SỐ CHÍNH PHƯƠNG

Thuật toán tìm trong đoạn, kiểm tra số lớn bằng modulo, và dùng thư viện Big Integer trong C++

1. Khái niệm số chính phương

Một số nguyên không âm x được gọi là số chính phương nếu tồn tại một số nguyên k sao cho:

x = k^2

Ví dụ: 0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, ...

Nói cách khác, số chính phương là bình phương của một số nguyên.

2. Thuật toán tìm các số chính phương trong đoạn [L, R]

2.1. Ý tưởng

Ta cần tìm các số chính phương nằm trong đoạn từ L đến R. Không nên duyệt từng số trong đoạn rồi kiểm tra, vì đoạn có thể rất dài.

Thay vào đó, ta tìm các số nguyên k sao cho:

L <= k^2 <= R

Suy ra:

sqrt(L) <= k <= sqrt(R)

Vì vậy:

- Số k nhỏ nhất là a = ceil(sqrt(L)).
- Số k lớn nhất là b = floor(sqrt(R)).
- Các số chính phương cần in là a^2, (a + 1)^2, ..., b^2.

2.2. Ví dụ

Cho đoạn L = 10, R = 50.

- sqrt(10) ~ 3.16, nên a = 4.
- sqrt(50) ~ 7.07, nên b = 7.

Các số chính phương là:

4^2, 5^2, 6^2, 7^2

Kết quả: 16 25 36 49.

2.3. Thuật toán dạng giả mã

Nhập L, R
a = ceil(sqrt(L))
b = floor(sqrt(R))
Với i chạy từ a đến b, in i × i.

2.4. Code C++ cơ bản

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    long long L, R;
    cin >> L >> R;

    long long a = ceil(sqrt(L));
    long long b = floor(sqrt(R));

    for (long long i = a; i <= b; i++) {
        cout << i * i << " ";
    }

    return 0;
}
```

Code C++ tìm số chính phương trong đoạn [L, R].

2.5. Code C++ tránh sai số căn bậc hai

Với số lớn, hàm sqrt() có thể có sai số nhỏ do dùng số thực. Có thể hiệu chỉnh lại bằng các vòng lặp kiểm tra.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    long long L, R;
    cin >> L >> R;

    long long a = sqrt(L);
    while (a * a < L) a++;
    while (a > 0 && (a - 1) * (a - 1) >= L) a--;

    long long b = sqrt(R);
    while ((b + 1) * (b + 1) <= R) b++;
    while (b * b > R) b--;

    for (long long i = a; i <= b; i++) {
        cout << i * i << " ";
    }

    return 0;
}
```

Code C++ hiệu chỉnh biên để tránh sai số khi dùng sqrt().

2.6. Độ phức tạp

Thuật toán chỉ duyệt theo số lượng căn nguyên trong đoạn, tức khoảng:

O(sqrt(R) - sqrt(L) + 1)

Cách này tốt hơn nhiều so với duyệt toàn bộ từ L đến R khi đoạn rất lớn.

3. Kiểm tra số chính phương rất lớn bằng modulo

Nếu số cần kiểm tra rất lớn, ví dụ có hàng trăm hoặc hàng nghìn chữ số, kiểu long long không lưu được. Một hướng xử lý nhanh là kiểm tra theo modulo.

3.1. Ý tưởng

Một số chính phương khi chia cho một số m chỉ có thể cho một số dư nhất định. Ta gọi các số dư đó là tập số dư bình phương.

Ví dụ với modulo 10, số chính phương chỉ có thể tận cùng bằng:

0, 1, 4, 5, 6, 9

Vì vậy, nếu một số tận cùng bằng 2, 3, 7 hoặc 8 thì chắc chắn không phải số chính phương.

3.2. Thuật toán tổng quát

- Lưu số rất lớn N dưới dạng chuỗi.
- Chọn một vài modulo, ví dụ: 64, 63, 65, 11, 13, 17, ...
- Với mỗi modulo m, tính r = N mod m.
- Tạo trước tập các số dư có dạng x^2 mod m.
- Nếu r không thuộc tập đó thì kết luận NO.
- Nếu vượt qua nhiều modulo thì kết luận khả năng cao là YES.

Lưu ý: Cách này là kiểm tra nhanh theo xác suất. Nếu kết quả là NO thì chắc chắn không phải số chính phương. Nếu kết quả là YES thì chỉ là khả năng cao, chưa phải chứng minh tuyệt đối.

3.3. Code C++ kiểm tra số lớn dạng chuỗi

```cpp
#include <bits/stdc++.h>
using namespace std;

long long modString(const string &s, long long mod) {
    long long res = 0;
    for (char c : s) {
        res = (res * 10 + (c - '0')) % mod;
    }
    return res;
}

bool checkByMod(const string &s, int mod) {
    vector<bool> squareResidue(mod, false);

    for (int i = 0; i < mod; i++) {
        squareResidue[(1LL * i * i) % mod] = true;
    }

    int r = modString(s, mod);
    return squareResidue[r];
}

bool isProbablySquare(const string &s) {
    vector<int> mods = {64, 63, 65, 11, 13, 17, 19, 23, 29, 31};

    for (int m : mods) {
        if (!checkByMod(s, m)) {
            return false;
        }
    }
    return true;
}

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

    if (isProbablySquare(n)) cout << "YES";
    else cout << "NO";

    return 0;
}
```

Code C++ kiểm tra số chính phương lớn bằng nhiều phép chia modulo.

4. Thư viện Big Integer trong C++

Big Integer là thư viện dùng để xử lý số nguyên rất lớn, vượt quá giới hạn của các kiểu dữ liệu thông thường như int, long long.

Trong C++, có thể dùng thư viện Boost Multiprecision:

#include <boost/multiprecision/cpp_int.hpp>

Kiểu dữ liệu chính là cpp_int. Kiểu này có thể lưu các số nguyên có rất nhiều chữ số, miễn là bộ nhớ còn đủ.

4.1. Ví dụ dùng cpp_int

```cpp
#include <bits/stdc++.h>
#include <boost/multiprecision/cpp_int.hpp>

using namespace std;
using namespace boost::multiprecision;

int main() {
    cpp_int a, b;
    cin >> a >> b;

    cout << a + b << '\n';
    cout << a * b << '\n';

    return 0;
}
```

Code C++ nhập hai số rất lớn rồi tính tổng và tích bằng cpp_int.

4.2. Một số phép toán với cpp_int

Với cpp_int, ta có thể sử dụng các phép toán quen thuộc:

- a + b: cộng.
- a - b: trừ.
- a * b: nhân.
- a / b: chia nguyên.
- a % b: chia lấy dư.
- a == b, a < b, a > b: so sánh.

5. Kiểm tra số chính phương chính xác bằng cpp_int

Với cpp_int, ta không dùng trực tiếp sqrt() như long long. Ta có thể tìm căn nguyên bằng tìm kiếm nhị phân.

Mục tiêu là tìm số nguyên x sao cho:

x^2 = n

Nếu tìm được x thì n là số chính phương, ngược lại không phải.

5.1. Ý tưởng tìm kiếm nhị phân

- Đặt left = 0, right = n.
- Lấy mid = (left + right) / 2.
- Nếu mid * mid = n thì trả về YES.
- Nếu mid * mid < n thì tìm tiếp bên phải.
- Nếu mid * mid > n thì tìm tiếp bên trái.

5.2. Code C++ kiểm tra chính phương với cpp_int

```cpp
#include <bits/stdc++.h>
#include <boost/multiprecision/cpp_int.hpp>

using namespace std;
using namespace boost::multiprecision;

bool isPerfectSquare(cpp_int n) {
    if (n < 0) return false;

    cpp_int left = 0, right = n;

    while (left <= right) {
        cpp_int mid = (left + right) / 2;
        cpp_int sq = mid * mid;

        if (sq == n) return true;

        if (sq < n) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return false;
}

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

    if (isPerfectSquare(n)) cout << "YES";
    else cout << "NO";

    return 0;
}
```

Code C++ kiểm tra chính phương chính xác bằng cpp_int và tìm kiếm nhị phân.

6. Khi nào dùng cách nào?

Trường hợp	Cách nên dùng
N <= 10^18	Dùng long long và sqrt(), có hiệu chỉnh sai số.

Cần liệt kê số chính phương trong đoạn [L, R]	Duyệt i từ ceil(sqrt(L)) đến floor(sqrt(R)), in i^2.
N rất lớn, chỉ cần lọc nhanh	Dùng kiểm tra modulo theo chuỗi.
N rất lớn, cần kết quả chính xác	Dùng cpp_int và tìm kiếm nhị phân căn nguyên.

Ghi nhớ: nếu đề bài chỉ yêu cầu kiểm tra số trong phạm vi long long, không cần dùng cpp_int. Nếu đề bài cho số có rất nhiều chữ số, cần dùng chuỗi, modulo hoặc cpp_int tùy yêu cầu chính xác.