



IT001 - NHẬP MÔN LẬP TRÌNH

CHƯƠNG 6: HÀM ĐỆ QUY - RECURSION

Đệ quy là một kỹ thuật lập trình đầy sức mạnh và bí ẩn. Đệ quy (hay còn gọi là recursion) là phương pháp giải quyết vấn đề bằng cách chia nhỏ nó thành những bài toán con giống hệt nhau, sau đó sử dụng chính hàm đó để giải quyết từng bài toán con. Trong chương này, chúng ta sẽ tìm hiểu về đệ quy và cách xây dựng hàm đệ quy để giải quyết một số bài toán đệ quy cơ bản.

Khoa Khoa học Máy tính



NỘI DUNG

- 6.1 Khái niệm đệ quy
- 6.2 Khái niệm và Cấu trúc hàm đệ quy
- 6.3 Các loại đệ quy
- 6.4 Các bước xây dựng hàm đệ quy
- 6.5 Nguyên tắc hoạt động của hàm đệ quy
- 6.6 Ví dụ các bài toán đệ quy cơ bản
- 6.7 Các vấn đề đệ quy thông dụng
- 6.8 Ưu điểm và khuyết điểm của phương pháp đệ quy



NỘI DUNG

- 6.1 Khái niệm đệ quy
- 6.2 Khái niệm và Cấu trúc hàm đệ quy
- 6.3 Các loại đệ quy
- 6.4 Các bước xây dựng hàm đệ quy
- 6.5 Nguyên tắc hoạt động của hàm đệ quy
- 6.6 Ví dụ các bài toán đệ quy cơ bản
- 6.7 Các vấn đề đệ quy thông dụng
- 6.8 Ưu điểm và khuyết điểm của phương pháp đệ quy



6.1 Khái niệm đệ quy



6.1 Khái niệm đệ quy

- Ví dụ: Cho $S(n) = 1 + 2 + 3 + \dots + n$

=> $S(10)$? $S(11)$?

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

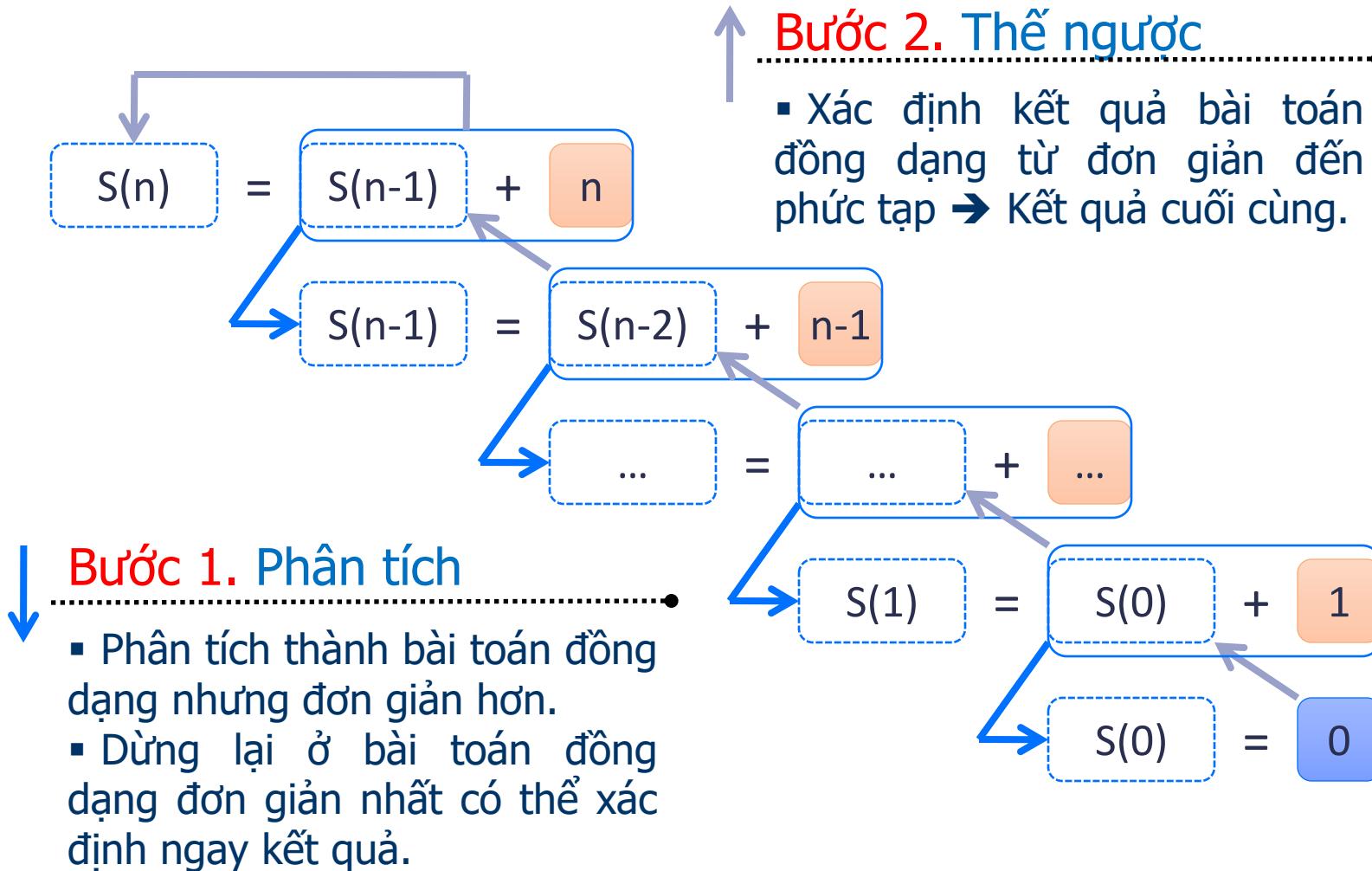
$$S(11) = 1 + 2 + \dots + 10 + 11 = 66$$

$$= S(10) + 11$$

$$= 55 + 11 = 66$$



6.1 Khái niệm đệ quy





6.1 Khái niệm đệ quy

- 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 $S(n-1)$.
 - Hai điều kiện quan trọng trong việc giải bài toán đệ quy
 - Tồn tại bước đệ quy
 - Điều kiện dừng

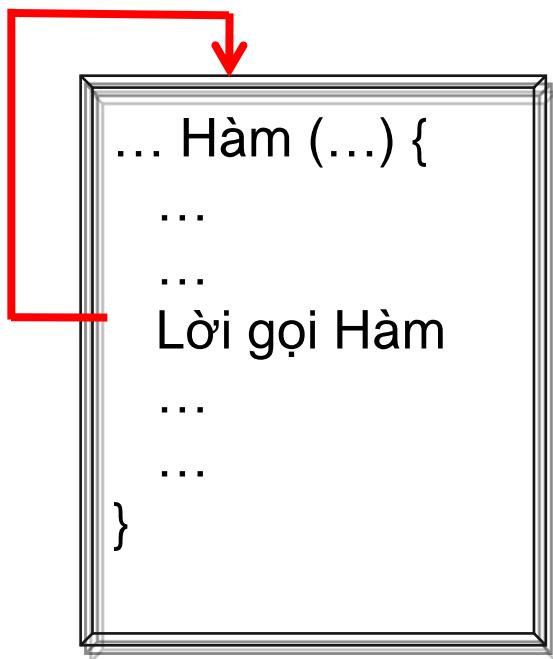


6.2 Khái niệm và cấu trúc hàm đệ quy

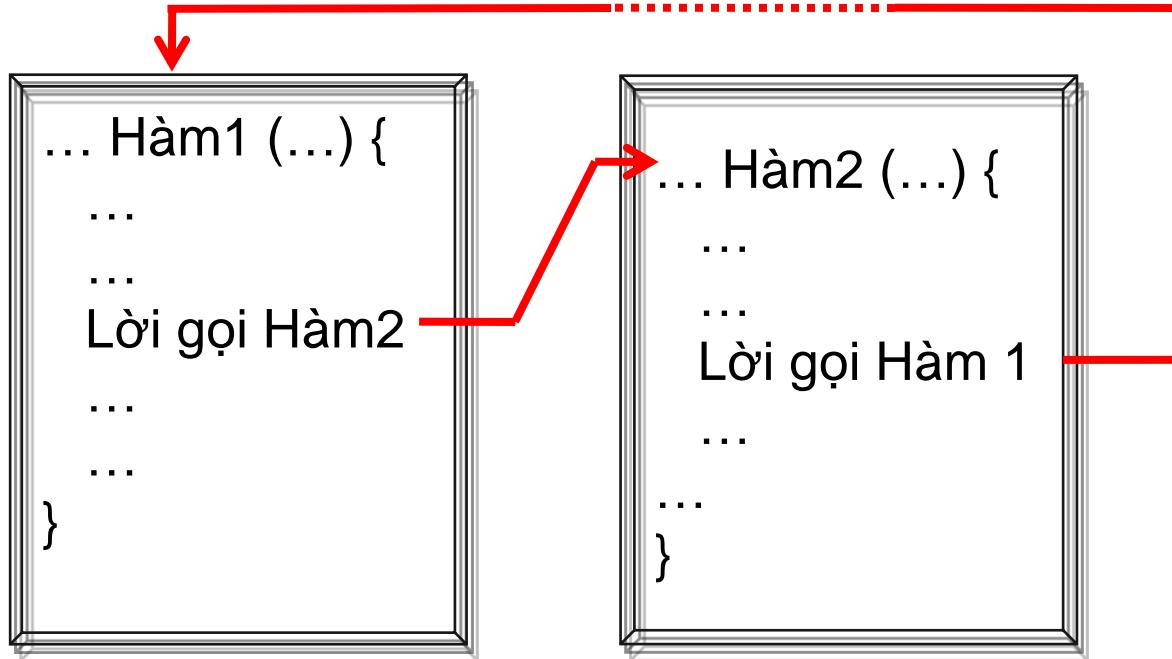


6.2 Khái niệm hàm đệ quy

- 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.



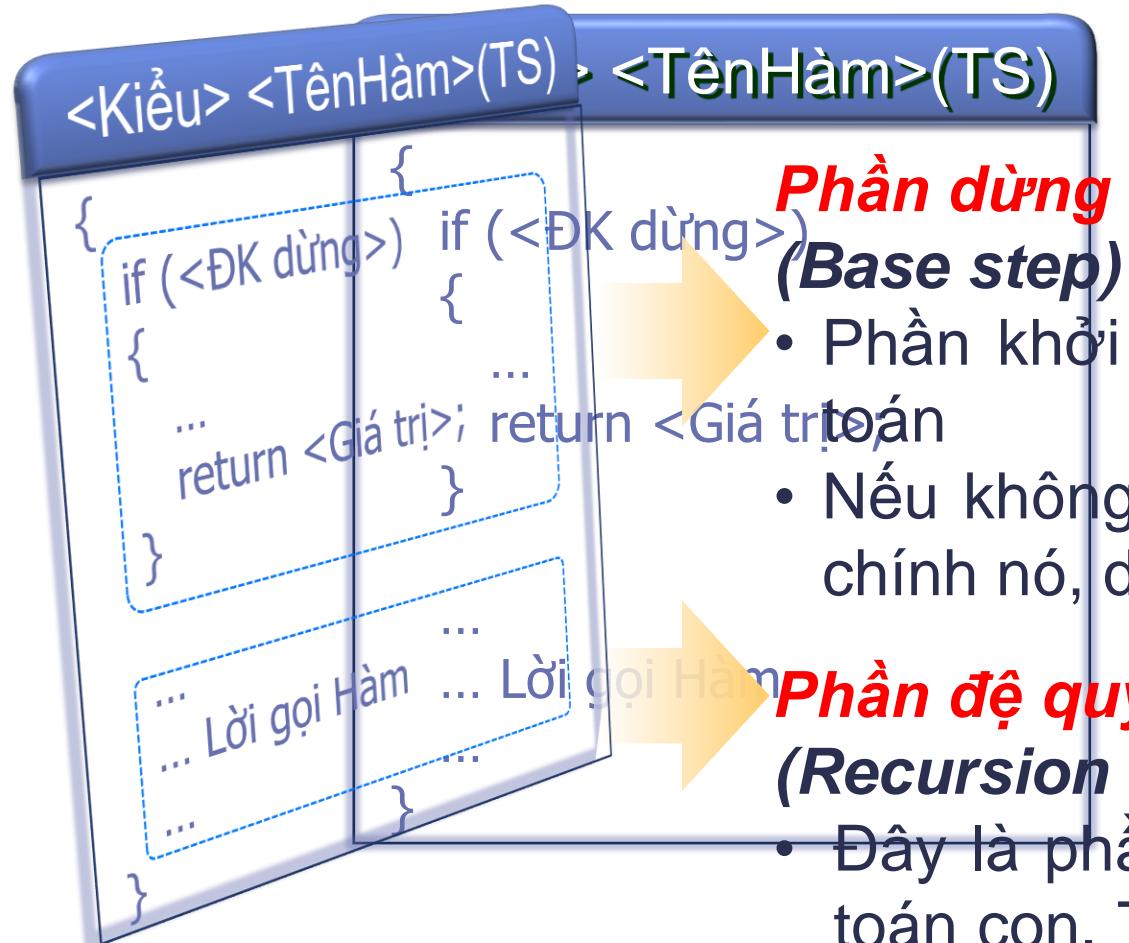
Đệ quy trực tiếp



Đệ quy gián tiếp



Cấu trúc hàm đệ quy



Phân dừng (Base step)

- Phân khởi tính toán hoặc điểm kết thúc của thuật toán
- Nếu không có điều kiện dừng, hàm sẽ liên tục gọi chính nó, dẫn đến tràn bộ nhớ.

Phân đệ quy (Recursion step)

- Đây là phần gọi lại chính hàm đó để giải quyết bài toán con. Tham số của lần gọi đệ quy thường được thay đổi để phù hợp với bài toán con.



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.

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

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

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

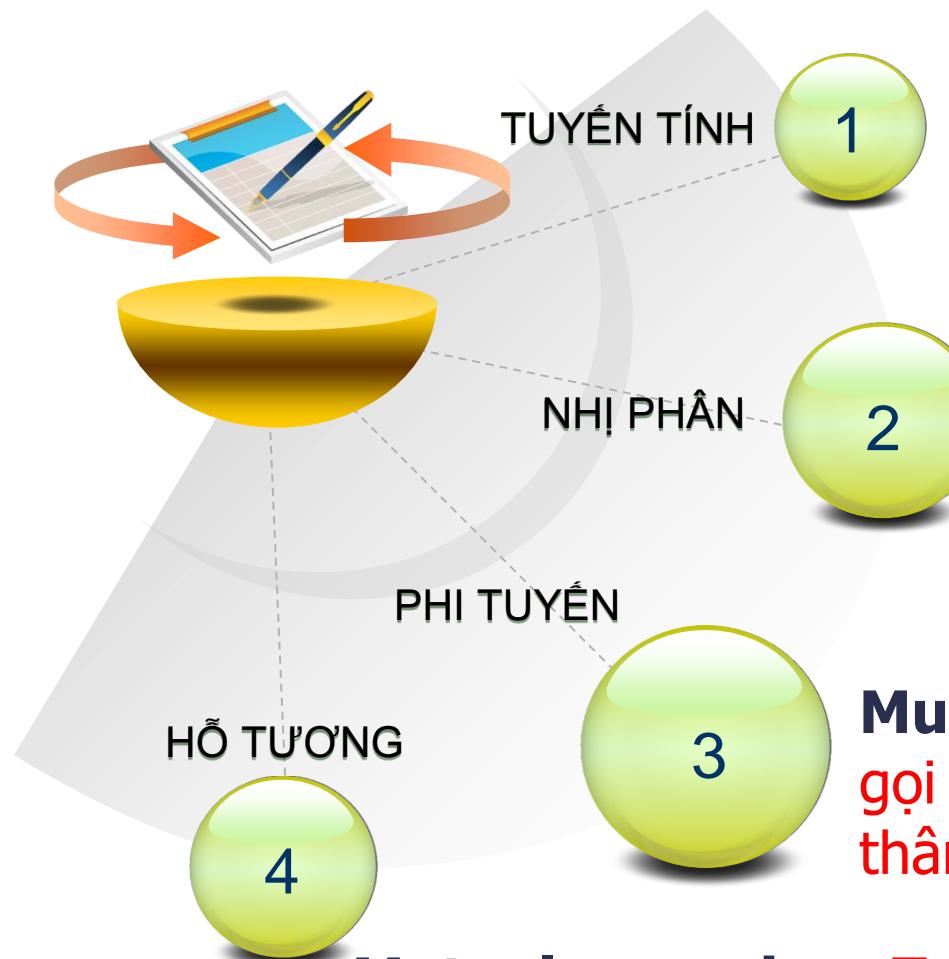
$$\Rightarrow S(2) = 1 + S(1) \Rightarrow S(1) = ???$$



6.3 Các loại đệ quy



6.3 Các loại đệ quy



Single recursion: Trong thân hàm có duy nhất một lời gọi hàm gọi lại chính nó một cách tường minh.

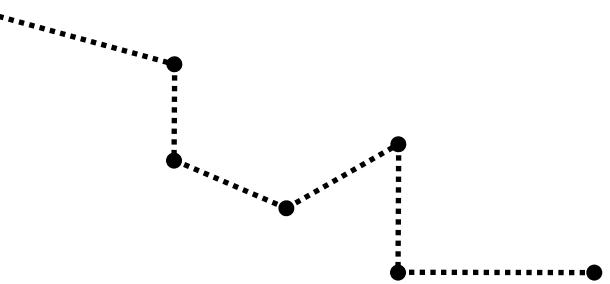
Binary recursion: Trong thân hàm có hai lời gọi hàm gọi lại chính nó một cách tường minh.

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

Mutual recursion: Trong thân hàm này có lời gọi hàm tới hàm kia và bên trong thân hàm kia có lời gọi hàm tới hàm này.



Đệ quy tuyến tính



Cấu trúc chương trình

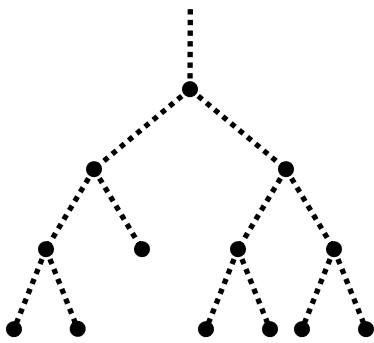
```
<Kiểu> TênHàm(<TS>) {  
    if (<ĐK dừng>) {  
        ...  
        return <Giá Trị>;  
    }  
    ... TênHàm(<TS>); ...  
}
```

Ví dụ

```
int giao_thua(int n){  
    if (n == 0) return 1;  
    return n * giao_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 nhị phân



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:

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

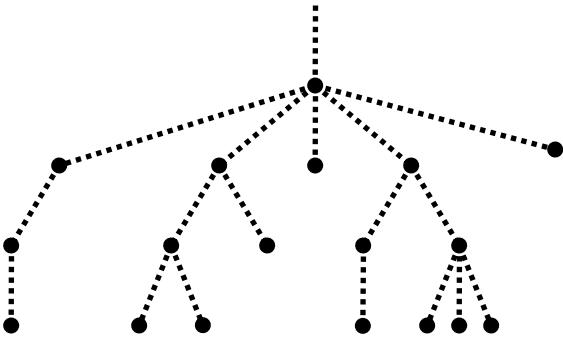
(Dãy Fibonacci: 0 1 1 2 3 5 8 13 21 ...)

∴ 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 phi tuyến



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:

$$x_0 = 1$$

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

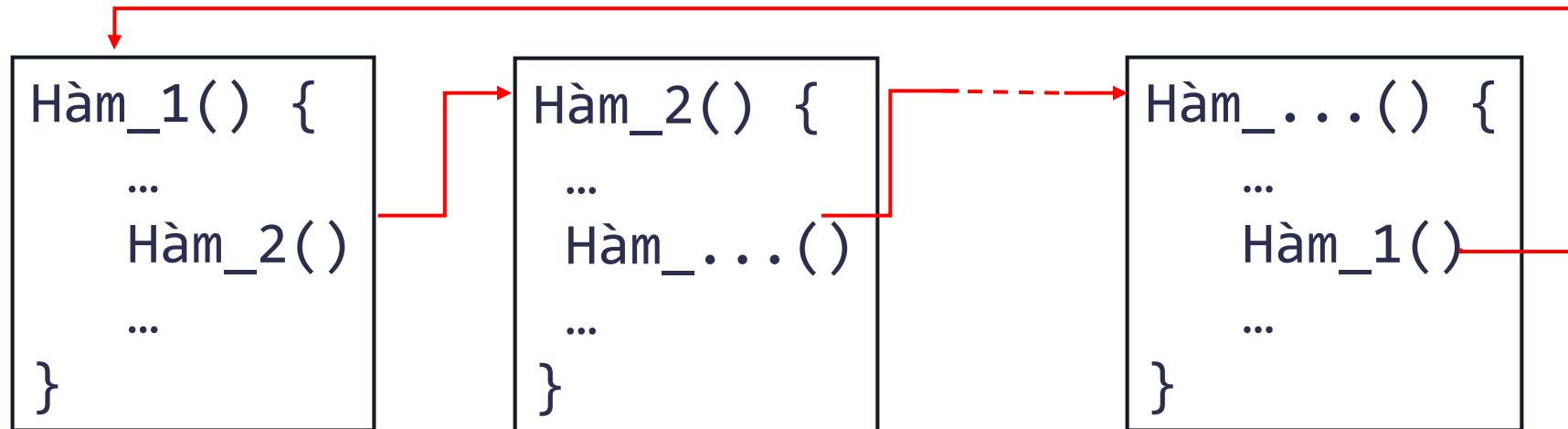
∴ Chương trình ∴

```
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;  
}
```



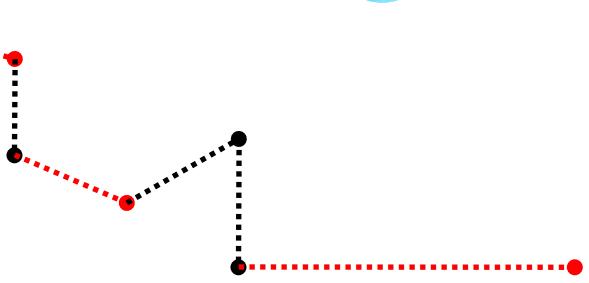
Đệ quy hỗ tương

- 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.





Đệ quy hỗn tương



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:

$$\begin{aligned}x(0) &= 1 \text{ với } n=0, y(0) = 1 \text{ với } n=0 \\x(n) &= x(n - 1) + y(n - 1) \\y(n) &= 2*x(n - 1)*y(n - 1)\end{aligned}$$

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



6.4 Các bước xây dựng hàm đệ quy



6.4 Các bước xây dựng hàm đệ quy

Thông số hóa bài toán

Tìm các trường hợp cơ sở

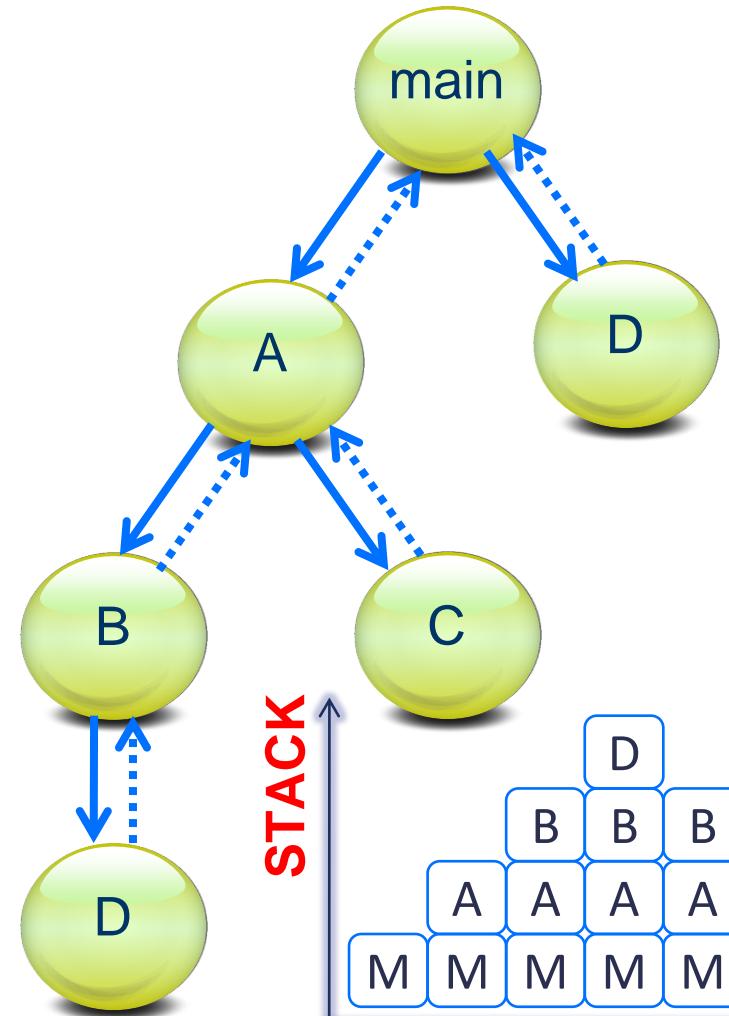
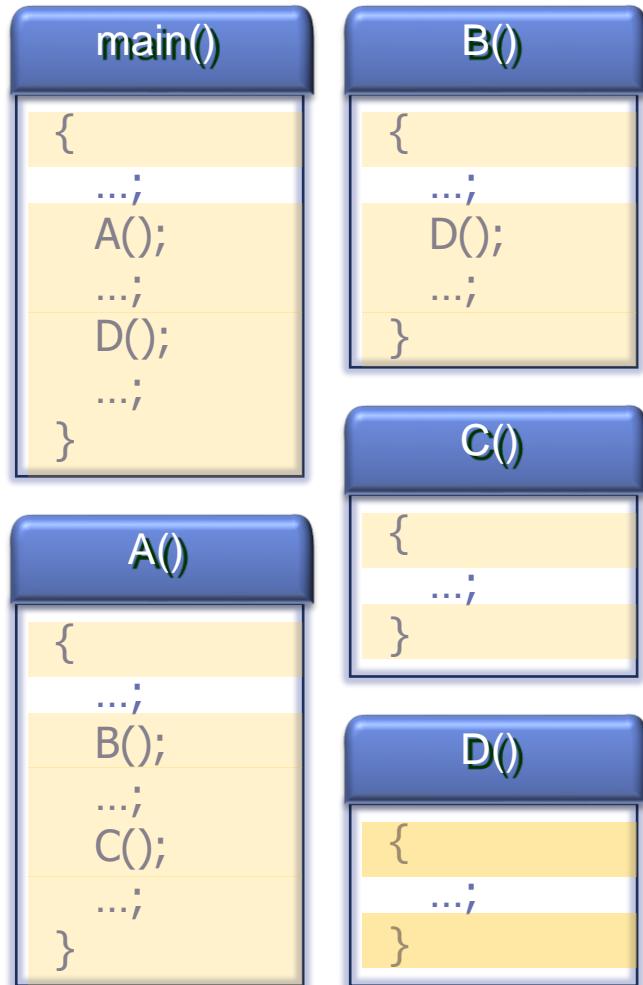
Tìm công thức trường hợp tổng quát



6.5 Nguyên tắc hoạt động của hàm đệ quy



Cơ chế gọi hàm và STACK



Ví dụ cơ chế hoạt động của stack



Nhận xét

- Cơ chế gọi hàm dùng STACK trong C++ phù hợp cho giải thuật đệ quy vì:
 - Lưu thông tin trạng thái còn dang mở 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.

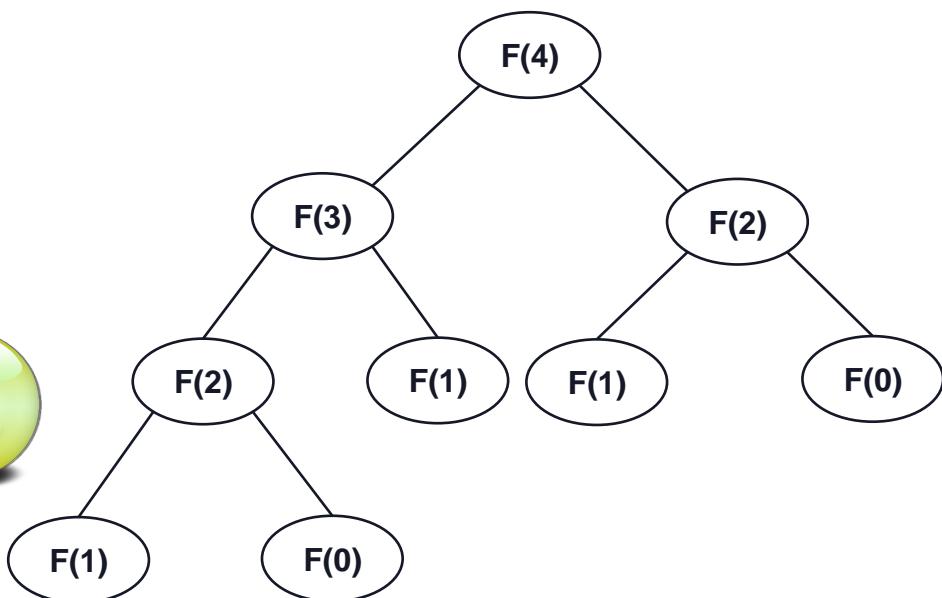
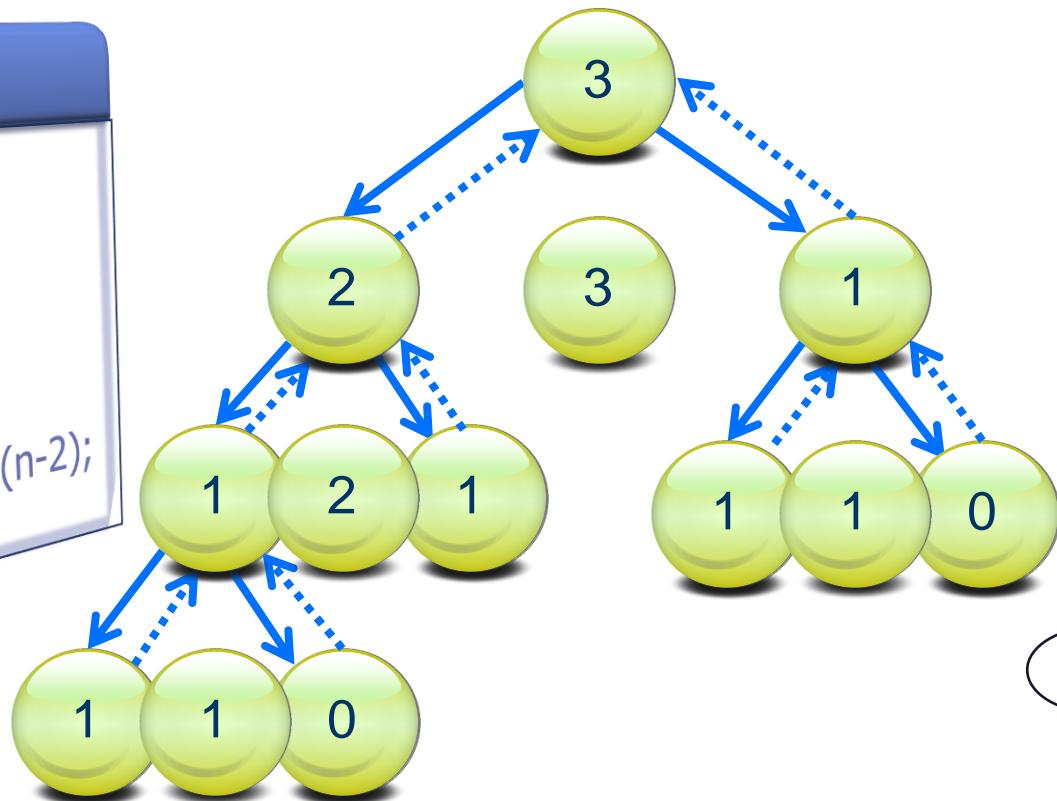


Ví dụ gọi hàm đệ quy

- Tính số hạng ở vị trí 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 ở vị trí 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(2)	F(2)	F(2)						
	F(4)																
main																	



Phân tích giải thuật đệ quy sử dụng cây 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.



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.



Ví dụ về StackOverflow

```
#include <iostream>

void print(){
    std::cout << "Hello";
    print();
    std::cout << "Good Luck Student";
}

int main(){
    print();
    return 0;
}
```

```
#include <iostream>

void func(int num) {
    std::cout << num << std::endl;
    func(num + 1);
}

int main() {
    func(1);
    return 0;
}
```



6.6 Ví dụ các bài toán đệ quy cơ bản



Ví dụ 1

- Đề bài: Tính tổng các số từ 1 đến n với n nhập từ bàn phím.

```
int sum1(int n) {  
    int i, kq = 0;  
    for (i = 1; i <= n; i++)  
        kq = kq + i;  
    return kq;  
}
```

```
int sum2(int n) {  
    if (n == 0) return 0;  
    return n + sum2(n - 1);  
}
```

```
void sum3(int n, int &s) {  
    if (n == 0) return ;  
    s += n;  
    sum3(n - 1, s);  
}
```

```
int main() {  
    int n, s;  
    cin >> n;  
    cout << sum1(n) << endl;  
    cout << sum2(n) << endl;  
    s=0; sum3(n, s); cout << s << endl;  
    return 0;  
}
```



Ví dụ 2

- Đề bài: Viết hàm tính giai thừa của số nguyên không âm n nhập từ bàn phím.

```
long double gaii_thua1(int n) {  
    long double kq = 1;  
    for (int i = 1; i <= n; i++)  
        kq = kq * i;  
    return kq;  
}
```

```
void gaii_thua3(int n, long double &s) {  
    if (n == 0) return;  
    s *= n;  
    gaii_thua3(n - 1, s);  
}
```

```
long double gaii_thua2(int n) {  
    if (n == 0) return 1;  
    return n * gaii_thua2(n - 1);  
}
```

```
int main() {  
    long double n, s; cin >> n;  
    cout << gaii_thua1(n) << endl;  
    cout << gaii_thua2(n) << endl;  
    s=1; gaii_thua3(n, s); cout << s;  
    return 0;  
}
```



Tính thời gian chạy

```
#include <time.h>
...
int main() {
    clock_t start = clock();
    for(int i = 0; i < 1000000; i++)
        giai_thua1(10000);
    cout << "Khong de quy: " << (double)(clock() - start)/CLOCKS_PER_SEC << " giay !" << endl;

    start = clock();
    for(int i = 0; i < 1000000; i++)
        giai_thua2(10000);
    cout << "De quy: " << (double)(clock() - start)/CLOCKS_PER_SEC << " giay !";
    return 0;
}
```

Kết quả:

Khong de quy: 28.667 giay !
De quy: 47.68 giay !



Ví dụ 3

- Đề bài: Viết hàm tính lũy thừa x^n .

```
#include<iostream>
using namespace std;
double LuyThua(double x, int n) {
    if (n == 0) return 1;
    return x * LuyThua(x, n - 1);
}

int main() {
    double x;
    int n;
    cout << "Nhập x, n: "; cin >> x >> n;
    cout << "Giá trị x^n = " << LuyThua(x, n);
}
```



Ví dụ 4

- Đề bài: Viết hàm tìm USCLN của 2 số a, b được nhập vào từ bàn phím.

```
#include <iostream>
```

```
int UCLN1(int a, int b) {
    if (a==b) return a;
    if (a > b)
        return UCLN1(a - b, b);
    return UCLN1(a, b - a);
}
```

```
int UCLN2(int a, int b) {
// Lưu ý: đầu vào a>=b
    if (!a || !b) return a+b;
    return UCLN2(b, a%b);
}
```

- **Cách 1:** Lưu ý điều kiện ban đầu $a>0, b>0$
$$\begin{cases} \text{UCLN}(a, a)=a \\ \text{UCLN}(a, b)=\text{UCLN}(a-b, b) \text{ nếu } a > b \\ \text{UCLN}(a, b)=\text{UCLN}(a, b-a) \text{ nếu } b>a \end{cases}$$

- **Cách 2:**
$$\begin{cases} \text{UCLN}(a, 0)=a \\ \text{UCLN}(a, b)=\text{UCLN}(b, a \bmod b) \end{cases}$$

(Điều kiện ban đầu: $a \geq b$, nên nếu $a < b$ thì hoán đổi a, b trước khi thực hiện)



Ví dụ 5

- Đề bài: Tính tổng các chữ số của số nguyên dương n.

```
#include <iostream>

int sum(int n){
    if(n==0) return 0;
    return sum(n/10)+n%10;
}

void sum(int n, int &s){
    if(n==0) return;
    s += n%10;
    return sum(n/10, s);
}
```

```
int main() {
    int n, s;
    std::cin >> n;
    std::cout << sum(n);

    s=0;
    sum(n, s);
    std::cout << s << std::endl;
    return 0;
}
```



Ví dụ 6

- Đề bài: Tính hàm Ackerman, biết rằng hàm Ackermann được định nghĩa như sau:

$$A(m, n) = \begin{cases} n + 1 & , m = 0 \\ A(m - 1, 1) & , m > 0 \text{ và } n = 0 \\ A(m - 1, A(m, n - 1)) & , m > n \text{ và } n > 0 \end{cases}$$

```
int Ackerman(int m, int n) {  
    if (m == 0) return (n + 1);  
    else if (n == 0) return Ackerman(m - 1, 1);  
    return Ackerman(m - 1, Ackerman(m, n - 1));  
}
```



6.7 Các vấn đề đệ quy thông dụng



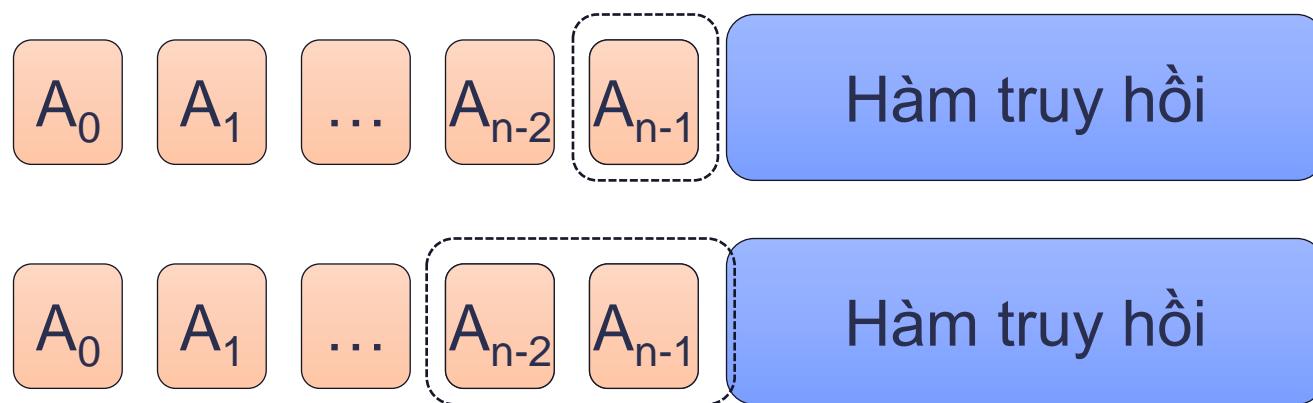
6.7 Các vấn đề dễ quy thông dụng





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.
- Bài toán điển hình của hệ thức truy hồi: Tính Giai thừa, Tính số Fibonacci.





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:
 - $V_h = 2V_{h-1}$
 - $V_0 = 2$
- Đệ quy tuyến tính với $V(h)=2*V(h-1)$ và điều kiện dừng $V(0) = 2$



Hệ thức truy hồi

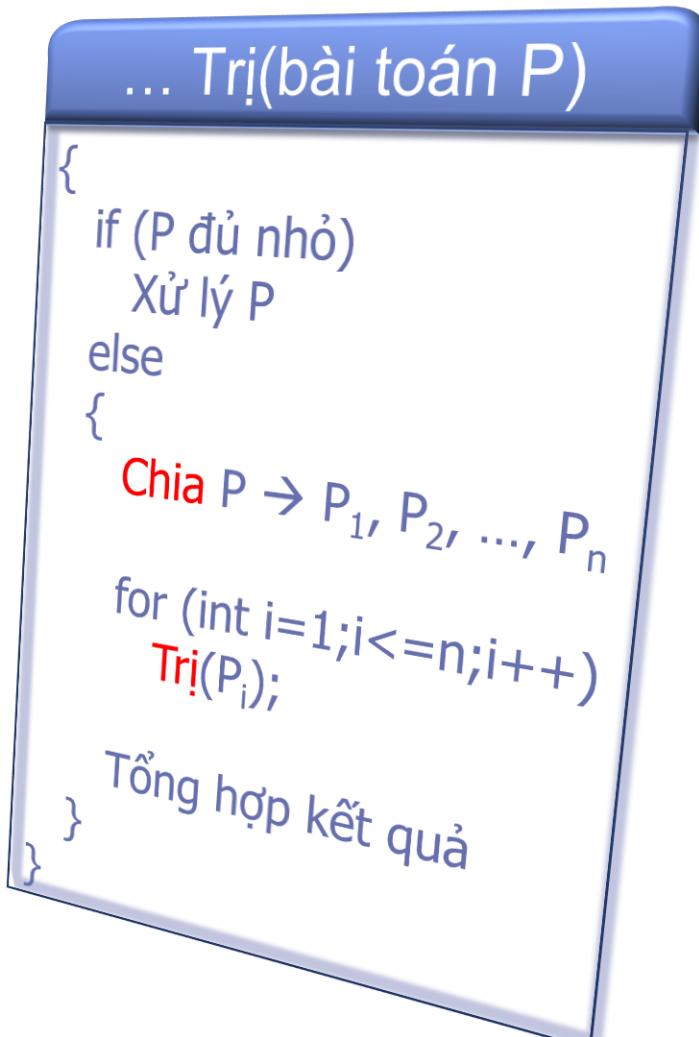
- **Ví dụ 2:** Gửi ngân hàng 1000 USD, lãi suất 12%/năm. Số tiền có được sau 30 năm là bao nhiêu?
 - **Giải pháp:**
 - Gọi T_n là số tiền có được sau n năm.
 - Ta có:
 - $T_n = T_{n-1} + 0.12T_{n-1} = 1.12T_{n-1}$
 - $V(0) = 1000$
- Đệ quy tuyến tính với $T(n)=1.12*T(n-1)$ và điều kiện dừng $V(0) = 1000$



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.





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 $[\text{mid} + 1, r]$
- ➔ Sử dụng đệ quy nhị phân.



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 $Y_L = Y_{2n-1} \dots Y_n, Y_N = Y_{n-1} \dots Y_0 \Rightarrow Y = 10^n Y_L + Y_N$
- $\Rightarrow X * Y = 10^{2n} X_L Y_L + 10^n (X_L Y_L + X_N Y_N) + X_N Y_N$
- và $X_L Y_L + X_N Y_N = (X_L - X_N)(Y_N - Y_L) + X_L Y_L + X_N Y_N$

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



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.



Lần ngược (Backtracking)

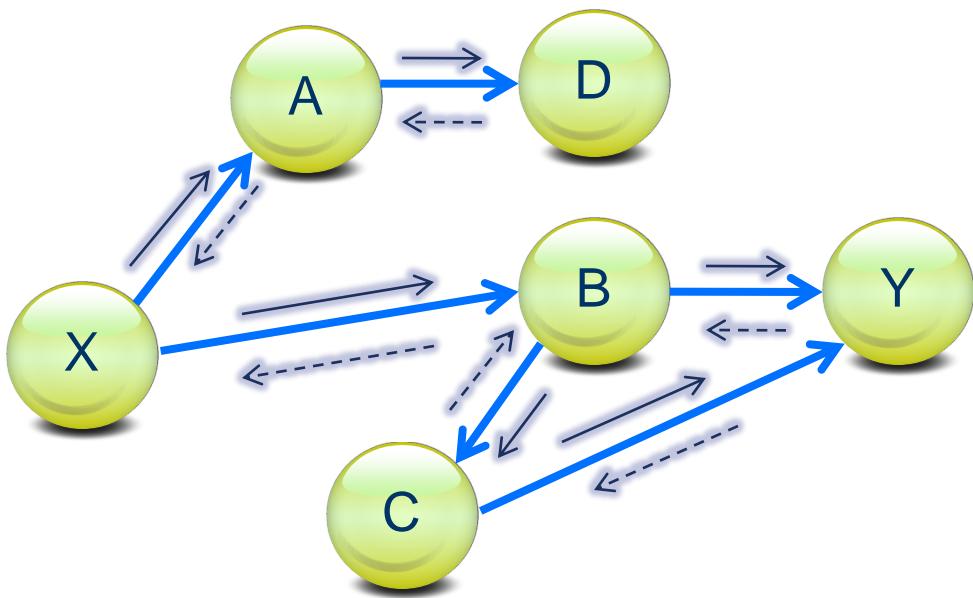
- **Khái niệm:**

- 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.



Lần ngược (Backtracking)

- Ví dụ: Tìm đường đi từ X đến Y.





Giải bài toán Tháp Hà Nội

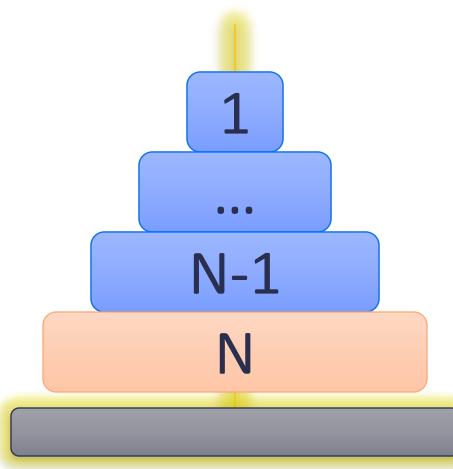
- **Mô tả bài toán:**

- 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.

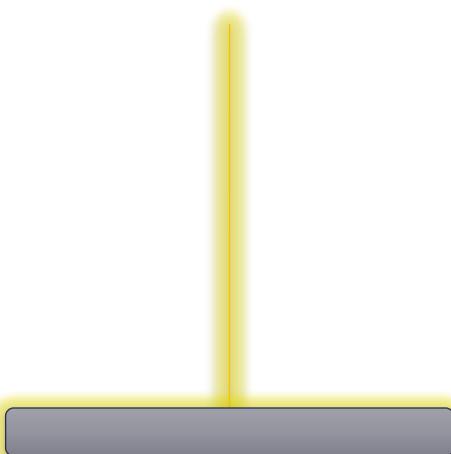


Tháp Hà Nội

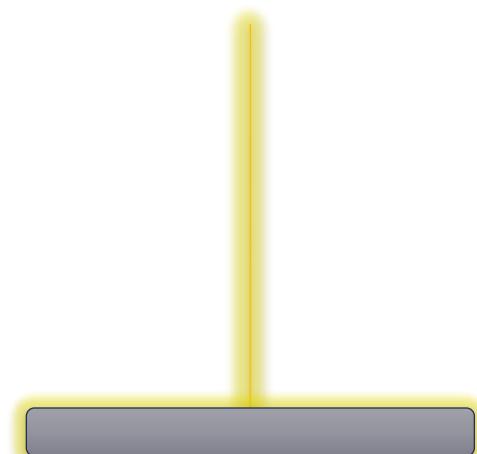
$$N \text{ đĩa } A \rightarrow C = ? \text{ đĩa } A \rightarrow B + Đĩa N A \rightarrow C + N-1 \text{ đĩa } B \rightarrow C$$



Cột nguồn A



Cột trung gian B



Cột đích C



6.8 Ưu điểm và khuyết điểm của phương pháp đệ quy



6.8 Ưu điểm và khuyết điểm của phương pháp đệ quy

- **Ưu điểm:**

- Giải quyết vấn đề phức tạp một cách đơn giản
- Mã đơn giản và ngắn gọn, tiết kiệm thời gian viết mã nguồn
- Đệ quy có thể áp dụng cho nhiều loại vấn đề khác nhau

- **Khuyết điểm:**

- Khó khăn trong việc theo dõi quá trình thực thi
- Khó khăn trong quá trình gỡ lỗi
- Tốn nhiều bộ nhớ và thời gian thực hiện
- Nguy cơ tràn bộ nhớ



Bài tập

- Câu 1: Giải các bài sau bằng đệ quy:
 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 $n!$
- Câu 2: Tìm số thứ n của dãy Fibonacci
- Câu 3: Tìm số thứ n của dãy pandovan
- Câu 4: Hiển thị n dòng của tam giác Pascal.
 - $a[i][0] = a[i][i] = 1$
 - $a[i][k] = a[i-1][k-1] + a[i-1][k]$
- Câu 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 $0 < k < n$



Chúc các em học tốt!

