Chuyên đề
Đệ Quy và Quy Hoạch Động

TA. Nguyễn Hải Đăng

Định nghĩa

- **Đệ quy:** Một kỹ thuật gọi hàm định nghĩa chính nó.
    - Đệ quy thường được sử dụng để giải quyết các bài toán mang tính liệt kê hoặc duyệt tuần tự
- **Quy hoạch động:** Một phương pháp giải các bài toán có thể sử dụng đệ quy theo hướng *khử đệ quy* nhằm giảm bớt việc tính toán trùng lắp trong đệ quy.
    - Quy hoạch động thường được sử dụng để giải quyết các bài toán đếm số lượng, tìm cực trị.

Bài toán mẫu - Dãy Fibonacci

- Dãy số Fibonacci được định nghĩa như sau:
  - Fibo(0) = 0
  - Fibo(1) = 1
  - Fibo(n) = Fibo(n-1) + Fibo(n-2)
- Dãy Fibonacci:
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …
- Bài toán: cho số nguyên dương n, tìm số Fibonacci thứ n. Fibo(n) = ?
- Ví dụ:
  - Input: n = 10
  - Output: Fibo(10) = 55

Các hướng tiếp cận và phân tích bài toán

- Gồm 2 hướng tiếp cận chính:
  - Top-down: góc nhìn theo hướng đệ quy
  - Bottom-up: góc nhìn theo hướng quy hoạch động

Các hướng tiếp cận và phân tích bài toán

- Top-down:
    - Chia nhỏ trường hợp lớn ra thành các trường hợp con
    - Tìm lời giải cho các trường hợp con để ra được kết quả cho bài toán
- Đây là hướng tiếp cận tương đối phức tạp vì thiếu sự chắc chắn.
- Tuy nhiên, trong một số bài toán, đây là cách tiếp cận dễ nhất để nhìn ra hướng giải quyết.
- Mấu chốt: Rã bài toán lớn thành các bài toán con để tìm lời giải.
- *Lưu ý các trường hợp trùng nhau*.

Các hướng tiếp cận và phân tích bài toán

- Bottom-up:
  - Giải quyết bài toán với *trường hợp nhỏ nhất* (ví dụ dãy với 1 phần tử)
  - Dùng kết quả của trường hợp nhỏ để tìm lời giải cho các trường hợp lớn hơn (dãy với 2 phần tử, 3 phần tử, …)
- Đây là hướng tiếp cận trực quan và chắc chắn.
- <u>Mấu chốt:</u> tiếp cận và xây dựng lời giải cho trường hợp mới bằng cách sử dụng *kết quả của các trường hợp* ***đã được tính sẵn*** trước đó.

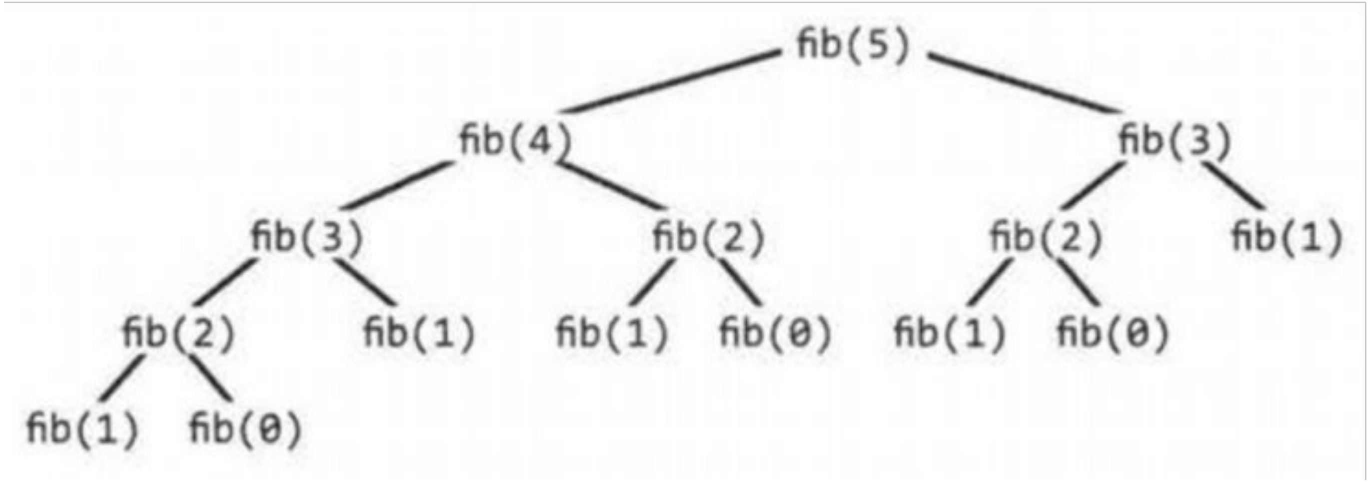
Bài toán Fibonacci với Đệ quy (góc nhìn top-down)

```
int fibo(int n) {
    if (n==0 || n==1) return n;
    return fibo(n-1) + fibo(n-2);
}
fibo_n = fibo(n);
```

- Độ phức tạp thuật toán:
  - Về thời gian: O(n) hay O(n^2) hay lớn hơn?
  - Về không gian: O(n) or O(log2(n))?

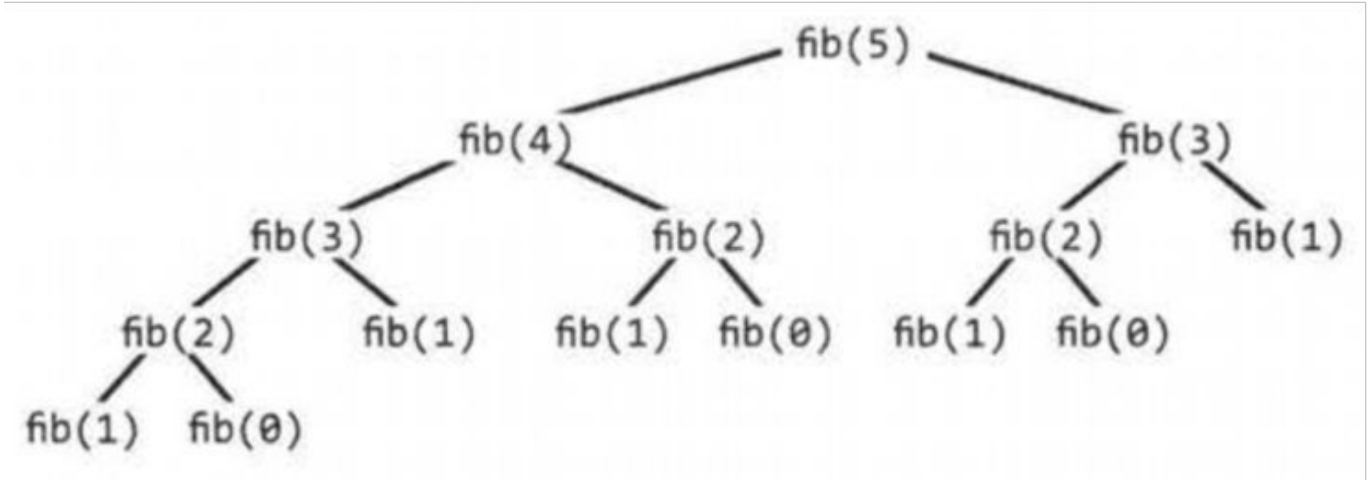
Bài toán Fibonacci với Đệ quy (góc nhìn top-down)

- Gợi ý: Nên sử dụng cây để biểu diễn và phân tích độ phức tạp của một thuật toán đệ quy.



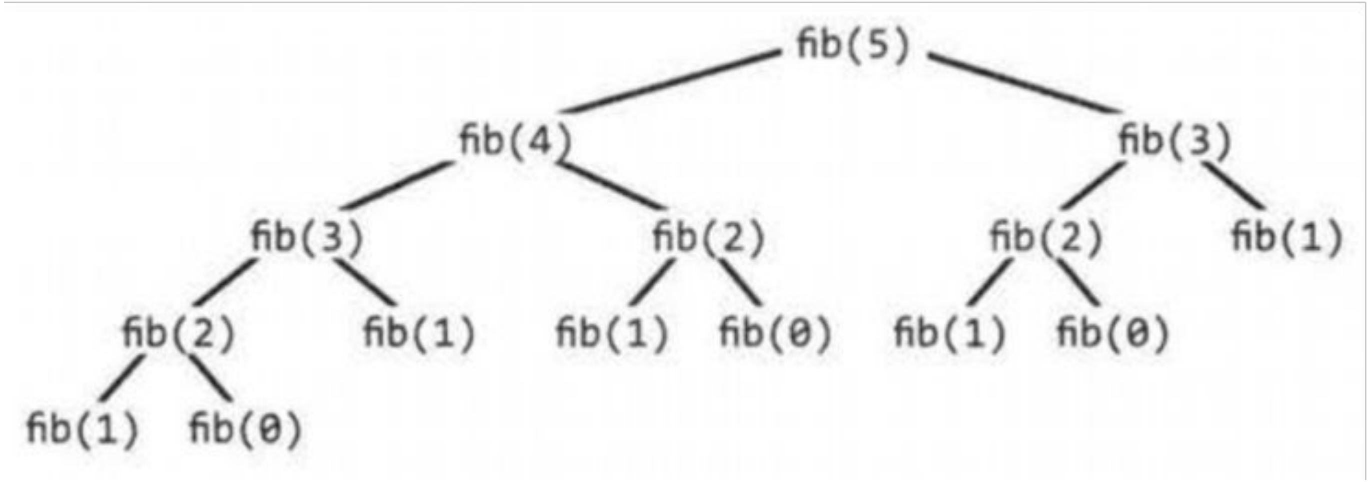
Bài toán Fibonacci với Đệ quy (góc nhìn top-down)

- **Tổng số lượng các đỉnh** trên cây chính là *độ phức tạp về mặt thời gian* của thuật toán.



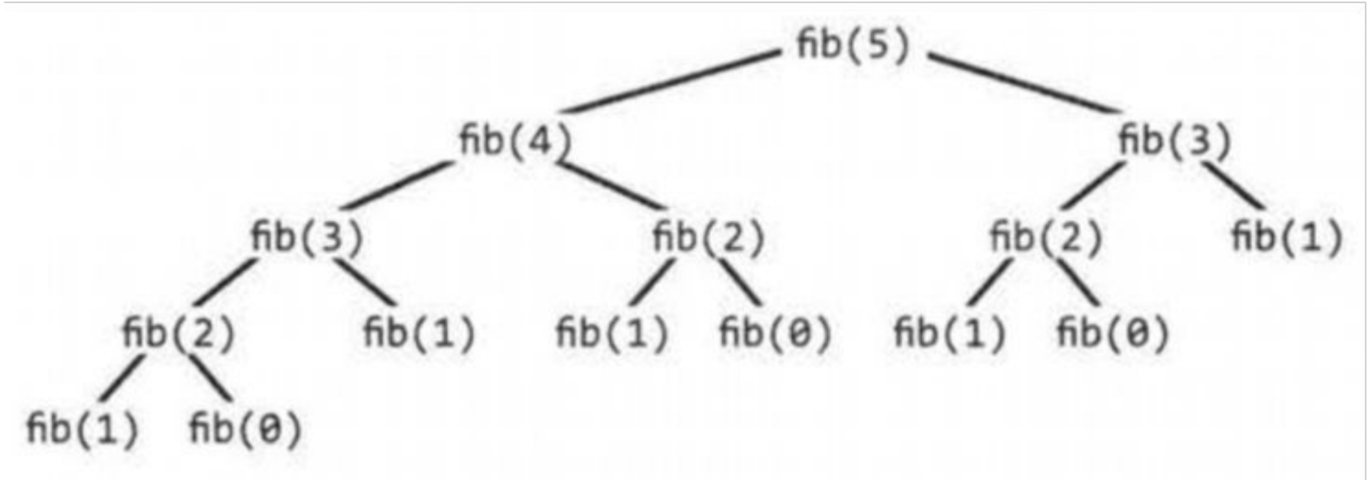
Bài toán Fibonacci với Đệ quy (góc nhìn top-down)

- Tại mỗi đỉnh:
  - Thời gian: O(1)
  - Không gian: O(1)



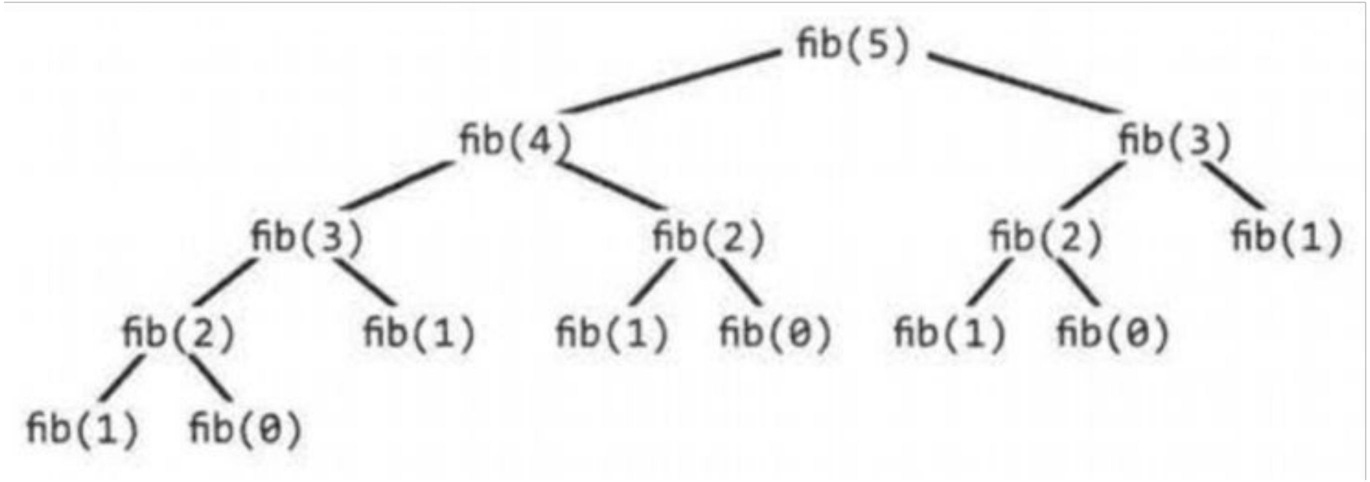
Bài toán Fibonacci với Đệ quy (góc nhìn top-down)

- Tổng thời gian: O(2^n)
- Tổng không gian: O(n) - chiều cao của cây, tại sao?

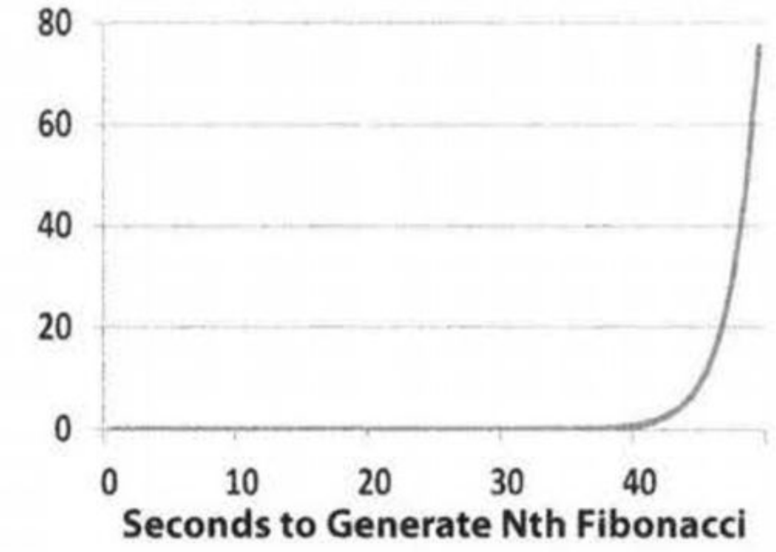


Bài toán Fibonacci với Đệ quy (góc nhìn top-down)

- Tổng không gian: O(n) - chiều cao của cây, tại sao?
- Mỗi lần gọi đệ quy, chương trình lưu lại thông tin của hàm hiện tại vào bộ nhớ stack. Thuật toán gọi đệ quy tối đa n lần, sử dụng O(n) bộ nhớ.

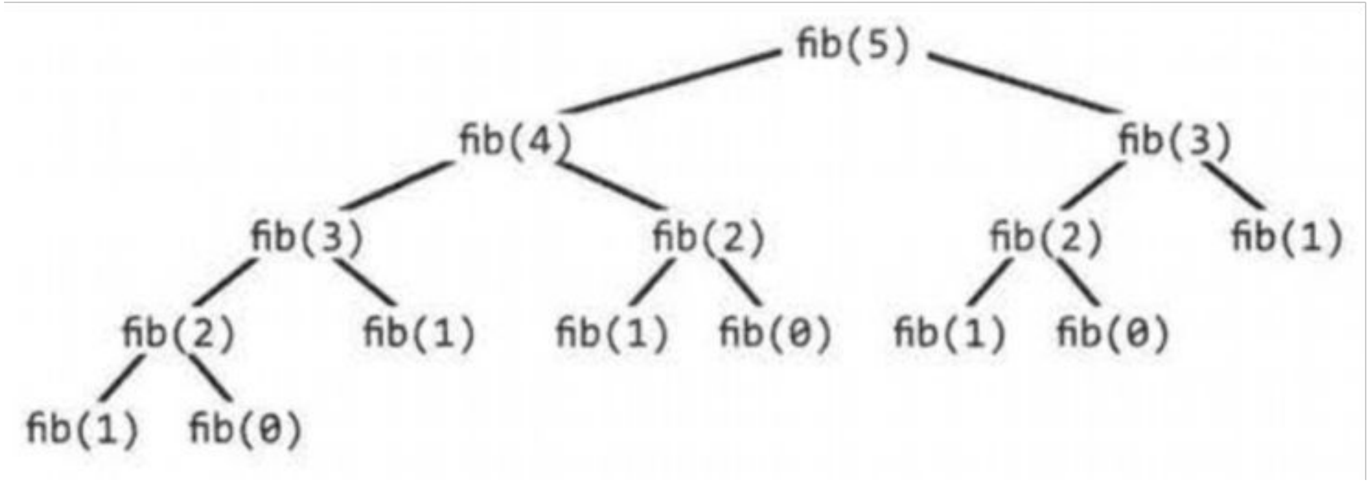


Bài toán Fibonacci với Đệ quy (góc nhìn top-down)

- 

Bài toán Fibonacci với Đệ quy (góc nhìn top-down)

- Chú ý: một số đỉnh trùng nhau trên cây.



Bài toán Fibonacci với Memoization

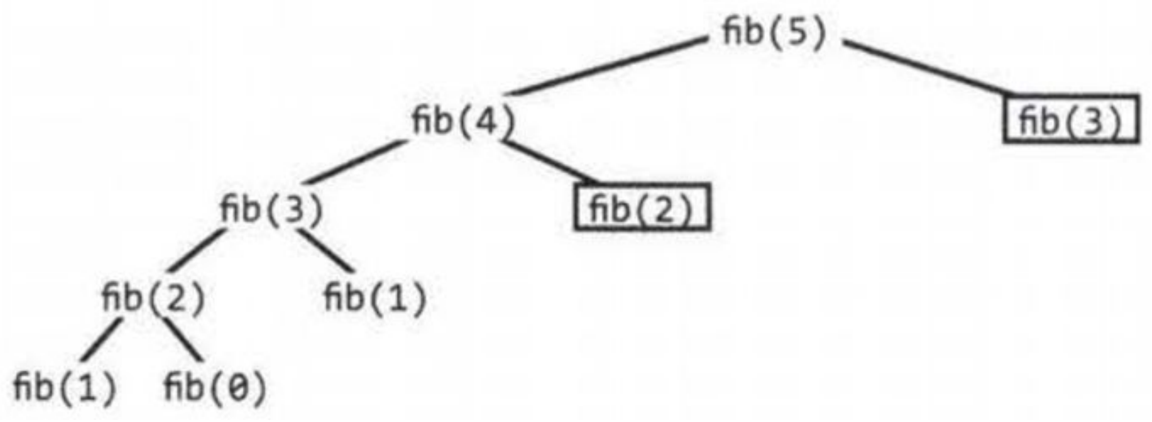
```
int fibo(int n) {
    return fibo(n, new int[n+1])
}

int fibo(int i, int[] cache) {
    if (i==0 || i==1) return i;
    if (cache[i] == 0) {
         cache[i] = fibo(i-1, cache) + fibo(i-2, cache)
    }
    return cache[i];
}
fibo_n = fibo(n);
```

* **Memoization**: https://stackoverflow.com/questions/6184869/what-is-the-difference-between-memoization-and-dynamic-programming

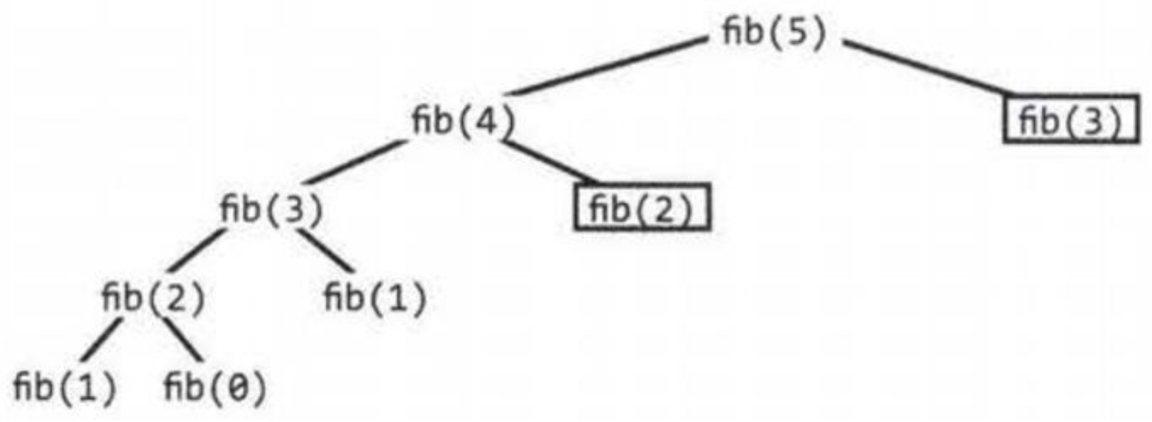
Bài toán Fibonacci với Memoization

- Cây đệ quy chứa ít đỉnh hơn.



Bài toán Fibonacci với Memoization

- Tổng thời gian: O(n)
- Tổng không gian: O(n)



Bài toán Fibonacci với Quy hoạch động (góc nhìn bottom-up)

```
int fibo(int n) {
    if (n==0) return 0;
    else if (n==1) return 1;

    int[] cache = new int[n+1];
    cache[0] = 0;
    cache[1] = 1;
    for (int i=2; i<=n; i++) {
        cache[i] = cache[i-1] + cache[i-2];
    }
    return cache[n];
}
```

Bài toán Fibonacci với Quy hoạch động (góc nhìn bottom-up)

```
int fibo(int n) {
    if (n==0) return 0;
    else if (n==1) return 1;

    int[] cache = new int[n+1];
    cache[0] = 0;
    cache[1] = 1;
    for (int i=2; i<=n; i++) {
        cache[i] = cache[i-1] + cache[i-2];
    }
    return cache[n];
}
```

- Tổng thời gian: O(n)
- Tổng không gian: O(n)

Bài toán Fibonacci với Quy hoạch động (góc nhìn bottom-up)

```
int fibo(int n) {
    if (n==0) return 0;
    else if (n==1) return 1;

    int[] cache = new int[n+1];
    cache[0] = 0;
    cache[1] = 1;
    for (int i=2; i<=n; i++) {
        cache[i] = cache[i-1] + cache[i-2];
    }
    return cache[n];
}
```

- Tổng thời gian: O(n)
- Tổng không gian: O(n) => O(1)

Bài toán Fibonacci với Quy hoạch động (góc nhìn bottom-up)

```
int fibo(int n) {
   if (n==0) return 0;
   int a=0;
   int b=1;
   for (int i=2; i<=n; i++) {
      int c = a+b;
      a = b;
      b = c;
   }
   return a+b;
}
```

- Tổng thời gian: O(n)
- Tổng không gian: O(1)

Bài toán Fibonacci với Quy hoạch động (góc nhìn bottom-up)

```
int fibo(int n) {
   if (n==0) return 0;
   int a=0;
   int b=1;
   for (int i=2; i<=n; i++) {
      int c = a+b;
      a = b;
      b = c;
   }
   return a+b;
}
```

- Tổng thời gian: O(n) => O(log2(n))?
- Tổng không gian: O(1)

Bài toán Fibonacci với phương pháp nhân ma trận (nâng cao)

- Donald E. Knuth đưa ra ma trận đơn vị:

$$Q = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Bài toán Fibonacci với phương pháp nhân ma trận (nâng cao)

- Donald E. Knuth đưa ra ma trận đơn vị:

$$Q = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

Bài toán Fibonacci với phương pháp nhân ma trận (nâng cao)

- Donald E. Knuth đưa ra ma trận đơn vị:

$$Q = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

$$F_n = Q^{n-1}(1,1)$$

Bài toán Fibonacci với phương pháp nhân ma trận (nâng cao)

- Donald E. Knuth đưa ra ma trận đơn vị:

$$Q = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

$$F_n = Q^{n-1}(1,1)$$

- Chứng minh: sử dụng **quy nạp**

Bài toán Fibonacci với phương pháp nhân ma trận (nâng cao)

- Donald E. Knuth đưa ra ma trận đơn vị:

$$Q = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

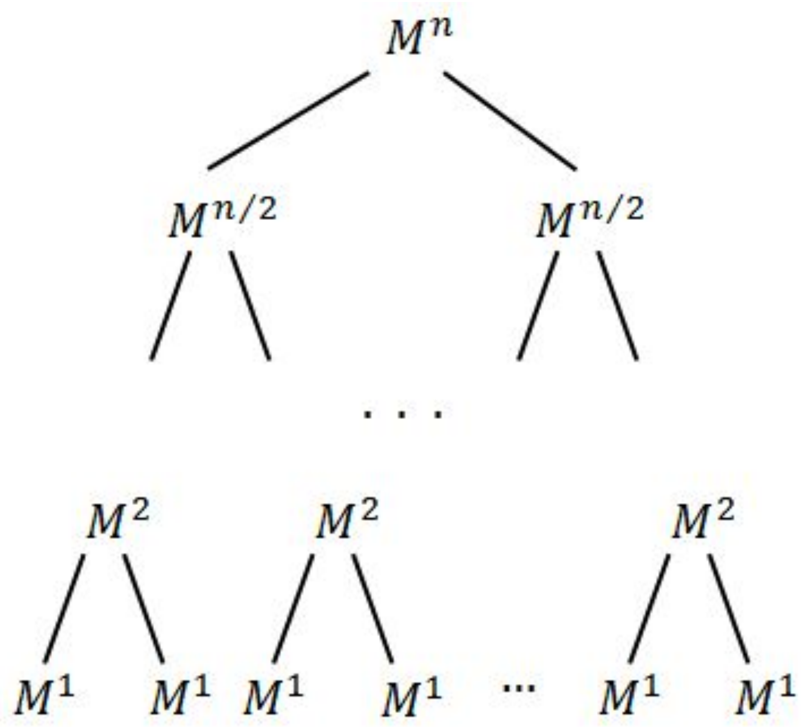
$$F_n = Q^{n-1}\,(1,1)$$

- Độ phức tạp?

Bài toán Fibonacci với phương pháp nhân ma trận (nâng cao)

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

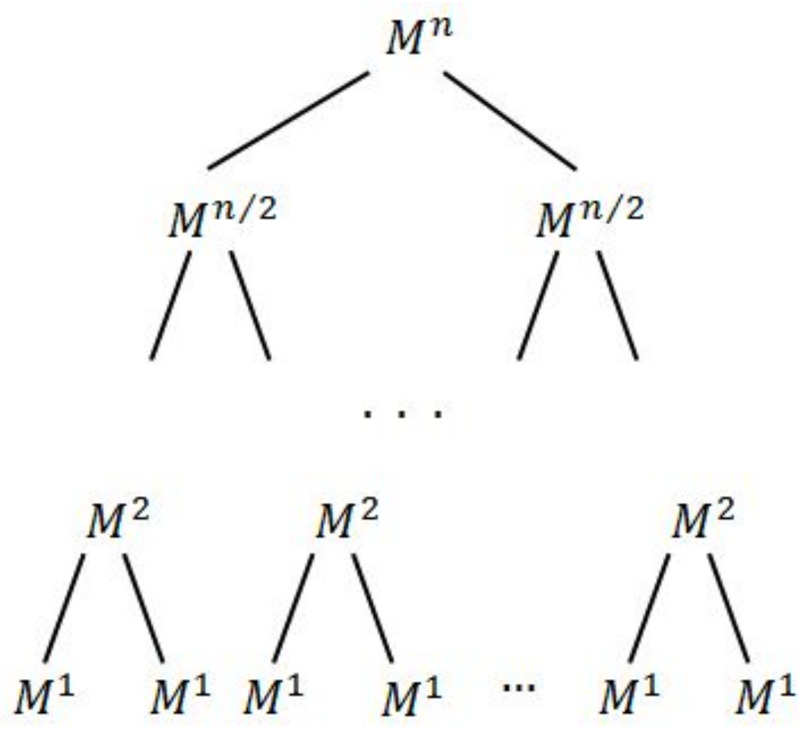
- Tổng thời gian: O(log2(n))
- Tổng không gian: O(log2(n))
- Chạy nhanh khi n rất lớn.



Bài toán Fibonacci với phương pháp nhân ma trận (nâng cao)

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

- Tổng thời gian: O(log2(n))
- Tổng không gian: O(log2(n))
- Chạy nhanh khi n rất lớn.



Code: https://kukuruku.co/post/the-nth-fibonacci-number-in-olog-n/

Bài luyện tập

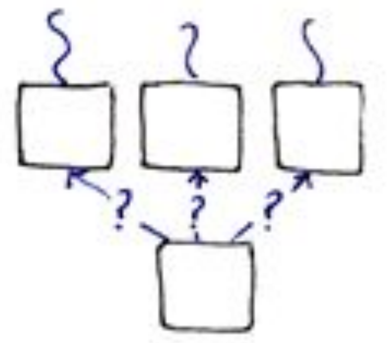
- **Bậc thang:** Một đứa trẻ đứng dưới cầu thang có n bậc thang. Đứa trẻ có thể bước lên 1 bậc, 2 bậc, hoặc 3 bậc một lúc. Hãy đưa ra một thuật toán để đếm số cách khác nhau để đứa trẻ bước lên được bậc thứ n. Đồng thời, phân tích độ phức tạp thuật toán.
- **Robot:** Một con robot hút bụi được đặt trên một nền gạch có nhiều ô kích thước m*n. Robot bắt đầu công việc hút bụi của mình ở góc trên bên trái nền gạch. Robot chỉ có thể di chuyển theo hướng đông hoặc hướng nam. Biết rằng một số ô gạch có chứa vật cản và robot không thể đi vào những ô này. Hãy đề xuất một thuật toán đếm số cách khác nhau để robot có thể di chuyển đến góc phải dưới của nền gạch. Đồng thời, phân tích độ phức tạp thuật toán.

Bài luyện tập

- **Mua vé:** Có N người đang xếp hàng mua vé. Người thứ i mua vé cho mình thì mất Ti phút. Tuy nhiên, nếu người thứ i mua luôn vé cho cả người đứng sau thì tổng thời gian sẽ mất Ri phút. Tìm tổng thời gian khi những người xếp hàng mua vé một cách tối ưu nhất.
- **Hai dãy số:** Cho 2 số nguyên dương m, n và 2 dãy số nguyên A1, A2, ..., Am và B1, B2, ..., Bn. Hãy loại đi một số phần tử của 2 dãy sao cho các số còn lại của 2 dãy (giữ nguyên thứ tự cũ) tạo thành 2 dãy giống nhau và có độ dài k là lớn nhất.

Bài luyện tập

- **Ma trận:** Cho một ma trận chứa MxN số nguyên. Một con robot đứng ở lề dưới (biên dưới) của ma trận và mong muốn di chuyển sang lề bên trên (biên trên) của ma trận. Robot có thể chọn vị trí bắt đầu bất kì ở hàng dưới cùng và kết thúc ở bất kì ô nào ở hàng trên cùng của ma trận. Khi robot đang ở ô [i,j], nó chỉ có thể di chuyển đến 3 ô [i-1,j-1], [i-1,j], và [i-1,j+1]. Hãy tìm tổng giá trị của đường đi nhỏ nhất của robot.



Một số trang web luyện tập

- **Leetcode:** https://leetcode.com/tag/dynamic-programming/
- **HackerRank:** https://www.hackerrank.com/domains/algorithms?filters%5Bsubdomains%5D%5B%5D=dynamic-programming