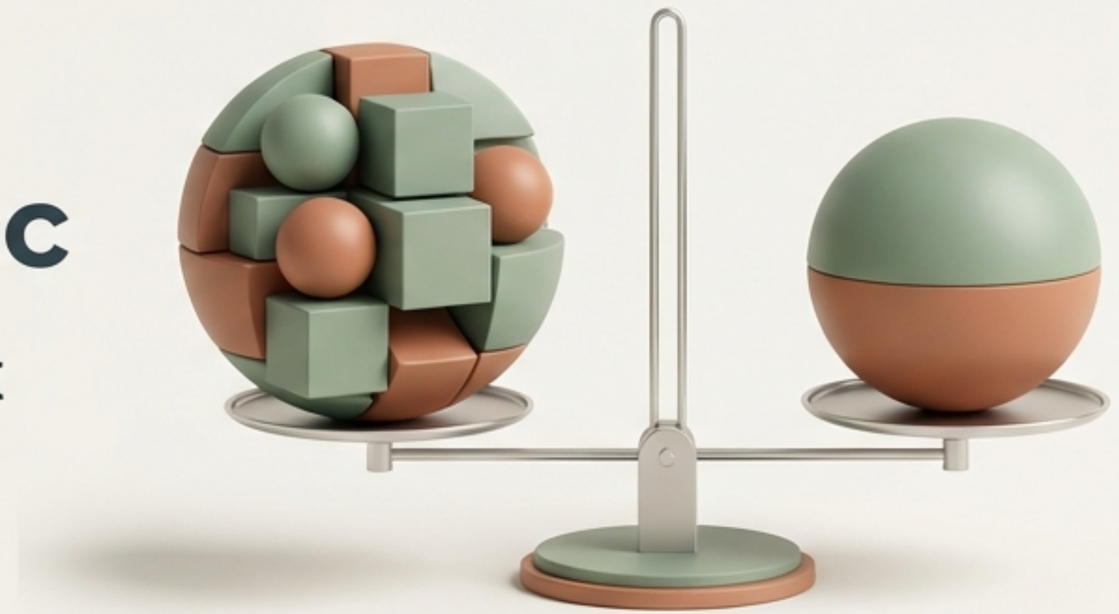
Vẻ Đẹp Của Sự Đủ Đầy Trong Toán Học

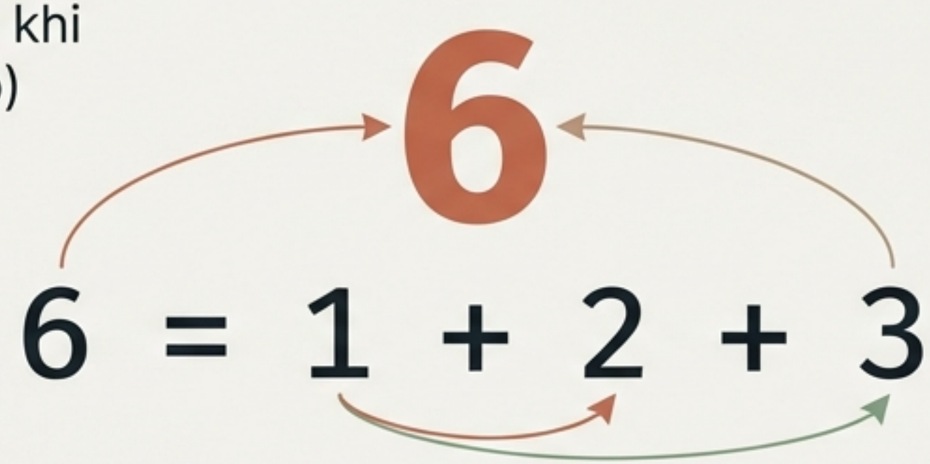
Khi những con số ghép lại thành một bức tranh hoàn hảo.

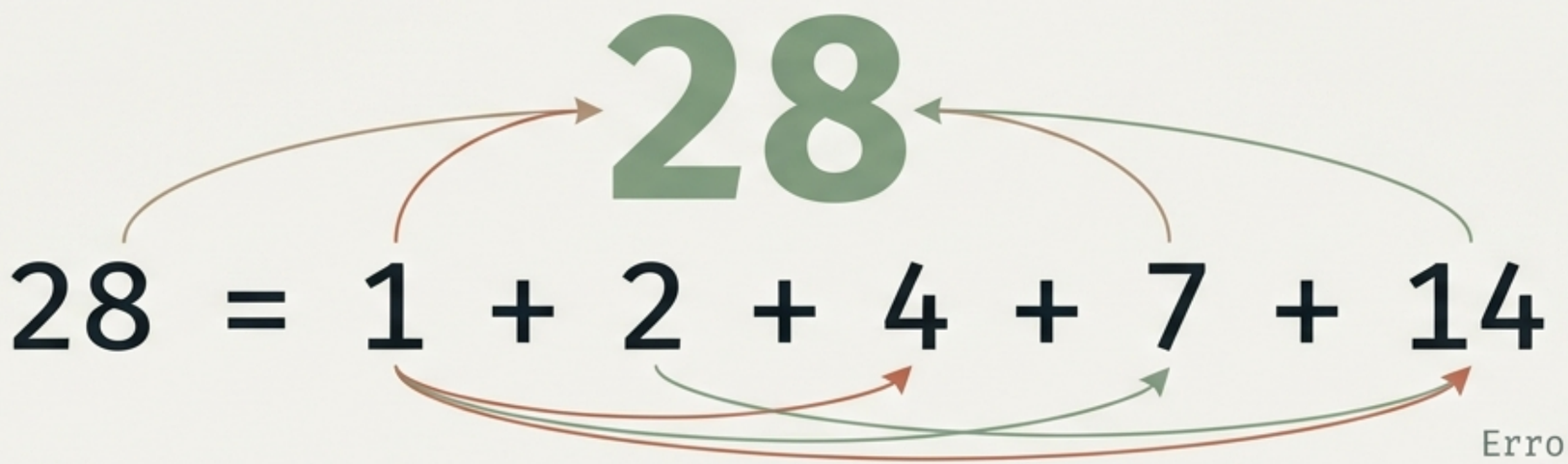


Error404-Labs.Info.Vn

Tổng Các Ước Số Tạo Nên Chính Nó

Một số được gọi là hoàn hảo khi tổng các ước số (nhỏ hơn nó) cộng lại bằng đúng giá trị ban đầu.





Một Giải Pháp C Gọn Gàng Và Tối Ưu

```c
int kiemTraSoHoanHao(int n) {
    if(n <= 1) return 0;
    int tong = 1;
    for(int i = 2; i <= n / i ; i++) {
        if(n % i == 0) {
            tong += i;
            int doi = n / i;
            if(i != doi) {
                tong += doi;
            }
        }
    }
    return tong == n;
}
```

Gác Cổng Nghiêm Ngặt Bảo Vệ Logic Cốt Lõi

Số 0, số âm, và số 1 không bao giờ là số hoàn hảo. Xử lý sớm để tránh lãng phí tài nguyên tính toán.

```
int kiemTraSoHoanHao(int n) {
    if(n <= 1) return 0;
    int tong = 1;
    for(int i = 2; i <= n / i ; i++) {
        if(n % i == 0) {
            tong += i;
            int doi = n / i;
            if(i != doi) {
                tong += doi;
            }
        }
    }
    return tong == n;
}
```

Khởi Đầu Bằng Một Chân Lý Luôn Đúng

Vì 1 luôn là ước của mọi số lớn hơn 1, chúng ta khởi tạo tổng bằng 1. Không cần lãng phí thời gian duyệt từ 1.

```
int kiemTraSoHoanHao(int n) {
    if(n <= 1) return 0;
    int tong = 1;
    for(int i = 2; i <= n / i ; i++) {
        if(n % i == 0) {
            tong += i;
            int doi = n / i;
            if(i != doi) {
                tong += doi;
            }
        }
    }
    return tong == n;
}
```

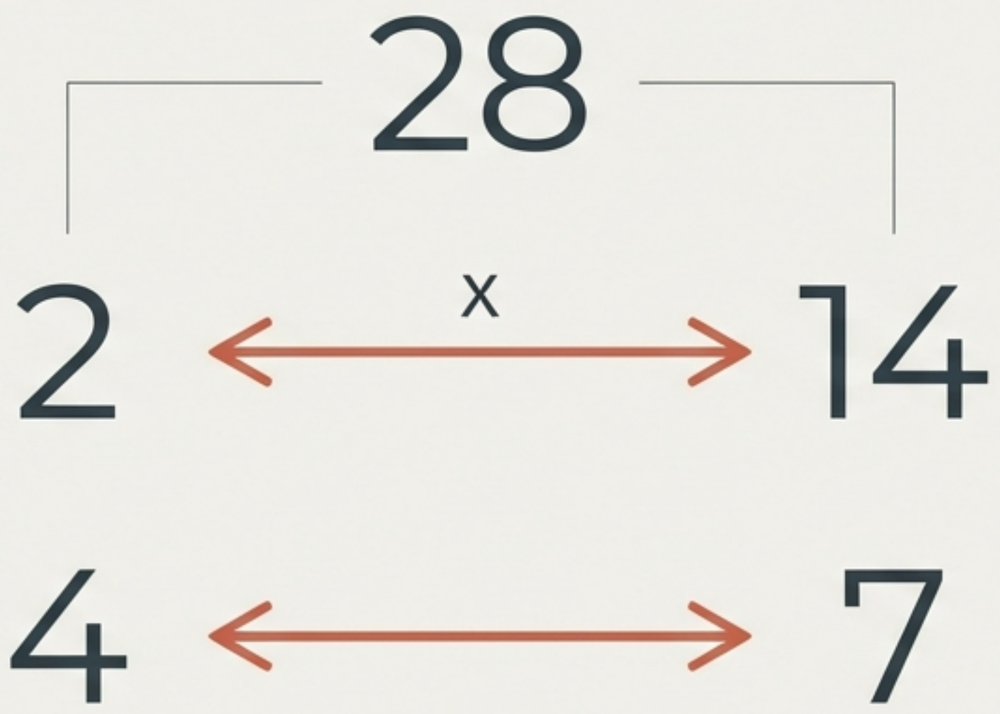
Cú Nhảy Vọt Về Hiệu Năng Xử Lý

Viết **i <= n / i** tương đương với **i * i <= n**. Chúng ta chỉ cần duyệt vòng lặp đến căn bậc hai của n.

```
int kiemTraSoHoanHao(int n) {
    if(n <= 1) return 0;
    int tong = 1;
    for(int i = 2; i <= n / i; i++)
    if(n % i == 0) {
        tong += i;
        int doi = n / i;
        if(i != doi) {
            tong += doi;
        }
    }
    return tong == n;
}
```

Mọi Ước Số Luôn Đi Theo Cặp

Nếu tìm thấy một ước số, người anh em của nó cũng tự động lộ diện. Không cần tiếp tục tìm kiếm một cách thừa thãi.



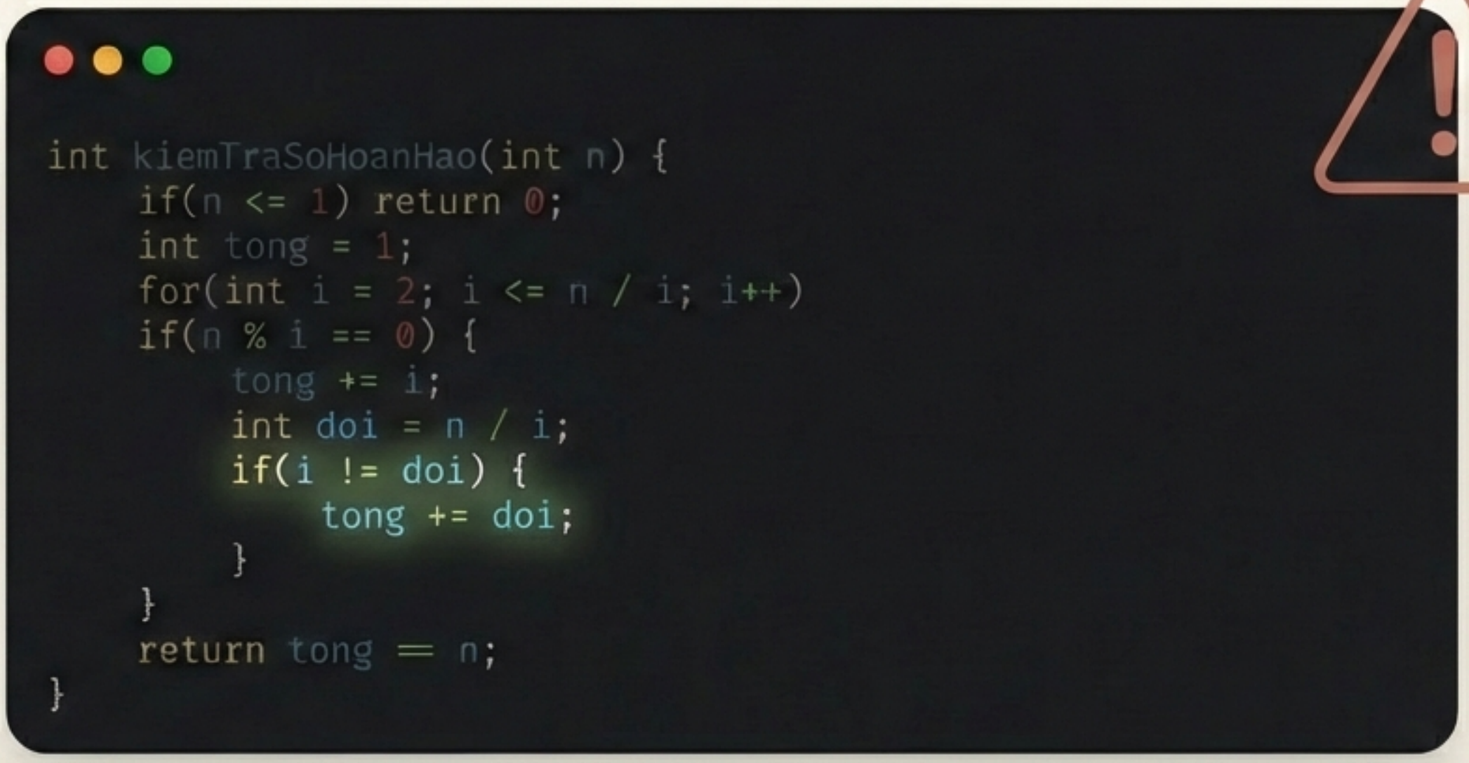
Bắt Trọn Cặp Đối Xứng Qua Một Phép Chia

Biến doi chính là ước đối xứng của i. Khi i là một ước, n / i tự động là nửa kia của cặp đôi.

```
int kiemTraSoHoanHao(int n) {
    if(n <= 1) return 0;
    int tong = 1;
    for(int i = 2; i <= n / i; i++)
    if(n % i == 0) {
        tong += i;
        int doi = n / i;
        if(i != doi) {
            tong += doi;
        }
    }
    return tong == n;
}
```

Né Tránh Cạm Bẫy Của Số Chính Phương

Nếu n = 36, ước số là 6 × 6. Lúc này i = 6 và doi = 6. Kiểm tra điều kiện này giúp thuật toán không bao giờ cộng một con số hai lần.



```
int kiemTraSoHoanHao(int n) {
    if(n <= 1) return 0;
    int tong = 1;
    for(int i = 2; i <= n / i; i++)
    if(n % i == 0) {
        tong += i;
        int doi = n / i;
        if(i != doi) {
            tong += doi;
        }
    }
    return tong == n;
}
```

Lời Khẳng Định Cuối Cùng

Chỉ cần so sánh tổng các ước với chính nó. Rõ ràng, sạch sẽ và cực kỳ dễ đọc.

```
int kiemTraSoHoanHao(int n) {
    if(n <= 1) return 0;
    int tong = 1;
    for(int i = 2; i <= n / i; i++)
    if(n % i == 0) {
        tong += i;
        int doi = n / i;
        if(i != doi) {
            tong += doi;
        }
    }
    return tong == n;
}
```

4 Cột Mốc Của Một Thuật Toán Tinh Tế

Bám sát định nghĩa toán học

Không cộng trùng lặp

Không cộng chính nó vào tổng

Tối ưu độ phức tạp xuống $O(\sqrt{n})$

Tư Duy Tối Ưu Mới Là Phần Đáng Giá Nhất

Giải quyết một bài toán nhỏ là cách tốt nhất để rèn luyện một tư duy thuật toán lớn.