GIẢI THUẬT ĐỆ QUY

CÁC KỸ THUẬT THIẾT KẾ GIẢI THUẬT

Nội dung

- Khái niệm đệ quy.
- Thiết kế giải thuật đệ quy

Khái niệm đệ quy

- Đệ quy là một khái niệm cơ bản trong toán học và khoa học máy tính. Một đối tượng được gọi là đệ quy nếu nó hoặc một phần của nó được định nghĩa thông qua khái niệm về chính nó.
- Trong lĩnh vực lập trình: 1 chương trình gọi là đệ quy nếu nó gọi lại chính nó.
- Chương trình đệ quy luôn kiểm tra điều kiện dừng:
  - Nếu không thỏa mãn, tiếp tục gọi đệ quy.
  - Nếu thỏa mãn ➔ không gọi chính nó nữa, chấm dứt đệ quy.

Khái niệm đệ quy

- Ví dụ:
  - Định nghĩa số tự nhiên:
    - 0 là số tự nhiên.
    - Nếu k là số tự nhiên thì k+1 cũng là số tự nhiên.
  - Định nghĩa xâu kí tự (chuỗi kí tự) bằng đệ quy:
    - Xâu rỗng là xâu kí tự
    - Một chữ cái bất kì ghép với 1 xâu sẽ tạo thành xâu mới
  - Định nghĩa hàm giai thừa n!:
    - Khi n = 0, định nghĩa 0! = 1
    - Khi n>0, định nghĩa n! = (n-1)! * n

Khái niệm đệ quy

❑ Ví dụ

```
private int Factor(int n){
    if (n==0) return 1;
    else return n*Factor(n-1);
}
```

❑ Đặc điểm của chương trình đệ quy:

- Chương trình này có thể gọi chính nó
- Khi chương trình gọi chính nó thì mục đích là để giải quyết một vấn đề tương tự nhưng nhỏ hơn.
- Vấn đề nhỏ hơn này một lúc nào đó sẽ đơn giản tới mức chương trình có thể tự giải quyết mà không cần phải gọi chính nó nữa.

Khái niệm đệ quy

- Để xây dựng 1 chương trình đệ quy cần tồn tại 1 công thức đệ quy. Công thức này gồm 2 phần:
  - Phần đệ quy: recursive case
  - Phần cơ bản: base case
- Ưu điểm của chương trình đệ quy:
  - Có thể thực hiện một lượng lớn các thao tác tính toán thông qua 1 đoạn chương trình ngắn gọn.
  - Có thể định nghĩa một tập vô hạn các đối tượng thông qua 1 số hữu hạn lời phát biểu.

Điều kiện viết chương trình đệ quy

- Vấn đề cần xử lý phải giải quyết được bằng cách đệ quy.
- Ngôn ngữ dùng viết chương trình phải hỗ trợ đệ quy.
- Các loại đệ quy:
  - đệ quy trực tiếp: một hàm gọi tới chính nó
  - đệ quy gián tiếp: Một hàm gọi tới hàm khác, hàm khác này gọi tới hàm ban đầu

Khi nào không nên sử dụng đệ quy

- Khi một hàm đệ quy gọi chính nó thì tập các đối tượng được sử dụng trong hàm này như: biến, hằng, cấu trúc… và các thông số cần cho việc chuyển giao điều khiển sẽ được sinh ra.
- Sử dụng đệ quy đôi khi tạo ra những tính toán thừa, không cần thiết do tính chất tự động gọi thực hiện thủ tục khi chưa kết thúc đệ quy.
- Nếu chương trình có thể viết dưới dạng lặp hoặc các cấu trúc lệnh khác thì không nên sử dụng đệ quy.

Ví dụ

- *Ví dụ 1 : Lập hàm tính n! bằng đệ quy*

```
int GT(int n)
{
        if (n==0)       // điểm dừng
                return 1;
        else
                return n*GT(n-1);
}
```

6. Đệ quy (Recursion)

Minh họa

Gọi hàm **`answer <- GT(5)`**

CT chính: Chưa xong: answer <- GT(5)

6. Đệ quy (Recursion)

Minh họa

GT. 1st: N=5,	Chưa xong: 5*GT(4)

CT chính:	Chưa xong: answer <- GT(5)

6. Đệ quy (Recursion)

Minh họa

GT. 2nd: N=4,	**Chưa xong: 4*GT(3)**
GT. 1st: N=5,	**Chưa xong: 5*GT(4)**
CT chính:	**Chưa xong: answer <- GT(5)**

6. Đệ quy (Recursion)

Minh họa

GT. 3rd: N=3,	**Chưa xong: 3*GT(2)**
GT. 2nd: N=4,	**Chưa xong: 4*GT(3)**
GT. 1st: N=5,	**Chưa xong: 5*GT(4)**
CT chính:	**Chưa xong: answer <- GT(5)**

6. Đệ quy (Recursion)

Minh họa

GT. 4th: N=2,	**Chưa xong: 2*GT(1)**
GT. 3rd: N=3,	**Chưa xong: 3*GT(2)**
GT. 2nd: N=4,	**Chưa xong: 4*GT(3)**
GT. 1st: N=5,	**Chưa xong: 5*GT(4)**
CT chính:	**Chưa xong: answer <- GT(5)**

6. Đệ quy (Recursion)

Minh họa

GT. 5th: N=1,	Chưa xong: 1*GT(0)
GT. 4th: N=2,	Chưa xong: 2*GT(1)
GT. 3rd: N=3,	Chưa xong: 3*GT(2)
GT. 2nd: N=4,	Chưa xong: 4*GT(3)
GT. 1st: N=5,	Chưa xong: 5*GT(4)
CT chính:	Chưa xong: answer <- GT (5)

6. Đệ quy (Recursion)

Minh họa

GT. 6th: N=0,	xong: returns 1
GT. 5th: N=1,	Chưa xong: 1*GT(0)
GT. 4th: N=2,	Chưa xong: 2*GT(1)
GT. 3rd: N=3,	Chưa xong: 3*GT(2)
GT. 2nd: N=4,	Chưa xong: 4*GT(3)
GT. 1st: N=5,	Chưa xong: 5*GT(4)
CT chính:	Chưa xong: answer <- GT(5)

6. Đệ quy (Recursion)

Minh họa

GT. 5th: N=1,	xong: returns 1*1
GT. 4th: N=2,	Chưa xong: 2*GT(1)
GT. 3rd: N=3,	Chưa xong: 3*GT(2)
GT. 2nd: N=4,	Chưa xong: 4*GT(3)
GT. 1st: N=5,	Chưa xong: 5*GT(4)
CT chính:	Chưa xong: answer <- GT(5)

6. Đệ quy (Recursion)

Minh họa

GT. 4th: N=2,	xong: returns 2*1
GT. 3rd: N=3,	Chưa xong: 3*GT(2)
GT. 2nd: N=4,	Chưa xong: 4*GT(3)
GT. 1st: N=5,	Chưa xong: 5*GT(4)
CT chính:	Chưa xong: answer <- GT(5)

6. Đệ quy (Recursion)

Minh họa

GT. 3rd: N=3,	**xong: returns 3*2**
GT. 2nd: N=4,	**Chưa xong: 4*GT(3)**
GT. 1st: N=5,	**Chưa xong: 5*GT(4)**
CT chính:	**Chưa xong: answer <- GT(5)**

6. Đệ quy (Recursion)

Minh họa

GT. 2nd: N=4,	**xong: returns 4*6**
GT. 1st: N=5,	**Chưa xong: 5*GT(4)**
CT chính:	**Chưa xong: answer <- GT(5)**

6. Đệ quy (Recursion)

Minh họa

GT. 1st: N=5, xong: returns 5*24
CT chính: Chưa xong: answer <- GT(5)

6. Đệ quy (Recursion)

Minh họa

CT chính: xong: answer <- 120

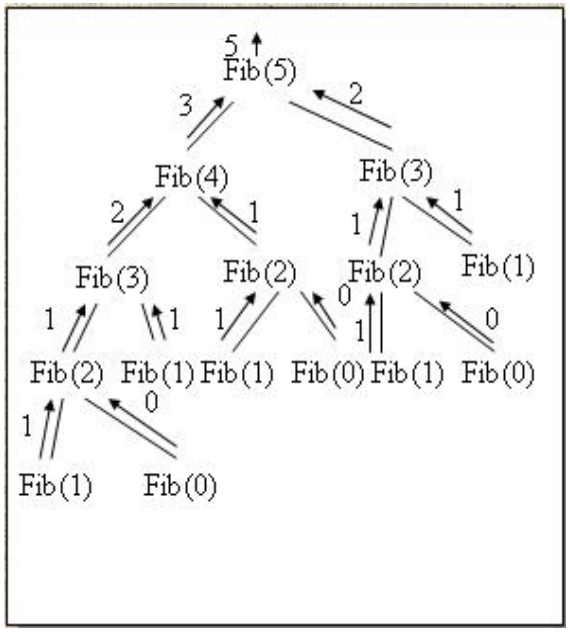
Ví dụ

- Dãy Fibonaci được định nghĩa như sau:
  - F(0) = 0;
  - F(1) = 1;
  - Với n>1 thì F(n) = F(n-1) + F(n-2)
- Phương thức đệ quy để tính dãy Fibonaci

```
int Fib(int i){
    if(i==0) return 0;
    if(i==1) return 1;
    return Fib (i-1)+Fib (i-2);
}
```

Base case

Recusive case



Ví dụ

- Phương thức dùng vòng lặp để tính dãy Fibonaci:

```
int Fib(int n){
    int fib1=1,fib0=0, fibn, i;
    if(n<=1)
        return n;
    else{
        for(i=2;i<=n;i++) {
            fibn=fib1+fib0;
            fib0=fib1;
            fib1=fibn;
        }
        return fibn;
    }
}
```

Thiết kế giải thuật đệ quy

- Thuật toán Euclid tính ước số chung lớn nhất của hai số nguyên dương.
  - Ước số chung lớn nhất của hai số nguyên dương m, n là một số k lớn nhất sao cho m và n đều chia hết cho k.
  - Cách giải:
    - Cách 1: vét cạn
    - Cách 2: thuật toán Euclid: giả sử m>n ➔ USCLN (m, n) == USCLN (m mod n, n)
      - ✓Ví dụ:

        USCLN(108, 45) 108 chia cho 45 dư 18

        USCLN(45, 18) 45 chia cho 18 dư 9

        USCLN(18, 9) 18 chia cho 9 dư 0

        USCLN(9, 0) Kết quả USCLN của 108 và 45 là 9

Thiết kế giải thuật đệ quy

- Cài đặt thuật toán:

```
int USCLN(int m, int n){
    if(n==0) return m;
    else return USCLN(n, m%n);
}
```

Nhận xét

- Thông thường thay vì sử dụng lời giải đệ quy cho một bài toán, ta có thể thay thế bằng lời giải không đệ quy (khử đệ quy) bằng phương pháp lặp.

Ưu điểm	Khuyết điểm
Thuận lợi cho việc biểu diễn bài toán	Có khi không được tối ưu về thời gian
Gọn (đối với chương trình)	Có thể gây tốn bộ nhớ

- ➔ cố tránh sử dụng thủ tục đệ quy nếu thấy không cần thiết

Các dạng của đệ quy

- Chia để trị
- Quay lui

Giải thuật đệ quy dạng chia để trị

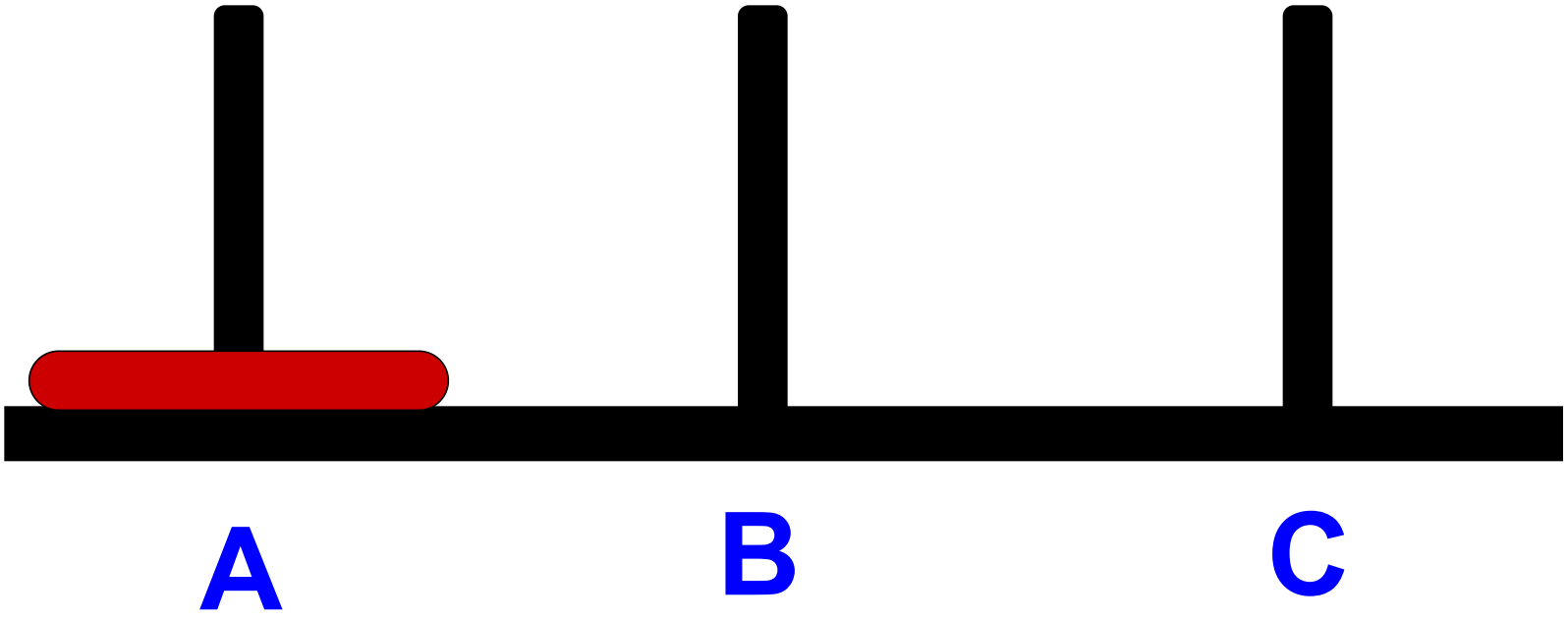
- Ý tưởng:
  - Phân chia bài toán thành 2 hoặc nhiều bài toán con có dạng tương tự và lần lượt giải quyết từng bài toán con này.
  - Các bài toán con này được coi là dạng đơn giản hơn của bài toán ban đầu.
- Ví dụ:
  - Quick sort
  - Bài toán tháp Hà Nội

Bài toán tháp Hà Nội

- Phát biểu bài toán: có 3 chiếc cọc và bộ n đĩa. Các đĩa này có kích thước khác nhau, các điã đều có lỗ để xuyên chúng qua đầu cọc.
- Ban đầu các đĩa đều nằm trên 1 cọc, trong đó đĩa nhỏ sẽ nằm trên đĩa lớn.
- Yêu cầu: chuyển bộ n đĩa từ cọc ban đầu A sang cọc đích C sao cho vẫn đảm bảo nguyên tắc đĩa nhỏ trên đĩa lớn dưới.

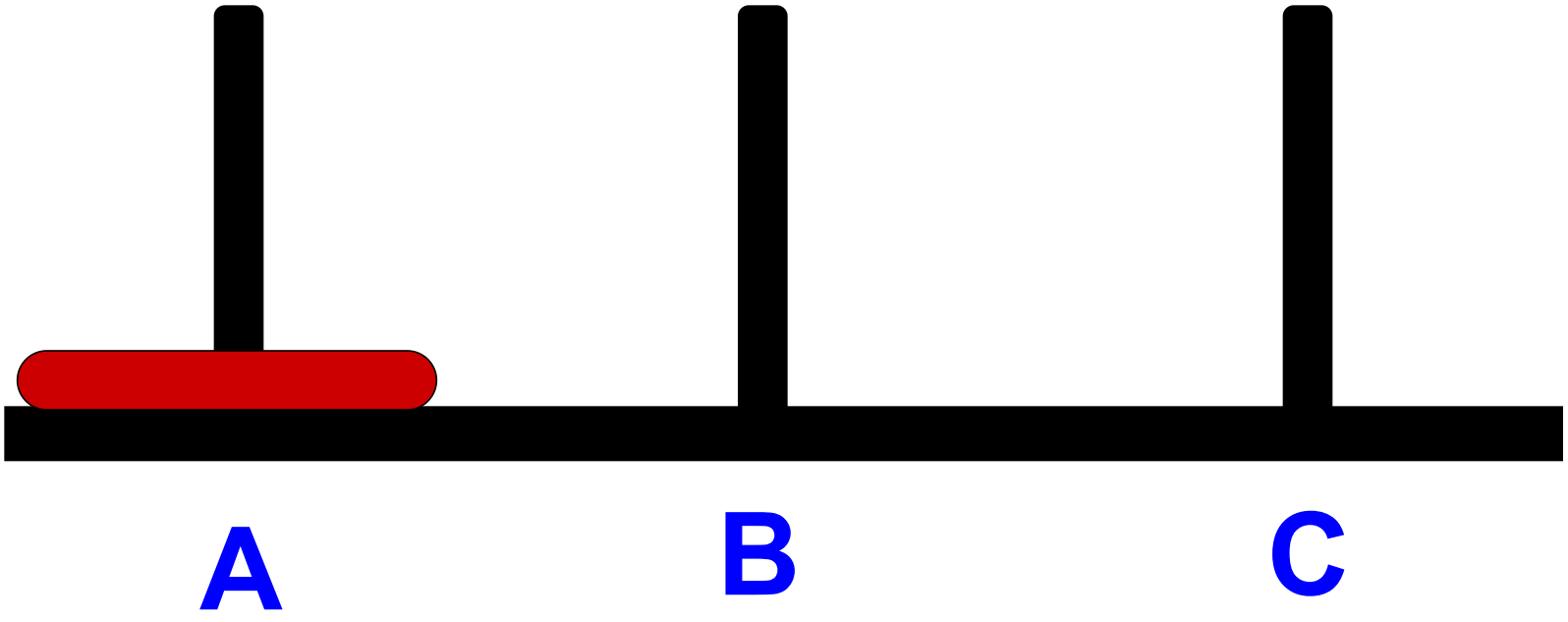
Giải một số bài tập đệ quy

1 đĩa



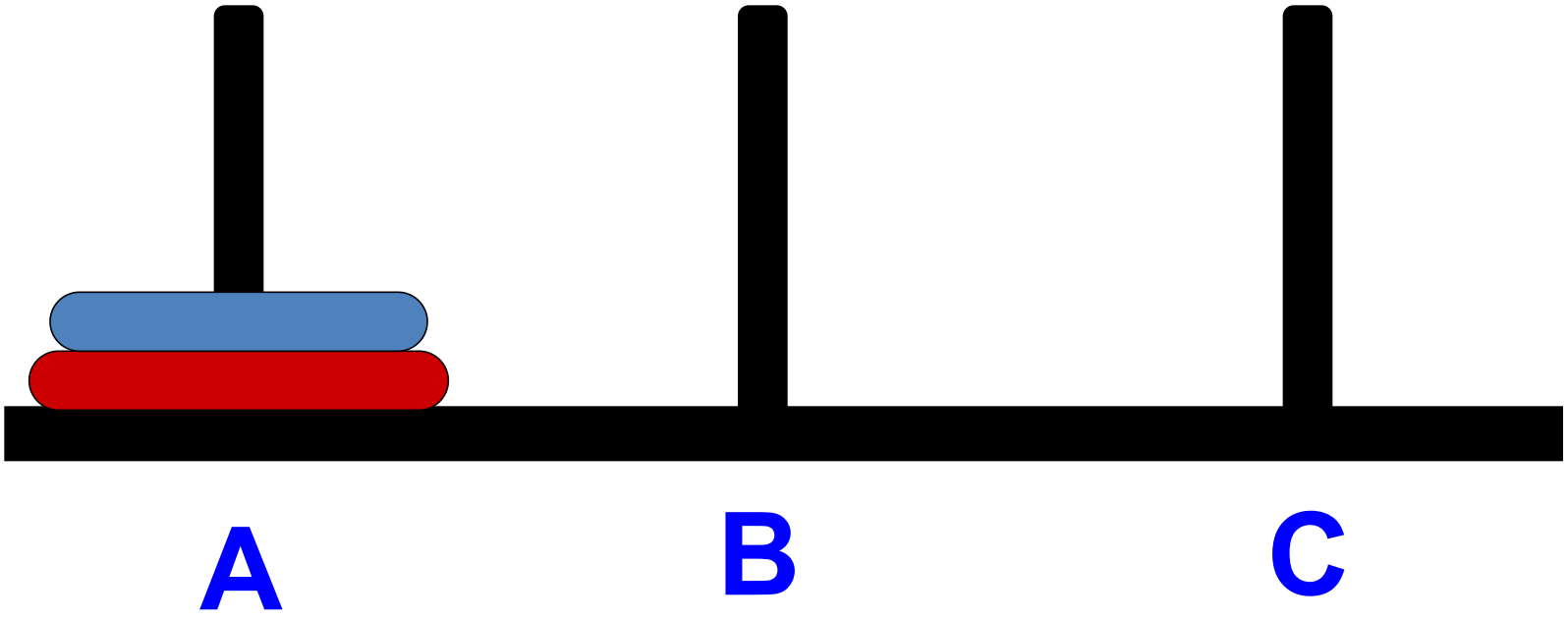
Giải một số bài tập đệ quy

1 đĩa



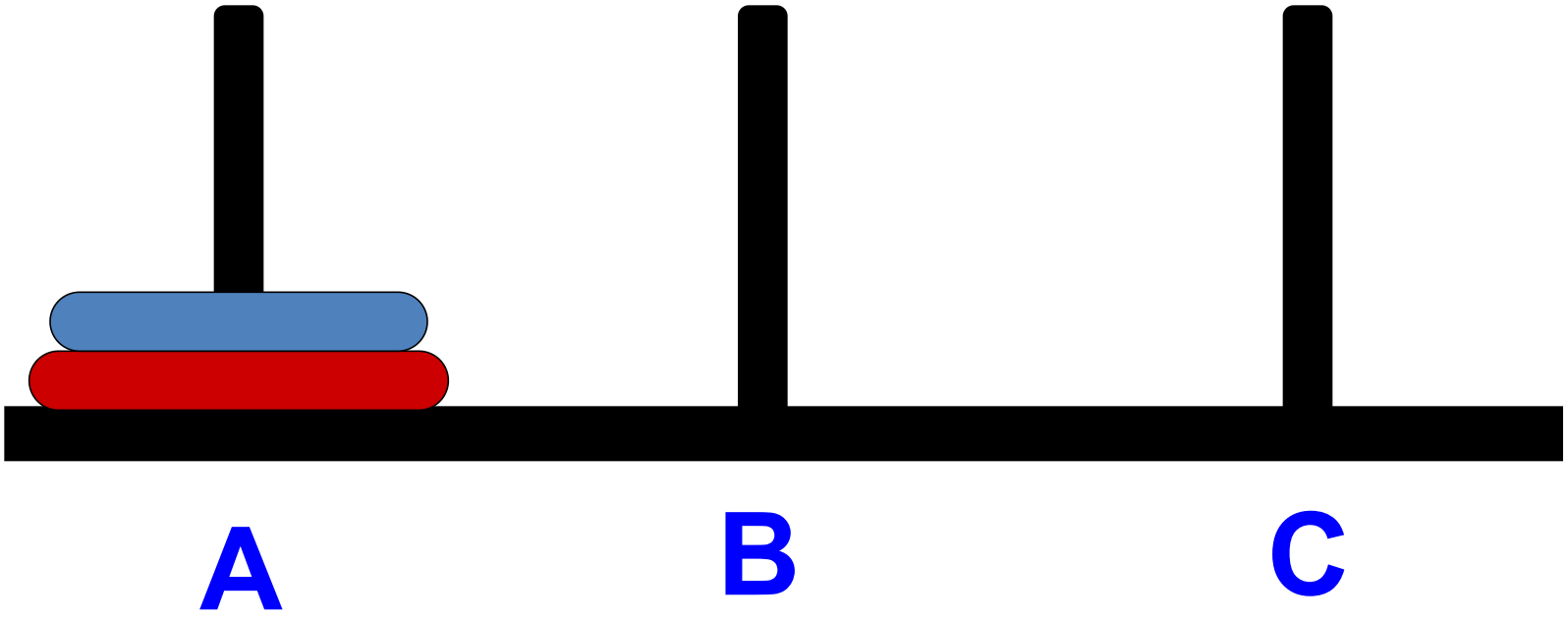
Giải một số bài tập đệ quy

2 đĩa



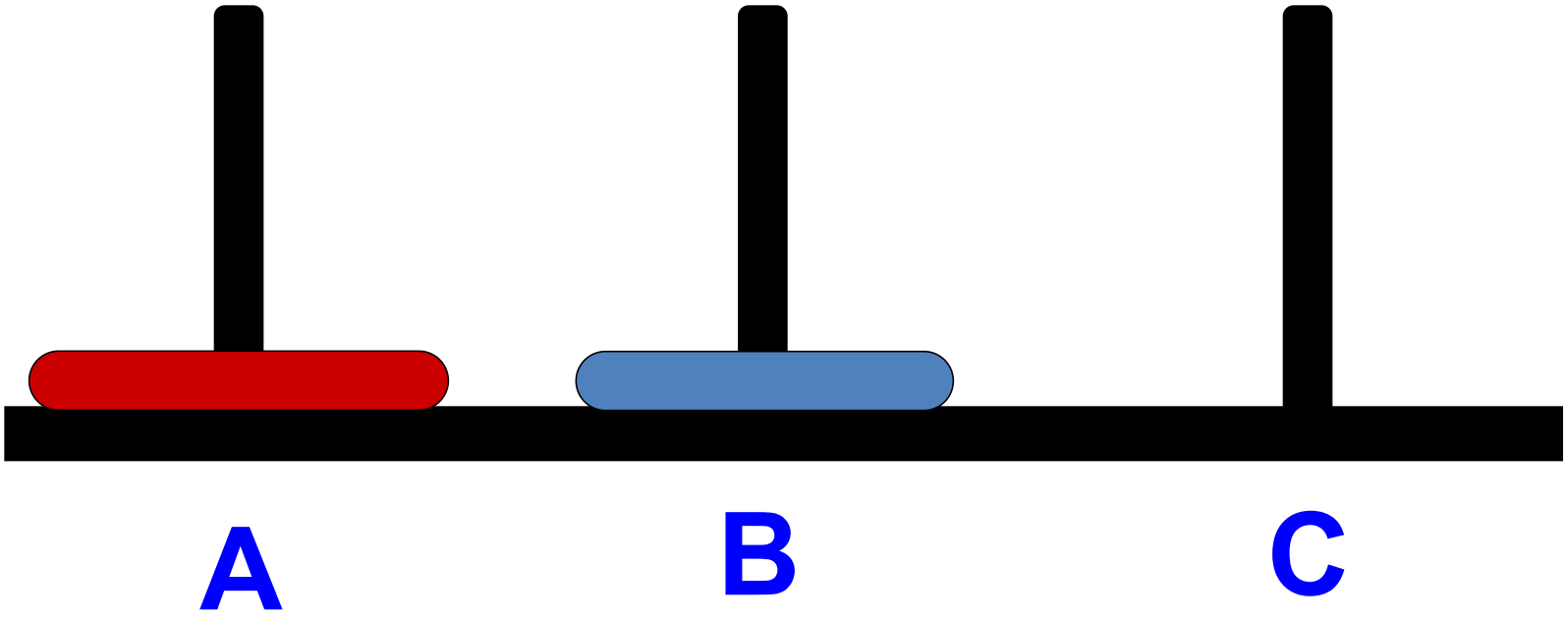
Giải một số bài tập đệ quy

2 đĩa



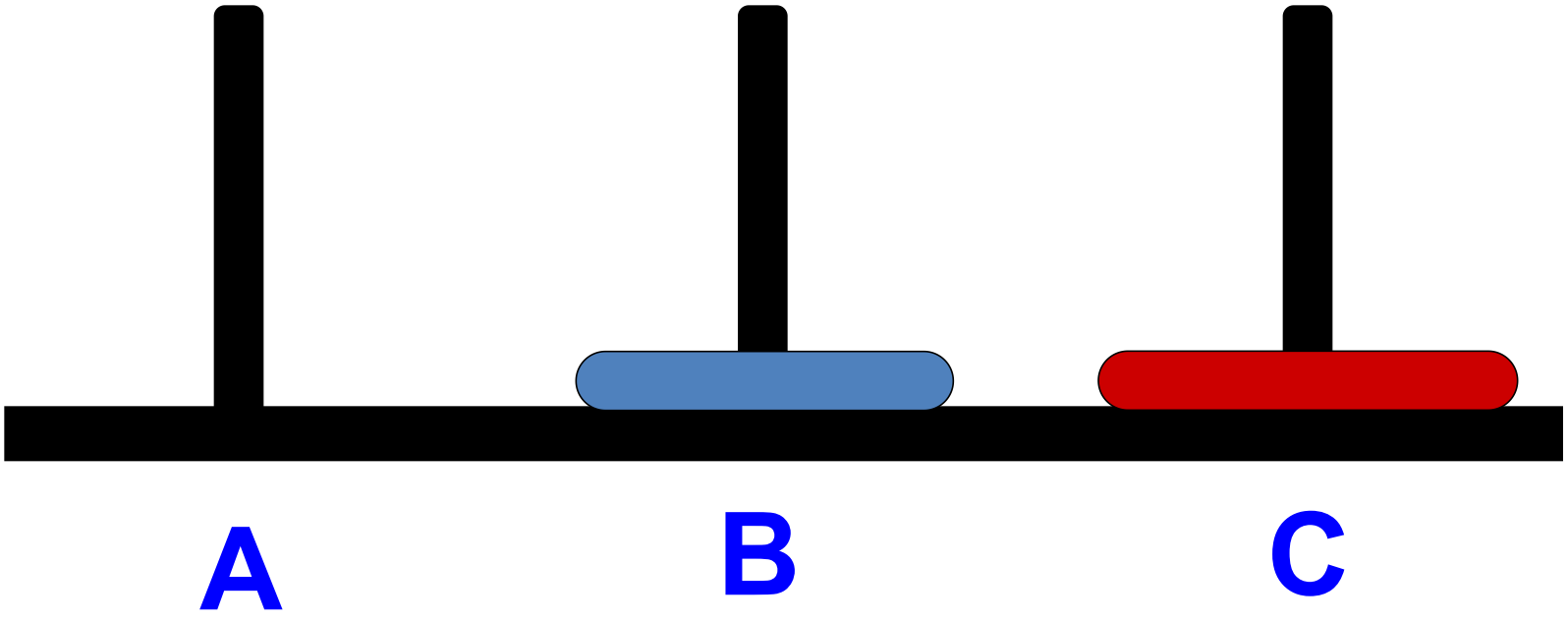
Giải một số bài tập đệ quy

2 đĩa

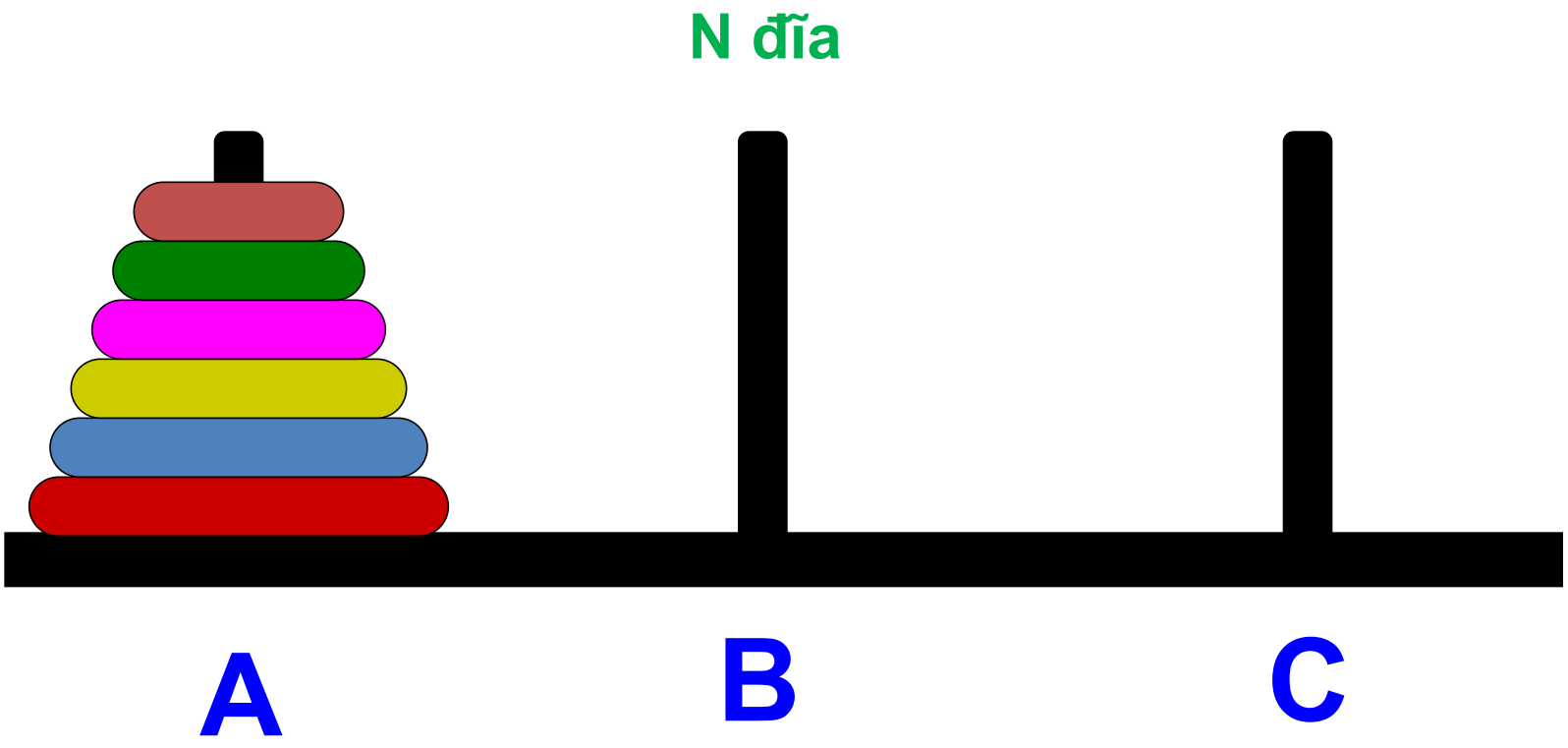


Giải một số bài tập đệ quy

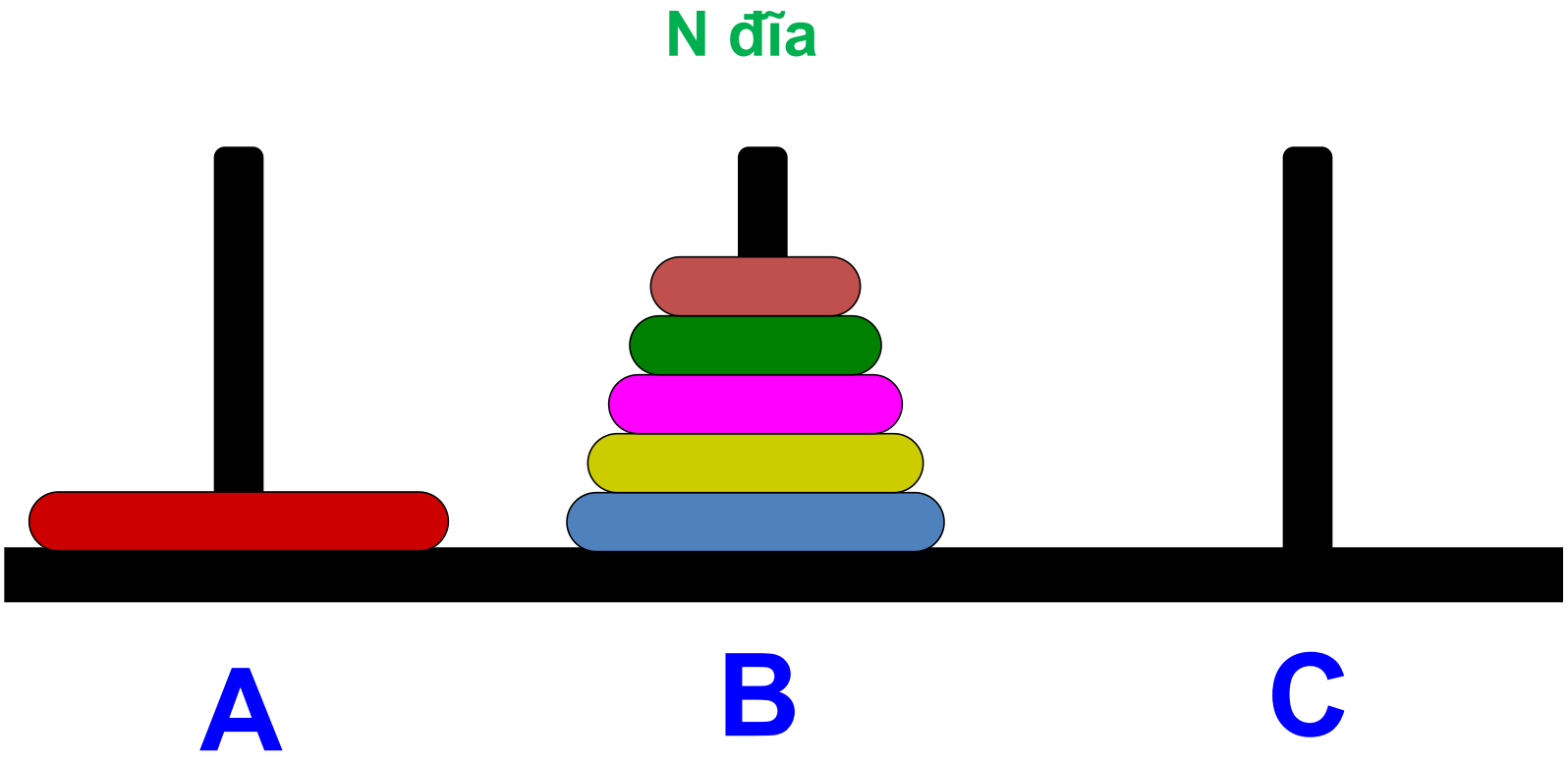
2 đĩa



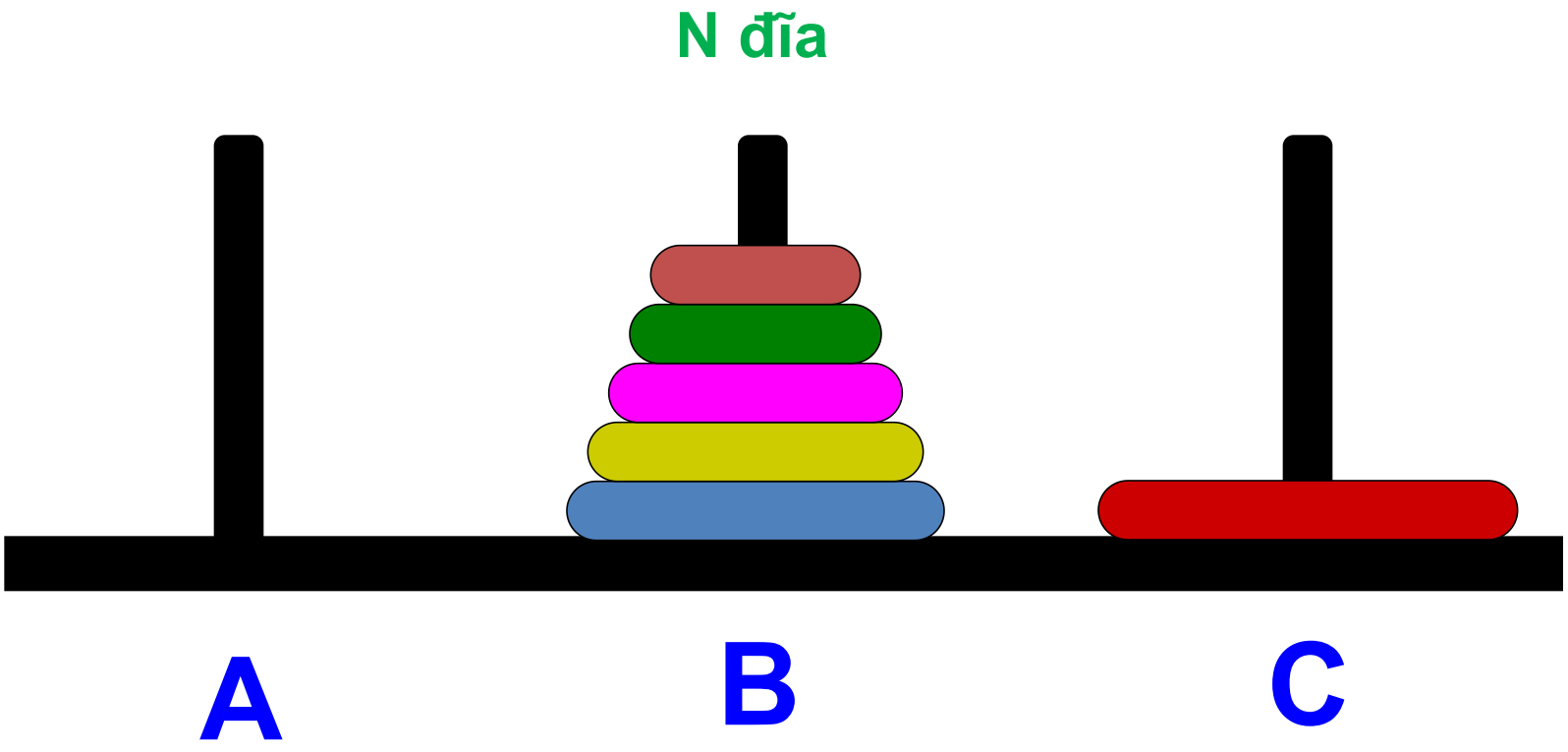
Giải một số bài tập đệ quy



Giải một số bài tập đệ quy



Giải một số bài tập đệ quy



Bài toán tháp Hà Nội

- Nhận xét:
  - Ở đây ta thấy bài toán chuyển n cọc thành bài toán đơn giản hơn là chuyển n-1 cọc.
  - Điểm dừng của thuật toán là n = 1, ta chuyển thẳng cọc này từ cọc ban đầu sang cọc đích.
  - Bài toán chuyển n đĩa được chia làm 2 bài toán nhỏ hơn là chuyển n-1 đĩa:
    - Lần 1: chuyển n-1 đĩa từ cọc A sang cọc trung gian B.
    - Lần 2: chuyển n-1 đĩa từ cọc B sang cọc đích C.

Bài toán tháp Hà Nội

- Cài đặt đệ quy

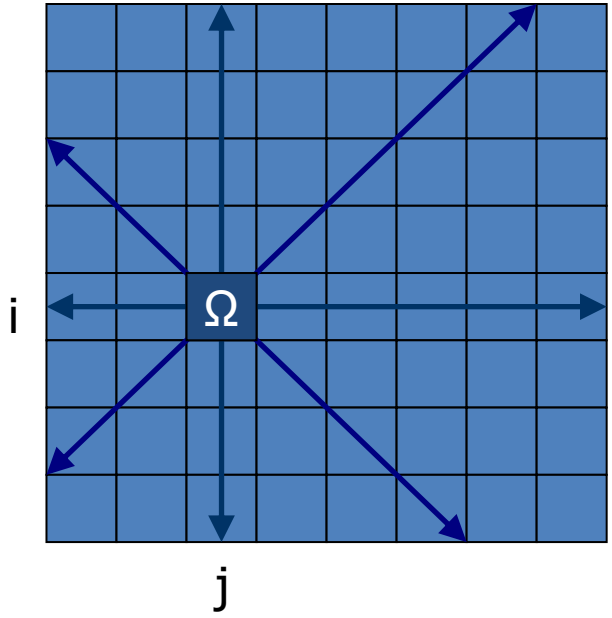
```
public void chuyen(int n, char a, char b) {
        System.out.println("Chuyen dia thu " + n + " tu coc " + a + " sang
    coc " + b);
    }

    public void thapHaNoi(int n, char a, char c, char b) {
        if (n == 1) {
            chuyen(1, a, c);
        } else {
            thapHaNoi(n-1, a, c, b);
            chuyen(n, a, c);
            thapHaNoi(n-1, b, a, c);
        }
    }
```

Giải thuật đệ quy dạng quay lui

- Ý tưởng
  - Các thuật toán thường có 1 qui tắc tính toán nào đó nhưng có những vấn đề không tuân theo qui tắc nào hết➔ dùng thử sai (trial-and-error) để giải quyết.
  - Đặc điểm của dạng toán này là:
    - ➢Không thể dùng biện pháp phân tích
    - ➢Phải dùng pp tính toán thủ công với sự kiên trì & độ chính xác cao.
  - Quá trình thử sai sẽ được thực hiện trên các bài toán đơn giản hơn.
    - ➢Thường được mô tả dưới dạng đệ quy và liên quan đến việc giải quyết 1 số hữu hạn bài toán con.
- Ví dụ:
  - Tám quân hậu
  - Mã đi tuần

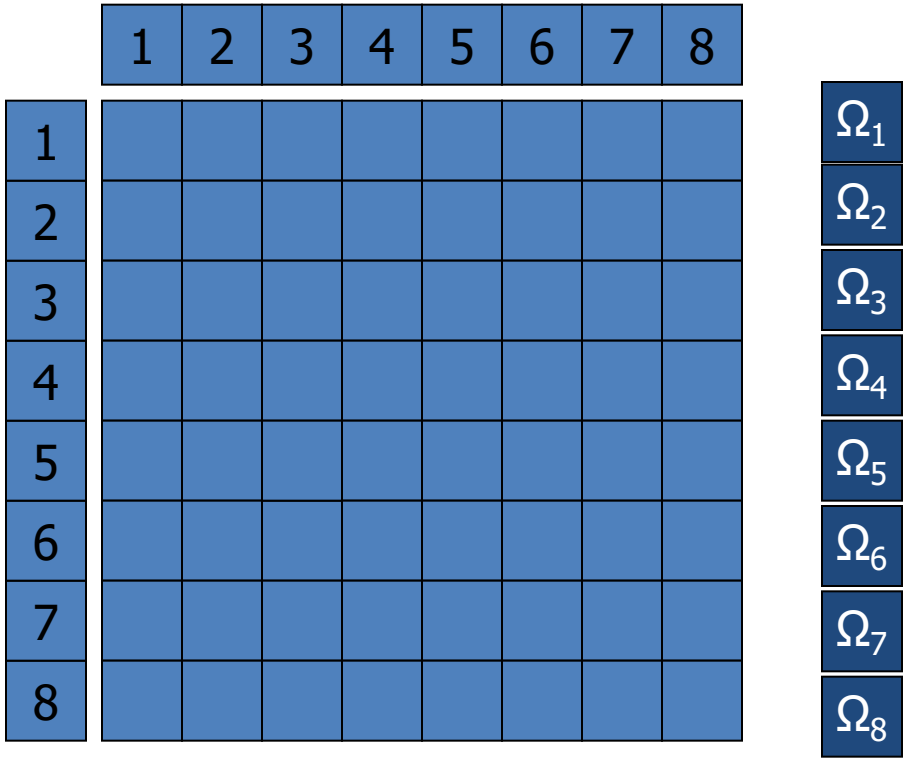
Bài toán 8 quân hậu

- Là một ví dụ nổi tiếng của việc sử dụng phương pháp thử sai & thuật toán quay lui.
- Phát biểu: Tìm cách đặt 8 quân hậu trên 1 bàn cờ mà không có 2 quân hậu nào có thể ăn được nhau.



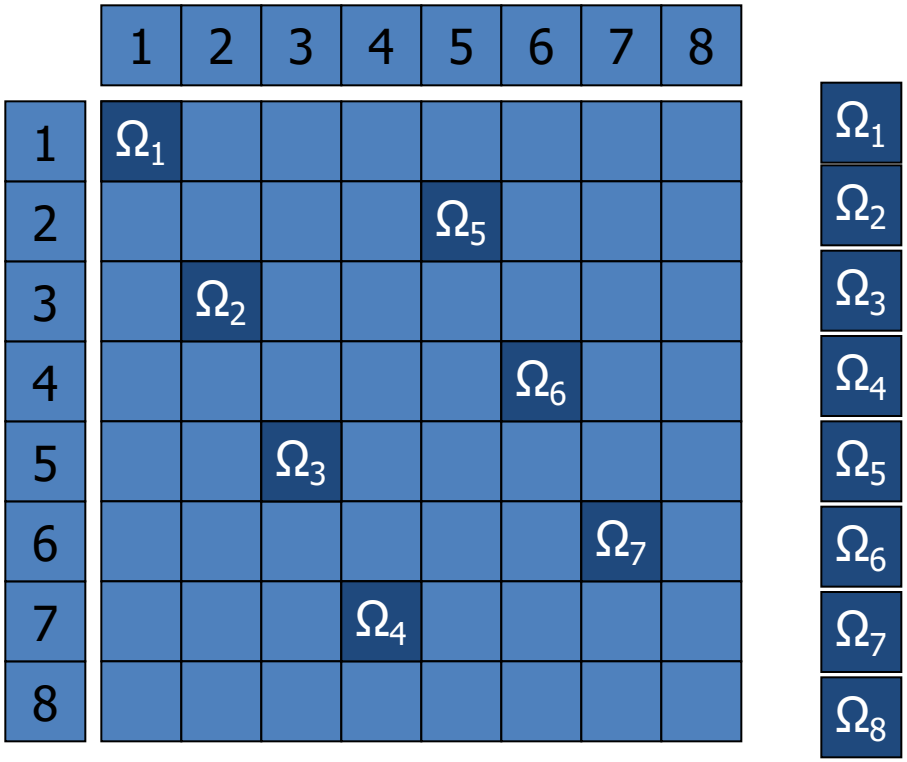
Bài toán 8 quân hậu

- Bài toán 8 quân hậu (mô phỏng)



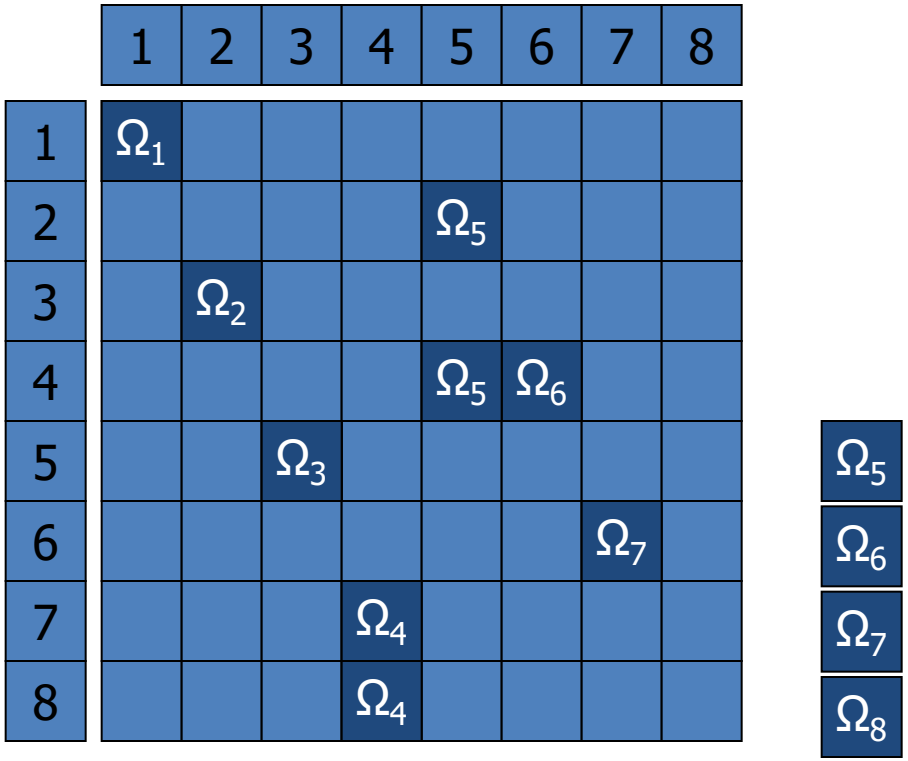
Bài toán 8 quân hậu

- Bài toán 8 quân hậu (mô phỏng)



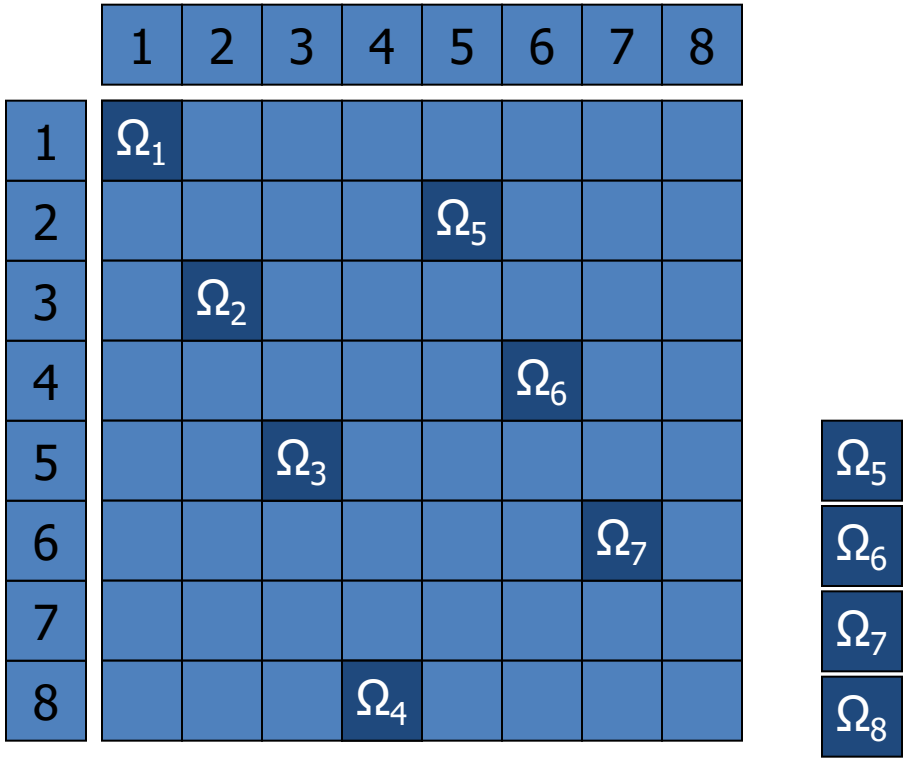
Bài toán 8 quân hậu

- Bài toán 8 quân hậu (mô phỏng)



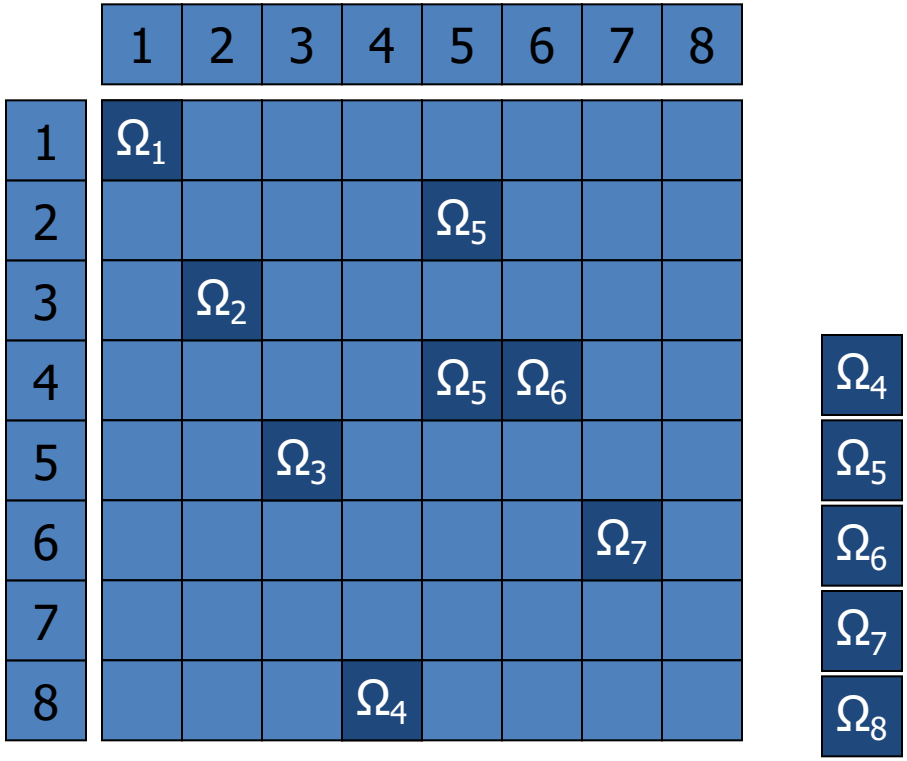
Bài toán 8 quân hậu

- Bài toán 8 quân hậu (mô phỏng)



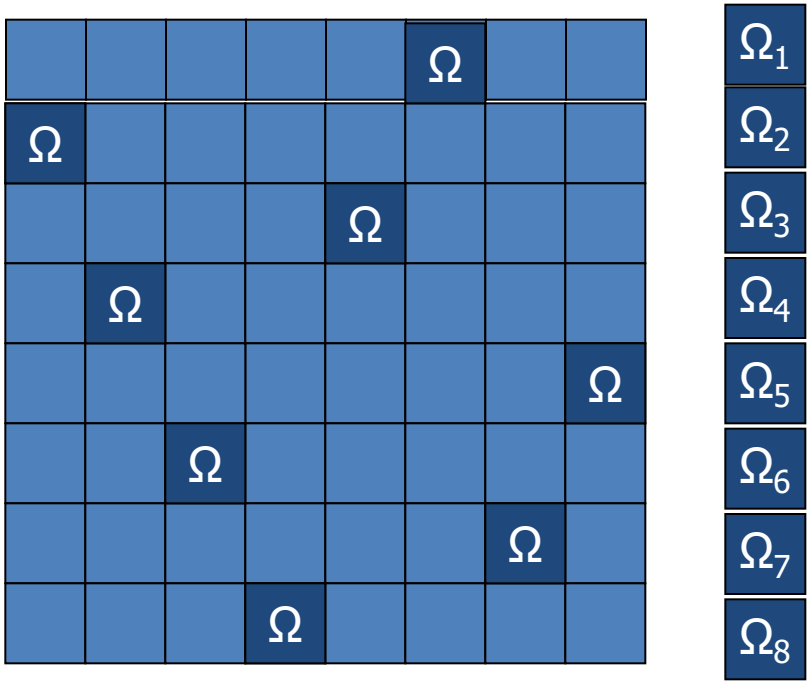
Bài toán 8 quân hậu

- Bài toán 8 quân hậu (mô phỏng)



Bài toán 8 quân hậu

- Bài toán 8 quân hậu



Bài toán 8 quân hậu

```
void datHau(int i){
    Khởi tạo danh sách các vị trí có thể đặt quân hậu tiếp theo;
    do{
        Lựa chọn vị trí đặt quân hậu tiếp theo;
        if Vị trí đặt an toàn{
                Đặt hậu;
                if i<8{
                datHau(i+1);
                if Không thành công➔Bỏ hậu đã đặt ra khỏi vị trí
                }
        }
    } while(Không thành công)&&(Vẫn còn lựa chọn)
}
```

Qui hoạch động

- Bài toán đặt ra: một số bài toán có thể chia nhỏ thành nhiều bài toán con, nhưng thời gian thu được sẽ tăng theo số mũ ➔ thuật toán trở nên vô giá trị.
- ➔ Solution: Qui hoạch động

Qui hoạch động

- Trên thực tế, việc chia thành các bài toán con thường chỉ chiếm thời gian là đa thức
- Trong trường hợp này một bài toán con sẽ được lặp lại nhiều lần trong quá trình tìm kiếm lời giải.
- Để khỏi mất thời gian mỗi khi giải quyết các bài toán con, các bạn sẽ lưu trữ các lời giải này để tra cứu về sau mỗi khi cần đến.
- Công việc này sẽ đòi hỏi độ phức tạp thuật toán là đa thức.

Qui hoạch động

- *Qui hoạch động*: Chúng ta sẽ lưu giữ tất cả các lời giải của các bài toán con lại không cần biết rằng chúng có được dùng lại nhiều lần về sau hay không, không quan tâm đến việc các lời giải này có cần thiết cho lời giải của bài toán chính của chúng ta hay không.

Tóm tắt

- Chương này chúng ta cần nắm vững các vấn đề sau:
  - Định ngĩa bằng đệ quy
  - Chương trình đệ quy
  - Ưu điểm và khuyết điểm của đệ quy
  - Khi nào không nên sử dụng đệ quy
  - Các dạng đệ quy và ví dụ:
    - Chia để trị
    - Quay lui
    - Qui hoạch động

HỎI ĐÁP