Data Structure & Algorithm

Algorithm

Recursion

(Đệ quy)

1. Định nghĩa

递 **đệ [đái]**

U+9012, tổng 10 nét, bộ sước 辶 + 7 nét
giản thể, hình thanh

Từ điển phổ thông

đưa, chuyển, giao

Từ điển trích dẫn

1. Giản thể của chữ 遞.

归 **quy**

U+5F52, tổng 5 nét, bộ kệ 彐 + 2 nét
giản thể, hình thanh

Từ điển phổ thông

trở về

Từ điển trích dẫn

1. Giản thể của chữ 歸.

1. Định nghĩa

- **Recursion – Đệ quy**: Một hàm tự gọi đến chính nó.

1. Định nghĩa

- **Recursion – Đệ quy**: Một hàm tự gọi đến chính nó.

```
public void printF()
{
    System.out.println("F");
    printF();
}
```

2. Khi nào sử dụng Đệ quy?

- Khi xuất hiện yếu tố đệ quy trong bài toán.

Định nghĩa đệ quy [sửa | sửa mã nguồn]

Ta có thể định nghĩa đệ quy (quy nạp) n! như sau

1. $0! = 1$
2. $(n + 1)! = n! \times (n + 1)$ với $n > 0$

3. Đặc điểm của Đệ Quy:

- Một bài toán chỉ áp dụng được phương pháp đệ quy nếu có đủ 2 yếu tố sau đây:

  - ✓ Bài toán cơ sở / Điều kiện dừng

$$0! = 1$$

  - ✓ Công thức quy nạp:
    Phải đưa về bài toán con nhỏ hơn (cuối cùng là bài toán cơ sở).

$$n! = n * (n-1)!$$

4. Thực hành 1

- Implement bài toán tính giai thừa.

✓ Bài toán cơ sở:

0! = 1 → GiaiThua(0) = 1

✓ Công thức quy nạp:

n! = n * (n-1)! → GiaiThua(n) = n * GiaiThua(n-1)

5. Cây đệ quy

- Cần hiểu rõ từng bước chương trình chạy như thế nào để không bị nhầm lẫn.

```
public static int GiaiThua(int n){
    // Bai toan co so
    if(n == 0)
        return 1;
    // Cong thuc quy nap
    int t = n * GiaiThua(n-1);
    return t;
}
```

6. Thực hành 2

- Implement bài toán tìm số Fibonaci thứ n

Dãy Fibonacci

Bách khoa toàn thư mở Wikipedia

Dãy Fibonacci là dãy vô hạn các số tự nhiên bắt đầu bằng hai phần tử 0 và 1 hoặc 1 và 1, các phần tử sau đó được thiết lập theo quy tắc *mỗi phần tử luôn bằng tổng hai phần tử trước nó*. Công thức truy hồi của dãy Fibonacci là:

$$F(n) := \begin{cases} 1, & \text{khi } n = 1; \\ 1, & \text{khi } n = 2; \\ F(n-1) + F(n-2) & \text{khi } n > 2. \end{cases}$$

6. Thực hành 2

- Implement bài toán tìm số Fibonaci thứ n

✓ Bài toán cơ sở?

Fibo(1) = 1;
Fibo(2) = 1;

✓ Công thức quy nạp?

Fibo(n) = Fibo(n-1) + Fibo(n-2)

6. Thực hành 2

- Implement bài toán tìm số Fibonaci thứ n
  - ✓ CODE
  - ✓ Cây đệ quy

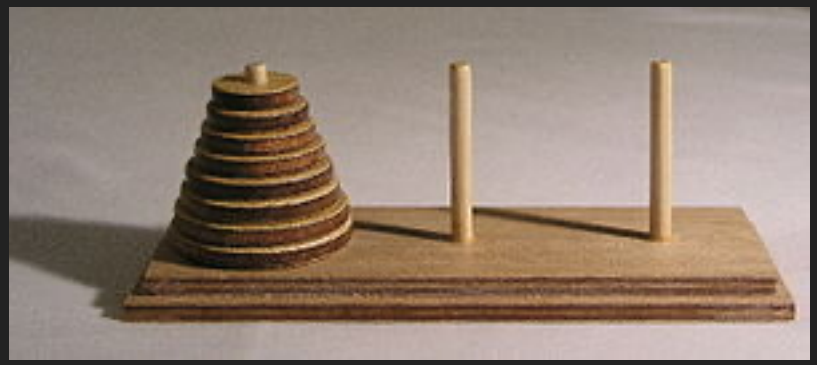
```
public static int Fibo(int n){
    // 1. Bai toan co so
    if(n <= 2)
        return 1;
    // 2. Cong thuc quy nap
    int Fn = Fibo(n-1) + Fibo(n-2);
    return Fn;
}
```

7. Một số bài toán khác

- ✓ Tính tổng từ 1 đến n: T(n) = n + T(n-1)
- ✓ Tính số mũ: A^n = A^n-1 * A
- ✓ In giá trị của một mảng số nguyên.
- ✓ Tìm Ước số chung lớn nhất - UCLN
- ✓ Tìm Bội số chung nhỏ nhất - BCNN
- ✓ Bài toán Tháp Hà Nội
- ✓ …

7. Một số bài toán khác

✓Bài toán Tháp Hà Nội



Luật chơi [sửa | sửa mã nguồn]

Dạng thường gặp nhất của trò chơi này gồm một bộ các đĩa kích thước khác nhau, có lỗ ở giữa, nằm xuyên trên ba cái cọc. Bài toán đố bắt đầu bằng cách sắp xếp các đĩa theo trật tự kích thước vào một cọc sao cho đĩa nhỏ nhất nằm trên cùng, tức là tạo ra một dạng hình nón. Yêu cầu của trò chơi là di chuyển toàn bộ số đĩa sang một cọc khác, tuân theo các quy tắc sau:

- Chỉ có 3 cột để di chuyển.
- Một lần chỉ được di chuyển một đĩa (không được di chuyển đĩa nằm giữa).
- Một đĩa chỉ có thể được đặt lên một đĩa lớn hơn (không nhất thiết hai đĩa này phải có kích thước liền kề, tức là đĩa nhỏ nhất có thể nằm trên đĩa lớn nhất).

7. Một số bài toán khác

✓Bài toán Tháp Hà Nội

Khách tham quan tương tác tại Universum, Bảo tàng Khoa học của Đại học Tự trị Quốc gia Mexico

8. Phân loại đệ quy

- Bài toán: In giá trị của một mảng số nguyên.
  - ✓ In các phần tử theo thứ tự từ 0 đến n-1.
  - ✓ In các phần tử theo thứ tự từ n-1 đến 0.

- Phân loại đệ quy: (Dựa theo thứ tự thực hiện phần logic và lời gọi hàm)
  - ✓ Đệ quy thực hiện công việc trước rồi Đệ quy sau.
  - ✓ Đệ quy Đệ quy trước rồi thực hiện công việc sau.

9. Ưu điểm và nhược điểm

❖ Ưu điểm:

- ✓ Ngắn gọn – dễ hình dung.
- ✓ Dễ implement (triển khai)

❖ Nhược điểm:

- ✓ Nếu xử lý không đúng dễ gây StackOverflow.
- ✓ Tốn hiệu năng về mặt thời gian và không gian bộ nhớ.

10. Khử và tối ưu đệ quy

- Tối ưu đệ quy:
  - ✓ Đệ quy có nhớ: Lưu lại các kết quả đã tính được trước đó.
- Khử đệ quy:
  - ✓ Tìm cách giải bài toán mà không cần sử dụng đến đệ quy.

11. Khi nào sử dụng đệ quy

✓ Khi chúng ta cần implement nhanh một chức năng nào đó để test hoặc kiểm thử một giải pháp (solution nào đó) mà chưa cần quan tâm đến hiệu năng vội.

✓ Khi tài nguyên khi sử đệ quy gần tương đương với phương pháp không sử dụng đệ quy.

✓ Khi đệ quy là cách làm duy nhất mà bạn biết. ☺

12. Practice - Thực hành

✓ Tìm dách sách bài tập với #recursion

Data Structure & Algorithm

Please Like and Subcribe