

Cấu trúc dữ liệu

Data Structure

Ts. Nguyễn Đức Thuần
BM Hệ thống Thông Tin



Tài liệu tham khảo

- [1] Hồ Thuần – Hồ Cẩm Hà – Trần Thiên Thành
 - CẤU TRÚC DỮ LIỆU, PHÂN TÍCH THUẬT TOÁN VÀ PT PHẦN MỀM
- [2] Đỗ Xuân Lôi
 - CẤU TRÚC DỮ LIỆU & GIẢI THUẬT
- [3] Lê Minh Hoàng
 - GIẢI THUẬT & LẬP TRÌNH
- [4] Đinh Mạnh Tường
 - CẤU TRÚC DỮ LIỆU & GIẢI THUẬT
- [5] Đặng Bình Phương
 - Slide bài giảng KỸ THUẬT LẬP TRÌNH
- [6] A.V. Aho, J.E. Hopcroft, J.D. Ullman
 - THE DESIGN & ANALYSIS OF COMPUTER ALGORITHMS
- [7] N.E. Wirth
 - DATA STRUCTURE & ALGORITHM
 - Email: ctdl59nt@gmail.com pw: nt59cautruc

Chương I

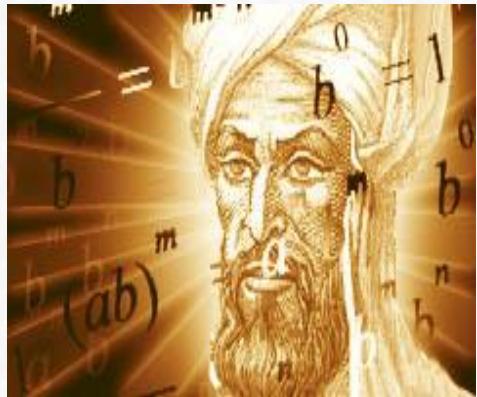
THIẾT KẾ & PHÂN TÍCH GIẢI THUẬT

Don't start coding

You must design a working algorithm first.



1. THUẬT TOÁN (Algorithm)



Al' Khwarizmi



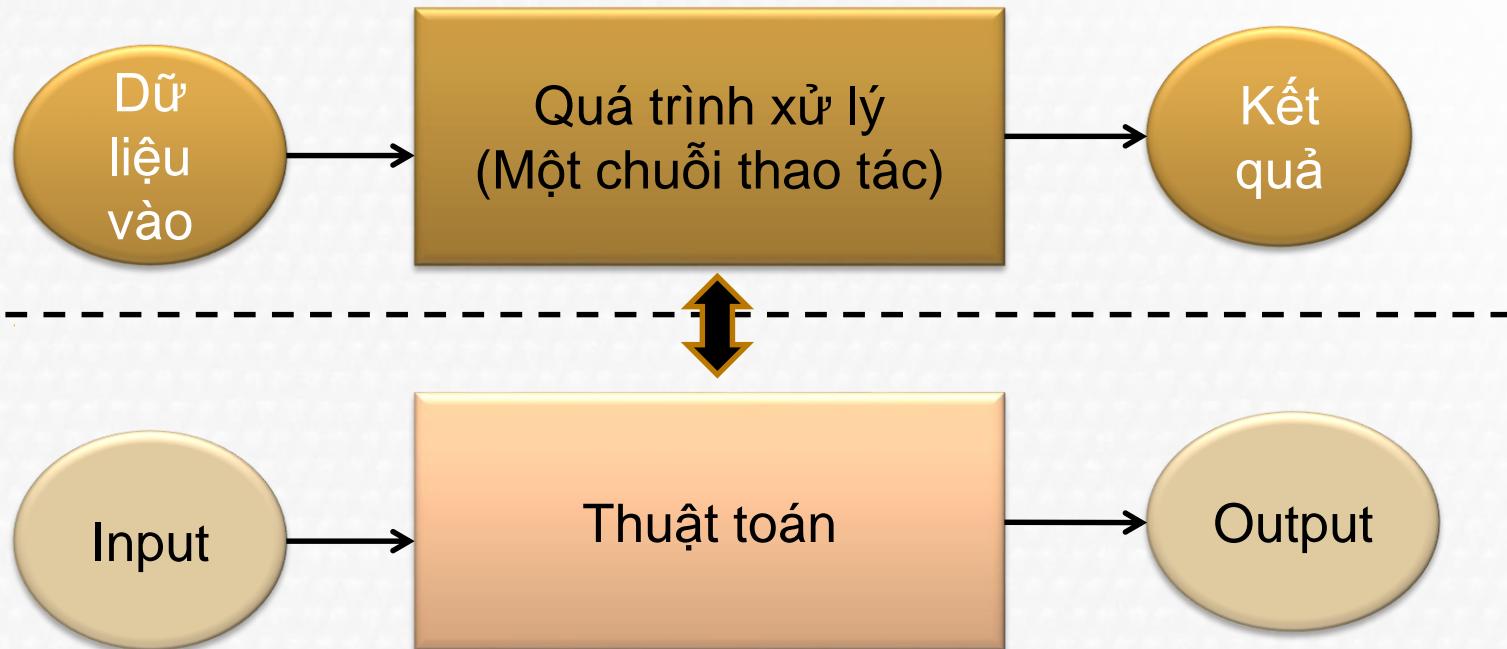
D.E Knuth



N.E Wirth

1. THUẬT TOÁN (Algorithm)

▪ Giải bài toán



1. THUẬT TOÁN (Algorithm)



1. THUẬT TOÁN (Algorithm)

- *Thuật toán để giải một bài toán là một dây hữu hạn các thao tác được sắp xếp theo một trật tự xác định sao cho sau khi thực hiện dây thao tác đó, từ Input của bài toán ta nhận được Output cần tìm.*
 - Có nhiều thuật toán để giải cho cùng một bài toán
 - Cần xác định thuật toán thích hợp

1. THUẬT TOÁN (Algorithm)

▪ Đặc trưng của thuật toán

- *Tính kết thúc*: Sau hữu hạn bước cho kết quả
- *Tính xác định*: Các bước riêng lẻ, cùng dữ liệu vào cho cùng kết quả ra
- *Dữ liệu vào/ra*
- *Tính phổ dụng*: Giải quyết cho 1 lớp bài toán
- *Tính hiệu quả*: Nhanh chóng, chính xác
 - Thời gian
 - Dung lượng lưu trữ

1. THUẬT TOÁN (Algorithm)

▪ Mô tả thuật toán

- Liệt kê: Nêu tuần tự các bước cần tiến hành
- Sơ đồ khối (flow chart): Biểu diễn các thao tác bằng các hình vẽ qui ước
- Mã giả (Peusedo Code): dùng ngôn ngữ tự nhiên + từ khóa qui ước thể hiện các thao tác

1. THUẬT TOÁN (Algorithm)

Ví dụ: Tìm giá trị lớn nhất của dãy số a_1, a_2, \dots, a_N ,

Input : Số nguyên dương N và dãy a_1, a_2, \dots, a_N .

Output : Tìm Max là giá trị lớn nhất của dãy đã cho.

Mô tả LIỆT KÊ

Thuật toán:

B₁: Nhập N, dãy a_1, a_2, \dots, a_N .

B₂: Đặt Max = a_1

B₃: i=2

B4: i>N \Rightarrow B₇

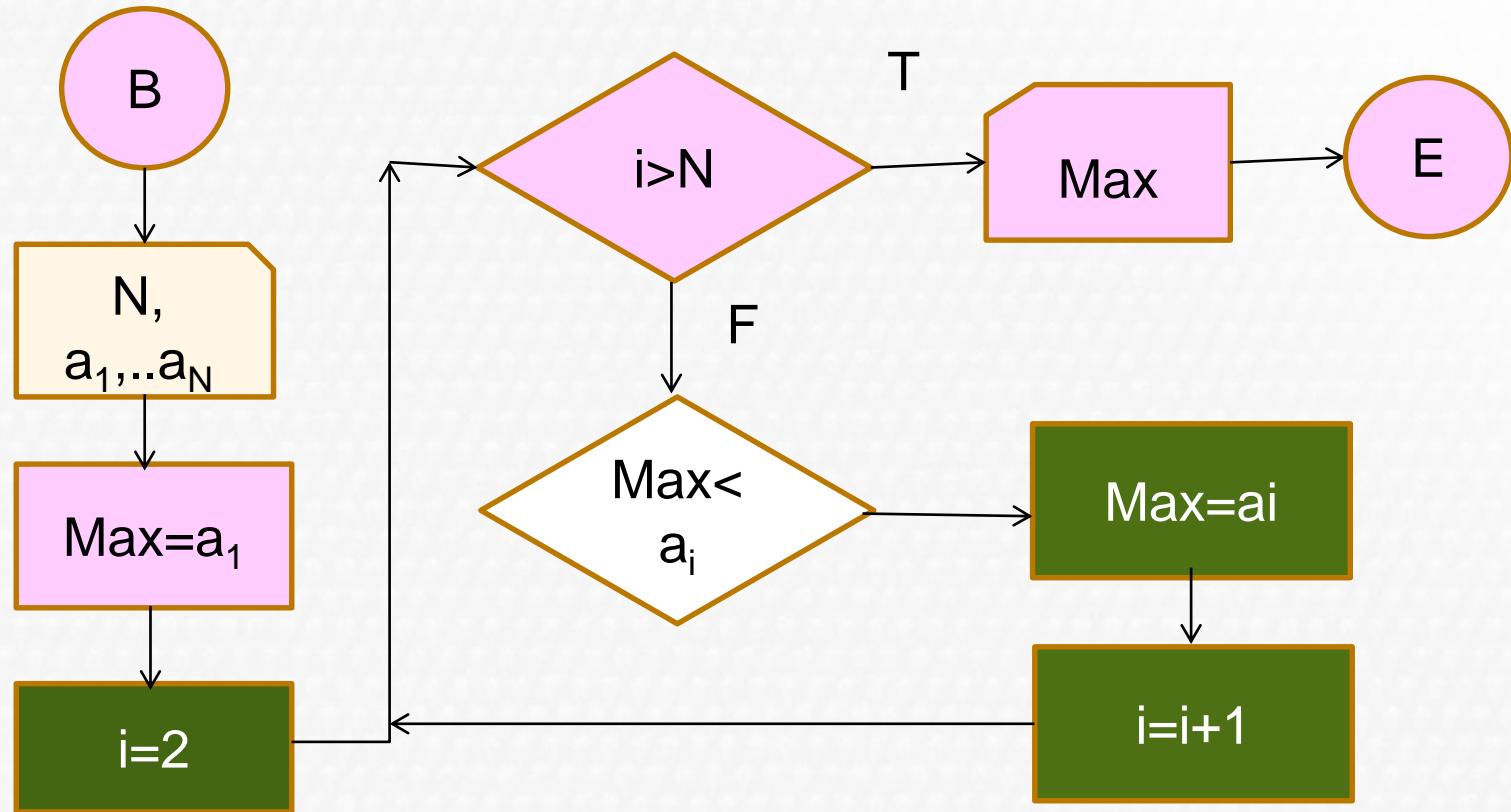
B5: Nếu Max < a_i thì Max=a_i

B6: i=i+1 \Rightarrow B₄

B7: Xuất Max \Rightarrow Kết thúc

1. THUẬT TOÁN (Algorithm)

Mô tả SƠ ĐỒ KHỐI



1. THUẬT TOÁN (Algorithm)

Mô tả MÃ GIẢ

Read (N, a[1..N])

Set Max = a[1];

i=2;

While i <= N do

{ If Max< a[i] then Max=a[i] ;

i=i+1; }

Return Max.

1. THUẬT TOÁN (Algorithm)

▪ Tiêu chí đánh giá thuật toán

- Tiêu chuẩn 1: Đơn giản, dễ hiểu, dễ cài đặt
- Tiêu chuẩn 2:
 - Thời gian thực hiện nhanh
 - Dung lượng lưu trữ bé

Nhận xét:

- Tiêu chuẩn 1 mang tính chủ quan
- Tiêu chuẩn 2 mang tính khách quan

1. THUẬT TOÁN (Algorithm)

■ Tổ chức dữ liệu:

- Mô hình dữ liệu: (Data Model) trừu tượng hóa dữ liệu: Đồ thị, tập hợp, danh sách, cây, ...
 - Phản ánh đúng thực tế + thao tác phù hợp
 - Tiết kiệm tài nguyên hệ thống
- Cấu trúc dữ liệu (Data Structure): chọn các đơn vị cấu trúc (construct) của NNLT để biểu diễn mô hình dữ liệu
 - Kiểu dữ liệu: Data Type = (V , O)
 - Ví dụ: Mảng, bản ghi, chuỗi, file, ..

2. ĐỘ PHÚC TẠP THUẬT TOÁN

▪ Để có thể so sánh thời gian chạy của các thuật toán:

- Không phụ thuộc vào chủ quan của con người
- Không phụ thuộc vào thiết bị

⇒ Xây dựng một hàm đánh giá phụ thuộc vào kích thước của dữ liệu gọi là Độ phức tạp thuật toán.

⇒ Hàm có dạng $T(<\text{kích thước bài toán}>)$

2. ĐỘ PHÚC TẠP THUẬT TOÁN

- Xây dựng hàm đánh giá độ phức tạp:
 - ❖ 1. Đếm số lượng các phép toán cơ bản
 - Phép toán cơ bản: +, -, *, /, phép gán, phép so sánh, thao tác đọc/ghi file ...

Ký hiệu: D_n : tập các dữ liệu vào kích thước n

$\text{cost}_A(d)$: số phép toán cơ bản thuật toán A xử lý dữ liệu d

Độ phức tạp trong trường hợp xấu nhất:

$$T(n) = \underset{d \in D_n}{\text{Min}} \quad \{\text{cost}(d)\}$$

Độ phức tạp trong trường hợp tốt nhất:

$$T(n) = \underset{d \in D_n}{\text{Max}} \quad \{\text{cost}(d)\}$$

2. ĐỘ PHỨC TẠP THUẬT TOÁN

Ví dụ: Tính tổng $S(n) = 1! + 2! + \dots + n!$

```

int P1(int n)
{
    int t, gt, i;
    t = 0;
    gt = 1;
    for(i= 1; i<= n; i++) {
        gt *= i;
        t += gt;
    }
    return t;
}

```

Cách 1

Số lần	Chi phí
1	c_1
1	c_2
n	c_3
n	c_4
n	c_5

Độ phức tạp: $(c_3 + c_4 + c_5)n + c_1 + c_2 \rightarrow T(n)$

```

int P2(int n)
{
    int t, gt, i, j;
    t = 0;
    for(i= 1; i<= n; i++) {
        gt = 1;
        for(j= 1; j<= i; j++)
            gt *= j;
        t += gt;
    }
    return t;
}

```

Cách thứ 2:

Số lần	Chi phí
1	c_1
n	c_2
$\sum_1^n i$	c_3
$\sum_1^n i$	c_4 $n(n+1)/2$
$\sum_1^n i$	c_5 $n(n+1)/2$
n	c_6

Độ phức tạp:

$$(c_4 + c_5)n^2/2 + (c_2 + c_3 + c_6 + (c_4 + c_5)/2)n + c_1 \rightarrow T(n)$$

2. ĐỘ PHÚC TẬP THUẬT TOÁN

Ví dụ trường hợp tốt nhất, xấu nhất

```
int TimKiem(int a[100], int n, int x)
{
    int i;
    for(i= 0; i< n; i++)
        if (a[i] == x)
            return i;
    return -1;
}
```

Số lần

$t = ?$

1

- Nếu x ở đầu mảng: $t = 1$ (best case)
- Nếu x ở cuối mảng: $t = n$ (worst case)

2. ĐỘ PHÚC TẠP THUẬT TOÁN

▪ Xây dựng hàm đánh giá độ phức tạp:

❖ 2. Dùng ký hiệu O() (big oh)

Việc đánh giá thời gian thực hiện thông qua số lượng phép toán cơ bản quá chi tiết \Rightarrow **khó khăn trong việc so sánh và phân lớp các thuật toán**

Để thể hiện bản chất hơn độ phức tạp của thuật toán phụ thuộc vào kích thước \Rightarrow Khái niệm O-lớn

2. ĐỘ PHÚC TẬP THUẬT TOÁN

▪ Khái niệm O()

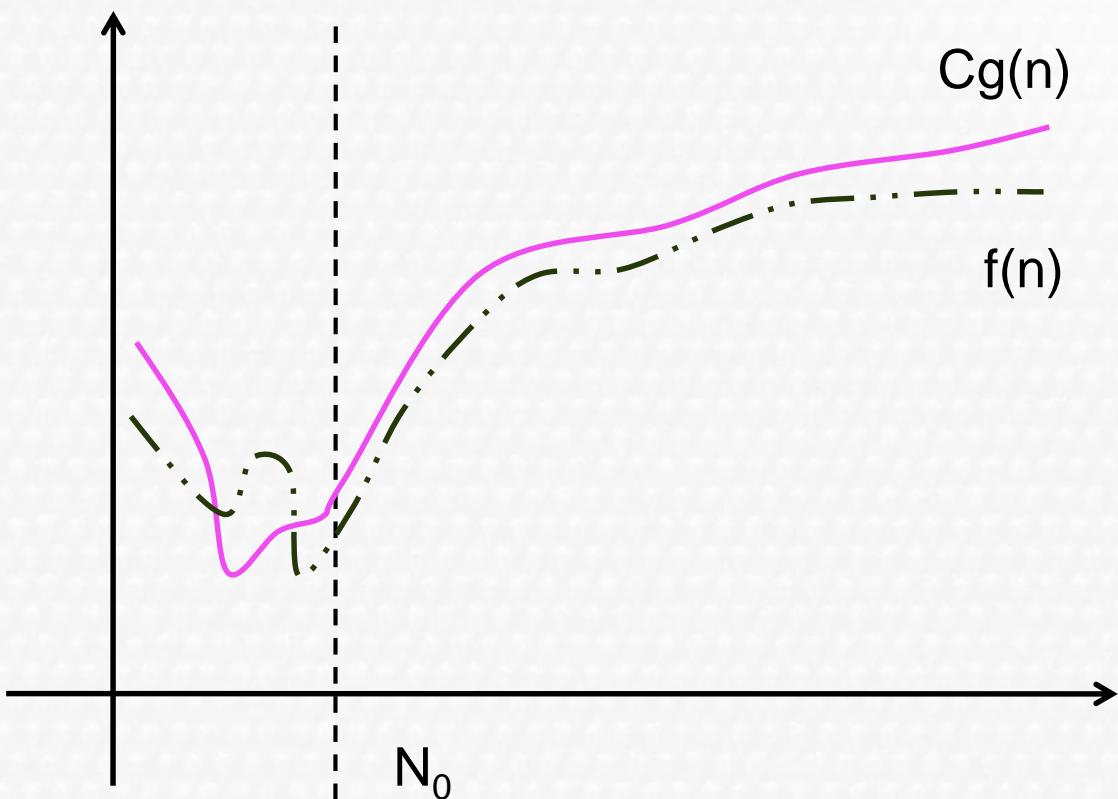
Cho $f(n)$, $g(n)$ là hai hàm xác định trên tập số nguyên không âm (tập số tự nhiên \mathbb{N}) có miền giá trị là \mathbb{R}^+ tập số thực không âm:

$$f, g: \mathbb{N} \rightarrow \mathbb{R}^+$$

Ta nói $f(n)$ là O -lớn $g(n)$, ký hiệu $f(n)=O(g(n))$ nếu và chỉ nếu:

$$\exists C, \exists N_0, \forall n \geq N_0 : f(n) \leq C.g(n)$$

2. ĐỘ PHÚC TẠP THUẬT TOÁN



Ký hiệu O-lớn cho ta phân lớp các hàm

2. ĐỘ PHÚC TẠP THUẬT TOÁN

Ký hiệu O-lớn	Tên gọi thường dùng
$O(1)$	Hằng
$O(\log n)$	Logarit
$O(n)$	Tuyến tính
$O(n \log n)$	nlogn
$O(n^2)$	Bình phương
$O(2^n)$	Mũ

2. ĐỘ PHỨC TẠP THUẬT TOÁN

▪ Các tính chất của O()

1. Tính phản xạ: $f(n) = O(f(n))$

2. Tính bắt đầu:

$$f(n)=O(g(n)), g(n)=O(t(n)) \Rightarrow f(n)=O(t(n))$$

3. Qui tắc hằng số:

$$f(n) = O(C \cdot g(n)) \Rightarrow f(n) = O(g(n))$$

4. Qui tắc cộng:

$$f_i(n)=O(g_i(n)), i=1,2 \Rightarrow f_1(n)+f_2(n) = O(\max(g_1(n),g_2(n)))$$

2. ĐỘ PHÚC TẠP THUẬT TOÁN

5. Qui tắc nhân:

$$f_i(n) = O(g_i(n)), i=1,2$$

$$\Rightarrow f_1(n) \times f_2(n) = O((g_1(n) \times g_2(n))$$

▪ Ứng dụng

- Câu lệnh đơn thực hiện một thao tác \Rightarrow QT hằng số
- Câu lệnh hợp thành là dãy các câu lệnh \Rightarrow QT tổng
- Câu lệnh rẽ nhánh dạng If ..then..else.. \Rightarrow QT Max
- Các câu lệnh lặp \Rightarrow QT Nhân

2. ĐỘ PHÚC TẠP THUẬT TOÁN

- Dãy tuân tự các lệnh đơn giản

$S_1; S_2; \dots S_k$, k: hằng số Độ phức tạp $O(1)$

- Vòng lặp đơn giản

for($i=1$; $i \leq n$; $i++$) {s}

s : câu lệnh đơn giản, độ phức tạp $O(1)$

Độ phức tạp $n.O(1)$ hay $O(n)$

- Vòng lặp lồng nhau

for($i=1$; $i < n$; $i++$)

 for($j=0$; $j < n$; $j++$) {s}

Độ phức tạp $n.O(n)$ hay $O(n^2)$

2. ĐỘ PHÚC TẠP THUẬT TOÁN

- Chỉ mục lặp không tuyến tính

$h=1$

```
while (h<=n)
```

```
{ s;
```

```
    h=2*h;
```

```
}
```

h nhận các giá trị 1, 2, 4, , , cho đến khi vượt n

Có $1+\log_2 n$ lần lặp: Độ phức tạp $O(\log_2 n)$

2. ĐỘ PHÚC TẠP THUẬT TOÁN

- Chỉ mục lặp phụ thuộc vào chỉ mục lặp khác

for($i=0$; $i < n$; $i++$)

 for($j=0$; $j < i$; $j++$) {s}

$$\sum_{i=0}^n \sum_{k=0}^i k$$

Độ phức tạp $\cong O(n^2)$

BÀI TẬP

1. XÁC ĐỊNH ĐỘ PHÚC TẬP CÁC ĐOẠN CHƯƠNG TRÌNH

a)

```
tong = 0;  
for(i=0; i<n; i++)  
    for(j=i; j<=i+3; j++)  
        tong += j;
```

c)

```
tong= 0;  
for(i=n; i>=1; i--)  
    for(j=i; j>=1; j--)  
        for(k=i; k<=i+3; k++)  
            tong+= i+ j+ k;
```

b)

```
tich = 0;  
for(i=n; i>0; i--)  
    for(j= 0; j<2*i; j++)  
        tich *= j;
```

d)

```
tong= 0;  
for(i=1; i<=n; i++)  
    for(j=1; j<=i; j++)  
        for(k=1; k<=2*i; k++)  
            tong+= i+ j+ k;
```

Gợi ý:

e)

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

f)

```
for(i=0; i<n-1; i++)  
    for(j=i+1; j<n; j++)  
        if (a[j]< a[i])  
        {  
            int t = a[i];  
            a[i]= a[j];  
            a[j]= t;  
        }
```

a)

$$\sum_{i=0}^{n-1} 4$$

b)

$$\sum_{i=0}^{n-1} (2 * i)$$

c)

$$\sum_{i=1}^n \sum_{j=1}^i 4$$

d)

$$\sum_{i=1}^n \sum_{j=1}^i (2 * i)$$

e)

$$\sum_{i=1}^n \sum_{j=1}^i j$$

f)

$$\sum_{i=0}^{n-1} n - i - 1$$

2. XÁC ĐỊNH ĐỘ PHÚC TẠP ĐOẠN CHƯƠNG TRÌNH

```
void NghiPhan(int n)
{
    int c, a[100];
    c = 0;
    while (n>0)
    {
        c++;
        a[c] = n%2;
        n= n/2;
    }
    printf("Ket qua la: ");
    for(i=c; i>=1; i--)
        printf("%d", a[i]);
}
```

Chương II

ĐÊ QUI

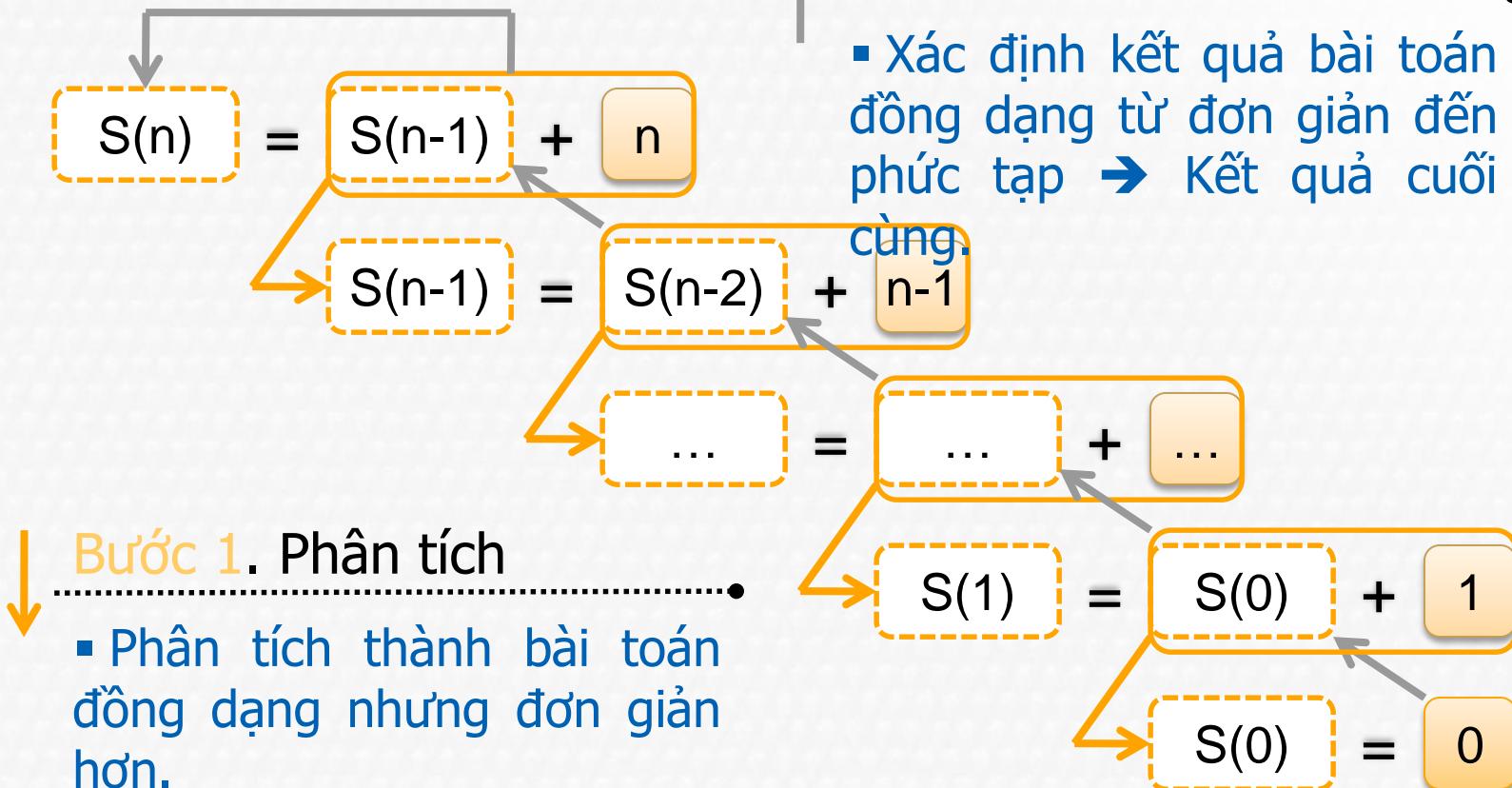
1. DẪN NHẬP

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

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

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

1. DẪN NHẬP



2. ĐỆ QUI

Khái niệm

Một khái niệm, định nghĩa được gọi là đệ qui nếu trong khái niệm/định nghĩa có chứa lại chính nó.

Ví dụ:

Người giàu:

Người có nhiều tiền

Người có cha mẹ là người giàu



2. ĐỆ QUI

- Ví dụ

$$n! = \begin{cases} 1, & n = 0 \\ (n-1)!n, & n > 0 \end{cases}$$

$$a^n = \begin{cases} 1, & n = 0 \\ a^{n-1} \cdot a, & n > 0 \end{cases}$$

- Cần phân biệt đệ qui và các khái niệm lặp, điệp từ ...

2. ĐỆ QUI

- Một khái niệm đê qui gồm 2 thành phần:
 - Thành phần neo (dùng) (anchor)
 - Thành phần đê qui

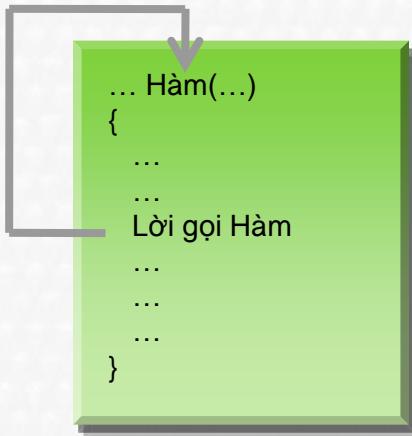
Phần đê quy thể hiện tính “quy nạp”

Phần neo đảm bảo cho tính dùng.

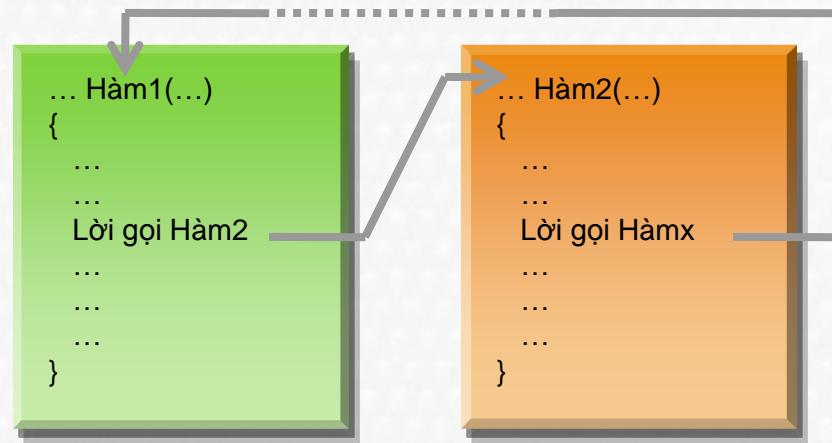
3. LẬP TRÌNH ĐỆ QUI

▪ 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.

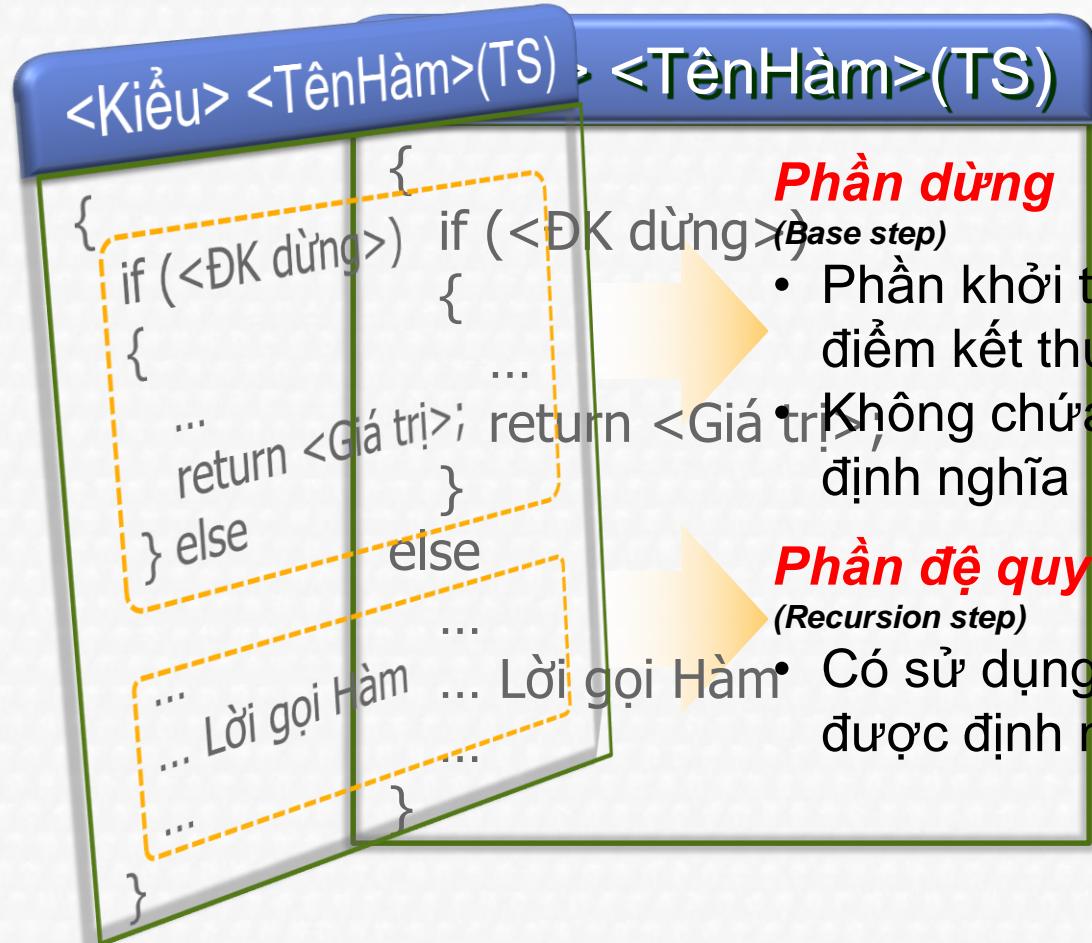


ĐQ trực tiếp



ĐQ gián tiếp

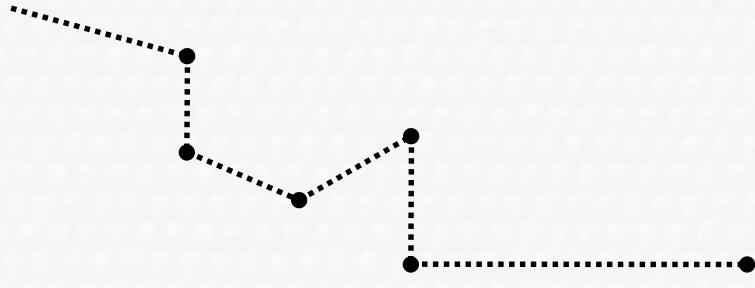
Cấu trúc hàm đệ quy



Phân loại



Đệ quy tuyến tính



Cấu trúc chương trình

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

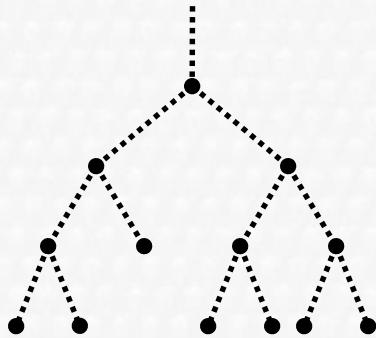
Ví dụ

Tính $S(n) = 1 + 2 + \dots + n$
→ $S(n) = S(n - 1) + n$
ĐK dừng: $S(0) = 0$

..: Chương trình :.

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

Đệ quy nhị phân



Cấu trúc chương trình

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

Ví dụ

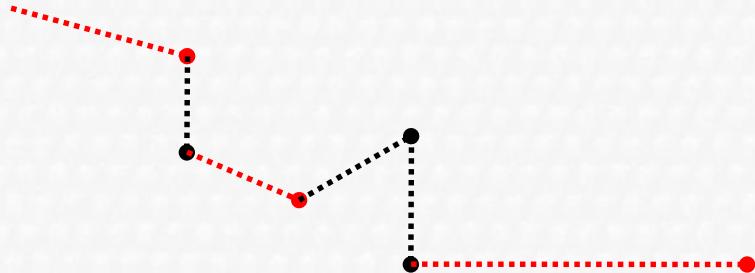
Tính số hạng thứ n của dãy Fibonacy:

$f(0) = f(1) = 1$
 $f(n) = f(n - 1) + f(n - 2)$, $n > 1$
ĐK dừng: $f(0)=1$ và $f(1)=1$

∴ Chương trình ∴
long Fibo (int n)

```
{  
    if (n == 0 || n == 1)  
        return 1; else  
    return Fibo(n-1)+Fibo(n-1);  
}
```

Đề quy hỗn hợp



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:

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

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

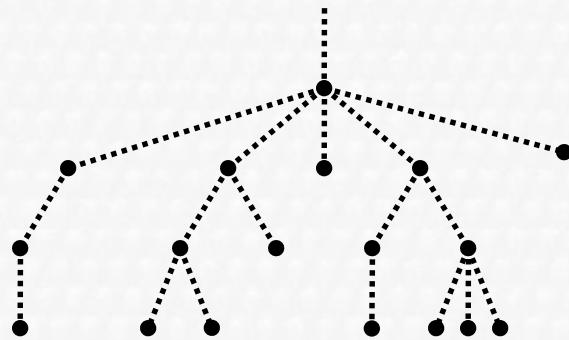
$$y(n) = 3*x(n - 1) + 2*y(n - 1)$$

ĐK dừng: $x(0) = 1, y(0) = 0$

∴ Chương trình :

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

Đệ quy phi tuyến



Cấu trúc chương trình

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

Ví dụ

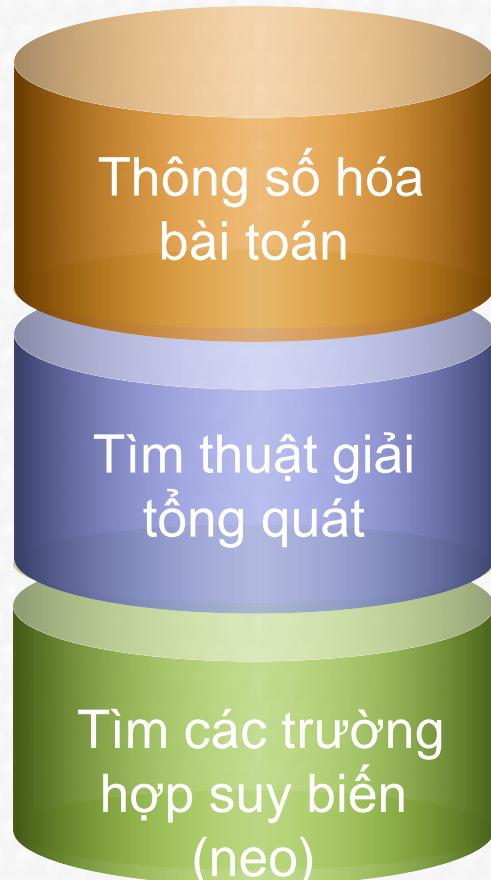
Tính số hạng thứ n của dãy:

$$\begin{aligned}x(0) &= 1 \\x(n) &= n^2x(0) + (n-1)^2x(1) + \dots \\&\quad + 2^2x(n-2) + 1^2x(n-1)\end{aligned}$$

ĐK dừng: $x(0) = 1$

.: Chương trình :.
long xn(int n)
{
 if (n == 0) return 1; else {
 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

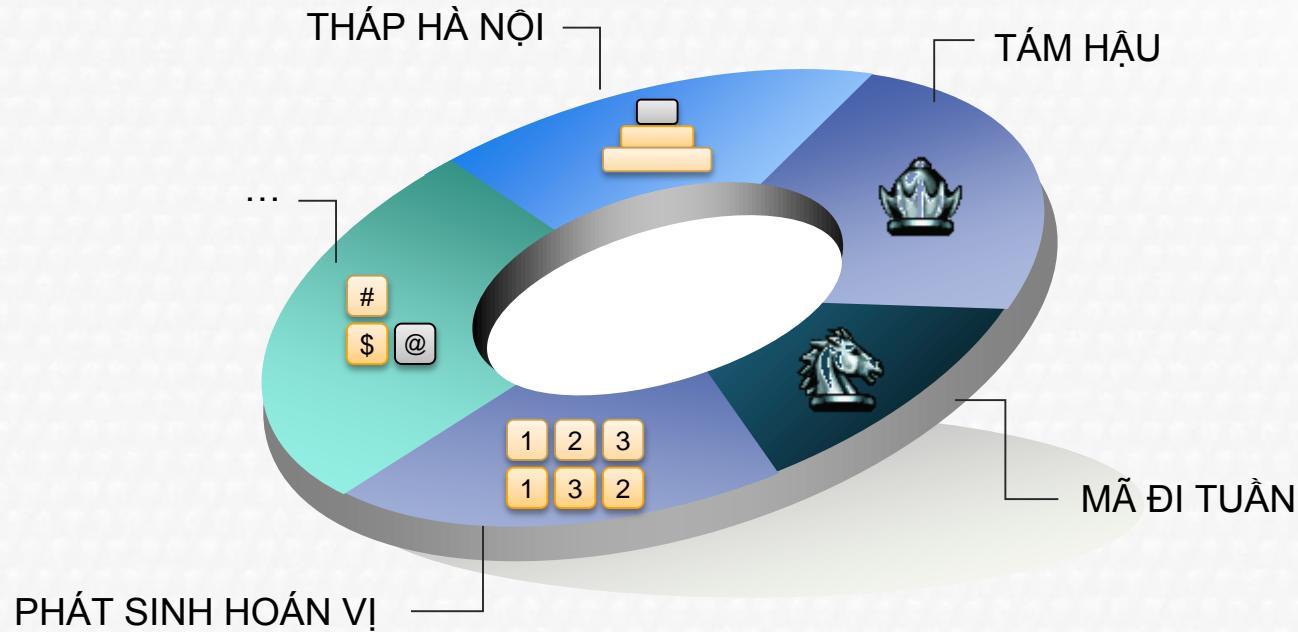


- Tổng quát hóa bài toán cụ thể thành bài toán tổng quát.
- Thông số hóa cho bài toán tổng quát
- VD: n trong hàm tính tổng $S(n)$, ...

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

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

Một số bài toán kinh điển



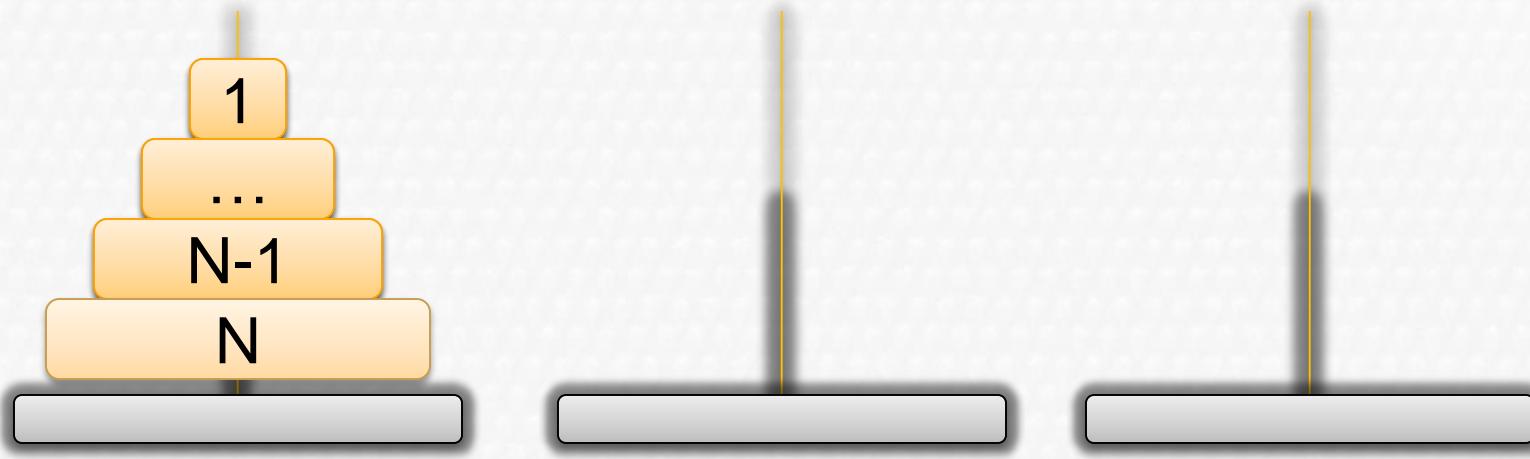
Tháp Hà Nội

■ 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 B làm cột trung gian.

Tháp Hà Nội

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



Chuyển n đĩa từ a → b, c là cọc trung gian

```
void HaNoi(int n, char a, char c, char b)
{
```

```
    if (n == 1)
```

Điều kiện dừng

```
        printf("\n %c -> %c", a, c);
```

```
    else
```

```
{
```

```
    HaNoi(n-1, a, b, c);
```

Chuyển n-1 đĩa từ a → b

```
    printf("\n %c -> %c", a, c);
```

```
    HaNoi(n-1, b, c, a);
```

Chuyển đĩa lớn từ a → c

```
}
```

```
}  
int main()
```

```
{
```

```
    HaNoi(5, 'A', 'C', 'B');
```

Chuyển n-1 đĩa từ b → c

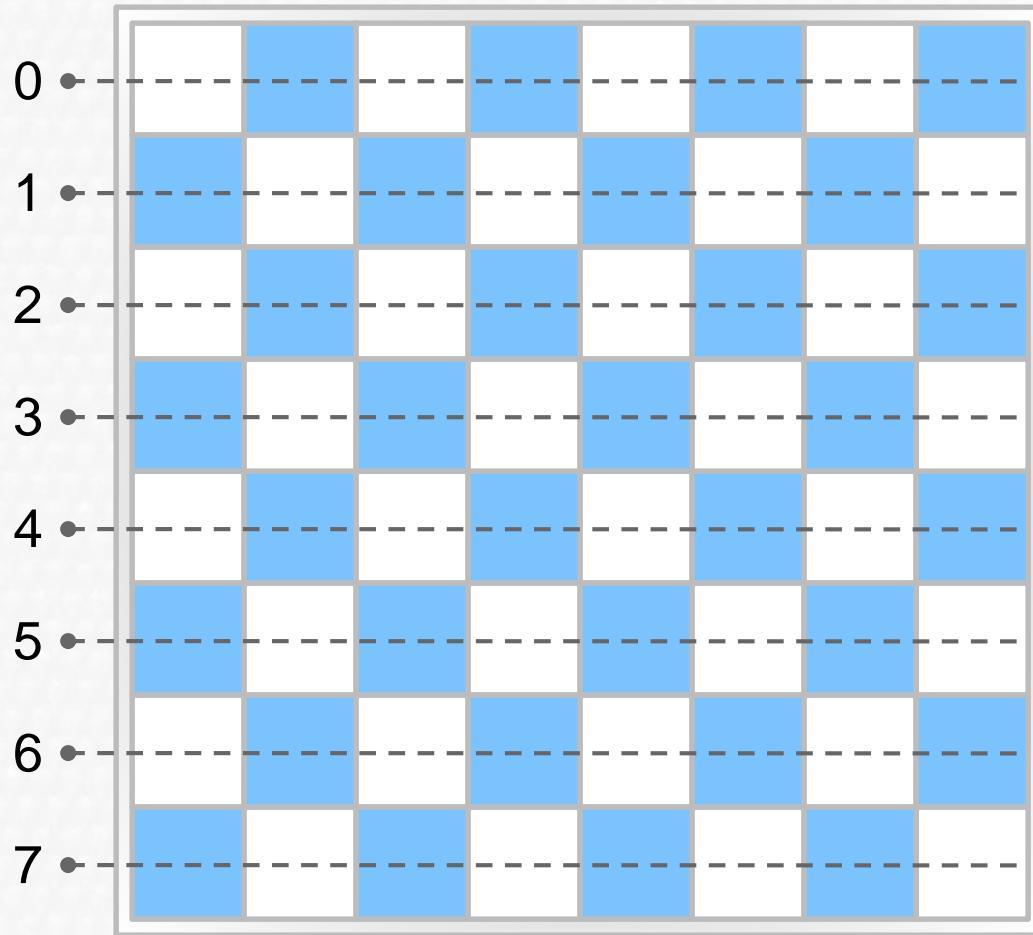
```
}
```

Tám hậu

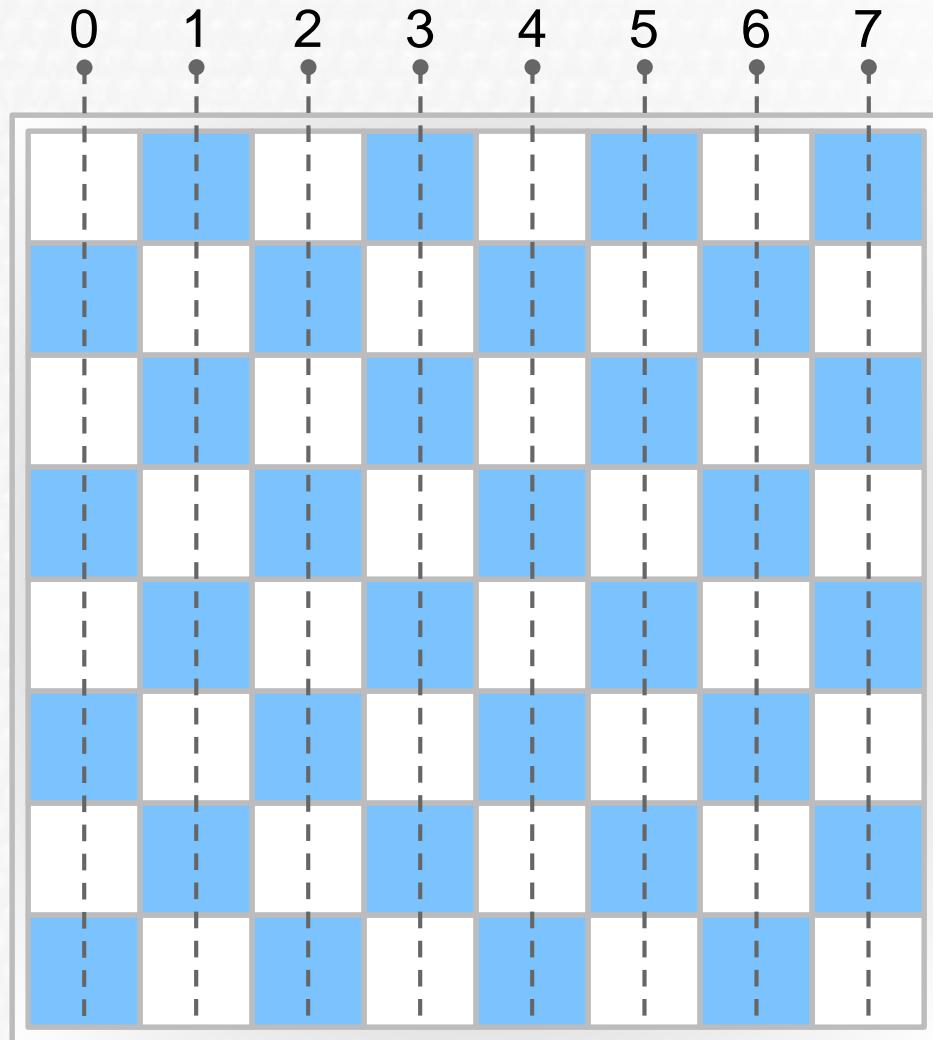
■ Mô tả bài toán

- Cho bàn cờ vua kích thước 8x8
- 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 dòng

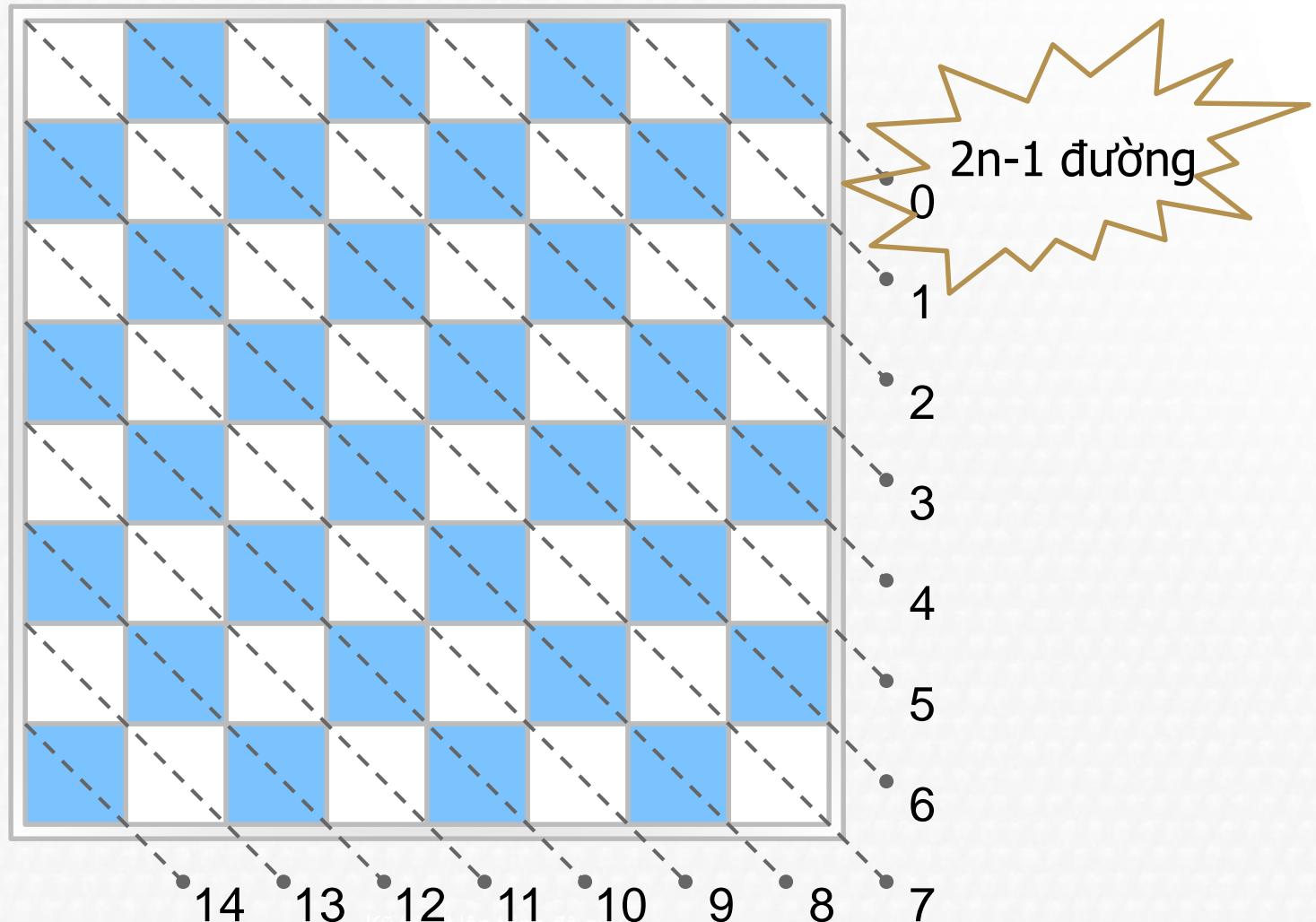


Tám hậu – Các cột

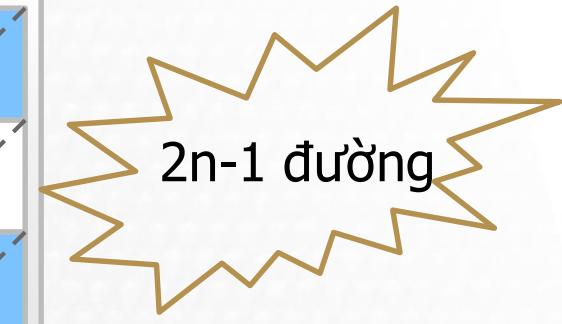
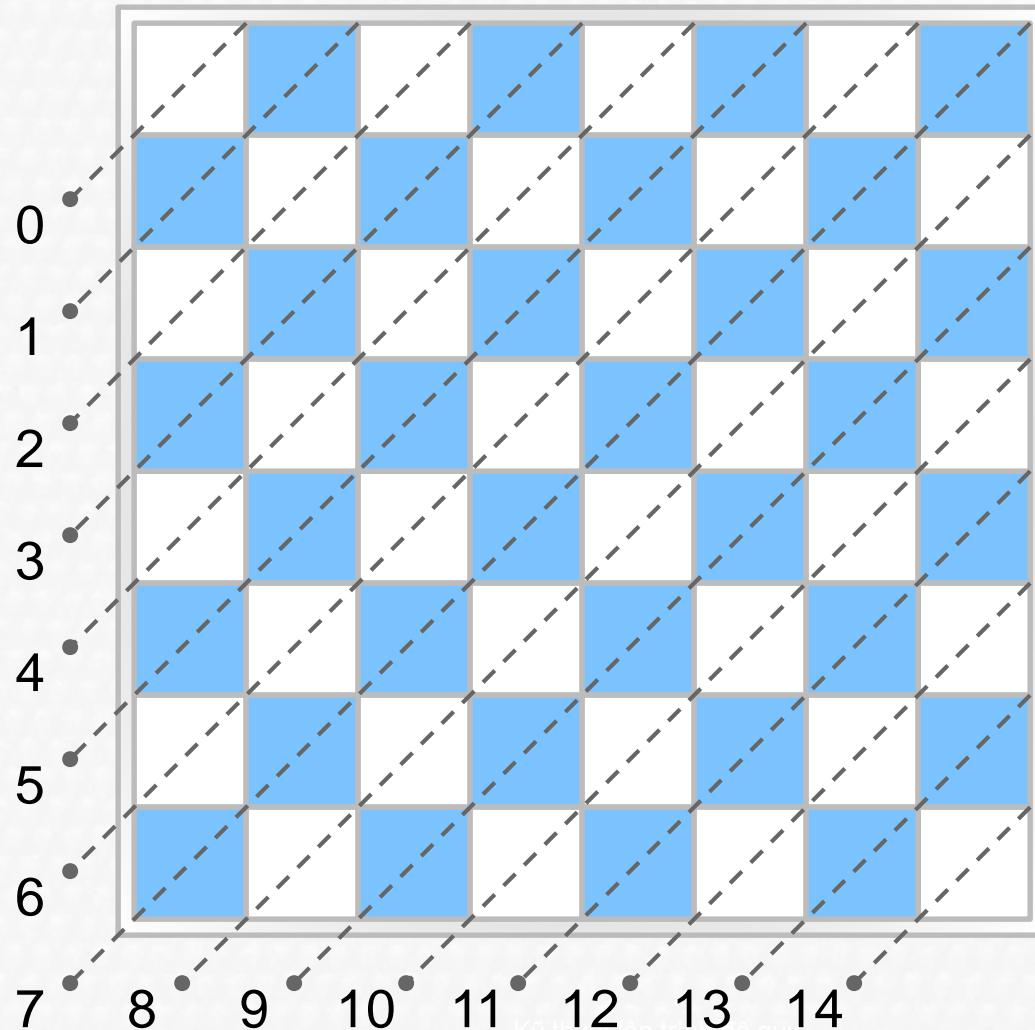


n đường

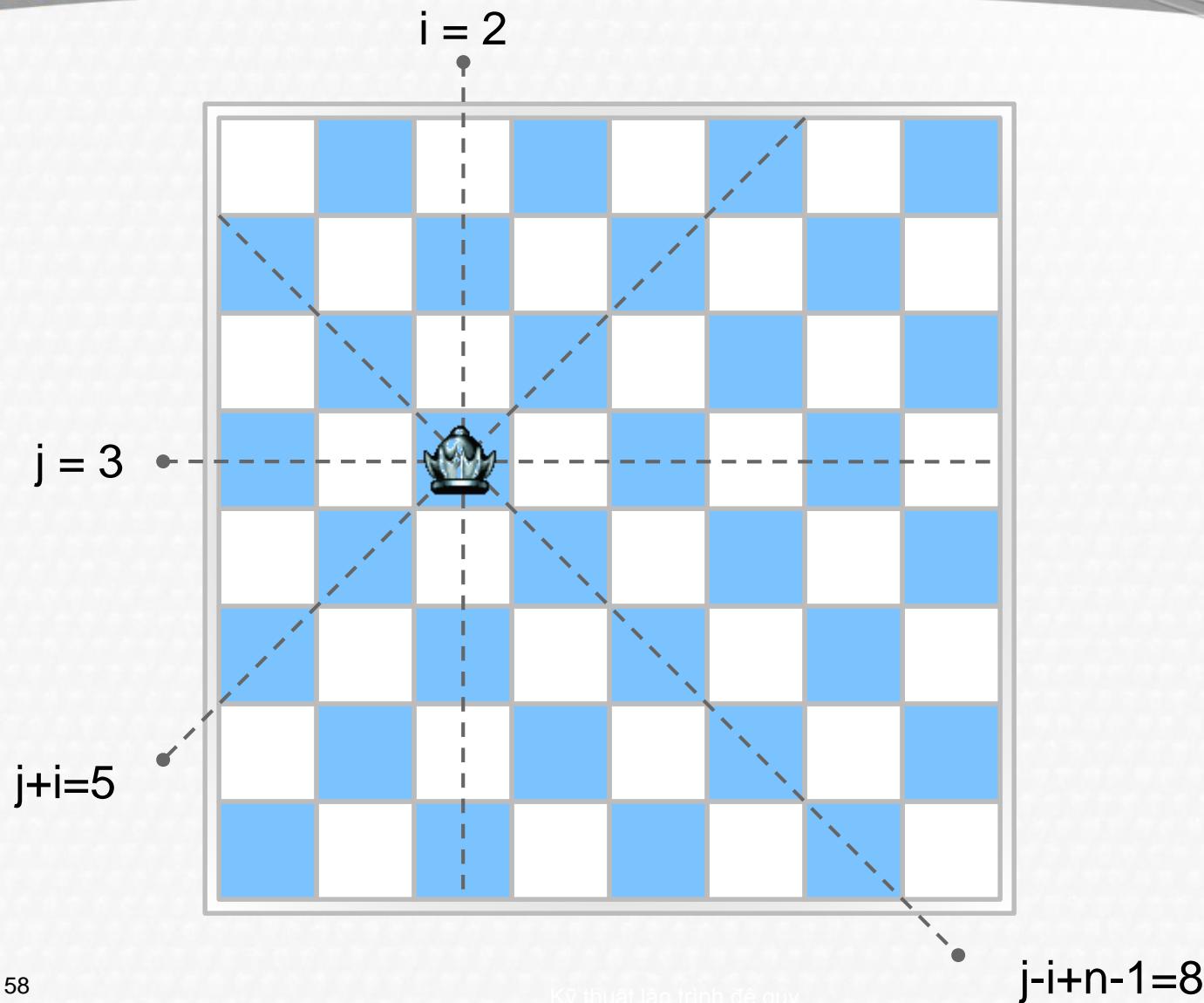
Tám hậu – Các đường chéo xuôi



Tám hậu – Các đường chéo ngược



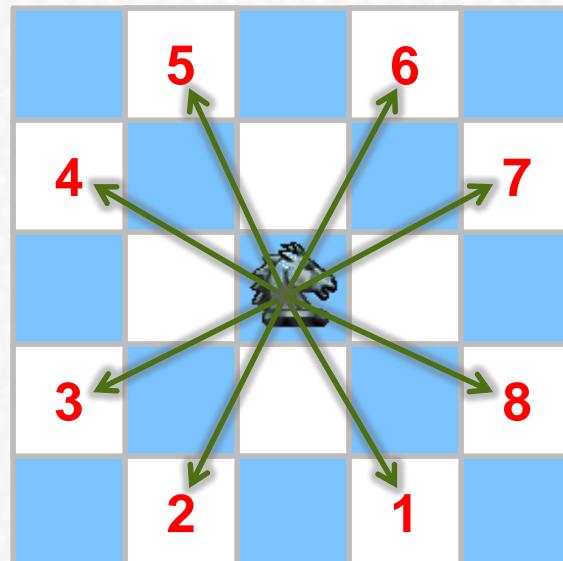
Tám hậu – Các dòng



Mã đi tuần

▪ 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:



Các vấn đề đệ quy thông dụng



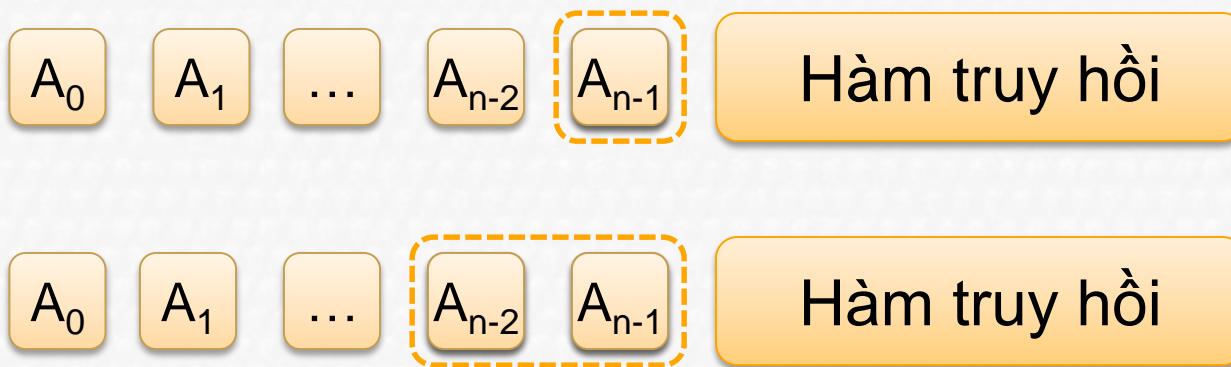
Đệ
quy??



a. 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:

- $V_h = 2V_{h-1}$
- $V_0 = 2$

➔ Đệ 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 $T(n)=1.12*T(n-1)$ và điều kiện dừng $V(0) = 1000$

b.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 → P1, P2, ..., Pn  
        for (int i=1;i<=n;i++)  
            Trị (Pi);  
    }  
    }  
    Tổng hợp kết quả
```

b.Chia để trị (divide & conquer)

▪ Ví dụ

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

▪ Giải pháp

- $\text{mid} = (\text{l} + \text{r}) / 2;$
 - Nếu $A[\text{mid}] = x \rightarrow$ trả về mid.
 - Ngược lại
 - Nếu $x < A[\text{mid}] \rightarrow$ tìm trong đoạn $[\text{l}, \text{mid} - 1]$
 - Ngược lại \rightarrow tìm trong đoạn $[\text{mid} + 1, \text{r}]$
- ➔ Sử dụng đệ quy nhị phân.

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.

c.Lần ngược (Backtracking)

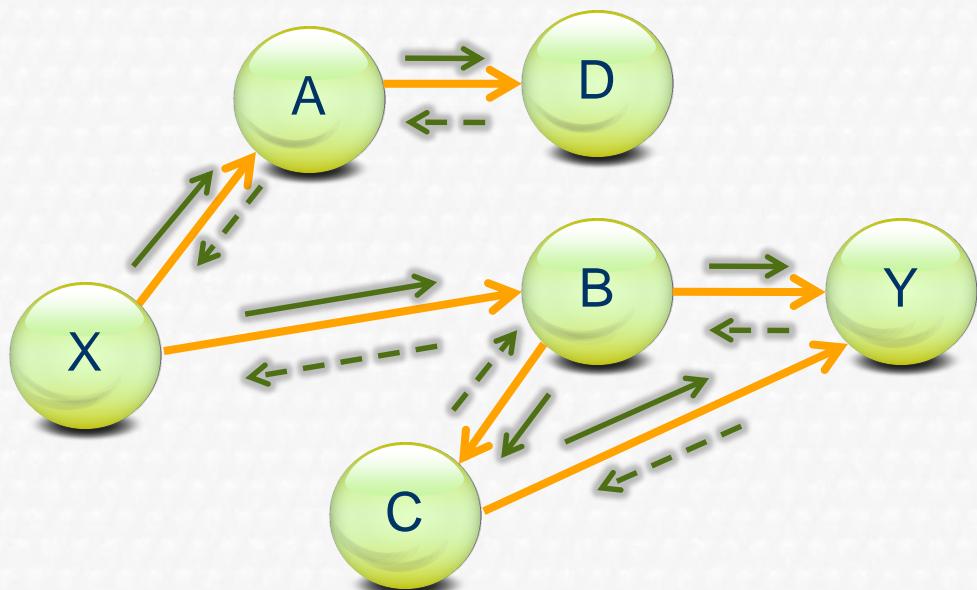
▪ Khái niệm

- 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.

c. Lần ngược (Back tracking)

■ Ví dụ

- Tìm đường đi từ X đến Y.



Thuật toán Thủ sai-Quay lui

```
void try(int i)
{ Duyệt < không gian trạng thái ứng viên ở bước i>
  { ghi nhận trạng thái chấp nhận được;
    if (i<n) try(i+1);
    else
      Kiểm tra nghiệm – xuất kết quả;
      xóa trạng thái đã ghi nhận;
  }
}
```

Nhận xét

▪ 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.

BÀI TẬP

LẬP CHƯƠNG TRÌNH CHO CÁC BÀI TẬP SAU

- **Bài 1:** Các bài tập trên mảng sử dụng đệ quy.
- **Bài 2:** Viết hàm đệ quy xác định chiều dài chuỗi.
- **Bài 3:** Hiển thị n dòng của tam giác Pascal.

$$- a[i][0] = a[i][i] = 1$$

$$- a[i][k] = a[i-1][k-1] + a[i-1][k]$$

Dòng 0: 1

Dòng 1: 1 1

Dòng 2: 1 2 1

Dòng 3: 1 3 3 1

Dòng 4: 1 4 6 4 1

LẬP CHƯƠNG TRÌNH CHO CÁC BÀI TẬP SAU

- **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 5:** Đổi 1 số thập phân sang cơ số khác.
- **Bài 6:** Tính các tổng truy hồi.
- **Bài 7:** Bài toán In ra hoán vị n số tự nhiên đầu tiên”.
- **Bài 8:** Bài toán “8 hậu”.
- **Bài 9:** Bài toán “Mã đi tuần”.

Chương III

DANH SÁCH

1. ĐỊNH NGHĨA

▪ Định nghĩa

- *Danh sách là một dãy hữu hạn các phần tử cùng loại được xếp theo một thứ tự tuyến tính.*

Danh sách L gồm các phần tử a_1, a_2, \dots, a_n

$$L = (a_1, a_2, \dots, a_n)$$

a_i : p.tử thứ i của danh sách,

n chiều dài của danh sách

1. ĐỊNH NGHĨA

- **Danh sách con**
- **Cho danh sách $L=(a_1, a_2, \dots, a_n)$,**
Danh sách $L'=(a_1, \dots, a_m)$, $1 \leq l \leq m \leq n$ được gọi là danh sách con của danh sách L
Ví dụ: $L=(6, 7, 9, 3, 5, 1, 8)$, $L'=(9, 3, 5)$
- **Các thao tác trên danh sách**

Tạo lập
Bổ sung
Loại bỏ

Sắp thứ tự
Tìm kiếm
Ghép danh sách

Trộn danh sách
Sao chép danh sách
...

1. ĐỊNH NGHĨA

▪ Danh sách đặc

❑ Các phần tử của danh sách nằm liên tiếp trong bộ nhớ.

– Tổ chức lưu trữ: mảng

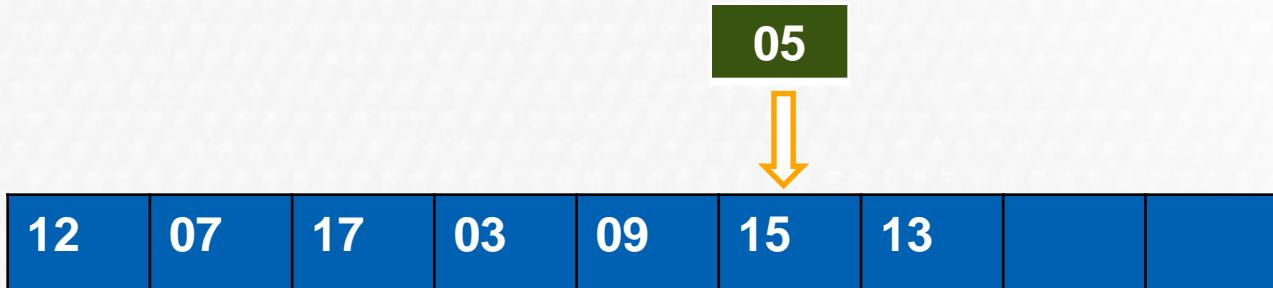
➤ Chú ý: Điểm khác biệt: Cấu trúc mảng và mô hình danh sách: **mảng số phần tử cố định, số phần tử của danh sách thay đổi** theo các thao tác chèn, xóa

▪ Danh sách liên kết:

❑ Các phần tử của danh sách liên kết với nhau qua thành phần chứa địa chỉ.

2. DANH SÁCH ĐẶC

- Chèn thêm phần tử



- Xóa phần tử



2. DANH SÁCH ĐẶC

■ Xác định địa chỉ của phần tử

– Mảng 1 chiều: Địa chỉ phần tử thứ i

- $\text{Add}(i) = l_0 + (i-1)d$

– Mảng 2 chiều $A_{n \times m}$: Địa chỉ phần tử thứ (i,j)

- $\text{Add}(i,j) = l_0 + (i-1)m.d + (j-1).d$

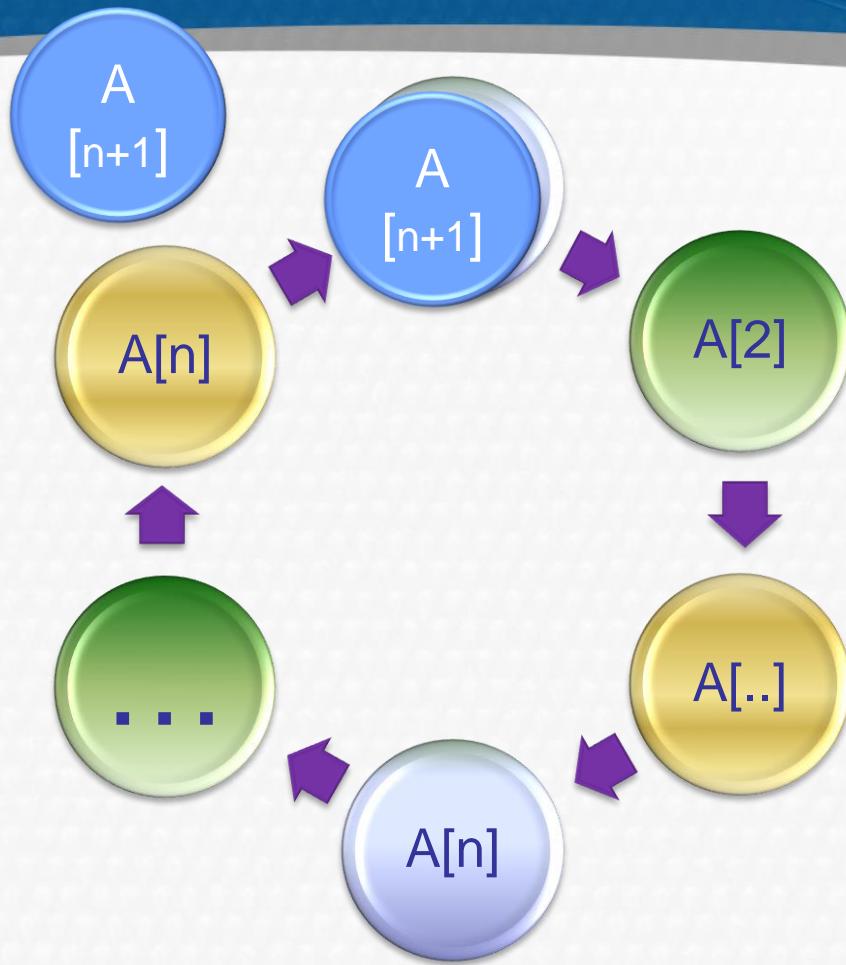
• Tổ chức lưu trữ:

- Mảng

- Chú ý trường hợp tràn (overflow):

Tổ chức lưu trữ vòng

2. DANH SÁCH ĐẶC



2. DANH SÁCH ĐẶC

▪ Ưu nhược điểm của danh sách đặc:

➤ Ưu:

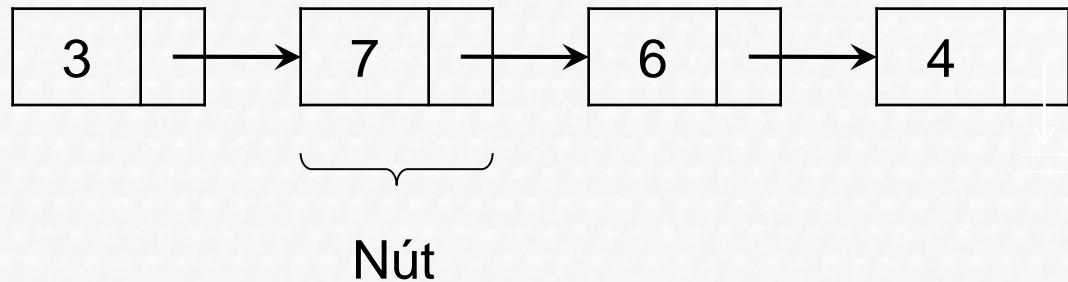
- Truy xuất các phần tử trực tiếp theo vị trí
- Có các thuật toán hiệu năng cao thao tác trên mảng: tìm kiếm, sắp xếp..
- Mật độ sử dụng 100%

➤ Nhược

- Các thuật toán chèn, xóa có độ phức tạp $O(n)$
- Lãng phí không gian nhớ khi có nhiều phần tử mang cùng giá trị.

3. DANH SÁCH LIÊN KẾT

■ 3.1 Danh sách liên kết đơn



- Các phần tử của một danh sách liên kết không cố định
→ không gian nhớ cấp phát cho ds liên kết phải **được cấp phát động**

3. DANH SÁCH LIÊN KẾT

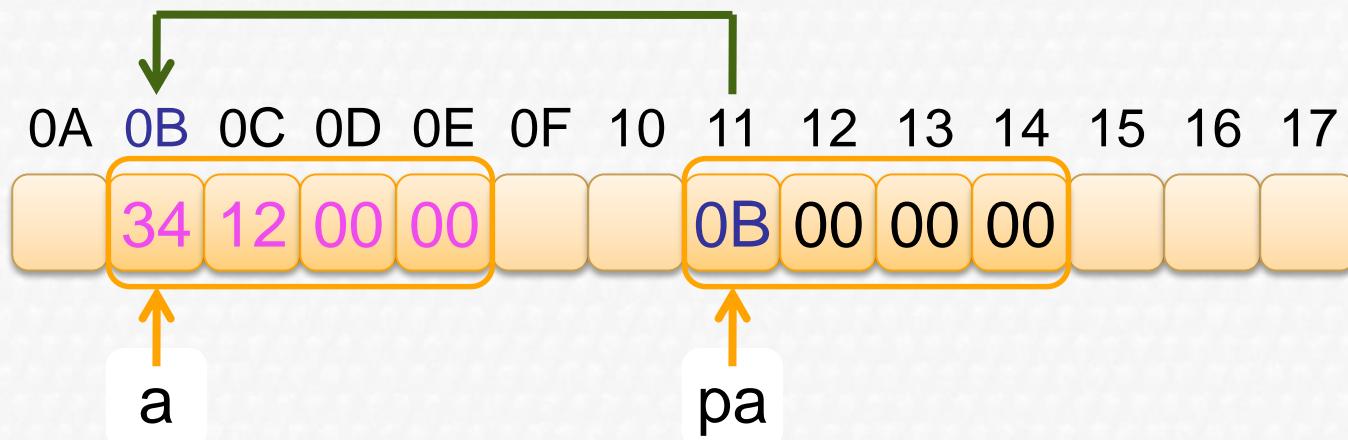
▪ Cấp phát động, biến kiểu con trỏ

Ô nhớ cấp phát động là những ô nhớ được cấp phát và thu hồi bằng lệnh trong chương trình. Để quản lý các ô nhớ cấp phát động sử dụng biến con trỏ:

- *Biến kiểu con trỏ dùng để lưu trữ địa chỉ của các biến khác. Mỗi biến kiểu con trỏ quản lý một ô nhớ cấp phát động có kiểu xác định.*

3. DANH SÁCH LIÊN KẾT

- Khai báo kiểu và biến con trỏ trong C
 - <kiểu dữ liệu> *<tên biến con trỏ>;
- Ví dụ: int *pa, a; // {a=0x1234; pa=&a;}
- p là biến con trỏ, dùng để chứa địa chỉ của một vùng nhớ (*p) chứa một số nguyên



3. DANH SÁCH LIÊN KẾT

■ Sử dụng từ khóa typedef

```
typedef <kiểu dữ liệu> *<tên kiểu con trỏ>;  
<tên kiểu con trỏ> <tên biến con trỏ>;
```

■ Ví dụ

```
typedef int *pint;  
int *p1;  
pint p2, p3;
```

■ Lưu ý khi khai báo kiểu dữ liệu mới

- Giảm bối rối khi mới tiếp xúc với con trỏ.
- Nhưng dễ nhầm lẫn với biến thường.

3. DANH SÁCH LIÊN KẾT

- Tạo vùng nhớ động để lưu trữ dữ liệu:

<biến con trỏ> = new <kiểu dữ liệu (biến con trỏ)>

Ví dụ: p= new int;

- Hằng NULL:

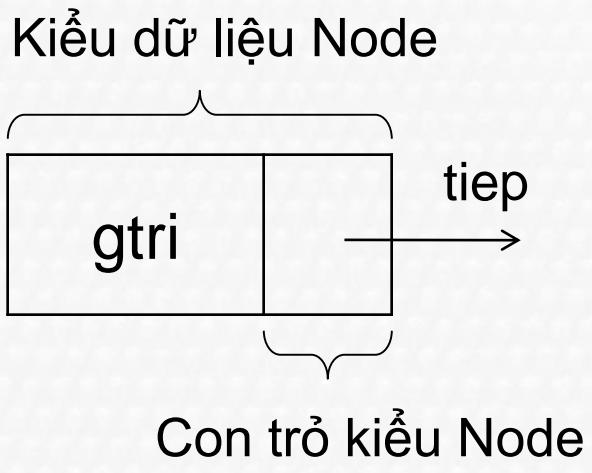
Khi biến con trỏ không mang địa chỉ của vùng nhớ nào (**rỗng**), được gán mang giá trị **NULL**

- Xóa vùng nhớ động

delete <biến con trỏ>

3. DANH SÁCH LIÊN KẾT

■ a. Khai báo cấu trúc nút



```
struct nut {  
    int gtri;  
    nut *tiep;}  
  
typedef nut Node;
```

3. DANH SÁCH LIÊN KẾT

- b. Khai báo biến con trỏ chứa đ/c nút đầu danh sách

Node *dau;

- c. Khởi tạo giá trị ban đầu

```
void khoitao(Node *&dau)  
{ dau=NULL;}
```

- d. Tạo danh sách

Nhập vào một dãy số nguyên, kết thúc việc nhập khi nhập giá trị 0. Các số nguyên được lưu thành một ds liên kết đơn.

3. DANH SÁCH LIÊN KẾT

```
void nhapds(Node *&dau)
{
    int tam;
    Node *p, *q;
    do { printf("Nhập 1 số nguyên (0:dung) ");
          scanf("%d",&tam);
          if (tam!=0) { p= new Node;
                         p->gtri=tam; p->tiep=NULL;
                         if (dau==NULL) dau=p;
                         else q->tiep= p;
                         q=p; } } while (tam!=0); }
```

3. DANH SÁCH LIÊN KẾT

▪ e. Duyệt danh sách

C1: *void duyetsds(Node *dau)*

{ Node *p;

p=dau;

while (p!=NULL)

{ Xử lý nút p; *p=p->tiep;*} }

C2: *void duyetsdsdq (Node *dau)*

{ if (*dau!=NULL*) {Xử lý nút *dau*;

duyetsdsdq(dau->tiep);}

3. DANH SÁCH LIÊN KẾT

- f. Tìm nút đầu tiên mang giá trị x

```
Node *tim(Node *dau, int x)
```

```
{ Node *p;
```

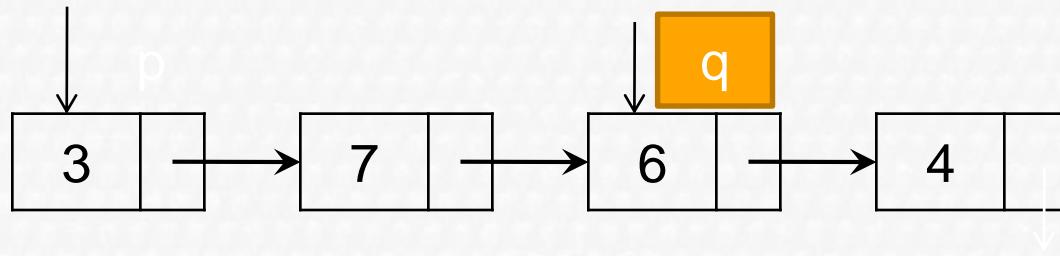
```
p=dau;
```

```
while (p!=NULL) && (p->gtri!=x)
```

```
p=p->tiep;
```

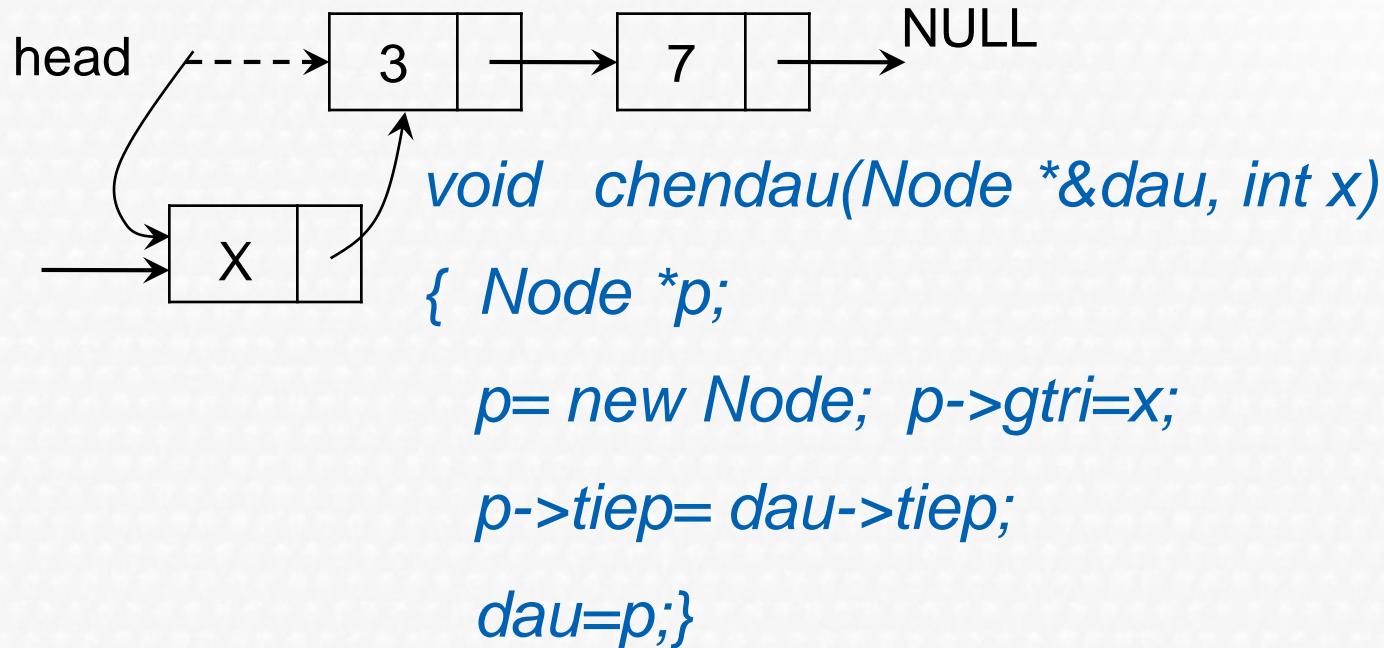
```
return p; }
```

```
q= tim(dau,6);
```



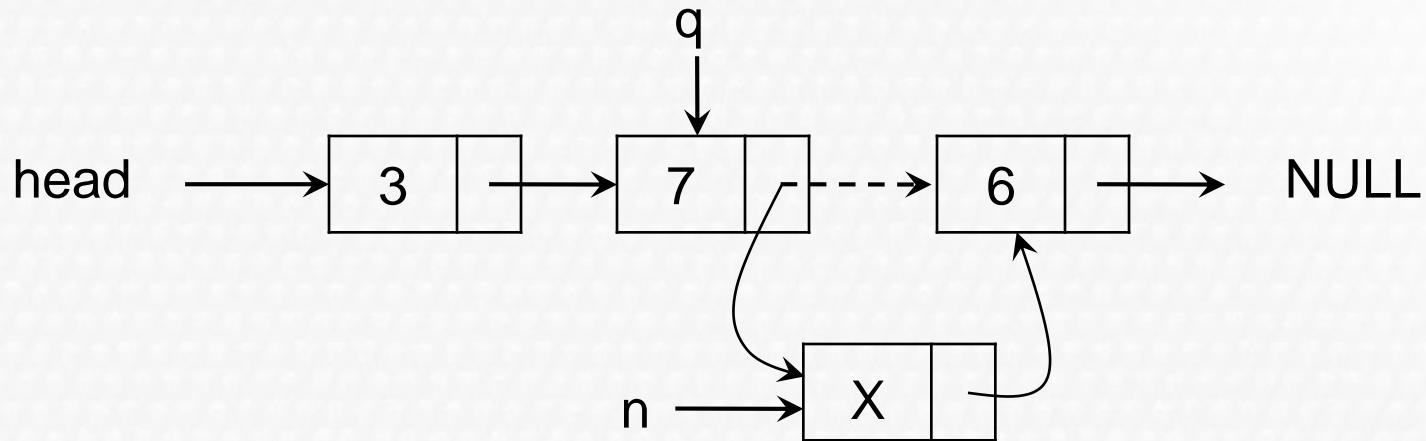
3. DANH SÁCH LIÊN KẾT

- g. Chèn một nút vào danh sách
- g1. Chèn một nút vào đầu danh sách:



3. DANH SÁCH LIÊN KẾT

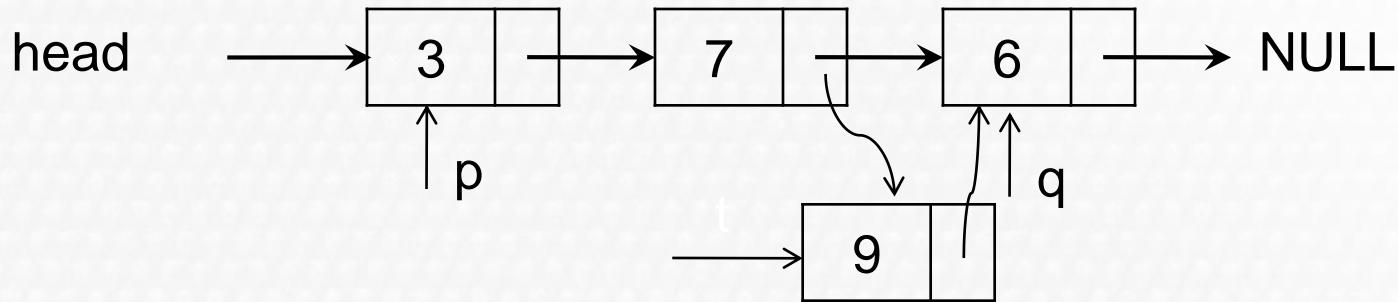
- g2. Chèn một nút vào sau nút q:



```
void chensau(Node *&dau, Node *q, int x)
{ Node *n;
n= new Node; n->gtri=x; n->tiep= q->tiep;
q->tiep=n;}
```

3. DANH SÁCH LIÊN KẾT

- g3. Chèn một nút vào trước nút q:



```
void chentruoc(Node *&dau, Node *q, int x)
{ Node *p, *t;
  p=dau;
  while (p->tiep !=q) p=p->tiep;
  t=new Node; t->gtri=x; t->tiep=q; p->tiep=t;}
```

3. DANH SÁCH LIÊN KẾT

- h. Xóa nút

- h1. Xóa nút đầu ds

- h2. Xóa nút q

- h3. Xóa nút đứng sau nút q

- h4. Xóa nút đứng trước nút q

Thao tác tương tự như chèn

3. DANH SÁCH LIÊN KẾT

i. Sắp xếp danh sách, Ý tưởng: Buble Sort

```
node *sapxep (node *dau)
```

```
{ node *p, *q;    int tam;
```

```
    p=dau;
```

```
    while (p->tiep !=NULL)
```

```
    { q=p->tiep;
```

```
        while (q!=NULL)
```

```
        { if (p->gtri > q->gtri)
```

```
            { tam=p->gtri; p->gtri=q->gtri; q->gtri=tam;}
```

```
            q=q->tiep;}
```

```
        p=p->tiep;}
```

```
    return dau;
```

3. DANH SÁCH LIÊN KẾT

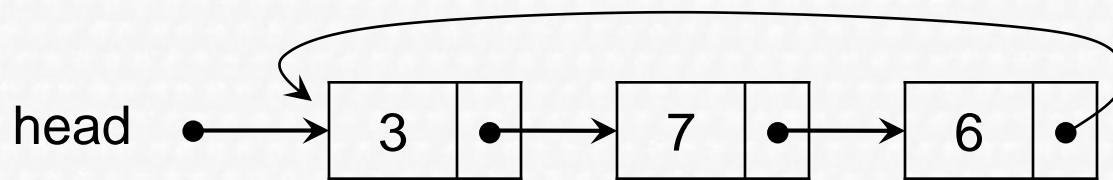
j. Đảo ngược danh sách:

```
void daonguoc(node *&dau)
{node *p, *q;
 q=NULL;
 while (dau!=NULL)
 { p=dau;  dau=dau->tiep; p->tiep=q;  q=p; }
dau=p;
}
```

3. DANH SÁCH LIÊN KẾT

2. Danh sách liên kết vòng:

Nút cuối của danh sách liên kết vòng không trỏ đến NULL mà trỏ về lại nút đầu.



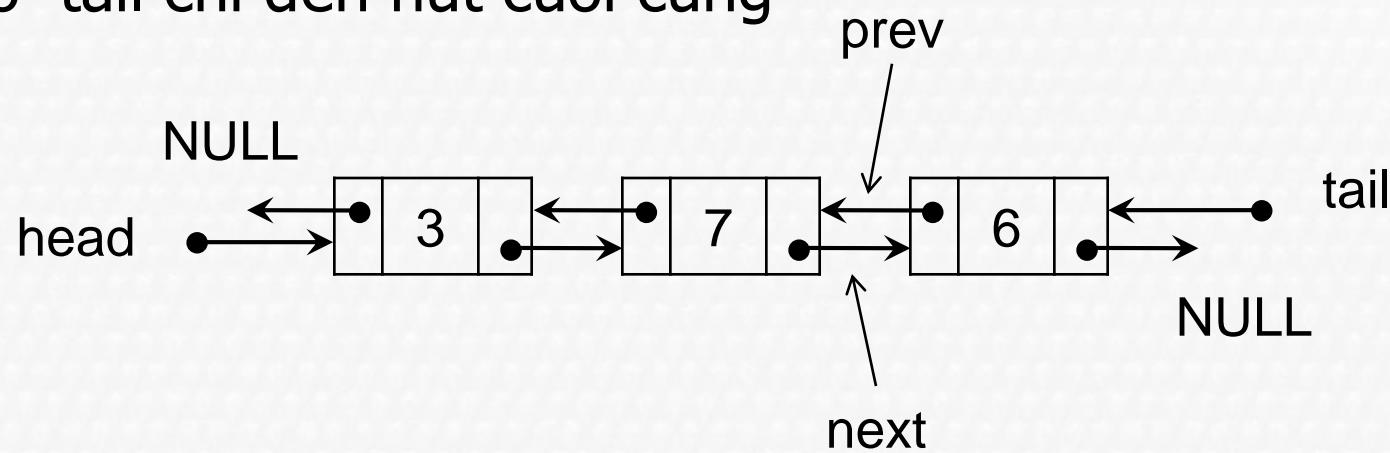
3. DANH SÁCH LIÊN KẾT

3. Danh sách liên kết kép:

Nút của danh sách liên kết kép gồm 2 con trỏ

- next : trỏ đến nút đứng sau
- prev: trỏ đến nút đứng trước

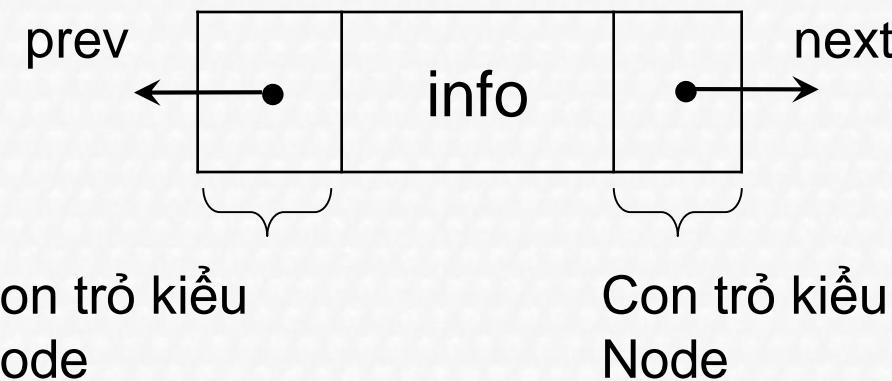
Con trỏ tail chỉ đến nút cuối cùng



3. DANH SÁCH LIÊN KẾT

Cấu trúc nút của danh sách liên kết kép:

```
struct Node  
{    int info;  
    Node *prev;  
    Node *next;  
};
```



3. DANH SÁCH LIÊN KẾT

a. Ưu điểm:

- Không thực hiện các thao tác dời mảng khi thêm hay xóa phần tử.
- Không hao phí bộ nhớ.

b. Nhược điểm:

- Không truy cập trực tiếp phần tử bằng chỉ số.

4. ỨNG DỤNG DS

A. Ngăn Xếp (Stack)

- 1) Khái niệm ngăn xếp
- 2) Ngăn xếp thực hiện bằng mảng
- 3) Ứng dụng của ngăn xếp

B. Hàng Đợi (Queue)

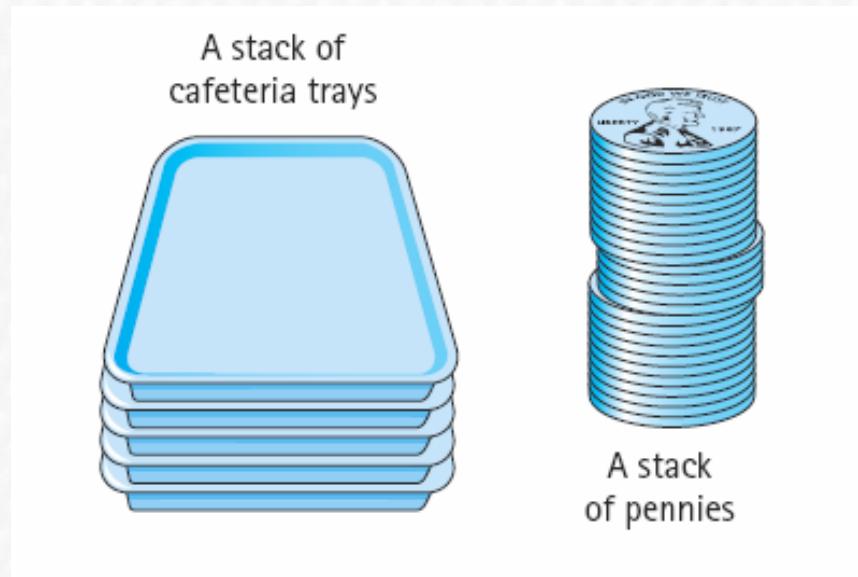
- 4) Khái niệm hàng đợi
- 5) Hàng đợi thực hiện bằng mảng
- 6) Ứng dụng của hàng đợi

a. Ngăn Xếp (Stack)

1. Khái niệm ngăn xếp:

Ngăn xếp là danh sách đặc biệt: **thêm và xóa phần tử được thực hiện theo nguyên tắc vào sau ra trước (LIFO – Last In First Out)**

Ví dụ: chồng đĩa



Thao tác trên Stack

- Push(x): thêm phần tử x vào Stack
- Pop(x) : lấy ra phần tử từ Stack cho vào biến x.
- View(x): xem phần tử kế tiếp sẽ được lấy ra.

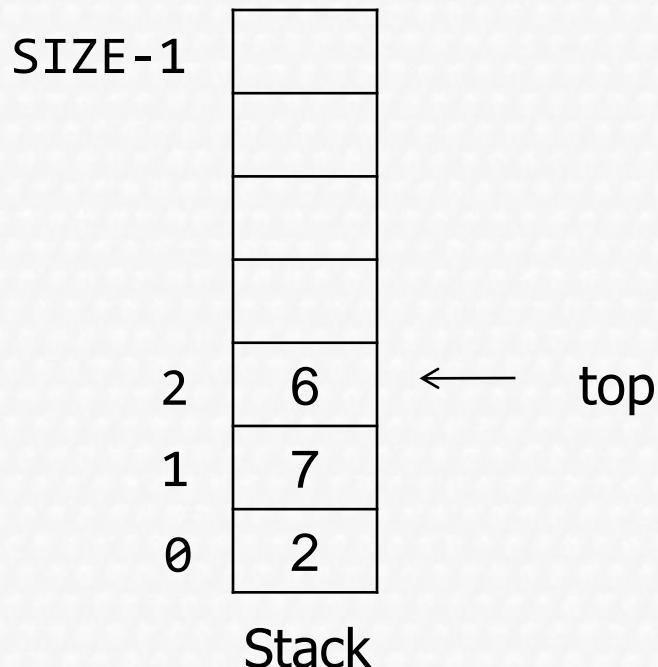
2) Biểu diễn ngăn xếp bằng mảng

- Ngăn xếp các số nguyên:

```
#define SIZE 20
```

```
struct Stack
```

```
{   int a[SIZE];      // Stack có kích thước là SIZE  
    int top;          // vị trí của đầu Stack  
};
```



top= -1 : stack rỗng
top= SIZE-1: stack đầy

- Cài đặt stack số nguyên

```
void Init(Stack &s)  
{   s.top= -1;}  
  
int Push(Stack &s, int x)  
{   if (s.top < SIZE- 1) {    // stack chưa đầy  
        s.top++;           // vị trí phần tử mới  
        s.a[s.top] = x;     // đưa pt mới vào stack  
        return 1;  
    }  
    else  return 0;         // stack đầy  
}
```

```
int Pop(Stack &s, int &x)
{
    if (s.top == -1) // stack rỗng, return 0
        return 0;
    else {
        x = s.a[s.top]; // lấy phần tử khỏi stack
        s.top = s.top - 1; // chỉ đến phần tử tiếp theo
        return 1; // Pop thành công return 1
    }
}
```

```
void main()
{
    Stack s;
    Init(s);
    Push(s, 2); // phần tử của Stack được đưa vào x
    Push(s, 3);
    while (Pop(s, x) == 1) printf("%d ", x);
}
```

Giống hàm Pop, xem nhưng
không lấy phần tử khỏi stack

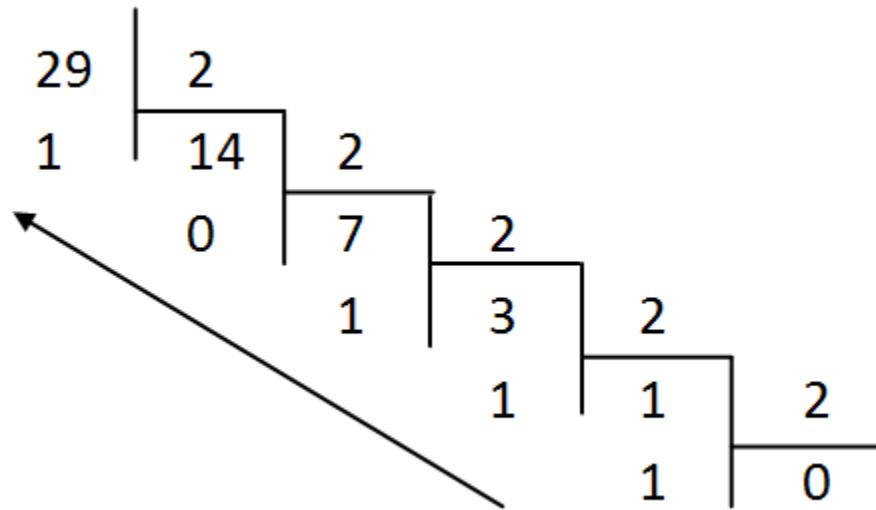


```
int View(Stack &s, int &x)
{
    if (s.top== -1)
        return 0;
    else {
        x = s.a[s.top];
        return 1;
    }
}
```

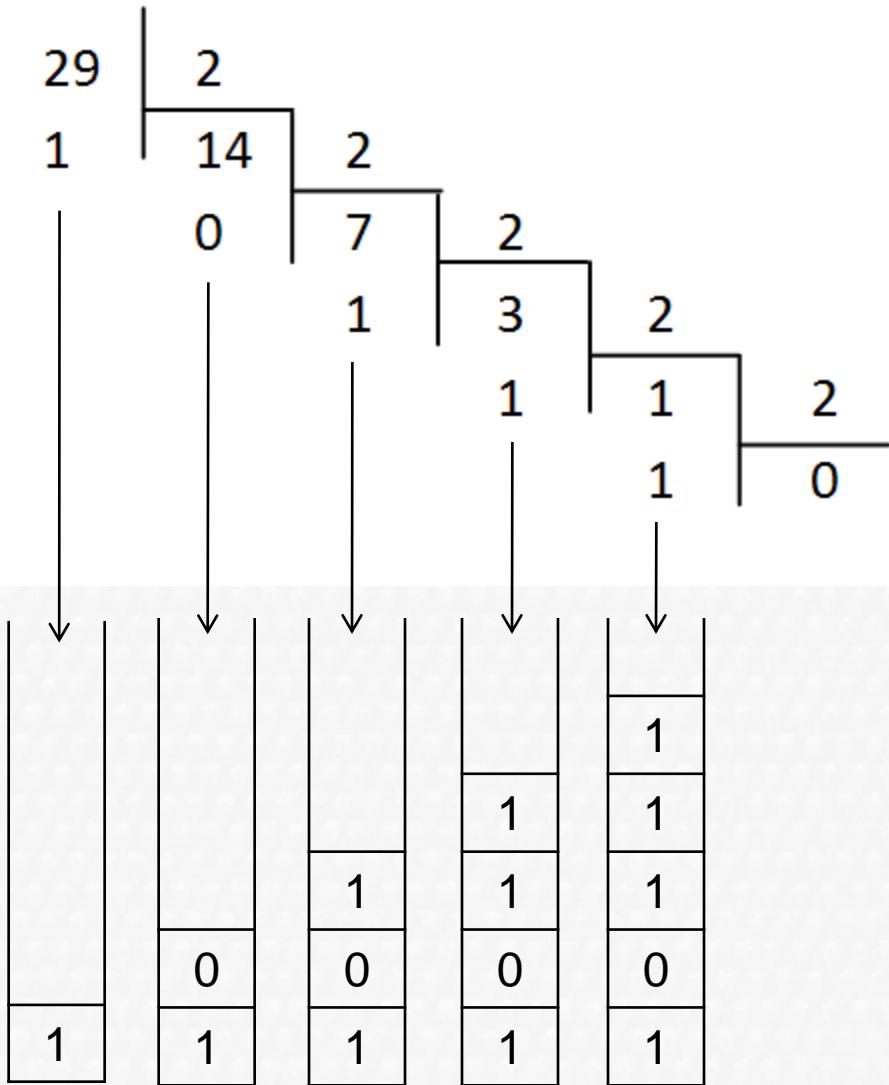
3) Ứng dụng của ngăn xếp

(Đổi số thập phân sang nhị phân)

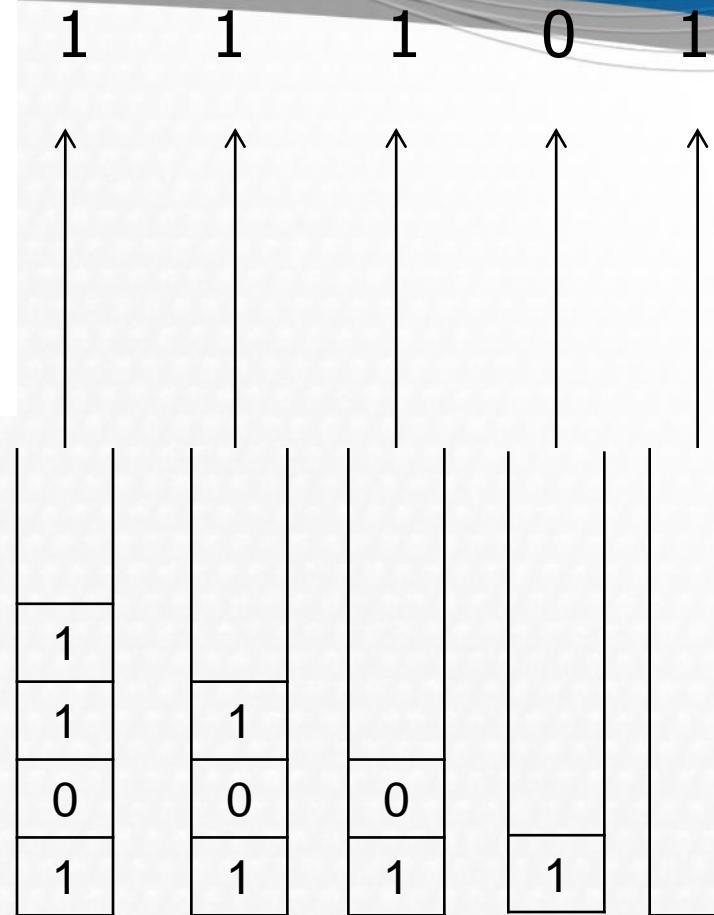
Ví dụ: đổi số 29 sang nhị phân



Kết quả: 11101



Push



Pop

```
void main()
{
    Stack s;
    int n, c;
    printf("Nhập vào số nguyên n:");
    scanf("%d", &n);
    Init(s);
    while (n>0) {
        Push(s, n%2);      // đưa phần dư vào stack
        n= n/2;           // tính phần thương }
    printf("Số nhị phân: ");
    while (Pop(s, c))
        printf("%d", c);  }
}
```

4) Ứng dụng của ngăn xếp

(Biểu thức hậu tố - ký pháp Ba Lan - *Polish notation*)

Jan Łukasiewicz -1920

Biểu thức trung tố:

Toán hạng

Toán tử

Toán hạng

Ví dụ: 6+7, A*B, 3^8, A+B*C - D

Để thay đổi thứ tự tính toán, dùng dấu ngoặc: (A+B)*(C - D)

b) Biểu thức hậu tố:

Toán hạng

Toán hạng

Toán tử

Ví dụ: 6 7 +, A B * , 3 8 ^,

Trung tố	Hậu tố	Bỏ ngoặc
$A + B * C$	$A (B C *) +$	$A B C * +$
$(A + B) * C$	$(A B +) C *$	$A B + C *$

Tính chất của biểu thức hậu tố:

- Được tính từ trái qua phải
- Không có thứ tự ưu tiên giữa các phép tính → không cần dấu ngoặc

Ví dụ: tính giá trị biểu thức: 1 2 3 * + 4 -

Lần tính	Kết quả
1	1 6 + 4 -
2	7 4 -
3	3

Bài tập:

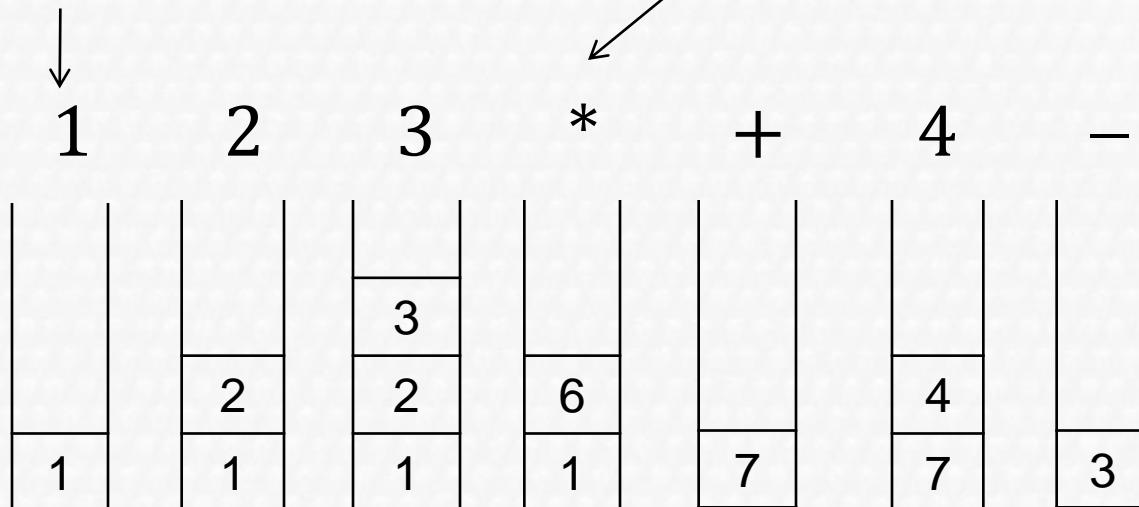
1. Tính: 1 2 3 4 - * +
2. Tính: 1 5 + 8 4 1 - - *

b) Dùng Stack tính giá trị biểu thức hậu tố:

Ví dụ: tính giá trị biểu thức $1\ 2\ 3\ *\ +\ 4\ -$

Toán tử: - Lấy 2 phần tử từ stack, tính.
- Đưa kết quả vào lại stack

Toán hạng: đưa vào stack



$$2\ 3\ *\ =6$$

$$7\ 4\ -\ =3$$

$$1\ 6\ +\ =7$$

Bài tập:

1. Tính: 1 2 3 4 - * +

2. Tính: 1 5 + 8 4 1 - - *

b) Chuyển đổi biểu thức trung tố thành hậu tố:

Xét biểu thức $A + (B * E - C * D / F)$

Toán hạng: → kết quả

Kết thúc: pop hết stack

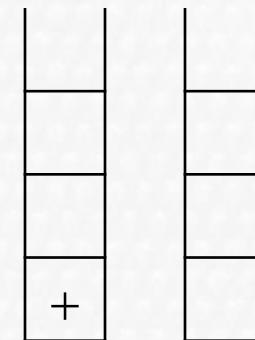
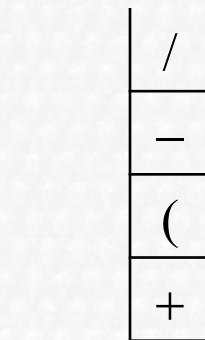
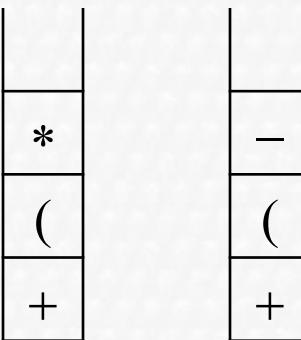
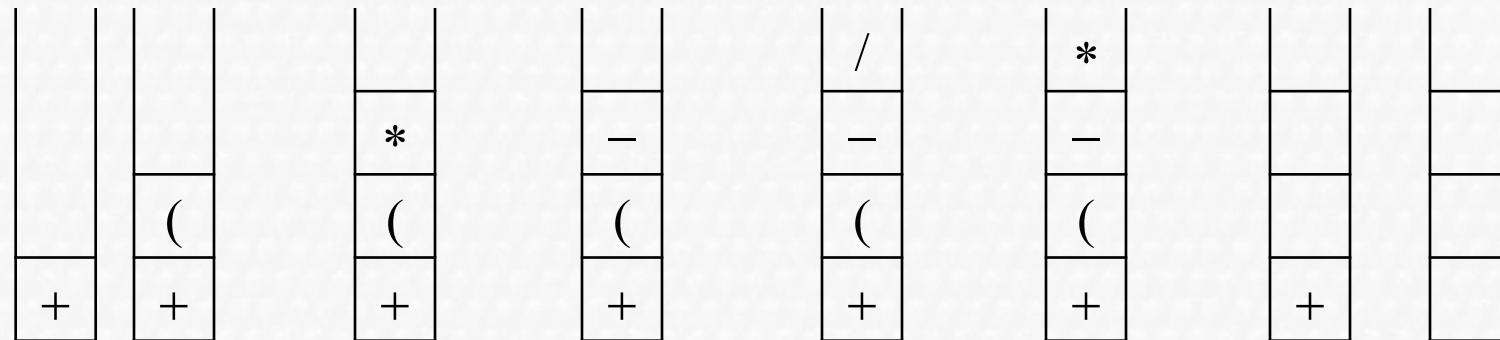
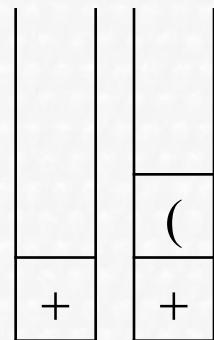
Toán tử: push stack

Dấu (: push stack

Dấu): pop stack
cho đến dấu (

Độ ưu tiên \leq toán tử ở stack:
pop stack → kết quả

A + (B * C - D / E * F)



A B C * D E / F * - +

Các thao tác thực hiện khi duyệt biểu thức trung tố

- Nếu là toán hạng: ghi ra biểu thức hậu tố kết quả
- Nếu là dấu (: push stack
- Nếu là dấu) : pop stack cho đến khi gặp dấu (, ghi ra biểu thức kết quả
- Nếu kết thúc biểu thức trung tố: pop hết stack, ghi ra biểu thức kết quả
- Nếu là toán tử:
 - Nếu toán tử trên đỉnh stack có độ ưu tiên cao hơn pop stack, ghi ra biểu thức kết quả
 - Push toán tử vào stack

Bài tập đổi biểu thức trung tố thành hậu tố

- 1) $A / (B + C * D - E) * F$
- 2) $(A - B) * (C - D * E + F) + G$

b. Hàng Đợi (Queue)

1. Khái niệm hàng đợi:

Hàng đợi là danh sách đặc biệt: thêm và xóa phần tử được thực hiện theo nguyên tắc **vào trước ra trước** (**FIFO** – First In First Out)

Ví dụ: xếp hàng

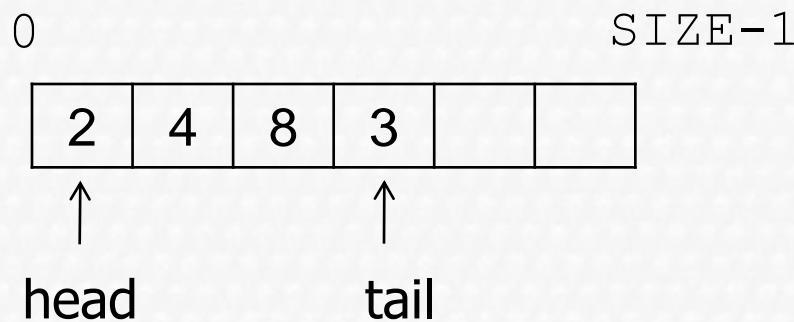


Thao tác trên Queue:

- Push(x): thêm một phần tử x vào Queue
- Pop(x) : lấy ra phần tử x ra khỏi Queue
- View(x): xem phần tử kế tiếp.

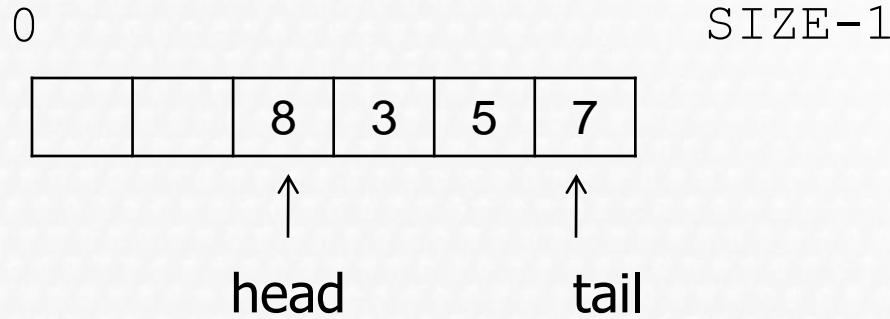
2) Biểu diễn hàng đợi bằng mảng

```
#define SIZE 20  
  
struct Queue  
{    int a[SIZE];      // Queue có kích thước là SIZE  
    int head, tail;    // Vị trí của đầu và cuối của queue  
};
```



Vấn đề 1:

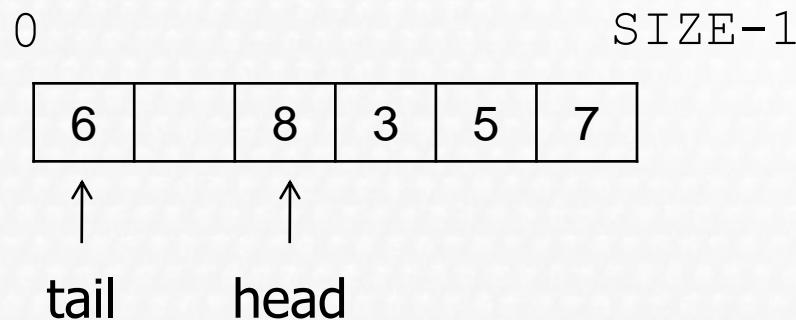
Khi pop Queue: nếu giữ head cố định và dồn mảng → tốn kém → tăng giá trị head sau khi pop



Vấn đề 2:

Nếu tail chỉ đến cuối mảng: cần push queue?

→ cho tail chỉ về đầu mảng



Các trường hợp của head và tail:

- Nếu $\text{head} = \text{tail}$: Queue có 1 phần tử
- Nếu $(\text{tail} + 1) \bmod \text{SIZE} = \text{head}$: Queue đầy
- Queue rỗng? $\text{Head} = \text{tail} = -1$

Cài đặt Queue

```
void Init(Queue &q)
{   q.head= q.tail= -1;}
```

```
int Push(Queue &q, int x)
{
    if ((q.tail + 1)% SIZE == head)
        return 0;                      // queue đầy.
    else {
        q.tail = (q.tail+ 1)% SIZE;    // vị trí phần tử mới.
        q.a[q.tail] = x;               // lưu phần tử mới.
        if (q.head == -1) q.head = 0;
        return 1;
    }
}
```

nếu trước đó queue rỗng
cần cập nhật head

```
int Pop(Queue &q, int &x)
{
    if ((q.head == -1))
        return 0;
    else {
        x = q.a[q.head];
        if (q.head == q.tail)
            q.head = q.tail = -1;
        else
            q.head = (q.head + 1) % SIZE;
    }
    return 1;
}
```

Queue rỗng

Nếu phần tử lấy ra
là pt cuối thì cập
nhật queue rỗng

Vị trí phần tử tiếp
theo sẽ được Pop

```
int View(Queue &q, int &x)
{
    if ((q.head== -1) return 0;
    else {
        x = q.a[q.head];
        return 1;
    }
}
```

BÀI TẬP

Dùng stack, viết chương trình phân tích 1 số thành thừa số nguyên tố theo thứ tự lớn trước nhỏ sau.

Ví dụ $n = 3960$, hiển thị: $3960 = 11 * 5 * 3 * 3 * 2 * 2 * 2$

Cám ơn đã theo dõi

