1. Mô hình bài toán

- Với một hệ số k, dãy phát sinh là

$$a_i = k + \sum_{j=1}^{i-1} j \;=\; k + \frac{(i-1)\,i}{2}.$$

- Ta chỉ lấy các a_i sao cho $a_i \le n$; giả sử số phần tử thỏa là m.
- Giá trị của dãy (tới phần tử m) là

$$S = \sum_{i=1}^{m} a_i = \sum_{i=1}^{m} \left(k + \tfrac{(i-1)i}{2}\right) = m\,k + \frac{1}{2}\sum_{i=1}^{m}(i-1)i = m\,k + \frac{m\,(m^2-1)}{6}.$$

- Nhiệm vụ: tìm k sao cho S lớn nhất.

2. Ý tưởng tối ưu

Thay vì thử mọi k, ta đảo sang thử mọi khả năng độ dài m (số phần tử) vì:

1. Với mỗi m, điều kiện $a_m \le n$ cho ta

$$k + \frac{(m-1)m}{2} \le n \quad \Longrightarrow \quad k \le n - \frac{(m-1)m}{2}.$$

   Để có giá trị S lớn nhất, ta chọn k ở mức tối đa thỏa:

$$k = n - \frac{(m-1)m}{2}.$$

2. Khi m quá lớn, $\frac{(m-1)m}{2} > n$ thì không còn phần tử nào, nên ta chỉ cần lặp m từ 1 đến khi $\frac{(m-1)m}{2} > n$. Thực chất độ lớn mốc này là khoảng $O(\sqrt{n})$.

Như vậy, ta chỉ lặp khoảng $\sqrt{n}$ lần, mỗi lần tính vài công thức đơn giản — tổng độ phức tạp $O(\sqrt{n})$, dư sức chạy với n lên đến 10^{12}.

Ý tưởng của đoạn code trên rất thẳng–tay và dễ hình dung:

1. **Duyệt mọi giá trị khởi đầu k**
   Ta biết k phải nằm trong khoảng từ 1 đến n (nếu $k > n$ thì ngay phần tử đầu tiên đã vượt, không có gì để xét). Vì vậy ta có một vòng lặp for duyệt k từ 1 đến n.

2. **Xây dãy với hiệu gia tăng dần**
   Với mỗi k, ta sinh dãy như sau:

   - Phần tử đầu: $a_1 = k$.
   - Phần tử thứ hai: $a_2 = k + 1$.
   - Phần tử thứ ba: $a_3 = k + 1 + 2$.
   - Phần tử thứ tư: $a_4 = k + 1 + 2 + 3$.

     ...

     Như vậy hiệu giữa hai phần tử liên tiếp là dãy 1, 2, 3, ... được tăng thêm từng bước.

3. **Cộng cho đến khi vượt n**
   Khi sinh dãy, ta giữ biến `val` cho giá trị hiện tại (bắt đầu bằng `k`) và biến `d` cho "bước nhảy" (bắt đầu bằng 1).

   - Trong mỗi vòng lặp `while (val <= n)`, ta cộng `val` vào tổng `sum`.
   - Rồi tăng `val += d` và `d++`.
   - Khi nào `val` trở nên $> n$, ta dừng sinh và ghi nhận `sum` vừa thu được.

4. **So sánh và lưu kết quả tốt nhất**
   Mỗi khi hoàn thành chuỗi cho một k, ta so sánh tổng `sum` với `max_sum` đã lưu. Nếu `sum` lớn hơn, ta cập nhật `max_sum = sum` và ghi nhớ `res_k = k`.

5. **Kết quả**
   Sau khi thử qua tất cả các k, `res_k` chính là giá trị khởi đầu cho phép lấy được tổng dãy "lớn nhất trước khi vượt n".

Ta có công thức cần chứng minh:

$$\frac{1}{2}\sum_{i=1}^{m}(i-1)\,i \;=\; \frac{m\,(m^2-1)}{6}.$$

Bước 1. Viết lại tổng:

$$\sum_{i=1}^{m}(i-1)\,i = \sum_{i=1}^{m}\left(i^2-i\right) = \sum_{i=1}^{m} i^2 \;-\; \sum_{i=1}^{m} i.$$

Bước 2. Áp dụng công thức:

- $\displaystyle\sum_{i=1}^{m} i = \frac{m(m+1)}{2}.$
- $\displaystyle\sum_{i=1}^{m} i^2 = \frac{m(m+1)(2m+1)}{6}.$

cpp

Copy Edit

```cpp
ll m_max = (ll)((1 + sqrt(1 + 8.0*(n-1))) / 2) + 2;
```

- Tính xấp xỉ giới hạn trên cho m sao cho $\frac{(m-1)m}{2} \leq n$. Công thức này xuất phát từ bất phương trình $m^2 - m - 2(n-1) \leq 0$.

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

#define npk ios::sync_with_stdio(false); cin.tie(nullptr); cout.tie(nullptr);

typedef long long ll;

ll n;

int main() {
    npk

    ll res_k,max_sum,k,sum,val,d,best_k,best_sum,m_max,S,m,t;
    cin >> n;

    if (n<=ll(1e6)) {
        res_k = 1, max_sum = 0;
        for (k = 1; k <= n; ++k) {
            sum = 0, val = k, d = 1;

            while (val <= n) {
                sum += val;
                val += d;
                ++d;
            }

            if (sum > max_sum) {
                max_sum = sum;
                res_k = k;
            }
        }

        cout << res_k;
    }
    else {
        best_k = 1;
        best_sum = 0;
        // Giới hạn m tới khi t = (m-1)*m/2 > n
        m_max = (ll)((1 + sqrt(1 + 8.0*n)) / 2) + 2;
        for(m = 1; m <= m_max; m++){
            t = (m-1) * m / 2;
            if (t > n) break;

            // k lớn nhất để có đủ m phần tử
            k = n - t;

            // Tính tổng S = m*k + m*(m^2-1)/6
            // Tất cả vừa vặn trong 64-bit với n<=1e12
            S = m*k + (m * (m*m - 1) / 6);

            // Cập nhật: ưu tiên sum lớn, nếu bằng thì k nhỏ
            if (S > best_sum || (S == best_sum && k < best_k)) {
                best_sum = S;
                best_k = k;
            }
        }

        cout << best_k;
    }
}

```