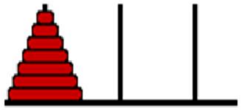
Chương 2

Thuật toán đệ qui

Bài giảng của
PGS.TS. NGUYỄN ĐỨC NGHĨA

Khoa học Máy tính
Đại học Bách khoa Hà nội



Nội dung

- **2.1. Khái niệm đệ qui**
- **2.2.** Thuật toán đệ qui
- **2.3.** Một số ví dụ minh hoạ
- **2.4.** Phân tích thuật toán đệ qui
- **2.5.** Đệ qui có nhớ
- **2.6.** Chứng minh tính đúng đắn của thuật toán đệ qui

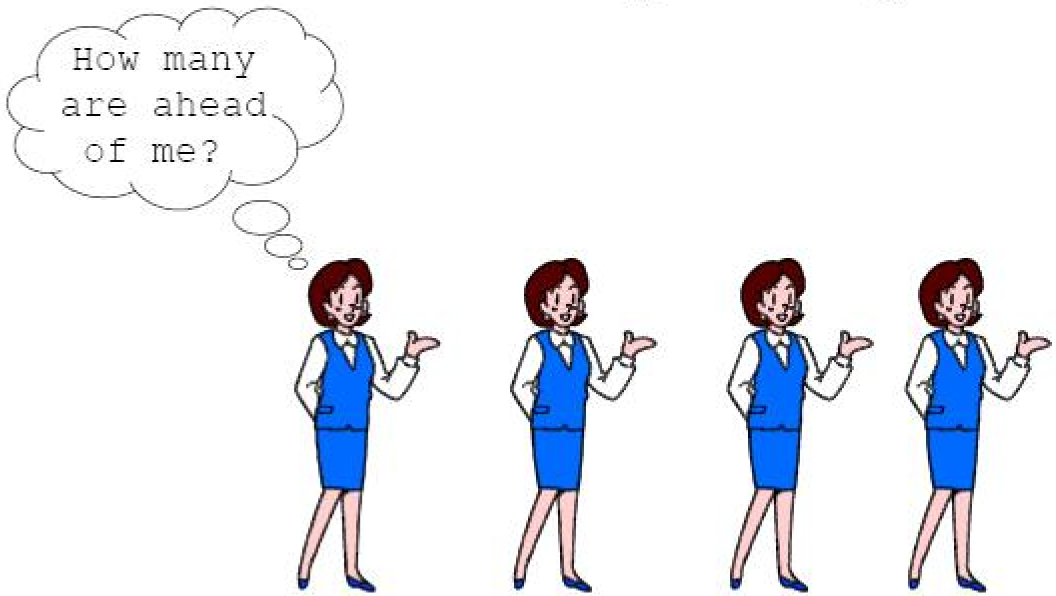
2.1. Khái niệm đệ qui

- 2.1.1 Khái niệm đệ qui
- 2.1.2 Thuật toán đệ qui

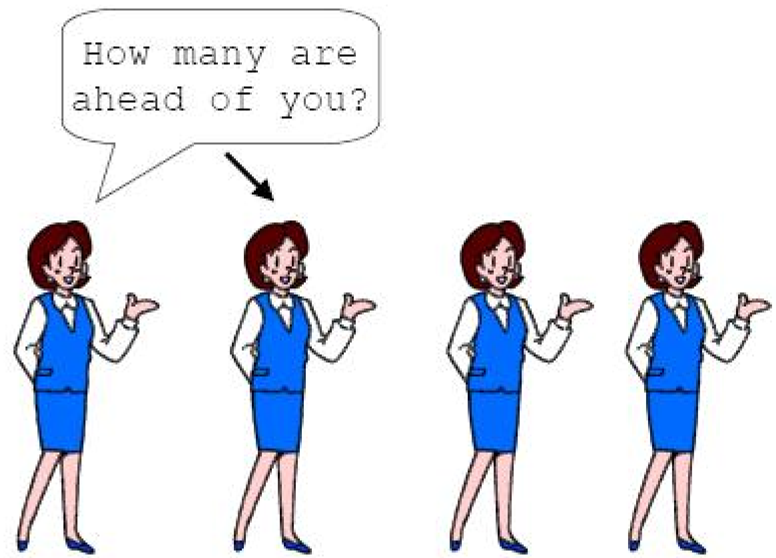
Khái niệm đệ qui

- Trong thực tế ta thường gặp những đối tượng bao gồm chính nó hoặc được định nghĩa dưới dạng của chính nó. Ta nói các đối tượng đó được xác định một cách đệ qui.
- **Ví dụ:**
  - Điểm quân số
  - Fractal
  - Các hàm được định nghĩa đệ qui
  - Tập hợp được định nghĩa đệ qui
  - Định nghĩa đệ qui của cây
  - ...

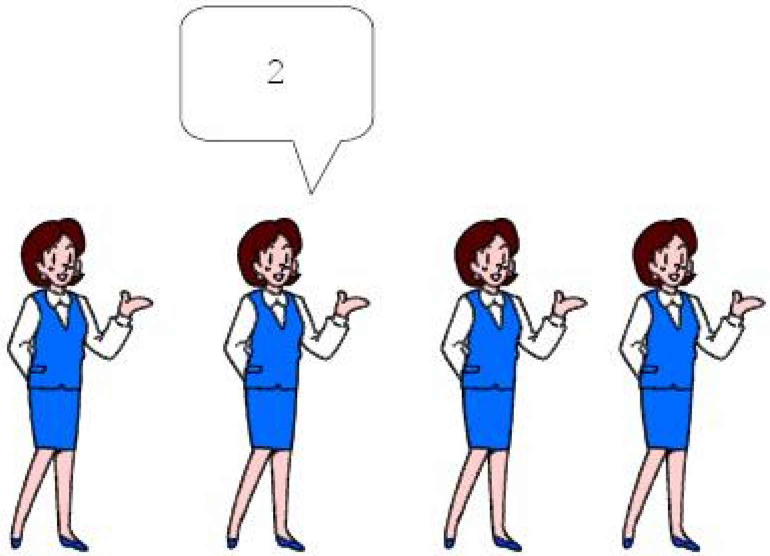
Đệ qui: Điểm quân



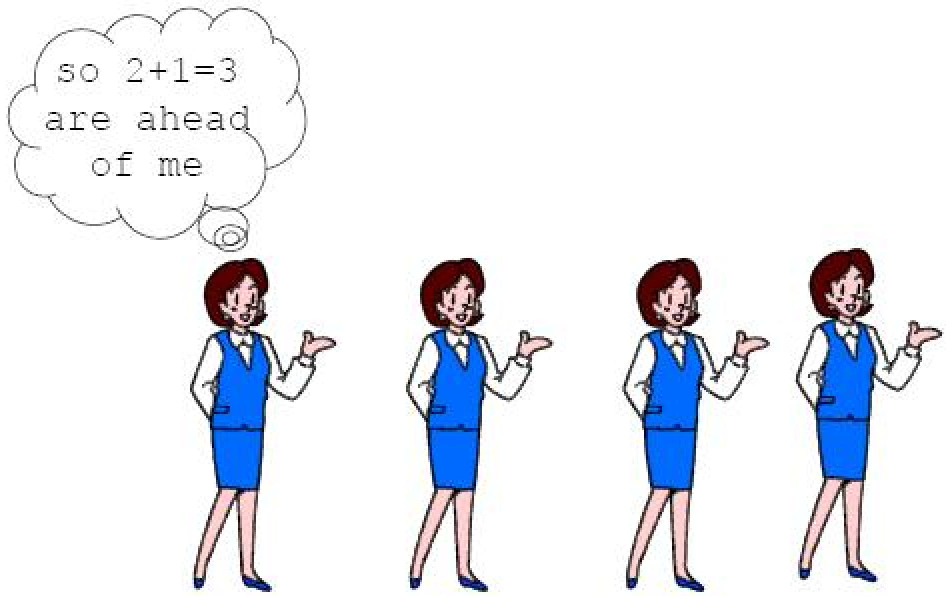
Đệ qui: Điểm quân



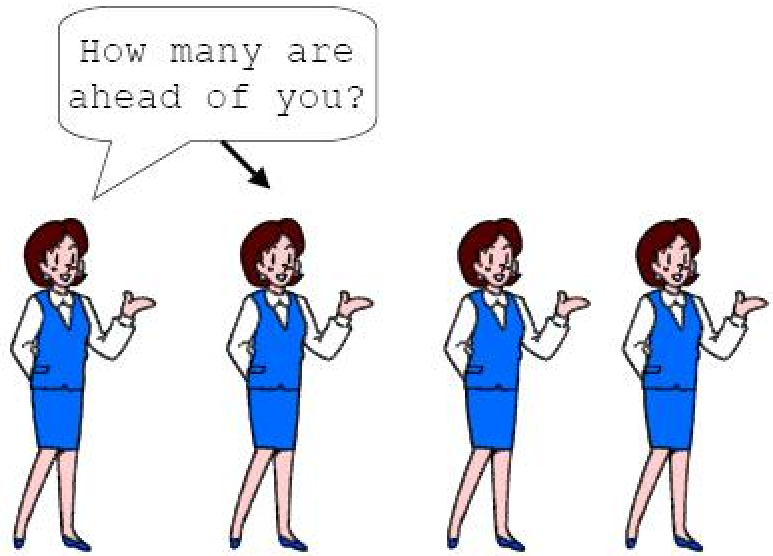
Đệ qui: Điểm quân



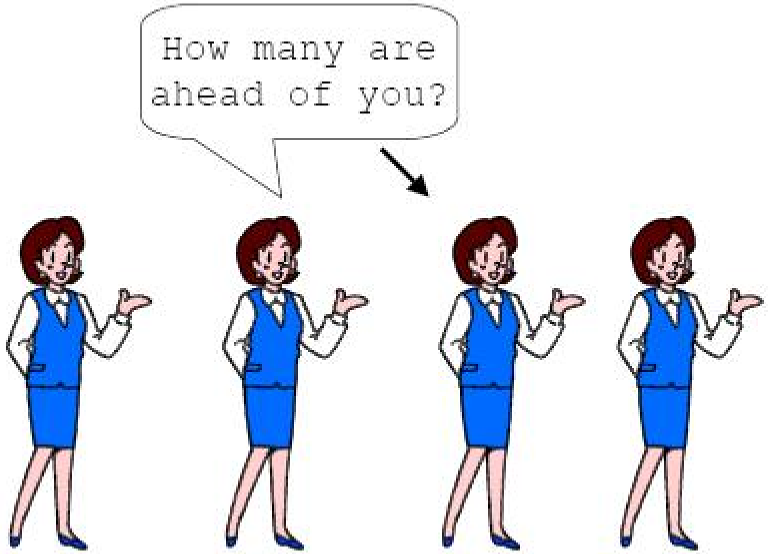
Đệ qui: Điểm quân



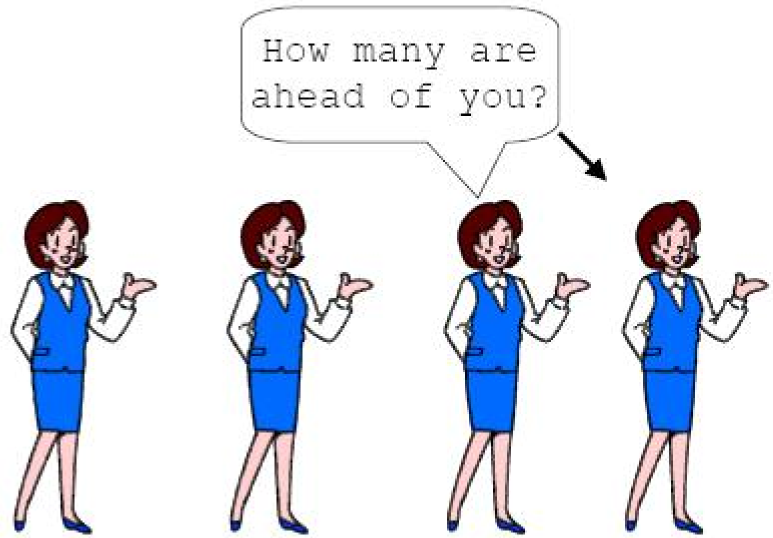
Đệ qui: Điểm quân



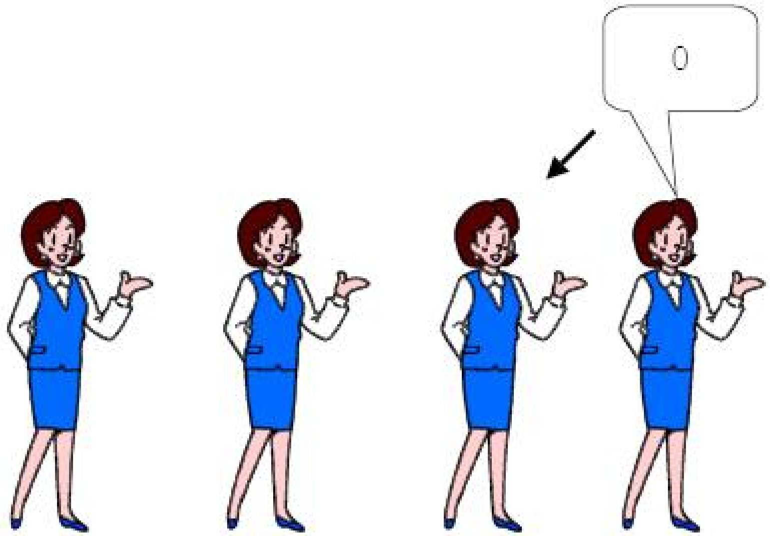
Đệ qui: Điểm quân



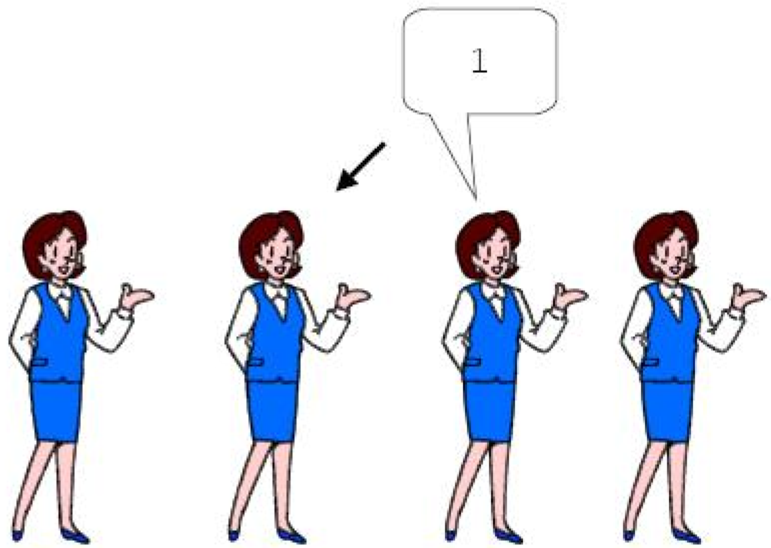
Đệ qui: Điểm quân



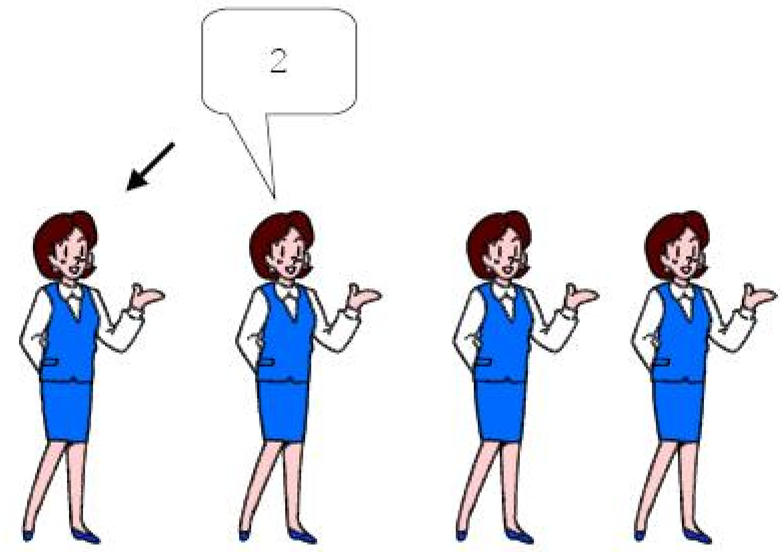
Đệ qui: Điểm quân



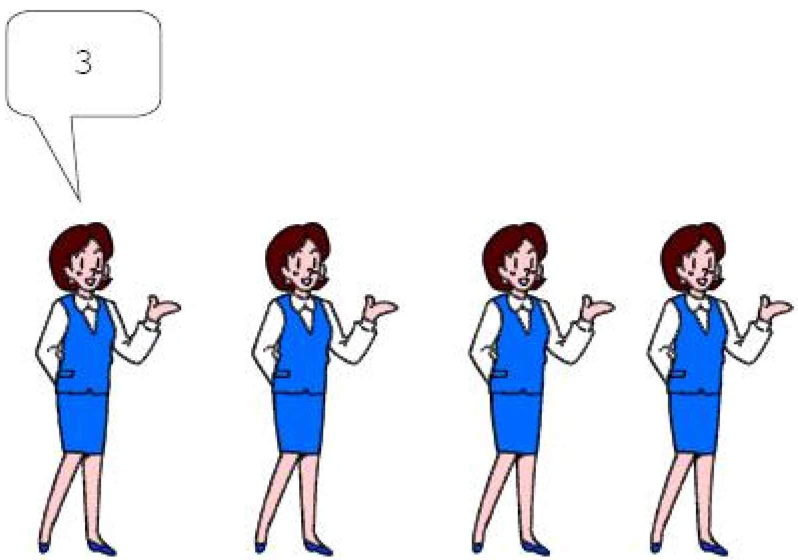
Đệ qui: Điểm quân



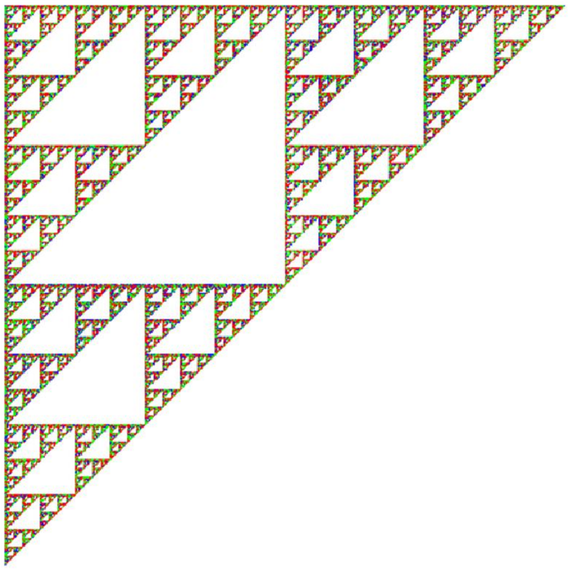
Đệ qui: Điểm quân



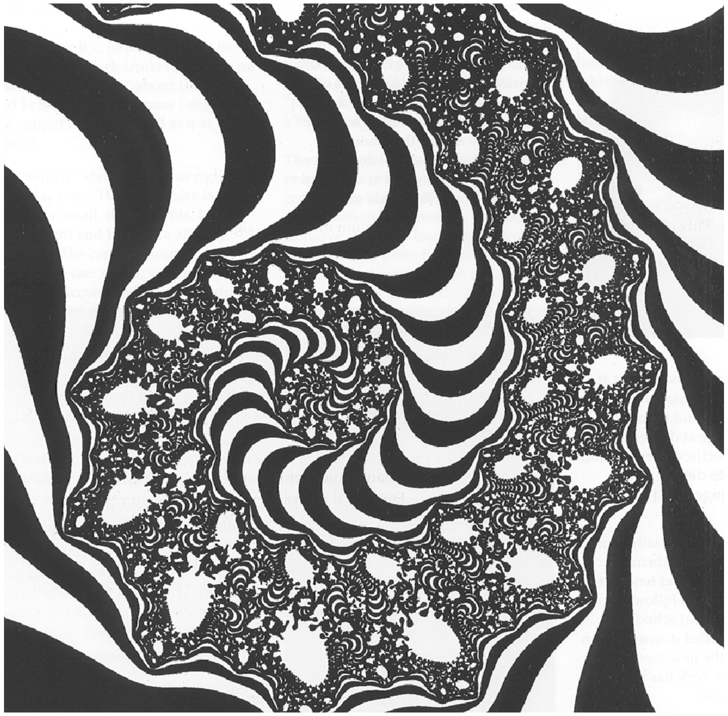
Đệ qui: Điểm quân



Fractals



fractals là ví dụ về hình ảnh được xây dựng một cách đệ qui (đối tượng lặp lại một cách đệ qui).



Hàm đệ qui (Recursive Functions)

Các hàm đệ qui được xác định phụ thuộc vào biến nguyên không âm *n* theo sơ đồ sau:

Bước cơ sở (Basic Step): *Xác định giá trị của hàm tại* $n=0$: $f(0)$.

Bước đệ qui (Recursive Step): *Cho giá trị của* $f(k)$, $k \le n$, *đưa ra qui tắc tính giá trị của* $f(n+1)$.

***Ví dụ* 1:**

$$f(0) = 3, \qquad n = 0$$

$$f(n+1) = 2f(n) + 3, \qquad n > 0$$

Khi đó ta có: $f(1) = 2 \times 3 + 3 = 9$, $f(2) = 2 \times 9 + 3 = 21$, ...

Hàm đệ qui (Recursive Functions)

Ví dụ 2: Định nghĩa đệ qui của $n!$:

$$f(0) = 1$$
$$f(n+1) = f(n) \times (n+1)$$

Để tính giá trị của hàm đệ qui ta thay thế dần theo định nghĩa đệ qui để thu được biểu thức với đối số càng ngày càng nhỏ cho đến tận điều kiện đầu.

Chẳng hạn:

đệ qui

$$5! = 5 \cdot 4! = 5 \cdot 4 \cdot 3! = 5 \cdot 4 \cdot 3 \cdot 2! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1! =$$
$$5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 \cdot 0! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 \cdot 1 = 120$$

điều kiện đầu

Hàm đệ qui (Recursive Functions)

***Ví dụ* 3:** Định nghĩa đệ qui của tổng $s_n = \sum_{k=1}^{n} a_k$

$$s_1 = a_1$$

$$s_n = s_{n\text{-}1} + a_n$$

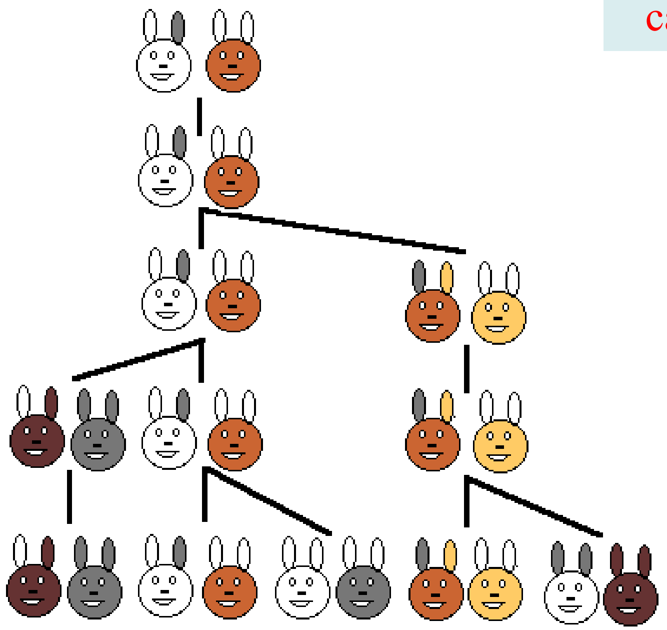
***Ví dụ* 4:** Dãy số Fibonacci

$$f(0) = 0, f(1) = 1,$$

$$f(n) = f(n\text{-}1) + f(n\text{-}2) \text{ với } n > 1$$

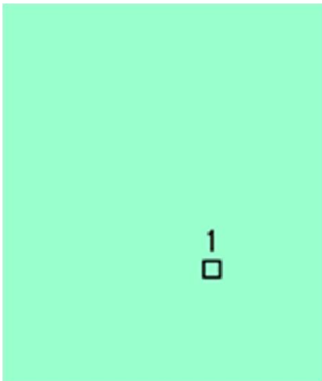
Fibonacci Numbers

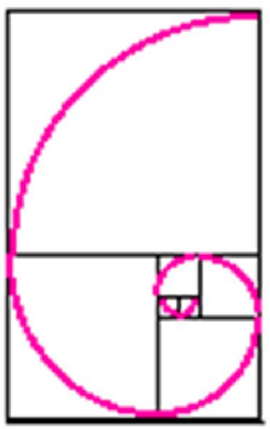
Sự phát triển của bày thỏ



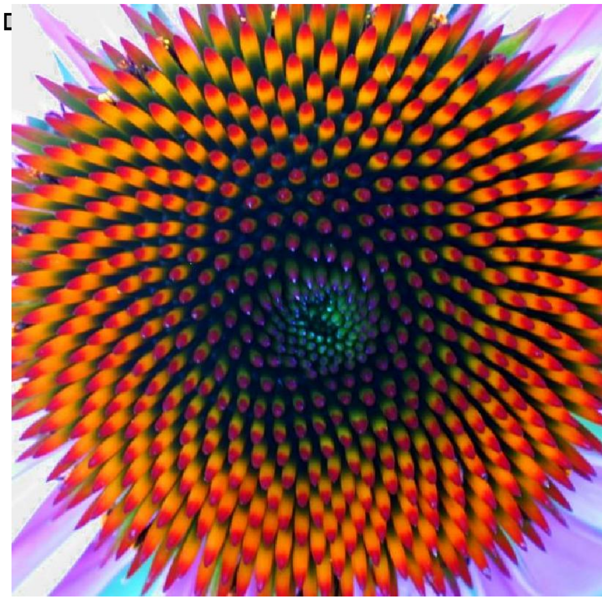
Số lượng cặp thỏ
1
1
2
3
5

Nói thêm về Fibonacci

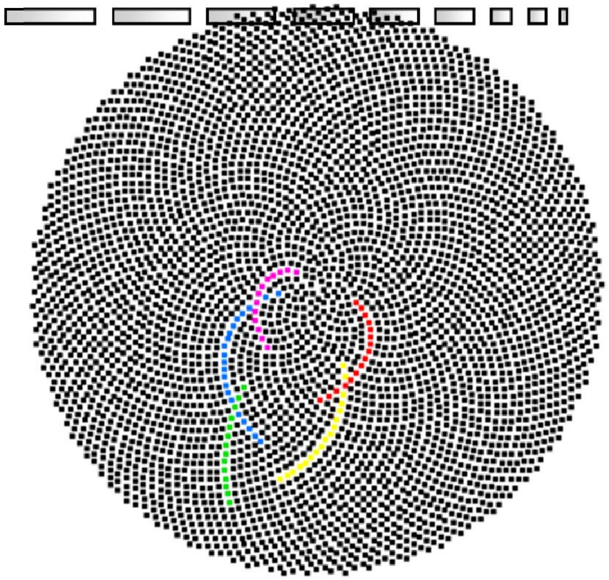


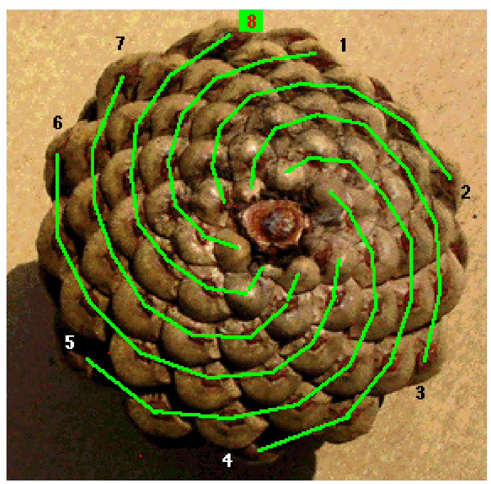


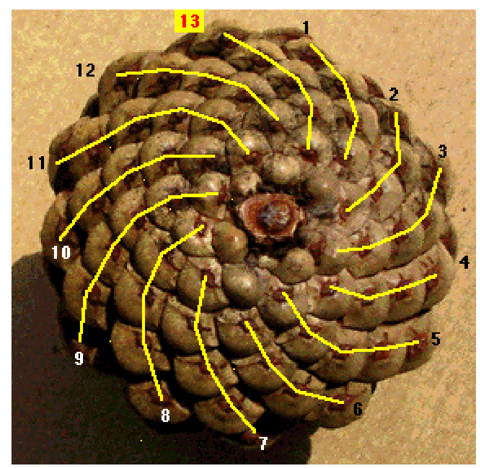
Số Fibonacci trong các cấu trúc tự nhiên



The left and right going spirals are neighboring Fibonacci numbers!



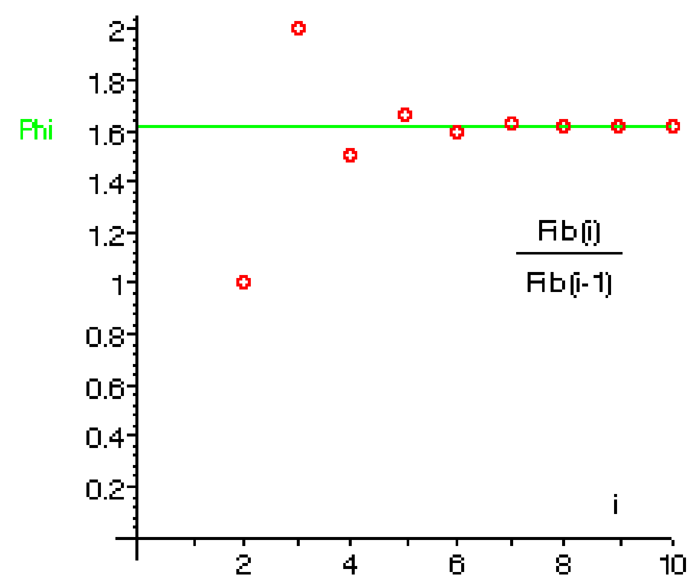




Tỷ lệ vàng (Golden Section)

$$\frac{x+y}{x}=\frac{x}{y}=\frac{1}{2}(1+\sqrt{5})=1.6180=Phi$$

$$\lim_{n\to\infty}=\frac{F(n+1)}{F(n)}=Phi$$



Tập hợp được xác định đệ qui (Recursively defined sets)

Tập hợp có thể xác định một cách đệ qui theo sơ đồ tương tự như hàm đệ qui:

Basis Step: *định nghĩa tập cơ sở* (chẳng hạn tập rỗng).

Recursive Step : *Xác định qui tắc để sản sinh tập mới từ các tập đã có.*

Ví dụ:

Basis Step: 3 là phần tử của *S*.

Recursive Step: nếu x thuộc S và y thuộc S thì $x+y$ thuộc S.

3

3+3=6

3+6 = 9 & 6+6=12

...

Định nghĩa đệ qui của xâu

Giả sử Σ = bảng chữ cái (alphabet).
Tập Σ^* các xâu (strings) trên bảng chữ cái Σ được định nghĩa đệ qui như sau:

Basic step: xâu rỗng là phần tử của Σ^*
Recursive step: nếu w thuộc Σ^* và x thuộc $\Sigma \Rightarrow$ wx thuộc Σ^*

Ví dụ: Tập các xâu nhị phân thu được khi $\Sigma=\{0,1\}$:
1) xâu rỗng
2) 0 & 1
3) 00 & 01 & 10 & 11
4) ...

Độ dài của xâu

- Định nghĩa đệ qui độ dài length(w) của xâu $w \in \Sigma^*$
  - (B) length(λ) = 0
  - (R) Nếu $w \in \Sigma^*$ và $x \in \Sigma$ thì length(wx) = length(w)+1.
- Định nghĩa đệ qui của tập các xâu nhị phân độ dài chẵn.
  - (B) $\lambda \in S$ (λ là xâu rỗng)
  - (R) Nếu $b \in S$ thì 0b0, 0b1, 1b0, 1b1 $\in S$.

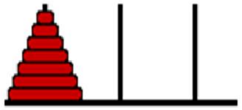
Định nghĩa đệ qui và Qui nạp

- Định nghĩa đệ qui và Qui nạp toán học có những nét tương đồng và là bổ sung cho nhau. Định nghĩa đệ qui thường giúp cho chứng minh bằng qui nạp các tính chất của các đối tượng được định nghĩa đệ qui. Ngược lại, các chứng minh bằng qui nạp toán học thường là cơ sở để xây dựng các thuật toán đệ qui để giải quyết nhiều bài toán.

Chứng minh bằng qui nạp toán học thường bao gồm hai phần:

1. **Bước cơ sở qui nạp** – giống như **bước cơ sở** trong định nghĩa đệ qui
2. **Bước chuyển qui nạp** – giống như **bước đệ qui**

Nội dung



2.1. Khái niệm đệ qui

2.2. Thuật toán đệ qui

2.3. Một số ví dụ minh hoạ

2.4. Phân tích thuật toán đệ qui

2.5. Đệ qui có nhớ

2.6. Chứng minh tính đúng đắn của thuật toán đệ qui

Định nghĩa

- ***Thuật toán đệ qui là thuật toán tự gọi đến chính mình với đầu vào kích thước nhỏ hơn.***
- Việc phát triển thuật toán đệ qui là thuận tiện khi cần xử lý với các đối tượng được định nghĩa đệ qui (chẳng hạn: tập hợp, hàm, cây, ...trong các ví dụ nêu trong mục trước).
- Các thuật toán được phát triển dựa trên phương pháp chia để trị thông thường được mô tả dưới dạng các thuật toán đệ qui (xem ví dụ mở đầu)
- Các ngôn ngữ lập trình cấp cao thường cho phép xây dựng các hàm (thủ tục) đệ qui, nghĩa là trong thân của hàm (thủ tục) có chứa những lệnh gọi đến chính nó. Vì thế, khi cài đặt các thuật toán đệ qui người ta thường xây dựng các hàm (thủ tục) đệ qui.

Cấu trúc của thuật toán đệ qui

- Thuật toán đệ qui thường có cấu trúc sau đây:

Thuật toán RecAlg(input)

begin

    if (kích thước của input là nhỏ nhất) **then**

        Thực hiện Bước cơ sở /* giải bài toán kích thước đầu vào nhỏ nhất */

    else

        RecAlg(input với kích thước nhỏ hơn); /* bước đệ qui */

        /* có thể có thêm những lệnh gọi đệ qui */

        Tổ hợp lời giải của các bài toán con để thu được **lời_giải;**

        return (lời_giải)

    endif

end;

Thuật toán chia để trị
(Divide and conquer)

- Thuật toán chia để trị là một trong những phương pháp thiết kế thuật toán cơ bản, nó bao gồm 3 thao tác
- Chia (Divide) bài toán cần giải ra thành một số bài toán con
  – Bài toán con có kích thước nhỏ hơn và có cùng dạng với bài toán cần giải
- Trị (Conquer) các bài toán con
  – Giải các bài toán con một cách đệ qui.
  – Bài toán con có kích thước đủ nhỏ sẽ được giải trực tiếp.
- Tổ hợp (Combine) lời giải của các bài toán con
  – Thu được lời giải của bài toán xuất phát.

Thuật toán chia để trị

- Sơ đồ của phương pháp có thể trình bày trong thủ tục đệ qui sau đây:

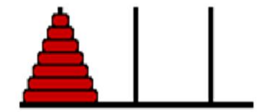
```
procedure D-and-C (n);
begin
    if  n ≤ n0 then
            Giải bài toán một cách trực tiếp
    else
    begin
        Chia bài toán thành a bài toán con kích thốc n/b;
        for (mỗi bài toán trong a bài toán con) do D-and-C(n/b);
        Tổng hợp lời giải của a bài toán con để thu đọc lời giải của bài toán gốc;
    end;
end;
```

Thuật toán chia để trị

- Các thông số quan trọng của thuật toán:
  - n_0 - kích thước nhỏ nhất của bài toán con (còn gọi là neo đệ qui). Bài toán con với kích thước n_0 sẽ được giải trực tiếp.
  - a - số lượng bài toán con cần giải
  - b - liên quan đến kích thước của bài toán con được chia

Ví dụ: Sắp xếp trộn (Merge Sort)

- **Bài toán:** Cần sắp xếp mảng A[1 .. n]:
- **Chia (Divide)**
  – Chia dãy gồm *n* phần tử cần sắp xếp ra thành 2 dãy, mỗi dãy có *n*/2 phần tử
- **Trị (Conquer)**
  – Sắp xếp mỗi dãy con một cách đệ qui sử dụng ***sắp xếp trộn***
  – Khi dãy chỉ còn một phần tử thì trả lại phần tử này
- **Tổ hợp (Combine)**
  – Trộn (Merge) hai dãy con được sắp xếp để thu được dãy được sắp xếp gồm tất cả các phần tử của cả hai dãy con



Nội dung

2.1. Khái niệm đệ qui

2.2. Thuật toán đệ qui

2.3. Một số ví dụ minh hoạ

2.4. Phân tích thuật toán đệ qui

2.5. Đệ qui có nhớ

2.6. Chứng minh tính đúng đắn của thuật toán đệ qui

Cài đặt các thuật toán đệ qui

- Để cài đặt các thuật toán đệ qui trên các ngôn ngữ lập trình cấp cao quen thuộc như Pascal, C, C++, ... ta thường xây dựng các hàm (thủ tục) đệ qui.
- Trong mục này ta xét một số ví dụ minh hoạ cài đặt thuật toán đệ qui:
  – Hàm đệ qui tính n!
  – Hàm đệ qui tính số Fibonacci
  – Hàm đệ qui tính hệ số nhị thức
  – Tìm kiếm nhị phân
  – Bài toán tháp Hà nội

Tính n!

- Hàm $f(n) = n!$ được định nghĩa đệ qui như sau

  $f(0) \equiv 0! = 1, \qquad n=0,$

  $f(n) = n\, f(n-1), \; n>0$

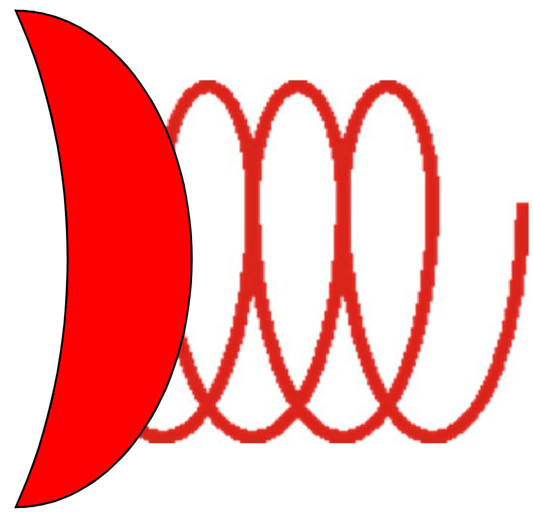
- Hàm đệ qui trên C:

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

Hoạt động của Fact(5)

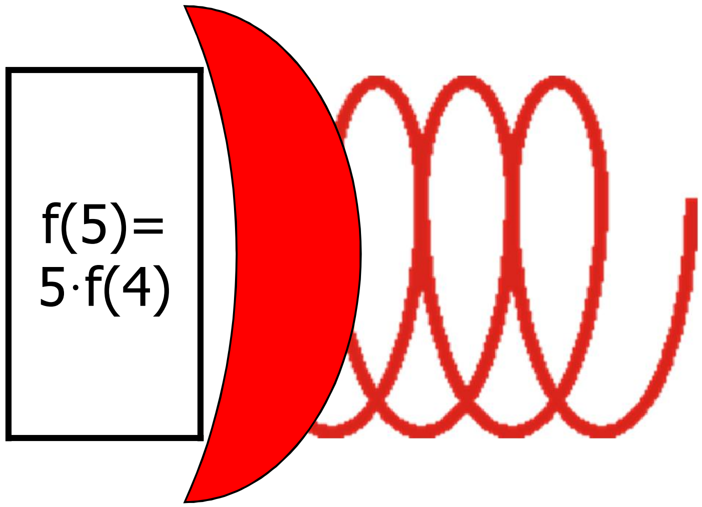
```
int Fact(int n){
  if (n==0) return 1;
  else return n*Fact(n-1);
}
```

Tính 5!



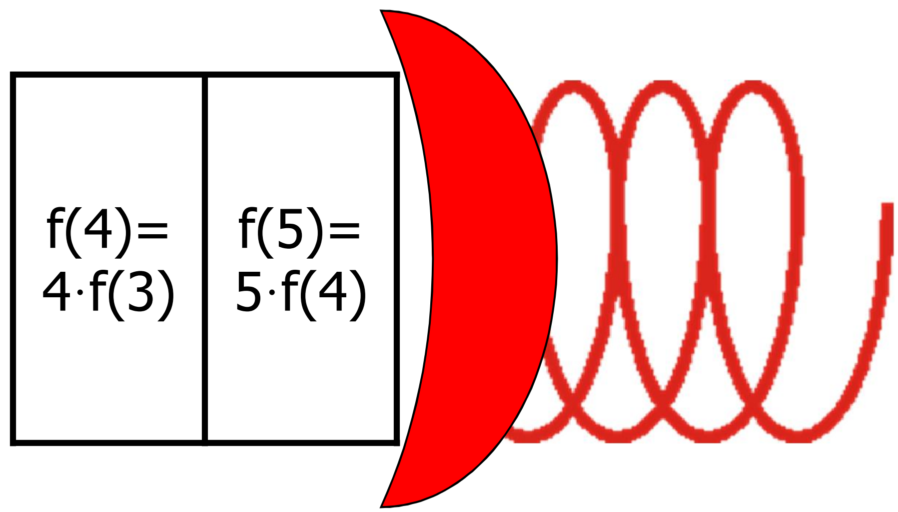
Hoạt động của Fact(5)

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



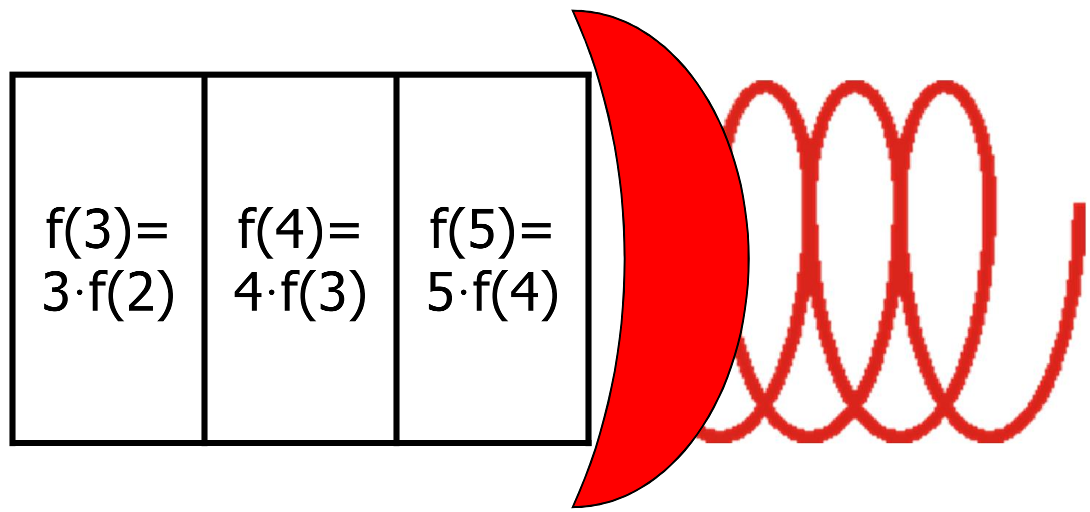
Hoạt động của Fact(5)

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



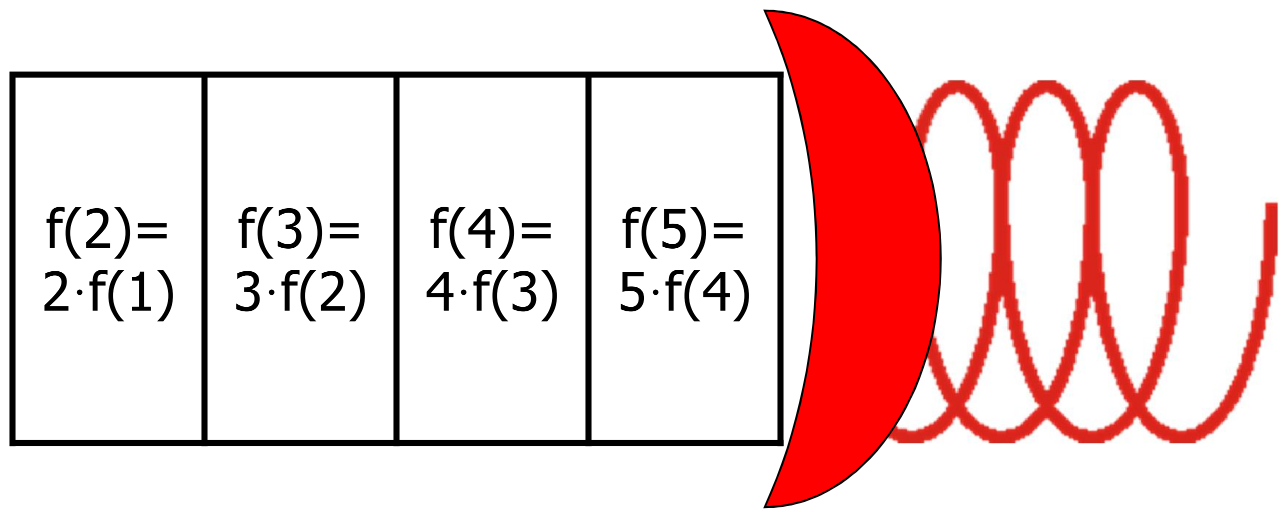
Hoạt động của Fact(5)

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



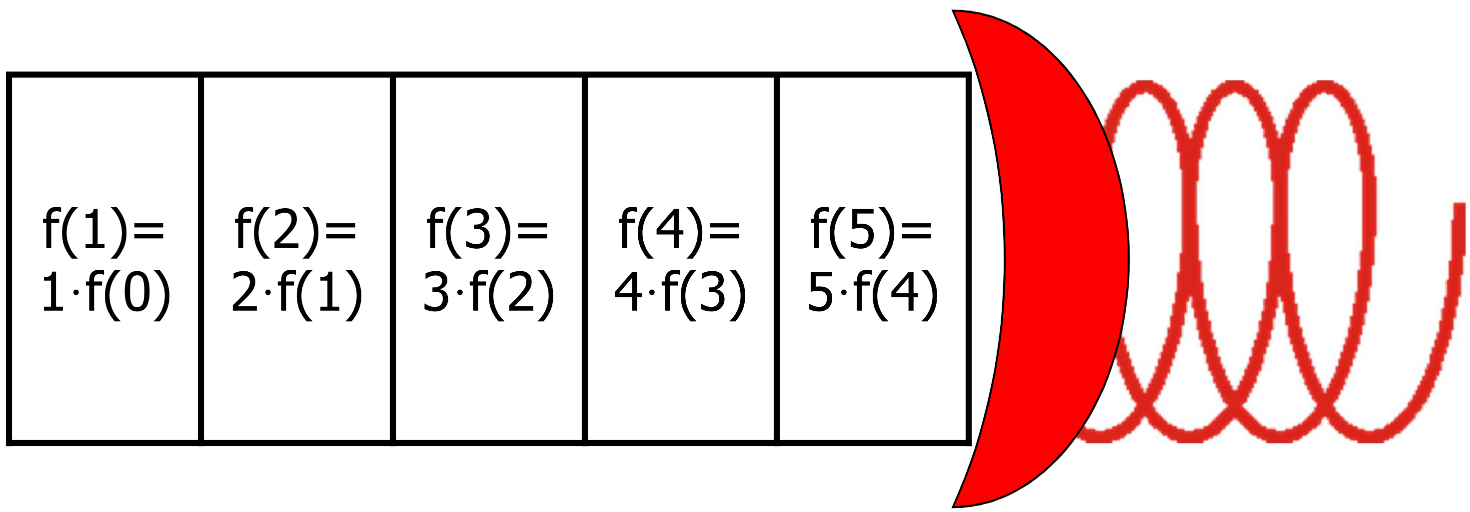
Hoạt động của Fact(5)

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



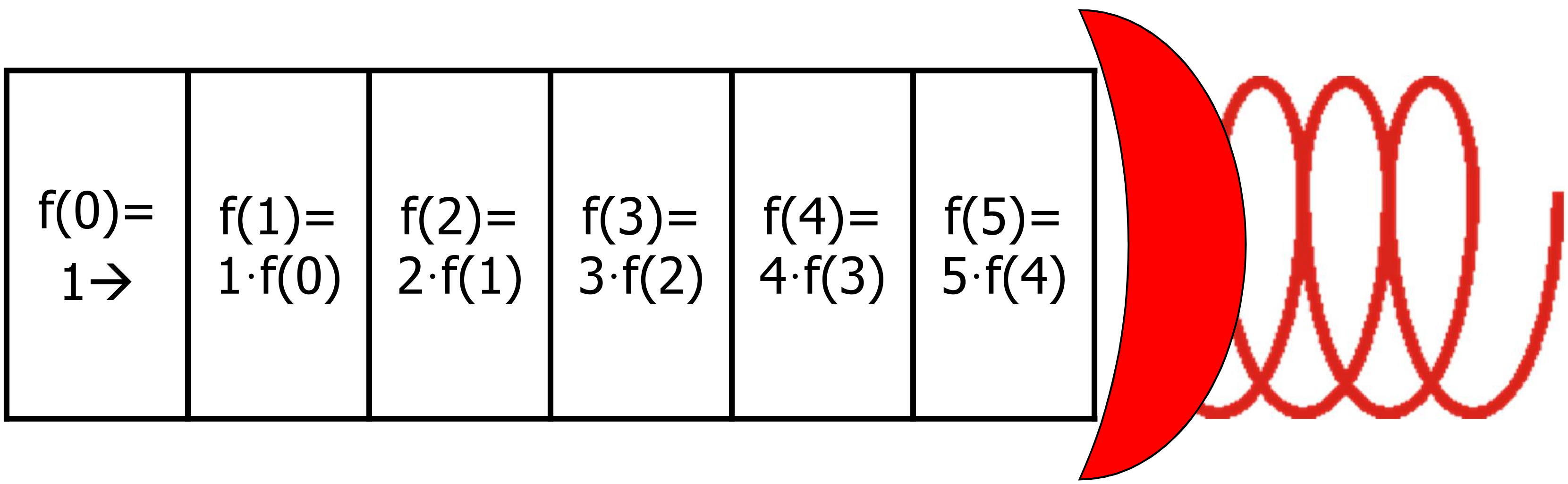
Hoạt động của Fact(5)

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



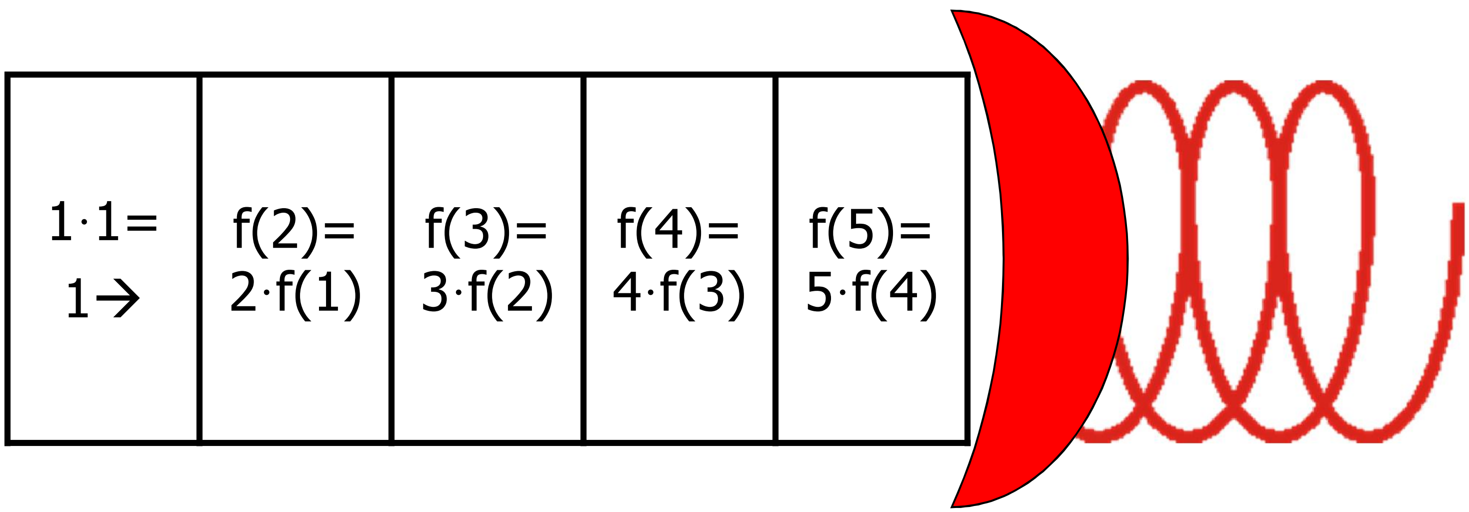
Hoạt động của Fact(5)

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



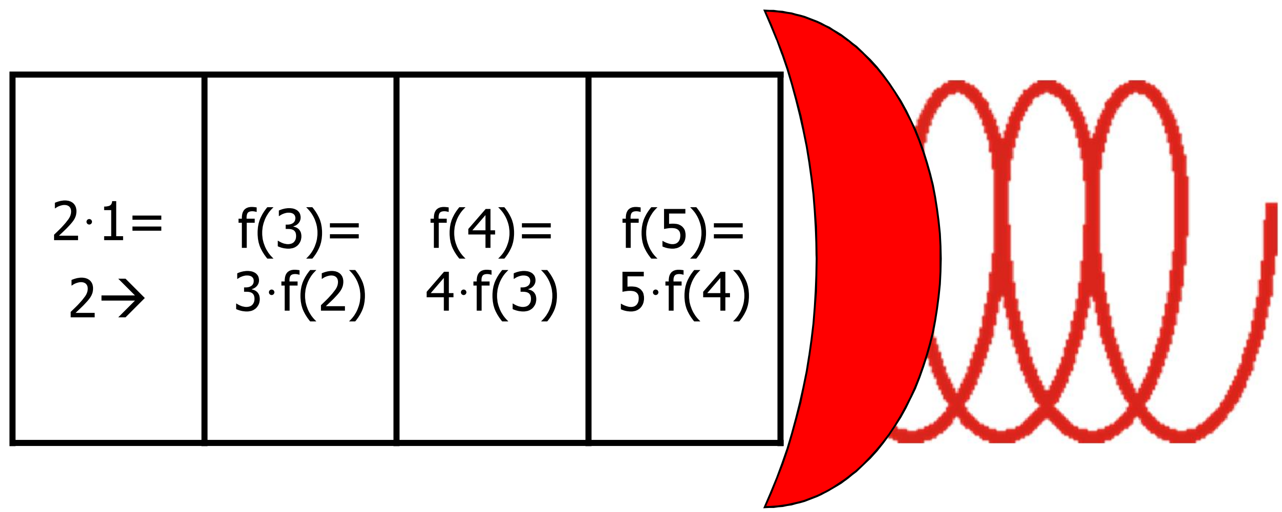
Hoạt động của Fact(5)

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



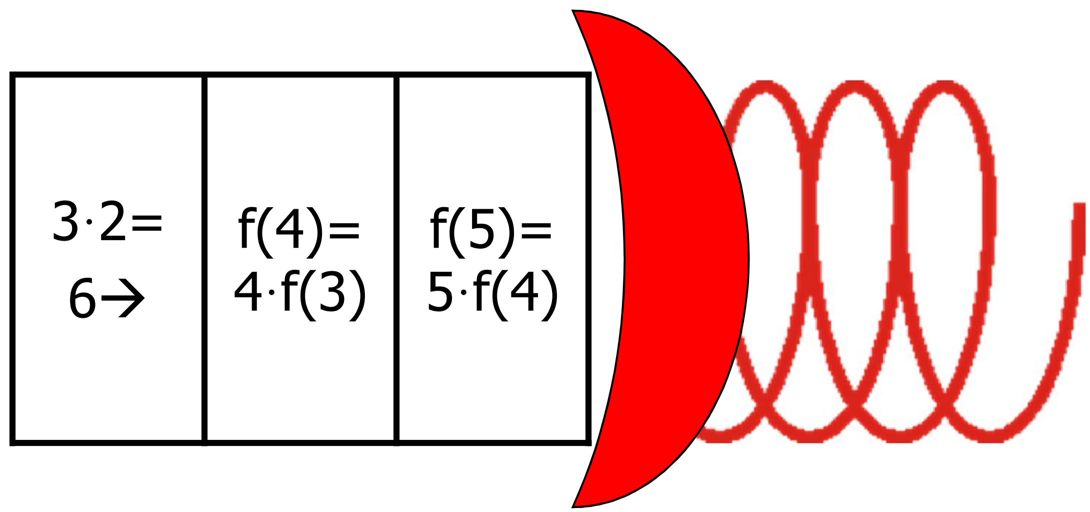
Hoạt động của Fact(5)

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



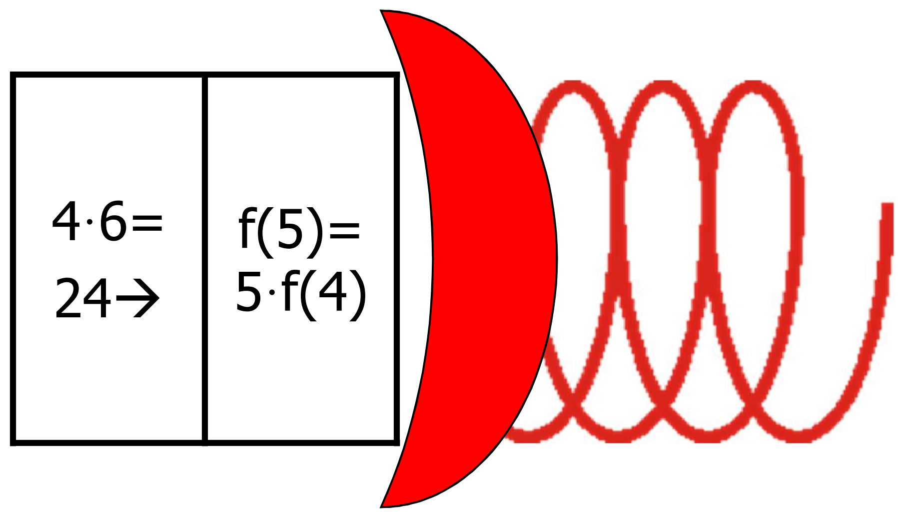
Hoạt động của Fact(5)

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



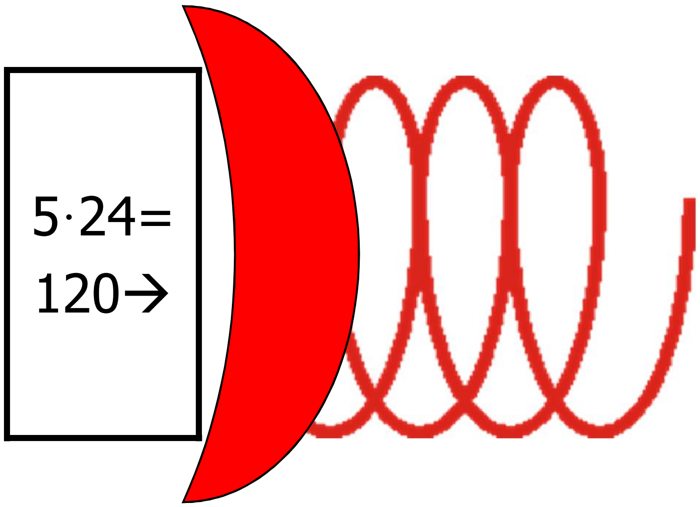
Hoạt động của Fact(5)

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



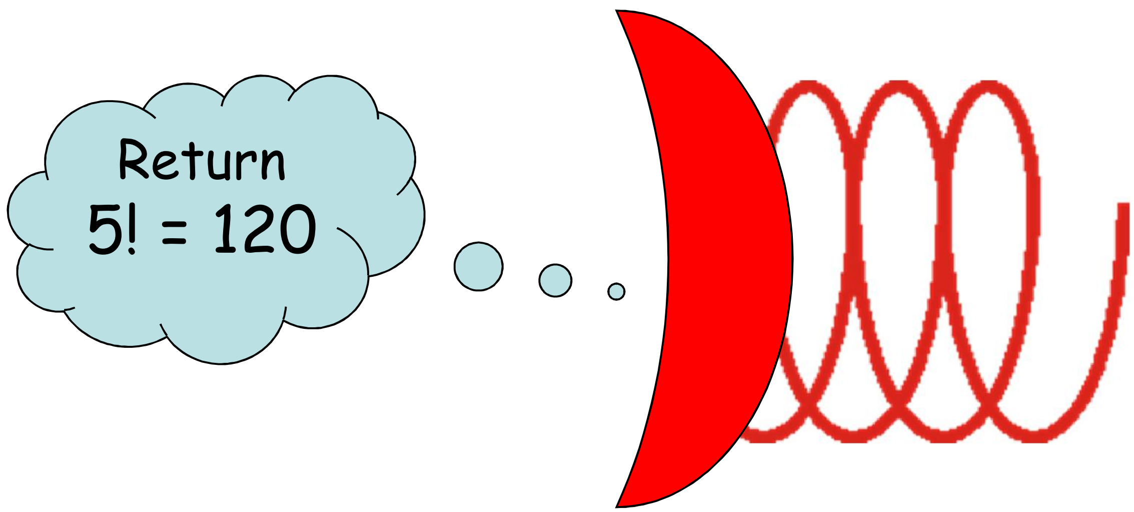
Hoạt động của Fact(5)

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



Hoạt động của Fact(5)

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



Tính số Fibonacci

- Dãy số Fibonacci được định nghĩa đệ qui như sau

$$F(0) = 0, \quad F(1) = 1;$$

$$F(n) = F(n\text{-}1)+F(n\text{-}2), \; n \geq 2.$$

- Hàm đệ qui trên C:

```
int FibRec(int n){
    if (n<=1)return n;
    else return FibRec(n-1)+FibRec(n-2);
}
```

Tính hệ số nhị thức

- Hệ số nhị thức $C(n,k)$ được định nghĩa đệ qui như sau

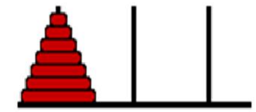
$$C(n,0) = 1, \ \ C(n,n) = 1; \quad \text{với mọi } n >= 0,$$

$$C(n,k) = C(n-1,k-1)+C(n-1,k), \ \ 0 < k < n$$

- Hàm đệ qui trên C:

```
int C(int n, int k){
    if ((k==0)||(k==n)) return 1;
    else return C(n-1,k-1)+C(n-1,k);
}
```

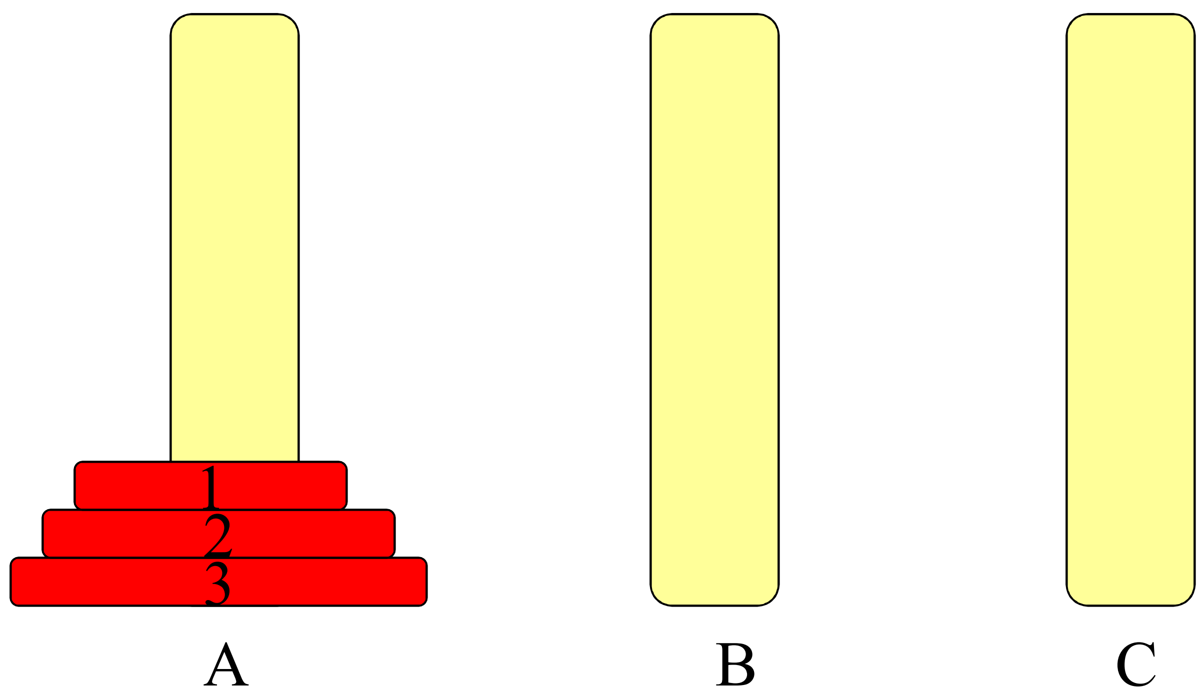
Bài toán tháp Hà nội (Hanoi Tower)



- Bài toán tháp Hà nội. Trò chơi tháp Hà nội được trình bày như sau: “*Có 3 cọc a, b, c. Trên cọc a có một chồng gồm n cái đĩa đường kính giảm dần từ dưới lên trên. Cần phải chuyển chồng đĩa từ cọc a sang cọc c tuân thủ qui tắc: mỗi lần chỉ chuyển* 1 *đĩa và chỉ được xếp đĩa có đường kính nhỏ hơn lên trên đĩa có đường kính lớn hơn. Trong quá trình chuyển được phép dùng cọc b làm cọc trung gian*”.
- Bài toán đặt ra là: Tìm cách chơi đòi hỏi số lần di chuyển đĩa ít nhất

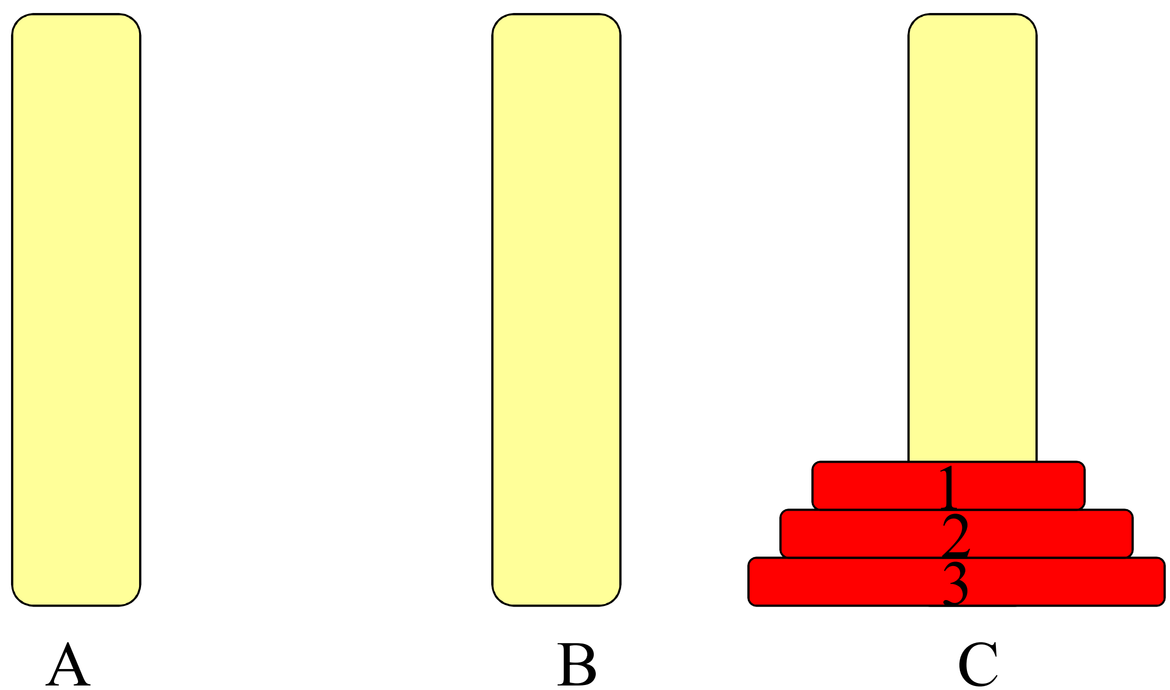
Tower of Hanoi - Trạng thái xuất phát

•Có 3 cọc và một chồng đĩa được sắp xếp theo thứ tự đường kính giảm dần từ dưới lên trên ở cọc A.



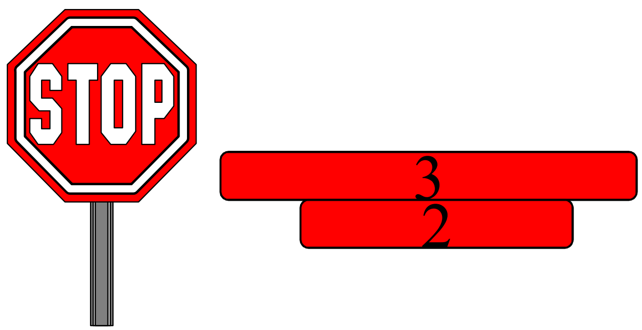
Tower of Hanoi - Trạng thái kết thúc

- Ta muốn chuyển chồng đĩa sang cọc C



Tower of Hanoi - Qui tắc chơi

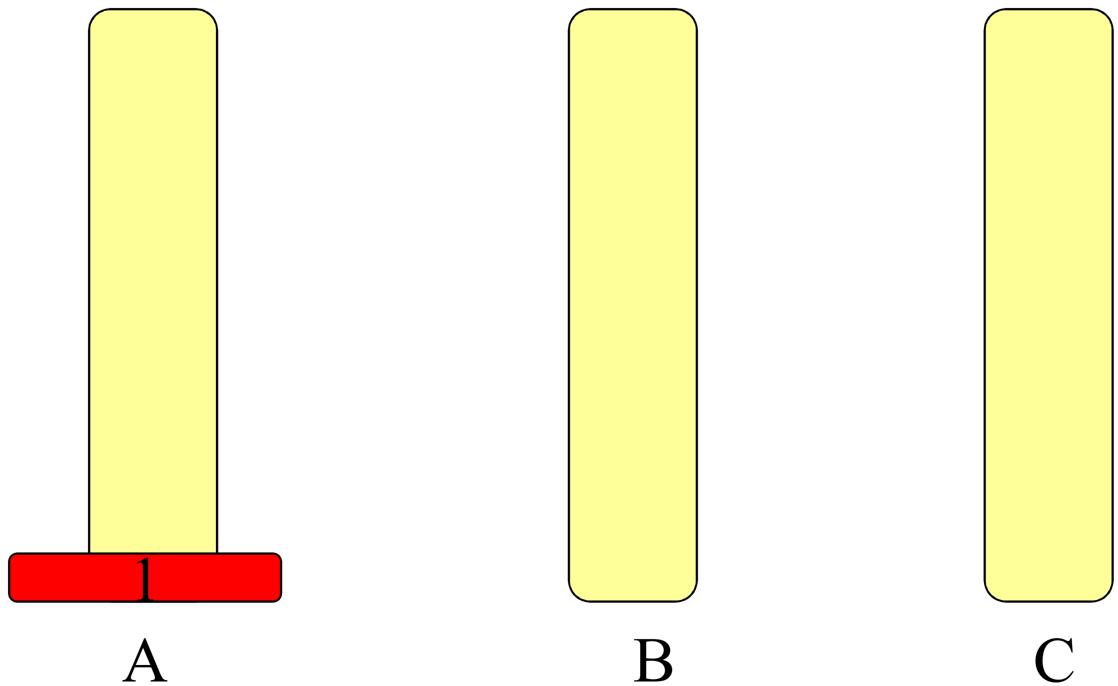
- Mỗi lần chỉ chuyển 1 đĩa
- Chỉ được đặt đĩa có đường kính nhỏ hơn lên trên điã có đường kính lớn hơn



Mục đích: Sử dụng ít lần di chuyển đĩa nhất

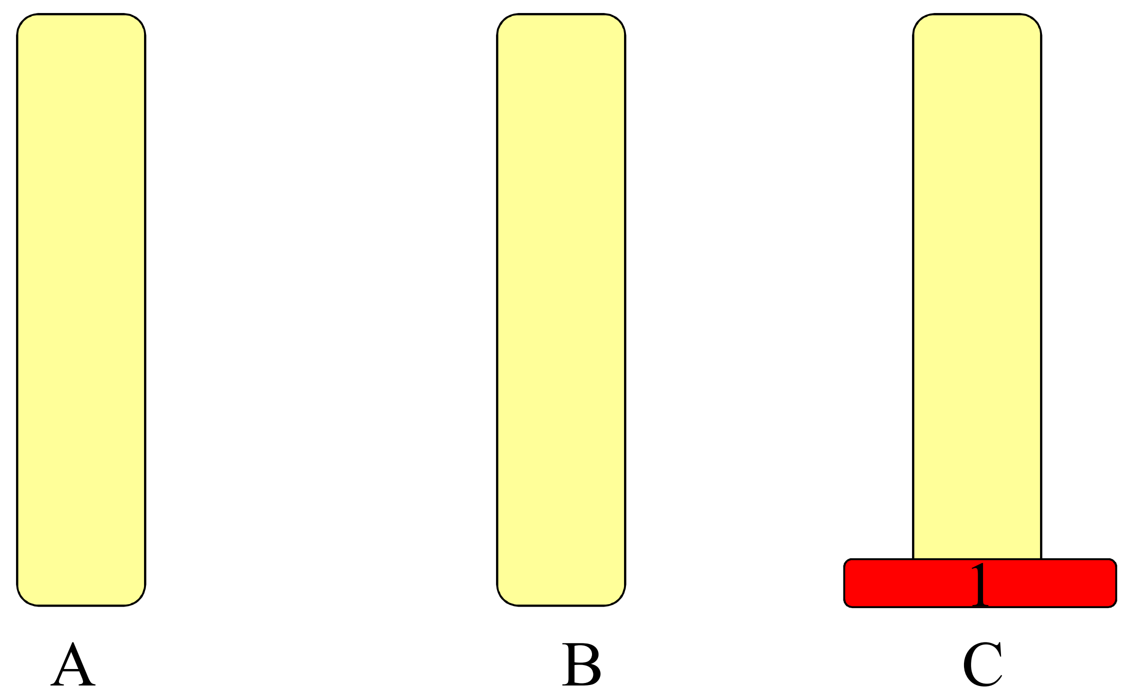
Tower of Hanoi - Chỉ có một đĩa

- Quá dễ!



Tower of Hanoi - Chỉ có một đĩa

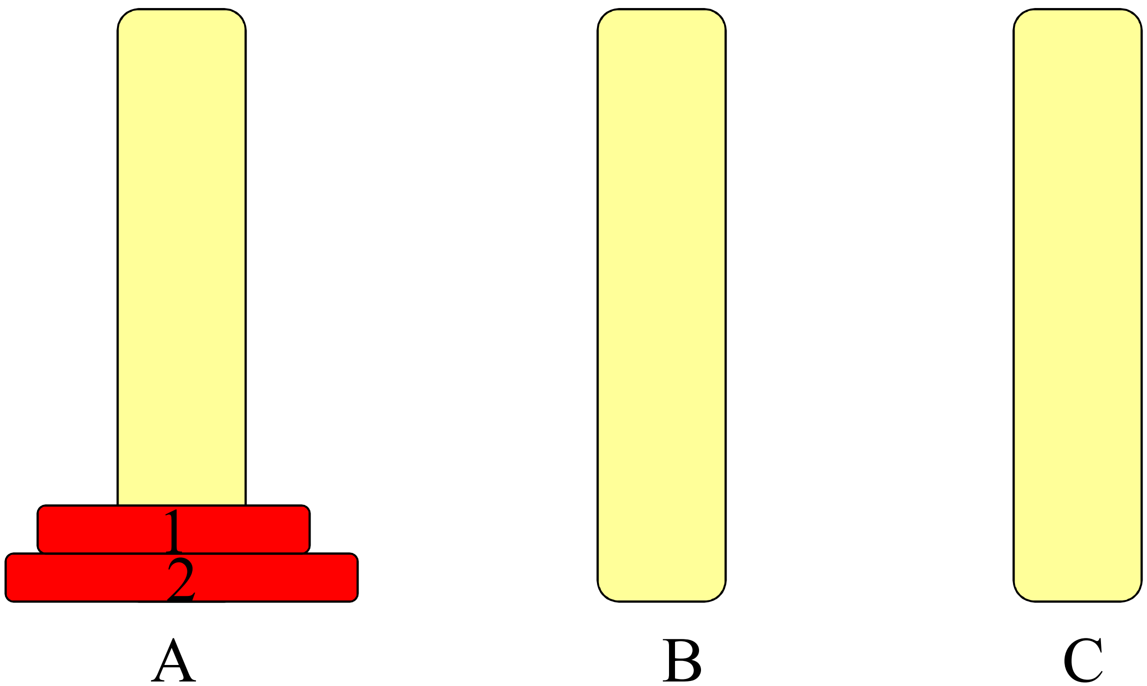
- Quá dễ! Chỉ cần 1 lần chuyển.



Tower of Hanoi - Hai đĩa

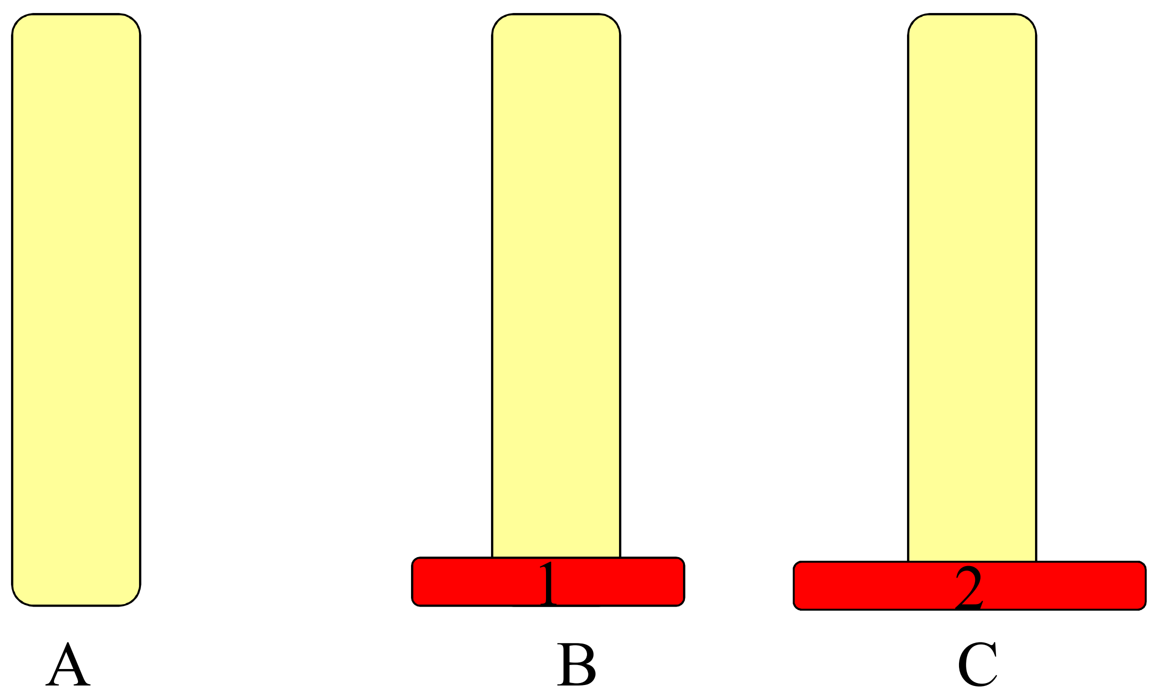
- Nhận thấy rằng ta phải chuyển đĩa 2 đến C. Bằng cách nào?

Trước hết chuyển đĩa 1 sang cọc B, sau đó đĩa 2 sang C



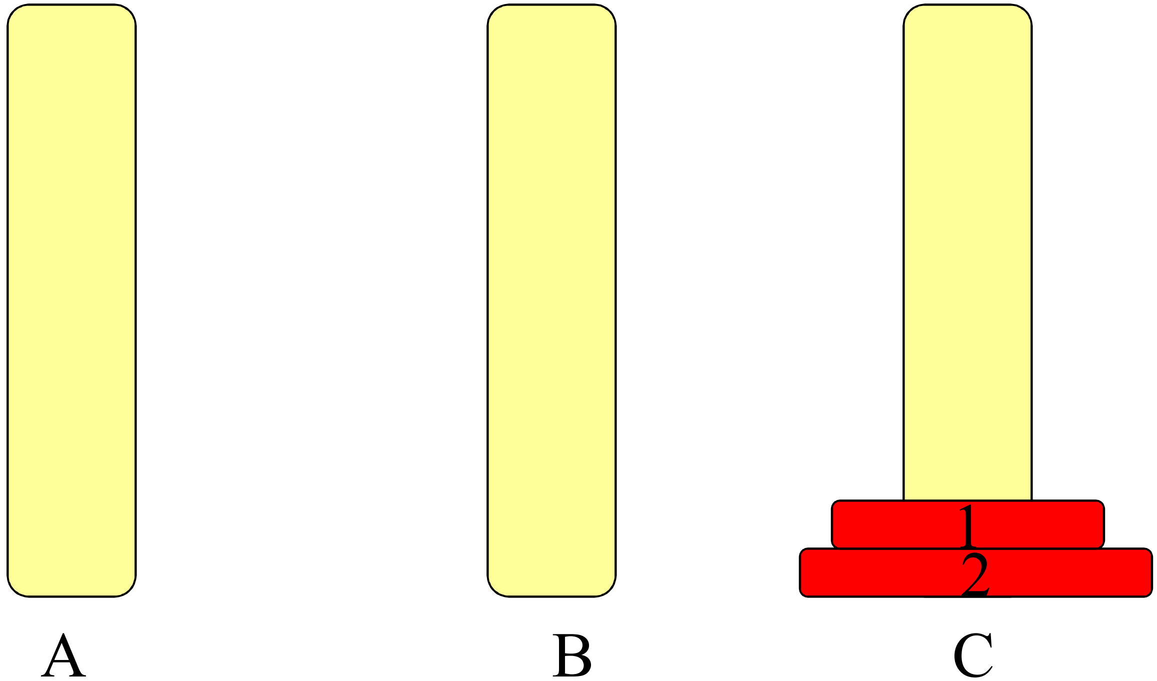
Tower of Hanoi - Hai đĩa

- Tiếp theo?
- Chuyển đĩa 1 sang C



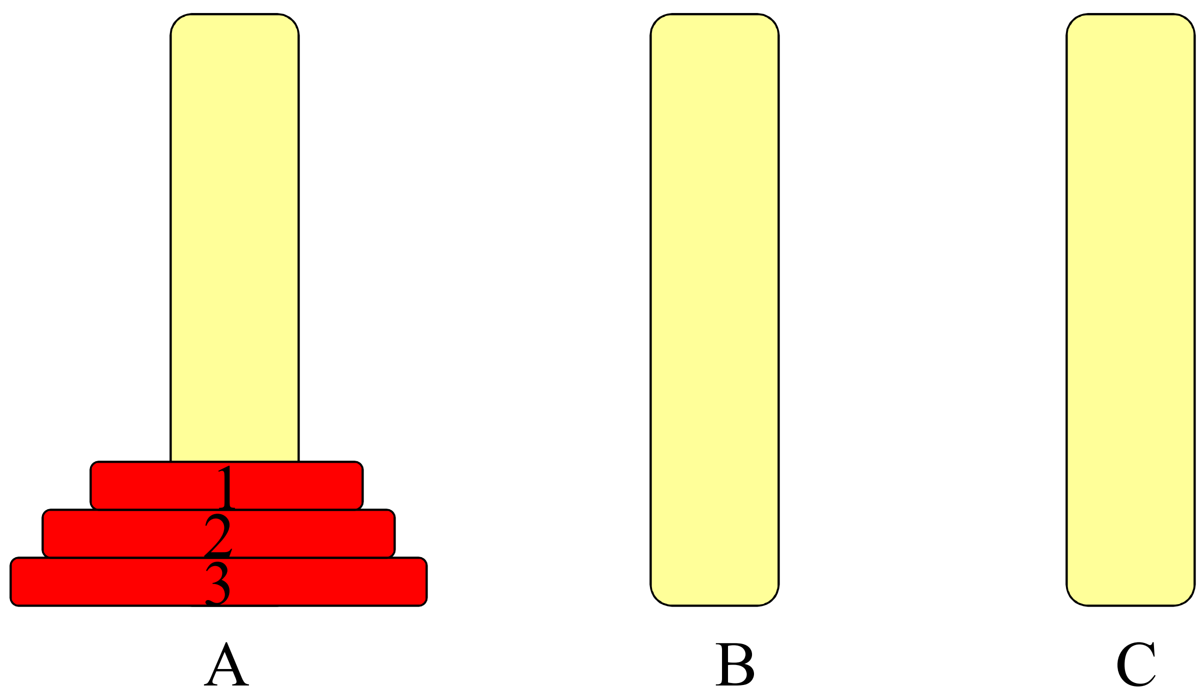
Tower of Hanoi - Hai đĩa

- Hoàn tất!



Tower of Hanoi - Ba đĩa

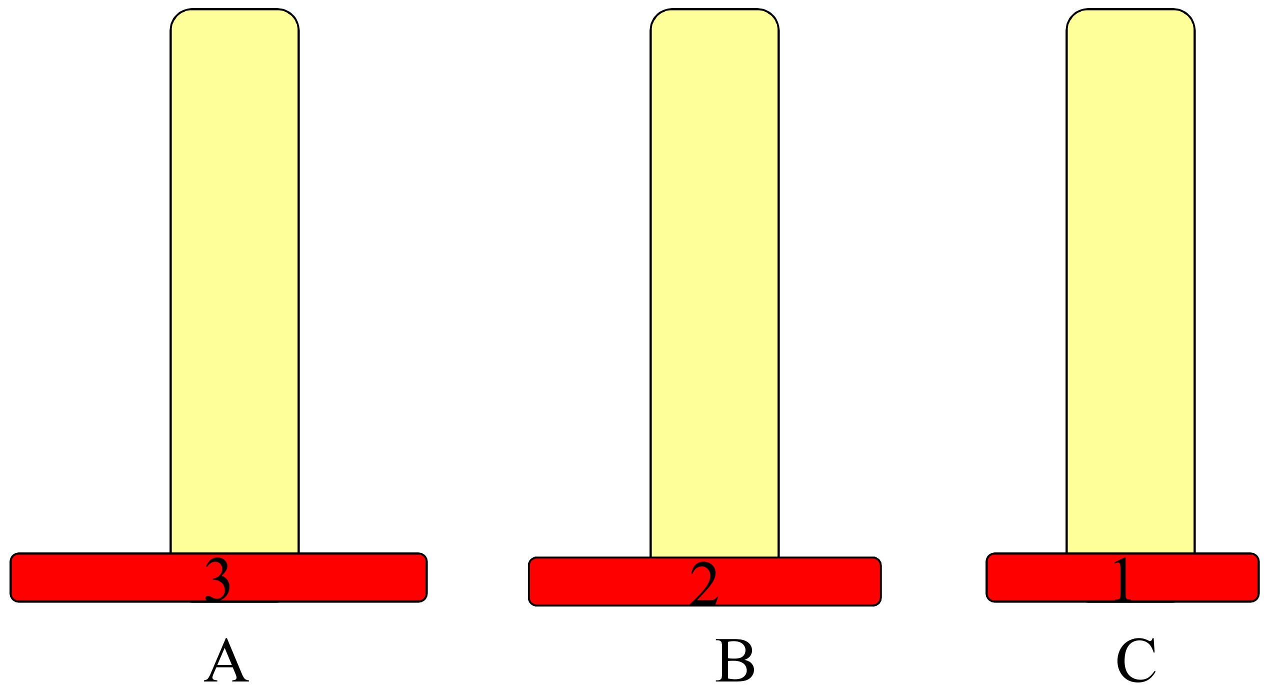
- Ta cần chuyển đĩa 3 sang C, bằng cách nào?
  - Chuyển đĩa 1 và 2 sang B (đệ qui)



Tower of Hanoi - Ba đĩa

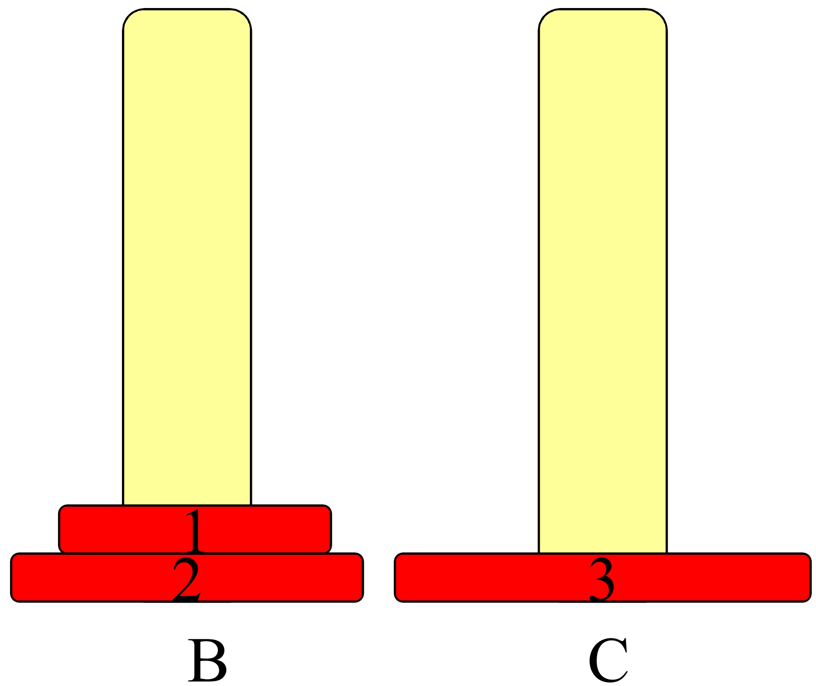
- Ta phải chuyển đĩa 3 sang C, bằng cách nào?
  - Chuyển đĩa 1 và 2 sang B (đệ qui)

➢ Sau đó chuyển đĩa 3 sang C



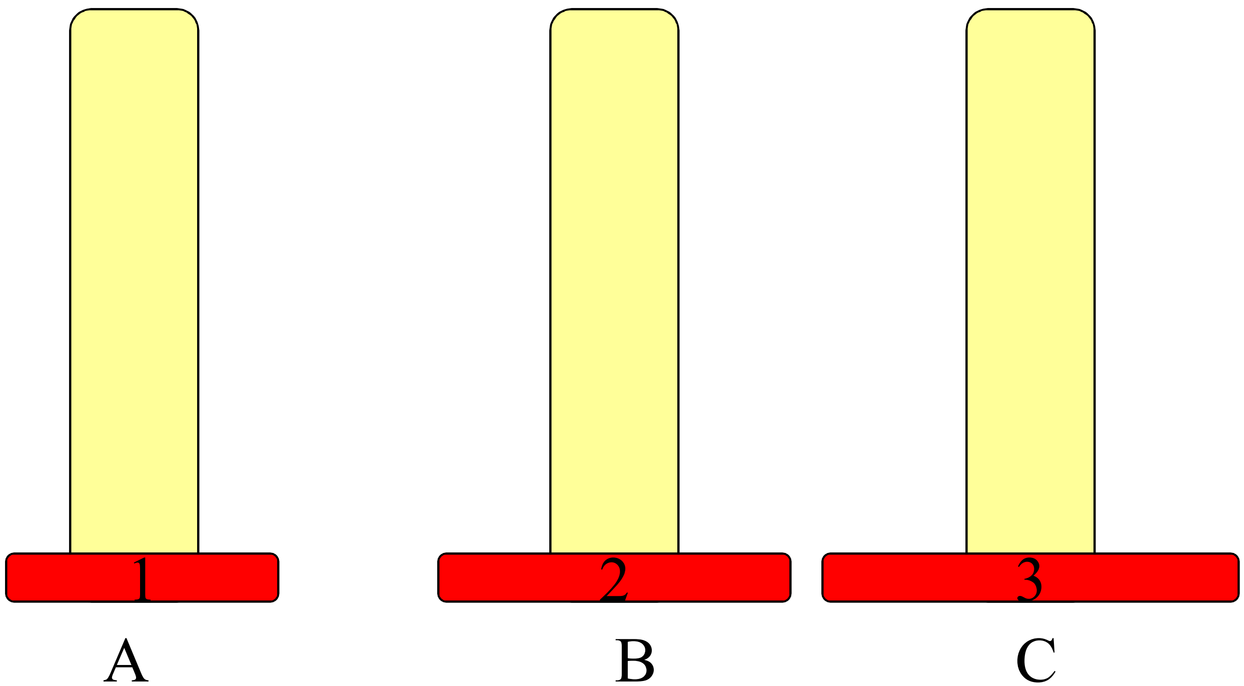
Tower of Hanoi - Ba đĩa

- Nhiệm vụ là: chuyển đĩa 1 và 2 sang C (tương tự như đã làm)



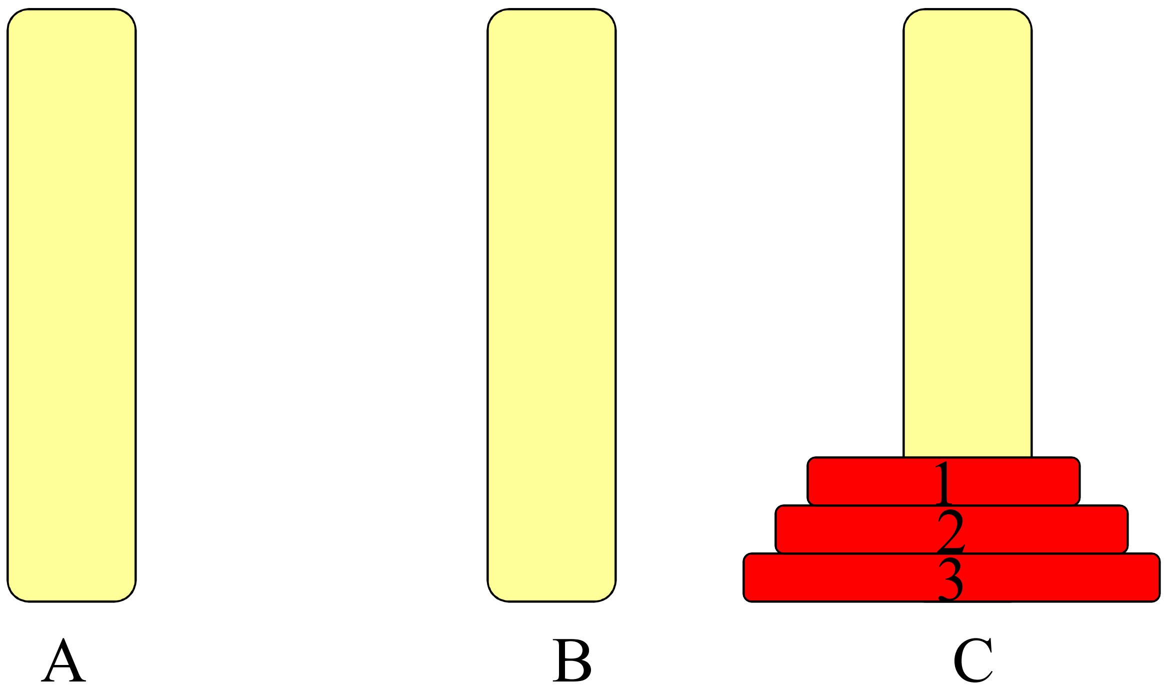
Tower of Hanoi - Ba đĩa

- Nhiệm vụ là: chuyển đĩa 1 và 2 sang C (tương tự như đã làm)



Tower of Hanoi - Ba đĩa

- Hoàn tất!



Bài toán tháp Hà nội

- Lập luận sau đây được sử dụng để xây dựng thuật toán giải quyết bài toán đặt ra
- Nếu n=1 thì ta chỉ việc chuyển đĩa từ cọc *a* sang cọc *c*.
- Giả sử $n \geq 2$. Việc di chuyển đĩa gồm các bước:

(i) Chuyển n-1 đĩa từ cọc ***a*** đến cọc ***b*** sử dụng cọc ***c*** làm trung gian. Bước này phải được thực hiện với số lần di chuyển đĩa nhỏ nhất, nghĩa là ta phải giải bài toán tháp Hà nội với n-1 đĩa.

(ii) Chuyển 1 đĩa (đĩa với đường kính lớn nhất) từ cọc ***a*** đến cọc ***c***.

(iii) Chuyển n-1 đĩa từ cọc ***b*** đến cọc ***c*** (sử dụng cọc ***a*** làm trung gian). Bước này cũng phải được thực hiện với số lần di chuyển đĩa nhỏ nhất, nghĩa là ta phải giải bài toán tháp Hà nội với n-1 đĩa.

Bài toán tháp Hà nội

- Bớc (i) và (iii) đòi hỏi giải bài toán tháp Hà nội với n-1 đĩa, vì vậy số lần di chuyển đĩa ít nhất cần thực hiện trong hai bớc này là $2h_{n-1}$. Do đó, nếu gọi h_n là số lần di chuyển đĩa ít nhất, ta có công thức đệ qui sau:

$$h_1 = 1,$$

$$h_n = 2h_{n-1} + 1,\ n \geq 2.$$

- Ta có thể tìm được công thức trực tiếp cho h_n bằng phương pháp thế:

$$
\begin{aligned}
h_n &= 2\,h_{n-1} + 1 \\
&= 2\,(2\,h_{n-2} + 1) + 1 = 2^2\,h_{n-2} + 2 + 1 \\
&= 2^2(2\,h_{n-3} + 1) + 2 + 1 \qquad = 2^3\,h_{n-3} + 2^2 + 2 + 1 \\
&\ldots \\
&= 2^{n-1}\,h_1 + 2^{n-2} + \ldots + 2 + 1 \\
&= 2^{n-1} + 2^{n-2} + \ldots + 2 + 1 \qquad \text{(do } h_1 = 1) \\
&= 2^n - 1
\end{aligned}
$$

Thuật toán tháp Hà nội

- Thuật toán có thể mô tả trong thủ tục đệ qui sau đây

```
procedure HanoiTower(n, a, b, c);
begin
    if n=1 then <chuyển đĩa từ cọc a sang cọc c>
    else
        HanoiTower(n-1,a,c,b);
        HanoiTower(1,a,b,c);
        HanoiTower(n-1,b,a,c);
end;
```

Cài đặt trên C

```c
#include <stdio.h>
#include <conio.h>

void move (int, char, char, char);
int i = 0;

int main()
{
   int disk;
   printf ("Please input number of disk: ");
   scanf ("%d", &disk);
   move (disk, '1', '3', '2');
   printf ("Total number of moves = %d", i);
   getch();
   return 0;
}
```

```c
void move (int n, char start, char finish, char spare)
{
   if (n == 1){
      printf ("Move disk from %c to %c\n", start, finish);
      i++;
      return;
   } else {
      move (n-1, start, spare, finish);
      move (1, start, finish, spare);
      move (n-1, spare, finish, start);
   }
}
```

Nội dung

2.1. Khái niệm đệ qui

2.2. Thuật toán đệ qui

2.3. Một số ví dụ minh hoạ

2.4. Phân tích thuật toán đệ qui

2.5. Đệ qui có nhớ

2.6. Chứng minh tính đúng đắn của thuật toán đệ qui

Phân tích thuật toán đệ qui

- Để phân tích thuật toán đệ qui ta thường tiến hành như sau:
  - Gọi ***T(n)*** là thời gian tính của thuật toán
  - Xây dựng công thức đệ qui cho ***T*(*n*)**.
  - *Giải công thức đệ qui* thu được để đưa ra đánh giá cho ***T(n)***
- Nói chung ta chỉ cần một đánh giá sát cho tốc độ tăng của $T(n)$ nên việc ***giải công thức đệ qui*** đối với $T(n)$ là đưa ra đánh giá tốc độ tăng của $T(n)$ trong ký hiệu tiệm cận

Phân tích FibRec

- **Ví dụ.** Thuật toán FibRec

```
int FibRec(int n){
   if (n<=1)return n;
   else
    return FibRec(n-1)+ FibRec(n-2);
}
```

- Gọi $T(n)$ là số phép toán cộng phải thực hiện trong lệnh gọi FibRec(n). Ta có

  $T(0) = 0$, $T(1) = 0$;

  $T(n) = T(n\text{-}1)+T(n\text{-}2)+1$, n>1

- Từ đó có thể chứng minh bằng qui nạp: $T(n) = \Theta(F_n)$

Phân tích FibRec

- Phân tích trên cho thấy FibRec có thời gian tính tăng với tốc độ $(1.6)^n$. Vì thế việc sử dụng FibRec là không hiệu quả.
- Để tính số Fibonacci thứ *n*, ta nên dùng **thủ tục lặp** FibIter sau đây

```
int FibIter(int n){
   int f1, f2, f3, k;
   if (n<=1)return n;
   else {  f1=0; f2=1;
           for (k=2;k<=n;k++){
               f3=f1+f2; f1=f2; f2=f3;}
           return f3;}
}
```

- **Chú ý:** Việc thay thế hàm đệ qui bởi hàm không đệ qui thường được gọi là việc **khử đệ qui**. Khử đệ qui không phải bao giờ cũng là dễ thực hiện như trong tình huống bài toán tính số Fibonacci.

Phân tích thời gian của thuật toán chia để trị

- Gọi $T(n)$ – thời gian giải bài toán kích thước n
- Thời gian của chia để trị được đánh giá dựa trên đánh giá thời gian thực hiện 3 thao tác của thuật toán:
  - **Chia** bài toán ra thành a bài toán con, mỗi bài toán kích thước n/b: đòi hỏi thời gian: $D(n)$
  - **Trị** (giải) các bài toán con: $aT(n/b)$
  - **Tổ hợp** các lời giải: $C(n)$
- **Vậy ta có công thức đệ qui sau đây để tính *T*(*n*) :**

$$T(n) = \begin{cases} \Theta(1) & \text{nếu } n \le n_0 \\ aT(n/b) + D(n) + C(n) & \text{nếu trái lại} \end{cases}$$

Sơ đồ thuật toán chia để trị

procedure D-and-C (n);

begin

    if $n \le n_0$ **then**

            Giải bài toán một cách trực tiếp

    else

    begin

        Chia bài toán thành ***a*** bài toán con kích thước ***n/b****;*

        for (mỗi bài toán trong ***b*** bài toán con) **do** D-and-C(***n/b***);

        Tổng hợp lời giải của ***a*** bài toán con để thu được lời giải của bài toán gốc;

    end;

end;

Giải công thức đệ qui: Định lý thợ rút gọn

- Ta cần đưa ra đánh giá dưới dạng hiện cho $T(n)$
- Định lý thợ cung cấp công cụ để ***đánh giá số hạng tổng quát*** của dãy số thoả mãn công thức đệ qui dạng

$$T(n) = a\,T(n/b) + cn^k$$

- trong đó:
  - $a \geq 1$ và $b \geq 1$, $c > 0$ là các hằng số
  - $T(n)$ được xác định với đối số nguyên không âm
  - Ta dùng n/b thay cho cả $\lfloor n/b \rfloor$ lẫn $\lceil n/b \rceil$

Định lý thợ rút gọn
Simplified Master Theorem

Giả sử $a \geq 1$ và $b > 1$, $c > 0$ là các hằng số. Xét $T(n)$ là công thức đệ qui

$$T(n) = a\, T(n/b) + c\, n^k$$

xác định với $n \geq 0$.

1. Nếu $a > b^k$, thì $T(n) = \Theta(\, n^{\log_b a}\,)$.
2. Nếu $a = b^k$, thì $T(n) = \Theta(\, n^k \log n\,)$.
3. Nếu $a < b^k$, thì $T(n) = \Theta(\, n^k\,)$.

Ví dụ áp dụng định lý thợ

$T(n) = aT(n/b)+cn^k$

1. Nếu $a > b^k$, thì $T(n) = \Theta(n^{\log_b a})$.
2. Nếu $a = b^k$, thì $T(n) = \Theta(n^k \lg n)$.
3. Nếu $a < b^k$, thì $T(n) = \Theta(n^k)$.

- **Ví dụ 1:** $T(n) = 3T(n/4) + cn^2$
  - Trong ví dụ này: a=3, b=4, k=2.
  - Do $3 < 4^2$, ta có tình huống 3, nên $T(n) = \Theta(n^2)$
- **Ví dụ 2.** $T(n) = 2T(n/2) + n^{0.5}$
  - Trong ví dụ này: a=2, b=2, k=1/2.
  - Do $2 > 2^{1/2}$ nên ta có tình huống 1. Vậy $T(n) = \Theta(n^{\log_b a}) = \Theta(n)$.
- **Ví dụ 3.** $T(n) = 16T(n/4) + n$
  - $a = 16$, $b = 4$, k=1.
  - Ta có 16 > 4, vì thế có tình huống 1. Vậy $T(n) = \Theta(n^2)$.
- **Ví dụ 4.** $T(n) = T(3n/7) + 1$
  - a=1, b=7/3, k=0.
  - Ta có $a = b^k$, suy ra có tình huống 2. Vậy $T(n)=\Theta(n^k \log n) = \Theta(\log n)$.

Thời gian tính của sắp xếp trộn
MERGE-SORT Running Time

- **Chia:**
  - tính q như là giá trị trung bình của p và r: $D(n) = \Theta(1)$
- **Trị:**
  - giải đệ qui 2 bài toán con, mỗi bài toán kích thước $n/2 \Rightarrow 2T(n/2)$
- **Tổ hợp:**
  - TRỘN (MERGE) trên các mảng con cỡ n phần tử đòi hỏi thời gian $\Theta(n)$

    $\Rightarrow C(n) = \Theta(n)$

$$T(n) = \begin{cases} \Theta(1) & \text{nếu } n = 1 \\ 2T(n/2) + \Theta(n) & \text{nếu } n > 1 \end{cases}$$

– **Suy ra theo định lý thợ: $T(n) = \Theta(n \log n)$**

Phân tích Bsearch

```
function Bsearch(x[1..n],s,f){
    m=(s+f)/2;
    if (y==x[m]) return m;
    else{
         if (y<x[m])
            return Bsearch(x,s,m-1);
         else  /* y>x[m] */
            return Bsearch(x,m+1,f)
    }
}
```

- Gọi $T(n)$ là thời gian tính của việc thực hiện Bsearch(x[1..n],1,n), ta có

$$T(1) = c$$

$$T(n) = T(n/2) + d$$

trong đó *c và d là các hằng số.*

- Từ đó theo định lý thợ, $\mathbf{T(n) = \Theta(\log n)}$.

Phân tích thuật toán tính C(n,k)

- Xét thuật toán đệ qui để tính hệ số nhị thức C(n,k):

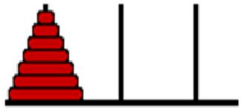
```
int C(int n, int k){
    if ((k==0)||(k==n)) return 1;
    else return C(n-1,k-1)+C(n-1,k);
}
```

- Gọi $C^*(n,k)$ là thời gian thực hiện lệnh gọi hàm C(n,k). Khi đó, dễ thấy $C^*(n,k)$ thoả mãn công thức đệ qui sau đây:

$$C^*(n,0) \geq 1, \quad C^*(n,n) \geq 1; \quad \text{với mọi } n >= 0,$$

$$C^*(n,k) \geq C^*(n-1,k-1)+C^*(n-1,k)+1, \quad 0 < k < n$$

- Từ đó, có thể chứng minh $C^*(n,k) \geq C_n^k$. Do đó thuật toán đệ qui nói trên để tính hệ số nhị thức là không hiệu quả.



Nội dung

2.1. Khái niệm đệ qui

2.2. Thuật toán đệ qui

2.3. Một số ví dụ minh hoạ

2.4. Phân tích thuật toán đệ qui

2.5. Đệ qui có nhớ

2.6. Chứng minh tính đúng đắn của thuật toán đệ qui

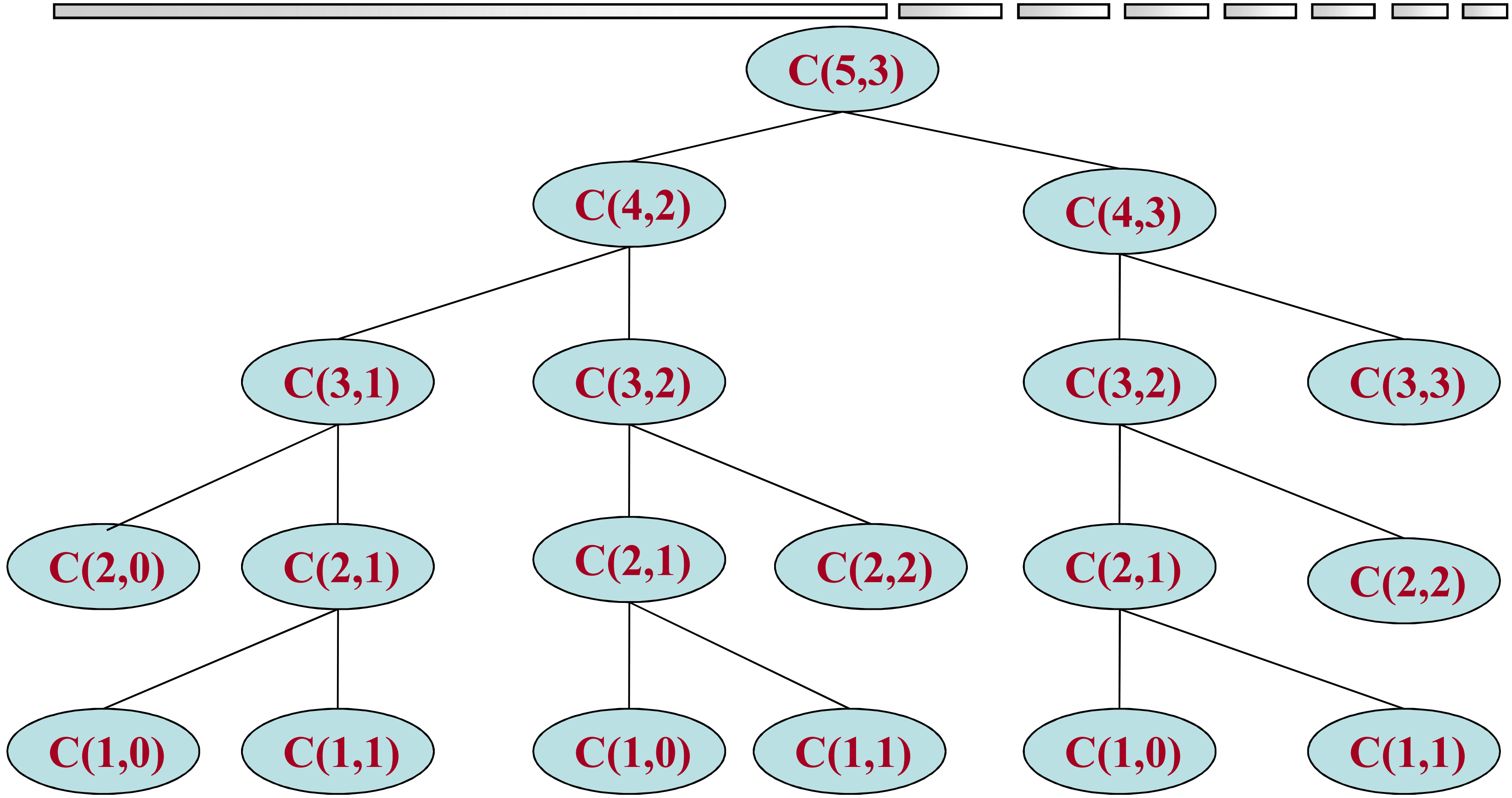
Đệ qui có nhớ

- Trong phần trước ta đã thấy các thuật toán đệ qui để tính số Fibonacci và tính hệ số nhị thức là kém hiệu quả.
- Để tăng hiệu quả của các thuật toán đệ qui mà không cần tiến hành xây dựng các thủ tục lặp hay khử đệ qui, ta có thể sử dụng kỹ thuật đệ qui có nhớ.
- Sử dụng kỹ thuật này, trong nhiều trường hợp, ta giữ nguyên được cấu trúc đệ qui của thuật toán và đồng thời lại đảm bảo được hiệu quả của nó. Nhược điểm lớn nhất của cách làm này là đòi hỏi về bộ nhớ.

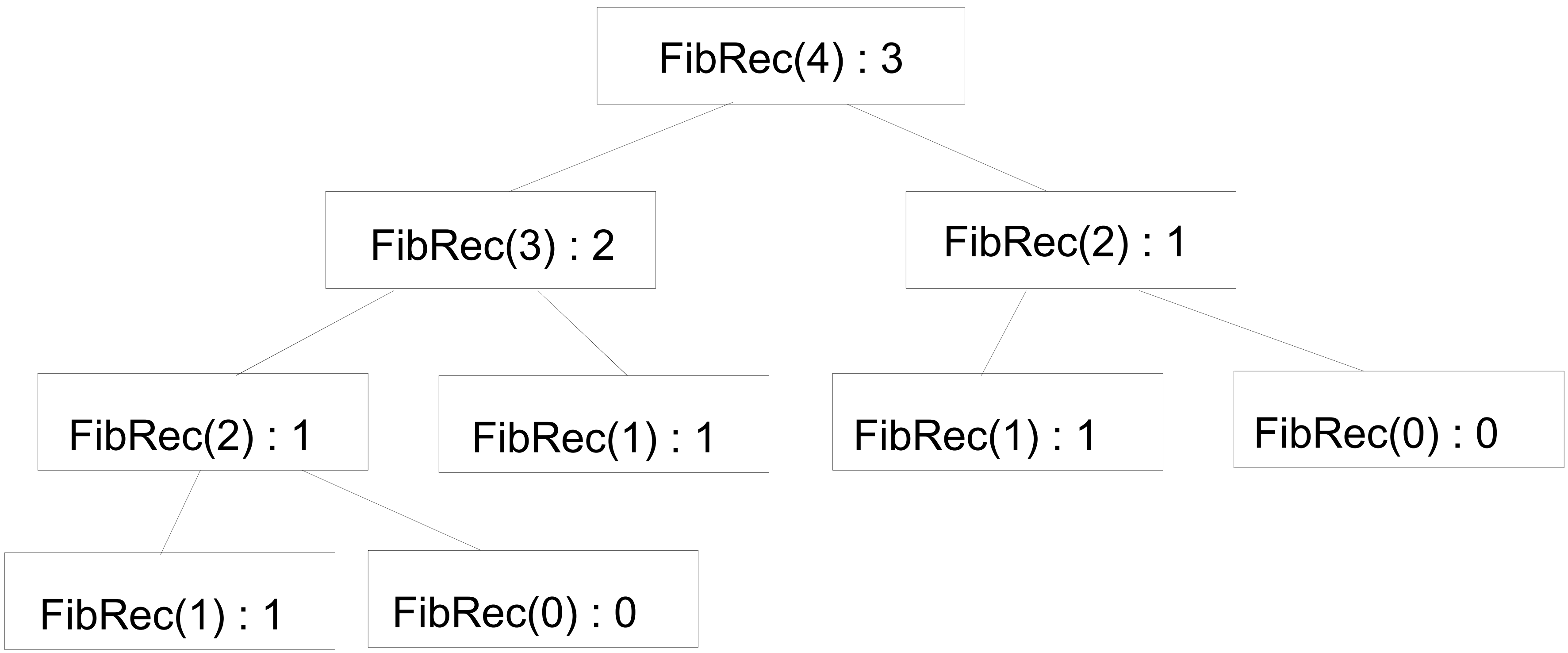
Bài toán con trùng lặp

- Nhận thấy là trong các thuật toán đệ qui là mỗi khi cần đến lời giải của một bài toán con ta lại phải trị nó một cách đệ qui. Do đó, có những bài toán con bị giải đi giải lại nhiều lần. Điều đó dẫn đến tính kém hiệu quả của thuật toán. Hiện tượng này gọi là hiện tượng bài toán con trùng lặp.
- Ta xét ví dụ thuật toán tính hệ số nhị thức C(5,3). Cây đệ qui thực hiện lệnh gọi hàm C(5,3) được chỉ ra trong hình sau đây

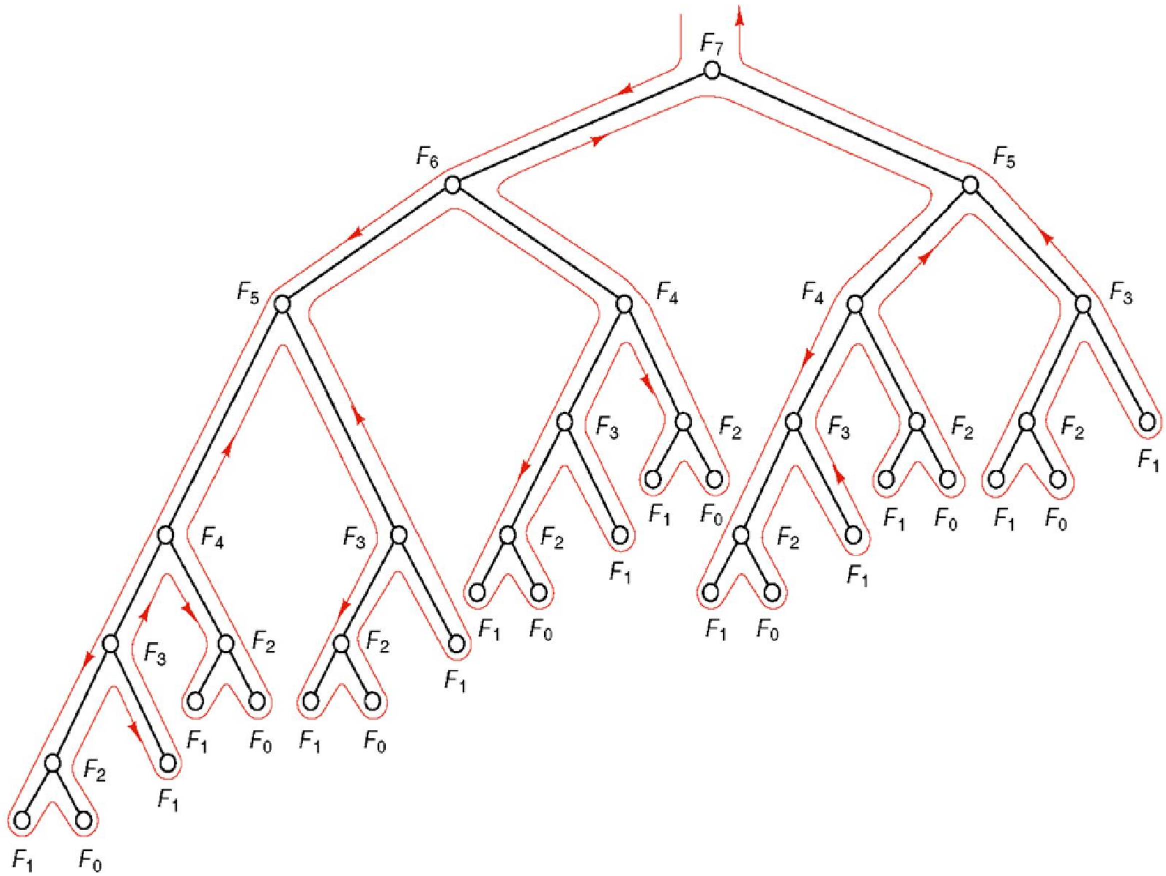
Bài toán con trùng lặp trong việc tính C(5,3)



Bài toán con trùng lặp khi tính FibRec(4)



Cây đệ qui tính F_7



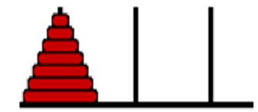
Đệ qui có nhớ

- Để khắc phục hiện tượng này, ý tưởng của ***đệ qui có nhớ*** là: Ta sẽ dùng biến ghi nhớ lại thông tin về lời giải của các bài toán con ngay sau lần đầu tiên nó được giải. Điều đó cho phép rút ngắn thời gian tính của thuật toán, bởi vì, mỗi khi cần đến có thể tra cứu mà không phải giải lại những bài toán con đã được giải trước đó.
- Xét ví dụ thuật toán đệ qui tính hệ số nhị thức. Ta đưa vào biến
- D[n,k] để ghi nhận những giá trị đã tính.
- Đầu tiên D[n,k]=0, mỗi khi tính được C(n,k) giá trị này sẽ được ghi nhận vào D[n,k]. Như vậy, nếu D[n,k]>0 thì điều đó có nghĩa là không cần gọi đệ qui hàm C(n,k)

Hàm tính C(n,k) có nhớ

```
Function C(n,k){
   if D[n,k]>0  return D[n,k]
   else{
          D[n,k] = C(n-1,k-1)+C(n-1,k);
          return D[n,k];
       }
}
```

- Trước khi gọi hàm C(n,k) cần khởi tạo mảng D[,] như sau:
- D[i,0] = 1, D[i,i]=1, với i=0,1,...,n



Nội dung

2.1. Khái niệm đệ qui

2.2. Thuật toán đệ qui

2.3. Một số ví dụ minh hoạ

2.4. Phân tích thuật toán đệ qui

2.5. Đệ qui có nhớ

2.6. Chứng minh tính đúng đắn của thuật toán đệ qui

Chứng minh tính đúng đắn của thuật toán đệ qui

- Để chứng minh tính đúng đắn của thuật toán đệ qui thông thường ta sử dụng qui nạp toán học.
- Ngược lại chứng minh bằng qui nạp cũng thường là cơ sở để xây dựng nhiều thuật toán đệ qui.
- Ta xét một số ví dụ minh hoạ

Ví dụ 1

- **Ví dụ 1.** Chứng minh hàm Fact(n) cho ta giá trị của n!

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

- CM bằng qui nạp toán học.
  - **Cơ sở qui nạp.** Ta có Fact(0) = 1 = 0!
  - **Chuyển qui nạp.** Giả sử Fact(n-1) cho giá trị của (n-1)!, ta chứng minh Fact(n) cho giá trị của n! Thực vậy, lệnh Fact(n) sẽ trả lại giá trị

$$n*Fact(n-1) = (\text{theo giả thiết qui nạp}) = n*(n-1)! = n!$$

Ví dụ 2

Xét hàm trên Pascal sau đây

```
function Count(x: integer): integer;
begin
    if x=0 then
    begin
        Count:=0; exit
    end
    else Count:= x mod 2 + Count(x div 2)
end;
```

- Hỏi hàm nói trên cho ta đặc trưng gì của số nguyên x? Chứng minh tính đúng đắn của khẳng định đề xuất.

Ví dụ 2

Xét hàm trên C sau đây

```
int Count(int x)
{
    if (x==0)
        return 0;
    else return (x % 2 + Count(x /2));
}
```

- Hỏi hàm nói trên cho ta đặc trưng gì của số nguyên x? Chứng minh tính đúng đắn của khẳng định đề xuất.

Ví dụ 3

- Trên mặt phẳng vẽ *n* đường thẳng ở vị trí tổng quát. Hỏi ít nhất phải sử dụng bao nhiêu màu để tô các phần bị chia bởi các đường thẳng này sao cho không có hai phần có chung cạnh nào bị tô bởi cùng một màu?
- *P*(*n*): Luôn có thể tô các phần được chia bởi *n* đường thẳng vẽ ở vị trí tổng quát bởi 2 màu xanh và đỏ sao cho không có hai phần có chung cạnh nào bị tô bởi cùng một màu.

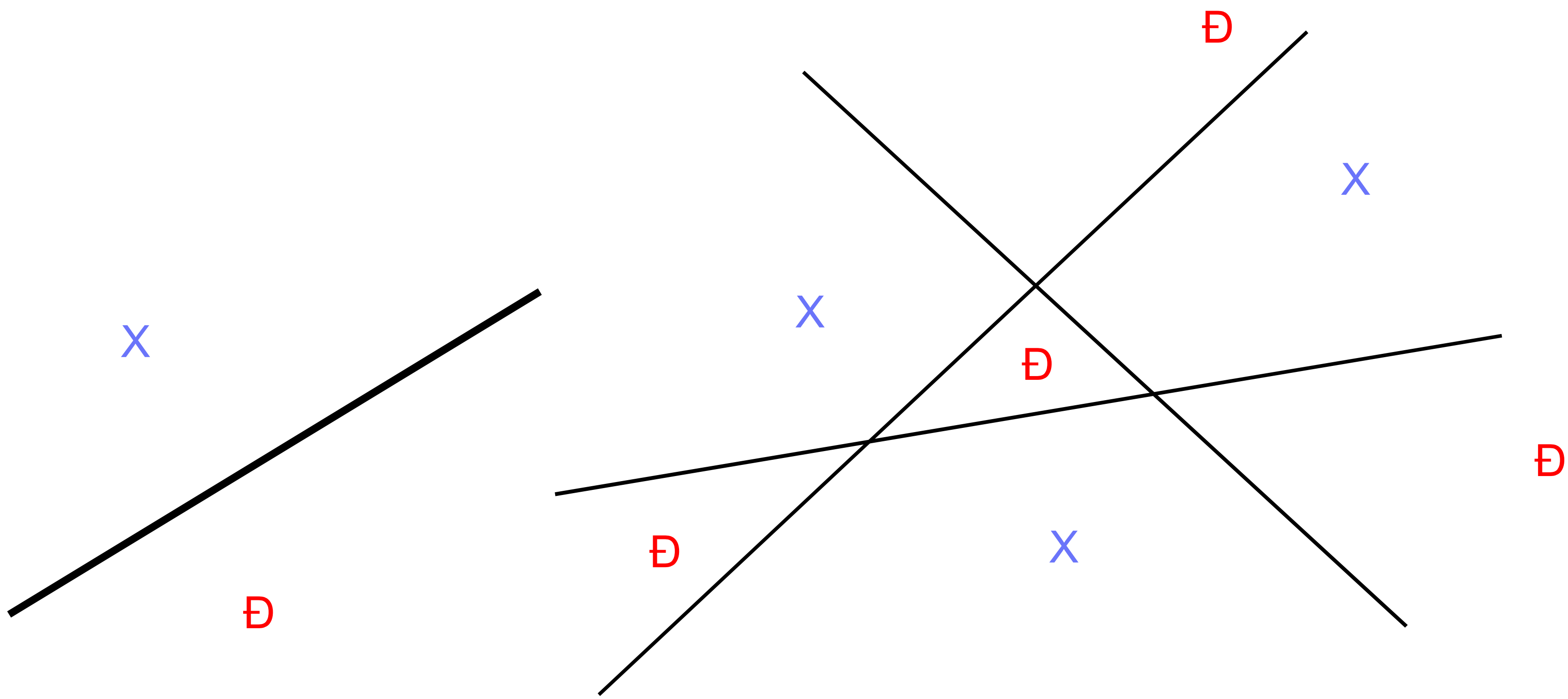
Ví dụ 3

- **Cơ sở qui nạp.** Khi n = 1, mặt phẳng được chia làm hai phần, một phần sẽ tô màu xanh, phần còn lại tô màu đỏ.
- **Chuyển qui nạp.** Giả sử khẳng định đúng với n-1, ta chứng minh khẳng định đúng với n.
- Thực vậy, trước hết ta vẽ n-1 đường thẳng. Theo giả thiết qui nạp có thể tô màu các phần sinh ra bởi hai màu thoả mãn điều kiện đặt ra.
- Bây giờ ta vẽ đường thẳng thứ n. Đường thẳng này chia mặt phẳng ra làm hai phần, gọi là phần A và B.

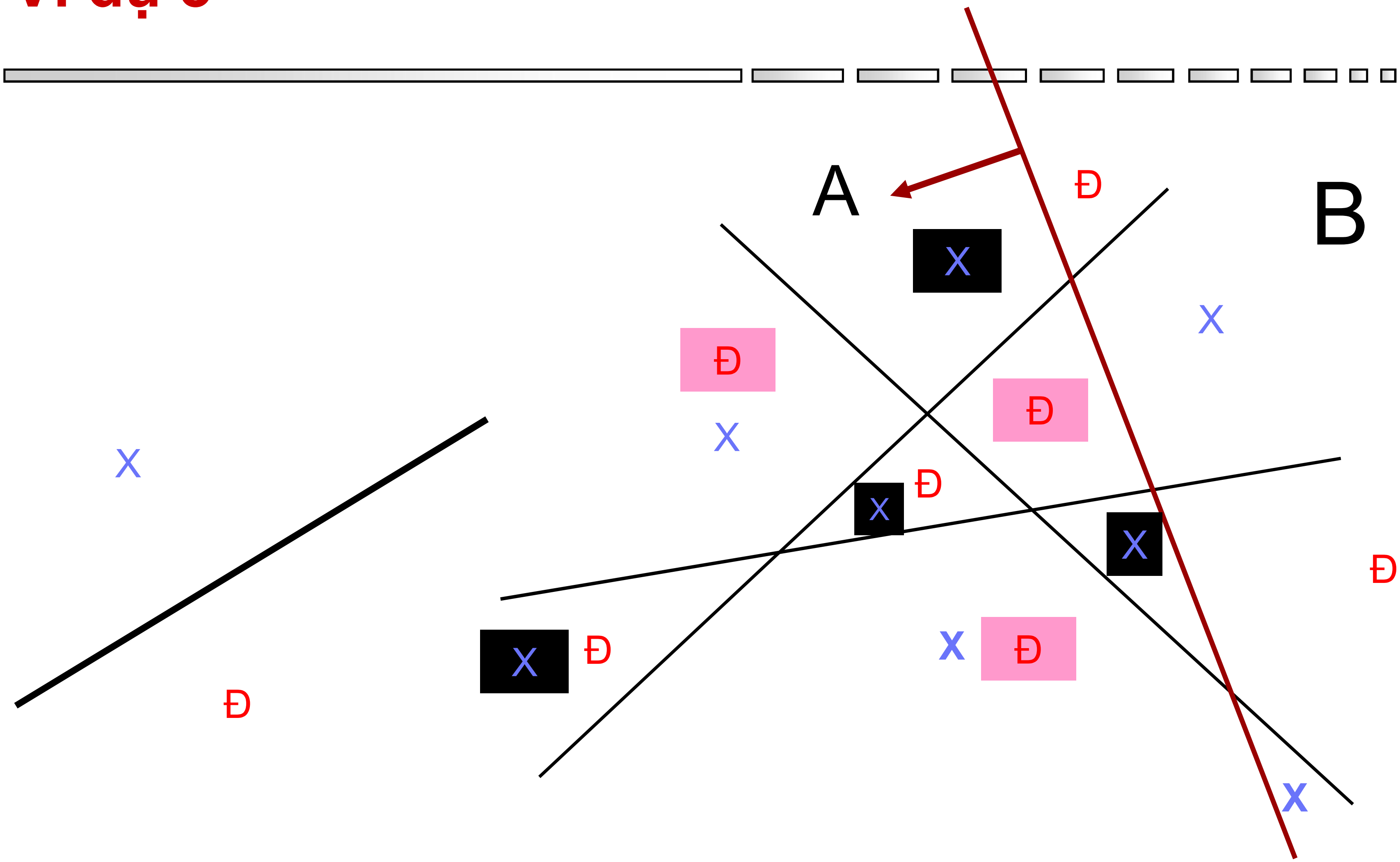
Ví dụ 3

- Các phần của mặt phẳng được chia bởi *n* đường thẳng ở bên nửa mặt phẳng B sẽ giữ nguyên màu đã tô trước đó. Trái lại, các phần trong nửa mặt phẳng A mỗi phần sẽ được tô màu đảo ngược xanh thành đỏ và đỏ thành xanh.
- Rõ ràng:
  - Hai phần có chung cạnh ở cùng một nửa mặt phẳng A hoặc B là không có chung màu.
  - Hai phần có chung cạnh trên đường thẳng thứ *n* rõ ràng cũng không bị tô cùng màu (do màu bên nửa A bị đảo ngược).
- Vậy $P(n)$ đúng. Theo qui nạp khẳng định đúng với mọi *n*.

Ví dụ 3



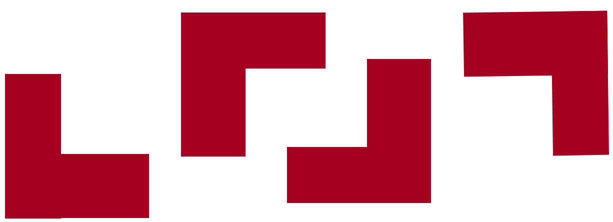
Ví dụ 3



Ví dụ 4. Phủ lưới $2^n \times 2^n$ bởi viên gạch chữ L

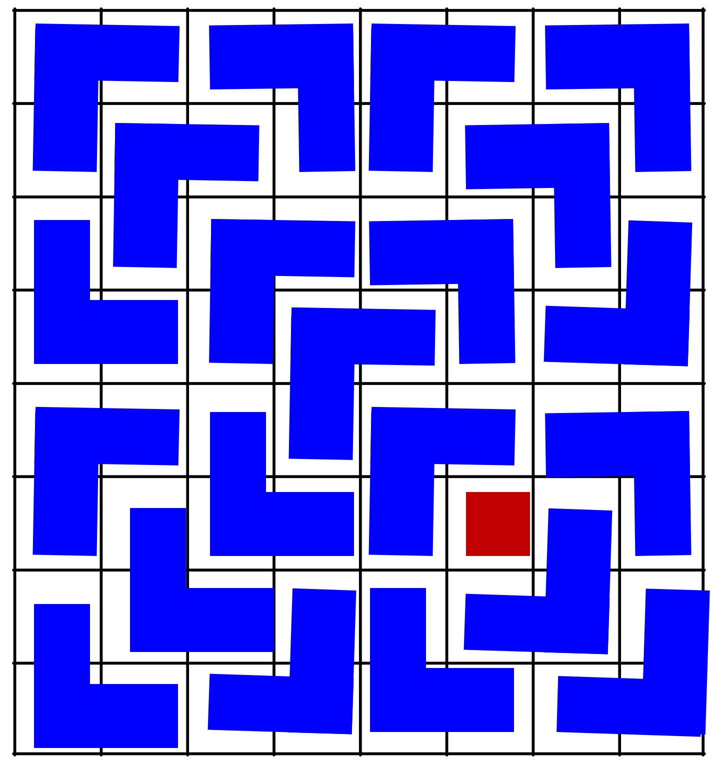
Cho lưới ô vuông kích thước $2^n \times 2^n$ bị đục mất một ô tùy ý.

Có thể phủ kín lưới bởi viên gạch chữ L ?



Khẳng định:

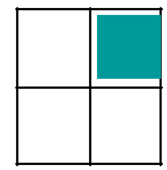
Luôn phủ được với mọi *n*.



Ví dụ: Lưới 8×8:

Chứng minh:

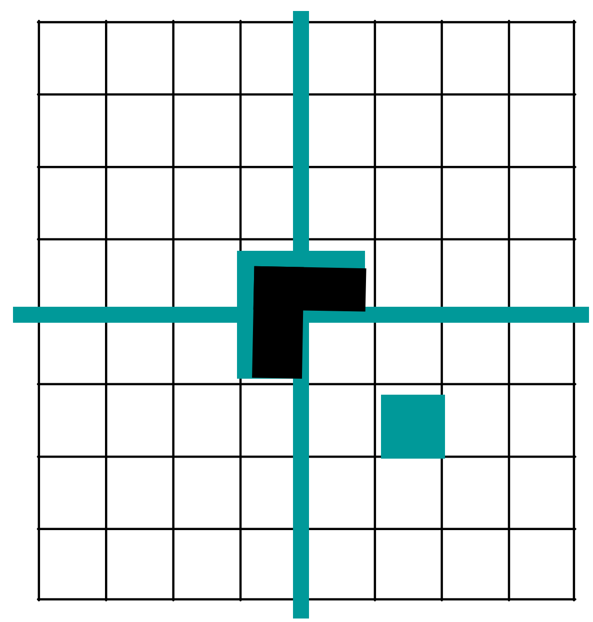
Cơ sở: Rõ ràng lưới 2×2 có thể phủ được.



Chuyển qui nạp: Giả sử có thể phủ kín lưới $2^n \times 2^n$. Để chỉ ra có thể phủ lưới $2^{n+1} \times 2^{n+1}$, ta chia lưới thành 4 lưới con:

Xét 3 ô ở trung tâm:

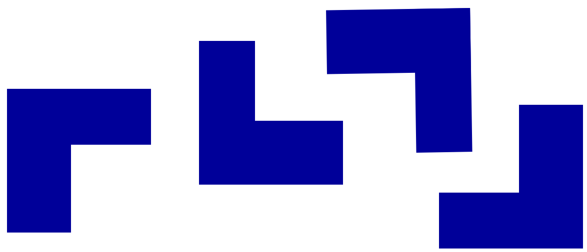
Đặt viên gạch L vào giữa.
Bốn lưới con mỗi lưới đều có kích thước $2^n \times 2^n$ và bị khuyết một ô, có thể phủ kín theo giả thiết qui nạp.

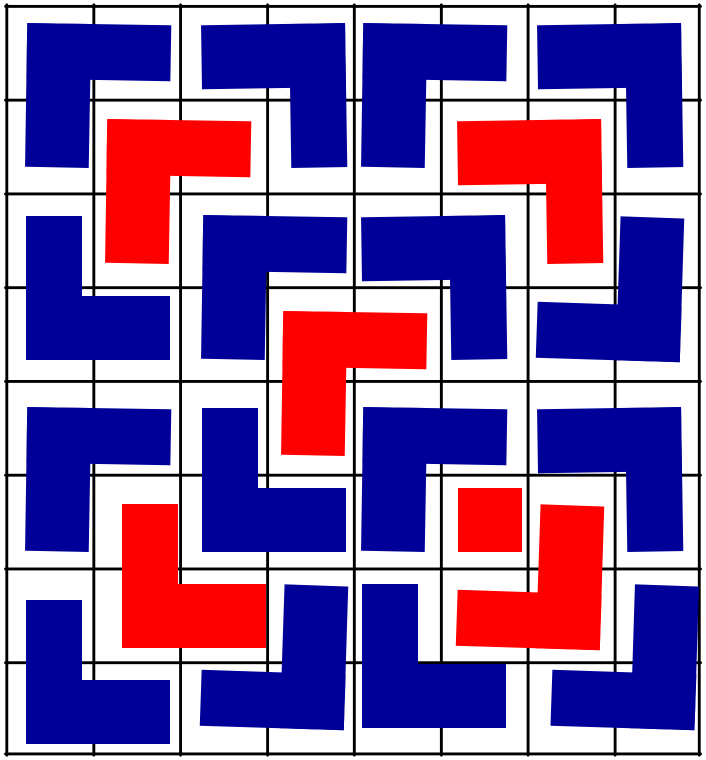


Lưu ý: Chứng minh bằng qui nạp mang tính xây dựng

Phủ lưới $2^n \times 2^n$

Chứng minh mang tính xây dựng. Nó chỉ ra cho ta cách phủ lưới sử dụng gạch chữ L:





Ví dụ lưới 8×8:

THUẬT TOÁN QUAY LUI

Backtracking Algorithm

SƠ ĐỒ THUẬT TOÁN

- Thuật toán quay lui (Backtracking Algorithm) là một thuật toán cơ bản đợc áp dụng để giải quyết nhiều vấn đề khác nhau.
- **Bài toán liệt kê (Q):** *Cho* $A_1, A_2, \ldots, A_n$ *là các tập hữu hạn. Ký hiệu*

  $X = A_1 \times A_2 \times \ldots \times A_n = \{ (x_1, x_2, \ldots, x_n):\ x_i \in A_i\ , i=1, 2, \ldots, n\}.$

  Giả sử ***P*** *là tính chất cho trên X. Vấn đề đặt ra là liệt kê tất cả các phần tử của X thoả mãn tính chất* ***P*****:**

  $D = \{\ x = (x_1, x_2, \ldots, x_n) \in X:\ x \text{ thoả mãn tính chất } \boldsymbol{P}\ \}.$
- Các phần tử của tập *D* đợc gọi là các lời giải chấp nhận đợc.

Ví dụ

- Bài toán liệt kê xâu nhị phân độ dài n dẫn về việc liệt kê các phần tử của tập

  $B^n = \{(x_1, ..., x_n): x_i \in \{0, 1\}, i=1, 2, ..., n\}.$

- Bài toán liệt kê các tập con m phần tử của tập $N = \{1, 2, ..., n\}$ đòi hỏi liệt kê các phần tử của tập:

  $S(m,n) = \{(x_1,..., x_m) \in N^m: 1 \leq x_1 < ... < x_m \leq n \}.$

- Tập các hoán vị của các số tự nhiên 1, 2, ..., n là tập

  $\Pi_n = \{(x_1,..., x_n) \in N^n: x_i \neq x_j ; i \neq j \}.$

Lời giải bộ phận

- **Định nghĩa.** Ta gọi lời giải bộ phận cấp k ($0 \leq k \leq n$) là bộ có thứ tự gồm k thành phần $(a_1, a_2, ..., a_k)$, trong đó $a_i \in A_i$, $i = 1, 2, ..., k$.
- Khi $k = 0$, lời giải bộ phận cấp 0 đọc ký hiệu là () và còn đọc gọi là lời giải rỗng.
- Nếu $k = n$, ta có lời giải đầy đủ hay đơn giản là một lời giải của bài toán.

Ý tưởng chung

- Thuật toán quay lui được xây dựng dựa trên việc xây dựng dần từng thành phần của lời giải.
- Thuật toán bắt đầu từ lời giải rỗng (). Trên cơ sở tính chất P ta xác định được những phần tử nào của tập A_1 có thể chọn vào vị trí thứ nhất của lời giải. Những phần tử như vậy ta sẽ gọi là những ứng cử viên (viết tắt là UCV) vào vị trí thứ nhất của lời giải. Ký hiệu tập các UCV vào vị trí thứ nhất của lời giải là S_1. Lấy $a_1 \in S_1$, bổ sung nó vào lời giải rỗng đang có ta thu được lời giải bộ phận cấp 1: (a_1).

Bước tổng quát

- Tại bước tổng quát, giả sử ta đang có lời giải bộ phận cấp k-1: $(a_1, a_2, ..., a_{k-1})$.
- Trên cơ sở tính chất P ta xác định được những phần tử nào của tập A_k có thể chọn vào vị trí thứ k của lời giải. Những phần tử như vậy ta sẽ gọi là những ứng cử viên (viết tắt là UCV) vào vị trí thứ k của lời giải khi k-1 thành phần đầu của nó đã được chọn là $(a_1, a_2, ..., a_{k-1})$. Ký hiệu tập các ứng cử viên này là S_k.

Xét hai tình huống

- $S_k \neq \varnothing$. Khi đó lấy $a_k \in S_k$, bổ sung nó vào lời giải bộ phận cấp k-1 đang có

$$(a_1, a_2, ..., a_{k-1})$$

ta thu đợc lời giải bộ phận cấp k:

$$(a_1, a_2, ..., a_{k-1}, a_k).$$

—Nếu $k = n$ thì ta thu đợc một lời giải,

—Nếu $k < n$, ta tiếp tục đi xây dựng thành phần thứ k+1 của lời giải.

Tình huống ngõ cụt

- $S_k=\varnothing$. Điều đó có nghĩa là lời giải bộ phận (a_1, a_2, ..., a_{k-1}) không thể tiếp tục phát triển thành lời giải đầy đủ. Trong tình huống này ta ***quay trở lại*** tìm ứng cử viên mới vào vị trí thứ k-1 của lời giải.
- Nếu tìm thấy UCV như vậy thì bổ sung nó vào vị trí thứ k-1 rồi lại tiếp tục đi xây dựng thành phần thứ k.
- Nếu không tìm được thì ta lại ***quay trở lại*** thêm một bước nữa tìm UCV mới vào vị trí thứ k-2, ... Nếu quay lại tận lời giải rỗng mà vẫn không tìm được UCV mới vào vị trí thứ 1, thì thuật toán kết thúc.

Thuật toán quay lui

Thuật toán quay lui có thể mô tả trong thủ tục đệ qui sau đây

procedure Bactrack(k: integer);
begin
    Xây dựng S_k;
        for y $\in S_k$ do (* Với mỗi UCV y từ S_k *)
        begin
            a_k := y;
            if k = n then <Ghi nhận lời giải (a_1, a_2, ..., a_k) >
            else Backtrack(k+1);
        end;
end;

Lệnh gọi để thực hiện thuật toán quay lui là:

Bactrack(1)

Hai vấn đề mấu chốt

- Để cài đặt thuật toán quay lui giải các bài toán tổ hợp cụ thể ta cần giải quyết hai vấn đề cơ bản sau:
  - Tìm thuật toán xây dựng các tập UCV S_k.
  - Tìm cách mô tả các tập này để có thể cài đặt thao tác liệt kê các phần tử của chúng (cài đặt vòng lặp qui ớc **for y** $\in$ $\mathbf{S_k}$ **do**).
- Hiệu quả của thuật toán liệt kê phụ thuộc vào việc ta có xác định đợc chính xác các tập UCV này hay không.

Chú ý

- Nếu chỉ cần tìm một lời giải thì cần tìm cách chấm dứt các thủ tục gọi đệ qui lồng nhau sinh bởi lệnh gọi **Backtrack(1)** sau khi ghi nhận đợc lời giải đầu tiên.
- Nếu kết thúc thuật toán mà ta không thu đợc một lời giải nào thì điều đó có nghĩa là bài toán không có lời giải.
- Thuật toán dễ dàng mở rộng cho bài toán liệt kê trong đó lời giải có thể mô tả nh là bộ (a_1, a_2, ..., a_n,...) độ dài hữu hạn, tuy nhiên giá trị của độ dài là không biết trớc và các lời giải cũng không nhất thiết phải có cùng độ dài.

Chú ý

- Khi đó chỉ cần sửa lại câu lệnh

if k = n then <Ghi nhận lời giải (a_1, a_2, ..., a_k) >

else Backtrack(k+1);

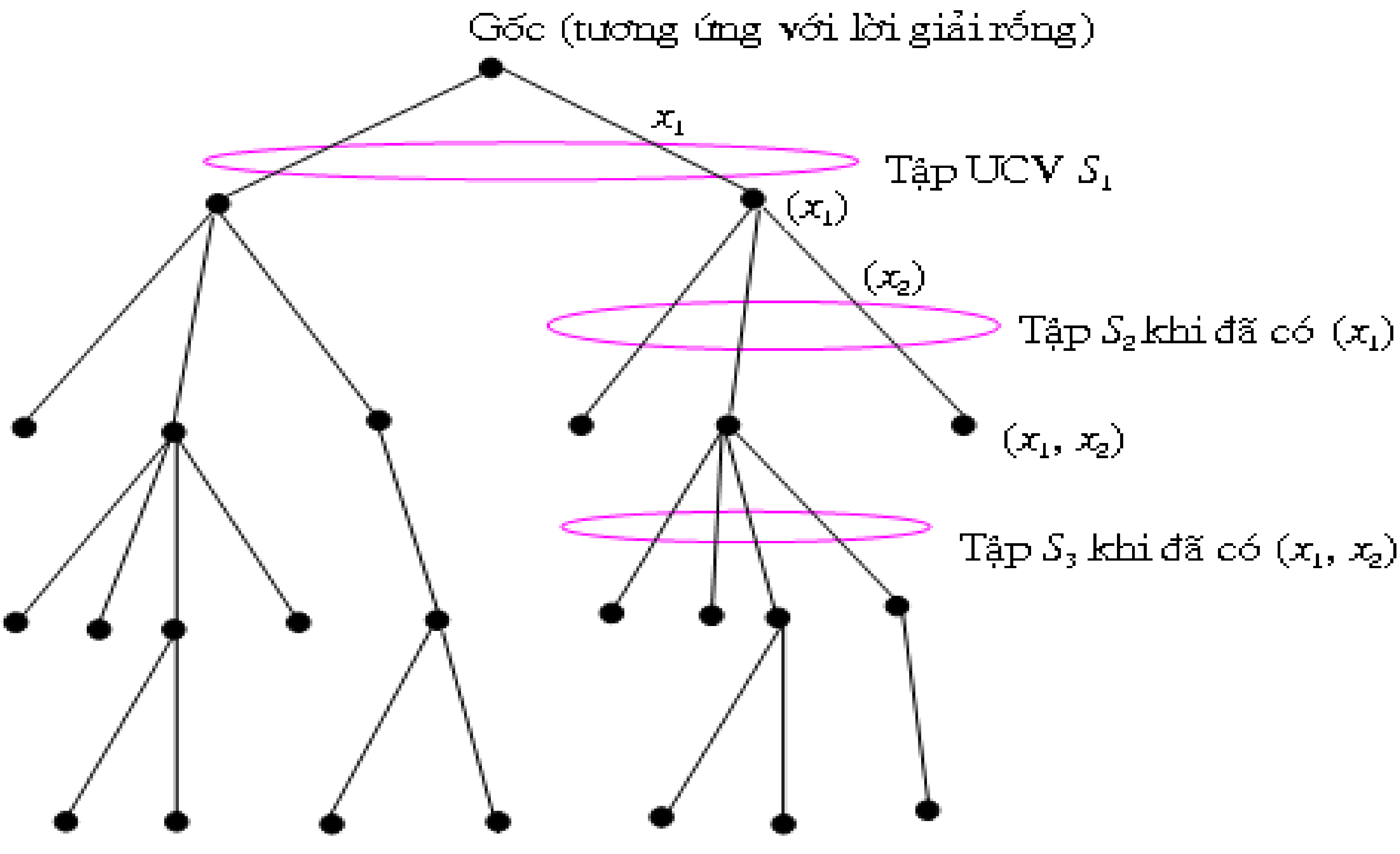
thành

if <(a_1, a_2, ..., a_k) là lời giải> then <Ghi nhận (a_1, a_2, ..., a_k) >

else Backtrack(k+1);

- Ta cần xây dựng hàm nhận biết (a_1, a_2, ..., a_k) đã là lời giải hay cha.

Cây liệt kê lời giải theo thuật toán quay lui

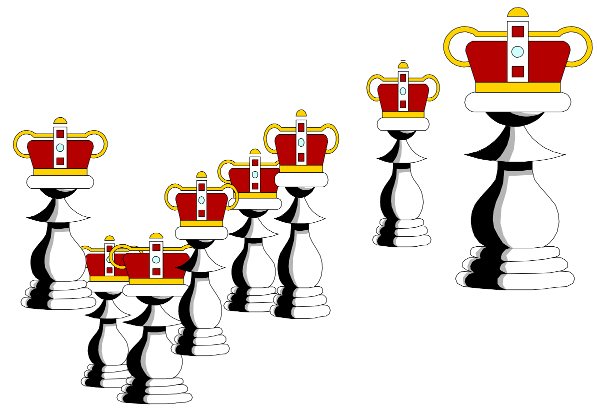


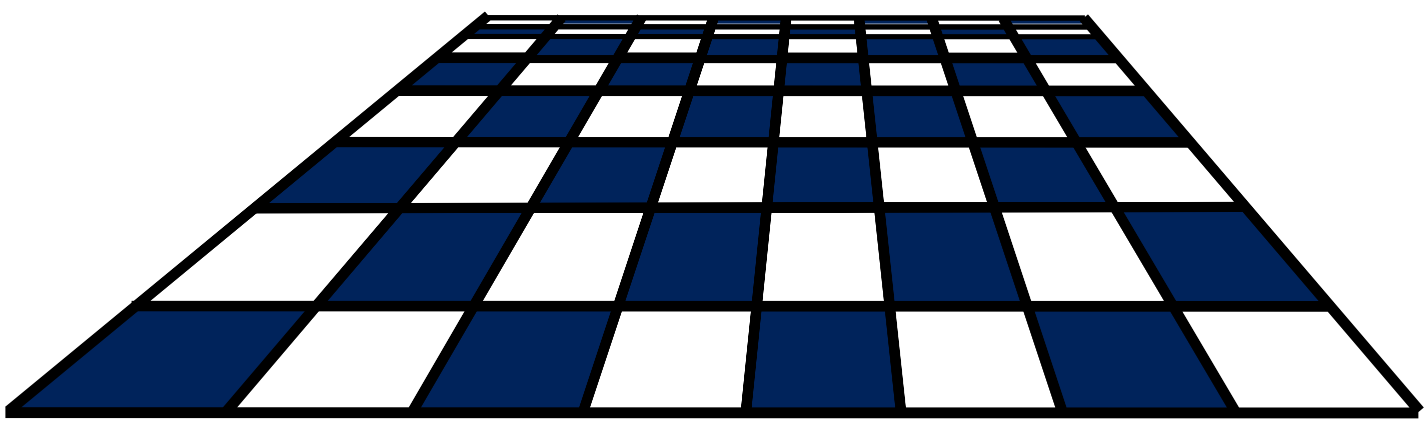
Bài toán xếp hậu

- Liệt kê tất cả các cách xếp *n* quân Hậu trên bàn cờ *n*×*n* sao cho chúng không ăn được lẫn nhau, nghĩa là sao cho không có hai con nào trong số chúng nằm trên cùng một dòng hay một cột hay một đường chéo của bàn cờ.

The *n*-Queens Problem

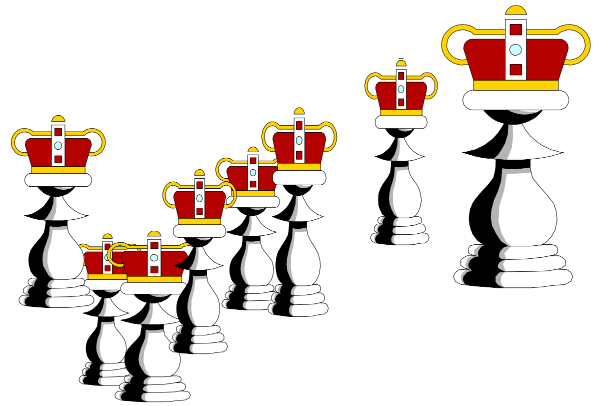
- Giả sử ta có 8 con hậu...
- ...và bàn cờ quốc tế

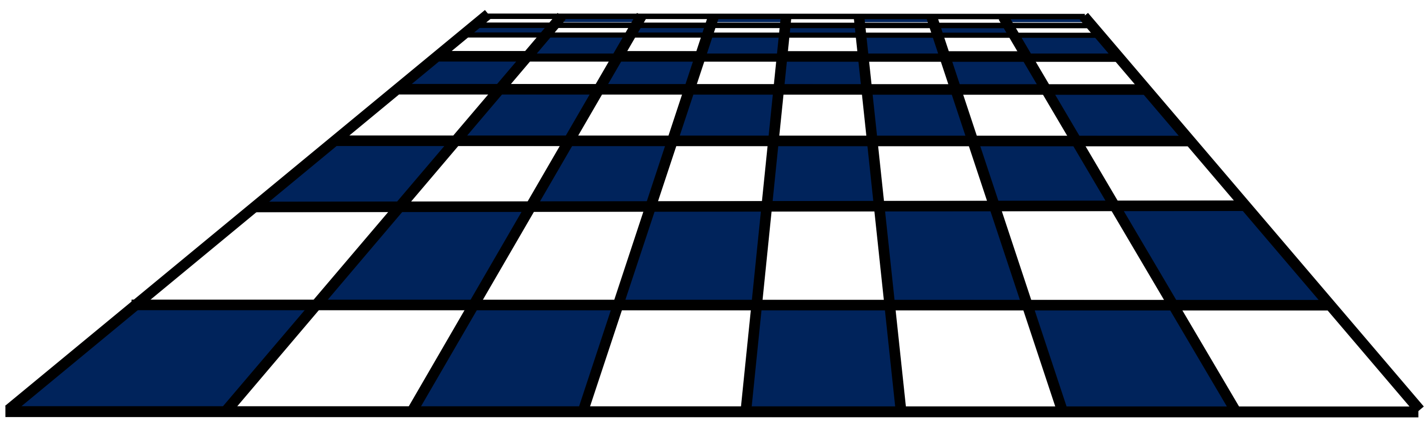




The *n*-Queens Problem

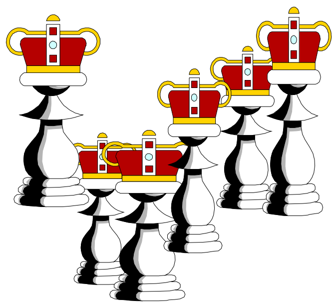
Có thể xếp các con hậu sao cho không có hai con nào ăn nhau hay không ?

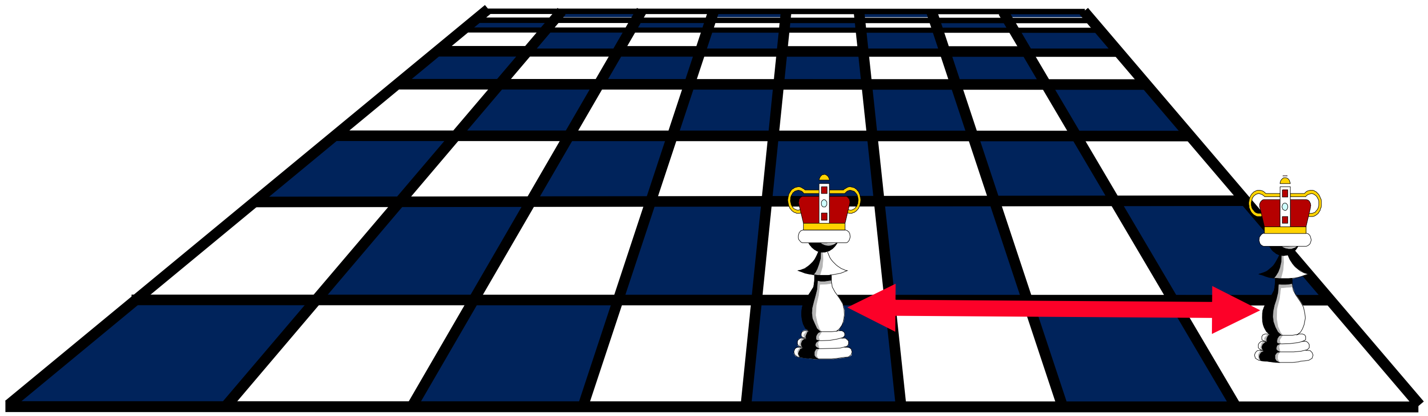




The n-Queens Problem

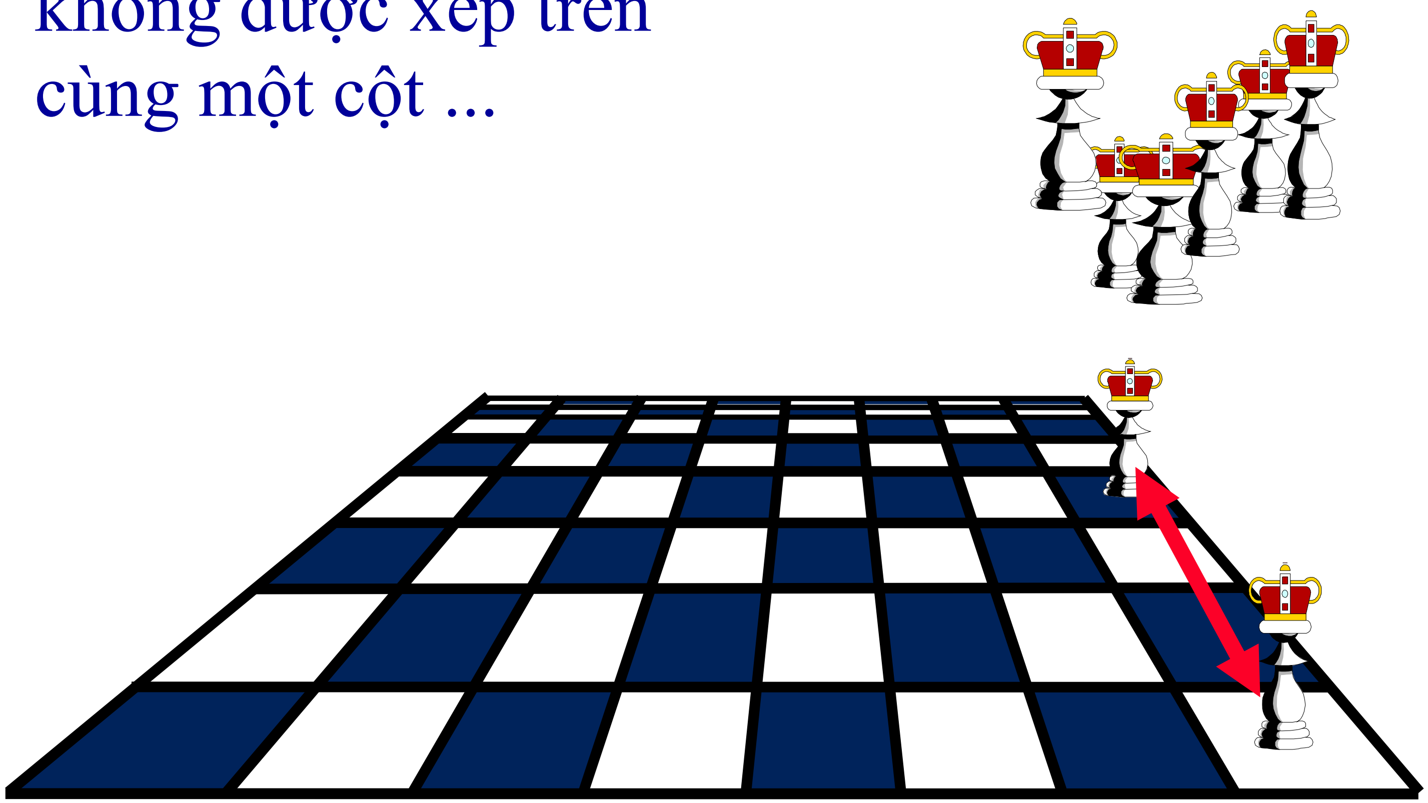
Hai con hậu bất kỳ
không được xếp trên
cùng một dòng ...





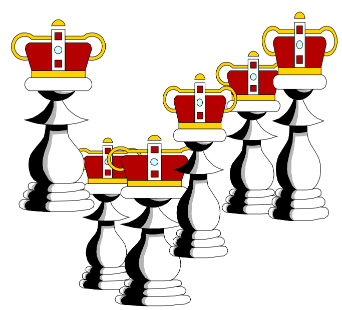
The n-Queens Problem

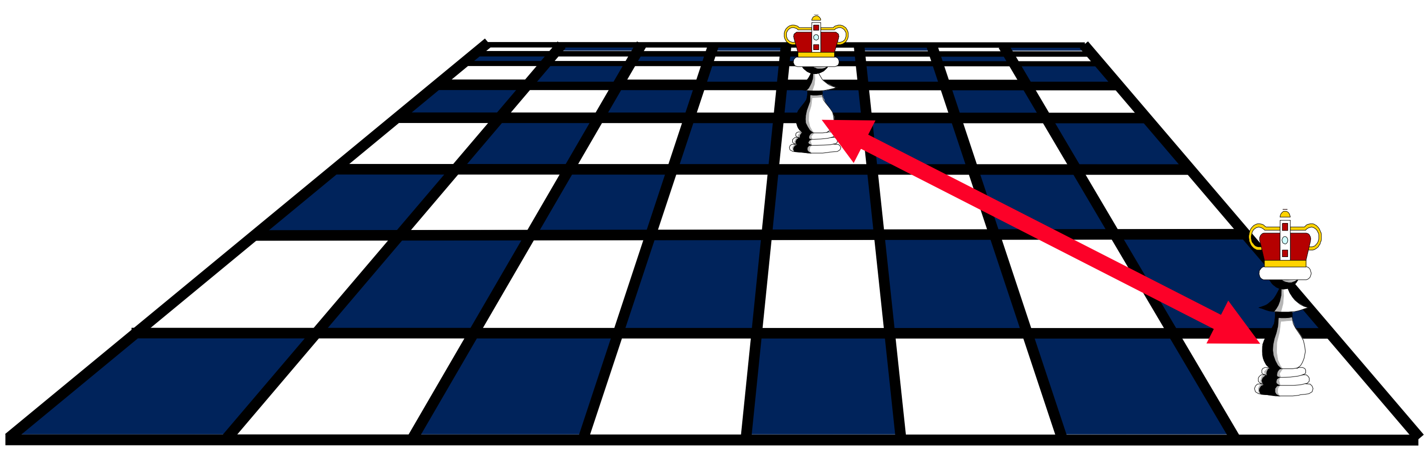
Hai con hậu bất kỳ không được xếp trên cùng một cột ...



The n-Queens Problem

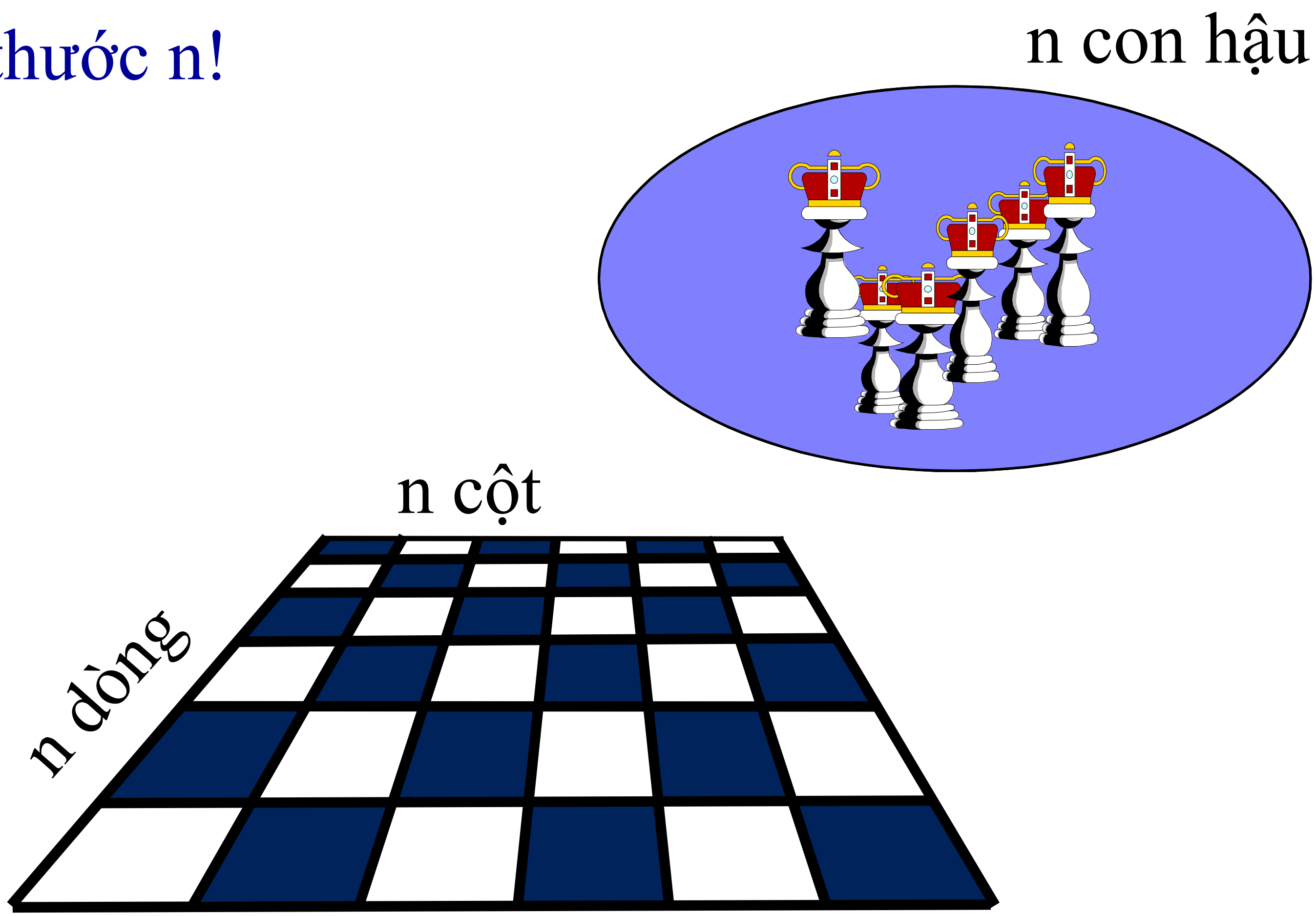
Hai con hậu bất kỳ
không được xếp trên
cùng một dòng, một cột
hay một đường chéo!





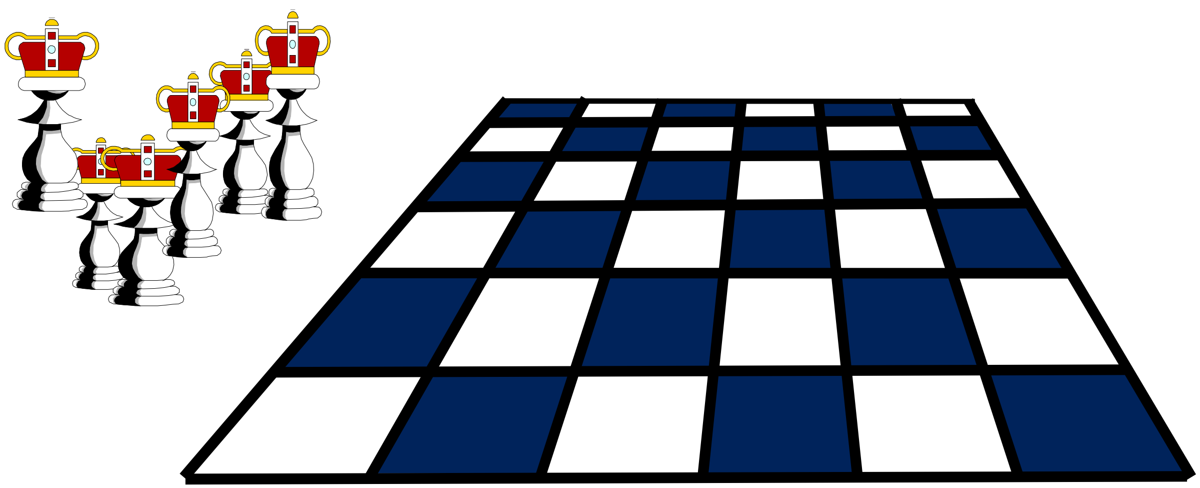
The n-Queens Problem

Kích thước n!



The n-Queens Problem

Xét bài toán xếp n con hậu lên bàn cờ kích thước n x n.



Biểu diễn lời giải

- Đánh số các cột và dòng của bàn cờ từ 1 đến n. Một cách xếp hậu có thể biểu diễn bởi bộ có n thành phần $(a_1, a_2, ..., a_n)$, trong đó a_i là toạ độ cột của con Hậu ở dòng i.
- Các điều kiện đặt ra đối với bộ $(a_1, a_2, ..., a_n)$:
  - $a_i \neq a_j$, với mọi $i \neq j$ (nghĩa là hai con hậu ở hai dòng i và j không đọc nằm trên cùng một cột);
  - $| a_i \square a_j | \neq | i \square j |$, với mọi $i \neq j$ (nghĩa là hai con hậu ở hai ô (a_i, i) và (a_j, j) không đọc nằm trên cùng một đờng chéo).

Phát biểu bài toán

- Như vậy bài toán xếp Hậu dẫn về bài toán liệt kê các phần tử của tập:

  $D=\{(a_1, a_2, \ldots, a_n)\in N^n\colon a_i \neq a_j$ và $|a_i - a_j| \neq |i - j|, i \neq j \}$.

Hàm nhận biết ứng cử viên

```c
int UCVh(int j, int k) {
// UCVh nhận giá trị 1
// khi và chỉ khi ∈ Sk
  int i;
  for (i=1; i<k; i++)
     if ((j == a[i]) || (fabs(j-a[i])== k-i))
          return 0;
  return 1;
}
```

```pascal
function UCVh(j,k: integer): boolean;
(* UCVh nhận giá trị true
   khi và chỉ khi  j ∈ Sk *)
var i: integer;
begin
   for i:=1 to  k-1 do
      if (j = a[i]) or (abs(j-a[i])= k-i) then
      begin
         UCVh:= false; exit;
      end;
  UCVh:= true;
end;
```

Chương trình trên PASCAL/C

```pascal
var   n, count: integer;
      a: array[1..20] of integer;
procedure Ghinhan;
var i: integer;
begin
      count := count+1; write(count:5, '. ');
      for i := 1 to n do write(a[i]:3); writeln;
end;

function UCVh(j,k: integer): boolean;
(* UCVh nhận giá trị true khi và chỉ khi j ∈ S_k *)
var i: integer;
begin
  for i:=1 to  k-1 do
     if (j = a[i]) or (abs(j-a[i])= k-i) then
     begin
        UCVh:= false; exit;
     end;
  UCVh:= true;
end;
```

```c
#include <stdio.h>
#include <math.h>
int   n, count;
int   a[20];
int Ghinhan() {
  int i;
  count++; printf("%i. ",count);
  for (i=1; i<=n;i++) printf("%i  ",a[i]);
  printf("\n");
}

int UCVh(int j, int k) {
/* UCVh nhận giá trị 1 khi vá chỉ khi j ∈ S_k */
  int i;
  for (i=1; i<k; i++)
     if ((j == a[i]) || (fabs(j-a[i])== k-i)) return 0;
  return 1;
}
```

```
procedure Hau(i: integer);
var j: integer;
begin
   for j := 1 to n do
     if  UCVh(j, i) then
      begin
         a[i] := j;
         if  i = n then Ghinhan
        else Hau(i+1);
    end;
end;

BEGIN     {Main program}
  write('n = '); readln(n);
  count := 0; Hau(1);
  If count = 0 then
    writeln('NO SOLUTION');
  write('Gõ Enter để kết thúc... ');
  readln;
END.
```

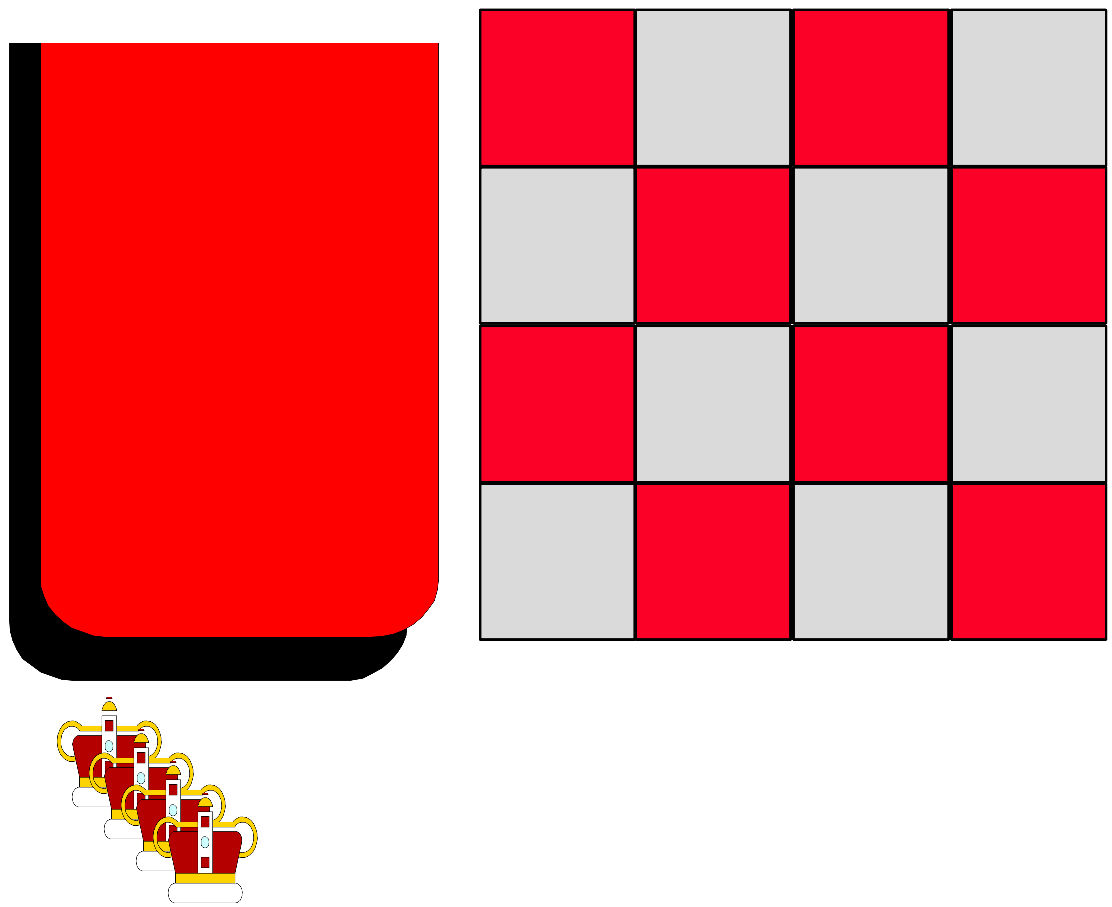
```
int Hau(int i){
   int j;
  for (j=1; j<=n; j++)
       if  (UCVh(j, i)){
           a[i] = j;
           if  (i == n) Ghinhan();
           else Hau(i+1);
       }
}

int main() {
   printf("Input n = "); scanf("%i",&n);
   printf("\n==== RESULT ====\n");
  count = 0; Hau(1);
   if (count == 0) printf("No solution!\n");
   getchar();
   printf("\n Press Enter to finish... ");
   getchar();
}
```

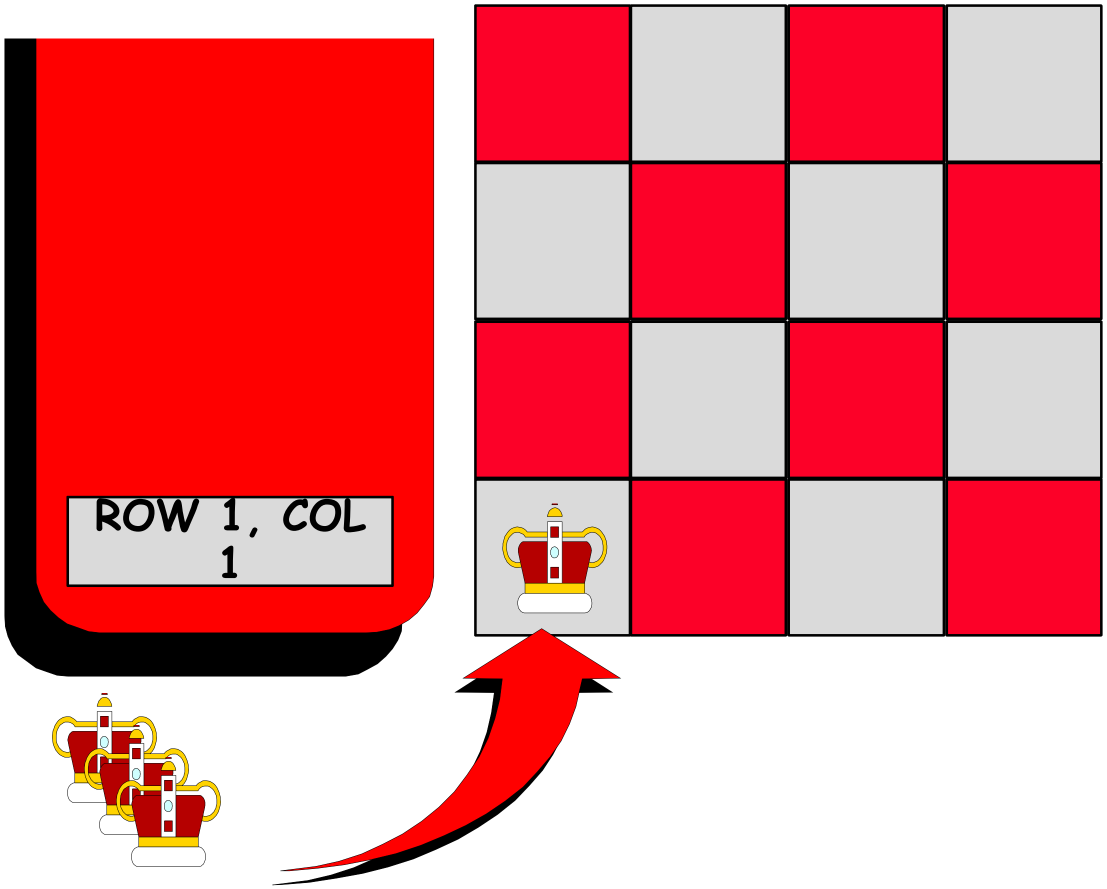
Chú ý

- Rõ ràng là bài toán xếp hậu không phải là luôn có lời giải, chẳng hạn bài toán không có lời giải khi *n*=2, 3. Do đó điều này cần được thông báo khi kết thúc thuật toán.
- Thuật toán trình bày ở trên là chưa hiệu quả. Nguyên nhân là ta đã không xác định được chính xác các tập UCV vào các vị trí của lời giải.

Thuật toán làm việc như thế nào

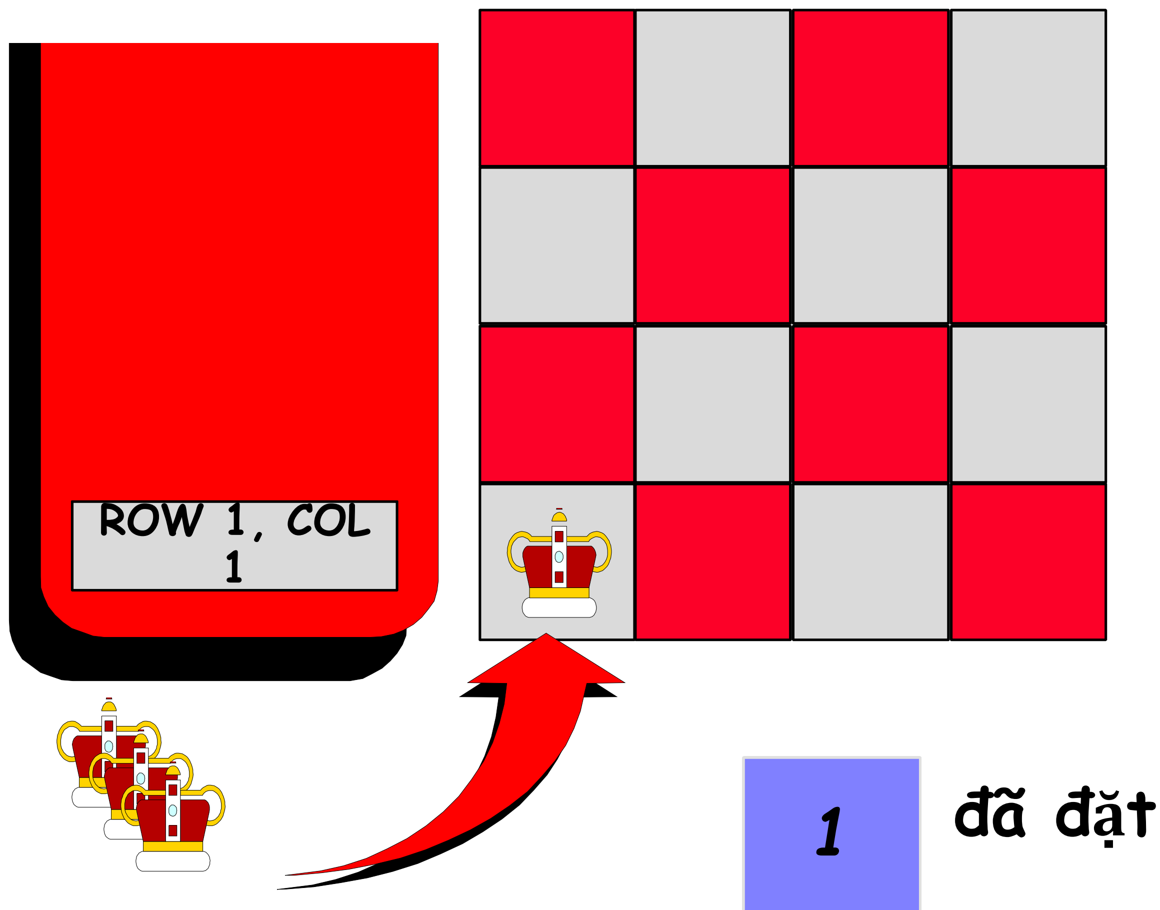


Thuật toán làm việc như thế nào



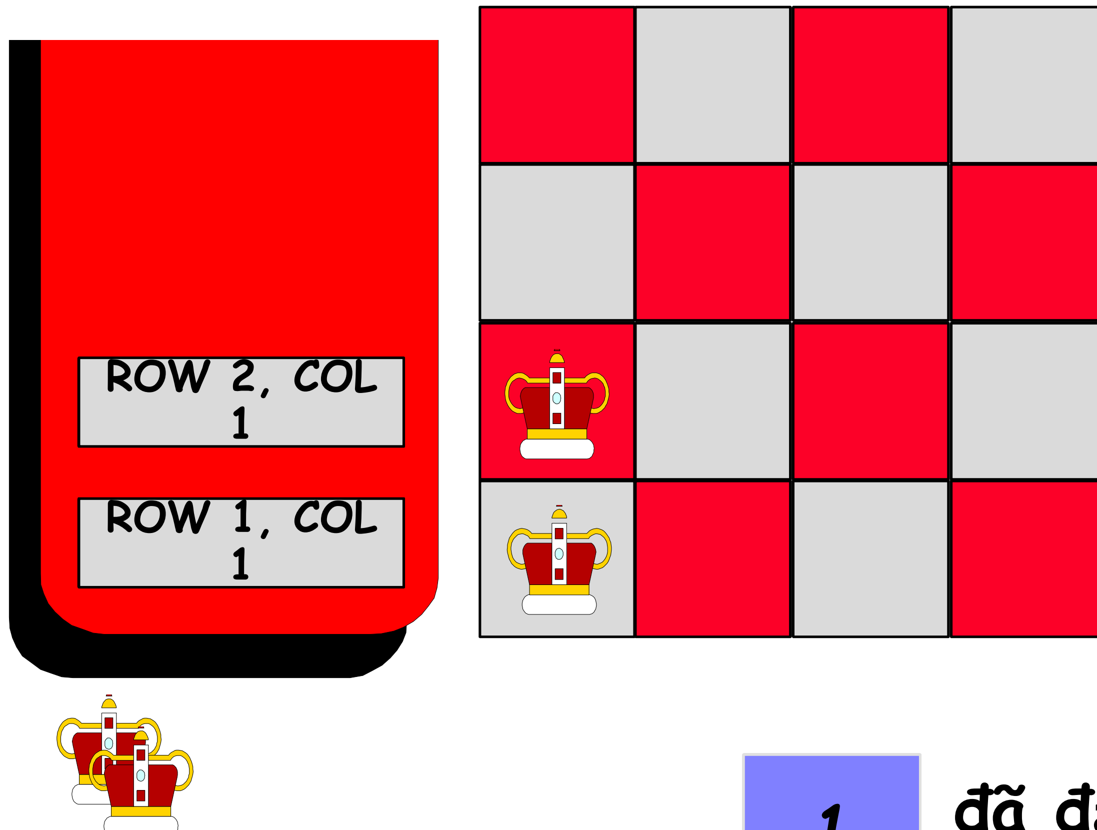
Thuật toán làm việc như thế nào

- Xếp con hậu ở dòng 1 vào vị trí cột 1



Thuật toán làm việc như thế nào

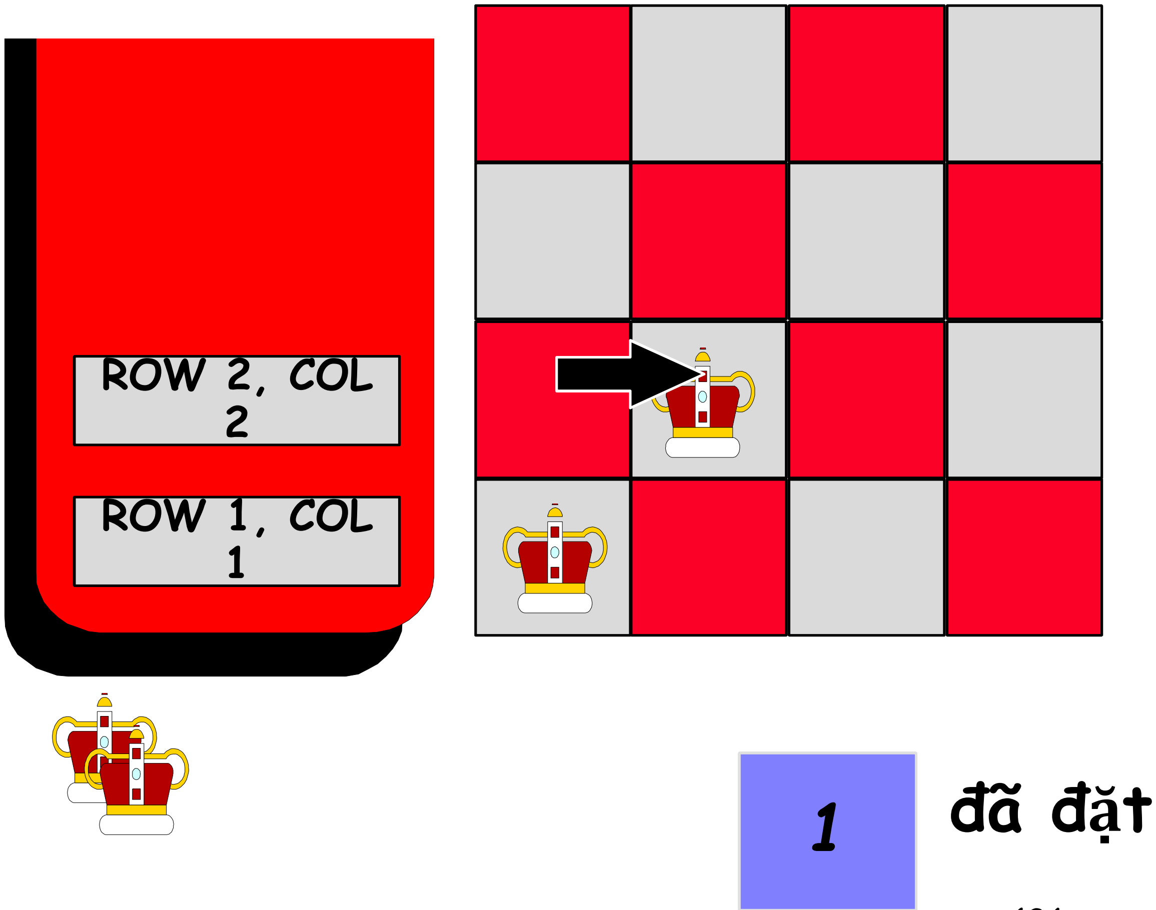
- Thử xếp con hậu ở dòng 2 vào vị trí cột 1



1 đã đặt

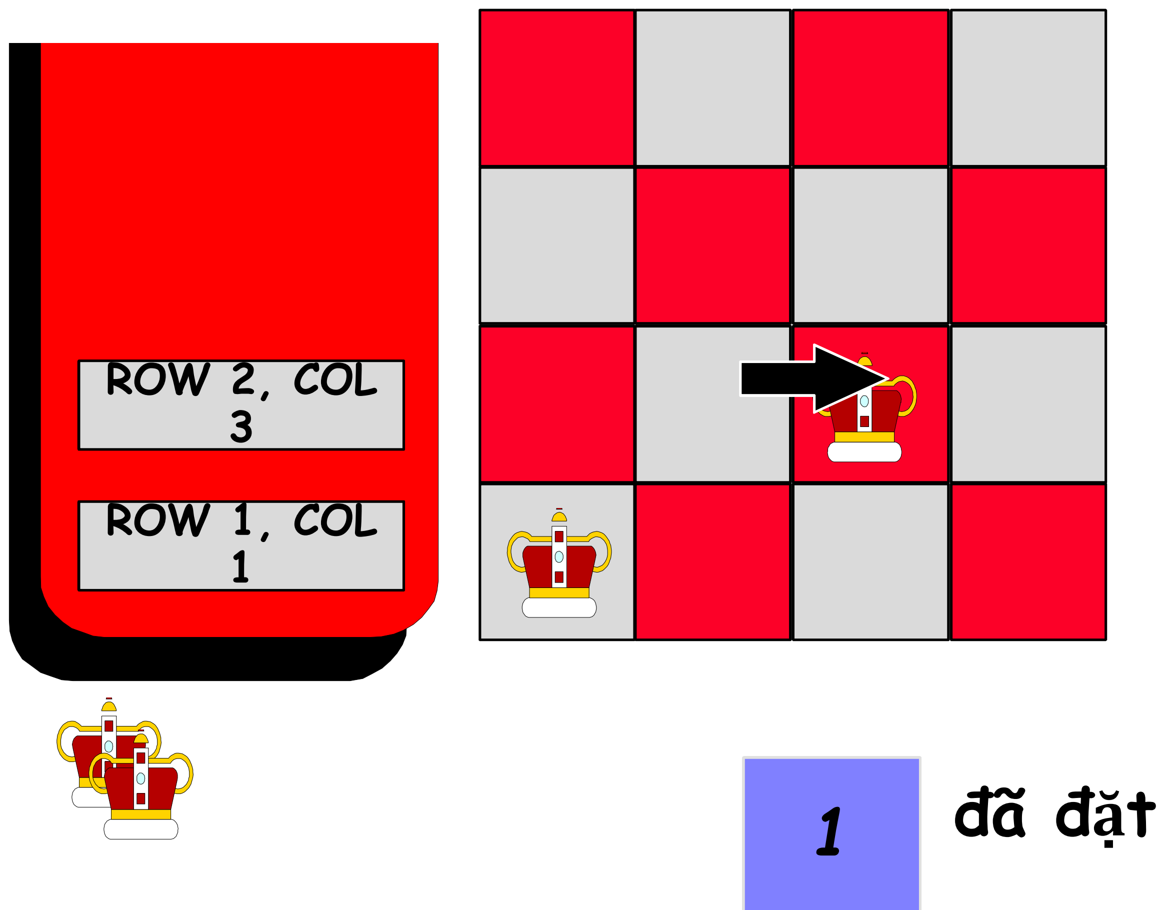
Thuật toán làm việc như thế nào

- Thử xếp con hậu ở dòng 2 vào vị trí cột 2



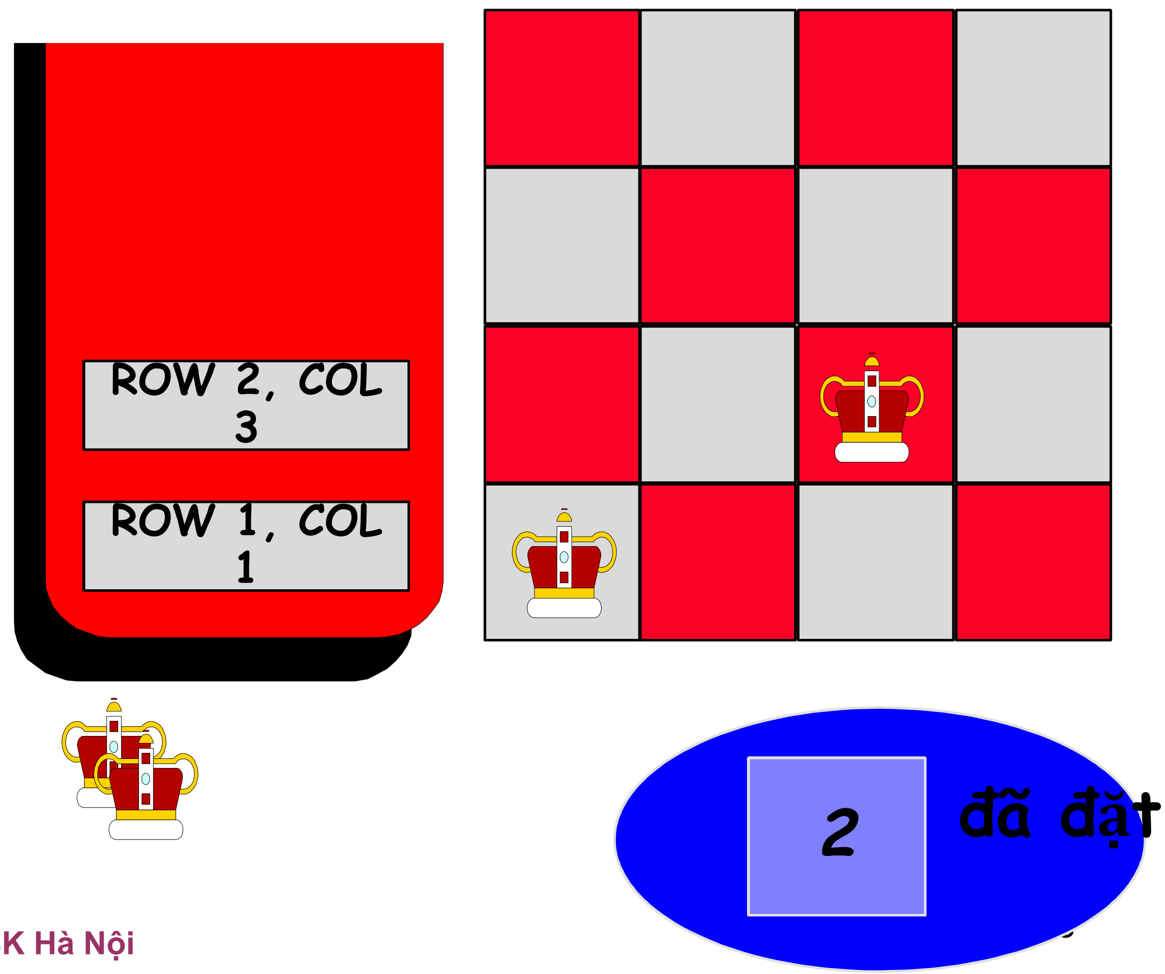
Thuật toán làm việc như thế nào

- Thử xếp con hậu ở dòng 2 vào vị trí cột 3



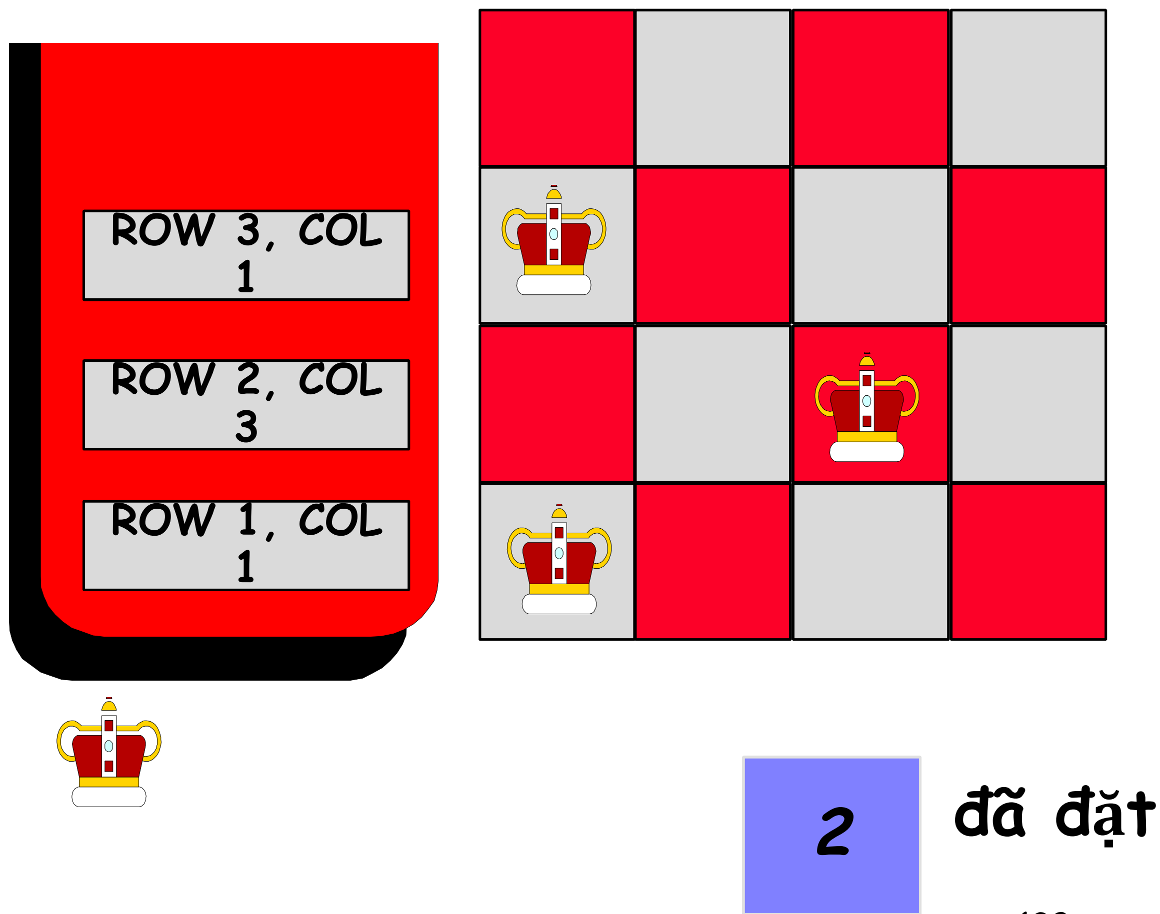
Thuật toán làm việc như thế nào

- Chấp nhận xếp con hậu ở dòng 2 vào vị trí cột 3



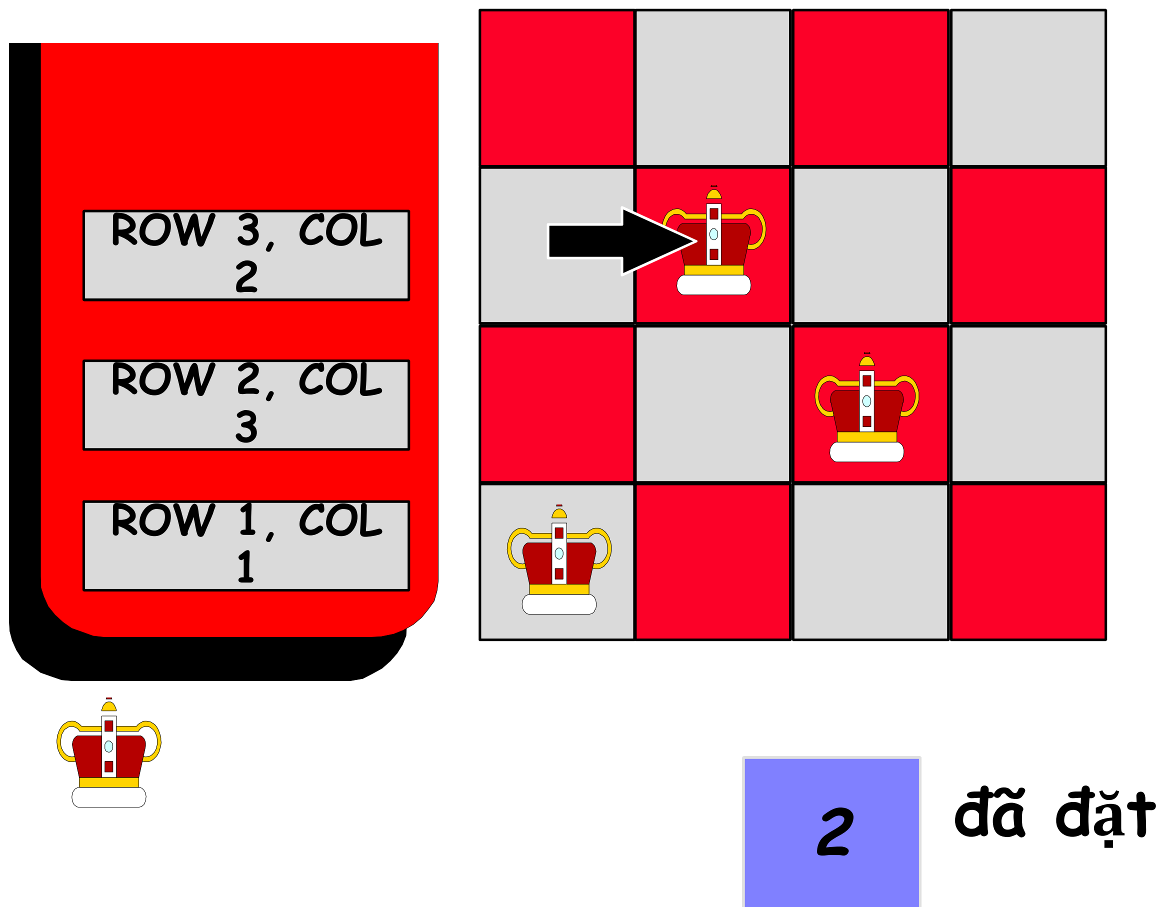
Thuật toán làm việc như thế nào

Thử xếp con hậu ở dòng 3 vào cột đầu tiên



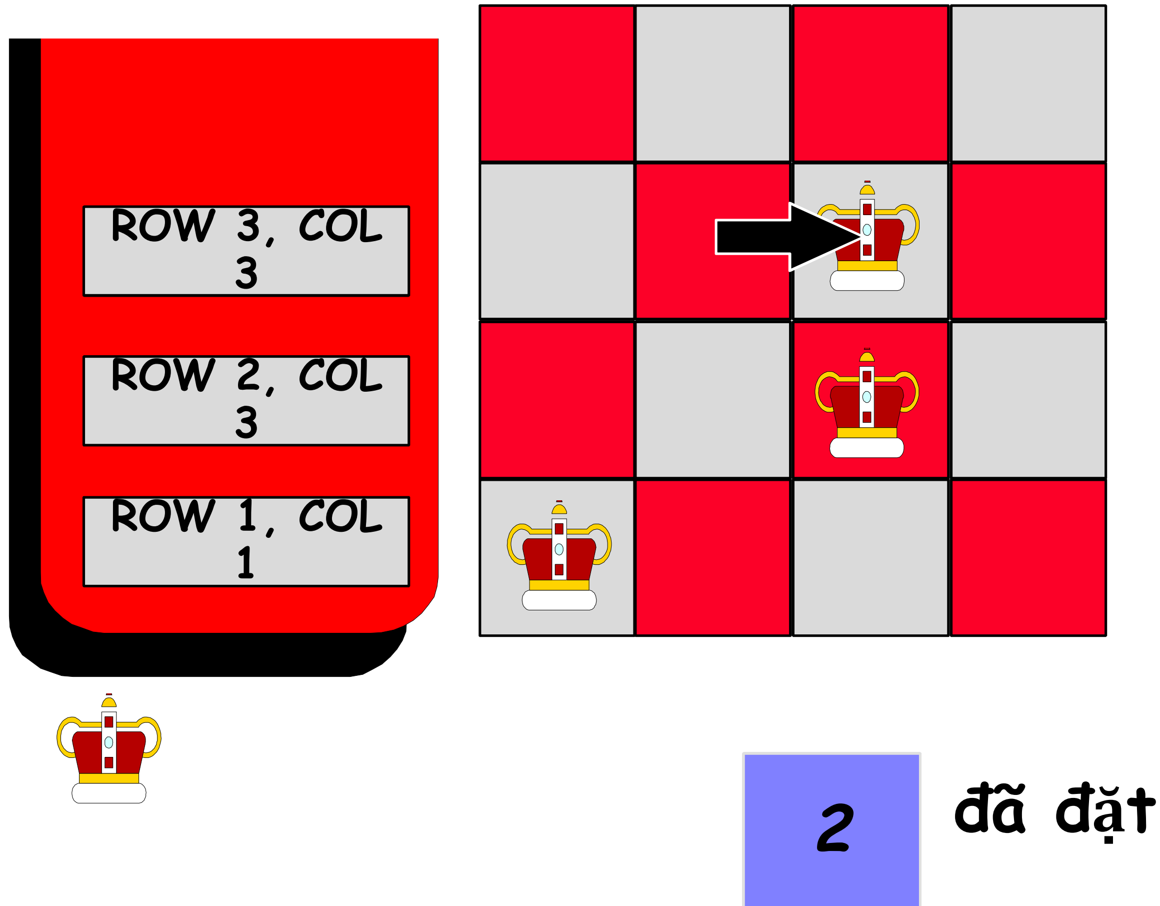
Thuật toán làm việc như thế nào

Thử cột tiếp theo



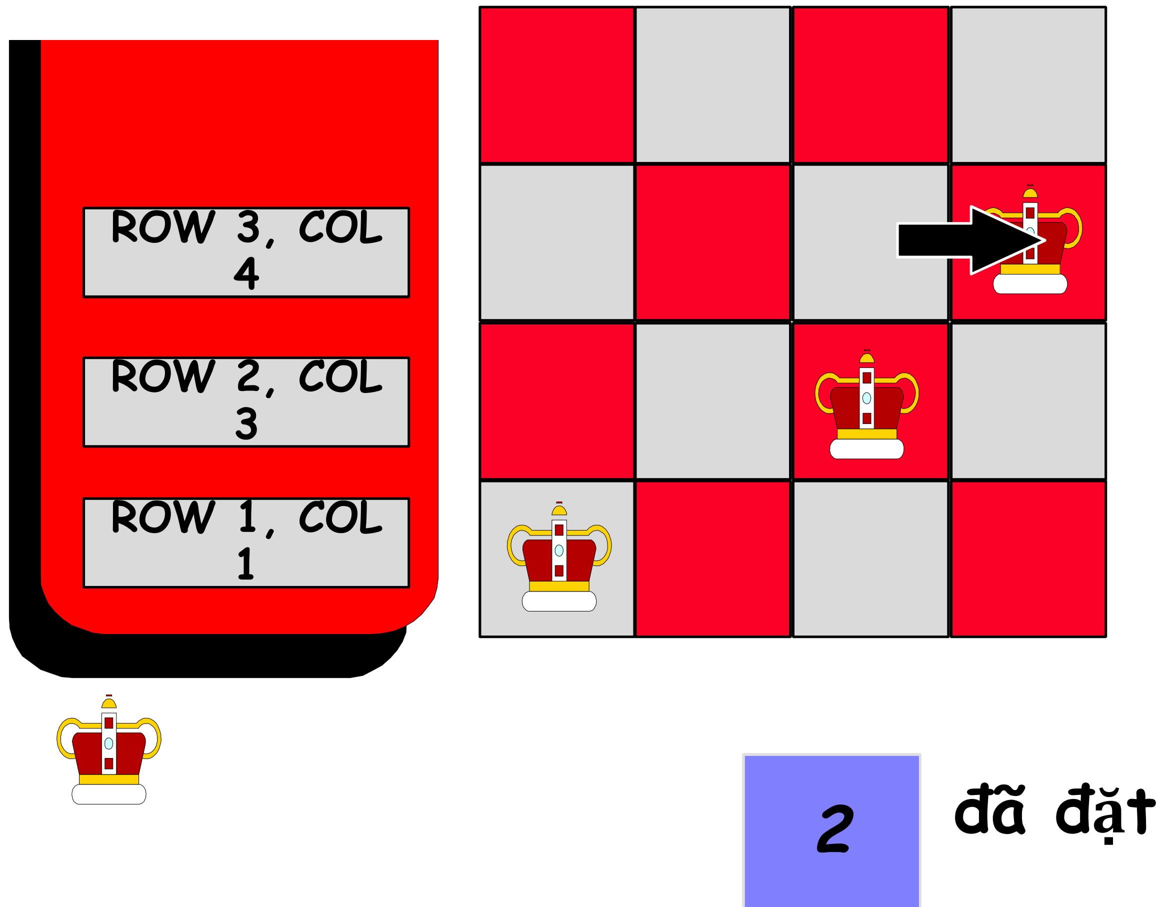
Thuật toán làm việc như thế nào

- Thử cột tiếp theo



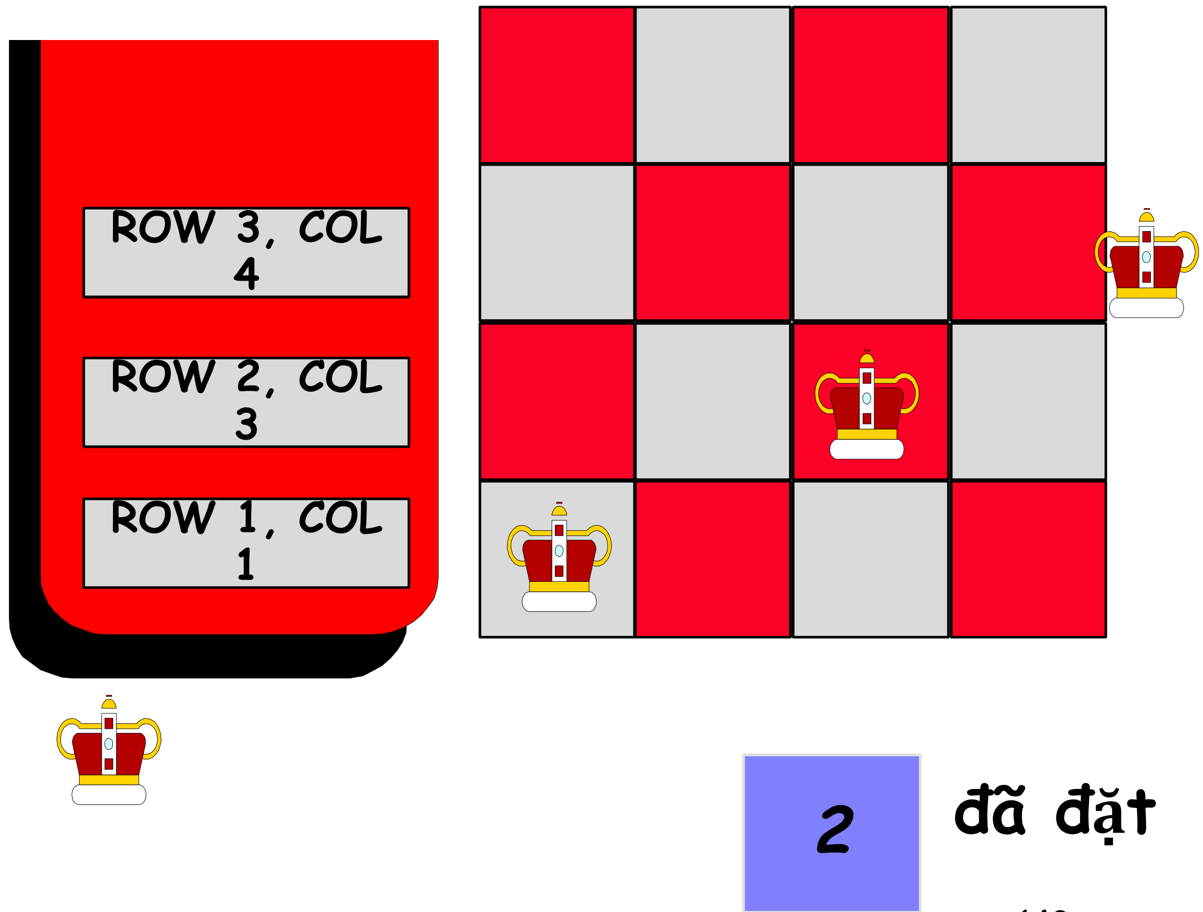
Thuật toán làm việc như thế nào

- Thử cột tiếp theo



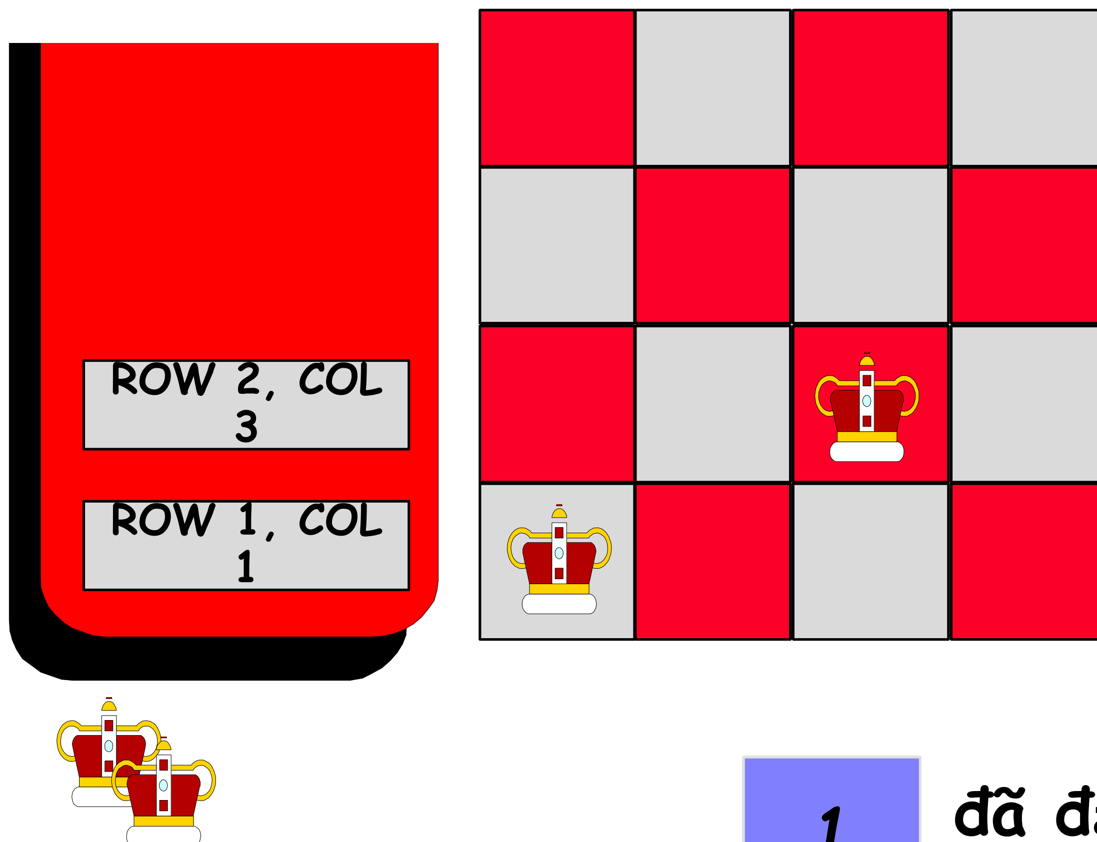
Thuật toán làm việc như thế nào

...không có vị trí đặt con hậu ở dòng 3.



Thuật toán làm việc như thế nào

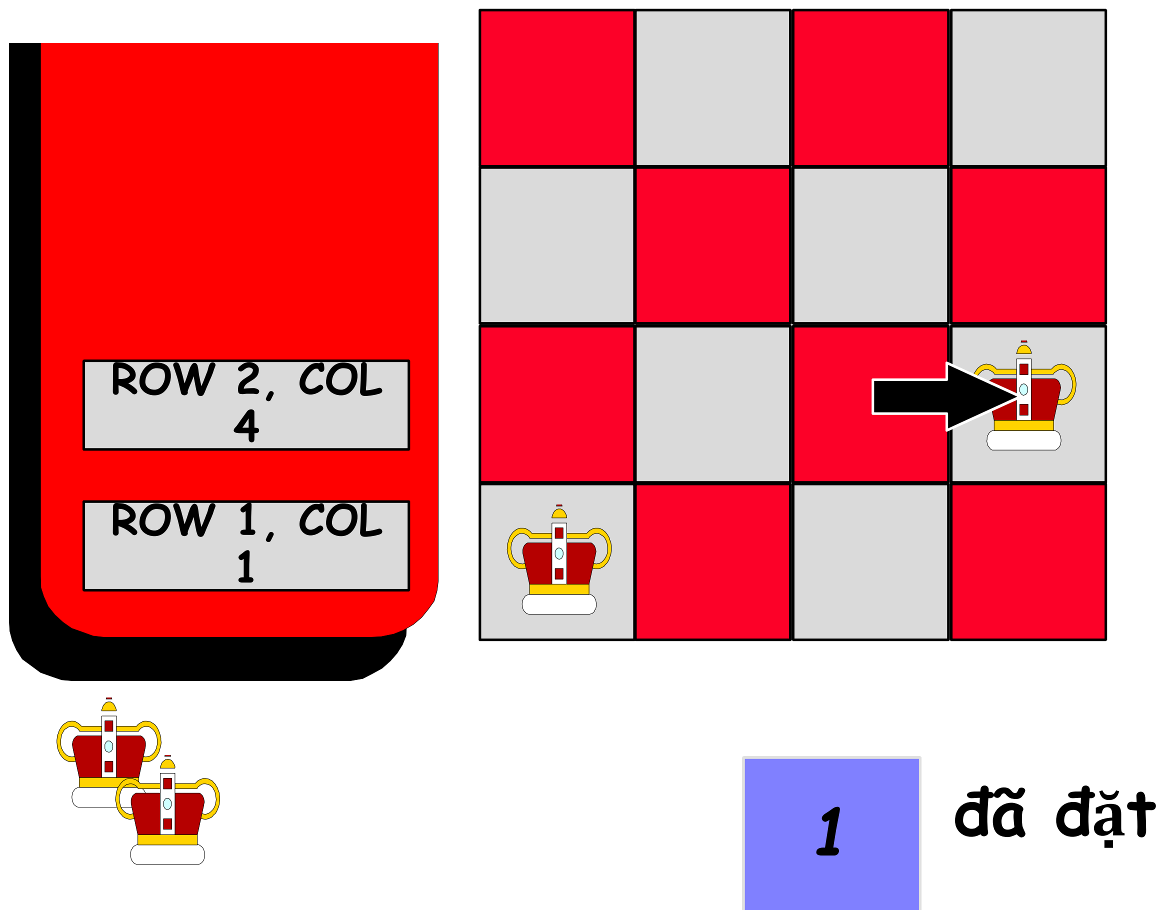
Quay lại dịch chuyển con hậu ở dòng 2



1 đã đặt

Thuật toán làm việc như thế nào

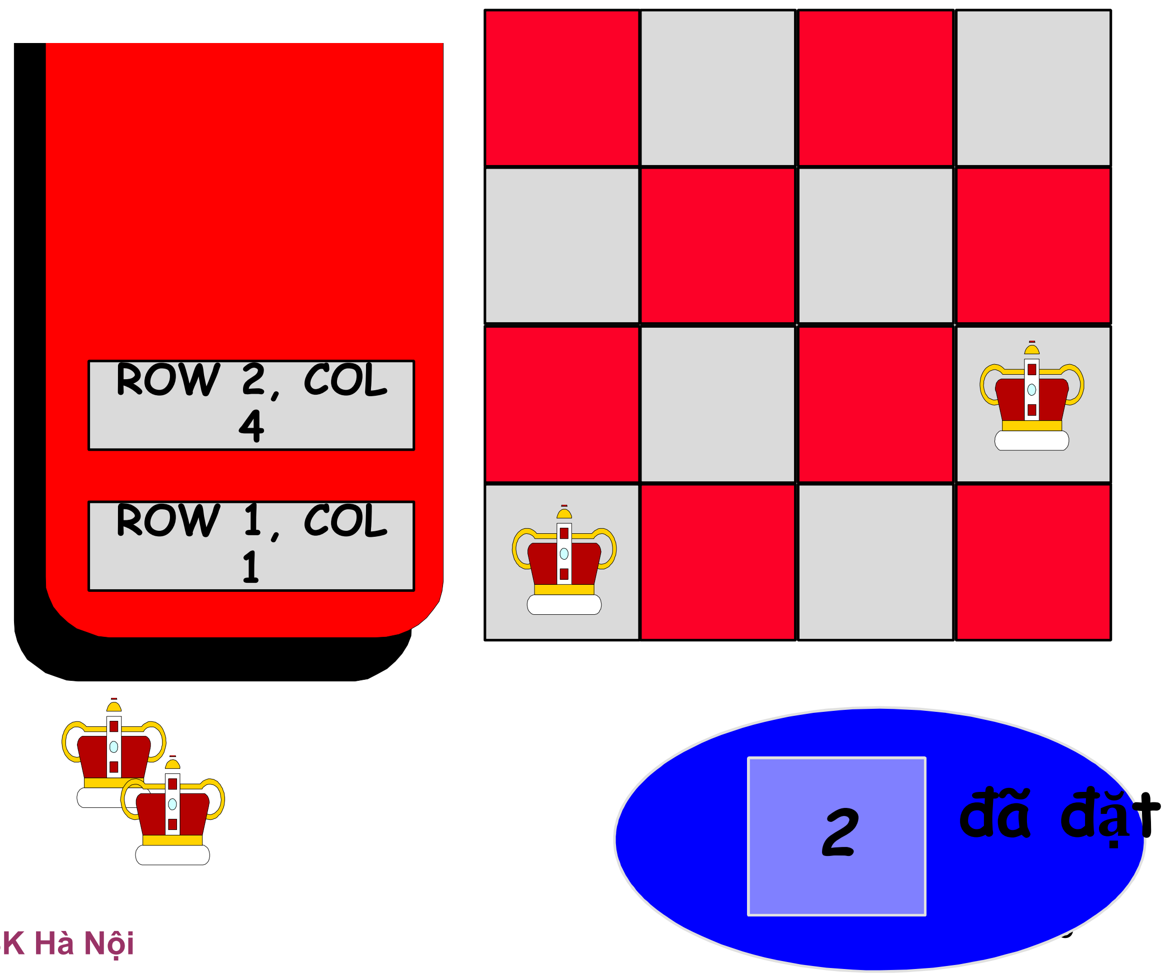
Đẩy con hậu ở dòng 2 sang cột thứ 4.



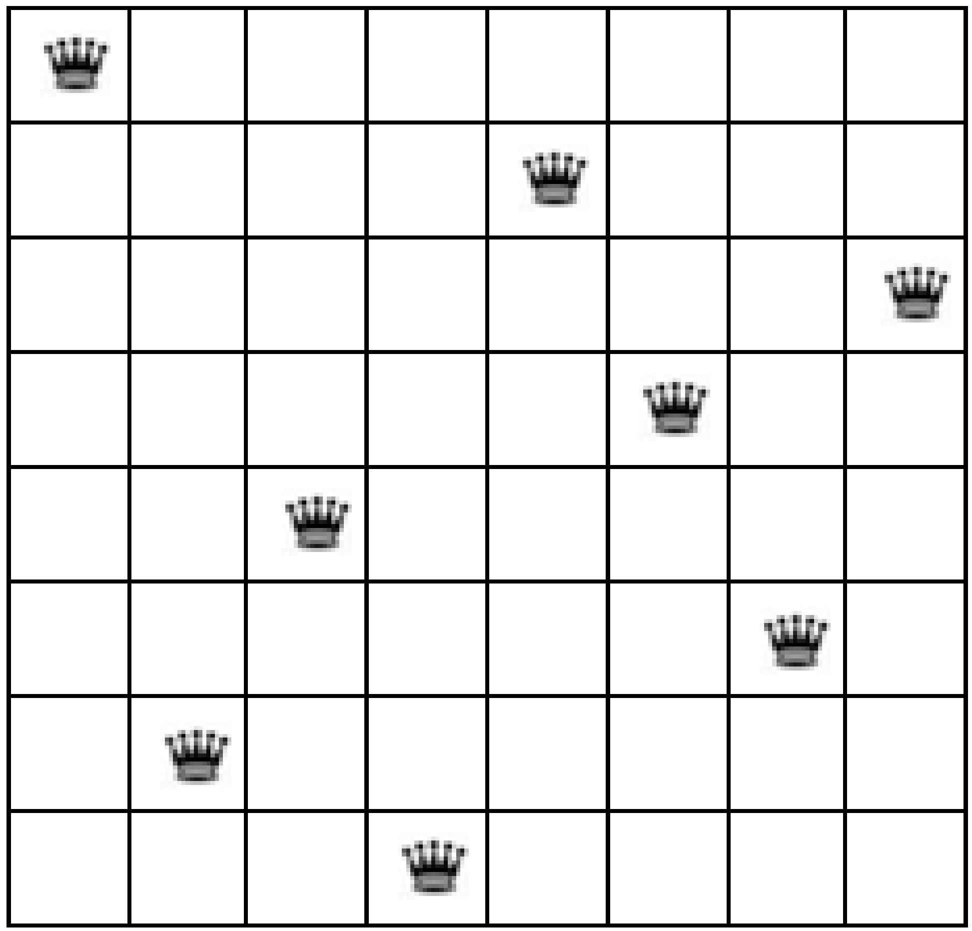
Thuật toán làm việc như thế nào

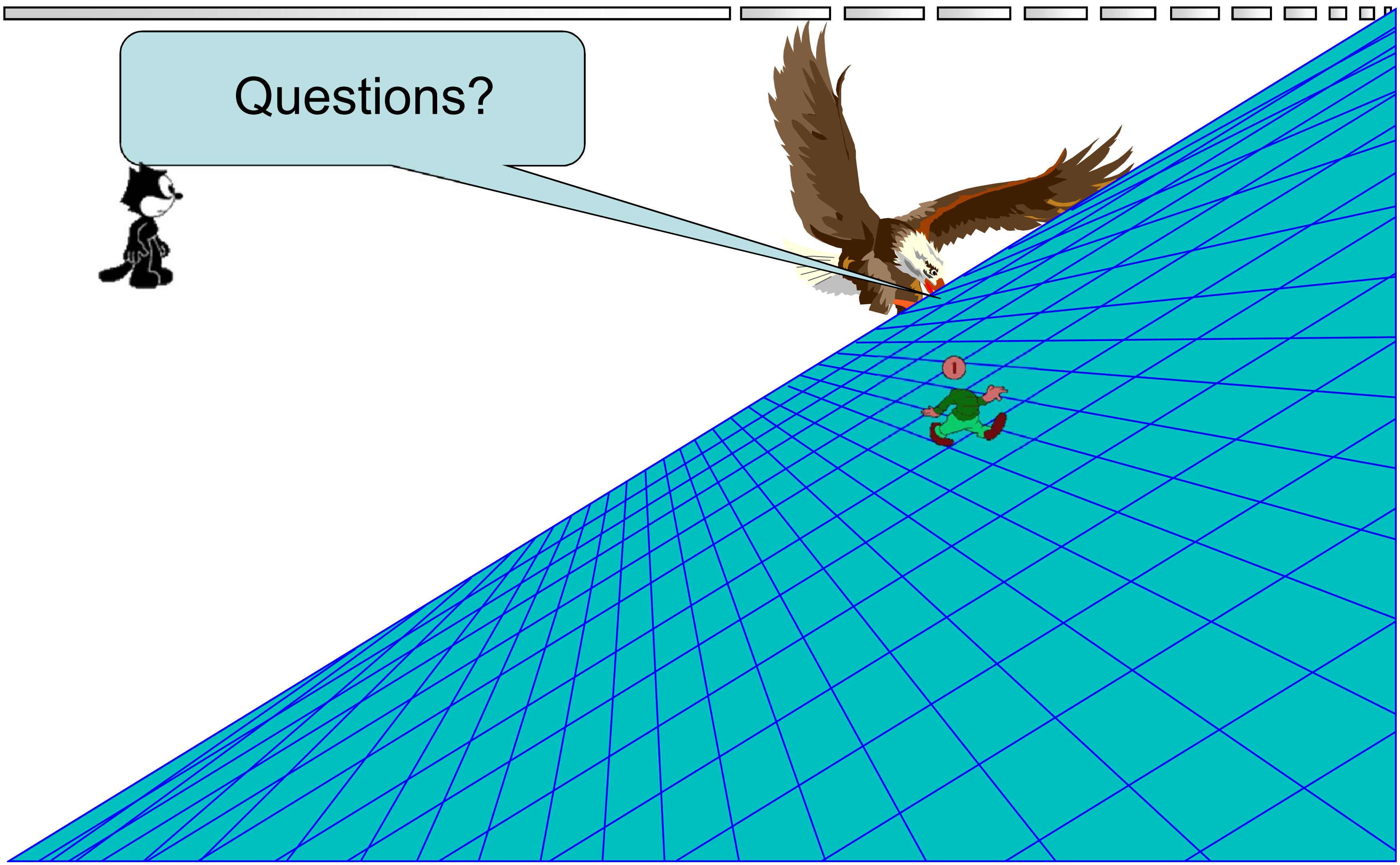
Xếp được con hậu ở dòng 2 ta tiếp tục xếp con hậu ở dòng

. . .



Một lời giải của bài toán xếp hậu khi *n* = 8




Questions?