Phần thứ nhất

LÝ THUYẾT TỔ HỢP

Combinatorial Theory

Fall 2009

Nội dung

Chương 3

BÀI TOÁN LIỆT KÊ

NỘI DUNG

1. **Giới thiệu bài toán**
2. Thuật toán và độ phức tạp
3. Phương pháp sinh
4. Thuật toán quay lui

Giới thiệu bài toán

- Bài toán đưa ra danh sách tất cả cấu hình tổ hợp thoả mãn một số tính chất cho trước được gọi là *bài toán liệt kê tổ hợp*.
- Do số lượng cấu hình tổ hợp cần liệt kê thường là rất lớn ngay cả khi kích thước cấu hình chưa lớn:
  - Số hoán vị của n phần tử là n!
  - Số tập con m phần tử của n phần tử là n!/(m!(n-m)!
- Do đó ần có quan niệm thế nào là giải bài toán liệt kê tổ hợp

Giới thiệu bài toán

- Bài toán liệt kê tổ hợp là giải được nếu như ta có thể *x*ác định một ***thuật toán*** để theo đó có thể lần lượt xây dựng được tất cả các cấu hình cần quan tâm.
- Một thuật toán liệt kê phải đảm bảo 2 yêu cầu cơ bản:
  - Không được lặp lại một cấu hình,
  - không được bỏ sót một cấu hình.

Chương 3. Bài toán liệt kê

1. Giới thiệu bài toán
2. **Thuật toán và độ phức tạp**
3. Phương pháp sinh
4. Thuật toán quay lui

Khái niệm bài toán tính toán

- **Định nghĩa.** ***Bài toán tính toán F là ánh xạ từ tập các xâu nhị phân độ dài hữu hạn vào tập các xâu nhị phân độ dài hữu hạn:***

  $$F : \{0, 1\}^* \rightarrow \{0, 1\}^*.$$

- **Ví dụ:**
- Mỗi số nguyên x đều có thể biểu diễn dưới dạng xâu nhị phân là cách viết trong hệ đếm nhị phân của nó.
- Hệ phương trình tuyến tính $Ax = b$ có thể biểu diễn dưới dạng xâu là ghép nối của các xâu biểu diễn nhị phân của các thành phần của ma trận A và vectơ b.
- Đa thức một biến:

  $$P(x) = a_0 + a_1 x + ... + a_n x^n,$$

  hoàn toàn xác định bởi dãy số $n, a_0, a_1, ..., a_n$, mà để biểu diễn dãy số này chúng ta có thể sử dụng xâu nhị phân.

Khái niệm thuật toán

- **Định nghĩa.** *Ta hiểu thuật toán giải bài toán đặt ra là một thủ tục xác định bao gồm một dãy hữu hạn các bước cần thực hiện để thu được đầu ra cho một đầu vào cho trước của bài toán.*
- Thuật toán có các đặc trưng sau đây:
  - **Đầu vào** (Input): Thuật toán nhận dữ liệu vào từ một tập nào đó.
  - **Đầu ra** (Output): Với mỗi tập các dữ liệu đầu vào, thuật toán đưa ra các dữ liệu tương ứng với lời giải của bài toán.
  - **Chính xác** (Precision): Các bước của thuật toán được mô tả chính xác.

Khái niệm thuật toán

- **Hữu hạn** (Finiteness): Thuật toán cần phải đưa được đầu ra sau một số hữu hạn (có thể rất lớn) bước với mọi đầu vào.
- **Đơn trị** (Uniqueness): Các kết quả trung gian của từng bước thực hiện thuật toán được xác định một cách đơn trị và chỉ phụ thuộc vào đầu vào và các kết quả của các bước trước.
- **Tổng quát** (Generality): Thuật toán có thể áp dụng để giải mọi bài toán có dạng đã cho.

Độ phức tạp của thuật toán

- ***Độ phức tạp tính toán*** của thuật toán được xác định như là lượng tài nguyên các loại mà thuật toán đòi hỏi sử dụng.
- Có hai loại tài nguyên quan trọng đó là thời gian và bộ nhớ.
- Việc tính chính xác được các loại tài nguyên mà thuật toán đòi hỏi là rất khó. Vì thế ta quan tâm đến việc đưa ra các đánh giá sát thực cho các đại lượng này.
- Trong giáo trình này ta đặc biệt quan tâm đến đánh giá thời gian cần thiết để thực hiện thuật toán mà ta sẽ gọi là ***thời gian tính*** của thuật toán.

Độ phức tạp của thuật toán

- Rõ ràng: Thời gian tính phụ thuộc vào dữ liệu vào.
- Ví dụ: Việc nhân hai số nguyên có 3 chữ số đòi hỏi thời gian khác hẳn so với việc nhân hai số nguyên có $3*10^9$ chữ số!
- **Định nghĩa.** *Ta gọi* ***kích thước dữ liệu đầu vào*** (*hay độ dài dữ liệu vào*) *là số bít cần thiết để biểu diễn nó.*
- Ví dụ: Nếu x, y là đầu vào cho bài toán nhân 2 số nguyên, thì kích thước dữ liệu vào của bài toán là $n = \lceil \log |x| \rceil + \lceil \log |y| \rceil$.
- Ta sẽ tìm cách đánh giá thời gian tính của thuật toán bởi một hàm của độ dài dữ liệu vào.

Phép toán cơ bản

- Đo thời gian tính bằng đơn vị đo nào?
- **Định nghĩa.** *Ta gọi **phép toán cơ bản** là phép toán có thể thực hiện với thời gian bị chặn bởi một hằng số không phụ thuộc vào kích thước dữ liệu.*
- Để tính toán thời gian tính của thuật toán ta sẽ đếm ***số phép toán cơ bản mà nó phải thực hiện.***

Các loại thời gian tính

- Chúng ta sẽ quan tâm đến:
  - Thời gian tối thiểu cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích thước *n*. Thời gian như vậy sẽ được gọi là ***thời gian tính tốt nhất*** của thuật toán với đầu vào kích thước *n*.
  - Thời gian nhiều nhất cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích thước *n*. Thời gian như vậy sẽ được gọi là ***thời gian tính tồi nhất*** của thuật toán với đầu vào kích thước *n*.
  - Thời gian trung bình cần thiết để thực hiện thuật toán trên tập hữu hạn các đầu vào kích thước *n*. Thời gian như vậy sẽ được gọi là ***thời gian tính trung bình*** của thuật toán.

Ký hiệu tiệm cận

Asymptotic Notation

- Θ, O, Ω
- Được xác định đối với các hàm nhận giá trị nguyên không âm
- Dùng để so sánh tốc độ tăng của hai hàm
- Được sử dụng để mô tả thời gian tính của thuật toán
- Thay vì nói chính xác, ta có thể nói thời gian tính là, chẳng hạn, $\Theta(n^2)$

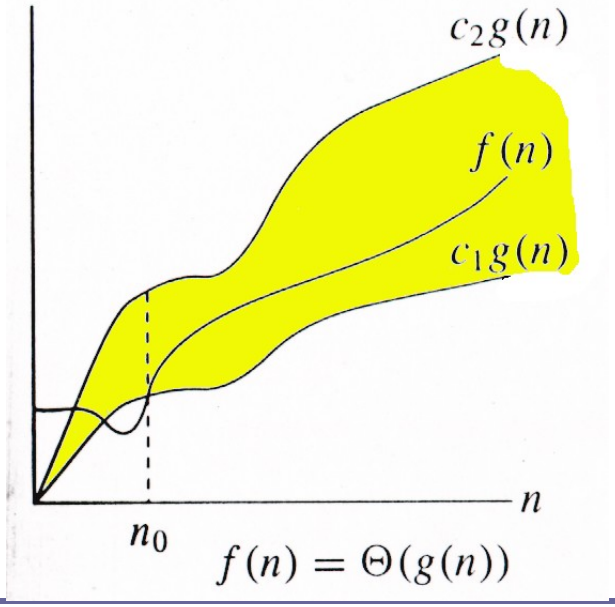
Ký hiệu Θ

Đối với hàm $g(n)$ cho trước, ta ký hiệu $\Theta(g(n))$ là tập các hàm

$\Theta(g(n)) = \{f(n) \mid$ tồn tại các hằng số c_1, c_2 và n_0 sao cho

$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$, với mọi $n \ge n_0\}$

Ta nói rằng $g(n)$ là ***đánh giá tiệm cận đúng*** cho $f(n)$



Ví dụ

- $10n^2 - 3n = \Theta(n^2)$
- Với giá trị nào của các hằng số n_0, c_1, và c_2 thì bất đẳng thức sau đây là đúng với $n \geq n_0$:

$$c_1 n^2 \leq 10n^2 - 3n \leq c_2 n^2$$

- Ta có thể lấy c_1 bé hơn hệ số của số hạng với số mũ cao nhất, còn c_2 lấy lớn hơn hệ số này, chẳng hạn:

$$c_1 = 9 < 10 < c_2 = 11,\ n_0 = 10.$$

- *Tổng quát, để so sánh tốc độ tăng của các đa thức, cần nhìn vào số hạng với số mũ cao nhất*

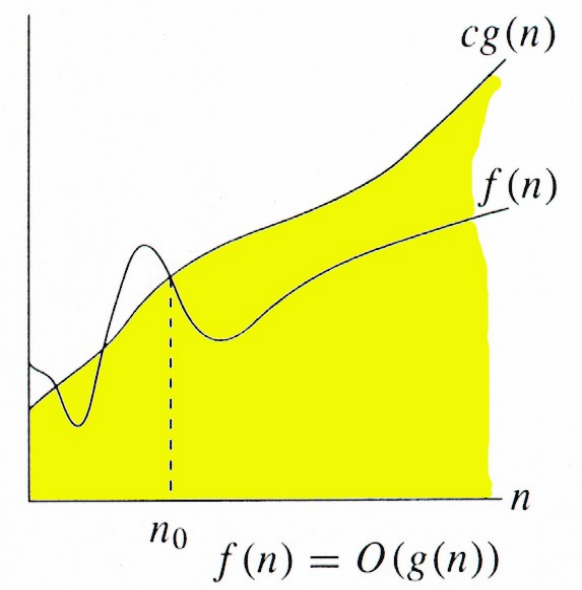
Ký hiệu *O*

Đối với hàm $g(n)$ cho trước, ta ký hiệu $O(g(n))$ là tập các hàm

$O(g(n)) = \{f(n) \mid$ tồn tại các hằng số dương c và n_0 sao cho:

$f(n) \leq cg(n)$ với mọi $n \geq n_0 \}$

Ta nói $g(n)$ là *cận trên tiệm cận* của $f(n)$



Ví dụ: Ký hiệu O lớn

- **Chứng minh:** $f(n) = n^2 + 2n + 1$ là $O(n^2)$
  - Cần chỉ ra: $n^2 + 2n + 1 \leq c*n^2$

    với c là hằng số nào đó và khi $n > n_0$ nào đó
- Ta có:

  $2n^2 \geq 2n$ khi $n \geq 1$

  và $n^2 \geq 1$ khi $n \geq 1$
- Vì vậy

  $n^2 + 2n + 1 \leq 4*n^2$ với mọi $n \geq 1$
- Như vậy hằng số $c = 4$, và $n_0=1$

Ví dụ: Ký hiệu O lớn

- Rõ ràng: Nếu $f(n)$ là $O(n^2)$ thì nó cũng là $O(n^k)$ với $k > 2$.
- **Chứng minh:** $f(n) = n^2 + 2n + 1 \notin O(n)$.
- Phản chứng. Giả sử trái lại, khi đó phải tìm được hằng số c và số n_0 để cho:

$$n^2 + 2n + 1 \leq c*n \text{ khi } n \geq n_0$$

- Suy ra

$$n^2 < n^2 + 2n + 1 \leq c*n \text{ với mọi } n \geq n_0$$

- Từ đó ta thu được:

$$n < c \text{ với mọi } n \geq n_0 \quad ?!$$

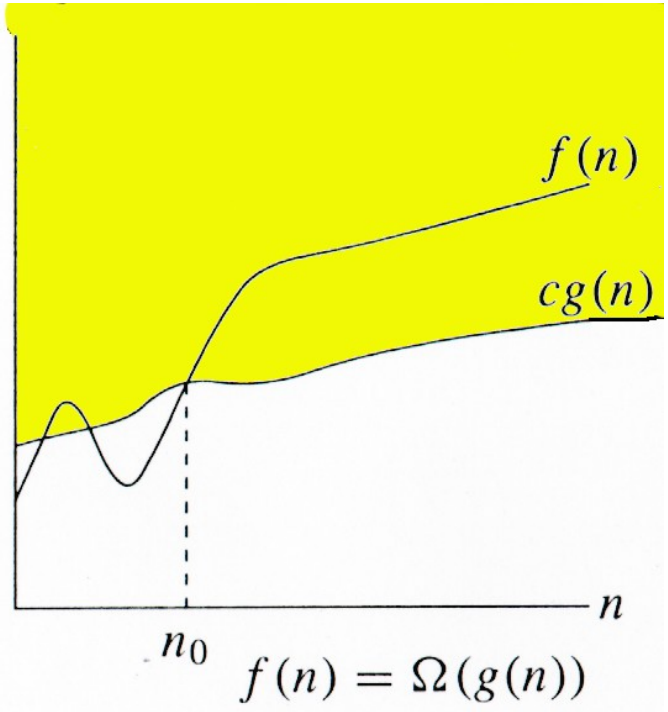
Ký hiệu Ω

Đối với hàm $g(n)$ cho trước, ta ký hiệu $\Omega(g(n))$ là tập các hàm:

$\Omega(g(n)) = \{f(n)|$ tồn tại các hằng số dương c và n_0 sao cho

$cg(n) \leq f(n)$

với mọi $n \geq n_0\}$



Ta nói $g(n)$ là *cận dưới tiệm cận* cho $f(n)$

Ví dụ: Ký hiệu Ω

- **Chứng minh:** $f(n) = n^2 - 2000n$ là $\Omega(n^2)$
  - Cần chỉ ra: $n^2 - 2000n \geq c*n^2$

    với c là hằng số nào đó và khi $n > n_0$ nào đó
- Ta có:

  $n^2 - 2000n \geq 0.5*n^2$ với mọi $n \geq 10000$

(vì $n^2 - 2000n - 0.5*n^2 = 0.5*n^2 - 2000n$

$= n(0.5*n - 2000) \geq 0$

khi $n \geq 10000$)

- Như vậy hằng số $c = 1$, và $n_0=1$

Ví dụ: Ký hiệu Ω

- Rõ ràng: Nếu $f(n)$ là $\Omega(n^2)$ thì nó cũng là $\Omega(n^k)$ với $k < 2$.
- **Chứng minh:** $f(n) = n^2 - 2000n \notin \Omega(n^3)$.
- Phản chứng. Giả sử trái lại, khi đó phải tìm được hằng số c và số n_0 để cho:

$$n^2 - 2000n \geq c*n^3 \text{ khi } n \geq n_0$$

- Suy ra

$$n^2 \geq n^2 - 2000n \geq c*n^3 \text{ khi } n \geq n_0$$

- Từ đó ta thu được:

$$1/c \geq n \text{ với mọi } n \geq n_0 \quad ?!$$

Liên hệ giữa Θ, Ω, *O*

- Đối với hai hàm bất kỳ $g(n)$ và $f(n)$,

$$f(n) = \Theta(g(n))$$

khi và chỉ khi

$$f(n) = O(g(n)) \text{ và } f(n) = \Omega(g(n)).$$

tức là

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

Chú ý

- Giá trị của n_0 và c ***không phải là duy nhất*** trong chứng minh công thức tiệm cận.
- Chứng minh rằng $100n + 5 = O(n^2)$
  - $100n + 5 \leq 100n + n = 101n \leq 101n^2$ với mọi $n \geq 5$

    $n_0 = 5$ và $c = 101$ là các hằng số cần tìm
  - $100n + 5 \leq 100n + 5n = 105n \leq 105n^2$ với mọi $n \geq 1$

    $n_0 = 1$ và $c = 105$ cũng là các hằng số cần tìm
- Chỉ cần tìm các hằng c và n_0 **nào đó** thoả mãn bất đẳng thức trong định nghĩa công thức tiệm cận.

Ký hiệu tiệm cận trong các đẳng thức

- Được sử dụng để thay thế các biểu thức chứa các toán hạng với tốc độ tăng chậm
- Ví dụ,

$$4n^3 + 3n^2 + 2n + 1 = 4n^3 + 3n^2 + \Theta(n)$$

$$= 4n^3 + \Theta(n^2) = \Theta(n^3)$$

- Trong các đẳng thức, $\Theta(f(n))$ thay thế cho một *hàm nào đó* $g(n) \in \Theta(f(n))$
  - Trong ví dụ trên, $\Theta(n^2)$ thay thế cho $3n^2 + 2n + 1$

So sánh các hàm số

$$f \leftrightarrow g \approx a \leftrightarrow b$$

$$f(n) = O(g(n)) \approx a \leq b$$

$$f(n) = \Omega(g(n)) \approx a \geq b$$

$$f(n) = \Theta(g(n)) \approx a = b$$

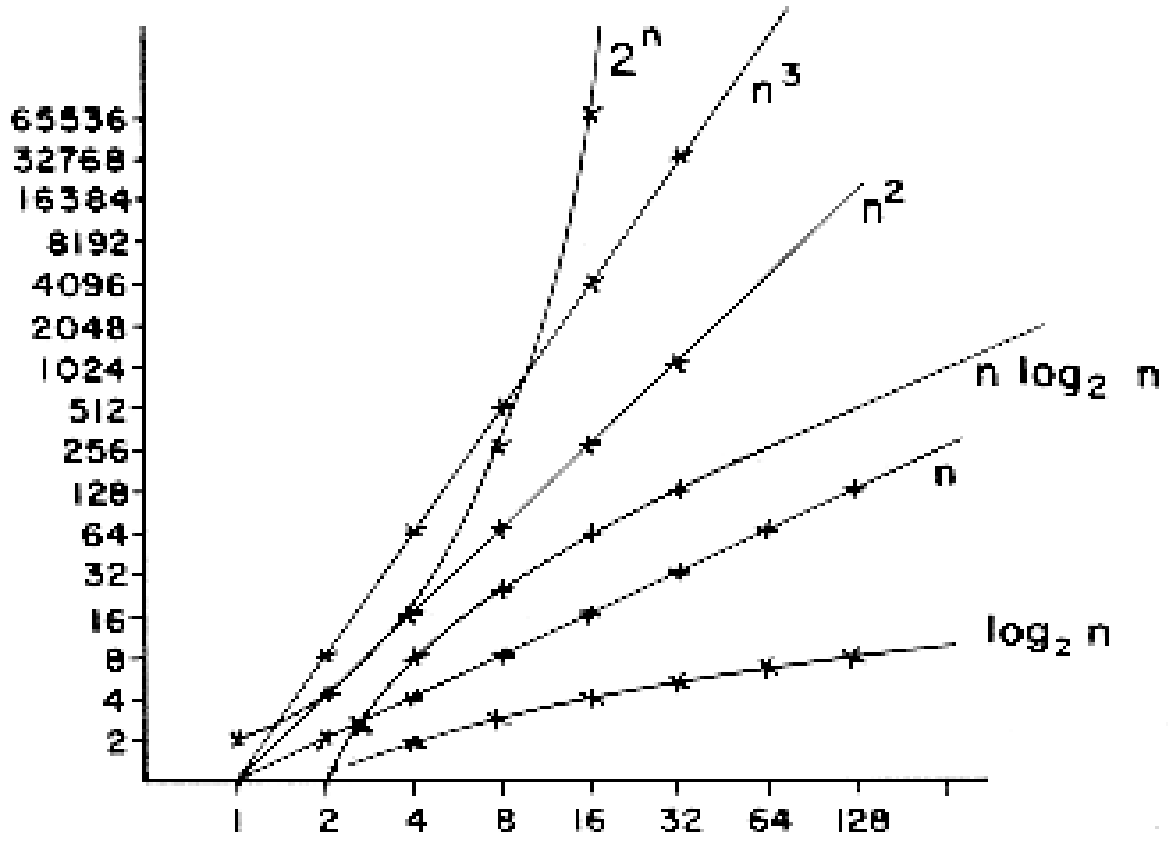
$$f(n) = o(g(n)) \approx a < b$$

$$f(n) = \omega\ (g(n)) \approx a > b$$

Cách nói về thời gian tính

- Nói "Thời gian tính là $O(f(n))$" hiểu là: Đánh giá trong tình huống tồi nhất (worst case) là $O(f(n))$. Thường nói: "Đánh giá thời gian tính trong tình huống tồi nhất là $O(f(n))$"
  - Nghĩa là thời gian tính trong tình huống tồi nhất được xác định bởi một hàm nào đó $g(n) \in O(f(n))$
- "Thời gian tính là $\Omega(f(n))$" hiểu là: Đánh giá trong tình huống tốt nhất (best case) là $\Omega(f(n))$. Thường nói: "Đánh giá thời gian tính trong tình huống tốt nhất là $\Omega(f(n))$"
  - *Nghĩa là thời gian tính trong tình huống tốt nhất được xác định bởi một hàm nào đó* $g(n) \in \Omega(f(n))$

Đồ thị của một số hàm cơ bản



Tên gọi của một số tốc độ tăng

Hàm	Tên gọi
$\log n$	log
n	tuyến tính
$n \log n$	n log n
n^2	bình phương
n^3	bậc 3
2^n	hàm mũ
n^k (k là hằng số dương)	đa thức

Ví dụ phân tích thuật toán

Ví dụ. Xét thuật toán tìm kiếm tuần tự để giải bài toán

Đầu vào: n và dãy số $s_1, s_2, \ldots, s_n$.

Đầu ra: Vị trí phần tử có giá trị *key* hoặc là n+1 nếu không tìm thấy.

```
function  Linear_Search(s,n,key);
begin
        i:=0;
        repeat
                i:=i+1;
        until  (i>n) or (key = si);
        Linear_Search := i;
end;
```

Ví dụ phân tích thuật toán

- Cần đánh giá thời gian tính tốt nhất, tồi nhất, trung bình của thuật toán với độ dài đầu vào là n. Rõ ràng thời gian tính của thuật toán có thể đánh giá bởi số lần thực hiện câu lệnh $i:=i+1$ trong vòng lặp **repeat**.
- Nếu $s_1 = key$ thì câu lệnh $i:=i+1$ trong thân vòng lặp repeat thực hiện 1 lần. Do đó thời gian tính tốt nhất của thuật toán là $\Theta(1)$.
- Nếu *key* không có mặt trong dãy đã cho, thì câu lệnh $i:= i+1$ thực hiện n lần. Vì thế thời gian tính tồi nhất của thuật toán là $O(n)$.

Ví dụ phân tích thuật toán

- Cuối cùng, ta tính thời gian tính trung bình của thuật toán.
  - Nếu *key* tìm thấy ở vị trí thứ i của dãy ($key = s_i$) thì câu lệnh $i := i+1$ phải thực hiện i lần ($i = 1, 2, ..., n$),
  - Nếu *key* không có mặt trong dãy đã cho thì câu lệnh $i := i+1$ phải thực hiện n lần.
- Từ đó suy ra số lần trung bình phải thực hiện câu lệnh $i := i+1$ là

$$[(1 + 2 + \ldots + n) + n] /(n+1) = [n(n+1)/2 + n]/(n+1).$$

- Ta có

$$n/4 \leq [n(n+1)/2 + n]/(n+1) \leq n. \text{ với mọi } n \geq 1.$$

- Vậy thời gian tính trung bình của thuật toán là $\Theta(n)$.

Chương 3. Bài toán liệt kê

1. Giới thiệu bài toán
2. Thuật toán và độ phức tạp
3. **Phương pháp sinh**
4. Thuật toán quay lui

3. PHƯƠNG PHÁP SINH

3.1. Sơ đồ thuật toán

3.2. Sinh các cấu hình tổ hợp cơ bản

- Sinh xâu nhị phân độ dài *n*
- Sinh tập con *m* phần tử của tập *n* phần tử
- Sinh hoán vị của *n* phần tử

SƠ ĐỒ THUẬT TOÁN

- Phương pháp sinh có thể áp dụng để giải bài toán liệt kê tổ hợp đặt ra nếu như hai điều kiện sau được thực hiện:

  1) *Có thể xác định được một thứ tự trên tập các cấu hình tổ hợp cần liệt kê. Từ đó có thể xác định được cấu hình đầu tiên và cấu hình cuối cùng trong thứ tự đã xác định.*

  2) *Xây dựng được thuật toán từ cấu hình chưa phải là cuối cùng đang có, đưa ra cấu hình kế tiếp nó.*

- Thuật toán nói đến trong điều kiện 2) được gọi là Thuật toán *Sinh kế tiếp*

Thuật toán sinh

```
procedure Generate;
Begin
    <Xây dựng cấu hình đầu tiên>;
    Stop:=false;
    while not stop do
    begin
    <Đưa ra cấu hình đang có>;
    if (cấu hình đang có chưa là cuối cùng)
        then   <Sinh_kế_tiếp
        else Stop:= true;
    end;
 End.
```

Giải thích

- Sinh_kế_tiếp là thủ tục thực hiện thuật toán sinh kế tiếp đã xây dựng trong điều kiện 2). Thủ tục này sẽ xây dựng cấu hình kế tiếp của cấu hình đang có trong thứ tự đã xác định.

- **Chú ý:** Do tập các cấu hình tổ hợp cần liệt kê là hữu hạn nên luôn có thể xác định được thứ tự trên nó. Tuy nhiên, thứ tự cần xác định sao cho có thể xây dựng được thuật toán Sinh kế tiếp.

3. PHƯƠNG PHÁP SINH

3.1. Sơ đồ thuật toán

3.2. Sinh các cấu hình tổ hợp cơ bản

- Sinh xâu nhị phân độ dài *n*
- Sinh tập con *m* phần tử của tập *n* phần tử
- Sinh hoán vị của *n* phần tử

Sinh các dãy nhị phân độ dài *n*

Sinh các dãy nhị phân độ dài n

- **Bài toán:** Liệt kê tất cả các dãy nhị phân độ dài n: $b_1\ b_2\ ...\ b_n$, trong đó $b_i \in \{0, 1\}$.

- Thứ tự tự nhiên:
- Xem mỗi dãy nhị phân $b = b_1\ b_2\ ...\ b_n$ là biểu diễn nhị phân của một số nguyên $p(b)$
- Ta nói dãy nhị phân $b = b_1\ b_2\ ...\ b_n$ ***đi trước*** dãy nhị phân $b' = b'_1\ b'_2\ ...\ b'_n$ trong thứ tự tự nhiên và ký hiệu là $b \prec b'$ nếu $p(b) < p(b')$.

41

Ví dụ

- **Ví dụ:** Khi n=3, các dãy nhị phân độ dài 3 được liệt kê theo thứ tự tự nhiên trong bảng bờn

- Dễ thấy thứ tự này trùng với thứ tự từ điển

b	$p(b)$
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Thuật toán sinh kế tiếp

- Dãy đầu tiên sẽ là 0 0 ... 0,
- Dãy cuối cùng là 1 1 ... 1.
- Giả sử $b_1\ b_2 \ldots b_n$ *là dãy đang có.*
- Nếu dãy này gồm toàn số 1, kết thúc,
- Trái lại, dãy kế tiếp nhận được bằng cách cộng thêm 1 (theo modun 2, có nhớ) vào dãy hiện tại.
- Từ đó ta có qui tắc sinh dãy kế tiếp như sau:
  - Tìm i đầu tiên (theo thứ tự $i=n, n\text{-}1, \ldots, 1$) thoả mãn $b_i = 0$.
  - Gán lại $b_i = 1$ và $b_j = 0$ với tất cả $j > i$. Dãy mới thu được sẽ là dãy cần tìm.

Ví dụ

- Xét dãy nhị phân độ dài 10: $b = 1101011111$.
- Ta có $i = 5$.
- Do đó, đặt $b_5 = 1$, và $b_i = 0$, $i = 6, 7, 8, 9, 10$, ta thu đươc xâu nhị phân kế tiếp là 1101100000.

```
1101011111
          1
----------
1101100000
```

Thuật toán sinh xâu kế tiếp

```
procedure  Next_Bit_String;
(*    Sinh xâu nhị phân kế tiếp theo thứ tự từ điển của
      xâu đang có   b1 b2 ... bn ≠ 1 1 ... 1          *)
begin
    i:=n;
    while  bi = 1 do
    begin
            bi = 0;
            i:=i-1;
    end;
    bi := 1;
end;
```

Sinh các tập con *m* phần tử của tập *n* phần tử

Sinh các tập con m phần tử của tập n phần tử

- Bài toán đặt ra là: Cho $X = \{1, 2, \ldots, n\}$. Hãy liệt kê các tập con m phần tử của X.
- Mỗi tập con m phần tử của X có thể biểu diễn bởi bộ có thứ tự gồm m thành phần

$$a = (a_1, a_2, \ldots, a_m)$$

thoả mãn

$$1 \le a_1 < a_2 < \ldots < a_m \le n.$$

Thứ tự từ điển

- Ta nói tập con $a = (a_1, a_2, \ldots, a_m)$ đi trớc tập con $a' = (a'_1, a'_2, \ldots, a'_m)$ trong thứ tự từ điển và ký hiệu là $a \prec a'$, nếu tìm đợc chỉ số k ($1 \leq k \leq m$) sao cho

$$a_1 = a'_1, a_2 = a'_2, \ldots, a_{k-1} = a'_{k-1},$$

$$a_k < a'_k .$$

Ví dụ

- Các tập con 3 phần tử của $X = \{1, 2, 3, 4, 5\}$ được liệt kê theo thứ tự từ điển như sau

1	2	3
1	2	4
1	2	5
1	3	4
1	3	5
1	4	5
2	3	4
2	3	5
2	4	5
3	4	5

Thuật toán sinh kế tiếp

- Tập con đầu tiên là (1, 2, ... , m)
- Tập con cuối cùng là (n-m+1, n-m+2, ..., n).

- Giả sử a=(a_1, a_2, ... , a_m) là tập con đang có chưa phải cuối cùng, khi đó tập con kế tiếp trong thứ tự từ điển có thể xây dựng bằng cách thực hiện các quy tắc biến đổi sau đối với tập đang có:
  - Tìm từ bên phải dãy a_1, a_2,..., a_m phần tử $a_i \neq n$-m+i,
  - Thay a_i bởi a_i + 1;
  - Thay a_j bởi $a_i + j - i$, với $j = i$+1, i+2,..., m.

Ví dụ

- ***Ví dụ:*** $n = 6, m = 4$.

  Giả sử đang có tập con (1, 2, 5, 6), cần xây dựng tập con kế tiếp nó trong thứ tự từ điển.

- Ta có i=2:

  (1, **2,** 5, 6)

  (3, **4,** 5, 6)

  thay $a_2 = 3$, và $a_3 = 4$, $a_4 = 5$, ta được tập con kế tiếp (1, 3, 4, 5).

51

Sinh m-tập kế tiếp

procedure *Next_Combination;*

(Sinh m-tập con kế tiếp theo thứ tự từ điển*

của tập con $(a_1, a_2,..., a_m) \neq (n\text{-}m+1,...,n)$ **)*

begin

$i:=m;$

while $a_i = n\text{-}m+i$ ***do*** $i:=i\text{-}1;$

$a_i := a_i + 1;$

for $j:=i+1$ ***to*** m ***do*** $a_j := a_i + j - i;$

end;

52

Sinh các hoán vị của tập *n* phần tử

Sinh các hoán vị của tập n phần tử

- **Bài toán**: Cho $X = \{1, 2, \ldots, n\}$, hãy liệt kê các hoán vị từ n phần tử của X.
- Mỗi hoán vị từ n phần tử của X có thể biểu diễn bởi bộ có thứ tự gồm n thành phần

$$a = (a_1, a_2, \ldots, a_n)$$

thoả mãn

$$a_i \in X, \ i = 1, 2, \ldots, n, \ a_p \neq a_q, p \neq q.$$

Thứ tự từ điển

- Ta nói hoán vị $a = (a_1, a_2, \ldots, a_n)$ đi trước hoán vị $a' = (a'_1, a'_2, \ldots, a'_n)$ trong thứ tự từ điển và ký hiệu là $a \prec a'$, nếu tìm được chỉ số k ($1 \leq k \leq n$) sao cho:

$$a_1 = a'_1, a_2 = a'_2, \ldots, a_{k-1} = a'_{k-1},$$

$$a_k < a'_k .$$

Ví dụ

- Các hoán vị từ 3 phần tử của $X=\{1, 2, 3\}$ đợc liệt kê theo thứ tự từ điển nh sau

1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

Thuật toán sinh kế tiếp

- Hoán vị đầu tiên: $(1, 2, \ldots, n)$
- Hoán vị cuối cùng: $(n, n\text{-}1, \ldots, 1)$.
- Giả sử $a = (a_1, a_2, \ldots, a_n)$ là hoán vị chưa phải cuối cùng, khi đó hoán vị kế tiếp nó có thể xây dựng nhờ thực hiện các biến đổi sau:
  - Tìm từ phải qua trái hoán vị đang có chỉ số j đầu tiên thoả mãn $a_j < a_{j+1}$ (nói cách khác: j là chỉ số lớn nhất thoả mãn $a_j < a_{j+1}$);
  - Tìm a_k là số nhỏ nhất còn lớn hơn a_j trong các số ở bên phải a_j ;
  - Đổi chỗ a_j với a_k ;
  - Lật ngược đoạn từ a_{j+1} đến a_n .

Ví dụ

- Giả sử đang có hoán vị (3, 6, 2, 5, 4, 1), cần xây dựng hoán vị kế tiếp nó trong thứ tự từ điển.
- Ta có chỉ số $j = 3$ ($a_3 = 2 < a_4 = 5$).
- Số nhỏ nhất còn lớn hơn a_3 trong các số bên phải của a_3 là $a_5 = 4$. Đổi chỗ a_3 với a_5 ta thu được (3, 6, 4, 5, 2, 1),
- Cuối cùng, lật ngược thứ tự đoạn $a_4\ a_5\ a_6$ ta thu được hoán vị kế tiếp (3, 6, 4, 1, 2, 5).

Sinh hoán vị kế tiếp

```
procedure Next_Permutation;
(*Sinh hoán vị kế tiếp  (a1, a2, ..., an)  ≠ (n, n-1, ..., 1)    *)
begin
    (* Tìm j là chỉ số lớn nhất thoả aj < aj+1 *)
    j:=n-1;  while aj > aj+1 do j:=j-1;
    (* Tìm ak là số nhỏ nhất còn lớn hơn aj ở bên phải aj *)
    k:=n;  while aj > ak do k:=k-1;
    Swap(aj, ak); (* đổi chỗ aj với ak *)
    (* Lật ngược đoạn từ aj+1 đến an *)
    r:=n; s:=j+1;
    while r>s do begin
        Swap(ar, as); (* đổi chỗ ar với as *)
            r:=r-1; s:= s+1;
    end;
end;
```

Chương 3. Bài toán liệt kê

1. Giới thiệu bài toán
2. Thuật toán và độ phức tạp
3. **Phương pháp sinh**
4. Thuật toán quay lui

3. THUẬT TOÁN QUAY LUI
Backtracking Algorithm

NỘI DUNG

3.1. Sơ đồ thuật toán

3.2. Liệt kê các cấu hình tổ hợp cơ bản

- Liệt kê xâu nhị phân độ dài n
- Liệt kê tập con m phần tử của tập n phần tử
- Liệt kê hoán vị

3.3. Bài toán xếp hậu

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.
- ***Bài toán liệt kê (Q):*** *Cho A_1, A_2,..., A_n là các tập hữu hạn. Ký hiệu*

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

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

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

- Các phần tử của tập *D* được gọi là các ***lời giải chấp nhận được.***

Ví dụ

- Tất cả các bài toán liệt kê tổ hợp cơ bản đều có thể phát biểu dới dạng bài toán (Q)
- Bài toán liệt kê xâu nhị phân độ dài n dẫn về việc liệt kê các phần tử của tập

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

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

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

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

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

Lời giải bộ phận

- **Định nghĩa.** Ta gọi lời giải bộ phận cấp k ($0 \le k \le n$) là bộ có thứ tự gồm k thành phần

  $$(a_1, a_2, ..., a_k),$$

  trong đó $a_i \in A_i$, $i = 1, 2, ..., k$.

- Khi $k = 0$, lời giải bộ phận cấp 0 được ký hiệu là () và còn được gọi là lời giải rỗng.
- Nếu $k = n$, ta có lời giải đầy đủ hay đơn giản là một lời giải của bài toán.

Ý tưởng chung

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

Bước tổng quát

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

Xét hai tình huống

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

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

  ta thu được lời giải bộ phận cấp k:

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

- Khi đó
  - Nếu $k = n$ thì ta thu được một lời giải,
  - Nếu $k < n$, ta tiếp tục đi xây dựng thành phần thứ k+1 của lời giải.

Tình huống ngõ cụt

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

Thuật toán quay lui

```
procedure Bactrack(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 gọi để thực hiện thuật toán quay lui là: Bactrack(1)

Hai vấn đề mấu chốt

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

Chú ý

- Nếu chỉ cần tìm một lời giải thì cần tìm cách chấm dứt các thủ tục gọi đệ qui lồng nhau sinh bởi lệnh gọi Backtrack(1) sau khi ghi nhận được lời giải đầu tiên.
- Nếu kết thúc thuật toán mà ta không thu được một lời giải nào thì điều đó có nghĩa là bài toán không có lời giải.

Chú ý

- Thuật toán dễ dàng mở rộng cho bài toán liệt kê trong đó lời giải có thể mô tả như là bộ (a_1, a_2, ..., a_n,...) độ dài hữu hạn, tuy nhiên giá trị của độ dài là không biết trước và các lời giải cũng không nhất thiết phải có cùng độ dài.
- Khi đó chỉ cần sửa lại câu lệnh

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

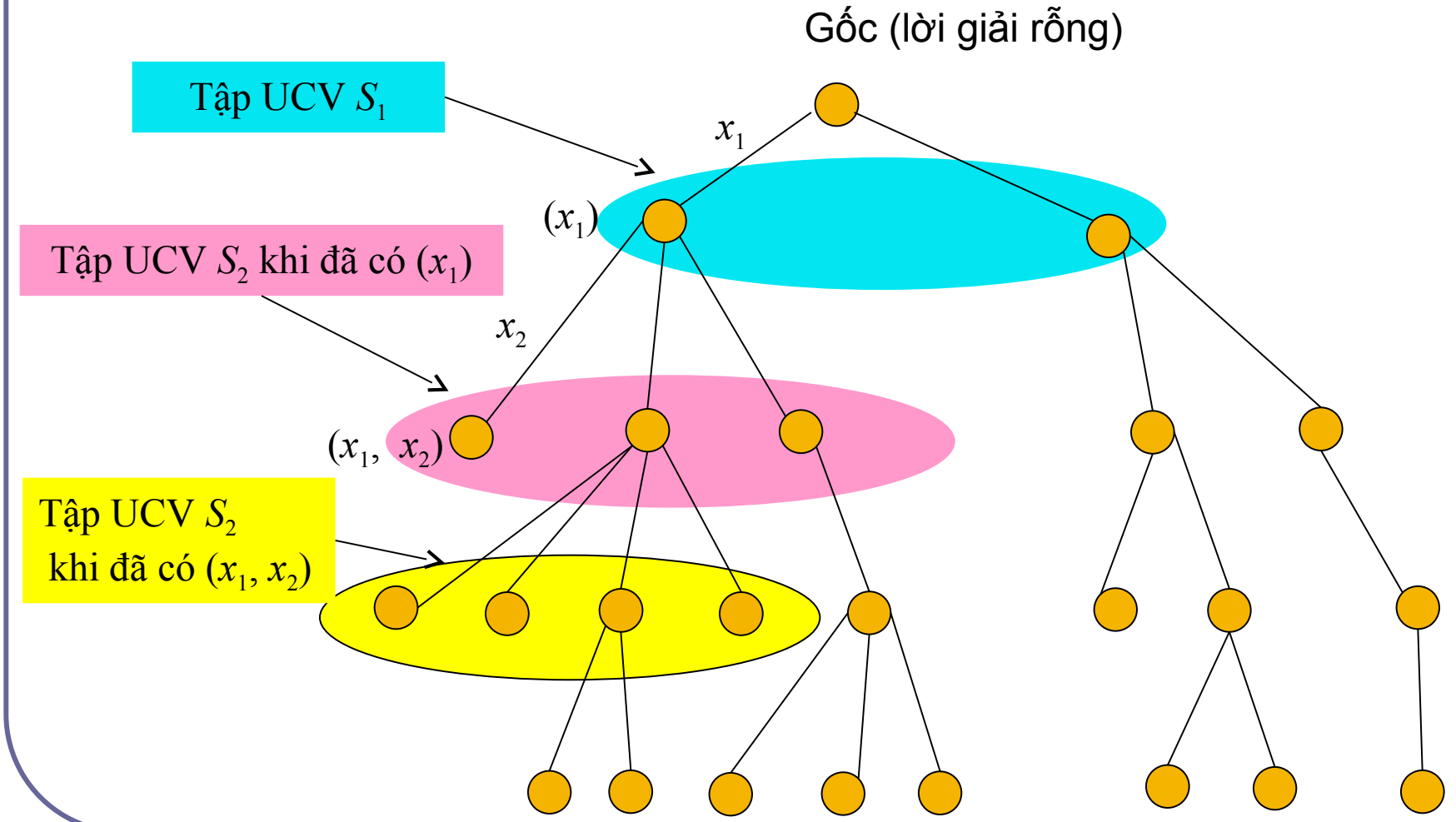
  else Backtrack(k+1);

  thành

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

  else Backtrack(k+1);
- Cần xây dựng hàm nhận biết (a_1, a_2, ..., a_k) đã là lời giải hay chưa.

Cây liệt kê lời giải theo thuật toán quay lui



Liệt kê xâu nhị phân độ dài *n*

Liệt kê xâu nhị phân độ dài *n*

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

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

- Ta xét cách giải quyết hai vấn đề cơ bản để cài đặt thuật toán quay lui:
  - Rõ ràng ta có $S_1 = \{0, 1\}$. Giả sử đã có xâu nhị phân cấp k-1 $(b_1, ..., b_{k-1})$, khi đó rõ ràng $S_k = \{0,1\}$. Nh vậy, tập các UCV vào các vị trí của lời giải đợc đã xác định.
  - Để cài đặt vòng lặp liệt kê các phần tử của S_k, dễ thấy là ta có thể sử dụng vòng lặp for

trên PASCAL: **for y:= 0 to 1 do**

hoặc trên C: **for (y=0;y<=1;y++)**

Chương trình trên Pascal

```
var           n: integer;
    b: array[1..20] of 0..1;
    count: word;

procedure Ghinhan;
var i: integer;
begin
    count := count+1;
    write(count:5, '. ');
    for i := 1 to n do write(b[i]:2);
    writeln;
end;
```

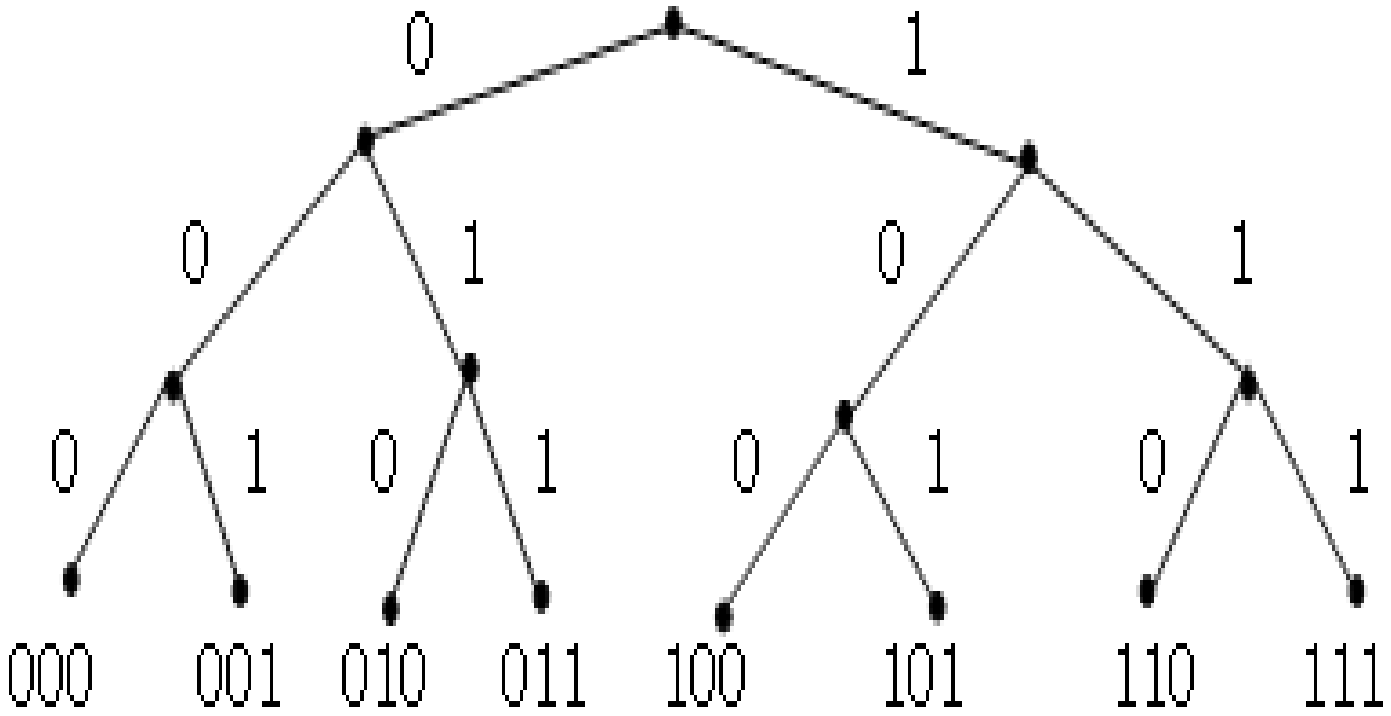
```
procedure Xau(i: integer);
var j: integer;
begin
    for j := 0 to 1 do
    begin
      b[i] := j;
      if  i = n then Ghinhan else Xau(i+1);
    end;
end;

BEGIN     {Main program}
    write('n = '); readln(n);
    count := 0; Xau(1);
    write('Gõ Enter để kết thúc... '); readln
END.
```

Chương trình trên C

```
include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int  n, count;
int b[20];

void Ghinhan() {
   int i, j;
    count++;
    printf("Xau thu %i. ",count);
    for  (i=1 ; i<= n ;i++) {
         j=b[i];
         printf("%i  ", j);
    }
    printf("\n");
}
```

```
void  Xau(int i){
   int j;
   for (j = 0; j<=1; j++) {
         b[i] = j;
         if (i==n)  Ghinhan();
         else  Xau(i+1);
    }
}

int main() {
     printf("n = ");
     scanf("%i ",&n); printf("\n");
     count = 0; Xau(1);
     printf("Count = %d ", count);
     getch();
}
```

Cây liệt kê dãy nhị phân độ dài 3



Liệt kê các *m*-tập con của *n*-tập

Liệt kê các m-tập con của n-tập

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

- Bài toán dẫn về: Liệt kê các phần tử của tập:

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

Giải quyết 2 vấn đề mấu chốt

- Từ điều kiện: $1 \le a_1 < a_2 < ... < a_m \le n$

  suy ra $S_1 = \{1, 2, ..., n-(m-1)\}$.

- Giả sử đã có tập con $(a_1, ..., a_{k-1})$.

  Từ điều kiện $a_{k-1} < a_k < ... < a_m \le n$, ta suy ra

  $S_k = \{a_{k-1}+1, a_{k-1}+2, ..., n-(m-k)\}$.

- Để cài đặt vòng lặp liệt kê các phần tử của S_k, dễ thấy là ta có thể sử dụng vòng lặp

  của PASCAL: **for** y:= a[k-1]+1 **to** n-m+k **do ...**

  của C: **for** (y=a[k-1]+1;y<=n-m+k;y++) **...**

Chương trình trên Pascal

```
var  n, m: integer;
     a: array[0..20] of byte;
     count: word;

procedure Ghinhan;
var i: integer;
begin
     count := count+1;
     write(count:5, '. ');
     for i := 1 to m do write(a[i]:4);
     writeln;
end;
```

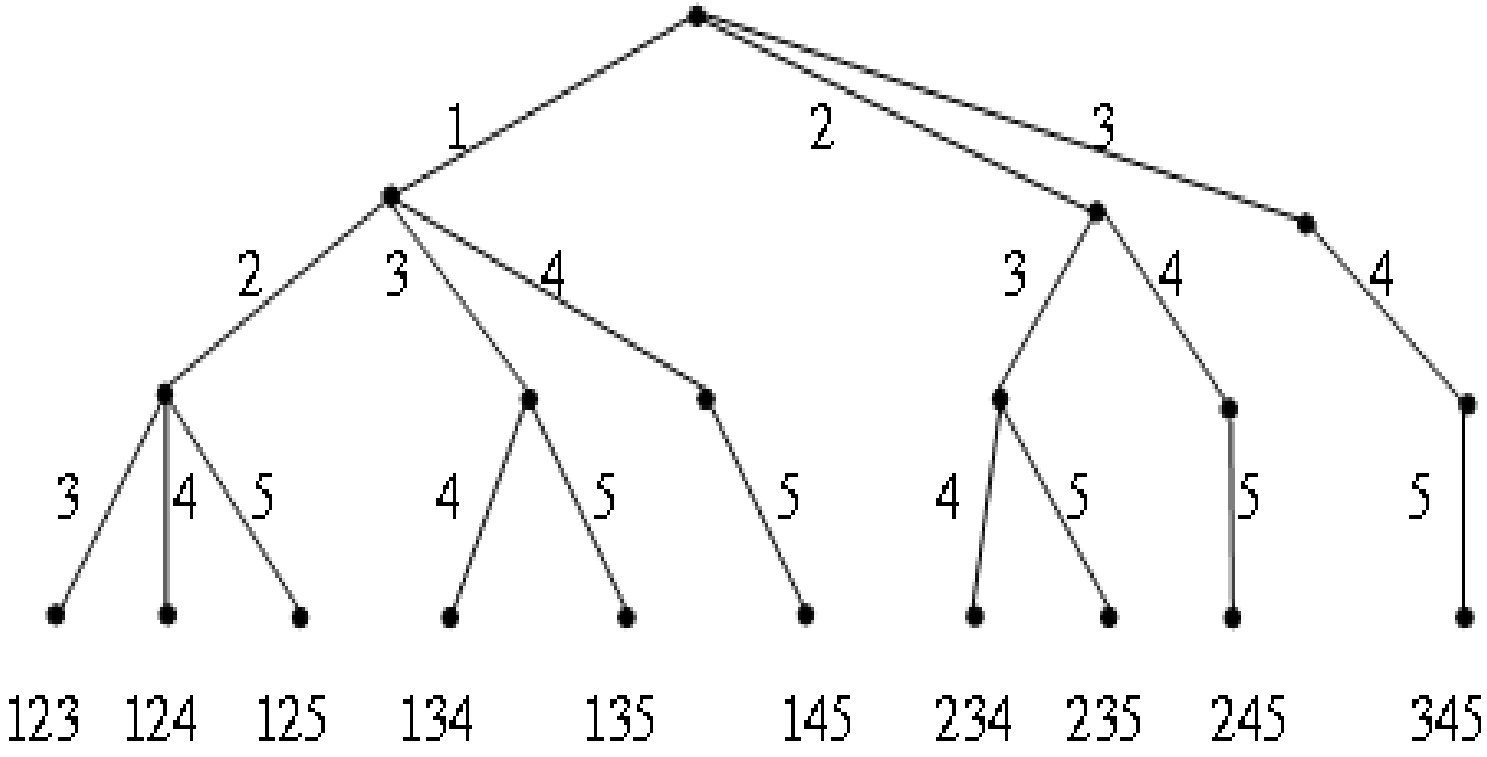
```
procedure MSet(i: integer);
var j: integer;
begin
   for j := a[i-1] +1 to n-m+i do
   begin
      a[i] := j;
      if i =m then Ghinhan else MSet(i+1);
   end;
end;

BEGIN     {Main program}
     write('n, m = '); readln(n, m);
     count := 0; MSet(1);
     write('Gõ Enter để kết thúc... '); readln
END.
```

Chương trình trên C

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int  n, m, count;
int a[20];
void Ghinhan() {
   int i, j;
    count++;
    printf("Tap con thu %i. ",count);
    for  (i=1 ; i<= m ;i++) {
         j=a[i];
         printf("%i  ", j);
    }
    printf("\n");
}
```

```
void  MSet(int i){
   int j;
   for (j = a[i-1] +1; j<= n-m+i; j++) {
         a[i] = j;
         if (i==m)  Ghinhan();
         else MSet(i+1);
    }
}
int main() {
     printf("n, m = ");
     scanf("%i %i",&n, &m); printf("\n");
     a[0]=0; count = 0; MSet(1);
     printf("Count = %d ", count);
     getch();
}
```

Cây liệt kê S(5,3)



Liệt kê hoán vị

Liệt kê hoán vị

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

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

- **Bài toán: Liệt kê tất cả các phần tử của Π_n**

Giải quyết 2 vấn đề mấu chốt

- Rõ ràng $S_1 = N$. Giả sử ta đang có hoán vị bộ phận $(a_1, a_2, ..., a_{k-1})$, từ điều kiện $a_i \neq a_j$, với mọi $i \neq j$ ta suy ra

$$S_k = N \setminus \{ a_1, a_2, ..., a_{k-1} \}.$$

- Nh vậy ta đã có cách xác định đợc tập các UCV vào các vị trí của lời giải.
- Mô tả S_k?

Mô tả S_k

Xây dựng hàm nhận biết UCV:

```
function UCV(j,k: integer): boolean;
(* UCV 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] then
       begin
            UCV:= false; exit;
       end;
   UCV:= true;
end;
```

Mô tả S_k

- Câu lệnh qui ước "**for y ∈ S_k do** " có thể cài đặt như sau

```
for y:=1 to n do
        if UCV(y,k) then
        begin
                    (*  y là UCV vào vị trí k  *)
                    . . .
            end
```

Chương trình trên Pascal

```
var
   n, count: integer;
   a: array[1..20] of integer;

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

function UCV(j,k: integer): boolean;
var i: integer;
begin
    for i:=1 to  k-1 do
       if   j = a[i]   then  begin
            UCV:= false; exit;
       end;
    UCV:= true;
 end;
```

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

BEGIN     {Main program}
   write('n = '); readln(n);
   count := 0;
   Hoanvi(1);
   write('Press Enter to finish... '); readln;
END.
```

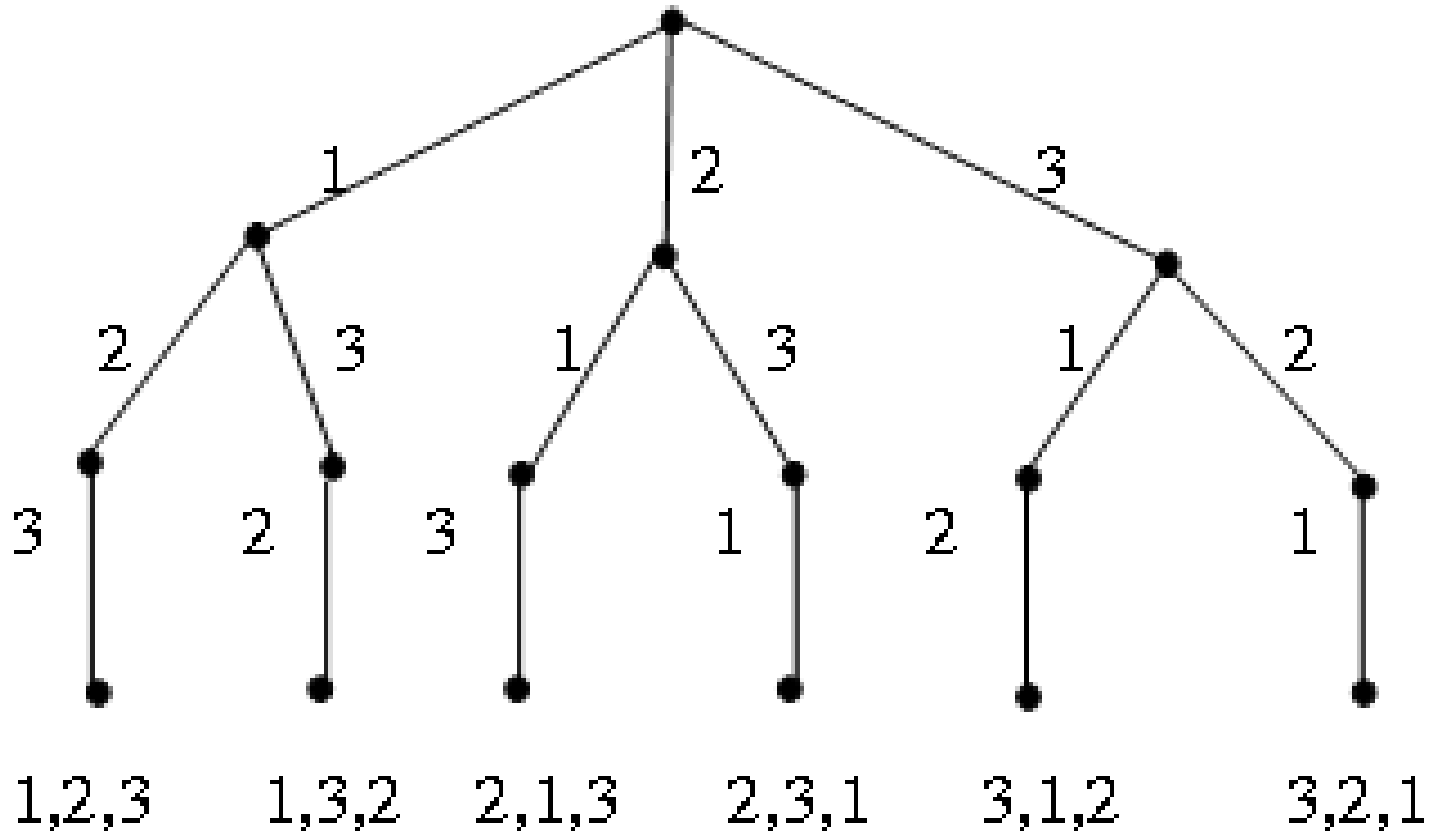
```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int  n, count;
int b[20];

int Ghinhan() {
   int i, j;
    count++;
    printf("Hoan vi thu %i. ",count);
    for  (i=1 ; i<= n ;i++) {
            printf("%i  ", b[i]);
    }
    printf("\n");
}
```

```
int UCV(int j, int k)
{
int i;
    for (i=1; i<=k-1; i++)
       if (j == b[i]) return 0;
    return 1;
}
```

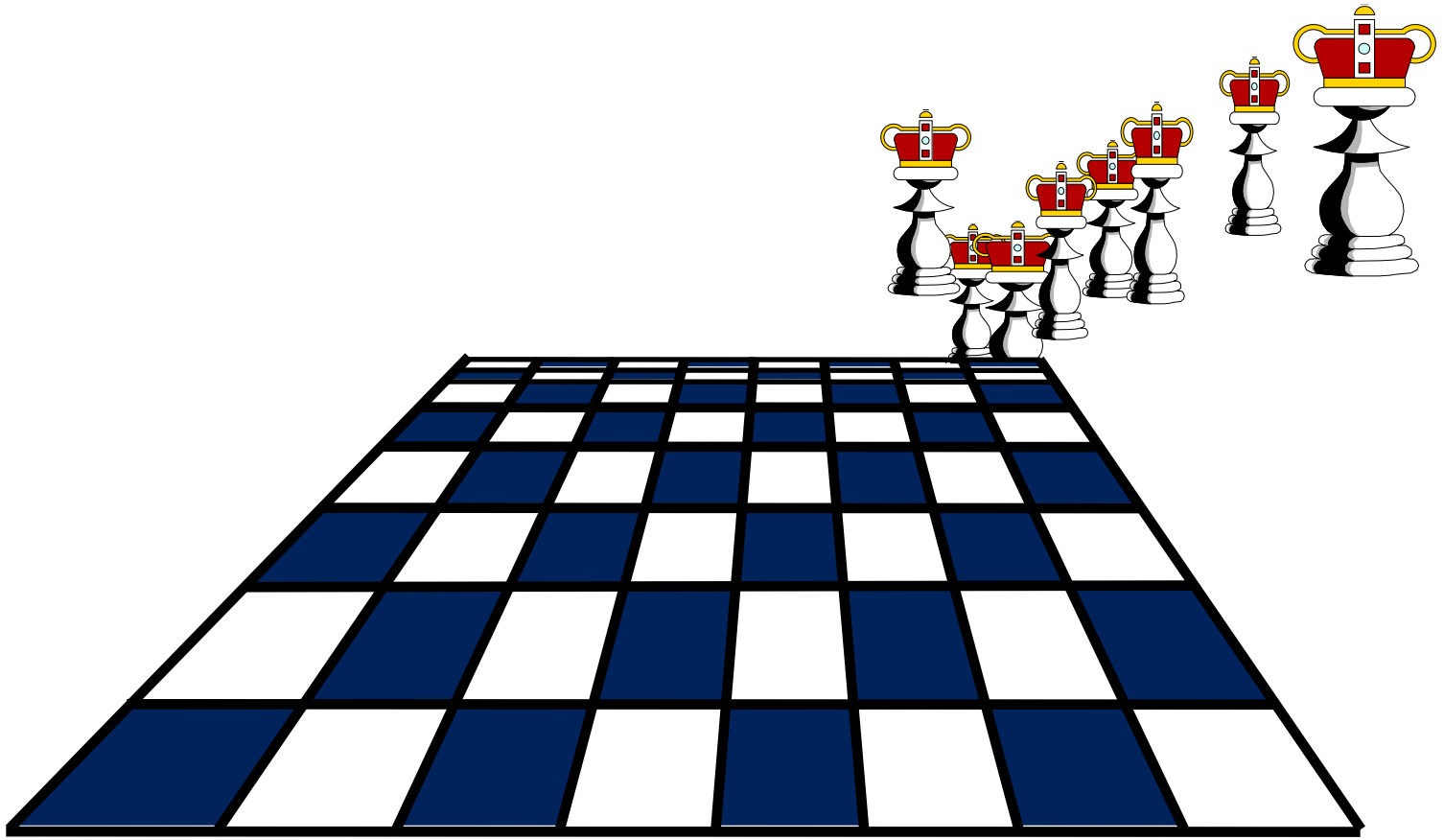
```
int  Hoanvi(int i){
   int j;
   for (j = 1; j<=n; j++)
      if (UCV(j,i)==1)
      {  b[i] = j;
         if (i==n)  Ghinhan();
         else  Hoanvi(i+1);
      }
}
```

```
int main() {
    printf(“==========\n");
    printf("n = ");
    scanf("%i",&n);
     printf("\n");
     count = 0; Hoanvi(1);
     printf("Count = %d ", count);
     getch();
}
```

Cây liệt kê hoán vị của 1, 2, 3

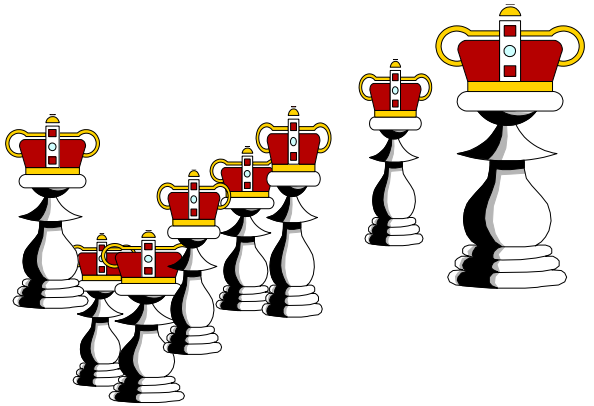


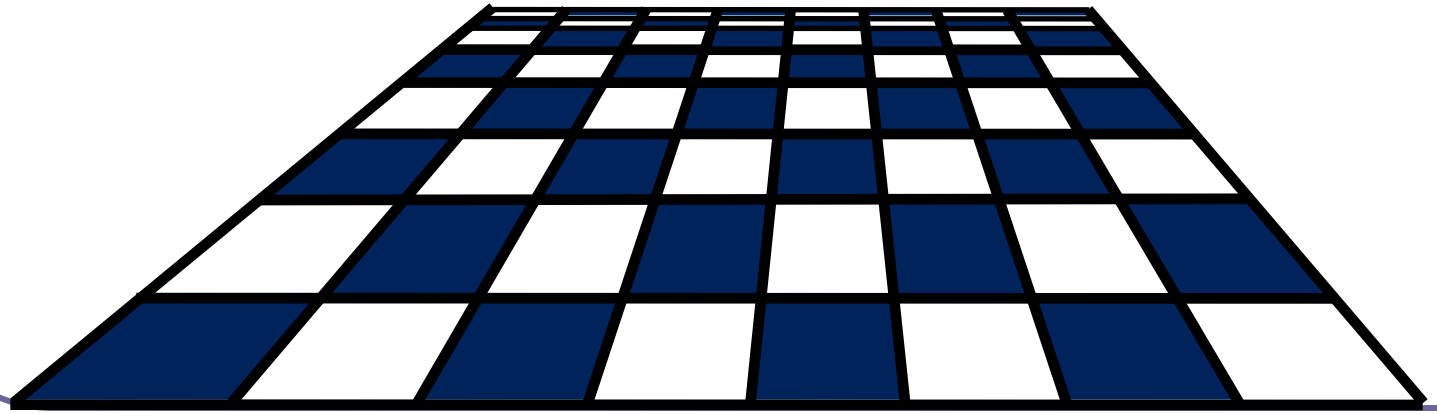
The *n*-Queens Problem



The *n*-Queens Problem

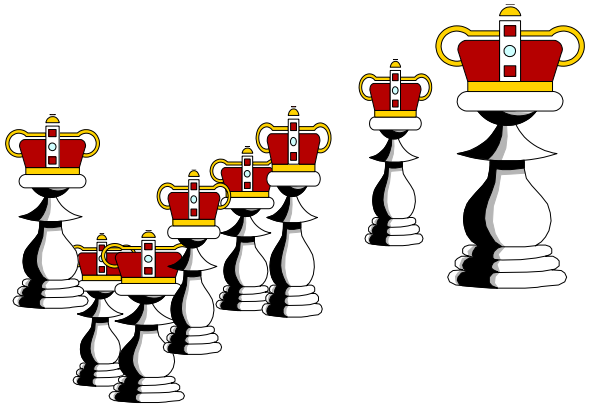
- Giả sử ta có 8 con hậu...
- ...và bàn cờ quốc tế

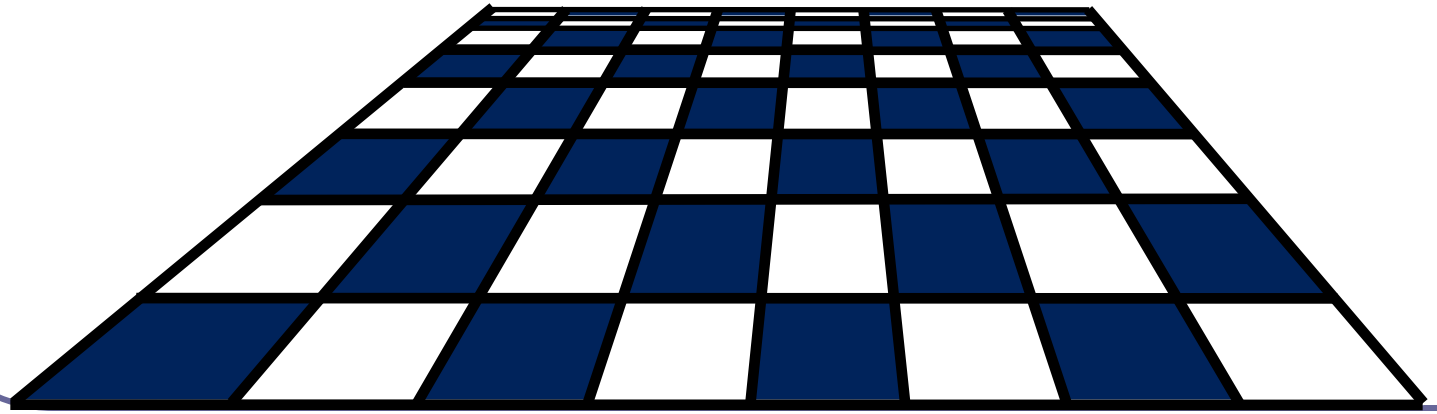




The *n*-Queens Problem

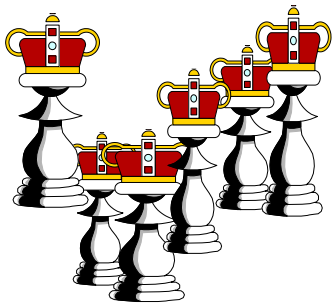
Có thể xếp các con hậu sao cho không có hai con nào ăn nhau hay không ?

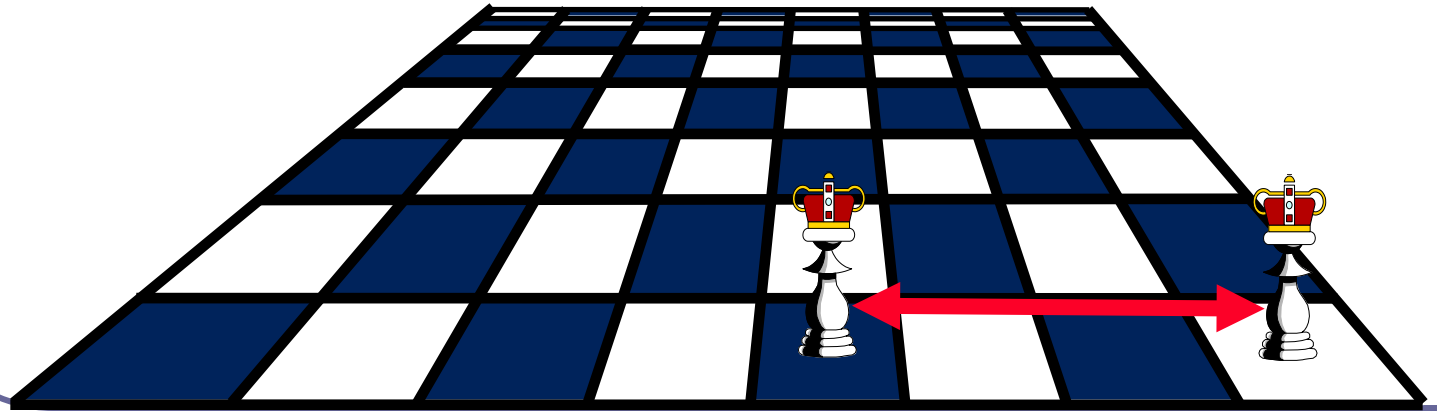




The n-Queens Problem

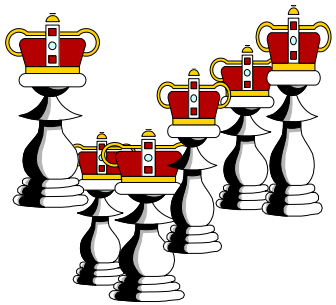
Hai con hậu bất kỳ không được xếp trên cùng một dòng ...

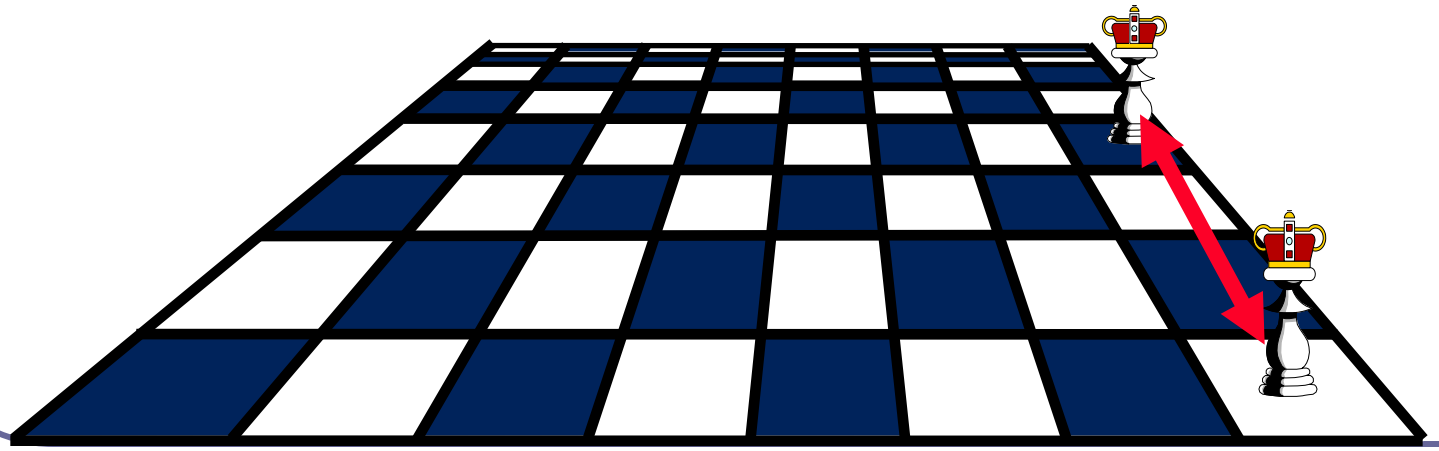




The n-Queens Problem

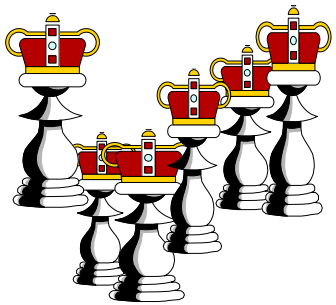
Hai con hậu bất kỳ không được xếp trên cùng một cột ...

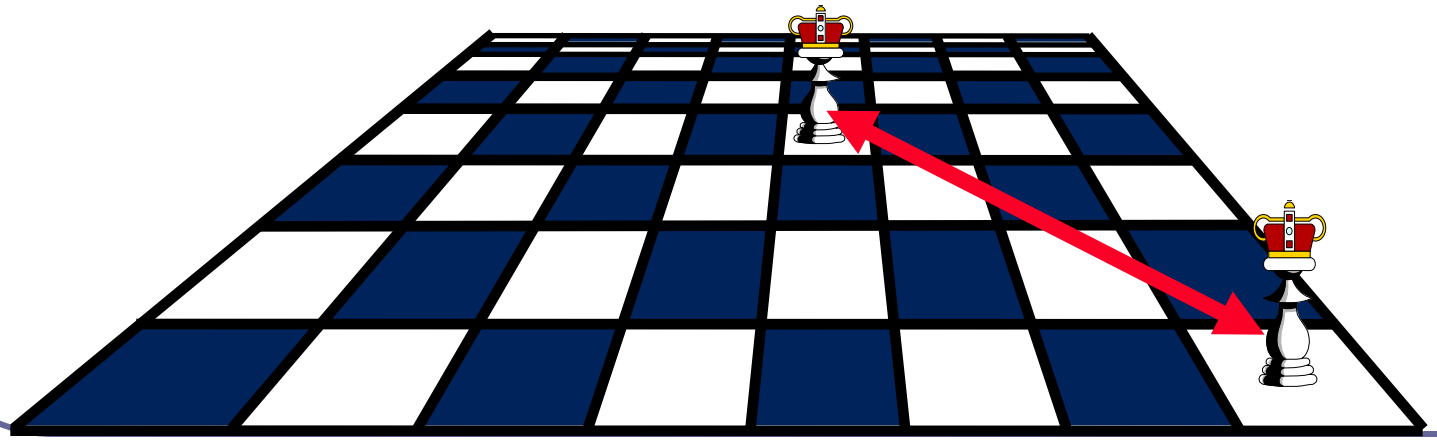




The n-Queens Problem

Hai con hậu bất kỳ
không được xếp trên
cùng một đường chéo!

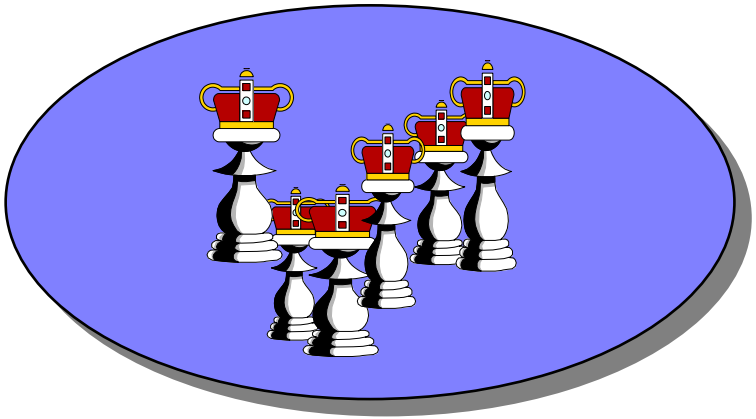


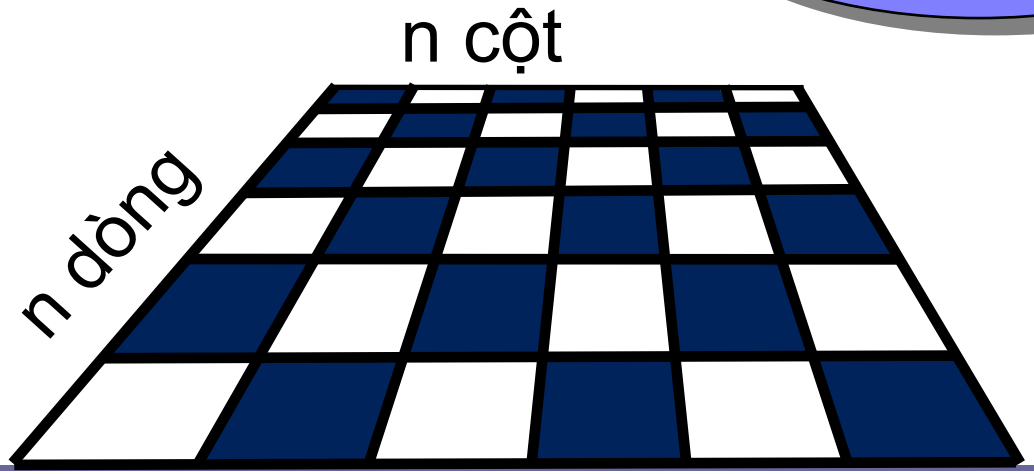


The n-Queens Problem

Kích thước n*n

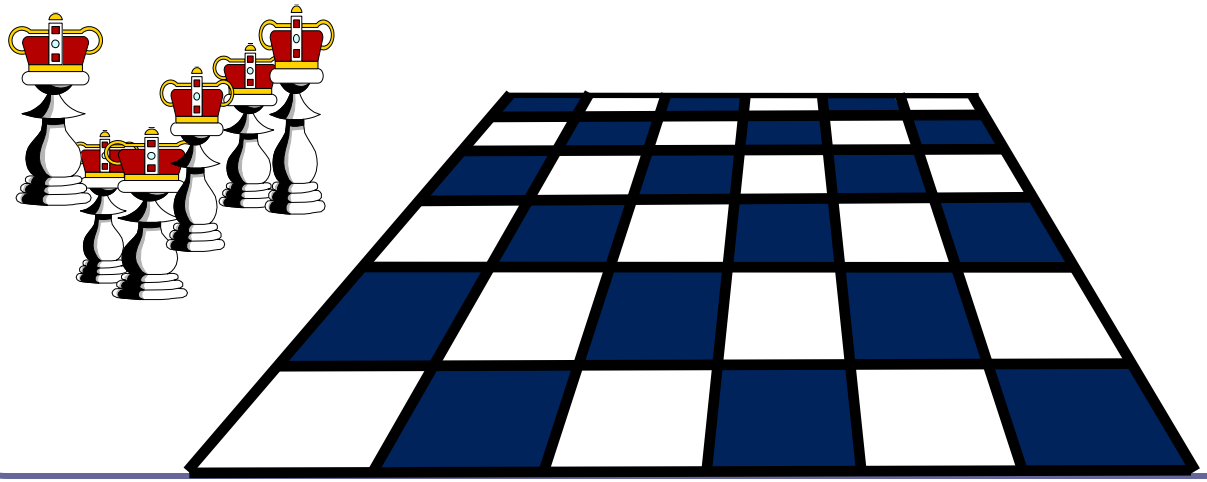
n con hậu





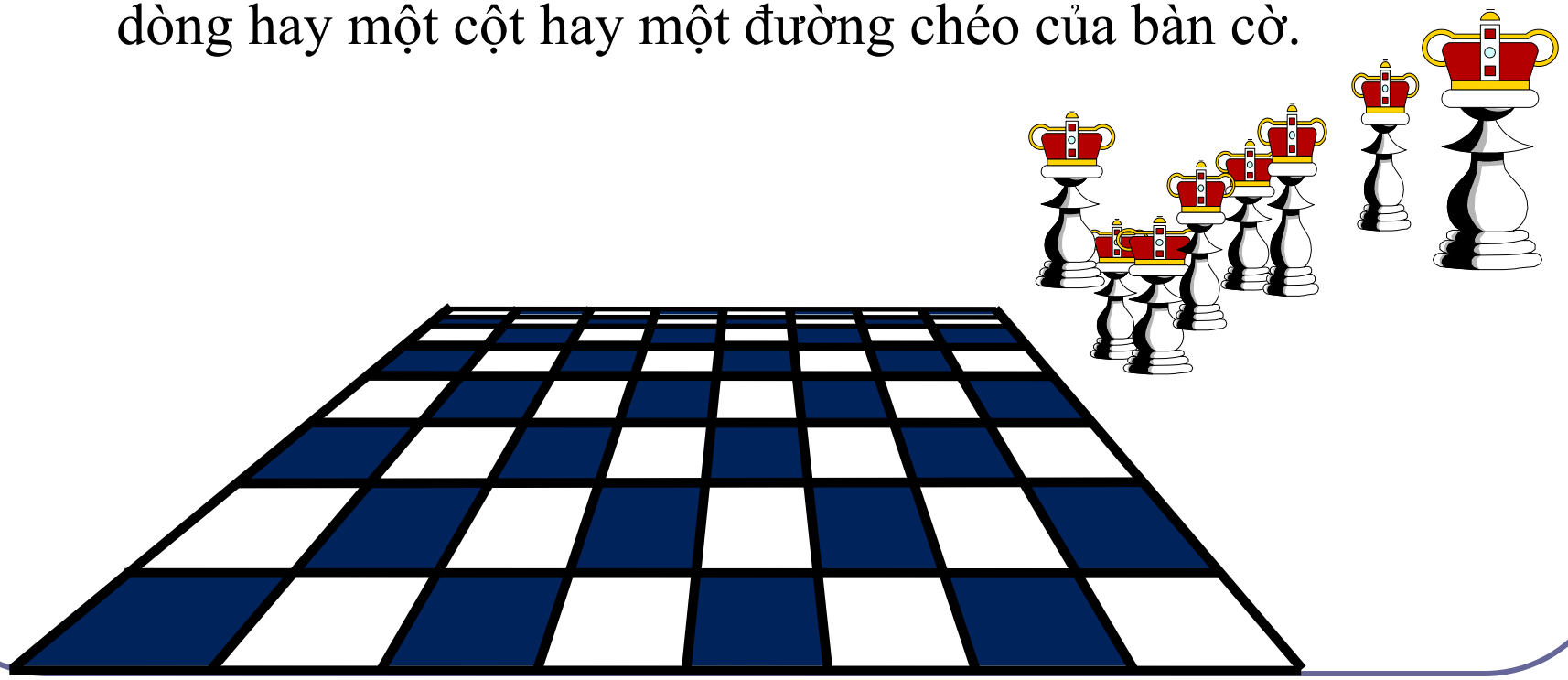
The n-Queens Problem

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



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.



Biểu diễn lời giải

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

Phát biểu bài toán

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

$$D=\{(a_1, a_2, ..., a_n)\in N^n:$$

$$a_i \neq a_j \text{ và } |a_i - a_j| \neq |i - j|, i \neq j \}.$$

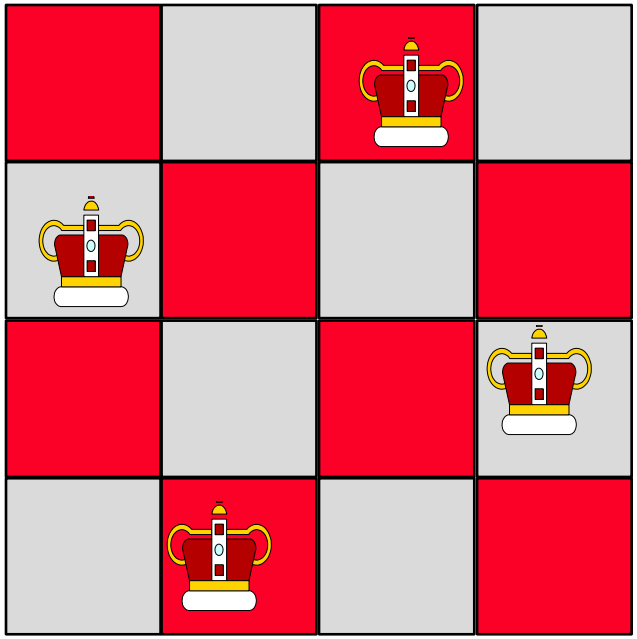
Hàm nhận biết ứng cử viên

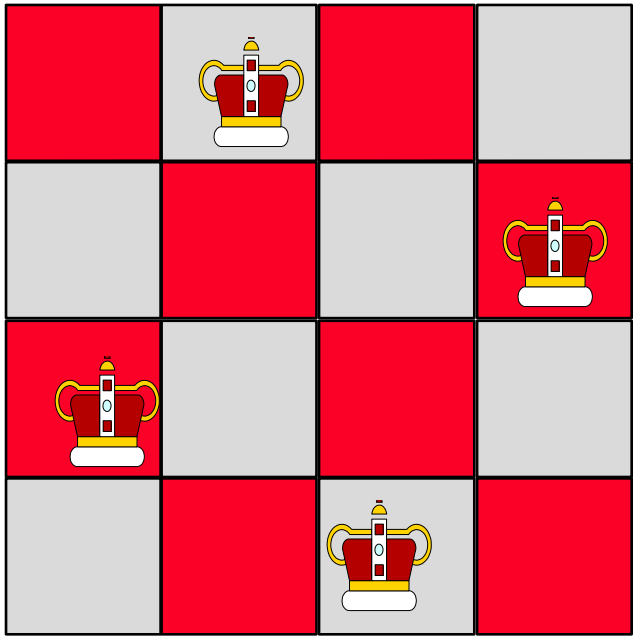
```
int UCVh(int j, int k) {
// UCVh nhận giá trị 1
// khi và chỉ khi j ∈ Sk
   int i;
   for (i=1; i<k; i++)
     if ((j == a[i])||(fabs(j-a[i])== k-i)) return 0;
   return 1;
}
```

Chương trình trên C

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

int UCVh(int j, int k) {
/* UCVh nhận giá trị 1 khi 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;
}
```

```
int Hau(int i){   int j;
   for (j=1; j<=n; j++)
         if  (UCVh(j, i)){
             a[i] = j;
             if  (i == n) Ghinhan();
             else Hau(i+1);
         }
}
int main() {
    printf("Input n = "); scanf("%i",&n);
    printf("\n==== RESULT ====\n");
    count = 0; Hau(1);
    if (count == 0) printf("No solution!\n");
    getch();
    printf("\n Press Enter to finish... ");
    getchar();
}
```

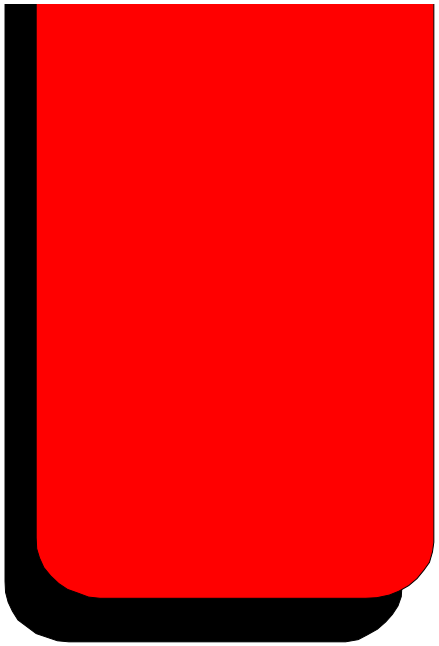


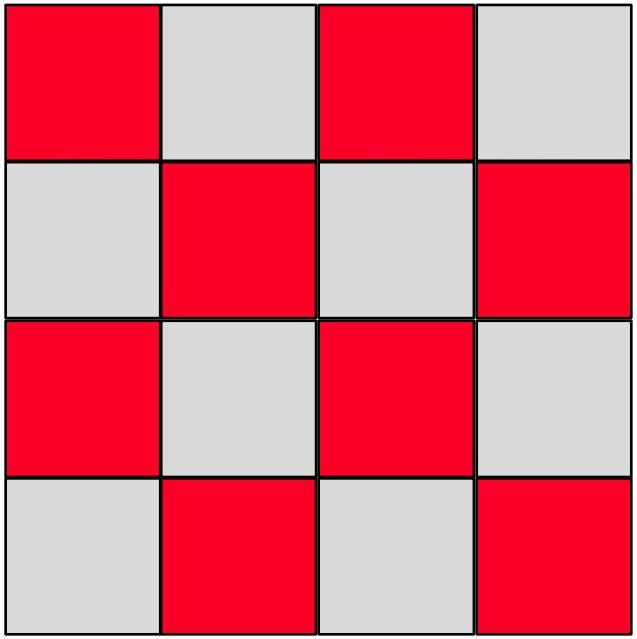


Chú ý

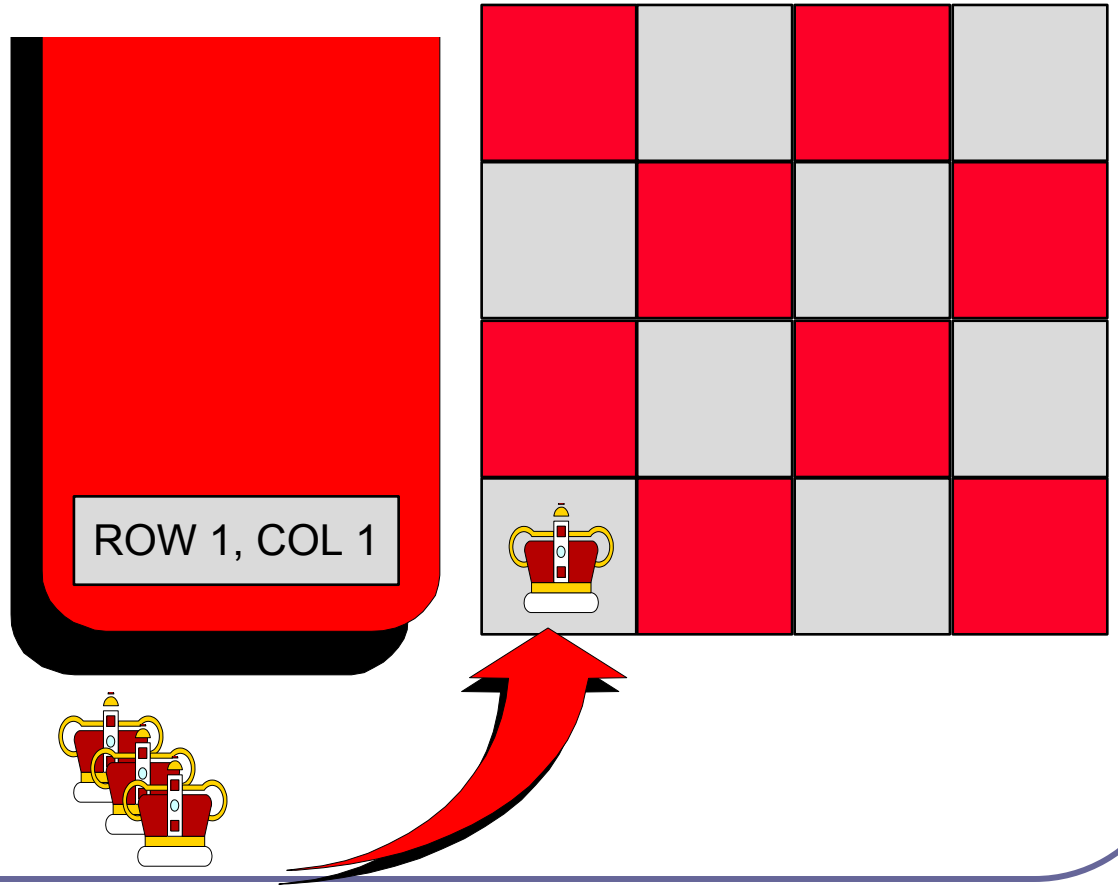
- Rõ ràng là bài toán xếp hậu không phải là luôn có lời giải, chẳng hạn bài toán không có lời giải khi n=2, 3. Do đó điều này cần được thông báo khi kết thúc thuật toán.
- Thuật toán trình bày ở trên là chưa hiệu quả. Nguyên nhân là ta đã không xác định được chính xác các tập UCV vào các vị trí của lời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

Thuật toán làm việc như thế nào



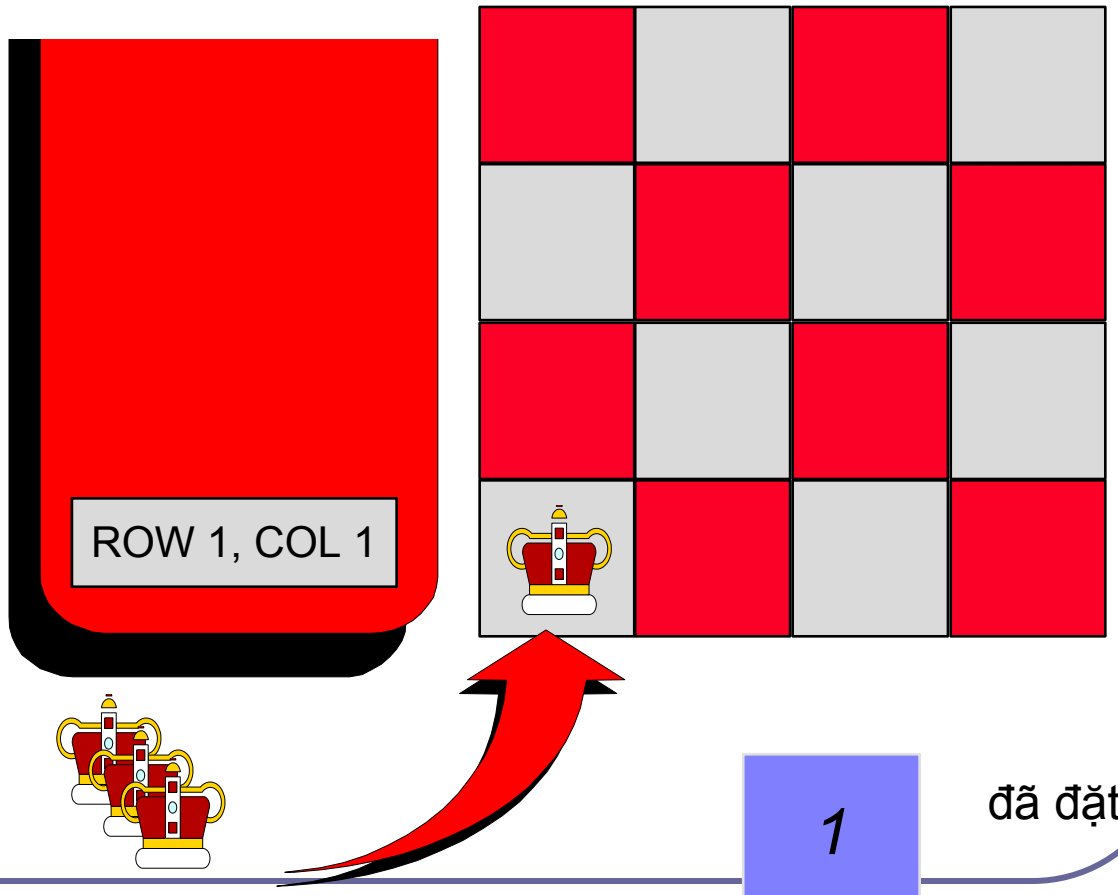


Thuật toán làm việc như thế nào



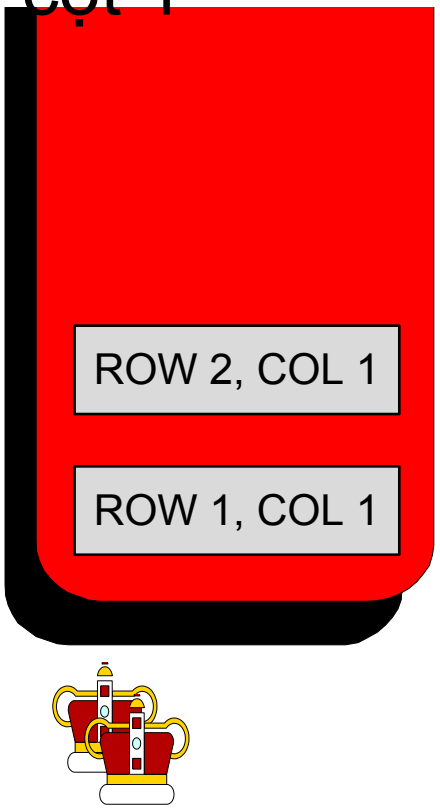
Thuật toán làm việc như thế nào

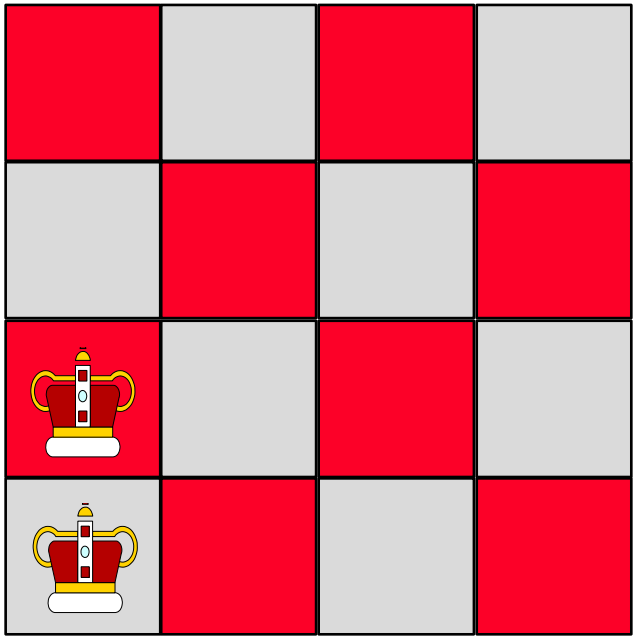
- Xếp con hậu ở dòng 1
  vào vị trí cột 1



Thuật toán làm việc như thế nào

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



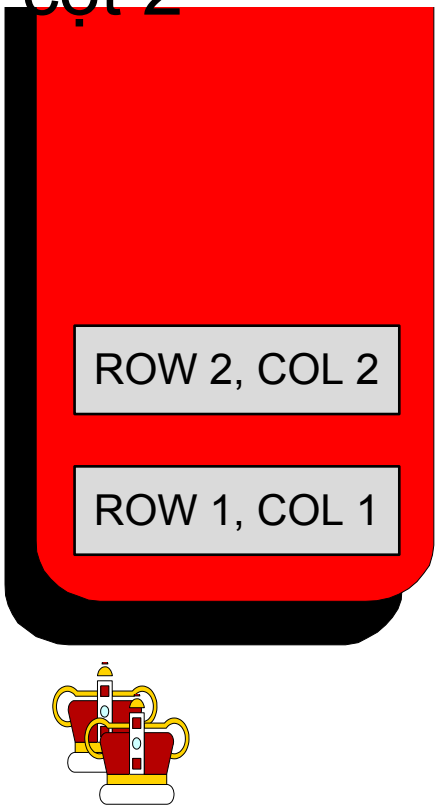


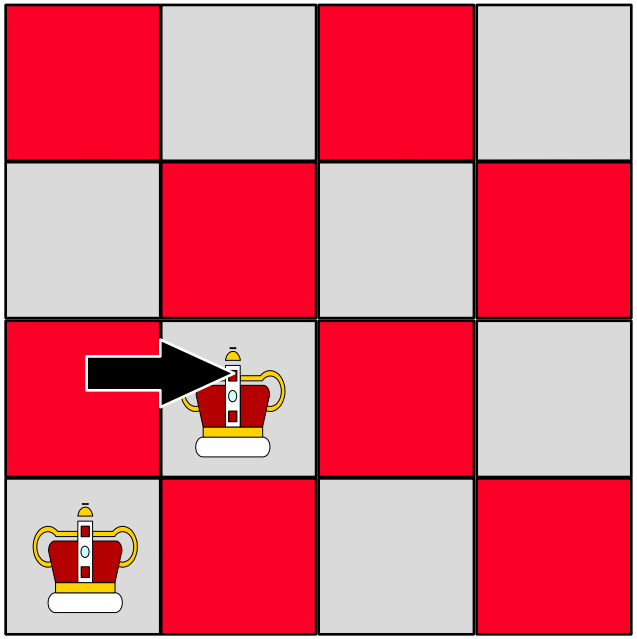
1

đã đặt

Thuật toán làm việc như thế nào

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



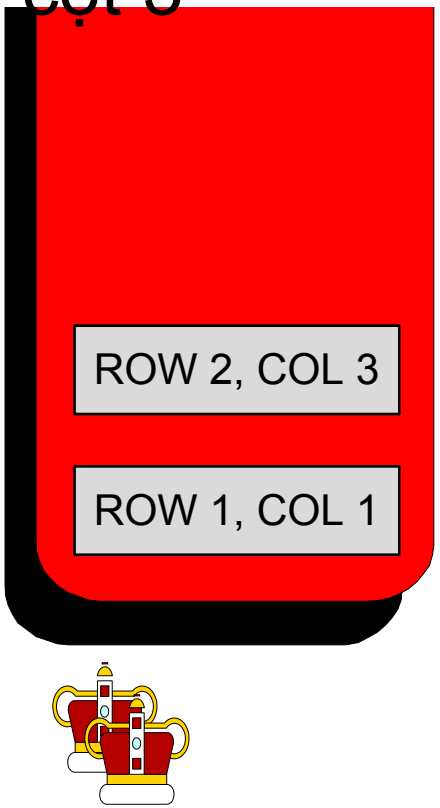


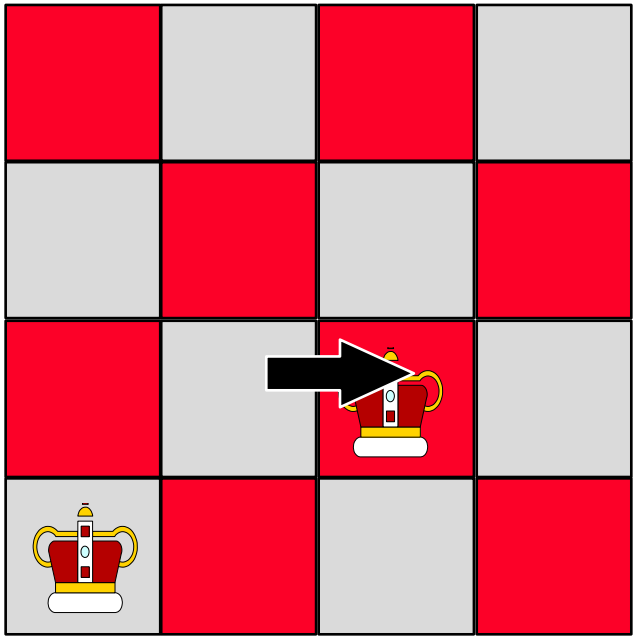
1

đã đặt

Thuật toán làm việc như thế nào

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



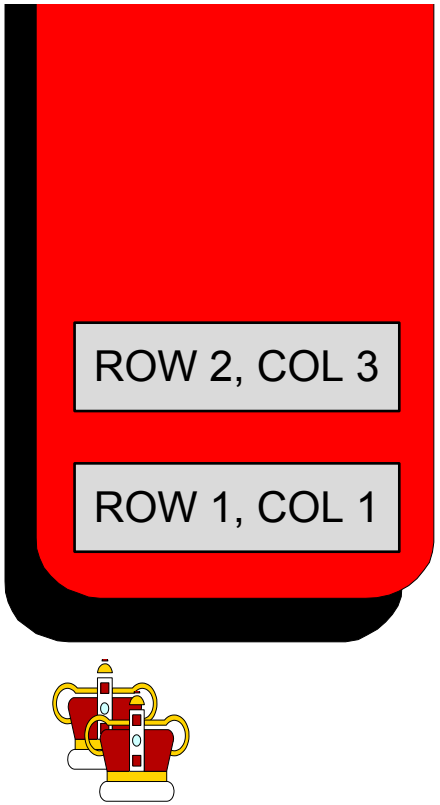


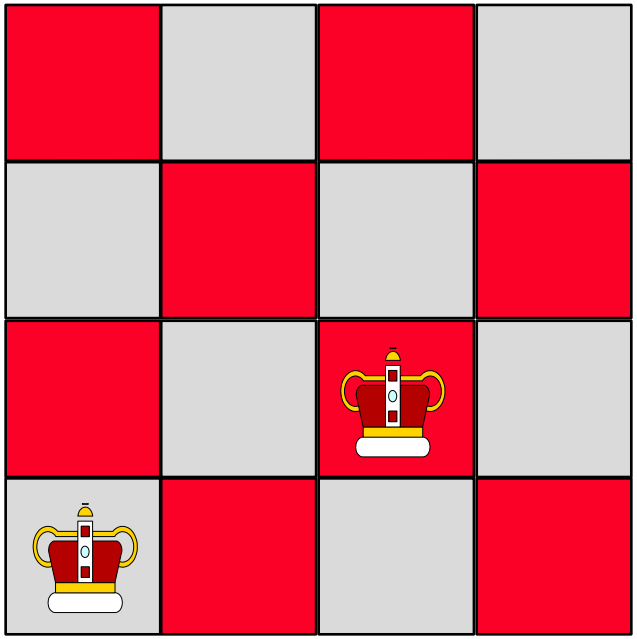
1

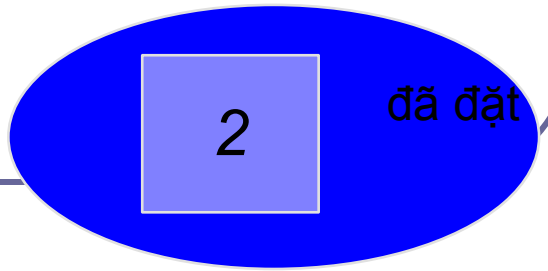
đã đặt

Thuật toán làm việc như thế nào

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

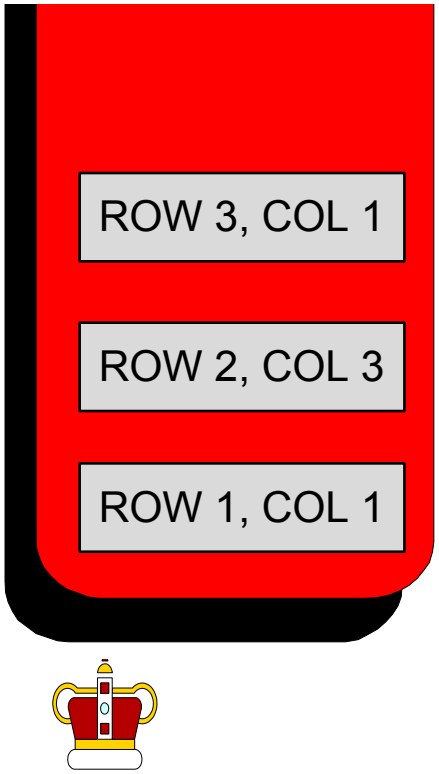


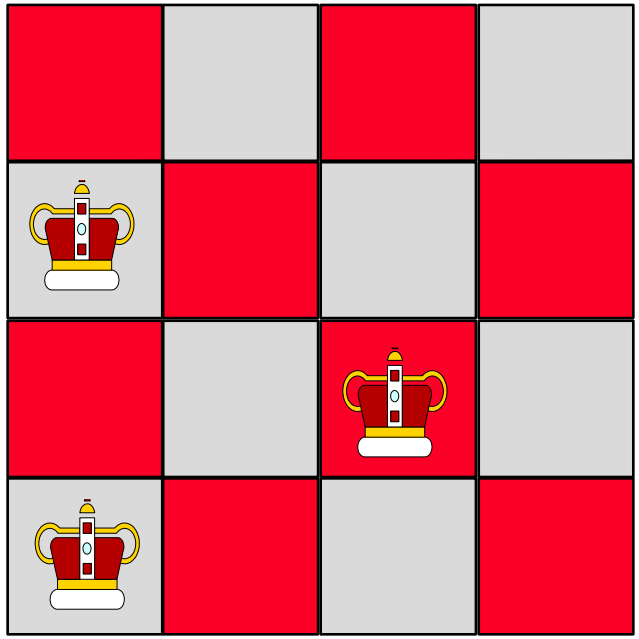




Thuật toán làm việc như thế nào

Thử xếp con hậu ở
dòng 3
vào cột đầu tiên



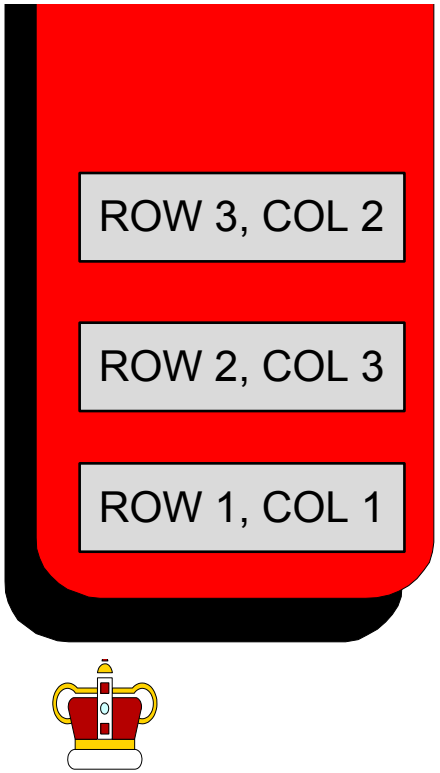


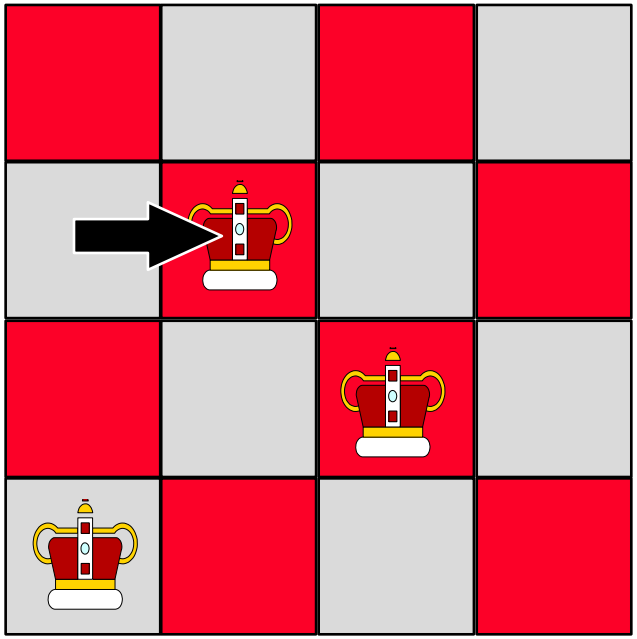
2

đã đặt

Thuật toán làm việc như thế nào

Thử cột tiếp theo



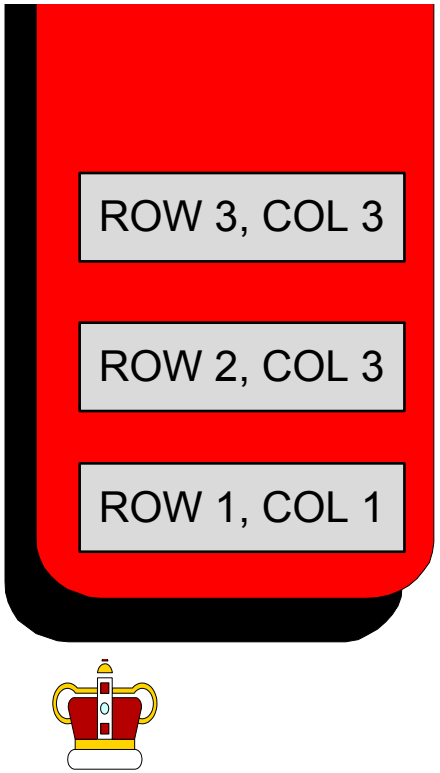


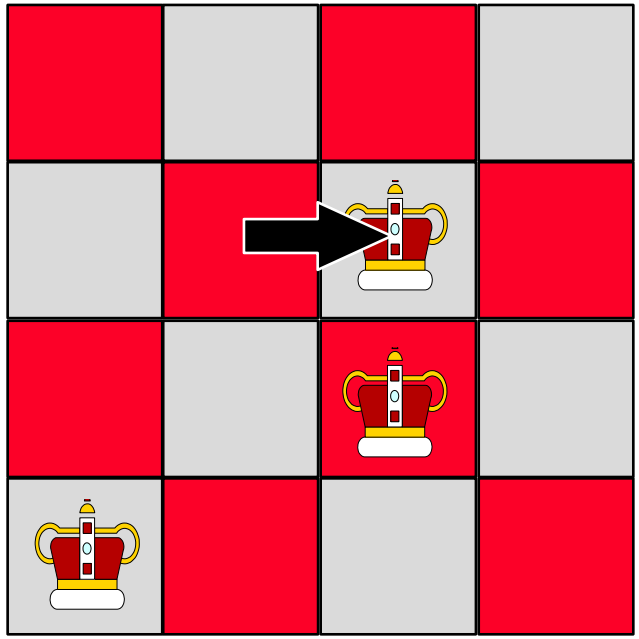
2

đã đặt

Thuật toán làm việc như thế nào

- Thử cột tiếp theo



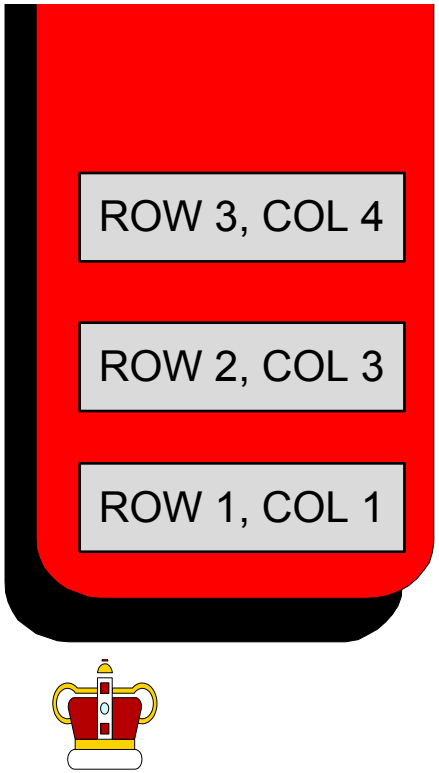


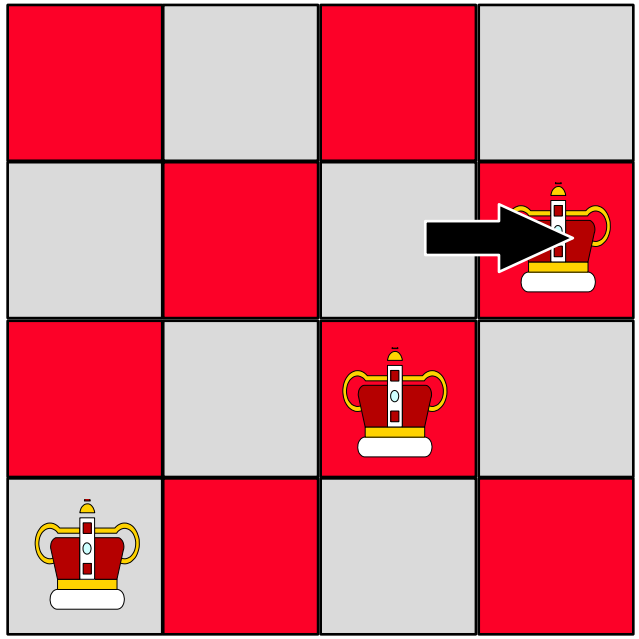
2

đã đặt

Thuật toán làm việc như thế nào

- Thử cột tiếp theo



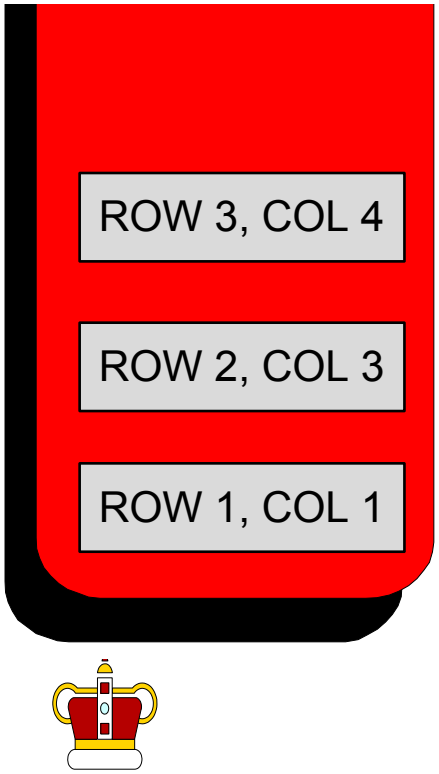


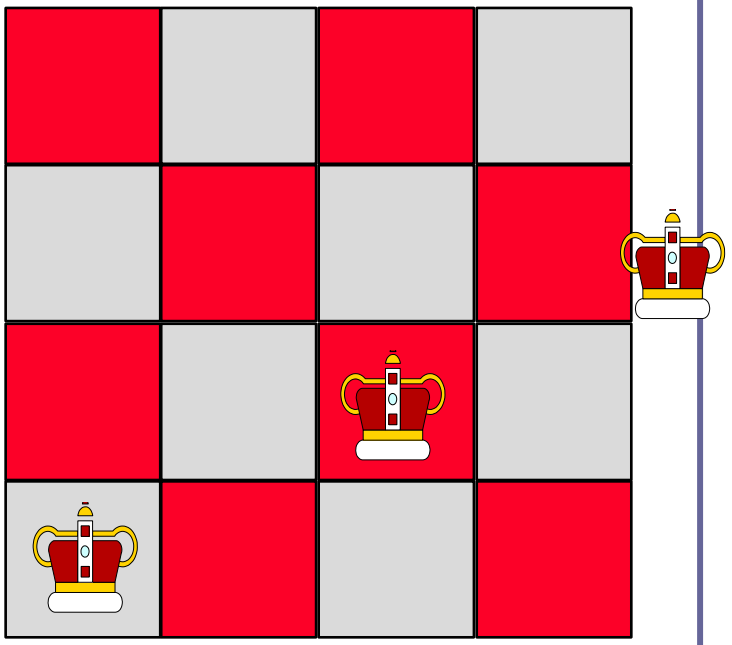
2

đã đặt

Thuật toán làm việc như thế nào

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



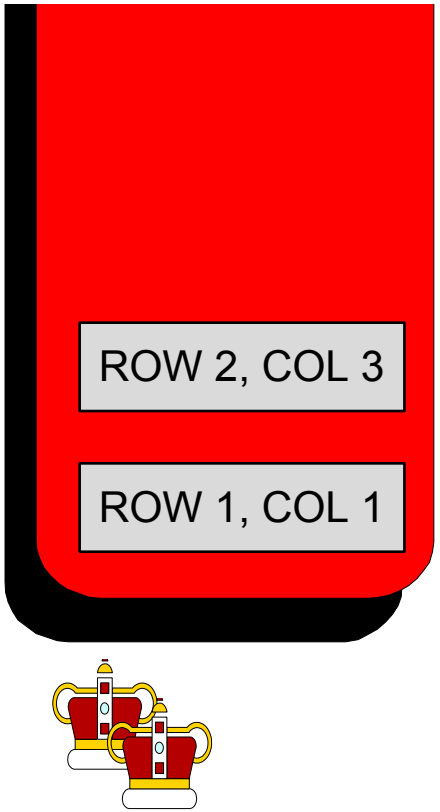


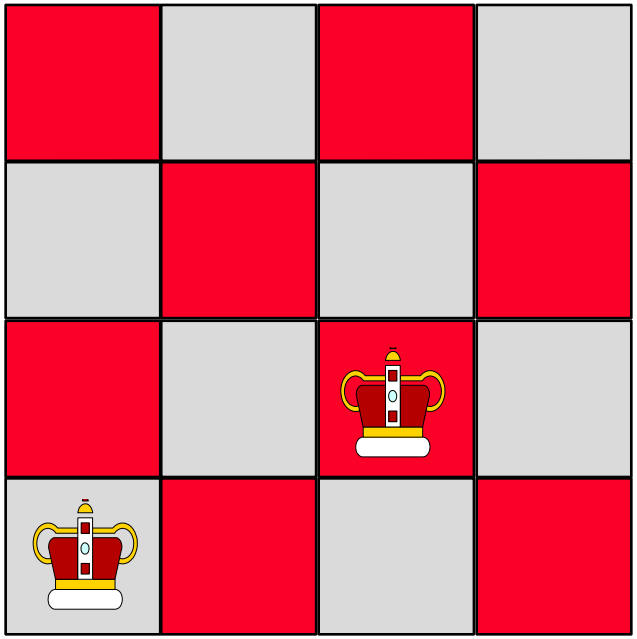
2

đã đặt

Thuật toán làm việc như thế nào

Quay lại dịch chuyển con hậu ở dòng 2



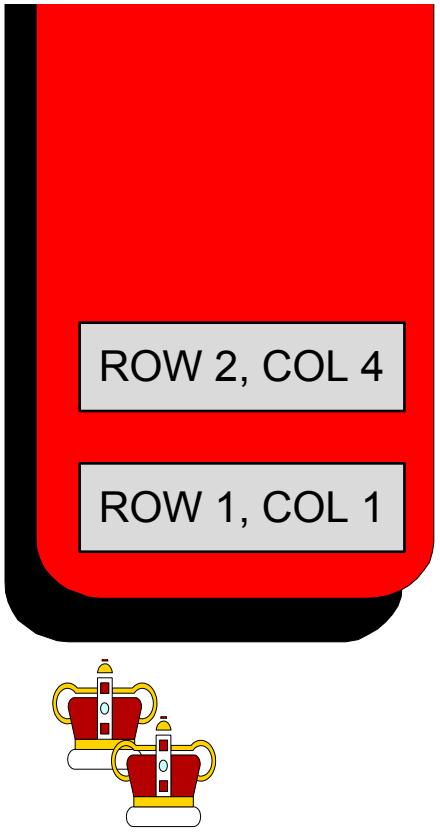


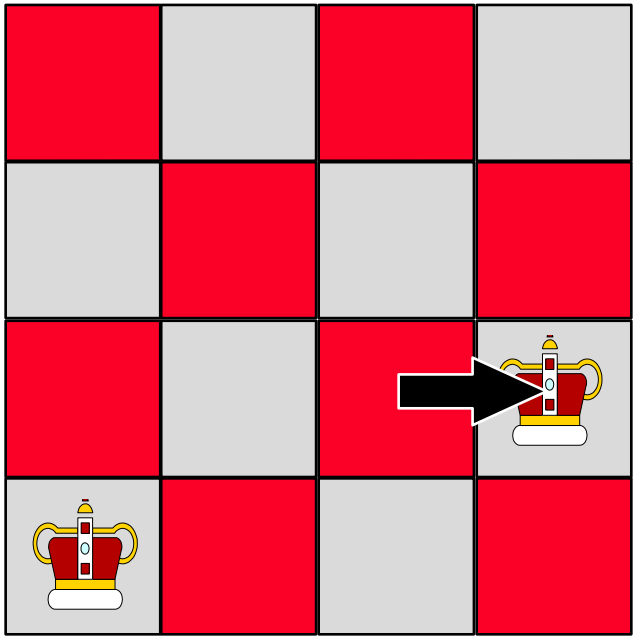
1

đã đặt

Thuật toán làm việc như thế nào

Đẩy con hậu ở dòng 2 sang cột thứ 4.

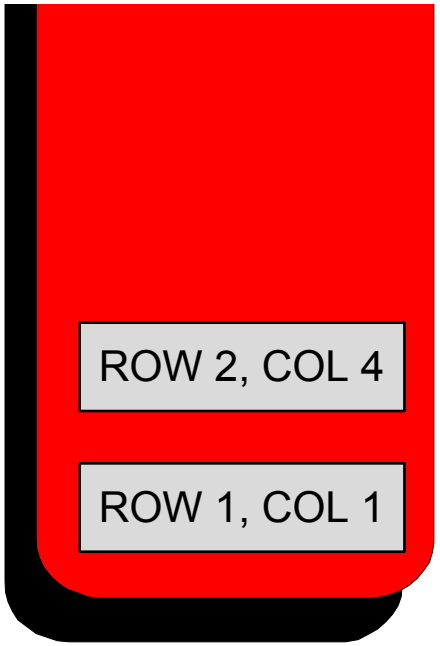


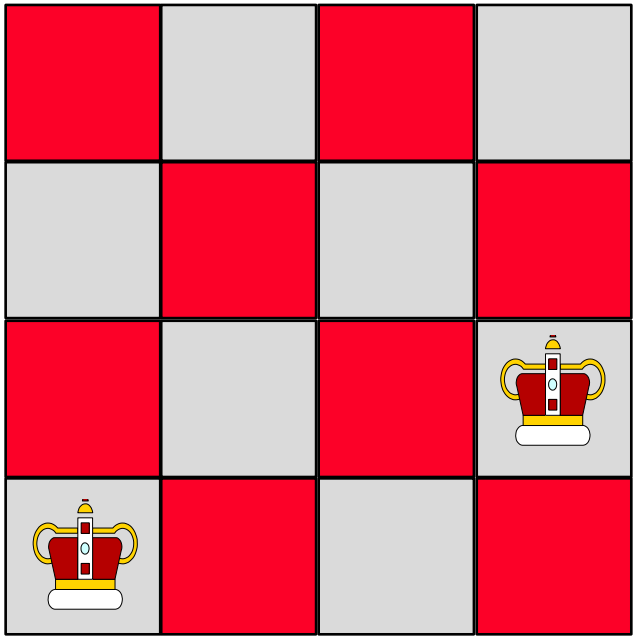


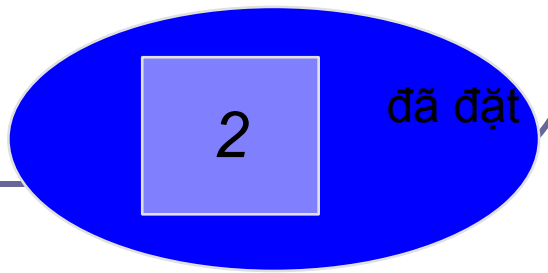
1 đã đặt

Thuật toán làm việc như thế nào

Xếp được con hậu ở dòng 2 ta tiếp tục xếp con hậu ở dòng 3

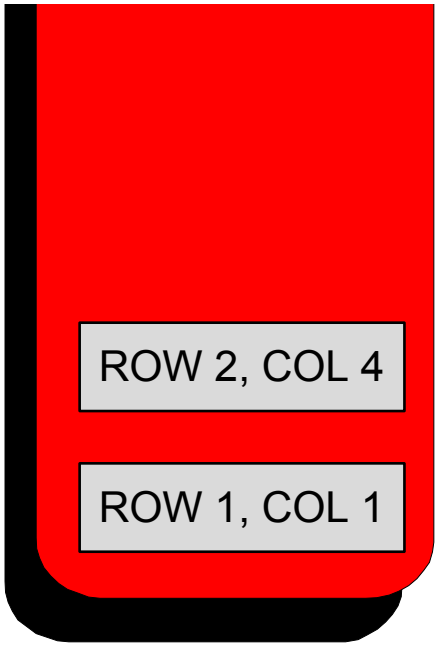


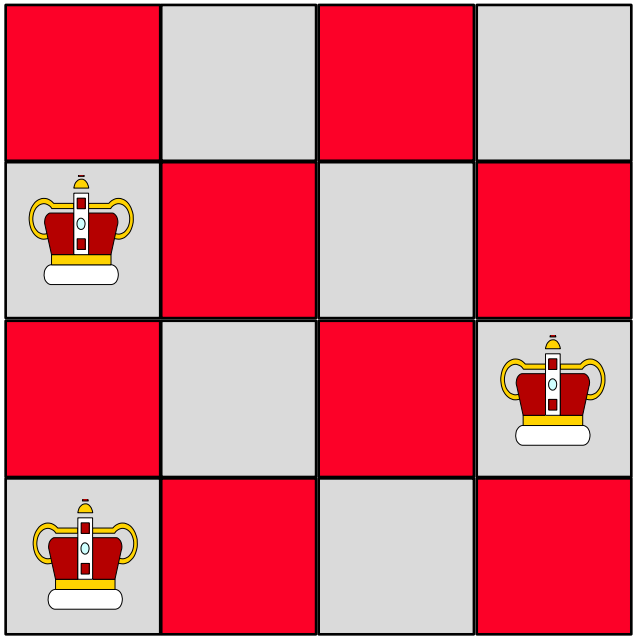


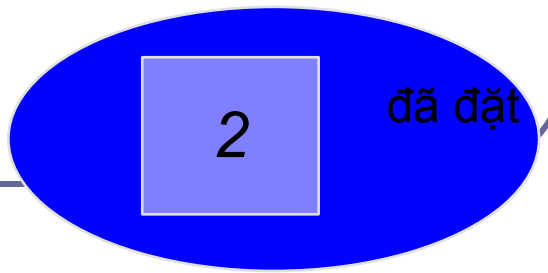


Thuật toán làm việc như thế nào

Thử xếp con hậu ở dòng 3 vào cột 1

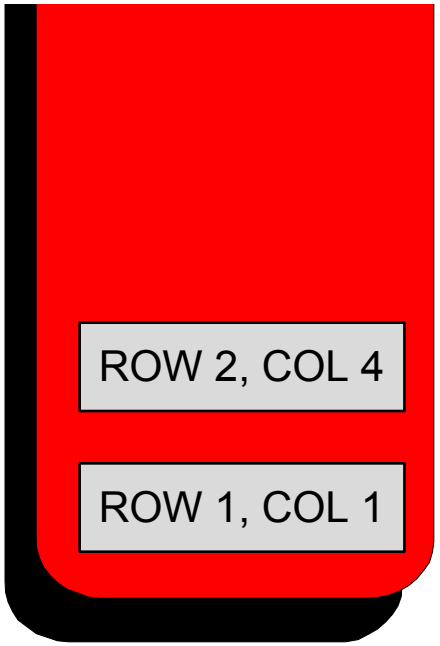


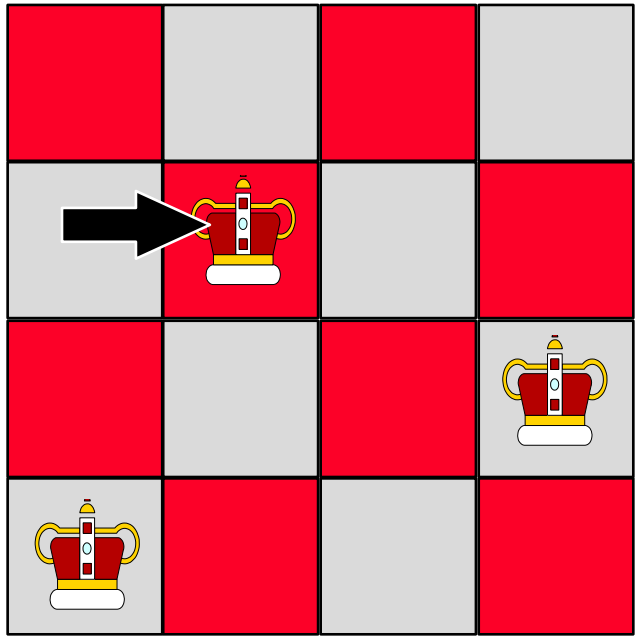




Thuật toán làm việc như thế nào

Thử xếp con hậu ở dòng 3 vào cột 2

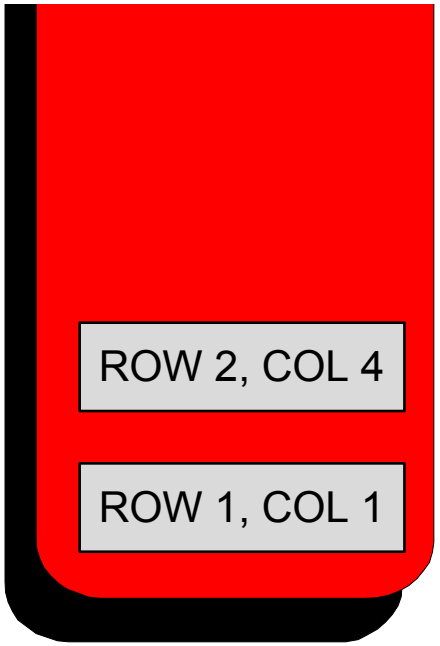


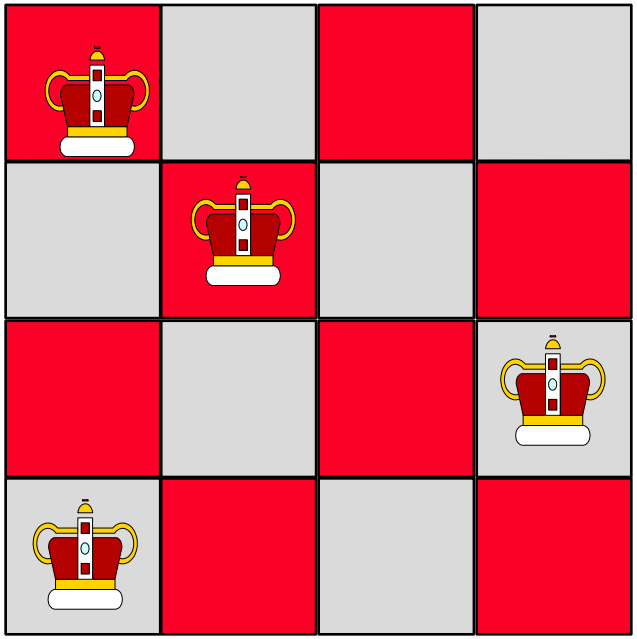


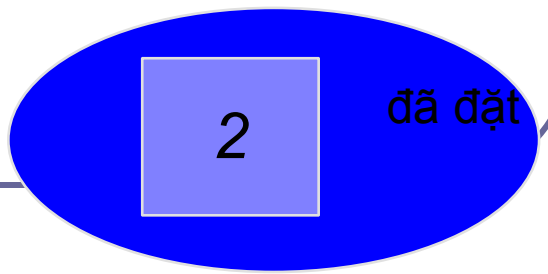
2 đã đặt

Thuật toán làm việc như thế nào

Xếp được con hậu ở dòng 3 ta tiếp tục xếp con hậu ở dòng 4: Thử cột 1

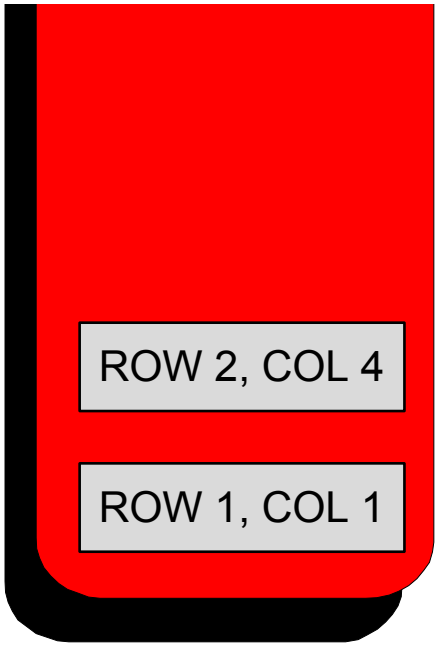


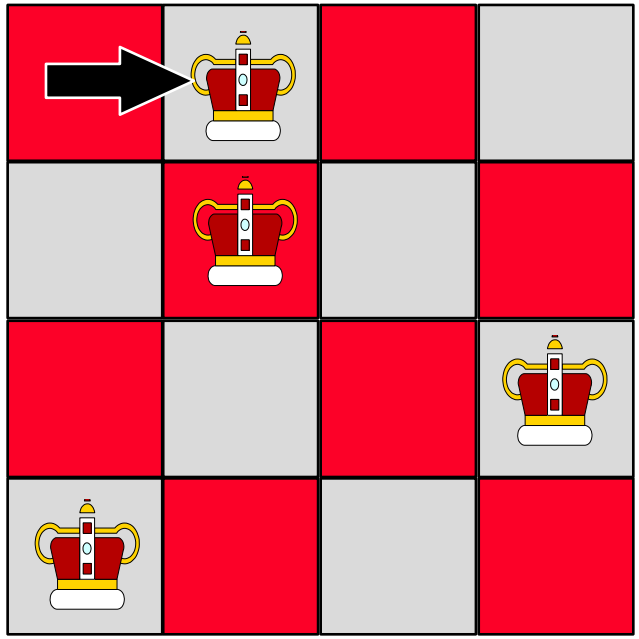




Thuật toán làm việc như thế nào

Thử xếp được con hậu ở dòng 4 vào cột 2

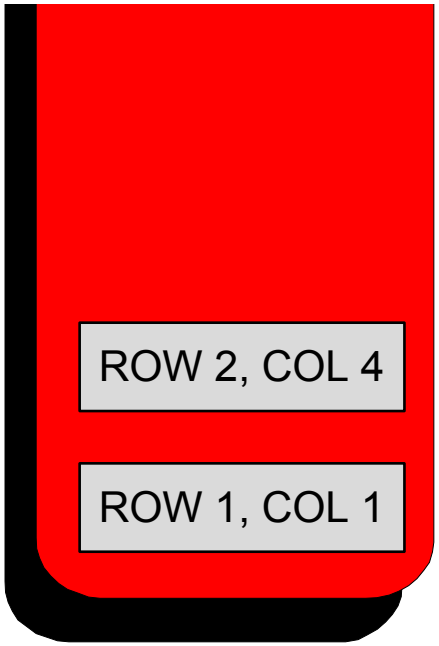


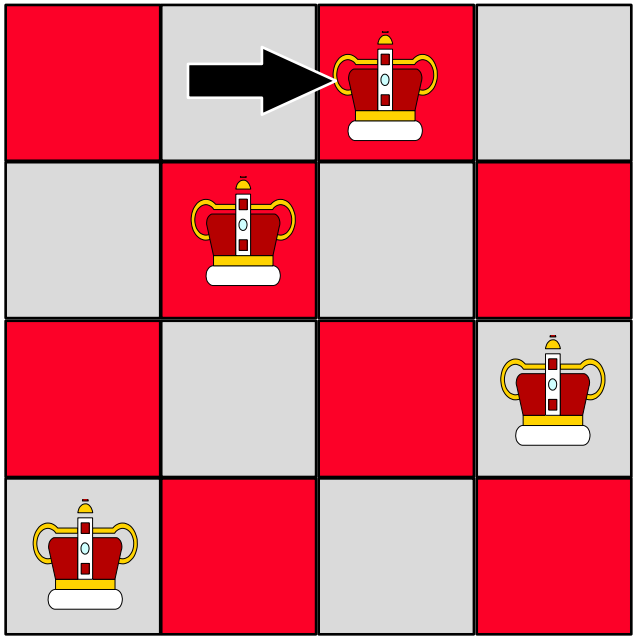


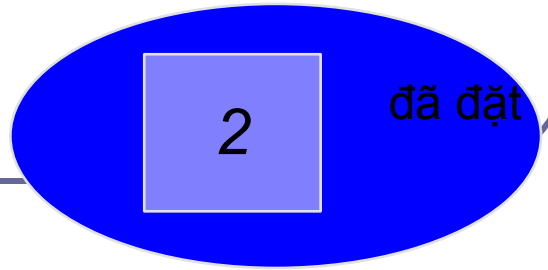
2 đã đặt

Thuật toán làm việc như thế nào

Thử xếp được con hậu ở dòng 4 vào cột 3

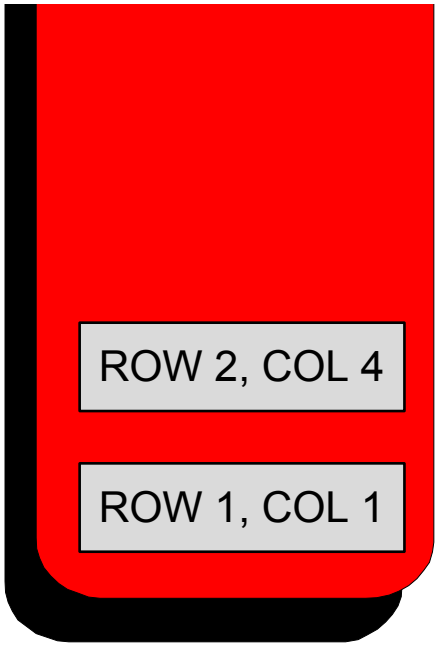


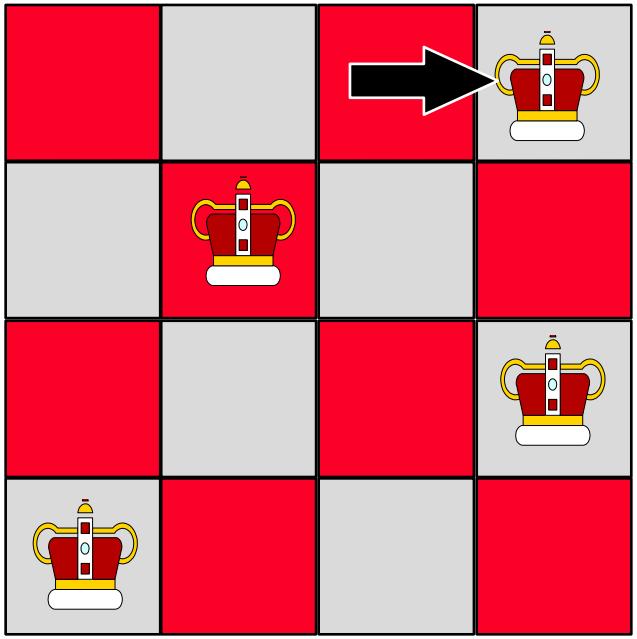


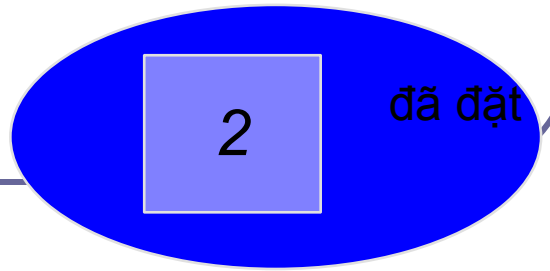


Thuật toán làm việc như thế nào

Thử xếp được con hậu ở dòng 4 vào cột 4

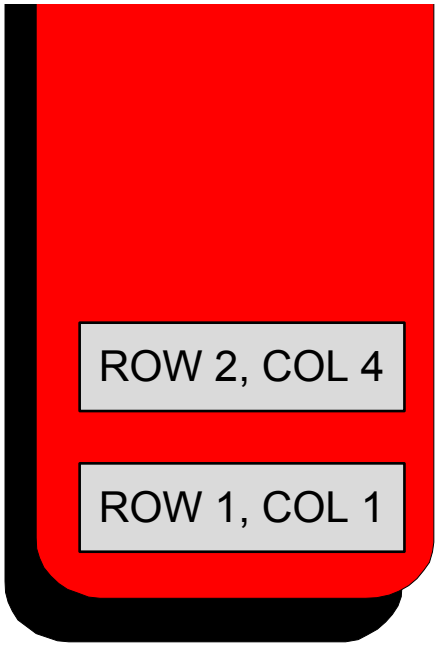


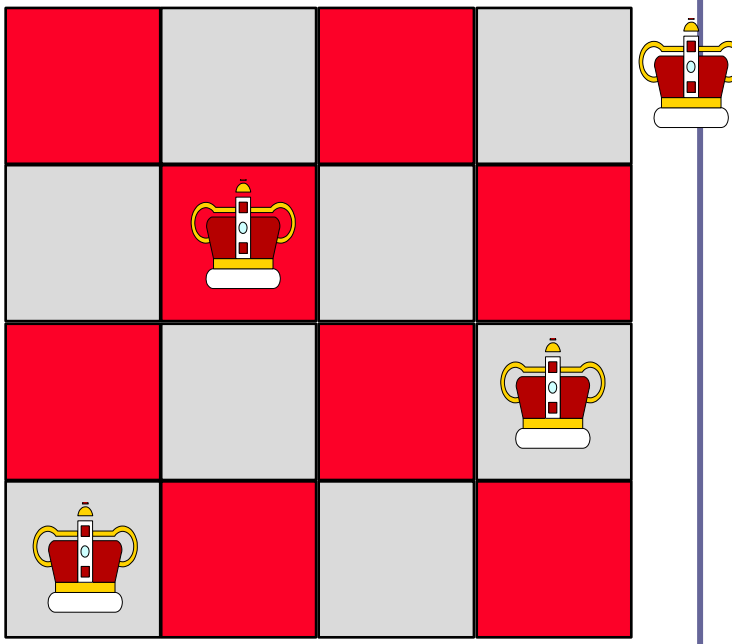


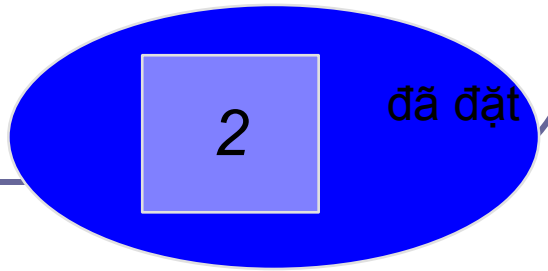


Thuật toán làm việc như thế nào

Không xếp được con hậu ở dòng 4

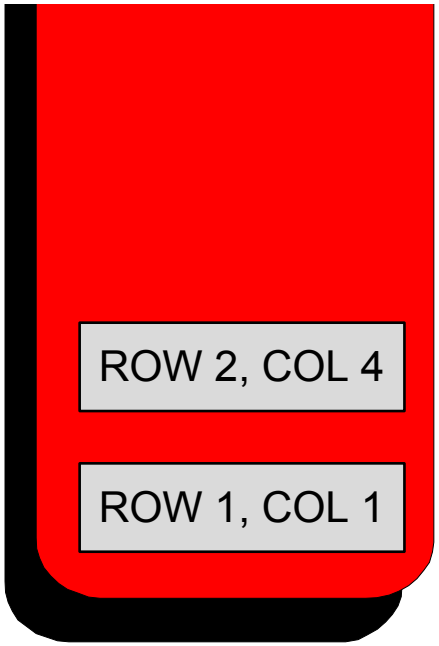


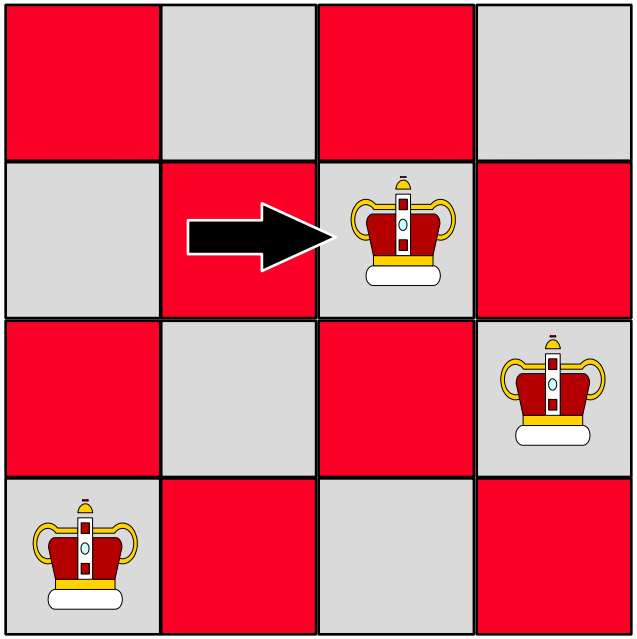


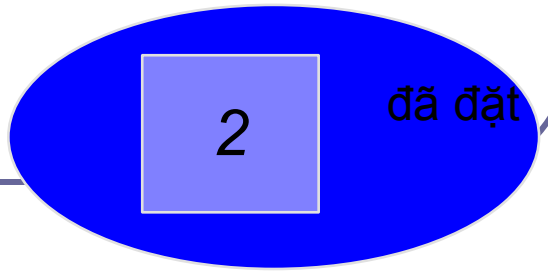


Thuật toán làm việc như thế nào

Quay lại tìm vị trí mới cho con hậu ở dòng 3: Thử cột 3

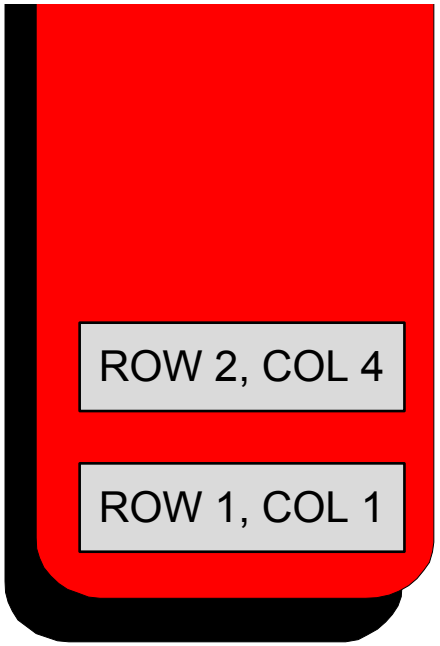


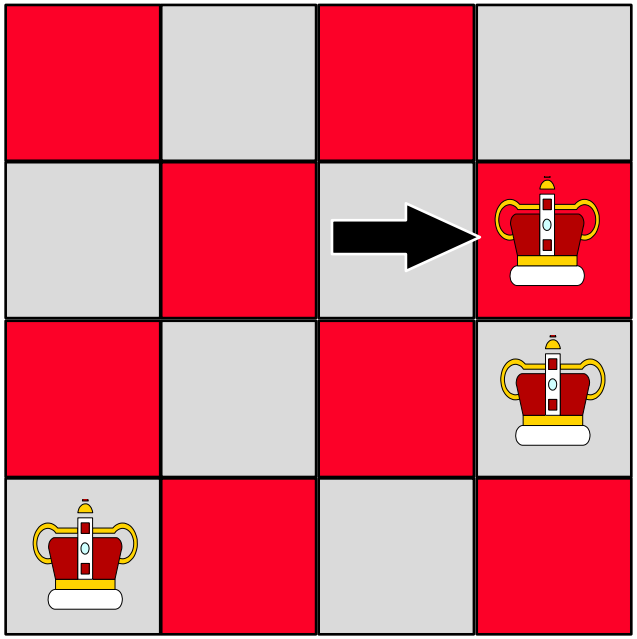


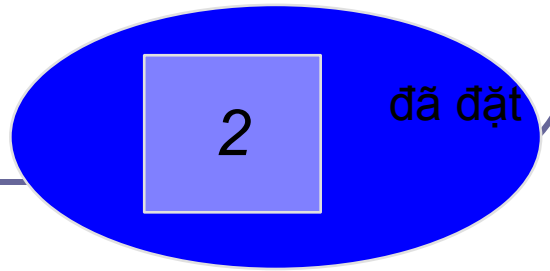


Thuật toán làm việc như thế nào

Quay lại tìm vị trí mới cho con hậu ở dòng 3: Thử cột 4

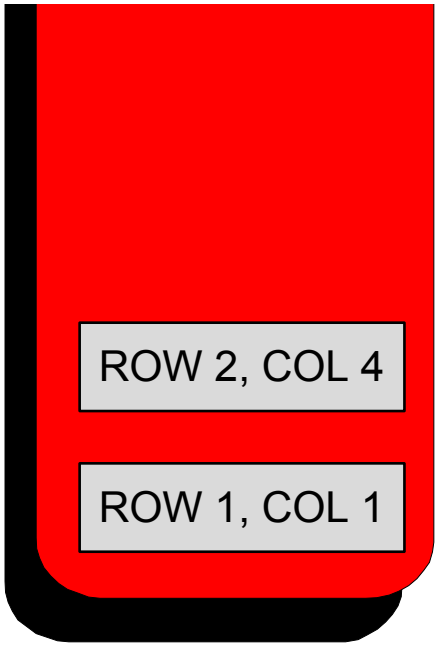


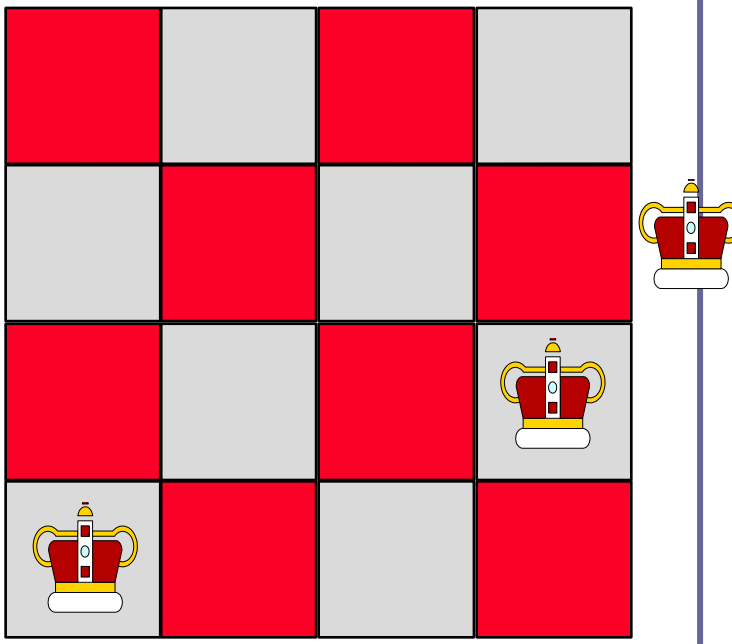




Thuật toán làm việc như thế nào

Không có cách xếp mới cho con hậu ở dòng 3

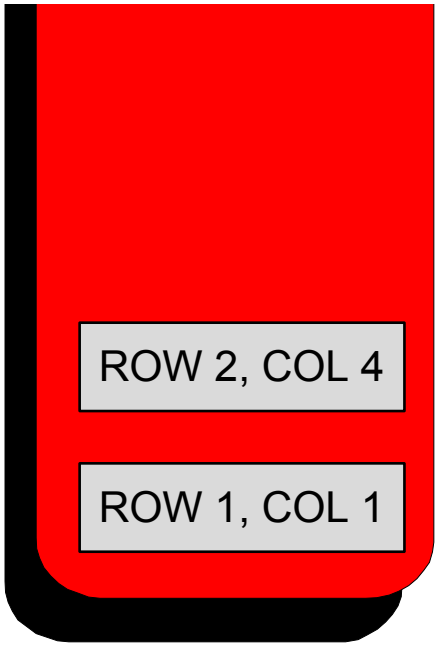


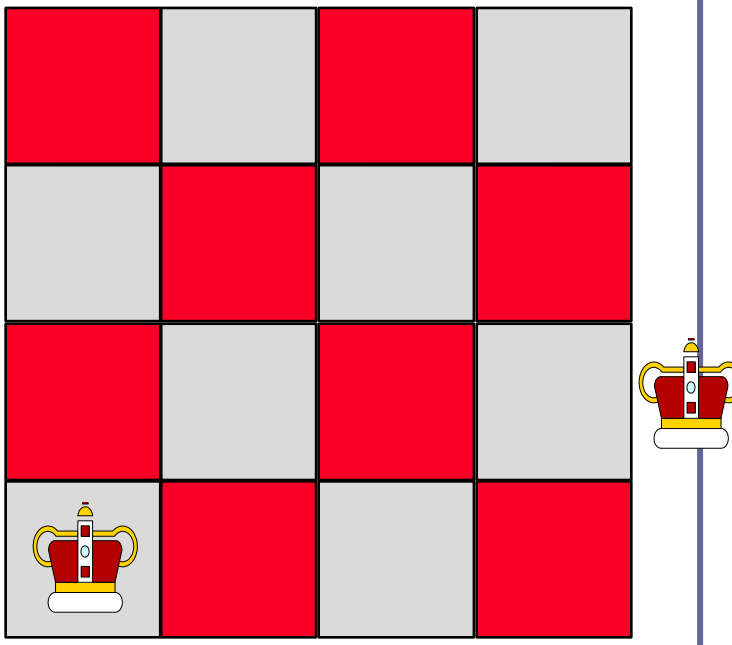


2 đã đặt

Thuật toán làm việc như thế nào

Quay lại tìm cách xếp mới cho con hậu ở dòng 2: Không có

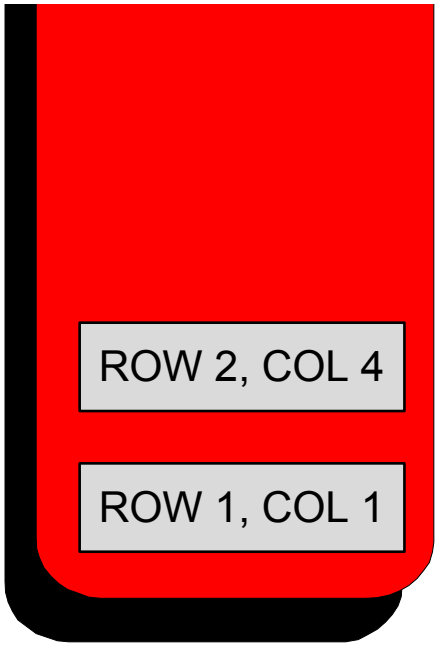


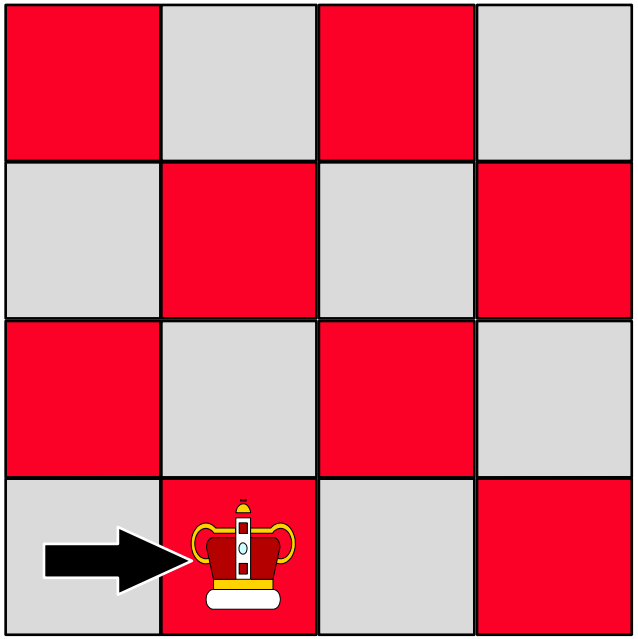


2 đã đặt

Thuật toán làm việc như thế nào

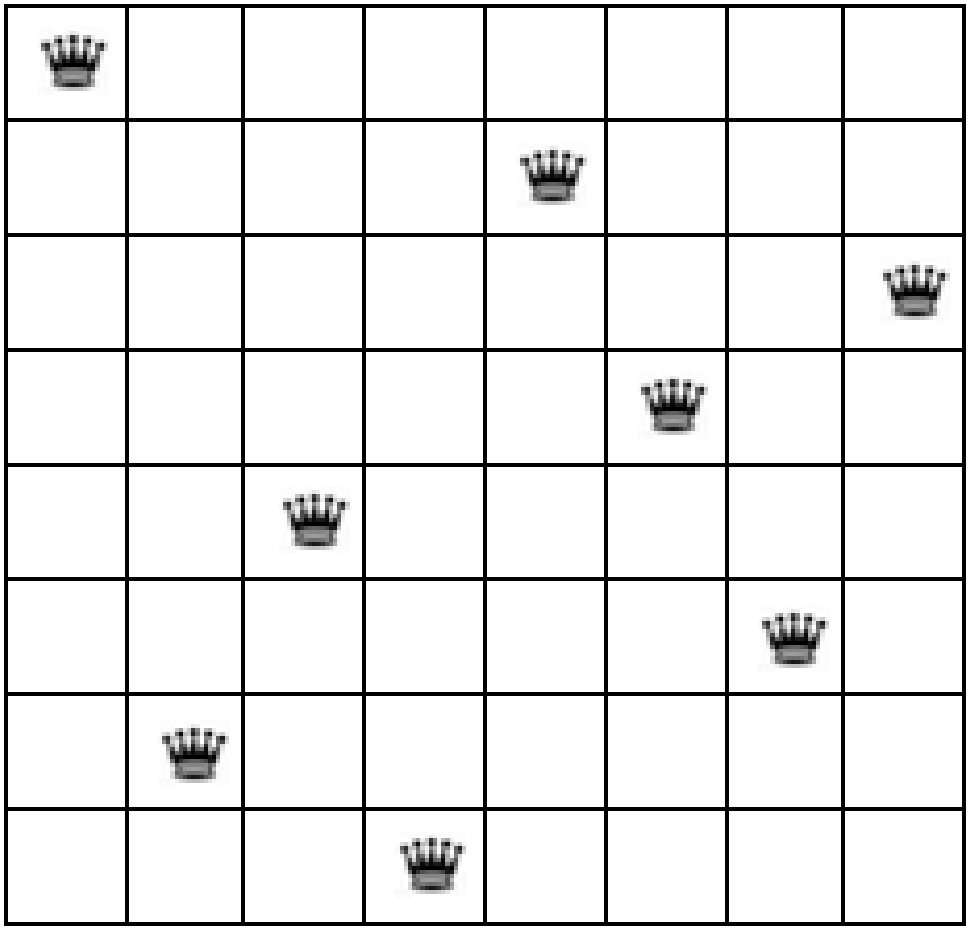
Quay lại tìm cách xếp mới cho con hậu ở dòng 1:
Chuyển sang cột 2





2 đã đặt

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



The End

Questions?

141