

## Bài 1A

# GIẢI THUẬT, CẤU TRÚC DỮ LIỆU

ThS. Nguyễn Thị Quỳnh Như  
Email: [ntqnhu@hou.edu.vn](mailto:ntqnhu@hou.edu.vn)

# Nội dung bài học

---

- Giải thuật và cấu trúc dữ liệu
- Phân tích và thiết kế giải thuật

# Đặt vấn đề

---

Cấu trúc dữ liệu+ giải thuật  
=chương trình

# Giải thuật và cấu trúc dữ liệu

- **Giải thuật** (thuật toán) là một dãy các câu lệnh chắc chắn và rõ ràng, xác định một trình tự các thao tác trên một số đối tượng nào đó sao cho sau một số hữu hạn bước thực hiện ta thu được kết quả mong muốn.
- Ví dụ: Thuật toán giải phương trình bậc 1:
  - Nếu  $a \neq 0$  thì  $x = -b/a$
  - ngược lại :
    - nếu  $b = 0$  thì phương trình vô số nghiệm
    - ngược lại thì phương trình vô nghiệm

# Các đặc trưng của thuật toán

- Tính đúng đắn
- Tính kết thúc
- Tính rõ ràng, chắc chắn
- Tính phổ dụng
- Tính hiệu quả

# Dữ liệu- Cấu trúc dữ liệu

- **Dữ liệu** là biểu diễn của các thông tin cần thiết cho bài toán: dữ kiện vào, ra, các kết quả trung gian...Nó có thể là 1 chữ số, 1 ký tự hay 1 con số, 1 từ ...
- **Cấu trúc dữ liệu** là cách tổ chức, liên kết dữ liệu với nhau.
- Cách biểu diễn một cấu trúc dữ liệu trong bộ nhớ được gọi là **cấu trúc lưu trữ**.
- Lưu ý: Một cấu trúc dữ liệu có thể có nhiều cấu trúc lưu trữ khác nhau

# Tiêu chuẩn lựa chọn CTDL

- **Ví dụ:** Tổ chức một danh sách hồ sơ sinh viên với mỗi sinh viên có nhiều thuộc tính có kiểu khác nhau.
- Tiêu chuẩn lựa chọn CTDL:
  - CTDL phải biểu diễn được đầy đủ các thông tin nhập và xuất của bài toán
  - CTDL phải phù hợp với các thao tác thuật toán mà ta lựa chọn để giải quyết bài toán
  - CTDL phải cài đặt được trên máy tính với ngôn ngữ lập trình đang sử dụng

# Biểu diễn thuật toán

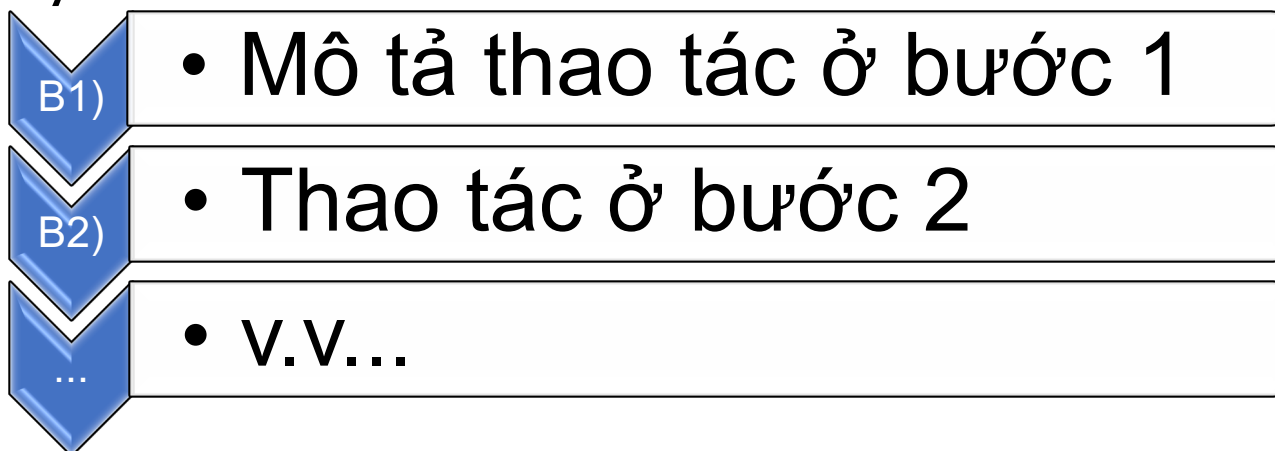
---

- Mô tả từng bước
- Sơ đồ khối



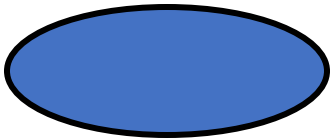



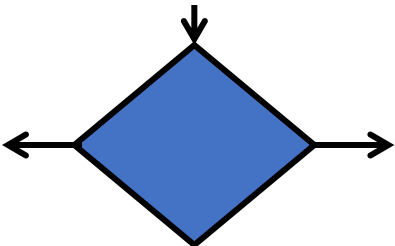
# Biểu diễn TT: mô tả từng bước

- Nêu trình tự từ bước 1 đến bước cuối cùng, ở mỗi sử dụng ngôn ngữ giả lập trình cùng với ngôn ngữ tự nhiên để trình bày.

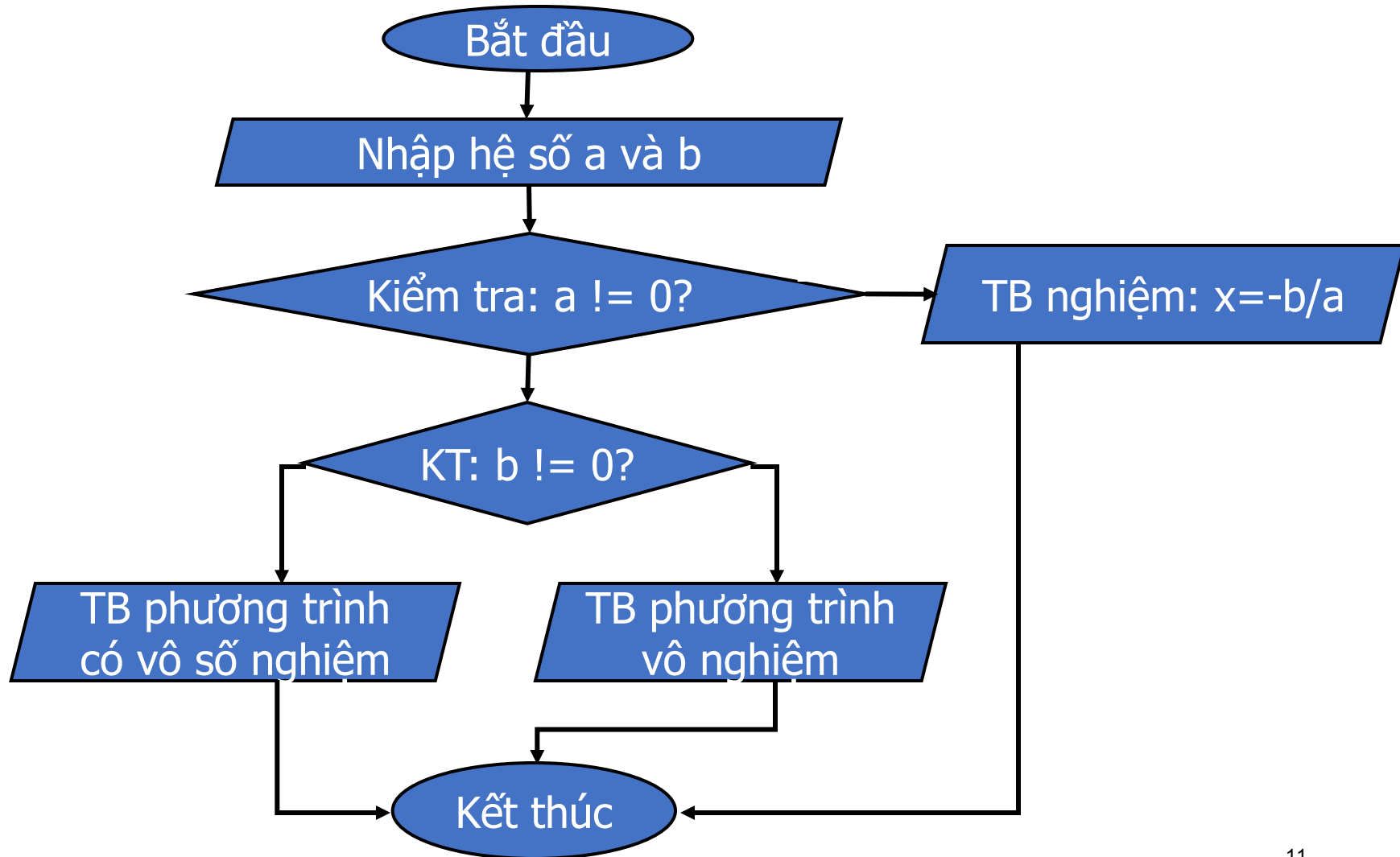


- Các thao tác được mô tả phải chính xác và rõ ràng, đơn nghĩa và dễ hiểu.

# Biểu diễn TT: Sơ đồ khối

Hình Vẽ	Ý Nghĩa
	Thể hiện sự bắt đầu và kết thúc thuật toán
	Thể hiện sự tính toán, xử lý
	Thể hiện cho thao tác nhập dữ liệu vào đưa ra kết quả in
	Thể hiện chiều đi của thuật toán
	Thể hiện sự lựa chọn đúng (Đ) hoặc sai (S), có một chiều đi vào và hai chiều đi ra tương ứng với hai trường hợp (Đ,S)

## Ví dụ: TT giải PT $a.x+b=0$



# Phân tích và thiết kế giải thuật

- **Thiết kế giải thuật:**

- **Môđun hoá bài toán:** là phương pháp chia nhỏ bài toán cần giải quyết thành các bài toán nhỏ hơn, dễ dàng giải quyết → chiến thuật "chia để trị" (divide and conquer) hay thiết kế top-down.
- **Ví dụ:** chương trình quản lý đầu sách thư viện nhằm phục vụ độc giả tra cứu sách.
  - **Input:** Có 1 file dữ liệu gồm các bản ghi về thông tin liên quan đến một đầu sách: tên sách, mã số, tác giả, NXB, năm XB, giá tiền.
  - **Yêu cầu:** cập nhật dữ liệu, tìm kiếm và in ấn

# Thiết kế giải thuật

- **Phương pháp tinh chỉnh từng bước:** là phương pháp thiết kế giải thuật gắn liền với lập trình. Phương pháp này phản ánh tinh thần quá trình mô đun hóa bài toán và thiết kế kiểu top-down
- **Ví dụ:** Lập chương trình sắp xếp dãy số theo thứ tự tăng dần
- **Phác thảo giải thuật:**
  - “Từ dãy số chưa sắp xếp đã cho tìm ra số nhỏ nhất, đặt vào cuối dãy đã được sắp xếp
  - Cứ lặp lại quy trình đó cho đến khi dãy chưa sắp xếp là rỗng”

# Tinh chỉnh thuật toán

- Bước tinh chỉnh thứ 1:

```
for (i=1; i<=n; i++)  
{ B1:   xét từ  $a_i$  đến  $a_n$  để tìm số nhỏ  
nhất  $a_j$   
      B2: đổi chỗ  $a_i$  và  $a_j$   
}
```

- Bước tinh chỉnh thứ 2:

- B1: tìm số nhỏ nhất  $a_j$

```
    j=i;  
    for (k=j+1; k<=n; k++)  
        if ( $a_k < a_j$ ) j=k;
```

- B2: đổi chỗ  $a_i$  và  $a_j$

```
        tg= $a_i$ ;  $a_i=a_j$ ;  $a_j=tg$ ;
```

# Phân tích giải thuật

- Phân tích thời gian thực hiện giải thuật:
  - Thời gian thực hiện chương trình phụ thuộc rất nhiều yếu tố, gồm kích thước của dữ liệu phần cứng máy tính, ngôn ngữ để viết chương trình, chương trình dịch...
  - Tuy nhiên các yếu tố này không giống nhau trên các máy tính, do đó ta chỉ có thể đánh giá được thời gian thực hiện khi  $n$  (số lượng dữ liệu) lớn, cách đánh giá như vậy gọi là đánh giá độ phức tạp của giải thuật.

# Đánh giá độ phức tạp của thuật toán

- Gọi  $n$  là số lượng dữ liệu, thời gian thực hiện  $T$  của giải thuật được biểu diễn như một hàm của  $n$ :  $T(n)$ 
  - Ta nói độ phức tạp của giải thuật là  $g(n)$  nếu tồn tại một hằng  $c$  và số  $n_0$  sao cho:  $T(n) \leq cg(n)$  khi  $n > n_0$ , nghĩa là  $T(n)$  bị chặn trên bởi 1 hằng số  $c \cdot g(n)$ , với mọi  $n$ .
- **Ký hiệu:**  $T(n) = O(g(n))$
- Thông thường các hàm thể hiện độ phức tạp giải thuật có dạng:  $\log_2 n$ ,  $n$ ,  $n \log_2 n$ ,  $n^2$ ,  $n^3$ ,  $2^n$ ,  $n!$ ,  $n^n$ .



# Xác định độ phức tạp của thuật toán

- **Quy tắc tổng:** Giả sử  $T_1(n)$  và  $T_2(n)$  là thời gian thực hiện 2 đoạn chương trình  $P_1$  và  $P_2$  mà  $T_1(n)=O(f(n))$ ,  $T_2(n)=O(g(n))$ .
- $P_1$  thực hiện rồi đến  $P_2$  thì  $T(n)=T_1(n)+T_2(n)=O(f(n)+g(n))$ , ta cũng có thể coi  $T(n)=O(\max(f(n),g(n)))$ .
- Ví dụ: `scanf(x);`  $\rightarrow O(1)$   
`x=x+5;`  $\rightarrow O(1)$   
 $\rightarrow$  thời gian thực hiện 2 lệnh trên có độ phức tạp  $O(\max(1,1))=O(1)$ .

# Xác định độ phức tạp của thuật toán (tt)

- **Quy tắc nhân:** Giả sử đoạn chương trình P có thời gian thực hiện là  $T(n)=O(f(n))$ . Khi đó, nếu thực hiện  $k(n)$  lần đoạn chương trình P với  $k(n)=O(g(n))$  thì độ phức tạp tính toán sẽ là  $O(g(n).f(n))$
- **Ví dụ:**

```
for (i=0; i<n; i++)  
    for (j=0; j<n; j++) x=x+1;
```
- Ta thấy thời gian thực hiện của  $x=x+1$  là 1 hàm có thời gian thực hiện là hằng số, ký hiệu  $O(1)$ .  
→ câu lệnh 

```
for (i=0; i<n; i++) x=x+1
```

 có thời gian thực hiện  $O(n*1)=O(n)$   
→ **Như vậy:**

```
for (i=0; i<n; i++)  
    for (j=0; j<n; j++) x=x+1;
```

 có thời gian thực hiện là  $O(n*n)=O(n^2)$ .

## Bài 1B

# ĐỆ QUY VÀ GIẢI THUẬT ĐỆ QUY

# Nội dung bài giảng

---

- Khái niệm đệ quy
- Giải thuật đệ quy
- Thiết kế giải thuật đệ quy
- Đệ quy quay lui (back tracking)

# 1. Khái niệm về đệ quy

- Đối tượng là đệ quy nếu được định nghĩa **qua chính nó** hoặc một đối tượng khác **cùng dạng với chính nó** bằng quy nạp
- Ví dụ:
  - Đặt hai chiếc gương đối diện nhau
  - Tính giai thừa

## 2. Giải thuật đệ quy

- Giải thuật đệ quy là lời giải một bài toán  $P$  được thực hiện bằng lời giải của bài toán  $P'$  có dạng giống như  $P$ ,  **$P'$  phải “nhỏ” hơn  $P$ .**
- **Định nghĩa một hàm đệ quy hay thủ tục đệ quy gồm:**
  - **Phần cơ định/phần suy biến:** phần này được thực hiện khi mà công việc quá đơn giản, có thể giải trực tiếp chứ không cần phải nhờ đến một bài toán con nào cả.
  - **Phần đệ quy/hạ bậc:** xác định những bài toán con và gọi đệ quy giải những bài toán con đó, có lời giải các bài toán con thì phối hợp chúng lại để giải bài toán.

## 2. Giải thuật đệ quy (2)

- Như vậy:

- Phần suy biến chứa các câu lệnh thực hiện đối với 1 giá trị cụ thể ban đầu của tham số.
- Phần hạ bậc chứa các câu lệnh được định nghĩa bằng các câu lệnh đã được định nghĩa trước đó với kích thước nhỏ hơn của tham số

- Lưu ý:

- Không phải lúc nào tính đệ quy của bài toán cũng hiện rõ và đơn giản.
- Muốn tìm ra được tính đệ quy cần trả lời các câu hỏi:
  - Có thể định nghĩa bài toán đã cho bằng bài toán cùng dạng nhưng nhỏ hơn hay không?
  - Kích thước của bài toán được giảm như thế nào ở mỗi lần gọi đệ quy?
  - Trường hợp suy biến của bài toán là gì?

### 3. Thiết kế giải thuật đệ quy

---

- Tính giai thừa
- Hàm Fibonacci



## a. Tính giai thừa

- **Bài toán:** Cho số nguyên  $n$ , tính giai thừa của  $n$   
 $\Leftrightarrow n! = n * (n-1)!$
- **Phân tích bài toán:**
  - Phần suy biến: định nghĩa kết quả hàm  $=1$  tại  $n=0$
  - Phần đệ quy: sẽ định nghĩa kết quả hàm qua giá trị của  $n$  và giai thừa của  $(n-1)$
- Ví dụ:
  - $3! = 3 * 2!$
  - $2! = 2 * 1!$
  - $1! = 1 * 0!$
  - $0! = 1$

# Cài đặt giải thuật

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

## b. Dãy số Fibonacci:

- Dãy số Fibonacci bắt nguồn từ bài toán cổ về sinh sản của các cặp thỏ.
- **Bài toán:**
  - Các con thỏ không bao giờ chết
  - Hai tháng sau khi ra đời, mỗi cặp thỏ sẽ sinh ra một cặp thỏ con (một đực, một cái)
  - Khi đã sinh ra rồi thì cứ mỗi tháng tiếp theo chúng lại sinh được một cặp mới
  - ***Yêu cầu của bài toán: Giả sử từ đầu tháng 1 có một cặp mới ra đời thì đến tháng thứ  $n$  sẽ có bao nhiêu cặp?***

# Ví dụ về dãy Fibonacci

- $n = 5$ 
  - tháng thứ 1: 1 cặp (a,b)
  - tháng thứ 2: 1 cặp (a,b) (*vẫn chưa đẻ*)
  - tháng thứ 3: 2 cặp (a,b) (c,d) (*để thêm 1 cặp*)
  - tháng 4: 3 cặp (a,b) (c,d) (e,f) (*cặp ban đầu đẻ tiếp*)
  - tháng 5: 5 cặp (a,b) (c,d) (e,f) (g,h) (i,k) (*cả cặp (a,b) và (c,d) đều đẻ*)
- **Công thức đệ quy:**
  - $F(1) = 1$                        $F(2) = 1$
  - $F(n) = F(n-1) + F(n-2)$

# Cài đặt giải thuật Fibonacci

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

## 4. Đệ quy quay lui (back tracking) và bài toán 8 quân hậu

- **Đặt vấn đề:** Một bàn cờ quốc tế là một bảng hình vuông gồm có 8 hàng, 8 cột. Quân hậu là một quân cờ có thể ăn bất kỳ quân nào nằm trên cùng một hàng, cùng một cột hay cùng một đường chéo.
- **Bài toán:** Hãy xếp 8 quân hậu trên bàn cờ sao cho không có quân hậu nào có thể ăn được quân hậu nào. Nghĩa là trên mỗi hàng, mỗi cột, mỗi đường chéo chỉ có 1 quân hậu mà thôi.

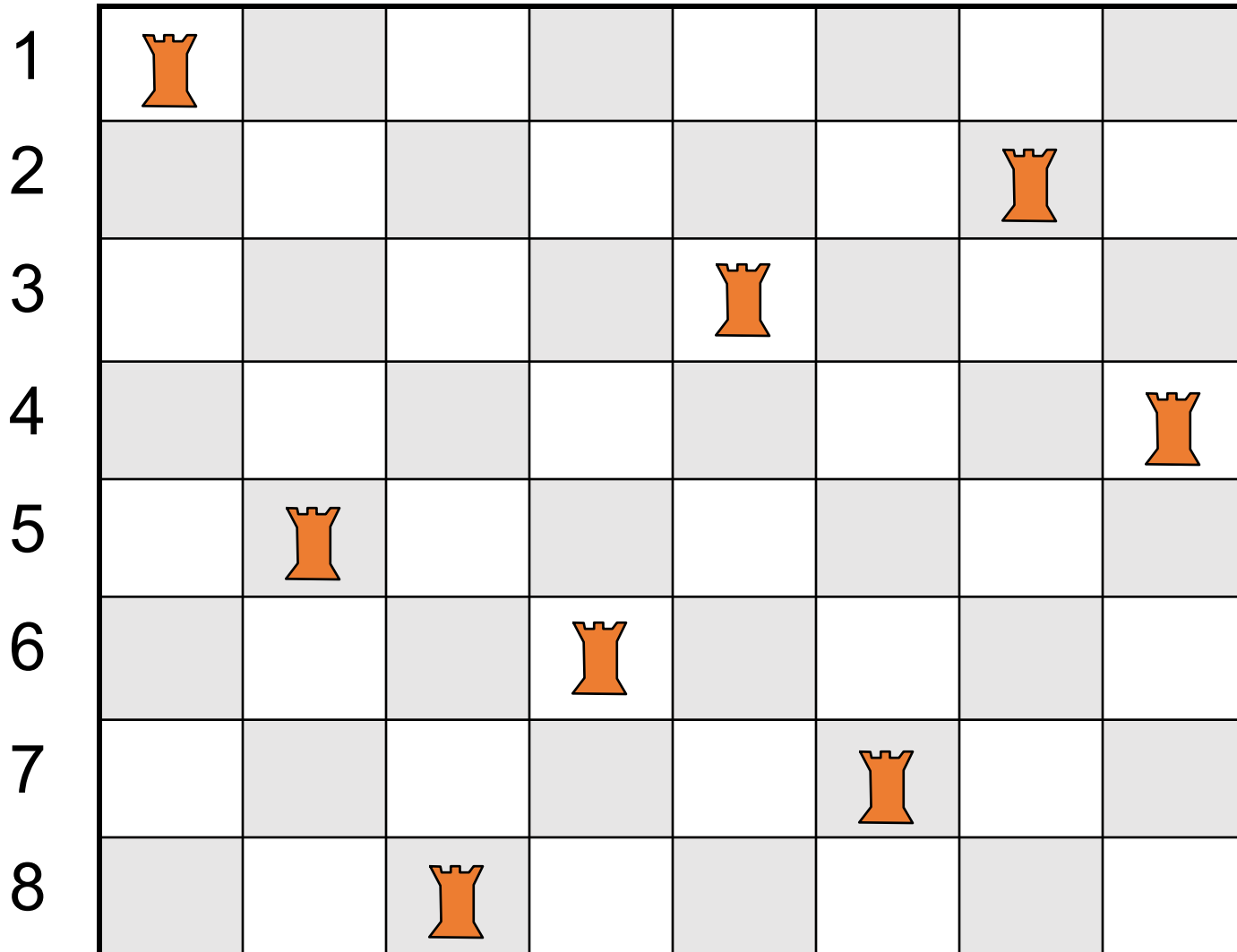
## 4. Độ quy quay lui (back tracking) và bài toán 8 quân hậu (2)

- Phác thảo giải thuật:

- Có bàn cờ có 8 cột, ta có 8 quân hậu → mỗi quân hậu nằm trên 1 cột.
- Do mỗi cột chỉ có thể có 1 quân hậu nên lựa chọn đối với quân hậu thứ  $j$ , ứng cột  $j$ , là đặt nó vào hàng nào để đảm bảo an toàn, nghĩa là không cùng hàng, cùng cột hay cùng đường chéo  $(j-1)$  với bất kỳ quân hậu nào trước đó

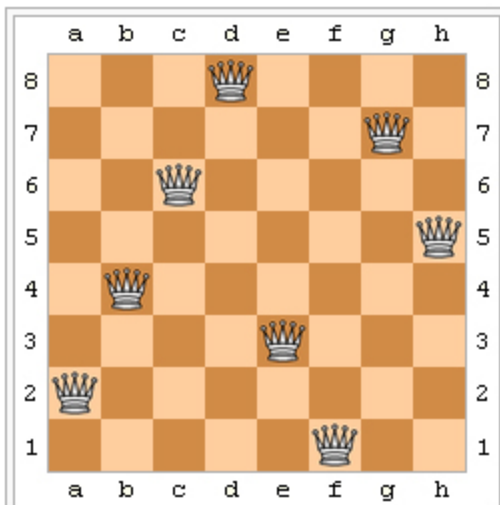
# Kết quả đầu tiên thu được:

1 2 3 4 5 6 7 8

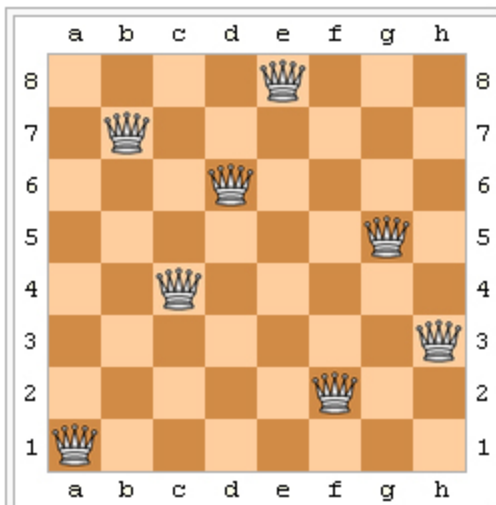




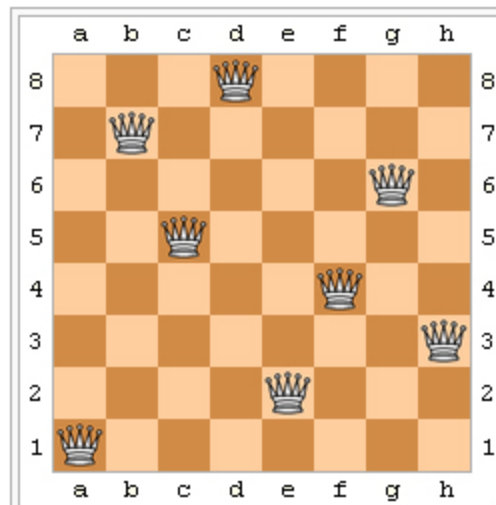
Có tất cả 92 lời giải, nếu không phân biệt các lời giải là ảnh của nhau qua phép đối xứng, phép quay bàn cờ thì chúng chỉ có 12 lời giải **đơn vị** như biểu diễn dưới đây:



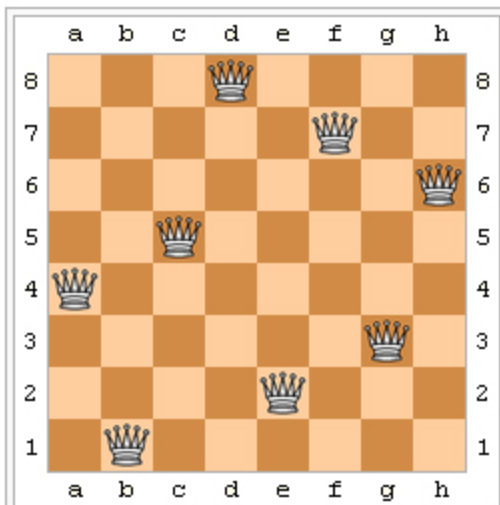
Lời giải 1



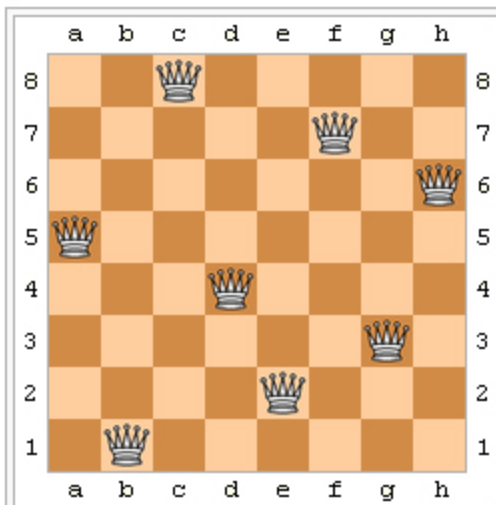
Lời giải 2



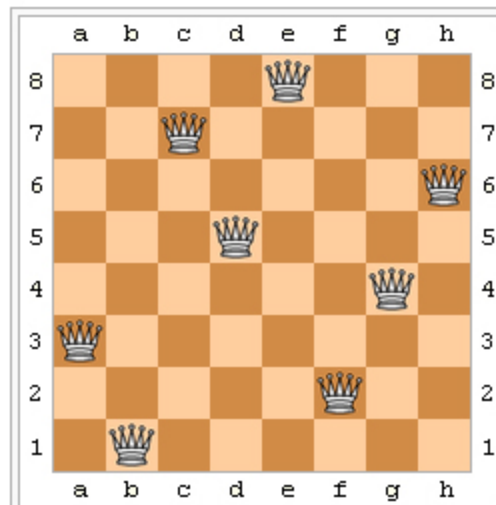
Lời giải 3



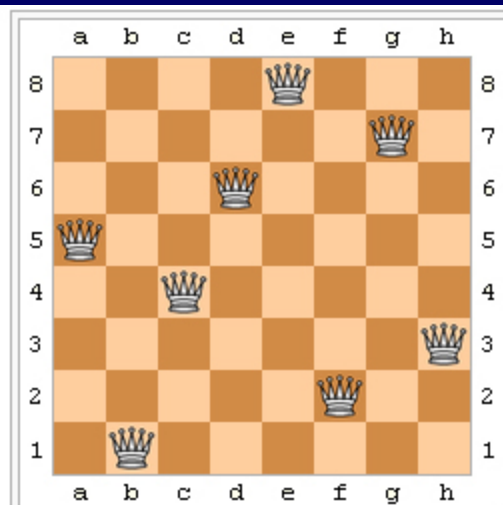
Lời giải 4



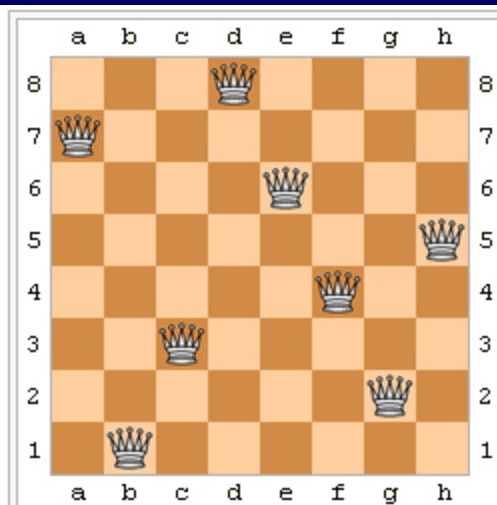
Lời giải 5



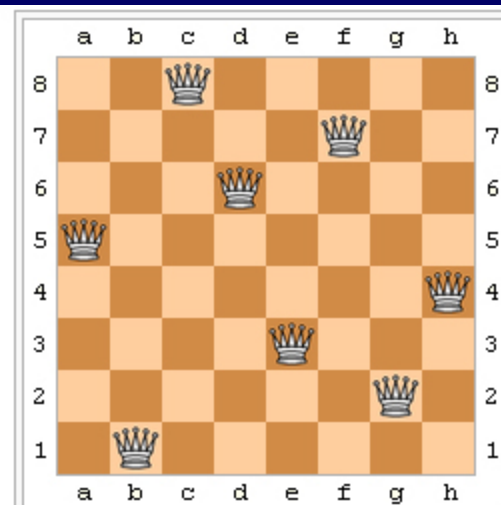
Lời giải 6



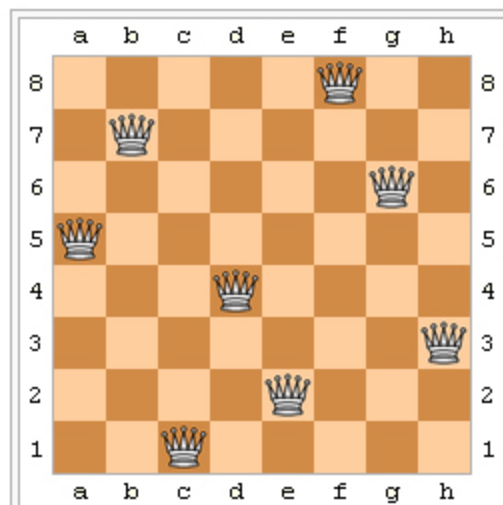
Lời giải 7



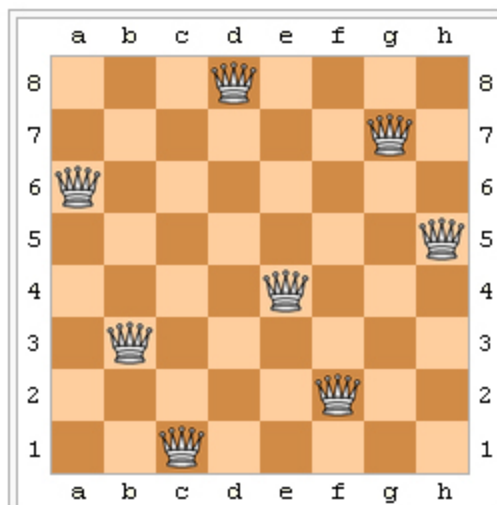
Lời giải 8



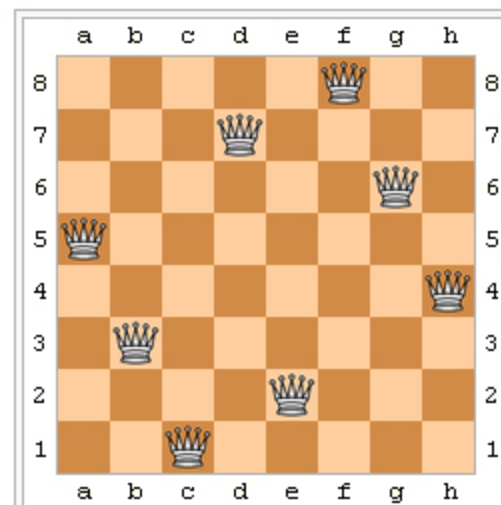
Lời giải 9



Lời giải 10



Lời giải 11



Lời giải 12

## 4. Độ quy quay lui (back tracking) và bài toán 8 quân hậu (3)

- Vậy nét đặc trưng cơ bản của phương pháp này là các bước đi tới lời giải hoàn toàn làm thử.
  - Nếu có một lựa chọn chấp nhận được thì ghi nhớ các thông tin cần thiết và tiến hành các bước thử tiếp theo.
  - Nếu không có một lựa chọn nào chấp nhận được thì làm lại bước trước, xóa bớt ghi nhớ và quay về chu trình thử với các lựa chọn còn lại.
  - Hành động này gọi là quay lui.
  - Giải thuật thể hiện tư tưởng này là giải thuật quay lui.

# Giải thuật bài toán 8 quân hậu

```
void try(j)
{Khởi tạo vị trí ban đầu cho quân hậu thứ j;
do
{ Chọn vị trí đặt quân hậu;
  if (an toàn)
  { đặt quân hậu;
    if (j<8) try(j+1) ;
    if (không thành công)
      cất quân hậu;
  }
}while (thành công) hoặc (hết chỗ) ;
}
```

# Tình hình giải thuật bài toán 8 quân hậu

- Gọi  $i$  vị trí đặt quân hậu thứ  $j$ :  $i \rightarrow [1..8]$
- An toàn???
  - Không có quân hậu nào đặt trên cùng dòng, cùng đường chéo lên và đường chéo xuống trên bàn cờ.
- **Đặt quân hậu:** đánh dấu dòng  $i$  và 2 đường chéo tương ứng đã có quân hậu
- **Cất quân hậu:** trả lại giá trị đã đánh dấu
- Không thành công???
  - Không tìm được vị trí đặt quân hậu

## 5. Hiệu lực của đệ quy

- Là công cụ rất mạnh để giải bài toán
- Có nhiều thuật toán giải bằng đệ quy nhanh hơn nhiều không đệ quy
- Nhiều giải thuật rất dễ mô tả dạng đệ quy nhưng lại rất khó mô tả với giải thuật không-đệ-quy.
- Hàm đệ quy kém hiệu quả vì: tốn bộ nhớ và gọi hàm quá nhiều lần. Tuy nhiên viết hàm đệ quy rất ngắn gọn
- **HÀM ĐỆ QUY: Vừa tốn bộ nhớ vừa chạy chậm**