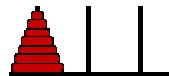


Chương 2

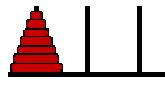
Thuật toán đệ qui



Nội dung

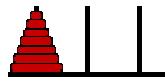


- 2.1. Khái niệm đệ qui**
- 2.2. Thuật toán đệ qui**
- 2.3. Một số ví dụ minh họa**
- 2.4. Phân tích thuật toán đệ qui**
- 2.5. Đệ qui có nhớ**
- 2.6. Chứng minh tính đúng đắn của thuật toán đệ qui**



2.1. Khái niệm đệ qui

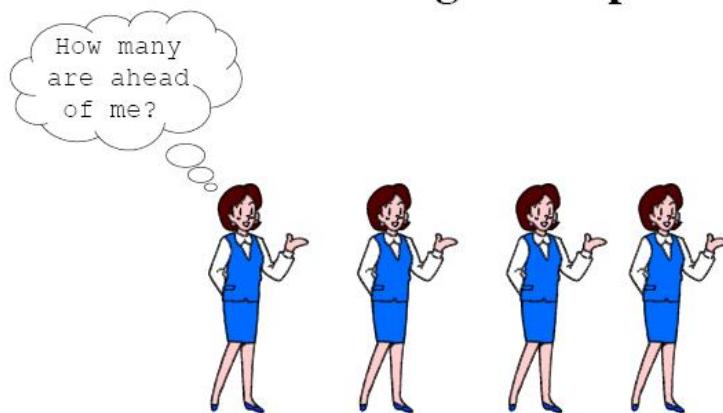
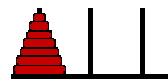
- 2.1.1 Khái niệm đệ qui
- 2.1.2 Thuật toán đệ qui



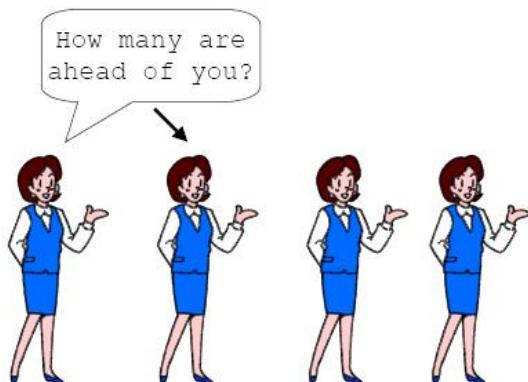
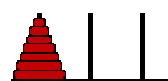
Khái niệm đệ qui

- 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
 - ...

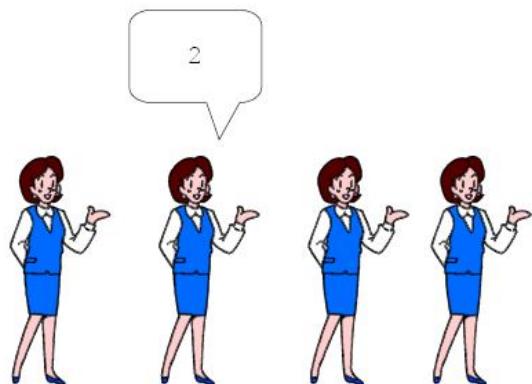
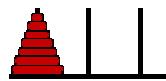
Đệ qui: Điểm quân



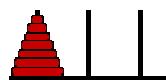
Đệ qui: Điểm quân



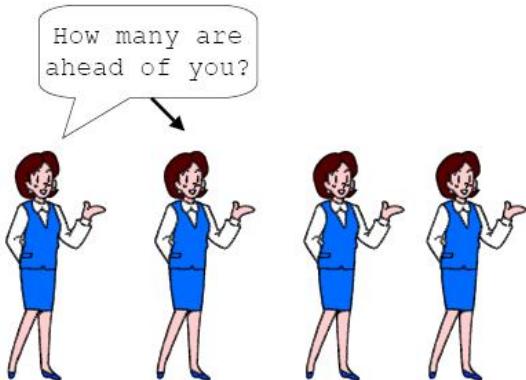
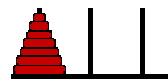
Đệ qui: Điểm quân



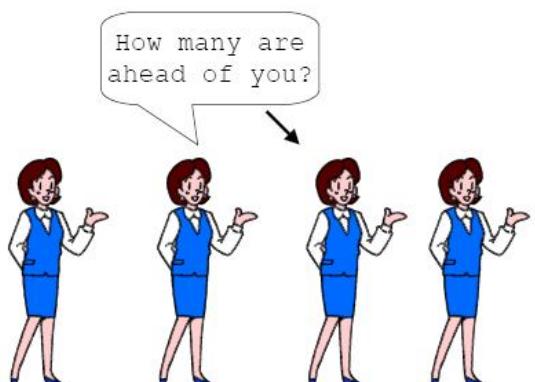
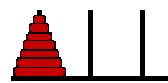
Đệ qui: Điểm quân



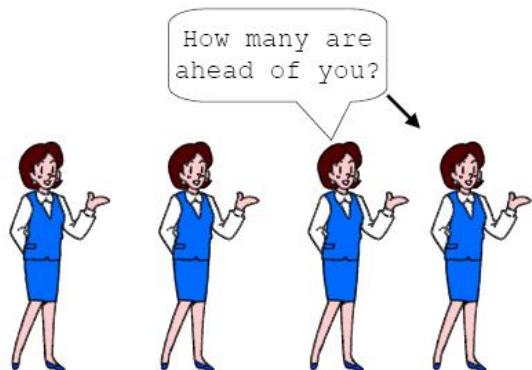
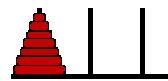
Đệ qui: Điểm quân



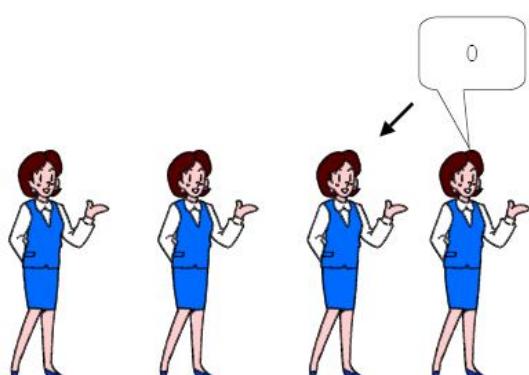
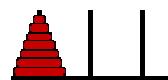
Đệ qui: Điểm quân



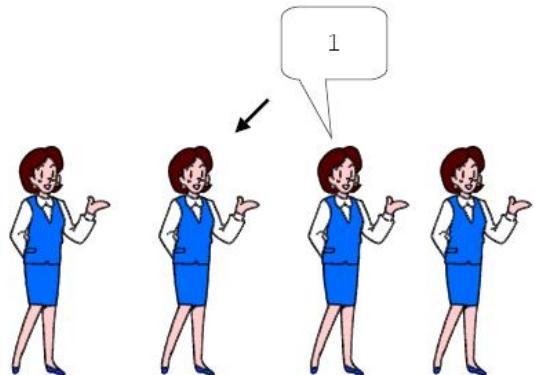
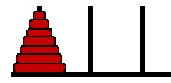
Đệ qui: Điểm quân



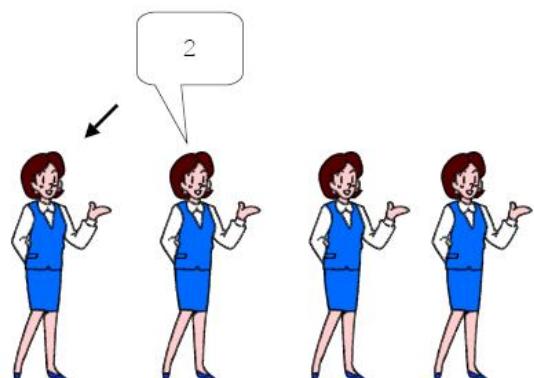
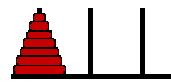
Đệ qui: Điểm quân



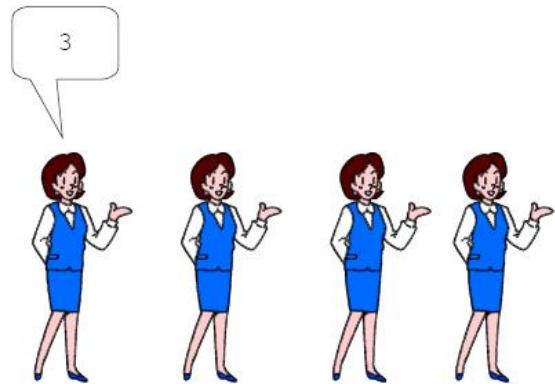
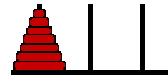
Đệ qui: Điểm quân



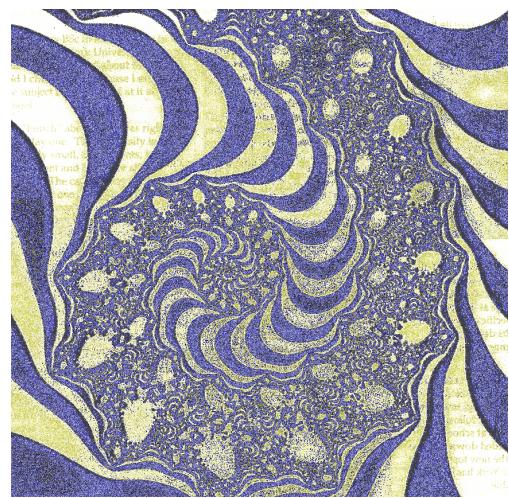
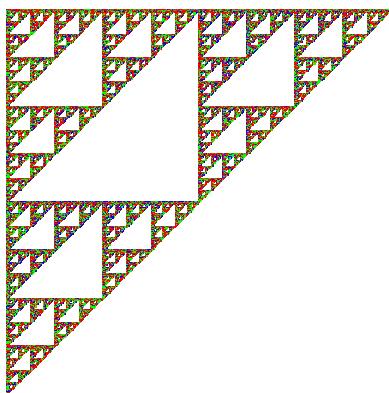
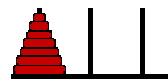
Đệ qui: Điểm quân



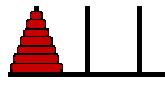
Đệ qui: Điểm quân



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 $n=0$: $f(0)$.

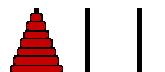
Bước đệ qui (Recursive Step): Cho giá trị của $f(k)$, $k \leq n$, đưa ra qui tắc tính giá trị của $f(n+1)$.

Ví dụ 1:

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

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

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



Hàm đệ qui (Recursive Functions)

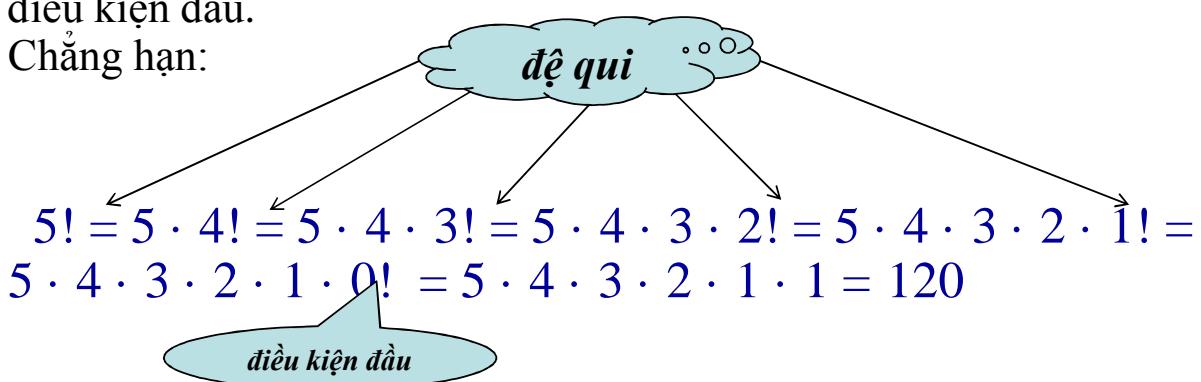
Ví dụ 2: Định nghĩa đệ qui của $n!$:

$$f(0) = 1$$

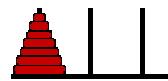
$$f(n+1) = f(n) \times (n+1)$$

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:



Hàm đệ qui (Recursive Functions)



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

$$s_1 = a_1$$

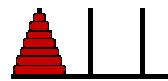
$$s_n = s_{n-1} + a_n$$

Ví dụ 4: Dãy số Fibonacci

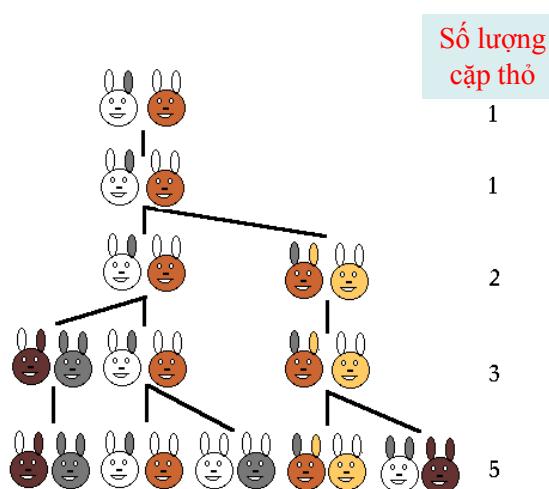
$$f(0) = 0, f(1) = 1,$$

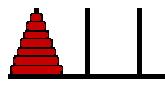
$$f(n) = f(n-1) + f(n-2) \text{ với } n > 1$$

Fibonacci Numbers

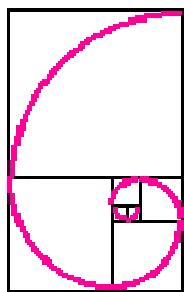
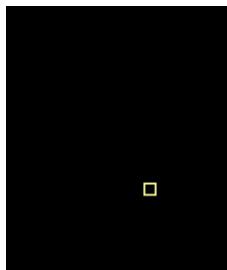


Sự phát triển của bầy thỏ

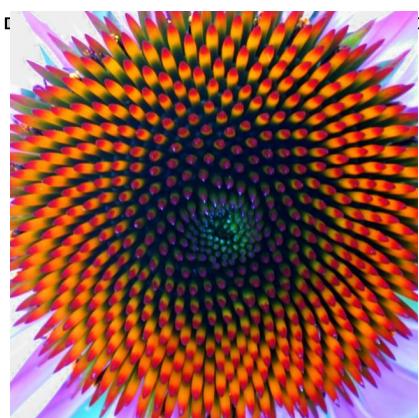
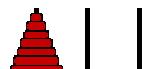




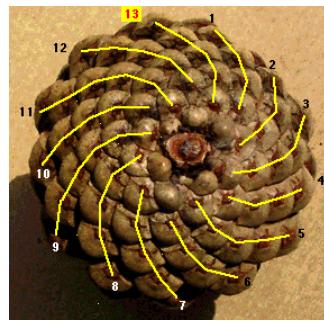
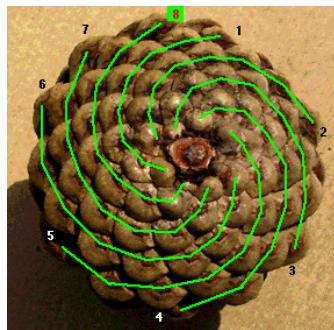
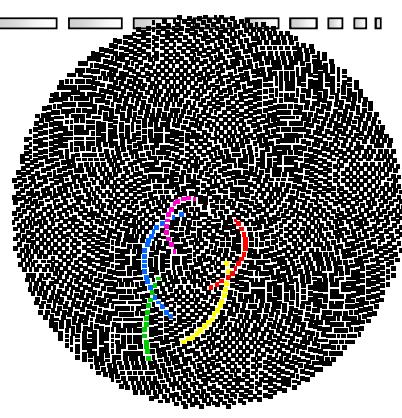
Nói thêm về Fibonacci

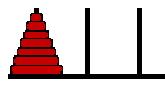


Số Fibonacci trong các cấu trúc tự nhiên

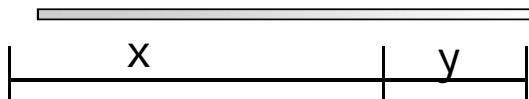


The left and right going
spirals are neighboring
Fibonacci numbers!



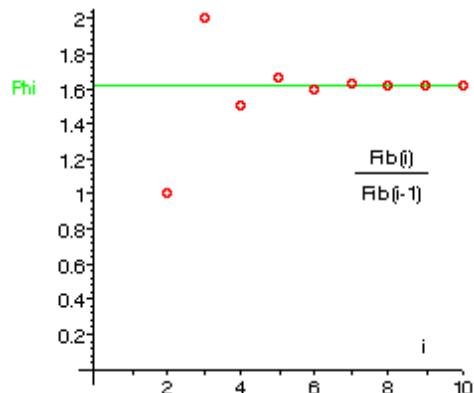


Tỷ lệ vàng (Golden Section)



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

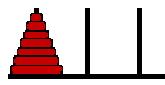
$$\lim_{n \rightarrow \infty} \frac{F(n+1)}{F(n)} = Phi$$



Cấu trúc dữ liệu và thuật toán - NGUYỄN ĐỨC NGHĨA, Bộ môn KHMT, ĐHBK Hà Nội

23

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.

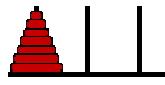
Recursive Step: nếu x thuộc S và y thuộc S thì x+y thuộc S.

3

3+3=6

3+6 = 9 & 6+6=12

...



Định nghĩa đệ qui của xâu

Giả sử Σ = bảng chữ cái (alphabet).

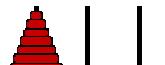
Tập Σ^* các xâu (strings) trên bảng chữ cái Σ được định nghĩa đệ qui như sau:

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 $\Sigma=\{0,1\}$:

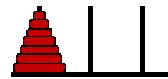
- 1) xâu rỗng
- 2) 0 & 1
- 3) 00 & 01 & 10 & 11
- 4) ...



Độ dài của xâu

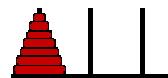
- Định nghĩa đệ qui độ dài $\text{length}(w)$ của xâu $w \in \Sigma^*$
 - (B) $\text{length}(\lambda) = 0$
 - (R) Nếu $w \in \Sigma^*$ và $x \in \Sigma$ thì $\text{length}(wx) = \text{length}(w) + 1$.
- Định nghĩa đệ qui của tập các xâu nhị phân độ dài chẵn.
 - (B) $\lambda \in S$ (λ là xâu rỗng)
 -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

Công thức toán học



- Một **công thức hợp lệ** của các biến, các số và các phép toán từ tập $\{+, -, *, /, ^\}$ có thể định nghĩa như sau:
- Cơ sở:** x là công thức hợp lệ nếu x là biến hoặc là số.
- Qui nạp:** Nếu f, g là các công thức hợp lệ thì
$$(f + g), (f - g), (f * g), (f / g), (f ^ g)$$
là công thức hợp lệ.
- Theo định nghĩa ta có thể xây dựng các công thức hợp lệ như:
$$(x - y); \quad ((z / 3) - y);\\ ((z / 3) - (6 + 5)); \quad ((z / (2 * 4)) - (6 + 5))$$

Cây có gốc



Cây là cấu trúc dữ liệu quan trọng thường được dùng để tổ chức tìm kiếm và sắp xếp dữ liệu

Cây có gốc (Rooted Trees): Cây có gốc bao gồm các nút, có một nút đặc biệt được gọi là gốc (root) và các cạnh nối các nút.

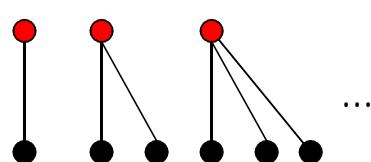
Basic Step: Một nút là cây có gốc.

Recursive Step: Giả sử T_1, \dots, T_n, \dots là các cây với gốc là r_1, \dots, r_n, \dots

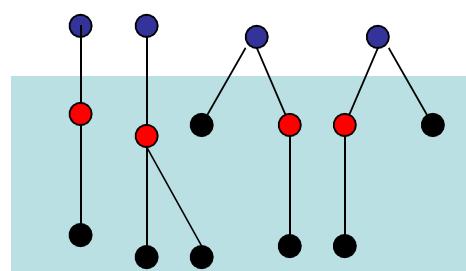
Nếu ta tạo một gốc mới r và nối gốc này với mỗi một trong số các gốc r_1, \dots, r_n, \dots bởi một cạnh tương ứng, ta thu được một cây có gốc mới.

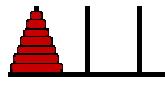
Basis step: ●

Step 1:



Step 2:





Cây nhị phân (Binary Trees)

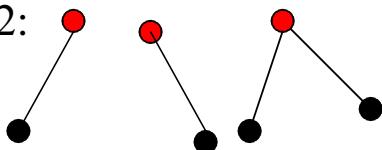
Cây nhị phân mở rộng (Extended Binary Trees):

Basic Step: tập rỗng là cây nhị phân mở rộng.

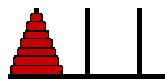
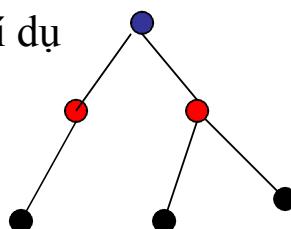
Recursive Step: Nếu T_1 và T_2 là các cây nhị phân mở rộng, thì cây $T_1.T_2$ sau đây cũng là cây nhị phân mở rộng: chọn một nút gốc mới và gắn T_1 với gốc bởi một cạnh như là cây con trái và gắn T_2 như là cây con phải.

Bước 1: ●

Bước 2:



Bước 3: ví dụ



Cây nhị phân đầy đủ

Cây nhị phân đầy đủ (Full binary Trees): Chỉ khác định nghĩa cây nhị phân mở rộng ở bước cơ sở:

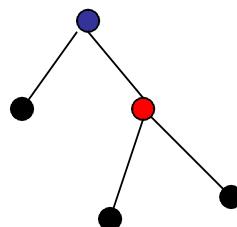
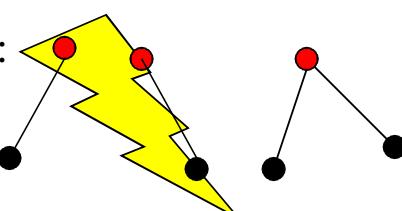
Basic Step: Một nút là cây nhị phân đầy đủ.

Recursive Step: Giống như trong cây nhị phân mở rộng.

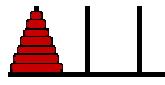
Kết quả là chúng ta không thể gắn cây rỗng vào bên trái cũng như bên phải.

Khởi tạo: ●

Bước 1:



Không phải! Cây nhị phân đầy đủ có 0 hoặc 2 nút con.



Một số định nghĩa

Ký hiệu $h(T)$ là độ cao của cây nhị phân đầy đủ. Định nghĩa đệ qui của $h(T)$:

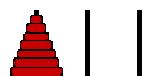
Basic Step: Chiều cao của cây T gồm duy nhất một nút gốc là $h(T)=0$.

Recursive Step: Nếu T_1 và T_2 là các cây nhị phân đầy đủ, thì cây nhị phân đầy đủ $T = T_1 \cdot T_2$ có chiều cao $h(T) = 1 + \max(h(T_1), h(T_2))$.

Ký hiệu $n(T)$ là số đỉnh trong cây. Ta có thể định nghĩa đệ qui $n(T)$ như sau:

Basic Step: Số đỉnh trong cây T gồm duy nhất một nút gốc là $n(T) = 1$;

Recursive Step: Nếu T_1 và T_2 là các cây nhị phân đầy đủ, thì số đỉnh của cây $T = T_1 \cdot T_2$ là $n(T) = 1 + n(T_1) + n(T_2)$.

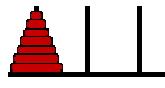


Định nghĩa đệ qui và Qui nạp

- 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:

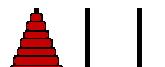
- Bước cơ sở qui nạp** – giống như bước cơ sở trong định nghĩa đệ qui
- Bước chuyển qui nạp** – giống như bước đệ qui



Nội dung

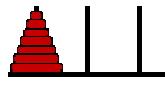


- 2.1. Khái niệm đệ qui
- 2.2. Thuật toán đệ qui
- 2.3. Một số ví dụ minh họa
- 2.4. Phân tích thuật toán đệ qui
- 2.5. Đệ qui có nhớ
- 2.6. Chứng minh tính đúng đắn của thuật toán đệ qui



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 */

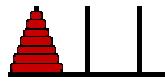
 /* có thể có thêm những lệnh gọi đệ qui */

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

return (**lời_giải**)

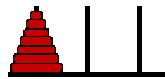
endif

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
```

Gọi bùi to ,n mét c ,ch trùc tiōp

```
    else
```

```
    begin
```

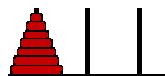
Chia bùi to ,n thunh a bài toán con kích thước n/b ;

```
        for (mci bùi to ,n trong a bùi to ,n con) do D-and-C(n/b);
```

Tæng hî p lêi gi¶li cña a bài toán con để thu được lời giải của bài toán gốc;

```
    end;
```

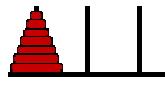
```
end;
```



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 đê 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)**

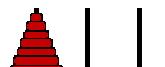
- Chia dãy gồm n phần tử cần sắp xếp ra thành 2 dãy, mỗi dãy có $n/2$ phần tử

- **Trị (Conquer)**

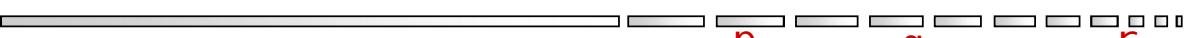
- Sắp xếp mỗi dãy con một cách đệ qui sử dụng *sắp xếp trộn*
- Khi dãy chỉ còn một phần tử thì trả lại phần tử này

- **Tổ hợp (Combine)**

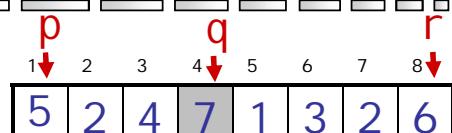
- Trộn (Merge) hai dãy con được sắp xếp để thu được dãy được sắp xếp gồm tất cả các phần tử của cả hai dãy con



Merge Sort



MERGE-SORT(A , p , r)



if $p < r$

▷ Kiểm tra điều kiện neo

then $q \leftarrow \lfloor (p + r)/2 \rfloor$

▷ Chia (Divide)

MERGE-SORT(A , p , q)

▷ Trị (Conquer)

MERGE-SORT(A , $q + 1$, r)

▷ Trị (Conquer)

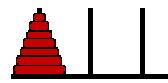
MERGE(A , p , q , r)

▷ Tổ hợp (Combine)

endif

- Lệnh gọi thực hiện thuật toán: **MERGE-SORT(A , 1, n)**

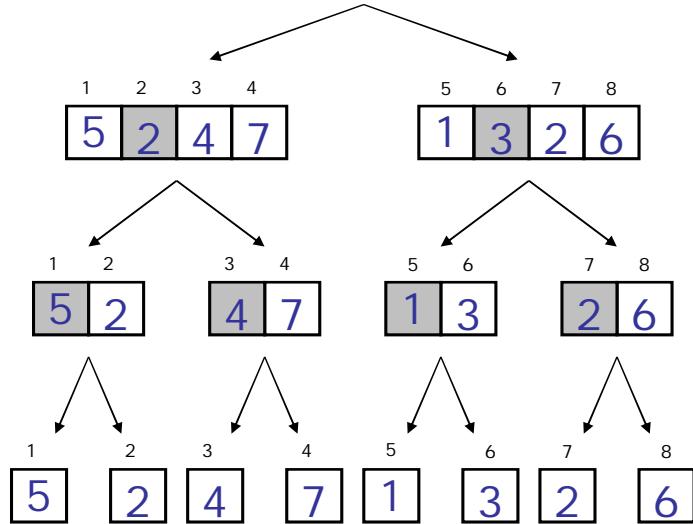
Ví dụ – n là luỹ thừa của 2



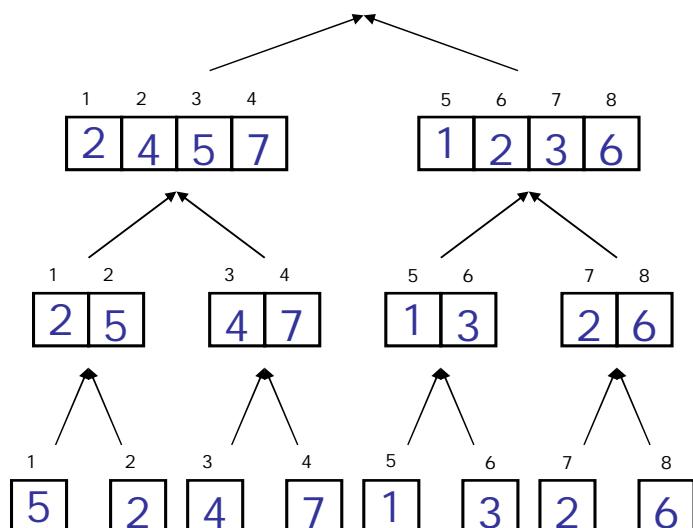
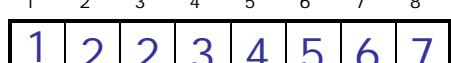
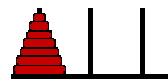
Ví dụ



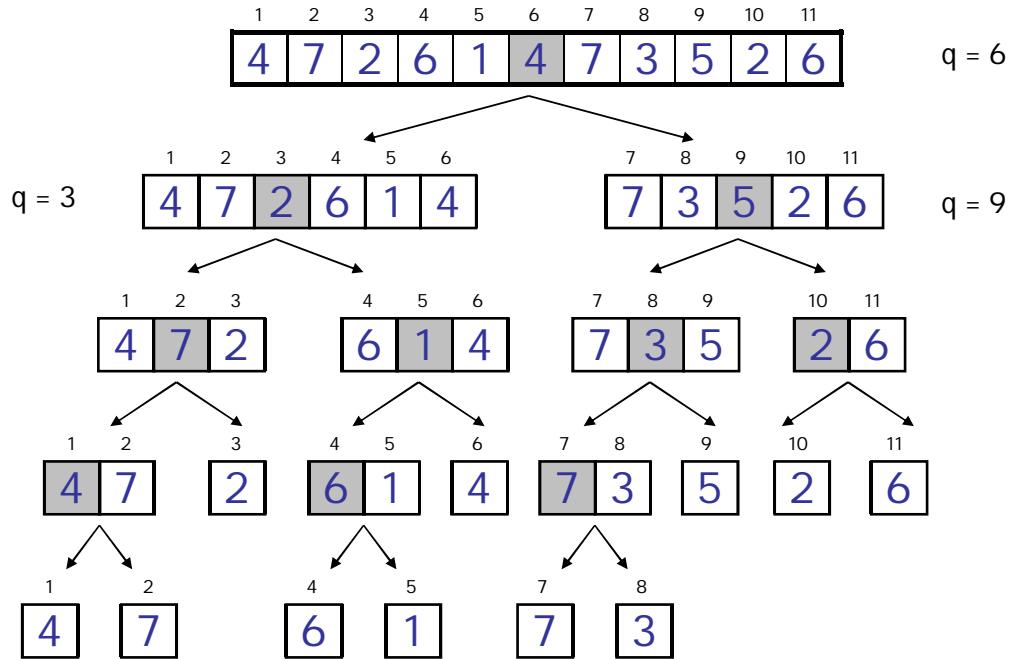
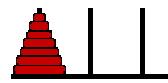
$q = 4$



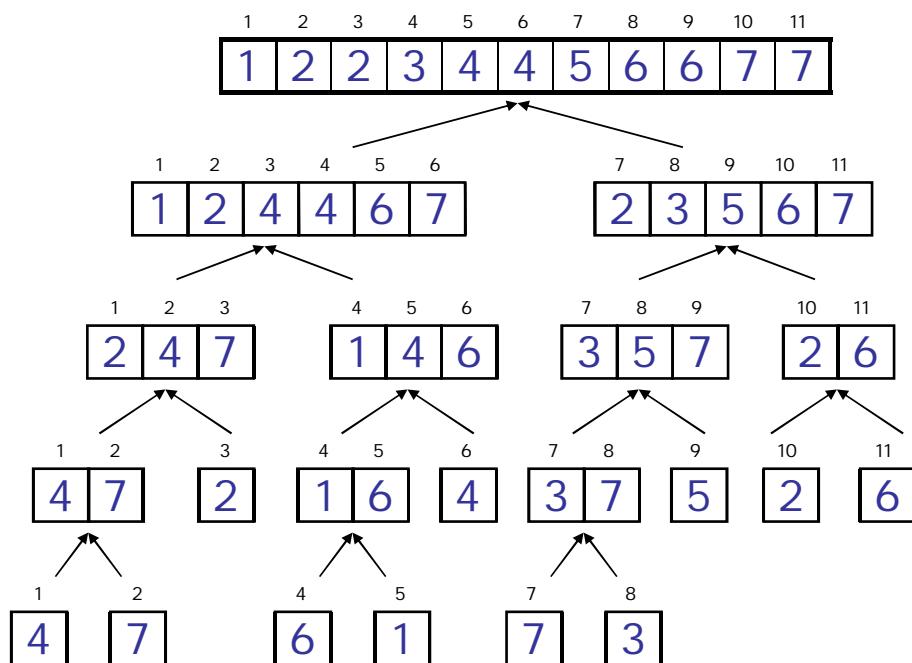
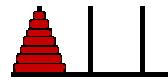
Ví dụ – n là luỹ thừa của 2

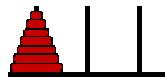


Ví dụ – n không là luỹ thừa của 2

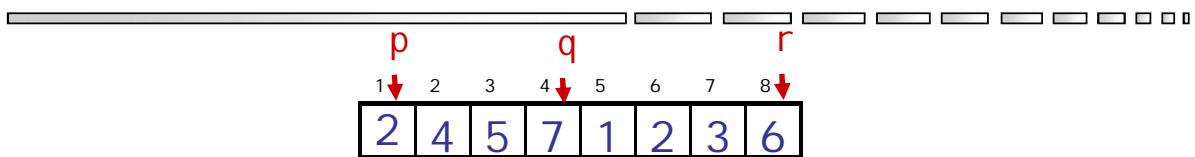


Example – n không là luỹ thừa của 2

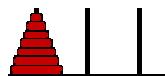




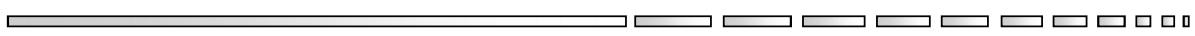
Trộn (Merging)



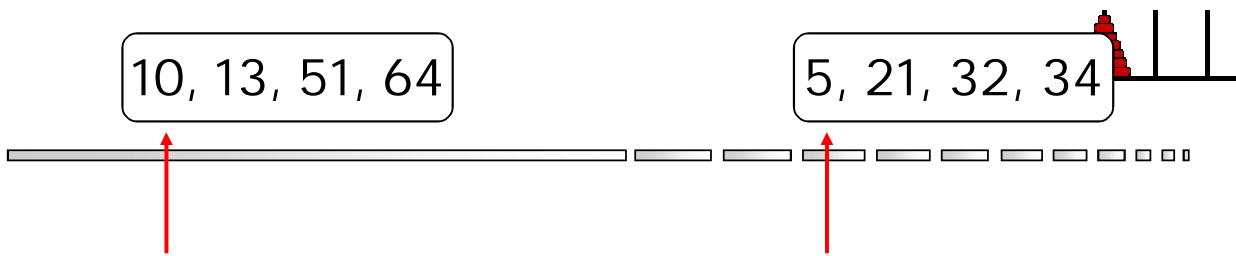
- **Đầu vào:** Mảng A và các chỉ số p, q, r sao cho $p \leq q < r$
 - Các mảng con $A[p \dots q]$ và $A[q + 1 \dots r]$ đã được sắp xếp
- **Đầu ra:** Mảng con được sắp xếp $A[p \dots r]$



Trộn



- **Ý tưởng của thuật toán trộn:**
 - Có hai dãy con đã được sắp xếp
 - *Chọn phần tử nhỏ hơn ở đầu hai dãy*
 - *Loại nó khỏi dãy con tương ứng và đưa vào dãy kết quả*
 - Lặp lại điều đó cho đến khi một trong hai dãy trở thành dãy rỗng
 - Các phần tử còn lại của dãy con kia sẽ được đưa nốt vào đuôi của dãy kết quả



Kết quả:

Để trộn hai dãy được sắp,
ta sẽ giữ hai con trỏ, mỗi con cho một dãy

So sánh hai phần tử được trỏ,
đưa phần tử **nhỏ hơn** vào dãy kết quả
và **di chuyển** con trỏ tương ứng



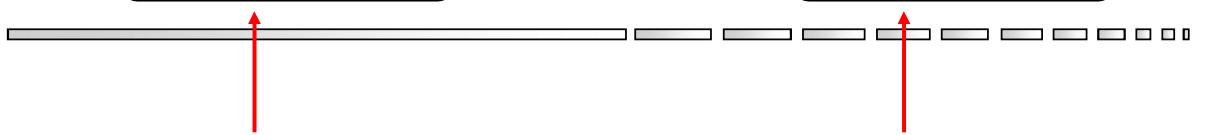
Kết quả:

5,

Tiếp theo lại so sánh hai phần tử
được chỉ ra bởi con trỏ;
đưa phần tử nhỏ hơn vào dãy kết quả
và tăng con trỏ tương ứng

10, 13, 51, 64

5, 21, 32, 34



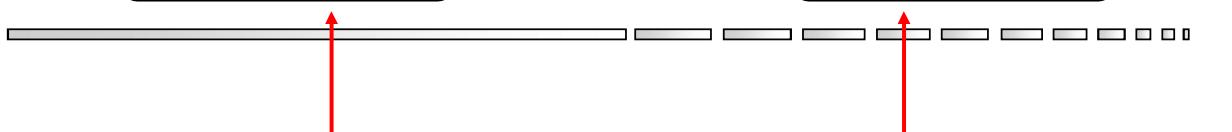
Kết quả:

5, 10,

Lặp lại quá trình ...

10, 13, 51, 64

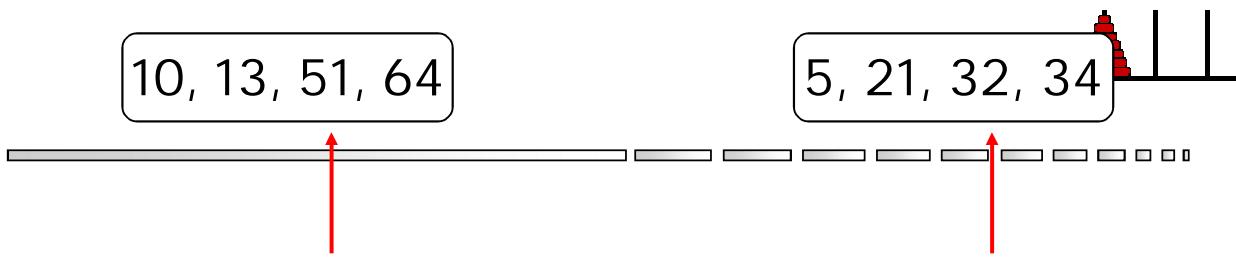
5, 21, 32, 34



Kết quả:

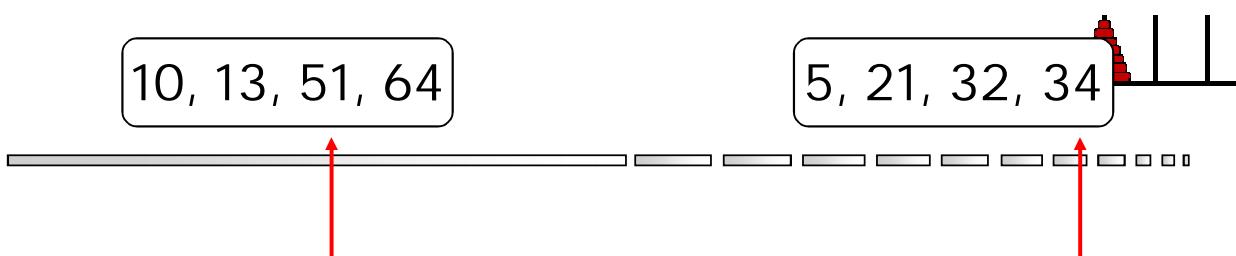
5, 10, 13

Lặp lại quá trình ...



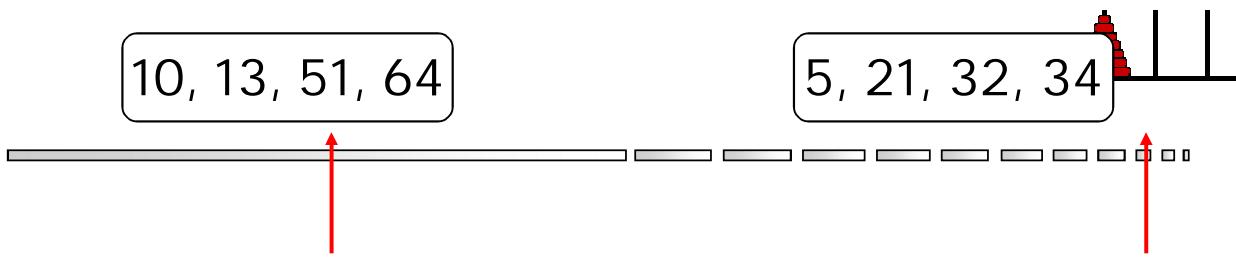
Kết quả: 5, 10, 13, 21

và cứ như vậy ...



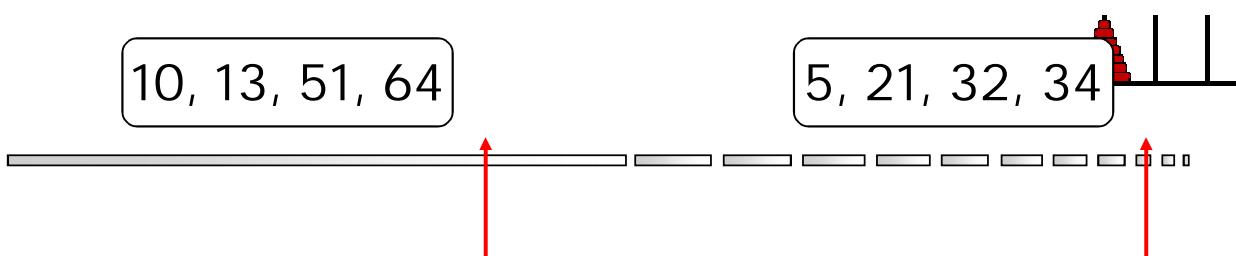
Kết quả: 5, 10, 13, 21, 32

...



Kết quả: 5, 10, 13, 21, 32, 34

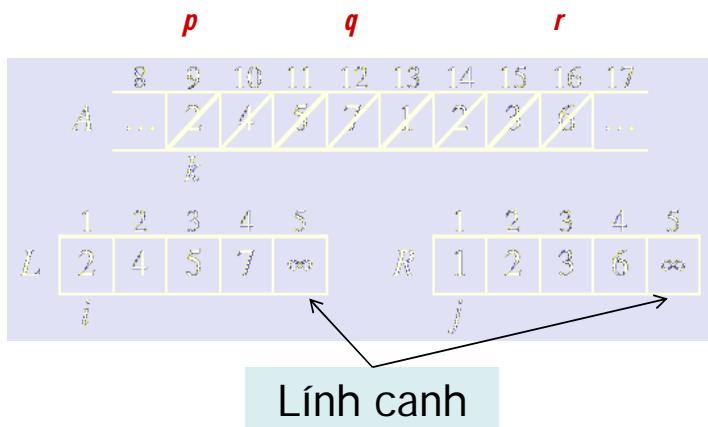
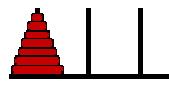
Khi ta đạt đến kết thúc của một dãy,
chỉ việc đưa các phần tử còn lại của dãy kia vào dãy kết quả



Kết quả: 5, 10, 13, 21, 32, 34, 51, 64

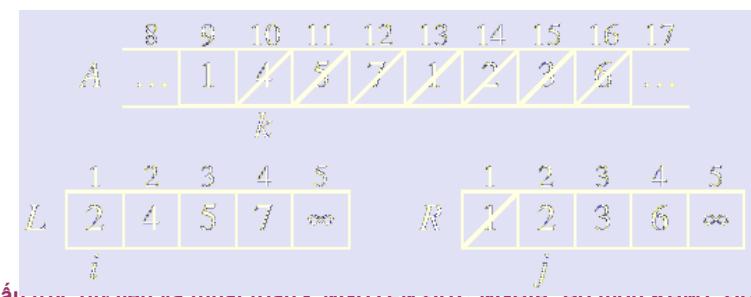
Ta thu được dãy được sắp xếp

Ví dụ: MERGE(A, 9, 12, 16)



Mảng A chứa 2 dãy
con đã được sắp xếp:
 $A[p..q]$ và $A[q+1..r]$

Sao ra 2 mảng con

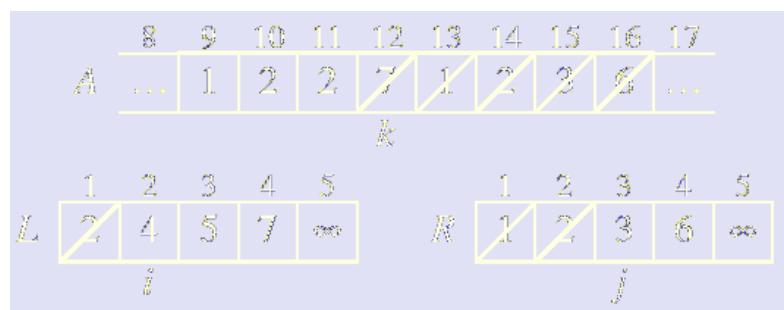
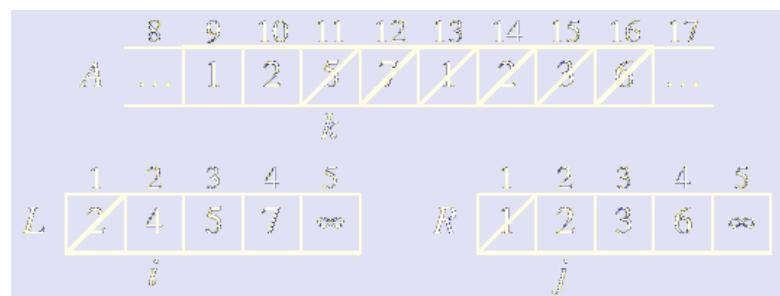
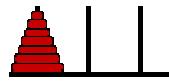


Trộn 2 mảng con và
đưa trả kết quả vào A

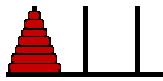
CẤU TRÚC DỮ LIỆU VÀ MÔN HỌC - NGUYỄN ĐỨC NGỌM, BỘ MÔN KINH TẾ, ĐHQGHN

55

Ví dụ: MERGE(A, 9, 12, 16)



Ví dụ (tiếp)



A	8	9	10	11	12	13	14	15	16	17
	...	1	2	2	3	1	2	3	6	...
										k

L	1	2	3	4	5	∞
	2	4	5	7	∞	
						i

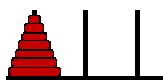
R	1	2	3	4	5	∞
	1	2	3	6	∞	
						j

A	8	9	10	11	12	13	14	15	16	17
	...	1	2	2	3	4	2	3	6	...
										k

L	1	2	3	4	5	∞
	2	4	5	7	∞	
						i

R	1	2	3	4	5	∞
	1	2	3	6	∞	
						j

Ví dụ (tiếp)



A	8	9	10	11	12	13	14	15	16	17
	...	1	2	2	3	4	5	3	6	...
										k

L	1	2	3	4	5	∞
	2	4	5	7	∞	
						i

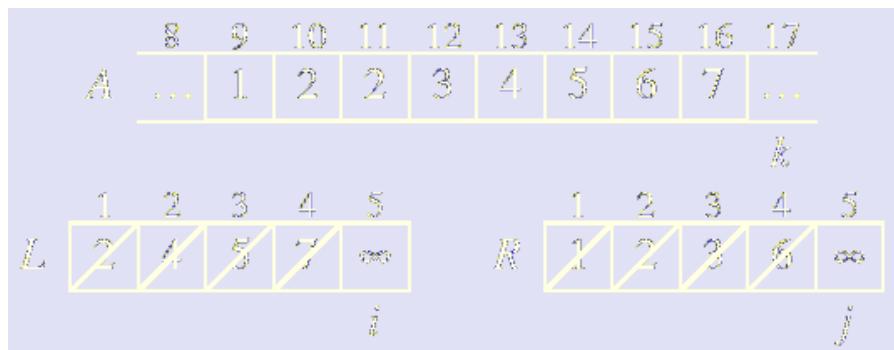
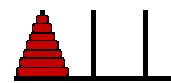
R	1	2	3	6	∞	
	1	2	3	6	∞	
						j

A	8	9	10	11	12	13	14	15	16	17
	...	1	2	2	3	4	5	6	6	...
										k

L	1	2	3	4	5	∞
	2	4	3	7	∞	
						i

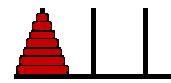
R	1	2	3	5	∞	
	1	2	3	5	∞	
						j

Ví dụ (tiếp)



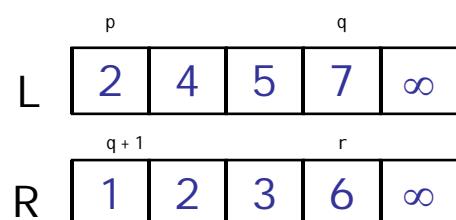
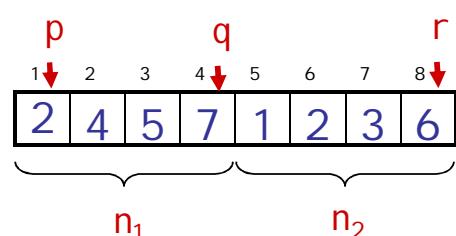
Trộn xong!

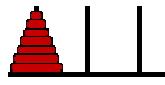
Trộn (Merge) - Pseudocode



MERGE(A, p, q, r)

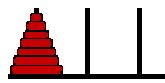
1. Tính n_1 và n_2
2. Sao n_1 phần tử đầu tiên vào $L[1 \dots n_1]$ và n_2 phần tử tiếp theo vào $R[1 \dots n_2]$
3. $L[n_1 + 1] \leftarrow \infty$; $R[n_2 + 1] \leftarrow \infty$
4. $i \leftarrow 1$; $j \leftarrow 1$
5. **for** $k \leftarrow p$ **to** r **do**
6. **if** $L[i] \leq R[j]$
7. **then** $A[k] \leftarrow L[i]$
8. *i* $\leftarrow i + 1$
9. **else** $A[k] \leftarrow R[j]$
10. *j* $\leftarrow j + 1$





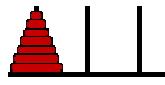
Thời gian tính của trộn

- Khởi tạo (tạo hai mảng con tạm thời L và R):
 - $\Theta(n_1 + n_2) = \Theta(n)$
- Đưa các phần tử vào mảng kết quả (vòng lặp **for** cuối cùng):
 - n lần lặp, mỗi lần đòi hỏi thời gian hằng số $\Rightarrow \Theta(n)$
- Tổng cộng thời gian của trộn là:
 - $\Theta(n)$



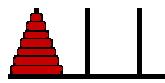
Nội dung

- 2.1. Khái niệm đệ qui
- 2.2. Thuật toán đệ qui
-  2.3. Một số ví dụ minh họa
- 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:
 - 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!

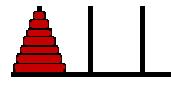
- Hàm $f(n) = n!$ được định nghĩa đệ qui như sau

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

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

- Hàm đệ qui trên C:

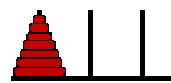
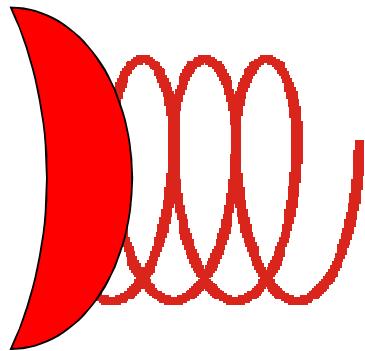
```
int Fact(int n){  
    if (n==0) return 1;  
    else return n*Fact(n-1);  
}
```



Hoạt động của Fact(5)

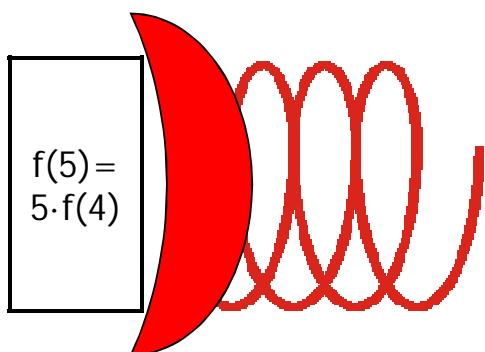
```
int Fact(int n){  
    if (n==0) return 1;  
    else return n*Fact(n-1);  
}
```

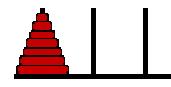
Tính 5!



Hoạt động của Fact(5)

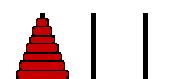
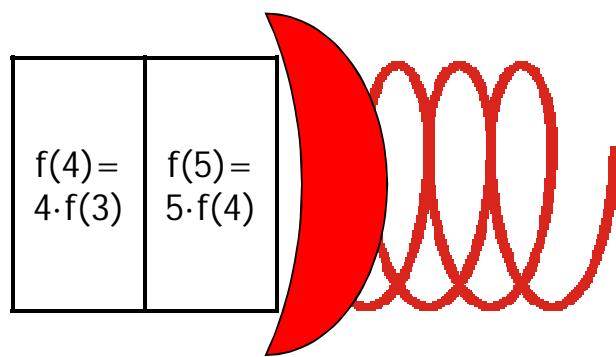
```
int Fact(int n){  
    if (n==0) return 1;  
    else return n*Fact(n-1);  
}
```





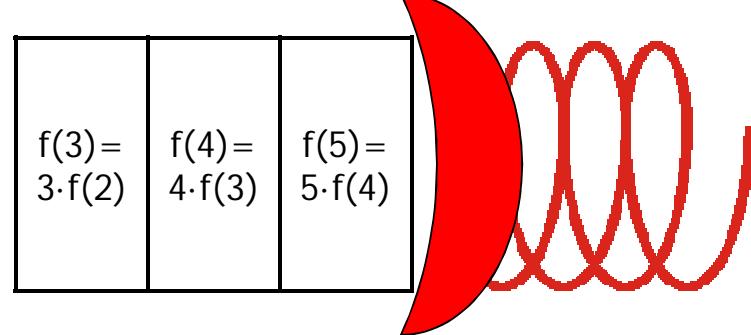
Hoạt động của Fact(5)

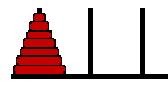
```
int Fact(int n){  
    if (n==0) return 1;  
    else return n*Fact(n-1);  
}
```



Hoạt động của Fact(5)

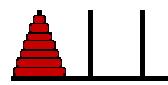
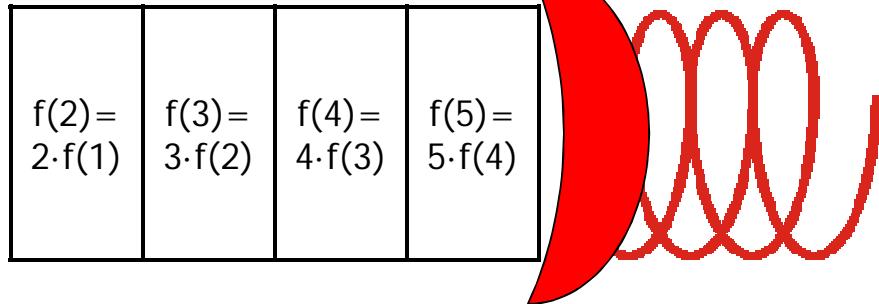
```
int Fact(int n){  
    if (n==0) return 1;  
    else return n*Fact(n-1);  
}
```





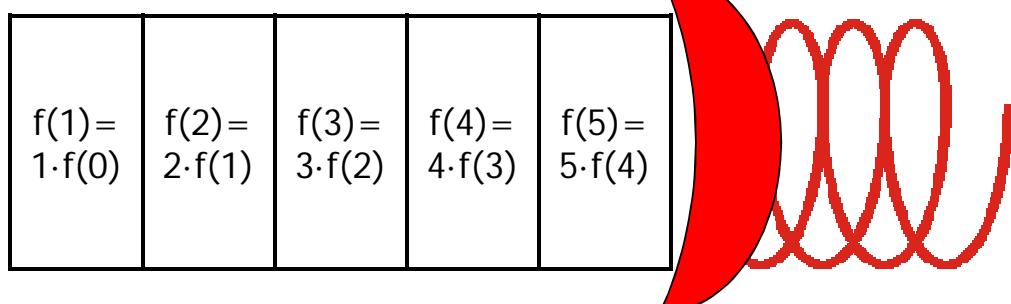
Hoạt động của Fact(5)

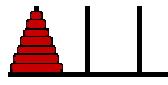
```
int Fact(int n){  
    if (n==0) return 1;  
    else return n*Fact(n-1);  
}
```



Hoạt động của Fact(5)

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

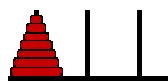
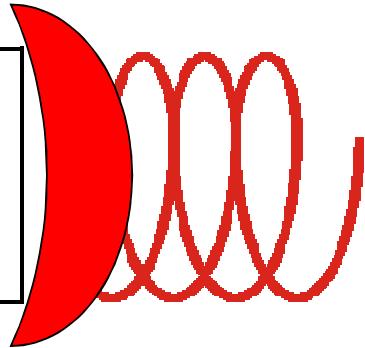




Hoạt động của Fact(5)

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

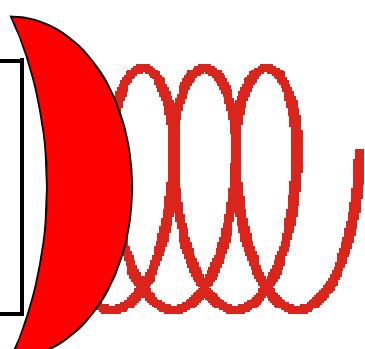
$f(0)=$ $1 \rightarrow$	$f(1)=$ $1 \cdot f(0)$	$f(2)=$ $2 \cdot f(1)$	$f(3)=$ $3 \cdot f(2)$	$f(4)=$ $4 \cdot f(3)$	$f(5)=$ $5 \cdot f(4)$
----------------------------	---------------------------	---------------------------	---------------------------	---------------------------	---------------------------

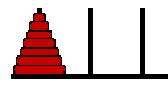


Hoạt động của Fact(5)

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

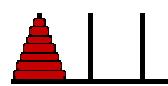
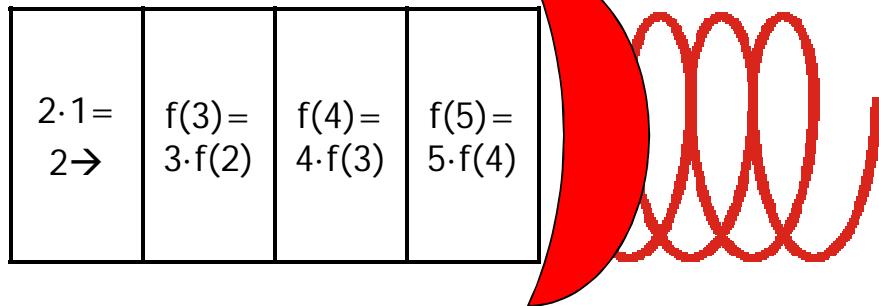
$1 \cdot 1=$ $1 \rightarrow$	$f(2)=$ $2 \cdot f(1)$	$f(3)=$ $3 \cdot f(2)$	$f(4)=$ $4 \cdot f(3)$	$f(5)=$ $5 \cdot f(4)$
---------------------------------	---------------------------	---------------------------	---------------------------	---------------------------





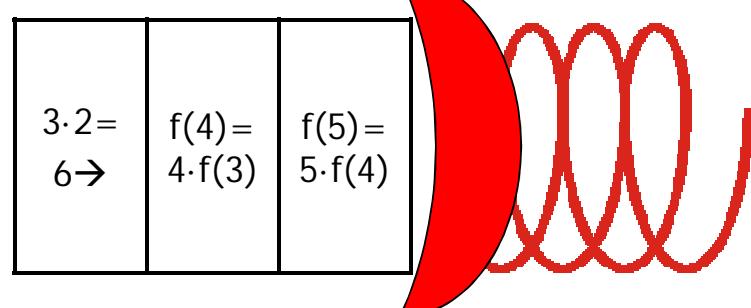
Hoạt động của Fact(5)

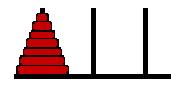
```
int Fact(int n){  
    if (n==0) return 1;  
    else return n*Fact(n-1);  
}
```



Hoạt động của Fact(5)

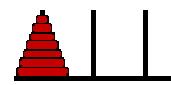
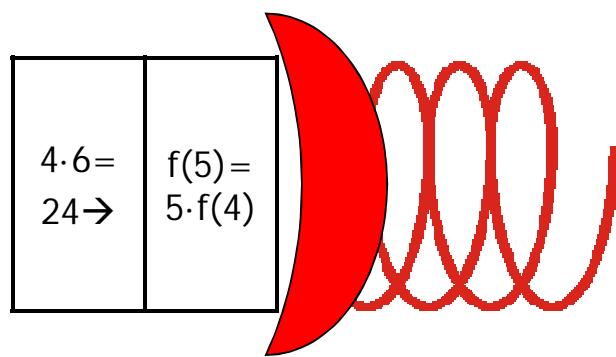
```
int Fact(int n){  
    if (n==0) return 1;  
    else return n*Fact(n-1);  
}
```





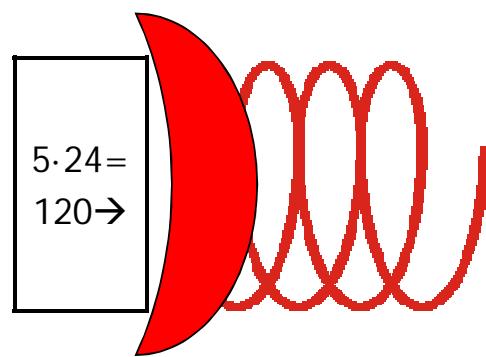
Hoạt động của Fact(5)

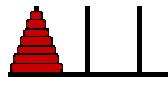
```
int Fact(int n){  
    if (n==0) return 1;  
    else return n*Fact(n-1);  
}
```



Hoạt động của Fact(5)

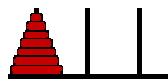
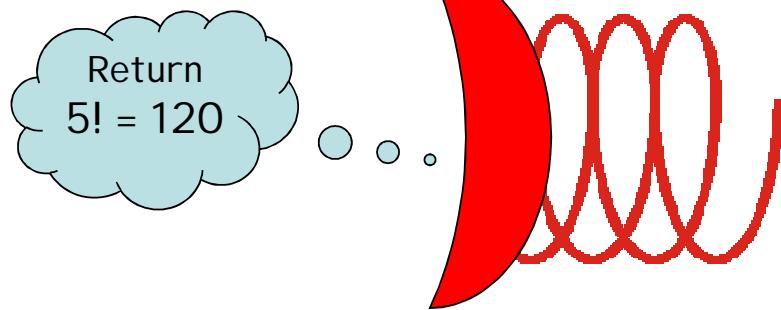
```
int Fact(int n){  
    if (n==0) return 1;  
    else return n*Fact(n-1);  
}
```





Hoạt động của Fact(5)

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



Tính số Fibonacci

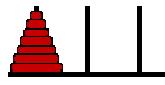
- Dãy số Fibonacci được định nghĩa đệ qui như sau

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

$$F(n) = F(n-1) + F(n-2), n \geq 2.$$

- 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

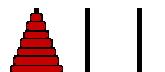
- Hệ số nhị thức $C(n,k)$ được định nghĩa đê qui như sau

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

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

- Hàm đê 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);  
}
```



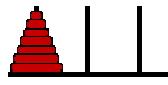
Tìm kiếm nhị phân

Bí quyết: Cho mảng số $x[1..n]$ được sắp xếp theo thứ tự không giảm vự sè y. Cacen tñm chø sè i ($1 \leq i \leq n$) sao cho $x[i] = y$.

Sđ đơn giản ta giả thiết rằng chỉ số như vậy là tồn tại. Thuết toán giải bài toán được xây dựng dựa trên lập luận sau: Số y cho trước

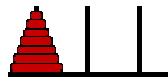
- hoặc lù b»ng phçn tö n»m è vþ trý è gi÷a m¶ng x
- hoặc lù n»m è nöa bªn tr,s i (L) cña m¶ng x
- hoặc lù n»m è nöa bªn ph¶i (R) cña m¶ng x.
(Tnh huèng L (R) x¶y ra chø khi y nhá h¬n (lí n h¬n) phçn tö è gi÷a cña m¶ng x.)

Ph»n tích tr»n dÉn ®Ón thuết toán sau ®©y:



Tìm kiếm nhị phân

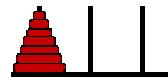
```
function Bsearch(x[1..n], start, finish) {  
    middle:= (start+finish)/2;  
    if (y = x[middle]) return middle;  
    else {  
        if (y < x[middle]) return Bsearch(x, start, middle-1);  
        else /* y > x[middle] */  
            return Bsearch(x, middle+1, finish)  
    }  
}
```



Hàm trên C

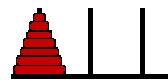
```
boolean binary_search_2(int* a, int n, int x) {  
/* Test xem x có mặt trong mảng a[] kích thước n. */  
    int i;  
    if (n > 0) {  
        i = n / 2;  
        if (a[i] == x)  
            return true;  
        if (a[i] < x)  
            return binary_search_2(&a[i + 1], n - i - 1, x);  
        return binary_search_2(a, i, x);  
    }  
    return false;  
}
```

Tìm kiếm nhị phân (xét cả trường hợp không tìm thấy)



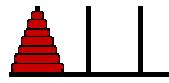
```
function Bsearch(x[1..n], start, finish) {  
    middle:= (start+finish)/2;  
    if (start==finish) { /* Base step */  
        if (x[middle]==y) return middle  
        else return notFound ;  
    }  
    if (y == x[middle]) return middle;  
    else {  
        if (y < x[middle]) return Bsearch(x, start, middle-1);  
        else /* y > x[middle] */  
            return Bsearch(x, middle+1, finish)  
    }  
}
```

Bài toán tháp Hà nội (Hanoi Tower)

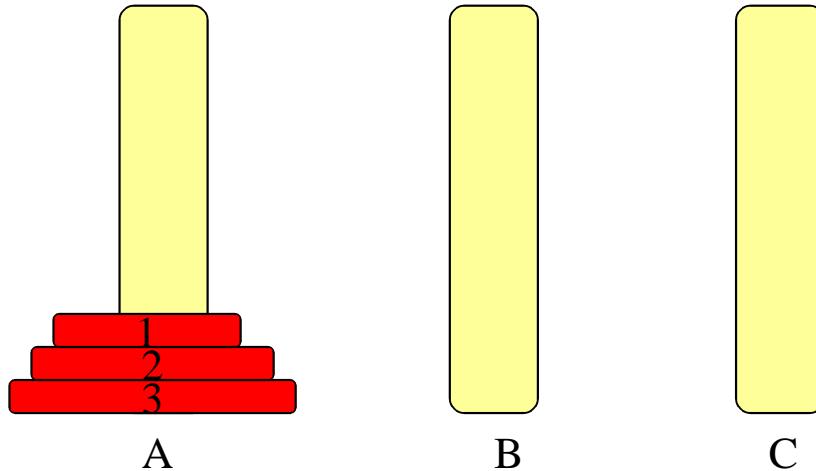


- Bài toán tháp Hà nội. Trò chơi tháp Hà nội được trình bày như sau: “*Có 3 cọc a, b, c. Trên cọc a có một chồng gồm n cái đĩa đường kính giảm dần từ dưới lên trên. Cần phải chuyển chồng đĩa từ cọc a sang cọc c tuân thủ qui tắc: mỗi lần chỉ chuyển 1 đĩa và chỉ được xếp đĩa có đường kính nhỏ hơn lên trên 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

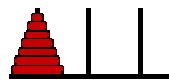
Tower of Hanoi - Trạng thái xuất phát



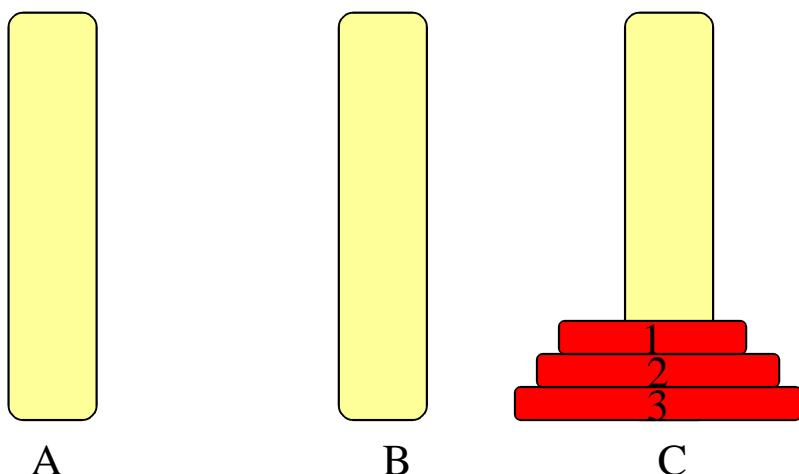
- Có 3 cọc và một chồng đĩa được sắp xếp theo thứ tự đường kính giảm dần từ dưới lên trên ở cọc A.



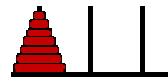
Tower of Hanoi - Trạng thái kết thúc



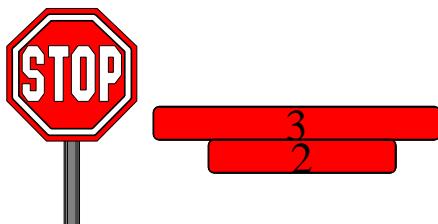
- Ta muốn chuyển chồng đĩa sang cọc C



Tower of Hanoi - Qui tắc chơi

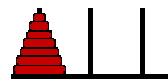


- Mỗi lần chỉ chuyển 1 đĩa
- Chỉ được đặt đĩa có đường kính nhỏ hơn lên trên đĩa có đường kính lớn hơn

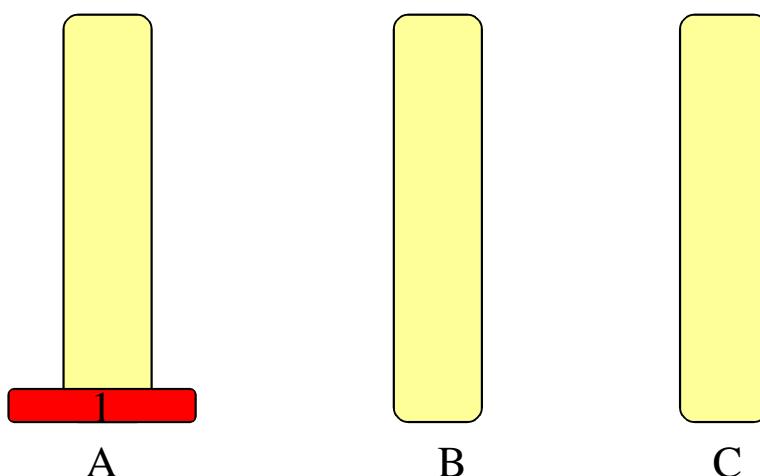


Mục đích: Sử dụng ít lần di chuyển đĩa nhất

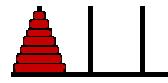
Tower of Hanoi - Chỉ có một đĩa



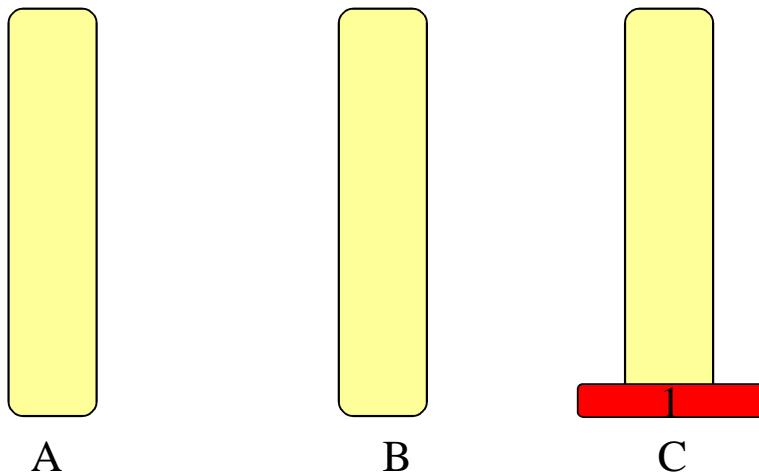
- Quá dễ!



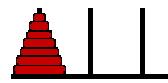
Tower of Hanoi - Chỉ có một đĩa



- Quá dễ! Chỉ cần 1 lần chuyển.

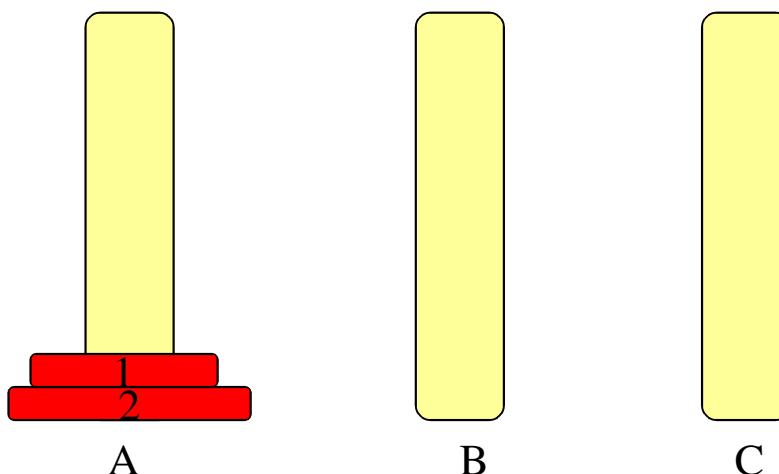


Tower of Hanoi - Hai đĩa

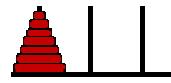


- Nhận thấy rằng ta phải chuyển đĩa 2 đến C. Bằng cách nào?

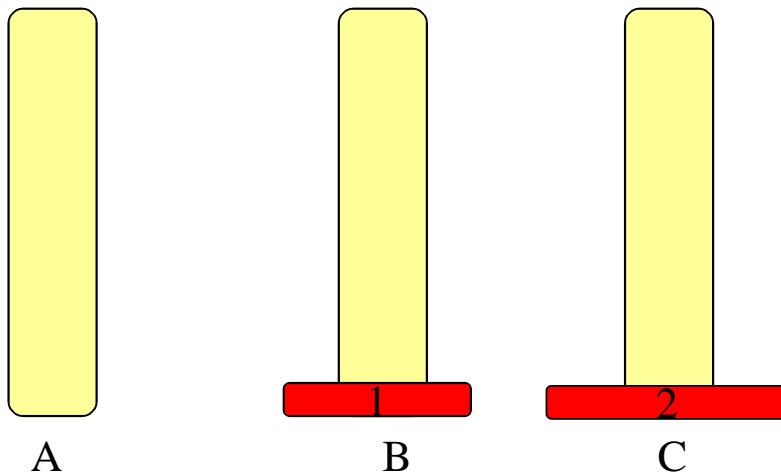
Trước hết chuyển đĩa 1 sang cọc B, sau đó đĩa 2 sang C



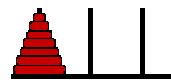
Tower of Hanoi - Hai đĩa



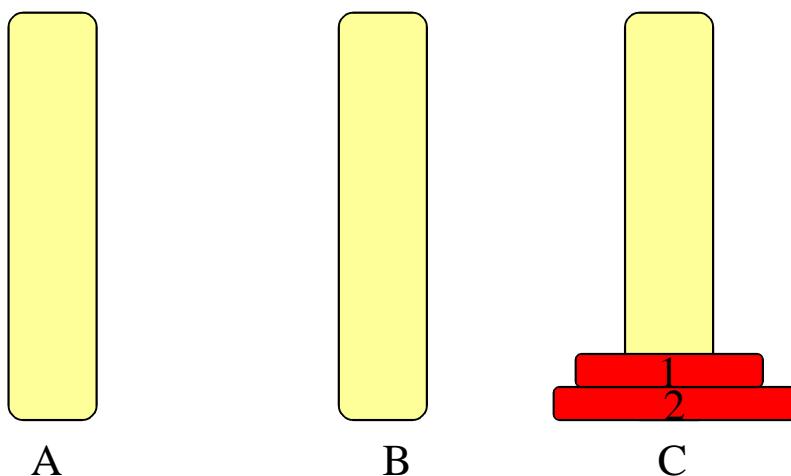
- Tiếp theo?
- Chuyển đĩa 1 sang C

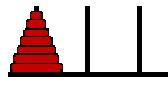


Tower of Hanoi - Hai đĩa



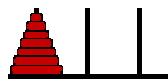
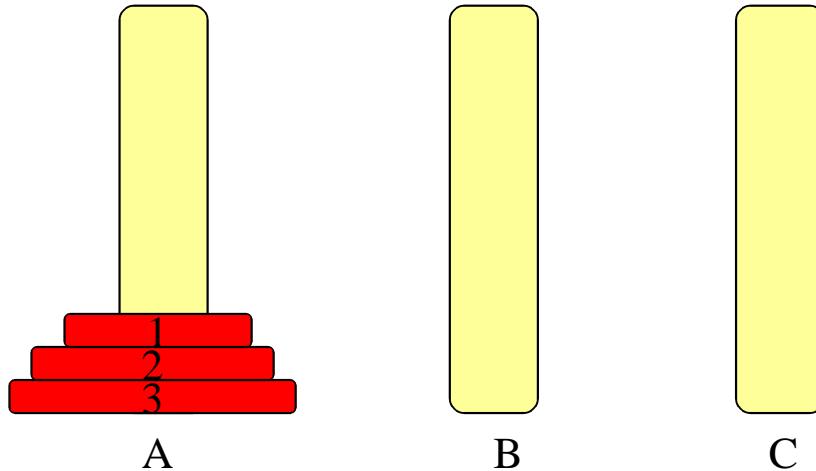
- Hoàn tất!





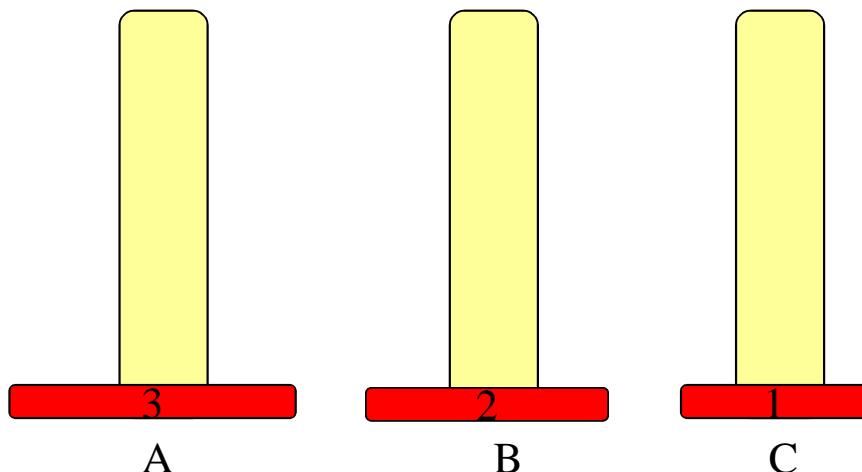
Tower of Hanoi - Ba đĩa

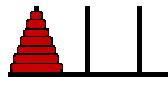
-
- Ta cần chuyển đĩa 3 sang C, bằng cách nào?
 - Chuyển đĩa 1 và 2 sang B (đệ qui)



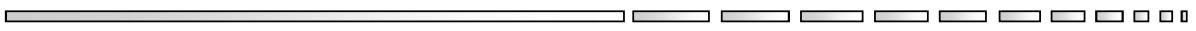
Tower of Hanoi - Ba đĩa

-
- 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

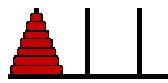
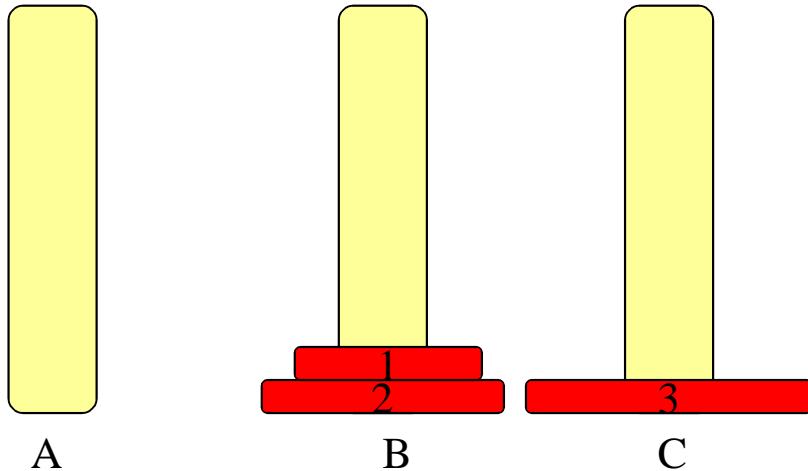




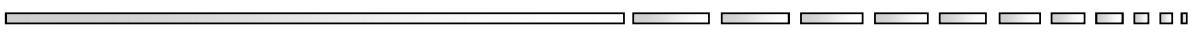
Tower of Hanoi - Ba đĩa



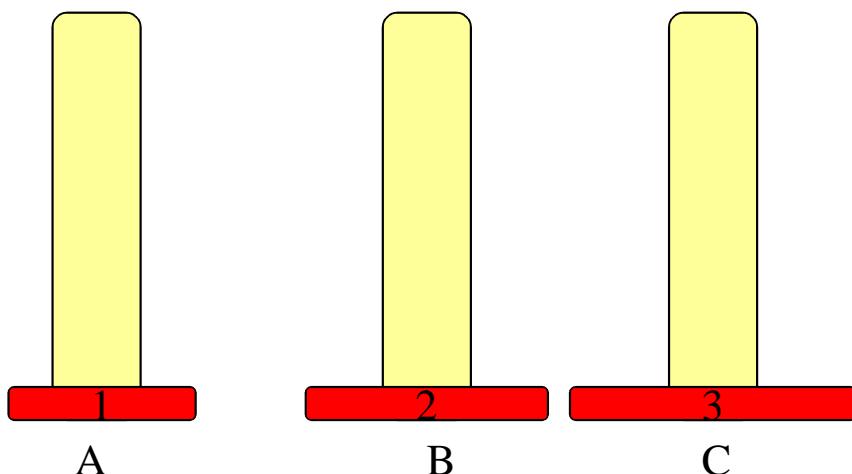
- Nhiệm vụ là: chuyển đĩa 1 và 2 sang C (tương tự như đã làm)



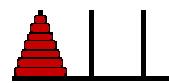
Tower of Hanoi - Ba đĩa



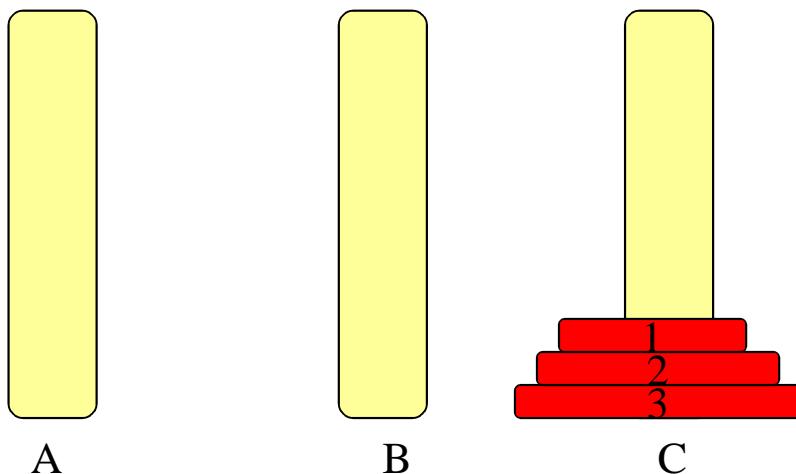
- Nhiệm vụ là: chuyển đĩa 1 và 2 sang C (tương tự như đã làm)



Tower of Hanoi - Ba đĩa



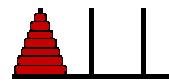
- Hoàn tất!



Cấu trúc dữ liệu và thuật toán - NGUYỄN ĐỨC NGHĨA, Bộ môn KHMT, ĐHBK Hà Nội

97

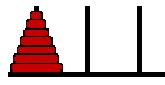
Bài toán tháp Hà nội



- 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.

Cấu trúc dữ liệu và thuật toán - NGUYỄN ĐỨC NGHĨA, Bộ môn KHMT, ĐHBK Hà Nội

98



Bài toán tháp Hà nội

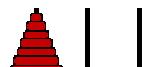
- Bước (i) và (iii) đòi hỏi giải bài toán tháp Hà nội với $n-1$ đĩa, và với n đĩa cần thực hiện trong hai bước này là $2h_{n-1}$. Do đó, nếu gãy h_n lùi về h_{n-1} di chuyển n đĩa, ta có công thức sau:

$$h_1 = 1,$$

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

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

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



Thuật toán tháp Hà nội

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

procedure HanoiTower(n , a, b, c);

begin

if $n=1$ **then** <chuyển đĩa từ cọc a sang cọc c>

else

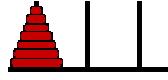
HanoiTower($n-1$, a, c, b);

HanoiTower(1, a, b, c);

HanoiTower($n-1$, b, a, c);

end;

Cài đặt trên C



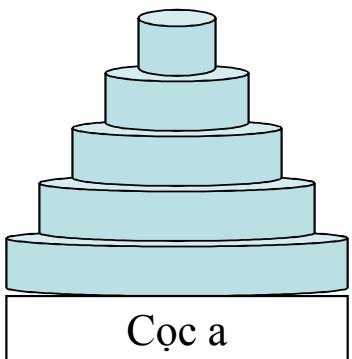
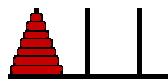
```
#include <stdio.h>
#include <conio.h>

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

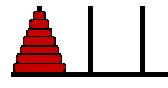
int main()
{
    int disk;
    printf ("Please input number of disk: ");
    scanf ("%d", &disk);
    move (disk, '1', '3', '2');
    printf ("Total number of moves = %d", i);
    getch();
    return 0;
}

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);
    }
}
```

Tower of Hanoi Example, n=5



Nội dung



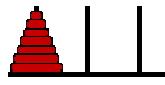
- 2.1. Khái niệm đệ qui
- 2.2. Thuật toán đệ qui
- 2.3. Một số ví dụ minh họa
- 2.4. Phân tích thuật toán đệ qui**
- 2.5. Đệ qui có nhớ
- 2.6. Chứng minh tính đúng đắn của thuật toán đệ qui



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

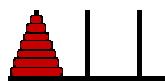
```
int FibRec(int n){  
    if (n<=1) return n;  
    else  
        return FibRec(n-1)+ FibRec(n-2);  
}
```

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

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

$$T(n) = T(n-1)+T(n-2)+1, n>1$$

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



Phân tích FibRec

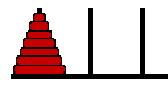
- Phân tích trên cho thấy FibRec có thời gian tính tăng với tốc độ $(1.6)^n$. Vì thế việc sử dụng FibRec là không hiệu quả.

- 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.

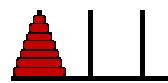
Phân tích thời gian của thuật toán chia để trị



- 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 quy sau đây để tính $T(n)$:

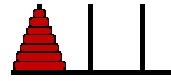
$$T(n) = \begin{cases} \Theta(1) & \text{nếu } n \leq n_0 \\ aT(n/b) + D(n) + C(n) & \text{nếu trái lại} \end{cases}$$

Sơ đồ thuật toán chia để trị



```
procedure D-and-C (n);  
begin  
  if n ≤ n0 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



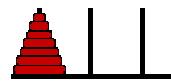
- Ta cần đưa ra đánh giá dưới dạng hiện cho $T(n)$
- Định lý thợ cung cấp công cụ để **đánh giá số hạng tổng quát** của dãy số thoả mãn công thức đệ qui dạng

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

- 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 $T(n) = \Theta(n^k)$.

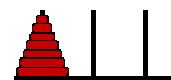
Ví dụ áp dụng định lý thố

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

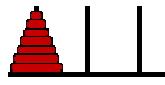
- Nếu $a > b^k$, thì $T(n) = \Theta(n^{\log_b a})$.
- Nếu $a = b^k$, thì $T(n) = \Theta(n^k \lg n)$.
- Nếu $a < b^k$, thì $T(n) = \Theta(n^k)$.

- 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 quy 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 thố:** $T(n) = \Theta(n \log n)$



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)  
    }  
}
```

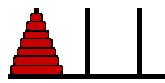
- Gọi $T(n)$ là thời gian tính của việc thực hiện Bsearch($x[1..n]$, $1,n$), ta có

$$T(1) = c$$

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

trong đó c và d là các hằng số.

- Từ đó theo định lý thố, $\textcolor{red}{T(n)} = \Theta(\log n)$.



Phân tích thuật toán tính $C(n,k)$

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

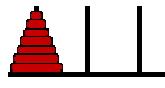
```
int C(int n, int k){  
    if ((k==0) || (k==n)) return 1;  
    else return C(n-1,k-1)+C(n-1,k);  
}
```

- 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 quy sau đây:

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

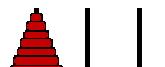
$$C^*(n,k) \geq C^*(n-1,k-1)+C^*(n-1,k)+1, \quad 0 < k < n$$

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



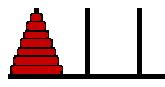
Nội dung

- 
- 2.1. Khái niệm đệ qui
 - 2.2. Thuật toán đệ qui
 - 2.3. Một số ví dụ minh họa
 - 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



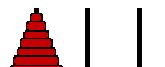
Đệ qui có nhớ

- 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.

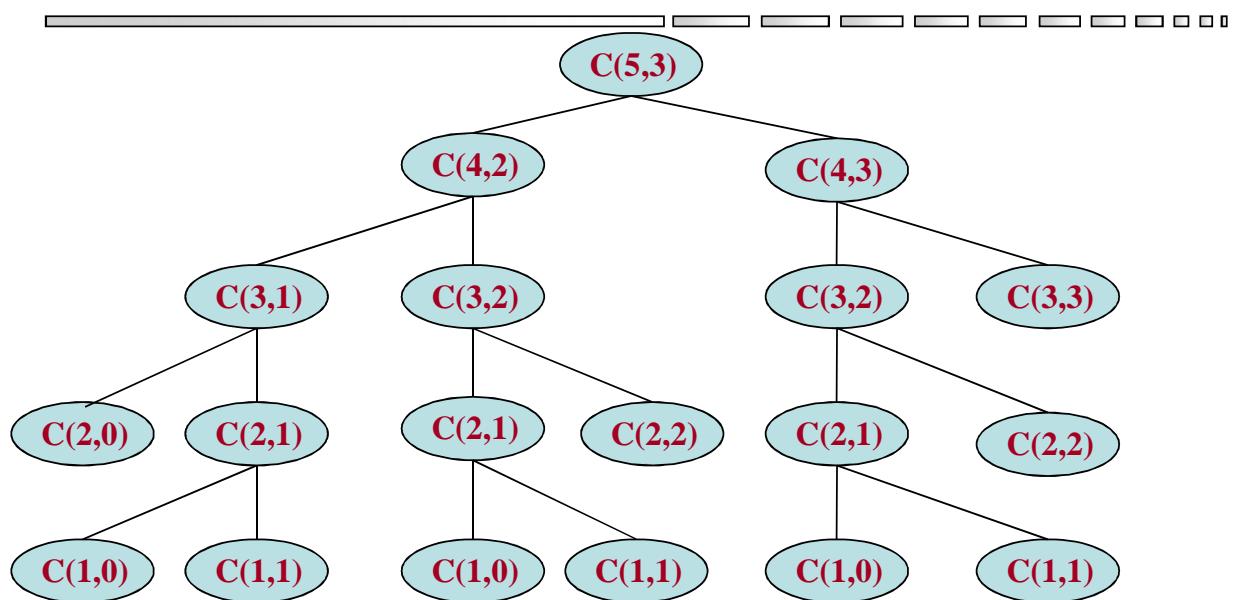


Bài toán con trùng lặp

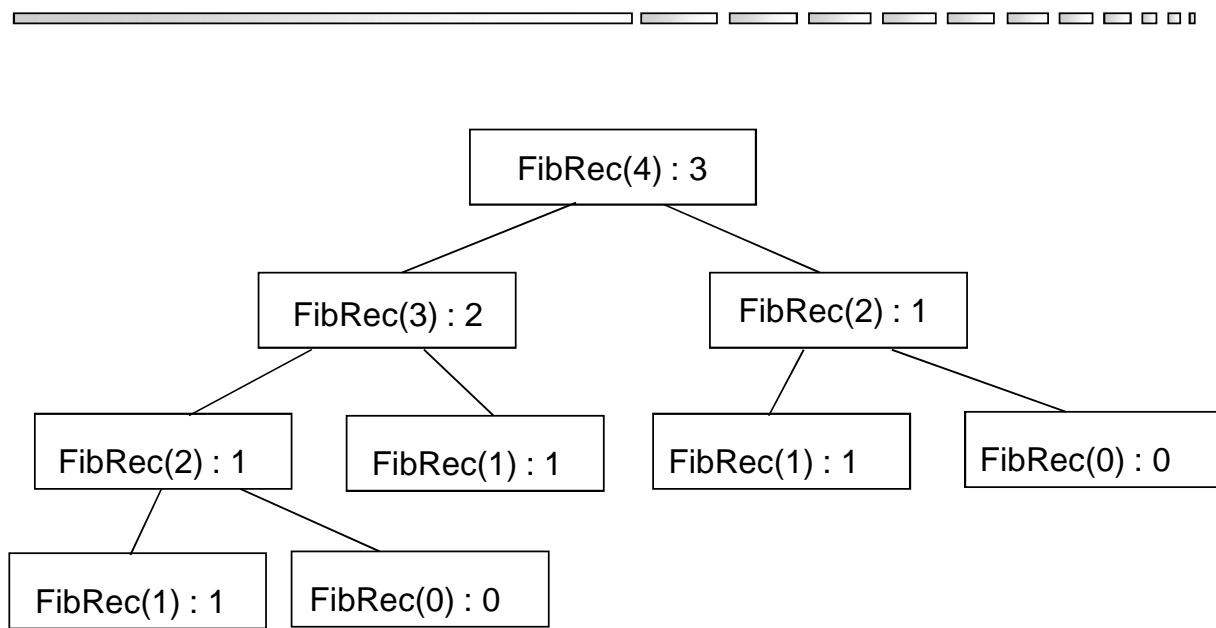
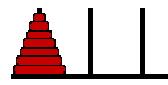
- Nhận thấy là trong các thuật toán đệ qui là mỗi khi cần đến lời giải của một bài toán con ta lại phải trả nó một cách đệ qui. Do đó, có những bài toán con bị giải đi giải lại nhiều lần. Điều đó dẫn đến tính kém hiệu quả của thuật toán. Hiện tượng này gọi là hiện tượng bài toán con trùng lặp.
- Ta xét 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



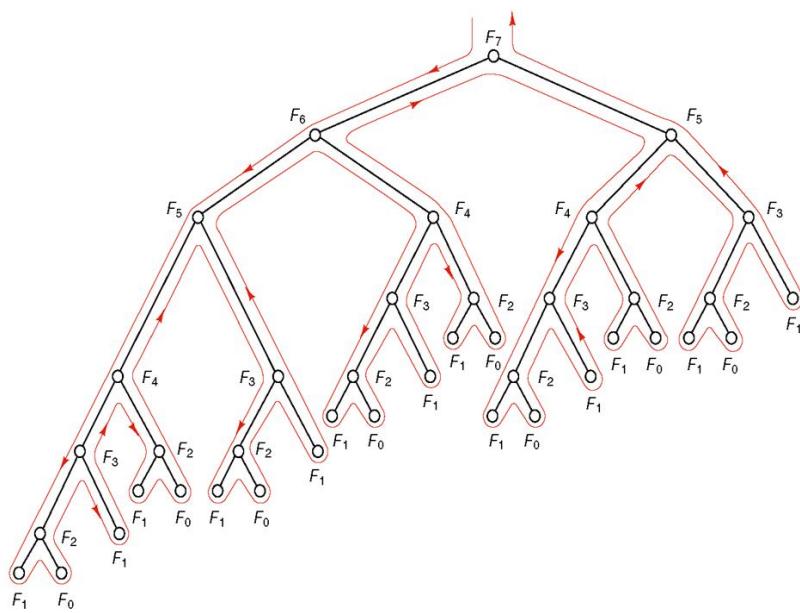
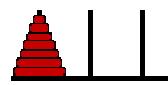
Bài toán con trùng lặp trong việc tính $C(5,3)$

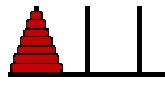


Bài toán con trùng lắp khi tính FibRec(4)



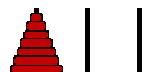
Cây đẽ qui tính F_7





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 $C(n,k)$

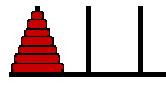


Hàm tính $C(n,k)$ có nhớ

```
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 $i=0,1,\dots,n$

Nội dung



2.1. Khái niệm đệ qui

2.2. Thuật toán đệ qui

2.3. Một số ví dụ minh họa

2.4. Phân tích thuật toán đệ qui

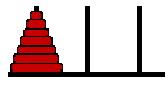
2.5. Đệ qui có nhớ

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 họa



Ví dụ 1

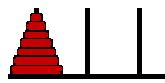
- **Ví dụ 1.** Chứng minh hàm Fact(n) cho ta giá trị của n!

```
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 $\text{Fact}(0) = 1 = 0!$
- **Chuyển qui nạp.** Giả sử $\text{Fact}(n-1)$ cho giá trị của $(n-1)!$, ta chứng minh $\text{Fact}(n)$ cho giá trị của $n!$ Thực vậy, lệnh $\text{Fact}(n)$ sẽ trả lại giá trị

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

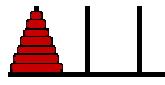


Ví dụ 2

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.

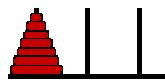


Ví dụ 2

Xét hàm trên C sau đây

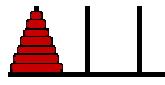
```
int Count(int x)
{
    if (x==0)
        return 0;
    else return (x % 2 + Count(x / 2));
}
```

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



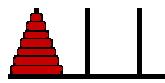
Ví dụ 3

- 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

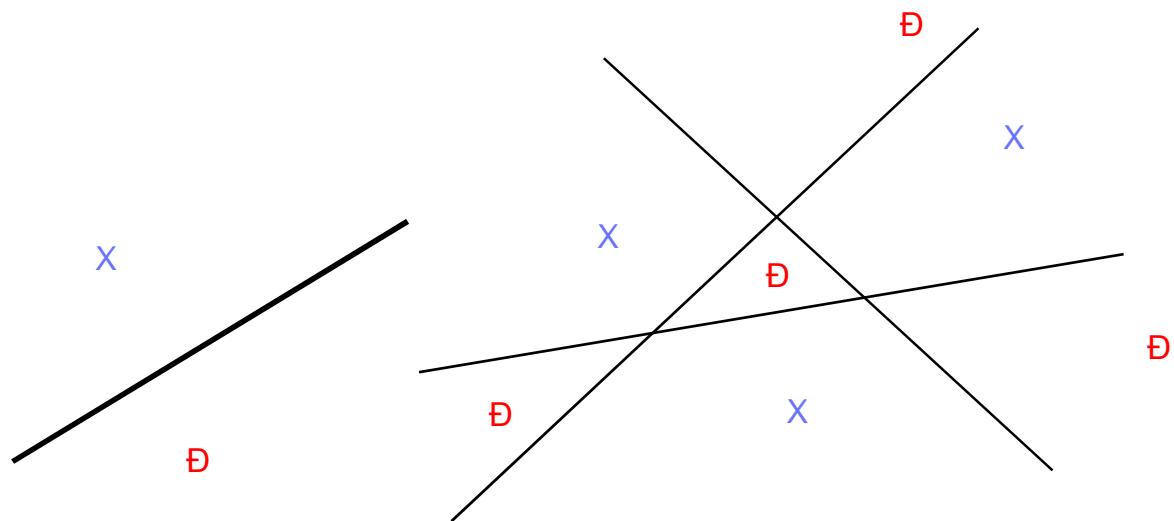
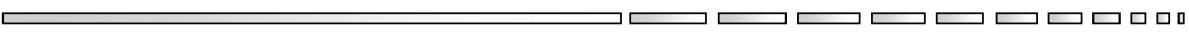
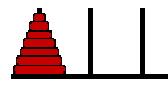
- **Cơ sở qui nạp.** Khi $n = 1$, mặt phẳng được chia làm hai phần, một phần sẽ tô màu xanh, phần còn lại tô màu đỏ.
- **Chuyển qui nạp.** Giả sử khẳng định đúng với $n-1$, ta chứng minh khẳng định đúng với n .
- Thực vậy, trước hết ta vẽ $n-1$ đường thẳng. Theo giả thiết qui nạp có thể tô màu các phần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



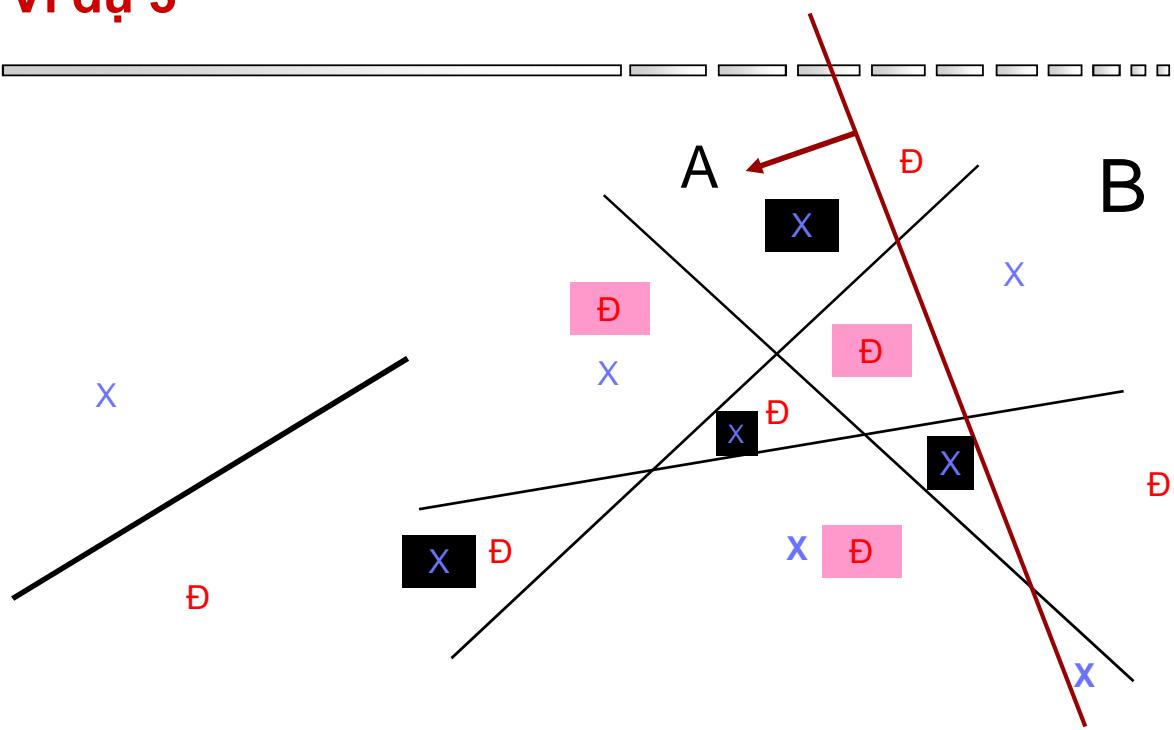
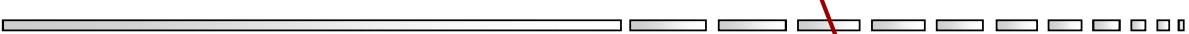
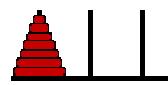
Ví dụ 3

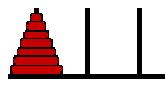
- Các phần của mặt phẳng được chia bởi n đường thẳng ở bên nửa mặt phẳng B sẽ giữ nguyên màu đã tô trước đó. Trái lại, các phần trong nửa mặt phẳng A mỗi phần sẽ được tô màu đảo ngược xanh thành đỏ và đỏ thành xanh.
- Rõ ràng:
 - Hai phần có chung cạnh ở cùng một nửa mặt phẳng A hoặc B là không có chung 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

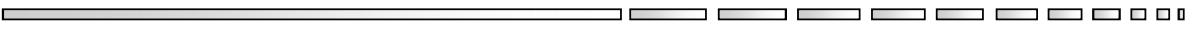


Ví dụ 3





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

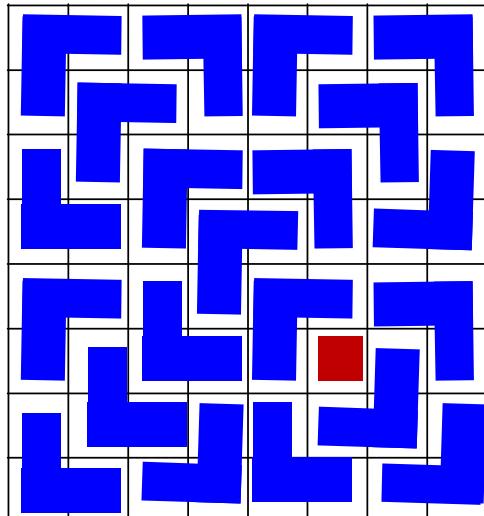


Cho lưới ô vuông kích thước $2^n \times 2^n$ bị đục mất một ô tùy ý.

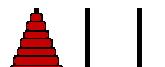
Có thể phủ kín lưới bởi viên gạch chữ L ?



Khẳng định:
Luôn phủ được với mọi n .

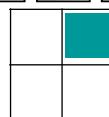


Ví dụ: Lưới 8×8 :



Chứng minh:

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

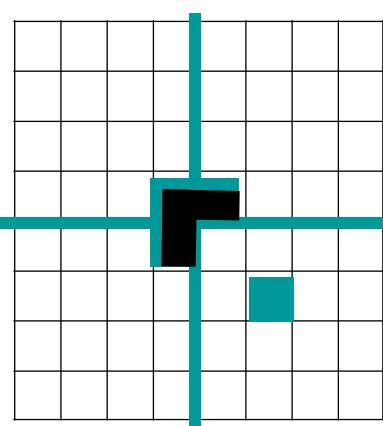


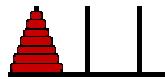
Chuyển qui nạp: Giả sử có thể phủ kín lưới $2^n \times 2^n$. Để chỉ ra có thể phủ lưới $2^{n+1} \times 2^{n+1}$, ta chia lưới thành 4 lưới con:

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.

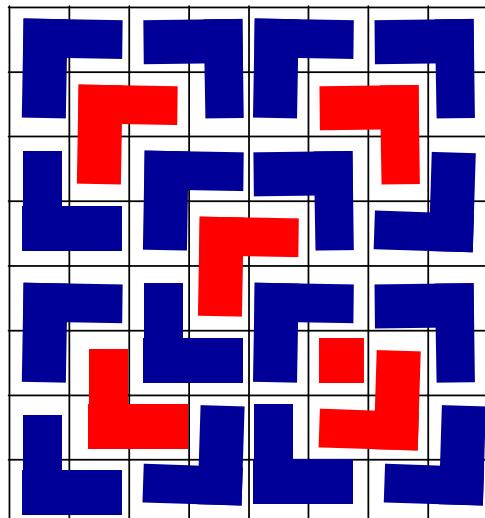
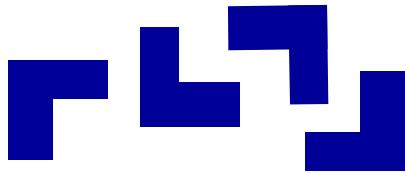
Lưu ý: Chứng minh bằng qui nạp mang tính xây dựng



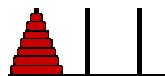


Phủ lưới $2^n \times 2^n$

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

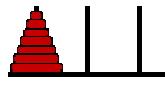


Ví dụ lưới 8×8 :



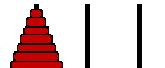
Ví dụ 5

- Kết thúc một giải vô địch bóng chuyền gồm n đội tham gia, trong đó các đội thi đấu vòng tròn một lượt người ta mời các đội trưởng của các đội ra đứng thành một hàng ngang để chụp ảnh.
- $P(n)$: Luôn có thể xếp n đội trưởng ra thành một hàng ngang sao cho ngoại trừ hai người đứng ở hai mép, mỗi người trong hàng luôn đứng cạnh một đội trưởng của đội thắng đội mình và một đội trưởng của đội thua đội mình trong giải.



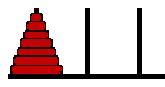
Ví dụ 5

- Cơ sở qui nạp: Rõ ràng $P(1)$ là đúng.
- Giả sử $P(n-1)$ là đúng, ta chứng minh $P(n)$ là đúng.
- Trước hết, ta xếp $n-1$ đội trưởng của các đội $1, 2, \dots, n-1$. Theo giả thiết qui nạp, luôn có thể xếp họ ra thành hàng ngang thỏa mãn điều kiện đầu bài. Không giảm tổng quát ta có thể giả thiết hàng đó là:
$$1 \rightarrow 2 \rightarrow \dots \rightarrow n-1.$$

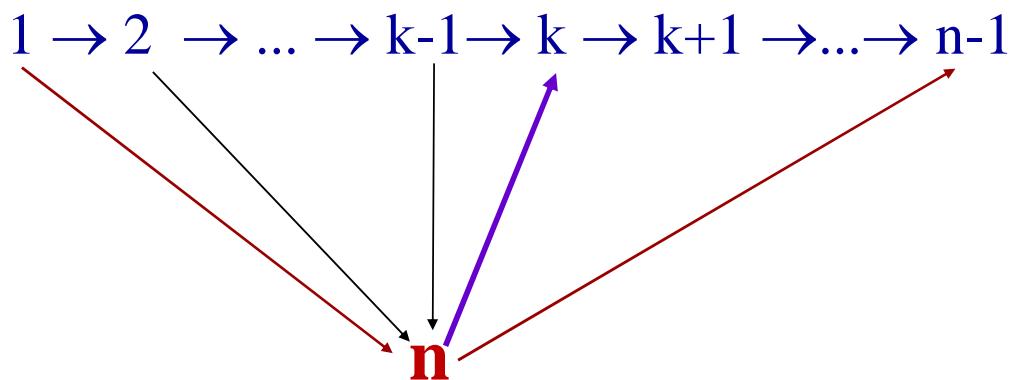


Ví dụ 5

- Bây giờ ta sẽ tìm chỗ cho đội trưởng của đội n . Có 3 tình huống:
 - Nếu đội n thắng đội 1 , thì hàng cần tìm là:
$$n \rightarrow 1 \rightarrow 2 \rightarrow \dots \rightarrow n-1.$$
 - Nếu đội n thua đội $n-1$, thì hàng cần tìm là:
$$1 \rightarrow 2 \rightarrow \dots \rightarrow n-1 \rightarrow n.$$
 - Nếu đội n thua đội 1 và thắng đội $n-1$.
 - Gọi k là chỉ số nhỏ nhất sao cho đội n thắng đội k .
 - Rõ ràng tồn tại k như vậy.

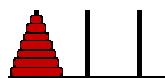


Ví dụ 5

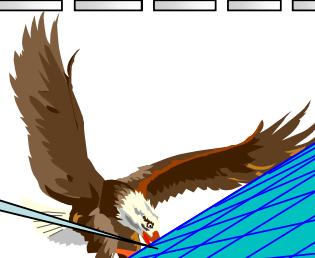


Hàng cần tìm:

$$1 \rightarrow \dots \rightarrow k-1 \rightarrow \mathbf{n} \rightarrow k \rightarrow k+1 \rightarrow \dots \rightarrow n-1$$



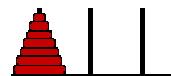
Questions?



THUẬT TOÁN QUAY LUI

Backtracking Algorithm

SƠ ĐỒ THUẬT TOÁN



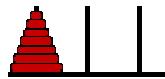
- Thuật toán quay lui (Backtracking Algorithm) là một thuật toán cơ bản được áp dụng để giải quyết nhiều vấn đề khác nhau.
- Bí quyết liệt k^a (Q):** Cho A_1, A_2, \dots, A_n là các tệp hữu hạn. Ký hiệu

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

Giả sử P là tệp chép cho tần X. Vẽ \emptyset và đặt ra là liệt k^a tệp cung cấp phán tử của X thoả mãn tệp chép P:

$$D = \{ x = (x_1, x_2, \dots, x_n) \in X : x \text{ thoả mãn tệp chép } P \}.$$

- Các phán tử của X thoả mãn tệp chép D được gọi là các lời giải chấp nhận được.



Ví dụ

- Bạn to, n liöt k^a x[©]u nh^b ph^cn ®é d^du i n d^eñ v^fØ viÖc liöt k^a c^g, c^h phⁱn tö cña t^jEp

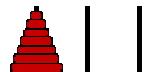
$$B^n = \{(x_1, \dots, x_n) : x_i \in \{0, 1\}, i=1, 2, \dots, n\}.$$

- Bạn to, n liöt k^a c^g, c^h t^jEp con m phⁱn tö cña t^jEp $N = \{1, 2, \dots, n\}$ ®Bi hái liöt k^a c^g, c^h phⁱn tö cña t^jEp:

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

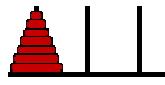
- T^jEp c^g, c^h ho, n vⁱ cña c^g, c^h sè tù nhi^an 1, 2, ..., n l^ju t^jEp

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



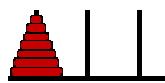
Lời giải bộ phận

- Sinh nghĩa.** Ta gäi l^jei giⁱli bé ph^en c^hEp k ($0 \leq k \leq n$) l^ju bé c^ha thø tù gäm k thønh phⁱn (a_1, a_2, \dots, a_k), trong ®ã $a_i \in A_i$, $i = 1, 2, \dots, k$.
- Khi $k = 0$, lời giải bộ phận cấp 0 được ký hiệu là () và còn được gäi l^ju l^jei giⁱli rçng.
- NÔu $k = n$, ta cä l^jei giⁱli ®Çy ®ñ hay ®¬n giⁱn l^ju mét l^jei giⁱli cña b^jui to, n.



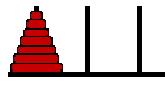
Ý tưởng chung

- 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 $k-1$: $(a_1, a_2, \dots, a_{k-1})$.
- 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 $(a_1, a_2, \dots, a_{k-1})$. Ký hiệu tập các ứng cử viên này là S_k .



Xét hai tình huống

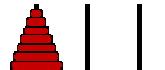
-
- $S_k \neq \emptyset$. Khi đó lối $a_k \in S_k$, bổ sung nã vào lối giải bé phết cấp $k-1$ sang cã

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

ta thu được lời giải bộ phận cấp k:

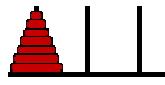
$$(a_1, a_2, \dots, a_{k-1}, a_k).$$

- Nếu $k = n$ thì ta thu được một lời giải,
- Nếu $k < n$, ta tiếp tục xem xét dùng thịnh phách thợ $k+1$ cãa lối giải.



Tình huống ngõ cụt

-
- $S_k = \emptyset$. Điều đó có nghĩa là lời giải bộ phận $(a_1, a_2, \dots, a_{k-1})$ 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 lối 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

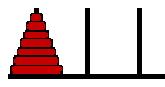
```
procedure Backtrack(k: integer);  
begin  
    Xây dựng Sk;  
    for y ∈ Sk do (* Với mỗi UCV y từ Sk *)  
        begin  
            ak := y;  
            if k = n then <Ghi nhận lời giải (a1, a2, ..., ak)>  
            else Backtrack(k+1);  
        end;  
    end;
```

Lệnh gõ ®ết thúc hiÖn thuËt to_n quay lui lµ:
Backtrack(1)



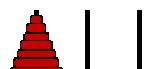
Hai vấn đề mâu chốt

- Đó cµi ®ết thuËt to_n quay lui gi¶li c_s c bµi to_n tæ hî p cô thÓ ta cÇn gi¶li quyÖt hai vÊn ®Ø c¬ b¶n sau:
 - Tìm thuËt to_n x©y dùng c_s c tËp UCV S_k.
 - Tìm c_s ch m« t¶ c_s c tËp nµy ®Ø cã thÓ cµi ®ết thao t_s c liÖt kª c_s c phÇn tö cña chúng (cài đặt vòng lặp qui ước **for y ∈ S_k do**).
- HiÖu qu¶ cña thuËt to_n liÖt kª phô thuéc vµo viÖc ta cã xác định được chính xác các tập UCV này hay không.



Chú ý

- Nếu chưa tìm mét lẻi gipi thì cần tìm cách chém đứt cách thõa mãn \oplus qui láng nhau sinh bởi lệnh gaji **Backtrack(1)** sau khi ghi nhận được lẻi gipi \oplus của tia.
- Nếu kết thúc thuết toán mực ta không thu được mét lẻi gipi nào thì \oplus là \oplus cả nghĩa là bụi tia không có lẻi gipi.
- Thuết toán dưới dạng mẻ riêng cho bụi tia liết k^a trong \oplus là lẻi gipi cả thõa mãn t \oplus như là bé ($a_1, a_2, \dots, a_n, \dots$) \oplus là bụi hữu hạn, tuy nhiên giao tiếp cña \oplus là bụi không biết trước vụ cách lẻi gipi cùng không nhận thiết phipi cả cing \oplus là bụi.



Chú ý

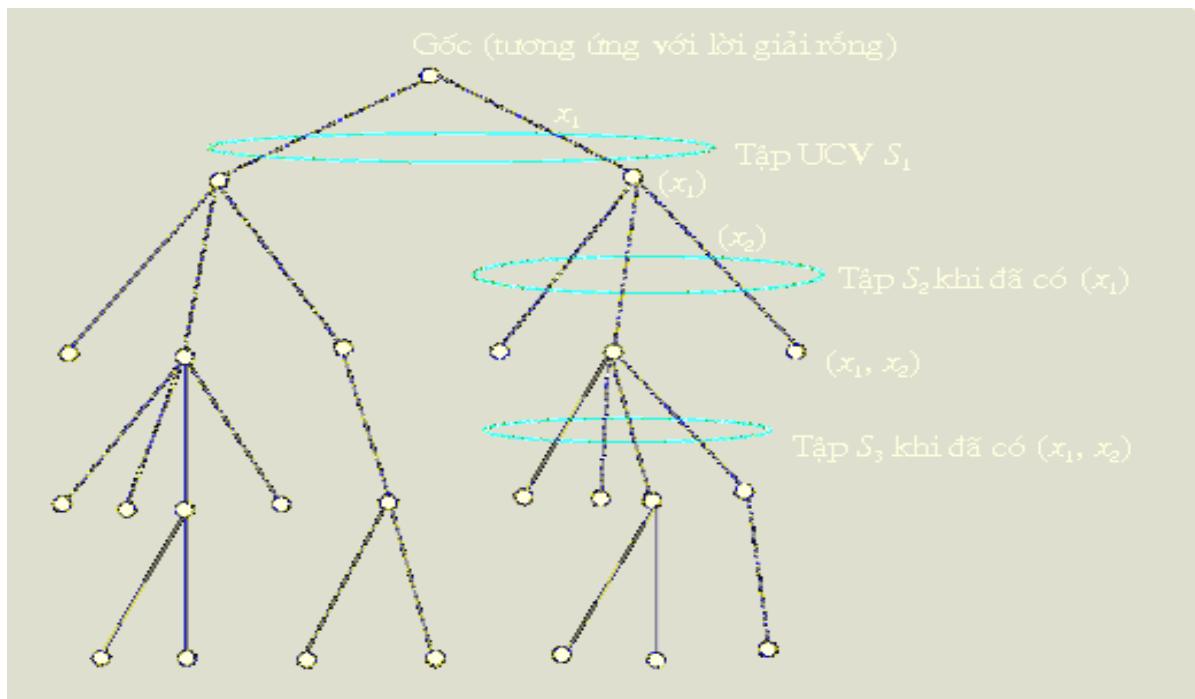
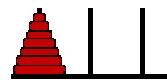
- Khi \oplus chưa sôa lⁱi c^ou lệnh

```
if k = n then <Ghi nhận lẻi gipi (a1, a2, ..., ak)>
else Backtrack(k+1);
```

thì

```
if <(a1, a2, ..., ak) lùi lẻi gipi> then <Ghi nhận (a1, a2, ..., ak)>
else Backtrack(k+1);
```
- Ta cần x^oy dùng h^um nhận biết (a₁, a₂, ..., a_k) \oplus . lùi lẻi gipi hay chưa.

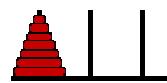
Cây liệt kê lời giải theo thuật toán quay lui



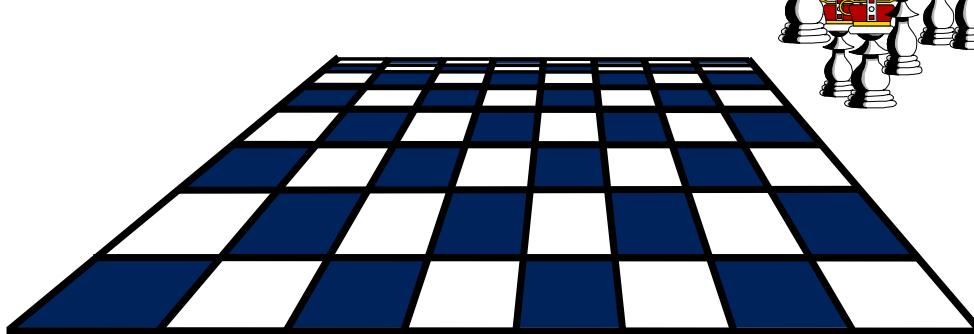
Cấu trúc dữ liệu và thuật toán - NGUYỄN ĐỨC NGHĨA, Bộ môn KHMT, ĐHBK Hà Nội

153

Bài toán xếp hậu

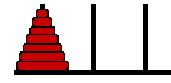


- Liệt kê tất cả các cách xếp n quân Hậu trên bàn cờ $n \times n$ sao cho chúng không ăn được lẫn nhau, nghĩa là sao cho không có hai con nào trong số chúng nằm trên cùng một dòng hay một cột hay một đường chéo của bàn cờ.



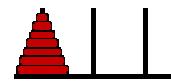
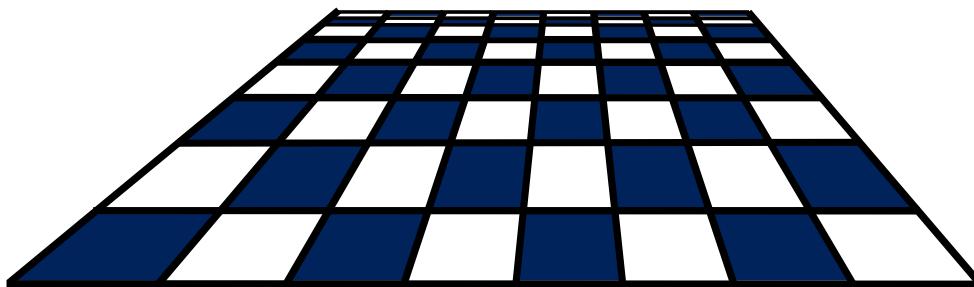
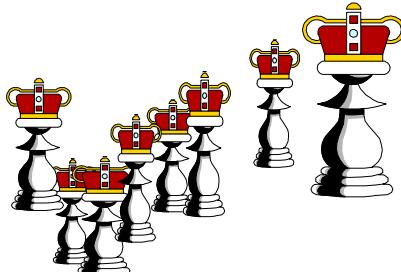
Cấu trúc dữ liệu và thuật toán - NGUYỄN ĐỨC NGHĨA, Bộ môn KHMT, ĐHBK Hà Nội

154



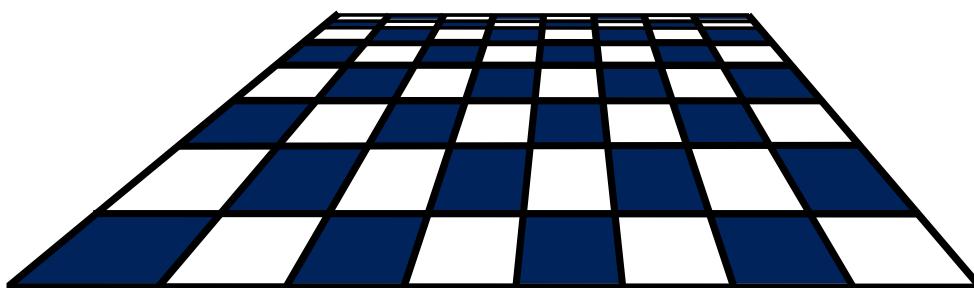
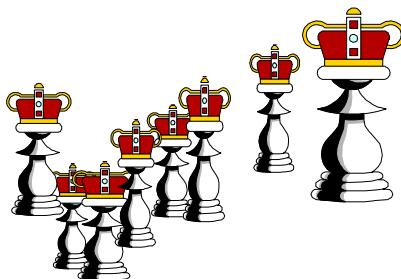
The *n*-Queens Problem

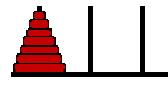
-
- Giả sử ta có 8 con hậu...
 - ...và bàn cờ quốc tế



The *n*-Queens Problem

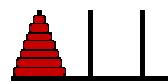
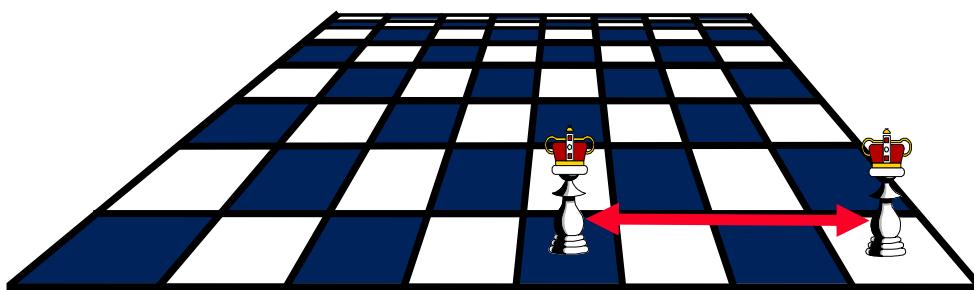
Có thể xếp các con hậu sao cho không có hai con nào ăn nhau hay không ?





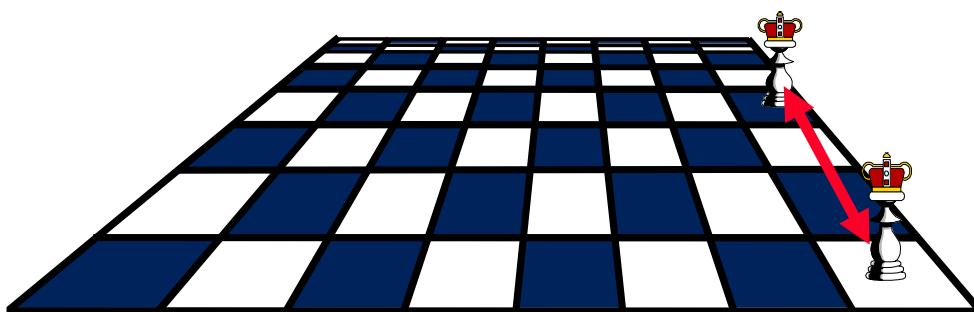
The n-Queens Problem

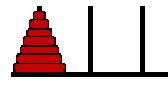
Hai con hậu bất kỳ
không được xếp trên
cùng một dòng ...



The n-Queens Problem

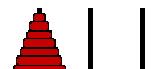
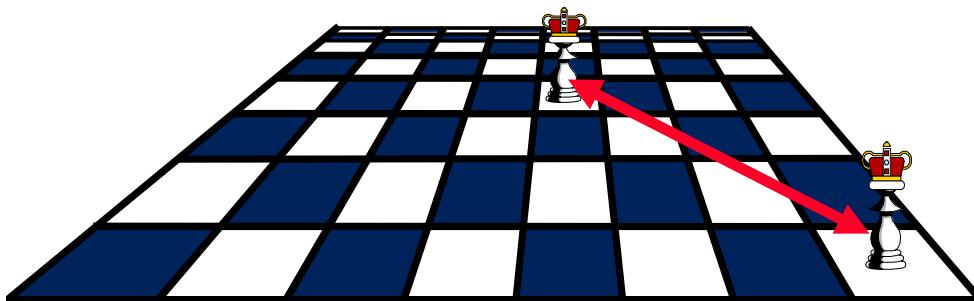
Hai con hậu bất kỳ
không được xếp trên
cùng một cột ...





The n-Queens Problem

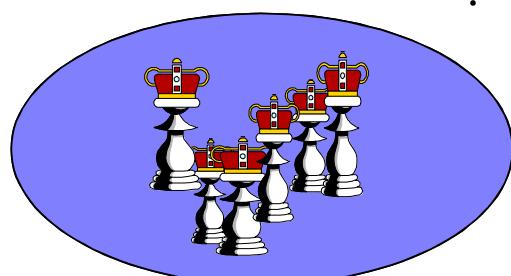
Hai con hậu bất kỳ
không được xếp trên
cùng một dòng, một cột
hay một đường chéo!



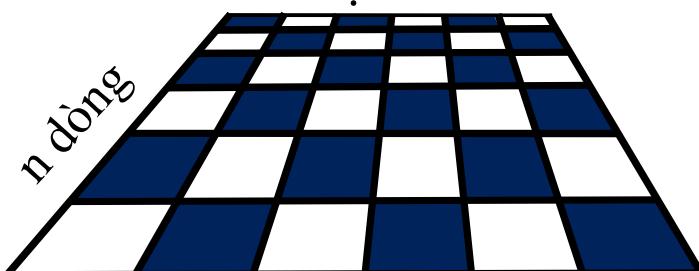
The n-Queens Problem

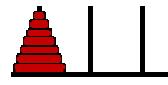
Kích thước n !

n con hậu

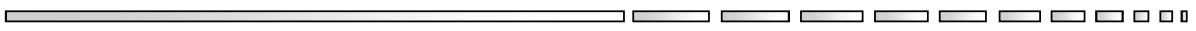


n cột

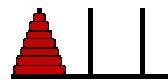
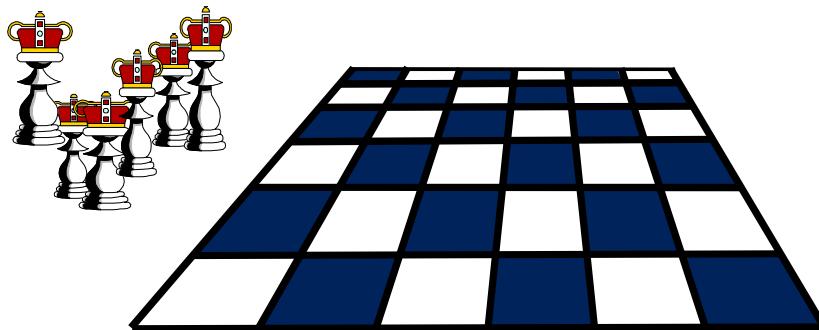




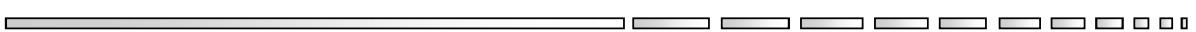
The n-Queens Problem



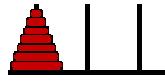
Xét bài toán xếp n con
hậu lên bàn cờ kích
thước $n \times n$.



Biểu diễn lời giải



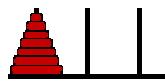
- Đánh số các cột và dòng của bàn cờ từ 1 đến n . Một cách xếp hậu có thể biểu diễn bởi bộ có n thành phần (a_1, a_2, \dots, a_n) , trong đó a_i là toạ độ cột của con Hậu ở dòng i .
- Các điều kiện đặt ra đối với bộ (a_1, a_2, \dots, a_n) :
 - $a_i \neq a_j$, ví i mäi $i \neq j$ (nghĩa là hai con h \ddot{E} u \ddot{e} hai d \ddot{B} ng i v \ddot{u} j kh \ddot{E} ng được n \gg m tr \ddot{a} n c \ddot{i} ng m \ddot{E} t c \ddot{E} t);
 - $|a_i - a_j| \neq |i - j|$, ví i mäi $i \neq j$ (nghĩa là hai con h \ddot{E} u \ddot{e} hai « (a_i, i) v \ddot{u} (a_j, j) kh \ddot{E} ng được n \gg m tr \ddot{a} n c \ddot{i} ng m \ddot{E} t đ \ddot{u} ng ch \ddot{D} o).



Phát biểu bài toán

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

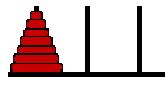
$$D = \{(a_1, a_2, \dots, a_n) \in N^n : a_i \neq a_j \text{ và } |a_i - a_j| \neq |i - j|, i \neq j\}.$$



Hàm nhận biết ứng cử viên

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

```
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

```

var n, count: integer;
    a: array[1..20] of integer;
procedure Ghinhан;
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 ∈ 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;

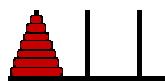
```

```

#include <stdio.h>
#include <math.h>
int n, count;
int a[20];
int Ghinhан() {
    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 và chỉ khi j ∈ Sk */
    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 Ghinhан
                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 ®Ó kôt thóc... ');
    readln;
END.

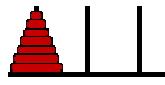
```

```

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

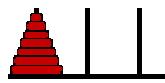
int main() {
    printf("Input n = "); scanf("%i",&n);
    printf("\n==== RESULT ====\n");
    count = 0; Hau(1);
    if (count == 0) printf("No solution!\n");
    getchar();
    printf("\n Press Enter to finish... ");
    getchar();
}

```

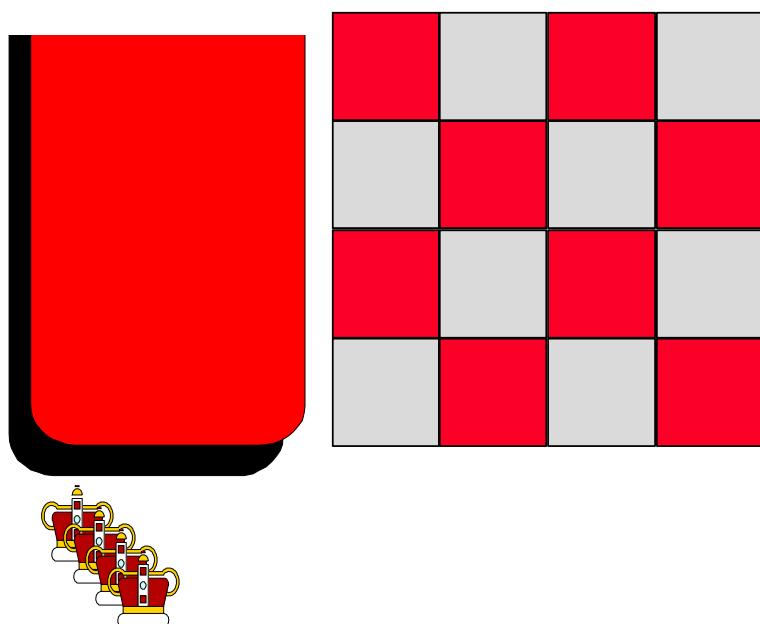


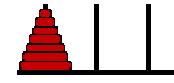
Chú ý

-
- 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.

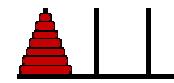
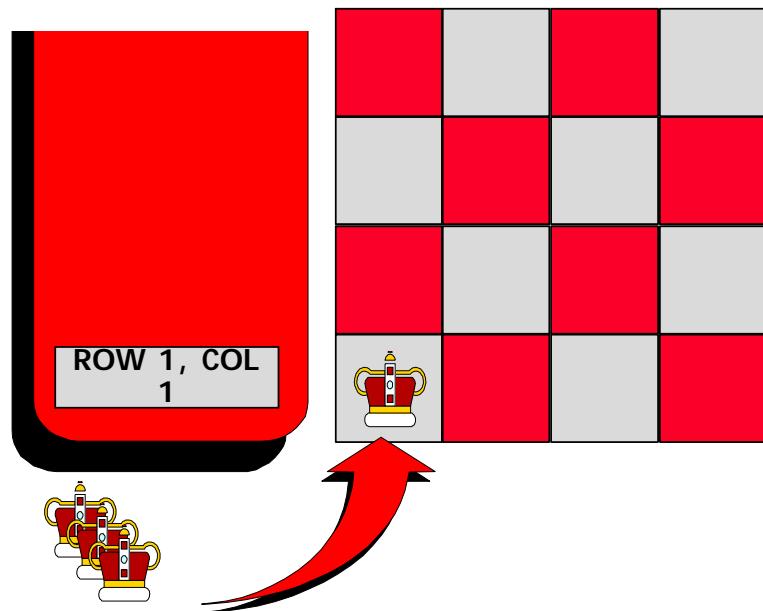
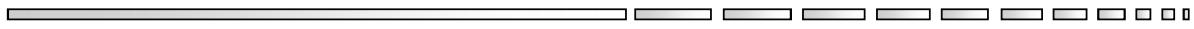


Thuật toán làm việc như thế nào





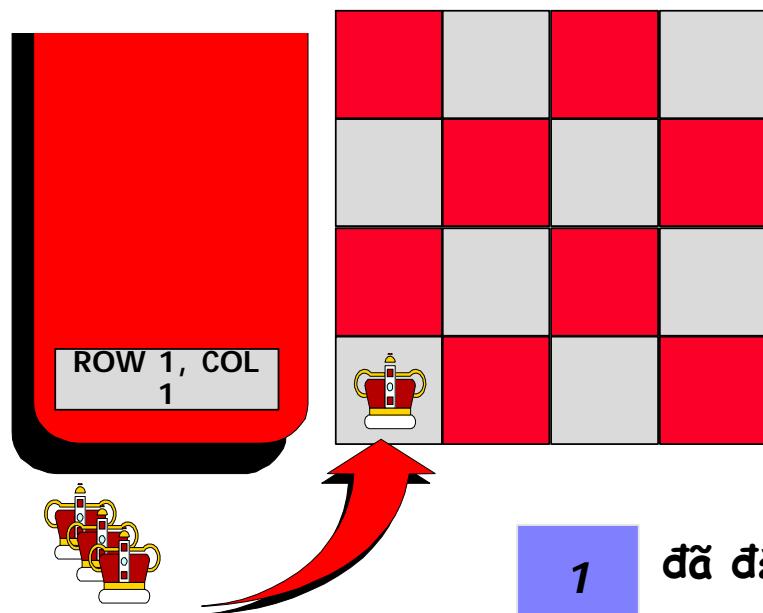
Thuật toán làm việc như thế nào

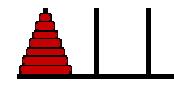


Thuật toán làm việc như thế nào



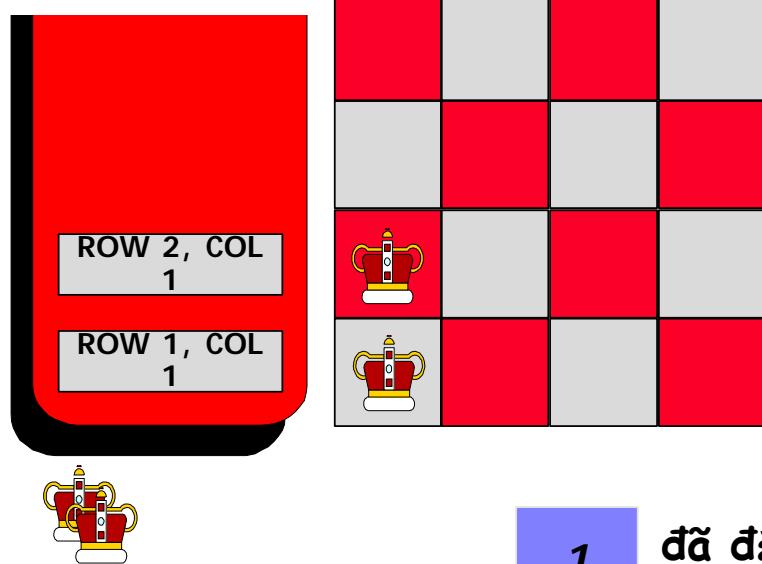
- Xếp con hậu ở dòng 1
vào vị trí cột 1





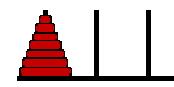
Thuật toán làm việc như thế nào

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



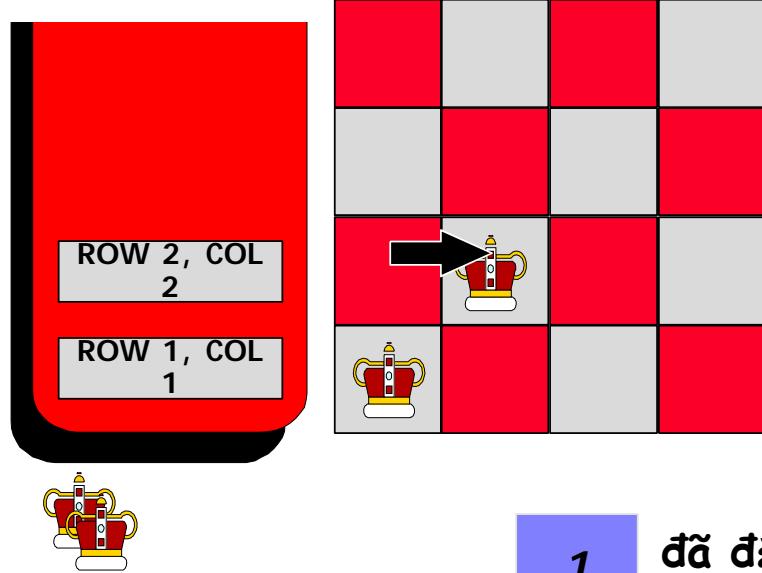
Cấu trúc dữ liệu và thuật toán - NGUYỄN ĐỨC NGHĨA, Bộ môn KHMT, ĐHBK Hà Nội

171



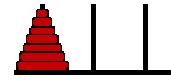
Thuật toán làm việc như thế nào

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



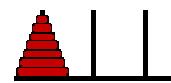
Cấu trúc dữ liệu và thuật toán - NGUYỄN ĐỨC NGHĨA, Bộ môn KHMT, ĐHBK Hà Nội

172



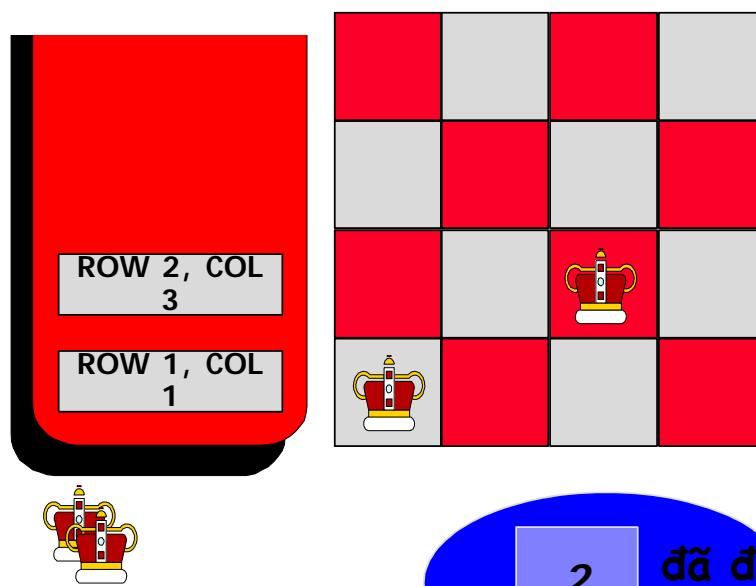
Thuật toán làm việc như thế nào

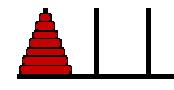
- Thứ xếp con hậu ở dòng 2 vào vị trí cột 3



Thuật toán làm việc như thế nào

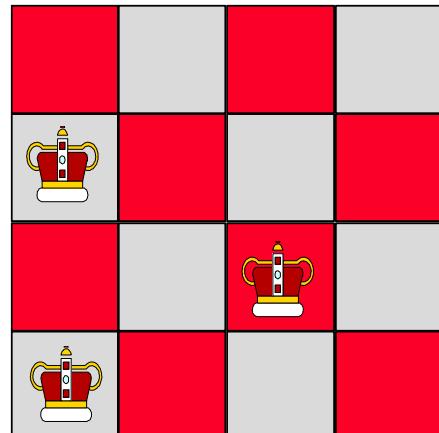
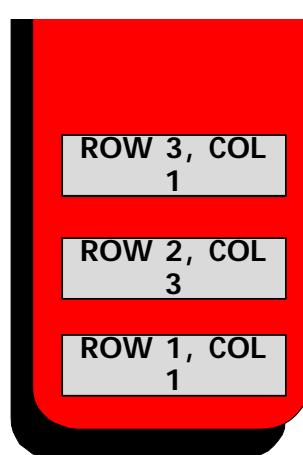
- Chấp nhận xếp con hậu ở dòng 2 vào vị trí cột 3



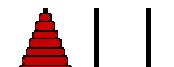


Thuật toán làm việc như thế nào

Thứ xếp con hậu ở
dòng 3
vào cột đầu tiên

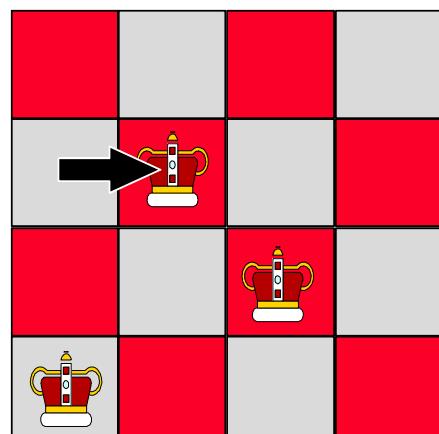
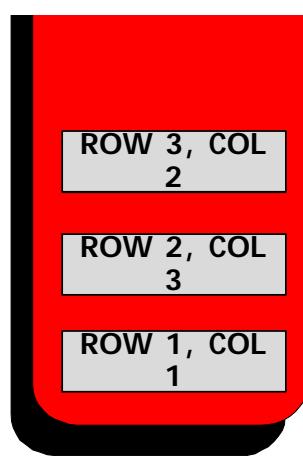


2 đã đặt

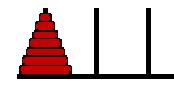


Thuật toán làm việc như thế nào

Thứ cột tiếp theo

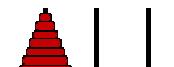
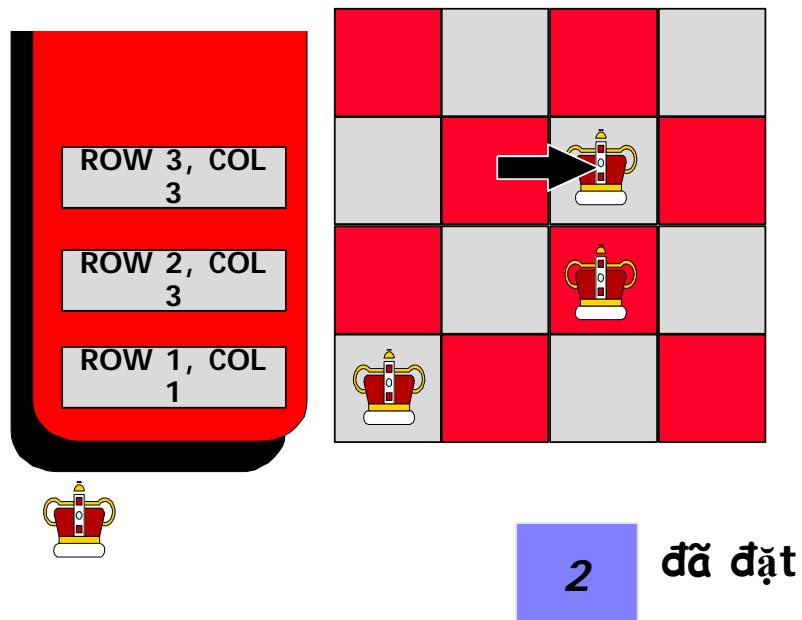


2 đã đặt



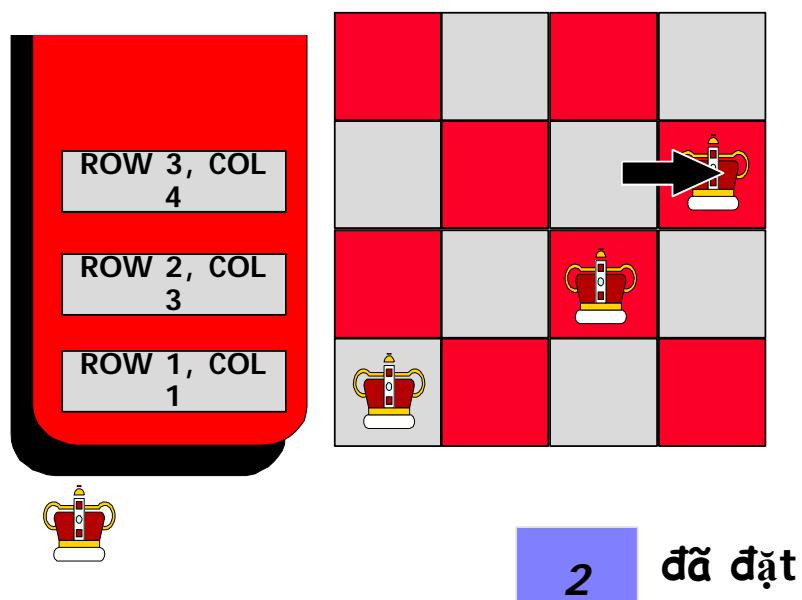
Thuật toán làm việc như thế nào

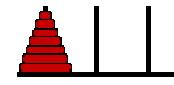
- Thứ cột tiếp theo



Thuật toán làm việc như thế nào

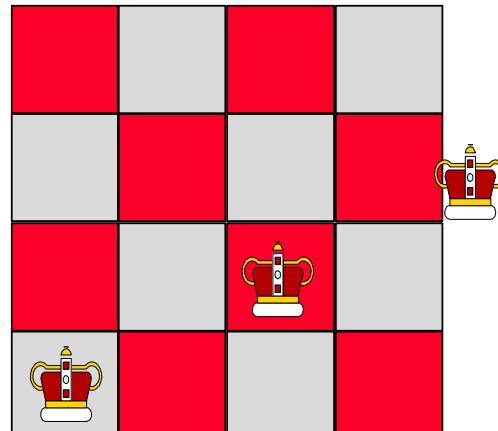
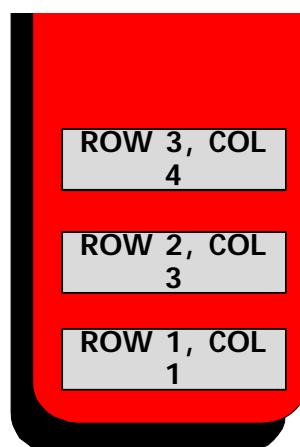
- Thứ cột tiếp theo





Thuật toán làm việc như thế nào

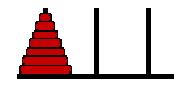
...không có vị trí đặt con hậu ở dòng 3.



2 đã đặt

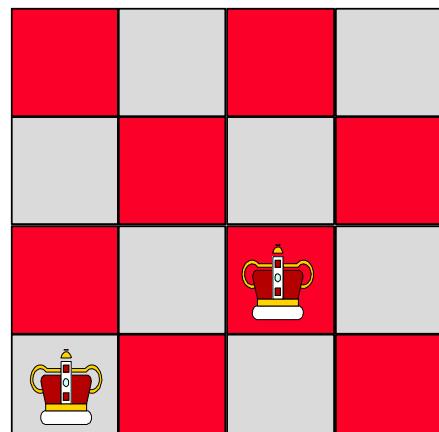
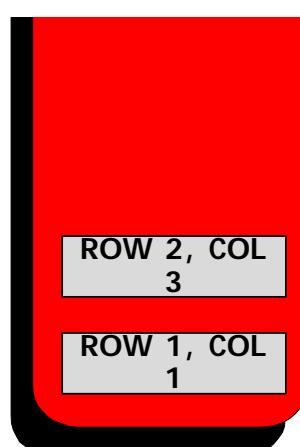
179

Cấu trúc dữ liệu và thuật toán - NGUYỄN ĐỨC NGHĨA, Bộ môn KHMT, ĐHBK Hà Nội



Thuật toán làm việc như thế nào

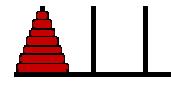
Quay lại dịch chuyển con hậu ở dòng 2



1 đã đặt

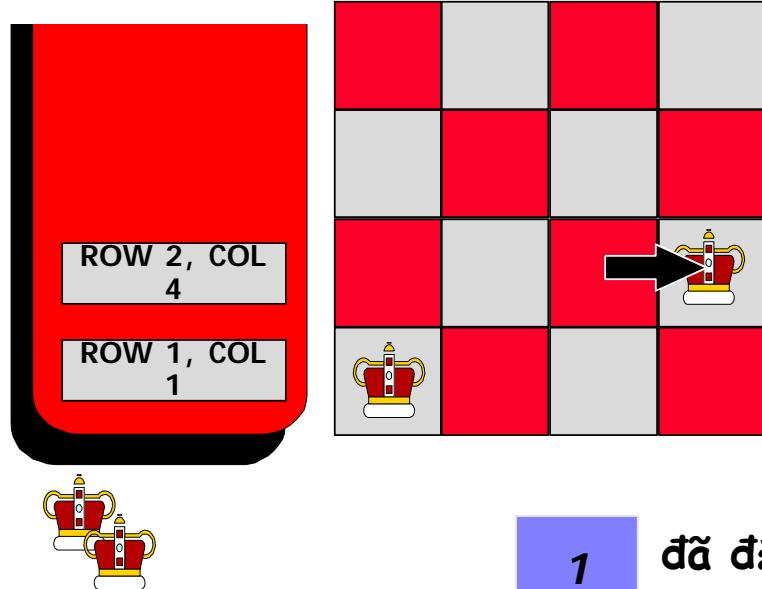
180

Cấu trúc dữ liệu và thuật toán - NGUYỄN ĐỨC NGHĨA, Bộ môn KHMT, ĐHBK Hà Nội



Thuật toán làm việc như thế nào

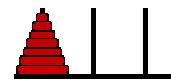
Đẩy con hậu ở dòng 2 sang cột thứ 4.



1 đã đặt

181

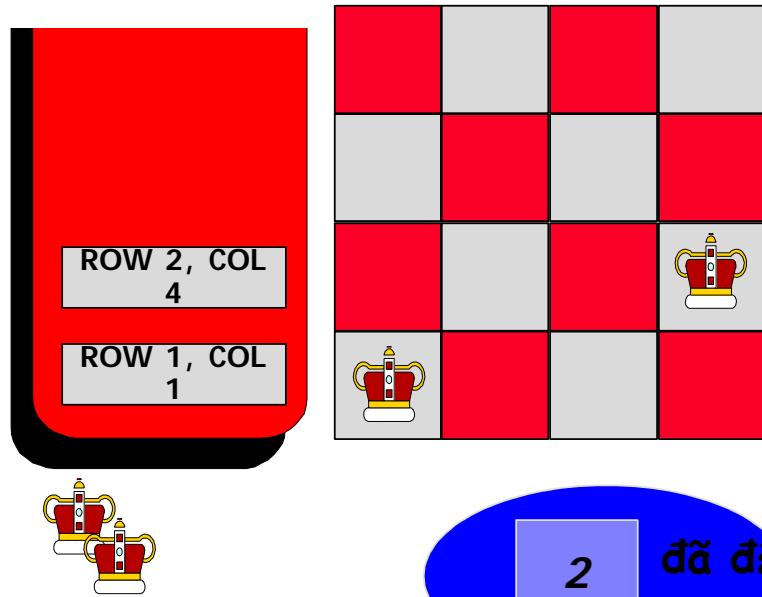
Cấu trúc dữ liệu và thuật toán - NGUYỄN ĐỨC NGHĨA, Bộ môn KHMT, ĐHBK Hà Nội



Thuật toán làm việc như thế nào

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

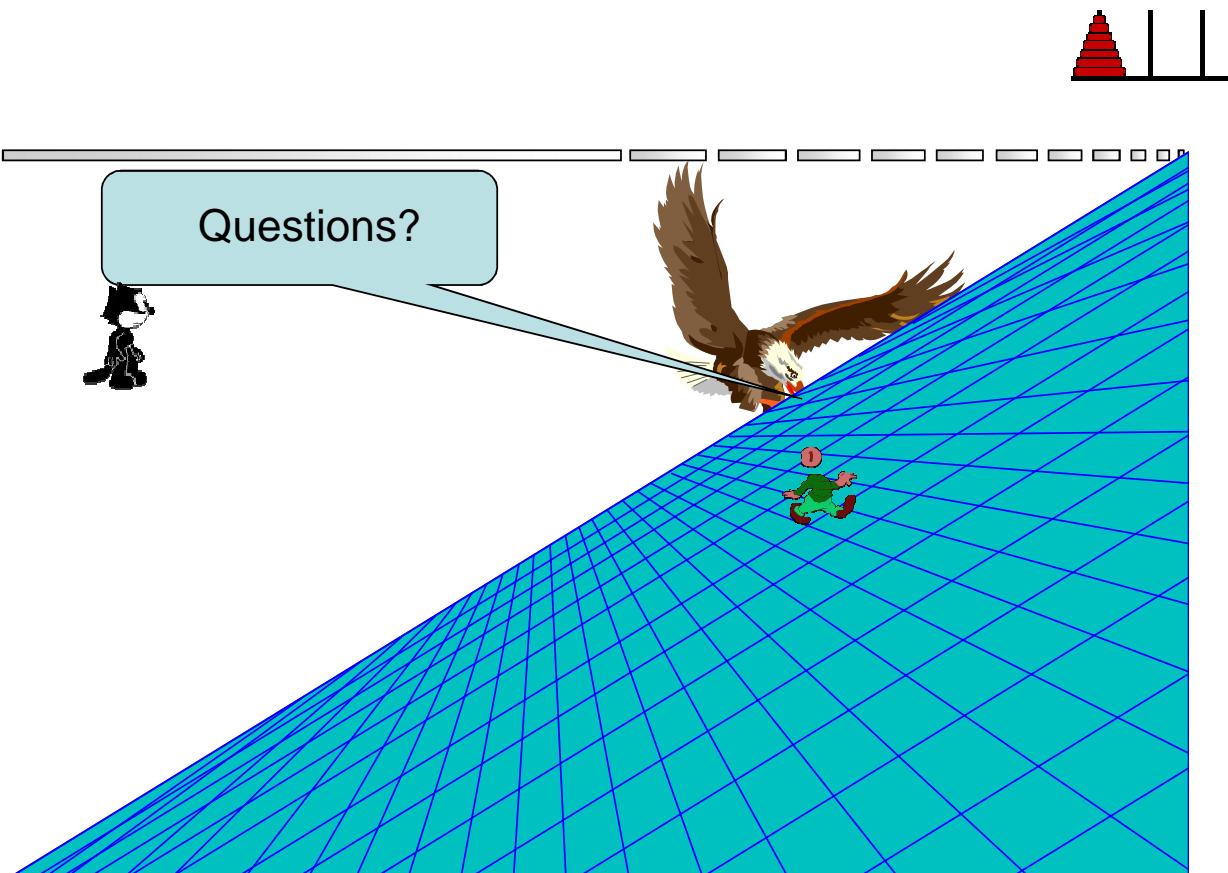
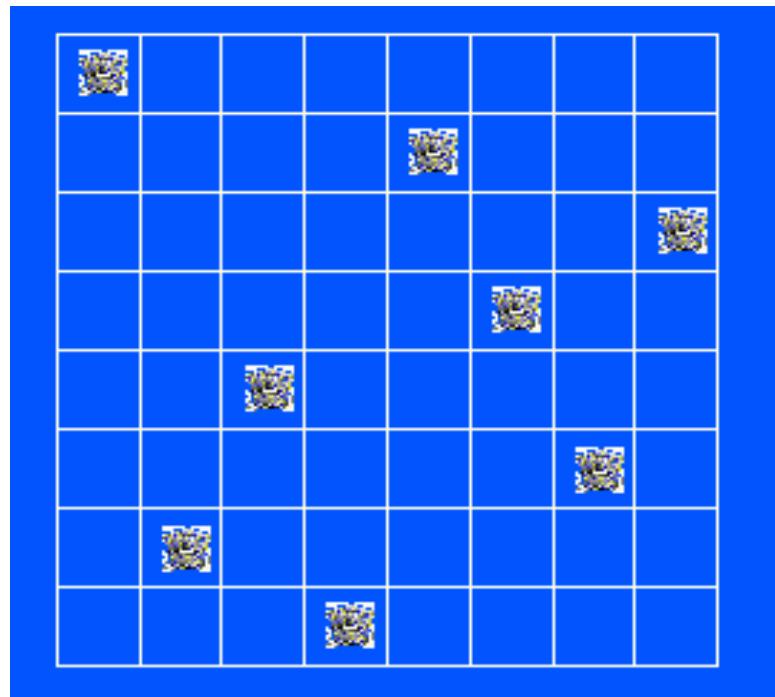
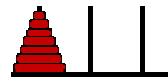
...

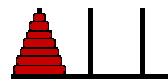


2 đã đặt

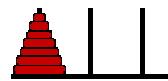
Cấu trúc dữ liệu và thuật toán - NGUYỄN ĐỨC NGHĨA, Bộ môn KHMT, ĐHBK Hà Nội

Mét lēi gi¶i cña b¶i to ,n xÔp hËu khi n = 8





1.9.1. Chứng minh bằng qui nạp toán học



- Đây là kỹ thuật chứng minh rất hữu ích khi ta phải chứng minh mệnh đề $P(n)$ là đúng với mọi số tự nhiên $n \geq n_0$.
- Tương tự như nguyên lý “hiệu ứng domino”.
- Sơ đồ chứng minh:

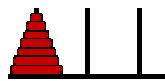
$P(n_0)$

$\forall n \geq n_0 (P(n) \rightarrow P(n+1))$

Kết luận: $\forall n \geq n_0 P(n)$

*“Nguyên lý qui nạp toán học
thứ nhất”*

*“The First Principle
of Mathematical Induction”*

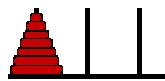
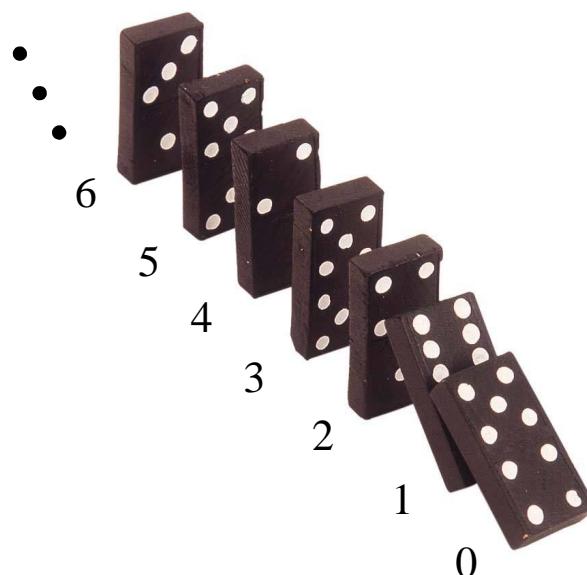


The “Domino Effect”

- **Bước #1:** Domino #0 đổ.
- **Bước #2:** Với mọi $n \in \mathbb{N}$, nếu domino # n đổ, thì domino # $n+1$ cũng đổ.
- **Kết luận:** Tất cả các quân bài domino đều đổ!



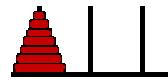
Chú ý:
điều này xảy ra
ngay cả khi
có vô hạn
quân bài domino!



Tính đúng đắn của qui nạp (The Well-Ordering Property)

- Tính đúng đắn của chứng minh qui nạp là hệ quả của “well-ordering property”: Mỗi tập con khác rỗng các số nguyên không âm đều có phần tử nhỏ nhất”.
 - $\forall \emptyset \neq S \subseteq \mathbb{N} : \exists m \in S : \forall n \in S : m \leq n$
- **Chứng minh tính đúng đắn của nguyên lý qui nạp.**
- Giả sử $P(n)$ không đúng với mọi n . Khi đó từ WOP suy ra tập $\{n | \neg P(n)\}$ có phần tử nhỏ nhất m . Ta có $P(m-1)$ là đúng, theo chứng minh qui nạp suy ra $P((m-1)+1) = P(m)$ là đúng?!

Sơ đồ chứng minh bằng qui nạp yếu

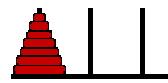


Giả sử ta cần chứng minh $P(n)$ là đúng $\forall n \geq m$.

- **Cơ sở qui nạp:** Chứng minh $P(m)$ là đúng.
- **Giả thiết qui nạp:** Giả sử $P(n)$ là đúng
- **Bước chuyển qui nạp:** Chứng minh $P(n+1)$ là đúng.
- **Kết luận:** Theo nguyên lý qui nạp ta có $P(n)$ là đúng $\forall n \geq m$.

Qui nạp mạnh

(Second Principle of Induction – Strong Induction)



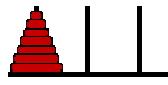
- Sơ đồ chứng minh:

$$\underbrace{P(m)}_{\forall n \geq m: (\forall m \leq k \leq n P(k))} \quad P \text{ là đúng trong } mọi \text{ tình huống trước} \\ \rightarrow P(n+1)$$

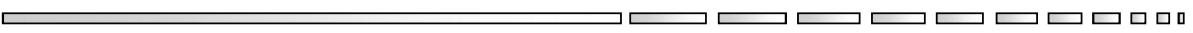
Kết luận $\forall n \geq m: P(n)$

- Sự khác biệt với sơ đồ qui nạp “yếu” ở chỗ:

- bước chuyển qui nạp sử dụng giả thiết *mạnh* hơn: $P(k)$ là đúng cho *mọi* số nhỏ hơn $m \leq k < n+1$, chứ không phải chỉ riêng với $k=n$ như trong nguyên lý qui nạp thứ nhất.



Sơ đồ chứng minh bằng qui nạp mạnh



Giả sử ta cần chứng minh $P(n)$ là đúng $\forall n \geq m$.

- **Cơ sở qui nạp:** Chứng minh $P(m)$ là đúng.
- **Giả thiết qui nạp:** Giả sử $P(k)$ là đúng $\forall m \leq k \leq n$.
- **Bước chuyển qui nạp:** Chứng minh $P(n+1)$ là đúng.
- **Kết luận:** Theo nguyên lý qui nạp ta có $P(n)$ là đúng $\forall n \geq m$.