SỐ HỌC TRONG LẬP TRÌNH

Giáo viên: Hoàng Đăng Quang - 0903781593

SỐ HỌC TRONG LẬP TRÌNH

1. Các khái niệm số học và ứng dụng
2. Lý thuyết đồng dư
3. Bài tập.

Giáo viên: Hoàng Đăng Quang - 0903781593

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Phép chia hết

- Nếu $a : b = q$ thì $a = bq$.
- Nếu $a : b = q$ và $q \neq 0$ thì $a : q = b$.

Phép chia có dư

- Khi $r = 0$ ta có phép chia hết.
- Khi $r \neq 0$ ta có phép chia có dư. Ta nói: a chia cho b được thương là q và số dư là r.
Kí hiệu: $a : b = q$ (dư r).

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Phép tính lũy thừa

Khi nhân hai luỹ thừa cùng cơ số, ta giữ nguyên cơ số và cộng các số mũ:

$$a^m . a^n = a^{m+n}.$$

Khi chia hai luỹ thừa cùng cơ số (khác 0), ta giữ nguyên cơ số và trừ các số mũ:

$$a^m : a^n = a^{m-n} \ (a \neq 0; m \geq n).$$

Quy ước: $a^0 = 1 \ (a \neq 0)$.

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Thứ tự thực hiện phép tính

Khi biểu thức có các phép tính cộng, trừ, nhân, chia, nâng lên luỹ thừa, ta thực hiện phép tính nâng lên luỹ thừa trước, rồi đến nhân và chia, cuối cùng đến cộng và trừ.

Khi biểu thức có chứa dấu ngoặc, ta thực hiện các phép tính trong dấu ngoặc trước.

Nếu biểu thức chứa các dấu ngoặc ( ), [ ], { } thì thứ tự thực hiện các phép tính như sau: ( ) $\longrightarrow$ [ ] $\longrightarrow$ { }.

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Quan hệ chia hết

- Nếu số dư trong phép chia a cho b bằng 0 thì a chia hết cho b, kí hiệu là $a \vdots b$.
- Nếu số dư trong phép chia a cho b khác 0 thì a không chia hết cho b, kí hiệu là $a \not\vdots b$.

Với a là số tự nhiên khác 0 thì:

- a là ước của a;
- a là bội của a;
- 0 là bội của a;
- 1 là ước của a.

Giáo viên: Hoàng Đăng Quang - 0903781593

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Tính chất chia hết

Với $a \geq b$:
Nếu $a \vdots m$ và $b \vdots m$ thì
$(a-b) \vdots m$.
Khi đó ta có:
$(a-b) : m = a : m - b : m$.

Nếu $a \vdots m$ thì $(a \,.\, b) \vdots m$
với mọi số tự nhiên b.

Nếu $a \vdots m$ và $b \vdots m$ thì
$(a+b) \vdots m$.
Khi đó ta có:
$(a+b) : m = a : m + b : m$.

Giáo viên: Hoàng Đăng Quang - 0903781593

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Dấu hiệu chia hết cho 2 và 5

Các số có chữ số tận cùng là 0, 2, 4, 6, 8 thì chia hết cho 2 và chỉ những số đó mới chia hết cho 2.

Các số có chữ số tận cùng là 0 hoặc 5 thì chia hết cho 5 và chỉ những số đó mới chia hết cho 5.

Giáo viên: Hoàng Đăng Quang - 0903781593

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Dấu hiệu chia hết cho 3 và 9

Các số có tổng các chữ số chia hết cho 3 thì chia hết cho 3 và chỉ những số đó mới chia hết cho 3.

Các số có tổng các chữ số chia hết cho 9 thì chia hết cho 9 và chỉ những số đó mới chia hết cho 9.

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Dấu hiệu chia hết cho 4, 6, 7, 8

Một số chia hết cho 4 nếu **hai chữ số cuối** của nó là một số **chia hết cho 4**. Đây là một quy tắc đơn giản để kiểm tra dấu hiệu chia hết cho 4

Một số chia hết cho 6 nếu nó **chia hết** cho cả **2 và 3**. Để kiểm tra một số có chia hết cho 6 hay không, bạn có thể áp dụng cả hai quy tắc kiểm tra chia hết cho 2 và 3

Quy tắc kiểm tra chia hết cho 7:

Lấy chữ số cuối của số và nhân nó với 2.

Trừ kết quả từ bước 1 từ phần còn lại của số khi loại bỏ chữ số cuối.

Nếu kết quả thu được chia hết cho 7, thì số ban đầu cũng chia hết cho 7.

Quy tắc kiểm tra chia hết cho 8:

Lấy ba chữ số cuối của số và tạo thành một số.

Nếu số tạo thành từ ba chữ số cuối chia hết cho 8, thì số ban đầu cũng chia hết cho 8.

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Số chính phương

Số chính phương là số mà căn bậc hai của nó là một số nguyên. Nói cách khác, một số chính phương là bình phương của một số nguyên.

Hàm kiểm tra số chính phương

Cách 1: sử dụng sqrt

```
 6   bool CheckPrefectSquare(ll so)
 7   {
 8       if (so < 1) return false;
 9       ll c = sqrt((long double)so);
10       return (c*c == so);
11   }
```

Cách 2: Sử dụng tìm kiếm nhị phân

```
13  bool CheckPrefectSquareBinSreach(ll so)
14  {
15      if (so < 1) return false;
16      ll left = 1, right = so;
17      while (left <= right) {
18          ll mid = left + (right - left) / 2;
19          if (mid > so/mid) right = mid-1;
20          else if (mid < so/mid) left = mid+1;
21              else return so % mid == 0;
22      }
23      return false;
24  }
```

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Cách 3: sử dụng phương trình Newton (tìm nghiệm gần đúng của phương trình f(x) = 0)

Gọi **N** là số muốn kiểm tra → ta cần tìm một số nguyên **x** sao cho $x^2 = N$

Chuyển về phương trình có dạng $x^2 - N = 0$

Ta cần tìm nghiệm nguyên của phương trình $f(x) = x^2 - N$. Nếu ta tìm được một nghiệm nguyên **x** mà $x^2 = N$ thì **N** là số chính phương

Công thức Newton x = (x + N / 2) / 2

Các bước thuật toán

Giá trị hiện tại **pre_x = 0**

Chọn một giá trị phỏng đoán **x = N/2**

Lặp lại khi giá trị phỏng đoán khác giá trị hiện tại

- Lưu giá trị phỏng đoán vào giá trị x hiện tại
- Áp dụng công thức Newton tính x mới
- Nếu x mới lớn hơn x hiện tại thì thoát vòng lặp

Nếu **x * x = N** thì N là số chính phương

```
26  bool CheckPrefectSquareNewton(ll so)
27  {
28      if (so < 1) return false;
29      ll pre_x = 0;
30      ll x = so/2;
31      while (x != pre_x) {
32          pre_x = x;
33          x = (x + so / x) / 2;
34          if (x > pre_x) {
35              x = pre_x;
36              break;
37          }
38      }
39      return x * x == so;
40  }
```

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Hàm in các số chính phương nằm trong đoạn a, b

Phân tích : Để liệt kê các số chính phương từ 1 tới n ta gọi số chính phương đó có dạng i^2 trong đó i là một số nguyên

Từ đó ta có bất phương trình $a \le i^2 \le b \Rightarrow \sqrt{a} \le i \le \sqrt{b}$, vậy để in ra các số chính phương từ 1 tới n bạn chỉ cần duyệt các số i từ 1 tới $\sqrt{n}$ và in ra bình phương của i là được.

```
15  void PrintPrefectSquare(ll a, ll b)
16  {
17      ll c1 = sqrt(a), c2 = sqrt(b);
18      if (c1*c1 < a || c1 < 1) c1++;
19      for (int i = c1; i <= c2; i++)
20          cout << i*i << " ";
21  }
```

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Số nguyên tố

Số nguyên tố là một số tự nhiên lớn hơn 1 và chỉ có hai ước số dương khác nhau là 1 và chính nó. Điều này có nghĩa là số nguyên tố không chia hết cho bất kỳ số tự nhiên nào khác ngoài 1 và chính nó.

Phương pháp ngây thơ (vét cạn) – độ phức tạp O(n)

Duyệt qua tất cả các số nguyên i từ 2 đến N-1 và nếu phát hiện i là ước của N thì N không phải là số nguyên tố. Duyệt đến cuối cùng mà không tìm được ước thì N là số nguyên tố

```
15  bool CheckPrime(ll so)
16  {
17      if (so < 2) return false;
18      for (int i = 2; i < so; ++i)
19          if (so % i == 0) return false;
20      return true;
21  }
```

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Phương pháp tối ưu duyệt ứng dụng căn bậc 2 – Độ phức tạp O($\sqrt{N}$)

Ý tưởng dựa trên nhận xét: nếu số N có một ước số d lớn hơn $\sqrt{N}$, thì nó cũng phải có một ước số tương ứng là N/d nhỏ hơn $\sqrt{N}$. Do đó, để tìm một ước của N (khác 1 và N), ta chỉ cần tìm kiếm trong đoạn [2, $\sqrt{N}$]. Nếu không tìm thấy ước nào trong đoạn này, thì N chắc chắn là số nguyên tố.

```
23  bool CheckPrime1(ll so)
24  {
25      if (so < 2) return false;
26      for (ll i = 2; i*i <= so; i++)
27          if (so % i == 0) return false;
28      return true;
29  }
```

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Phương pháp tối ưu dựa trên nhận xét 6k ± 1

Các số có dạng như 6k, 6k + 2, 6k + 4 đều chia hết cho 2, và dạng 6k + 3 chia hết cho 3

Mọi số nguyên tố **lớn hơn 3** đều có dạng **6k + 1** hoặc **6k + 5**. Mà **6k + 5** ta có thể viết lại là **6(k+1) – 1**. Vì vậy mọi số nguyên tố **lớn hơn 3** đều có dạng **6k ± 1**

Do đó, sau khi xử lý riêng các trường hợp N=2 và N=3, ta chỉ cần kiểm tra các số chia có dạng 6k ± 1 (tức là 5, 7, 11, 13,...) cho đến $\sqrt{N}$

Các bước thực hiện

Xử lý các trường hợp đặc biệt **N < 2, N%2, N%3, N = 2, N = 3**

Duyệt **i = 5** đến $\sqrt{N}$

Kiểm tra **N** có **chia hết** cho **i** (dạng 6k – 1) hoặc **N** có **chia hết** cho **i+2** (dạng 6k + 1)

Tăng **i** lên **6** để kiểm tra cặp **i, i+2** tiếp theo (Ví dụ 5, 7 thì cặp tiếp theo là 11, 13, ...)

Nếu kết thúc vòng lặp mà không phát hiện chia hết thì N là số nguyên tố

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Phương pháp tối ưu dựa trên nhận xét 6k ± 1

```
31  bool CheckPrime2(ll so)
32  {
33      if (so < 2) return false;
34      if (so % 2 == 0 || so % 3 == 0) return false;
35      if (so == 2 || so == 3) return true;
36      for (ll i = 5; i*i <= so; i+=6)
37          if (so % i == 0 || so % (i+2) == 0) return false;
38      return true;
39  }
```

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Sàng số nguyên tố Eratosthenes

Bước 1: Tạo mảng bool đánh dấu các số nguyên tố từ 2 đến giới hạn cần kiểm tra.

Bước 2: Bắt đầu từ số đầu tiên trong danh sách (là số 2), đánh dấu tất cả các bội số của nó là không nguyên tố.

Bước 3: Chọn số nguyên tố tiếp theo chưa được đánh dấu và lặp lại bước 2.

Bước 4: Lặp lại bước 3 cho đến khi bạn đã kiểm tra tất cả các số trong danh sách.

```
28  void SieveOfEratosthenes(ll n)
29  {
30      IsPrime[0] = 0; IsPrime[1] = 0;
31      for (ll i = 2; i*i <= n; i++)
32          if (IsPrime[i] == 1)
33              for (ll j = i*i; j <= n; j+=i)
34                  IsPrime[j] = 0;
35  }
```

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Kiểm tra tính nguyên tố của một dãy số: Mỗi phần tử trong dãy có giá trị **$1 < a_i <= 10^{14}$**

Sử dụng kỹ thuật **Sàng phân đoạn (Segmented Sieve)**:

Để tìm số nguyên tố trong đoạn **[L, R]**, chúng ta dựa trên nguyên lý: Một hợp số **N** luôn có ít nhất một ước nguyên tố nhỏ hơn hoặc bằng $\sqrt{N}$.

Vì **$R_{max} = 10^{14}$**, nên ta chỉ cần các số nguyên tố nhỏ làm **"mẫu thử"** trong phạm vi $\sqrt{N}$ **= 10^7**.

Các bước thuật toán:

Bước 1 (sàng cơ bản): Sử dụng Sàng Eratosthenes lọc ra các số nguyên tố từ 2 đến 10^7 và lưu vào mảng primes.

Bước 2 (sàng đoạn): Tạo một mảng đánh dấu cho đoạn [L, R]. Với mỗi số nguyên tố trong mảng primes[i] tìm được ở Bước 1, ta loại bỏ các bội số của primes[i] nằm trong đoạn [L, R].

Bước 3: Những số còn lại trong đoạn [L, R] chưa bị đánh dấu chính là số nguyên tố.

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Bước 1: Tạo sàng cơ bản

```
166  vector<int> sieve_base(int limit)
167  {
168      vector<bool> is_prime(limit + 1, true);
169      is_prime[0] = is_prime[1] = false;
170
171      for (int p = 2; p * p <= limit; p++) {
172          if (is_prime[p]) {
173              for (int i = p * p; i <= limit; i += p)
174                  is_prime[i] = false;
175          }
176      }
177      vector<int> primes;
178      for (int p = 2; p <= limit; p++) {
179          if (is_prime[p]) {
180              primes.push_back(p);
181          }
182      }
183      return primes;
184  }
```

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Bước 2, 3: Tạo mảng đánh dấu và thực hiện sàng phân đoạn

Tạo mảng **primes** chứa các số nguyên tố đã sàng được từ **2** đến 10^7

Tạo mảng **is_prime_range** có kích thước **R – L + 1**

is_preme_range[i] tương ứng với số thực tế L + i trong đoạn từ L đến R

*Vì ta không thể tạo một mảng có chỉ số lên đến 10^{14} nên ta dùng kỹ thuật **tịnh tiến mảng.***

Vì giới hạn R – L <= 10^6 nên ta cắt đoạn L đến R và đặt nó vào một mảng bắt đầu từ 0

Duyệt mảng **primes**

Tìm **bội số đầu tiên** của **primes[i]** nằm trong đoạn **[L, R]**

Gọi **start** là **bội số đầu tiên** của **primes[i]** lớn hơn hoặc bằng **L**, để tính được giá trị này ta thực hiện như sau:

start = K * primes[i] >= L

Để **start** là **bội số đầu tiên** của **primes[i] >= L** ta cần tìm số **K nhỏ nhất**

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

K = (L + primes[i] – 1) / primes[i] *(đây là kỹ thuật chia làm tròn lên)*

start = K * primes[i] *sẽ ra giá trị bội số đầu tiên của* ***primes[i]*** *lớn hơn hoặc bằng* ***L***

Ví dụ: L = 10, primes[i] = 3*, kết quả mong muốn là* ***12*** *(vì 9 < 10 nên bội tiếp theo là 12)*

Tính tử số: ***L + primes[i] – 1 = 10 + 3 – 1 = 12***

Chia nguyên để thu được ***K = 12 / 3 = 4***

Nhân lại ***K*** *với* ***primes[i] = 4 * 3 = 12***

start = (L + primes[i] – 1) / primes[i] * primes[i]

Xét trường hợp đặc biệt nếu **start < primes[i] * primes[i] thì start = primes[i]**

Duyệt từ **j = start** đến **R** đánh dấu các vị trí bội của **primes[i]** trong mảng **is_prime_range** thành **false.**

is_prime_range[j – L] = false

Để hiểu tại sao ta lại đánh dấu tại vị trí ***j – L*** *trong mảng* ***is_prime_range*** *ta xét ví dụ*

Tìm số nguyên tố trong đoạn ***L = 100****,* ***R = 105*** *→ mảng* ***is_prime_range*** *sẽ có kích thước*

105 – 100 + 1 = 6 *phần tử*

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Ta sẽ có ánh xạ giữa chỉ số mảng với giá trị thực tế:

i	***(L + i)***	***số thực tế (j)***	***trạng thái (is_prime_range[i])***
0	*100 + 0*	*100*	*false (vì chia hết cho 2)*
1	*100 + 1*	*101*	*true (số nguyên tố)*
2	*100 + 2*	*102*	*false (vì chia hết cho 2)*
3	*100 + 3*	*103*	*true (số nguyên tố)*
4	*100 + 4*	*104*	*false (vì chia hết cho 2)*
5	*100 + 5*	*105*	*false (vì chia hết cho 5)*

Xử lý trường hợp **L = 1** thì **is_prime_range[0] = false**

Để xuất các số nguyên tố trong đoạn **L** đến **R** ta duyệt mảng **is_prime_range**

Nếu **is_prime_range[i] = true** thì xuất **L + i** chính là số nguyên tố trong đoạn **L** đến **R** đã kiểm tra được

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

```cpp
// Hàm Sàng phân đoạn: Tìm số nguyên tố trong đoạn [L, R]
void segmented_sieve(long long L, long long R) {
    // 1. Tính giới hạn cần sàng sơ cấp: sqrt(R)
    long long limit = sqrt(R);
    vector<int> primes = sieve_base(limit);

    // 2. Tạo mảng đánh dấu cho đoạn [L, R]
    // Kích thước mảng là (R - L + 1).
    // is_prime_range[i] tương ứng với số thực tế là (L + i)
    long long n = R - L + 1;
    vector<bool> is_prime_range(n, true);

    // 3. Loại bỏ bội số của các số nguyên tố cơ sở trong đoạn [L, R]
    for (int p : primes) {
        // Tìm bội số đầu tiên của p nằm trong đoạn [L, R]
        // Công thức: (L + p - 1) / p * p giúp tìm bội số của p >= L
        long long start = (L + p - 1) / p * p;

        // Trường hợp đặc biệt: Nếu start nhỏ hơn p*p thì bắt đầu từ p*p
        // (để tránh đánh dấu chính số nguyên tố p là hợp số nếu p nằm trong [L, R])
        if (start < p * p) start = p * p;

        // Bắt đầu đánh dấu từ start, bước nhảy là p
        for (long long j = start; j <= R; j += p) {
            // Chỉ số trong mảng is_prime_range là (j - L)
            is_prime_range[j - L] = false;
        }
    }
```

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

```
215         // Xử lý trường hợp đặc biệt: số 1 không phải là số nguyên tố
216         if (L == 1) {
217             is_prime_range[0] = false;
218         }
219
220         // 4. In kết quả hoặc lưu trữ
221         // Ở đây tôi sẽ đếm số lượng để kiểm tra, bạn có thể in ra nếu muốn
222         long long count = 0;
223         cout << "Các số nguyên tố trong đoạn [" << L << ", " << R << "]:\n";
224
225         // Chỉ in tối đa 20 số đầu tiên để tránh tràn màn hình nếu đoạn quá lớn
226         int print_limit = 20;
227
228         for (int i = 0; i < n; i++) {
229             if (is_prime_range[i])
230                     cout << (L + i) << " ";
231         }
232     }
```

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Sàng số nguyên tố cải tiến

Sàng có công dụng đánh dấu các vị trí không phải là số nguyên tố kèm theo số lượng ước nguyên tố của số đó.

```
 7  void SangSNTNew(ll n)
 8  {
 9      for (int i = 2; i <= n/2; i++)
10          if (p[i] == 0)
11          {
12              ll j = i+i;
13              while (j <= n)
14              {
15                  p[j]++;
16                  j += i;
17              }
18      }
19  }
```

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Thuật toán phân tích ra thừa số nguyên tố

Số bằng 1 thì xuất 1 và kết thúc thuật toán

Duyệt i = 2 đến căn bậc hai của số muốn phân tích.

Trong khi số còn chia hết cho i thì:

Xuất i

Chia số cho i

Nếu số lớn hơn 1 thì xuất số

```
39  void PrimeFactorization (ll so)
40  {
41      if (so == 1) cout << so;
42      else for (int i = 2; i*i <= so; ++i) {
43              while (so % i == 0)
44                  cout << i << " ";
45                  so /= i;
46              }
47      if (so > 1) cout << so;
48  }
```

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Tính tổng ước của số nguyên

Phương pháp cơ bản: duyệt các số từ 1 đến N và kiểm tra tính chia hết của N với các số đó. Đây là cách làm dễ hiểu nhưng lại không tối ưu về mặt thời gian thực thi. Độ phức tạp thuật toán O(n)

Phương pháp tối ưu 1:

Phân tích số nguyên n ra thừa số nguyên tố có dạng

$n = p^{m1} * p^{m2} * p^{m3} * \dots * p^{mk}$

Tổng số ước của n sẽ được tính bằng công thức

$(m_1 + 1) * (m_2 + 1) * (m_3 + 1) * \dots * (m_k + 1)$

Ví dụ: $n = 100 = 2^2 * 5^2 = (2+1) * (2+1) = 9$ ước (1, 2, 4, 5, 10, 20, 25, 50, 100)

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Tính tổng ước của số nguyên

Phương pháp tối ưu 1:

```
81  ll CountDivisor1()
82  {
83      //pf là mảng chứa bậc các thừa số nguyên tố của n
84      ll res = 1;
85      for (ll i = 0; i <= d1; i++)
86          if (pf[i] > 0) res = res * (pf[i] + 1);
87      return res;
88  }
```

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Tính tổng ước của số nguyên

Phương pháp tối ưu 2: Trong phương pháp này, ta chỉ cần duyệt từ 1 đến $\sqrt{n}$ là đủ

Giải thích : Để tìm được tổng ước hay đếm ước của n ta cần xét tất cả các ước của n, vậy nếu duyệt từ 1 tới $\sqrt{n}$ thì sẽ không duyệt được các ước lớn hơn $\sqrt{n}$. Ví dụ với n = 60 thì $\sqrt{n}$ = 7 (lấy số nguyên) sẽ không xét được các ước như 20, 30, 60 ?

Với số tự nhiên n, bạn luôn có thể viết n thành tích của 2 ước của nó. Ví dụ với n bằng 60 thì bạn có thể viết thành 1x60, 2x30, 3x20, 4x15, 5x12, 6x10.

Giả sử viết n = a * b và a ≤ b trong đó a và b tương ứng với 2 ước của N thì chắc chắn a ≤ $\sqrt{n}$, vì nếu a > $\sqrt{n}$ thì b > $\sqrt{n}$ và khi đó tích của a và b sẽ vượt quá n.

Vậy nên khi xét tất cả các ước của N thì ta chỉ cần xét được ước nhỏ hơn ($\sqrt{n}$) và từ ước nhỏ hơn đó suy ra được ước còn lại.

Chú ý: Với n là số chính phương thì sẽ xảy ra trường hợp 2 ước a và b bằng nhau, khi đó bạn chỉ được xét 1 lần.

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Tính tổng ước của số nguyên

Phương pháp tối ưu 2:

```
107  ll CountDivisor3(ll so)
108  {
109      ll res = 0;
110      for (ll i = 1; i*i <= so; i++)
111          if (so % i == 0)
112              res += 2;
113      if (CheckPrefectSquare(so)) res--;
114      return res;
115  }
```

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Tính tổng giá trị các ước của số nguyên

```
125  ll SumDivisor2(ll so)
126  {
127      ll res = 0;
128      for (ll i = 1; i*i <= so; i++)
129          if (so % i == 0)
130              if (i != n/i) res += i + n/i;
131              else res += i;
132      return res;
133  }
```

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Ước chung, Ước chung lớn nhất

Số tự nhiên n được gọi là ước chung của hai số a và b nếu n vừa là ước của a vừa là ước của b.

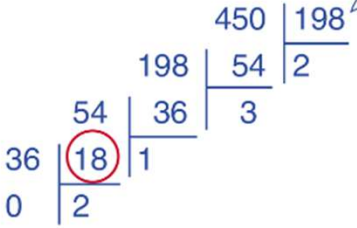
Số lớn nhất trong các ước chung của a và b được gọi là ước chung lớn nhất của a và b.

Ước chung của hai số là ước của ước chung lớn nhất của chúng.

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Thuật toán Euclid tìm ước chung lớn nhất của hai số

Ta có thể tìm ƯCLN(450, 198) theo các bước sau:



Bước 1. Chia số lớn cho số nhỏ

450 : 198 = 2 (dư 54) (1)

Bước 2. • Phép chia (1) còn dư nên lấy số chia đem chia cho số dư

198 : 54 = 3 (dư 36) (2)

• Phép chia (2) vẫn còn dư nên tiếp tục lấy số chia đem chia cho số dư

54 : 36 = 1 (dư 18) (3)

• Phép chia (3) vẫn còn dư nên tiếp tục lấy số chia đem chia cho số dư

36 : 18 = 2 (dư 0) (4)

• Phép chia (4) có số dư bằng 0, ta dừng lại

Bước 3. Số chia cuối cùng là ƯCLN phải tìm

ƯCLN(450, 198) = 18.

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Thuật toán Euclid tìm ước chung lớn nhất của hai số

Để tìm ước chung lớn nhất bằng thuật toán Ơ-clit, ta làm như sau:

Bước 1. Chia số lớn cho số nhỏ

Bước 2. Nếu phép chia còn dư thì ta lấy số chia đem chia cho số dư;

Ta cứ làm như vậy cho đến khi nhận được số dư bằng 0 thì dừng lại

Bước 3. Số chia cuối cùng là ước chung lớn nhất phải tìm.

Nhận xét: Người ta thường dùng thuật toán Ơ-clit để tìm ƯCLN của cặp số lớn.

Chẳng hạn, để tìm ƯCLN(336, 480), ta thực hiện các phép chia:

		480	336
	336	144	1
144	48	2	
0	3		

Vậy ƯCLN(336, 480) = 48.

```
95  ll GCD (ll s1, ll s2)
96  {
97      while (s2 > 0)
98      {
99          ll d = s1 % s2;
100         s1 = s2;
101         s2 = d;
102     }
103     return s1;
104 }
```

Lưu ý:
Nếu a, b <= 0 thì chỉ cần lấy giá trị tuyệt đối trước khi xử lý.

1. CÁC KHÁI NIỆM SỐ HỌC VÀ ỨNG DỤNG

Tìm ước chung lớn nhất của một dãy số

GCD(a1, a2, ..., an) = GCD(a1, GCD(a2, ..., GCD(an-1, an)...))

Ví dụ: tìm ước chung lớn nhất của [4, 12, 6]

GCD(4, 12) = 4

GCD(4, 6) = 2

GCD(4, 12, 6) = 2

1. CÁC KHÁI NIỆM SỐ HỌC

Hai số nguyên tố cùng nhau

Hai số nguyên tố cùng nhau là hai số có ước chung lớn nhất bằng 1.

Phân số tối giản là phân số có tử và mẫu là hai số nguyên tố cùng nhau.

Bội chung nhỏ nhất của 2 số

BCNN(a, b) = (a * b) / GCD(a, b)

Lưu ý: trong cách tính bội chung nhỏ nhất có phép nhân nên lưu ý trường hợp tràn miền giá trị trong quá trình tính nên ta cần thực hiện như sau để hạn chế tràn giá trị.

BCNN(a, b) = a * (b / GCD(a, b))

Giáo viên: Hoàng Đăng Quang - 0903781593

1. CÁC KHÁI NIỆM SỐ HỌC

Tổng Gauss

Dùng để tính tổng của một dãy số tự nhiên liên tiếp

$$\text{Tổng Gauss} = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Tính chất:

Dãy Số Cấp Số Cộng: Dãy số từ 1 đến n là một dãy số cấp số cộng với số hạng đầu là 1 và công sai là 1.

Tính Chất Đối Xứng: Tổng Gauss có tính chất đối xứng, nghĩa là nếu bạn chia đôi dãy số, tổng hai phần bằng nhau.

1. CÁC KHÁI NIỆM SỐ HỌC

Cấp số cộng

Là một dãy số mà sự khác biệt giữa hai số liên tiếp luôn không đổi. Số chênh lệch cố định này được gọi là công sai (d). Các số trong dãy cấp số cộng được xác định bởi công thức tổng quát của cấp số cộng.

Công thức tính số hạng thứ n trong dãy cấp số cộng	Công thức tính tổng của n số hạng đầu tiên cấp số cộng
$a_n = a_1 + (n-1) \cdot d$ • Trong đó: ○ a_n: Số hạng thứ n. ○ a_1: Số hạng đầu tiên. ○ n: Số thứ tự của số hạng. ○ d: Công sai.	$S_n = \frac{n}{2} \cdot (a_1 + a_n)$ • Trong đó: ○ S_n: Tổng của n số hạng đầu tiên.

1. CÁC KHÁI NIỆM SỐ HỌC

Cấp số nhân

là một dãy số trong đó mỗi số hạng (sau số hạng đầu tiên) đều là tích của số hạng liền trước đó với một hằng số không đổi, được gọi là công bội (common ratio).

Công thức tính số hạng thứ n trong dãy cấp số nhân

$$a_n = a_1 \cdot r^{(n-1)}$$

- Trong đó:
  - a_n: Số hạng thứ n.
  - a_1: Số hạng đầu tiên.
  - r: Công bội.
  - n: Số thứ tự của số hạng.

Công thức tính tổng của n số hạng đầu tiên cấp số nhân

- Công thức khi $r \neq 1$:

$$S_n = a_1 \cdot \frac{1-r^n}{1-r}$$

- Công thức khi $r = 1$ (đơn giản hơn vì dãy số không thay đổi):

$$S_n = a_1 \cdot n$$

- Trong đó:
  - S_n: Tổng của n số hạng đầu tiên.

Giáo viên: Hoàng Đăng Quang - 0903781593