

NHẬP MÔN LẬP TRÌNH

CHƯƠNG 5.2: HÀM ĐỆ QUY RECURSION FUNCTION

ThS. Nguyễn Thị Ngọc Diễm
diemntn@uit.edu.vn



Nội dung

1. Tổng quan về đệ quy
2. Các vấn đề đệ quy thông dụng
3. Phân tích giải thuật & khử đệ quy
 - a. Hệ thức Truy hồi (Retrieval)
 - b. Chia để trị (Divide and conquer)
 - c. Lăn ngược (backtracking)
4. Các bài toán kinh điển
 - a. Tháp Hà Nội
 - b. Tám hậu
 - c. Mã đi tuần
 - d. Phát sinh hoán vị



1. Tổng quan về đệ quy

- Cho $S(n) = 1 + 2 + 3 + \dots + n$
 $\Rightarrow S(10)? S(11)?$

$$S(10) = 1 + 2 + \dots + 10 = 55$$

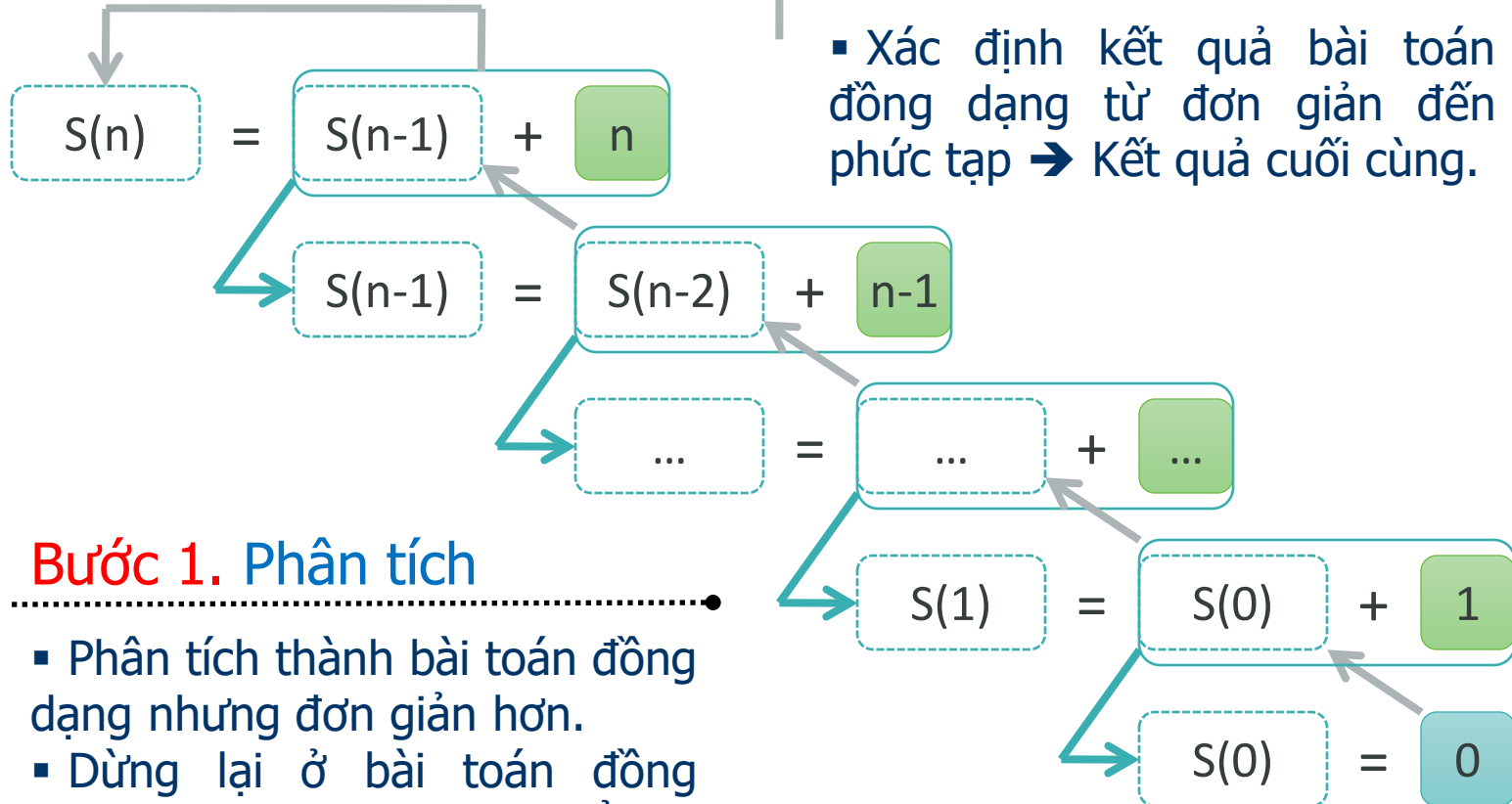
$$\begin{aligned} S(11) &= 1 + 2 + \dots + 10 + 11 = 66 \\ &= S(10) + 11 \\ &= 55 + 11 = 66 \end{aligned}$$



2 bước giải bài toán

Bước 2. Thể ngược

- Xác định kết quả bài toán đồng dạng từ đơn giản đến phức tạp → Kết quả cuối cùng.



Bước 1. Phân tích

- Phân tích thành bài toán đồng dạng nhưng đơn giản hơn.
- Dừng lại ở bài toán đồng dạng đơn giản nhất có thể xác định ngay kết quả.



Thuật ngữ

- **Recursion** – Đệ quy
- **Recursive** – Tính đệ quy.
- **Recursive problem** – vấn đề đệ quy



❖ Khái niệm

Một vấn đề mang tính đệ quy nếu như nó có thể được giải quyết thông qua kết quả của chính vấn đề đó nhưng với đầu vào đơn giản hơn.

❖ Ví dụ

Tổng $S(n)$ được tính thông qua tổng $S(n-1)$.

❖ 2 điều kiện quan trọng

- Tồn tại bước đệ quy.
- Điều kiện dừng.



Điều kiện dừng

- Trường hợp cơ bản – **base case** – là một input đủ nhỏ để ta có thể giải quyết vấn đề mà không cần lời gọi đệ quy.

$$n! = n \times (n - 1)!$$

$$\rightarrow 1! = 1 \times 0! \quad \rightarrow 0! = ? ? ? ?$$

$$S(n) = n + S(n - 1)$$

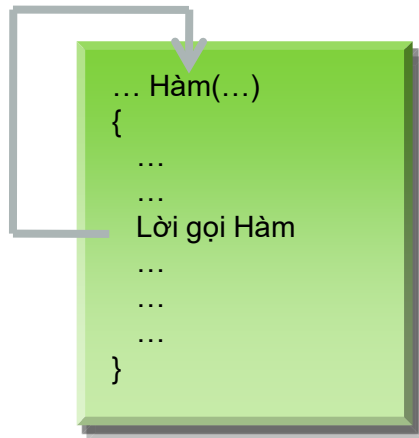
$$\rightarrow S(2) = 1 + S(1) \rightarrow S(1) = ? ? ? ?$$



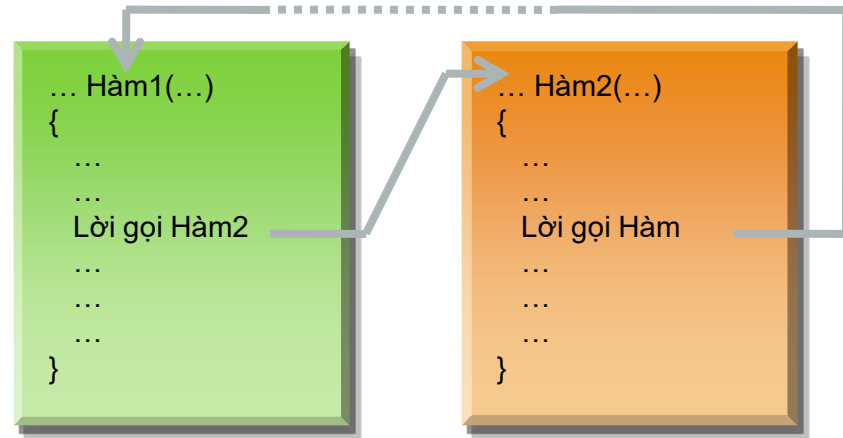
Khái niệm hàm đệ quy

- Khái niệm

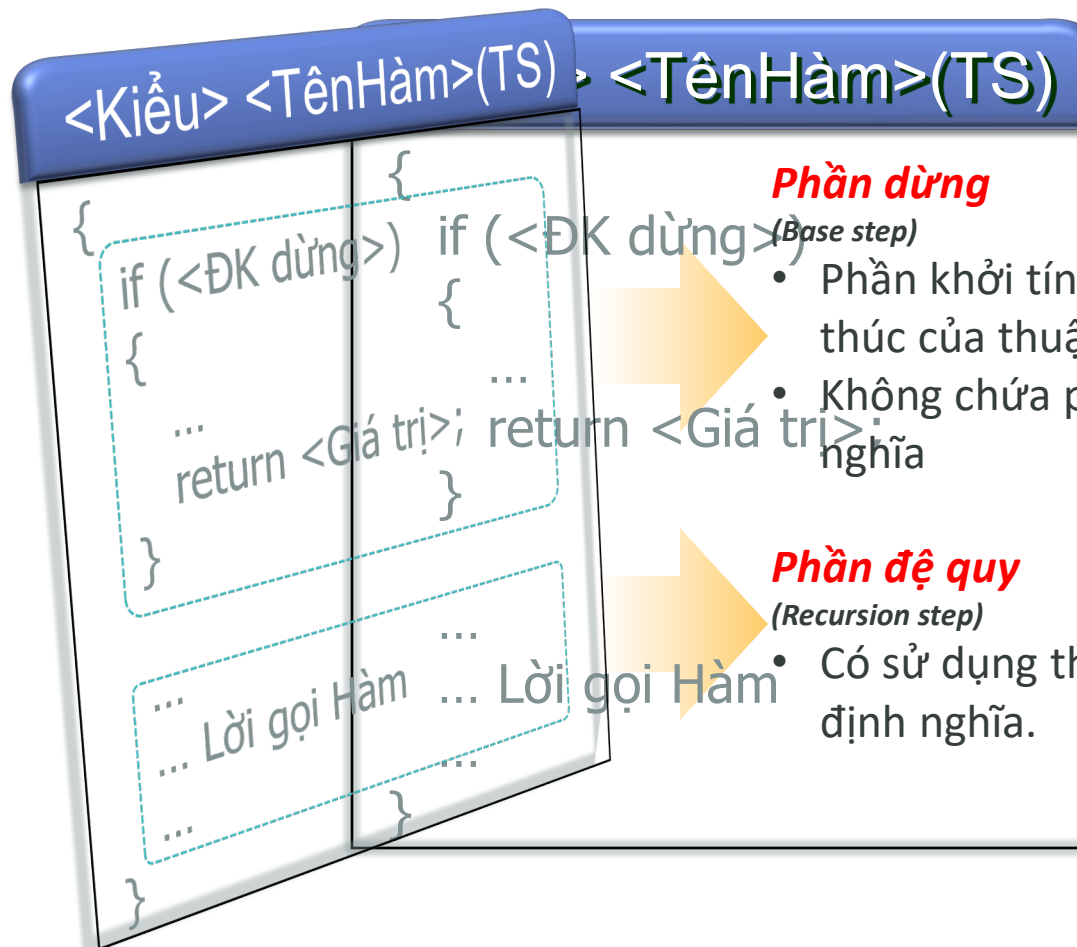
- Một hàm được gọi là đệ quy nếu bên trong thân của hàm đó có lời gọi hàm lại chính nó một cách trực tiếp hay gián tiếp.



ĐQ trực tiếp

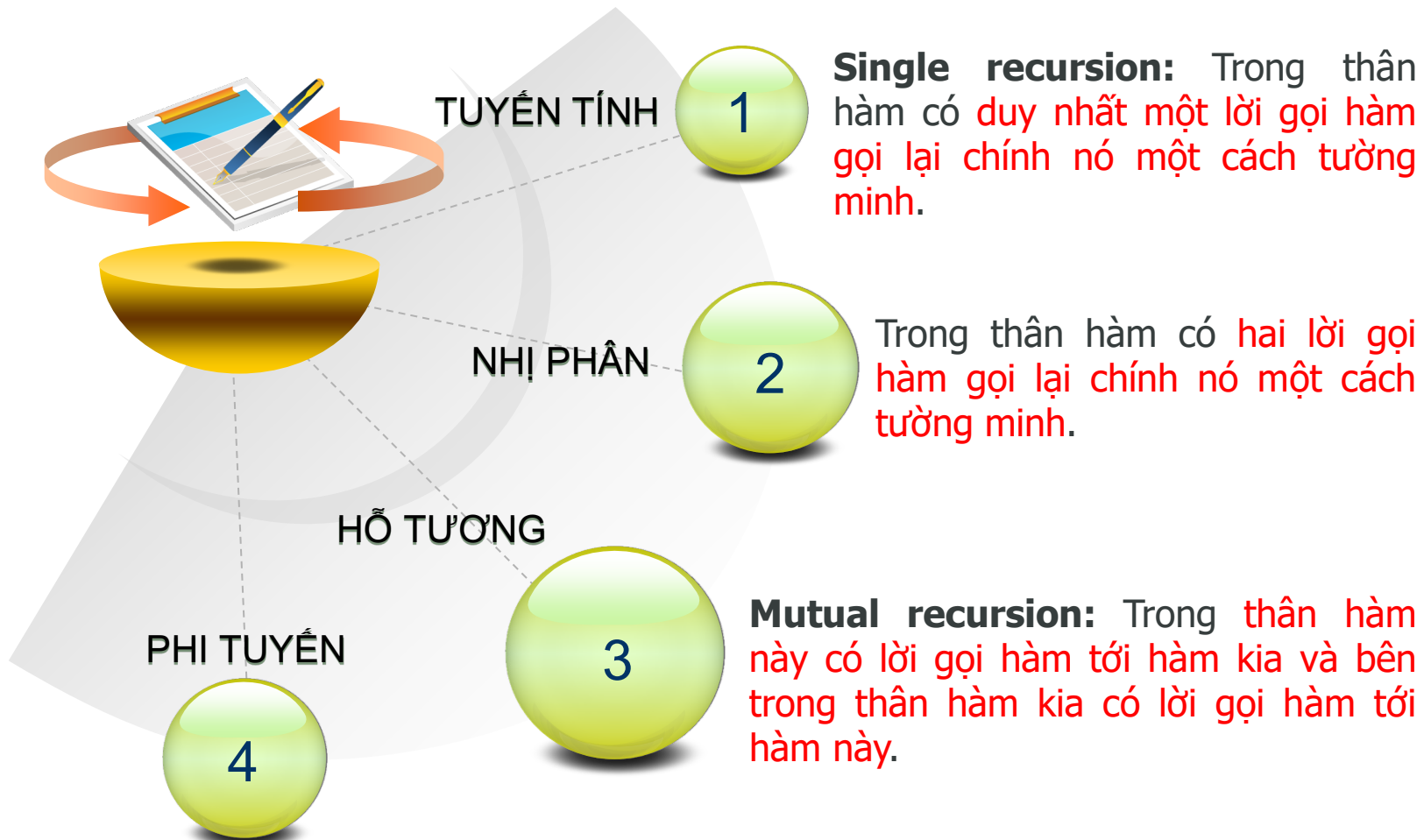


ĐQ gián tiếp

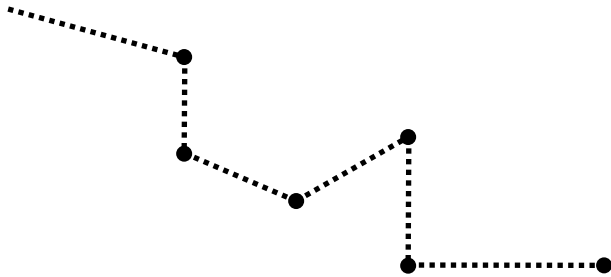




2. Các vấn đề đệ quy thông dụng



Multiple recursion: Trong thân hàm có lời gọi hàm lại chính nó được đặt bên trong thân vòng lặp.



Cấu trúc chương trình

```
<Kiểu> TênHàm(<TS>) {  
    if (<ĐK dừng>) {  
        ...  
        return <Giá Trị>;  
    }  
    ... TênHàm(<TS>); ...  
}
```

Ví dụ


```
int giai_thua(int n){  
    if (n == 0) return 1;  
    else return n * giai_thua(n -  
1);  
}  
  
int uscln(int a, int b){  
    if ((!a) || (!b)) return a+b;  
    if (a>b) return uscln(a - b, b);  
    return uscln(a, b - a);  
}
```




Đệ quy tuyến tính

- Đệ quy tuyến tính rất dễ chuyển sang vòng lặp có chức năng tương đương → Khử đệ quy

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



```
int giai_thua(int n){  
    int kq = 1;  
    for (int i = 1; i <= n; i++){  
        kq = kq * i;  
    }  
    return kq;  
}
```





Đệ quy tuyến tính

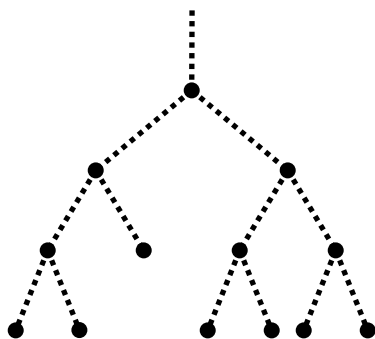
- Dạng vòng lặp thường chạy nhanh hơn đệ quy, dùng ít bộ nhớ hơn
→ chạy được input lớn hơn

```
int sum(int n){  
    if (n > 0)  
        return n + tong(n - 1);  
    else return 0;  
}
```

```
int sum_2(int n){  
    int i, kq = 0;  
    for (i = 1; i <= n; i++){  
        kq = kq + i;  
    }  
    return kq;  
}
```

```
int main(){  
    for(int i = 0; i < 1000; i++){  
        // cout << sum(100000) << endl;  
        cout << sum_2(100000) << endl;  
    }  
}
```

⇒ 0.5s
⇒ 0.25s



Cấu trúc chương trình

```
<Kiểu> TênHàm(<TS>) {  
    if (<ĐK dừng>) {  
        ...  
        return <Giá trị>;  
    }  
    ... TênHàm(<TS>);  
    ...  
    ... TênHàm(<TS>);  
    ...  
}
```

Ví dụ

Tính số hạng thứ n của dãy Fibonacci:

$$F(n) = \begin{cases} 0 & , n = 0 \\ 1 & , n = 1 \\ F(n - 1) + F(n - 2), & n \geq 2 \end{cases}$$

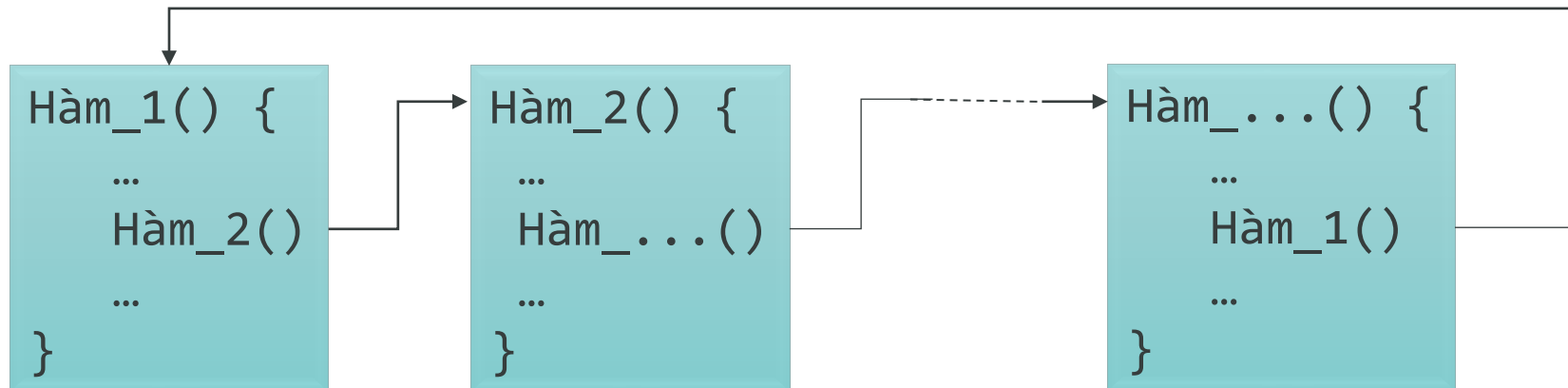
∴ Chương trình ∴

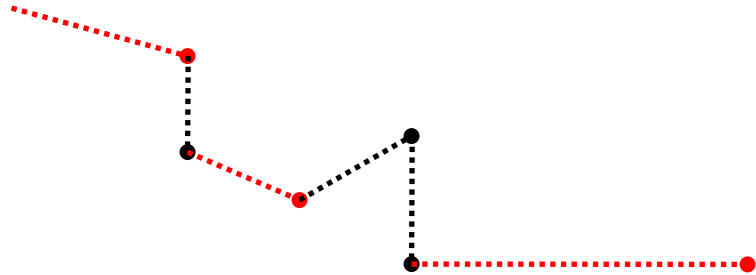
```
int fibonacci(int n){  
    if (n==0) return 0;  
    if (n==1) return 1;  
    return fibonacci(n - 1) +  
    fibonacci(n - 2);  
}
```



Đệ quy hỗ tương

- Đệ quy hỗ tương – mutual recursion. Còn gọi đệ quy gián tiếp – indirect recursion.
- Hàm không trực tiếp gọi lại chính nó mà gọi thông qua một (hoặc nhiều) hàm khác.





Cấu trúc chương trình

```
<Kiểu> TênHàm1(<TS>) {  
    if (<ĐK dừng>)  
        return <Giá trị>;  
    ... TênHàm2(<TS>); ...  
}  
  
<Kiểu> TênHàm2(<TS>) {  
    if (<ĐK dừng>)  
        return <Giá trị>;  
    ... TênHàm1(<TS>); ...  
}
```

Ví dụ

Tính số hạng thứ n của dãy:

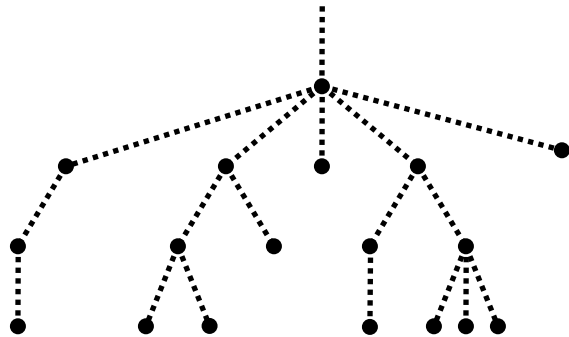
$$x(0) = 1, y(0) = 0$$

$$x(n) = x(n-1) + y(n-1)$$

$$y(n) = 3*x(n-1) + 2*y(n-1)$$

∴ Chương trình ∴

```
long yn(int n);  
long xn(int n) {  
    if (n == 0) return 1;  
    return xn(n-1)+yn(n-1);  
}  
long yn(int n) {  
    if (n == 0) return 0;  
    return 3*xn(n-1)+2*yn(n-1);  
}
```

Cấu trúc chương trình

```
<Kiểu> TênHàm(<TS>) {  
    if (<ĐK dừng>) {  
        ...  
        return <Giá Trị>;  
    }  
    ... Vòng lặp {  
        ... TênHàm(<TS>);  
        ...  
    }  
    ...  
}
```

Ví dụ

Tính số hạng thứ n của dãy sau:

$$x_0 = 1$$

$$x_n = n^2 \cdot x_0 + (n-1)^2 \cdot x_1 + \dots + 2^2 \cdot x_{n-2} + 1^2 \cdot x_{n-1}$$

∴ Chương trình ∴

```
long xn(int n) {  
    if (n == 0) return 1;  
  
    long s = 0;  
  
    for (int i=1; i<=n; i++)  
        s = s + i*i*xn(n-i);  
  
    return s;  
}
```



Các bước xây dựng hàm đệ quy

Thông số hóa
bài toán

- Tổng quát hóa bài toán cụ thể thành bài toán tổng quát.
- Thông số hóa cho bài toán tổng quát
- VD: n trong hàm tính tổng $S(n)$, ...

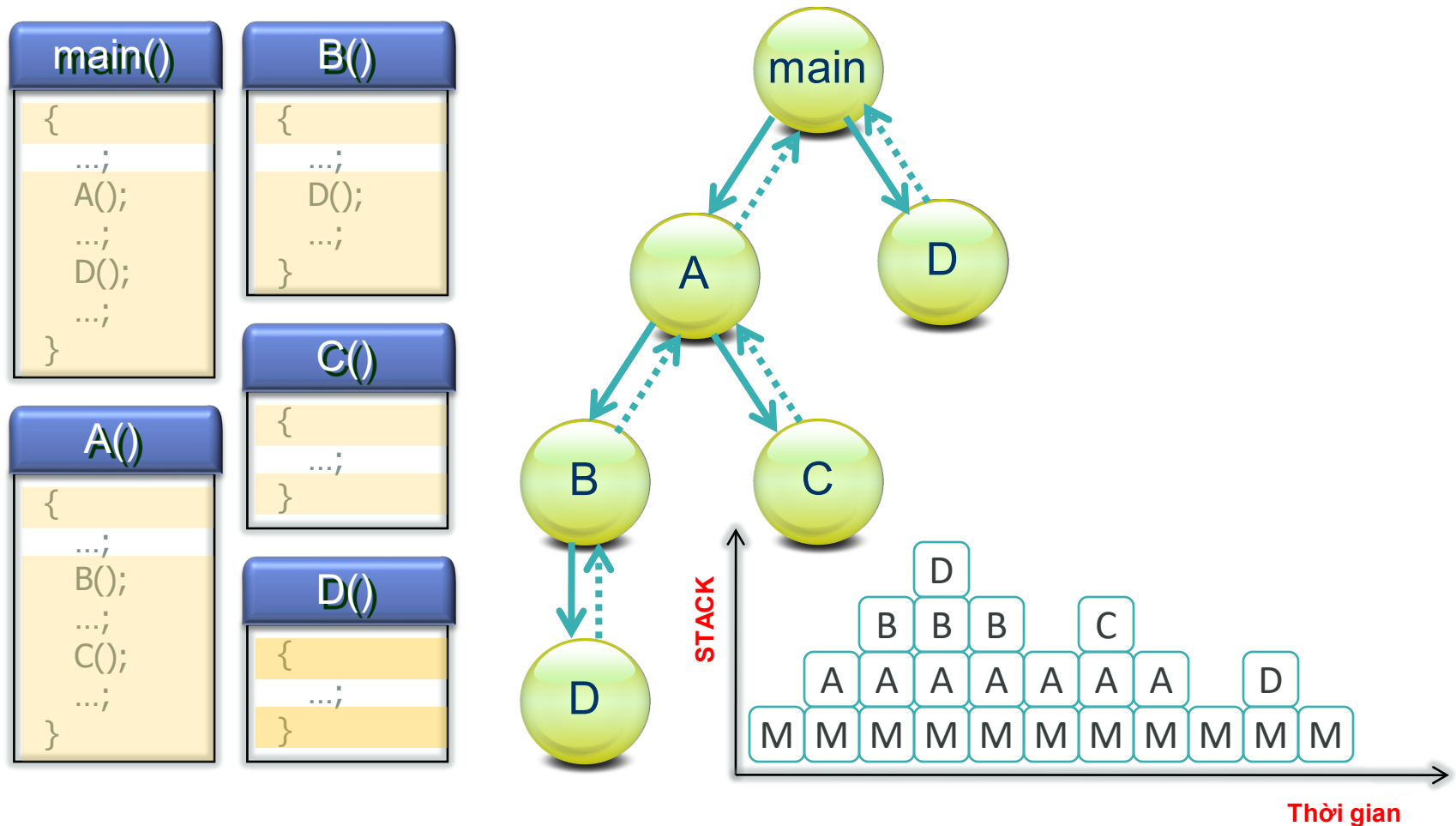
Tìm thuật giải
tổng quát

- Chia bài toán tổng quát ra thành:
 - Phần không đệ quy.
 - Phần như bài toán trên nhưng kích thước nhỏ hơn.
- VD: $S(n) = S(n - 1) + n$, ...

Tìm các trường
hợp suy biến

- Các trường hợp suy biến của bài toán.
- Kích thước bài toán trong trường hợp này là nhỏ nhất.
- VD: $S(0) = 0$

Cơ chế gọi hàm và STACK





- Cơ chế gọi hàm dùng STACK trong C++ phù hợp cho giải thuật đệ quy vì:
 - Lưu thông tin trạng thái còn dở dang mỗi khi gọi đệ quy.
 - Thực hiện xong một lần gọi cần khôi phục thông tin trạng thái trước khi gọi.
 - Lệnh gọi cuối cùng sẽ hoàn tất đầu tiên.

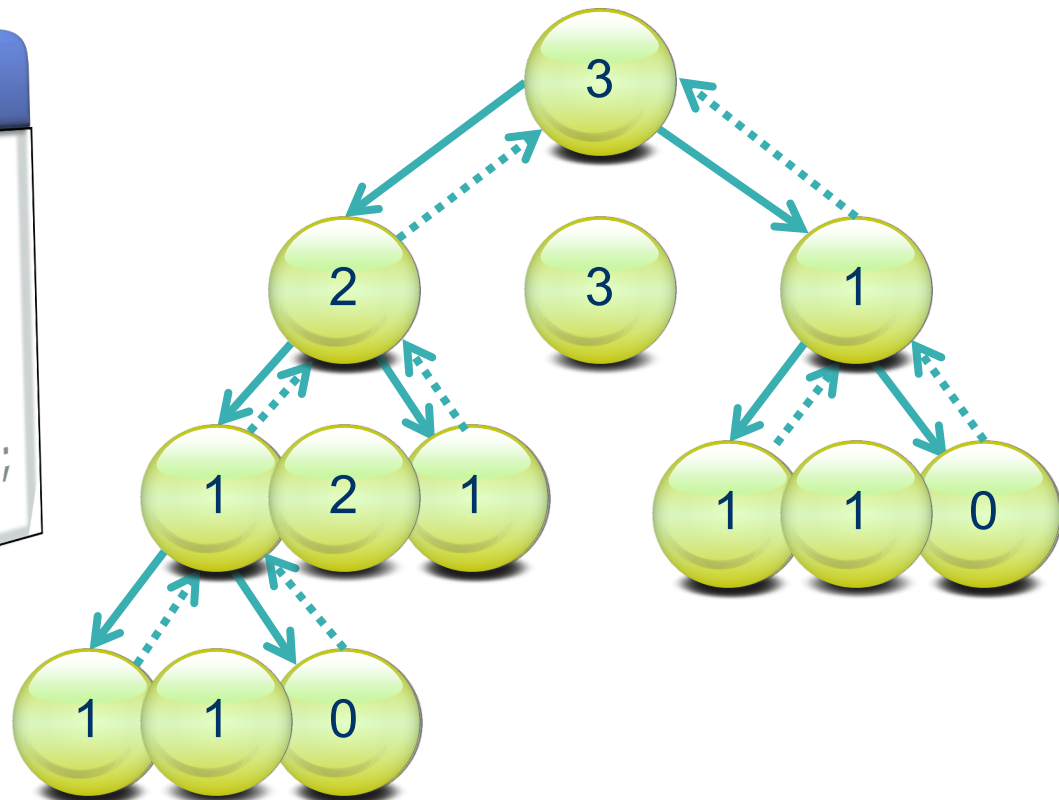


Ví dụ gọi hàm đệ quy

- Tính số hạng thứ 5 của dãy **Fibonacci**

F(0)	F(1)	F(2)	F(3)	F(4)	F(5)	F(6)	F(7)	F(8)	...
0	1	1	2	3	5	8	13	21	...

```
long F(int n)
{
    if (n == 0)
        return 0;
    if (n == 1)
        return 1;
    return F(n-1) + F(n-2);
}
```





Ví dụ gọi hàm đệ quy

- Tính số hạng thứ 5 của dãy **Fibonacci**

```
long F(int n)
{
    if (n == 0)
        return 0;
    if (n == 1)
        return 1;
    return F(n-1) + F(n-2);
}
```

				F(1)		F(0)										
			F(2)	F(2)	F(2)	F(2)	F(2)				F(1)		F(0)			
		F(3)	F(3)	F(3)	F(3)	F(3)	F(3)	F(3)		F(2)	F(2)	F(2)	F(2)	F(2)		
	F(4)	F(4)	F(4)	F(4)	F(4)	F(4)	F(4)	F(4)	F(4)	F(4)	F(4)	F(4)	F(4)	F(4)	F(4)	
main	main	main	main	main	main	main	main	main	main	main	main	main	main	main	main	main



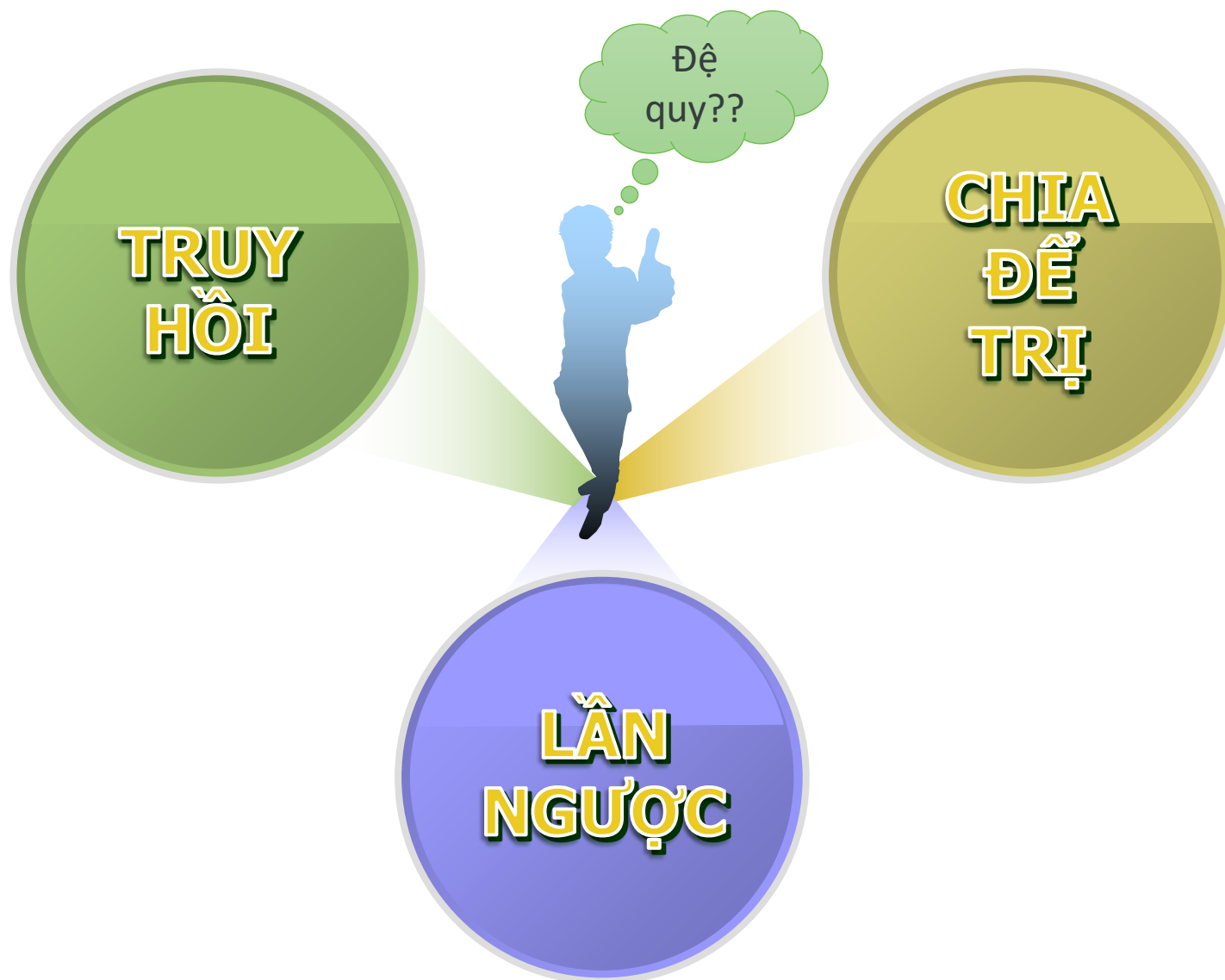
Một số lỗi thường gặp

- Công thức đệ quy chưa đúng, không tìm được bài toán đồng dạng đơn giản hơn (không hội tụ) nên không giải quyết được vấn đề.
- Không xác định các trường hợp suy biến (điều kiện dừng).
- Thông điệp thường gặp là **StackOverflow** :
 - Thuật giải đệ quy đúng nhưng số lần gọi đệ quy quá lớn làm tràn STACK.
 - Thuật giải đệ quy sai do không hội tụ hoặc không có điều kiện dừng.



- Làm lại các bài tập chỉ có 01 vòng lặp mà không dùng các từ khóa: for, while, do, goto
 1. Tính tổng các chữ số của số nguyên dương n
 2. Đếm số lượng chữ số của số nguyên dương n
 3. Tính giá trị của x lũy thừa y
 4. Tính giá trị của $n!$
- Tìm số thứ n của dãy Fibonacci
- Tìm số thứ n của dãy pandovan

3. Các vấn đề đệ quy thông dụng

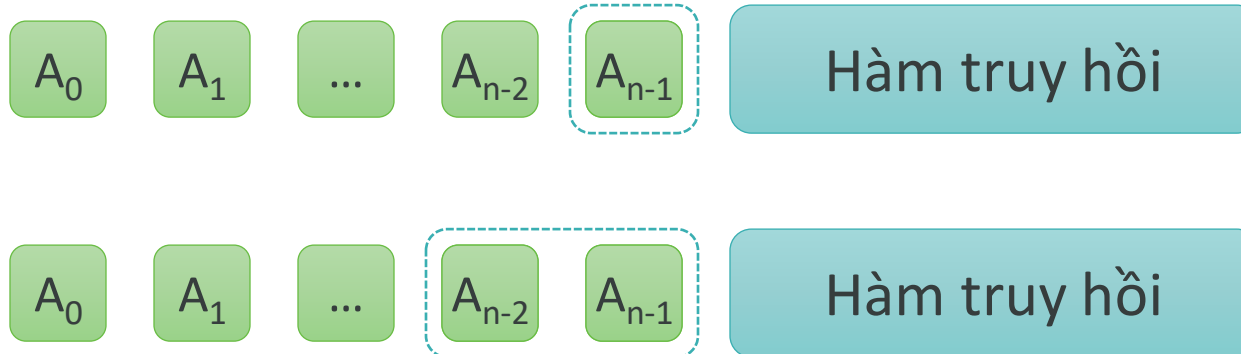




1. Hệ thức truy hồi

- Khái niệm

- Hệ thức truy hồi của 1 dãy A_n là công thức biểu diễn phần tử A_n thông qua 1 hoặc nhiều số hạng trước của dãy.





a. Hệ thức truy hồi

- Ví dụ 1

- Vi trùng cứ 1 giờ lại nhân đôi. Vậy sau 5 giờ sẽ có mấy con vi trùng nếu ban đầu có 2 con?

- Giải pháp

- Gọi V_h là số vi trùng tại thời điểm h .

- Ta có:

- $V_h = 2V_{h-1}$

- $V_0 = 2$

➔ Hệ quy tuyến tính với $V(h) = 2 * V(h-1)$ và điều kiện dừng $V(0) = 2$



a. Hệ thức truy hồi

- Ví dụ 2

- Gửi ngân hàng 1000 USD, lãi suất 12%/năm. Số tiền có được sau 30 năm là bao nhiêu?

- Giải pháp

- Gọi T_n là số tiền có được sau n năm.

- Ta có:

- $T_n = T_{n-1} + 0.12T_{n-1} = 1.12T_{n-1}$

- $V(0) = 1000$

➔ Đệ quy tuyến tính với $T(n)=1.12 \cdot T(n-1)$ và điều kiện dừng $V(0) = 1000$



a. Chia để trị (divide & conquer)

- Khái niệm
 - Chia bài toán thành nhiều bài toán con.
 - Giải quyết từng bài toán con.
 - Tổng hợp kết quả từng bài toán con để ra lời giải.

... Trị(bài toán P)

```
{  
  if (P đủ nhỏ)  
    Xử lý P  
  else  
  {  
    Chia  $P \rightarrow P_1, P_2, \dots, P_n$   
    for (int i=1; i<=n; i++)  
      Trị( $P_i$ );  
    Tổng hợp kết quả  
  }  
}
```



b. Chia để trị (divide & conquer)

- Ví dụ 1

- Cho dãy A đã sắp xếp thứ tự tăng. Tìm vị trí phần tử x trong dãy (nếu có)

- Giải pháp

- $\text{mid} = (l + r) / 2;$

- Nếu $A[\text{mid}] = x \rightarrow$ trả về mid.

- Ngược lại

- Nếu $x < A[\text{mid}] \rightarrow$ tìm trong đoạn $[l, \text{mid} - 1]$

- Ngược lại \rightarrow tìm trong đoạn $[\text{mid} + 1, r]$

- \rightarrow Sử dụng đệ quy nhị phân.



2. Chia để trị (divide & conquer)

- Ví dụ 2

- Tính tích 2 chuỗi số cực lớn X và Y

- Giải pháp

- $X = X_{2n-1} \dots X_n X_{n-1} \dots X_0$, $Y = Y_{2n-1} \dots Y_n Y_{n-1} \dots Y_0$

- Đặt $X_L = X_{2n-1} \dots X_n$, $X_N = X_{n-1} \dots X_0 \Rightarrow X = 10^n X_L + X_N$

- Đặt $Y_L = Y_{2n-1} \dots Y_n$, $Y_N = Y_{n-1} \dots Y_0 \Rightarrow Y = 10^n Y_L + Y_N$

- $\Rightarrow X * Y = 10^{2n} X_L Y_L + 10^n (X_L Y_N + X_N Y_L) + X_N Y_N$

- và $X_L Y_L + X_N Y_N = (X_L - X_N)(Y_N - Y_L) + X_L Y_L + X_N Y_N$

\Rightarrow Nhân 3 số nhỏ hơn (độ dài $\frac{1}{2}$) đến khi có thể nhân được ngay.



2. Chia để trị (divide & conquer)

- Một số bài toán khác
 - Bài toán tháp Hà Nội
 - Các giải thuật sắp xếp: QuickSort, MergeSort
 - Các giải thuật tìm kiếm trên cây nhị phân tìm kiếm, cây nhị phân nhiều nhánh tìm kiếm.
- Lưu ý
 - Khi bài toán lớn được chia thành các bài toán nhỏ hơn mà những bài toán nhỏ hơn này không đơn giản nhiều so với bài toán gốc thì **không nên** dùng kỹ thuật chia để trị.



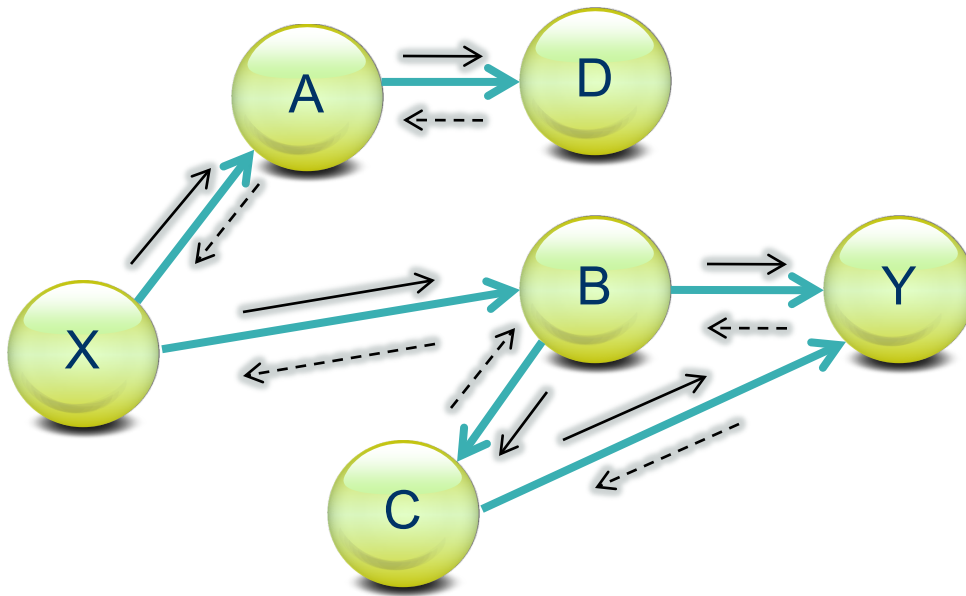
3. Lần ngược (Backtracking)

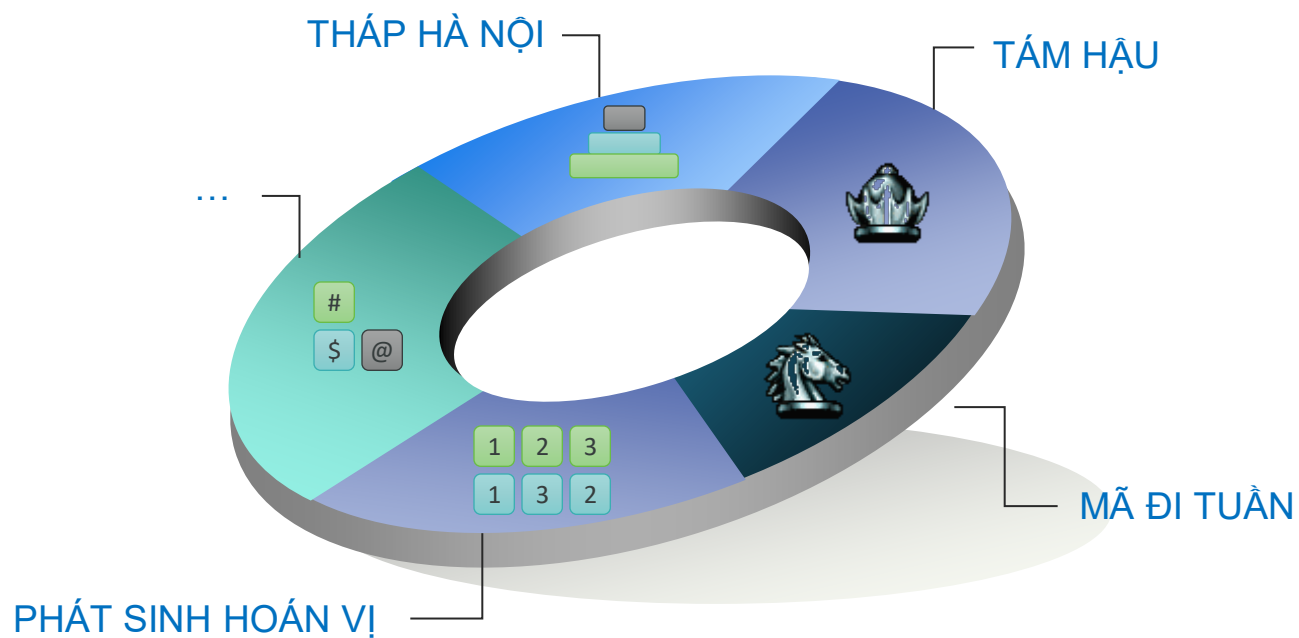
- Khái niệm
 - Tại bước có nhiều lựa chọn, ta chọn thử 1 bước để đi tiếp.
 - Nếu không thành công thì “lần ngược” chọn bước khác.
 - Nếu đã thành công thì ghi nhận lời giải này đồng thời “lần ngược” để truy tìm lời giải mới.
 - Thích hợp giải các bài toán kinh điển như bài toán 8 hậu và bài toán mã đi tuần.



3. Lần ngược (Backtracking)

- Ví dụ
 - Tìm đường đi từ **X** đến **Y**.



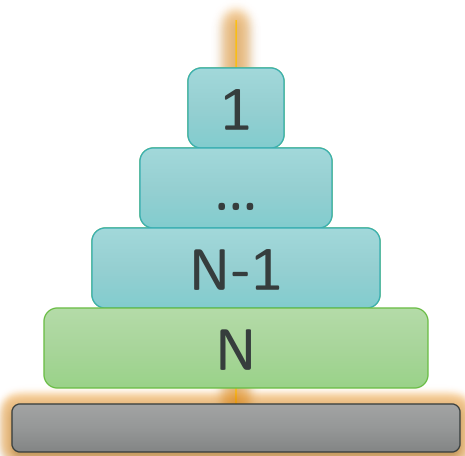




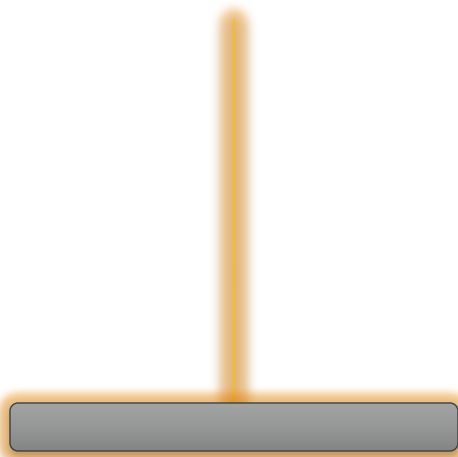
- Mô tả bài toán
 - Có 3 cột A, B và C và cột A hiện có N đĩa.
 - Tìm cách chuyển N đĩa từ cột A sang cột C sao cho:
 - Một lần chuyển 1 đĩa
 - Đĩa lớn hơn phải nằm dưới.
 - Có thể sử dụng các cột A, B, C làm cột trung gian.



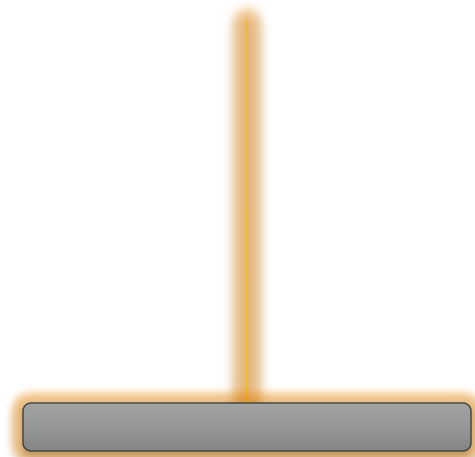
$$N \text{ đĩa } A \rightarrow C = ? \text{ đĩa } A \rightarrow B + \text{Đĩa } N \text{ } A \rightarrow C + N-1 \text{ đĩa } B \rightarrow C$$



Cột nguồn A



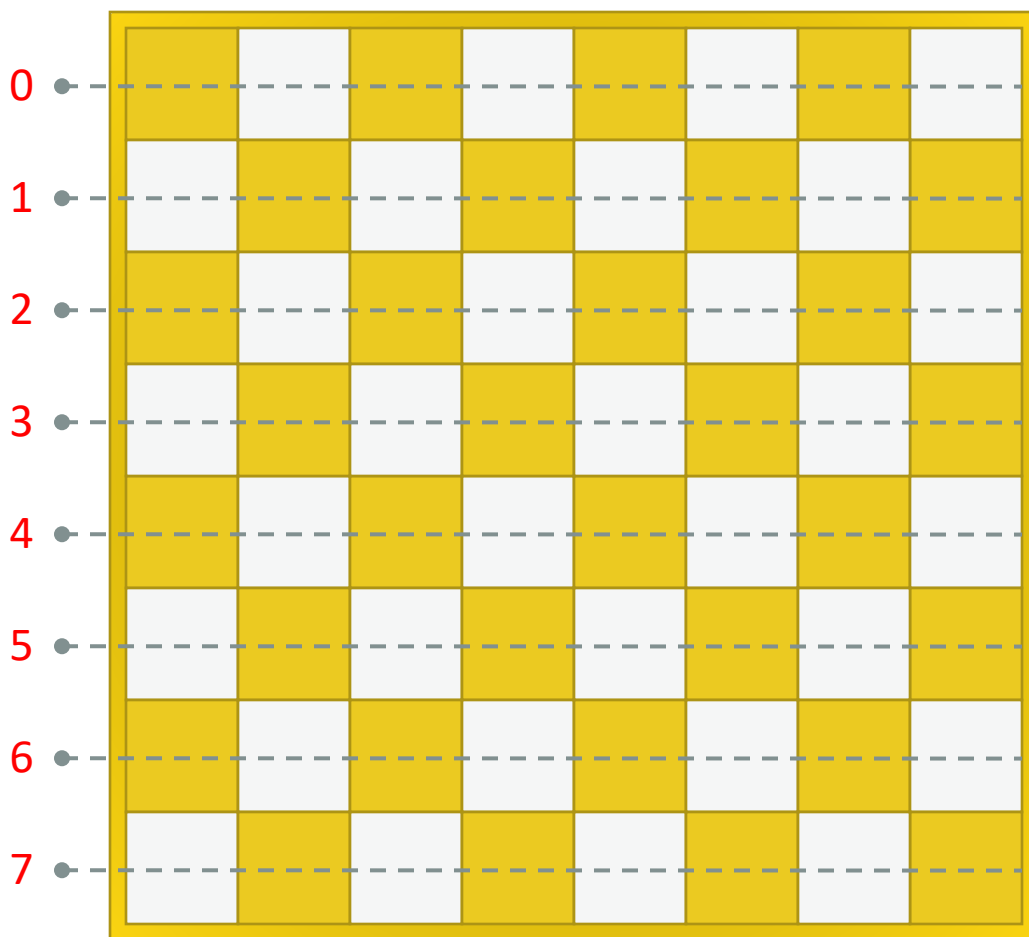
Cột trung gian B

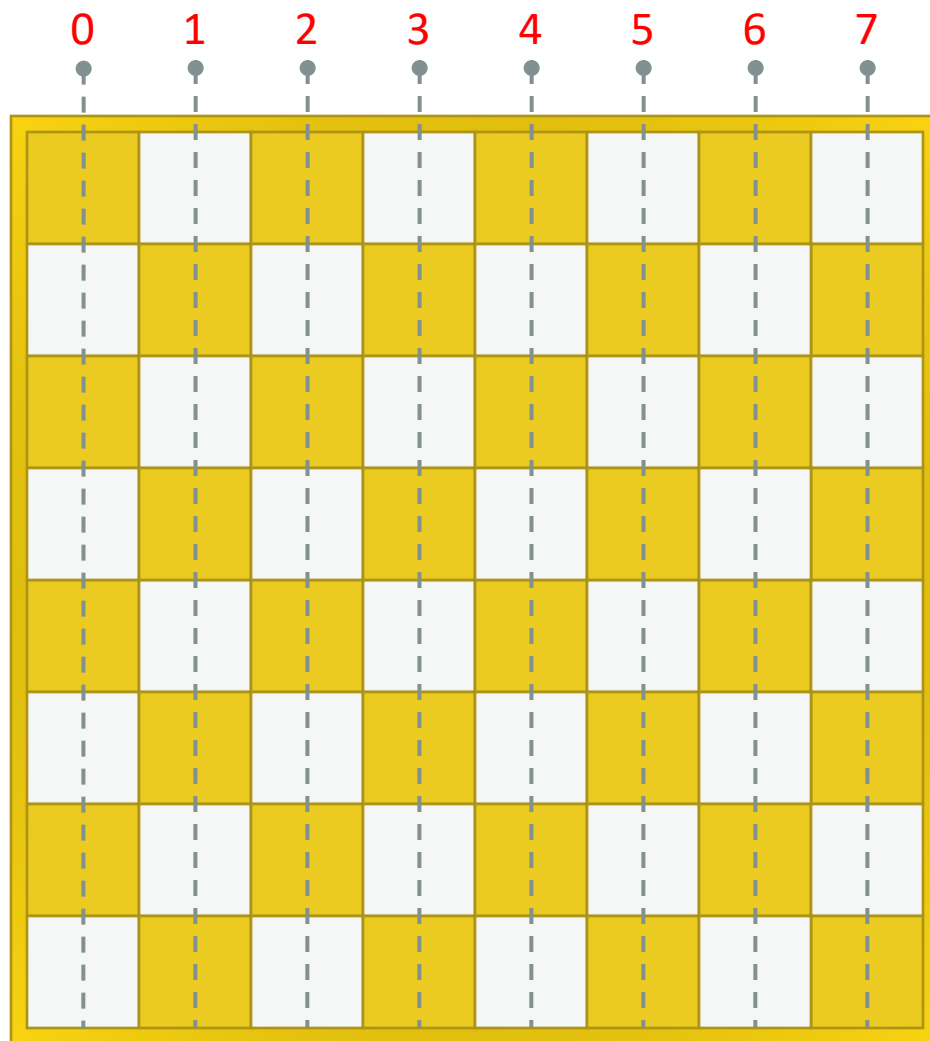


Cột đích C

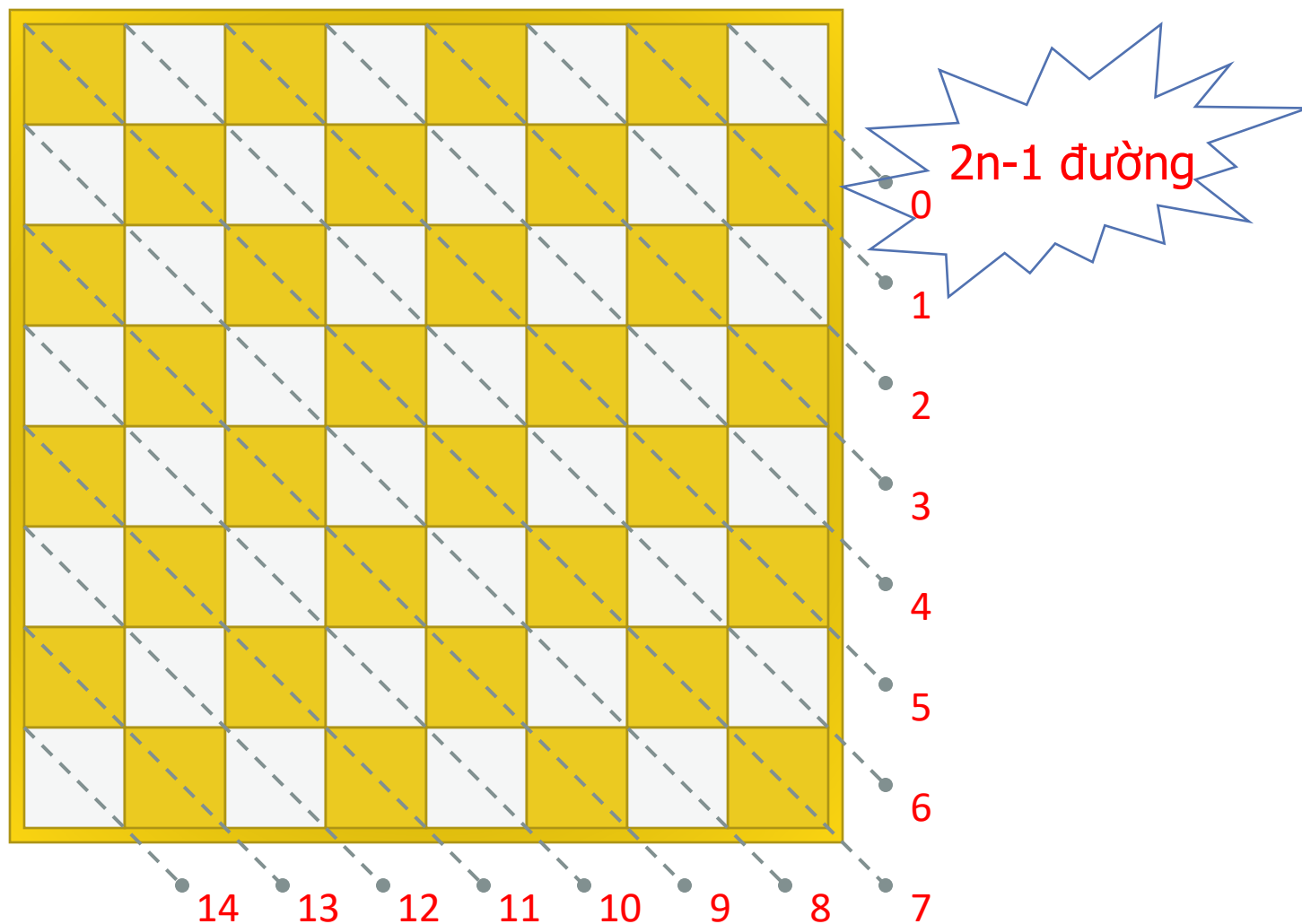


- Mô tả bài toán
 - Cho bàn cờ vua kích thước 8×8
 - Hãy đặt 8 hoàng hậu lên bàn cờ này sao cho không có hoàng hậu nào “ăn” nhau:
 - Không nằm trên cùng dòng, cùng cột
 - Không nằm trên cùng đường chéo xuôi, ngược.

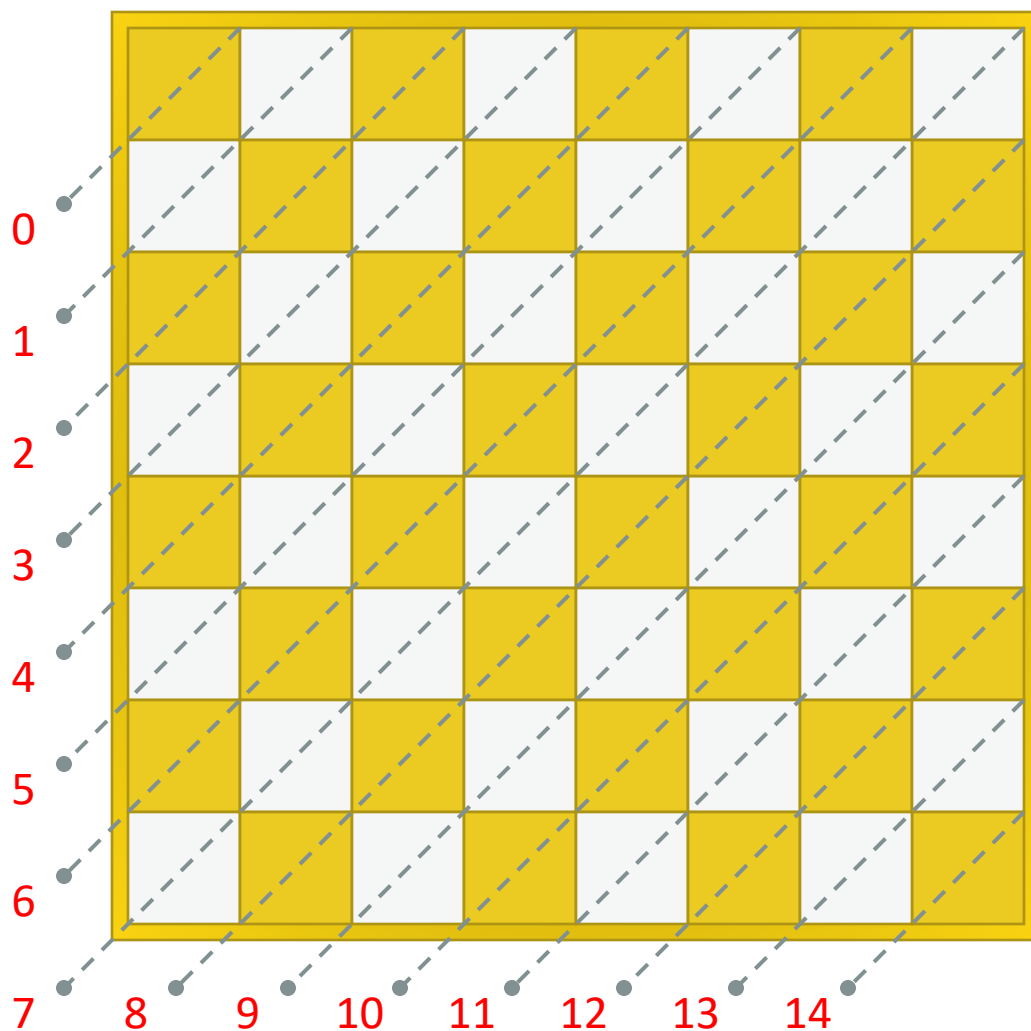




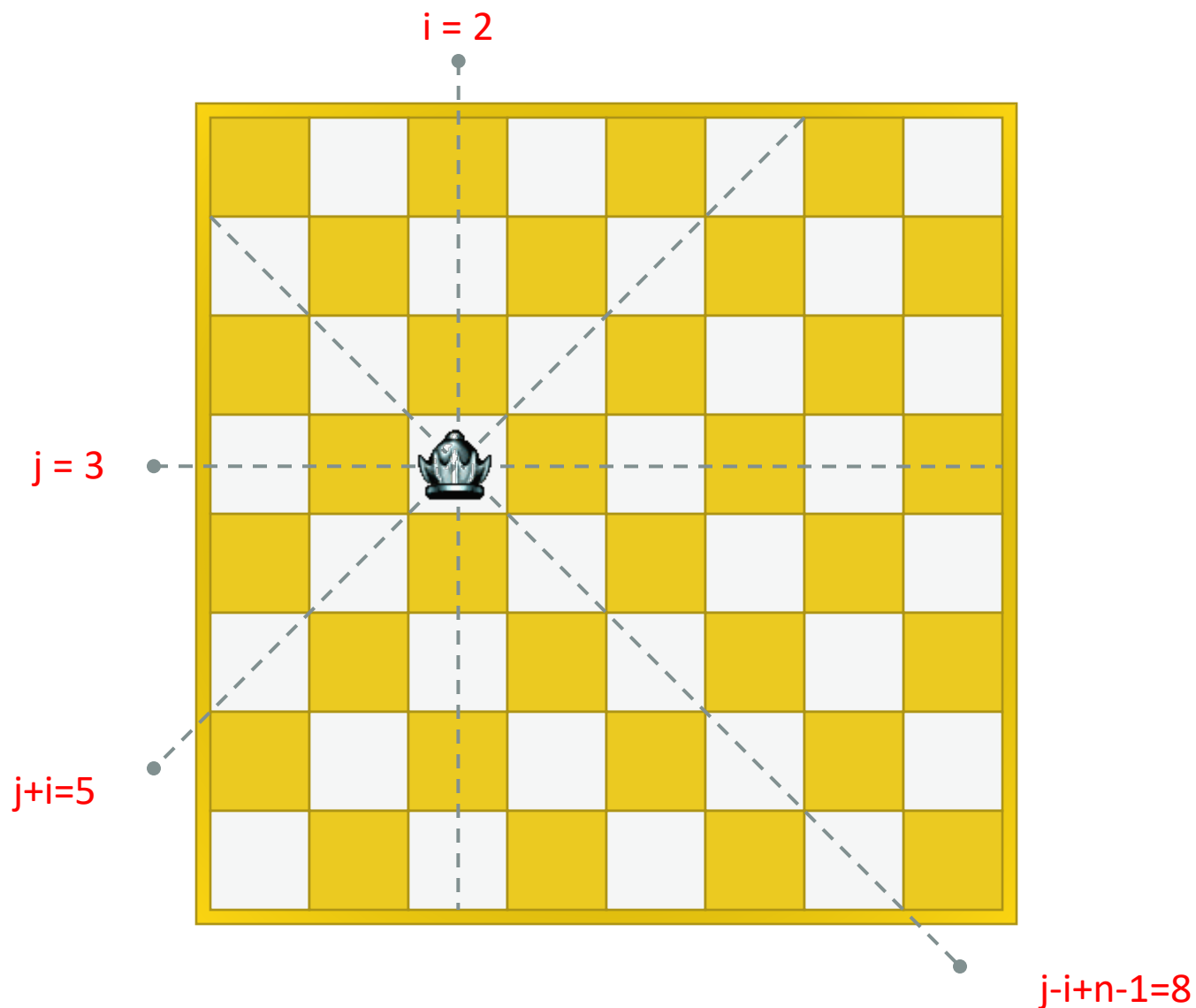
Tám hậu – Các đường chéo xuôi



Tám hậu – Các đường chéo ngược

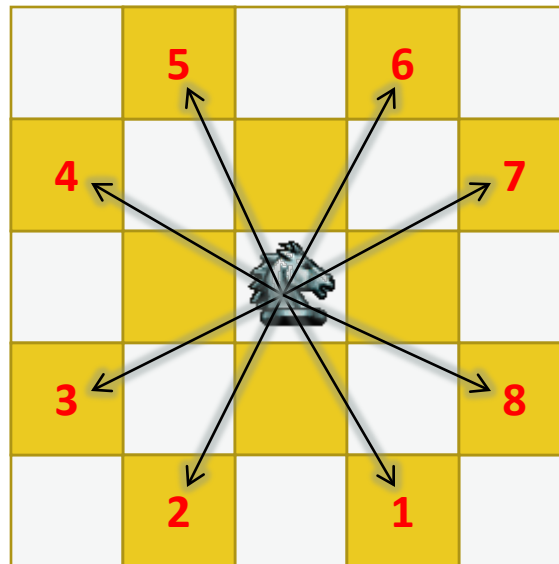


$2n-1$ đường





- Mô tả bài toán
 - Cho bàn cờ vua kích thước 8x8 (64 ô)
 - Hãy đi con mã 64 nước sao cho mỗi ô chỉ đi qua 1 lần (xuất phát từ ô bất kỳ) theo luật:





Phân tích giải thuật đệ quy

- Sử dụng cây đệ quy (recursive tree)
 - Giúp hình dung bước phân tích và thế ngược.
 - Bước phân tích: đi từ trên xuống dưới.
 - Bước thế ngược đi từ trái sang phải, từ dưới lên trên.
- Ý nghĩa
 - Chiều cao của cây \Leftrightarrow Độ lớn trong STACK.
 - Số nút \Leftrightarrow Số lời gọi hàm.



- Ưu điểm

- Sáng sủa, dễ hiểu, nêu rõ bản chất vấn đề.
- Tiết kiệm thời gian thực hiện mã nguồn.
- Một số bài toán rất khó giải nếu không dùng đệ quy.

- Khuyết điểm

- Tốn nhiều bộ nhớ, thời gian thực thi lâu.
- Một số tính toán có thể bị lặp lại nhiều lần.
- Một số bài toán không có lời giải đệ quy.



Khử đệ quy (Tham khảo)

- Khái niệm
 - Đưa các bài toán đệ quy về các bài toán không sử dụng đệ quy.
 - Thường sử dụng vòng lặp hoặc STACK tự tạo.
 - ...



- Nhận xét

- Chỉ nên dùng phương pháp đệ quy để giải các bài toán kinh điển như giải các vấn đề “chia để trị”, “lần ngược”.
- Vấn đề đệ quy không nhất thiết phải giải bằng phương pháp đệ quy, có thể sử dụng phương pháp khác thay thế (khử đệ quy)
- Tiện cho người lập trình nhưng không tối ưu khi chạy trên máy.
- Bước đầu nên giải bằng đệ quy nhưng từng bước khử đệ quy để nâng cao hiệu quả.



Bài tập thực hành

- Bài 1: Các bài tập trên mảng sử dụng đệ quy.
- Bài 2: Viết hàm xác định chiều dài chuỗi.
- Bài 3: Hiển thị n dòng của tam giác Pascal.
 - $a[i][0] = a[i][i] = 1$
 - $a[i][k] = a[i-1][k-1] + a[i-1][k]$
- Bài 4: Viết hàm đệ quy tính $C(n, k)$ biết
 - $C(n, k) = 1$ nếu $k = 0$ hoặc $k = n$
 - $C(n, k) = 0$ nếu $k > n$
 - $C(n, k) = C(n-1, k) + C(n-1, k-1)$ nếu $0 < k < n$



Bài tập thực hành

- Bài 5: Đổi 1 số thập phân sang cơ số khác.
- Bài 6: Bài toán 8 hậu
- Bài 7: Bài toán mã đi tuần
- Bài 8: Tính các tổng truy hồi.



Chúc các em học tốt!

