**Tuần 3: DIVIDE AND CONQUER**

# I. TÌM HIỂU VỀ CHIA ĐỂ TRỊ (DIVIDE AND CONQUER).

## 1. Định nghĩa:

- Chia để trị là một phương pháp quan trọng trong việc thiết kế các giải thuật. Ý tưởng của phương pháp này khá đơn giản và rất dễ hiểu:

Khi cần giải quyết một bài toán, ta sẽ tiến hành chia bài toán đó thành các bài toán con nhỏ hơn. Tiếp tục chia cho đến khi các bài toán nhỏ này không thể chia thêm nữa, khi đó ta sẽ giải quyết các bài toán nhỏ nhất này và cuối cùng kết hợp giải pháp của tất cả các bài toán nhỏ để tìm ra giải pháp của bài toán ban đầu.

Devide

Conquer

Combine

## 2. Nguyên lý:

Chia để trị được chia làm 3 bước rõ ràng:

- Bước 1: Chia (Divide/Break).

+ Ta chia bài toán ban đầu thành các bài toán con nhỏ hơn. Mỗi bài toán con nên là một phần của bài toán ban đầu. Ở bước này, ta sử dụng phương pháp đệ qui để chia nhỏ các bài toán cho đến khi không thể chia thêm nữa. Khi đó, các bài toán con được gọi là "atomic – nguyên tử" hoặc là Sub-Problem, nhưng chúng vẫn biểu diễn một phần nào đó của bài toán ban đầu. Hay nói cách khác những bài toán nhỏ này giống với bài toán ban đầu.

- Bước 2: Trị (Conquer/Solve).

+ Ta tiến hành giải quyết các bài toán con đã được chia.

- Bước 3: Kết hợp bài toán (Combine/Merge).

+ Từ kết quả của những bài toán con, ta kết hợp chúng một cách đệ quy từ bài toán đơn giản đến phức tạp, cuối cùng tìm ra được đáp án cho bài toán ban đầu.

## 3. Ví dụ:

- Bài toán Tháp Hà Nội.

- Sắp xếp trộn.

- Sắp xếp nhanh.

**II. BÀI TẬP MINH HỌA**

1. Bài toán Lũy Thừa

Graphical user interface, text, application, email

Description automatically generated

* Đối với các mũ nhỏ, ta có thể tính toán bình thường, nhưng trong trường hợp số lớn như này, ta cần tìm cách khác.
* Phương pháp: **Nhân Ấn Độ** với tư tưởng chia để trị.
  + Phép nhân với modulo

Calendar

Description automatically generated with medium confidence

Dễ thấy, ban đầu từ a x b, ta chia nhỏ và tính thông qua , và cứ tiếp tục chia nhỏ như thế nữa. Sau đó, kết quả con gộp lại để đưa ra được kết quả cuối cùng. Để tránh tràn số, ta cần kết hợp chia dư để có thể tính toán với các số lớn.

Code mẫu:

ll nhan(ll a, ll b)

{

    if(b == 0) return 0;

    ll t = nhan(a, b / 2) % big;

    if(b % 2 == 1)

    {

        return ((t + t) % big + a % big) % big;

    }

    else return (t + t) % big;

}

* Phép lũy thừa modulo

Text

Description automatically generated

Tương tự như phép nhân với modulo, ta chia nhỏ b thành rồi lấy lũy thừa, rồi lại tiếp tục chia nhỏ như thế, đến khi đạt được giá trị bằng 1 thì dừng lại. Các kết quả con, ta nhân gộp lại bằng chính thuật toán Nhân với Modulo, ta được kết quả cuối cùng. Lưu ý, chúng ta vẫn cần chia dư để tránh tràn số khi sử dụng với các số nguyên lớn.

Code mẫu:

ll mu(ll a, ll b)

{

    if(b == 0) return 1;

    ll t = mu(a, b / 2) % big;

    if(b % 2 == 1)

    {

        return nhan(nhan(t, t), a) % big;

    }

    else return nhan(t, t) % big;

}

* Độ phức tạp: Bài toán ban đầu từ O(N), sau khi sử dụng Nhân ấn độ, đã được rút gọn, trở thành **O(logN)**
* Code bài giải:

ll nhan(ll a, ll b)

{

    if(b == 0) return 0;

    ll t = nhan(a, b / 2) % big;

    if(b % 2 == 1)

    {

return ((t + t) % big + a % big) % big;

    }

    else return (t + t) % big;

}

ll mu(ll a, ll b)

{

    if(b == 0) return 1;

    ll t = mu(a, b / 2) % big;

    if(b % 2 == 1)

    {

        return nhan(nhan(t, t), a) % big;

    }

    else return nhan(t, t) % big;

}

int main()

{

    ios\_base::sync\_with\_stdio(false);cin.tie(0);cout.tie(0);

    int t;

    cin >> t;

    while(t--)

    {

        ll a, b;

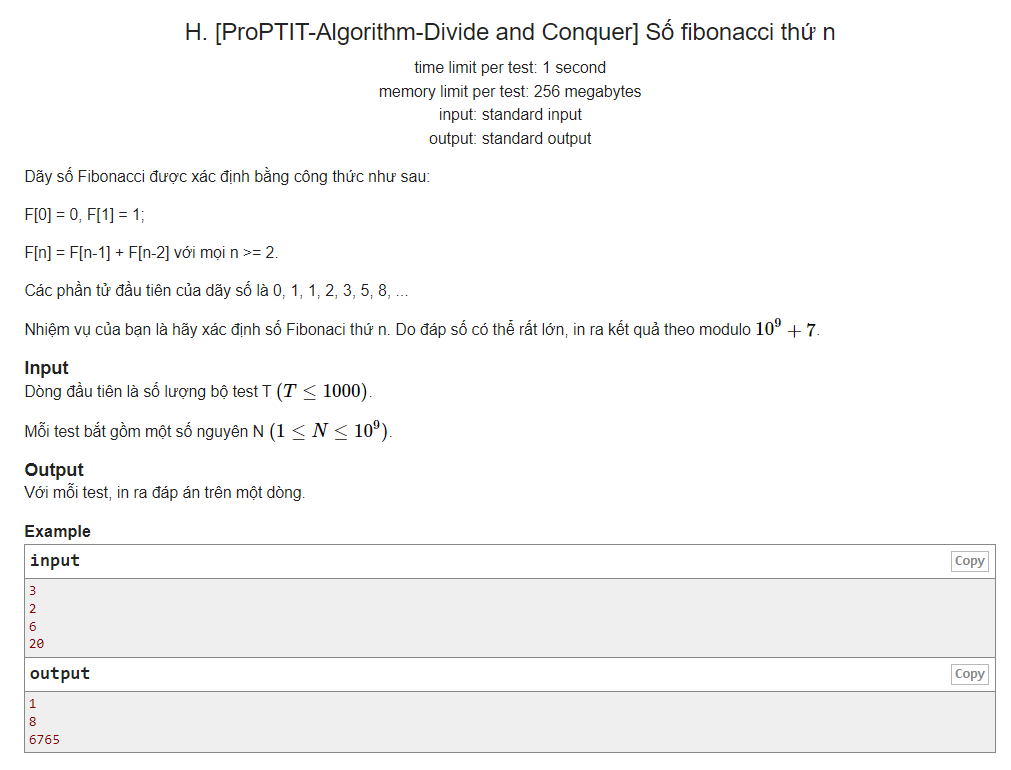
        cin >> a >> b;

        cout << mu(a, b) << endl;

    }

}

1. Bài toán Fibonacci thứ n.



Ở bài toán này, ta đã quen với việc sử dụng đệ quy và đệ quy có nhớ để làm bài. Cả 2 phương pháp này, đều phụ thuộc vào mảng đánh dấu, chính vì thế, để tìm số thứ 109 là điều rất khó khăn. Tuy nhiên, vẫn áp dụng tư tưởng chia để trị, ta hướng đến một phương pháp khác: Nhân Ma Trận.

Graphical user interface, application

Description automatically generated

*Giả sử ta có phép nhân ma trận sau.*

Để phép tính xảy ra thì a = b = c = 1 và d = 0.

Sau khi thực hiện tuần tự các phép biến đổi, ta chứng minh được công thức:

Diagram

Description automatically generated

Mà F(2) = F(1) = 1, F(0) = 0. Ta dễ dàng tính được F(n) với n biết trước, chỉ cần tính lũy thừa của ma trận.

Thừa kế kiến thức từ bài Lũy thừa Modulo, ta phát triển thêm thành Lũy thừa Modulo của Ma trận.

* Độ phức tạp: **O(logN)**

Code mẫu:

void nhan(ll f[2][2], ll a[2][2])

{

    ll x = (f[0][0] \* a[0][0] % big + f[0][1] \* a[1][0] % big) % big;

    ll y = (f[0][0] \* a[0][1] % big + f[0][1] \* a[1][1] % big) % big;

    ll z = (f[1][0] \* a[0][0] % big + f[1][1] \* a[1][0] % big) % big;

    ll t = (f[1][0] \* a[0][1] % big + f[1][1] \* a[1][1] % big) % big;

    f[0][0] = x;

    f[0][1] = y;

    f[1][0] = z;

    f[1][1] = t;

}

void mu(ll f[2][2], ll n)

{

    if(n <= 1) return;

    mu(f, n / 2);

    nhan(f, f);

    if(n % 2 == 1) nhan(f, a);

}

**Lưu ý:** đây là bài toán sử dụng ma trận 2x2, ngoài ra ta còn có thể mở rộng khi sử dụng ma trận lớn hơn bằng cách sử dụng vector hoặc map 2 chiều.