố bắt đầu bằng: $2, 3, 5, 7, 11, 13, 17, 19, \dots$. Trong lập trình thi đấu, số nguyên tố xuất hiện dày đặc trong nhiều dạng bài toán khác nhau:

- **Mã hóa và Hashing:** Các số nguyên tố lớn (như $10^9 + 7$ hoặc $10^9 + 9$) thường được sử dụng làm modulo để giảm thiểu xung đột trong các hàm băm (hash functions) xử lý chuỗi hoặc cấu trúc dữ liệu.
- **Phân tích thừa số nguyên tố:** Định lý Cơ bản của Số học khẳng định mọi số nguyên lớn hơn 1 đều có thể biểu diễn duy nhất dưới dạng tích các thừa số nguyên tố. Điều này là cơ sở cho các bài toán về ước số, bội số chung nhỏ nhất (LCM), và các hàm số học.[4]
- **Lý thuyết trò chơi:** Nhiều trò chơi tổ hợp dựa trên việc bốc sỏi hoặc chia số (như Nim game) sử dụng tính chất của số nguyên tố để xác định trạng thái thắng thua.

1.2 Ràng buộc Tính toán và Độ phức tạp

Một yếu tố then chốt phân biệt lập trình thi đấu với toán học thuần túy là giới hạn về thời gian thực thi và bộ nhớ. Trên Codeforces, giới hạn thời gian thông thường cho một bài toán là từ 1.0 đến 2.0 giây. Các máy chấm hiện đại có thể thực hiện khoảng 10^8 phép tính cơ bản mỗi giây. Sự hiểu biết về giới hạn này định hình cách tiếp cận thuật toán [5]:

Giới hạn N	Độ phức tạp cho phép	Thuật toán khả thi
$N \leq$ đến 20	$O(2^N$ hoặc $O(N!$	Đệ quy, Quay lui, Duyệt bitmask
$N \leq$	$O(N^3$ hoặc $O(N^4$	Floyd-Warshall, Duyệt trâu
$N \leq$	$O(N^2)$	Duyệt 2 vòng lặp lồng nhau
$N \leq$	$O(N \log N$	Sắp xếp, Segment Tree, Sàng Eratosthenes (biến thể)
$N \leq$	$O(N)$ hoặc $O(N \log \log N)$	Sàng Eratosthenes tuyến tính
$N \leq$	$O(\sqrt{N})$	Kiểm tra nguyên tố cơ bản
$N \leq 10^{18}$	$O(\log N$ hoặc $O(1)$	Toán học công thức, Miller-Rabin

Đối với phân khúc newbie (800-1000), việc nắm vững thuật toán $O(\sqrt{N})$ và Sàng Eratosthenes là đủ để giải quyết hầu hết các vấn đề mà không cần đến các thuật toán xác suất phức tạp như Miller-Rabin hay Pollard's rho.[7]

---

2. Phân tích Chuyên sâu: Kiểm tra Tính nguyên tố

$O(\sqrt{N})$

Kiểm tra xem một số nguyên dương N có phải là số nguyên tố hay không là bài toán cơ bản nhất. Phương pháp tiếp cận ngây thơ (naive approach) là duyệt qua tất cả các số từ 2 đến $N-1$ và kiểm tra xem có số nào là ước của N hay không. Tuy nhiên, phương pháp này có độ phức tạp $O(N)$, hoàn toàn không khả thi khi N lên tới 10^9 hoặc 10^{12}.[5]

2.1 Cơ sở Toán học của Thuật toán Tối ưu

Để tối ưu hóa, ta dựa vào một định lý quan trọng trong số học sơ cấp liên quan đến sự phân bố của các ước số.

Định lý: Nếu một số nguyên dương N là hợp số (composite number), thì nó phải có ít nhất một ước số (khác 1) nhỏ hơn hoặc bằng căn bậc hai của chính nó ($\sqrt{N}$).

Chứng minh chi tiết:

Giả sử N là một hợp số. Theo định nghĩa, N có thể được viết dưới dạng tích của hai số nguyên a và b sao cho $1 < a \leq b < N$, nghĩa là $N = a \times b$.

Chúng ta sẽ chứng minh bằng phương pháp phản chứng. Giả sử rằng cả hai nhân tử a và b đều lớn hơn $\sqrt{N}$.

Khi đó ta có bất đẳng thức:

$$a > \sqrt{N}$$

$$b > \sqrt{N}$$Nhân hai bất đẳng thức này lại với nhau (vì $a, b > 0$), ta được:

$$a \times b > \sqrt{N} \times \sqrt{N}$$

$$N > N$$

Điều này dẫn đến một mâu thuẫn vô lý (N không thể lớn hơn chính nó). Do đó, giả thiết ban đầu là sai. Bắt buộc phải có ít nhất một trong hai thừa số a hoặc b nhỏ hơn hoặc bằng $\sqrt{N}$.

Vì ta đã giả định $a \leq b$, nên chắc chắn $a \leq \sqrt{N}$.[4]

Hệ quả thực tiễn: Để kiểm tra tính nguyên tố của N, chúng ta không cần duyệt đến $N-1$. Chỉ cần duyệt từ 2 đến $\lfloor \sqrt{N} \rfloor$. Nếu không tìm thấy ước số nào trong khoảng này, ta có thể khẳng định chắc chắn N là số nguyên tố. Điều này giảm không gian tìm kiếm từ 10^9 xuống còn $\approx 31,622$ phép tính, một sự cải thiện hiệu suất khổng lồ.

2.2 Cài đặt Thuật toán và Các Lưu ý Kỹ thuật

Dưới đây là cài đặt chuẩn mực trong C++ cho thuật toán kiểm tra nguyên tố $O(\sqrt{N})$. Cần chú ý đặc biệt đến các trường hợp biên và kiểu dữ liệu để tránh lỗi.[5]

C++

```
bool isPrime(long long n) {
    // Trường hợp biên: số nhỏ hơn 2 không phải là số nguyên tố
    if (n <= 1) return false;

    // Vòng lặp kiểm tra từ 2 đến căn bậc hai của n
    // Sử dụng i * i <= n thay vì i <= sqrt(n) để tránh sai số số thực
    for (long long i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            return false; // Tìm thấy ước số, n là hợp số
        }
    }
    return true; // Không tìm thấy ước số nào, n là số nguyên tố
}
```

Phân tích kỹ thuật cài đặt:

1. **Tránh hàm sqrt():** Việc sử dụng hàm sqrt(n) trong điều kiện vòng lặp (i <= sqrt(n)) có thể gây ra sai số làm tròn đối với các số nguyên rất lớn hoặc tốn kém chi phí tính toán số thực. Thay vào đó, điều kiện i * i <= n hoạt động hoàn toàn trên số nguyên và đảm bảo độ chính xác tuyệt đối. Tuy nhiên, cần lưu ý rủi ro tràn số (overflow) nếu i * i vượt quá giới hạn của kiểu dữ liệu long long, mặc dù với $N \leq 10^{18}$, i chỉ lên tới 10^9, và $i^2 \approx 10^{18}$ vẫn nằm trong giới hạn an toàn.[4]

2. **Tối ưu hóa bước nhảy:** Chúng ta biết rằng số chẵn duy nhất là số nguyên tố là số 2. Tất cả các số chẵn lớn hơn 2 đều là hợp số. Do đó, ta có thể kiểm tra riêng tính chia hết cho 2, sau đó chỉ duyệt các số lẻ trong vòng lặp (bước nhảy i += 2). Điều này giảm một nửa số lần lặp, tuy độ phức tạp tiệm cận vẫn là $O(\sqrt{N})$ nhưng hằng số thời gian giảm đáng kể.[9]

C++

```
bool isPrimeOptimized(long long n) {
    if (n <= 1) return false;
    if (n == 2) return true;
    if (n % 2 == 0) return false; // Loại bỏ số chẵn > 2

    // Chỉ kiểm tra các số lẻ: 3, 5, 7,...
    for (long long i = 3; i * i <= n; i += 2) {
        if (n % i == 0) return false;
    }
    return true;
}
```

---

3. Lý thuyết Chuyên sâu: Sàng Eratosthenes (Sieve of Eratosthenes)

Trong khi thuật toán $O(\sqrt{N})$ rất hiệu quả để kiểm tra một số đơn lẻ, nó trở nên chậm chạp khi bài toán yêu cầu kiểm tra tính nguyên tố cho hàng triệu số hoặc liệt kê tất cả các số nguyên tố trong một khoảng lớn. Ví dụ, để liệt kê các số nguyên tố đến $N = 10^7$, việc chạy hàm isPrime cho từng số sẽ tốn $O(N\sqrt{N})$ thời gian tổng cộng, dẫn đến TLE. Giải pháp tối ưu cho bài toán này là Sàng Eratosthenes.[8]

3.1 Nguyên lý Hoạt động

Sàng Eratosthenes hoạt động dựa trên nguyên tắc loại trừ. Thay vì kiểm tra từng số xem nó có ước hay không, ta giả định tất cả các số đều là số nguyên tố, sau đó lần lượt loại bỏ các bội số của từng số nguyên tố đã biết.

Quy trình thực hiện:

1. Khởi tạo một mảng boolean isPrime có kích thước $N+1$, gán tất cả giá trị là true.
2. Đánh dấu isPrime và isPrime là false (theo định nghĩa).
3. Bắt đầu từ số nguyên tố đầu tiên $p = 2$. Duyệt qua tất cả các bội của p bắt đầu từ p^2 (tức là $p^2, p^2 + p, p^2 + 2p, \cdots$) và đánh dấu chúng là false (hợp số).
4. Tìm số nguyên tố tiếp theo (số tiếp theo trong mảng vẫn có giá trị true) và lặp lại bước 3.
5. Quá trình kết thúc khi $p^2 > N$. Các giá trị còn lại là true trong mảng chính là các số nguyên tố.[13]

3.2 Tại sao bắt đầu sàng từ p^2?

Một tối ưu quan trọng trong Sàng Eratosthenes là khi xét đến số nguyên tố p, ta không cần đánh dấu các bội số nhỏ hơn p^2 (như $2p, 3p, \ldots, (p-1)p$). Lý do là các bội số này đã được đánh dấu bởi các thừa số nguyên tố nhỏ hơn p. Ví dụ, khi xét $p = 5$, bội số $10(2 \times 5)$ đã bị đánh dấu khi sàng số 2; bội số $15(3 \times 5)$ đã bị đánh dấu khi sàng số 3. Do đó, ta bắt đầu trực tiếp từ $25(5^2)$. Điều này giúp loại bỏ các thao tác trùng lặp.[8]

3.3 Phân tích Độ phức tạp Thời gian và Không gian

Độ phức tạp của Sàng Eratosthenes được tính bằng tổng số phép toán đánh dấu bội số.

Số phép toán tỉ lệ thuận với tổng:

$$\frac{N}{2} + \frac{N}{3} + \frac{N}{5} + \frac{N}{7} + \cdots + \frac{N}{p}$$

với $p \leq \sqrt{N}$.

Theo Định lý Mertens thứ hai, tổng nghịch đảo của các số nguyên tố nhỏ hơn hoặc bằng N tiệm cận với $\ln(\ln N)$. Do đó, độ phức tạp thời gian tổng thể là:

$$O(N \log(\log N))$$

Với $N = 10^7$, $\log(\log N) \approx \log(16) \approx 4$, nghĩa là thuật toán chạy gần như tuyến tính,

cực kỳ nhanh và phù hợp với giới hạn thời gian 1 giây.[8]

Về không gian, mảng isPrime yêu cầu $O(N)$ bộ nhớ.

- Với $N = 10^7$: Mảng bool tốn khoảng 10 MB (chấp nhận được).
- Với $N = 10^8$: Mảng bool tốn khoảng 100 MB. Giới hạn bộ nhớ thường là 256MB hoặc 512MB, nên vẫn khả thi.
- **Tối ưu hóa bộ nhớ:** Có thể sử dụng std::vector<bool> hoặc std::bitset để giảm lượng bộ nhớ xuống 8 lần (1 bit cho mỗi số thay vì 1 byte). Với $N = 10^8$, bitset chỉ tốn khoảng 12.5 MB.[8]

3.4 Cài đặt Mẫu (C++)

Đoạn mã sau minh họa cách cài đặt Sàng Eratosthenes để tìm các số nguyên tố đến 10^7 :

C++

```
#include <vector>
#include <iostream>

const int MAXN = 10000005; // Kích thước tối đa 10^7
std::vector<bool> is_prime(MAXN, true); // Sử dụng vector<bool> để tối ưu bộ nhớ

void sieve() {
    is_prime = is_prime = false; // 0 và 1 không phải là số nguyên tố
    for (int i = 2; i * i < MAXN; i++) {
        if (is_prime[i]) {
            // Bắt đầu đánh dấu từ i*i, bước nhảy là i
            // Ép kiểu long long cho i*i để tránh tràn số khi i gần 10^5
            for (int j = i * i; j < MAXN; j += i) {
                is_prime[j] = false;
            }
        }
    }
}
```

---

4. Phân tích Chi tiết 20 Bài tập Codeforces (Rating 800-1000)

Phần này sẽ đi sâu vào phân tích 20 bài tập được chọn lọc kỹ lưỡng từ Codeforces. Các bài tập này được phân loại dựa trên kỹ thuật số học cần sử dụng, từ cơ bản đến ứng dụng Sàng. Mỗi bài tập sẽ được mổ xẻ về đề bài, tư duy toán học, và cách cài đặt để giúp người học nắm bắt bản chất vấn đề.

Dữ liệu về bài tập được tổng hợp từ các nguồn nghiên cứu và phân loại theo mức độ phổ biến cũng như tính giáo dục cao.[2]

Nhóm 1: Tư duy Số học Cơ bản & Toán Tổ hợp (Rating 800)

Nhóm bài này tập trung vào các tính chất cơ bản của số nguyên tố (chẵn lẻ, ước số) và không yêu cầu thuật toán phức tạp.

1. Bachgold Problem

- **Mã bài:** 749A
- **Rating:** 800
- **Nguồn:** [2]

- **Tóm tắt đề bài:** Cho số nguyên dương n. Hãy biểu diễn n thành tổng của nhiều số nguyên tố nhất có thể.
- **Phân tích Toán học:** Để tối đa hóa số lượng số hạng trong tổng, ta cần chọn các số nguyên tố có giá trị nhỏ nhất. Số nguyên tố nhỏ nhất là 2.
    - Nếu n chẵn: Ta có thể biểu diễn n hoàn toàn bằng các số 2 ($n = 2 + 2 + \cdots + 2$). Số lượng số hạng là $n/2$.
    - Nếu n lẻ: Vì tổng của các số chẵn luôn là số chẵn, ta bắt buộc phải sử dụng ít nhất một số nguyên tố lẻ để tổng là số lẻ. Số nguyên tố lẻ nhỏ nhất là 3. Ta tách $n = 3 + (n - 3)$. Phần còn lại $(n - 3)$ là số chẵn, có thể biểu diễn toàn bộ bằng số 2.
- **Giải thuật:** Kiểm tra tính chẵn lẻ của n. Nếu chẵn in $n/2$ số 2. Nếu lẻ in một số 3 và $(n - 3)/2$ số 2.
- **Độ phức tạp:** $O(n)$ để in kết quả (do output lớn), nhưng logic tính toán là $O(1)$.

2. Design Tutorial: Learn from Math

- **Mã bài:** 472A
- **Rating:** 800

- **Nguồn:** [19]
- **Tóm tắt đề bài:** Cho n ($n \geq 12$), hãy tìm hai hợp số x và y sao cho $x + y = n$.
- **Phân tích Toán học:** Hợp số là số có ước khác 1 và chính nó. Bài toán yêu cầu tách n thành tổng hai hợp số.
    - Nếu n là số chẵn: Ta có thể chọn $x = 4$ (hợp số nhỏ nhất). Khi đó $y = n - 4$. Vì n chẵn và $n \geq 12$, y cũng là số chẵn lớn hơn 8, do đó y chia hết cho 2 và là hợp số.
    - Nếu n là số lẻ: Ta có thể chọn $x = 9$ (hợp số lẻ nhỏ nhất). Khi đó $y = n - 9$. Vì n lẻ, y là số chẵn. Vì $n \geq 12$, $y \geq 3$, và y chẵn nên y là hợp số.
- **Giải thuật:** Kiểm tra n chẵn hay lẻ và in ra cặp $(4, n-4)$ hoặc $(9, n-9)$.
- **Độ phức tạp:** $O(1)$.

3. Maximum GCD

- **Mã bài:** 1370A
- **Rating:** 800
- **Nguồn:** [22]
- **Tóm tắt đề bài:** Cho số nguyên n. Tìm giá trị lớn nhất của $\gcd(a, b)$ với $1 \leq a < b \leq n$.
- **Phân tích Toán học:** Gọi $g = \gcd(a, b)$. Điều này có nghĩa là g là ước của cả a và b. Do đó, a và b đều là bội của g. Vì $a < b$, bội nhỏ nhất có thể là $a = g$ và $b = 2g$. Để $b \leq n$, ta phải có $2g \leq n$, suy ra $g \leq n/2$. Giá trị lớn nhất có thể đạt được chính là $\lfloor n/2 \rfloor$.
- **Giải thuật:** In ra $n/2$.
- **Độ phức tạp:** $O(1)$.

4. EhAb AnD gCd

- **Mã bài:** 1325A
- **Rating:** 800
- **Nguồn:** [2]
- **Tóm tắt đề bài:** Cho số nguyên x. Tìm hai số nguyên dương a, b sao cho

$\text{GCD}(a, b) + \text{LCM}(a, b) = x$.

- **Phân tích Toán học:** Bài toán nghe có vẻ phức tạp nhưng có một lời giải tầm thường (trivial solution). Ta biết rằng $\text{GCD}(1, k) = 1$ và $\text{LCM}(1, k) = k$ với mọi số nguyên dương k.
  - Thay $a = 1$, ta cần tìm b sao cho $1 + b = x \Rightarrow b = x - 1$.
  - Cặp $(1, x - 1)$ luôn thỏa mãn: $\text{GCD}(1, x - 1) + \text{LCM}(1, x - 1) = 1 + (x - 1) = x$.
- **Giải thuật:** Luôn in ra 1 và x-1.
- **Độ phức tạp:** $O(1)$.

5. Subtract or Divide

- **Mã bài:** 1451A
- **Rating:** 800
- **Nguồn:** [27]
- **Tóm tắt đề bài:** Cho số n. Mỗi bước có thể giảm n đi 1 hoặc chia n cho một ước số của nó (khác 1 và chính nó). Tìm số bước ít nhất để biến n thành 1.
- **Phân tích Toán học:** Đây là bài toán tham lam (greedy) kết hợp số học.
  - $n = 1$: 0 bước.
  - $n = 2$: 1 bước (chia 2 hoặc trừ 1).
  - $n = 3$: 2 bước (trừ 1 thành 2, rồi chia 2 thành 1).
  - $n > 3$ và chẵn: 1 bước chia cho $n/2$ để được 2, sau đó 1 bước nữa để về 1. Tổng 2 bước.
  - $n > 3$ và lẻ: 1 bước trừ 1 để thành số chẵn, sau đó áp dụng quy tắc số chẵn (2 bước nữa). Tổng 3 bước.
- **Giải thuật:** Kiểm tra các trường hợp nhỏ và tính chẵn lẻ.
- **Độ phức tạp:** $O(1)$.

6. Equation

- **Mã bài:** 1269A
- **Rating:** 800
- **Nguồn:** [2]
- **Tóm tắt đề bài:** Tìm hai hợp số a, b sao cho $a - b = n$.

- **Phân tích Toán học:** Tương tự bài 472A, ta cần tận dụng tính chất của hợp số. Ta biết $9k$ và $8k$ đều là hợp số với mọi $k \geq 1$. Hiệu của chúng là $9k - 8k = k$.
  - Thay $k = n$, ta có cặp số $9n$ và $8n$.
  - Vì $n \geq 1$, $9n$ chia hết cho 9 (và > 9) nên là hợp số. Tương tự $8n$ là hợp số.
- **Giải thuật:** In ra $9n$ và $8n$.
- **Độ phức tạp:** $O(1)$.

7. GCD Sum

- **Mã bài:** 1498A
- **Rating:** 800
- **Nguồn:** [30]
- **Tóm tắt đề bài:** Định nghĩa $\text{gcdSum}(x) = \gcd(x, \text{tổng các chữ số của } x)$. Tìm số nguyên nhỏ nhất $y \geq n$ sao cho $\text{gcdSum}(y) > 1$.
- **Phân tích Toán học:**
  - Ta cần tìm $y \geq n$ sao cho y và tổng chữ số của y có ước chung lớn hơn 1.
  - Dễ thấy mọi số chia hết cho 3 đều có tổng các chữ số chia hết cho 3, do đó $\text{gcdSum}(y) \geq 3 > 1$.
  - Cứ mỗi 3 số liên tiếp chắc chắn có một số chia hết cho 3. Do đó, khoảng cách từ n đến kết quả y rất nhỏ (thường chỉ 0, 1 hoặc 2 đơn vị).
- **Giải thuật:** Duyệt i từ n, $n+1, \cdots$ và kiểm tra điều kiện $\gcd(i, \text{sumDigits}(i)) > 1$. Vòng lặp sẽ kết thúc rất nhanh.
- **Độ phức tạp:** Gần như $O(\log n)$ (do tính tổng chữ số và GCD), số bước lặp là hằng số.

8. Fair Division

- **Mã bài:** 1472B
- **Rating:** 800
- **Nguồn:** [2]
- **Tóm tắt đề bài:** Có n viên kẹo, mỗi viên nặng 1g hoặc 2g. Có chia đều được thành 2 phần bằng nhau không?
- **Phân tích Toán học:** Tổng trọng lượng S phải là số chẵn.
  - Nếu S lẻ: Không thể chia đôi.

    - Nếu S chẵn: Ta cần tạo ra trọng lượng $S/2$.
        - Nếu số lượng viên kẹo 1g (cnt_1) là số chẵn và khác 0: Luôn chia được.
        - Nếu $cnt_1 = 0$: Số lượng viên kẹo 2g (cnt_2) phải chẵn mới chia được (vì ta không thể bẻ đôi viên 2g).
- **Giải thuật:** Đếm số lượng 1 và 2, kiểm tra điều kiện chẵn lẻ.
- **Độ phức tạp:** $O(N)$ để đếm.

9. Sum of 2050

- **Mã bài:** 1517A
- **Rating:** 800
- **Nguồn:** [2]
- **Tóm tắt đề bài:** Biểu diễn số n thành tổng các số dạng 2050×10^k. Tìm số lượng số hạng ít nhất.
- **Phân tích Toán học:**
    - Mọi số hạng đều chia hết cho 2050. Vậy n phải chia hết cho 2050. Nếu không, in -1.
    - Nếu $n = 2050 \times Q$, bài toán trở thành biểu diễn Q thành tổng các lũy thừa của 10 (10^k). Số lượng số hạng chính là tổng các chữ số của Q trong hệ thập phân.
- **Giải thuật:** Kiểm tra $n \% 2050$. Nếu bằng 0, tính tổng các chữ số của thương.
- **Độ phức tạp:** $O(\log n)$.

10. Red and Blue Beans

- **Mã bài:** 1519A
- **Rating:** 800
- **Nguồn:** [33]
- **Tóm tắt đề bài:** Có r hạt đỏ và b hạt xanh. Chia vào các gói sao cho mỗi gói có ít nhất 1 đỏ, 1 xanh và chênh lệch số lượng mỗi màu không quá d.
- **Phân tích Toán học:** Giả sử $r < b$. Để tối ưu, mỗi gói ta chỉ bỏ 1 hạt đỏ. Số gói tối đa là r. Số hạt xanh trong mỗi gói trung bình là b/r. Để thỏa mãn điều kiện, mỗi gói tối đa chứa $1 + d$ hạt xanh.
    - Vậy ta cần kiểm tra xem tổng số hạt xanh có vượt quá sức chứa tối đa của r gói hay không: $b \le r \times (1 + d)$.

- **Giải thuật:** Kiểm tra $\min(r, b) \times (1 + d) \geq \max(r, b)$.
- **Độ phức tạp:** $O(1)$.

Nhóm 2: Kiểm tra Nguyên tố & Tính chất Số học (Rating 900)

Nhóm này bắt đầu yêu cầu cài đặt các hàm kiểm tra nguyên tố hoặc sử dụng vòng lặp thông minh hơn.

11. Almost Prime

- **Mã bài:** 26A
- **Rating:** 900
- **Nguồn:** [2]
- **Tóm tắt đề bài:** Một số được gọi là "gần nguyên tố" nếu nó có đúng 2 ước nguyên tố phân biệt. Đếm số lượng số như vậy trong khoảng $[1, n]$ với $n \leq 3000$.
- **Phân tích Thuật toán:**
  - Với n nhỏ, ta có thể duyệt từng số từ 1 đến n.
  - Với mỗi số, phân tích thừa số nguyên tố và đếm số lượng thừa số khác nhau.
  - Cách tối ưu hơn (dạng Sàng): Dùng một mảng count khởi tạo bằng 0. Duyệt các số nguyên tố p từ 2 đến n. Với mỗi p, tăng count[k] lên 1 cho mọi bội số k của p. Cuối cùng đếm các số có count[i] == 2.
- **Giải thuật:** Cài đặt Sàng biến thể (đếm số ước nguyên tố).
- **Độ phức tạp:** $O(N \log \log N)$.

12. Prime Square

- **Mã bài:** 1436B
- **Rating:** 900
- **Nguồn:** [2]
- **Tóm tắt đề bài:** Tạo ma trận $n \times n$ gồm các số không âm sao cho mọi số trong ma trận KHÔNG phải số nguyên tố, nhưng tổng hàng và tổng cột đều LÀ số nguyên tố.
- **Phân tích Toán học:** Đây là bài toán xây dựng (Constructive).
  - Ta có thể điền ma trận bằng số 0 và 1. Tuy nhiên 1 không phải số nguyên tố (thỏa mãn điều kiện số hạng), nhưng tổng có thể không nguyên tố.
  - Chiến lược: Điền số 1 vào đường chéo chính và đường chéo phụ (dịch chuyển 1 hàng). Khi đó mỗi hàng và cột sẽ có đúng 2 số 1. Tổng là 2 (số nguyên tố). Các số khác là 0 (hợp lệ vì đề bài cho phép hợp số hoặc số không nguyên tố).
  - Trường hợp đặc biệt: Nếu n lẻ, việc điền chéo có thể khiến một ô trùng nhau, làm tổng là 1. Cần điều chỉnh nhỏ hoặc sử dụng mẫu khác. Một cách tổng quát: Tìm số

nguyên tố $p \geq n$. Điền đường chéo bằng 1, và các ô khác sao cho tổng là p. Nhưng đơn giản nhất là mô hình chu trình ma trận hoán vị sao cho mỗi hàng có đúng 2 số 1.

- **Giải thuật:** Ma trận A với $A_{i,i} = 1$ và $A_{i,(i+1)\%n} = 1$. Tổng mỗi hàng/cột là 2 (nguyên tố). Các phần tử là 0 hoặc 1 (không nguyên tố).
- **Độ phức tạp:** $O(N^2)$.

13. Prime Subtraction

- **Mã bài:** 1238A
- **Rating:** 900
- **Nguồn:** [38]
- **Tóm tắt đề bài:** Cho x, y ($x > y$). Hỏi có thể biến x thành y bằng cách trừ các số nguyên tố hay không?
- **Phân tích Toán học:** Bài toán tương đương với: Hiệu số $D = x - y$ có thể biểu diễn thành tổng các số nguyên tố không?
  - Nếu $D = 1$: Không thể, vì không có số nguyên tố nào bằng 1.
  - Nếu $D > 1$: Luôn có thể. Theo giả thuyết Goldbach (và thực tế đơn giản hơn), mọi số tự nhiên > 1 đều có thể biểu diễn thành tổng các số 2 và 3.
    - Nếu D chẵn: Tổng của nhiều số 2.
    - Nếu D lẻ: Tổng của một số 3 và nhiều số 2.
- **Giải thuật:** Nếu $x - y == 1$ in "NO", ngược lại "YES".
- **Độ phức tạp:** $O(1)$.

14. Odd Divisor

- **Mã bài:** 1475A
- **Rating:** 900
- **Nguồn:** [2]
- **Tóm tắt đề bài:** Kiểm tra xem số n có ước số lẻ nào lớn hơn 1 hay không.
- **Phân tích Toán học:**
  - Một số có ước lẻ lớn hơn 1 khi và chỉ khi nó **không phải** là lũy thừa của 2.
  - Nếu n là lũy thừa của 2 ($n = 2^k$), mọi ước của nó đều là 2^j (số chẵn, trừ số 1).
  - Ngược lại, nếu n có bất kỳ thừa số nguyên tố lẻ nào, nó sẽ có ước lẻ.

- **Giải thuật:** Kiểm tra xem n có phải lũy thừa của 2 không bằng phép toán bit: (n & (n - 1)) == 0. Nếu đúng, in "NO", ngược lại "YES".
- **Độ phức tạp:** $O(1)$.

15. New Year's Number

- **Mã bài:** 1475B
- **Rating:** 900
- **Nguồn:** [2]
- **Tóm tắt đề bài:** Số n có thể biểu diễn thành $n = 2020x + 2021y$ với $x, y \geq 0$ không?
- **Phân tích Toán học:**
  - Biến đổi: $n = 2020x + 2020y + y = 2020(x + y) + y$.
  - Đặt $k = x + y$, ta có $n = 2020k + y$.
  - Điều này tương đương với phép chia có dư: n chia cho 2020 được thương là k và dư là y.
  - Điều kiện cần là số dư y phải nhỏ hơn hoặc bằng thương k (vì $x = k - y$ và $x \geq 0$ nên $k \geq y$).
- **Giải thuật:** Tính div = n / 2020 và mod = n % 2020. Kiểm tra mod <= div.
- **Độ phức tạp:** $O(1)$.

16. Strange Partition

- **Mã bài:** 1471A
- **Rating:** 900
- **Nguồn:** [43]
- **Tóm tắt đề bài:** Cho mảng a. Ta có thể gộp các phần tử liền kề lại với nhau. Hãy tìm giá trị nhỏ nhất và lớn nhất của hàm $f = \sum \lceil a_i / x \rceil$.
- **Phân tích Toán học:** Hàm trần (ceil) có tính chất: $\lceil (a + b)/x \rceil \leq \lceil a/x \rceil + \lceil b/x \rceil$.
  - Tức là gộp lại luôn làm tổng hàm trần nhỏ đi hoặc bằng.
  - Để đạt cực đại (Max): Không gộp gì cả. Tính tổng các $\lceil a_i / x \rceil$.
  - Để đạt cực tiểu (Min): Gộp tất cả lại thành một số duy nhất $\sum a_i$. Tính $\lceil (\sum a_i)/x \rceil$.
- **Giải thuật:** Tính tổng mảng sum và tổng các ceil sum_ceil. In kết quả.

- **Độ phức tạp:** $O(N)$.

17. Orac and Factors

- **Mã bài:** 1350A
- **Rating:** 900
- **Nguồn:** [45]
- **Tóm tắt đề bài:** Cho n, thực hiện k lần phép biến đổi: $n \to n + f(n)$, với $f(n)$ là ước nhỏ nhất lớn hơn 1 của n.
- **Phân tích Toán học:**
  - Nếu n chẵn: $f(n) = 2$. n trở thành $n + 2$, vẫn là số chẵn. Vậy sau k bước, n tăng thêm $2k$.
  - Nếu n lẻ: $f(n)$ là một số lẻ (ước nhỏ nhất của số lẻ là số lẻ). $n \to n + f(n) = \text{lẻ} + \text{lẻ} = \tilde{\text{chẵn}}$. Sau bước đầu tiên, số trở thành chẵn và quay về trường hợp trên.
- **Giải thuật:** Nếu n chẵn, in $n + 2k$. Nếu lẻ, tìm ước nhỏ nhất p (dùng vòng lặp $O(\sqrt{n})$), in $n + p + 2(k - 1)$.
- **Độ phức tạp:** $O(\sqrt{N})$.

Nhóm 3: Ứng dụng Sàng & Toán Nâng cao (Rating 1000)

Nhóm này yêu cầu kỹ năng cài đặt Sàng Eratosthenes hoặc tư duy logic phức tạp hơn về ước số.

18. Noldbach Problem

- **Mã bài:** 17A
- **Rating:** 1000
- **Nguồn:** [47]
- **Tóm tắt đề bài:** Kiểm tra xem có ít nhất k số nguyên tố trong khoảng $[2, n]$ có thể biểu diễn dưới dạng $1 + p_i + p_{i+1}$ (tổng 1 và hai số nguyên tố liên tiếp).
- **Giải thuật:**
  1. Dùng Sàng Eratosthenes tạo list các số nguyên tố đến n ($n \leq 1000$).
  2. Duyệt qua list số nguyên tố: tính $S = p_i + p_{i+1} + 1$.
  3. Nếu $S \leq n$ và S là số nguyên tố (tra cứu mảng Sàng), tăng biến đếm.

4. So sánh biến đếm với k.

- **Độ phức tạp:** Sàng $O(N \log \log N)$.

19. Different Divisors

- **Mã bài:** 1474B
- **Rating:** 1000
- **Nguồn:** [49]
- **Tóm tắt đề bài:** Tìm số a nhỏ nhất có ít nhất 4 ước số, sao cho hiệu giữa hai ước số bất kỳ ít nhất là d.
- **Phân tích Toán học:**
  - Các ước số phải là $1, p, q, pq$ (với p, q là số nguyên tố).
  - Ước thứ nhất là 1. Ước thứ hai là p, phải thỏa mãn $p - 1 \geq d \Rightarrow p \geq 1 + d$.
  - Ước thứ ba là q, phải thỏa mãn $q - p \geq d \Rightarrow q \geq p + d$.
  - Kết quả là $p \times q$.
- **Giải thuật:** Tìm số nguyên tố đầu tiên $\geq 1 + d$ (gọi là p). Tìm số nguyên tố đầu tiên $\geq p + d$ (gọi là q). In $p \times q$. Cần hàm kiểm tra nguyên tố.
- **Độ phức tạp:** Tìm kiếm số nguyên tố rất nhanh vì mật độ số nguyên tố dày đặc.

20. Phoenix and Puzzle

- **Mã bài:** 1515B
- **Rating:** 1000
- **Nguồn:** [34]
- **Tóm tắt đề bài:** Có thể ghép n tam giác vuông cân (giống nhau) thành một hình vuông không?
- **Phân tích Toán học:**
  - Một hình vuông có thể được tạo thành từ 2 tam giác (ghép cạnh huyền) hoặc 4 tam giác (ghép góc vuông vào tâm).
  - Tổng quát: Hình vuông kích thước $k \times k$ được tạo từ k^2 hình vuông nhỏ.
  - Nếu ghép từ mẫu 2 tam giác: $n = 2 \times k^2$.
  - Nếu ghép từ mẫu 4 tam giác: $n = 4 \times k^2$.
- **Giải thuật:** Kiểm tra xem n có chia hết cho 2 và thương là số chính phương, HOẶC n chia hết cho 4 và thương là số chính phương.

- **Độ phức tạp:** $O(1)$ với hàm sqrt.

---

5. Chiến lược Tối ưu hóa và Sai lầm Thường gặp

Đối với Newbie (Rating 800-1000), việc mắc lỗi khi cài đặt các thuật toán số học là điều không tránh khỏi. Dưới đây là tổng hợp các sai lầm phổ biến và cách khắc phục:

5.1 Sai lầm về Độ phức tạp (TLE)

- **Lỗi:** Sử dụng vòng lặp từ 1 đến N để kiểm tra nguyên tố cho nhiều số (Test cases $t = 1000, n = 10^9$).
- **Khắc phục:** Với N lớn, luôn dùng $O(\sqrt{N})$. Với nhiều test case và N nhỏ (10^6), bắt buộc phải Sàng trước (Precomputation) bên ngoài vòng lặp test case.

5.2 Sai lầm về Tràn số (Overflow)

- **Lỗi:** Tính i * i trong vòng lặp for (int i = 2; i * i <= n; i++) khi n là long long nhưng i là int. Khi i đạt 46341, i^2 sẽ vượt quá giới hạn của int (khoảng 2×10^9) gây tràn số âm, dẫn đến vòng lặp vô hạn.
- **Khắc phục:** Luôn khai báo i là long long hoặc ép kiểu 1LL * i * i.

5.3 Sai lầm về Trường hợp biên

- **Lỗi:** Coi 1 là số nguyên tố. Coi 2 không phải là số nguyên tố vì nó chẵn.
- **Khắc phục:** Ghi nhớ định nghĩa: Số nguyên tố ≥ 2. Số 1 không phải nguyên tố cũng không phải hợp số. Xử lý riêng trường hợp $N = 1$.

5.4 Lạm dụng hàm pow và sqrt

- **Lỗi:** Sử dụng pow(a, b) để tính lũy thừa nguyên, hoặc so sánh sqrt(n) == floor(sqrt(n)). Sai số dấu phẩy động (floating point precision) có thể dẫn đến kết quả sai (ví dụ pow(5, 2) có thể ra 24.999999 $\rightarrow$ 24).
- **Khắc phục:** Sử dụng hàm lũy thừa nhị phân (Binary Exponentiation) cho số nguyên. Với kiểm tra số chính phương, dùng round(sqrt(n)) và bình phương lại để kiểm tra.

---

6. Kết luận

Việc làm chủ thuật toán kiểm tra tính nguyên tố $O(\sqrt{N})$ và Sàng Eratosthenes là bước đi đầu tiên và quan trọng nhất trên hành trình chinh phục đỉnh cao của lập trình thi đấu. Không chỉ dừng lại ở việc "copy-paste" mã nguồn, người học cần thấu hiểu bản chất toán học đằng sau mỗi dòng lệnh – từ định lý về phân bố ước số đến chuỗi điều hòa trong phân tích độ phức tạp.

20 bài tập Codeforces được phân tích trong báo cáo này đóng vai trò là "giao diện" kết nối lý thuyết khô khan với thực tiễn sinh động. Bằng cách giải quyết trọn vẹn các bài tập này, người mới bắt đầu sẽ xây dựng được phản xạ thuật toán nhạy bén, tạo tiền đề vững chắc để tiến lên các mức rating cao hơn (Pupil, Specialist) và tiếp cận các chủ đề nâng cao như Số học Module hay Lý thuyết đồ thị.

Bảng Tổng hợp Bài tập Tham khảo

Tên bài	Mã bài	Rating	Dạng bài
Bachgold Problem	749A	800	Constructive / Greedy
Design Tutorial: Learn from Math	472A	800	Composite Numbers
Maximum GCD	1370A	800	GCD Properties
EhAb AnD gCd	1325A	800	Constructive / GCD
Subtract or Divide	1451A	800	Greedy / Parity
Equation	1269A	800	Composite Construction
GCD Sum	1498A	800	Brute Force / GCD
Fair Division	1472B	800	Implementation / Math
Sum of 2050	1517A	800	Digit DP / Math

Red and Blue Beans	1519A	800	Math / Inequalities
Almost Prime	26A	900	Sieve / Factor Counting
Prime Square	1436B	900	Matrix Construction
Prime Subtraction	1238A	900	Number Theory Logic
Odd Divisor	1475A	900	Bitwise / Powers of 2
New Year's Number	1475B	900	Diophantine / Brute Force
Strange Partition	1471A	900	Math / Ceilings
Orac and Factors	1350A	900	Smallest Prime Factor
Noldbach Problem	17A	1000	Sieve of Eratosthenes
Different Divisors	1474B	1000	Prime Gaps / Sieve
Phoenix and Puzzle	1515B	1000	Square Numbers

Nguồn trích dẫn

1. Problem - C - Codeforces | PDF | Numbers - Scribd, truy cập vào tháng 1 22, 2026, https://www.scribd.com/document/853102667/Problem-C-Codeforces
2. List of short Codeforces problems with a statement of 1000 characters or less. - GitHub, truy cập vào tháng 1 22, 2026, https://github.com/alwa97/codeforces-1000
3. Complete Competitive Programming Notebook For CP-31 (800-1000 Rating) | PDF - Scribd, truy cập vào tháng 1 22, 2026, https://www.scribd.com/document/901594877/Complete-Competitive-Programming-Notebook-for-CP-31-800-1000-Rating

4. Primality Testing | Brilliant Math & Science Wiki, truy cập vào tháng 1 22, 2026, https://brilliant.org/wiki/prime-testing/
5. Number Theory: Primality Test in O(sqrt(n)) - DEV Community, truy cập vào tháng 1 22, 2026, https://dev.to/priyanka__488/number-theory-primality-test-in-o-sqrt-n-dde
6. Prime numbers within a range-Sieve of Eratosthenes | by Rohan Gupta - Medium, truy cập vào tháng 1 22, 2026, https://medium.com/theleanprogrammer/prime-numbers-within-a-range-sieve-of-eratosthenes-b3055de35352
7. Sieve of Eratosthenes | Competitive Programming with Python - YouTube, truy cập vào tháng 1 22, 2026, https://www.youtube.com/watch?v=-wlYdmPiHcA
8. Sieve of Eratosthenes - Algorithms for Competitive Programming, truy cập vào tháng 1 22, 2026, https://cp-algorithms.com/algebra/sieve-of-eratosthenes.html
9. The Prime Number Test: A Lesson in Efficiency | by kodaman - Medium, truy cập vào tháng 1 22, 2026, https://medium.com/@binboavetonik/the-prime-number-test-a-lesson-in-efficiency-b6e773237a55
10. L01: Primality Test O(SQRT(N)) | NT 2025 Course - YouTube, truy cập vào tháng 1 22, 2026, https://www.youtube.com/watch?v=oLxPBQ7dv_U
11. L01 : Primality Test in O(sqrt(N)) Time | Number Theory | CodeNCode - YouTube, truy cập vào tháng 1 22, 2026, https://www.youtube.com/watch?v=GXY6orMXU4c
12. Sieve of Eratosthenes Algorithm - Topcoder, truy cập vào tháng 1 22, 2026, https://www.topcoder.com/thrive/articles/sieve-of-eratosthenes-algorithm
13. Count Primes: Sieve of Eratosthenes Algorithm Explained (LeetCode 204) - YouTube, truy cập vào tháng 1 22, 2026, https://www.youtube.com/watch?v=lxVSPVWRTao
14. Sieve of Eratosthenes - Algorithms for Competitive Programming, truy cập vào tháng 1 22, 2026, https://cp-algorithms.web.app/algebra/sieve-of-eratosthenes.html
15. List of 100 beginner friendly questions on codeforces - GitHub, truy cập vào tháng 1 22, 2026, https://github.com/ankitvashisht12/100-questions-of-codeforces
16. Problem - 2093C - Codeforces | PDF | Prime Number - Scribd, truy cập vào tháng 1 22, 2026, https://www.scribd.com/document/910881428/Problem-2093C-Codeforces
17. Competitive Coding Challenges | PDF | Mathematics - Scribd, truy cập vào tháng 1 22, 2026, https://www.scribd.com/document/801869252/Questions
18. Codeforces-Solution/749 A. Bachgold Problem.cpp at master - GitHub, truy cập vào tháng 1 22, 2026, https://github.com/SaruarChy/Codeforces-Solution/blob/master/749%20A.%20Bachgold%20Problem.cpp
19. CodeforcesForBeginners - - YouTube, truy cập vào tháng 1 22, 2026, https://www.youtube.com/hashtag/codeforcesforbeginners
20. Bootstrap Module 2 | PDF | Technology & Engineering - Scribd, truy cập vào tháng 1 22, 2026, https://www.scribd.com/document/740606198/Bootstrap-Module-2

21. vikashpatel24/CodeForces-Solution: My solutions of codeforces practice questions - GitHub, truy cập vào tháng 1 22, 2026, https://github.com/vikashpatel24/CodeForces-Solution
22. Maximum GCD || Codeforces Round #651 (Div. 2) - YouTube, truy cập vào tháng 1 22, 2026, https://www.youtube.com/watch?v=1AOLn4B4BS0
23. Codeforces Problems - Kaggle, truy cập vào tháng 1 22, 2026, https://www.kaggle.com/datasets/lborgav/codeforces-problems
24. Codeforces-Solution/1370 A. Maximum GCD.cpp at master - GitHub, truy cập vào tháng 1 22, 2026, https://github.com/SaruarChy/Codeforces-Solution/blob/master/1370%20A.%20Maximum%20GCD.cpp
25. Number Theory: GCD && LCM - Virtual Judge, truy cập vào tháng 1 22, 2026, https://vjudge.net/contest/580126
26. 1325 A. EhAb AnD gCd.cpp - SaruarChy/Codeforces-Solution - GitHub, truy cập vào tháng 1 22, 2026, https://github.com/SaruarChy/Codeforces-Solution/blob/master/1325%20A.%20EhAb%20AnD%20gCd.cpp
27. 題库 - Hydro, truy cập vào tháng 1 22, 2026, https://hydro.ac/p?page=633
28. Solutions to Codeforces Problems - GitHub, truy cập vào tháng 1 22, 2026, https://github.com/kantuni/Codeforces
29. tanvir-robin/CP-Solutions: Bunches of codes that submitted to several online judges by me., truy cập vào tháng 1 22, 2026, https://github.com/tanvir-robin/CP-Solutions
30. 2223. Sum of Scores of Built Strings - In-Depth Explanation - AlgoMonster, truy cập vào tháng 1 22, 2026, https://algo.monster/liteproblems/2223
31. rakib181/Codeforces - GitHub, truy cập vào tháng 1 22, 2026, https://github.com/rakib181/Codeforces
32. B. Find The Array | Educational Codeforces Round 100 | CODE EXPLAINER - YouTube, truy cập vào tháng 1 22, 2026, https://www.youtube.com/watch?v=FUE95IGXkNg
33. Problems - Contest Mania, truy cập vào tháng 1 22, 2026, https://contestmania.web.app/problems
34. kbakdev/codeforces: Solution to Codeforces - GitHub, truy cập vào tháng 1 22, 2026, https://github.com/kbakdev/codeforces
35. Almost Prime (Codeforces 26A) | Enhanced Sieve of Eratosthenes + Number Theory Basics, truy cập vào tháng 1 22, 2026, https://www.youtube.com/watch?v=FYglrC--m-E
36. CodeForces/26A - Almost Prime.cpp at master - GitHub, truy cập vào tháng 1 22, 2026, https://github.com/fuwutu/CodeForces/blob/master/26A%20-%20Almost%20Prime.cpp
37. B. Prime Square | Codeforces Round #678 (Div. 2) - YouTube, truy cập vào tháng 1 22, 2026, https://www.youtube.com/watch?v=ZiaF9LE-20M
38. Prime Subtraction - CodeForces 1238A - Virtual Judge, truy cập vào tháng 1 22, 2026, https://vjudge.net/problem/CodeForces-1238A

39. S02E15 : CodeForces 900 Easy Rating for Beginners | TECH_ED - YouTube, truy cập vào tháng 1 22, 2026, https://www.youtube.com/watch?v=qwy_A9ojZzE
40. 26 Odd Divisor | Video Solution | 900 Rated | TLE CP-31 Sheet | Best Codeforces Problems, truy cập vào tháng 1 22, 2026, https://www.youtube.com/watch?v=UQ_rdYE4YAA
41. 1475 A. Odd Divisor.cpp - SaruarChy/Codeforces-Solution - GitHub, truy cập vào tháng 1 22, 2026, https://github.com/SaruarChy/Codeforces-Solution/blob/master/1475%20A.%20Odd%20Divisor.cpp
42. Codeforces-Solution/1475 B. New Year's Number.cpp at master - GitHub, truy cập vào tháng 1 22, 2026, https://github.com/SaruarChy/Codeforces-Solution/blob/master/1475%20B.%20New%20Year's%20Number.cpp
43. Codeforces-Solution/1471 A. Strange Partition.cpp at master - GitHub, truy cập vào tháng 1 22, 2026, https://github.com/SaruarChy/Codeforces-Solution/blob/master/1471%20A.%20Strange%20Partition.cpp
44. Strange Partition | Codeforces | @TLE_Eliminators 's CP31 | 900 Rated | Q-27 - YouTube, truy cập vào tháng 1 22, 2026, https://www.youtube.com/watch?v=GJrRe0bq248
45. jspw/Compititive-Programming: Problems solved from Codeforces and hackerRank in c/c++, truy cập vào tháng 1 22, 2026, https://github.com/jspw/Compititive-Programming
46. Codeforces Round #641 (Div. 2)题目+题解（A、B、C）原创 - CSDN博客, truy cập vào tháng 1 22, 2026, https://blog.csdn.net/SDAU_LGX/article/details/106095337
47. CodeForces rating 1000 题目集 - Virtual Judge, truy cập vào tháng 1 22, 2026, https://vjudge.net/article/4925
48. A2OJ Category: Codeforces Div. 2 - A Problems - earthshakira.github.io, truy cập vào tháng 1 22, 2026, https://earthshakira.github.io/a2oj-clientside/server/Category407.html
49. #23 Different Divisors | Video Solution | 1000 Rated | TLE CP-31 Sheet | Best Codeforces Problems - YouTube, truy cập vào tháng 1 22, 2026, https://www.youtube.com/watch?v=0w7MPodjtK8
50. 1474 B Different Divisors.cpp - dg-029/Codeforces-solutions - GitHub, truy cập vào tháng 1 22, 2026, https://github.com/dg-029/Codeforces-solutions/blob/master/1474%20B%20Different%20Divisors.cpp
51. tridibsamanta/Codeforces_Solutions: My solution to some problems listed on Codeforces., truy cập vào tháng 1 22, 2026, https://github.com/tridibsamanta/Codeforces_Solutions