

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT



GV : ThS. Trần Văn Thọ

E-mail : thotv@hufi.edu.vn

NỘI DUNG MÔN HỌC

Chương 1: TỔNG QUAN (OVERVIEW)

Chương 2: KỸ THUẬT TÌM KIẾM (SEARCHING)

Chương 3: KỸ THUẬT SẮP XẾP (SORTING)

Chương 4: DANH SÁCH (LIST)

Chương 5: CÂY (TREE)

Chương 6: BẢNG BĂM (HASH)

TÀI LIỆU HỌC TẬP

- [1] Trần Văn Thọ, *Cấu trúc dữ liệu và giải thuật*, Trường ĐH CNTP Tp.HCM, 2019.
- [2] Trần Văn Thọ, *Thực hành cấu trúc dữ liệu và giải thuật*, Trường ĐH CNTP Tp.HCM, 2019.

TÀI LIỆU THAM KHẢO

- [1]. Trần Hạnh Nhi và Dương Anh Đức, *Nhập môn cấu trúc dữ liệu và thuật toán*.
- [2]. Đỗ Xuân Lôi, *Cấu trúc dữ liệu và giải thuật*, NXB Khoa học kỹ thuật, 1996.
- [3]. Nguyễn Trung Trực, *Cấu trúc dữ liệu*. Đại học bách khoa TP. HCM.
- [4]. Nguyễn Hồng Chương, *Cấu trúc dữ liệu ứng dụng và đặt bằng C*, Nhà xuất bản Thành phố Hồ Chí Minh, 2003.
- [5]. Nguyễn Việt Hương, *Ngôn ngữ lập trình C++ và cấu trúc dữ liệu*, Nhà xuất bản Giáo Dục, 2000.

TÀI LIỆU THAM KHẢO

- [6]. Tác giả Robert Sedgewick, *Cẩm nang thuật toán* (bản dịch),
- [7]. Tác giả MARK ALLEN WEISS, *Data Structures and Algorithm Analysis*, InC. The Benjamin/Cummings Publishing Company, 1993.
- [8]. William J.Collins, *Data Structures an object Oriented Approach*, Addison Wesley, 1992.
- [9]. *Handbook of Algorithms and Data Structures*, Informatik, ETH Zurich, 1999. Gaston H.Gonnet.
- [10]. Tác giả N. Width, *Datastruct & Algorithm = Programs*.

Môn: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT



CHƯƠNG 1: TỔNG QUAN

NỘI DUNG CHƯƠNG 1

1.1. Một số bài toán

1.2. Giới thiệu vai trò của cấu trúc dữ liệu

1.3. Mối quan hệ giữa cấu trúc dữ liệu và giải thuật

1.4. Xây dựng giải thuật

1.5. Cấu trúc dữ liệu cơ sở

1.6. Bài tập

Một số bài toán

❖ Ví dụ 1: Sắp xếp danh sách tuyển sinh

Năm 2016, ĐHCNTP có N thí sinh tham gia tuyển sinh, hãy viết chương trình sắp xếp các thí sinh theo thứ tự giảm dần của tổng điểm thi ba môn: Toán, Lý, Hóa

Stt	Họ tên	Toán	Lý	Hóa	Tổng
1	Trần Anh Tuấn	7	8	7	22
2	Bùi Ngọc Thăng	10	10	9	29
3	Lê Sỹ Vinh	10	8	8	28
4	Nguyễn Thị Ánh	8	10	9	27
...

Một số bài toán

❖ Ví dụ 2: Sắp xếp danh sách website (google search)

Google có danh sách N website. Website x có một độ ưu tiên là $f(x)$. Hãy sắp xếp các website trên theo độ ưu tiên giảm dần.

Một số bài toán

❖ Ví dụ 3: Danh bạ điện thoại

Viết một chương trình quản lý danh bạ điện thoại của toàn bộ thành phố Hồ Chí Minh, sao cho các thao tác sau được hiệu quả nhất:

1. Kiểm traMột số điện thoại.
2. ThêmMột số điện thoại.
3. XóaMột số điện thoại.

Một số bài toán

❖ Ví dụ 4: Tìm đường đi ngắn nhất

Xây dựng hệ thống phần mềm chỉ đường đi tốt nhất cho người dùng khi muốn di chuyển giữa 2 địa điểm A và B

1. Đường đi ngắn nhất.
2. Đường đi qua ít đèn xanh – đèn đỏ nhất.
3. Đường đi ít tắc (ít kẹt xe) nhất .

Một số bài toán

140 Lê Trọng Tấn, Tây Thạnh, Tân Phú, Ho Chi Minh City, Vietnam to 120 Trường Chinh, Quận 11, Ho Chi Minh City, Vietnam - Google Maps - Mozilla Firefox

File Edit View History Bookmarks Yahoo! Tools Help

https://maps.google.com

Most Visited Getting Started Latest Headlines NEWS NEWS-EN IT HUTECH EMAIL TRAINING MUSIC VIDEO RELAX UPLOAD WEB DEV OTHER localhost

Share page: 0937330664

tvtho2000 - Yahoo! ... 140 Lê Trọng Tấn, Tân Phú ... Elearning

+You Search Images Maps Play YouTube News Gmail Documents Calendar More

tvtho2000@gmail.com

Satellite Photos

Get directions My places

Car Pedestrian

A 140 Lê Trọng Tấn, Tân Phú, Ho Chi Minh City, Vietnam
B 120 Trường Chinh, Ho Chi Minh City, Vietnam

Did you mean a different: 120 Trường Chinh, Ho Chi Minh City, Vietnam

Add Destination - Show options

GET DIRECTIONS

Suggested routes

- Dương Đức Hiền and Ché Lan Viên 1.6 km, 6 mins
- Lê Trọng Tấn and Ché Lan Viên 1.8 km, 7 mins
- Lê Trọng Tấn and Trường Chinh 2.4 km, 7 mins
- Or Walk 18 mins

Driving directions to 120 Trường Chinh,

Map data ©2012 Google Edit in Google Map Maker

Một số bài toán

Trường Cao Đẳng Công Nghiệp Thực Phẩm Tp. Hcm to HC - Opera House - Google Maps - Mozilla Firefox

File Edit View History Bookmarks Yahoo! Tools Help

https://maps.google.com

Most Visited Getting Started Latest Headlines NEWS NEWS-EN IT HUTECH EMAIL TRAINING MUSIC VIDEO RELAX UPLOAD WEB DEV OTHER localhost

Share page: 0937330664

! (149 chưa đọc) - tvtho2000 - Yahoo! ... Trường Cao Đẳng Công Nghiệp Thực... Elearning

+You Search Images Maps Play YouTube News Gmail Documents Calendar More

tvtho2000@gmail.com

Google

Get directions My places

A hoc Công nghiệp Thực phẩm Tp. Hồ Chí Minh
B nhà hát thành phố HCM

Did you mean a different: nhà hát thành phố HCM

Add Destination - Show options

GET DIRECTIONS

Suggested routes

Cách Mạng Tháng 8 9.4 km, 17 mins
Nam Kỳ Khởi Nghĩa 9.1 km, 19 mins
Lê Văn Sỹ 9.3 km, 19 mins

Driving directions to HC - Opera House

A Trường Cao Đẳng Công Nghiệp Thực Phẩm Tp. Hcm

Map data ©2012 Google - Edit in Google Map Maker

Một số bài toán

❖ Ví dụ 5: Xây dựng hệ thống từ điển

Viết chương trình từ điển Anh – Việt, cho phép thực hiện các thao tác sau:

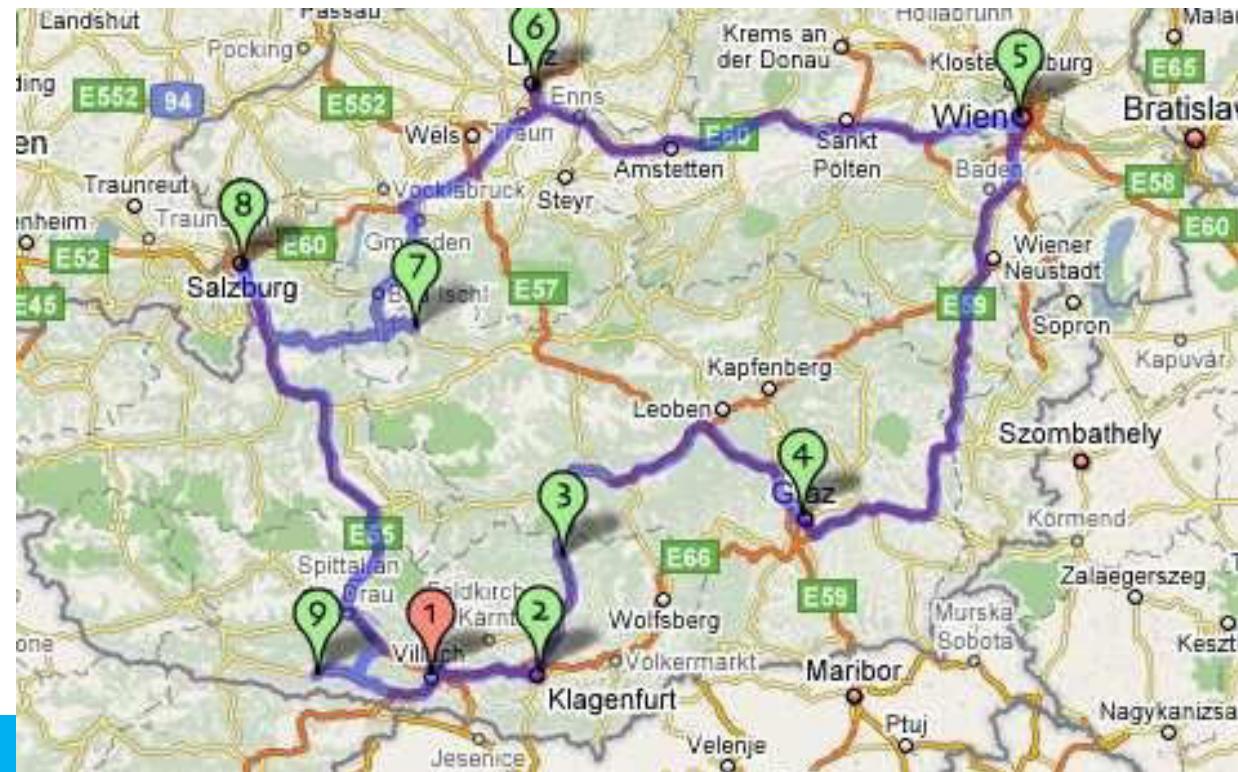
1. Tìm một từ.
2. Thêm một từ.
3. Xóa một từ.
4. Sửa một từ.
5. Tìm từ đồng nghĩa .
6. ...

Một số bài toán

❖ Ví dụ 6: Người bán hàng (traveling salesman problem - TSP)

Một người bán hàng cần đến thăm N khách hàng ở N địa điểm khác nhau. Tìm một hành trình cho người bán hàng trên sao cho:

1. Mỗi địa điểm thăm đúng 1 lần, sau đó quay về điểm xuất phát.
2. Tổng chi phí đi lại là ít nhất.



Một số bài toán

- ❖ **Thuật toán:** Thăm địa điểm gần nhất (nearest neighbor tour)
Từ điểm xuất phát, lần lượt đi thăm các điểm theo quy tắc: “đến thăm điểm chưa được thăm gần với điểm hiện tại nhất”.



Một số bài toán



Nearest neighbor tour: $1 \rightarrow 2 \rightarrow 3 \rightarrow X \rightarrow 7 \rightarrow 8 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 9 \rightarrow 1$

Đường đi tối ưu: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 7 \rightarrow X \rightarrow 9 \rightarrow 1$

Tâm quan trọng của CTDL & giải thuật

- ❖ Thực hiện một đề án tin học là chuyển bài toán thực tế thành bài toán có thể giải quyết trên máy tính.
- ❖ Một bài toán thực tế bất kỳ đều bao gồm **dữ liệu** và **các yêu cầu xử lý trên dữ liệu** đó để xây dựng một mô hình tin học phản ánh được bài toán thực tế cần chú trọng đến hai vấn đề:
 - Tổ chức biểu diễn các đối tượng thực tế.
 - Xây dựng các thao tác xử lý dữ liệu.

Tâm quan trọng của CTDL & giải thuật

- **Tổ chức biểu diễn các đối tượng thực tế:** Mô hình tin học của bài toán, cần phải tổ chức sao cho:
 - Vừa phản ánh chính xác dữ liệu thực tế.
 - Vừa dễ dàng dùng máy tính để xử lý.

→ *Xây dựng cấu trúc dữ liệu.*
- **Xây dựng các thao tác xử lý dữ liệu:** Từ những yêu cầu thực tế, cần tìm ra các giải thuật tương ứng để xác định trình tự các thao tác máy tính phải thi hành để cho ra kết quả mong muốn → Đây là bước *xây dựng giải thuật* cho bài toán.

Mối quan hệ giữa CTDL & giải thuật

- ❖ Mối quan hệ giữa cấu trúc dữ liệu và giải thuật

Cấu trúc dữ liệu + Giải thuật = Chương trình

Niklaus Wirth

- ❖ Khi có **cấu trúc dữ liệu tốt** và **giải thuật phù hợp** thì xây dựng **chương trình chỉ phụ thuộc thời gian**.
- ❖ Một chương trình máy tính chỉ hoàn thiện khi có đầy đủ **cấu trúc dữ liệu** và **giải thuật**.

❖ MỘT SỐ CẤU TRÚC DỮ LIỆU:

- Mảng (1 chiều, 2 chiều, ...)
- Danh sách liên kết.
- Ngăn xếp và hàng đợi.
- Cây.
- Bảng băm.
- Đồ thị.
- ...

❖ MỘT SỐ GIẢI THUẬT:

- Đệ quy và giải thuật đệ quy.
- Xử lý chuỗi, biểu thức (Ký pháp tiền tố, trung tố, hậu tố).
- Sắp xếp.
- Tìm kiếm.

Xây dựng giải thuật

❖ Các bước cơ bản khi tiến hành giải các bài toán tin học:

- Xác định bài toán.
- Tìm CTDL biểu diễn bài toán.
- Tìm thuật toán (giải thuật).
- Lập trình.
- Kiểm thử (kiểm chứng).
- Tối ưu chương trình.

Xây dựng giải thuật

❖ Phân tích thời gian thực hiện của giải thuật:

- Các ký pháp để đánh giá độ phức tạp tính toán.
- Xác định độ phức tạp tính toán của giải thuật.
- Độ phức tạp tính toán với tình trạng DL vào.
- Chi phí thực hiện thuật toán.

Các kiểu dữ liệu

- ❖ Máy tính chỉ có thể lưu trữ dữ liệu ở dạng nhị phân.
- ❖ Nếu muốn phản ánh được dữ liệu đa dạng, thì cần phải xây dựng những **phép ánh xạ**, những **qui tắc tổ chức phức tạp** che lên tầng dữ liệu nhị phân thô sơ.

Các kiểu dữ liệu

❖ Nhằm đưa ra những khái niệm logic về hình thức lưu trữ khác nhau được gọi là kiểu dữ liệu.

- Các kiểu dữ liệu cơ sở
- Các kiểu dữ liệu có cấu trúc
- Kiểu dữ liệu Union
- Kiểu dữ liệu con trỏ
- Kiểu dữ liệu tập tin

Các kiểu dữ liệu

Định nghĩa kiểu dữ liệu:

Kiểu dữ liệu **T** được xác định bởi một bộ **<V, O>**, với:

- ❖ **V**: Tập các giá trị hợp lệ mà một đối tượng kiểu **T** có thể lưu trữ.
- ❖ **O**: Tập các thao tác xử lý có thể thi hành trên đối tượng kiểu **T**.

Các kiểu dữ liệu

Định nghĩa kiểu dữ liệu:

Ví dụ: Giả sử có kiểu dữ liệu mẫu tự alpha = $\langle V_c, O_c \rangle$ với:

$$V_c = \{a-z, A-Z\}$$

$$O_c = \{\text{Lấy mã ASCII của ký tự, đổi ký tự thành ký tự hoa}\}$$

→ Dữ liệu lưu trữ chiếm số bytes trong bộ nhớ gọi là kích thước của kiểu dữ liệu.

Các kiểu dữ liệu

Các thuộc tính của một kiểu dữ liệu:

- ❖ Tên kiểu dữ liệu
- ❖ Miền giá trị của dữ liệu
- ❖ Kích thước của dữ liệu
- ❖ Tập các toán tử tác động lên kiểu dữ liệu

Các kiểu dữ liệu cơ sở

Các kiểu dữ liệu cơ sở trong C:

- ❖ Kiểu số nguyên
- ❖ Kiểu số thực
- ❖ Kiểu ký tự
- ❖ Kiểu luận lý

Tên kiểu	Kích thước	Miền giá trị
char	1 byte	-128 -> 127
unsigned char	1 byte	0 -> 255
int	2 byte	-32768 -> 32767
unsigned int	2 byte	0 - -> 65335
long	4 byte	-2 ³² -> 2 ³¹
unsigned long	4 byte	0 -> 2 ³²
float	4 byte	3.4E-38....3.4E38
double	8 byte	1.7E-308....1.7E308
long double	10 byte	3.4E-4932....1.1E4932

Các kiểu dữ liệu có cấu trúc

❖ **Kiểu chuỗi ký tự:** là kiểu dữ liệu có cấu trúc đơn giản nhất và thường các ngôn ngữ lập trình đều định nghĩa nó như một kiểu cơ bản.

- Trong C các hàm xử lý chuỗi được đặt trong thư viện ***string.lib.***
- VD: **char S[10] ;** //chuỗi S có chiều dài tối đa là 10 ký tự (kể cả ký tự kết thúc)

char S[] = “ABCDEF”;

char *S = “ABCDEF”;

Các kiểu dữ liệu có cấu trúc

❖ **Kiểu mảng:** là kiểu dữ liệu trong đó mỗi phần tử của nó là một tập hợp có thứ tự các giá trị có cùng cấu trúc được lưu trữ liên tiếp nhau trong bộ nhớ.

- Mảng một chiều:

<Kiểu dữ liệu> <Tên biến> [<Số phần tử>];

- Mảng nhiều chiều:

**<Kiểu dữ liệu> <Tên biến> [<Số phần tử 1>]
[<Số phần tử 2>]...;**

Các kiểu dữ liệu có cấu trúc

❖ **Kiểu mẫu tin:** Kiểu mẫu tin cũng tương tự như mảng nhưng mỗi phần tử của nó là tập hợp các giá trị có thể khác cấu trúc.

- Kiểu mẫu tin thường được dùng để mô tả những đối tượng có cấu trúc phức tạp.
- **Ví dụ:**

```
struct PERSON {  
    char Hoten[];  
    int NamSinh;  
    char NoiSinh[];  
    char GioiTinh; // 0:Nữ, 1: Nam  
    char DiaChi[];  
};
```

Các kiểu dữ liệu có cấu trúc

- ❖ **Kiểu dữ liệu con trỏ:** Cho trước kiểu $T = \langle V, O \rangle$. Kiểu con trỏ ký hiệu T_p chỉ đến các phần tử có kiểu T được định nghĩa như sau: $T_p = \langle V_p, O_p \rangle$, Trong đó:
 - $V_p = \{\{các\ địa\ chỉ\ có\ thể\ lưu\ trữ\ nhũng\ đối\ tượng\ kiểu\ T\}, NULL\}$
 - $O_p = \{các\ thao\ tác\ định\ địa\ chỉ\ của\ một\ đối\ tượng\ kiểu\ T\} k\hbox{i} biết\ con\ trỏ\ chỉ\ đ\hbox{e}n\ đối\ tượng\ đ\hbox{e}o\}$

Các kiểu dữ liệu có cấu trúc

❖ Kiểu dữ liệu con trỏ:

- Kiểu con trỏ là kiểu dữ liệu cơ sở dùng lưu địa chỉ của một đối tượng dữ liệu khác.
- Biến thuộc kiểu con trỏ T_p là biến mà giá trị của nó là địa chỉ của một vùng nhớ ứng với một biến kiểu T , hoặc là giá trị **NULL**

Các kiểu dữ liệu có cấu trúc

❖ Kiểu dữ liệu con trỏ:

- Kích thước biến con trỏ tùy thuộc vào quy ước số byte trong từng mô hình bộ nhớ và từng ngôn ngữ lập trình cụ thể.
- Biến con trỏ trong C++ có kích thước 2 hoặc 4 bytes tùy vào con trỏ NEAR hay FAR
- Cú pháp định nghĩa dữ liệu kiểu con trỏ.

```
typedef <kiểu cơ sở> *<kiểu con trỏ>;
```

Các kiểu dữ liệu có cấu trúc

❖ Kiểu dữ liệu con trỏ:

Các thao tác cơ bản trên kiểu con trỏ:

- Khi một biến con trỏ p lưu trữ địa chỉ của đối tượng x ta nói “p trỏ đến x”
- Gán địa chỉ của một vùng nhớ con trỏ p:

$p = <\text{địa chỉ}>;$

$p = <\text{địa chỉ}> + <\text{giá trị nguyên}>$

- Truy xuất (xem) nội dung của đối tượng p trỏ đến ($*p$)

Các kiểu dữ liệu có cấu trúc

❖ Kiểu dữ liệu tập tin:

- Tập tin là kiểu dữ liệu đặc biệt, kích thước tối đa của tập tin phụ thuộc không gian đĩa.
- Việc đọc, ghi dữ liệu trên tập tin là mất thời gian, không an toàn dữ liệu.
- Thông thường chuyển dữ liệu trong tập tin (một phần hay toàn bộ) vào bộ nhớ trong để xử lý.

Bài tập

- ❖ Xem lại việc sử dụng mảng một chiều
- ❖ Xem lại việc sử dụng con trỏ trong C++
- ❖ Xem lại các thao tác với tập tin
- ❖ Xem lại việc sử dụng kiểu dữ liệu có cấu trúc
- ❖ Bài tập trong giáo trình chương 1

NHỮNG DẠNG BÀI TOÁN LIÊN QUAN

❖ BÀI TOÁN LIỆT KÊ:

- Một số kiến thức đại số tổ hợp:

1. CHỈNH HỢP LẬP
2. CHỈNH HỢP KHÔNG LẬP.
3. HOÁN VỊ
4. TỒ HỢP

NHỮNG DẠNG BÀI TOÁN LIÊN QUAN

❖ BÀI TOÁN LIỆT KÊ:

■ Phương pháp sinh (Generation):

1. SINH CÁC DÃY NHỊ PHÂN ĐỘ DÀI N.
2. LIỆT KÊ CÁC TẬP CON K PHẦN TỬ ...
3. LIỆT KÊ CÁC HOÁN VỊ

NHỮNG DẠNG BÀI TOÁN LIÊN QUAN

❖ BÀI TOÁN LIỆT KÊ:

■ Thuật toán quay lui:

1. LIỆT KÊ CÁC DÃY NHỊ PHÂN ĐỘ DÀI N
2. LIỆT KÊ CÁC TẬP CON K PHẦN TỬ
3. LIỆT KÊ CÁC CHỈNH HỢP KHÔNG LẶP CHẬP K .
4. BÀI TOÁN PHÂN TÍCH SỐ
5. BÀI TOÁN XẾP HẬU

NHỮNG DẠNG BÀI TOÁN LIÊN QUAN

❖ BÀI TOÁN LIỆT KÊ:

■ Kỹ thuật nhánh cành:

1. BÀI TOÁN TỐI ƯU.....
2. SỰ BÙNG NỔ TỒ HỢP.....
3. MÔ HÌNH KỸ THUẬT NHÁNH CÂN.
4. BÀI TOÁN NGƯỜI DU LỊCH
5. DÃY ABC

Thank for you attention!



See you next week!

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT



GV : ThS. Trần Văn Thọ

E-mail : thotv@hufi.edu.vn

NỘI DUNG

- 1.1. Tìm kiếm (Linear Search, Binary Search).
- 1.2. Sắp xếp (Interchange Sort, Selection Sort, Quick Sort)

TÌM KIẾM (SEARCH)

❖ Giả sử mỗi phần tử gồm có 2 phần:

- Một thành phần *khóa* (**Key**) để nhận diện có kiểu dữ liệu *KeyType*.
- Các thành phần còn lại là *thông tin* (**Info**) có kiểu dữ liệu *InfoType*, như vậy mỗi phần tử có cấu trúc dữ liệu như sau:

```
struct ItemType
```

```
{
```

```
    KeyType Key; //Khóa tìm kiếm  
    InfoType Info; //Thông tin liên quan  
};
```

TÌM KIẾM (SEARCH)

Tuy nhiên, để đơn giản trong phần trình bày lý thuyết ta đồng nhất 2 phần **Key** và **Info** là một. Kiểu dữ liệu mô phỏng là số nguyên (*int*).

Ta có định nghĩa kiểu dữ liệu mới có tên **ItemType** như sau:

```
typedef int ItemType;
```

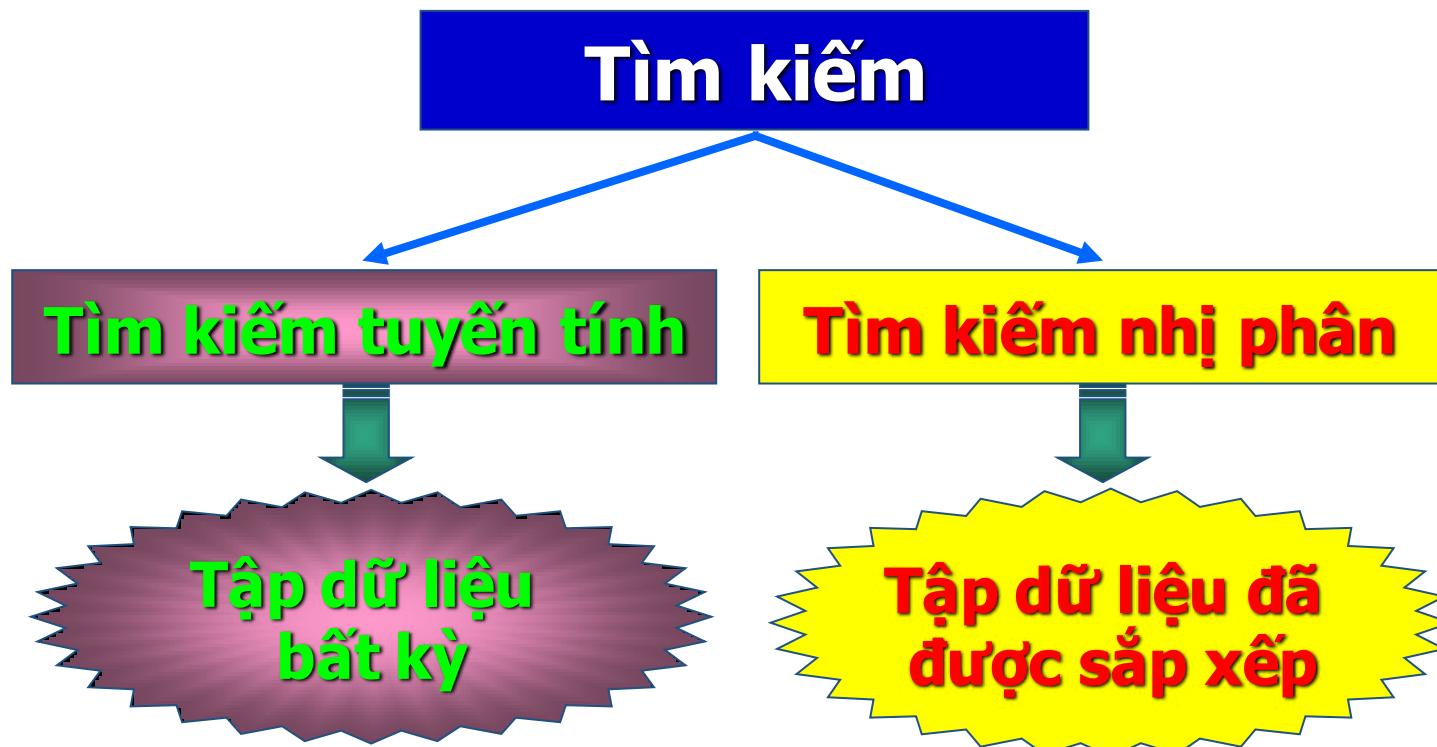
TÌM KIẾM (SEARCH)

- ❖ **Bài toán đặt ra:** Giả sử có một mảng A gồm n phần tử. Cần xác định xem một phần tử có giá trị là X có xuất hiện trong mảng A hay không? Nếu có thì phần tử có giá trị bằng phần tử X là phần tử tại vị trí thứ mấy trong mảng A?

TÌM KIẾM (SEARCH)

Các giải thuật tìm kiếm nội đưa ra 2 cách tìm kiếm:

- **Tìm kiếm tuyến tính** (Linear Search)
- **Tìm kiếm nhị phân** (Binary Search)



TÌM KIẾM TUYẾN TÍNH (LINEAR SEARCH)

❖ Ý tưởng:

So sánh lần lượt từng phần tử của mảng A với giá trị X cần tìm. Bắt đầu xuất phát từ phần tử đầu tiên cho đến khi tìm thấy **hoặc** tìm hết mảng mà vẫn không tìm thấy phần tử nào có giá trị bằng X.

TÌM KIẾM TUYẾN TÍNH (LINEAR SEARCH)

❖ Thuật toán: (mô phỏng theo C)

B1: Gán $i = 0$; // bắt đầu từ phần tử đầu tiên

B2: So sánh $A[i]$ với giá trị X, có 2 khả năng:

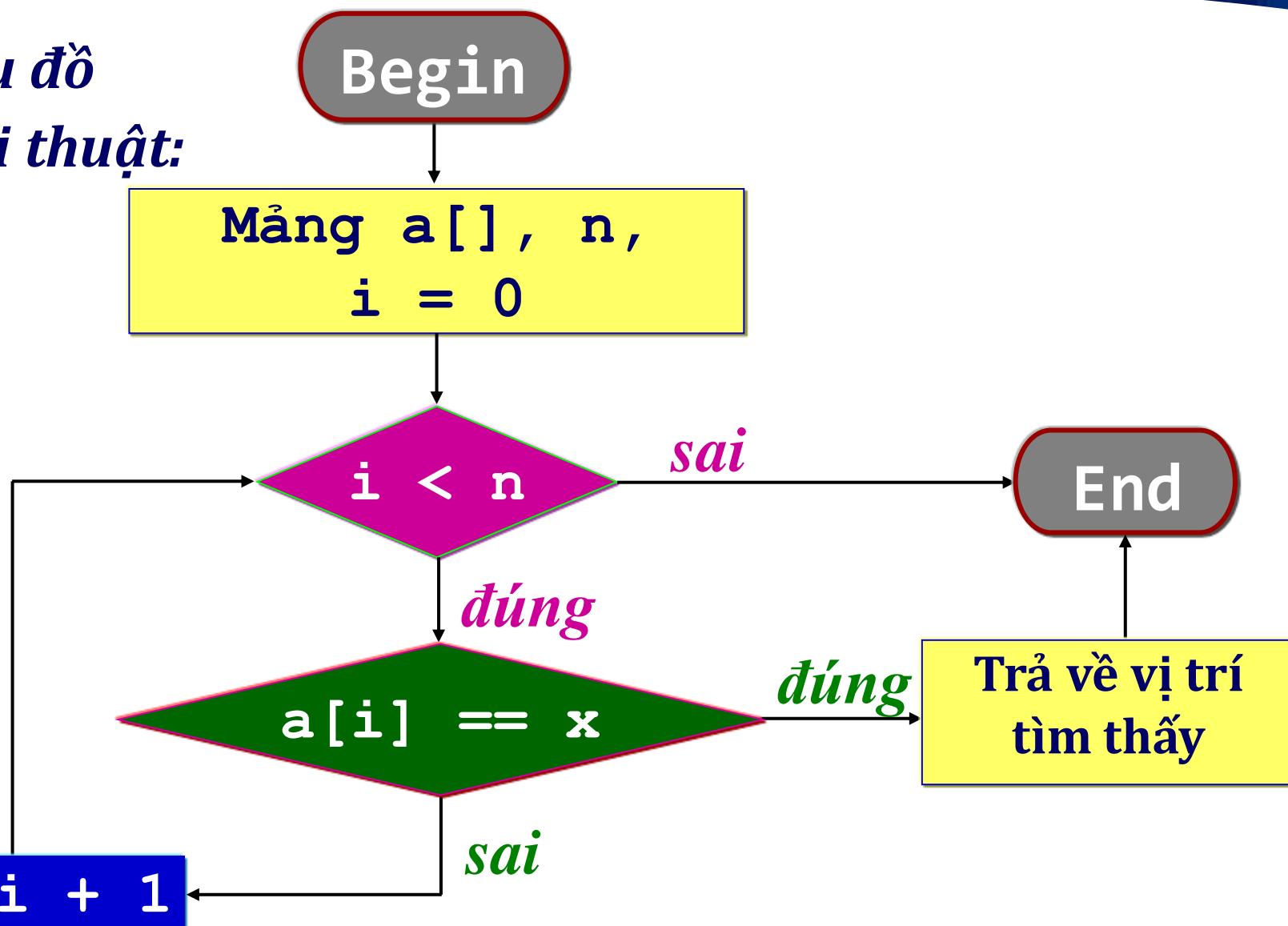
- $A[i] == X \rightarrow$ Tìm thấy. Dừng
- $A[i] != X \rightarrow$ Sang B3

B3: Tăng i thêm 1 //Xét phần tử tiếp theo trong mảng

- **Nếu $i == n$:** Hết mảng, không tìm thấy. **Dừng**
- **Ngược lại:** Quay trở lại B2.

TÌM KIẾM TUYẾN TÍNH (LINEAR SEARCH)

❖ **Lưu đồ
giải thuật:**



TÌM KIẾM TUYẾN TÍNH (LINEAR SEARCH)

Ví dụ

Cho dãy số a, giá trị tìm $X = 8$:

A

12	2	5	8	1	6	4
0	1	2	3	4	5	6

$X = 8$

Tìm được

Không

Không

Không bằng

1

6

4

Tìm thấy $A[3] = X \rightarrow$ Trả về 3. Dừng

TÌM KIẾM TUYẾN TÍNH (LINEAR SEARCH)

❖ Cài đặt:

```
int linearSearch(ItemType a[], int n, ItemType x)
```

```
{ //Trả về vị trí tìm thấy hoặc -1
```

```
    for(int i = 0; i < n; i++)
```

```
        if(a[i] == x)
```

```
            return i; //tìm thấy tại i
```

```
    return -1; //không tìm thấy
```

```
}
```

TÌM KIẾM TUYẾN TÍNH (LINEAR SEARCH)

- ❖ **Nhận xét:** Giải thuật tìm kiếm tuyến tính không phụ thuộc vào việc các phần tử trong mảng có được sắp xếp thứ tự (tăng/giảm) hay chưa?

TÌM KIẾM NHỊ PHÂN (BINARY SEARCH)

❖ Ý tưởng: Giả sử dãy đã sắp thứ tự tăng

Tại mỗi bước tiến hành **so sánh** x với phần tử nằm vị trí **giữa** của dãy tìm kiếm hiện hành, dựa vào kết quả so sánh này để quyết định giới hạn dãy cần tìm ở bước kế tiếp là **nữa bên trái** hay **nữa bên phải** của dãy tìm kiếm hiện hành.

TÌM KIẾM NHỊ PHÂN (BINARY SEARCH)

❖ Thuật toán: (mô phỏng theo C)

B1: Gán **Left** = 0; **Right** = $n - 1$;

B2: Tính **Mid** = (**Left** + **Right**) / 2;

B3: So sánh **A[Mid]** với khóa **X** có 3 khả năng xảy ra:

- Nếu **A[Mid]** = **X** thì **Tìm thấy. Dừng.**

- Nếu **A[Mid]** > **X** //Sẽ tìm trong dãy **A[Left]... A[Mid-1]**
thì gán lại **Right** = **Mid - 1**;

- Nếu **A[Mid]** < **X** //Sẽ tìm trong dãy **A[Mid+1]... A[Right]**
thì gán lại **Left** = **Mid + 1**;

B4: Nếu **Left** ≤ **Right** thì

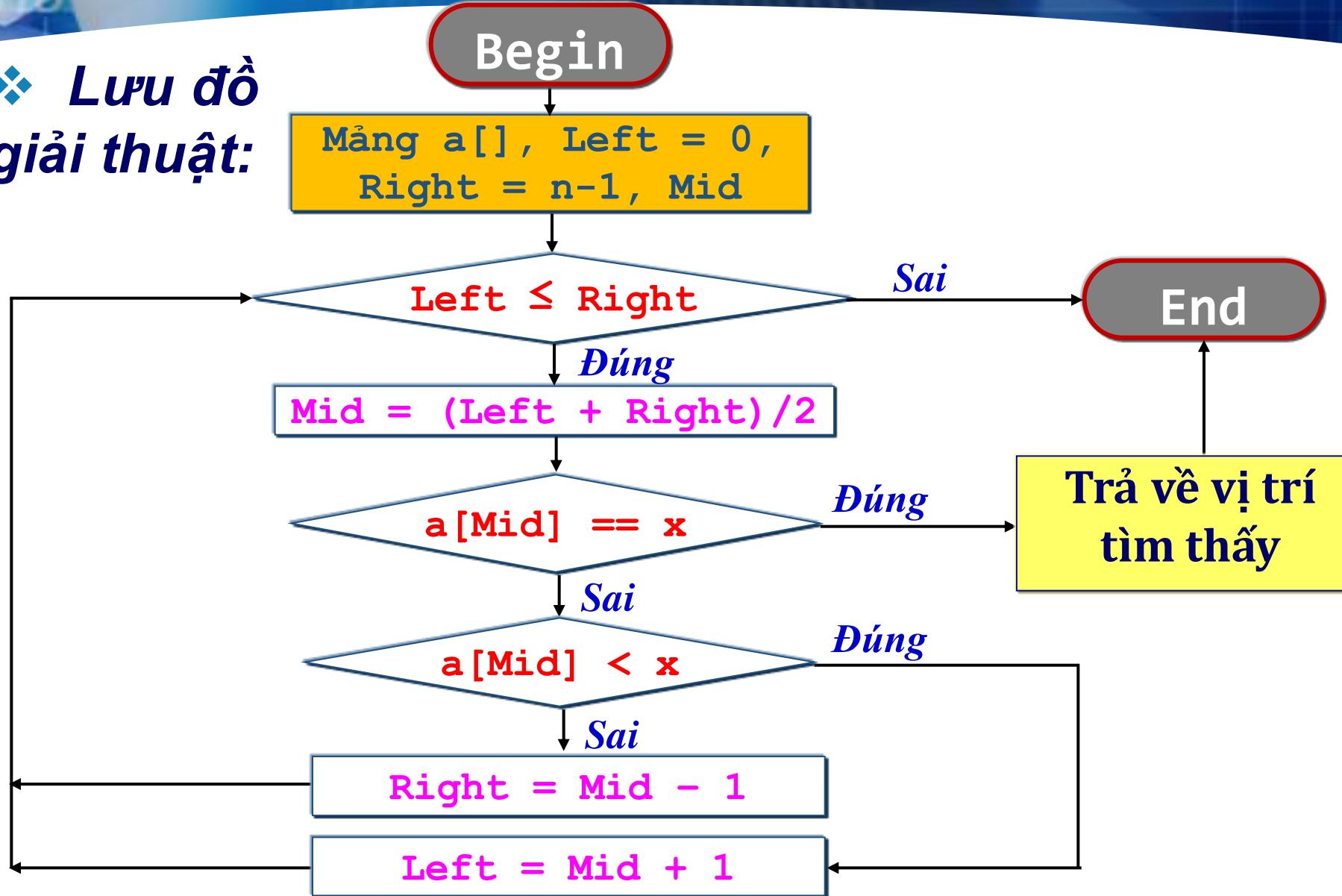
Quay trở lại B2

Ngược lại

Kết thúc

TÌM KIẾM NHỊ PHÂN (BINARY SEARCH)

❖ *Lưu đồ
giải thuật:*



TÌM KIẾM NHỊ PHÂN (BINARY SEARCH)

ví dụ

Cho dãy số gồm 8 phần tử bên dưới và $X = 8$:

1	2	4	5	6	8	12	15
---	---	---	---	---	---	----	----

$X = 8$

1	2	4	5	6	8	12	15
---	---	---	---	---	---	----	----

Left = 0

Mid = 3

Right = 7

Đoạn tìm kiếm

1	2	4	5	6	8	12	15
---	---	---	---	---	---	----	----

Left

Right = 7

Đoạn tìm kiếm

Tìm thấy

Tìm thấy $A[5] = X \rightarrow$ Trả về 5. Dừng

TÌM KIẾM NHỊ PHÂN (BINARY SEARCH)

❖ Cài đặt (*không dùng đệ qui*):

```
int binarySearch(ItemType a[], int n, ItemType x)
```

```
{ //Trả về vị trí tìm thấy: Mid, hoặc trả về: -1
```

```
    int Left = 0, Right = n - 1, Mid;
```

```
    while(Left <= Right) {
```

```
        Mid = (Left + Right) / 2;
```

```
        if(a[Mid] == x)          return Mid; //tìm thấy
```

```
        else if(a[Mid] > x)      Right = Mid - 1;
```

```
        else                      Left = Mid + 1;
```

```
}
```

```
return -1; //không tìm thấy
```

```
}
```

TÌM KIẾM NHỊ PHÂN (BINARY SEARCH)

❖ Cài đặt (dùng đệ quy):

```
int binarySearch_Recursive(ItemType a[], int Left, int Right,  
ItemType x)
```

```
{//Trả về vị trí tìm thấy: Mid, hoặc trả về: -1
```

```
    if (Left > Right) return -1; //không tìm thấy
```

```
    int Mid = (Left + Right)/2;
```

```
    if (a[Mid] == x)
```

```
        return Mid; //tìm thấy tại vị trí giữa
```

```
    else if (a[Mid] > x)
```

```
        return binarySearch_Recursive(a, Left, Mid-1, x);
```

```
    else
```

```
        return binarySearch_Recursive(a, Mid+1, Right, x);
```

```
}
```

TÌM KIẾM NHỊ PHÂN (BINARY SEARCH)

❖ *Nhận xét:*

- Giải thuật tìm kiếm nhị phân tiết kiệm thời gian hơn rất nhiều so với giải thuật tìm tuyến tính.
- Tuy nhiên khi muốn áp dụng giải thuật tìm kiếm nhị phân cần phải xét đến thời gian sắp xếp dãy dữ liệu để thỏa mãn điều kiện dãy phải có thứ tự. Thời gian này không nhỏ do đó ta cần phải cân nhắc kỹ nhu cầu thực tế để chọn 1 trong 2 giải thuật tìm kiếm trên sao cho có lợi nhất.

SẮP XẾP (SORT)

❖ Định nghĩa sắp xếp:

Sắp xếp là quá trình xử lý một danh sách các phần tử để đặt lại (*bố trí lại*) chúng theo một thứ tự nhất định nhằm thỏa mãn một điều kiện (*tiêu chuẩn*) nào đó dựa trên nội dung thông tin lưu trữ tại mỗi phần tử.

SẮP XẾP (SORT)

❖ Giả sử mỗi phần tử gồm có 2 phần là:

- Một thành phần *khóa* (**Key**) để nhận diện có kiểu dữ liệu *KeyType*.
- Các thành phần còn lại là *thông tin* (**Info**) có kiểu dữ liệu *InfoType*, như vậy mỗi phần tử có cấu trúc dữ liệu như sau:

struct ItemType

{

KeyType

Key; //Khóa sắp xếp

InfoType

Info; //Các Thông tin liên quan

};

ĐỊNH NGHĨA SẮP XẾP

Tuy nhiên, để đơn giản trong phần trình bày lý thuyết ta đồng nhất 2 phần **Key** và **Info** là một. Kiểu dữ liệu mô phỏng là số nguyên (int).

Ta có định nghĩa kiểu dữ liệu mới có tên **ItemType** như sau:

```
typedef int ItemType;
```

GIẢI THUẬT ĐỔI CHỖ TRỰC TIẾP

❖ Khái niệm nghịch thế:

Xét 1 mảng các số a_1, a_2, \dots, a_n .

Nếu có $i < j$ mà $a_i > a_j$ thì ta gọi đó là 1 nghịch thế.

Ví dụ: Cho dãy số a như sau: 14 5 7 8 3

Vậy dãy trên ta có các cặp nghịch thế sau: (14, 5); (5, 3); (7, 3); (8, 3); ...

GIẢI THUẬT ĐỔI CHỖ TRỰC TIẾP

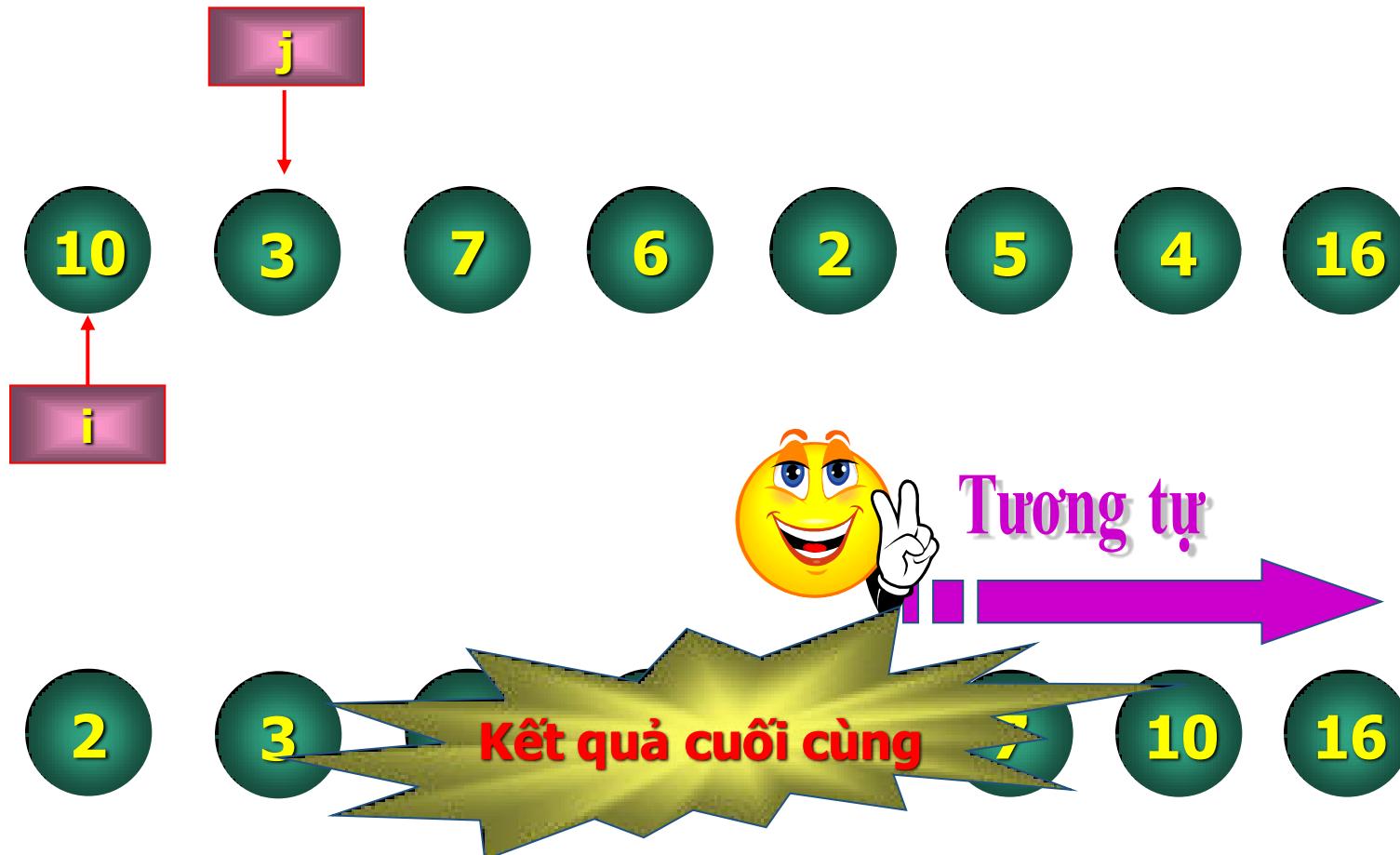
❖ Ý tưởng giải thuật:

Xuất phát từ đầu dãy, **tìm phần tử nào tạo nghịch thế** với phần tử đang xét thì **đổi chỗ 2 phần tử này** với nhau, lần lượt xét phần tử kế tiếp cho đến cuối dãy. Xem như phần tử đầu tiên được sắp xếp và không cần xét đến nó nữa. Tăng vị trí kiểm tra lên 1.

Lặp lại công việc như trên với các phần tử tiếp theo trong dãy. Cho đến khi dãy được sắp xếp hoàn toàn.

GIẢI THUẬT ĐỔI CHỖ TRỰC TIẾP

❖ Ví dụ: Cho dãy số a như sau:



GIẢI THUẬT ĐỔI CHỖ TRỰC TIẾP

❖ Cài đặt:

```
void interchangeSort_Asc(ItemType a[ ], int n)
```

```
{
```

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

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

```
            if(a[i] > a[j])
```

```
                swap(a[i], a[j]);
```

```
}
```

HOÁN VỊ HAI PHẦN TỬ

Trong đó, hàm Swap là hàm hoán vị hai phần tử (có kiểu dữ liệu là *ItemType*) cho nhau, như sau:

```
void swap(ItemType &x, ItemType &y)
```

```
{
```

```
    ItemType tam;
```

```
    tam = x;
```

```
    x = y;
```

```
    y = tam;
```

```
}
```

GIẢI THUẬT ĐỔI CHỖ TRỰC TIẾP

❖ *Đánh giá giải thuật:*

Ta có bảng đánh giá thuật toán sau:

Trường hợp	Số lần so sánh	Số lần gán
Tốt nhất	$\sum_{i=0}^{n-2} ((n-1) - (i+1) + 1) = \frac{n(n-1)}{2}$	0
Xấu nhất	$\sum_{i=0}^{n-2} ((n-1) - (i+1) + 1) = \frac{n(n-1)}{2}$	$3 \sum_{i=0}^{n-2} ((n-1) - (i+1) + 1) = 3 \frac{n(n-1)}{2}$

Vậy độ phức tạp của thuật toán là **O(n²)**.

GIẢI THUẬT CHỌN TRỰC TIẾP

❖ Ý tưởng giải thuật:

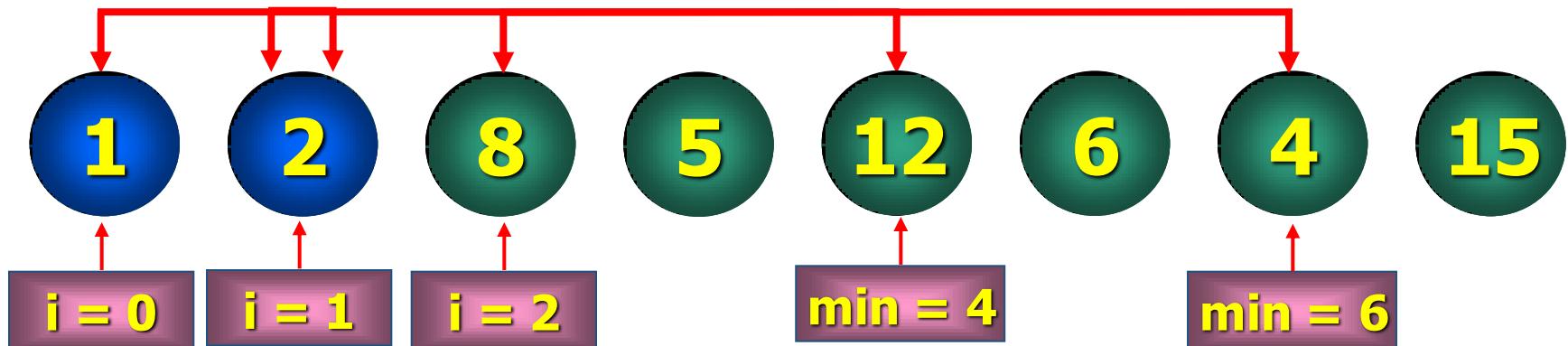
Chọn phần tử nhỏ nhất trong **n** phần tử ban đầu, **đổi chỗ với phần tử vị trí thứ 1** của dãy hiện hành; sau đó *không quan tâm đến nó nữa*.

Bắt đầu từ vị trí thứ 2; **Chọn phần tử nhỏ nhất** trong **n-1** phần tử còn lại, **đổi chỗ với phần tử vị trí thứ 2** của dãy hiện hành;

Lặp lại quá trình đó cho đến khi dãy hiện hành chỉ còn **1** phần tử thì dừng.

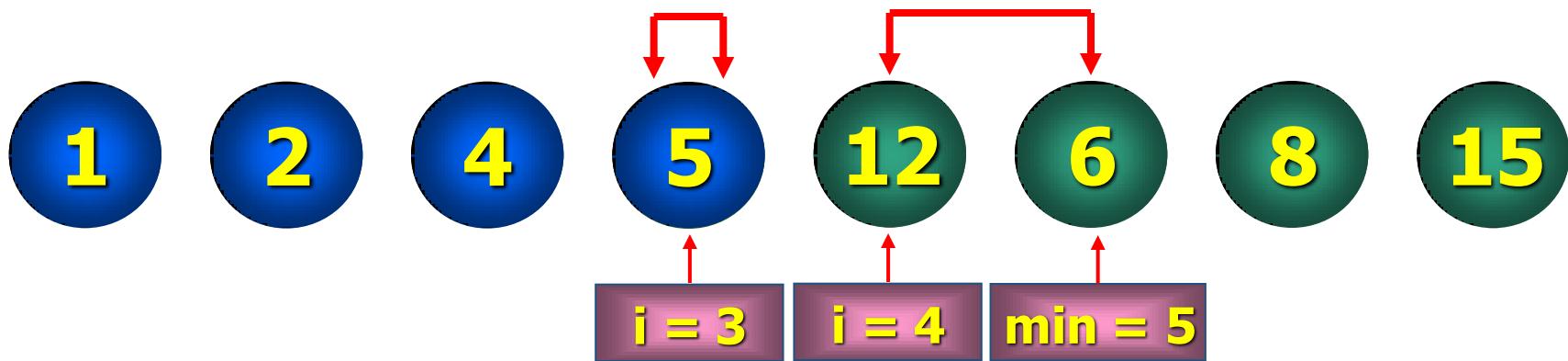
GIẢI THUẬT CHỌN TRỰC TIẾP

Ví dụ:



GIẢI THUẬT CHỌN TRỰC TIẾP

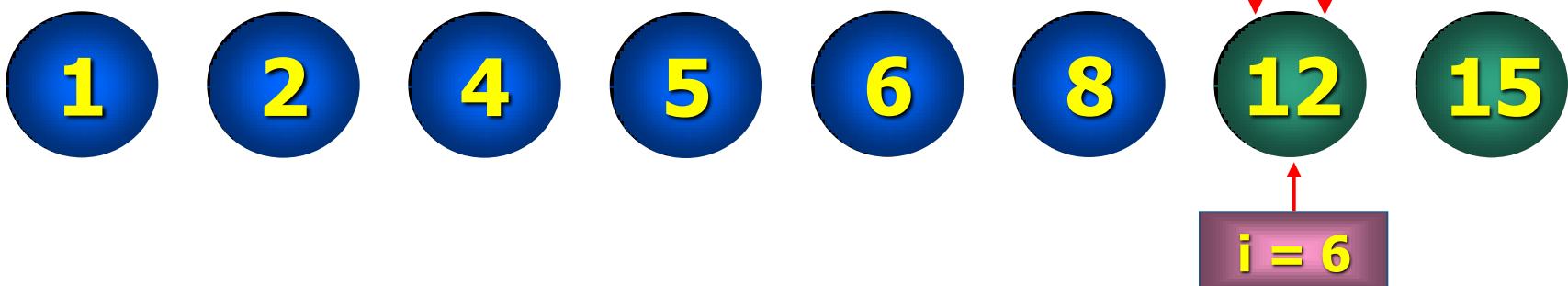
Ví dụ:



GIẢI THUẬT CHỌN TRỰC TIẾP

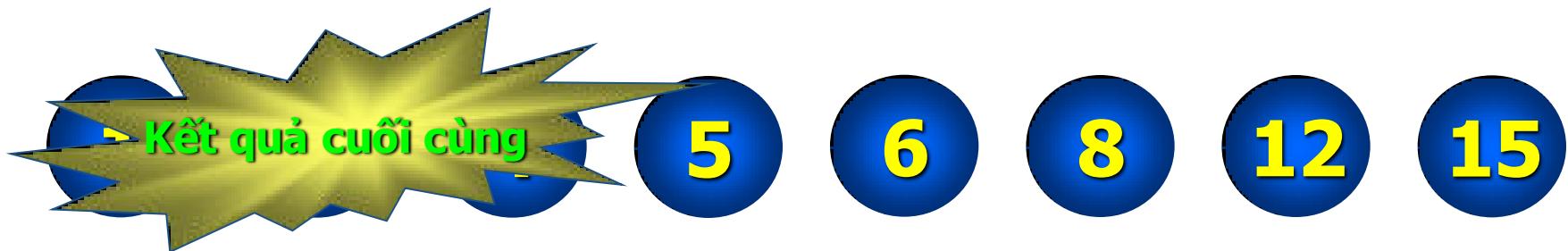
Ví dụ:

Tương tự



GIẢI THUẬT CHỌN TRỰC TIẾP

Ví dụ:



GIẢI THUẬT CHỌN TRỰC TIẾP

❖ Cài đặt:

```
void selectionSort_Asc(ItemType a[ ], int n) {  
    for(int i = 0; i < n - 1; i++) {  
        int min = i;  
        for(int j = i + 1; j < n; j++)  
            if(a[min] > a[j])  
                min = j;  
        swap(a[i], a[min]);  
    }  
}
```

GIẢI THUẬT CHỌN TRỰC TIẾP

❖ Cài đặt (cải tiến):

```
void selectionSort_Asc_Improve(ItemType a[ ], int n) {  
    for(int i = 0; i < n - 1; i++) {  
        int min = i;  
        for(int j = i + 1; j < n; j++)  
            if(a[min] > a[j])  
                min = j;  
        if(min != i) //nếu có 1 phần tử khác  
            swap(a[i], a[min]);  
    }  
}
```

GIẢI THUẬT CHỌN TRỰC TIẾP

❖ **Đánh giá giải thuật:**

Ta có bảng đánh giá thuật toán sau:

Trường hợp	Số lần so sánh	Số lần gán
Tốt nhất	$\sum_{i=0}^{n-2} ((n-1) - (i+1) + 1) = \frac{n(n-1)}{2}$	$\sum_{i=0}^{n-2} (4) = 4(n-1)$
Xấu nhất	$\sum_{i=0}^{n-2} ((n-1) - (i+1) + 1) = \frac{n(n-1)}{2}$	$\sum_{i=0}^{n-2} (4 + n - i - 1) = \frac{(n-1)(n+8)}{2}$

Vậy độ phức tạp của thuật toán là **O(n²)**.

GIẢI THUẬT QUICKSORT

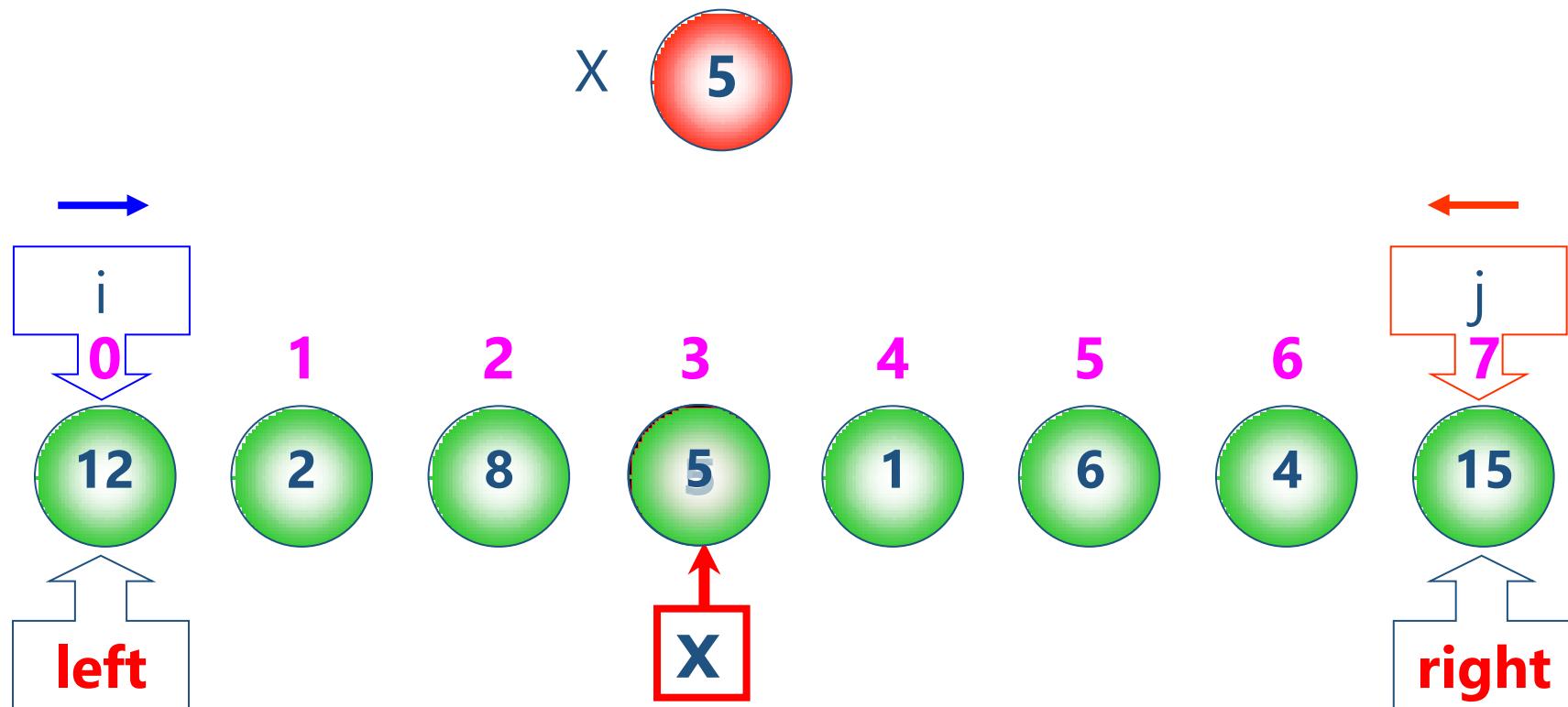
❖ Ý tưởng giải thuật:

Giải thuật QuickSort dựa trên việc phân hoạch dãy ban đầu thành 3 phần:

- **Dãy con 1:** gồm các phần tử $a_0 \dots a_{i-1}$ có giá trị không lớn hơn x.
- **Dãy con 2:** gồm các phần tử $a_i \dots a_j$ có giá trị bằng x.
- **Dãy con 3:** gồm các phần tử $a_{j+1} \dots a_{n-1}$ giá trị không nhỏ hơn x.
- Lặp lại quá trình trên với các dãy con.

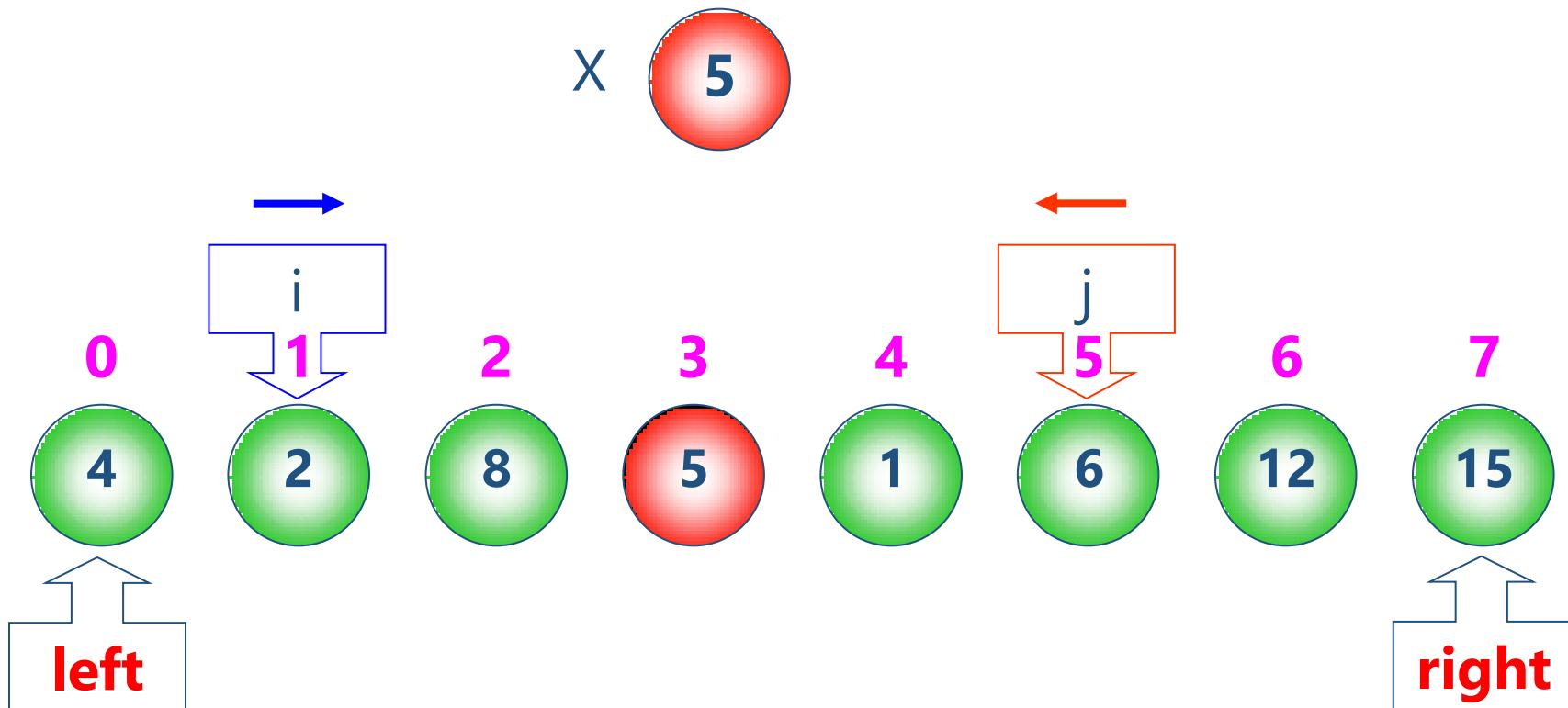
GIẢI THUẬT QUICKSORT – VÍ DỤ

➤ Phân hoạch đoạn [0,7]



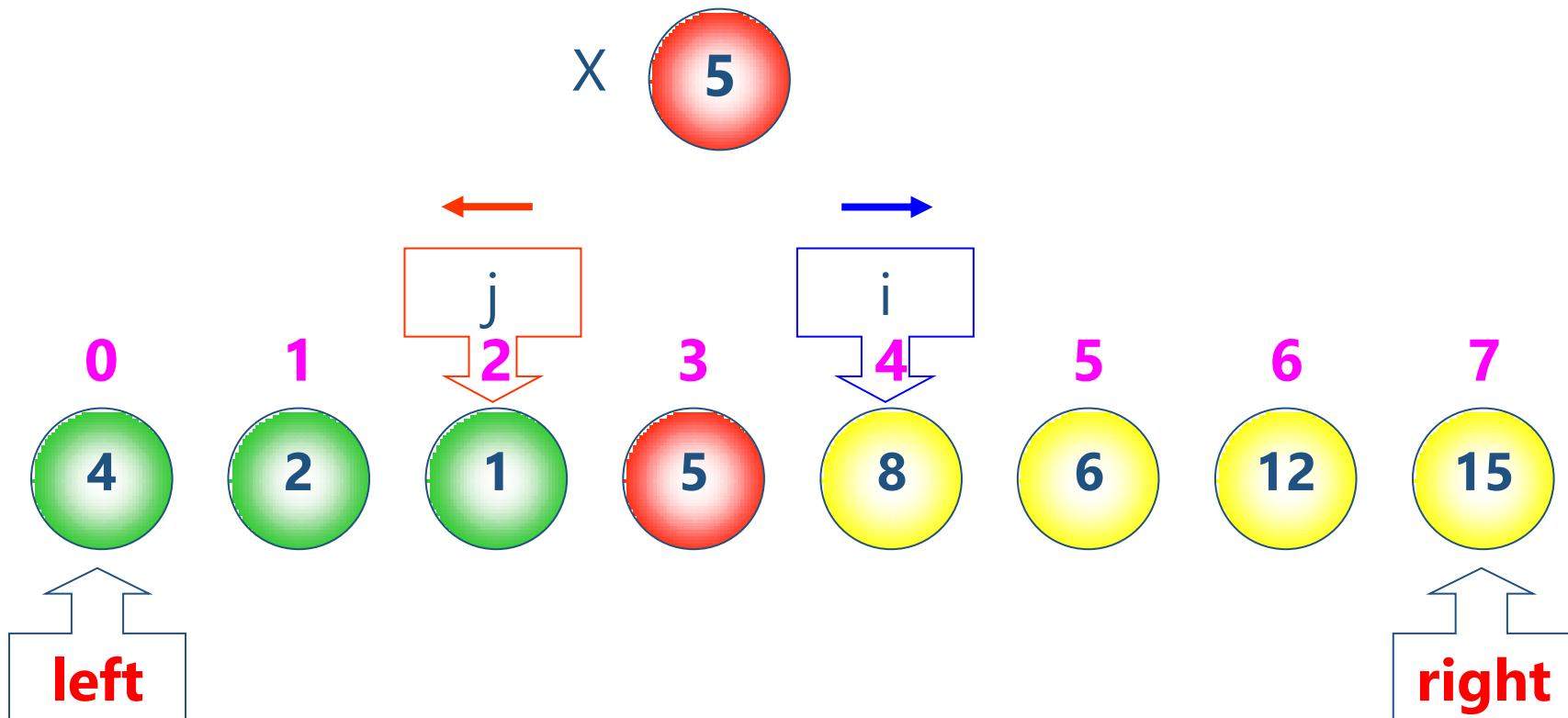
GIẢI THUẬT QUICKSORT – VÍ DỤ

➤ Phân hoạch đoạn [0,7]



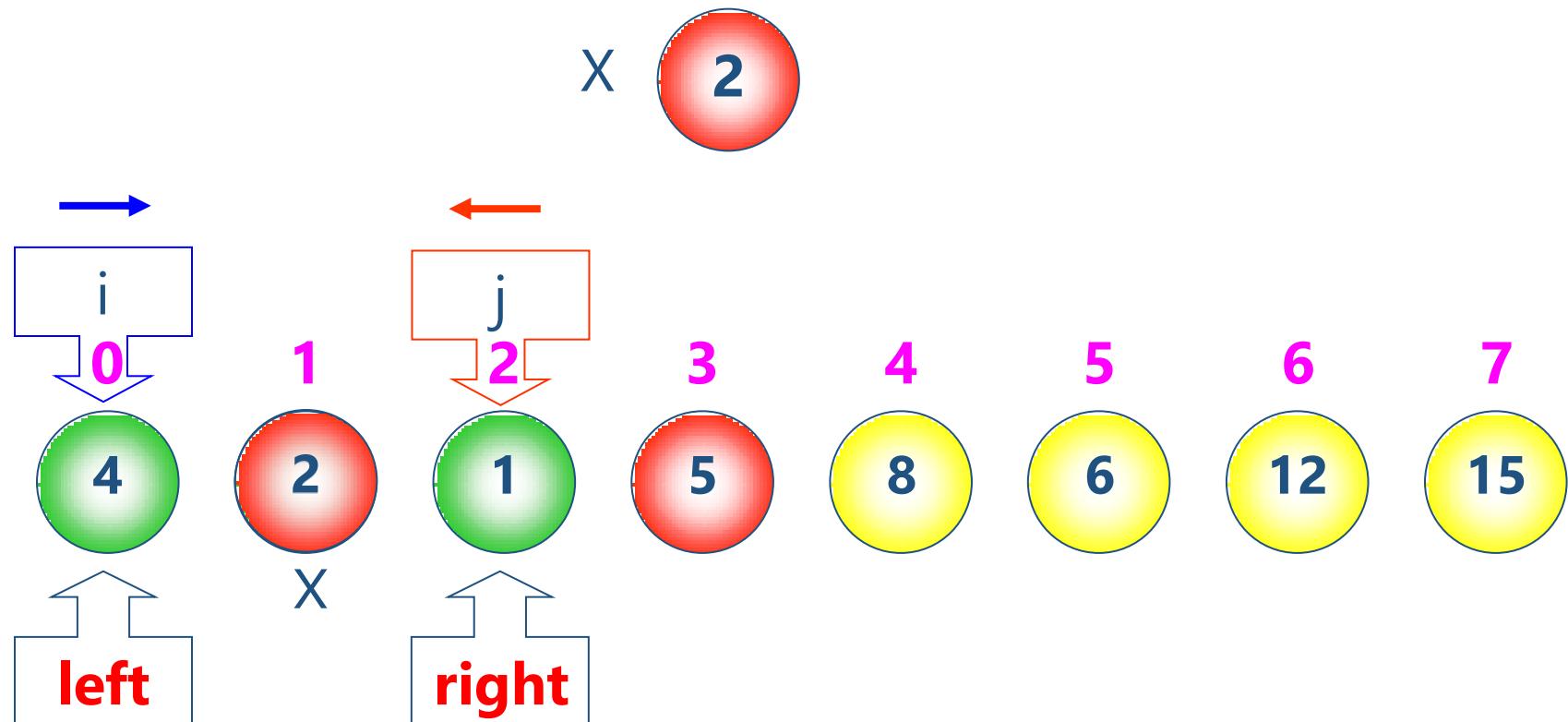
GIẢI THUẬT QUICKSORT – VÍ DỤ

➤ Phân hoạch đoạn $[0,7]$



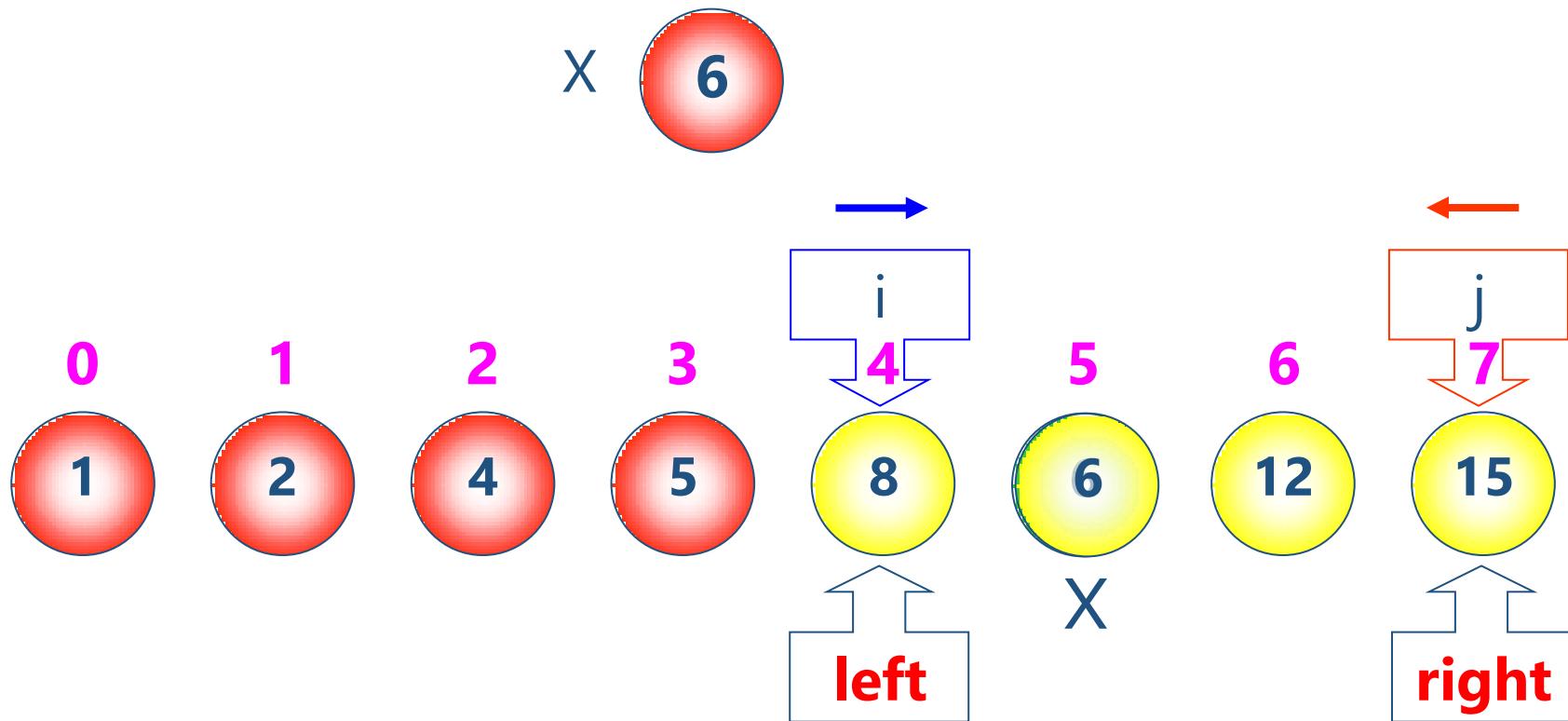
GIẢI THUẬT QUICKSORT – VÍ DỤ

➤ Phân hoạch đoạn [0,2]



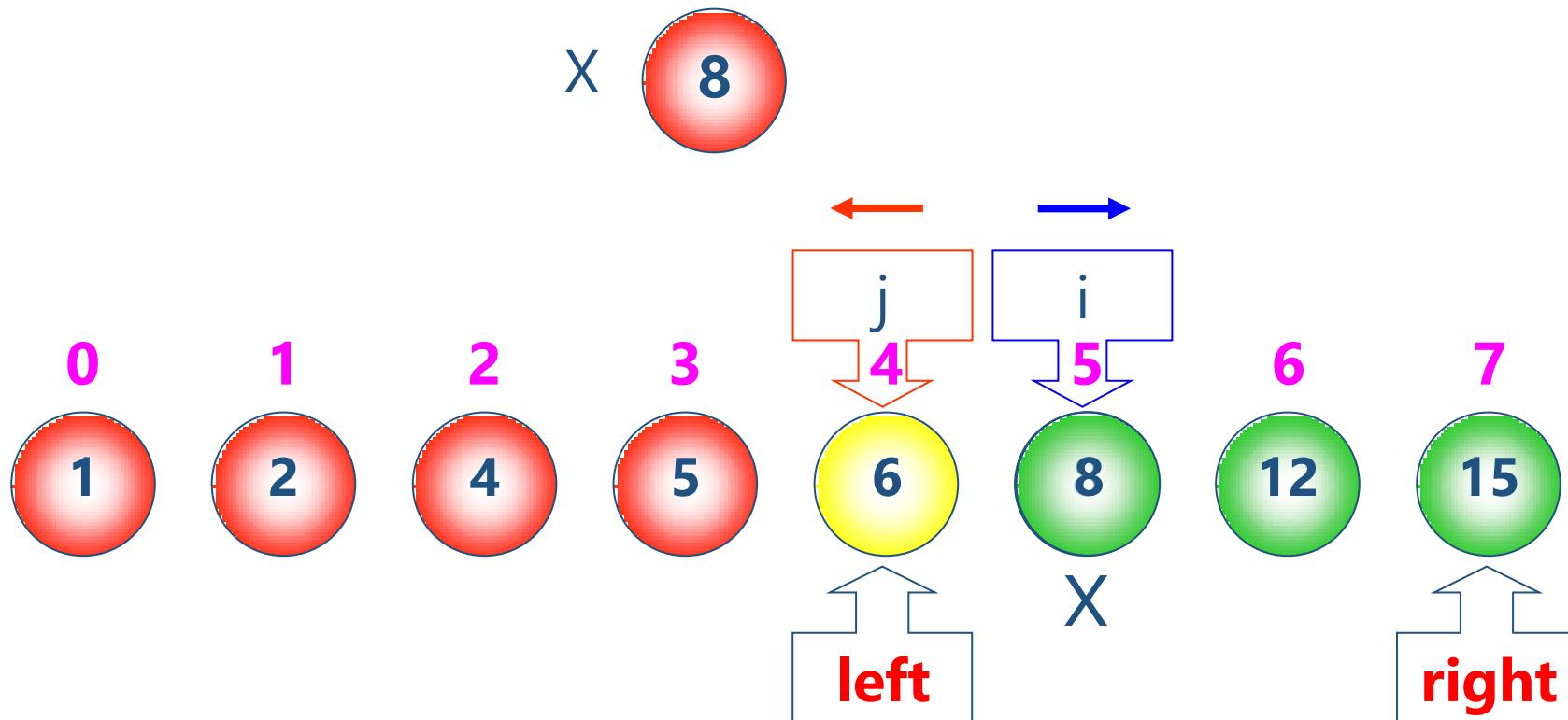
GIẢI THUẬT QUICKSORT – VÍ DỤ

➤ Phân hoạch đoạn [4,7]



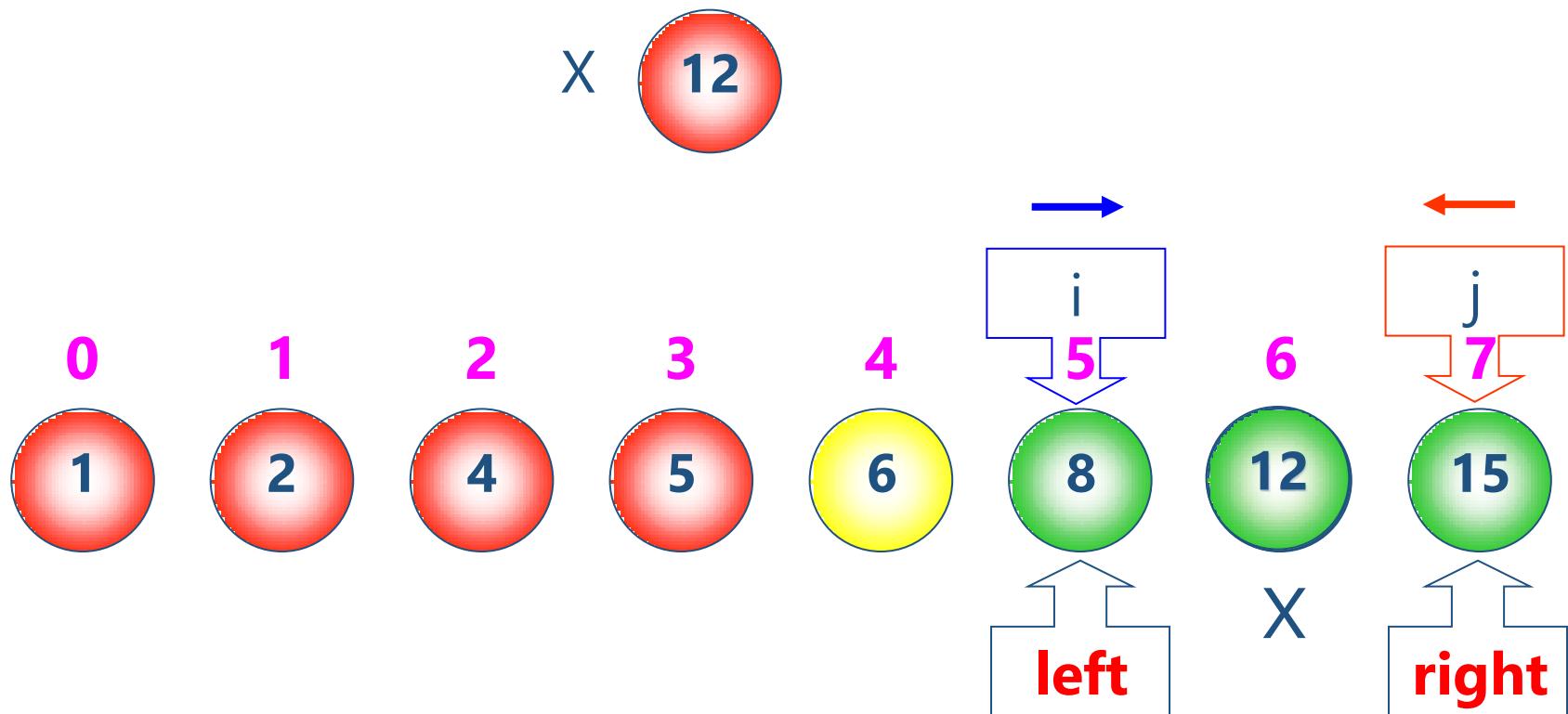
GIẢI THUẬT QUICKSORT – VÍ DỤ

➤ Phân hoạch đoạn [4,7]

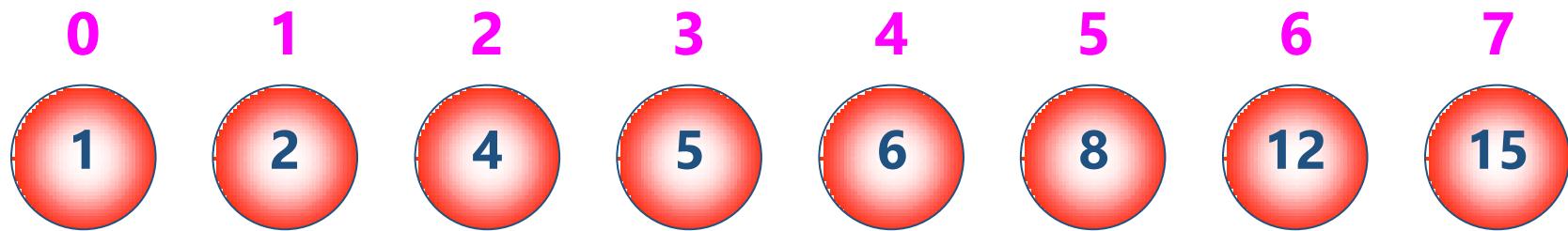


GIẢI THUẬT QUICKSORT – VÍ DỤ

➤ Phân hoạch đoạn [5,7]



GIẢI THUẬT QUICKSORT – VÍ DỤ



Kết quả cuối cùng

GIẢI THUẬT QUICKSORT

❖ Cài đặt:

void quickSort_Asc(ItemType a[], int Left, int Right) {

int i = Left, j = Right, Mid = (Left+Right)/2;

ItemType x = a[Mid];

do {

 while(a[i] < x) i++; // lặp cho đến khi a[i] >= x

 while(a[j] > x) j--; // lặp cho đến khi a[j] <= x

 if(i <= j) {

swap(a[i], a[j]);

 i++; j--;

 }

} while(i < j);

if (Left < j)

if (Right > i)

Phân hoạch bên trái

Phân hoạch bên phải

quickSort_Asc(a, Left, j);

quickSort_Asc(a, i, Right);

}

GIẢI THUẬT QUICKSORT

❖ Đánh giá giải thuật:

Ta có bảng đánh giá thuật toán sau:

Trường hợp	Độ phức tạp
Tốt nhất	$O(n * \log_2(n))$
Trung bình	$O(n * \log_2(n))$
Xấu nhất	$O(n^2)$

GIẢI THUẬT QUICKSORT

❖ Nhận xét:

- Giá trị mốc x được chọn sẽ có tác động đến hiệu quả thực hiện thuật toán vì nó quyết định đến số lần phân hoạch.
- Số lần phân hoạch sẽ ít nhất nếu ta chọn được x là phần tử trung bình của dãy.
- Tuy nhiên do tìm phần tử trung bình có chi phí quá cao nên *trong thực tế người ta thường hay chọn phần tử giữa dãy làm mốc.*

Thank for you attention!



See you next week!

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT



GV : ThS. Trần Văn Thọ

E-mail : thotv@hufi.edu.vn

Môn: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT



CHƯƠNG III: DANH SÁCH LIÊN KẾT

MỤC TIÊU

- ❖ Tìm hiểu các loại CTDL danh sách liên kết.
- ❖ Tìm hiểu những thao tác trên danh sách liên kết đơn.

Từ đó ứng dụng vào một số bài toán cụ thể.

Nội dung

❖ Danh sách liên kết đơn (**Single Linked List**):

- Giới thiệu
- Minh họa
- Cài đặt các thao tác
- Ứng dụng.

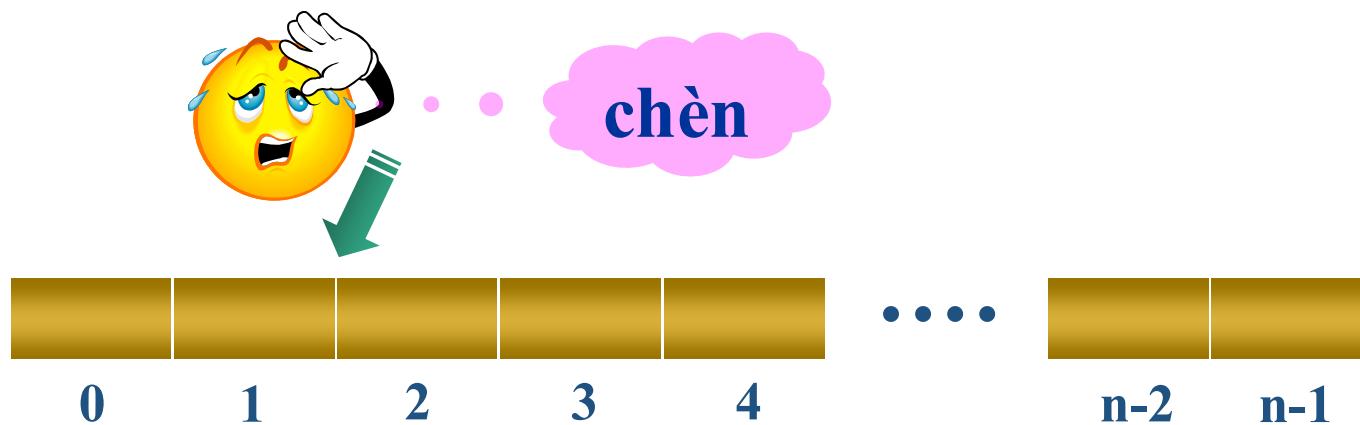
Single Linked List

(Danh sách liên kết đơn)

Giới thiệu

❖ Mảng 1 chiều

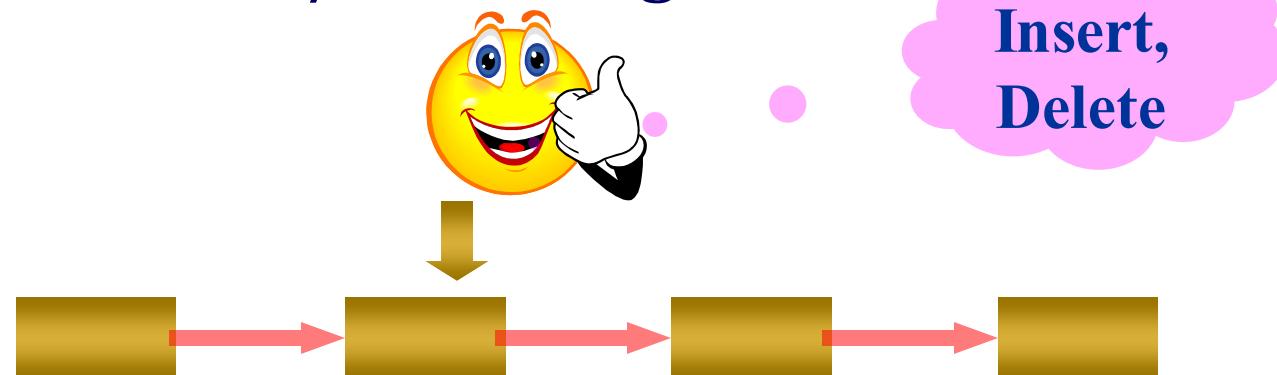
- Có kích thước cố định (*cấp phát tĩnh*)
- Thêm/ xóa có độ phức tạp cao
- Các phần tử tuần tự theo chỉ số $0 \Rightarrow n-1$
- Truy cập ngẫu nhiên



Giới thiệu

❖ Danh sách liên kết

- Cấp phát động lúc chạy chương trình
- Các phần tử nằm rải rác ở nhiều nơi trong bộ nhớ
- Kích thước danh sách chỉ bị giới hạn do RAM
- Thao tác thêm/xóa đơn giản



Con trỏ: Pointer

int i, *pi;

Địa chỉ

1FF0

i

1FE4

pi

pi = &i;

1FF0

i
*pi

1FE4

pi

i = 10
*pi = 10

1FF0

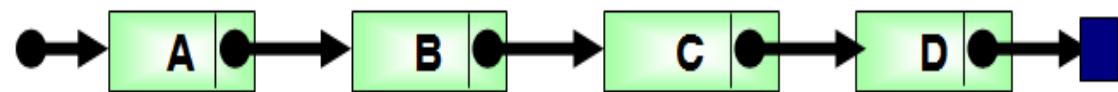
i
*pi

1FE4

pi

Định nghĩa

❖ **Danh sách liên kết đơn** là chuỗi các phần tử (**Node**), được tổ chức theo thứ tự tuyến tính. Mỗi phần tử liên kết với 1 phần tử đứng liền kề sau trong danh sách.

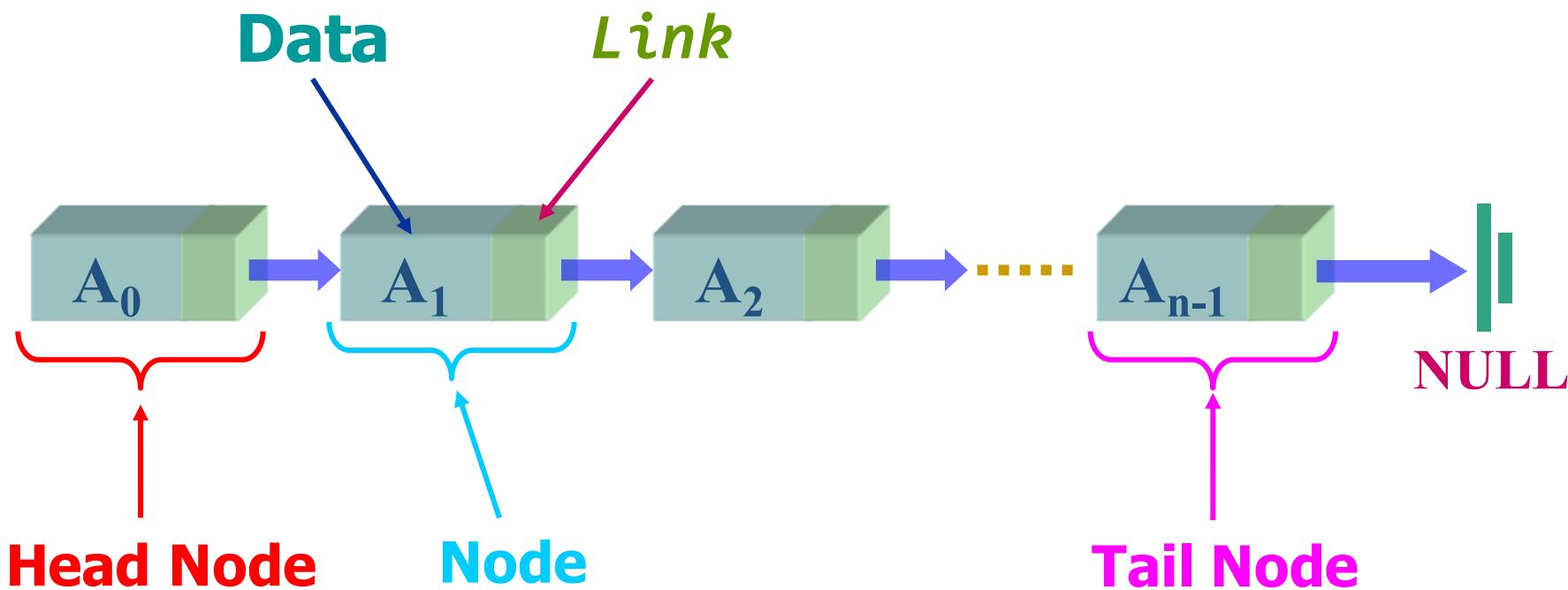


- ❖ **Mỗi phần tử (Node) gồm 2 thành phần:**
- **Thành phần dữ liệu (Data):** Lưu trữ thông tin về bản thân phần tử (**Information**).

A diagram showing a node structure with two adjacent boxes. The left box is labeled "Data" in red text. The right box is labeled "link" in pink text.
 - **Thành phần liên kết (Link):** Lưu địa chỉ phần tử đứng kế sau (**Next**) trong danh sách, hoặc lưu trữ giá trị **NONE** nếu là phần tử cuối danh sách.

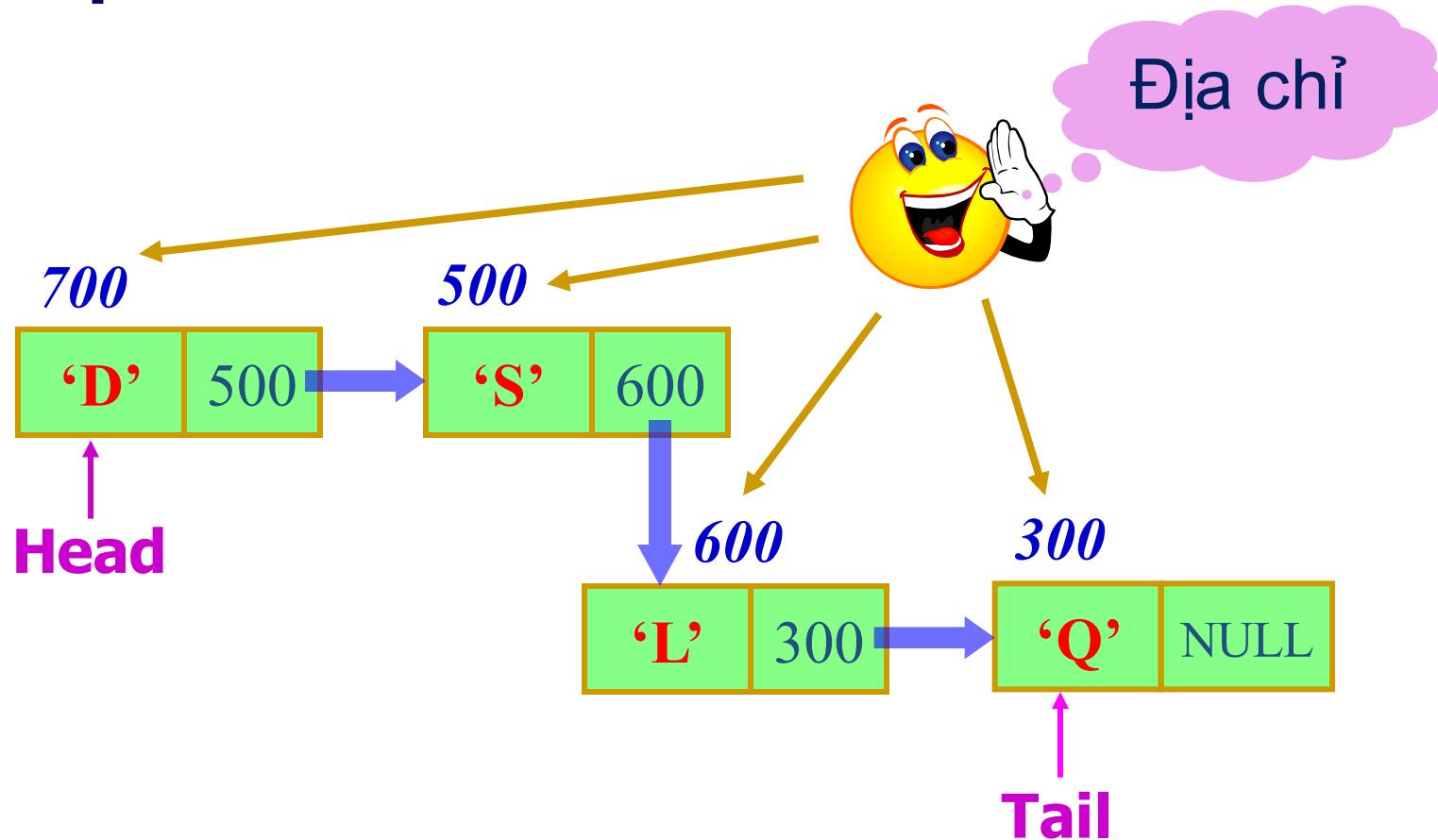
Minh Họa

❖ Mô tả danh sách



Minh Họa

Ví dụ:



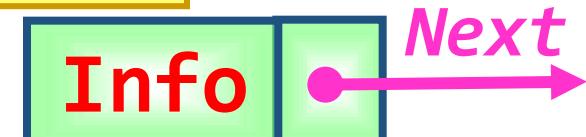
Khai báo phần tử của danh sách

❖ Khai báo cấu trúc một nút (Node):

```
struct SNode  
{  
    ItemType Info;  
    SNode* Next;  
};
```

❖ Trong đó ItemType:

- Là kiểu dữ liệu được định nghĩa trước.
- Chứa thông tin (hay dữ liệu) của từng Node.



ItemType có thể là: int, long, float, SV, NV,....

Khai báo phần tử của danh sách

```
struct SNode  
{  
    ItemType Info;  
    SNode* Next;  
};
```

Cấu trúc
SNode



```
struct SNode  
{  
    int Info;  
    SNode* Next;  
};
```

```
struct SNode  
{  
    SinhVien Info;  
    SNode* Next;  
};
```

Khai báo phần tử của danh sách

```
typedef int ItemType;  
struct SNode  
{  
    ItemType Info;  
    SNode* Next;  
};
```

Kiểu Dữ liệu Node
Dữ liệu của Node
là một số nguyên
trỏ đến nút kế sau

Khai báo kiểu con trỏ SNode

```
typedef SNode* SNodePtr;
```

Khai báo cấu trúc dữ liệu DSLK đơn

```
struct SList
```

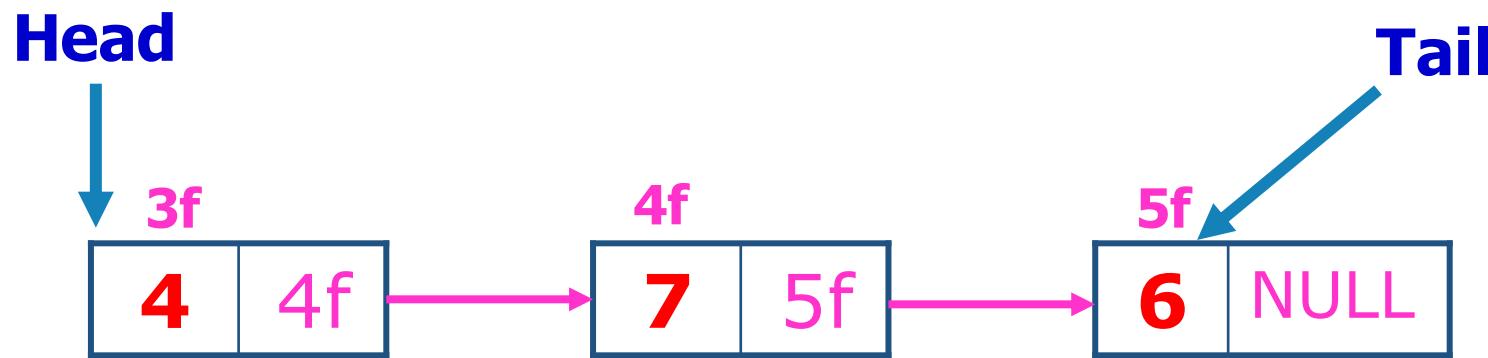
```
{// Tạo mới kiểu danh sách liên kết đơn tên SList
```

```
    SNode* Head; //Lưu địa chỉ SNode đầu tiên trong List
```

```
    SNode* Tail; //Lưu địa chỉ của SNode cuối cùng trong List
```

```
};
```

Ví dụ tổ chức DSLK đơn trong bộ nhớ



Trong ví dụ trên thành phần dữ liệu là 1 số nguyên

Thao tác cơ bản trên DSLK đơn

- ❖ Khởi tạo một danh sách rỗng.
- ❖ Kiểm tra danh sách rỗng.
- ❖ Tạo một nút mới chứa khóa x.
- ❖ Thêm phần tử có khóa x vào danh sách.
- ❖ Xóa phần tử có khóa x khỏi danh sách.
- ❖ Tìm kiếm phần tử có khóa bằng với x trong danh sách.
- ❖ Sắp xếp danh sách.
- ❖ Duyệt danh sách.

Phản minh họa
sẽ dùng
ItemType là int

Khởi tạo danh sách rỗng

- ❖ Gán địa chỉ của 2 con trỏ Head và Tail đồng thời bằng **NULL** (*trỏ NULL*).

```
void initSList(SList &sl)
{
    sl.Head = NULL;
    sl.Tail = NULL;
}
```

Kiểm tra danh sách rỗng

- ❖ Danh sách rỗng nếu con trỏ Head là **NULL**.

```
int isEmpty(SList s1)
{
    if(s1.Head == NULL)
        return 1;
    else
        return 0;
}
```

Tạo nút mới chứa giá trị x bất kỳ

❖ Hàm trả về địa chỉ phần tử mới tạo:
SNode* createSNode(ItemType x)

{

SNode* **p** = new SNode;

if(**p** == **NULL**) {

 printf("Không đủ bộ nhớ!");

 getch();

 return **NULL**;

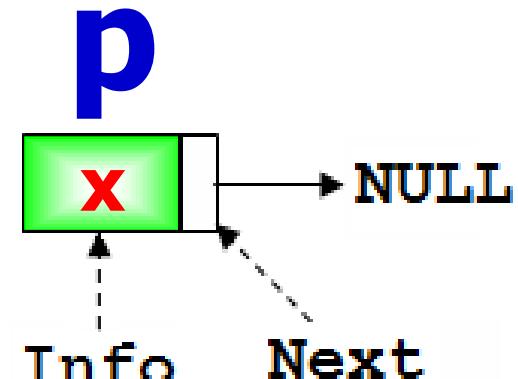
}

p→Info = x;

p→Next = **NULL**;

 return **p**;

}



Thao tác thêm phần tử vào danh sách

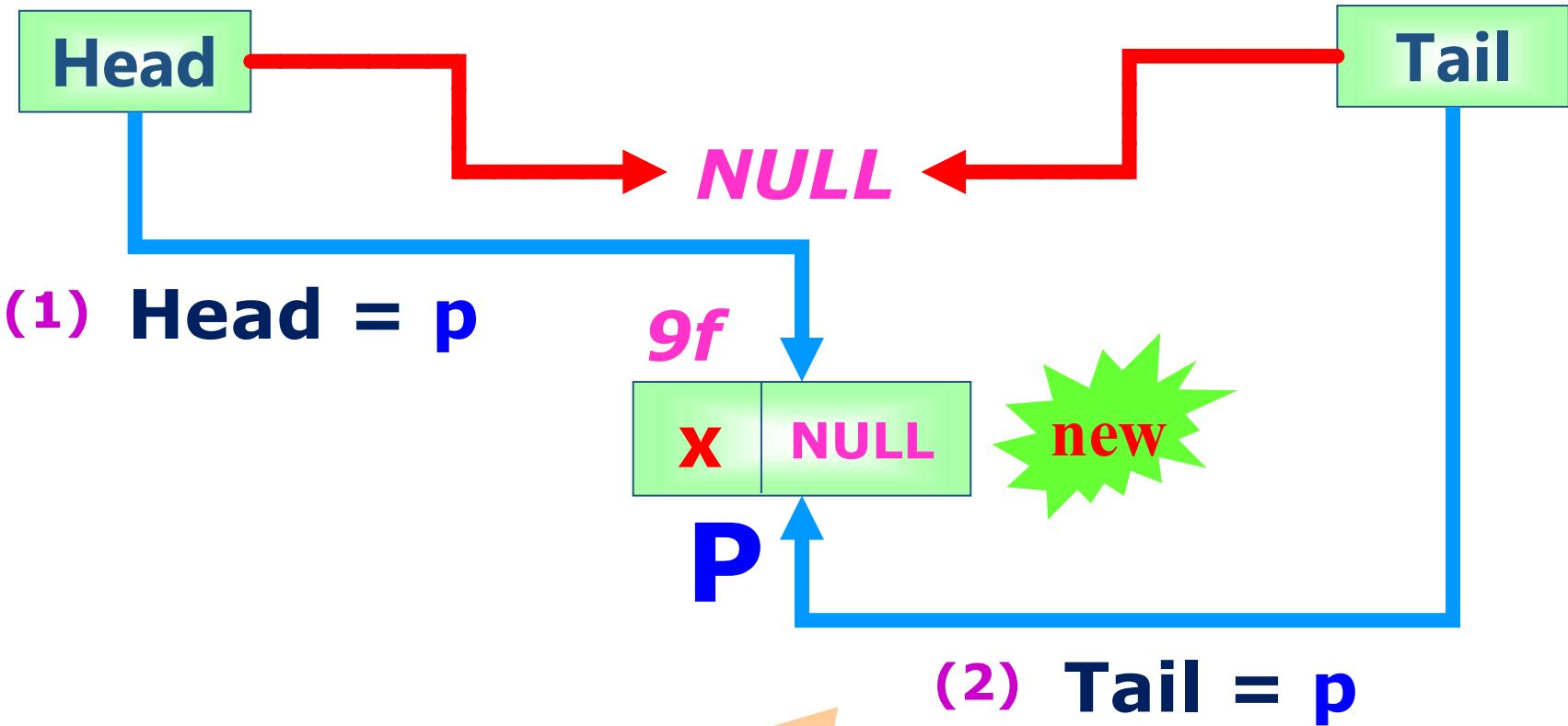
- **Nguyên tắc thêm:** Khi thêm 1 phần tử vào SList thì có làm cho các con trỏ Head, Tail thay đổi?
- **Các vị trí cần thêm 1 phần tử vào SList:**
 - Thêm phần tử vào đầu SList
 - Thêm phần tử vào cuối SList
 - Thêm phần tử p vào sau 1 phần tử q trong SList

Thêm phần tử vào đầu danh sách

Giải thuật:

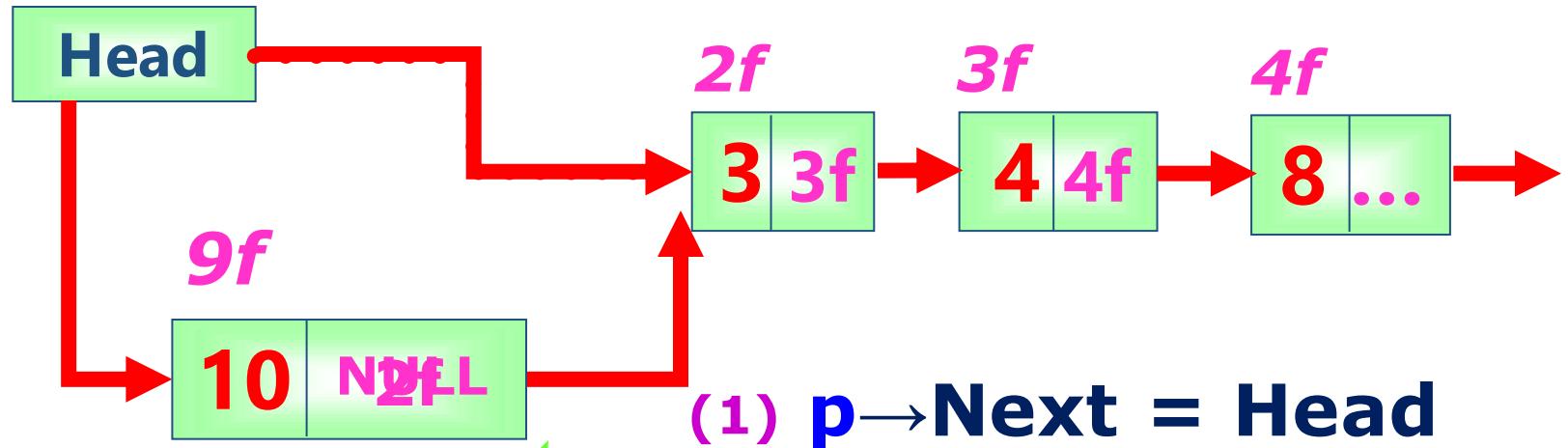
- **Bước 1:** Nếu phần tử muốn thêm p không tồn tại thì không thực hiện.
- **Bước 2:** Nếu SList rỗng thì
 - + Gán Head = p;
 - + Gán Tail = p; //hoặc Tail = Head;
- **Bước 3:** Ngược lại
 - + Gán p→Next = Head;
 - + Gán Head = p;

Ví dụ minh họa Thêm phần tử vào đầu DS



Sử dụng hàm CreateSNode

Ví dụ minh họa Thêm phần tử vào đầu DS



Head = p **P** **new**

Sử dụng hàm CreateSNode

Thêm phần tử vào đầu danh sách

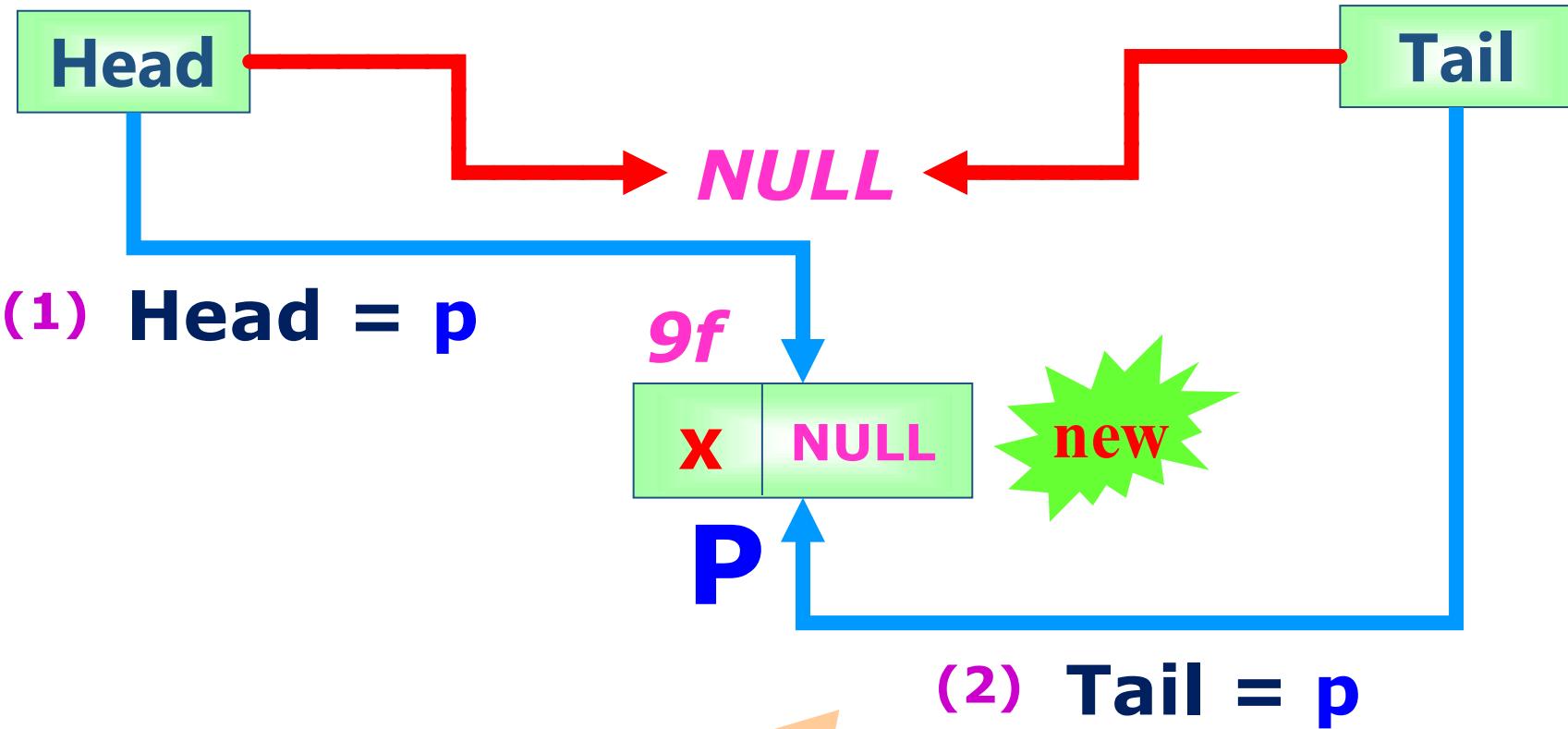
```
int insertHead(SList &sl, SNode* p) {  
    if(p == NULL) return 0;  
    if(isEmpty(sl) == 1) {  
        sl.Head = p;  
        sl.Tail = p;  
    } //p làm nút đầu đồng thời là nút cuối  
    else {  
        p->Next = sl.Head;      (1)  
        sl.Head = p;            (2)  
    }  
    return 1;  
}
```

Thêm phần tử vào cuối danh sách

Giải thuật:

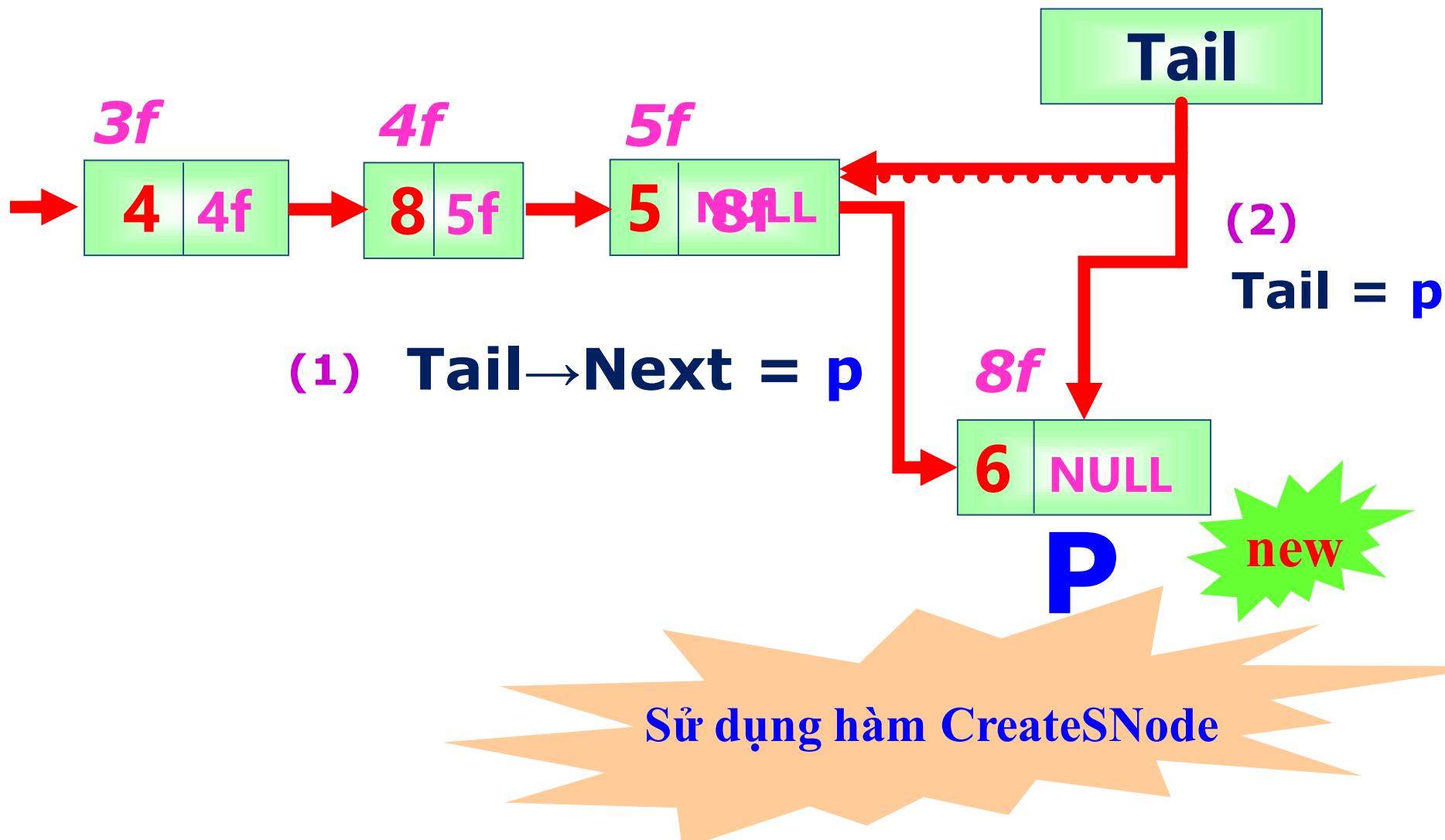
- **Bước 1:** Nếu phần tử muốn thêm p không tồn tại thì không thực hiện.
- **Bước 2:** Nếu SList rỗng thì:
 - + Gán Head = p;
 - + Gán Tail = p;
- **Bước 3:** Ngược lại
 - + Gán Tail→Next = p;
 - + Gán Tail = p;

Ví dụ minh họa Thêm phần tử vào cuối DS



Sử dụng hàm CreateSNode

Ví dụ minh họa Thêm phần tử vào cuối DS



Thêm phần tử vào cuối danh sách

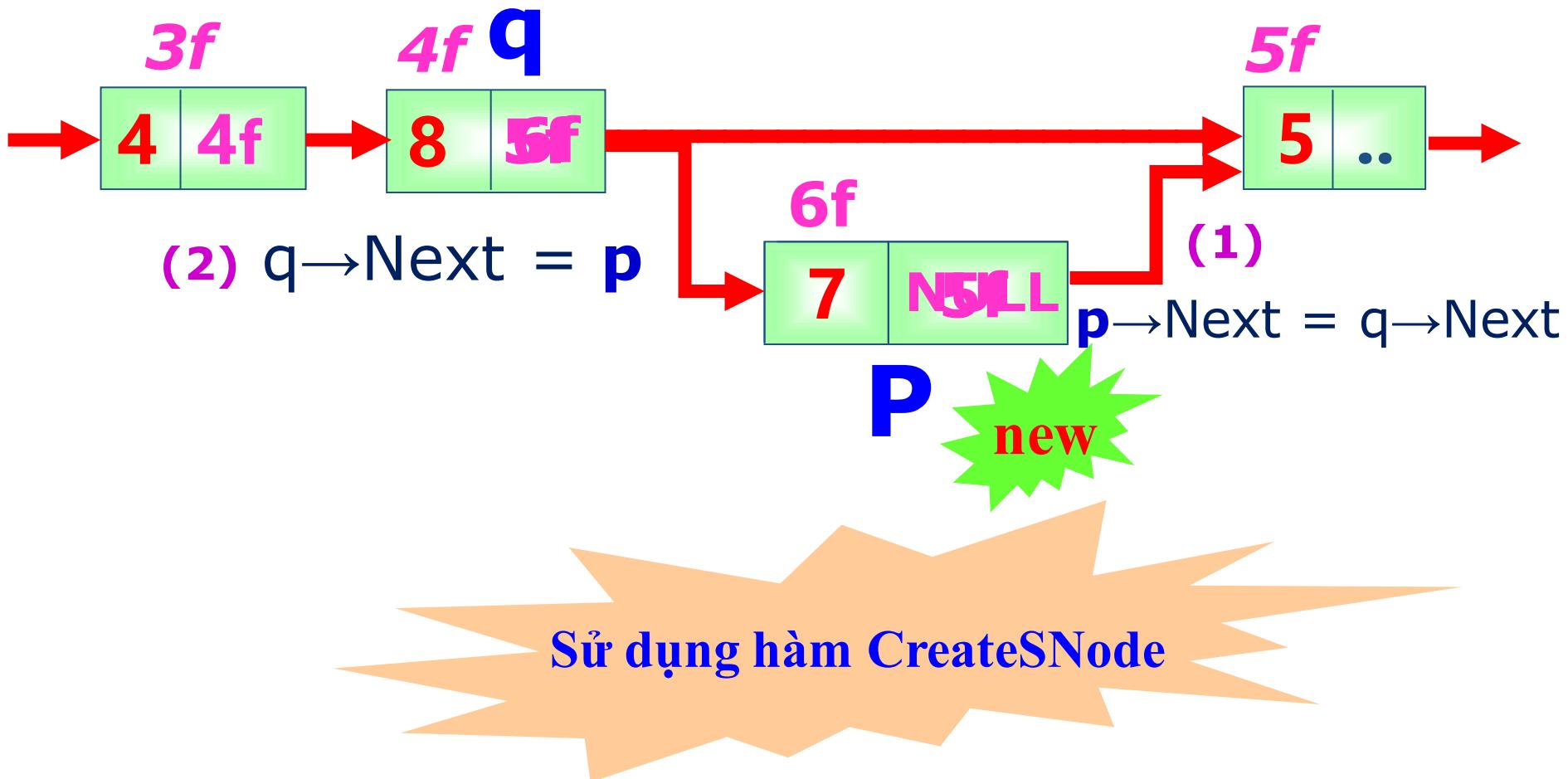
```
int insertTail(SList &sl, SNode* p) {  
    if(p == NULL) return 0;  
    if(isEmpty(sl) == 1) {  
        sl.Head = p;  
        sl.Tail = p;  
    } //p làm nút đầu đồng thời là nút cuối  
    else {  
        sl.Tail->Next = p;      (1)  
        sl.Tail = p;            (2)  
    }  
    return 1;  
}
```

Thêm phần tử p vào sau q của danh sách

Giải thuật:

- **Bước 1:** Nếu phần tử q hoặc phần tử muốn thêm p không tồn tại thì không thực hiện.
- **Bước 2:**
 - + Gán $p \rightarrow \text{Next} = q \rightarrow \text{Next};$
 - + Gán $q \rightarrow \text{Next} = p;$
 - + Nếu ***q là phần tử cuối*** thì:
Gán Tail = p;

Ví dụ minh họa Thêm p vào sau q



Thêm phần tử p vào sau q của danh sách

```
int insertAfter(SList &sl, SNode* q, SNode* p)
```

```
{
```

```
    if(q == NULL || p == NULL)
```

```
        return 0;
```

```
    p->Next = q->Next;      (1)
```

```
    q->Next = p;            (2)
```

```
    if(sl.Tail == q)
```

```
        sl.Tail = p;
```

```
    return 1;
```

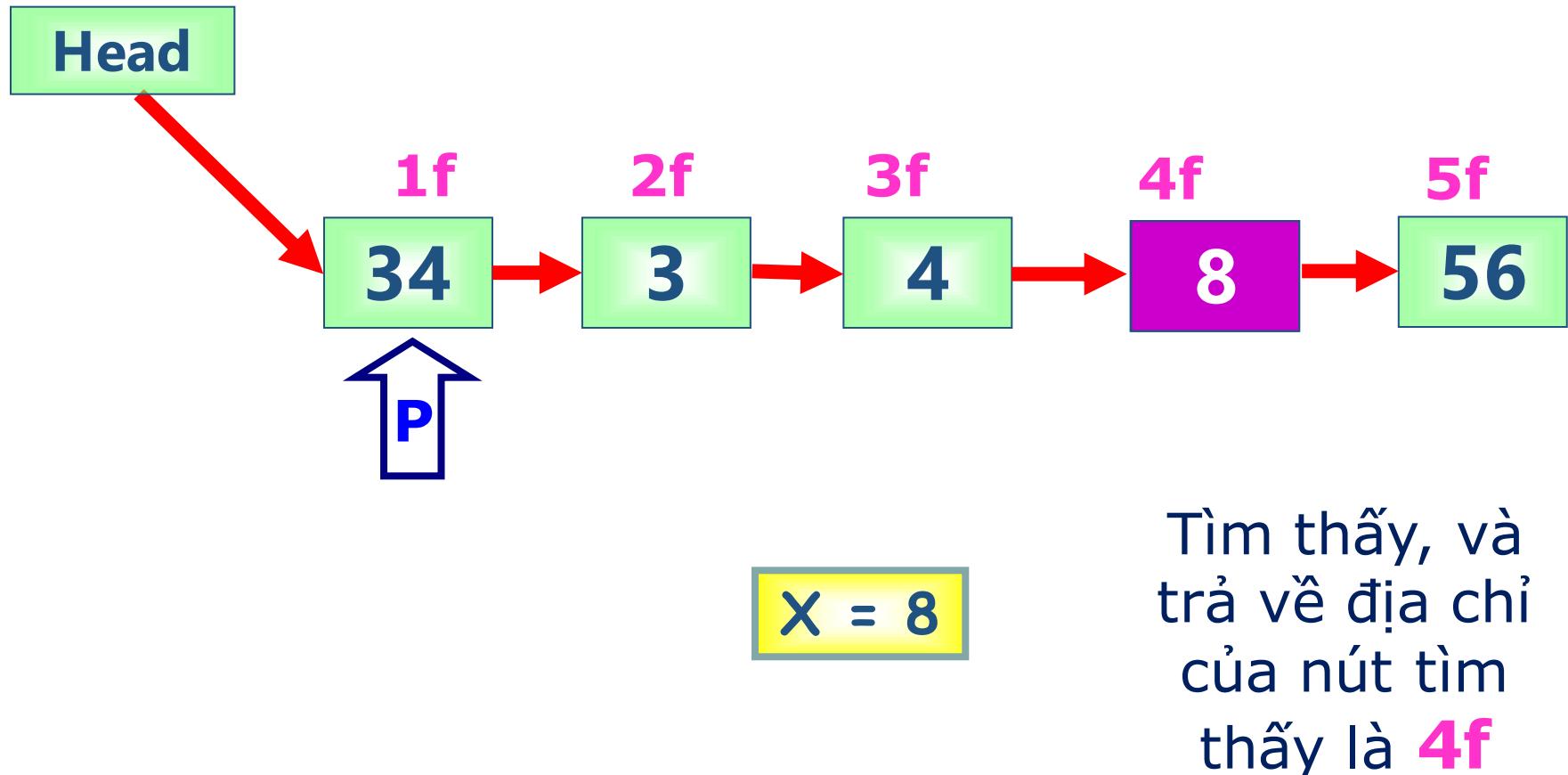
```
}
```

Tìm kiếm phần tử có giá trị bằng x

Giải thuật:

- **Bước 1:** Nếu (`Head == NULL`) thì: trả về `NULL`;
- **Bước 2:** Gán `p = Head; //địa chỉ của phần tử đầu trong SList`
- **Bước 3:**
Lặp lại trong khi (`p != NULL` và `p→Info != x`) thì thực hiện:
`p = p→Next; //xét phần tử kế sau`
- **Bước 4:** Trả về `p; //nếu (p != NULL) thì p lưu địa chỉ của phần tử có khóa bằng x, hoặc NULL là không có phần tử cần tìm.`

Minh họa tìm phần tử trong DSLK



Tìm kiếm phần tử có giá trị bằng x

```
SNode* findSNode(SList sl, ItemType x)
{ //Hàm trả về con trỏ trỏ đến phần tử muốn tìm
    SNode* p = sl.Head;
    while(p != NULL)
    {
        if(p->Info == x)
            return p; //Tìm thấy, trả về địa chỉ
        p = p->Next;
    }
    return NULL; //Không tìm thấy, trả về NULL
}
```

Tìm kiếm phần tử có giá trị bằng x

```
SNode* findSNode(SList sl, ItemType x)
{ //Hàm trả về con trỏ trỏ đến phần tử muốn tìm
    SNode* p;
    for(p = sl.Head; p != NULL; p = p→Next)
        if(p→Info == x)
            return p; //Tìm thấy, trả về địa chỉ
    return NULL; //Không tìm thấy, trả về NULL
}
```

Thao tác xóa phần tử trong danh sách

➤ Nguyên tắc xóa:

- Phải cô lập phần tử cần xóa trước khi xóa nó.
- Khi xóa 1 phần tử trong SList thì có làm cho các con trỏ Head, Tail thay đổi?

➤ Các vị trí cần xóa 1 phần tử trong SList:

- Xóa phần tử đầu/cuối SList
- Xóa phần tử p sau/trước 1 phần tử q trong SList
- Xóa phần tử có giá trị x bất kỳ trong SList
- Xóa toàn bộ SList

➤ Giải phóng vùng nhớ của nút bằng hàm **delete**

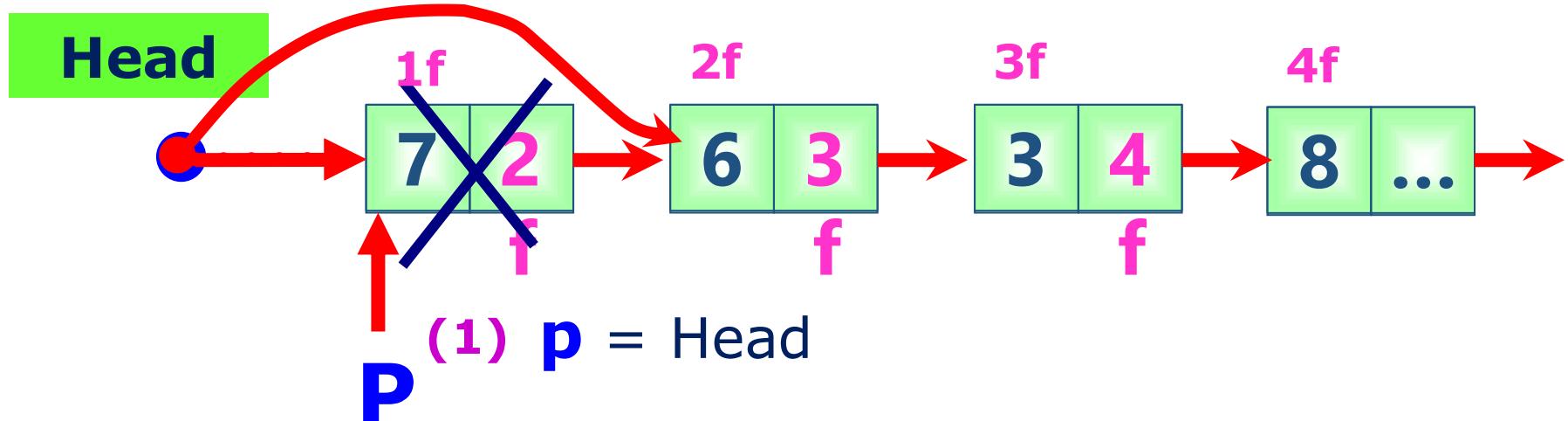
Xóa phần tử đầu của danh sách

Giải thuật:

- **Bước 1:** Nếu (`Head == NULL`) thì: không thực hiện gì cả.
- **Bước 2:** Khi (`Head != NULL`) thì: thực hiện các công việc sau:
 - + **Bước 2.1:** Gán `p = Head;`
 - + **Bước 2.2:** Gán `Head = Head→Next;`
 - + **Bước 2.3:** Nếu (`Head == NULL`) thì: Gán `Tail = NULL;`
 - + **Bước 2.4:** Lưu lại thông tin nút bị xóa;
 - + **Bước 2.5:** `delete p;` //Hủy nút do `p` trỏ đến (con trỏ `p`)

Ví dụ minh họa hủy nút đầu DS

(2) Head = Head→Next



(3) delete p

Xóa phần tử đầu danh sách

```
int deleteHead(SList &sl, ItemType &x)
```

```
{
```

```
    if(isEmpty(sl) == 1) return 0;
```

```
    SNode* p = sl.Head;      (1)
```

```
    sl.Head = sl.Head→Next; (2)
```

```
    if(sl.Head == NULL)
```

```
        sl.Tail = NULL;
```

```
    x = p→Info; //lưu ItemType của nút cần hủy
```

```
delete p; //Xóa nút p
```

```
    return 1; //Xóa thành công
```

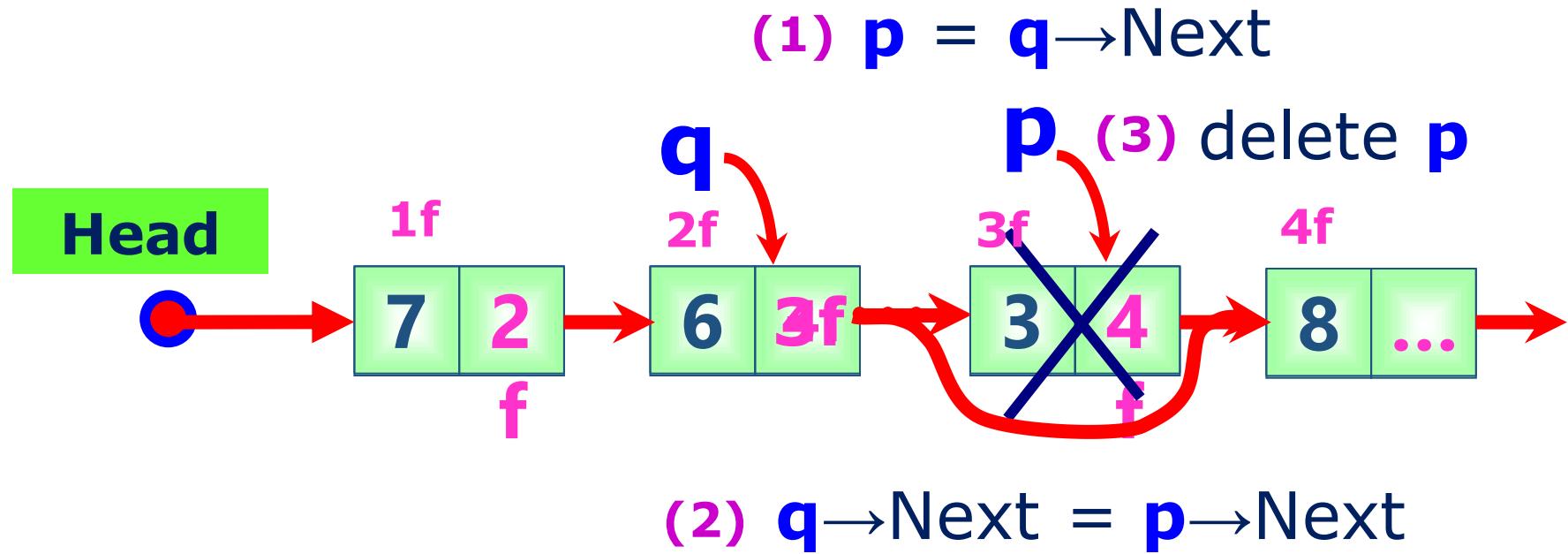
```
}
```

Xóa phần tử p sau phần tử q của danh sách

Giải thuật:

- **Bước 1:** Nếu ($q == \text{NULL}$ hoặc $q \rightarrow \text{Next} == \text{NULL}$) thì: không thực hiện.
- **Bước 2:** Khi ($q != \text{NULL}$ và $q \rightarrow \text{Next} != \text{NULL}$) thì: thực hiện các công việc sau:
 - + **Bước 2.1:** Gán $p = q \rightarrow \text{Next}$;
 - + **Bước 2.2:** Gán $q \rightarrow \text{Next} = p \rightarrow \text{Next}$;
 - + **Bước 2.3:** Nếu ($\text{Tail} == p$) thì: Gán $\text{Tail} = q$;
 - + **Bước 2.4:** Lưu lại thông tin nút bị xóa;
 - + **Bước 2.5:** **delete p;** //Hủy nút do p trỏ đến (con trỏ p)

Ví dụ minh họa hủy nút p sau nút q



Xóa phần tử sau q trong danh sách

```
int deleteAfter(SList& sl, SNode* q, ItemType &x)
```

```
{
```

```
    if(q == NULL || q→Next == NULL) return 0;
```

```
    SNode* p = q→Next; (1)
```

```
    q→Next = p→Next; (2)
```

```
    if(sl.Tail == p)
```

```
        sl.Tail = q;
```

```
    x = p→Info;
```

```
delete p; //Xóa nút p
```

```
return 1; //Xóa thành công
```

```
}
```

Xóa phần tử có khóa x của danh sách

Giải thuật:

- **Bước 1:** Nếu (`Head == NULL`) thì: không thực hiện gì cả.
- **Bước 2:** Tìm phần tử p có khoá bằng x, và q là phần tử đứng kế trước p.
- **Bước 3:** Nếu (`p == NULL`) thì: Không có nút chứa x nên không thực hiện.
- **Bước 4:** *//Ngược lại (`p != NULL`) → nghĩa là tồn tại nút p chứa khóa x*
 - + Nếu (`p == Head`) thì: *//p là nút đầu danh sách*
DeleteHead(sl, x); *//Gọi hàm xóa nút đầu*
 - + Ngược lại:
DeleteAfter(sl, q, x); *//Xóa nút p chứa x kế sau nút q*

Xóa phần tử có khóa x của danh sách

```
int deleteSNodeX(SList &sl, ItemType x) {  
    if(isEmpty(sl) == 1) return 0; //Không thực hiện được  
    SNode* p = sl.Head;  
    SNode* q = NULL; //sẽ trỏ đến nút kế trước p  
    while( (p != NULL) && (p→Info != x) )  
        {//vòng lặp tìm nút p chứa x, q là nút kế trước p  
            q = p; p = p→Next;  
        }  
    if(p == NULL) return 0; //không tìm thấy phần tử có khóa x  
    if(p == sl.Head) //p có khóa bằng x là nút đầu danh sách  
        deleteHead(sl, x);  
    else //xóa nút p có khóa x nằm kế sau nút q  
        deleteAfter(sl, q, x);  
    return 1; //Thực hiện thành công  
}
```

Xóa phần tử có khóa x của danh sách

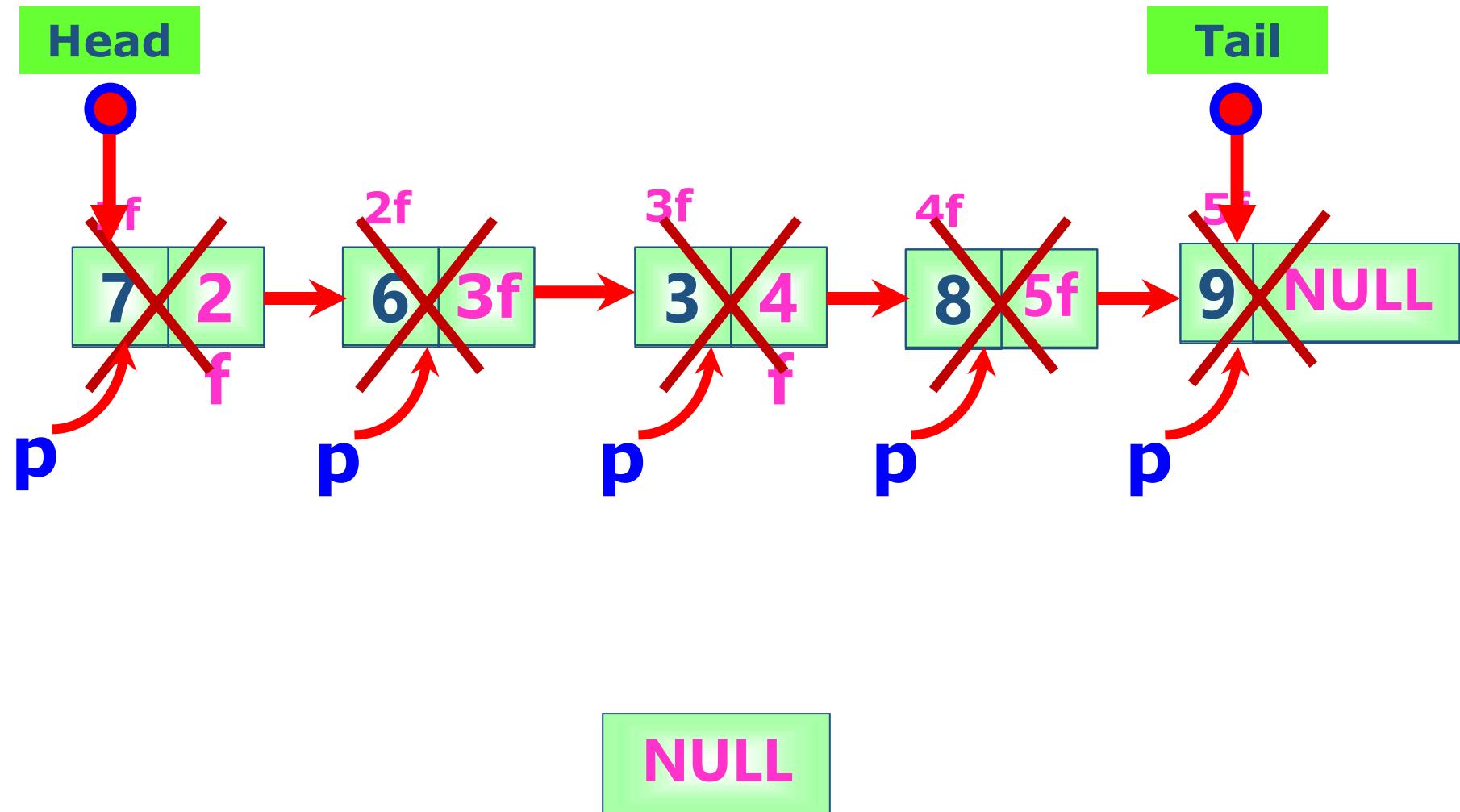
```
int deleteSNodeX(SList &sl, ItemType x) {  
    if(isEmpty(sl) == 1) return 0; //Không thực hiện được  
    SNode* p = findSNode(sl, x);  
    if(p == NULL) return 0; //Thực hiện không thành công  
    if(p == sl.Head)  
        deleteHead(sl, x);  
    else  
    {  
        SNode* q = sl.Head; //tìm nút q kế trước nút p  
        while(q->Next != p)  
            q = q->Next;  
        deleteAfter(sl, q, x);  
    }  
    return 1; //Thực hiện thành công  
}
```

Xóa toàn bộ danh sách

Giải thuật:

- **Bước 1:** Nếu (Head == NULL) thì: không thực hiện gì cả.
- **Bước 2:** Lặp lại trong khi (danh sách còn phần tử) thì thực hiện lần lượt những việc sau:
 - + **Bước 2.1:** Gán p = Head;
 - + **Bước 2.2:** Gán Head = Head→Next; //Cập nhật Head
 - + **Bước 2.3:** Hủy con trỏ p;
- **Bước 3:** Gán Tail = NULL; //bảo toàn tính nhất quán khi danh sách rỗng

Ví dụ minh họa Hủy danh sách



Xóa toàn bộ danh sách

```
void deleteAllSList(SList &sl)  
{  
    while(sl.Head != NULL)  
    {//trong khi còn phần tử trong SList  
        SNode *p = sl.Head;  
        sl.Head = sl.Head→Next;  
        delete p; //Xóa nút p  
    }  
    sl.Tail = NULL;  
}
```

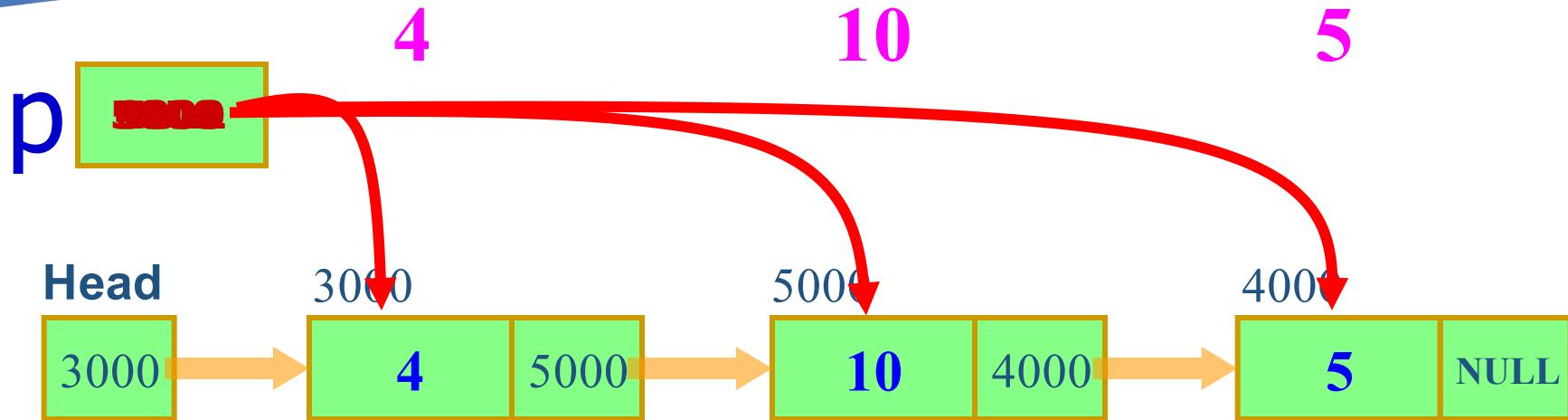
Xử lý danh sách

```
void processSList(SList sl) {  
    if(isEmpty(sl) == 1) {  
        printf("Danh sách rỗng!");  
        return;  
    }  
    SNode *p = sl.Head;  
    while(p != NULL) {  
        processSNode(p); //xử lý cụ thể tùy trường hợp  
        p = p→Next;  
    }  
}
```

Duyệt in danh sách: *Cách 1*

```
void showSList(SList sl) {  
    if(isEmpty(sl) == 1) {  
        printf("Danh sách rỗng!");  
        return;  
    }  
    SNode *p = sl.Head;  
    while(p != NULL) {  
        printf("%4d", p→Info); //Xuất ra màn hình  
        p = p→Next;  
    }  
}
```

Minh họa duyệt in danh sách



```
SNode* p = sl.Head;  
while(p != NULL)  
{  
    printf("%4d", p->Info);  
    p = p->Next;  
}
```



Duyệt in danh sách: cách 2

```
void showSList(SList sl)
{
    if(isEmpty(sl) == 1)
    {
        printf("\nDanh sach rong!");
        return;
    }
    printf("\nNoi dung cua danh sach la: ");
    for(SNode* p=sl.Head; p!=NULL; p=p→Next)
        printf("%4d", p→Info);
}
```

Tìm phần tử lớn nhất

```
int findMax(SList sl)
```

{*//Hàm trả về giá trị phần tử lớn nhất*

```
    int max = sl.Head→Info;
```

```
    for(SNode* p = sl.Head→Next; p != NULL; p =  
        p→Next)
```

```
        if(max < p→Info)
```

```
            max = p→Info;
```

```
    return max;
```

}

Tìm phần tử lớn nhất

```
int findMax(SList sl)
{//Hàm trả về giá trị phần tử lớn nhất
    int max = sl.Head→Info;
    SNode* p = sl.Head→Next;
    while(p != NULL)
    {
        if(max < p→Info)
            max = p→Info;
        p = p→Next;
    }
    return max;
}
```

Tìm phần tử lớn nhất

❖ Tìm địa chỉ chứa phần tử lớn nhất:

SNode* **findMaxPosition**(SList **sl**)

{*//Hàm trả về con trỏ trỏ đến phần tử lớn nhất*

SNode* **pos** = **sl**.Head;

for(SNode* p = **sl**.Head→Next; p != NULL; p =
p→Next)

 if(**pos**→Info < p→Info)

pos = p;

return **pos**;

}

Sắp xếp bằng InterchangeSort

```
void interchangeSort_Asc(SList &sl)
{
    SNode *p, *q;
    for(p = sl.Head; p→Next!=NULL; p = p→Next)
        for(q = p→Next; q!=NULL; q = q→Next)
            if(p→Info > q→Info)
                swap(p→Info, q→Info);
}
```

Sắp xếp bằng SelectionSort

```
void selectionSort_Asc(SList &sl)
{
    SNode *q, *p, *min;
    for(p = sl.Head; p→Next!=NULL; p = p→Next)
    {
        min = p;
        for(q = p→Next; q!=NULL; q = q→Next)
            if(min→Info > q→Info)
                min = q;
        swap(p→Info, min→Info);
    }
}
```

Bài tập bổ sung

1. Đếm số Node có trong danh sách
2. Đếm số phần tử dương trong danh sách
3. Đếm số phần tử lớn hơn phần tử kề sau
4. Kiểm tra mọi phần tử trong danh sách có chẵn không?
5. Kiểm tra danh sách có được sắp xếp tăng?

Bài tập bổ sung

6. Tạo danh sách **sl1** chứa các phần tử dương trong danh sách đã cho.
7. Xóa 1 phần tử có khoá là x
8. Thêm phần tử x vào danh sách đã có thứ tự (tăng) sao cho sau khi thêm vẫn có thứ tự (tăng).

Bài tập bổ sung

9. Xóa các phần tử trùng nhau trong danh sách, chỉ giữ lại duy nhất một phần tử (*)
10. Trộn hai danh sách có thứ tự tăng thành một danh sách cũng có thứ tự tăng. (*)
11. Chèn một phần tử có khoá x vào vị trí pos bất kỳ trong danh sách.

Thank for your attention!



See you Next week!

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT



GV : ThS. Trần Văn Thọ

E-mail : thotv@hufi.edu.vn

Môn: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

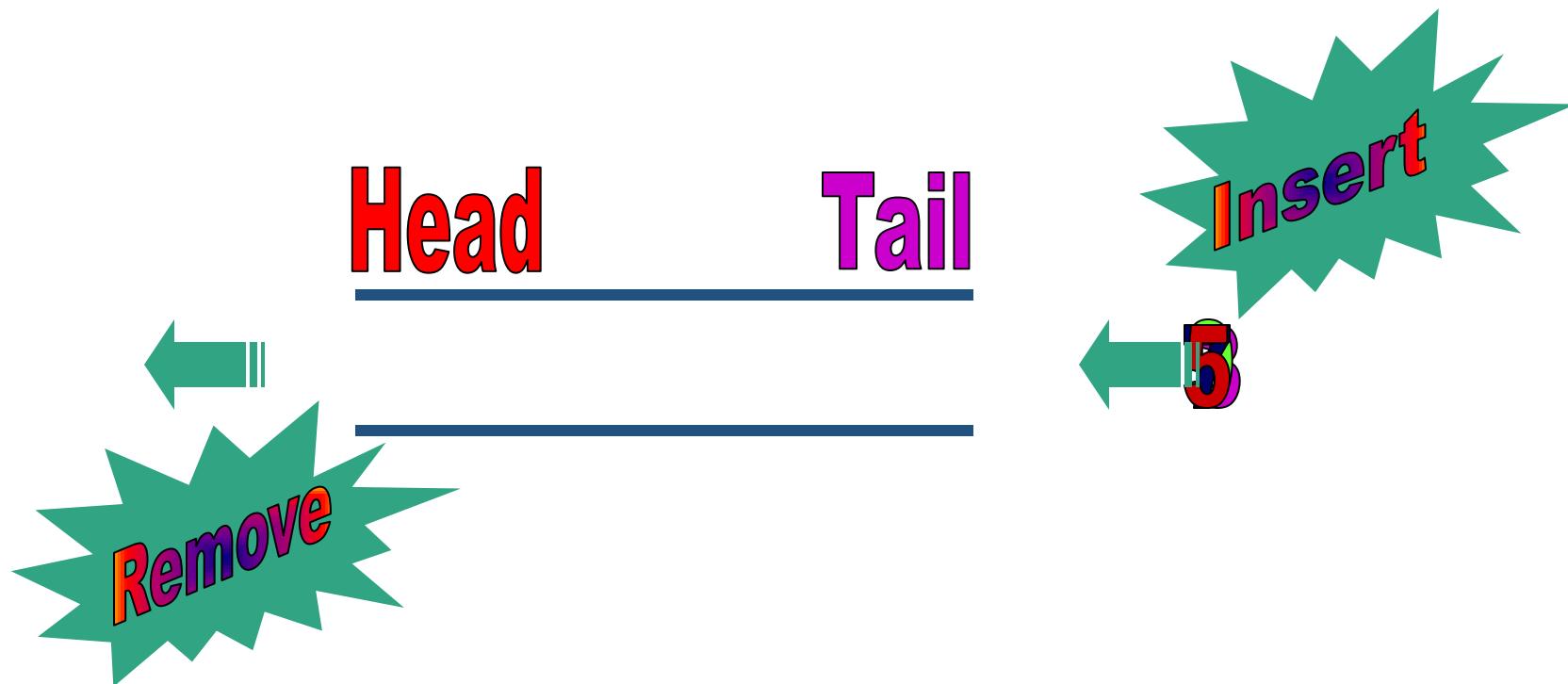


CHƯƠNG IV: STACK VÀ QUEUE

Queue

GIỚI THIỆU QUEUE

- ❖ Cơ chế làm việc: First In First Out - FIFO
- ❖ Thêm vào ở cuối, và lấy ra ở đầu



MÔ TẢ QUEUE

Queue dùng DSLK:

- Con trỏ **Head** trỏ đầu hàng đợi
- Con trỏ **Tail** trỏ đến cuối hàng đợi
- Thao tác **Remove** diễn ra ở **Head**
- Thao tác **Insert** diễn ra ở **Tail**
- Thao tác thêm xóa dễ dàng ở hai đầu

ỨNG DỤNG QUEUE

- Hàng đợi được ứng dụng trong những bài toán giải quyết theo cơ chế “*Vào trước ra trước*” – **FIFO (First In First Out)**:
 - Các ứng dụng đặt vé xe lửa, vé máy bay, công chứng giấy tờ, máy in, ...
 - Các hệ thống rút tiền, ...

TỔ CHỨC QUEUE THEO DSLK

❖ Tạo cấu trúc Node cho Queue

```
struct QueueNode
{
    ItemType Info;
    QueueNode* Next;
};

struct Queue
{
    QueueNode *Head, *Tail;
};
```

TẠO MỚI 1 NODE

```
QueueNode* createQueueNode(ItemType x)
```

```
{
```

```
    QueueNode* p = new QueueNode;
```

```
    if(p == NULL) return NULL;
```

```
    p->Info = x;
```

```
    p->Next = NULL;
```

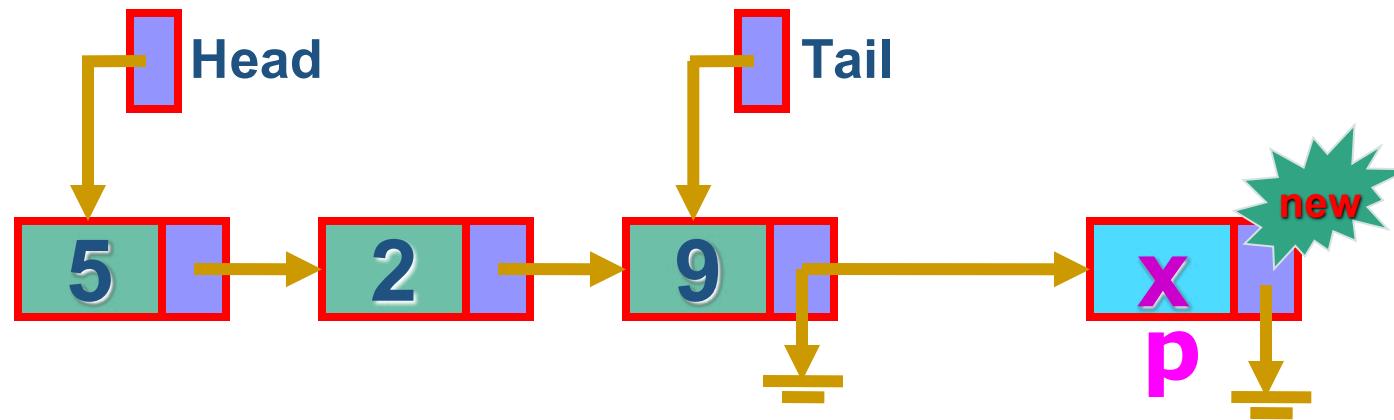
```
    return p;
```

```
}
```

KIỂM TRA QUEUE RỖNG

```
int isEmpty(Queue q)
{ //Nếu hàng đợi rỗng: trả về 1, ngược
lại trả về 0
    if(q.Head == NULL)
        return 1;
    else
        return 0;
    //return (!q.Head);
}
```

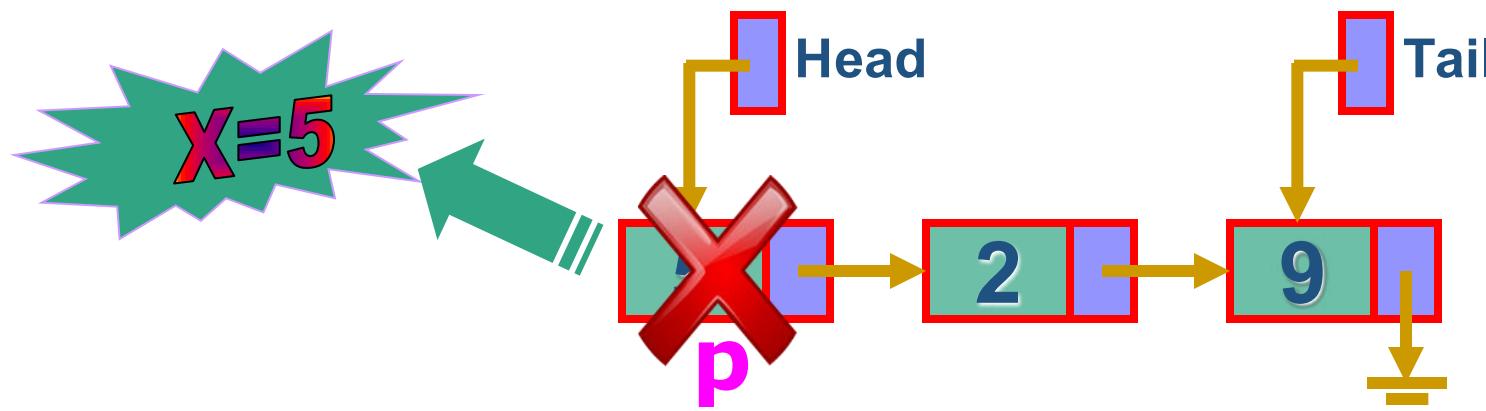
THÊM PHẦN TỬ VÀO QUEUE



THÊM PHẦN TỬ VÀO QUEUE

```
int insert(Queue &q, QueueNode* p)
{
    if(p == NULL)    return 0;
    if(isEmpty(q) == 1)
        q.Head = p;
    else
        q.Tail→Next = p;
    q.Tail = p;
    return 1;
}
```

LẤY PHẦN TỬ KHỎI QUEUE



LẤY PHẦN TỬ KHỎI QUEUE

```
int remove(Queue &q, ItemType &x) {  
    if(isEmpty(q) == 1)    return 0;  
    QueueNode *p = q.Head;  
    q.Head = q.Head→Next;  
    if(q.Head == NULL)  
        q.Tail = NULL;  
    x = p→Info;  
    delete p;  
    return 1;  
}
```

THAM KHẢO THÊM

TỔ CHỨC QUEUE THEO MẢNG 1 CHIỀU

TỔ CHỨC QUEUE THEO MẢNG 1 CHIỀU

Tạo cấu trúc dữ liệu cho QUEUE

```
#define MAXSIZE 100 //chứa 100 Phần  
tử  
struct Queue  
{  
    int n; //Số phần tử hiện hành  
    ItemType Info[MAXSIZE];  
    int Head; //Chỉ số của phần tử đầu  
    int Tail; //Chỉ số của phần tử cuối  
};
```

KHỞI TẠO QUEUE

```
void initQueue(Queue &q)  
{  
    q.n = 0;  
    q.Head = -1;  
    q.Tail = -1;  
}
```

KIỂM TRA QUEUE RỖNG/ ĐẦY

```
int isEmpty(Queue q)  
{  
    return (q.n == 0);  
}
```

```
int isFull(Queue q)  
{  
    return (q.n == MAXSIZE);  
}
```

THÊM PHẦN TỬ VÀO QUEUE

```
int insert(Queue &q, ItemType x)
{ //Add Tail of Queue
    if(isFull(q) == 1) return 0;
    q.Tail=(q.Tail+1) % MAXSIZE;
    q.Info[q.Tail] = x;
    q.n++;
    return 1;
}
```

LẤY PHẦN TỬ KHỎI QUEUE

```
int remove(Queue &q, ItemType &x)
{//Remove Head of Queue
    if(isEmpty(q) == 1) return 0;
    q.Head = (q.Head + 1) % MAXSIZE;
    x = q.Info[q.Head];
    q.n--;
    return 1;
}
```

Thank for your attention!



See you next week!

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

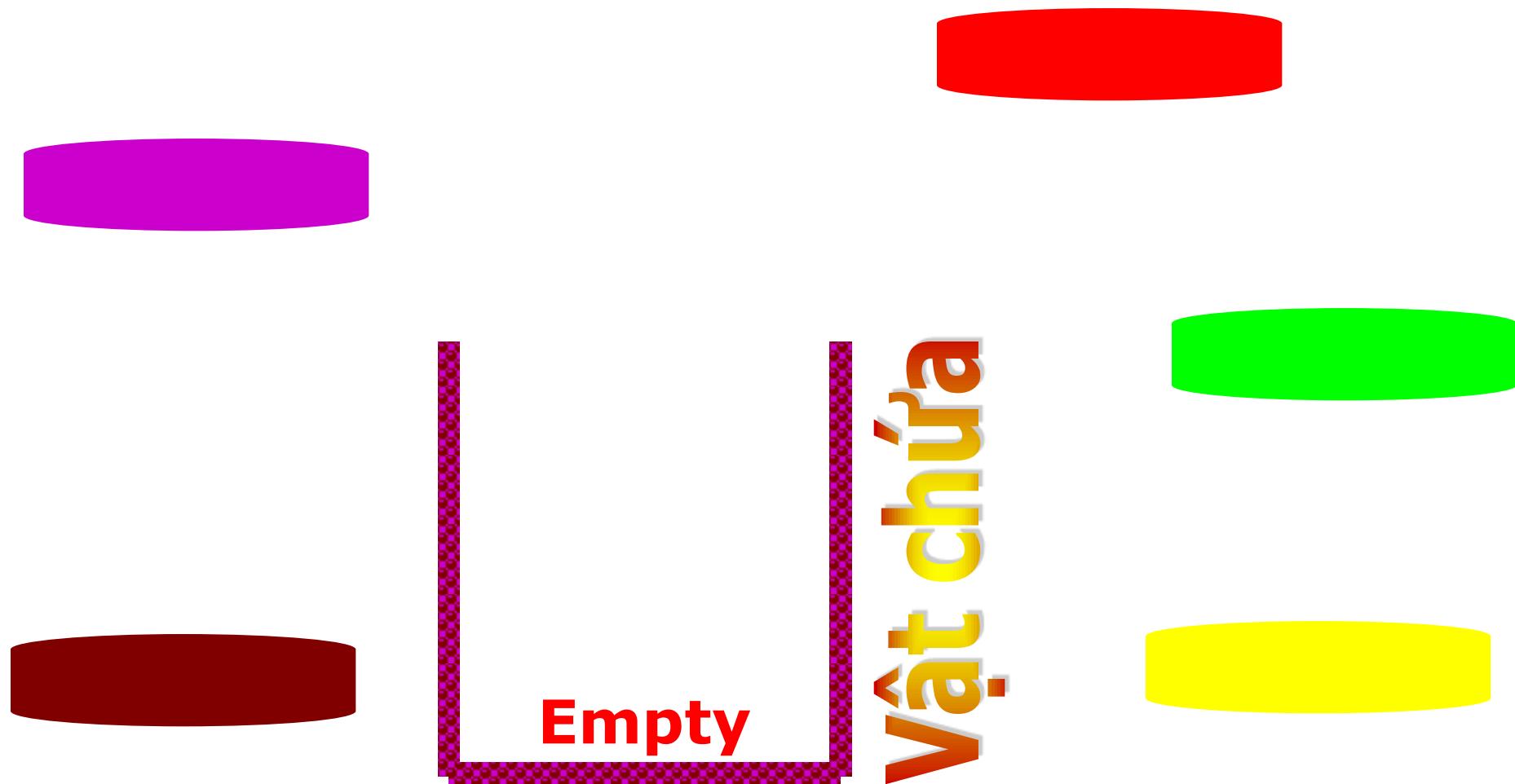


GV : ThS. Trần Văn Thọ

E-mail : thotv@hufi.edu.vn

VÍ DỤ DẪN NHẬP 1

- ❖ Quan sát việc xếp và lấy đĩa như sau:

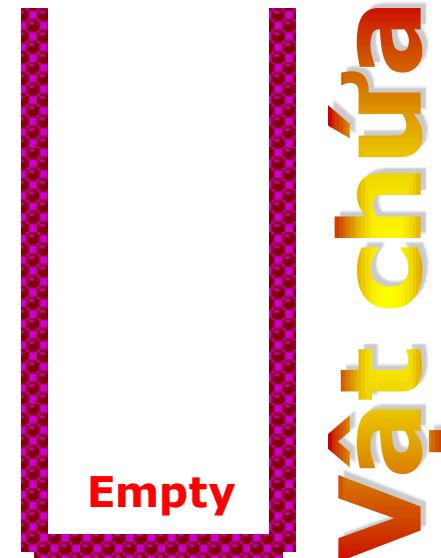


VÍ DỤ DẪN NHẬP 2

Ví dụ: Đổi số $n = 25$ từ hệ thập phân sang hệ nhị phân

Kết quả: $25_{10} =$

$$\begin{array}{r} 25 \\ \hline 2 | 12 + \text{dư } 1 \\ \hline 12 \\ \hline 2 | 6 + \text{dư } 0 \\ \hline 6 \\ \hline 2 | 3 + \text{dư } 0 \\ \hline 3 \\ \hline 2 | 1 + \text{dư } 1 \\ \hline 1 \\ \hline 2 | 0 + \text{dư } 1 \\ \hline \end{array}$$



Môn: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT



CHƯƠNG IV: STACK VÀ QUEUE

Stack

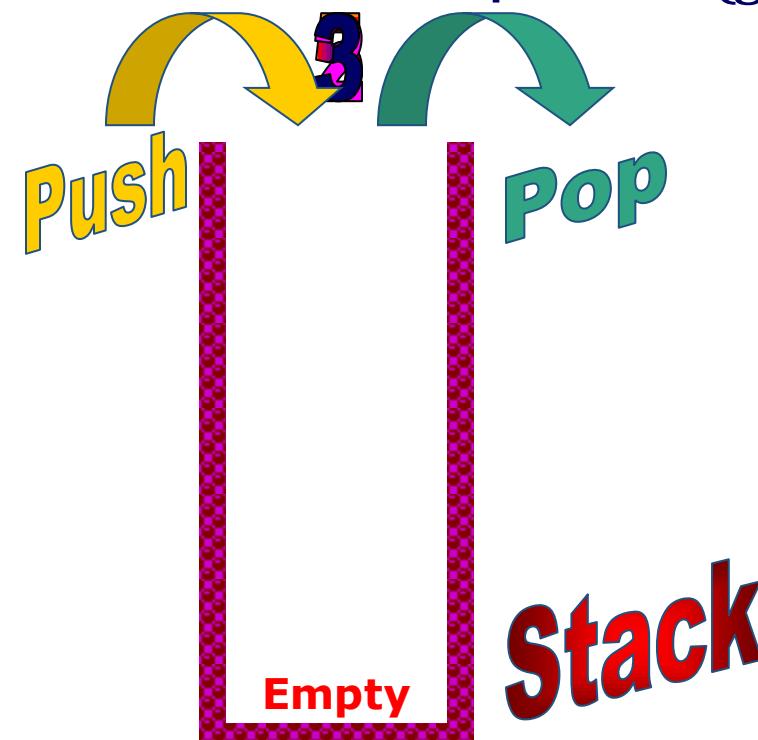
GIỚI THIỆU

- ❖ **Stack** là một đối tượng chứa các đối tượng khác, các đối tượng được **thêm vào** và **lấy ra** chỉ được thực hiện ở một đầu.
- ❖ Vì thế, việc thêm một đối tượng vào Stack hoặc lấy một đối tượng ra khỏi Stack được thực hiện theo cơ chế **LIFO** (**Last In First Out - Vào sau ra trước**).
- ❖ Các đối tượng có thể được thêm vào Stack bất kỳ lúc nào nhưng chỉ có **đối tượng thêm vào sau cùng** mới được phép **lấy ra khỏi Stack trước**.

GIỚI THIỆU

❖ Stack hỗ trợ 2 thao tác chính:

- **Push:** Thêm một đối tượng (*phần tử*) vào Stack.
 - **Pop:** Xóa một đối tượng (*phần tử*) ra khỏi Stack.
- cả 2 thao tác này chỉ diễn ra ở một đầu (*gọi là đỉnh*) của Stack.



GIỚI THIỆU

❖ Stack hỗ trợ các thao tác phụ khác:

- **IsEmpty:** Kiểm tra xem Stack có rỗng không.
- **GetTop:** Trả về giá trị của phần tử nằm ở đỉnh Stack mà không hủy nó ra khỏi Stack.

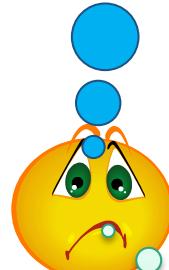
GIỚI THIỆU

- ❖ **Stack thích hợp lưu trữ các loại dữ liệu mà trình tự truy xuất ngược với trình tự lưu trữ.**
- ❖ **Một số ứng dụng của Stack:**
 - Trong trình biên dịch/thông dịch, khi thực hiện Stack được sử dụng để lưu môi trường của các thủ tục.
 - Tổ chức lưu vết các quá trình tìm kiếm theo chiều sâu và quay lui, vét cạn như: tìm đường đi ngắn nhất, bài toán 8 hậu, mã đi tuần,...
 - Khử đệ qui: Tháp Hà Nội, QuickSort, ...
 - Tính giá trị biểu thức toán học, đổi cơ số, đảo chuỗi, ...

HIỆN THỰC STACK

Mảng 1 chiều

Kích thước Stack
khi quá thiếu, lúc
quá thừa



Push / Pop
hơi phức tạp

Danh sách LK

Cấp phát
động!



Push/Pop
khá dễ dàng

TỔ CHỨC STACK THEO D.SÁCH LIÊN KẾT

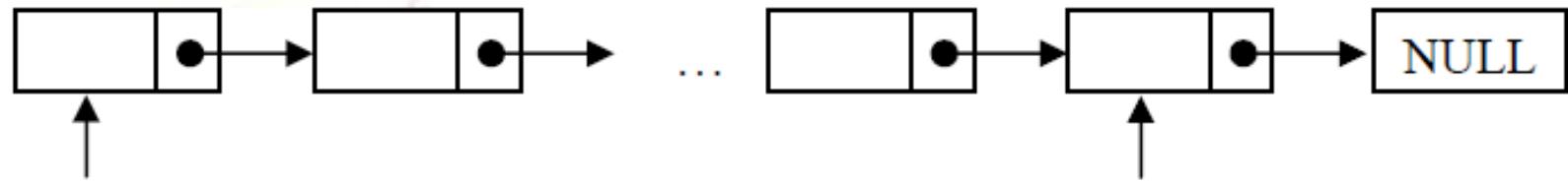
Danh sách Liên kết

Cấp phát động!



Hiện thực Stack dùng DSLK

- ❖ Có thể tạo một Stack bằng cách sử dụng một danh sách liên kết đơn (DSLK).
- ❖ Theo tính chất của danh sách liên kết đơn → phần tử nằm ở đầu danh sách là dễ thêm và xóa nhất.
- ❖ Do đó ta chọn:
 - Phần tử nằm ở đầu danh sách chính là đỉnh của ngăn xếp.
 - Phần tử nằm ở cuối danh sách là đáy ngăn xếp.



Đỉnh ngăn xếp

Đáy ngăn xếp

Hiện thực Stack dùng DSLK

- ❖ **Khai báo ngăn xếp bằng danh sách liên kết:**
 - Chỉ cần 1 biến để lưu phần tử đầu tiên trong ngăn xếp (Top).
- ❖ **Khởi tạo ngăn xếp rỗng:**
 - Biến Top được khởi tạo là NULL.
- ❖ **Kiểm tra ngăn xếp rỗng:**
 - Ngăn xếp rỗng khi Top là NULL.
- ❖ **Kiểm tra ngăn xếp đầy:**
 - Ngăn xếp không có giới hạn về kích thước do đó phương thức này không sử dụng.

Hiện thực Stack dùng DSLK

❖ Thêm một phần tử vào ngăn xếp:

- Tạo ra phần tử mới.
- Cho biến Next của phần tử này trở tới Top hiện thời.
- Top bây giờ sẽ là phần tử mới này.

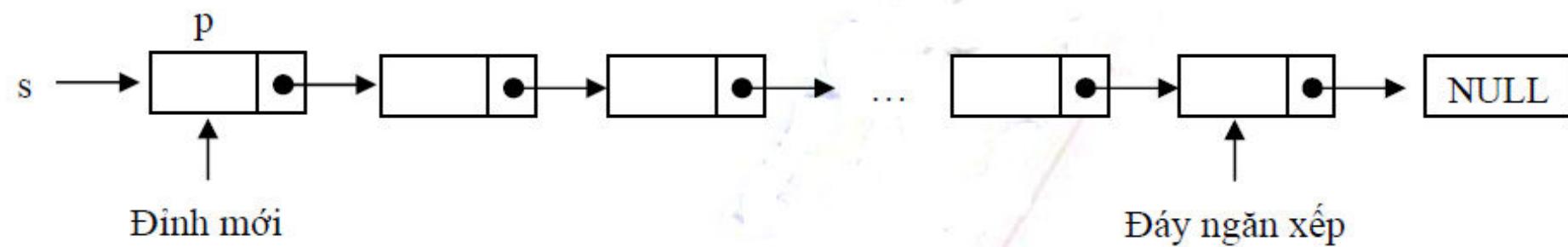
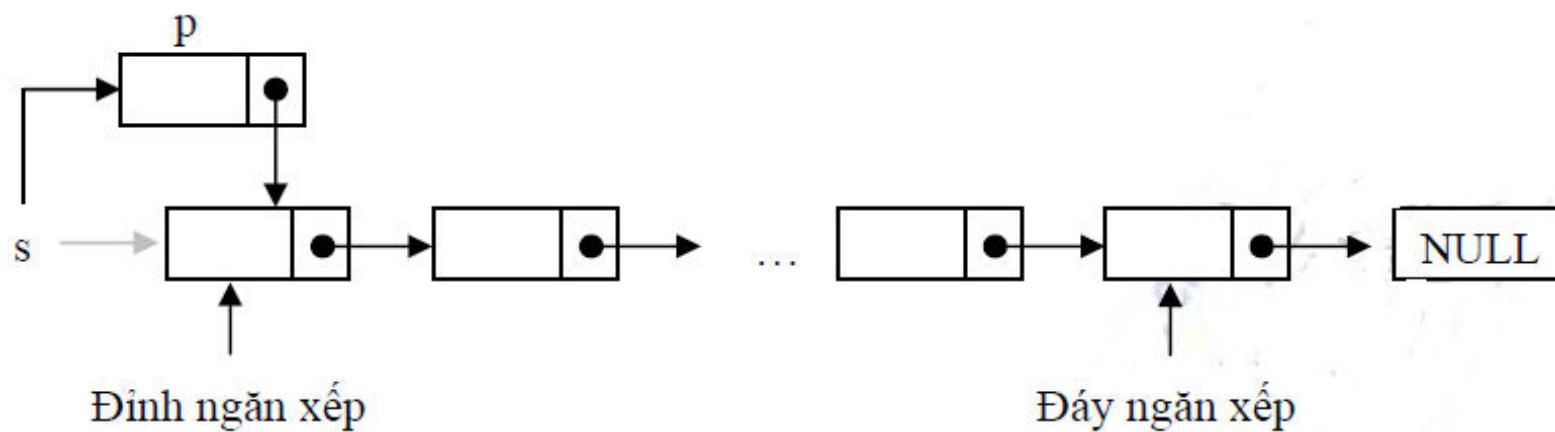
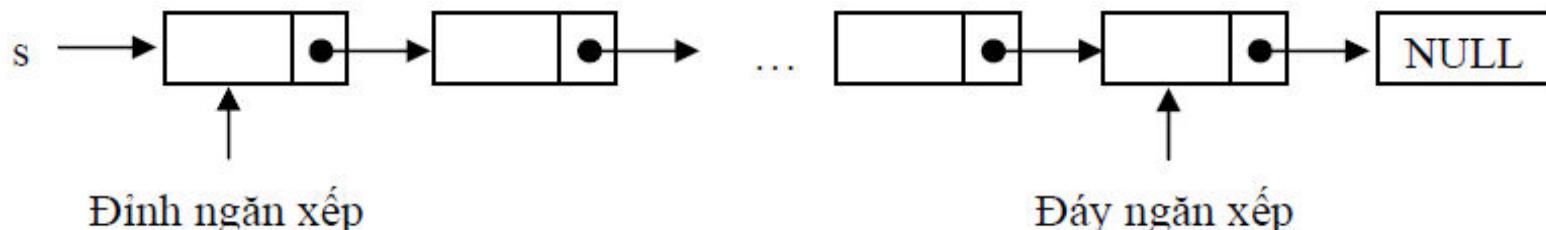
❖ Lấy một phần tử trong ngăn xếp:

- Phần tử được lấy chính là Top của ngăn xếp.

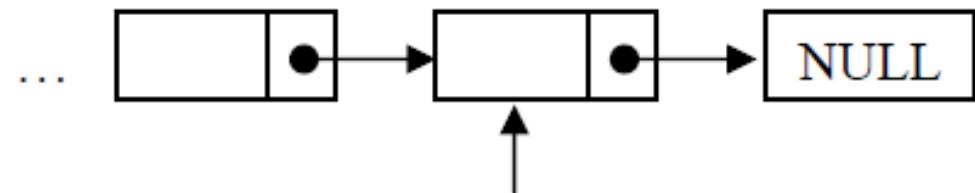
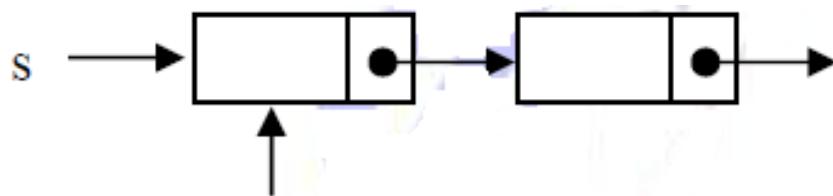
❖ Xóa 1 phần tử khỏi ngăn xếp:

- Top bây giờ là Top → Next.

Thêm một phần tử

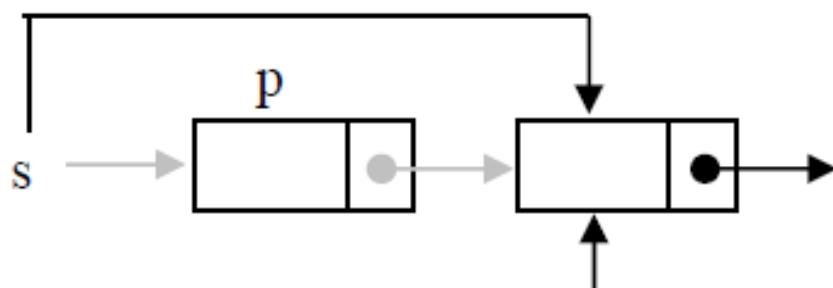


Xóa một phần tử

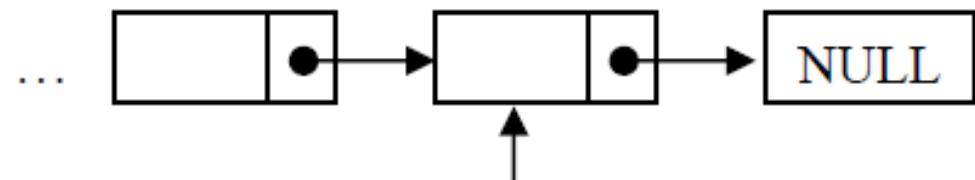


Đỉnh ngăn xếp

Đáy ngăn xếp



Đỉnh mới



Đáy ngăn xếp

KHAI BÁO STACK

Tạo cấu trúc dữ liệu cho Stack theo DSLK đơn:

```
typedef int ItemType;
struct StackNode
{
    ItemType Info;          //Thành phần dữ liệu của StackNode
    StackNode* Next;        //Con trỏ đến StackNode kế sau
};
struct Stack
{
    StackNode* Top;         //Con trỏ đến đỉnh của Stack
};
```

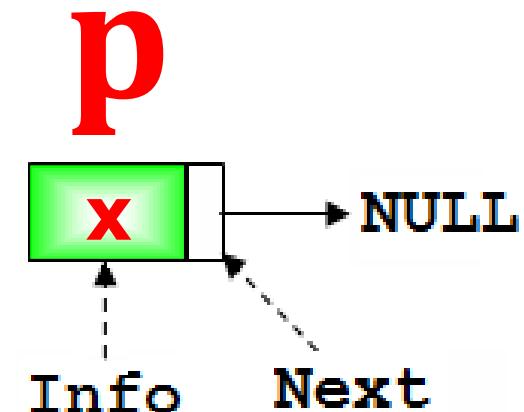
Tạo nút mới chứa giá trị x bất kỳ

❖ Hàm trả về địa chỉ phần tử mới tạo:

StackNode* **createStackNode**(ItemType x)

{

```
StackNode* p = new StackNode;
if(p == NULL) {
    printf("Không đủ bộ nhớ!");
    getch();
    return NULL;
}
p->Info = x;
p->Next = NULL;
return p;
}
```



CÁC THAO TÁC CƠ BẢN TRÊN STACK

- ❖ Stack là một cấu trúc dữ liệu trừu tượng tuyến tính có hai thao tác chính:
 - **push(Stack, O)**: Thêm đối tượng **O** vào **Stack**.
 - **pop(Stack, O)**: Nếu Stack không rỗng thì lấy đối tượng **O** ở đỉnh ra khỏi **Stack** và trả về giá trị của nó.
- ❖ Stack cũng có một số thao tác khác:
 - **initStack(Stack)**: Khởi tạo **Stack** rỗng.
 - **isEmpty(Stack)**: Kiểm tra **Stack** có rỗng không.

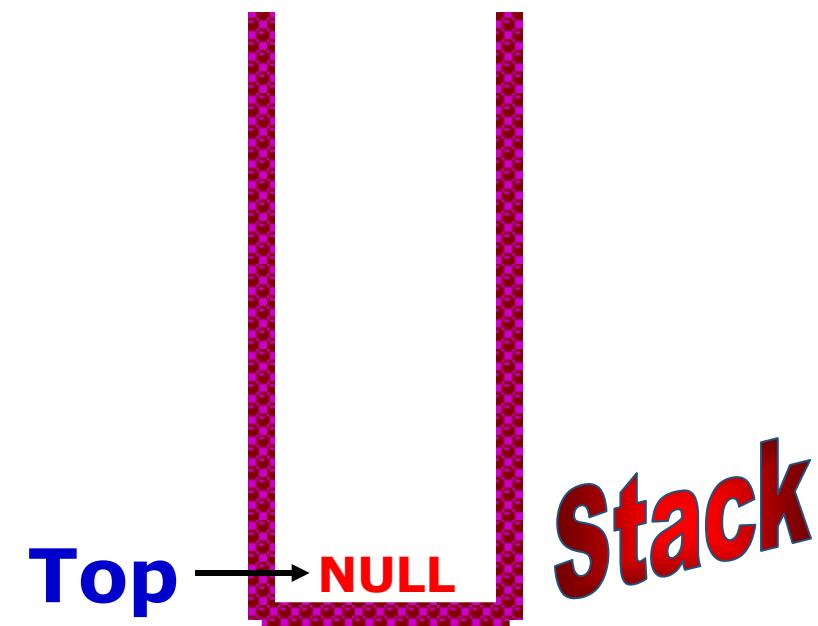
KHỞI TẠO STACK

```
void initStack(Stack &s)
```

```
{
```

```
    s.Top = NULL;
```

```
}
```



KIỂM TRA STACK RỖNG

```
int isEmpty(Stack s)
```

```
{//1: Stack rỗng, 0: Stack không rỗng
```

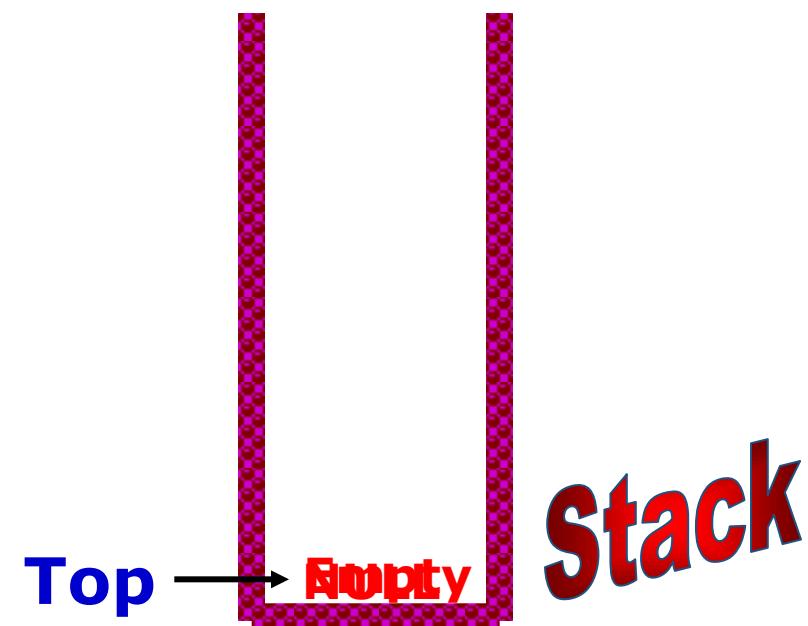
```
    if(s.Top == NULL)
```

```
        return 1;
```

```
    else
```

```
        return 0;
```

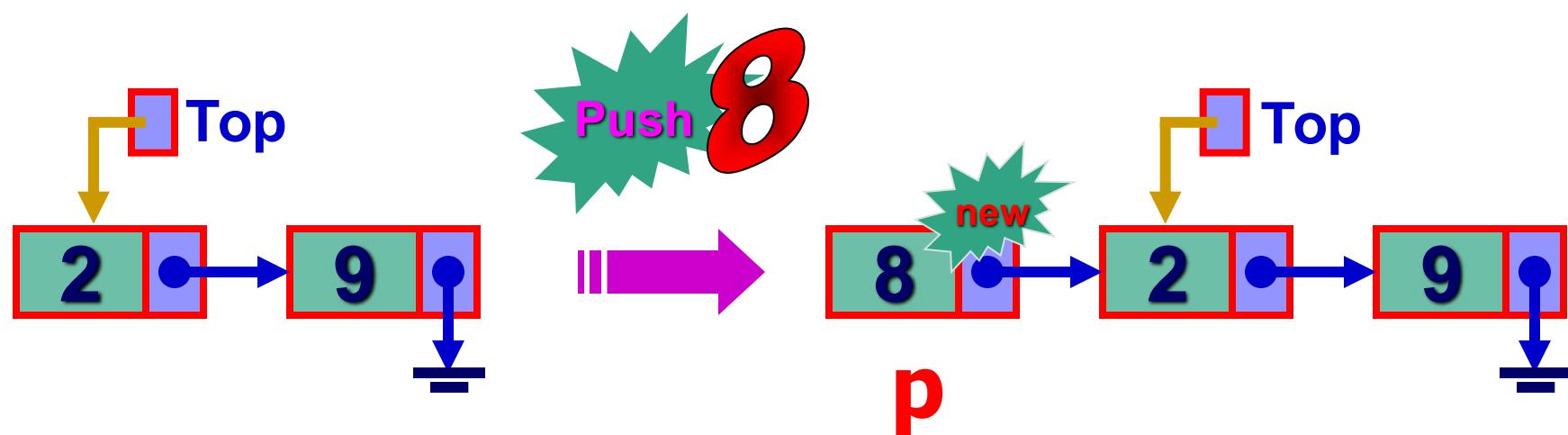
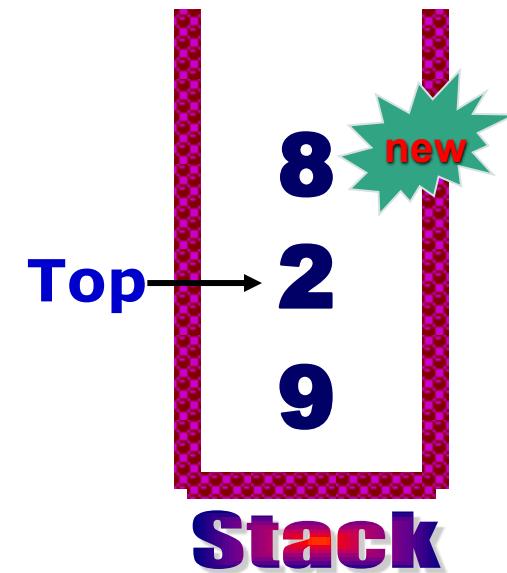
```
}
```



THÊM PHẦN TỬ VÀO STACK

❖ Push

- Tạo nút mới
- Đưa nút mới vào đầu Stack



THÊM PHẦN TỬ VÀO STACK

```
int push(stack &s, StackNode* p)
```

```
{
```

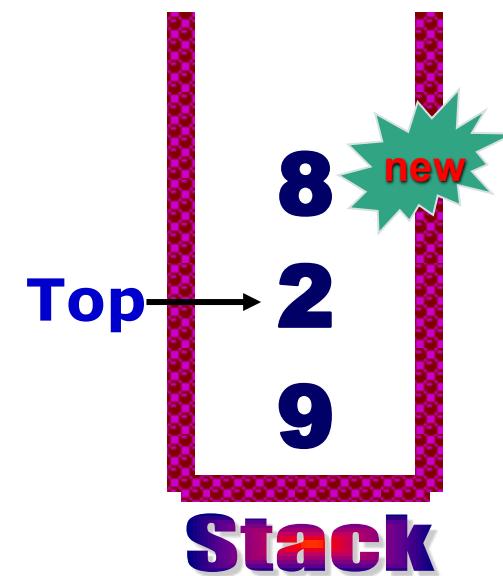
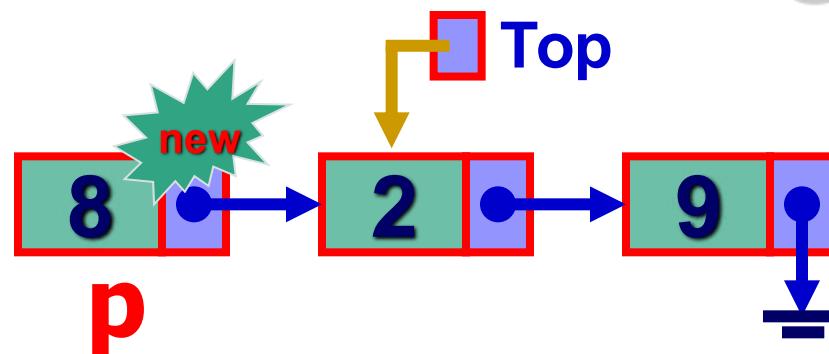
```
    if(p == NULL) return 0;
```

```
    p->Next = s.Top;
```

```
    s.Top = p;
```

```
    return 1;
```

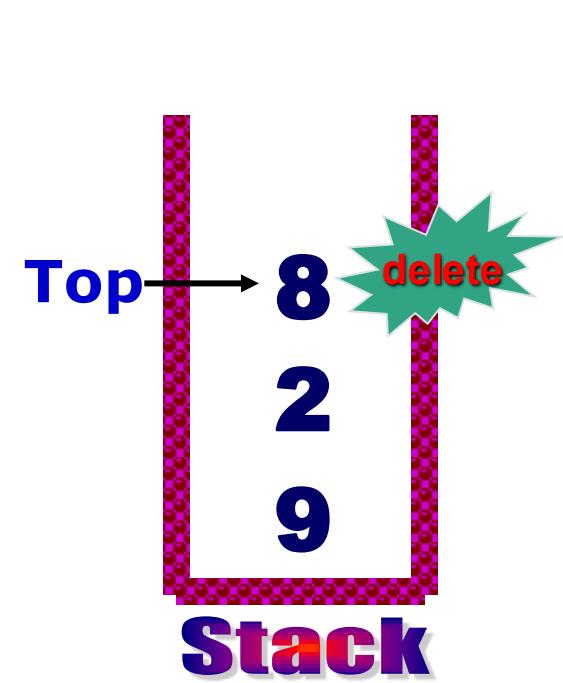
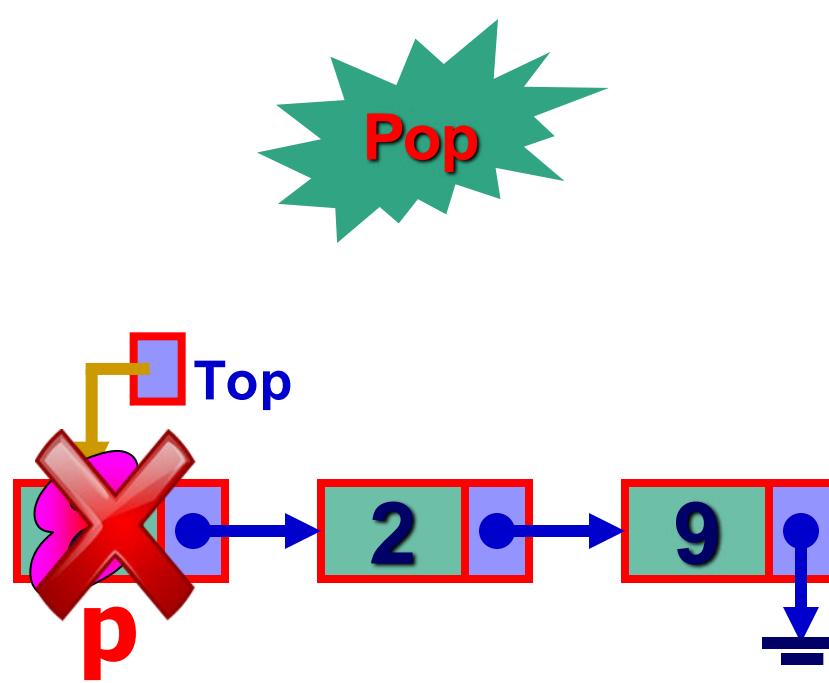
```
}
```



LẤY PHẦN TỬ ĐẦU RA KHỎI STACK

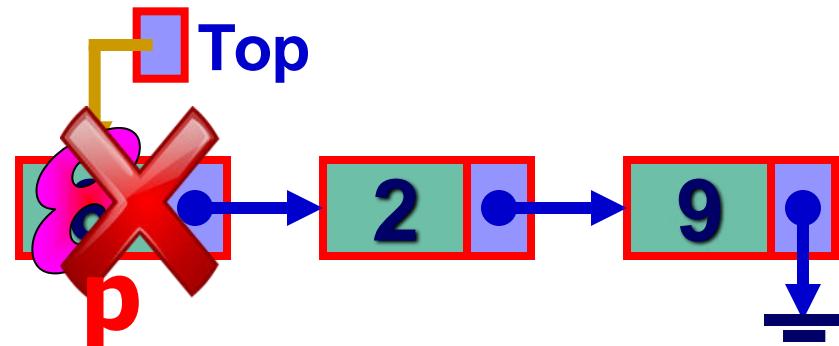
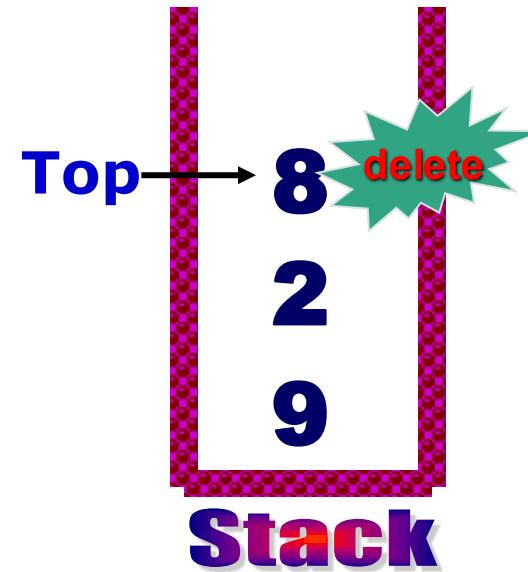
❖ Pop:

- Lấy phần tử đầu ra khỏi danh sách.
- Trả về nội dung và giải phóng nút.



LẤY PHẦN TỬ RA KHỎI STACK

```
int pop(Stack &s, ItemType &x)
{
    if(isEmpty(s) == 1)
        return 0;
    StackNode* p = s.Top;
    s.Top = s.Top->Next;
    x = p->Info;
    delete p;
    return 1;
}
```



ỨNG DỤNG STACK

**Stack có thể được ứng dụng để giải quyết
một số bài toán như:**

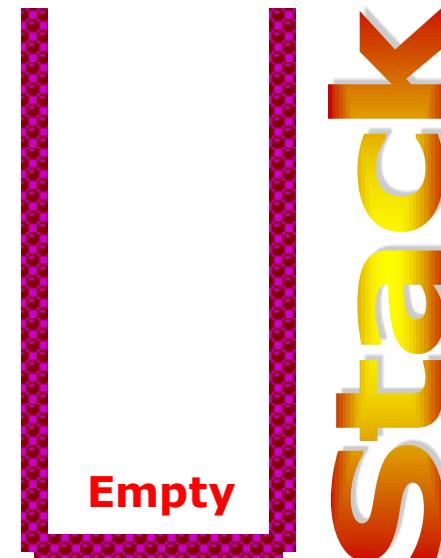
- Đổi cơ số, Đảo chuỗi ký tự, Tính giá trị biểu thức.
- Khử đệ quy: Bài toán tháp **HaNoi**, Thuật toán sắp xếp **QuickSort**, ...
- Áp dụng cho các bài toán dùng mô hình **LIFO** khác như: trình biên dịch/thông dịch, Tổ chức lưu vết các quá trình tìm kiếm theo chiều sâu và quay lui, vét cạn như: tìm đường đi ngắn nhất, bài toán 8 hậu, mã đi tuẫn, ...

ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

Ví dụ: Đổi số $n = 25$ từ hệ thập phân sang hệ nhị phân

Kết quả: $25_{10} =$ 2

$$\begin{array}{r} 25 \\ \hline 2 | 12 + \text{dư } 1 \\ \hline 12 \\ \hline 2 | 6 + \text{dư } 0 \\ \hline 6 \\ \hline 2 | 3 + \text{dư } 0 \\ \hline 3 \\ \hline 2 | 1 + \text{dư } 1 \\ \hline 1 \\ \hline 2 | 0 + \text{dư } 1 \\ \hline \end{array}$$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

Thuật toán:

Bước 1: Khai báo **Stack**.

Bước 2: Khởi tạo **Stack** (*làm cho Stack rỗng*).

Bước 3: Kiểm tra trong khi n còn khác 0 thì lặp lại các công việc sau:

- Tính số dư của phép chia n cho a (*với a là cơ số*).
- Gán lại n bằng phần nguyên của phép chia n cho a.
- Bỏ số dư vừa tính được vào **Stack**.

Bước 4: Kiểm tra trong khi **Stack** còn chưa rỗng thì lần lượt lấy các giá trị ở đỉnh **Stack** ra sử dụng.

ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

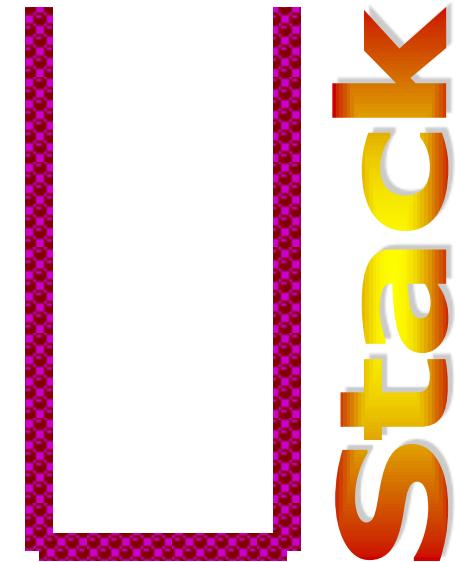
```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```

ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{ //Đổi từ hệ 10 sang một hệ số a bất kỳ.
    int sodu, x, cn = n;
    Stack stack;
    initStack(stack);
    while(n != 0)
    {
        sodu = n%a;
        n /= a;
        push(stack, sodu);
    }
    printf("Kết quả %ld = ", cn);
    while(isEmpty(stack) == 0)
    {
        pop(stack, x);
        printf("%3d", x);
    }
}
```

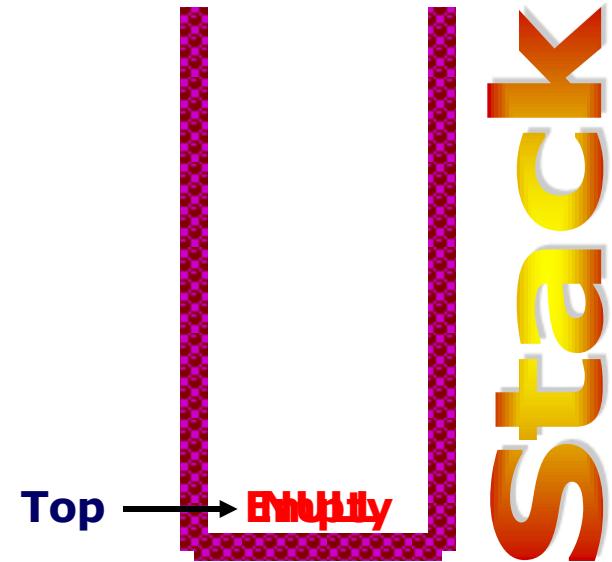
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



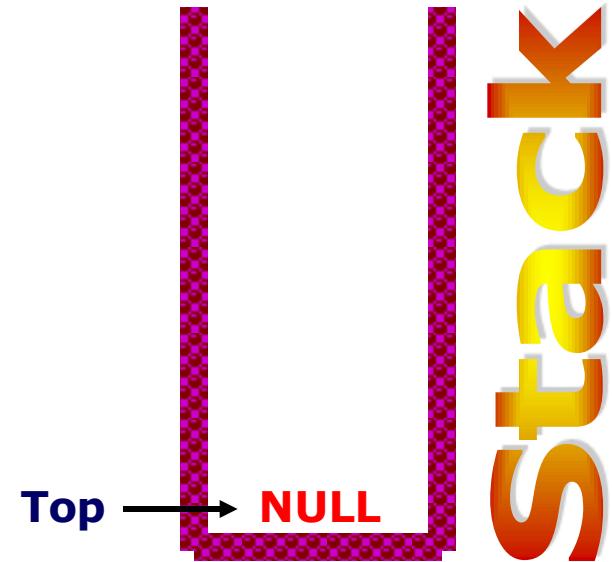
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
    int sodu, x, cn = n;
    Stack stack;
    initStack(stack);
    while(n != 0)
    {
        sodu = n%a;
        n /= a;
        push(stack, sodu);
    }
    printf("Kết quả %ld = ", cn);
    while(isEmpty(stack) == 0)
    {
        pop(stack, x);
        printf("%3d", x);
    }
}
```



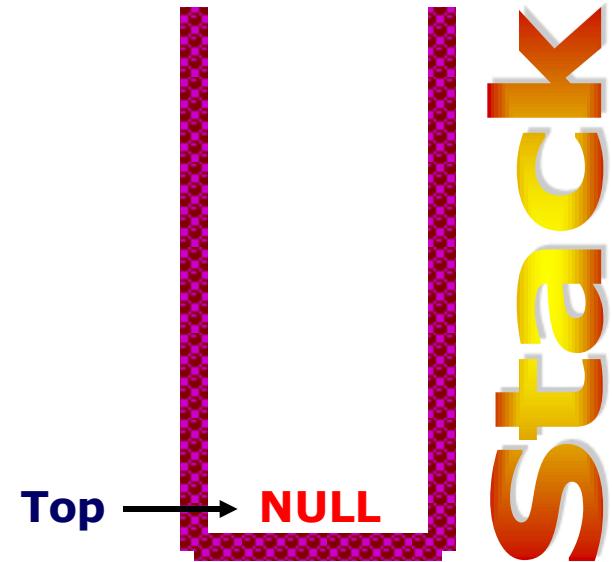
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



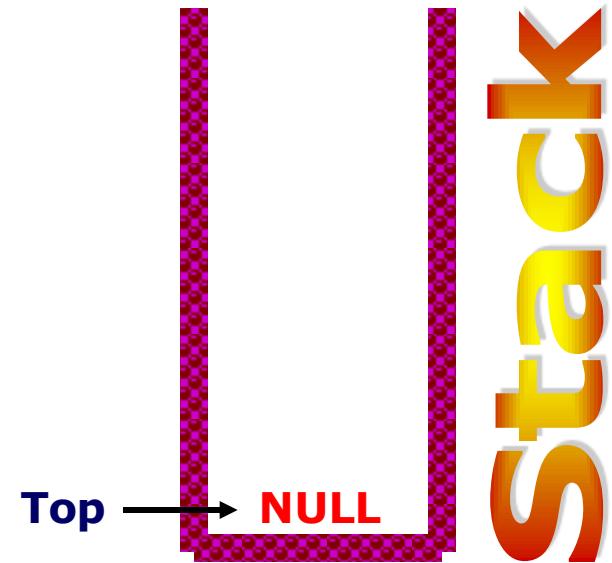
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



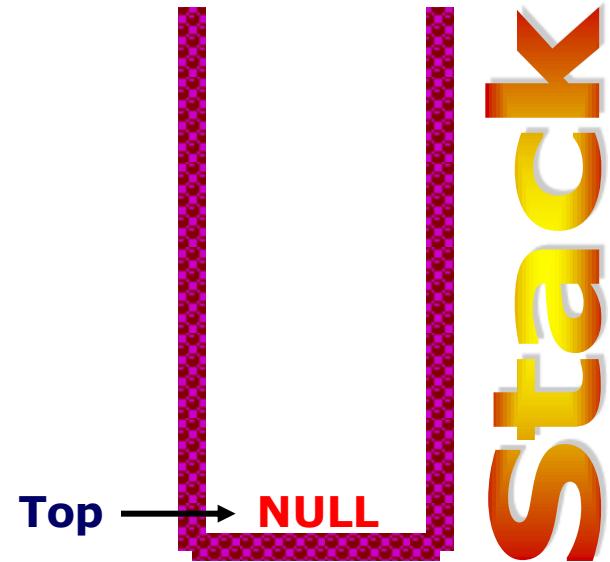
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;                sodu = 1
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



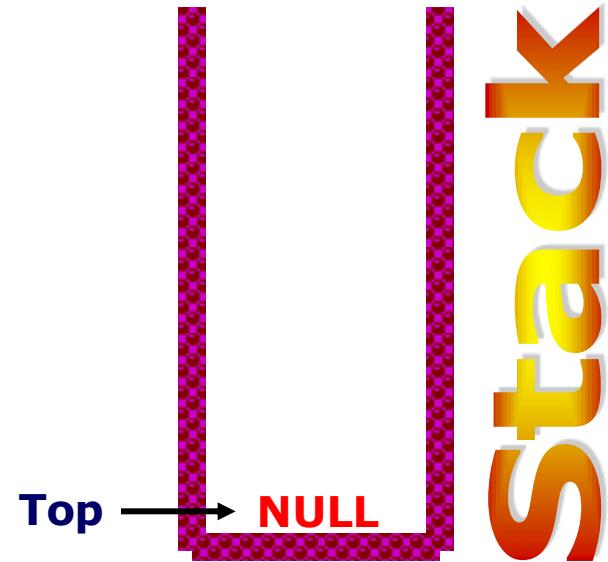
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;                sodu = 1
    n /= a;                   n = 12
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



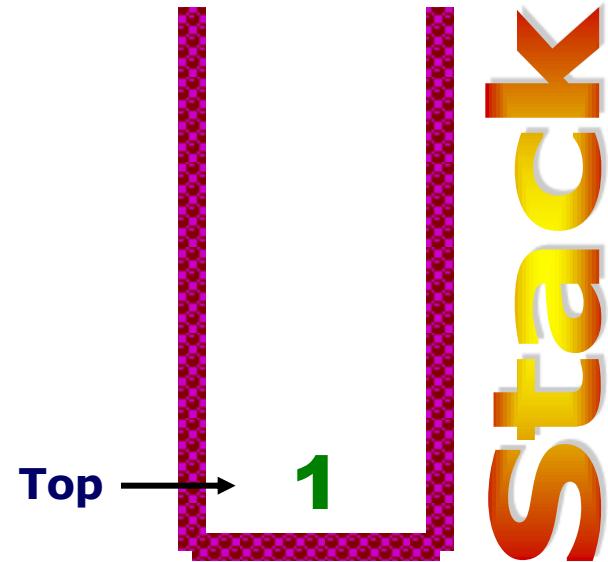
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 1
    n /= a;              n = 12
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



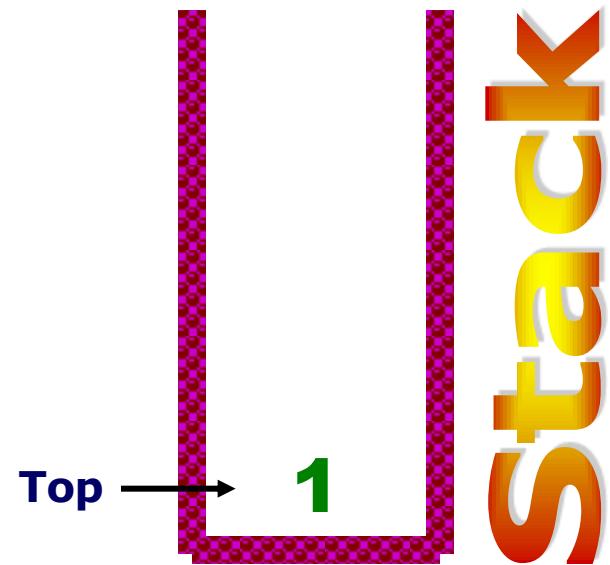
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 1
    n /= a;              n = 12
    push(stack, sodu);  push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



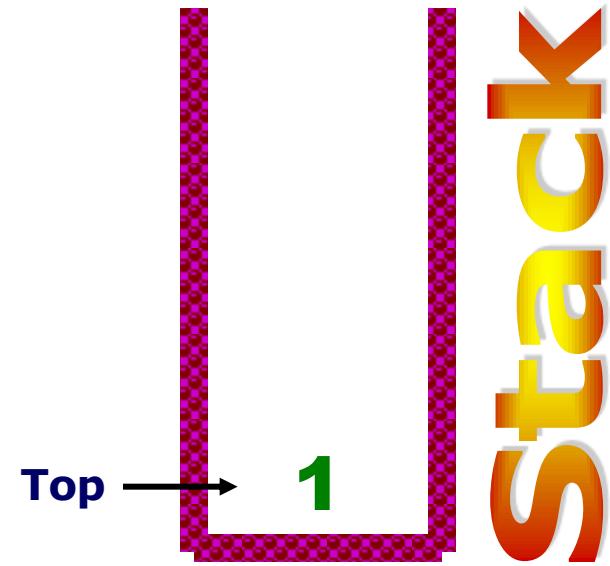
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 1
    n /= a;              n = 12
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



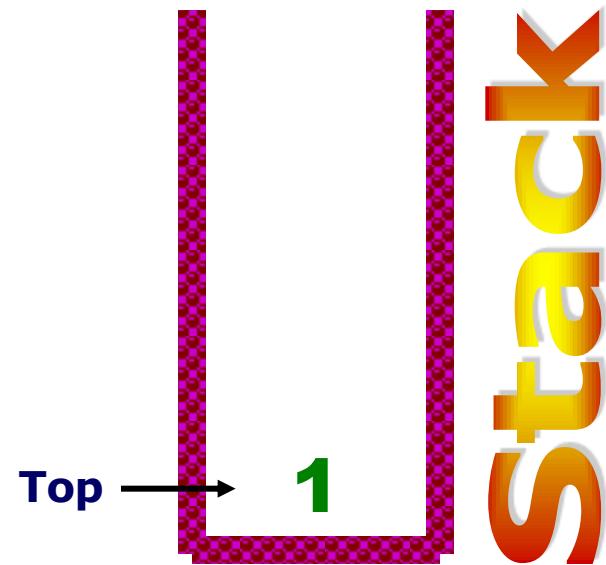
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 1
    n /= a;              n = 12
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



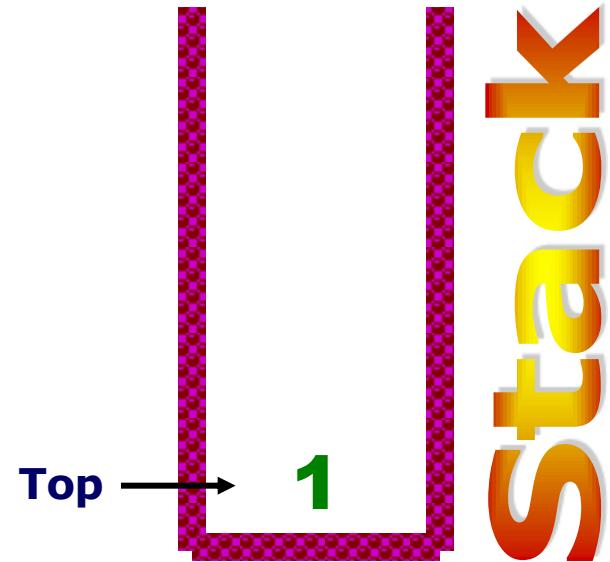
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;           sodu = 0
    n /= a;              n = 12
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



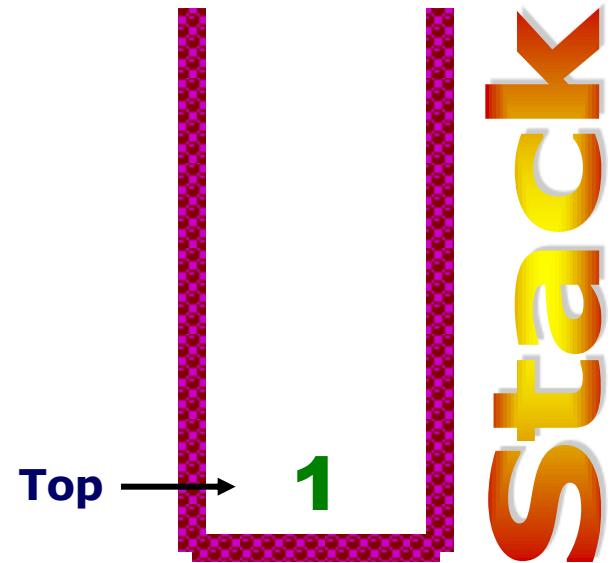
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;           sodu = 0
    n /= a;               n = 6
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



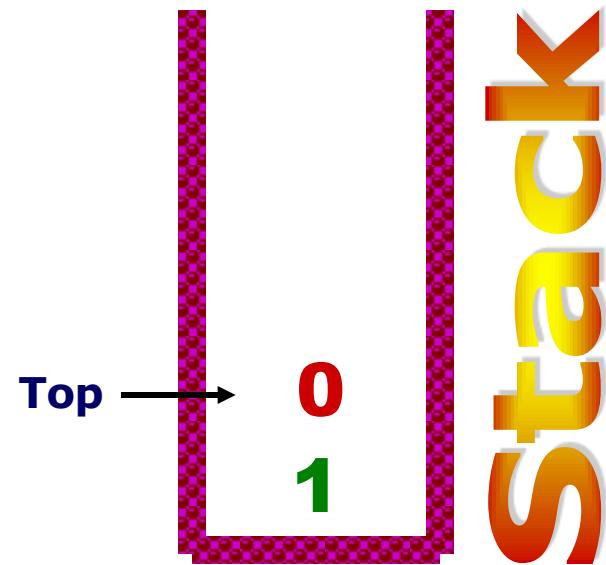
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;           sodu = 0
    n /= a;               n = 6
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



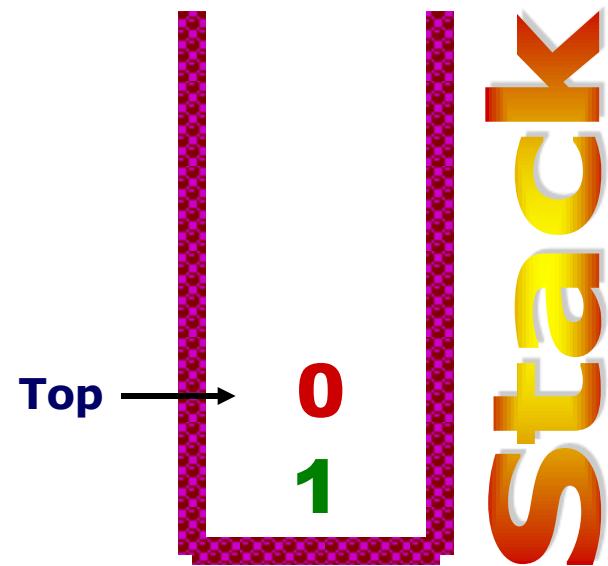
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 0
    n /= a;              n = 6
    push(stack, sodu);  ←
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



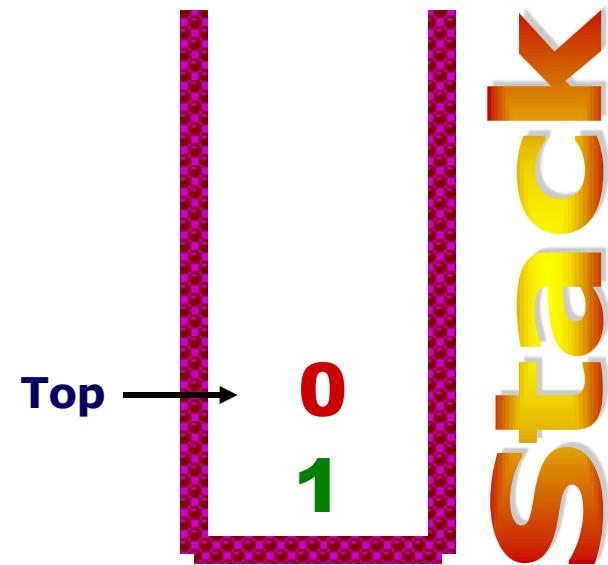
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 0
    n /= a;              n = 6
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



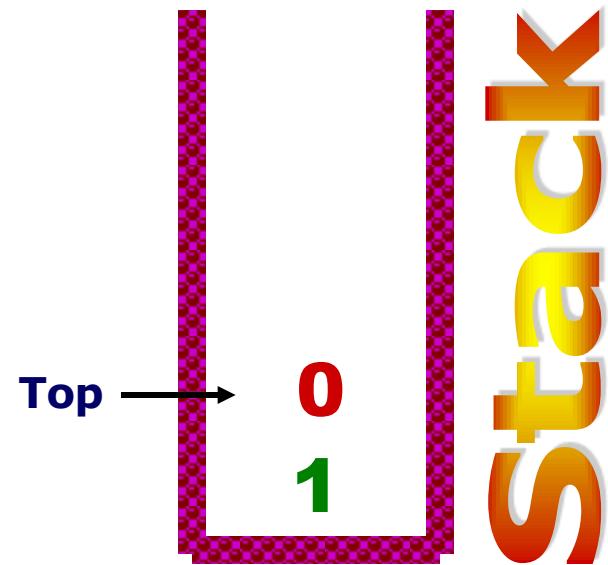
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 0
    n /= a;              n = 6
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



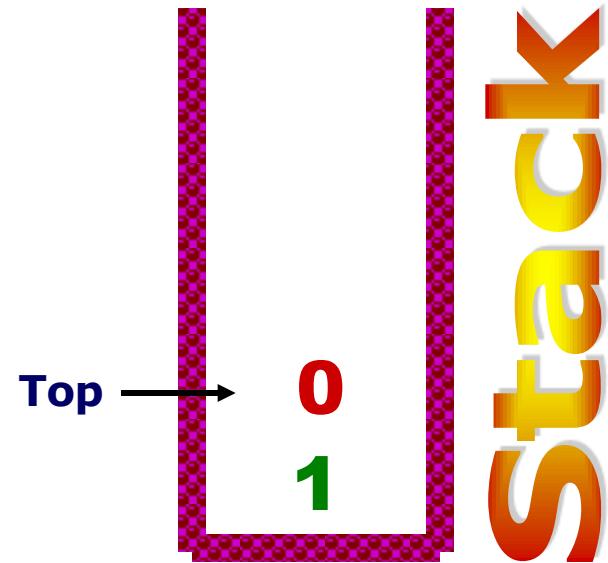
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 0
    n /= a;              n = 6
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



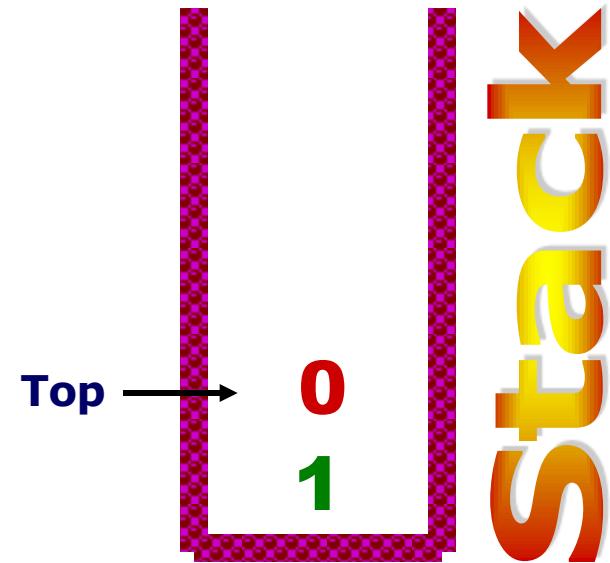
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;           sodu = 0
    n /= a;               n = 3
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



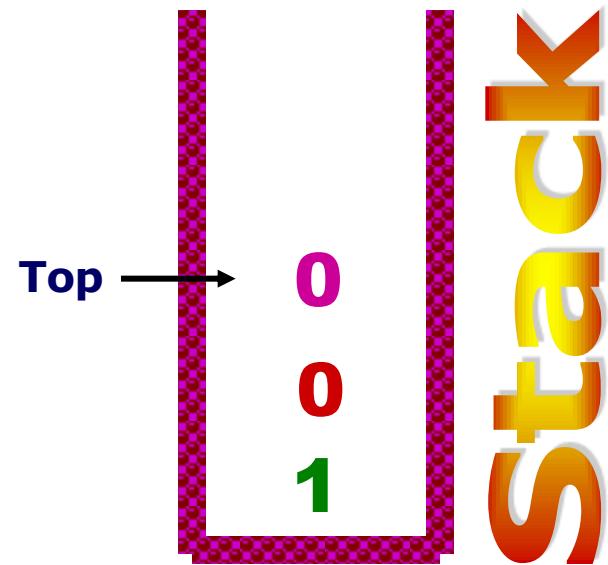
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 1
    n /= a;              n = 3
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



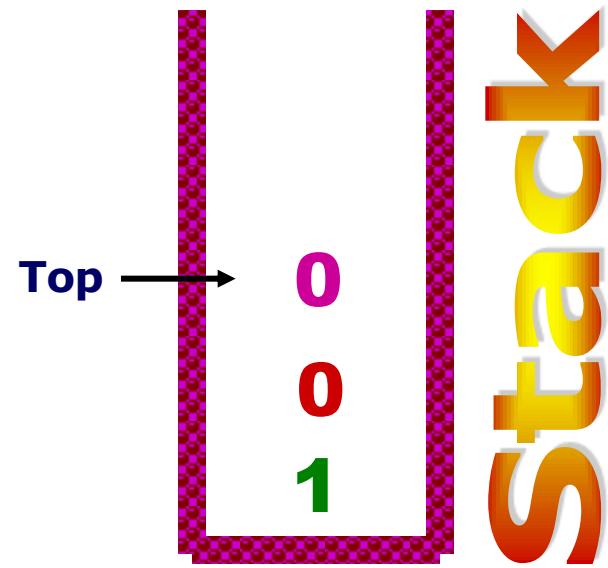
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 0
    n /= a;              n = 3
    push(stack, sodu);  ←
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



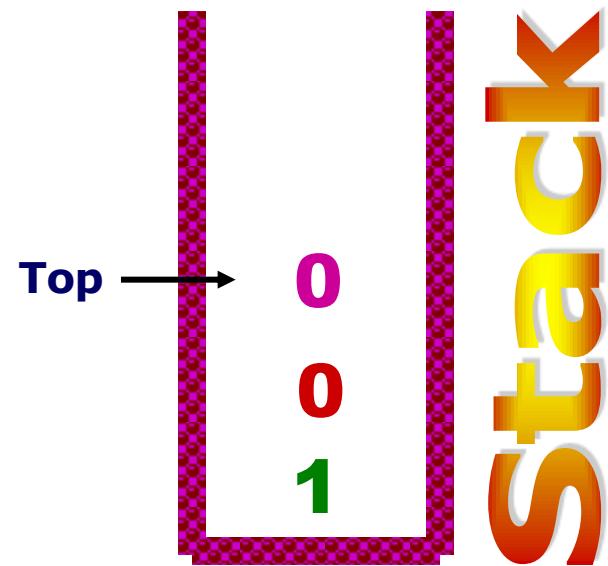
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 0
    n /= a;              n = 3
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



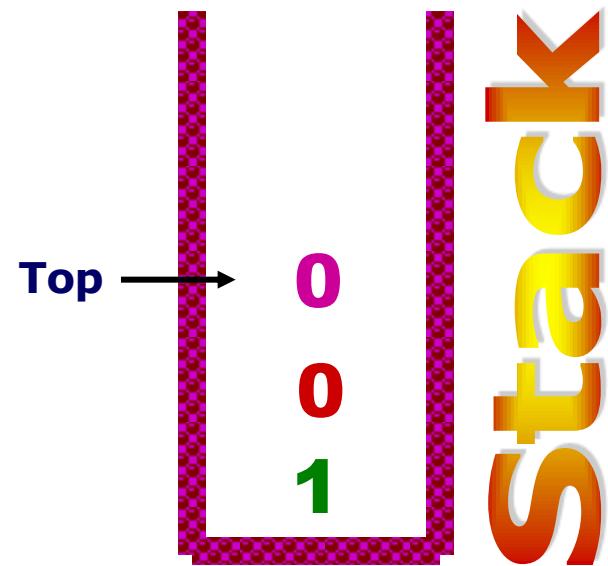
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 0
    n /= a;              n = 3
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



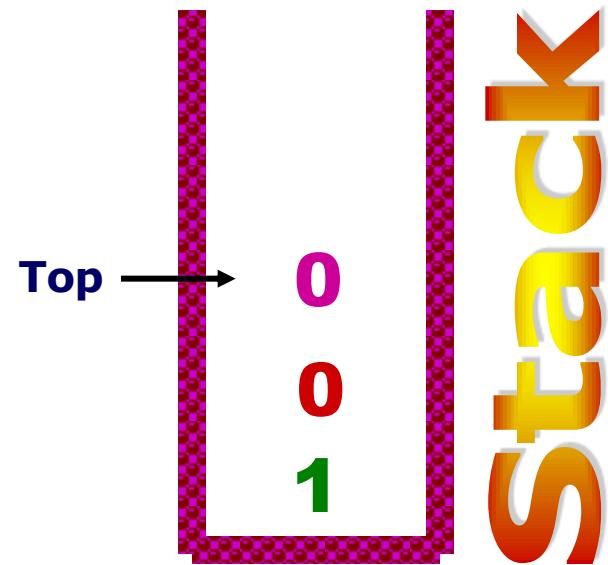
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;           sodu = 1
    n /= a;              n = 3
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



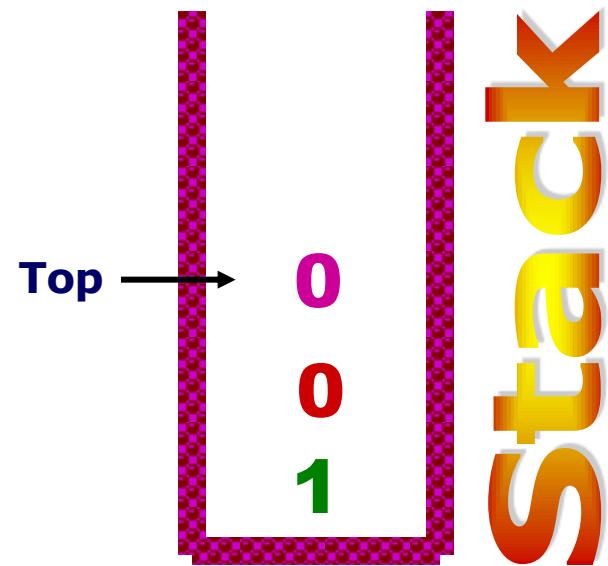
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;           sodu = 1
    n /= a;               n = 1
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



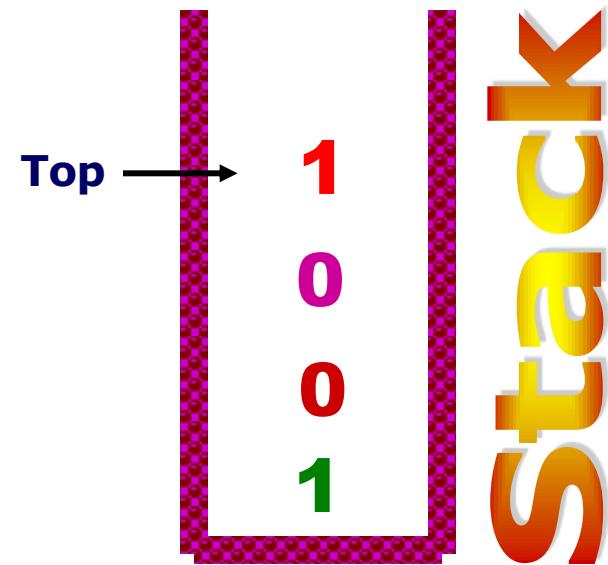
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 1
    n /= a;              n = 1
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



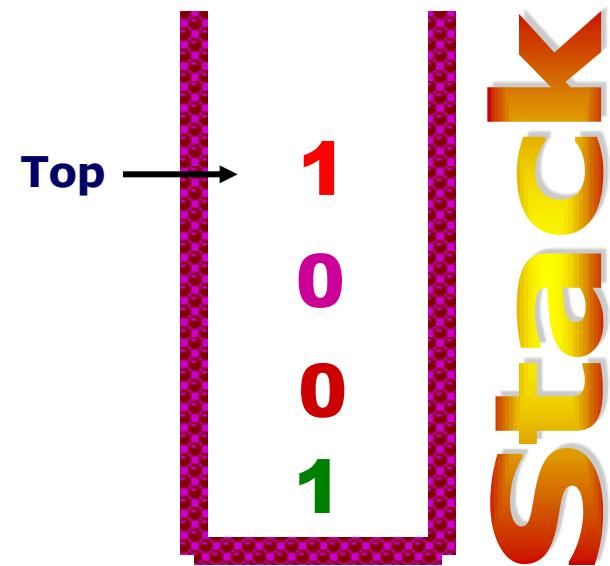
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 1
    n /= a;              n = 1
    push(stack, sodu);  ←
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



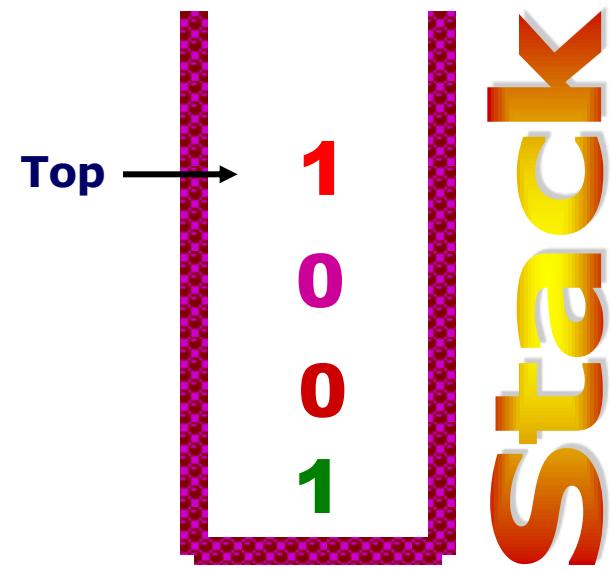
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 1
    n /= a;              n = 1
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



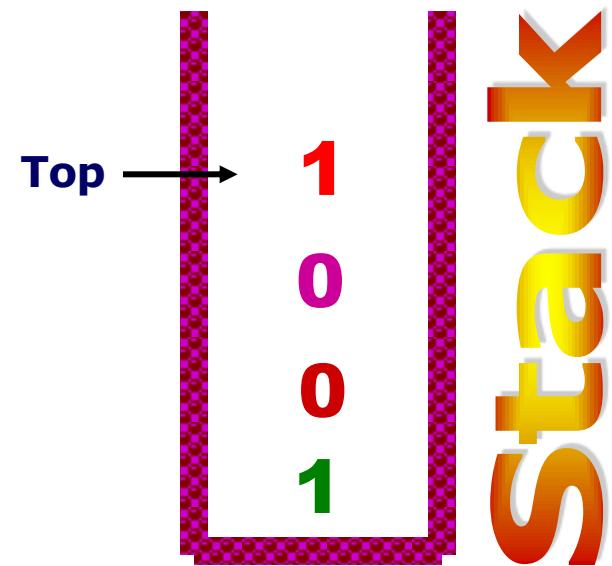
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 1
    n /= a;              n = 1
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



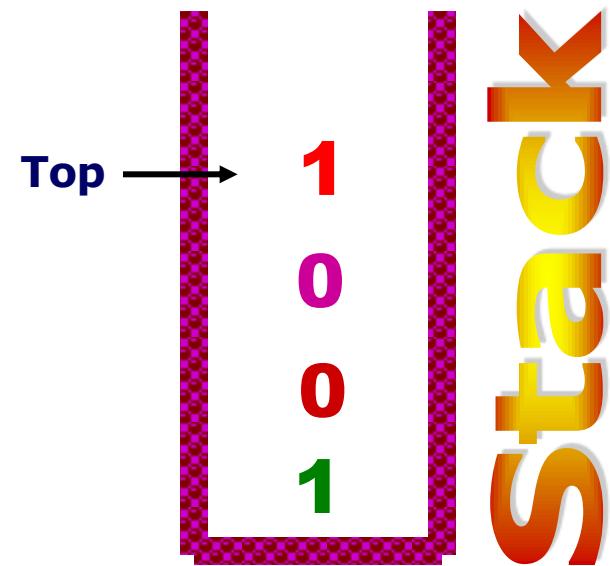
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;           sodu = 1
    n /= a;              n = 1
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



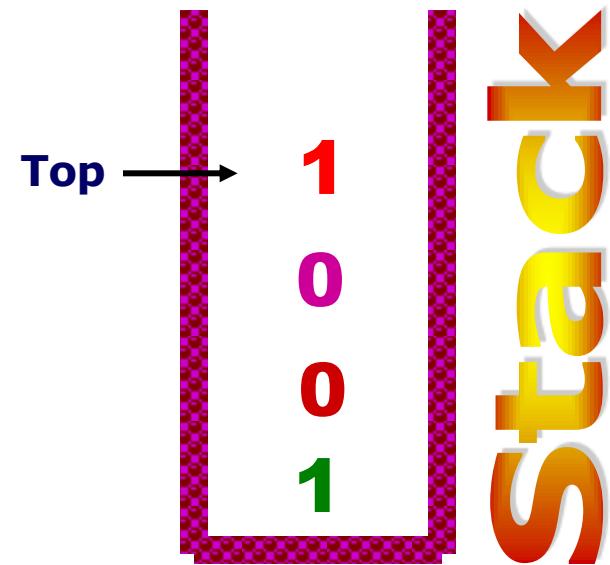
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;           sodu = 1
    n /= a;               n = 0
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



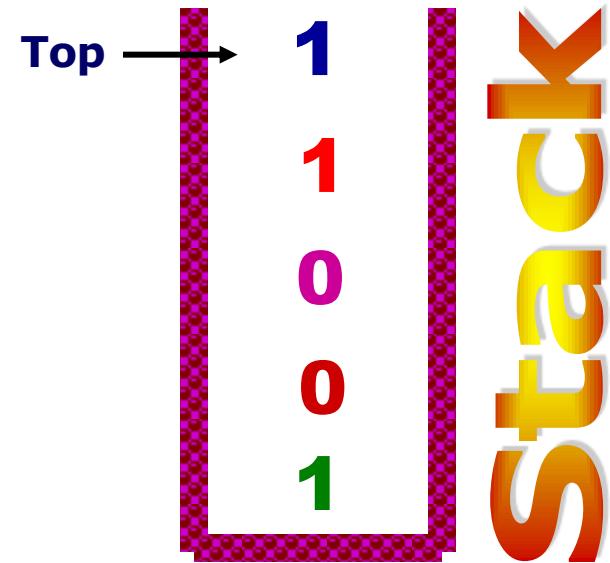
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 1
    n /= a;              n = 0
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



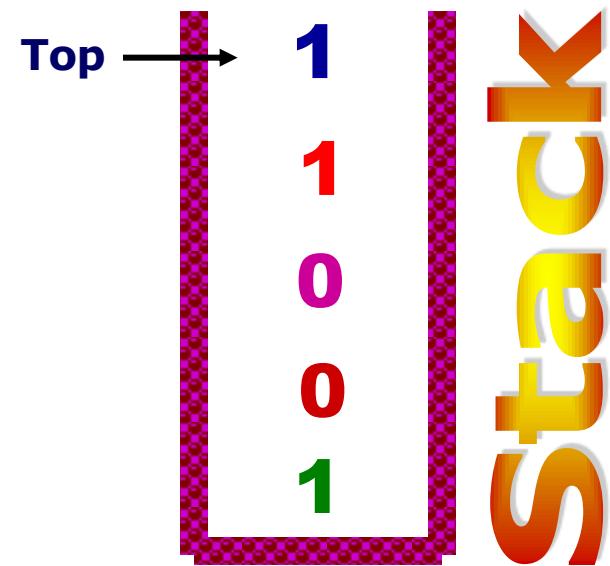
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 1
    n /= a;              n = 0
    push(stack, sodu);  ←
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



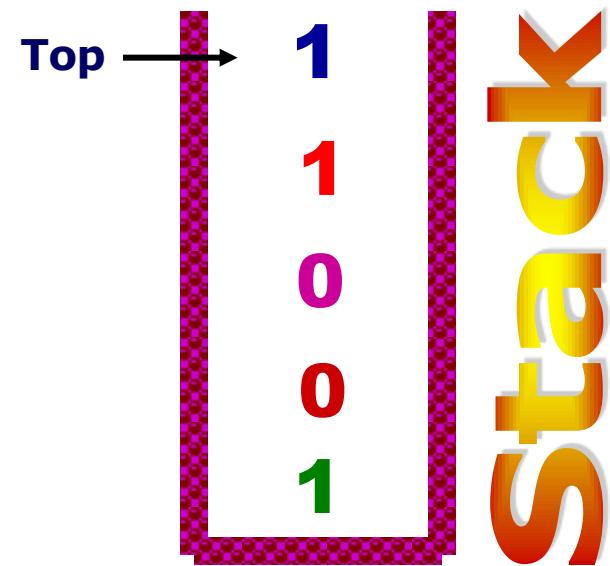
ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 1
    n /= a;              n = 0
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

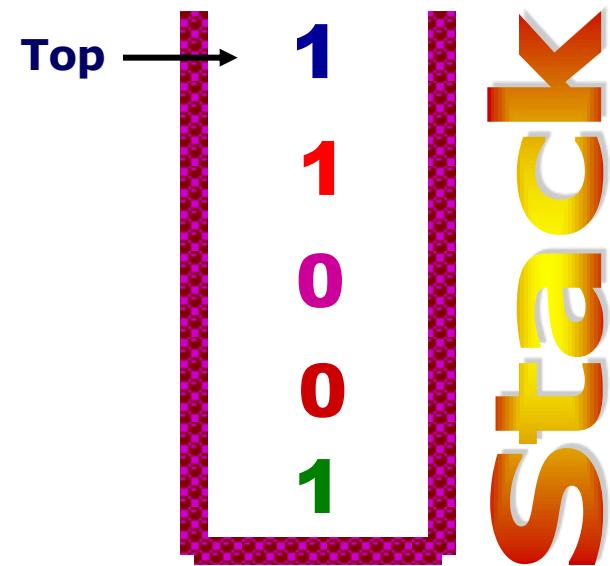
```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;          sodu = 1
    n /= a;              n = 0
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```

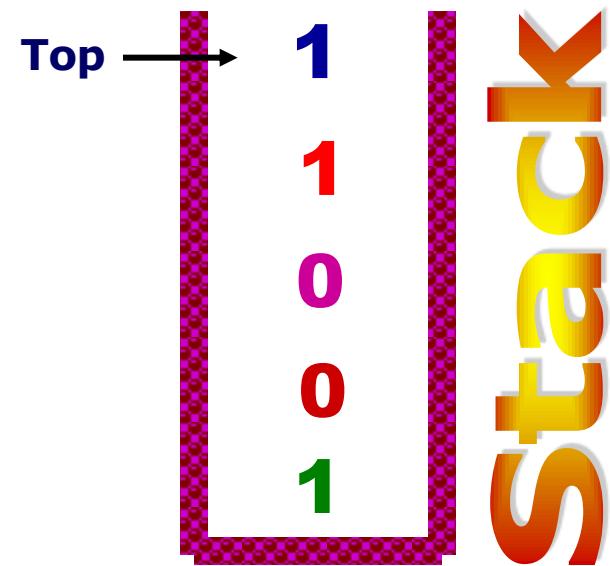
Kết quả $25_{10} =$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```

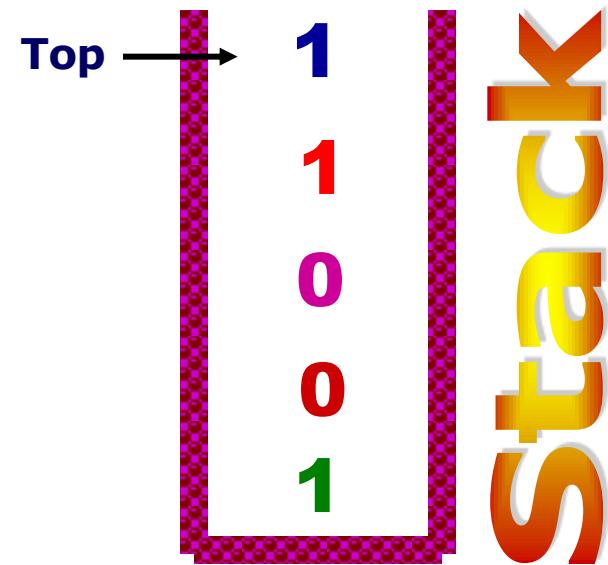
Kết quả $25_{10} =$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```

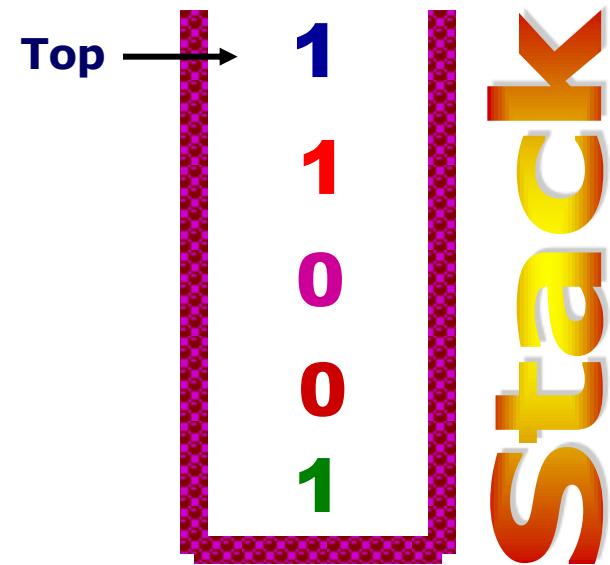
Kết quả $25_{10} =$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 1
    printf("%3d", x);
}
}
```

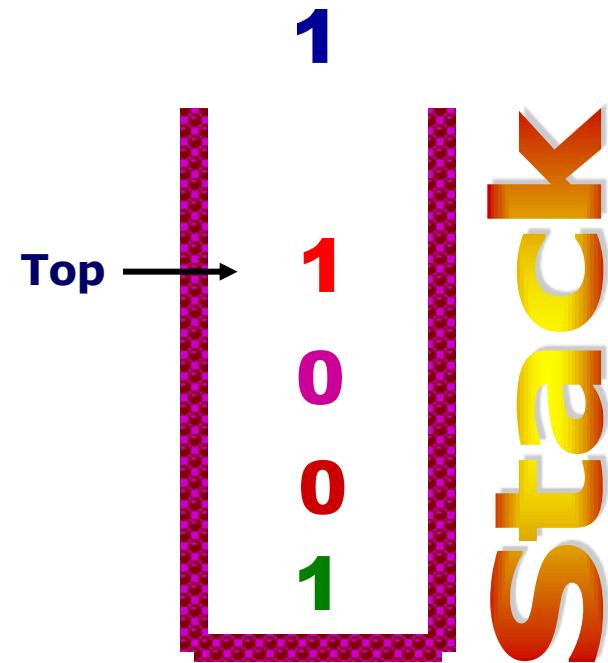
Kết quả $25_{10} =$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 1
    printf("%3d", x);
}
}
```

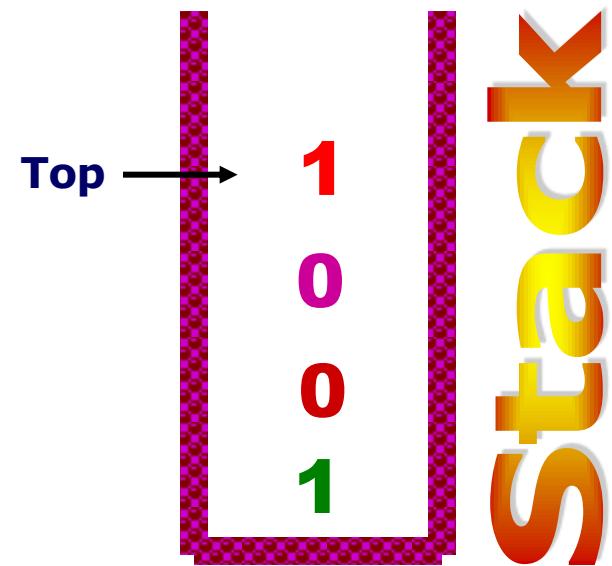
Kết quả $25_{10} =$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);          x = 1
    printf("%3d", x);
}
}
```

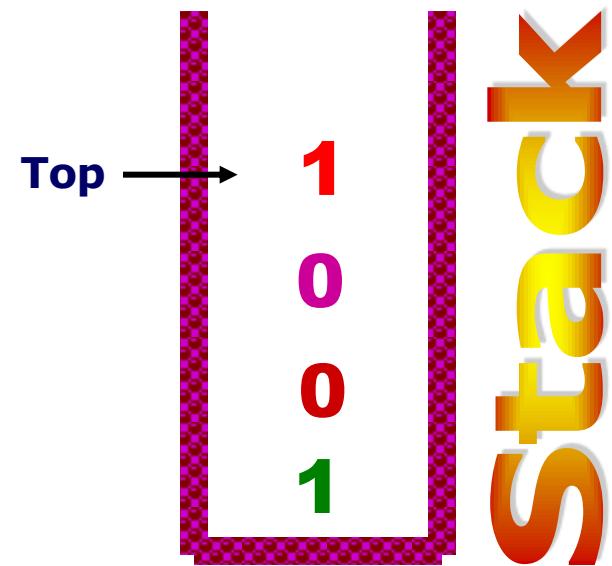
Kết quả $25_{10} = 1$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 1
    printf("%3d", x);
}
}
```

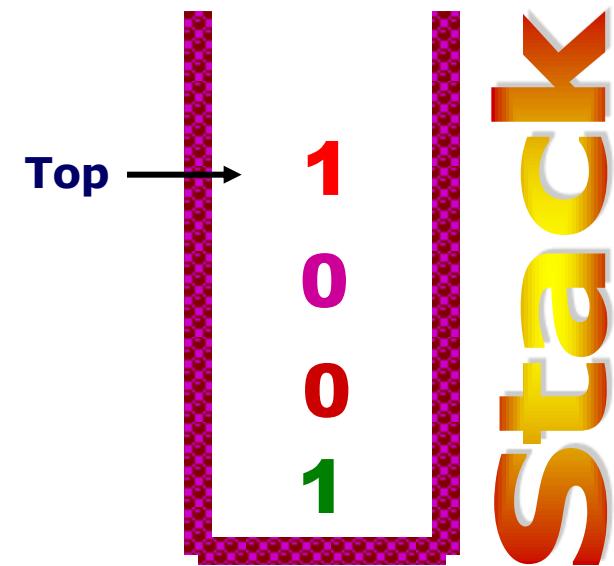
Kết quả $25_{10} = 1$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 1
    printf("%3d", x);
}
}
```

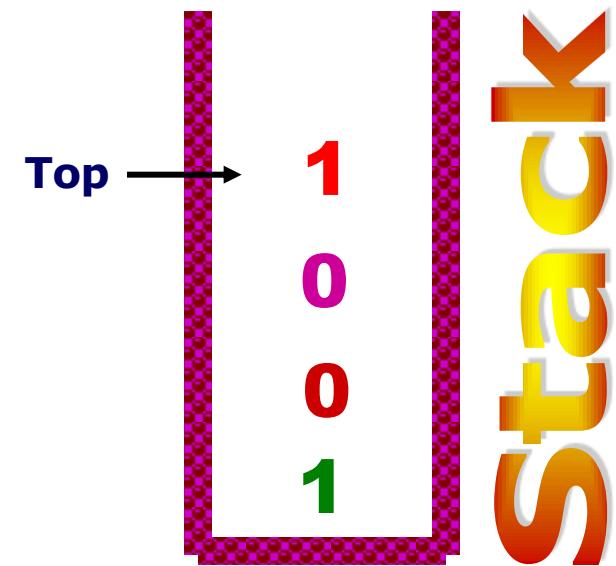
Kết quả $25_{10} = 1$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 1
    printf("%3d", x);
}
}
```

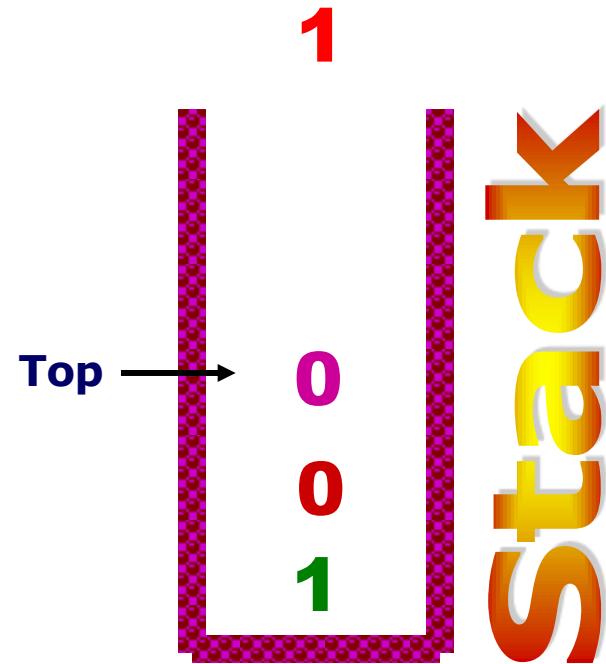
Kết quả $25_{10} = 1$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 1
    printf("%3d", x);
}
}
```

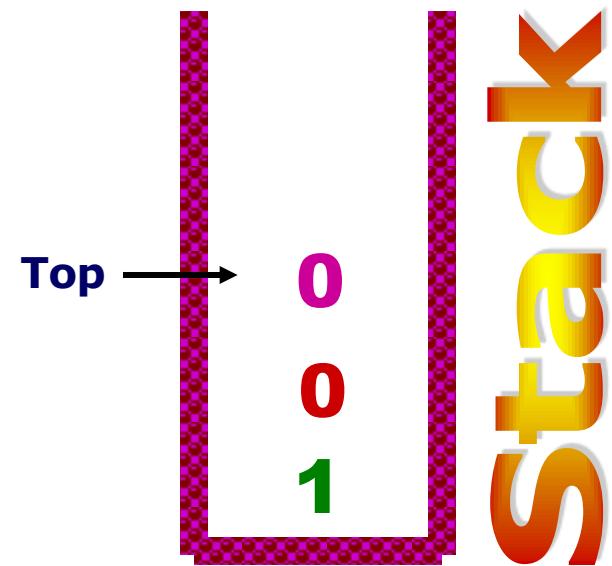
Kết quả $25_{10} = 1$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);          x = 1
    printf("%3d", x);
}
}
```

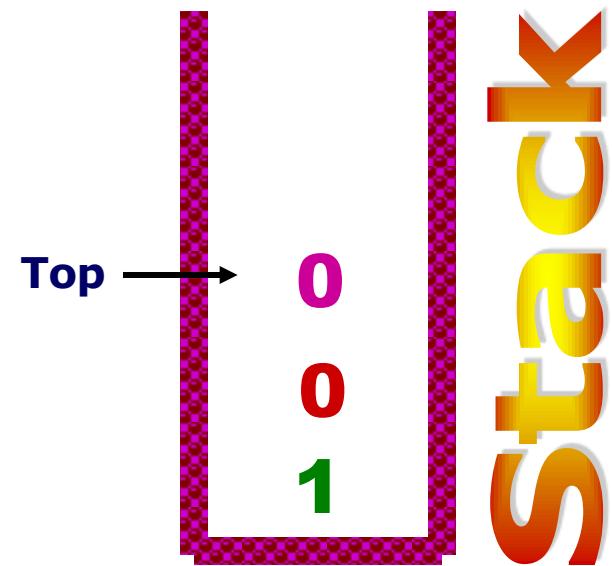
Kết quả $25_{10} = 1\ 1$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 1
    printf("%3d", x);
}
}
```

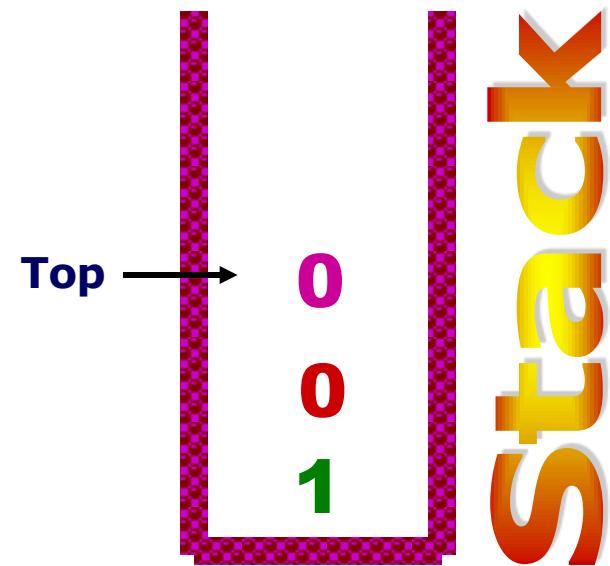
Kết quả $25_{10} = 1\ 1$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 1
    printf("%3d", x);
}
```

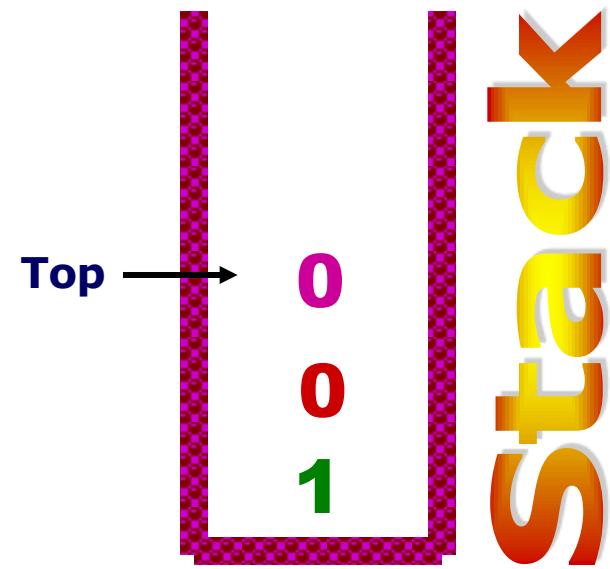
Kết quả $25_{10} = 1\ 1$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 0
    printf("%3d", x);
}
```

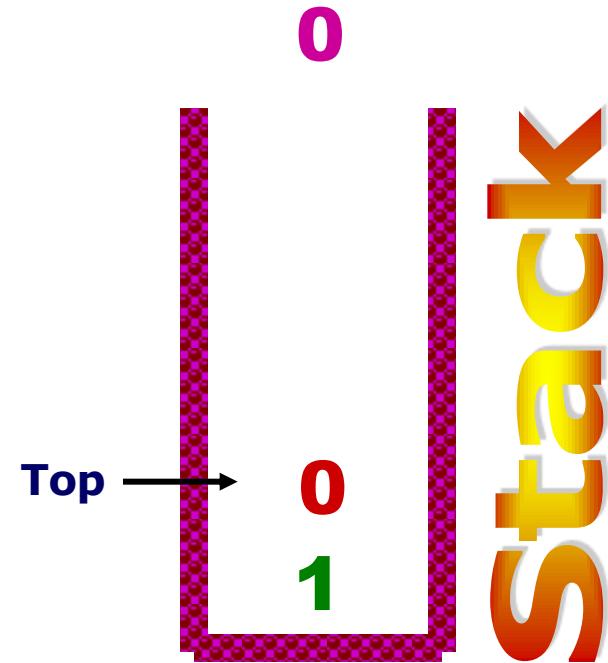
Kết quả $25_{10} = 1 \ 1$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 0
    printf("%3d", x);
}
}
```

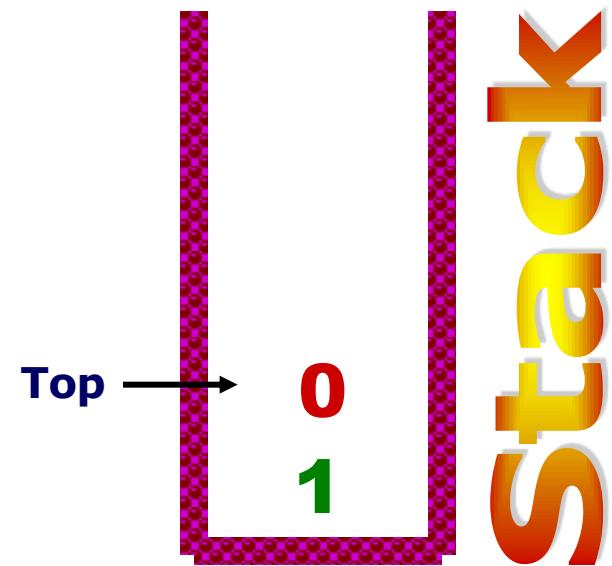
Kết quả $25_{10} = 1 \ 1$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);          x = 0
    printf("%3d", x);
}
}
```

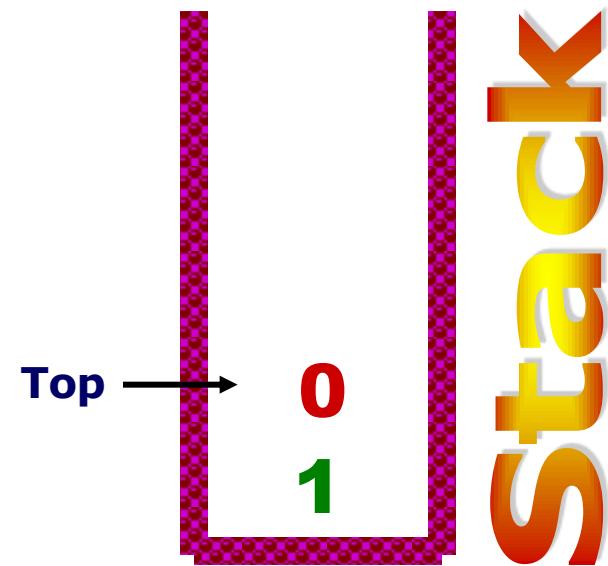
Kết quả $25_{10} = 1\ 1\ 0$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);          x = 0
    printf("%3d", x);
}
}
```

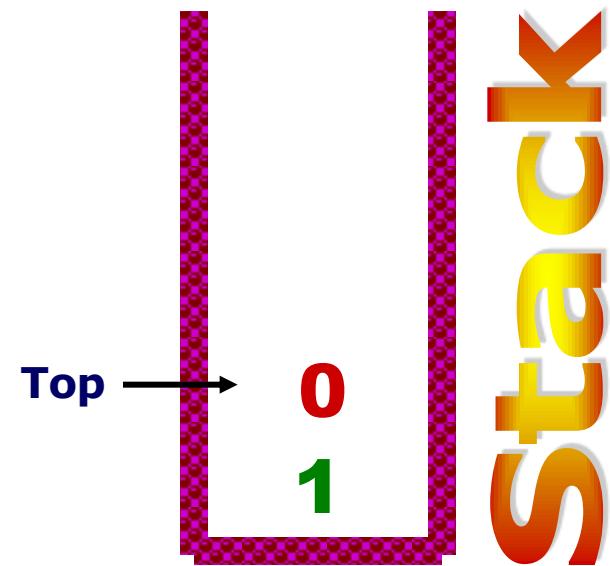
Kết quả $25_{10} = 1\ 1\ 0$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 0
    printf("%3d", x);
}
```

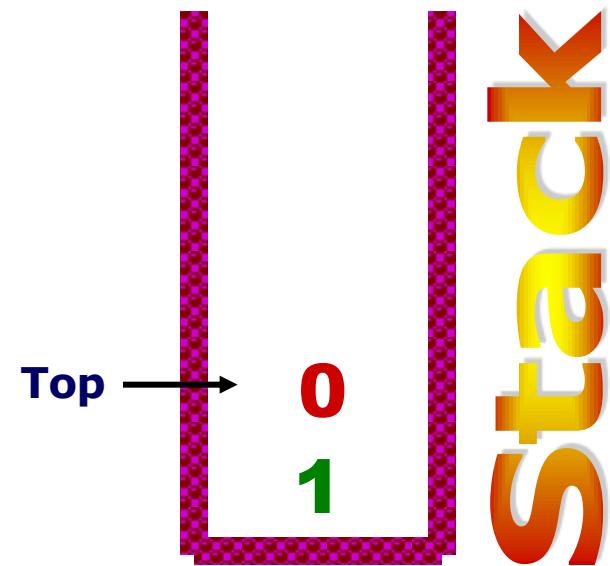
Kết quả $25_{10} = 1\ 1\ 0$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 0
    printf("%3d", x);
}
}
```

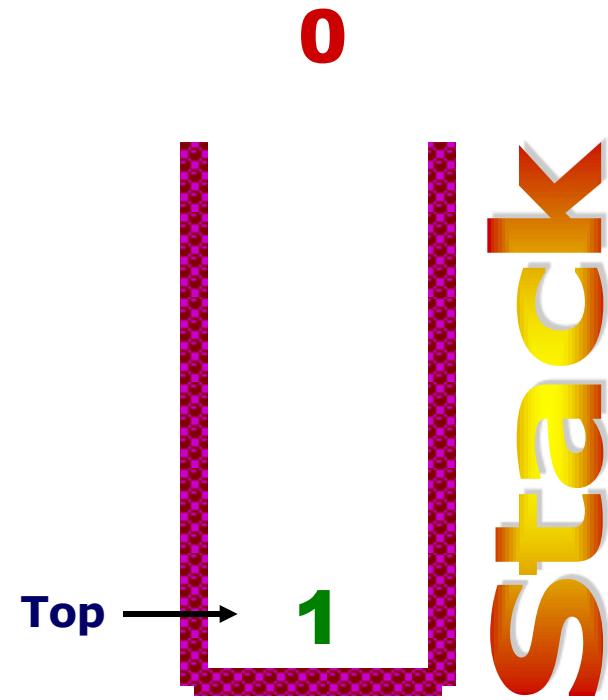
Kết quả $25_{10} = 1\ 1\ 0$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 0
    printf("%3d", x);
}
}
```

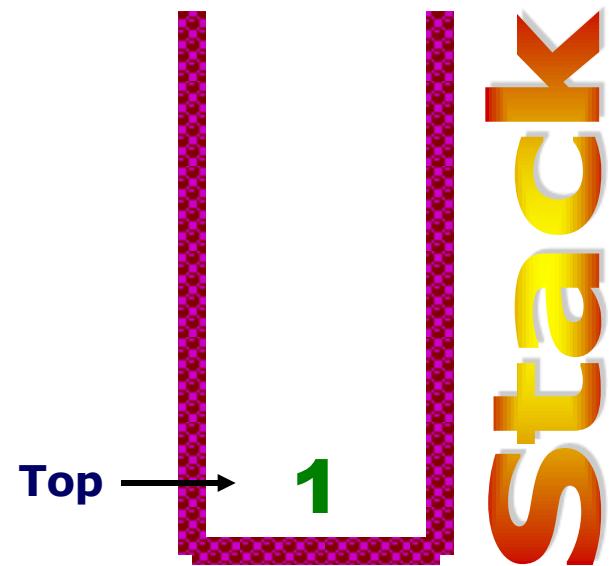
Kết quả $25_{10} = 1\ 1\ 0$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);          x = 0
    printf("%3d", x);
}
}
```

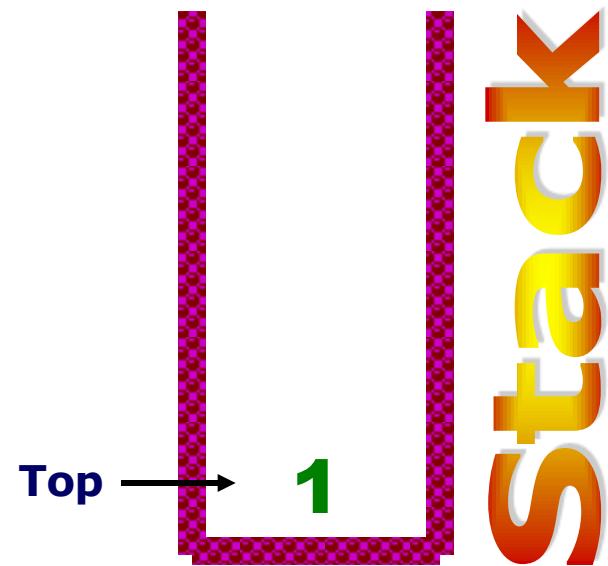
Kết quả $25_{10} = 1\ 1\ 0\ 0$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 0
    printf("%3d", x);
}
}
```

Kết quả $25_{10} = 1\ 1\ 0\ 0$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
```

```
int sodu, x, cn = n;
```

```
Stack stack;
```

```
initStack(stack);
```

```
while(n != 0)
```

```
{
```

```
    sodu = n%a;
```

```
    n /= a;
```

```
    push(stack, sodu);
```

```
}
```

```
printf("Kết quả %ld = ", cn);
```

```
while(isEmpty(stack) == 0)
```

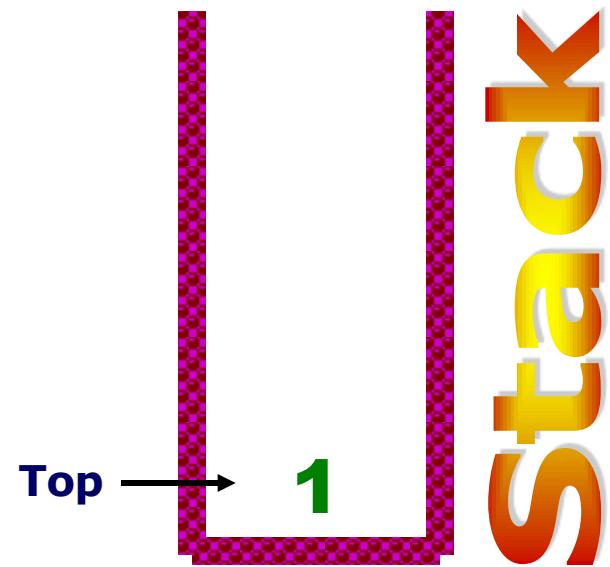
```
{
```

```
    pop(stack, x);      x = 0
```

```
    printf("%3d", x);
```

```
}
```

Kết quả $25_{10} = 1\ 1\ 0\ 0$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
```

```
int sodu, x, cn = n;
```

```
Stack stack;
```

```
initStack(stack);
```

```
while(n != 0)
```

```
{
```

```
    sodu = n%a;
```

```
    n /= a;
```

```
    push(stack, sodu);
```

```
}
```

```
printf("Kết quả %ld = ", cn);
```

```
while(isEmpty(stack) == 0)
```

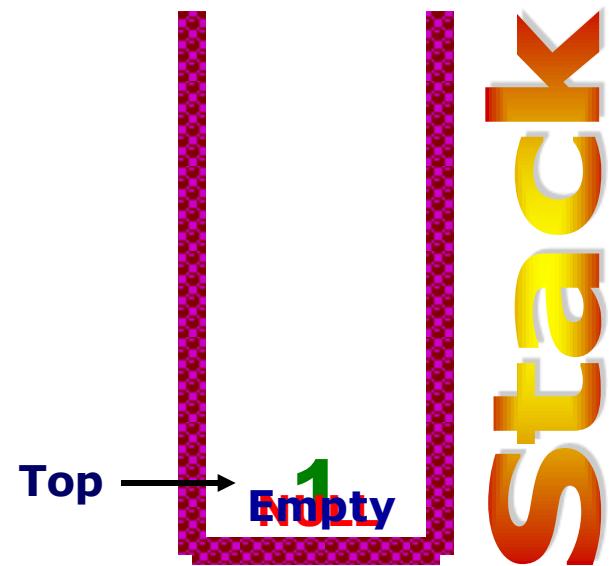
```
{
```

```
    pop(stack, x);      x = 1
```

```
    printf("%3d", x);
```

```
}
```

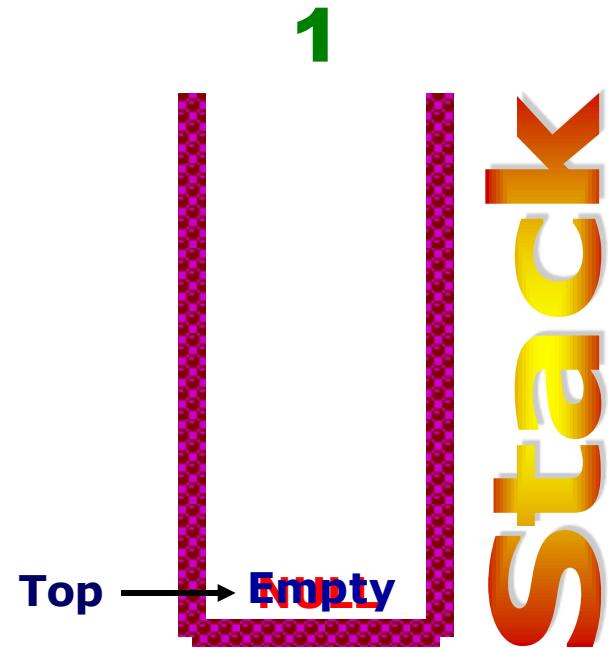
Kết quả $25_{10} = 1\ 1\ 0\ 0$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 1
    printf("%3d", x);
}
}
```

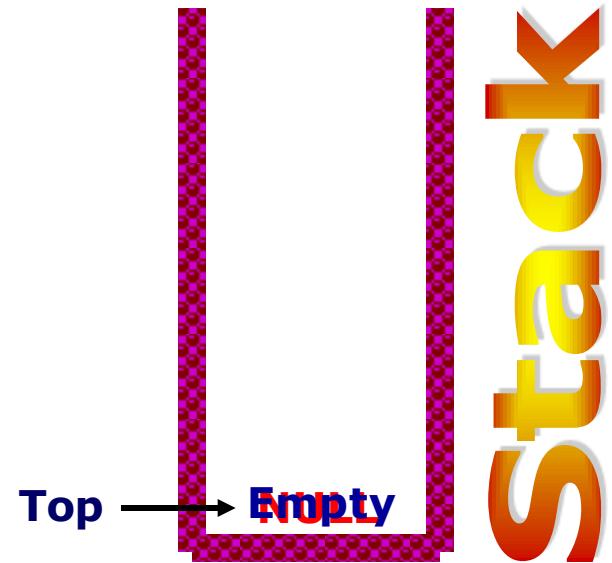
Kết quả $25_{10} = 1\ 1\ 0\ 0$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);          x = 1
    printf("%3d", x);
}
}
```

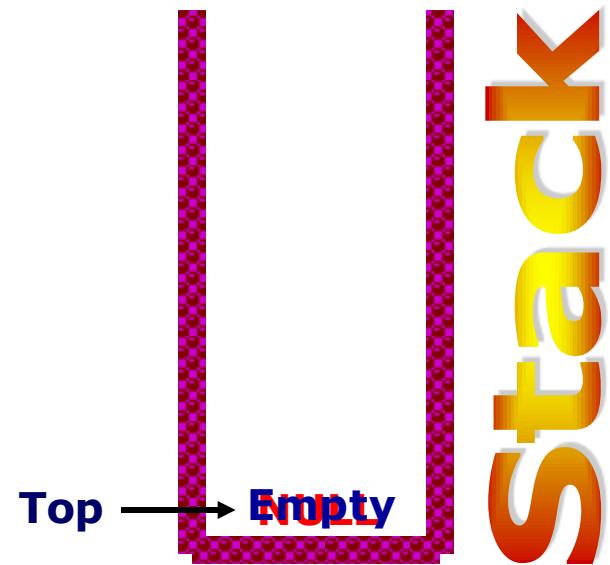
Kết quả $25_{10} = 1\ 1\ 0\ 0\ 1$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 1
    printf("%3d", x);
}
}
```

Kết quả $25_{10} = 1\ 1\ 0\ 0\ 1$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
```

```
int sodu, x, cn = n;
```

```
Stack stack;
```

```
initStack(stack);
```

```
while(n != 0)
```

```
{
```

```
    sodu = n%a;
```

```
    n /= a;
```

```
    push(stack, sodu);
```

```
}
```

```
printf("Kết quả %ld = ", cn);
```

```
while(isEmpty(stack) == 0)
```

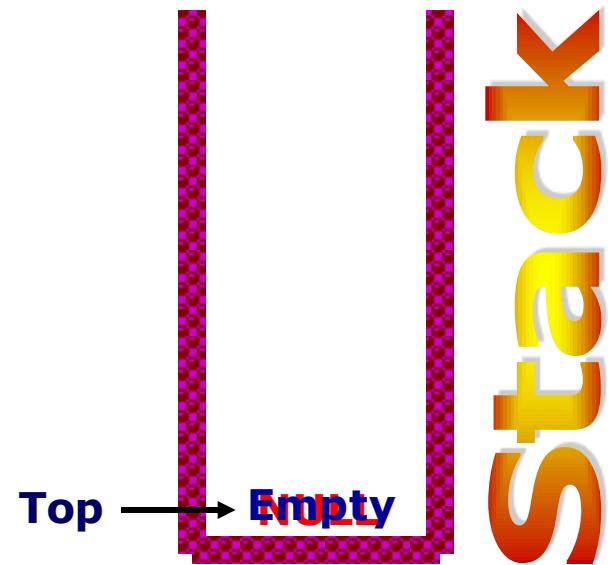
```
{
```

```
    pop(stack, x);      x = 1
```

```
    printf("%3d", x);
```

```
}
```

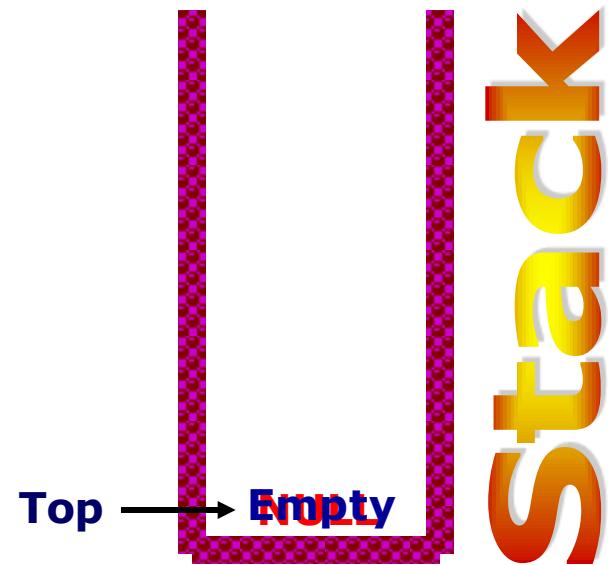
Kết quả $25_{10} = 1\ 1\ 0\ 0\ 1$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);      x = 1
    printf("%3d", x);
}
}
```

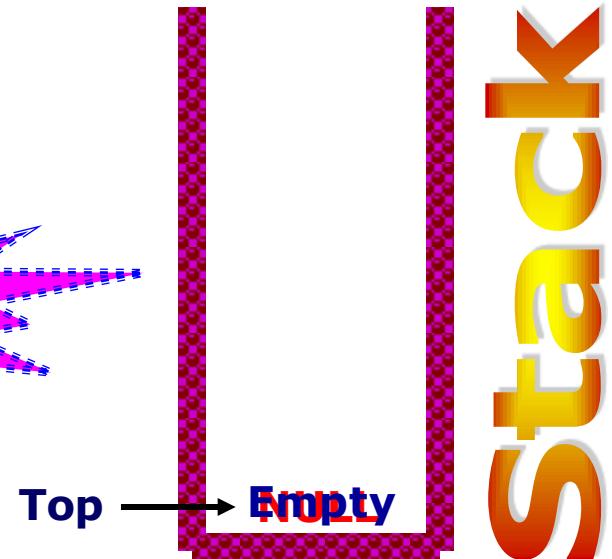
Kết quả $25_{10} = 1\ 1\ 0\ 0\ 1$



ỨNG DỤNG STACK ĐỂ ĐỔI CƠ SỐ

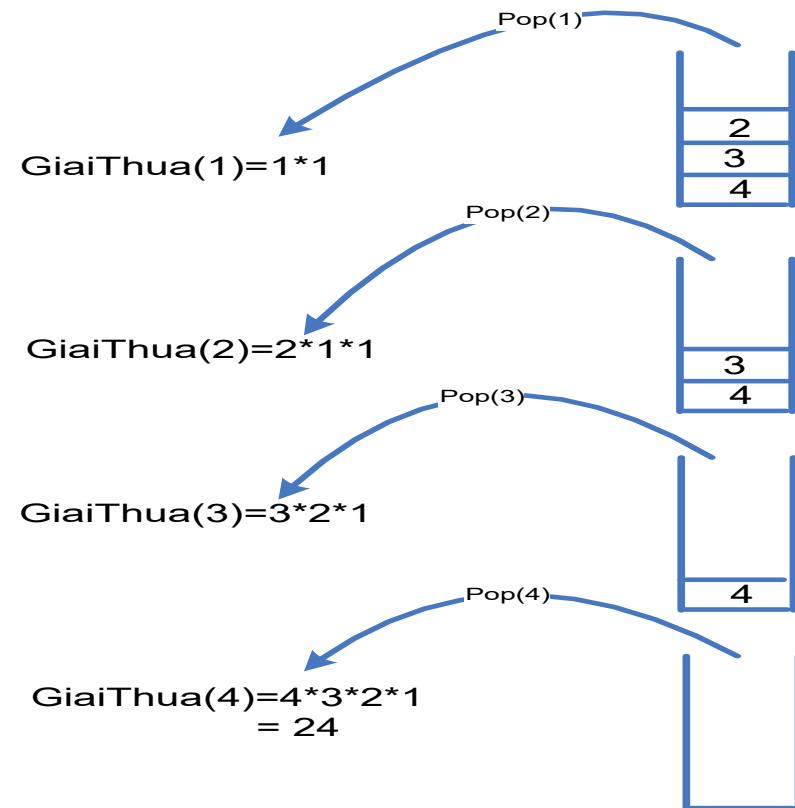
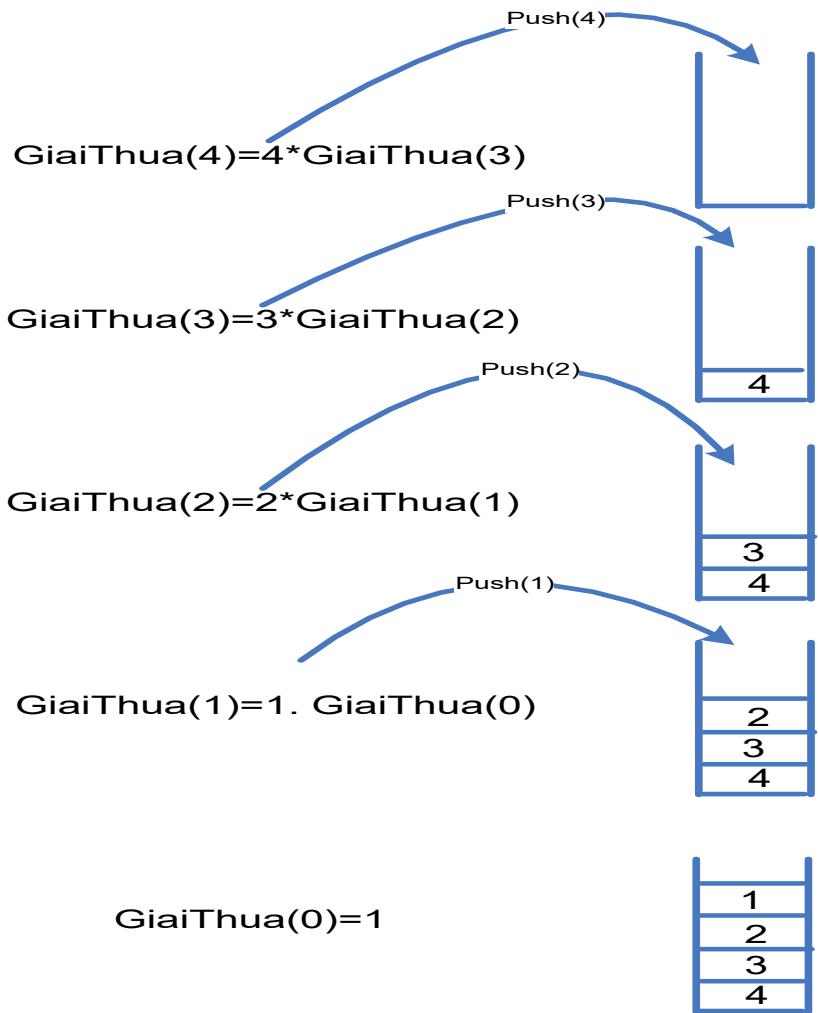
```
void doiCoSo(int n, int a)    Giả sử n = 25, a = 2
{//Đổi từ hệ 10 sang một hệ số a bất kỳ.
int sodu, x, cn = n;
Stack stack;
initStack(stack);
while(n != 0)
{
    sodu = n%a;
    n /= a;
    push(stack, sodu);
}
printf("Kết quả %ld = ", cn);
while(isEmpty(stack) == 0)
{
    pop(stack, x);
    printf("%3d", x);
}
}
```

Kết quả $25_{10} = 1\ 1\ 0\ 0\ 1$



BÀI TẬP 1: TÍNH GIAI THỪA

- Dùng ngăn xếp khử đệ quy bài toán tính giai thừa (minh họa)



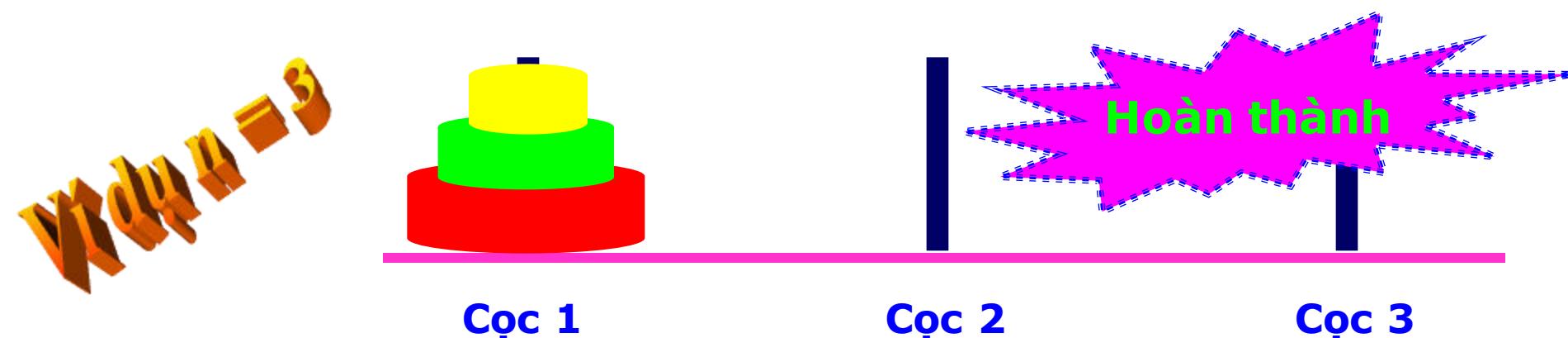
BÀI TẬP 1: TÍNH GIAI THỪA

```
long int tinhGiaiThua(int n) {  
    DataTypeItem x;  
    Stack stack;  
    initStack(stack);  
    while(n > 0)  
        {//Thêm Lần Lượt nội dung vào stack  
        StackNode* p = createStackNode(n);  
        if(push(stack, p) == 0) break;  
        n--;  
    }  
    long int kq = 1;  
    while(1)  
        {//Lấy Lần Lượt nội dung từ stack  
        if(pop(stack, x) == 0) break;  
        kq *= x;  
    }  
    return kq;  
}
```

BÀI TẬP 2: THÁP HÀ NỘI

❖ Tháp Hà Nội:

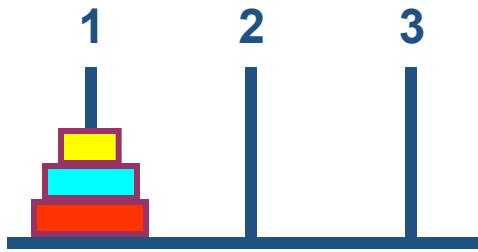
- Có **n** đĩa đặt chồng lên nhau tại **cọc 1** sao cho đĩa lớn nằm dưới và đĩa nhỏ nằm trên.
- Chuyển **n** đĩa này sang **cọc 3**, cho phép lấy **cọc 2** làm trung gian, mỗi lần chỉ cho phép di chuyển 1 đĩa và vẫn giữ nguyên tắc đĩa lớn nằm dưới và đĩa nhỏ nằm trên.



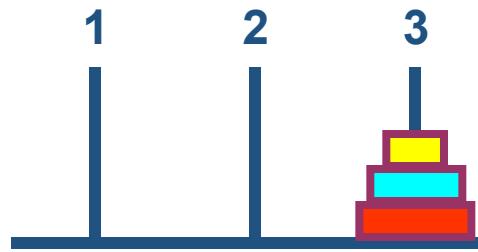
BÀI TẬP 2: THÁP HÀ NỘI

❖ Tháp HaNoi

Move(số đĩa, cọc nguồn, cọc đích, cọc tạm)



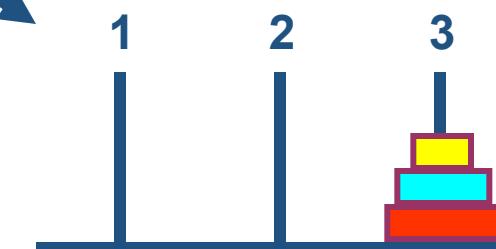
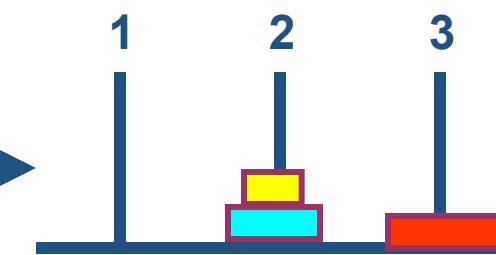
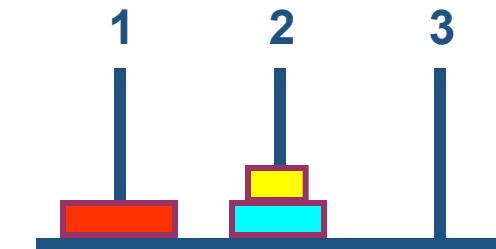
Kết quả



Move(n-1, 1, 2, 3);

Move(1, 1, 3, 2);

Move(n-1, 2, 3, 1);



BÀI TẬP 2: THÁP HÀ NỘI

❖ Stack khử đệ quy:

- $\{n, src, des\}$
- Các cọc đánh số 1, 2, 3
- Cọc temp = $6 - (src + des)$

1. Push \Rightarrow Stack: {n, 1, 3} • •

Stack lưu
trữ thứ tự
ngược

BÀI TẬP 2: THÁP HÀ NỘI

2. Pop: Stack $\Rightarrow \{n, \text{src}, \text{des}\}$

- Nếu $n = 1$ thì: move src \Rightarrow des
- Ngược lại:
 - $\text{temp} = 6 - (\text{src} + \text{des})$
 - Push \Rightarrow Stack: $\{n - 1, \text{temp}, \text{des}\}$
 - Push \Rightarrow Stack: $\{1, \text{src}, \text{des}\}$
 - Push \Rightarrow Stack: $\{n - 1, \text{src}, \text{temp}\}$

BÀI TẬP 3: THUẬT TOÁN QUICKSORT

Ví dụ: Giải thuật QuickSort dùng Stack để khử đệ qui:

- ❖ **Bước 1:** Left=0; Right=N-1;
- ❖ **Bước 2:** Chọn phần tử giữa $x=a[(Left+Right) / 2]$
- ❖ **Bước 3:** Phân hoạch (Left, Right) thành (Left1, Right1) và (Left2, Right2) bằng cách xét:
 - y thuộc (Left1, Right1) nếu $y \leq x$
 - y thuộc (Left2, Right2) ngược lại
- ❖ **Bước 4:** Nếu phân hoạch (Left2, Right2) có nhiều hơn 1 phần tử thì thực hiện:
 - Cất (Left2, Right2) vào Stack
 - Nếu (Left1, Right1) có nhiều hơn 1 phần tử thì thực hiện:
 - Left = Left1
 - Right = Right1
 - Quay lên bước 2
 - Ngược lại
 - Lấy (Left, Right) ra khỏi Stack, nếu Stack khác rỗng thì quay lên bước 2, ngược lại thì dừng

BÀI TẬP 3: THUẬT TOÁN QUICKSORT

Cụ thể ý tưởng QuickSort khử đệ quy như sau:

- Push \Rightarrow Stack: {Left, Right}
- Trong khi Stack còn chưa rỗng thì thực hiện:
 - Pop: Stack \Rightarrow {Left, Right}
 - Tính $i = \text{Left}$, $j = \text{Right}$, $\text{Mid} = (\text{Left} + \text{Right}) / 2 \rightarrow$ phần tử chốt x là phần tử giữa.
 - Thực hiện các công việc sau {
 - Trong khi (phần tử thứ $i <$ phần tử chốt) tăng i thêm 1.
 - Trong khi (phần tử thứ $j >$ phần tử chốt) giảm j bớt 1.
 - Nếu ($i \leq j$) { + Hoán vị 2 phần tử thứ i và thứ j với nhau.
 - + Tăng i thêm 1, đồng thời giảm j bớt 1 }
 - } Lặp lại trong khi ($i < j$);

BÀI TẬP 3: THUẬT TOÁN QUICKSORT

- Nếu ($i < Right$) thì **Push** \Rightarrow **Stack**: $\{i, Right\}$.
- Nếu ($j > Left$) thì **Push** \Rightarrow **Stack**: $\{Left, j\}$.
- Quá trình trên lặp lại cho đến khi **Stack** rỗng.

BÀI TẬP 3: THUẬT TOÁN QUICKSORT

Bài tập: Cài đặt thuật giải QuickSort không dùng đệ quy.

- Dùng DSLK, mỗi nút chứa thông tin biên trái và biên phải của đoạn chưa được sắp xếp.
- Áp dụng ý tưởng của slide trước để cài đặt.

BÀI TẬP 4: TÍNH GIÁ TRỊ BIỂU THỨC

❖ Thuật toán Ba Lan ngược (Reverse Polish Notation - RPN)

- Định nghĩa RPN:
 - Biểu thức toán học trong đó các toán tử được viết sau toán hạng và không dùng dấu ngoặc.
- Phát minh bởi Jan Lukasiewics một nhà khoa học Ba Lan vào những năm 1950.

BÀI TẬP 4: TÍNH GIÁ TRỊ BIỂU THỨC

Infix : toán tử viết giữa toán hạng

Postfix (RPN): toán tử viết sau toán hạng

Prefix : toán tử viết trước toán hạng

Ví dụ:

INFIX

A + B

A * B + C

A * (B + C)

A - (B - (C - D))

A - B - C - D

RPN (POSTFIX)

A B +

A B * C +

A B C + *

A B C D - - -

A B - C - D -

PREFIX

+ A B

+ * A B C

* A + B C

- A - B - C D

- - - A B C D

BÀI TẬP 4: TÍNH GIÁ TRỊ BIỂU THỨC

Kỹ thuật gạch dưới:

1. Duyệt từ trái sang phải của biểu thức cho đến khi gặp toán tử.
2. Gạch dưới 2 toán hạng ngay trước toán tử và kết hợp chúng bằng toán tử trên.
3. Lặp đi lặp lại cho đến hết biểu thức.

Ví dụ: $2 * ((3 + 4) - (5 - 6))$

→ 2 3 4 + 5 6 - - *

→ 2 3 4 + 5 6 - - *

→ 2 7 5 6 - - *

→ 2 7 5 6 - - *

→ 2 7 -1 - - *

→ 2 7 -1 - - * → 2 8 * → 2 8 * → 16

Chuyển infix thành postfix

- ❖ 1. Khởi tạo Stack rỗng (*chứa các phép toán*).
- ❖ 2. Lặp cho đến khi kết thúc biểu thức:
 - Đọc 01 phần tử của biểu thức
(01 phần tử có thể là hằng, biến, phép toán, ')' hay '(').
 - Nếu phần tử là:
 - 2.1 '**(**' : đưa vào Stack.
 - 2.2 '**)**' : lấy các phần tử của Stack ra cho đến khi lấy xong '**(**'.
 - 2.3 Một phép toán: **^ + - * /**
 - Nếu **Stack rỗng**: đưa vào Stack.
 - Nếu Stack khác rỗng và **phép toán có độ ưu tiên cao hơn phần tử ở đầu Stack**: đưa vào Stack.
 - Nếu Stack khác rỗng và **phép toán có độ ưu tiên thấp hơn hoặc bằng phần tử ở đầu Stack**:
 - Lấy phần tử ở đỉnh Stack ra;
 - Sau đó lặp lại việc so sánh với phần tử ở đỉnh Stack.
 - 2.4 **Hằng hoặc biến**: đưa vào postfix.
 - ❖ 3. Lấy hết tất cả các phần tử của Stack ra.

Độ ưu tiên:	
^	cao nhất
* , /	cao nhì
+ , -	cao ba

Ví dụ:

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

PostFix



Push (



Display A



Push +



Display



Push *



Display



Read)



Pop *, Display *,



Pop +, Display +,



Push /



Push (



Display



Push -



Push (



Display E



Push -



Display F



Read)



Pop -, Display -, Pop (



Read)



Pop -, Display -, Pop (



Pop /, Display /

A

AB

ABC

ABC*

ABC*+

ABC*+D

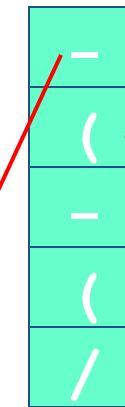
ABC*+DE

ABC*+DEF

ABC*+DEF-

ABC*+DEF--

ABC*+DEF---



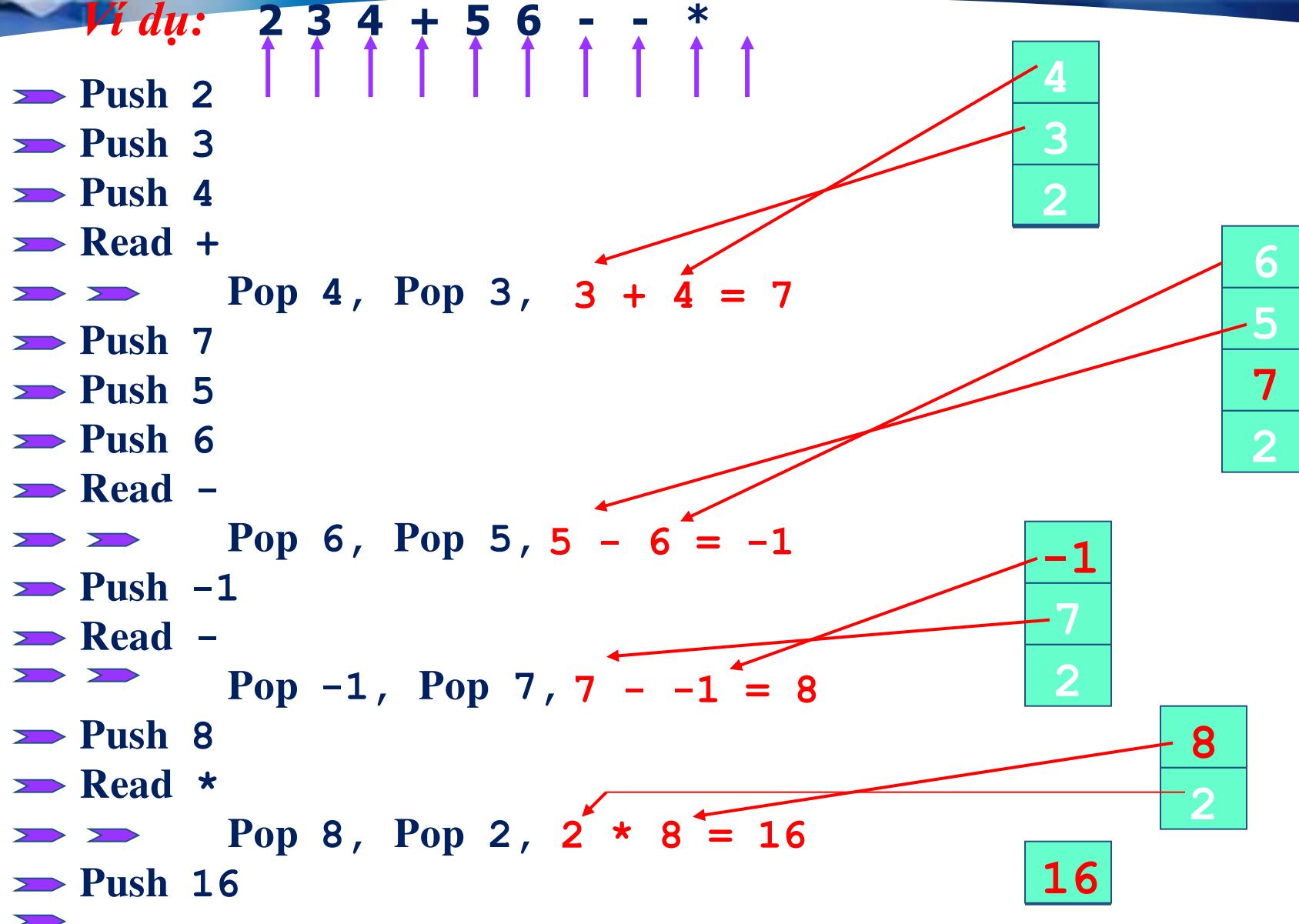
—

BÀI TẬP 4: TÍNH GIÁ TRỊ BIỂU THỨC

1. Khởi tạo Stack rỗng (*chứa hằng hoặc biến*).
2. Lặp cho đến khi kết thúc biểu thức:
 - Đọc 01 phần tử của biểu thức (*hằng, biến, phép toán*).
 - Nếu phần tử là hằng hay biến: đưa vào Stack.
 - Ngược lại:
 - Lấy ra 2 phần tử của Stack ra.
 - Áp dụng phép toán cho 2 phần tử vừa lấy ra.
 - Đưa kết quả vừa tính được vào lại Stack.
3. Giá trị của biểu thức chính là phần tử cuối cùng của Stack.

$$2 * ((3 + 4) - (5 - 6))$$

Ví dụ:



TÍNH GIÁ TRỊ BIỂU THỨC

❖ Giai đoạn 1: Chuyển biểu thức trung tố sang hậu tố

Trung tố (infix)

$(6 / 2 + 3) * (7 - 4)$

Hậu tố (Postfix)

$6\ 2\ /\ 3\ +\ 7\ 4\ -\ *$



TÍNH GIÁ TRỊ BIỂU THỨC

Chuyển từ trung tố về hậu tố.

❖ Duyệt qua từng phần tử trong infix (C)

- Nếu C là toán hạng thì bỏ vào postfix
- Nếu C là '(' thì push \Rightarrow **Stack**.
- Nếu C là ')' thì lấy tất cả phần tử trong Stack ra bỏ vào postfix cho đến khi gặp '('.

TÍNH GIÁ TRỊ BIỂU THỨC

- Nếu C là toán tử:
 - Lấy các toán tử có độ ưu tiên cao hơn bỏ vào postfix.
 - Bỏ toán tử vào Stack.

Ví dụ: $(2 * 3 + 8 / 2) * (5 - 1)$

TÍNH GIÁ TRỊ BIỂU THỨC

$$(2 * 3 + 8 / 2) * (5 - 1)$$

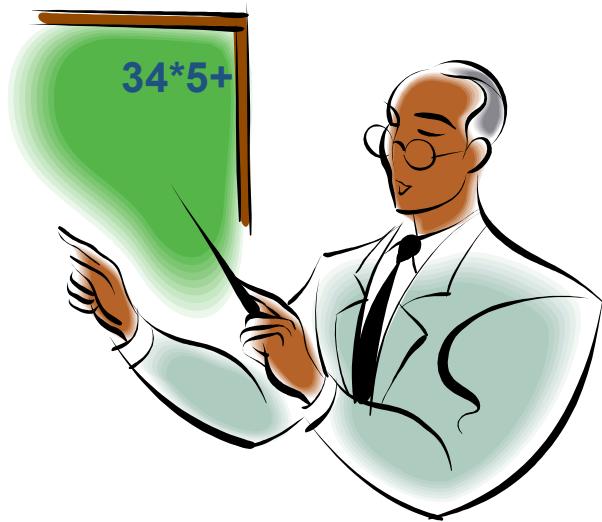
Đọc	Xử lý	Stack	postfix
(Đẩy vào Stack	(
2	Bỏ vào postfix	(2
*	Do ở đỉnh Stack chứa '(' nên bỏ dấu '*' vào Stack.	(*	2
3	Bỏ vào postfix	(*	2 3
+	Do '+' có độ ưu tiên thấp hơn '*' ở đỉnh Stack nên ta lấy '*' ra và bỏ vào postfix. Do ở đỉnh Stack chứa '(' nên bỏ dấu '+' vào Stack.	(+	2 3 *
8	Bỏ vào postfix	(+	2 3 * 8
/	Do '/' có độ ưu tiên cao hơn '+' ở đỉnh Stack nên đưa '/' vào Stack.	(+ /	2 3 * 8
2	Bỏ vào postfix	(+ /	2 3 * 8 2

TÍNH GIÁ TRỊ BIỂU THỨC

Đọc	Xử lý	Stack	postfix
)	Lấy trong Stack ra cho đến khi gặp ngoặc '('		2 3 * 8 2 / +
*	Đưa vào Stack	*	2 3 * 8 2 / +
(Đưa vào Stack	* (2 3 * 8 2 / +
5	Bỏ vào postfix	* (2 3 * 8 2 / + 5
-	Do trên đỉnh Stack chứa '(' nên bỏ dấu '-' vào Stack.	* (-	2 3 * 8 2 / + 5
1	Bỏ vào postfix	* (-	2 3 * 8 2 / + 5 1
)	Lấy trong Stack ra cho đến khi gặp ngoặc mở '('	*	2 3 * 8 2 / + 5 1 -
	Lấy những phần tử còn lại trong Stack và bỏ vào Stack.		2 3 * 8 2 / + 5 1 - *

TÍNH GIÁ TRỊ BIỂU THỨC

❖ Giai đoạn 2: Tính giá trị biểu thức postfix



Jan Lukasiewicz

Postfix không cần có dấu ngoặc vẫn có thể tính đúng bằng cách đọc lần lượt biểu thức từ trái qua phải và dùng một Stack để lưu trữ kết quả trung gian.

TÍNH GIÁ TRỊ BIỂU THỨC

Ý tưởng:

- ❖ **Khởi tạo Stack = {Ø}**
- ❖ **Đọc lần lượt các phần tử từ trái, kiểm tra**
 - Nếu toán hạng: Push \Rightarrow Stack
 - Nếu toán tử: lấy hai toán hạng, thực hiện phép toán, kết quả Push vào Stack
- ❖ **Sau khi đọc xong, trong Stack còn duy nhất một phần tử \Rightarrow kết quả!**

TÍNH GIÁ TRỊ BIỂU THỨC

$$(2 * 3 + 8 / 2) * (5 - 1)$$

$$2 \ 3 * 8 \ 2 / + 5 \ 1 - *$$

Đọc	Xử lý	Stack	Output
2,3	Đưa vào Stack	2, 3	
*	2*3	6	
8	Đưa vào Stack	6, 8	
2	Đưa vào Stack	6, 8, 2	
/	Lấy 8/2	6, 4	
+	Lấy 6 + 4	10	
5	Đưa vào Stack	10, 5	
1	Đưa vào Stack	10, 5, 1	
-	Lấy 5 - 1	10, 4	
*	Lấy 10 * 4	40	40

BÀI TẬP

$$3 * (2 + 6 * 2 / 3 - 1) - 2 * 3 / 2 + 1$$

Thank for your attention!



See you next week!

THAM KHẢO THÊM

TỔ CHỨC STACK THEO MÀNG 1 CHIỀU

TỔ CHỨC THEO MÀNG 1 CHIỀU

Mảng 1 chiều

Kích thước Stack khi
quá thiếu, lúc quá thừa



Hiện thực Stack dùng mảng

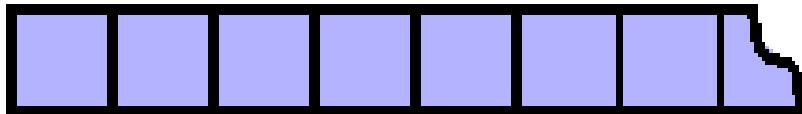
- ❖ Có thể tạo một Stack bằng cách khai báo một mảng 1 chiều với kích thước tối đa là MAXSIZE (ví dụ: *MAXSIZE = 1000*)
 - Stack có thể chứa tối đa MAXSIZE phần tử đánh số từ 0 đến MAXSIZE - 1
- ❖ Phần tử nằm ở đỉnh Stack sẽ có chỉ số là Top (*lúc đó trong Stack đang chứa Top + 1 phần tử*)
- ❖ Như vậy, để khai báo một Stack, ta cần một mảng 1 chiều Stack, và 1 biến số nguyên Top cho biết chỉ số của đỉnh Stack:

KHAI BÁO STACK

Tạo cấu trúc dữ liệu cho Stack

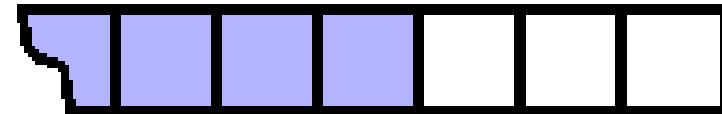
```
#define MAXSIZE 50 //Stack chứa 50 phần tử
```

```
typedef int ItemType;
struct Stack
{
    int Top; //Số phần tử
    ItemType Info[MAXSIZE];
};
```



0 1 2

...



Top MAXSIZE-1

THAO TÁC VỚI STACK

❖ Các thao tác trên Stack

initStack

isEmpty

isFull

push

pop

getTop

getSize

Hiện thực Stack dùng mảng

❖ Ngăn xếp được cài đặt bằng mảng phải có:

- Một biến Top dùng để lưu vị trí con trỏ hiện tại.
- Một mảng dùng để lưu giá trị của ngăn xếp.

❖ Khởi tạo ngăn xếp rỗng:

- Ngăn xếp rỗng là ngăn xếp không chứa bất kỳ phần tử nào → Đỉnh ngăn xếp không trỏ tới phần tử nào.

❖ Kiểm tra ngăn xếp rỗng:

- Ngăn xếp rỗng khi $\text{Top} = -1$

❖ Kiểm tra ngăn xếp đầy:

- Ngăn xếp đầy khi $\text{Top} = \text{MAXSIZE}$

Hiện thực Stack dùng mảng

❖ Thêm một phần tử vào ngăn xếp:

- Kiểm tra ngăn xếp đầy.
- Top tăng lên 1 đơn vị.
- Phần tử tại vị trí Top bây giờ sẽ là phần tử cần thêm vào.

❖ Lấy một phần tử trong ngăn xếp:

- Kiểm tra ngăn xếp rỗng.
- Trả về phần tử tại vị trí Top.

❖ Xóa 1 phần tử khỏi ngăn xếp:

- Kiểm tra ngăn xếp rỗng.
- Top giảm đi 1 đơn vị.

Hiện thực Stack dùng mảng

Nhận xét:

- Các thao tác trên đều làm việc với chi phí **O(1)**
- Việc cài đặt Stack thông qua mảng một chiều đơn giản và khá hiệu quả.
- Tuy nhiên, hạn chế lớn nhất của phương án cài đặt này là giới hạn về kích thước của Stack (*MAXSIZE*).
 - Giá trị của *MAXSIZE* có thể quá nhỏ so với nhu cầu thực tế hoặc quá lớn sẽ làm lãng phí bộ nhớ.
 - Nên hạn chế việc cài đặt ngăn xếp bằng mảng.

KHỞI TẠO STACK

```
void initStack(Stack &s)
{ //số phần tử ban đầu = 0
    s.Top = -1;
}
```

KIỂM TRA STACK RỖNG/ ĐẦY

```
int isEmpty(Stack s)
{
    return (s.Top == -1);
}

int isFull(Stack s)
{
    return (s.Top == MAXSIZE);
}
```

THÊM PHẦN TỬ VÀO STACK

```
int push(stack &s, ItemType x)
{
    if(isFull(s) == 1)
        return 0;
    s.Top++;
    s.Info[s.Top] = x;
    return 1;
}
```

Có thể viết gọn lại:
s.Info[++s.Top] = x;

LẤY PHẦN TỬ RA KHỎI STACK

```
int pop(Stack &s, ItemType &x)
{
    if (isEmpty(s) == 1)
        return 0;
    x = s.Info[s.Top];
s.Top--;
    return 1;
}
```

Có thể viết gọn lại:
x = s.Info[s.Top--];

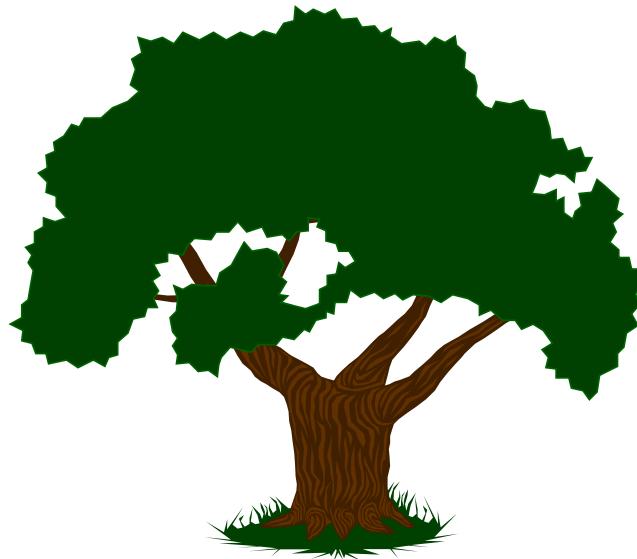
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT



GV : ThS. Trần Văn Thọ

E-mail : thotv@hufi.edu.vn

Đây là gì? Nó có những bộ phận chính nào?



Môn: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

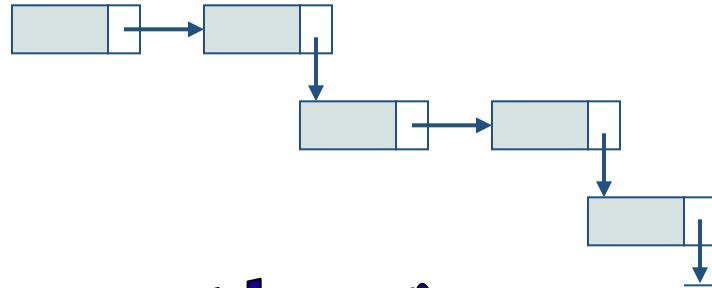


**CHƯƠNG V:
CÂY**

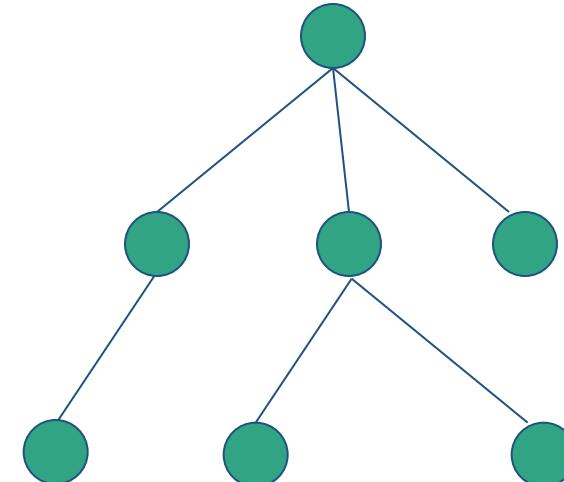
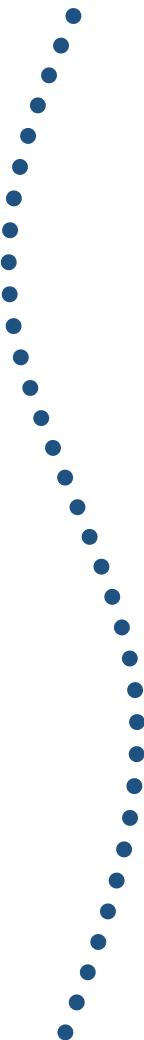
Nội dung

- Cấu trúc cây
- Cây nhị phân
- Cây nhị phân tìm kiếm
- Cây nhị phân tìm kiếm cân bằng

Cấu trúc dữ liệu



Linear



Hierarchical structures

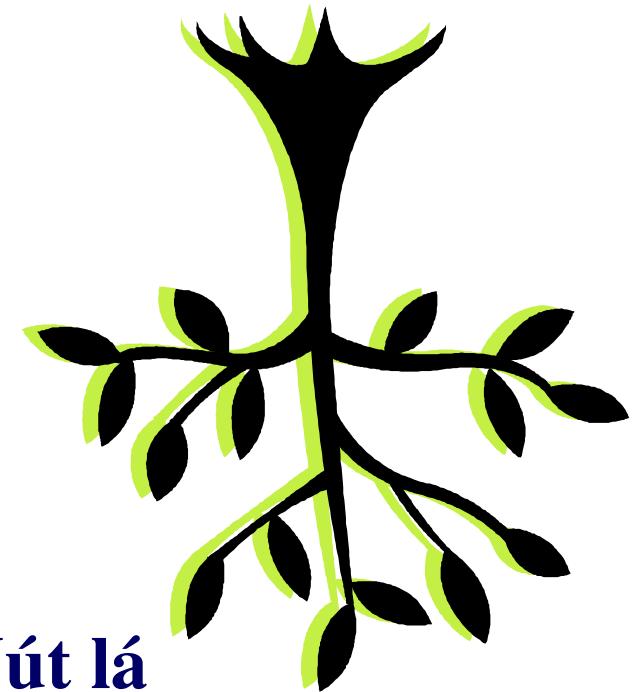
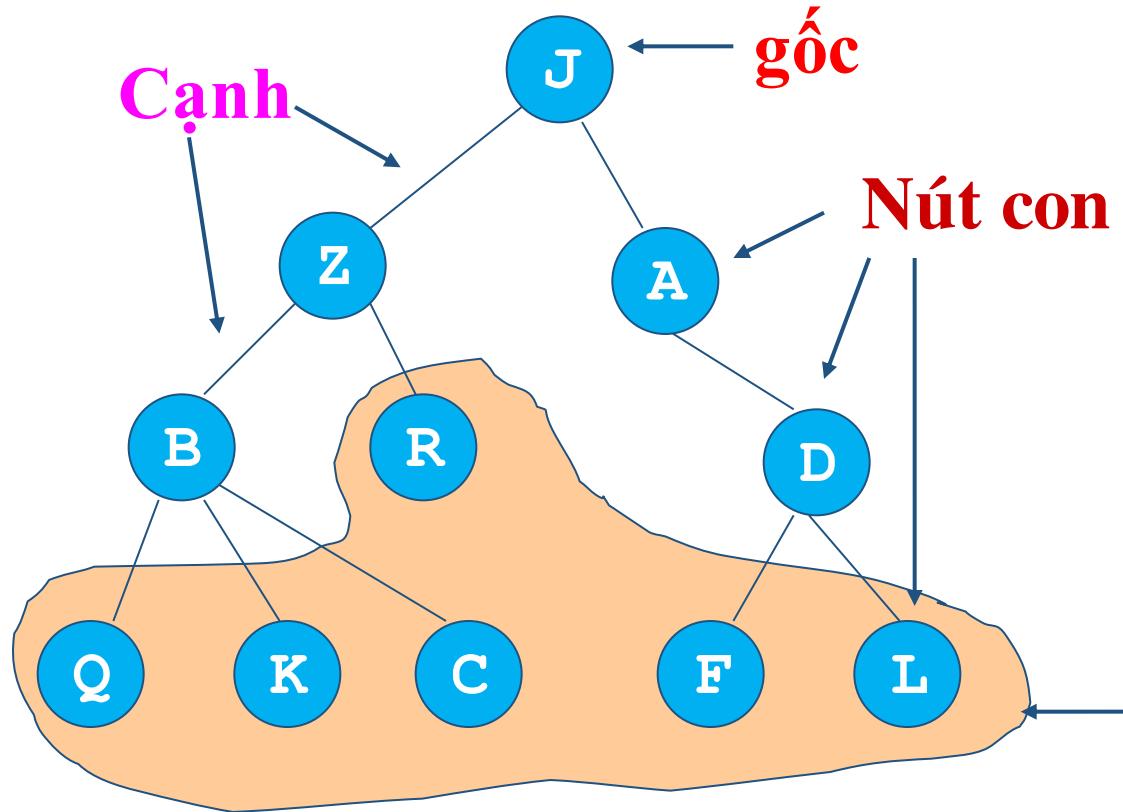
Cấu trúc cây

- Tập hợp các nút và cạnh nối các nút đó.
- Có một nút gọi là *gốc*.
- Quan hệ one-to-many giữa các nút.
- Có duy nhất một đường đi từ gốc đến nút con.
- Các loại cây:
 - **Nhi phân**: mỗi nút có $\{0, 1, 2\}$ nút con
 - **Tam phân**: mỗi nút có $\{0, 1, 2, 3\}$ nút con
 - **n-phân**: mỗi nút có $\{0, 1, \dots, n\}$ nút con

Cấu trúc cây



Khái niệm



Nút lá

Khái niệm

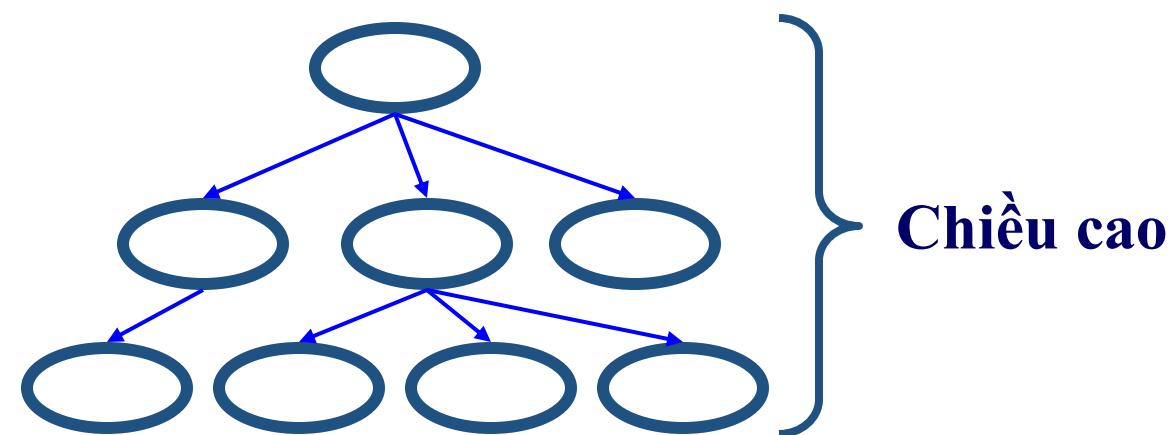
❖ Thuật ngữ

- **Nút gốc:** không có nút cha
- **Nút lá:** không có nút con
- **Nút trong:** không phải nút lá và nút gốc
- **Chiều cao:** khoảng cách từ gốc đến lá của nhánh cao nhất

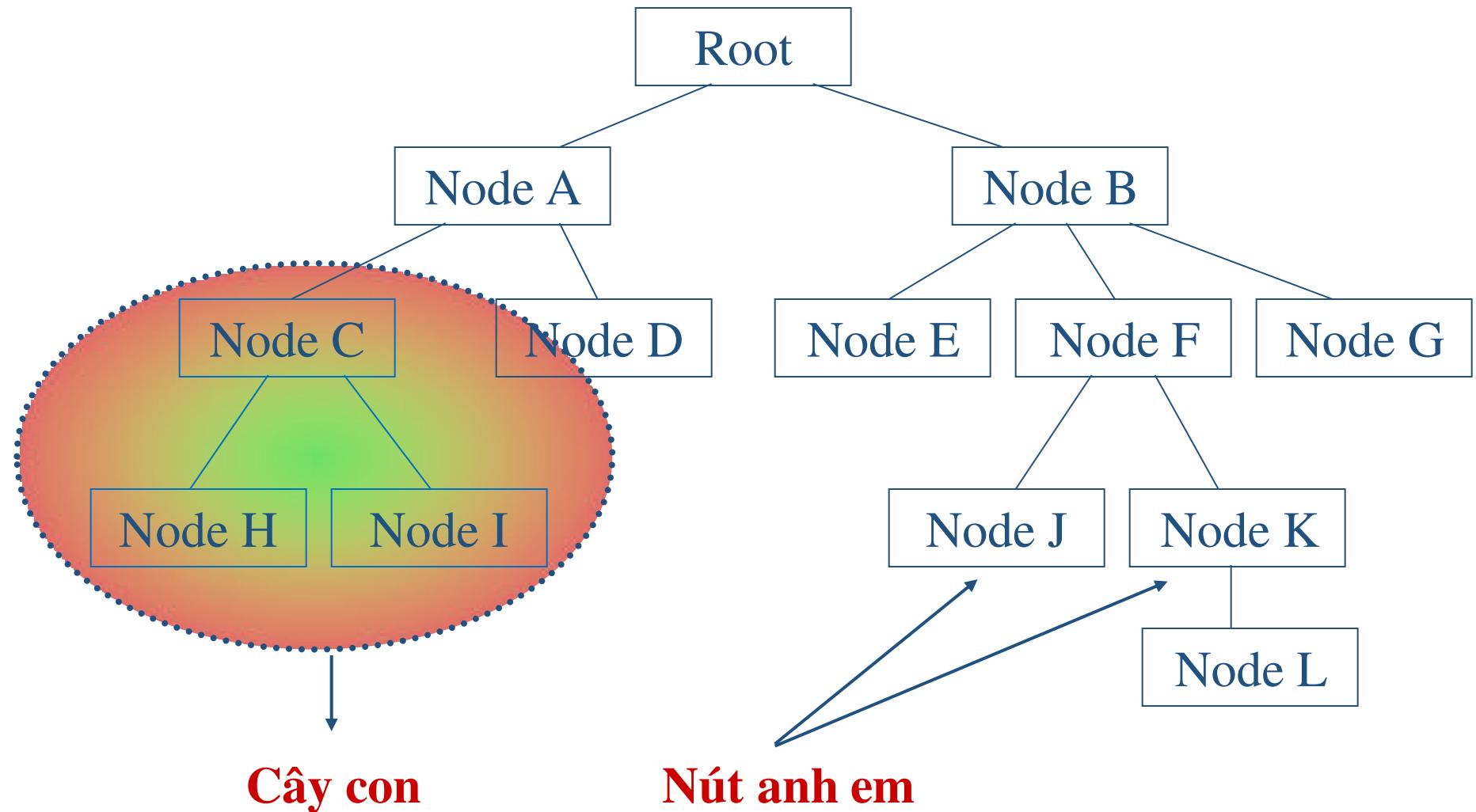
Nút gốc

Nút trong

Nút lá



Khái niệm



Nội dung

- Cấu trúc cây
- **Cây nhị phân**
- Cây nhị phân tìm kiếm
- Cây nhị phân tìm kiếm cân bằng

Cây nhị phân

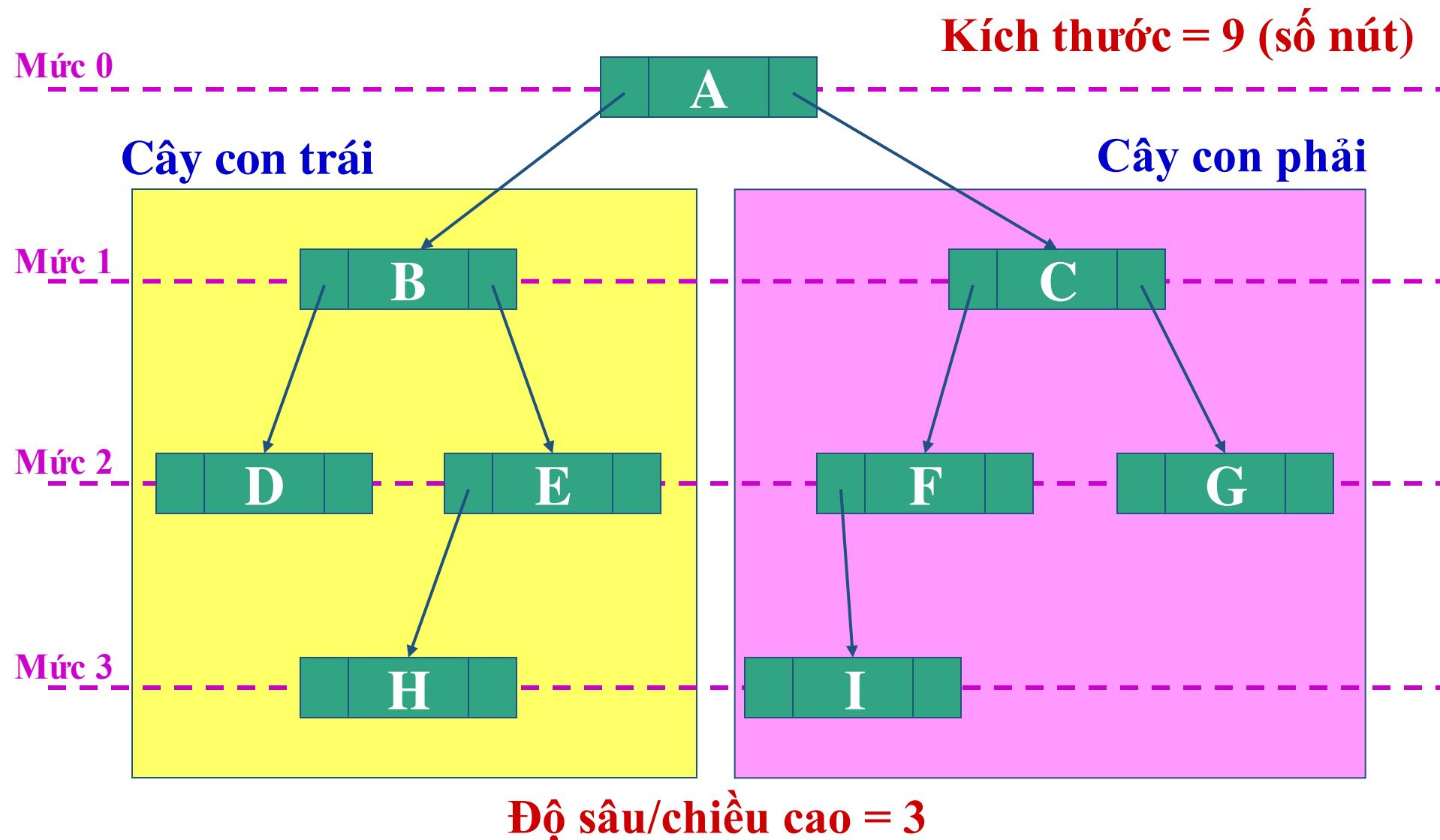
- Cây nhị phân là một đồ thị có hướng, mỗi nút có tối đa 2 nút con, hay *Cây nhị phân là cây có bậc bằng hai (bậc của mỗi nút tối đa bằng 2)*.
- Cấu trúc cây đơn giản nhất
- Tại mỗi nút gồm các 3 thành phần
 - Phần **Data**: chứa giá trị, thông tin của nút
 - Phần **Left**: Liên kết đến nút con trái (nếu có)
 - Phần **Right**: Liên kết đến nút con phải (nếu có)



Cây nhị phân

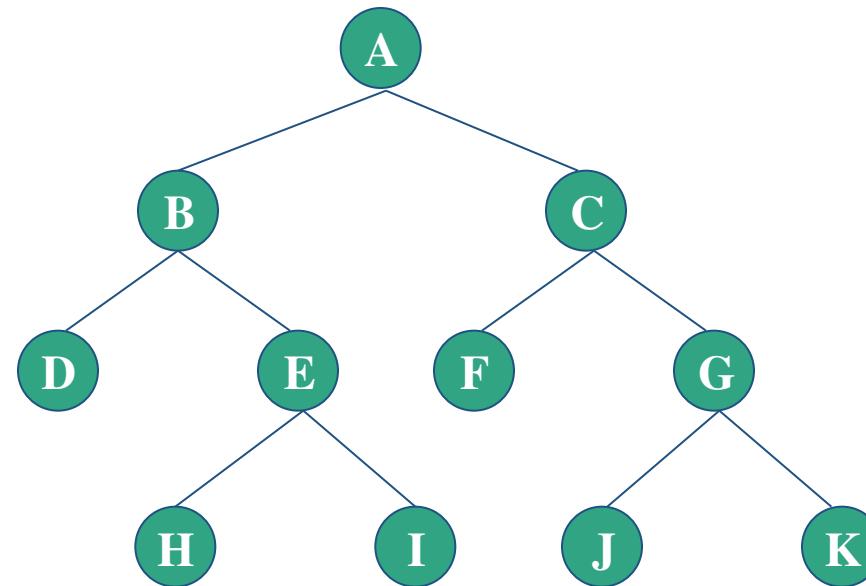
- **Cây nhị phân có thể rỗng** (không có nút nào)
- **Cây nhị phân khác rỗng** có 1 nút gốc
 - Có duy nhất 1 đường đi từ gốc đến 1 nút con.
 - Nút không có nút con bên trái và con bên phải là nút lá.

Cây nhị phân



Cây nhị phân

- **Cây nhị phân đúng:**
 - Nút gốc và nút trung gian có đúng 2 con
- **Cây nhị phân đúng có n nút lá thì số nút trên cây $2^* n - 1$.**



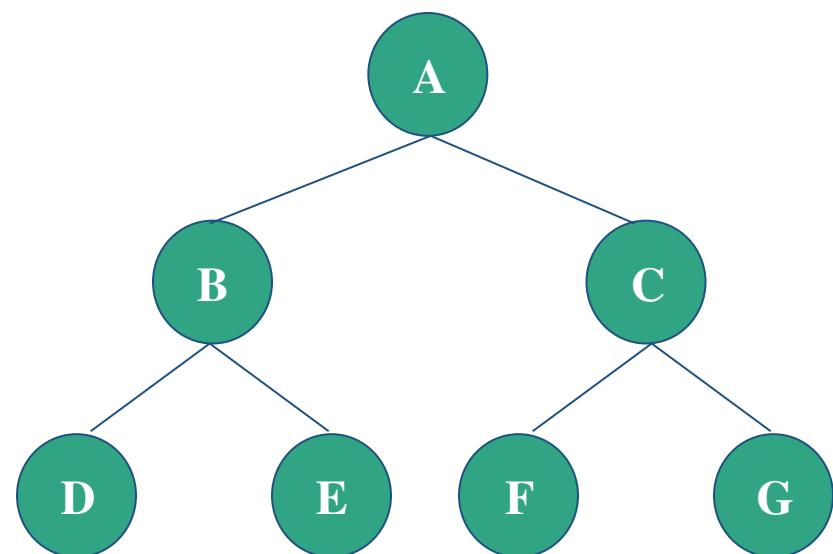
Cây nhị phân

- **Cây nhị phân đầy đủ với chiều sâu d**
 - Phải là cây nhị phân đúng
 - Tất cả nút lá có chiều sâu d

Số nút = $(2^{d+1} - 1)$

Số nút trung gian = ?

Biết số nút tính d của cây
nhị phân đầy đủ



Cấu trúc cây nhị phân

■ Cấu trúc của cây nhị phân.

```
typedef int ItemType;
```

```
struct TNode
```

```
{ //Cấu trúc của một nút
```

```
    ItemType Info;
```

```
    TNode* Left;
```

```
    TNode* Right;
```

```
};
```

```
struct BTree
```

```
{ //Cấu trúc của một cây
```

```
    TNode* Root;
```

```
};
```

ItemType *Info*;

TNode* *Left*;

TNode* *Right*;



Chứa thông tin của nút



Trỏ đến nút con trái

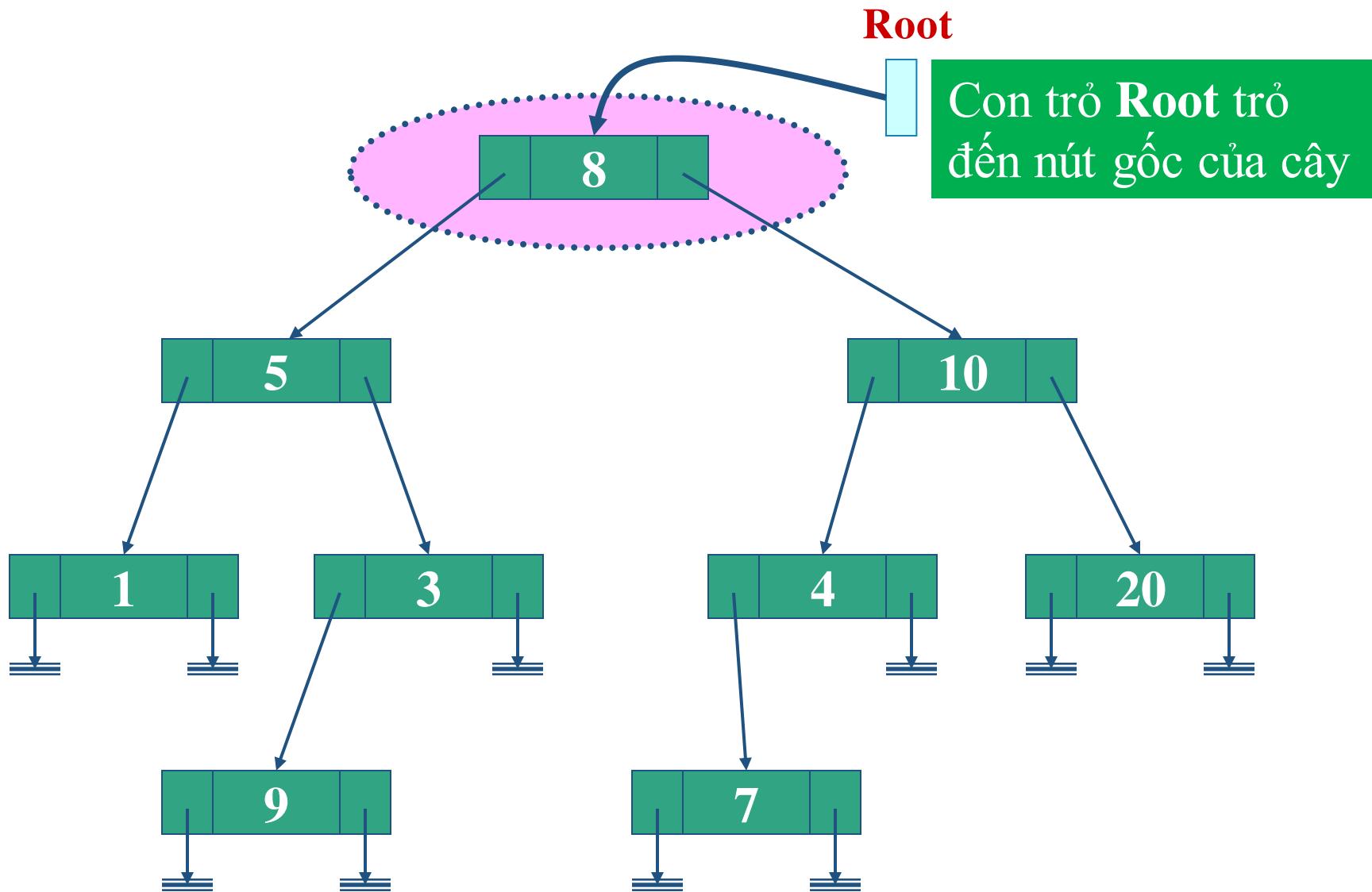


Trỏ đến nút con phải



Con trỏ đến nút gốc của cây

Sơ đồ cây nhị phân



Các thao tác cơ bản

- **Khởi tạo cây**
- **Tạo nút mới.**
- **Thêm nút con trái T**
- **Thêm nút con phải T**
- **Thêm nút có giá trị x**
- **Duyệt cây**
- **Xóa con trái T**
- **Xóa con phải T**
- **Xóa nút có giá trị x**
- **Xóa cây.**

Khởi tạo cây nhị phân rỗng

```
void initBTree(BTree &bt)
{
    bt.Root = NULL;
}
```

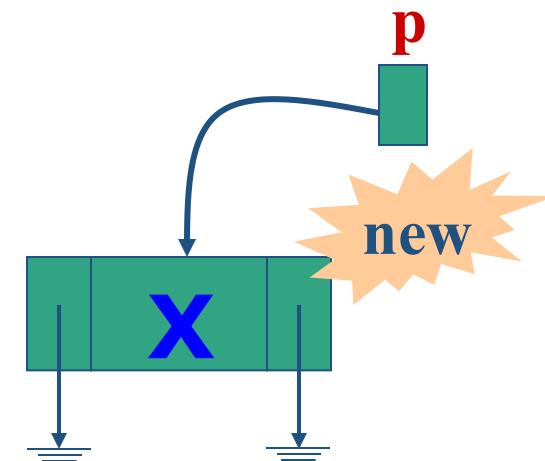
Tạo nút mới

TNode* *createTNode*(ItemType x)

{

```
TNode* p = new TNode;  
if(!p) return NULL;  
p->Info = x;  
p->Left = NULL;  
p->Right = NULL;  
return p;
```

}



Thêm nút con bên trái của node T

```
int insertTNodeLeft(TNode* T, ItemType x)
{
    if(T == NULL)
        return 0; //Không tồn tại nút T
    if(T->Left != NULL)
        return 0; //Đã tồn tại nút con trái
    TNode* p = createTNode(x);
    T->Left = p;
    return 1;
}
```

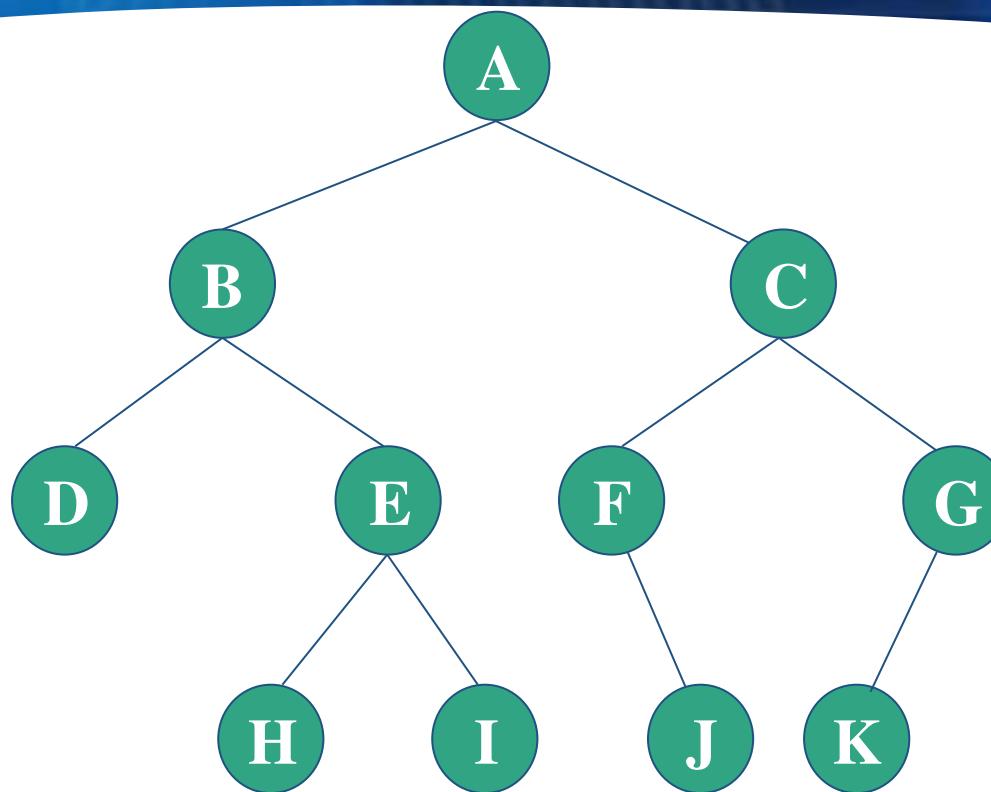
Thêm nút con bên phải của node T

```
int insertTNodeRight(TNode* T, ItemType x)
{
    if(T == NULL)
        return 0; //Không tồn tại nút T
    if(T→Right != NULL)
        return 0; //Đã tồn tại nút con phải
    TNode* p = createTNode(x);
    T→Right = p;
    return 1;
}
```

Duyệt cây

- **Do cây là cấu trúc không tuyến tính**
- **Có 6 cách duyệt cây nhị phân:**
 - Duyệt theo thứ tự trước PreOrder:
traverseNLR, traverseNRL
 - Duyệt theo thứ tự giữa InOrder:
traverseLNR, traverseRNL
 - Duyệt theo thứ tự sau PostOrder:
traverseLRN, traverseRLN

Duyệt cây



PreOrder (traverseNLR, traverseNRL)

InOrder (traverseLNR, traverseRNL)

PostOrder (traverseLRN, traverseRLN)

Duyệt Node Left Right

```
void traverseNLR(TNode* root)
```

```
{
```

```
if(root == NULL) return;
```

<xử lý gốc>;

```
traverseNLR(root→Left);
```

```
traverseNLR(root→Right);
```

```
}
```

Duyệt Node Left Right

```
void traverseNRL(TNode* root)
{
    if(root == NULL) return;
    <xử lý gốc>
    traverseNRL(root→Right);
    traverseNRL(root→Left);
}
```

Duyệt Left Node Right

```
void traverseLNR(TNode* root)
```

```
{
```

```
    if(root == NULL) return;
```

```
    traverseLNR(root→Left);
```

<xử lý gốc>;

```
    traverseLNR(root→Right);
```

```
}
```

Duyệt Left Node Right

```
void traverseRNL(TNode* root)
```

```
{
```

```
    if(root == NULL) return;
```

```
    traverseRNL(root→Right);
```

<xử lý gốc>;

```
    traverseRNL(root→Left);
```

```
}
```

Duyệt Left Right Node

```
void traverseLRN(TNode* root)
```

```
{
```

```
    if(root == NULL) return;
```

```
    traverseLRN(root→Left);
```

```
    traverseLRN(root→Right);
```

<xử lý gốc>;

```
}
```

Duyệt Left Right Node

```
void traverseRLN(TNode* root)
```

```
{
```

```
    if(root == NULL) return;
```

```
    traverseRLN(root→Right);
```

```
    traverseRLN(root→Left);
```

<xử lý gốc>;

```
}
```

Xóa nút con trái của node T

```
int deleteTNodeLeft(TNode* T)
{//Nút này phải là nút lá
    if(T == NULL) return 0;
    TNode* p = T → Left;
    if(p == NULL) return 0;
    if(p→Left != NULL || p→Right != NULL)
        return 0;
    delete p;
    return 1;
}
```

Xóa nút con phải của node T

```
int deleteTNodeRight(TNode* T)
{//Nút này phải là nút lá
    if(T == NULL) return 0;
    TNode* p = T→Right;
    if(p == NULL) return 0;
    if(p→Left != NULL || p→Right != NULL)
        return 0;
    delete p;
    return 1;
}
```

Tìm nút có khóa x

TNode* **findTNode**(TNode* root, ItemType x)

{

 if(!root) return NULL;

 if(root→Info == x) return root;

 TNode* p = **findTNode**(root→Left, x);

 if(p) return p;

 return **findTNode**(root→Right, x);

}

Xóa cây

```
int deleteTree(TNode* &root)
{
    if(!root) return 0;
    deleteTree(root→Left);
    deleteTree(root→Right);
    delete root;
    return 1;
}
```

Các thao tác mở rộng

- Đếm số nút trên cây.
- Đếm số nút lá trên cây.
- Đếm số nút trong.
- Xác định độ sâu/chiều cao của cây.
- Đếm số nút có giá trị bằng x.
- Tìm giá trị nhỏ nhất/lớn nhất trên cây.
- Tính tổng các giá trị trên cây.
- Xuất ra màn hình các nút ở mức thứ k.
- Đếm các nút lá ở mức k

Đếm số nút trên cây

```
int countTNode(TNode* root)
{
    if(!root) return 0;
    int cnl = countTNode(root→Left);
    int cnr = countTNode(root→Right);
    return (1 + cnl + cnr);
}
```

Đếm số nút lá

```
int countTNodeLeaf(TNode* root)
{
    if(!root) return 0;
    int cnl = countTNodeLeaf(root→Left);
    int cnr = countTNodeLeaf(root→Right);
    if(!root→Left && !root→Right)
        return (1 + cnl + cnr);
    return (cnl + cnr);
}
```

Đếm số nút không phải nút lá

```
int countTNodeNoLeaf(TNode* root)
{
    if(!root) return 0;
    int cnl=countTNodeNoLeaf(root→Left);
    int cnr=countTNodeNoLeaf(root→Right);
    if(root→Left || root→Right)
        return (1 + cnl + cnr);
    return (cnl + cnr);
}
```

Đếm số nút trong (*nút trung gian*)

```
int countTNodeMedium(TNode* root)
{
    int cn = countTNode(root);
    if(cn <= 2) return 0;
    int cnl = countTNodeLeaf(root);
    return (cn - cnl - 1); //trừ nút gốc
}
```

Đếm số nút trong (*nút trung gian*)

```
int countTNodeMedium(TNode* root)
{
    int n = countTNodeNoLeaf(root);
    if(n > 0)
        return (n - 1); //trừ nút gốc
    return 0;
}
```

Tính tổng

```
int sumTNode(TNode* root)
{
    if(!root) return 0;
    int suml = sumTNode(root→Left);
    int sumr = sumTNode(root→Right);
    return (root→Info + suml + sumr);
}
```

Tính chiều cao của cây

```
int highBTree(TNode* root)
{
    if(!root) return 0;
    int hl = highBTree(root→Left);
    int hr = highBTree(root→Right);
    if(hl > hr)
        return (1 + hl);
    else
        return (1 + hr);
}
```

Đếm số nút có giá trị bằng x

```
int countTNodeX(TNode* root, int x)
{
    if(!root) return 0;
    int nlx = countTNodeX(root→Left, x);
    int nrx = countTNodeX(root→Right, x);
    if(root→Info == x)
        return (1 + nlx + nrx);
    return (nlx + nrx);
}
```

Tìm giá trị lớn nhất trên cây nhị phân

```
int Max(int a, int b) {return (a>b) ? a : b; }

int maxTNode(TNode* root)
{
    if(!root→Left && !root→Right)
        return (root→Info);

    int maxl = maxTNode(root→Left);
    int maxr = maxTNode(root→Right);
    return Max(root→Info, Max(maxl, maxr));
}
```

Tìm giá trị lớn nhất trên cây nhị phân

```
int maxTNode(TNode* root) {  
    if(!root→Left && !root→Right)  
        return (root→Info);  
  
    int maxl = maxTNode(root→Left);  
    int maxr = maxTNode(root→Right);  
  
    int max = root→Info;  
    if(max < maxl) max = maxl;  
    if(max < maxr) max = maxr;  
  
    return max;  
}
```

Xuất ra màn hình các nút ở mức k

```
void showTNodeLevelK(TNode* root, int k)
{
    if(!root) return;
    if(k == 0) //đến tầng cần tìm
        printf("%4d", root→Info);
    k--;
    //mức k giảm dần về 0
    showTNodeLevelK(root→Left, k);
    showTNodeLevelK(root→Right, k);
}
```

Đếm các nút lá ở mức k

```
int countTNodeLeafLevelK(TNode* root, int k)
```

```
{
```

```
    if(!root) return 0;
```

```
    if(k==0 && !root->Left && !root->Right)  
        return (1);
```

```
    k--; //mức k giảm dần về 0
```

```
    int nl=countTNodeLeafLevelK(root->Left,k);
```

```
    int nr=countTNodeLeafLevelK(root->Right,k);
```

```
    return (nl + nr);
```

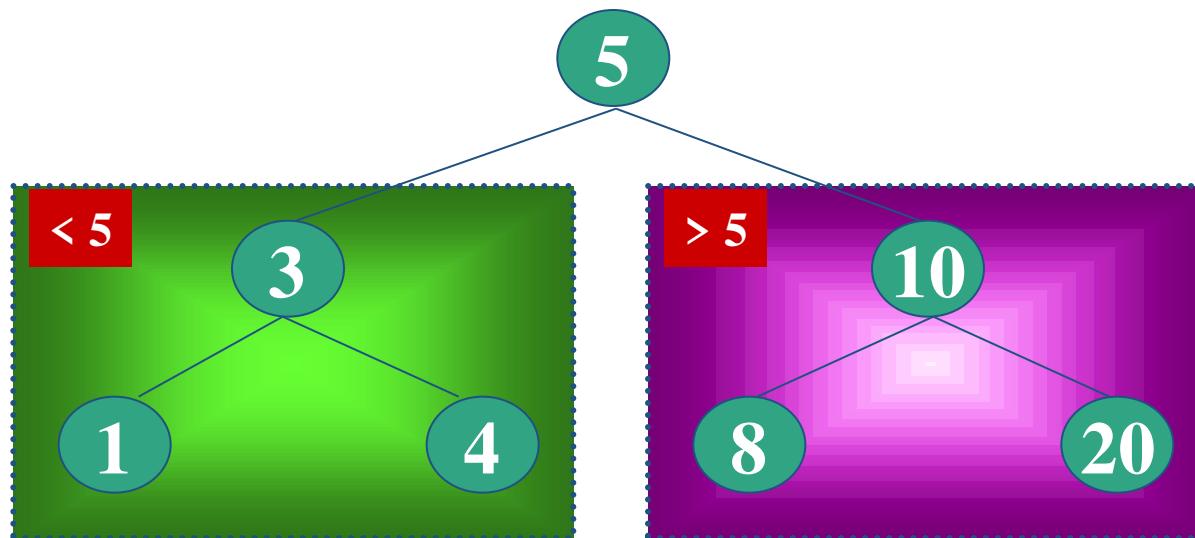
```
}
```

Nội dung

- Cấu trúc cây
- Cây nhị phân
- **Cây nhị phân tìm kiếm (BST- Binary Search Tree)**
- Cây nhị phân tìm kiếm cân bằng

Khái niệm

- **BST là cây nhị phân mà mỗi nút thoả:**
 - Giá trị của tất cả nút con trái $<$ nút gốc
 - Giá trị của tất cả nút con phải $>$ nút gốc



CẤU TRÚC CÂY NHỊ PHÂN TÌM KIẾM

- Cấu trúc dữ liệu của cây nhị phân tìm kiếm.

```
typedef int ItemType;
```

```
struct TNode
```

```
{ //Cấu trúc của một nút
```

```
    ItemType Info;
```

```
    TNode* Left;
```

```
    TNode* Right;
```

```
};
```

```
struct BSTree
```

```
{ //Cấu trúc của một cây
```

```
    TNode* Root;
```

```
};
```

Chứa thông tin của nút

Trỏ đến nút con trái

Trỏ đến nút con phải

Con trỏ đến nút gốc của cây

Cây nhị phân tìm kiếm

■ Xây dựng cây BST

- Tìm kiếm/ thêm
- Xóa

■ Luôn duy trì tính chất

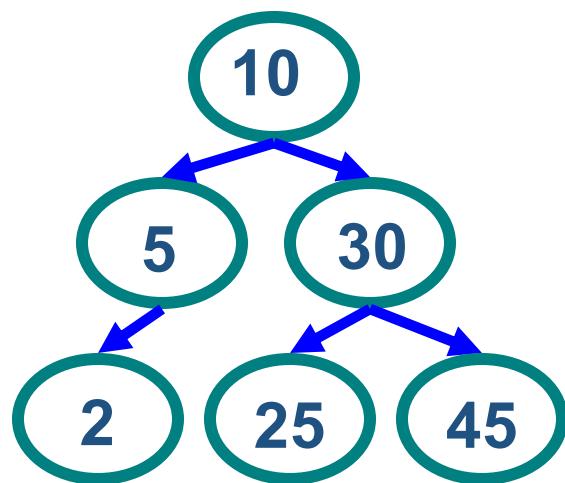
- Giá trị nhỏ hơn ở bên cây con trái
- Giá trị lớn hơn ở bên cây con phải

Tìm kiếm

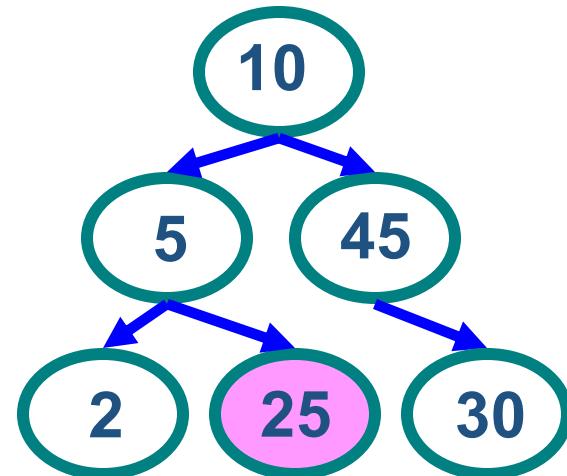
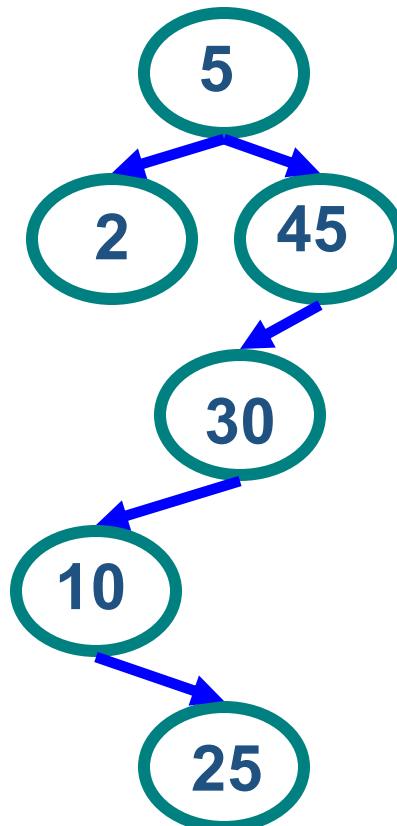
■ Xuất phát từ gốc

- Nếu gốc = NULL => không tìm thấy
- Nếu khóa x = khóa nút gốc => tìm thấy
- Ngược lại nếu khóa x < khóa nút gốc =>
Tìm trên cây bên trái
- Ngược lại => tìm trên cây bên phải

Ví dụ

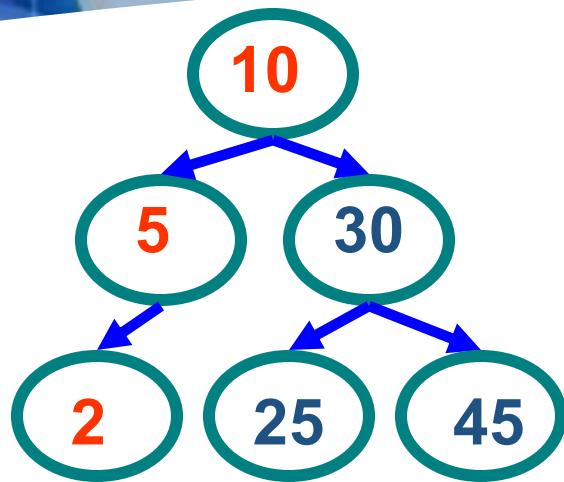


Binary
search trees



Non-binary
search tree

Ví dụ tìm $x = 2$



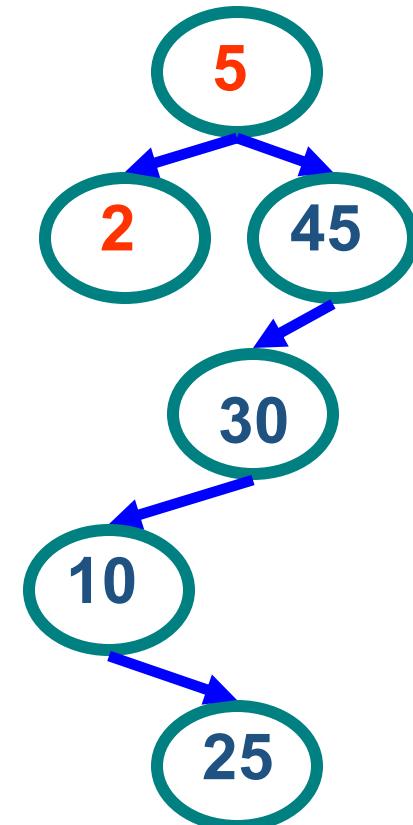
$10 > 2$, Left

$5 > 2$, Left

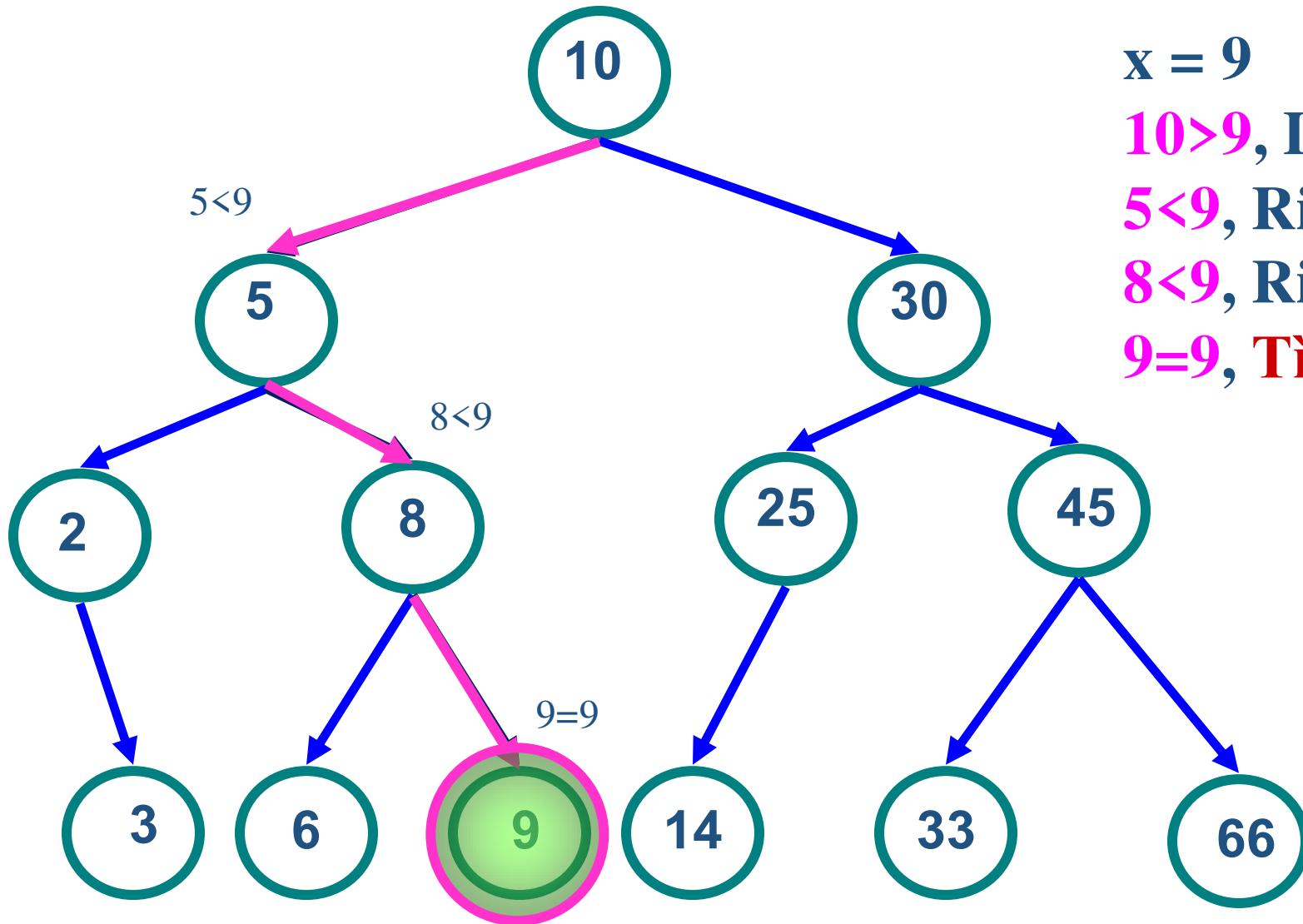
$2 = 2$, Tìm thấy

$5 > 2$, Left

$2 = 2$, Tìm thấy

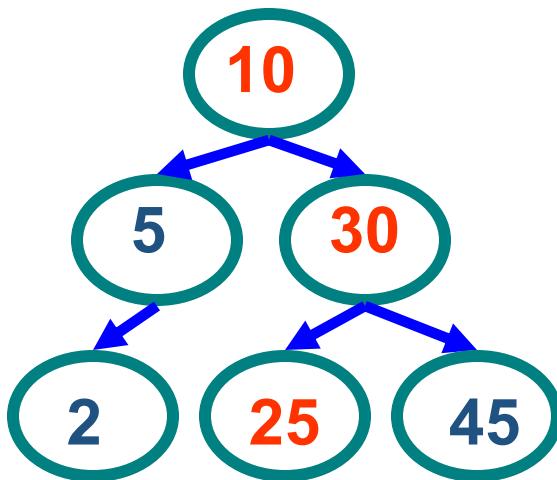


Ví dụ tìm $x=9$



$x = 9$
 $10 > 9$, Left
 $5 < 9$, Right
 $8 < 9$, Right
 $9 = 9$, Tìm thấy

Ví dụ tìm $x = 25$



$10 < 25$, Right

$30 > 25$, Left

$25 = 25$, Tìm thấy

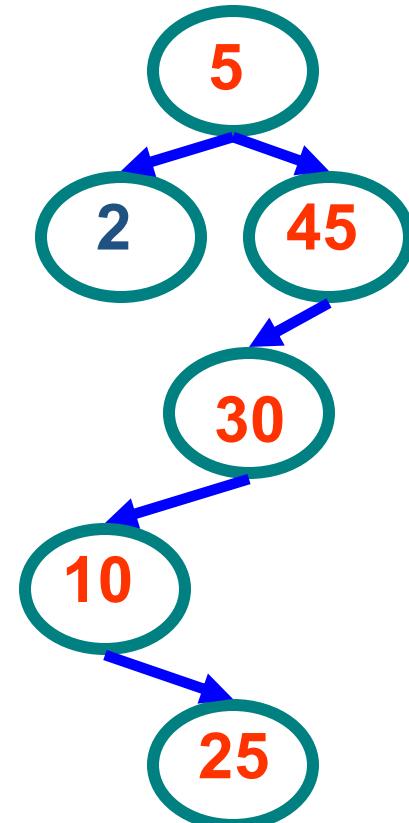
$5 < 25$, Right

$45 > 25$, Left

$30 > 25$, Left

$10 < 25$, Right

$25 = 25$, Tìm thấy



Tìm giá trị x

```
TNode* findTNodeX(TNode* root, ItemType x)
{// Dùng đệ quy
    if(!root) return NULL;
    if(root→Info == x)
        return root;
    if(root→Info > x)
        return findTNodeX(root→Left, x);
    else
        return findTNodeX(root→Right, x);
}
```

Tìm giá trị x

```
TNode* findTNodeX(TNode* root, ItemType x)
{
    //Không dùng đệ quy
    TNode* p = root;
    while( p && p→Info != x )
    {
        if( p→Info > x )          p = p→Left;
        else                      p = p→Right;
    }
    return p;
}
```

Tìm giá trị lớn nhất trên cây NPTK

TNode* **maxTNodeBSTree**(TNode* root)

{//Hàm tìm nút có giá trị lớn nhất trên cây (là nút bên phải nhất)

 TNode* p=root;

 while(p→Right != NULL)

 p = p→Right;

 return (p);

}

Duyệt cây

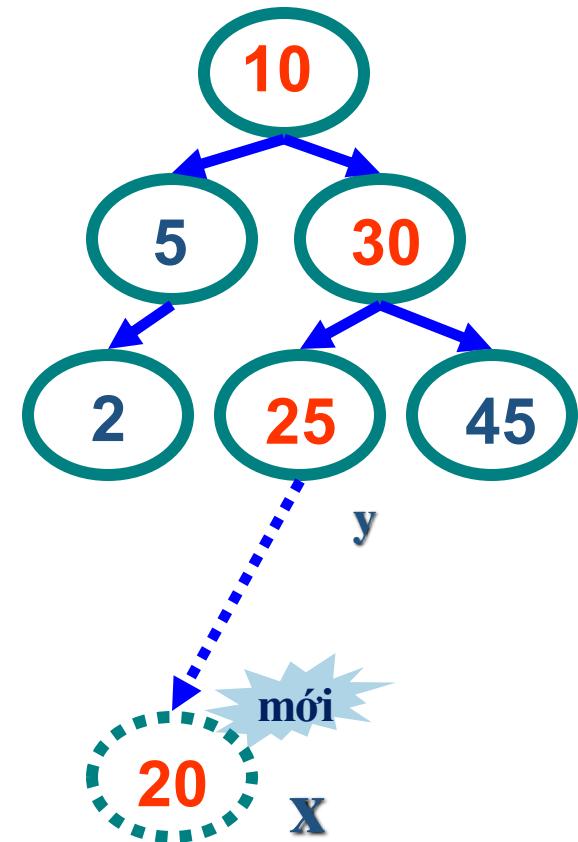
- **Lưu ý:** Cây nhị phân tìm kiếm duyệt theo traverseLNR thì thứ tự khóa tăng dần.
- **Ví dụ:** Đếm số nút lớn hơn x

Đếm số phần tử lớn hơn x

```
int countGreaterThanX(TNode* root, ItemType x)
{
    if(!root) return 0;
    int nlx = countGreaterThanX(root→Left, x);
    int nrx = countGreaterThanX(root→Right, x);
    if( root→Info > x )
        return (1 + nlx + nrx);
    return (nlx + nrx);
}
```

Thêm x vào cây

- Thực hiện tìm kiếm giá trị x
- Tìm đến cuối nút y(nếu x không tồn tại trong cây)
- Nếu $x < y$, thêm nút lá x bên trái của y
- Nếu $x > y$, thêm nút lá x bên phải của y



Thêm x vào cây

```
int insertTNode(TNode* &root, TNode* p)
{//Ham chen 1 nut vao cay NPTK
    if(p == NULL) return 0; //Thêm không thành công
    if(root == NULL) { //Cây đang rỗng
        root = p;
        return 1; //Thêm thành công
    }
    if(root->Info == p->Info)
        return 0; //Bị trùng khóa nên không thêm nữa
    if(p->Info < root->Info)
        insertTNode(root->Left, p); //thêm vào nhánh trái
    else
        insertTNode(root->Right, p); //thêm vào nhánh phải
    return 1; //Thêm thành công
}
```

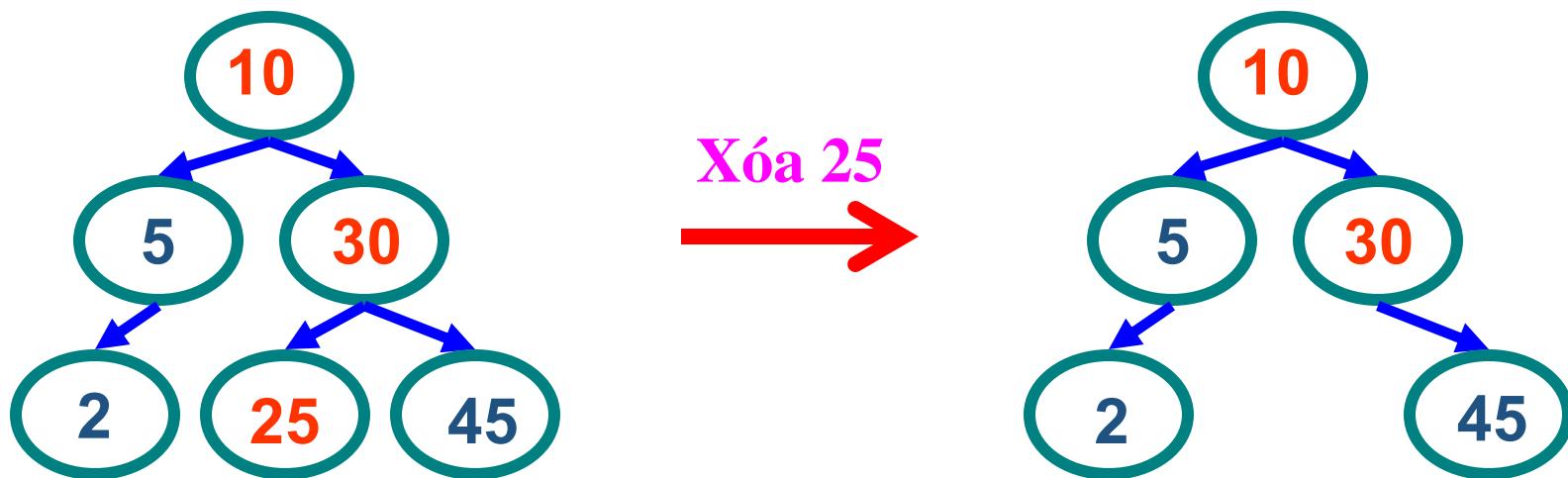
Xóa phần tử khỏi cây

Xóa nhưng phải đảm bảo vẫn là cây BST

- Thực hiện tìm nút có giá trị x.
- Nếu nút là nút lá, xóa nút.
- Ngược lại
 - Thay thế nút bằng một trong hai nút sau.
 - Y là nút lớn nhất của cây con bên trái
 - Z là nút nhỏ nhất của cây con bên phải
 - Chọn nút Y hoặc Z để thay thế.
 - Giải phóng nút có giá trị x.

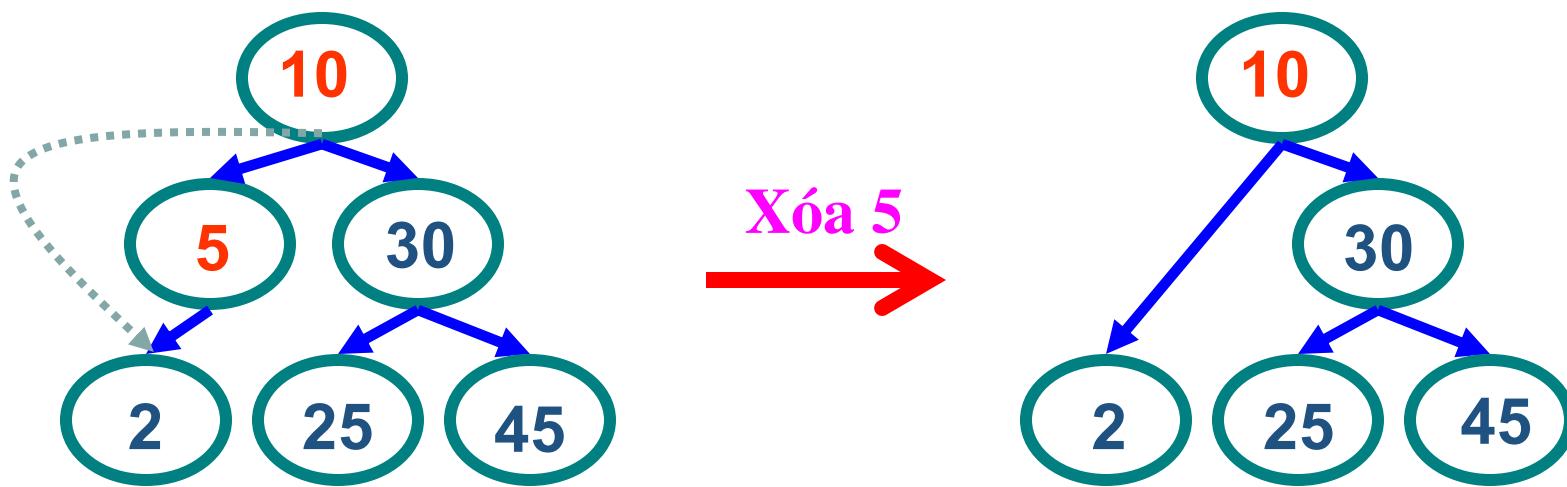
Ví dụ xóa $x = 25$

- Trường hợp 1: nút p là nút lá, xóa bình thường



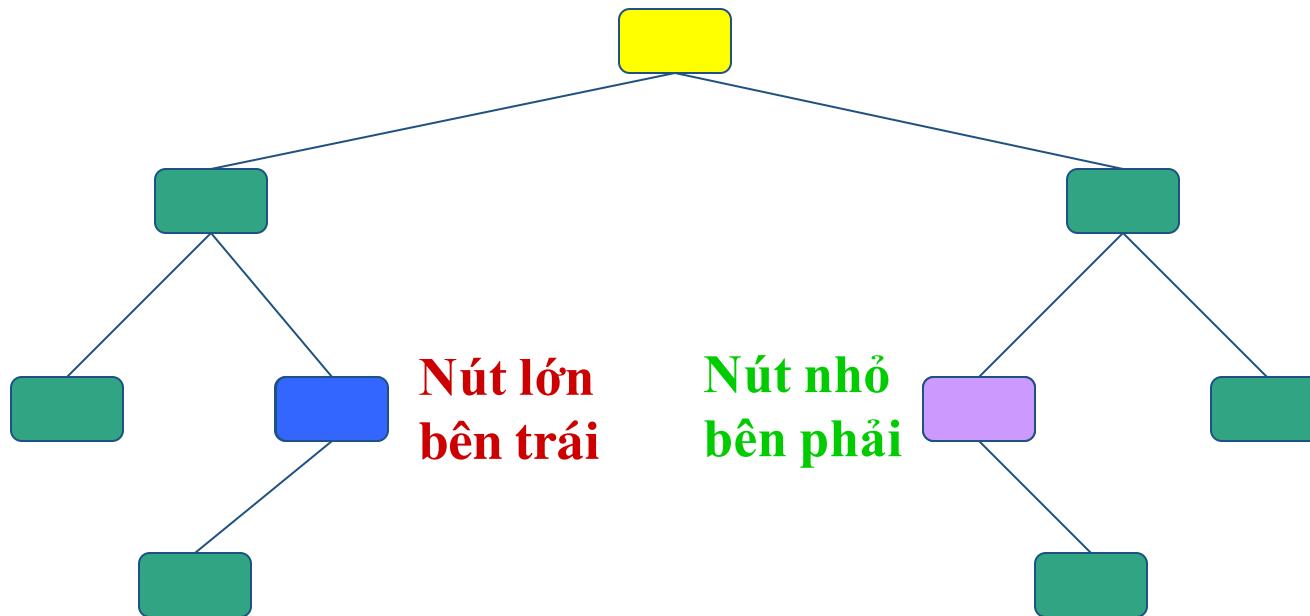
Ví dụ xóa $x = 5$

- Trường hợp 2: p chỉ có 1 cây con, cho nút cha của p trả tối nút con duy nhất của nó, rồi hủy p



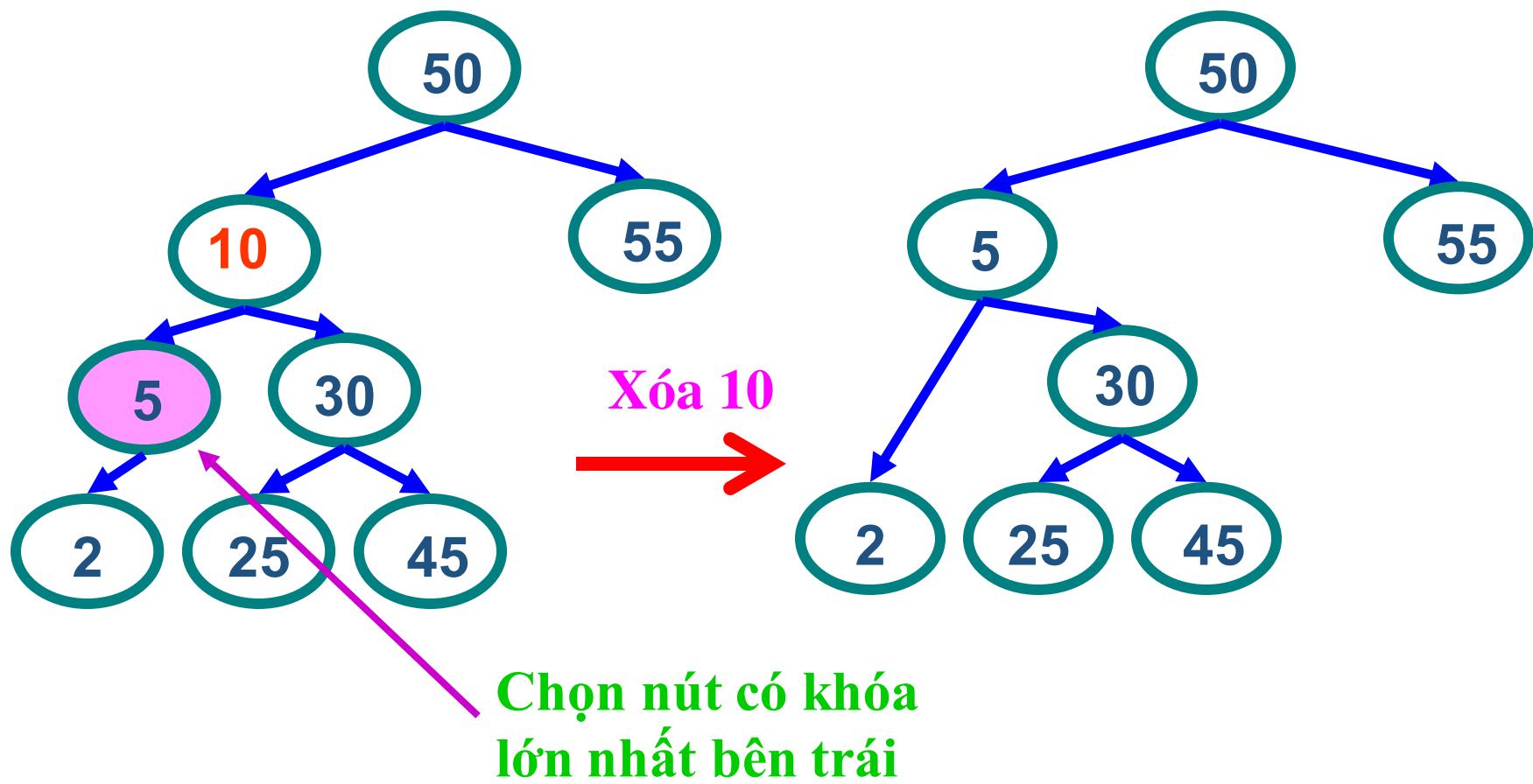
Ví dụ

- **Trường hợp 3:** nút p có 2 cây con, chọn nút thay thế theo 1 trong 2 cách như sau:
 - Nút lớn nhất trong cây con bên trái
 - Nút nhỏ nhất trong cây con bên phải



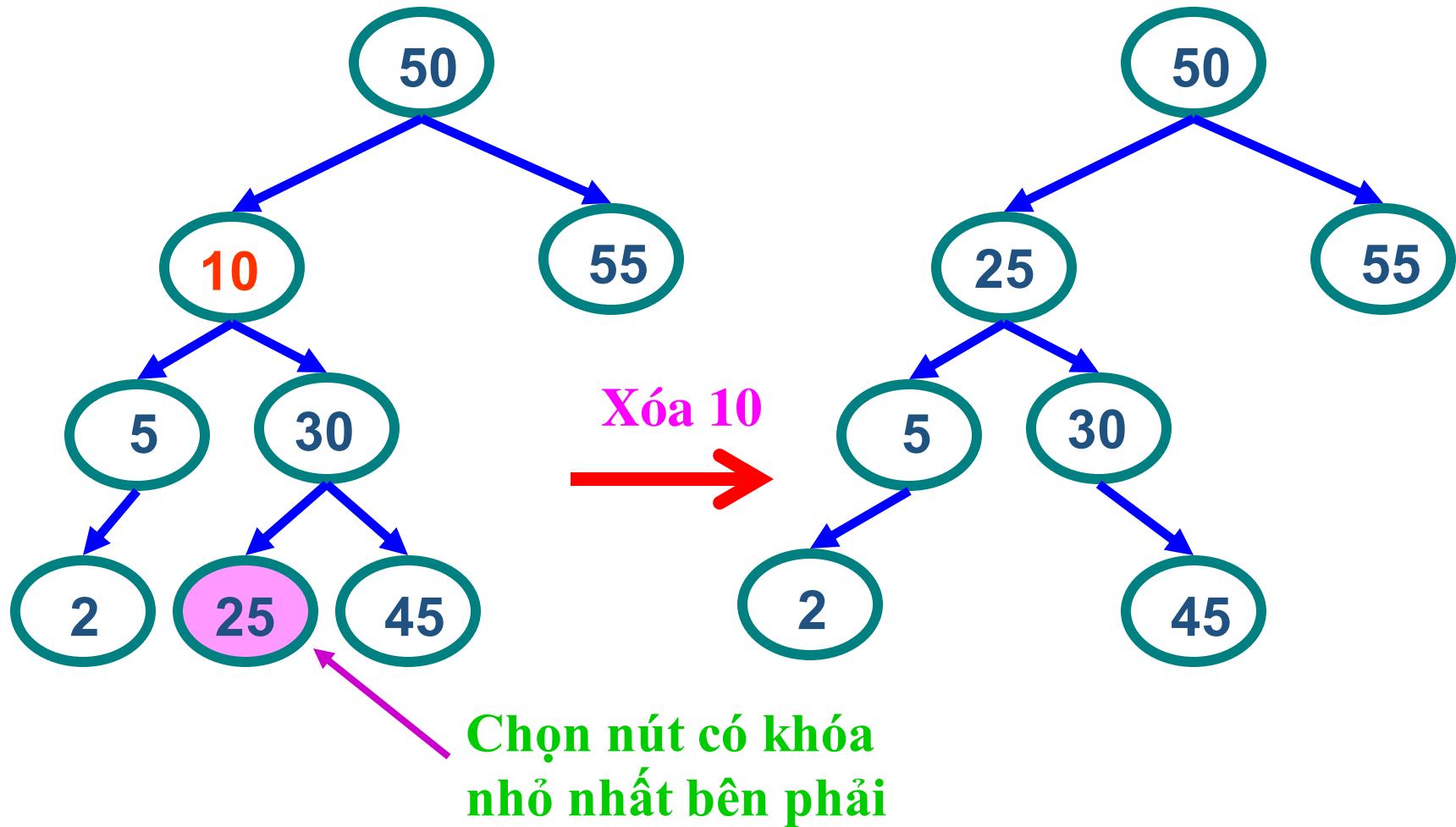
Ví dụ

- Xóa nút 10: cách 1

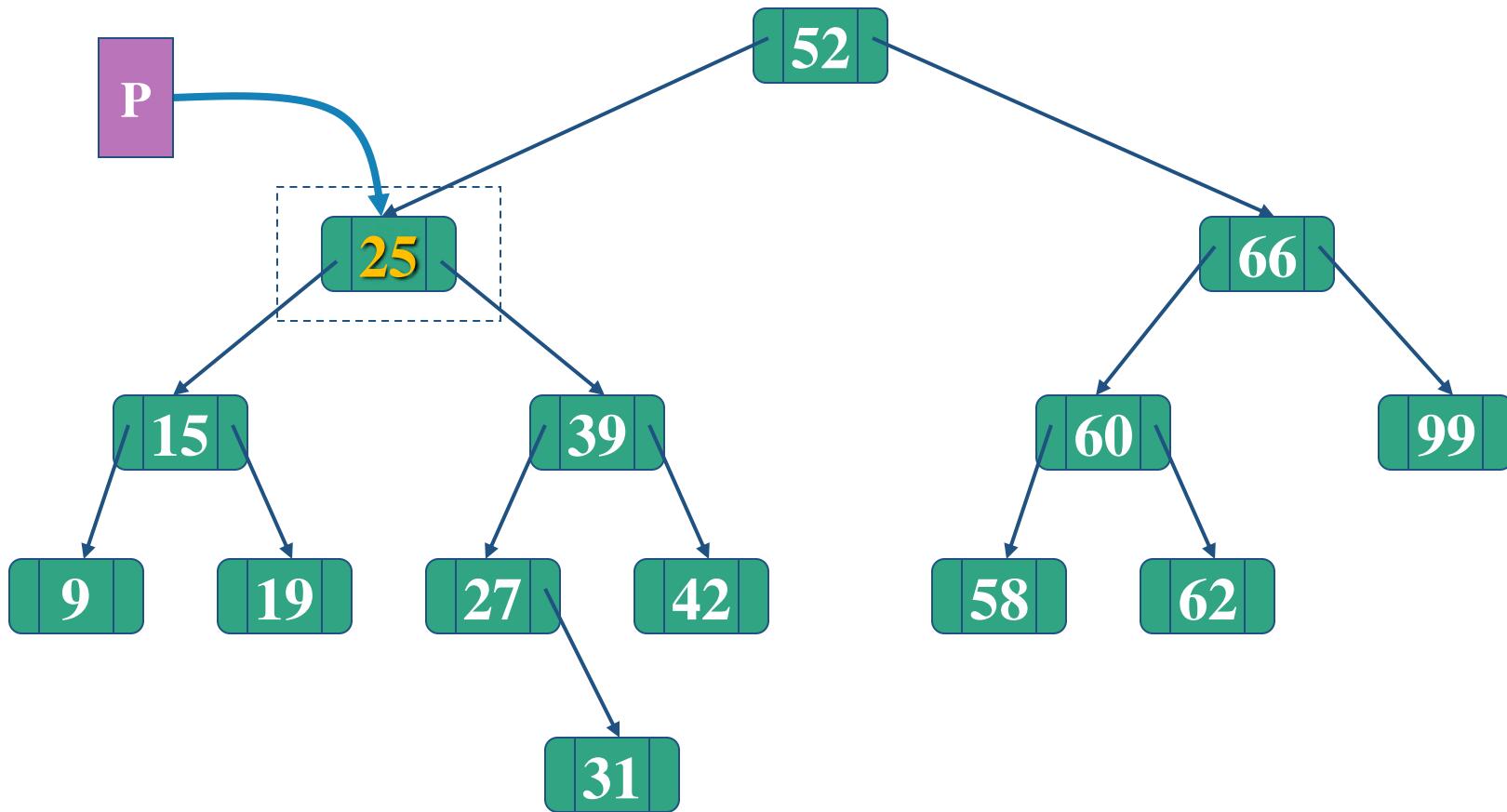


Ví dụ

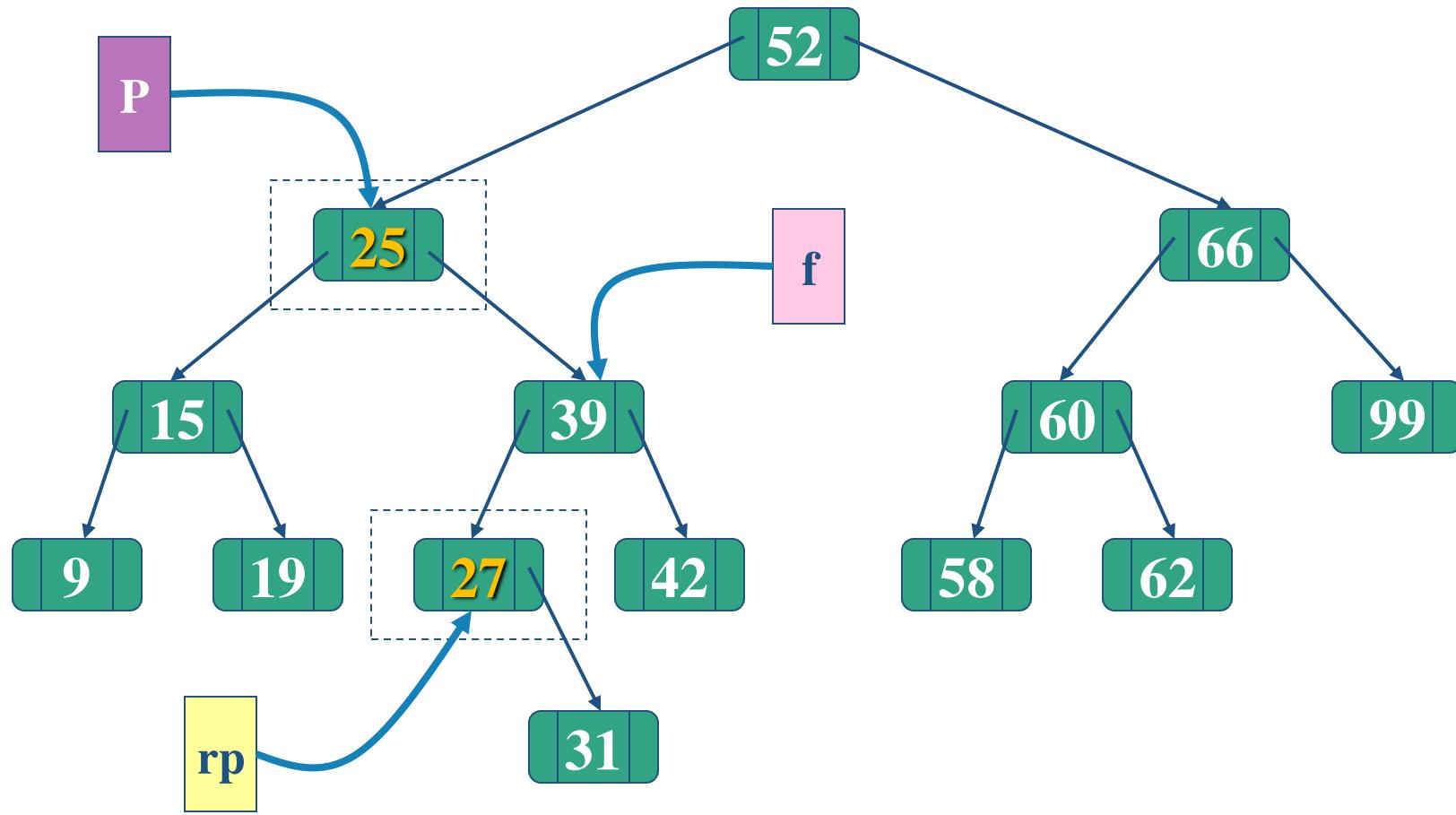
- Xóa nút 10: cách 2



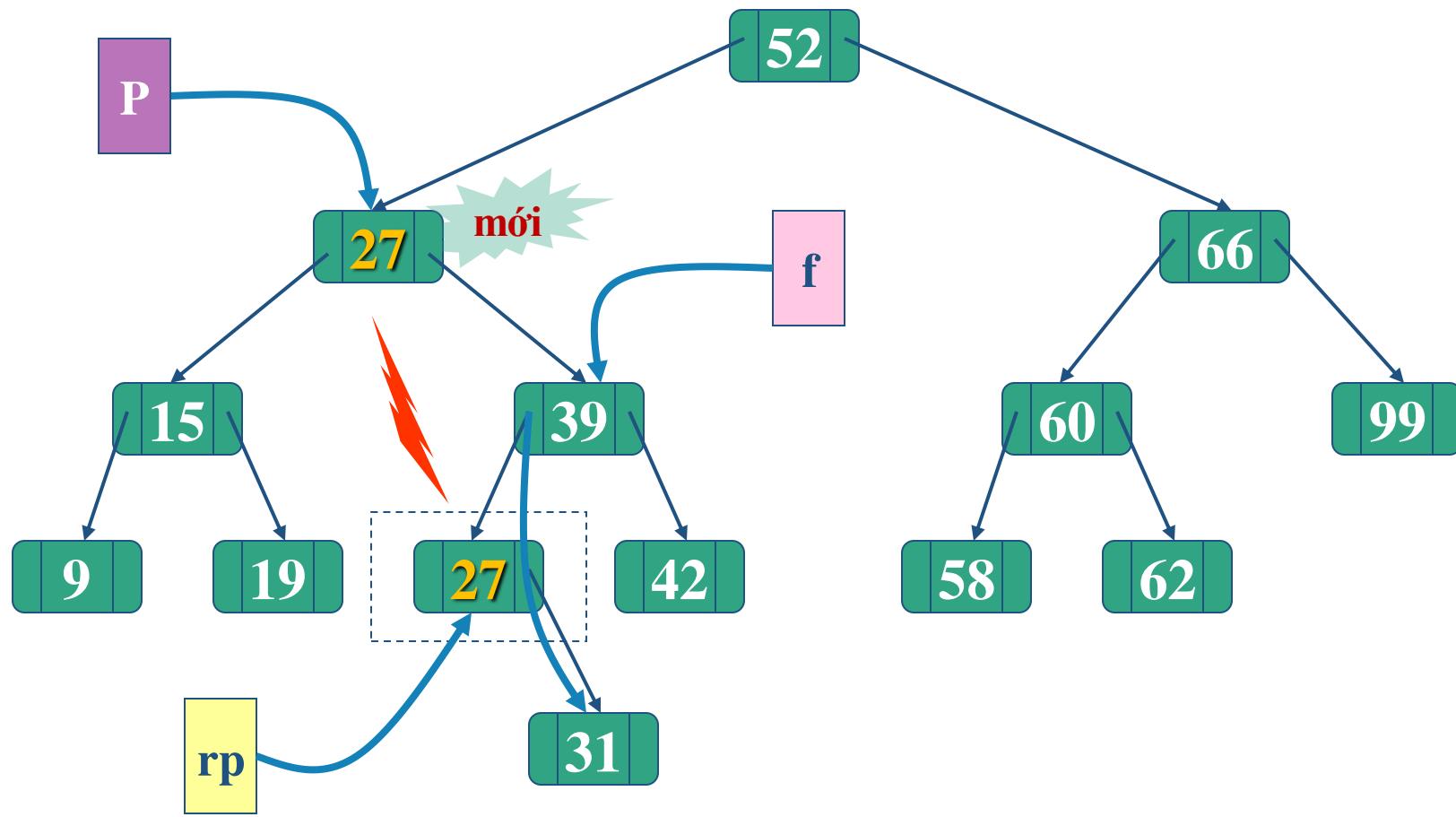
Ví dụ xóa $x = 25$



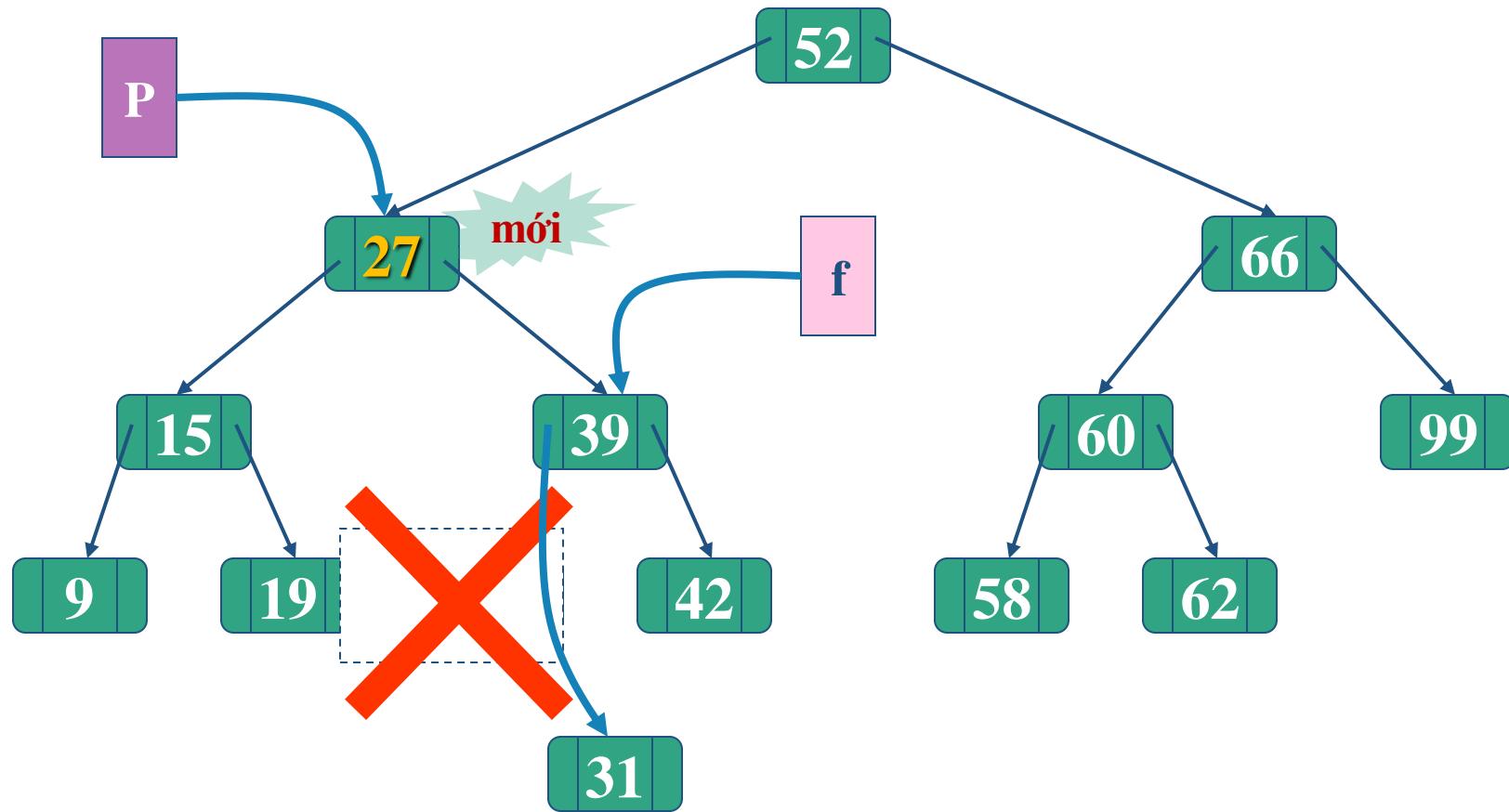
Ví dụ xóa $x = 25$



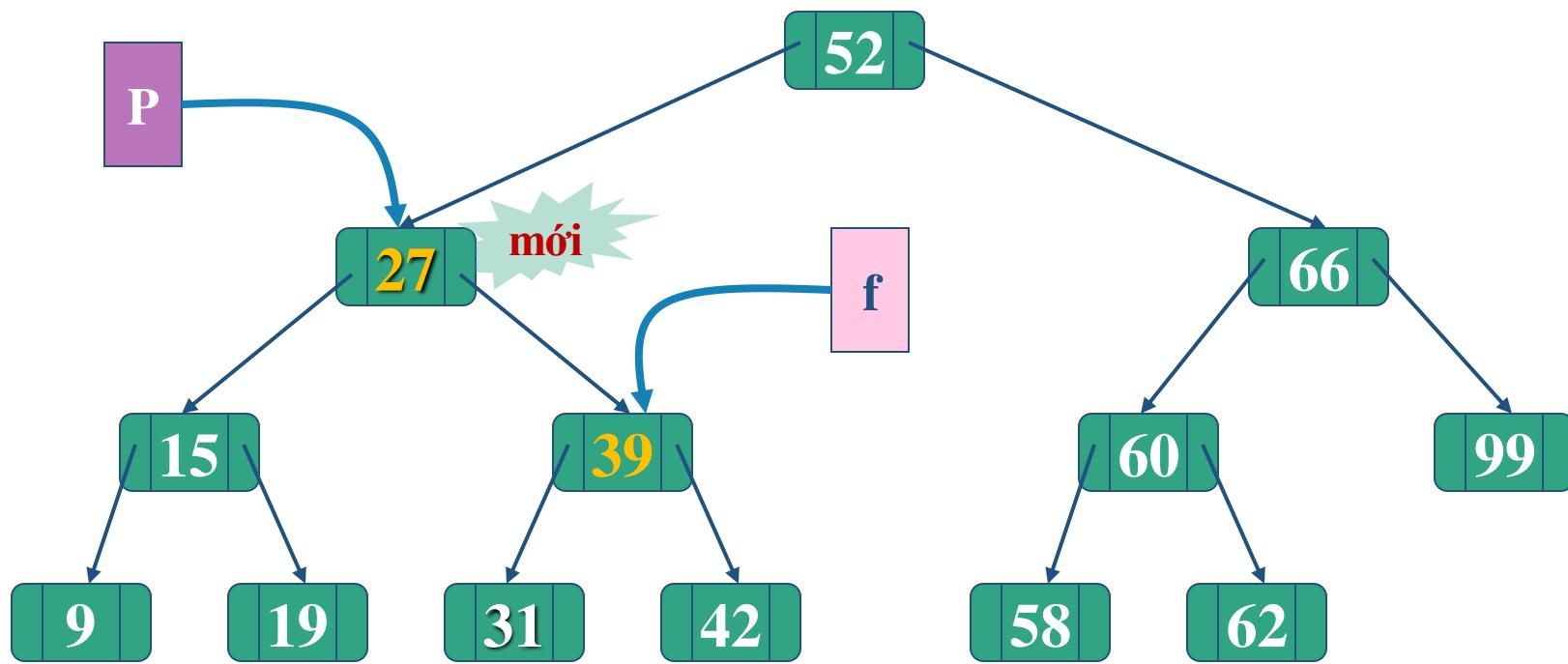
Ví dụ xóa $x = 25$



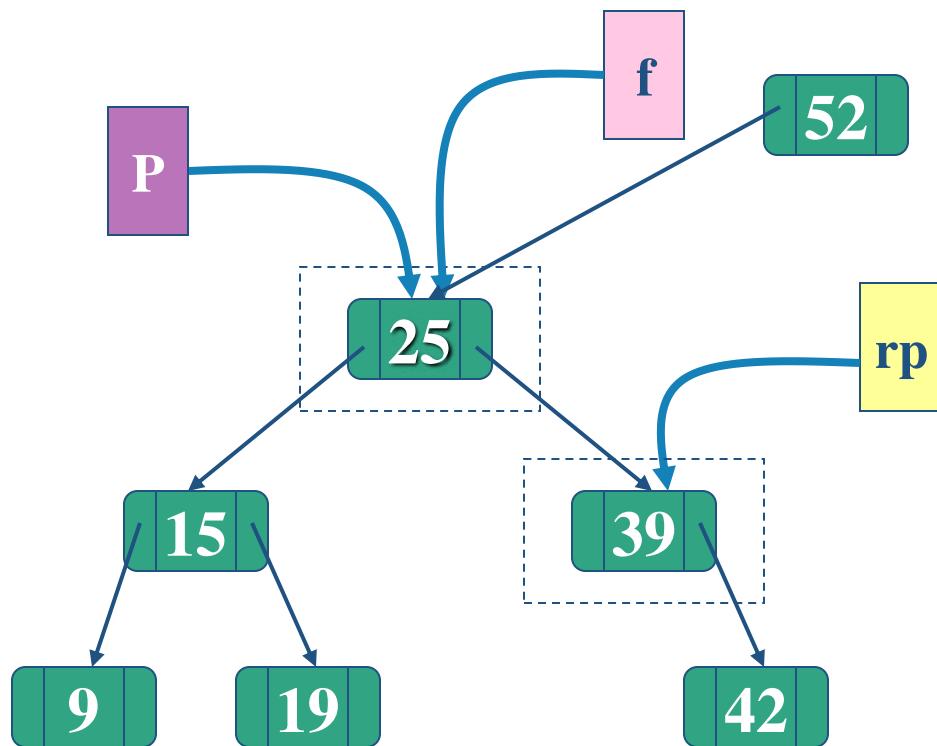
Ví dụ xóa $x = 25$



Ví dụ xóa $x = 25$

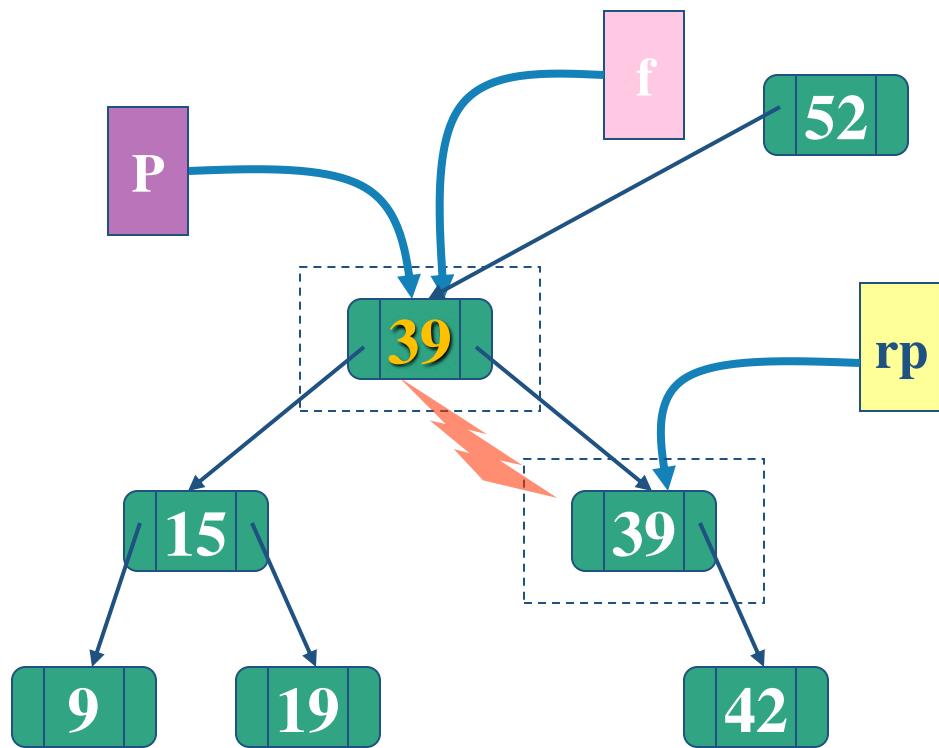


Ví dụ xóa $x = 25$



Trường hợp đặc biệt:
f == p
Nút thế mạng rp là
nút con phải của nút
p cần xóa

Ví dụ xóa $x = 25$

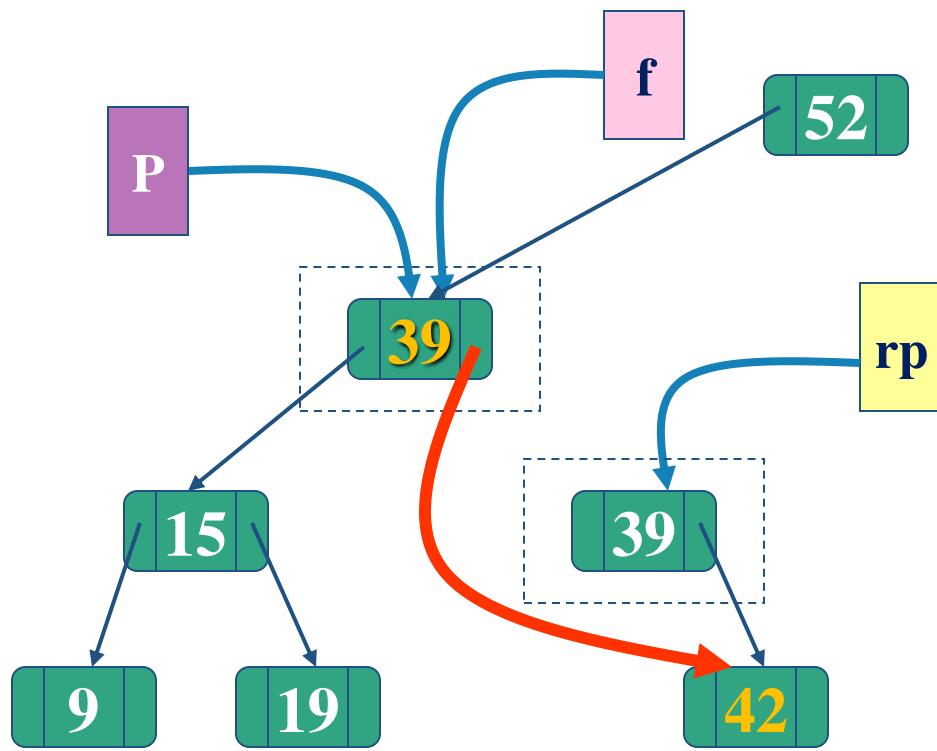


Trường hợp đặc biệt:
f == p

Nút thế mạng rp là
nút con phải của nút
p cần xóa

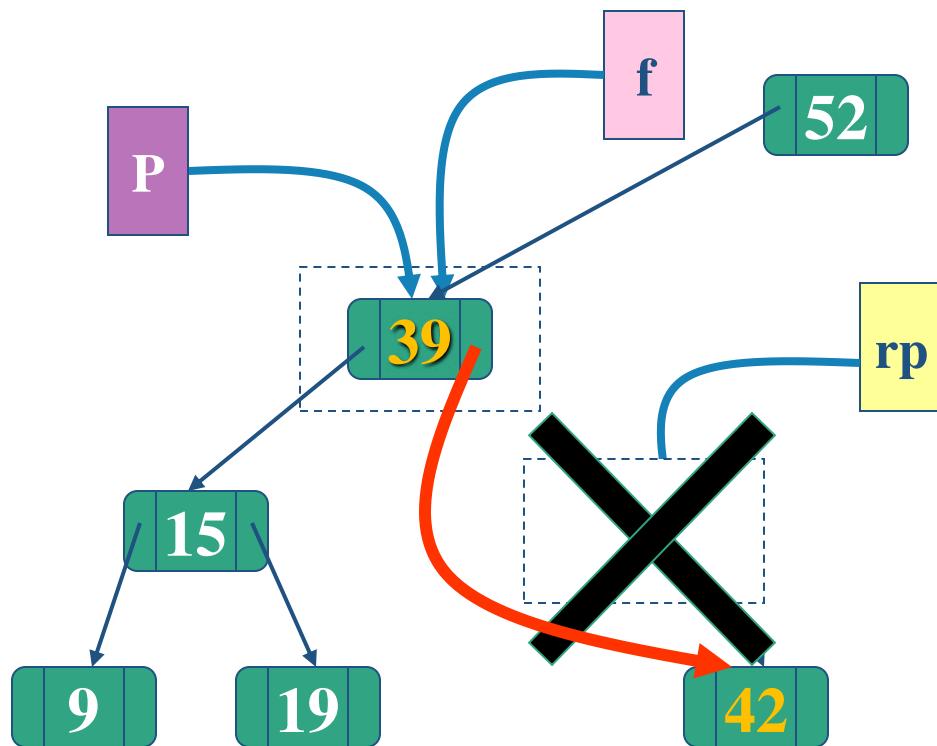
- Đưa giá trị của nút
rp lên nút p

Ví dụ xóa $x = 25$



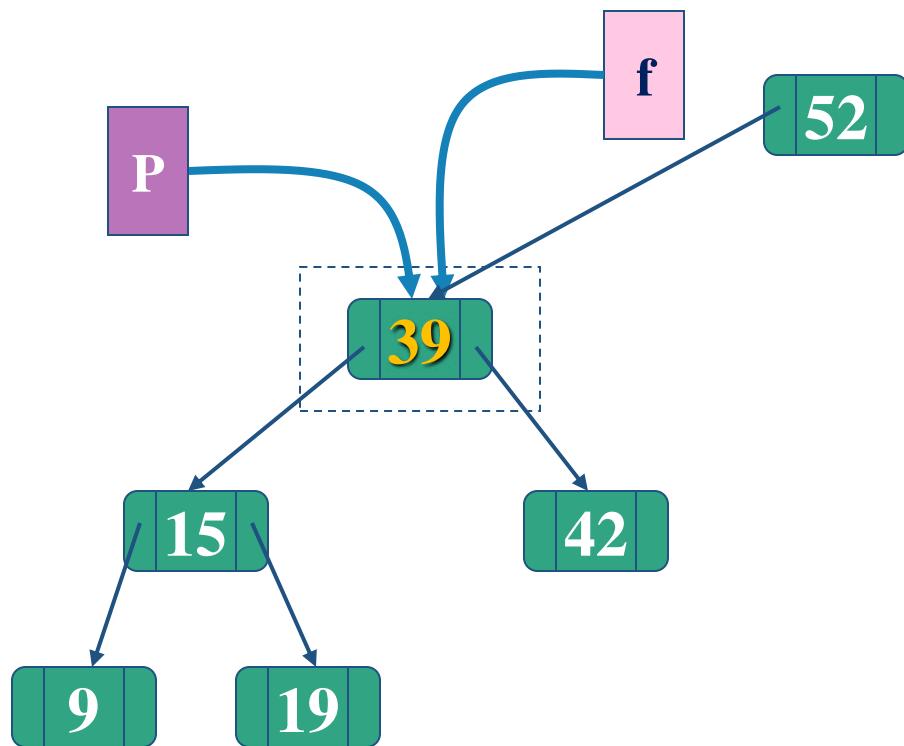
Trường hợp đặc biệt:
f == p
Nút thế mạng rp là
nút con phải của nút
p cần xóa
- Chuyển liên kết
phải của p đến liên
kết phải của rp

Ví dụ xóa $x = 25$



Trường hợp đặc biệt:
f == p
Nút thế mạng rp là
nút con phải của nút
p cần xóa
- xóa nút rp

Ví dụ xóa $x = 25$



Trường hợp đặc biệt:

f == p

Nút thế mạng rp là nút con phải của nút p cần xóa

- Sau khi xóa

Giải thuật

Nếu $T = \text{NULL}$ \Rightarrow thoát

Nếu $T \rightarrow \text{Info} > x \Rightarrow \text{Xoa}(T \rightarrow \text{Left}, x)$

Nếu $T \rightarrow \text{Info} < x \Rightarrow \text{Xoa}(T \rightarrow \text{Right}, x)$

Nếu $T \rightarrow \text{Info} = x$

$p = T$

Nếu T có 1 nút con thì T trỏ đến nút con đó

Giải thuật

Ngược lại có 2 con

Gọi $f = p$ và $rp = p \rightarrow Right$;

Tìm nút rp : $rp \rightarrow Left = NULL$ và nút f là nút cha nút rp

Thay đổi giá trị nội dung của p và rp

Nếu $f = p$ (trường hợp đặc biệt) thì:

$f \rightarrow Right = rp \rightarrow Right$;

Ngược lại:

$f \rightarrow Left = rp \rightarrow Right$;

Xóa rp ; // xóa nút thế mạng rp

Xóa nút có giá trị là x

```
int deleteTNode(TNode* &root, ItemType x)
{
    if(!root) return 0;
    if(root->Info > x) //tìm bên trái
        return deleteTNode(root->Left, x);
    else if(root->Info < x) //tìm bên phải
        return deleteTNode(root->Right, x);
    else
    {
        TNode* p = root;
```

Xóa nút có giá trị là x

```
if(!root→Left)//khi cay con khong co nhanh trai  
    root = root→Right;  
else if(!root→Right)//khi cay con khong co nhanh phai  
    root = root→Left;  
else  
{//khi cay con co ca 2 nhanh, chon min cua nhanh phai de the mang  
    TNode* p = root;  
    TNode* rp = findTNodeReplace(p);  
    deleteTNode(rp);  
}  
}  
}
```

Tìm nút thê mạng cho nút bị xóa

TNode* **findTNodeReplace**(TNode*&p)

{*//Ham tim nut rp nho nhattren cay con phai de the mang cho nut p*

 TNode* f = p,* rp = p→Right;

 while(rp→Left != NULL) {

 f = rp; //Luu cha cua rp

 rp = rp→Left; //rp qua ben trai

}

 p→Info = rp→Info; *//tim duoc phan tu the mang cho p la rp*

 if(f == p) *//neu cha cua rp la p*

 f→Right = rp→Right;

 else

 f→Left = rp→Right;

 return rp; *//Tra ve nut rp la nut the mang cho p*

}

Bài tập

- **Tìm phần tử max âm trên cây.**
- **Đếm có bao nhiêu phần tử chẵn trên cây.**

Thank for you attention!



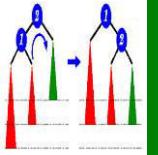
See you next week!

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT



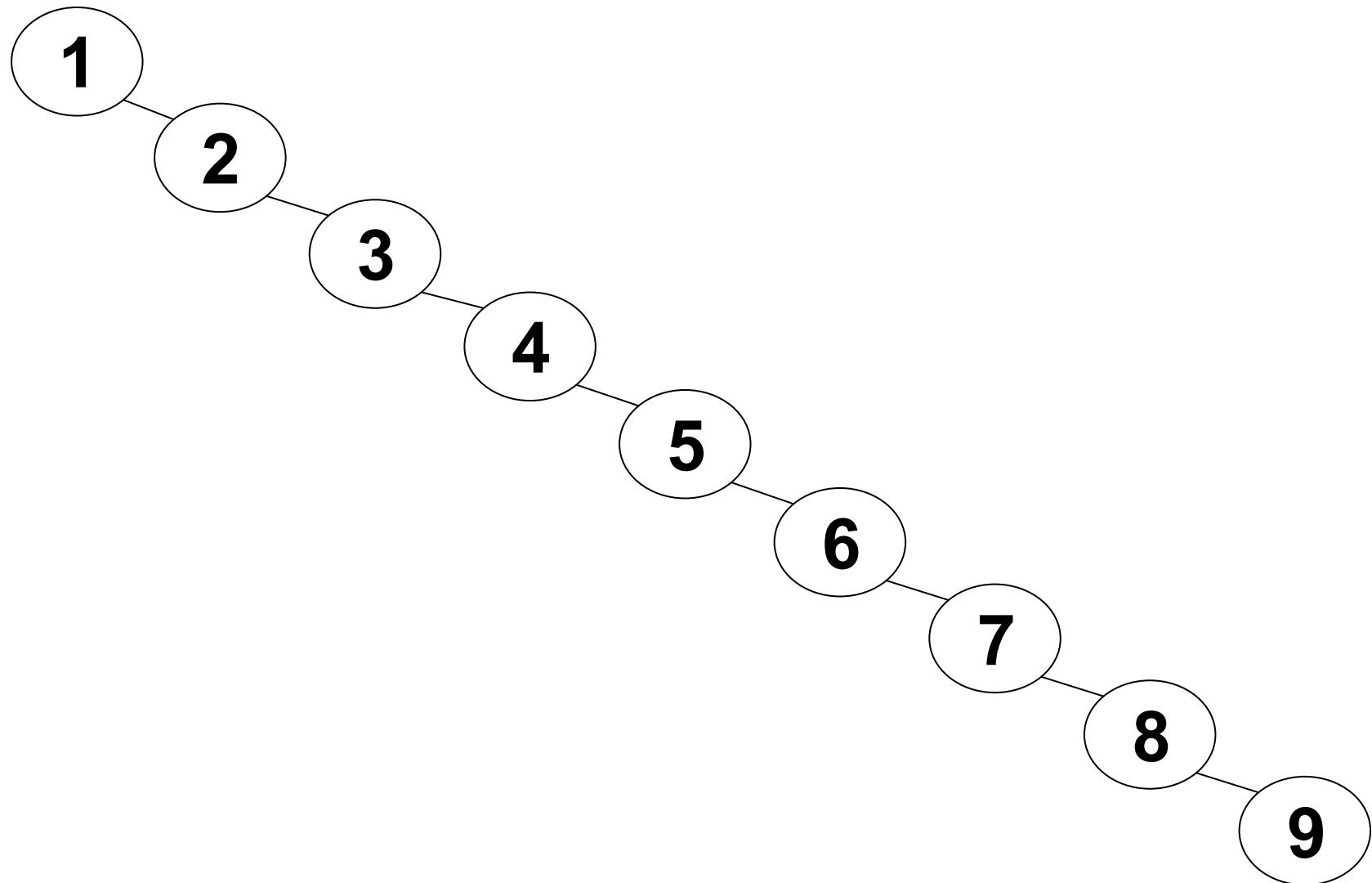
GV : ThS. Trần Văn Thọ

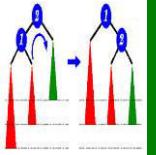
E-mail : thotv@hufi.edu.vn



Đánh giá tìm kiếm

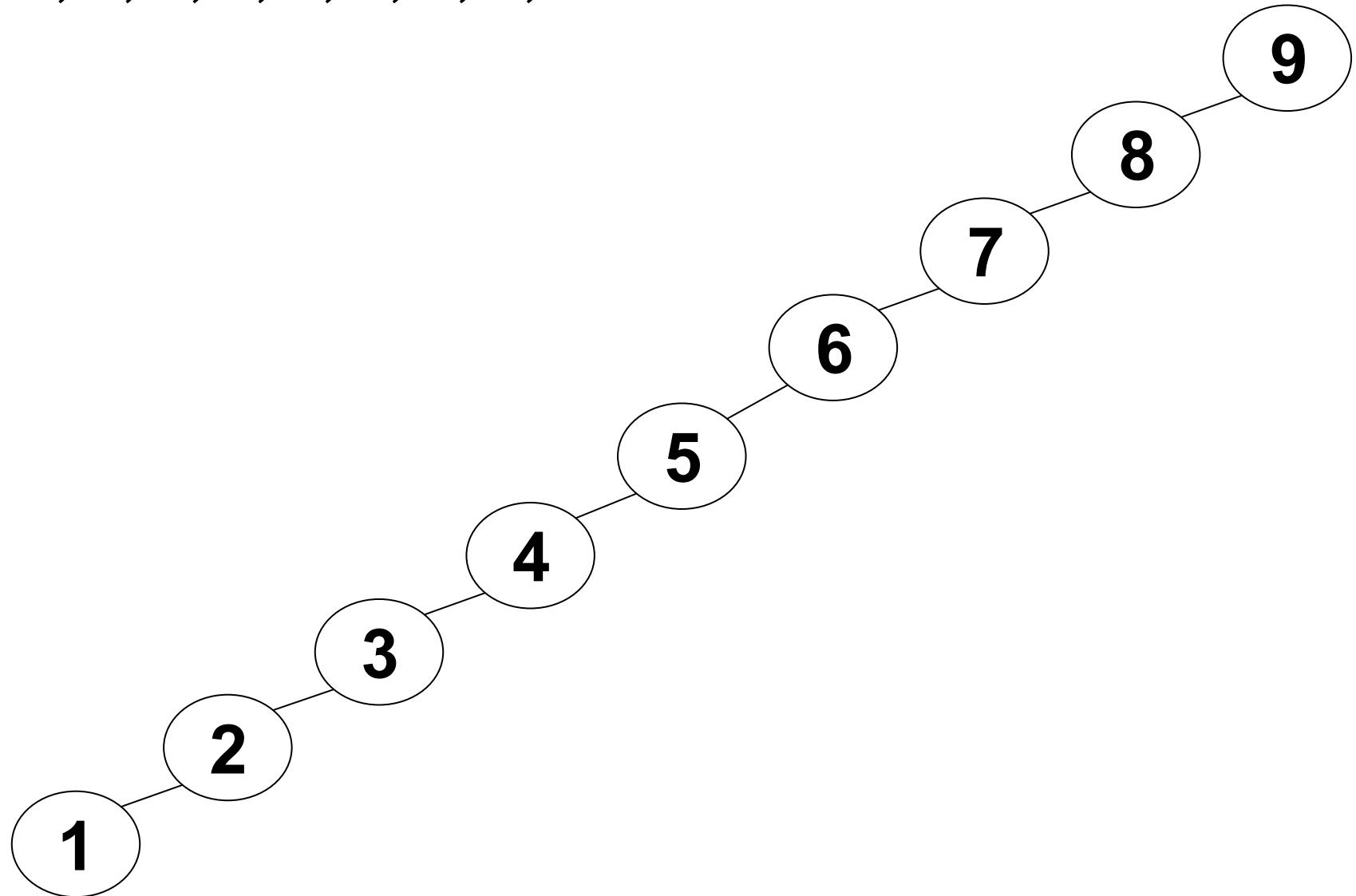
- ❖ 1, 2, 3, 4, 5, 6, 7, 8, 9

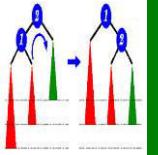




Đánh giá tìm kiếm

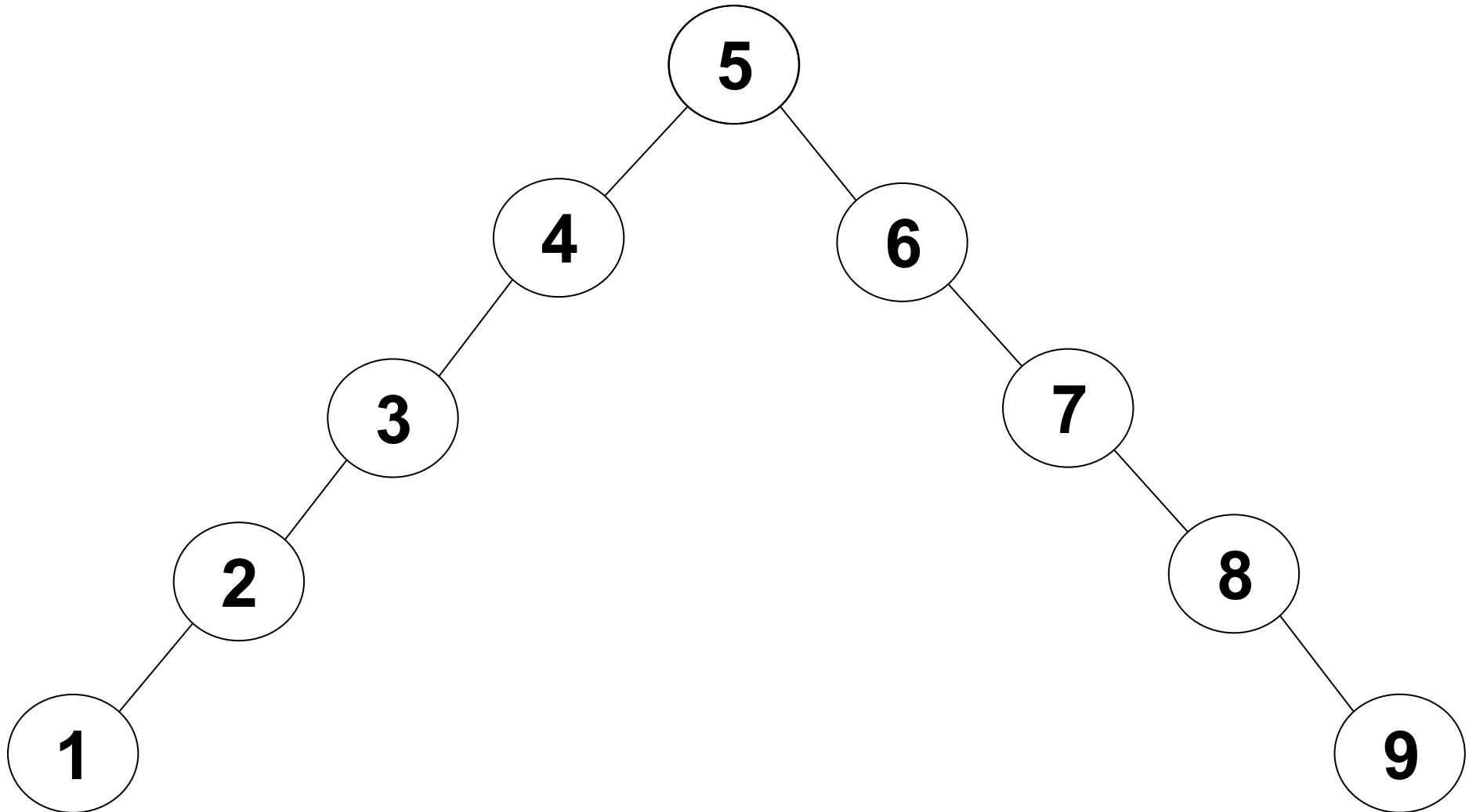
- ❖ 9, 8, 7, 6, 5, 4, 3, 2, 1





Đánh giá tìm kiếm

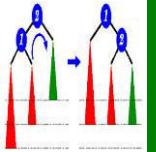
- ❖ 5, 4, 3, 2, 1, 6, 7, 8, 9



Môn: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

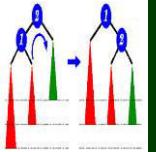


CHƯƠNG V: CÂY NHỊ PHÂN TÌM KIẾM CÂN BẰNG



AVL Tree - Giới thiệu

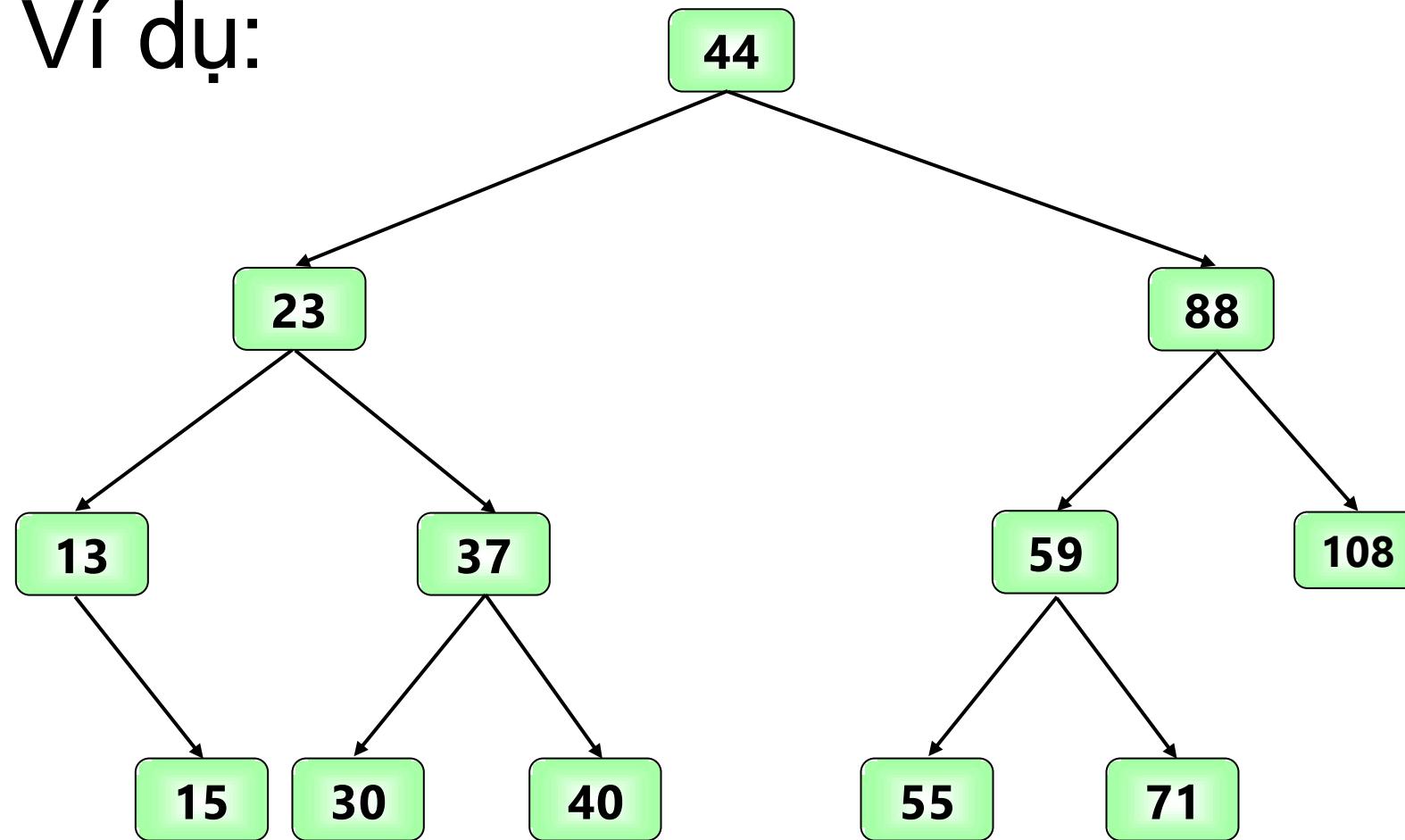
- ❖ Phương pháp thêm trên Cây NPTK có thể có những biến dạng mất cân đối nghiêm trọng
 - **Chi phí cho việc tìm kiếm** trong trường hợp xấu nhất đạt tới **n**
 - VD: 1 triệu nút \Rightarrow chi phí tìm kiếm = 1.000.000 nút
- ❖ Nếu có một cây tìm kiếm nhị phân cân bằng hoàn toàn, **chi phí cho việc tìm kiếm** chỉ xấp xỉ **$\log_2 n$**
 - VD: 1 triệu nút \Rightarrow chi phí tìm kiếm = $\log_2 1.000.000 \approx 20$ nút
- ❖ G.M. Adelson - Velsky và E.M. Landis đã đề xuất một tiêu chuẩn cân bằng (gọi là cân bằng **AVL**)
 - Cây AVL có chiều cao **$O(\log_2(n))$**

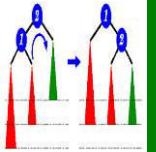


AVL Tree - Định nghĩa

- Cây nhị phân tìm kiếm cân bằng là cây mà **tại mỗi nút** của nó **độ cao** của cây con trái và của cây con phải **chênh lệch nhau không quá một**.

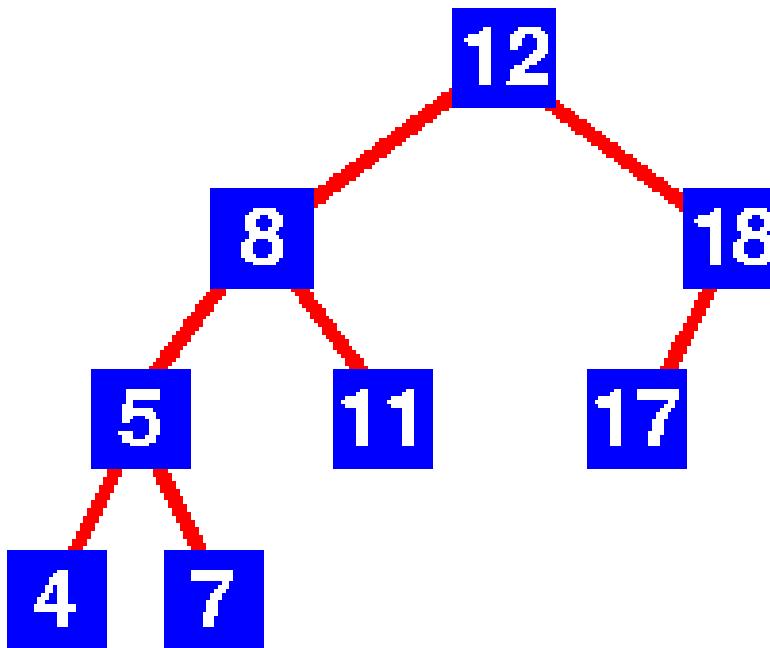
Ví dụ:



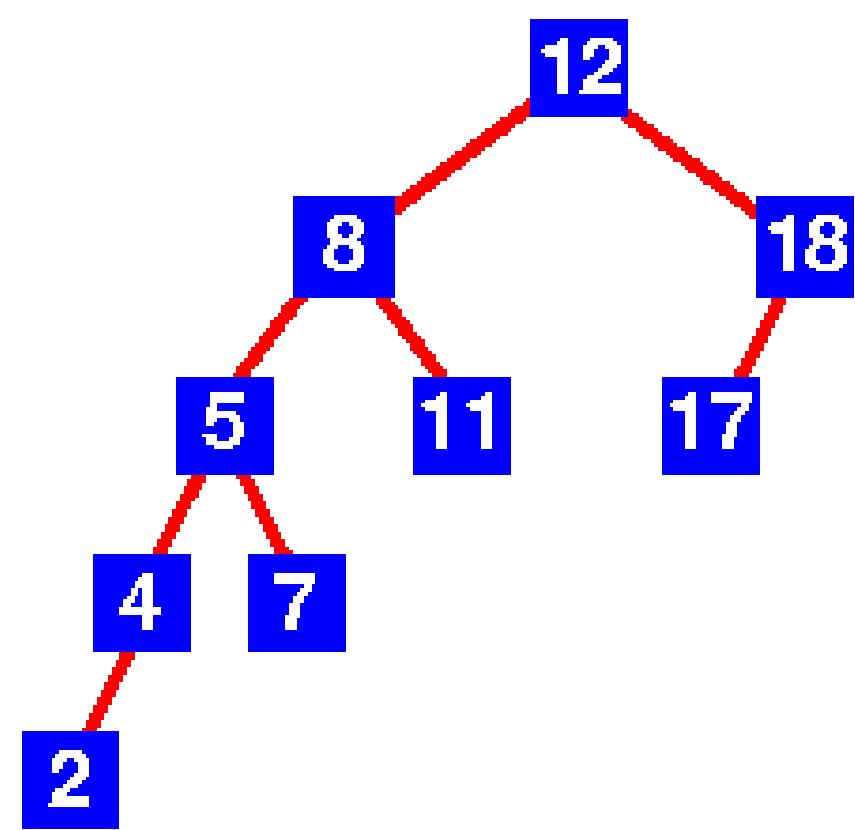


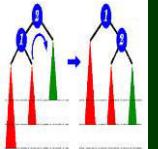
AVL Tree – Ví dụ

AVL Tree ?



AVL Tree ?





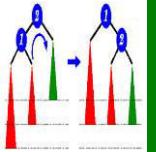
Tổ chức dữ liệu

❖ Chỉ số cân bằng của một nút:

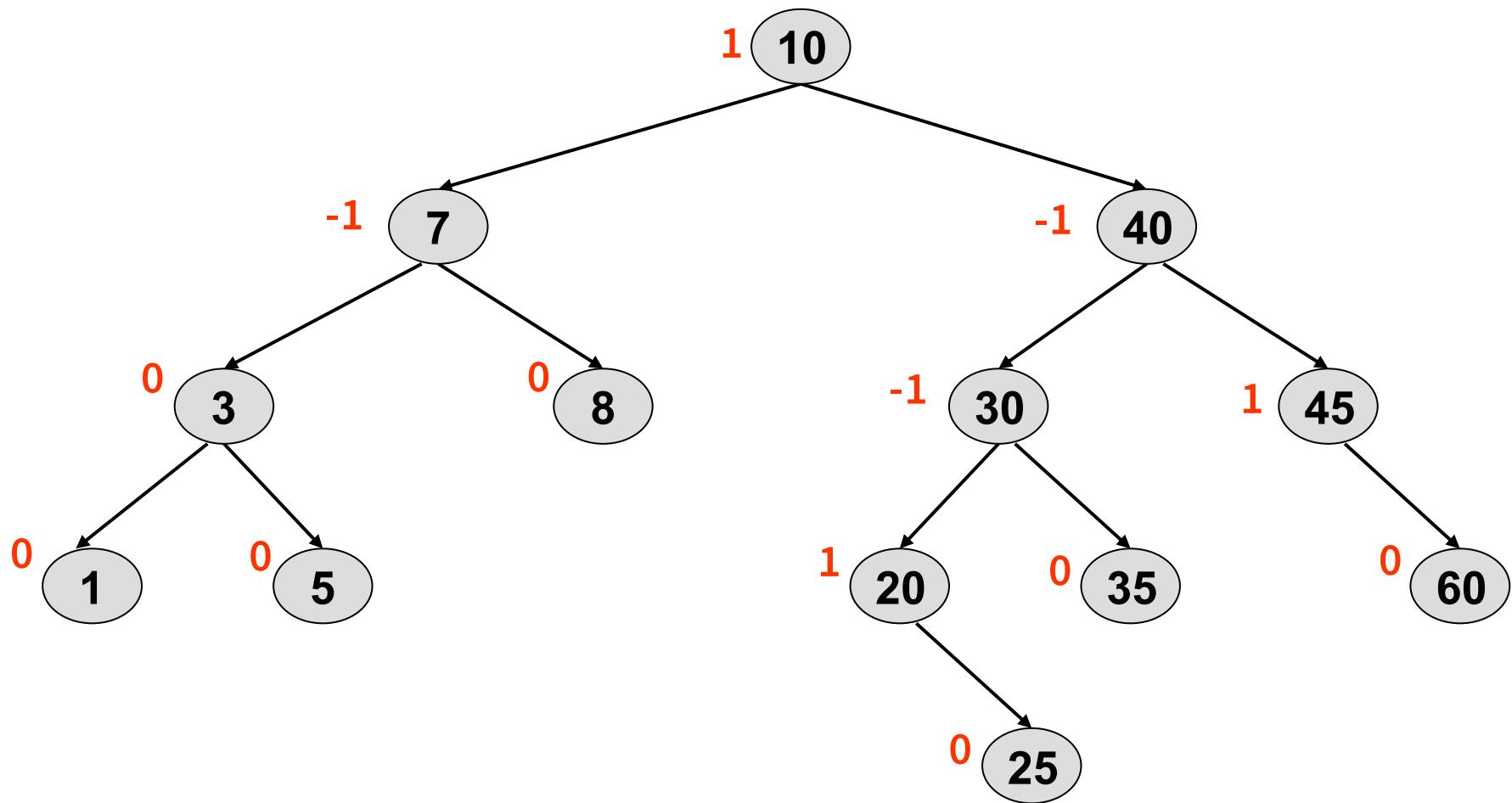
- **Định nghĩa:** Chỉ số cân bằng của một nút là **hiệu** của chiều cao cây con phải và cây con trái của nó.
- Đối với một cây cân bằng, chỉ số cân bằng (CSCB) của mỗi nút chỉ có thể mang một trong ba giá trị: -1, 0, và 1:

❖ Các giá trị hợp lệ:

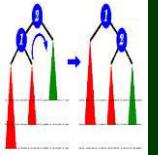
- **CSCB(p)=0** \Leftrightarrow Độ cao cây trái (p) = Độ cao cây phải (p)
- **CSCB(p)=1** \Leftrightarrow Độ cao cây trái (p) < Độ cao cây phải (p)
- **CSCB(p)=-1** \Leftrightarrow Độ cao cây trái (p) > Độ cao cây phải (p)



Ví dụ - Chỉ số cân bằng của nút

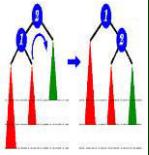


- Hãy cho biết chỉ số cân bằng của mỗi nút trên cây?
- Đây có phải là cây AVL không?



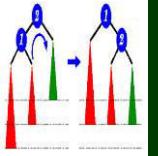
Tổ chức dữ liệu (tt)

```
#define LH -1 //cây con trái cao hơn
#define EH 0 //cây con trái bằng cây con phải
#define RH 1 //cây con phải cao hơn
typedef <Kiểu dữ liệu của nút> ItemType;
struct AVLNode
{
    int balFactor; //chỉ số cân bằng của cây
    ItemType Info;
    AVLNode* Left;
    AVLNode* Right;
};
struct AVLTree
{
    AVLNode* Root;
};
```



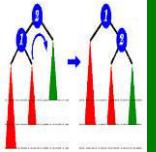
Cấp phát nút mới cho cây AVL

```
AVLNode* createAVLNode(ItemType x) {  
    AVLNode* p = new AVLNode;  
    if(p == NULL) {  
        printf("\nKhong du bo nho cap phat nut moi!");  
        getch();  
        return NULL;  
    }  
    p->balFactor = 0;  
    p->Info = x;  
    p->Left = NULL;  
    p->Right = NULL;  
    return p;  
}
```



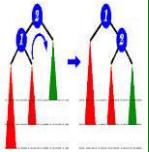
Khởi tạo cây AVL

```
void initAVLTree(AVLTree &avl)
{
    avl.Root = NULL;
}
```



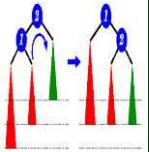
AVL Tree – Biểu diễn

- ❖ Các thao tác đặc trưng của cây AVL:
 - **Thêm** một phần tử vào cây AVL
 - **Xóa** một phần tử trên cây AVL
 - **Cân bằng lại** một cây vừa bị mất cân bằng (**Rotation**)
- ❖ Trường hợp **thêm** một phần tử trên cây AVL được thực hiện giống như thêm trên cây NPTK, tuy nhiên sau khi thêm phải cân bằng lại cây.
- ❖ Trường hợp **xóa** một phần tử trên cây AVL được thực hiện giống như hủy trên cây NPTK và cũng phải cân bằng lại cây.
- ❖ Việc **cân bằng lại** một cây sẽ phải thực hiện sao cho chỉ ảnh hưởng tối thiểu đến cây nhằm giảm thiểu chi phí cân bằng



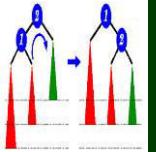
Các trường hợp mất cân bằng

- ❖ Không khảo sát tính cân bằng của 1 cây nhị phân bất kỳ mà chỉ quan tâm đến khả năng mất cân bằng xảy ra khi **thêm hoặc xóa** một nút trên cây AVL.
- ❖ Các trường hợp mất cân bằng:
 - Sau khi thêm (xóa) cây con **trái lệch trái** (left of left)
 - Sau khi thêm (xóa) cây con **trái lệch phải** (right of left)
 - Sau khi thêm (xóa) cây con **phải lệch phải** (right of right)
 - Sau khi thêm (xóa) cây con **phải lệch trái** (left of right)



Các thao tác trên cây cân bằng

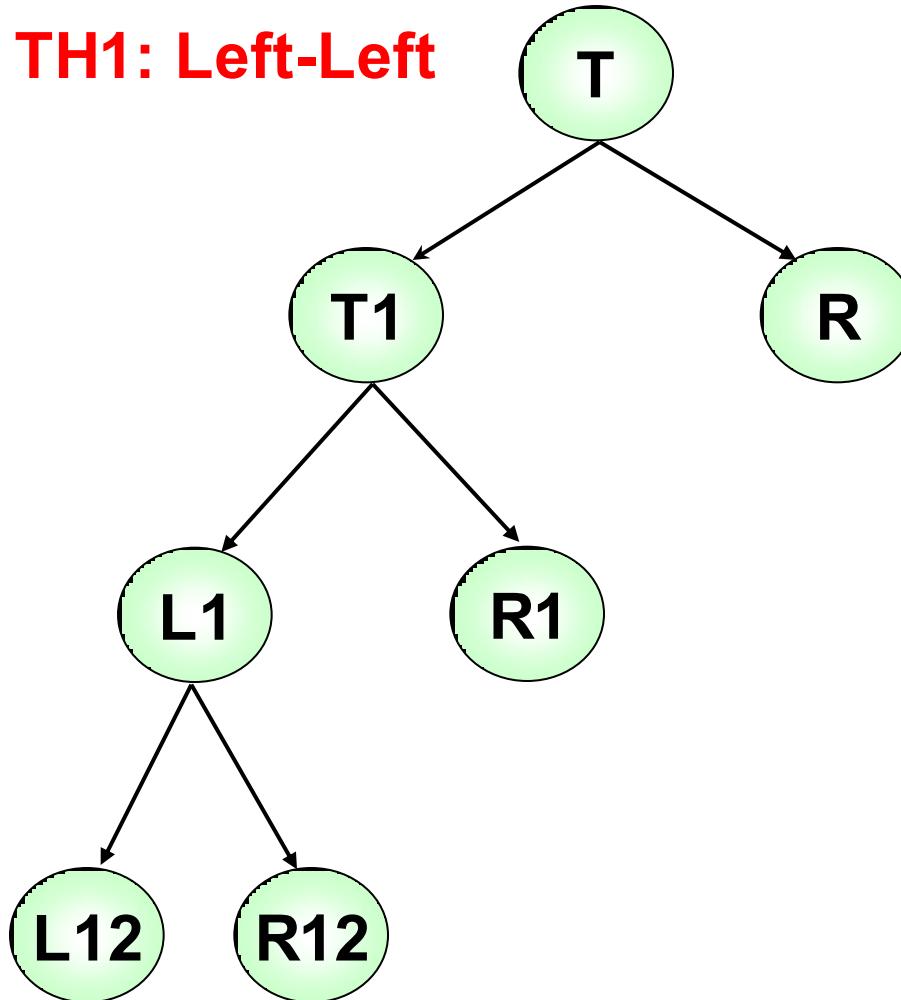
- ❖ Cây có khả năng mất cân bằng khi thay đổi chiều cao:
 - Thêm bên trái \rightarrow Lệch nhánh trái
 - Thêm bên phải \rightarrow Lệch nhánh phải
 - Hủy bên phải \rightarrow Lệch nhánh trái
 - Hủy bên trái \rightarrow Lệch nhánh phải
- ❖ **Cân bằng lại cây:** tìm cách bố trí lại cây sao cho chiều cao 2 cây con cân đối:
 - Kéo nhánh cao bù cho nhánh thấp
 - Phải bảo đảm cây vẫn là cây AVL.



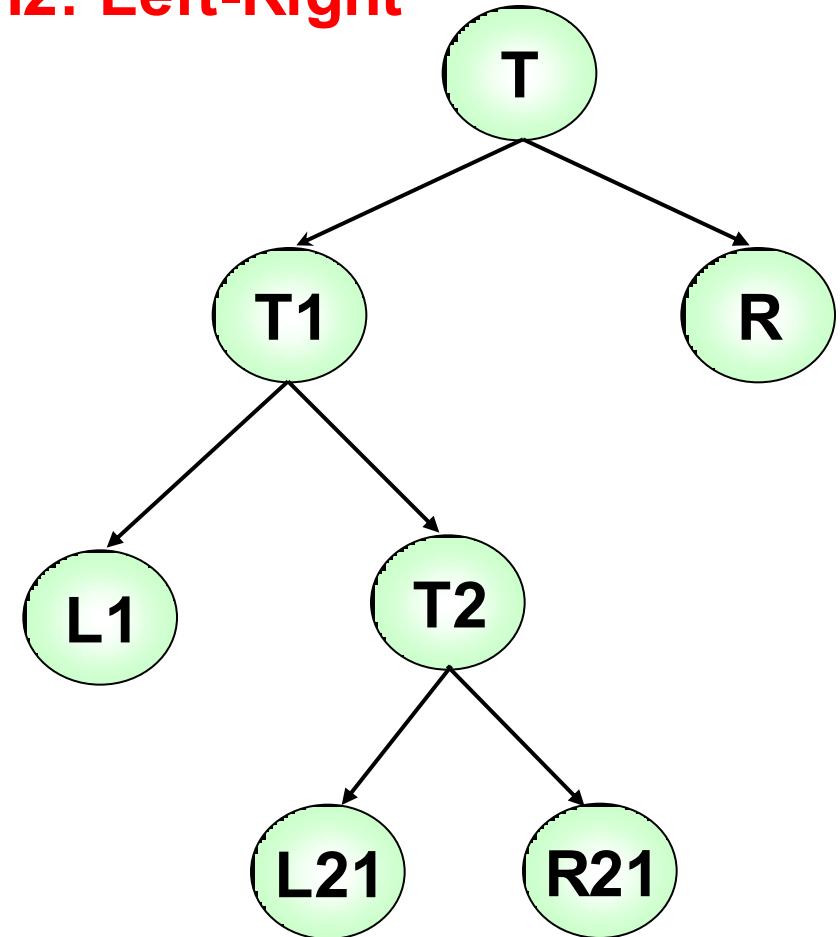
Các trường hợp mất cân bằng do lệch trái

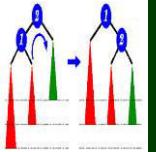
➤ Cây mất cân bằng tại nút T

TH1: Left-Left



TH2: Left-Right

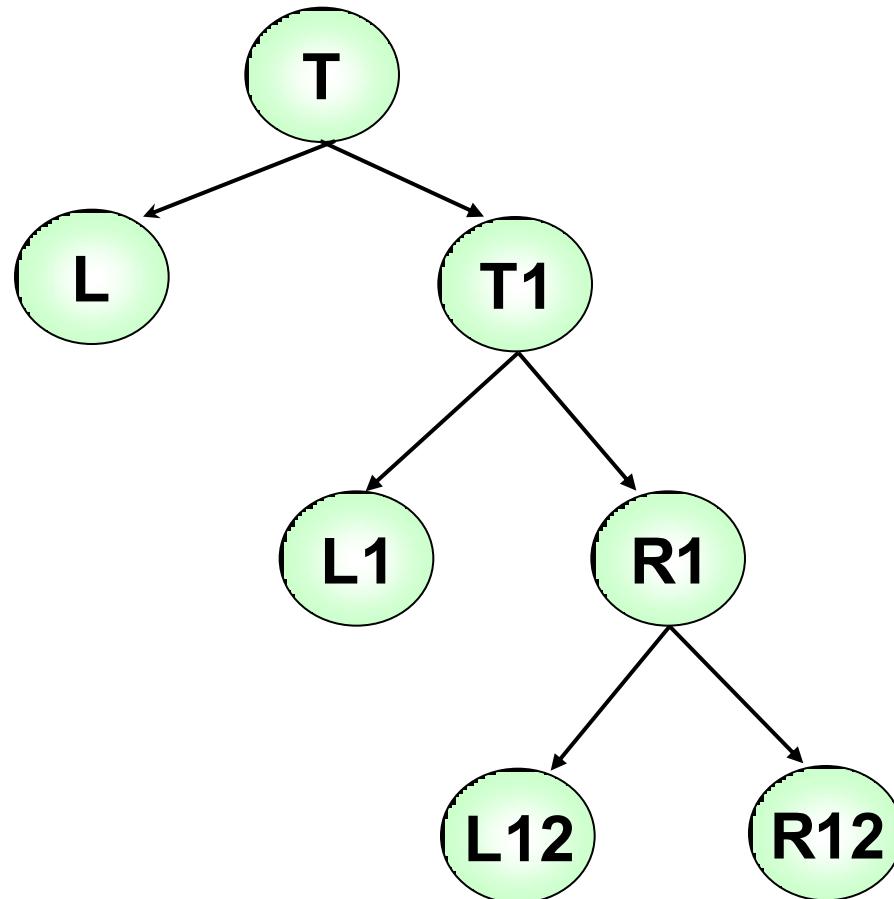




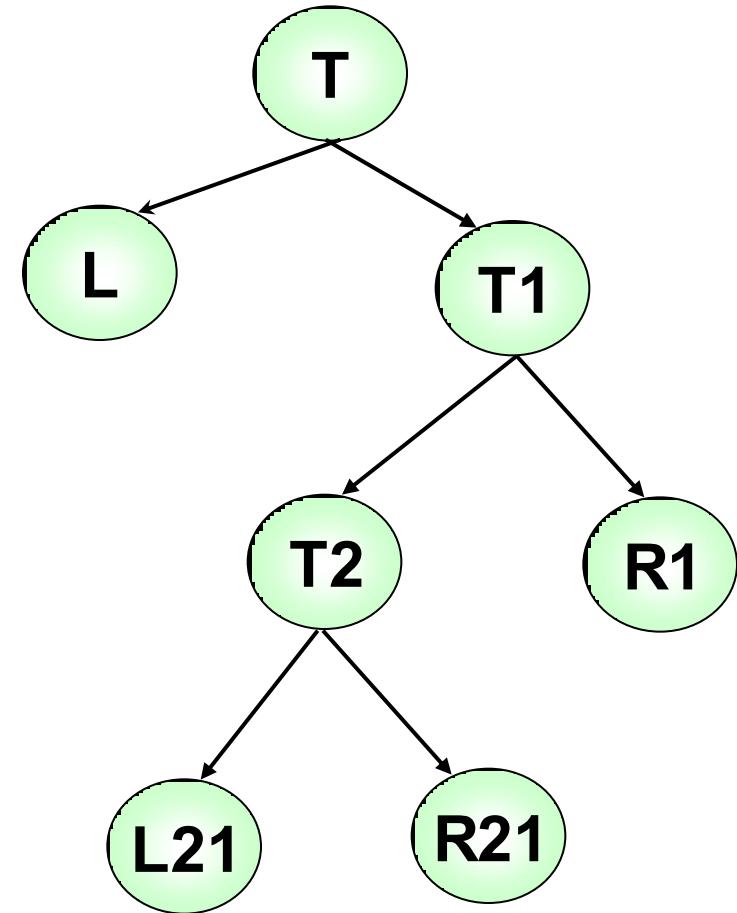
Các trường hợp mất cân bằng do lệch phải

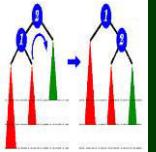
➤ Cây mất cân bằng tại nút T

TH3: Right-Right



TH4: Right-Left

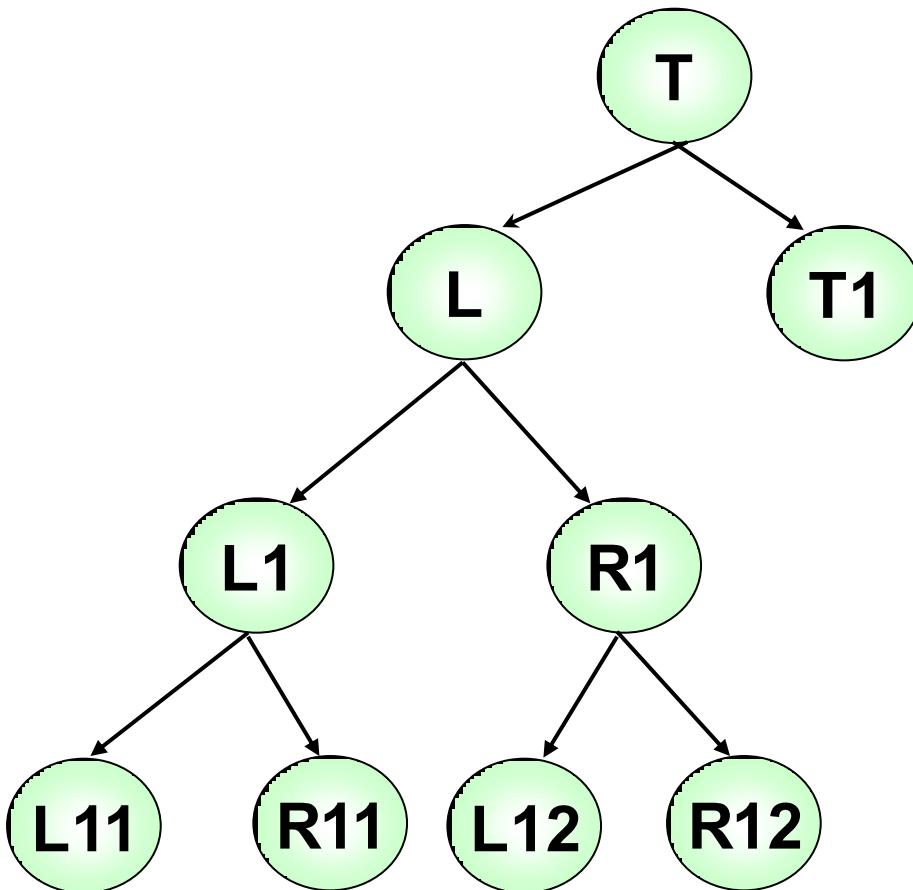




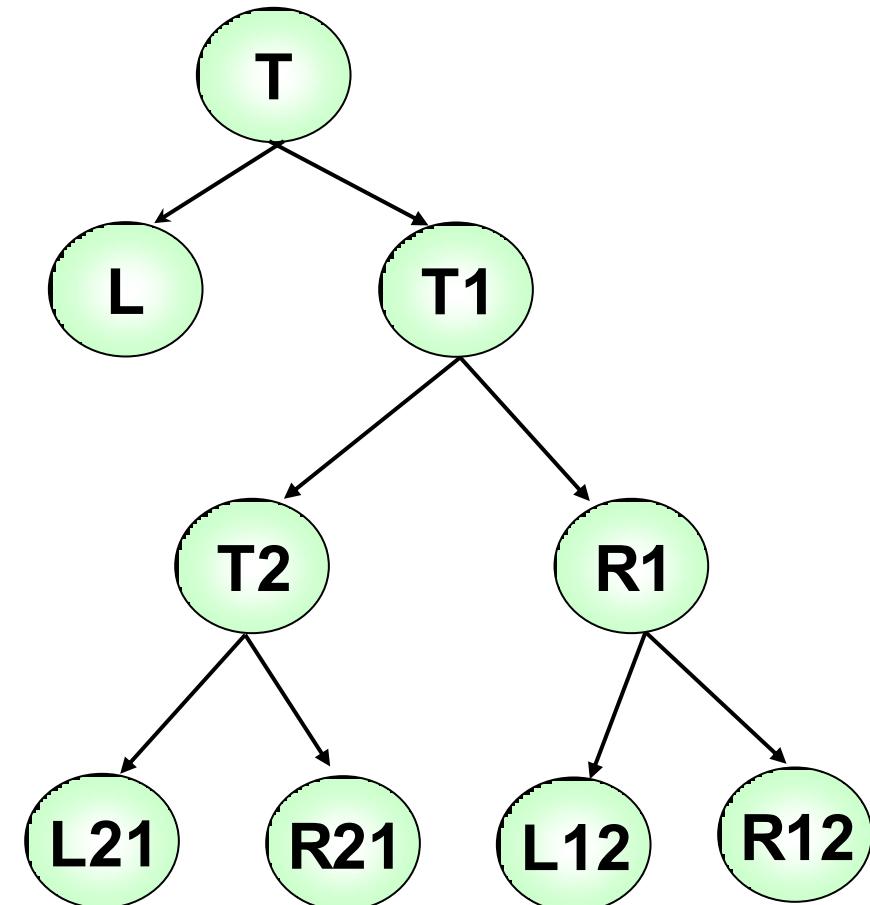
Các trường hợp mất cân bằng do xóa

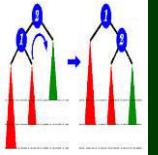
➤ Cây mất cân bằng tại nút T

TH5: Left-Balance

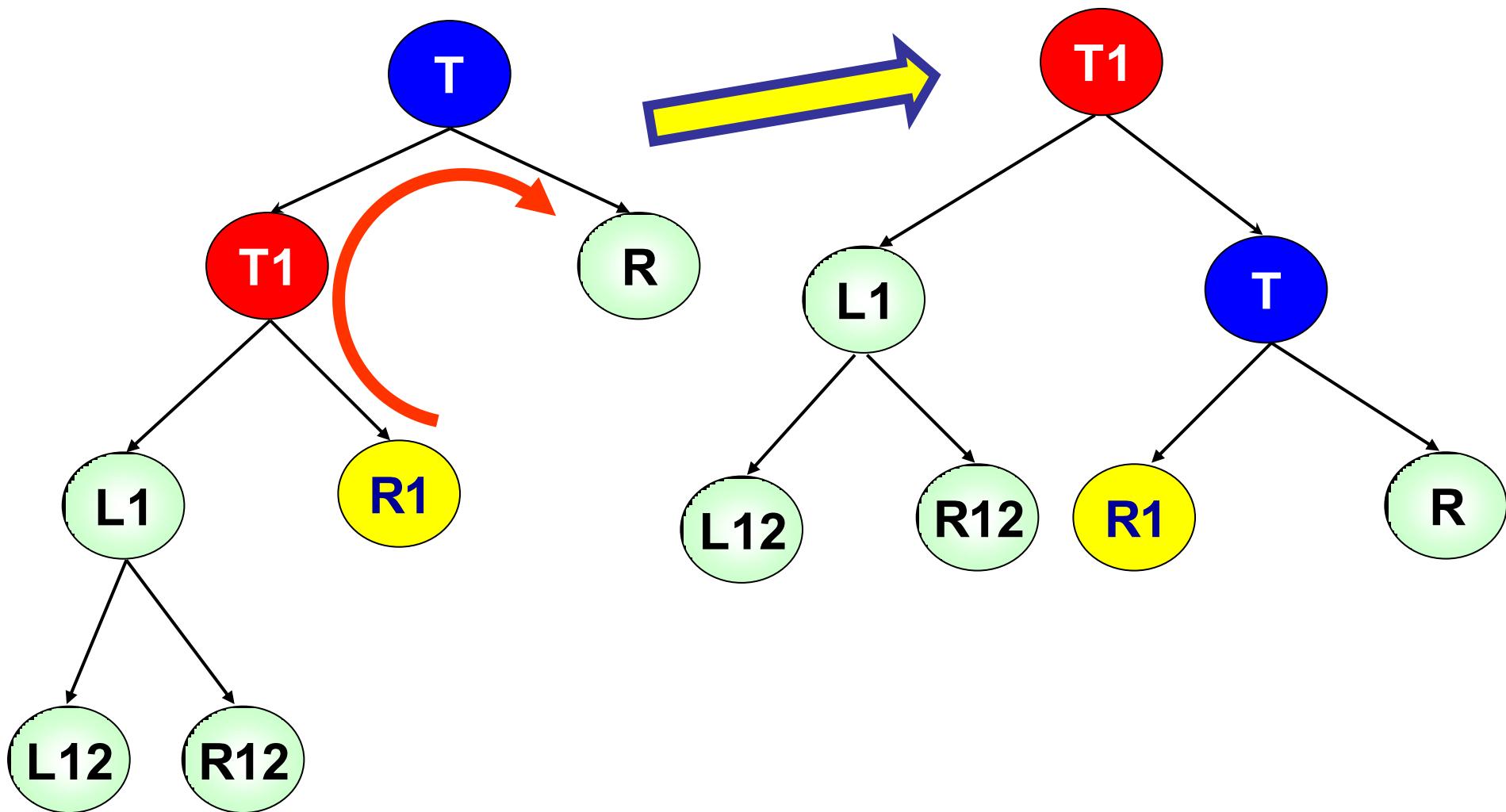


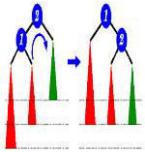
TH6: Right-Balance





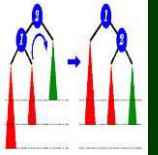
Cân bằng lại trường hợp 1



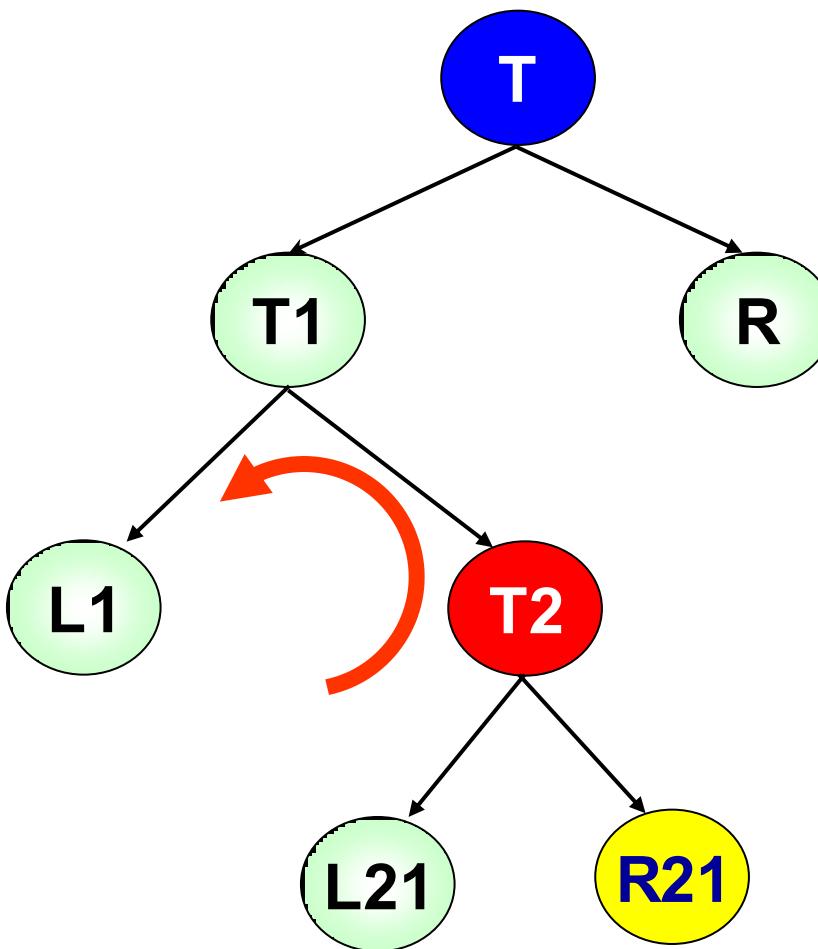


Cài đặt cân bằng lại cho trường hợp 1

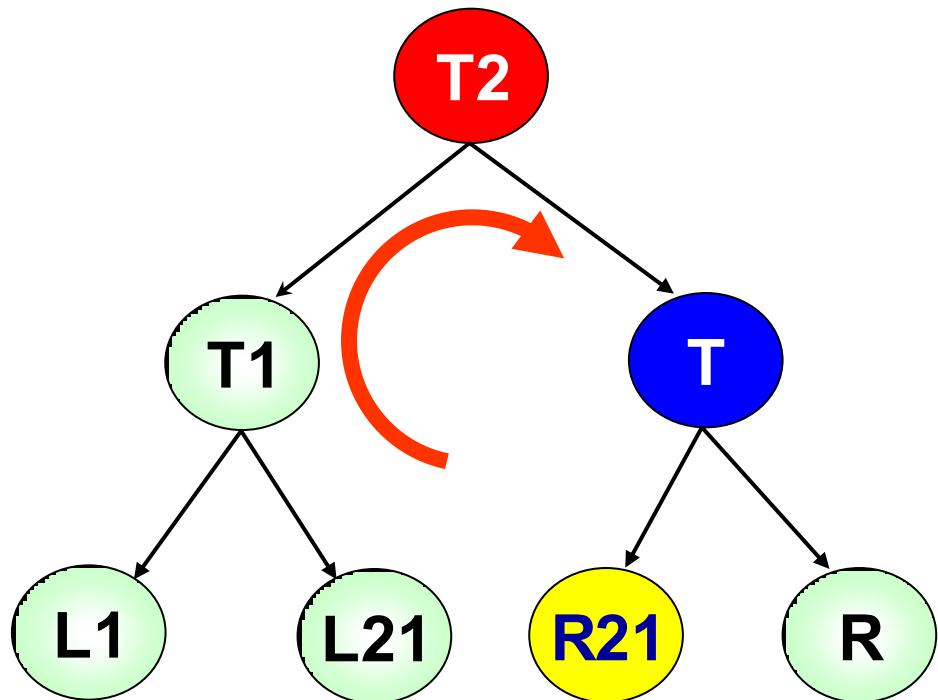
```
void rotateLL(AVLTree &T)
{
    AVLNode *T1 = T→Left;
    T→Left = T1→Right;
    T1→Right = T;
    switch(T1→balFactor)
    {
        case LH: T→balFactor = EH;
                    T1→balFactor = EH; break;
        case EH: T→balFactor = LH;
                    T1→balFactor = RH; break;
    }
    T = T1;
}
```



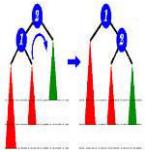
Cân bằng lại trường hợp 2



Xoay trái tại nút T_1

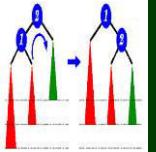


Xoay phải tại nút T_2

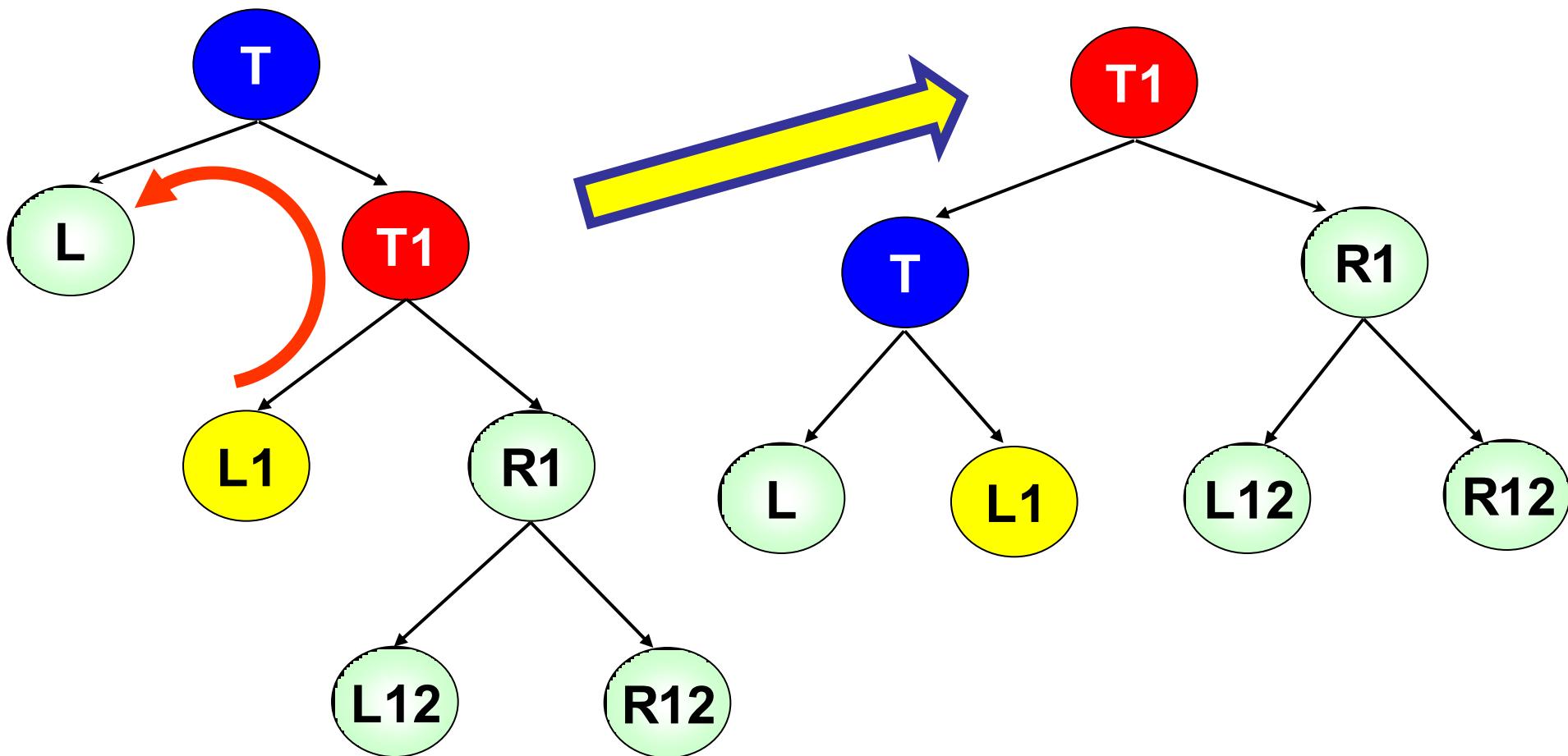


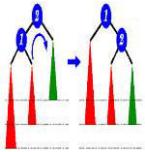
Cài đặt cân bằng lại cho trường hợp 2

```
void rotateLR(AVLTree &T)
{
    AVLNode *T1 = T→Left;
    AVLNode *T2 = T1→Right;
    T1→Right = T2→Left; T2→Left = T1;
    T→Left = T2→Right; T2→Right = T;
    switch(T2→balFactor)
    {
        case LH:      T→balFactor = RH;
                      T1→balFactor = EH; break;
        case EH:      T→balFactor = EH;
                      T1→balFactor = EH; break;
        case RH:      T→balFactor = EH;
                      T1→balFactor = LH; break;
    }
    T2→balFactor = EH; T = T2;
}
```



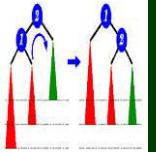
Cân bằng lại trường hợp 3



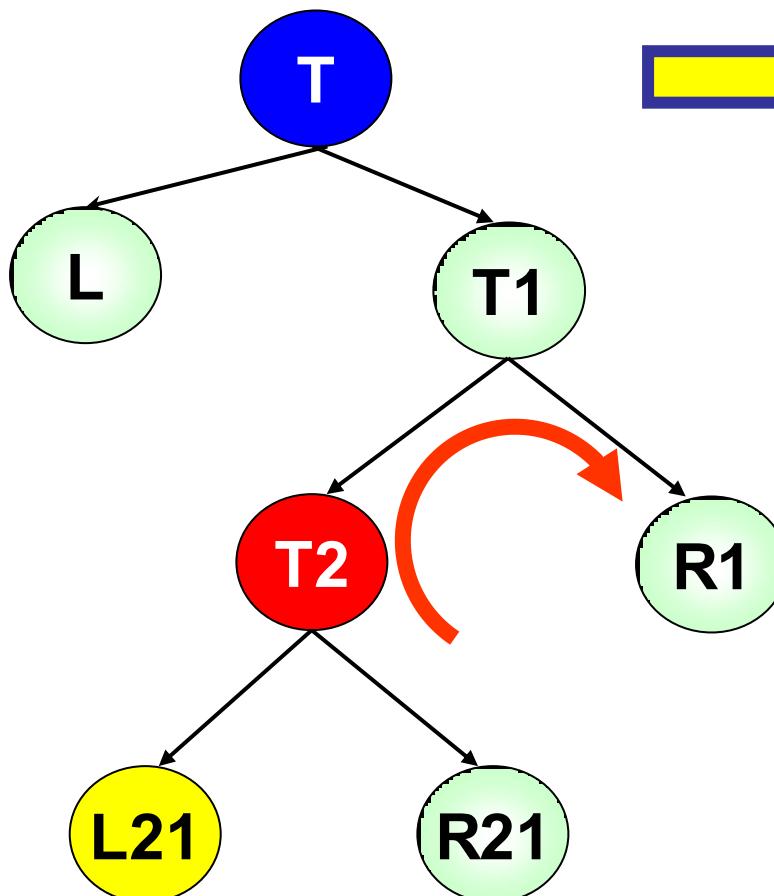


Cài đặt cân bằng lại cho trường hợp 3

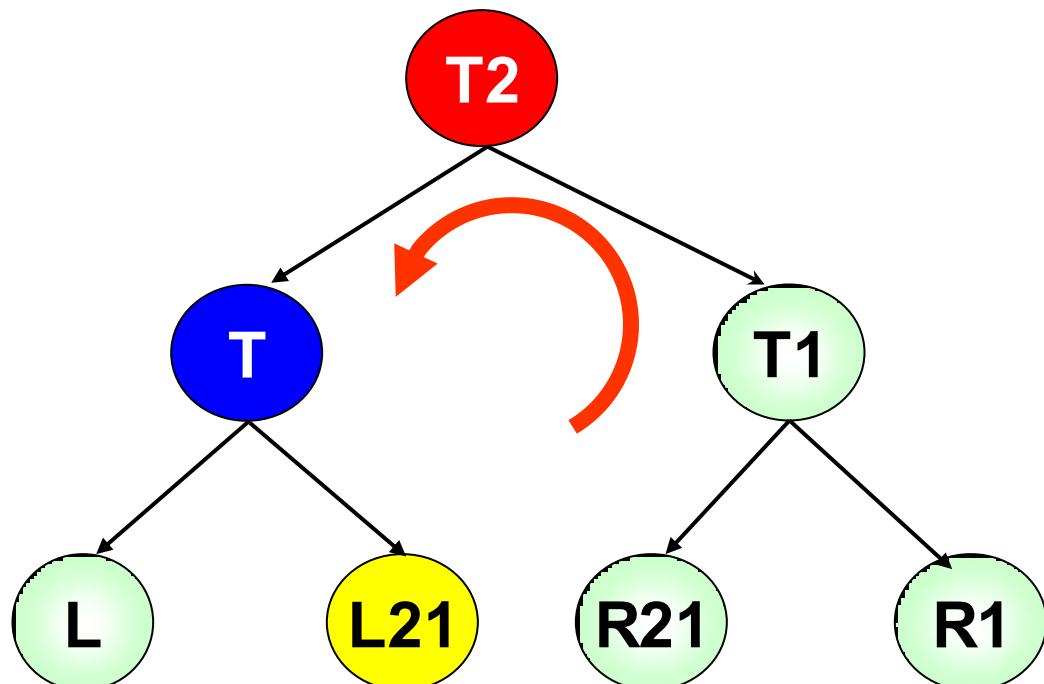
```
void rotateRR(AVLTree &T)
{
    AVLNode *T1 = T→Right;
    T→Right = T1→Left;          T1→Left = T;
    switch(T1→balFactor)
    {
        case RH:      T→balFactor = EH;
                        T→balFactor = EH; break;
        case EH:      T→balFactor = RH;
                        T1→balFactor = LH; break;
    }
    T = T1;
}
```



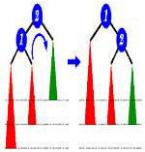
Cân bằng lại trường hợp 4



Xoay phải tại nút T_1

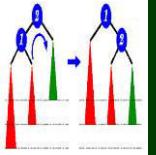


Xoay trái tại nút T_2

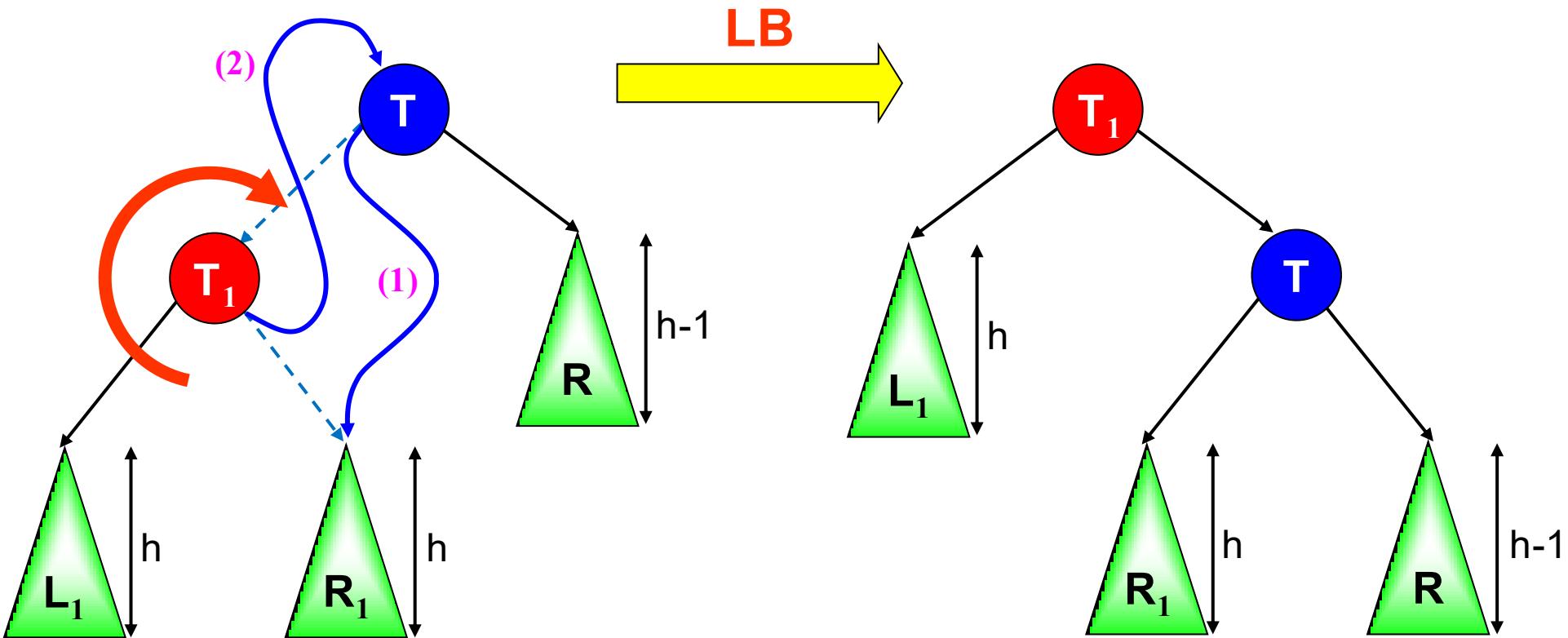


Cài đặt cân bằng lại cho trường hợp 4

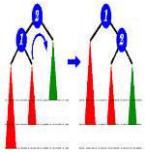
```
void rotateRL(AVLTree &T)
{
    AVLNode *T1 = T→Right;
    AVLNode *T2 = T1→Left;
    T1→Left = T2→Right;      T2→Right = T1;
    T→Right = T2→Left;      T2→Left = T;
    switch(T2→balFactor)
    {
        case RH:          T→balFactor = LH;
                           T1→balFactor = EH; break;
        case EH:          T→balFactor = EH;
                           T1→balFactor = EH; break;
        case LH:          T→balFactor = EH;
                           T1→balFactor = RH; break;
    }
    T2→balFactor = EH; T = T2;
}
```



Cân bằng lại trường hợp 5

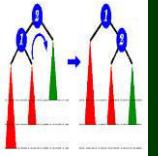


$T \rightarrow \text{Left} = T_1 \rightarrow \text{Right}; (1)$
 $T_1 \rightarrow \text{Right} = T; (2)$

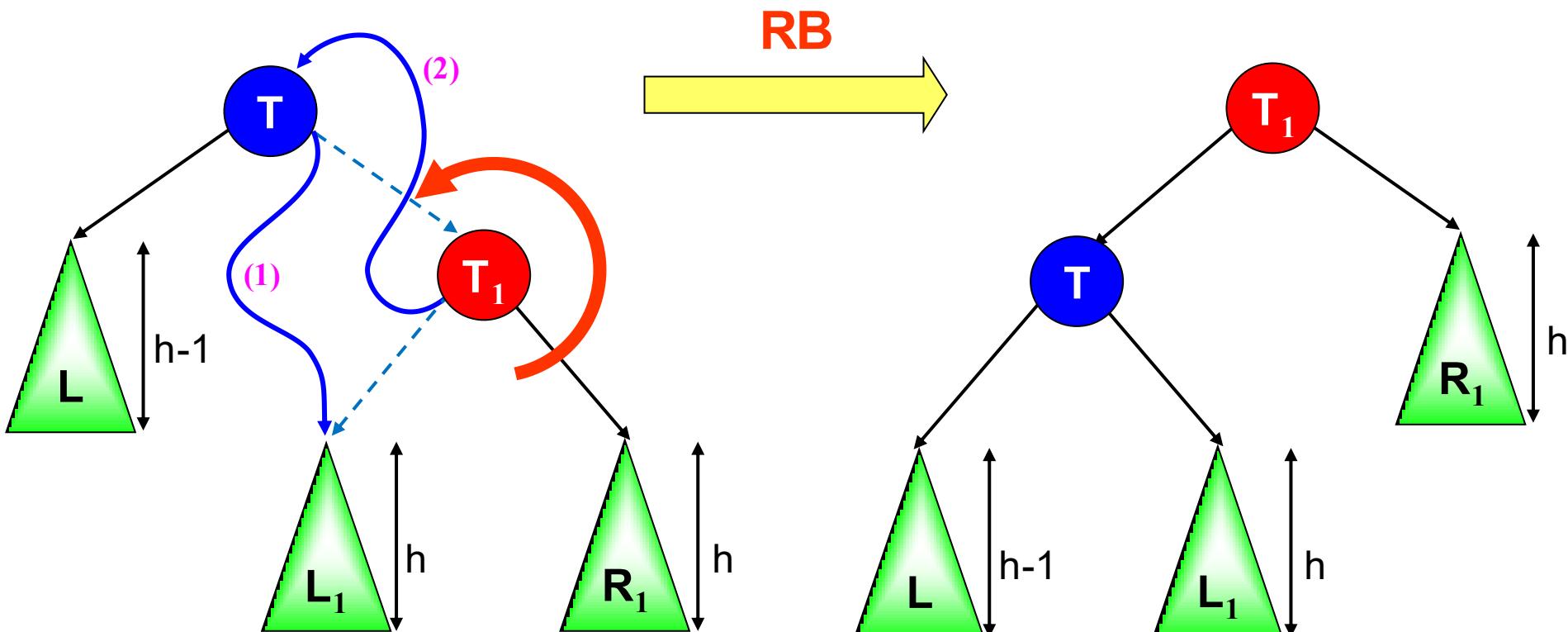


Cài đặt cân bằng lại cho trường hợp 5

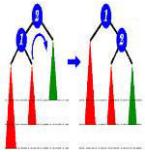
Phần cài đặt cho trường hợp **LB** này được cân bằng lại hoàn toàn giống với **LL**.



Cân bằng lại trường hợp 6

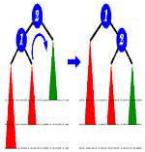


$T \rightarrow \text{Right} = T_1 \rightarrow \text{Left}; \quad (1)$
 $T_1 \rightarrow \text{Left} = T; \quad (2)$



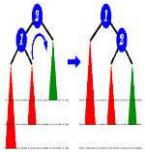
Cài đặt cân bằng lại cho trường hợp 6

Phần cài đặt cho trường hợp **RB** (Right Balance) này được cân bằng lại hoàn toàn giống với **RR** (Right Right).



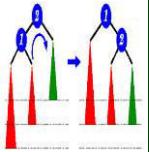
Cài đặt cân bằng lại cây AVL khi lệch trái

```
int balanceLeft(AVLNode* &T)
{//khi cây T lệch bên trái cần cân bằng lại
    AVLNode* T1=T→Left;
    switch(T1→balFactor)
    {
        case LH:   LL(T);           return 2;
        case EH:   LL(T);           return 1;
        case RH:   LR(T);           return 2;
    }
    return 0; //Trường balance bị sai
}
```



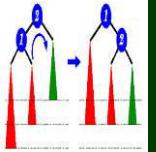
Cài đặt cân bằng lại cây AVL khi lệch phải

```
int balanceRight(AVLNode* &T)
{//khi cây T lệch bên phải cần cân bằng lại
    AVLNode* T1=T→Right;
    switch(T1→balFactor)
    {
        case LH:   RL(T);           return 2;
        case EH:   RR(T);          return 1;
        case RH:   RR(T);          return 2;
    }
    return 0; //Trường balance bị sai
}
```



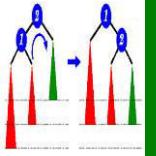
Thêm 1 nút vào cây AVL

- Thêm bình thường như trường hợp cây NPTK
- Nếu cây tăng trưởng chiều cao thì:
 - Lần ngược về gốc để phát hiện nút bị mất cân bằng.
 - Tiến hành cân bằng lại nút đó bằng thao tác cân bằng thích hợp.
- Việc cân bằng lại chỉ cần thực hiện 1 lần nơi mất cân bằng
- **Code:** Xem giáo trình lý thuyết



Hủy 1 nút ra khỏi cây AVL

- Hủy bình thường như trường hợp cây NPTK
- Nếu cây giảm chiều cao:
 - Lần ngược về gốc để phát hiện nút bị mất cân bằng.
 - Tiến hành cân bằng lại nút đó bằng thao tác cân bằng thích hợp.
 - Tiếp tục lần ngược lên nút cha...
- Việc cân bằng lại có thể lan truyền lên tận gốc.
- **Code:** Xem giáo trình lý thuyết



Lưu ý

- ❖ 2 trường hợp bị mất cân bằng LR, RL: *thì lấy cháu thay cho ông nội.*
- ❖ 4 trường hợp còn lại: LL, LB, RR, RB: *thì lấy con thay cho cha.*

Thank for you attention!



Goodbye! I wish you luck!