Đệ quy

Khái niệm

Khái niệm

Đệ quy là khái niệm dùng để chỉ *những đối tượng* mà khi định nghĩa, người ta lại dùng *chính những đối tượng đó*.

Ví dụ

Người ta có thể định nghĩa n! (n giai thừa) như sau:

- 0! = 1.
- n! = n * (n-1)! với n ≥ 1.

Đây là định nghĩa đệ quy vì có sử dụng lại chính khái niệm giai thừa trong định nghĩa.

Ví dụ

Ta có thể định nghĩa a^n đệ quy như sau:

- $a^0 = 1$
- $a^n = a^{n/2} * a^{n/2}$, nếu n chẵn
- $a^n = a^{n/2} * a^{n/2} * a$, nếu n lẻ

Ví dụ

Dãy Fibonacci:

- $F_0 = 1$
- $F_1 = 1$
- $F_{n+2} = F_{n+1} + F_n$, với $n \geq 0$

Khái niệm

Trong khoa học máy tính, đệ quy chỉ những hàm mà gọi lại chính nó trong quá trình thực thi.

Ví dụ

```
# ví dụ hàm tính số fibonacci thứ n
def fibo(n):
    if n <= 1:
        return 1
    else:
        return fibo(n-1) + fibo(n-2)
```

Ví dụ

```
# tính giai thừa bằng đệ quy
def tính_giai_thừa(n):
    if n == 0:
        return 1
    else:
        return n * tính_giai_thừa(n-1)
```

Đặc điểm

Qua các ví dụ trên, ta có thể thấy một định nghĩa (hay hàm) đệ quy luôn có 2 thành phần:

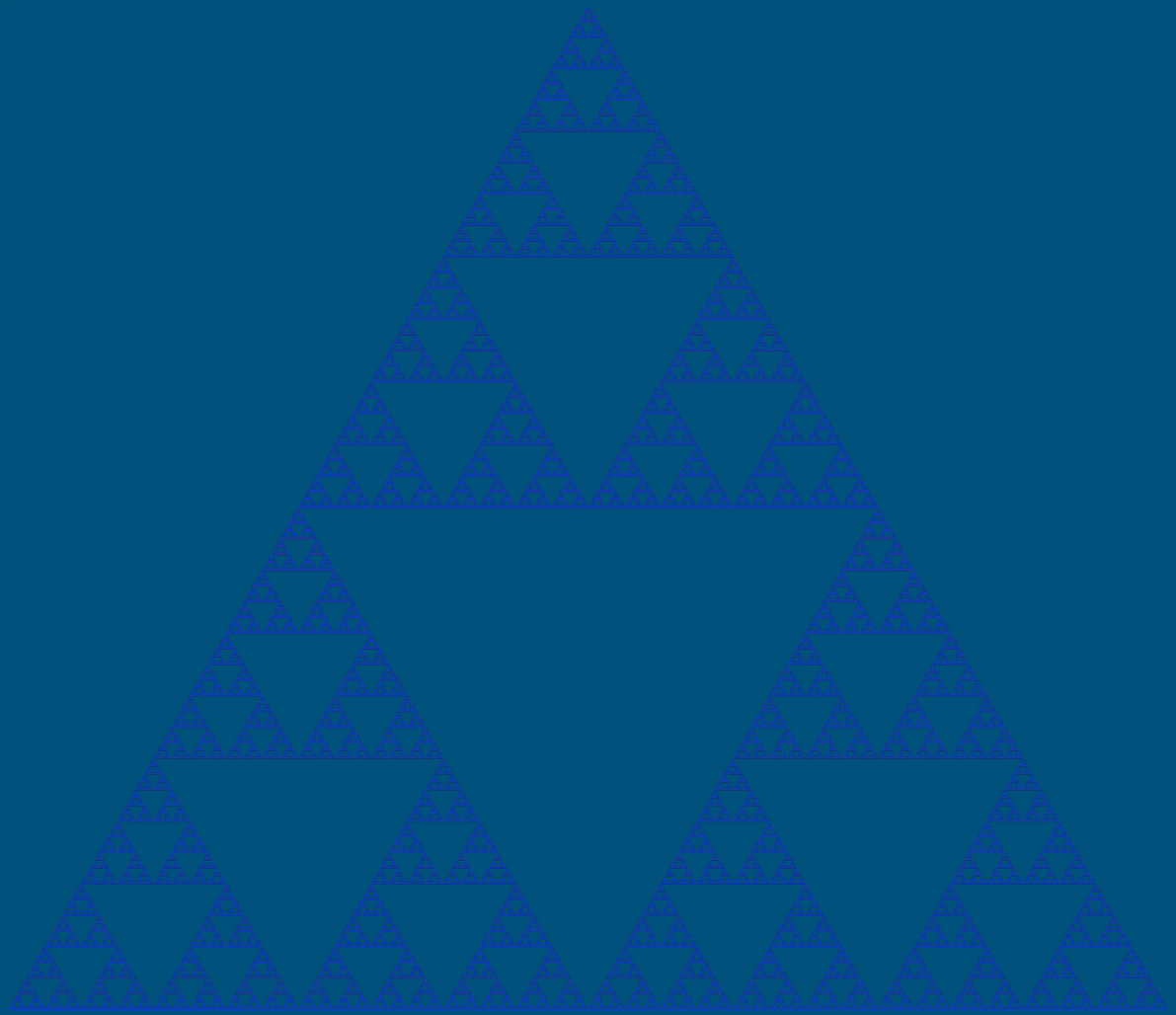
- Một thành phần định nghĩa các trường hợp cơ bản, gọi là phần neo.
- Một thành phần chứa đệ quy.

Xây dựng thuật toán đệ quy

Để xây dựng một giải thuật đệ quy, ta có thể thực hiện các bước sau:

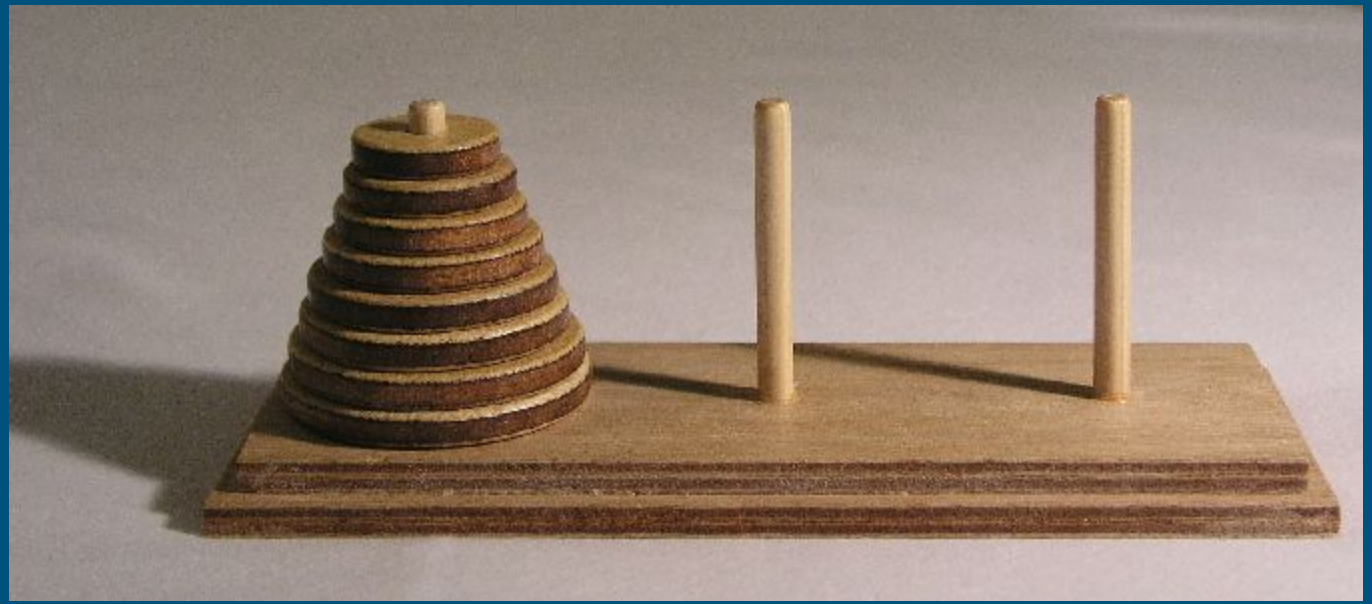
- Mô hình hóa bài toán.
- Giải các trường hợp đơn giản (*tìm phần neo*).
- Giải các trường hợp phức tạp bằng cách đưa về trường hợp đơn giản hơn (*xây dựng phần đệ quy*).

Ví dụ



Ví dụ

Bài toán Tháp Hà Nội: Cho trước ba chiếc cọc chôn thẳng đứng. Trên một cọc đã xếp n (n = 1, 2...) đĩa có đường kính khác nhau xuyên qua lỗ thủng ở giữa. Đĩa to nằm dưới, đĩa nhỏ hơn nằm trên. Vấn đề đặt ra là làm thế nào để chuyển từng chiếc một toàn bộ số đĩa đó sang chiếc cọc thứ ba thông qua chiếc cọc thứ hai với yêu cầu trong quá trình chuyển, ở cả ba cọc luôn có trật tự đĩa to nằm dưới đĩa bé nằm trên.



Ví dụ

Tương truyền rằng: Trong một ngôi đền thiêng liêng có ba chiếc cọc bằng kim cương, được chôn chặt, thẳng đứng. Ông trời đã xếp 64 chiếc đĩa vàng có các đường kính khác nhau và có lỗ ở giữa vào một chiếc cọc theo thứ tự to trước nhỏ sau. Những tu sĩ Brahmin đã tiếp nối nhau trong một thời gian dài để di chuyển 64 đĩa vàng của Tòa tháp Brahma. Khi công việc hoàn thành, Tòa tháp và Brahmin sẽ đổ, và lúc đó là thời điểm kết thúc của vũ trụ!

Mô hình hóa bài toán

- Gọi ba cái cột lần lượt là A, B và C.
- Có n cái đĩa ở cột A.
- Mỗi bước chỉ được di chuyển 1 đĩa từ cột này sang cột khác, một bước hợp lệ là sau khi di chuyển thì đĩa nhỏ phải ở trên đĩa to. Ta ký hiệu bước di chuyển hợp lệ là x > y (x và y là tên cột).
- Mục tiêu là xác định các bước đi hợp lệ để di chuyển n cái đĩa từ cột A sang cột C với cột B là cột trung gian.

Ta ký hiệu thuật toán ở trên là T(n, A, C, B).

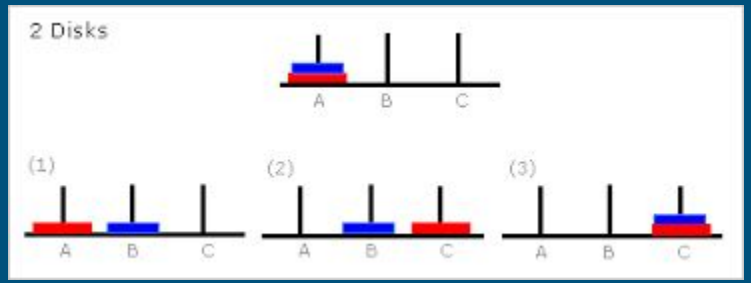
Giải các trường hợp đơn giản

Với trường hợp T(1, A, C, B) chúng ta cần thực hiện bước A > C là hoàn thành.

Giải các trường hợp đơn giản

Với trường hợp T(2, A, C, B) chúng ta cần thực hiện ba bước:

1. A > B
2. A > C
3. B > C



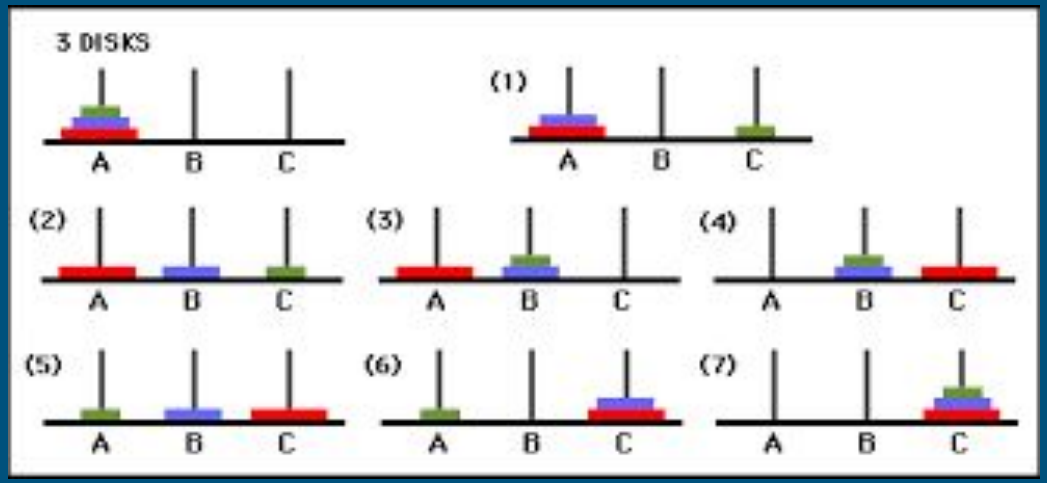
Giải các trường hợp đơn giản

Bài tập: Tìm T(3, A, C, B)?

Giải các trường hợp đơn giản

Đáp án: Với T(3, A, C, B), chúng ta thực hiện theo bảy bước:

1. A > C
2. A > B
3. C > B
4. A > C
5. B > A
6. B > C
7. A > C

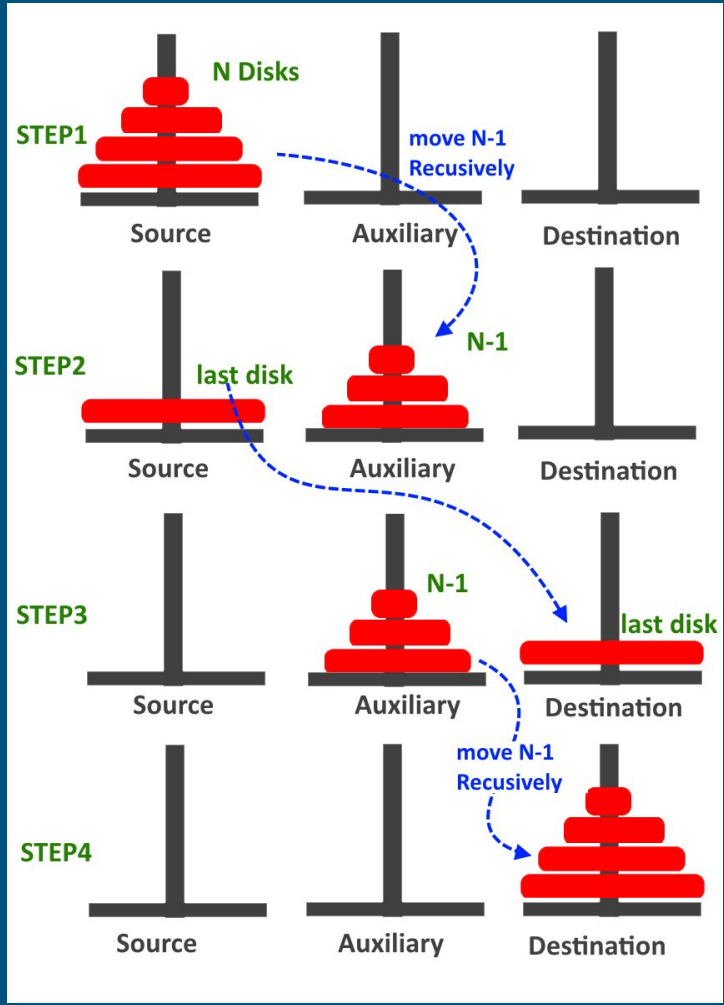


Giải trường hợp tổng quát

Vậy, trong trường hợp tổng quát T(n, A, C, B), chúng ta có thể thực hiện theo cách nào?

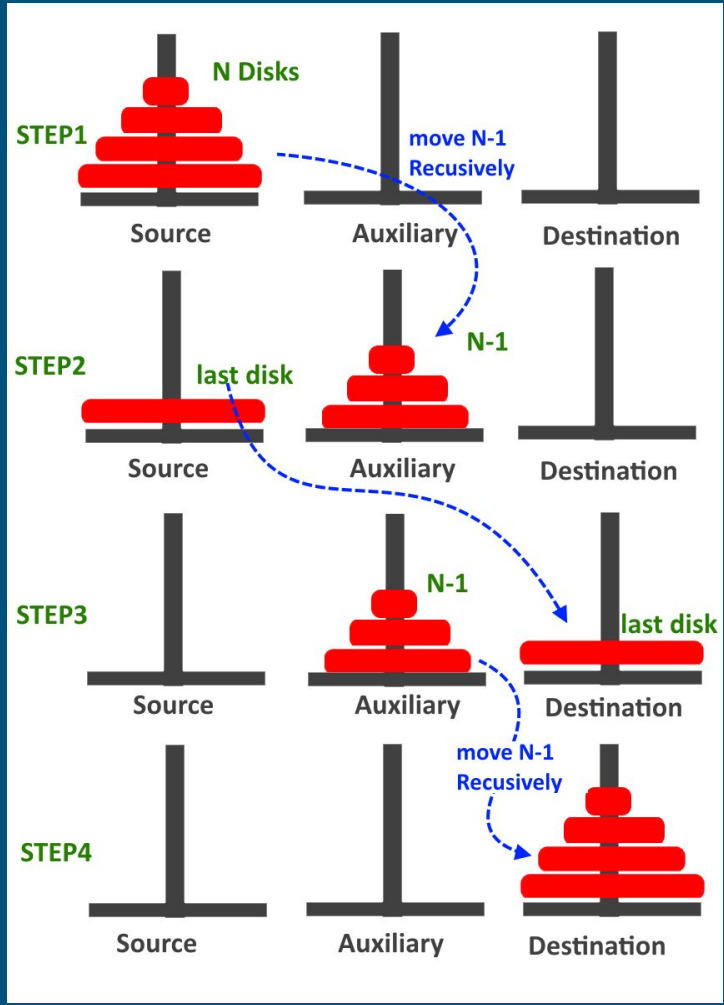
Giải trường hợp tổng quát

Có thể thấy rằng, chúng ta phải thực hiện một bước, đó là chuyển đĩa to nhất (đĩa dưới cùng) từ cột A sang cột C.



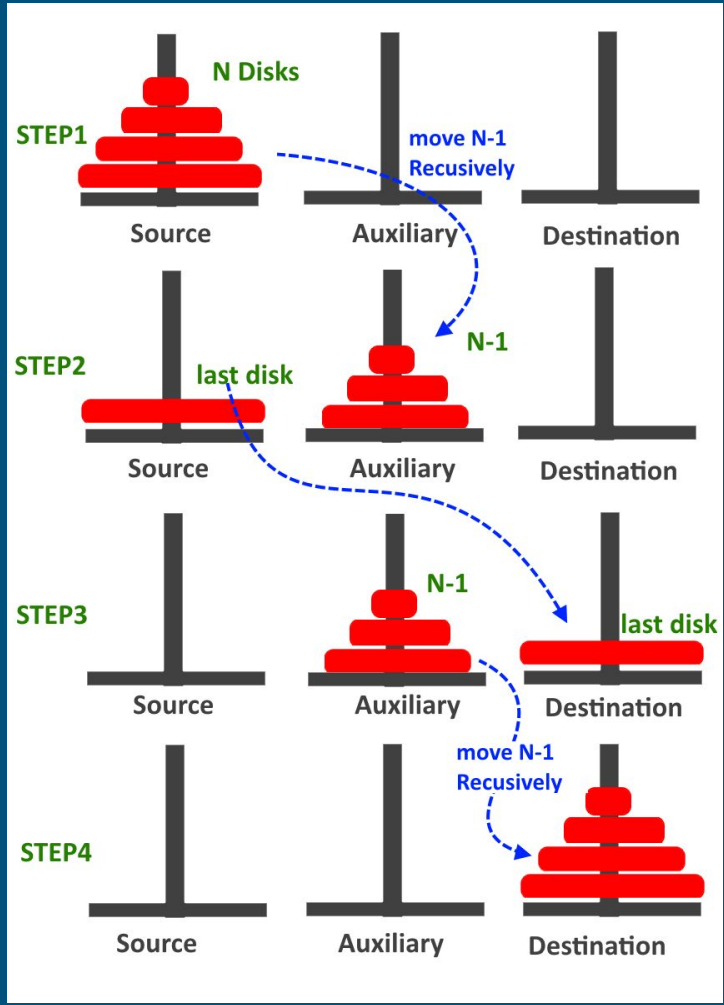
Giải trường hợp tổng quát

Để thực hiện điều đó, chúng ta cần di chuyển những đĩa phía trên sang cột B bằng cách thực hiện T(n-1, A, B, C).



Giải trường hợp tổng quát

Sau khi thực hiện bước chuyển đĩa lớn nhất từ A sang C, chúng ta chỉ cần thực hiện việc chuyển những đĩa ở cột B sang cột C với cột A là trung gian, tức là thực hiện T(n-1, B, C, A).



Giải trường hợp tổng quát

```
# Vậy, ta có thuật toán hoàn chỉnh như sau
# Trường hợp neo của bài toán này là T(3, A, C, B)
T(n, A, C, B):
    T(n-1, A, B, C)
    A > C
    T(n-1, B, C, A)
```

Giải trường hợp tổng quát

Theo thuật toán trên, để di chuyển n đĩa chúng ta cần dùng 2^n - 1 phép di chuyển.

Để dễ hình dung, giả sử các tu sĩ mất một giây để di chuyển một đĩa vàng, các tu sĩ sẽ mất 2^{64} - 1 giây, tương đương với 585 tỷ năm.

Cài đặt đệ quy

Ưu điểm

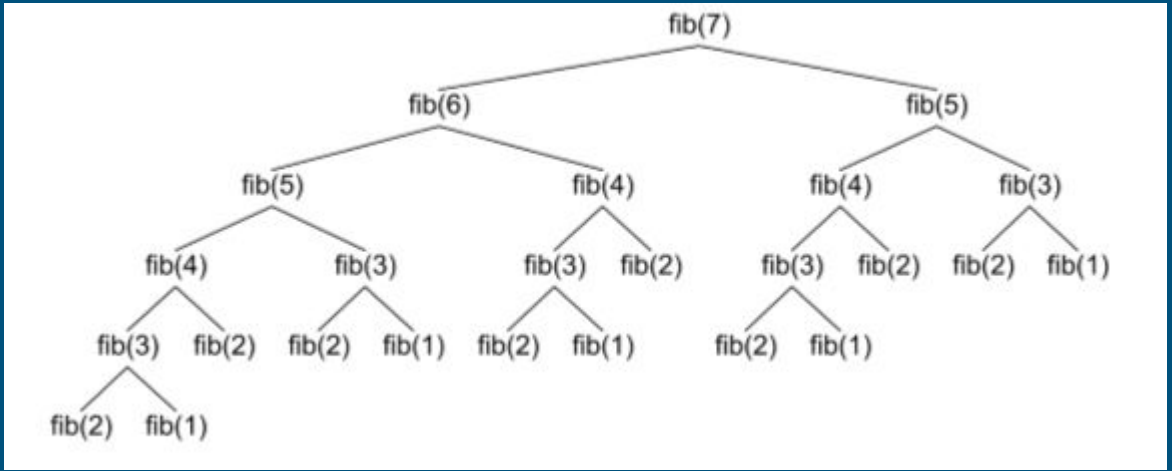
- Dễ hiểu.
- Dễ cài đặt.
- Thích hợp cho các bài toán duyệt cây.

Nhược điểm

- Dễ gây tràn stack.
- Thời gian thực hiện chậm nếu không tối ưu code (do phải tốn tài nguyên tính toán các kết quả trùng lặp)

Ví dụ

Để làm rõ hơn nhược điểm, chúng ta xem quá trình tính toán số fibo(7)



Ví dụ (tiếp)

Có thể thấy, để tính toán fibo(7) chúng ta cần lặp lại quá trình tính toán fibo(5) 2 lần, từ đó dẫn đến lãng phí tài nguyên.

Một số kỹ thuật

Để tránh việc lãng phí tài nguyên này, chúng ta có một số hướng giải quyết:

- Khử đệ quy.
- Ghi nhớ giá trị.

Khử đệ quy

Khử đệ quy là quá trình viết lại chương trình dưới dạng không chứa đệ quy (không chứa phần gọi lại chính nó), có thể thực hiện theo vài phương pháp:

- Sử dụng vòng lặp.
- Sử dụng các công thức tường minh.
- ...

Khử đệ quy

```
#khử đệ quy bằng vòng lặp
def tính_giai_thừa(n):
    if n < 2:
        res = 1
    else:
        res = 1
        for i = 2, ..., n:
            res = res * i
    return res
```

Khử đệ quy

```
#khử đệ quy bằng công thức tường minh
def fibo(n):
    s = 5 ** 0.5
    phi = (1 + s) / 2
    return (phi ** n - (1 / phi) ** n) / s
```

Khử đệ quy (tiếp)

Người ta chỉ ra rằng mọi giải thuật đệ quy đều có thể viết lại dưới dạng không đệ quy, tuy nhiên, không có giải pháp khử đệ quy tổng quát cho mọi trường hợp.

Ghi nhớ giá trị

Một vấn đề của đệ quy đó là việc tính toán các giá trị trùng lặp nhiều lần khiến chương trình bị chậm, do đó một hướng giải quyết là lưu trữ lại những giá trị đã tính toán để tái sử dụng.

Ví dụ

```
# Tính số fibonacci bằng cách lưu trữ giá trị
def new_fibo(n):
    a = []                          # tạo mảng lưu giá trị
    a[0] = a[1] = 1
    for i = 2, ..., n:
        a[i] = a[i-1]+a[i-2]  # tính a[i] bằng các giá trị đã lưu
    return a[n]
```

Ví dụ

Đối với việc thuật toán sử tính số Fibonacci:

- Nếu sử dụng cách gọi đệ quy bình thường, độ phức tạp là $O(1.6^n)$.
- Nếu sử dụng cách lưu trữ giá trị, độ phức tạp sẽ là $O(n)$.