

HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT



ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

TUẦN 1 : CÁC KHÁI NIỆM CƠ BẢN

ONE LOVE. ONE FUTURE.

1. Ví dụ minh họa
2. Một số khái niệm cơ bản về thuật toán
3. Ký hiệu tiệm cận
4. Kỹ thuật phân tích thuật toán

MỤC TIÊU

Sau bài học này, người học có thể:

1. Hiểu được một số khái niệm cơ bản về thuật toán
2. Biết ký hiệu tiệm cận dùng để đánh giá độ phức tạp thuật toán
3. Biết cách phân tích độ phức tạp của thuật toán



1. Ví dụ minh họa
2. Một số khái niệm cơ bản về thuật toán
3. Ký hiệu tiệm cận
4. Kỹ thuật phân tích thuật toán

1. VÍ DỤ MINH HỌA

- Bài toán tìm dãy con lớn nhất:
 - Cho dãy số gồm n số: $a_0, a_1, a_2, \dots, a_{n-1}$
 - Dãy gồm liên tiếp các số a_i, a_{i+1}, \dots, a_j với $0 \leq i \leq j \leq n-1$ được gọi là **dãy con** của dãy đã cho và $\sum_{k=i}^j a_k$ được gọi là **trọng lượng** của dãy con này
 - **Yêu cầu:** Hãy tìm trọng lượng lớn nhất của các dãy con, tức là tìm **cực đại giá trị** $\sum_{k=i}^j a_k$. Ta gọi dãy con có trọng lượng lớn nhất là **dãy con lớn nhất**.
- **Ví dụ:** Cho dãy số -2, **11**, **-4**, **13**, -5, 2 thì cần đưa ra câu trả lời là 20 (dãy con lớn nhất là 11, -4, 13 với giá trị = $11 + (-4) + 13 = 20$)

1. VÍ DỤ MINH HỌA

- Cách 1: Duyệt toàn bộ

- Duyệt tất cả các dãy con có thể có của dãy đã cho: a_i, a_{i+1}, \dots, a_j với $0 \leq i \leq j \leq n-1$, và tính tổng của mỗi dãy con để tìm ra trọng lượng lớn nhất.

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



1. VÍ DỤ MINH HỌA

▪ Cách 1: Duyệt toàn bộ

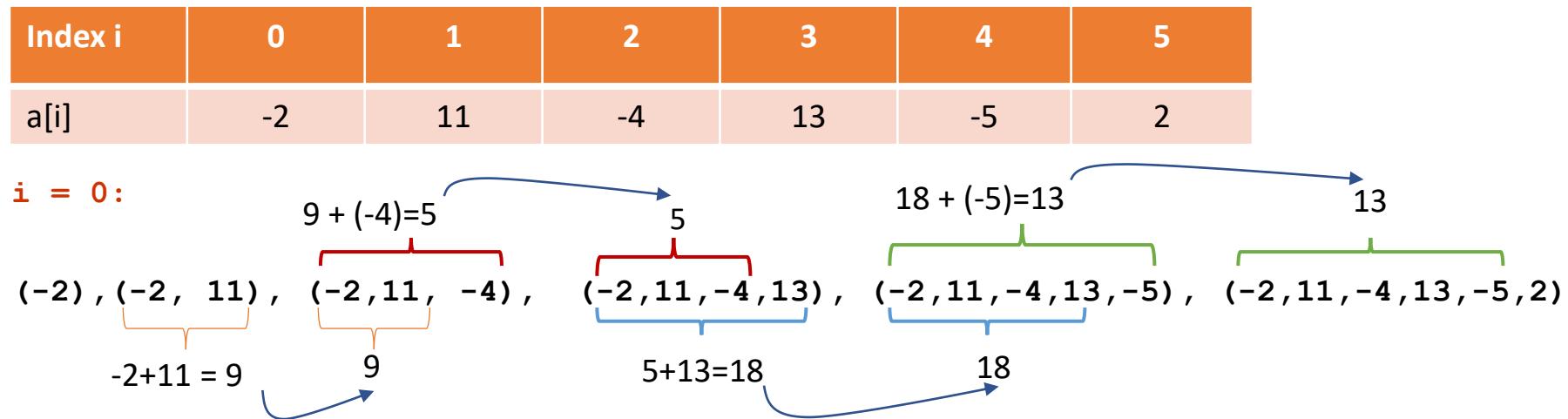
- Duyệt tất cả các dãy con có thể có của dãy đã cho: a_i, a_{i+1}, \dots, a_j với $0 \leq i \leq j \leq n-1$, và tính tổng của mỗi dãy con để tìm ra trọng lượng lớn nhất.
- Phân tích thuật toán:** Ta sẽ tính số lượng phép cộng mà thuật toán phải thực hiện, tức là đếm xem dòng lệnh **sum += a[k]** phải thực hiện bao nhiêu lần. Số lượng phép cộng là:

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

$$\begin{aligned} \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j-i+1) &= \sum_{i=0}^{n-1} (1+2+\dots+(n-i)) = \sum_{i=0}^{n-1} \frac{(n-i)(n-i+1)}{2} \\ &= \frac{1}{2} \sum_{k=1}^n k(k+1) = \frac{1}{2} \left[\sum_{k=1}^n k^2 + \sum_{k=1}^n k \right] = \frac{1}{2} \left[\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2} \right] \\ &= \frac{n^3}{6} + \frac{n^2}{2} + \frac{n}{3} \end{aligned}$$

1. VÍ DỤ MINH HỌA

■ Cách 2: Duyệt toàn bộ có cải tiến



- Nhận thấy, ta có thể tính tổng các phần tử từ vị trí i đến j từ tổng của các phần tử từ i đến $j-1$ chỉ bằng 1 phép cộng:

$$\sum_{k=i}^j a[k] = a[j] + \sum_{k=i}^{j-1} a[k]$$

Tổng các phần tử từ i đến j

Tổng các phần tử từ i đến $j-1$

1. VÍ DỤ MINH HỌA

- Cách 2: Duyệt toàn bộ có cải tiến

$$\sum_{k=i}^j a[k] = a[j] + \underbrace{\sum_{k=i}^{j-1} a[k]}$$

Tổng các phần tử từ i đến j

Tổng các phần tử từ i đến $j-1$

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

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



1. VÍ DỤ MINH HỌA

- Cách 2: Duyệt toàn bộ có cải tiến

- **Phân tích thuật toán:** Ta sẽ tính số lượng phép cộng mà thuật toán phải thực hiện, tức là đếm xem dòng lệnh **sum += a[j]** phải thực hiện bao nhiêu lần. Số lượng phép cộng là:

$$\sum_{i=0}^{n-1} (n-i) = n + (n-1) + \dots + 1 = \frac{n^2}{2} + \frac{n}{2}$$

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



1. VÍ DỤ MINH HỌA

- Số lượng phép cộng mà mỗi thuật toán cần thực hiện là:

- Cách 1. Duyệt toàn bộ $\frac{n^3}{6} + \frac{n^2}{2} + \frac{n}{3}$

- Cách 2. Duyệt toàn bộ có cải tiến $\frac{n^2}{2} + \frac{n}{2}$

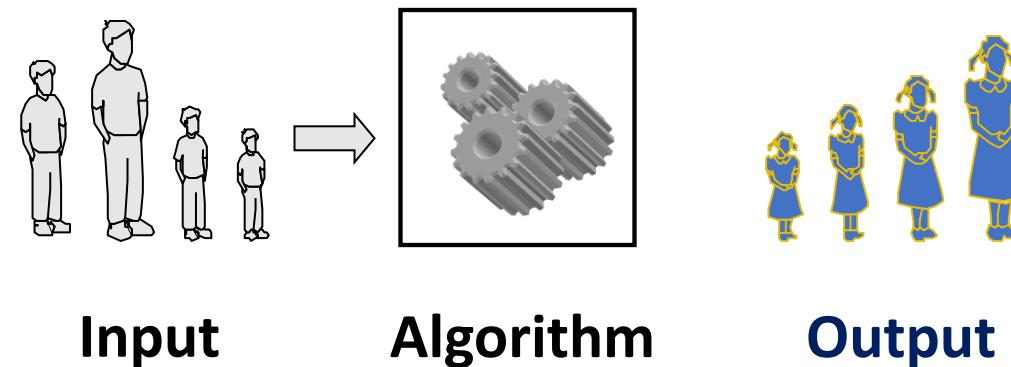
- Cùng một bài toán, ta đã đề xuất 2 thuật toán đòi hỏi số lượng phép toán khác nhau, và vì thế sẽ đòi hỏi thời gian tính khác nhau.
- Bảng dưới đây cho thấy thời gian tính của 2 thuật toán trên, với giả thiết: máy tính có thể thực hiện 10^8 phép cộng trong một giây

Độ phức tạp	n=10	Thời gian	n=100	Thời gian	n=10 ⁴	Thời gian	n=10 ⁶	Thời gian
n ³	10 ³	10 ⁻⁵ giây	10 ⁶	10 ⁻² giây	10 ¹²	2.7 giờ	10 ¹⁸	115 ngày
n ²	100	10 ⁻⁶ giây	10000	10 ⁻⁴ giây	10 ⁸	1 giây	10 ¹²	2.7 giờ

1. Ví dụ minh họa
2. Một số khái niệm cơ bản về thuật toán
3. Ký hiệu tiệm cận
4. Kỹ thuật phân tích thuật toán

2. MỘT SỐ KHÁI NIỆM CƠ BẢN VỀ THUẬT TOÁN

- Thuật toán (Algorithm) giải bài toán đặt ra là một thủ tục xác định bao gồm **một dãy hữu hạn các bước cần thực hiện** để thu **được đầu ra (output)** từ **một đầu vào cho trước (input)** của bài toán.



- Một số đặc trưng cơ bản của thuật toán:

- Chính xác
- Hữu hạn
- Đơn trị

2. MỘT SỐ KHÁI NIỆM CƠ BẢN VỀ THUẬT TOÁN

- Độ phức tạp của thuật toán:

- Khi nói đến hiệu quả của một thuật toán, ta quan tâm đến chi phí cần dùng để thực hiện nó:
 - 1) Dễ hiểu, dễ cài đặt, dễ sửa đổi ?
 - 2) Thời gian sử dụng CPU ? **THỜI GIAN**
 - 3) Tài nguyên bộ nhớ ? **BỘ NHỚ**



2. MỘT SỐ KHÁI NIỆM CƠ BẢN VỀ THUẬT TOÁN

- Độ phức tạp của thuật toán:

- Làm thế nào để đo được thời gian tính?
 - Thời gian tính của thuật toán phụ thuộc vào dữ liệu vào (kích thước tăng, thì thời gian tăng).
 - Vì vậy, người ta tìm cách đánh giá thời gian tính của thuật toán bởi một hàm của độ dài dữ liệu đầu vào. Tuy nhiên, trong một số trường hợp, kích thước dữ liệu đầu vào là như nhau, nhưng thời gian tính lại rất khác nhau.
 - Ví dụ: Để tìm số nguyên tố đầu tiên có trong dãy: ta duyệt dãy từ trái sang phải
 - Dãy 1: 3 9 8 12 15 20 (thuật toán dừng ngay khi xét phần tử đầu tiên)
 - Dãy 2: 9 8 3 12 15 20 (thuật toán dừng khi xét phần tử thứ ba)
 - Dãy 3: 9 8 12 15 20 3 (thuật toán dừng khi xét phần tử cuối cùng)



2. MỘT SỐ KHÁI NIỆM CƠ BẢN VỀ THUẬT TOÁN

- Các loại thời gian tính của thuật toán:

- **Thời gian tính tốt nhất (Best-case)**

$T(n)$: thời gian tối thiểu cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích thước n .

- **Thời gian tính trung bình (Average-case)**

$T(n)$: thời gian trung bình cần thiết để thực hiện thuật toán trên tập hữu hạn các đầu vào kích thước n .

- **Thời gian tính tồi nhất (Worst-case)**

$T(n)$: thời gian nhiều nhất cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích thước n .



2. MỘT SỐ KHÁI NIỆM CƠ BẢN VỀ THUẬT TOÁN

- Có hai cách để đánh giá thời gian tính:

- **Từ thời gian chạy thực nghiệm:**

- cài đặt thuật toán, rồi chọn các bộ dữ liệu đầu vào thử nghiệm
 - chạy chương trình với các dữ liệu đầu vào kích thước khác nhau
 - sử dụng hàm `clock()` để đo thời gian chạy chương trình

```
clock_t startTime = clock();
doSomeOperation();
clock_t endTime = clock();
clock_t clockTicksTaken = endTime - startTime;
double timeInSeconds = clockTicksTaken / (double) CLOCKS_PER_SEC;
```

- **Lý thuyết: khái niệm xấp xỉ tiệm cận**



1. Ví dụ minh họa
2. Một số khái niệm cơ bản về thuật toán
3. Ký hiệu tiệm cận
4. Kỹ thuật phân tích thuật toán

3. KÝ HIỆU TIỆM CẬN

- Các ký hiệu tiệm cận (asymptotic notation):

$\Theta, \Omega, O, \omega$

- Được sử dụng để mô tả thời gian tính của thuật toán, mô tả tốc độ tăng của thời gian chạy phụ thuộc vào kích thước dữ liệu đầu vào.
- Ví dụ, khi nói thời gian tính của thuật toán cỡ $\Theta(n^2)$, tức là, thời gian tính tỉ lệ thuận với n^2 cộng thêm các đa thức bậc thấp hơn.



3. KÝ HIỆU TIỆM CẬN

3.1. Ký hiệu tiệm cận theta Θ

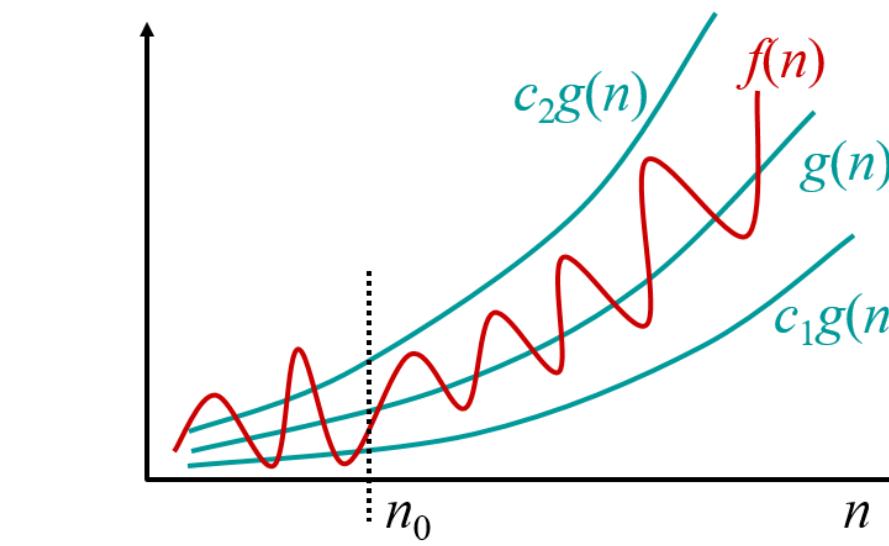
- Đối với hàm $g(n)$ cho trước, ta kí hiệu $\Theta(g(n))$ là tập các hàm:

$\Theta(g(n)) = \{f(n): \text{tồn tại các hằng số } c_1, c_2 \text{ và } n_0 \text{ sao cho:}$

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \text{ với mọi } n \geq n_0\}$$

(tập tất cả các hàm có cùng tốc độ tăng với hàm $g(n)$)

- Khi ta nói một hàm là theta của hàm khác, nghĩa là không có hàm nào đạt tới giá trị vô cùng nhanh hơn.



3. KÝ HIỆU TIỆM CẬN

3.1. Ký hiệu tiệm cận theta Θ

- Ví dụ: **Chứng minh rằng $10n^2 - 3n = \Theta(n^2)$**

Ta cần chỉ ra với những giá trị nào n_0, c_1, c_2 thì bất đẳng thức trong định nghĩa của kí hiệu theta là đúng:

$$c_1 n^2 \leq f(n) = 10n^2 - 3n \leq c_2 n^2 \quad \forall n \geq n_0$$

Gợi ý: lấy c_1 nhỏ hơn hệ số của số hạng với số mũ cao nhất, và c_2 lấy lớn hơn.

→ Chọn: $c_1 = 1, c_2 = 11, n_0 = 1$ thì ta có

$$n^2 \leq 10n^2 - 3n \leq 11n^2, \text{ với } n \geq 1$$

$$\rightarrow \forall n \geq 1: 10n^2 - 3n = \Theta(n^2)$$

Chú ý: Với các hàm đa thức: để so sánh tốc độ tăng, ta cần nhìn vào số hạng có số mũ cao nhất

3. KÝ HIỆU TIỆM CẬN

3.2. Ký hiệu tiệm cận O lớn O

- Đối với hàm $g(n)$ cho trước, ta kí hiệu $O(g(n))$ là tập các hàm:

$O(g(n)) = \{f(n): \text{tồn tại các hằng số dương } c \text{ và } n_0 \text{ sao cho:}$

$$f(n) \leq cg(n) \text{ với mọi } n \geq n_0\}$$

(tập tất cả các hàm có **tốc độ tăng nhỏ hơn hoặc bằng** tốc độ tăng của $g(n)$)

- $O(g(n))$ là tập các hàm đạt tới giá trị vô cùng không nhanh hơn $g(n)$.
- Ví dụ: Chứng minh rằng $2n + 10 = O(n)$

→ $f(n) = 2n+10, g(n) = n$

- Cần tìm hằng số c và n_0 sao cho:

$$2n + 10 \leq cn \text{ với mọi } n \geq n_0$$

$$\rightarrow (c - 2)n \geq 10$$

$$\rightarrow n \geq 10/(c - 2)$$

→ chọn $c = 3$ và $n_0 = 10$

3. KÝ HIỆU TIỆM CẬN

3.2. Ký hiệu tiệm cận O lớn O

- Chú ý: Có $f(n) = 50n^3 + 20n + 4$ là $O(n^3)$
 - Cũng đúng khi nói $f(n)$ là $O(n^3+n)$
 - Không hữu ích, vì n^3 có tốc độ tăng lớn hơn rất nhiều so với n , khi n lớn
 - Cũng đúng khi nói $f(n)$ là $O(n^5)$
 - Đúng, nhưng $g(n)$ nên có tốc độ tăng càng gần với tốc độ tăng của $f(n)$ càng tốt, thì đánh giá thời gian tính mới có giá trị
- Quy tắc đơn giản: Bỏ qua các số hạng có số mũ thấp hơn và các hằng số
 - Ví dụ:
 - Tất cả các hàm sau đều là $O(n)$: $n, 3n, 61n + 5, 22n - 5, \dots$
 - Tất cả các hàm sau đều là $O(n^2)$: $n^2, 9n^2, 18n^2 + 4n - 53, \dots$
 - Tất cả các hàm sau đều là $O(n \log n)$: $n(\log n), 5n(\log 99n), 18 + (4n - 2)(\log(5n + 3)), \dots$

3. KÝ HIỆU TIỆM CẬN

3.3. Ký hiệu tiệm cận Omega Ω

- Đối với hàm $g(n)$ cho trước, ta kí hiệu $\Omega(g(n))$ là tập các hàm:

$\Omega(g(n)) = \{f(n): \text{tồn tại các hằng số dương } c \text{ và } n_0 \text{ sao cho:}$

$$cg(n) \leq f(n) \text{ với mọi } n \geq n_0\}$$

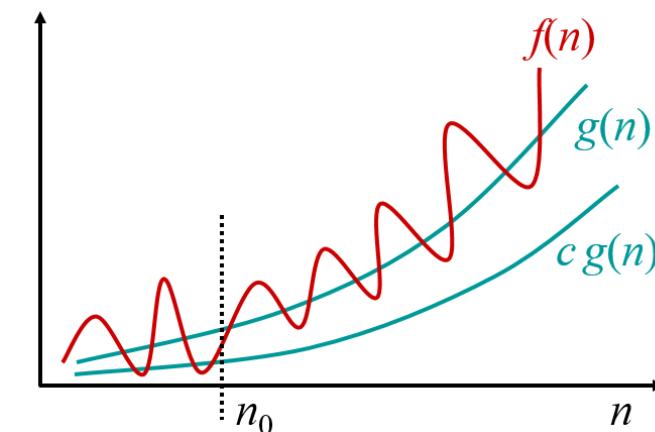
(tập tất cả các hàm có **tốc độ tăng lớn hơn hoặc bằng** tốc độ tăng của $g(n)$)

- $\Omega(g(n))$ là tập các hàm đạt tới giá trị vô cùng không chậm hơn $g(n)$.

- Ví dụ: Chứng minh rằng $5n^2 = \Omega(n)$

Cần tìm c và n_0 sao cho $cn \leq 5n^2$ với $n \geq n_0$

Bất đẳng thức đúng với $c = 1$ và $n_0 = 1$



1. Ví dụ minh họa
2. Một số khái niệm cơ bản về thuật toán
3. Ký hiệu tiệm cận
4. Kỹ thuật phân tích thuật toán

4. KỸ THUẬT PHÂN TÍCH THUẬT TOÁN

- **Cấu trúc tuần tự:**

Thời gian tính của chương trình “P; Q”, với P và Q là hai đoạn chương trình thực thi một thuật toán, P thực hiện trước, rồi đến Q là: $Time(P; Q) = Time(P) + Time(Q)$ hoặc ta có thể dùng kí hiệu tiệm cận theta: $Time(P; Q) = \Theta(\max(Time(P), Time(Q)))$ với $Time(P)$, $Time(Q)$ là thời gian tính của P và Q.

- **Vòng lặp FOR**

for i =1 to m do P(i);

Giả sử thời gian thực hiện $P(i)$ là $t(i)$, khi đó thời gian thực hiện vòng lặp for là $\sum_{i=1}^m t(i)$



4. KỸ THUẬT PHÂN TÍCH THUẬT TOÁN

- **Vòng lặp FOR lồng nhau**

```
for i =1 to n do  
    for j =1 to m do P(j);
```

Giả sử thời gian thực hiện $P(j)$ là $t(j)$, khi đó thời gian thực hiện vòng lặp lồng nhau này là:

- **Cấu trúc If/Else**

```
if (điều_kiện) then P;  
else Q;  
endif;
```

Thời gian thực hiện câu lệnh if/else

$$= \text{thời gian kiểm tra (điều_kiện)} + \max(\text{Time}(P), \text{Time}(Q))$$



4. KỸ THUẬT PHÂN TÍCH THUẬT TOÁN

- Ví dụ

Case1: for (i=0; i<n; i++)

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

$O(n^2)$

$O(n^2)$

$O(n^2)$



4. KỸ THUẬT PHÂN TÍCH THUẬT TOÁN

- **Câu lệnh đặc trưng:** là câu lệnh được thực hiện thường xuyên ít nhất là cũng như bất kỳ câu lệnh nào trong thuật toán.
- Nếu giả thiết thời gian thực hiện mỗi câu lệnh là bị chặn bởi hằng số thì thời gian tính của thuật toán sẽ cùng cỡ với số lần thực hiện câu lệnh đặc trưng. Do đó, để đánh giá thời gian tính, người ta đếm số lần thực hiện câu lệnh đặc trưng.
- Ví dụ 1: Hàm tính số Fibonacci $f_0=0; f_1=1; f_n=f_{n-1} + f_{n-2}$

```
function Fibiter(n)
    i=0;
    j=1;
    for k=1 to n-1 do
        j = i + j;
        i = j - i;
    return j;
```

Số lần thực hiện câu lệnh đặc trưng là n

➔ Thời gian chạy Fibiter là $O(n)$

4. KỸ THUẬT PHÂN TÍCH THUẬT TOÁN

- Ví dụ 2: Bài toán dãy con lớn nhất
 - Thuật toán 1: Duyệt toàn bộ

```
int maxSum = a[0];
for (int i=0; i<n; i++) {
    for (int j=i; j<n; j++) {
        int sum = 0;
        for (int k=i; k<=j; k++)
            sum += a[k];
        if (sum > maxSum)
            maxSum = sum;
    }
}
```

Chọn câu lệnh đặc trưng là **sum+=a[k]**

→ Thời gian tính của thuật toán: $O(n^3)$



4. KỸ THUẬT PHÂN TÍCH THUẬT TOÁN

- Ví dụ 2: Bài toán dãy con lớn nhất
 - Thuật toán 2: Duyệt toàn bộ có cải tiến

```
int maxSum = a[0];
for (int i=0; i<n; i++) {
    int sum = 0;
    for (int j=i; j<n; j++) {
        sum += a[j];
        if (sum > maxSum)
            maxSum = sum;
    }
}
```

Chọn câu lệnh đặc trưng là **sum+=a[j]**

→ Thời gian tính của thuật toán: $O(n^2)$



4. KỸ THUẬT PHÂN TÍCH THUẬT TOÁN

- Ví dụ 3: Đưa ra đánh giá tiệm cận O lớn cho thời gian tính $T(n)$ của đoạn chương trình sau:

a)

```
int x = 0;  
for (int i = 1; i <= n; i *= 2) x=x+1;
```

b)

```
int x = 0;  
for (int i = n; i > 0; i /= 2) x=x+1;
```



TỔNG KẾT VÀ GỢI MỞ

1. Bài học đã trình bày các khái niệm cơ bản về thuật toán và độ phức tạp thuật toán
2. Tiếp sau bài này, người học sẽ được học về đệ quy - sơ đồ chung và một số ví dụ

A large, faint watermark of the HUST logo is visible across the entire slide, consisting of a grid of red dots.

HUST

THANK YOU !

HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT



ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

TUẦN 2 : ĐỆ QUY, ĐỆ QUY CÓ NHỚ

ONE LOVE. ONE FUTURE.

NỘI DUNG

- Khái niệm cơ bản
- Sơ đồ chung đệ quy
- Phân tích thuật toán đệ quy
- Đệ quy có nhớ



KHÁI NIỆM CƠ BẢN

- Đối tượng đệ quy: được xác định thông qua chính nó nhưng với quy mô nhỏ hơn
 - Hàm đệ quy
 - Bước cơ sở: Xác định giá trị của hàm với một số giá trị tham số ban đầu
 - Bước đệ quy: Xác định mối quan hệ giữa hàm phụ thuộc vào chính hàm đó nhưng với tham số nhỏ hơn
- Cơ sở: $F(n) = 1$, với $n = 1$
 - Đệ quy: $F(n) = F(n-1) + n$, với $n > 1$
- Cơ sở: $C(k, n) = 1$, với $k = 0$ hoặc $k = n$
 - Đệ quy: $C(k, n) = C(k-1, n-1) + C(k, n-1)$, với các trường hợp khác



KHÁI NIỆM CƠ BẢN

- Đối tượng đệ quy: được xác định thông qua chính nó nhưng với quy mô nhỏ hơn
- Tập hợp được xác định đệ quy
 - Bước cơ sở: xác định các phần tử đầu tiên của tập hợp
 - Bước đệ quy: xác định luật cho biết các phần tử lớn hơn thuộc tập hợp từ các phần tử ban đầu

- Cơ sở: $3 \in S$
- Đệ quy: Nếu x và $y \in S$ thì $x + y \in S$



SƠ ĐỒ CHUNG ĐỀ QUY

- Thuật toán đệ quy là thuật toán tự gọi đến chính mình với đầu vào kích thước nhỏ hơn.
- Thuật toán đệ quy thường được dùng khi cần xử lý với các đối tượng được định nghĩa đệ quy.
 - Ví dụ: hàm tính dãy số Fibonacci được định nghĩa đệ quy:
 - $f(0) = 0, f(1) = 1,$
 - $f(n) = f(n-1) + f(n-2)$ với $n > 1$
- Các ngôn ngữ lập trình bậc cao thường cho phép xây dựng các hàm đệ quy, nghĩa là trong thân của hàm có chứa những lệnh gọi đến chính nó. Vì thế, khi cài đặt các thuật toán đệ quy, người ta thường xây dựng các hàm đệ quy.



SƠ ĐỒ CHUNG ĐỆ QUY

```
DeQuy(input) {  
    if (kích thước của input là nhỏ nhất) then  
        Thực hiện Bước cơ sở; /* giải bài toán kích thước đầu vào nhỏ nhất */  
    else {  
        DeQuy(input với kích thước nhỏ hơn); /* bước đệ quy */  
        /* Chú ý: có thể có thêm những lệnh gọi đệ quy */  
        Tổ hợp lời giải của các bài toán con để thu được lời_giải;  
        return lời_giải;  
    }  
}
```



VÍ DỤ

■ Ví dụ 1: Tính $n!$ theo công thức đệ quy:

$$f(0) = 1$$

$$f(n) = n * f(n-1)$$

(7) return 6

```
factorial(3):  
    3 == 0 ? NO  
    return 3*factorial(2)
```

(1) Trong hàm factorial(3)
Gọi tới factorial(2)

```
factorial(2):  
    2 == 0 ? NO  
    return 2*factorial(1)
```

(2) Trong hàm factorial(2)
Gọi tới factorial(1)

(5) return 1

```
factorial(1):  
    1 == 0 ? NO  
    return 1*factorial(0)
```

(3) Trong hàm factorial(1)
Gọi tới factorial(0)

(4) return 1

```
factorial(0):  
    0 == 0 ? YES  
    return 1
```

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

Thực thi hàm factorial(3) sẽ dừng cho đến khi hàm factorial(2) trả về kết quả

Khi hàm factorial(2) trả về kết quả, hàm factorial(3) tiếp tục được thực hiện

- **Ví dụ 2: Tính dãy số Fibonacci:**

$$F(0) = 0; F(1) = 1$$

$$F(n) = F(n-1) + F(n-2) \text{ với } n \geq 2$$

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

VÍ DỤ: BÀI TOÁN THÁP HÀ NỘI

▪ Ví dụ 3: Bài toán Tháp Hà Nội:

Bài toán Tháp Hà Nội được trình bày như sau: “Có 3 cọc a, b, c. Trên cọc a có một chồng gồm n cái đĩa, đường kính giảm dần từ dưới lên trên. Cần phải chuyển chồng đĩa từ cọc a sang cọc c tuân thủ quy tắc:

1. Mỗi lần chỉ chuyển 1 đĩa
2. Chỉ được xếp đĩa có đường kính nhỏ hơn lên trên đĩa có đường kính lớn hơn. Trong quá trình chuyển được phép dùng cọc b làm cọc trung gian.

Bài toán đặt ra là: Hãy liệt kê các bước di chuyển đĩa cần thực hiện để hoàn thành nhiệm vụ đặt ra của bài toán.



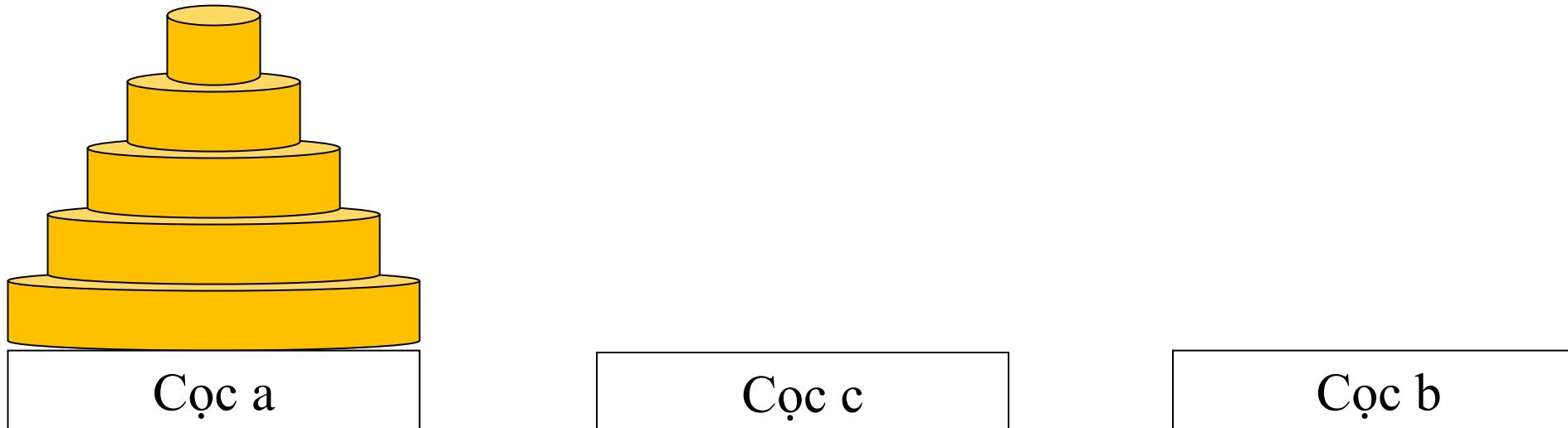
VÍ DỤ: BÀI TOÁN THÁP HÀ NỘI

Chuyển n đĩa từ cọc a sang cọc c sử dụng cọc b làm trung gian:

HanoiTower(n, a, c, b);

Việc di chuyển đĩa gồm 3 giai đoạn:

- (1) Chuyển $n-1$ đĩa từ cọc a sang cọc b , sử dụng cọc c làm trung gian
- (2) Chuyển 1 đĩa (đĩa với đường kính lớn nhất) từ cọc a sang cọc c
- (3) Chuyển $n-1$ đĩa từ cọc b sang cọc c , sử dụng cọc a làm trung gian



VÍ DỤ: BÀI TOÁN THÁP HÀ NỘI

Chuyển n đĩa từ cọc a sang cọc c sử dụng cọc b làm trung gian:

HanoiTower(n, a, c, b);

Việc di chuyển đĩa gồm 3 giai đoạn:

(1) Chuyển $n-1$ đĩa từ cọc a sang cọc b , sử dụng cọc c làm trung gian

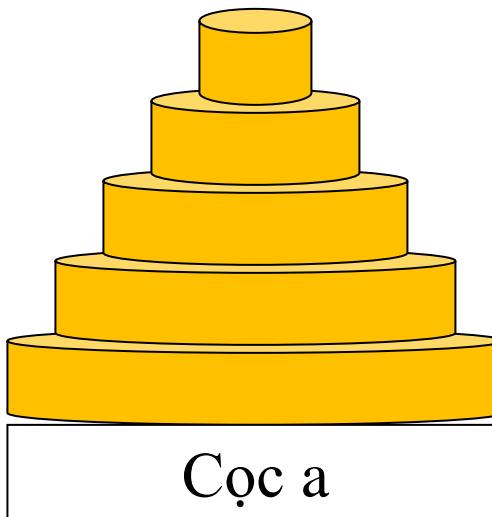
Giải bài toán kích thước $n-1$: **HaNoiTower(n-1, a, b, c)**

(2) Chuyển 1 đĩa (đĩa với đường kính lớn nhất) từ cọc a sang cọc c

Giải bài toán kích thước $n = 1$ (chỉ cần 1 bước di chuyển đĩa): **HaNoiTower(1, a, c, b)**

(3) Chuyển $n-1$ đĩa từ cọc b sang cọc c , sử dụng cọc a làm trung gian

Giải bài toán kích thước $n-1$: **HaNoiTower(n-1, b, c, a)**



VÍ DỤ: BÀI TOÁN THÁP HÀ NỘI

Ví dụ 3: Bài toán tháp Hà Nội: $n = 5$

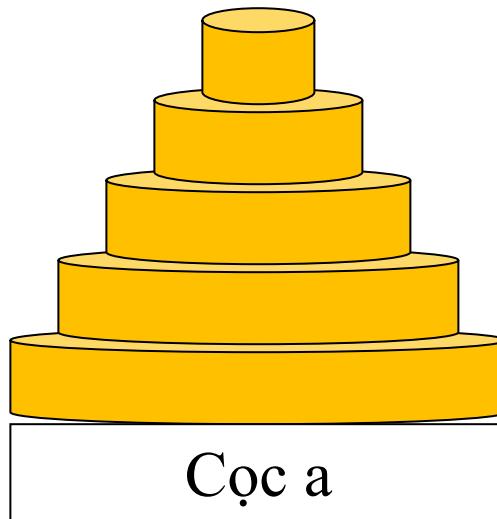
(1) Chuyển $n-1$ đĩa từ cọc a sang cọc b, sử dụng cọc c làm trung gian

Giải bài toán kích thước $n-1$: `HaNoiTower(n-1, a, b, c)`

(2) Chuyển 1 đĩa (đĩa với đường kính lớn nhất) từ cọc a sang cọc c

(3) Chuyển $n-1$ đĩa từ cọc b sang cọc c, sử dụng cọc a làm trung gian

Giải bài toán kích thước $n-1$: `HaNoiTower(n-1, b, c, a)`



VÍ DỤ: BÀI TOÁN THÁP HÀ NỘI

Thuật toán có thể mô tả trong thủ tục đệ qui sau đây:

HanoiTower(n, a, c, b) { //chuyển n đĩa từ cọc a sang cọc c sử dụng cọc b làm trung gian:

```
if (n==1) then <chuyển đĩa từ cọc a sang cọc c>
else {
    HanoiTower(n-1,a,b,c);
    HanoiTower(1,a,c,b);
    HanoiTower(n-1,b,c,a);
}
```

} Việc di chuyển đĩa gồm 3 giai đoạn:

(1) Chuyển $n-1$ đĩa từ cọc a sang cọc b, sử dụng cọc c làm trung gian

Giải bài toán kích thước $n-1$: **HaNoiTower**($n-1$, a, b, c)

(2) Chuyển 1 đĩa (đĩa với đường kính lớn nhất) từ cọc a sang cọc c

Giải bài toán kích thước $n = 1$ (chỉ cần 1 bước di chuyển đĩa): **HaNoiTower**(1, a, c, b)

(3) Chuyển $n-1$ đĩa từ cọc b sang cọc c, sử dụng cọc a làm trung gian

Giải bài toán kích thước $n-1$: **HaNoiTower**($n-1$, b, c, a)



VÍ DỤ: BÀI TOÁN THÁP HÀ NỘI

```
1 #include<stdio.h>
2
3 int i = 0;
4
5 void HanoiTower (int n, char xuatphat, char dich, char trunggian)
6 {
7     if (n == 1) {
8         printf("Dich chuyen dia tu coc %c den coc %c\n",xuatphat,dich);
9         i++;
10        return;
11    } else {
12        HanoiTower (n-1, xuatphat, trunggian, dich);
13        HanoiTower (1, xuatphat, dich, trunggian);
14        HanoiTower (n-1, trunggian, dich, xuatphat);
15    }
16 }
17
18 int main()
19 {
20     int n;
21     printf("Nhập số đĩa n = "); scanf("%d", &n);
22     HanoiTower (n, 'a', 'c', 'b');
23     printf("Tổng số lần di chuyển đĩa = %d", i);
24     return 0;
25 }
```

```
Nhập số đĩa n = 3
Dich chuyen dia tu coc a den coc c
Dich chuyen dia tu coc a den coc b
Dich chuyen dia tu coc c den coc b
Dich chuyen dia tu coc a den coc c
Dich chuyen dia tu coc b den coc a
Dich chuyen dia tu coc b den coc c
Dich chuyen dia tu coc a den coc c
Tổng số lần di chuyển đĩa = 7
```



PHÂN TÍCH THUẬT TOÁN ĐỆ QUY

- Để phân tích thuật toán đệ quy ta thường tiến hành như sau:
 - Gọi $T(n)$ là thời gian tính của thuật toán
 - Xây dựng công thức đệ quy cho $T(n)$
 - Giải công thức đệ quy thu được để đưa ra đánh giá cho $T(n)$
- Nói chung ta chỉ cần một đánh giá sát cho tốc độ tăng của $T(n)$ nên việc **giải công thức đệ quy** đối với $T(n)$ là đưa ra đánh giá tốc độ tăng của $T(n)$ trong ký hiệu tiệm cận.



PHÂN TÍCH THUẬT TOÁN ĐỆ QUY

Ví dụ: Tính $n!$ theo công thức đệ quy:

$$f(0) = 1$$

$$f(n) = n * f(n-1)$$

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

- Gọi $T(n)$ là số phép toán nhân phải thực hiện trong lệnh gọi `factorial(n)`.
- Ta có:

$$T(0) = 0,$$

$$T(n) = T(n-1) + 1, n \geq 1$$



PHÂN TÍCH THUẬT TOÁN ĐỆ QUY

- Gọi $T(n)$ là số phép toán nhân phải thực hiện trong lệnh gọi factorial(n). Ta có:
 $T(0) = 0,$
 $T(n) = T(n-1) + 1, n \geq 1$
- Giải công thức đệ quy $T(n)$, ta có:

$$\begin{aligned} T(n) &= T(n-1) + 1 && \text{thay thế } T(n-1) \\ &= T(n-2) + 1 + 1 && \text{thay thế } T(n-2) \\ &= T(n-3) + 1 + 1 + 1 \\ &= T(n-3) + 3 \\ &= \dots \\ &= T(n-k) + k \end{aligned}$$

$$\begin{aligned} T(n) &= T(n-n) + n && \text{chọn } k = n \\ &= T(0) + n \\ &= n \end{aligned}$$

Vậy $T(n) = O(n)$



ĐỆ QUY CÓ NHỚ

- Bài toán con trùng lặp

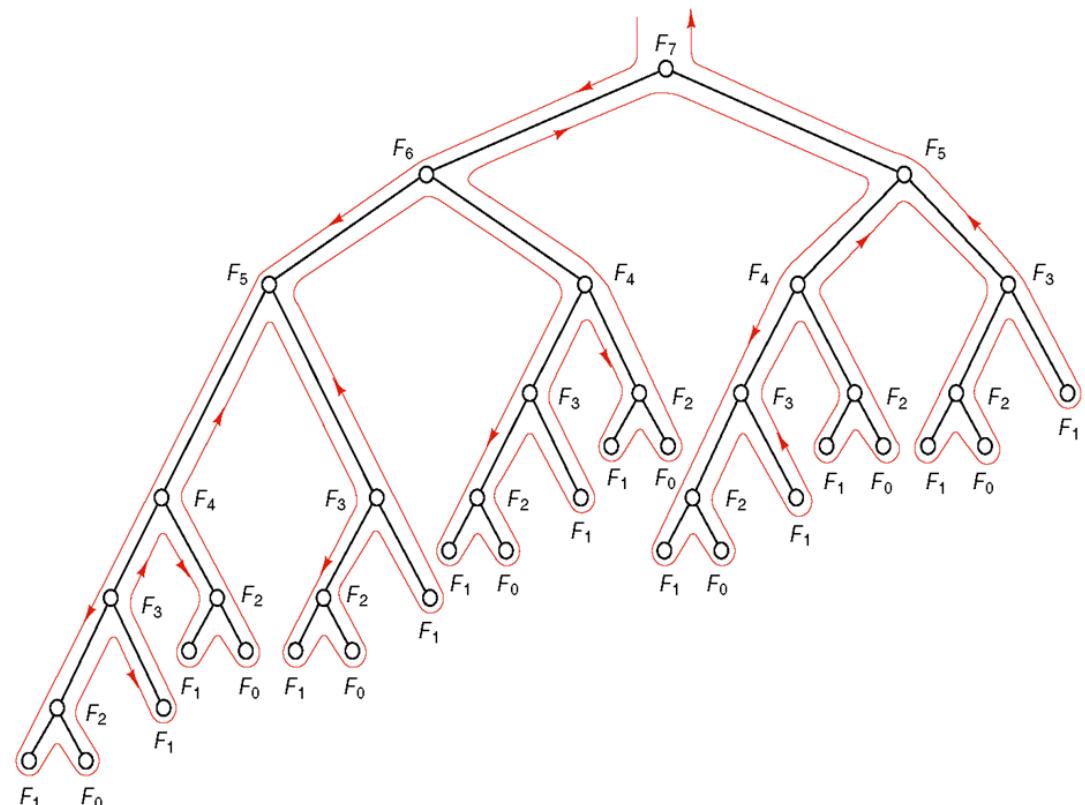
Ví dụ 1: Tính dãy số Fibonacci:

$$F(0) = 0; F(1) = 1$$

$$F(n) = F(n-1) + F(n-2) \text{ với } n \geq 2$$

- Trong thuật toán đệ quy, mỗi khi cần đến lời giải của một bài toán con ta lại phải trả về nó một cách đệ quy. Do đó, có những bài toán con bị giải đi giải lại nhiều lần. Điều đó dẫn đến tính kém hiệu quả của thuật toán. Hiện tượng này gọi là hiện tượng bài toán con trùng lặp.

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



ĐỆ QUY CÓ NHỚ

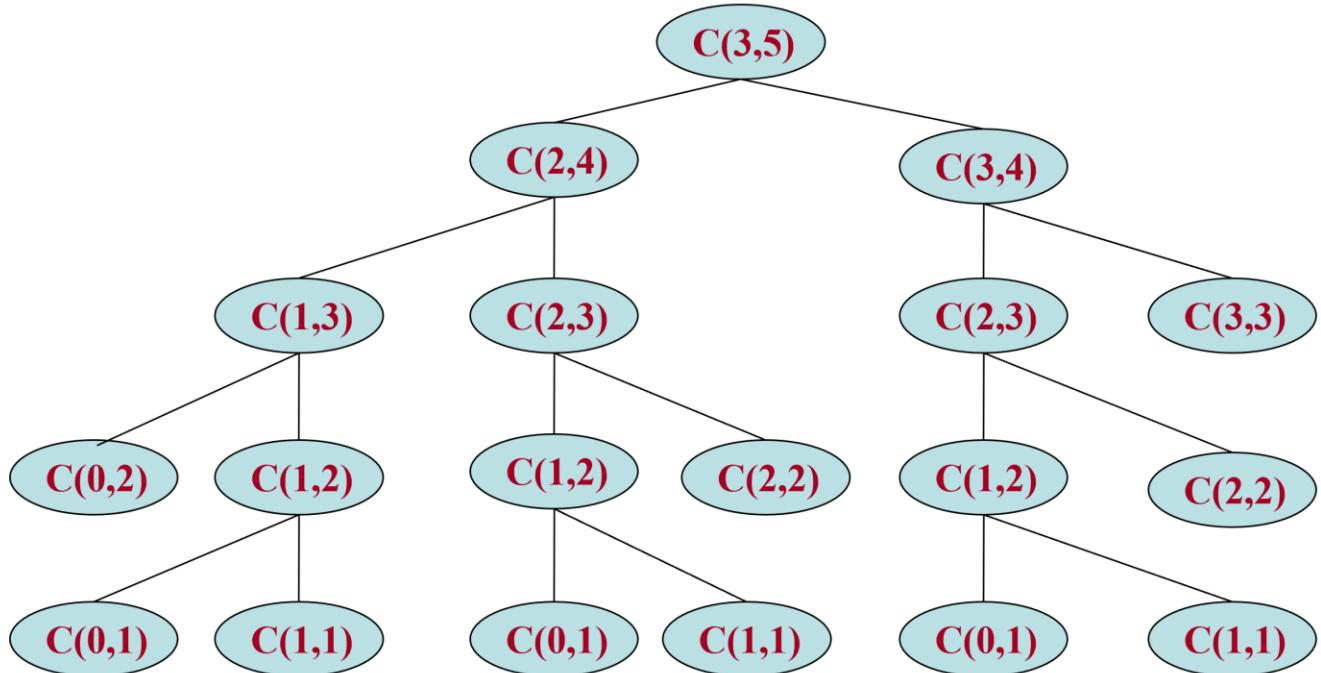
- Bài toán con trùng lặp

Ví dụ 2: Tính hệ số nhị thức:

$$C(0,n) = 1, \quad C(n,n) = 1; \quad \text{với mọi } n \geq 0,$$

$$C(k,n) = C(k-1,n-1) + C(k,n-1), \quad 0 < k < n$$

```
int C(int k, int n){  
    if (k == 0 || k == n) return 1;  
    else return C(k-1,n-1)+C(k,n-1);  
}
```



ĐỆ QUY CÓ NHỚ

- Trong hai ví dụ trên, ta đã thấy các thuật toán đệ quy để tính số Fibonacci và tính hệ số nhị thức là kém hiệu quả.
- Để tăng hiệu quả của các thuật toán đệ quy, ta có thể sử dụng kỹ thuật **đệ quy có nhớ**.
 - Sử dụng kỹ thuật đệ quy có nhớ, trong nhiều trường hợp, ta giữ nguyên được cấu trúc đệ quy của thuật toán và đồng thời lại đảm bảo được hiệu quả của nó. Nhược điểm lớn nhất của cách làm này là đòi hỏi về bộ nhớ.



ĐỆ QUY CÓ NHỚ

- **Ý tưởng của đệ quy có nhớ:**

- Dùng biến ghi nhớ lại thông tin về lời giải của các bài toán con ngay sau lần đầu tiên nó được giải. Điều đó cho phép rút ngắn thời gian tính của thuật toán, bởi vì, mỗi khi cần đến có thể tra cứu mà không phải giải lại những bài toán con đã được giải trước đó.



ĐỆ QUY CÓ NHỚ

- Ví dụ 1: Tính dãy số Fibonacci:

$$F(0) = 0; F(1) = 1$$

$$F(n) = F(n-1) + F(n-2) \text{ với } n \geq 2$$

Đệ quy không có nhớ:

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

Đệ quy có nhớ:

```
void init() {  
    M[0] = 0; M[1] = 1;  
    for (int i = 2; i <= n; i++) M[i] = 0;  
}  
  
int F(int n){  
    if (n != 0 && M[n] == 0)  
        M[n] = F(n-1) + F(n-2);  
    return M[n];  
}
```

Trước khi gọi hàm $F(n)$ cần gọi hàm $init()$ để khởi tạo các phần tử trong mảng $M[]$ như sau:

$M[0] = 0, M[1] = 1,$
 $M[i] = 0, \text{ với } i \geq 2.$



ĐỆ QUY CÓ NHỚ

- Ví dụ 2: Tính hệ số nhị thức:

$$C(0,n) = 1, \quad C(n,n) = 1; \quad \text{với mọi } n \geq 0,$$

$$C(k,n) = C(k-1,n-1) + C(k,n-1), \quad 0 < k < n$$

Đệ quy không có nhớ:

```
int C(int k, int n){  
    if (k == 0 || k == n)  
        return 1;  
    else  
        return C(k-1,n-1) + C(k,n-1);  
}
```

Đệ quy có nhớ:

```
void init() {  
    for (int i = 0; i <= k; i++)  
        for(int j = 0; j <= n; j++) M[i][j] = 0;  
}  
  
int C(int k, int n) {  
    if (k == 0 || k == n) M[k][n] = 1;  
    else {  
        if(M[k][n] == 0) M[k][n] = C(k-1,n-1) + C(k,n-1);  
    }  
    return M[k][n];  
}
```

Trước khi gọi hàm $C(k, n)$ cần gọi hàm $init()$ để khởi tạo các phần tử trong mảng $M[][] = 0$





HUST

THANK YOU !

HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT



ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

TUẦN 3 : ĐỆ QUY QUAY LUI

ONE LOVE. ONE FUTURE.

NỘI DUNG

- Sơ đồ chung đệ quy quay lui
- Bài toán liệt kê xâu nhị phân
- Bài toán liệt kê hoán vị
- Bài toán điền số Sudoku



SƠ ĐỒ CHUNG

- Bài toán liệt kê tổ hợp: Liệt kê các bộ $x = (x[1], x[2], \dots, x[k], x[k+1], \dots, x[n])$ với $x[i] \in A_i$, $i = 1, 2, \dots, n$ và thỏa mãn tập các ràng buộc P cho trước.
- Ví dụ:
 - “Bài toán liệt kê xâu nhị phân độ dài n ” dẫn về liệt kê các bộ $x = (x[1], x[2], \dots, x[k], x[k+1], \dots, x[n])$ với $x[i] \in \{0, 1\}$, $i = 1, 2, \dots, n$
 - “Bài toán liệt kê xâu nhị phân độ dài n có số lượng bit 0 là một số chẵn” dẫn về liệt kê các bộ $x = (x[1], x[2], \dots, x[k], x[k+1], \dots, x[n])$ với $x[i] \in \{0, 1\}$, $i = 1, 2, \dots, n$ và thỏa mãn ràng buộc: số lượng phần tử $x[i] = 0$ với $i = 1, 2, \dots, n$ là một số chẵn.
- Thuật toán quay lui cho phép giải các bài toán liệt kê tổ hợp. Có hai cách cài đặt thuật toán quay lui: đệ quy hoặc không đệ quy.



SƠ ĐỒ CHUNG

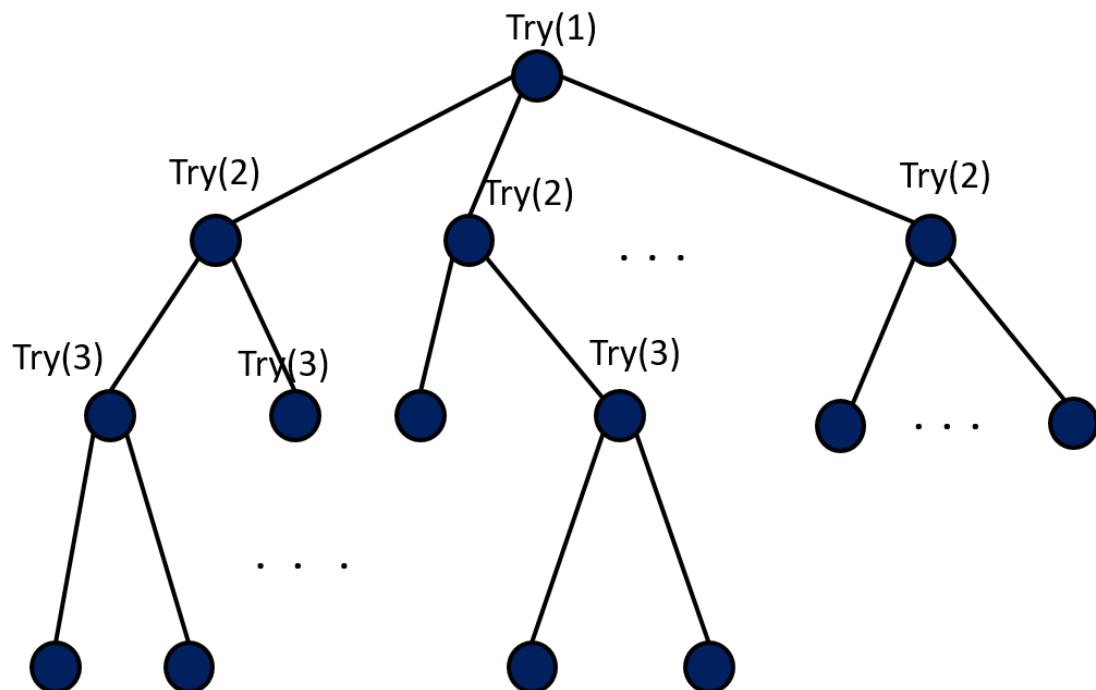
- Bài toán liệt kê tổ hợp: Liệt kê các bộ $x = (x[1], x[2], \dots, x[k], x[k+1], \dots, x[n])$ với $x[i] \in A_i$, $i = 1, 2, \dots, n$ và thỏa mãn tập các ràng buộc P cho trước.
- Lệnh gọi để thực hiện thuật toán đệ quy quay lui là: **Try(1);**
- Nếu chỉ cần tìm một lời giải thì cần tìm cách chấm dứt các thủ tục gọi đệ qui lồng nhau sinh bởi lệnh gọi Try(1) sau khi ghi nhận được lời giải đầu tiên.
- Nếu kết thúc thuật toán mà ta không thu được một lời giải nào thì điều đó có nghĩa là bài toán không có lời giải.

```
Try(k){//thử các giá trị có thể gán cho x[k]
    for v in candidates(k) do {
        if (check(v,k)) then {
            x[k] = v;
            [Update the data structure D]
            if (k == n) then solution();
            else Try(k+1);
            [Recover the data structure D]
        }
    }
}
```



SƠ ĐỒ CHUNG

- Bài toán liệt kê tổ hợp: Liệt kê các bộ $x = (x[1], x[2], \dots, x[k], x[k+1], \dots, x[n])$ với $x[i] \in A_i$, $i = 1, 2, \dots, n$ và thỏa mãn tập các ràng buộc P cho trước.
- Lệnh gọi để thực hiện thuật toán đệ quy quay lui là: **Try(1);**



```
Try(k){ // thử các giá trị có thể gán cho x[k]
    for v in candidates(k) do {
        if (check(v,k)) then {
            x[k] = v;
            [Update the data structure D]
            if (k == n) then solution();
            else Try(k+1);
            [Recover the data structure D]
        }
    }
}
```

BÀI TOÁN LIỆT KÊ XÂU NHỊ PHÂN

- Cho số nguyên dương $n \geq 1$. Hãy liệt kê tất cả các xâu nhị phân độ dài n theo thứ tự từ điển.
- Ví dụ: $n = 3$, ta có các xâu nhị phân độ dài 3 cần liệt kê theo thứ tự như sau:
 - 000
 - 001
 - 010
 - 011
 - 100
 - 101
 - 110
 - 111



BÀI TOÁN LIỆT KÊ XÂU NHỊ PHÂN

- Cho số nguyên dương $n \geq 1$. Hãy liệt kê tất cả các xâu nhị phân độ dài n theo thứ tự từ điển.
- Biểu diễn lời giải: mỗi xâu nhị phân được biểu diễn bởi mảng $(x[1], x[2], \dots, x[n])$ trong đó $x[k] \in \{0,1\}$ là bít thứ k trong xâu nhị phân.

```
Try(k){ //thử các giá trị có thể gán cho x[k]
    for v in candidates(k) do {
        if (check(v,k)) then {
            x[k] = v;
            [Update the data structure D]
            if (k == n) then solution();
            else Try(k+1);
            [Recover the data structure D]
        }
    }
}
```

Cần xác định:
▪ candidates(k)
▪ check(v, k)

```
void Try(int k){
    for (int v = 0; v <= 1; v++){
        x[k] = v;
        if (k == n) solution();
        else Try(k+1);
    }
}
```



BÀI TOÁN LIỆT KÊ XÂU NHỊ PHÂN

- Cho số nguyên dương $n \geq 1$. Hãy liệt kê tất cả các xâu nhị phân độ dài n theo thứ tự từ điển.
- Biểu diễn lời giải: mỗi xâu nhị phân được biểu diễn bởi mảng $(x[1], x[2], \dots, x[n])$ trong đó $x[k] \in \{0,1\}$ là bít thứ k trong xâu nhị phân.

```
Try(k){ // thử các giá trị có thể gán cho x[k]
    for v in candidates(k) do {
        if (check(v,k)) then {
            x[k] = v;
            [Update the data structure D]
            if (k == n) then solution();
            else Try(k+1);
            [Recover the data structure D]
        }
    }
}
```

Cần xác định:
▪ candidates(k)
▪ check(v, k)

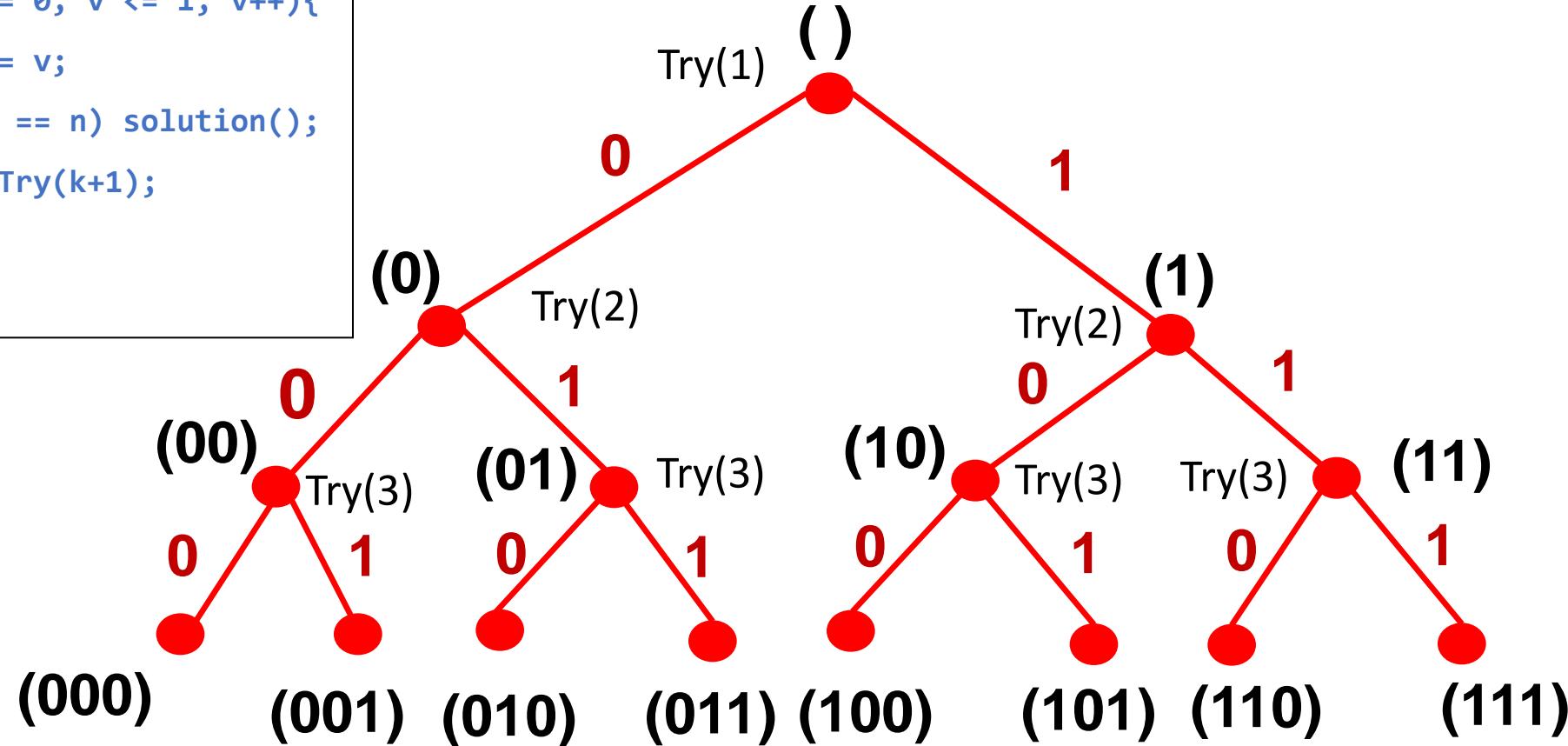
```
void Try(int k){
    for (int v = 0; v <= 1; v++){
        x[k] = v;
        if (k == n) solution();
        else Try(k+1);
    }
}
```



BÀI TOÁN LIỆT KÊ XÂU NHỊ PHÂN

```
void Try(int k){  
    for (int v = 0; v <= 1; v++){  
        x[k] = v;  
        if (k == n) solution();  
        else Try(k+1);  
    }  
}
```

Cây liệt kê các xâu nhị phân độ dài 3



BÀI TOÁN LIỆT KÊ HOÁN VỊ

- Cho số nguyên dương $n \geq 1$. Hãy liệt kê tất cả các hoán vị của n số $1, 2, \dots, n$ theo thứ tự từ điển.
- Ví dụ: $n = 3$, ta có các hoán vị của $1, 2, 3$ theo thứ tự từ điển như sau:
 - $(1, 2, 3)$
 - $(1, 3, 2)$
 - $(2, 1, 3)$
 - $(2, 3, 1)$
 - $(3, 1, 2)$
 - $(3, 2, 1)$



BÀI TOÁN LIỆT KÊ HOÁN VỊ

- Biểu diễn lời giải: mỗi hoán vị n phần tử được biểu diễn bởi mảng $(x[1], x[2], \dots, x[n])$ trong đó:

- $x[k] \in \{1, 2, \dots, n\}$ là phần tử thứ k trong hoán vị

- $x[k] \neq x[1], x[2], \dots, x[k-1], x[k+1], \dots, x[n]$

try(k)*// thử các giá trị có thể gán cho $x[k]$*

```
try(k){  
    for v in candidates(k) do {  
        if (check(v, k)) then {  
            x[k] = v;  
            [Update the data structure D]  
            if (k == n) then solution();  
            else try(k+1);  
            [Recover the data structure D]  
        }  
    }  
}
```

Cần xác định:
▪ candidates(k)
▪ check(v, k)

```
void Try(int k){  
    for (int v = 1; v <= n; v++){  
        if (check(v, k)) {  
            x[k] = v;  
            if (k == n) solution();  
            else Try(k+1);  
        }  
    }  
}
```



BÀI TOÁN LIỆT KÊ HOÁN VỊ

```
#include <stdio.h>

int n;
int x[100];
void solution(){
    for(int k = 1; k <= n; k++)
        printf("%d ",x[k]);
    printf("\n");
}
int check(int v, int k) {
    for (int i = 1; i <= k-1; i++)
        if (x[i] == v) return 0;
    return 1;
}
```

```
void Try(int k){
    for (int v = 1; v <= n; v++){
        if (check(v, k)) {
            x[k] = v;
            if (k == n) solution();
            else Try(k+1);
        }
    }
}
int main(){
    scanf("%d",&n);
    Try(1);
}
```



BÀI TOÁN LIỆT KÊ HOÁN VỊ

- Kỹ thuật đánh dấu
 - $\text{used}[v] = 1$: v đã xuất hiện
 - $\text{used}[v] = 0$: v chưa xuất hiện

```
try(k){//thử các giá trị có thể gán cho  $x[k]$ 
    for  $v$  in candidates( $k$ ) do {
        if (check( $v,k$ )) then {
             $x[k] = v;$ 
            [Update the data structure  $D$ ]
            if ( $k == n$ ) then solution();
            else try( $k+1$ );
            [Recover the data structure  $D$ ]
        }
    }
}
```

```
void Try(int k){
    for(int v = 1; v <= n; v++){
        if (used[v]==0){
            x[k] = v;
            used[v] = 1;
            if (k == n) solution();
            else Try(k+1);
            used[v] = 0;
        }
    }
}

int main(){
    scanf("%d",&n);
    for(int v = 1; v <= n; v++) used[v] = 0;
    Try(1);
}
```



BÀI TOÁN LIỆT KÊ HOÁN VỊ

```
#include <stdio.h>
int n;
int x[100];
int used[100];
void solution(){
    for(int k = 1; k <= n; k++)
        printf("%d ",x[k]);
    printf("\n");
}
```

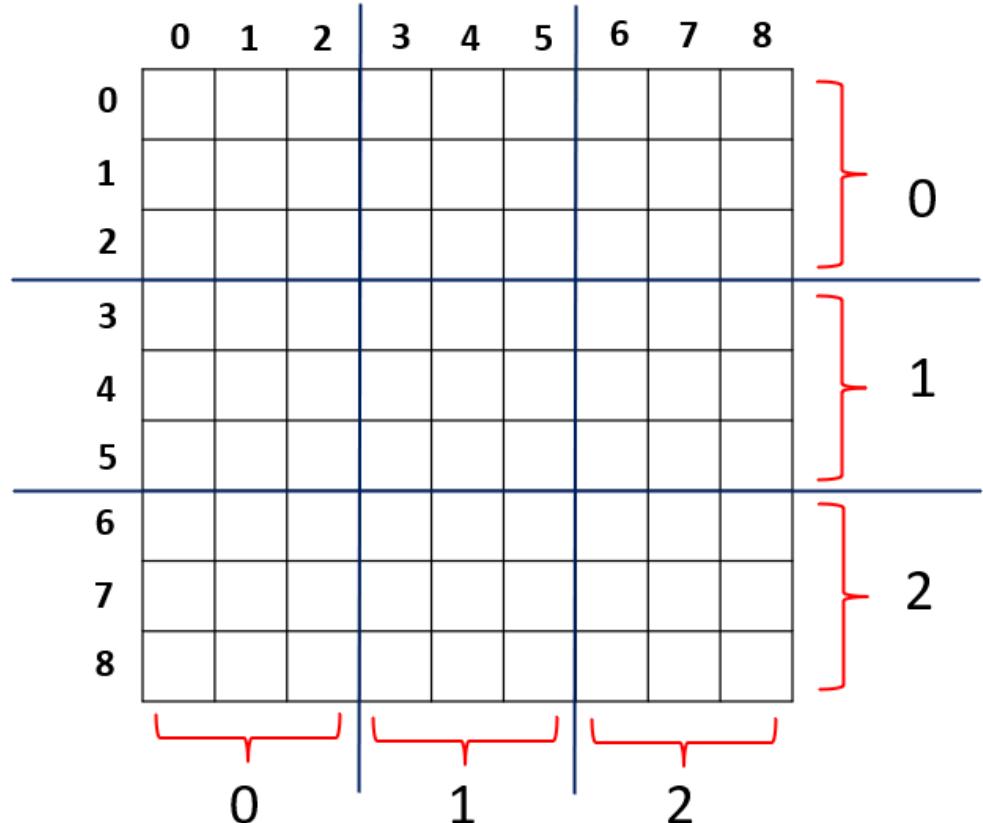
```
void Try(int k){
    for(int v = 1; v <= n; v++){
        if (used[v]==0){
            x[k] = v;
            used[v] = 1;
            if (k == n) solution();
            else Try(k+1);
            used[v] = 0;
        }
    }
}
int main(){
    scanf("%d",&n);
    for(int v = 1; v <= n; v++) used[v] = 0;
    Try(1);
}
```



BÀI TOÁN ĐIỀN SỐ SUDOKU

- Phát biểu bài toán:

- Cho lưới ô vuông 9×9 được chia thành 9 lưới 3×3 .
- Các hàng, cột được đánh số 0, 1, 2, ..., 8
- Liệt kê tất cả các cách điền các số 1, 2, ..., 9 vào các ô thuộc lưới ô vuông 9×9 sao cho:
 - Các số trên mỗi dòng là khác nhau,
 - Các số trên mỗi cột là khác nhau,
 - Các số trên mỗi lưới 3×3 là khác nhau.



BÀI TOÁN ĐIỀN SỐ SUDOKU

- Phát biểu bài toán:

- Cho lưới ô vuông 9×9 được chia thành 9 lưới 3×3 .
- Các hàng, cột được đánh số 0, 1, 2, ..., 8
- Liệt kê tất cả các cách điền các số 1, 2, ..., 9 vào các ô thuộc lưới ô vuông 9×9 sao cho:
 - Các số trên mỗi dòng là khác nhau,
 - Các số trên mỗi cột là khác nhau,
 - Các số trên mỗi lưới 3×3 là khác nhau.

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	1	4	3	6	5	8	9	7
3	6	5	8	9	7	2	1	4
8	9	7	2	1	4	3	6	5
5	3	1	6	4	2	9	7	8
6	4	2	9	7	8	5	3	1
9	7	8	5	3	1	6	4	2

BÀI TOÁN ĐIỀN SỐ SUDOKU

- Biểu diễn lời giải: $X[i, j]$ là giá trị số được điền vào ô hàng i cột j ($i, j = 0, 1, 2, \dots, 8$)
- Mảng đánh dấu
 - $\text{markR}[r, v] = 1$: giá trị v đã xuất hiện trên hàng r và $\text{markR}[r, v] = 0$, ngược lại ($r = 0, 1, \dots, 8$ và $v = 1, 2, \dots, 9$)
 - $\text{markC}[c, v] = 1$: giá trị v đã xuất hiện trên cột c và $\text{markC}[c, v] = 0$, ngược lại ($c = 0, 1, 2, \dots, 8$ và $v = 1, 2, \dots, 9$)
 - $\text{markS}[i, j, v] = 1$: giá trị v đã xuất hiện trong lưới 3×3 hàng thứ i và cột thứ j và $\text{markS}[i, j, v] = 0$, ngược lại ($i, j = 0, 1, 2$ và $v = 1, 2, \dots, 9$)

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	1	4	3	6	5	8	9	7
3	6	5	8	9	7	2	1	4
8	9	7	2	1	4	3	6	5
5	3	1	6	4	2	9	7	8
6	4	2	9	7	8	5	3	1
9	7	8	5	3	1	6	4	2

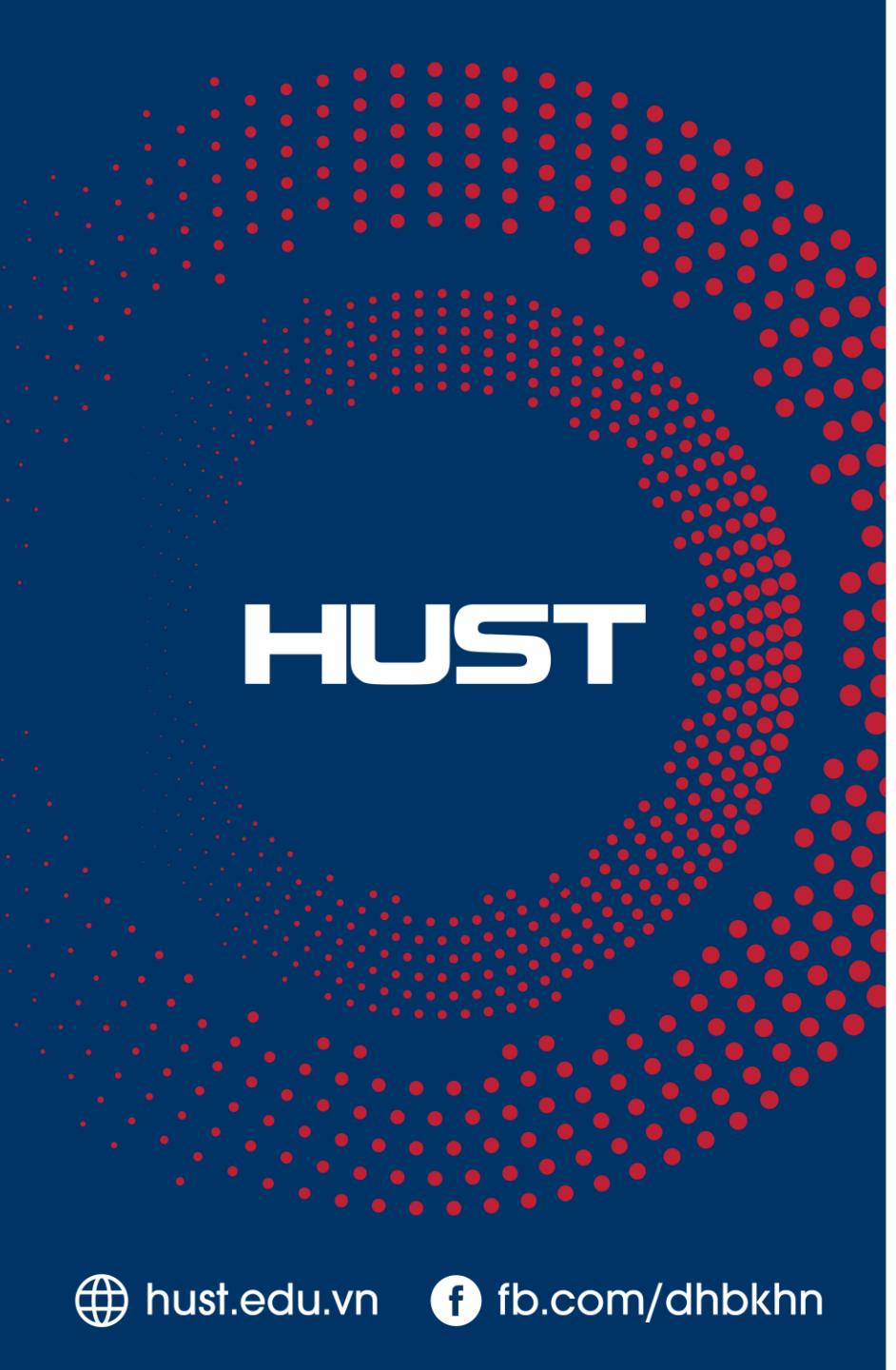
BÀI TOÁN ĐIỀN SỐ SUDOKU

- Thứ tự duyệt các ô để thử giá trị: từ trên xuống dưới và từ trái qua phải
- Hàm đệ quy Try(r, c): thử giá trị cho ô hàng r cột c

```
check(v, r, c){  
    if markR[r,v] = 1 then return 0;  
    if markC[c,v] = 1 then return 0;  
    if markS[r/3,c/3,v] = 1 then return 0;  
    return 1;  
}
```

```
Try(r, c){  
    for v = 1 to 9 do {  
        if (check(v,r,c)) then {  
            X[r,c] = v;  
            markR[r,v] = 1; markC[c,v] = 1; markS[r/3,c/3,v] = 1;  
            if r = 8 and c = 8 then solution();  
            else {  
                if c = 8 then Try(r+1, 0); else Try(r, c+1);  
            }  
            markR[r,v] = 0; markC[c,v] = 0; markS[r/3,c/3,v] = 0;  
        }  
    }  
}
```



The background of the slide features a dark blue vertical bar on the left side. On this bar, there is a graphic element consisting of a series of red dots arranged in a curved, flowing pattern that tapers towards the top right.

HUST

THANK YOU !

HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN



ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

TUẦN 4 : NHÁNH VÀ CÂN

ONE LOVE. ONE FUTURE.

NỘI DUNG

- Sơ đồ chung nhánh và cận
- Bài toán người du lịch



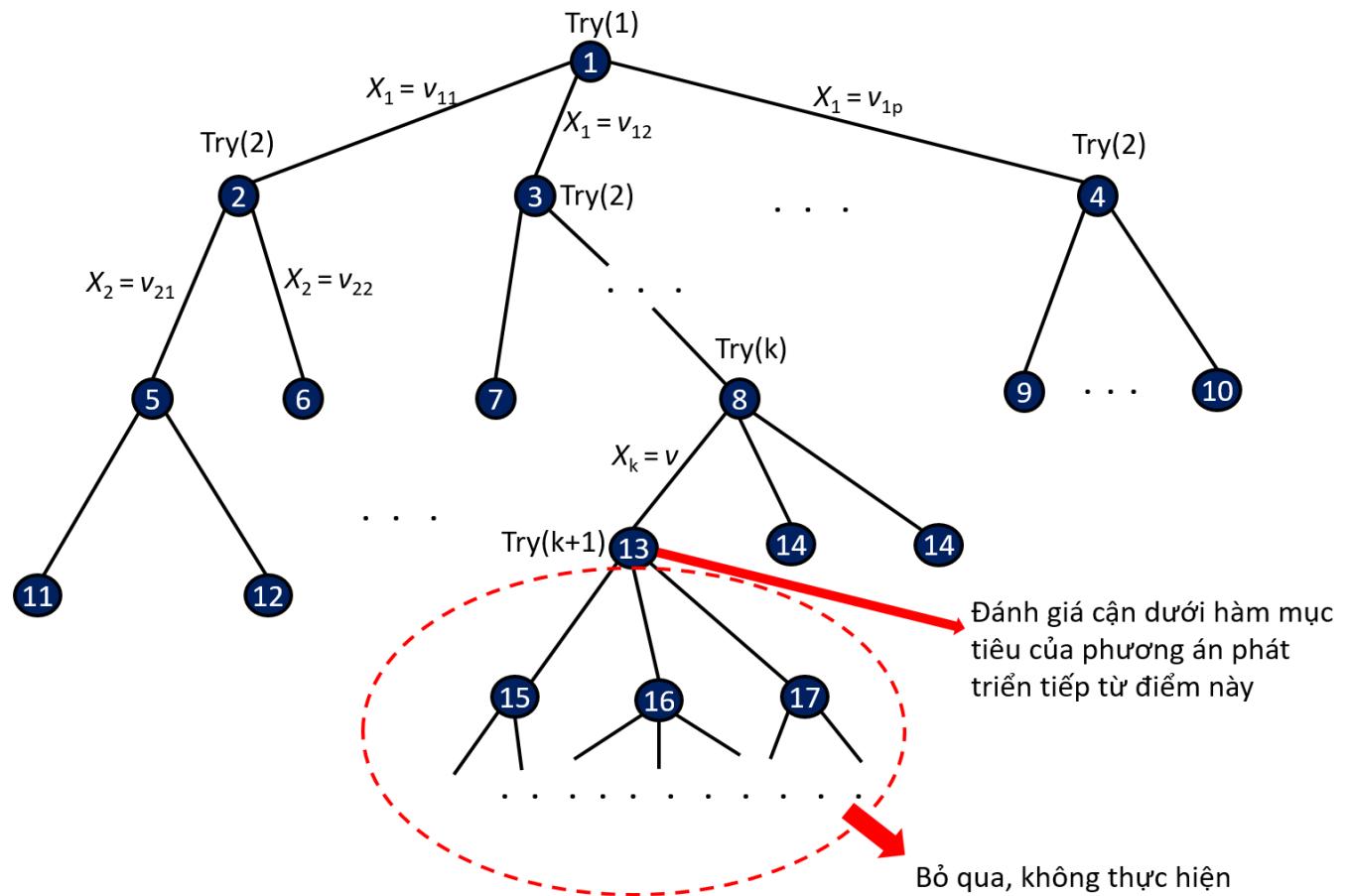
SƠ ĐỒ CHUNG

- Nhánh và cận (Branch and Bound): một trong số các phương pháp để giải bài toán tối ưu tổ hợp
 - Dùng kỹ thuật quay lui để liệt kê tất cả các phương án, từ đó giữ lại phương án tốt nhất
 - Dùng đánh giá cận (cận trên với bài toán tìm max và cận dưới đối với bài toán tìm min) để cắt bỏ bớt không gian tìm kiếm trong quá trình liệt kê



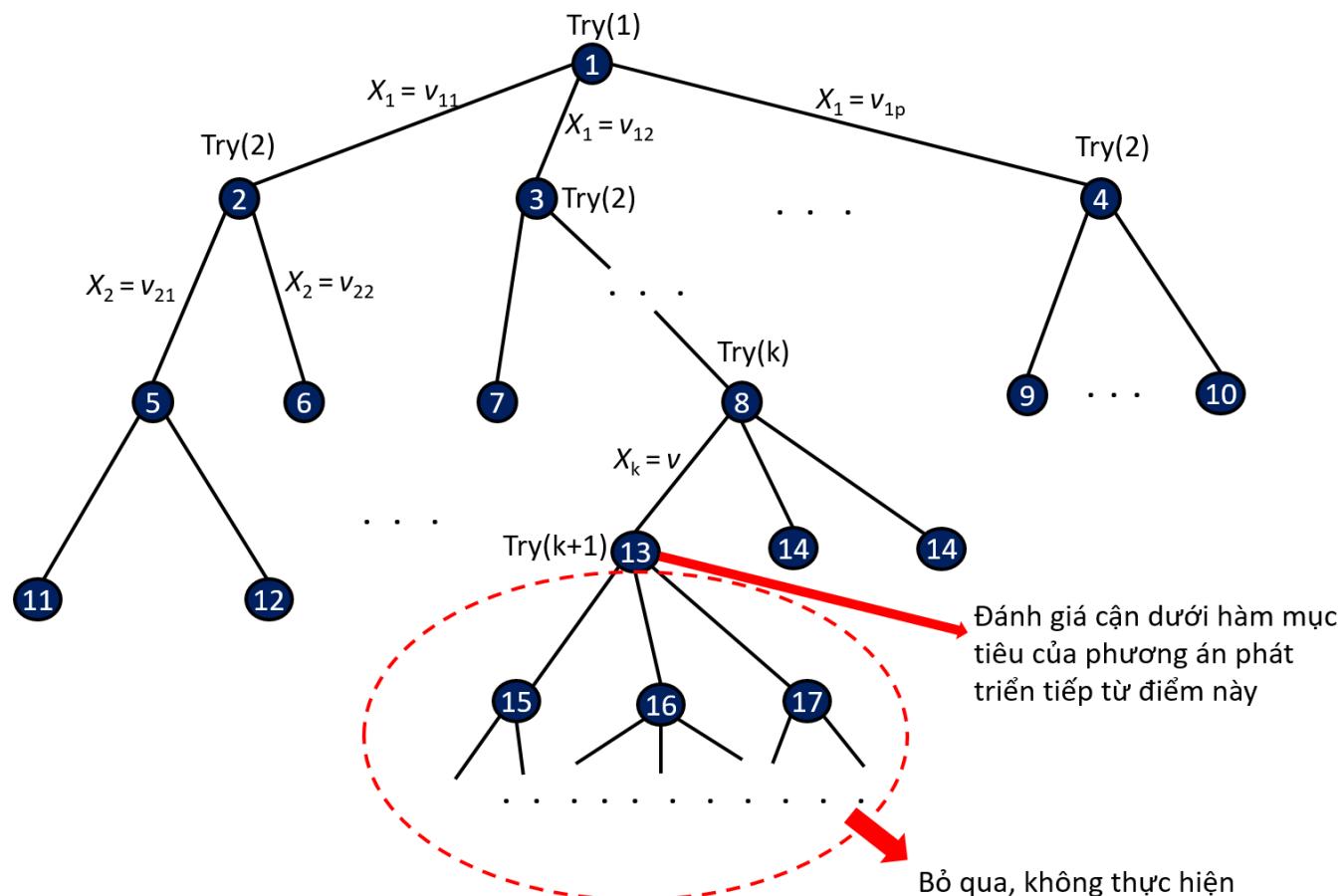
SƠ ĐỒ CHUNG

- Xét bài toán tìm cực tiểu của hàm mục tiêu trong đó lời giải được biểu diễn bởi 1 bộ các biến $X = (X_1, X_2, \dots, X_n)$.
- Hàm Try(k) dùng để thử giá trị cho biến X_k trong quá trình liệt kê
- Ký hiệu f^* : giá trị hàm mục tiêu của phương án tốt nhất đã tìm được



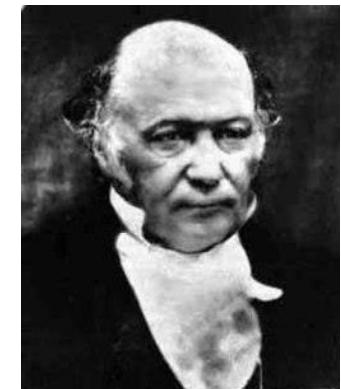
SƠ ĐỒ CHUNG

- Sau khi gán giá trị v cho X_k , ta đánh giá cận dưới g của hàm mục tiêu của các phương án phát triển tiếp từ điểm 13
- Nếu g lớn hơn hoặc bằng f^* thì không phát triển tiếp lời giải từ điểm 13



BÀI TOÁN NGƯỜI DU LỊCH

- Phát biểu bài toán:
 - Một người du lịch muốn đi tham quan n thành phố $1, 2, \dots, n$
 - *Hành trình là cách đi xuất phát từ thành phố 1 đi qua tất cả các thành phố còn lại, mỗi thành phố đúng một lần, rồi quay trở lại thành phố xuất phát 1.*
 - Biết $c(i, j)$ là chi phí đi từ thành phố i đến thành phố j ($i, j = 1, 2, \dots, n$)
 - Tìm hành trình với tổng chi phí là nhỏ nhất.
- Một số nhận xét:
 - Số lượng hành trình của người du lịch là $(n-1)!$
 - Ta có tương ứng 1-1 giữa một hành trình của người du lịch:
$$1 \rightarrow x[2] \rightarrow x[3] \rightarrow \dots \rightarrow x[n] \rightarrow 1$$
với một hoán vị $x = (x[2], x[3], \dots, x[n])$ của $n-1$ số tự nhiên $2, 3, \dots, n$.
 - Chi phí hành trình: $f(x) = c(1, x[2]) + c(x[2], x[3]) + \dots + c(x[n-1], x[n]) + c(x[n], 1)$



Sir William Rowan Hamilton [1]
(1805 – 1865)

BÀI TOÁN NGƯỜI DU LỊCH

- Giải bằng phương pháp duyệt toàn bộ:
 - Hành trình: $x = (1, x[2], x[3], \dots, x[n], 1)$

```
try(k){//thử các giá trị có thể gán cho x[k]
    for v in candidates(k) do {
        if (check(v,k)) then {
            x[k] = v;
            [Update the data structure D]
            if (k == n) then solution();
            else try(k+1);
            [Recover the data structure D]
        }
    }
}
```

Cần xác định:
1) candidates(k)
2) check(v, k)

```
Init:
•  $f^* = +\infty; f = 0; x[1] = 1;$ 
• for (int v = 2; v<=n; v++) visited[v]=0;

void Try(int k) {
    for (int v = 2; v<=n; v++) {
        if (!visited[v]) {
            x[k] = v;
            visited[v] = 1;
            f = f + c(x[k-1],x[k]);
            if (k == n) { //Update record
                ftemp = f + c(x[n],x[1]);
                if (ftemp < f*) f* = ftemp;
            }
            else Try(k + 1);
            f = f - c(x[k-1],x[k]);
            visited[v] = 0;
        }
    }
}
```



BÀI TOÁN NGƯỜI DU LỊCH

Giải bằng phương pháp nhánh và cân

Tính cận:

- Ký hiệu $c_{min} = \min \{ c(i, j) , i, j = 1, 2, \dots, n, i \neq j \}$ là chi phí đi lại nhỏ nhất giữa các thành phố
- Cân ước lượng chi phí hành trình đầy đủ cho nhánh hiện tại tương ứng với hành trình bộ phận $(1, u_2, \dots, u_k)$ đã đi qua k thành phố: $1 \rightarrow u_2 \rightarrow \dots \rightarrow u_{k-1} \rightarrow u_k$
- Nếu cận dưới $g(1, u_2, \dots, u_k) \geq f^*$ thì không đi tiếp từ hành trình bộ phận $(1, u_2, \dots, u_k)$



BÀI TOÁN NGƯỜI DU LỊCH

- Cần ước lượng lượng chi phí hành trình đầy đủ cho nhánh hiện tại tương ứng với hành trình bộ phận $(1, u_2, \dots, u_k)$ đã đi qua k thành phố: $1 \rightarrow u_2 \rightarrow \dots \rightarrow u_{k-1} \rightarrow u_k$
 - Chi phí phải trả theo hành trình bộ phận $(1, u_2, \dots, u_k)$ là
$$\sigma = c(1, u_2) + c(u_2, u_3) + \dots + c(u_{k-1}, u_k)$$
 - Để phát triển thành hành trình đầy đủ:
$$1 \rightarrow u_2 \rightarrow \dots \rightarrow u_{k-1} \rightarrow u_k \rightarrow u_{k+1} \rightarrow u_{k+2} \rightarrow \dots \rightarrow u_n \rightarrow 1$$
- Ta còn phải đi $n-k+1$ đoạn đường nữa, mỗi đoạn đường có chi phí không ít hơn c_{min} , nên đoạn đường chưa đi có chi phí ít ra là: $(n-k+1) c_{min}$
- Vậy nếu đã đi hành trình bộ phận $(1, u_2, \dots, u_k)$ thì đoạn đường còn lại dù đi thế nào thì tổng chi phí cũng lớn hơn hoặc bằng $g(1, u_2, \dots, u_k) = \sigma + (n-k+1) c_{min}$



BÀI TOÁN NGƯỜI DU LỊCH

- Hàm Try(k) tìm lời giải tối ưu cho bài toán người du lịch có sử dụng kỹ thuật **nhánh và cận**

```
Main(){  
    //Init:  
    f* = +∞; f = 0; x[1] = 1;  
    for v = 2 to n do visited[v] = false;  
    Try(2);  
    print(f*);  
}
```

```
Try(k) {  
    for v = 2 to n do {  
        if not visited[v] {  
            x[k] = v;  
            visited[v] = true;  
            f = f + c(x[k-1],x[k]);  
            if k = n then { //Update record  
                ftemp = f + c(x[n],x[1]);  
                if (ftemp < f*) f* = ftemp;  
            }  
            else {  
                g = f + (n-k+1)*cmin;  
                if g < f* then Try(k+1);  
            }  
            f = f - c(x[k-1],x[k]);  
            visited[v] = false;  
        }  
    }  
}
```



A large, faint watermark of the HUST logo is visible across the entire slide, consisting of a grid of red dots forming the letters 'HUST' and a stylized 'U' shape.

HUST

THANK YOU !

HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.



ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

Sơ đồ thuật toán tham lam, chia để trị

ONE LOVE. ONE FUTURE.

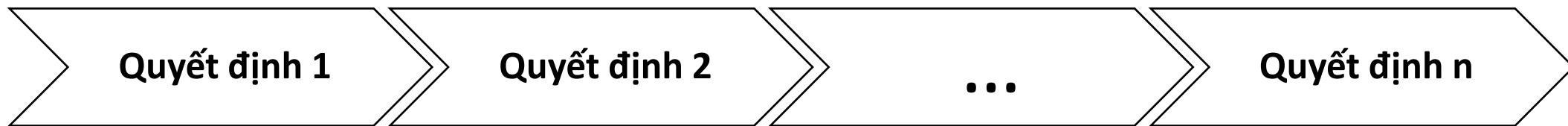
A. Thuật toán tham lam

B. Thuật toán chia để trị



Giới thiệu

- Thuật toán tham lam (Greedy algorithm) là cách tiếp cận **đơn giản và có tính trừu tượng cao** để giải các bài toán tối ưu, đặc biệt là tối ưu tổ hợp.
- Quá trình tìm lời giải bằng thuật toán tham lam được **chia thành nhiều giai đoạn**



- Thuật toán tham lam (Greedy algorithm) là cách tiếp cận **đơn giản và có tính trừu tượng cao** để giải các bài toán tối ưu, đặc biệt là tối ưu tổ hợp.
- Quá trình tìm lời giải bằng thuật toán tham lam được **chia thành nhiều giai đoạn**

Sơ đồ thuật toán

- Ký hiệu:
 - S : Lời giải cần tìm
 - C : Tập các ứng cử viên
 - $\text{select}(C)$: Hàm chọn ra ứng cử viên tiềm năng nhất
 - $\text{solution}(S)$: Hàm trả về TRUE nếu S là một lời giải hợp lệ, ngược lại hàm trả về FALSE
 - $\text{feasible}(S)$: Hàm trả về TRUE nếu S không vi phạm ràng buộc, ngược lại hàm trả về FALSE

```
Greedy() {  
    S = Ø;  
    while C ≠ Ø and not solution(S){  
        x = select(C);  
        C = C \ {x};  
        if feasible(S ∪ {x}) {  
            S = S ∪ {x};  
        }  
    }  
    return S;  
}
```



Sơ đồ chung thuật toán

Khởi tạo, tập lời giải rỗng

Chừng nào tập ứng cử viên còn khác rỗng
và S chưa phải là lời giải hợp lệ

Lựa chọn x là ứng cử viên tiềm năng nhất
và loại x khỏi tập ứng cử viên C

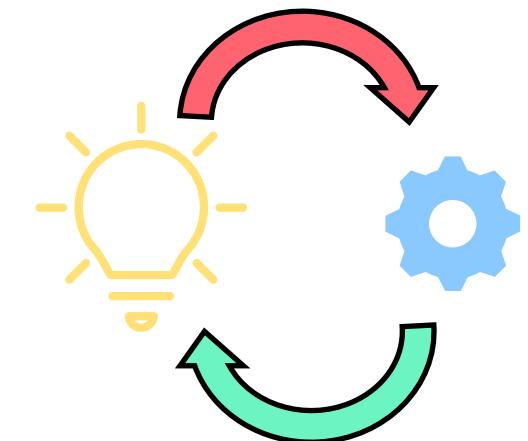
Nếu bổ sung x vào lời giải đang xây dựng
nếu không vi phạm ràng buộc

```
Greedy() {  
    S = Ø;  
    while C ≠ Ø and not solution(S){  
        x = select(C);  
        C = C \ {x};  
        if feasible(S ∪ {x}) {  
            S = S ∪ {x};  
        }  
    }  
    return S;  
}
```



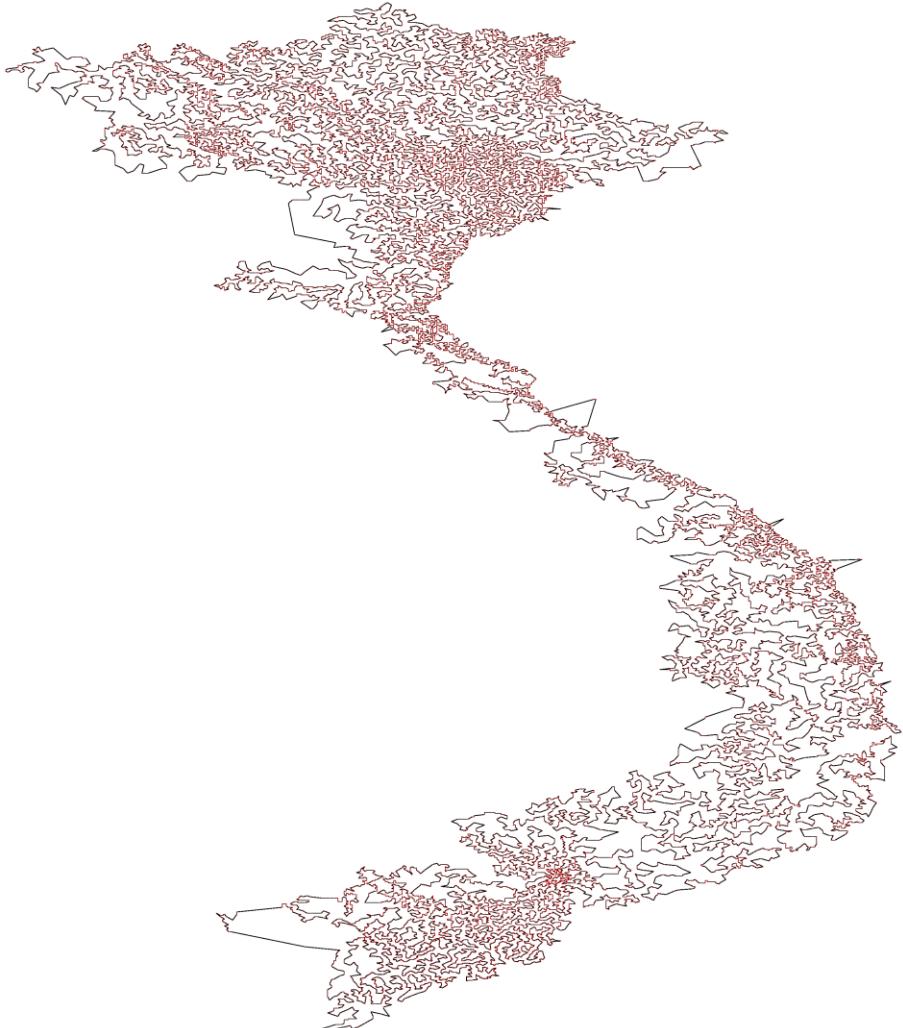
Đặc điểm

- Thuật toán đơn giản → Dễ đề xuất và cài đặt
- Trong một số bộ dữ liệu đầu vào và bài toán, thuật toán tham lam cho lời giải tối ưu. Tuy nhiên, đa số trường hợp, đặc biệt là các bài toán thực tế có độ phức tạp cao, thuật toán này không đảm bảo lời giải trả ra là tối ưu
- Thuật toán khá hiệu quả trong những bài toán thực tế có nhiều ràng buộc và độ phức tạp cao.



Ví dụ: Người giao hàng

- Bài toán người bán hàng (Traveling Salesman Problem – TSP) là một bài toán tối ưu kinh điển trong lĩnh vực tối ưu tổ hợp
- Bài toán thuộc lớp NP-Hard (Chưa tồn tại thuật toán chạy trong thời gian đa thức giải hiểu quả)
- Phát biểu: Người bán hàng cần tìm một hành trình qua một tập hợp các thành phố sao cho hành trình có tổng chi phí di chuyển là nhỏ nhất và mỗi thành phố chỉ được ghé qua đúng một lần trước khi quay về thành phố xuất phát.

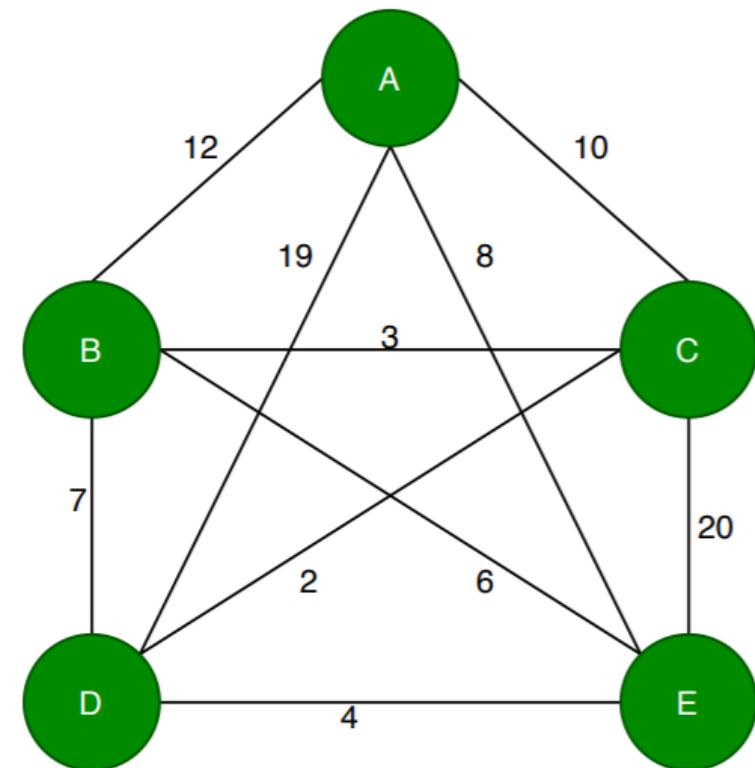


Nguồn: <https://www.math.uwaterloo.ca>

Ví dụ: Người giao hàng

- Thiết kế:

- S : Lời giải cần tìm
- C : Tập các ứng cử viên
- $\text{select}(C)$: Hàm chọn ra ứng cử viên tiềm năng nhất
- $\text{solution}(S)$: Hàm trả về TRUE nếu S là một lời giải hợp lệ, ngược lại hàm trả về FALSE
- $\text{feasible}(S)$: Hàm trả về TRUE nếu S không vi phạm ràng buộc, ngược lại hàm trả về FALSE



Ví dụ: Người giao hàng

- **Bài toán:** Thiết kế thuật toán tham lam cho bài toán TSP



- **Thiết kế:**

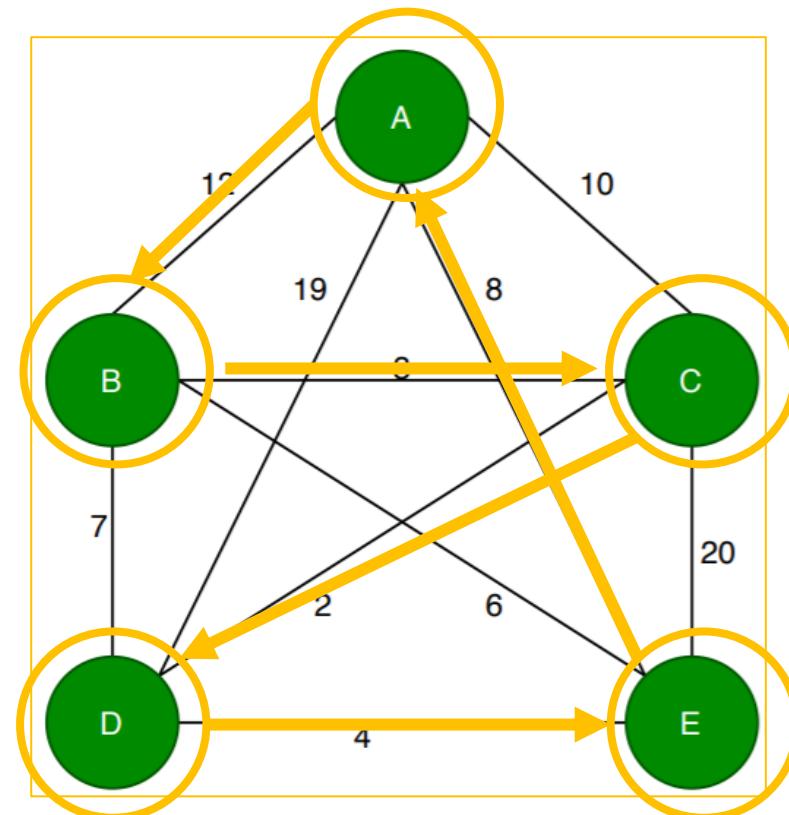
- S : Lời giải cần tìm
- C : Tập các ứng cử viên
- $\text{select}(C)$: Hàm chọn ra ứng cử viên tiềm năng nhất
- $\text{solution}(S)$: Hàm trả về TRUE nếu S là một lời giải hợp lệ, ngược lại hàm trả về FALSE
- $\text{feasible}(S)$: Hàm trả về TRUE nếu S không vi phạm ràng buộc, ngược lại hàm trả về FALSE

Ví dụ: Người giao hàng

- **Bài toán:** Thiết kế thuật toán tham lam cho bài toán TSP

- **Biểu diễn thuật toán:**

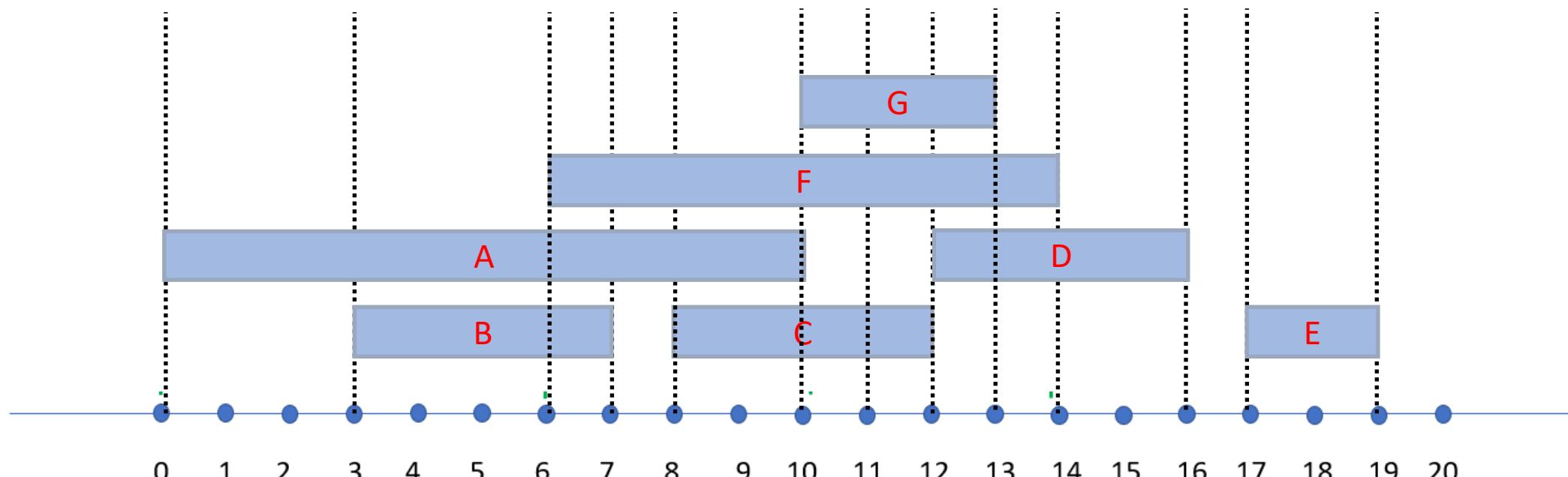
```
Greedy() {  
    S = Ø;  
    while C ≠ Ø and not  
        solution(S){  
        x = select(C);  
        C = C \ {x};  
        if feasible(S ∪ {x}) {  
            S = S ∪ {x};  
        }  
    }  
    return S;  
}
```



Xuất phát từ B: B → C → D → E → A → B

Ví dụ: Bài toán tập đoạn con không giao nhau cực đại

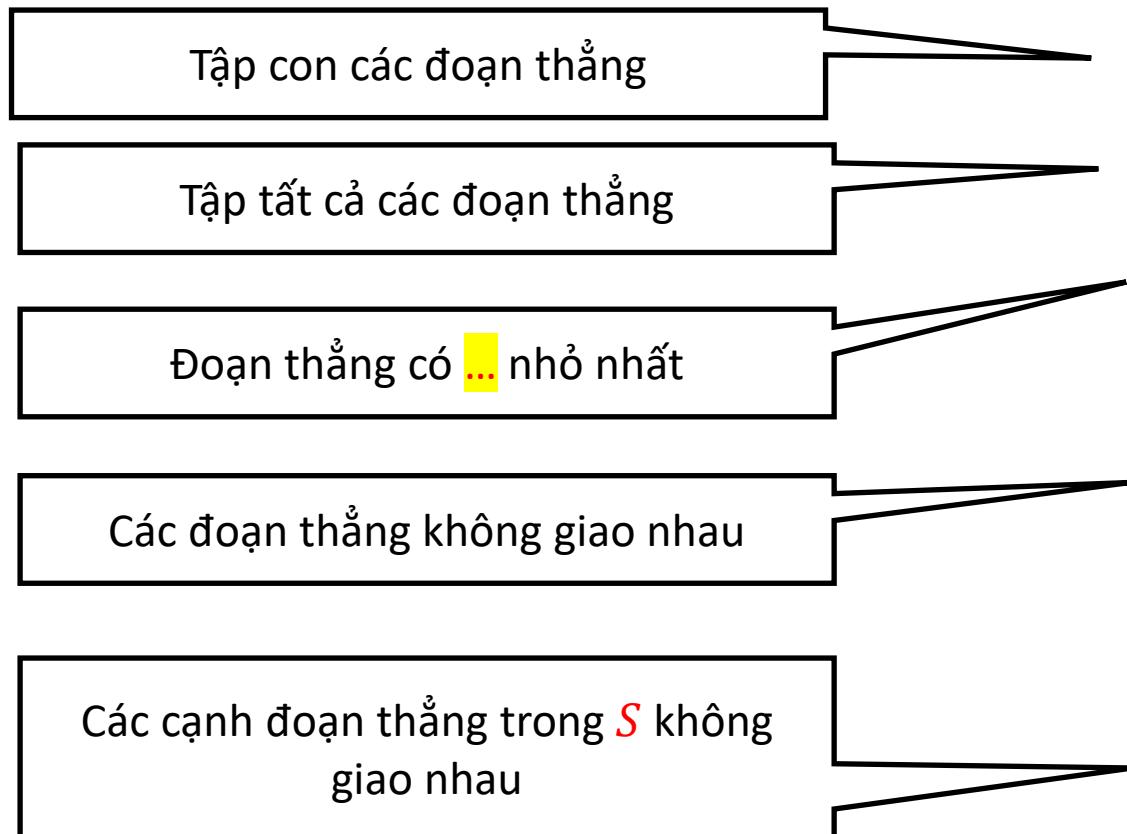
- Bài toán: Cho tập các đoạn thẳng $X = \{(a_1, b_1), \dots, (a_n, b_n)\}$ trong đó $a_i < b_i, \forall i \in \{1, \dots, n - 1\}$ là toạ độ đầu mứt của đoạn thứ i trên đường thẳng, với mọi $i \in \{1, \dots, n\}$. Tìm tập con các đoạn đôi một không giao nhau có số lượng phần tử lớn nhất.



Lời giải tối ưu: $S = \{B, C, D, E\}$

Ví dụ: Bài toán tập đoạn con không giao nhau cực đại

■ Thiết kế: Đề xuất thuật toán tham lam

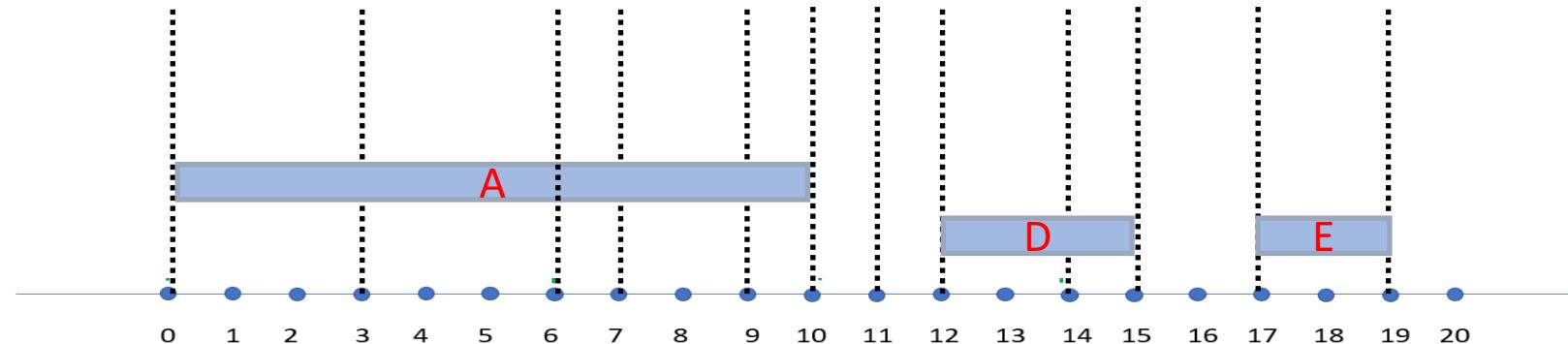
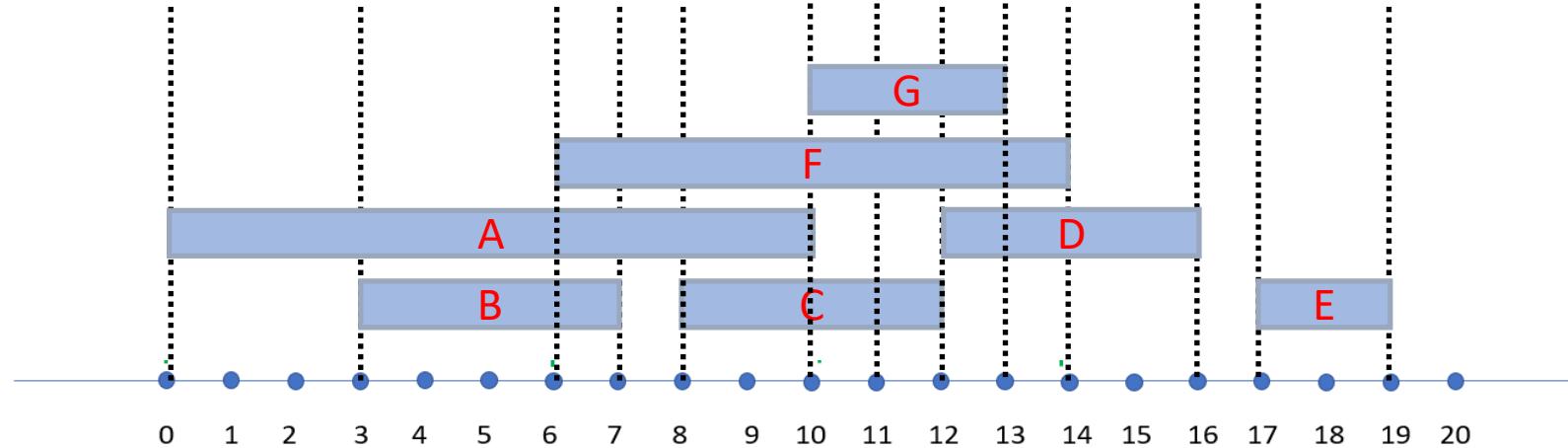
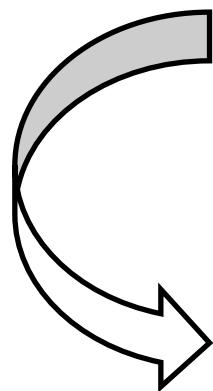


- S : Lời giải cần tìm
- C : Tập các ứng cử viên
- $\text{select}(C)$: Hàm chọn ra ứng cử viên tiềm năng nhất
- $\text{solution}(S)$: Hàm trả về TRUE nếu S là một lời giải hợp lệ, ngược lại hàm trả về FALSE
- $\text{feasible}(S)$: Hàm trả về TRUE nếu S không vi phạm ràng buộc, ngược lại hàm trả về FALSE

Ví dụ: Bài toán tập đoạn con không giao nhau cực đại

Đoạn thẳng có a_i nhỏ nhất

• $\text{select}(C)$: Hàm chọn ra ứng cử viên tiềm năng nhất

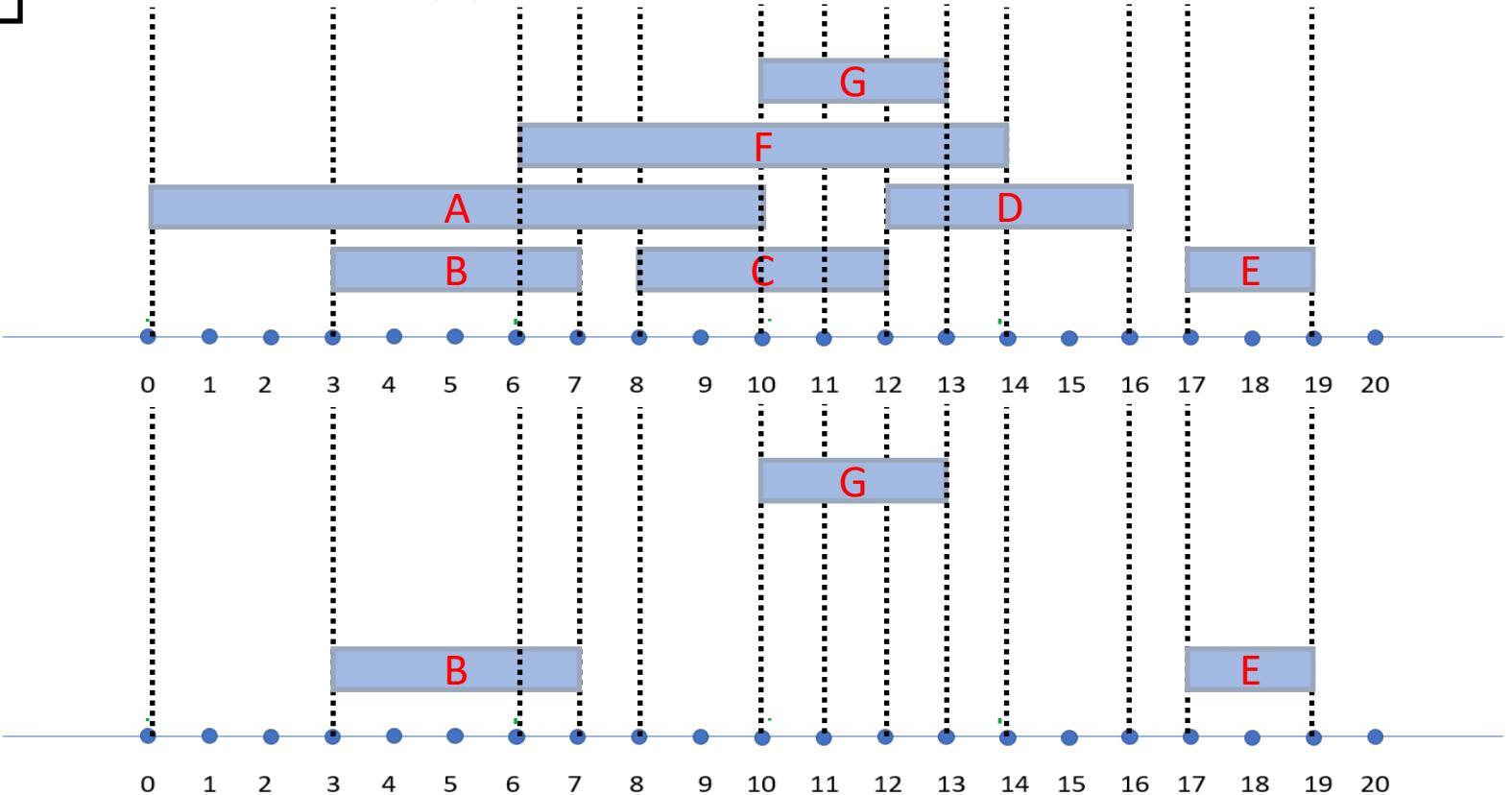
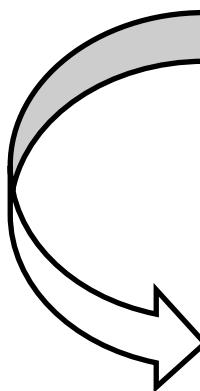


Lời giải này có 3 đoạn → Không tối ưu vì lời giải tối ưu có 4 đoạn ($S = \{B, C, D, E\}$)

Ví dụ: Bài toán tập đoạn con không giao nhau cực đại

Đoạn thẳng có $b_i - a_i$ nhỏ nhất

• $\text{select}(C)$: Hàm chọn ra ứng cử viên tiềm năng nhất

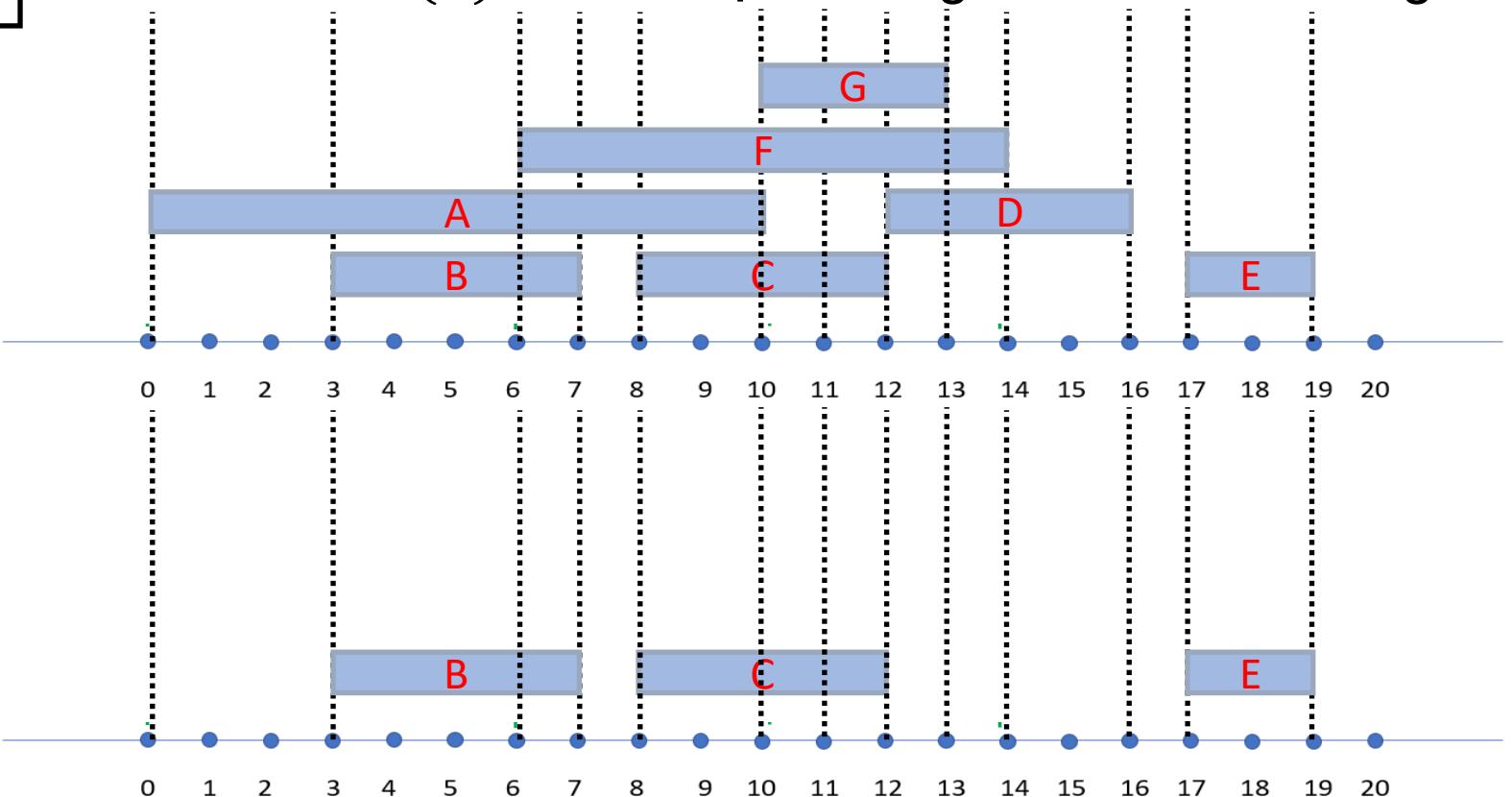
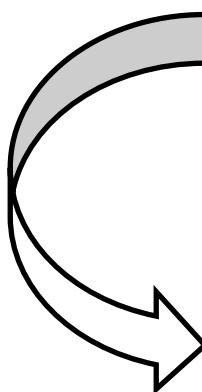


Lời giải này có 3 đoạn \rightarrow Không tối ưu vì lời giải tối ưu có 4 đoạn ($S = \{B, C, D, E\}$)

Ví dụ: Bài toán tập đoạn con không giao nhau cực đại

Đoạn thẳng có b_i nhỏ nhất

• $\text{select}(C)$: Hàm chọn ra ứng cử viên tiềm năng nhất



Thuật toán tìm được lời giải tối ưu có 4 đoạn.

Bài tập về nhà: Chứng minh thuật toán tham lam này luôn trả ra lời giải tối ưu

BÀI TẬP LUYỆN TẬP

- **Đề bài:** Có n công việc $1, 2, \dots, n$. Công việc i có thời hạn hoàn thành là $d[i]$ và có lợi nhuận khi được đưa vào thực hiện là $p[i]$ ($i = 1, \dots, n$). Biết rằng chỉ được nhiều nhất 1 công việc được thực hiện tại mỗi thời điểm và khi thời gian thực hiện xong mỗi công việc đều là 1 đơn vị. Hãy tìm cách chọn ra các công việc để đưa vào thực hiện sao cho tổng lợi nhuận thu được là nhiều nhất đồng thời mỗi công việc phải hoàn thành trước hoặc đúng thời hạn.
- **Hình thức:** Làm tại nhà và nộp lại trên hệ thống chấm code



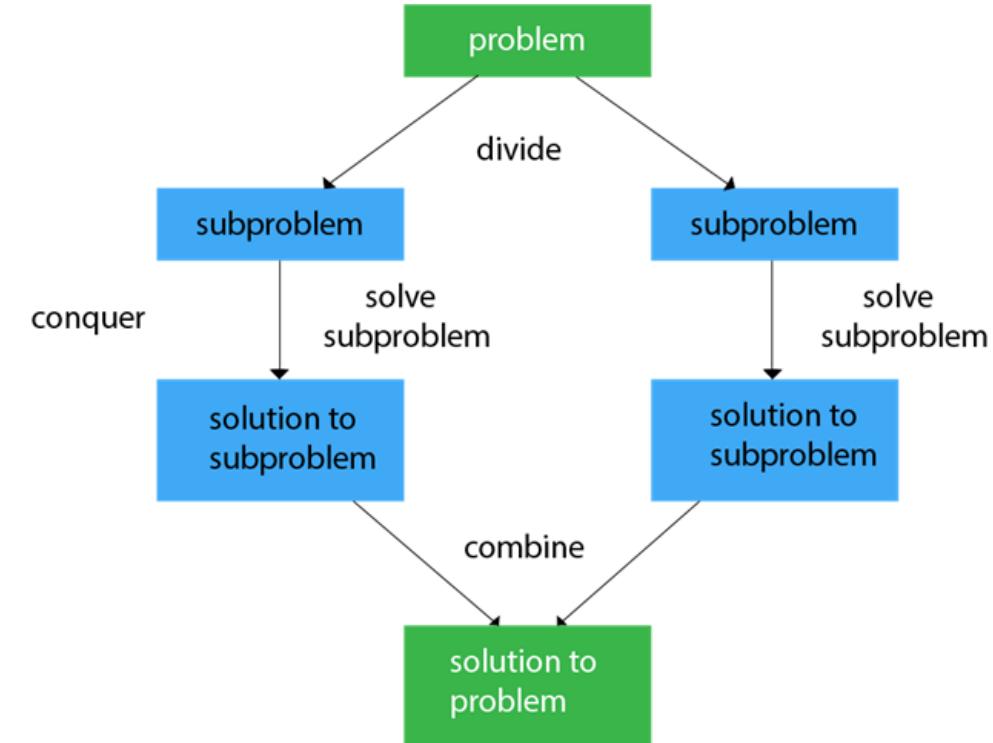
A. Thuật toán tham lam

B. Thuật toán chia để trị



Giới thiệu

- Thuật toán chia để trị Divide and Conquer algorithm) là kỹ thuật mạnh trong Khoa học máy tính và Toán học.
- Ý tưởng: là làm dễ bài toán bằng cách chia nhỏ bài toán phức tạp thành những bài toán con đơn giản hơn (CHIA), giải riêng rẽ các bài toán con (TRỊ) và cuối cùng là tổng hợp các kết quả bài toán con để thu được lời giải bài toán ban đầu.



Nguồn: <https://www.javatpoint.com/divide-and-conquer-introduction>



- Sử dụng thuật toán: Sử dụng **Đệ quy quay lui** để thể hiện sơ đồ chung của thuật toán **chia để trị**
- → Dễ hiểu và thể hiện rõ ý tưởng của thuật toán

```
DC( $P_n$ ) {  
    if  $n \leq n_0$  {  
        solve_problem( $p_n$ );  
    } else {  
        SP = divide( $P_n$ );  
        foreach  $p \in SP$  {  
             $s(p) = DC(p)$   
        }  
        S = aggregate( $s(p_1), \dots s(p_k)$ )  
    }  
    return S;  
}
```

Sơ đồ thuật toán

```
DC( $P_n$ ) {  
    if  $n \leq n_0$  {  
        solve_problem( $p_n$ );  
    } else {  
        SP = divide( $P_n$ );  
        foreach  $p \in SP$  {  
             $s(p) = DC(p)$   
        }  
        S = aggregate( $s(p_1), \dots s(p_k)$ )  
    }  
    return S;  
}
```

n_0 là kích thước nhỏ của bài toán, mà chúng ta có thể chỉ ra lời giải của bài một cách dễ dàng. Đây là bước cơ sở trong kỹ thuật Đệ quy.

Chia nhỏ bài toán cha thành nhiều bài toán con

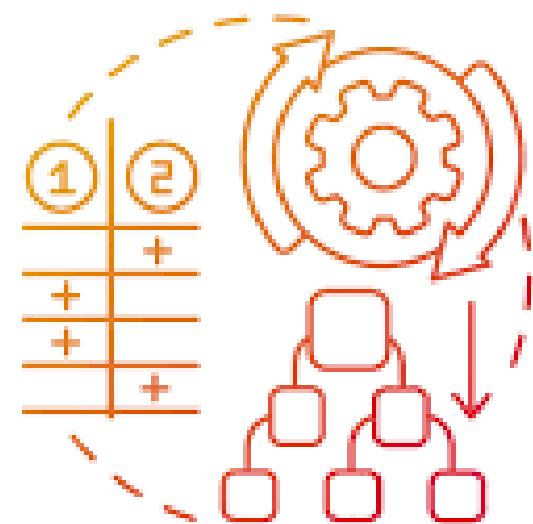
Giải từng bài toán con

Tổng hợp kết quả các bài toán con để xây dựng lời giải bài toán cha (bài toán ban đầu)



Đặc điểm

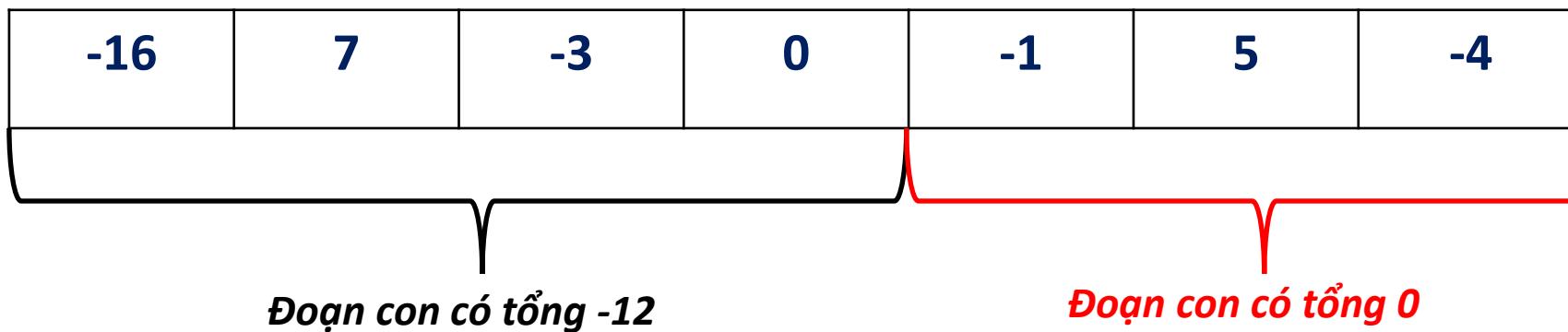
- Ý tưởng thuật toán “Chia nhỏ” rất tự nhiên và giống với cách tiếp cận giải quyết vấn đề của con người
- Các bài toán con độc lập, có nghĩa là việc giải một bài toán con không ảnh hưởng đến giải pháp của các bài toán con đồng mức khác
- Việc chia bài toán đảm bảo không mất nghiêm
- Một số ứng dụng nổi bật
 - Tính toán song song
 - Tìm kiếm



DIVIDE AND CONQUER

Ví dụ: Đoạn con có tổng lớn nhất

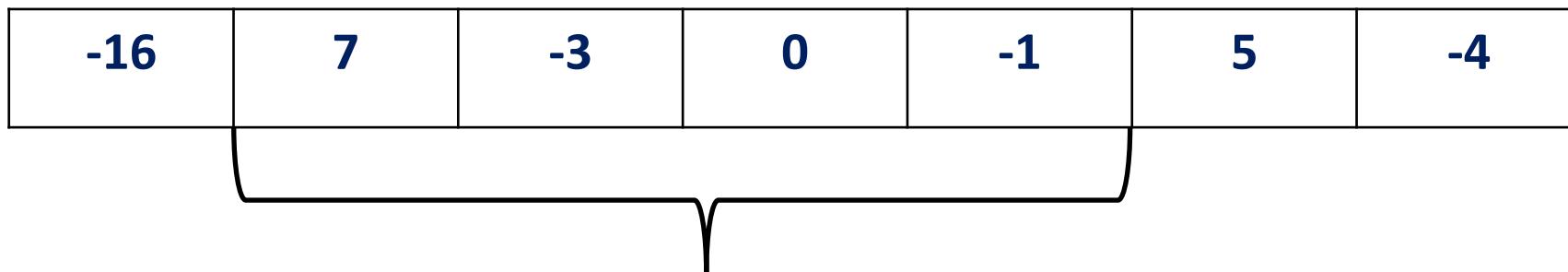
- **Phát biểu:** Cho dãy n số nguyên (a_1, a_2, \dots, a_n) , hãy tìm đoạn con bao gồm các phần tử liên tiếp của dãy sao cho tổng các phần tử được chọn là cực đại.
- **Ví dụ:** Cho dãy 7 số nguyên sau



- *Lời giải tối ưu (dãy con có tổng cực đại bằng 8): 7, -3, 0, -1, 5*
- Thuật toán vét cạn có độ phức tạp $O(n^2)$, chúng ta có thể làm tốt hơn với thuật toán chia để trị hay không?

Ví dụ: Đoạn con có tổng lớn nhất

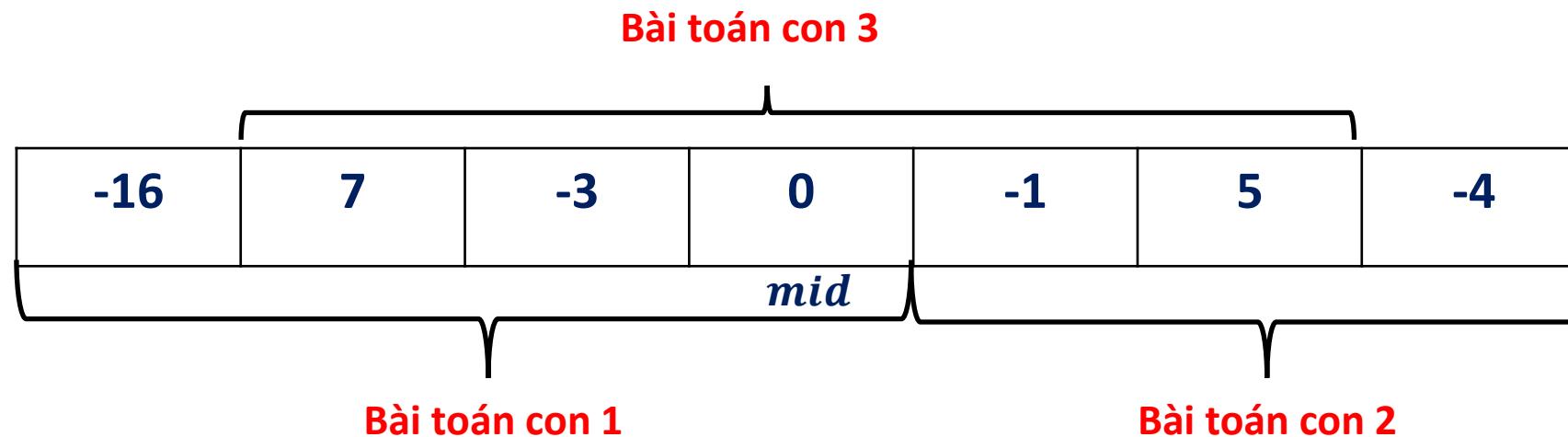
- **Xác định bài toán con:** Bài toán con của bài toán tìm đoạn con cực đại của dãy (a_1, \dots, a_n) là bài toán tìm đoạn con cực đại của đoạn con $a_i, a_{i+1}, \dots, a_{i+j}$, $i \geq 1$, $i + j \leq n$.



Bài toán con: Tìm đoạn con cực đại của đoạn $(7, -3, 0, -1)$

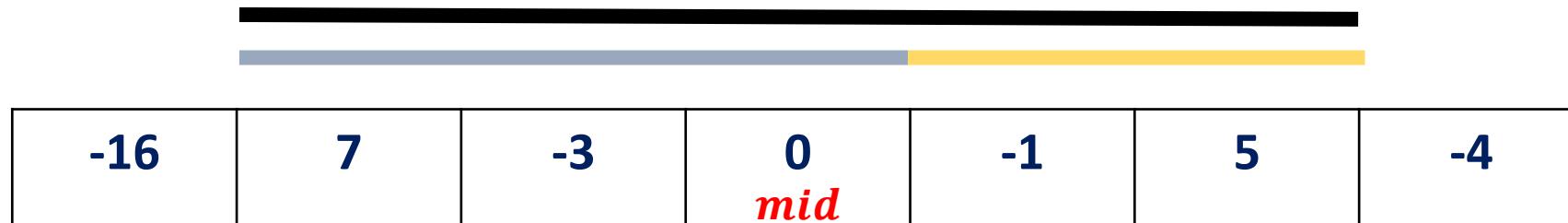
Ví dụ: Đoạn con có tổng lớn nhất

- **Chia:** Chia đôi dãy n phần tử tại điểm dữa $mid = \text{round}(\frac{n+1}{2})$
 - Bài toán con 1: Giống bài toán cha, tìm dãy con cực đại của dãy $1, \dots, mid$
 - Bài toán con 2: Giống bài toán cha, tìm dãy con cực đại của dãy $mid + 1, \dots, n$
 - Bài toán con 3: Tìm dãy con cực đại có chứa phần tử mid và không biết thành phần đầu tiên và cuối cùng



Ví dụ: Đoạn con có tổng lớn nhất

- Giải bài toán con:** Bài toán con 1 và 2 có thể giải bằng đệ quy. Bài toán con 3 giải được trực tiếp do có ràng buộc phải chứa phần tử mid . Lời giải bài toán 3 là hợp của đoạn con cực đại kết thúc tại mid và đoạn con cực đại bắt đầu từ $mid + 1$. Để giải 2 bài toán nhỏ này, chúng ta chỉ cần duyệt từ mid lùi về 1 và từ $mid + 1$ tiến về n , với độ phức tạp $O(n)$



Ví dụ: Đoạn con có tổng lớn nhất

- **Xác định bài toán con:** Bài toán con của bài toán tìm đoạn con cực đại của dãy (a_1, \dots, a_n) là bài toán tìm đoạn con cực đại của đoạn con $a_i, a_{i+1}, \dots, a_{i+j}$, $i \geq 1$, $i + j \leq n$.
- **Chia:** Chia đôi dãy n phần tử tại điểm dữa $mid = round(\frac{n+1}{2})$
 - Bài toán con 1: Giống bài toán cha, tìm dãy con cực đại của dãy $1, \dots, mid$
 - Bài toán con 2: Giống bài toán cha, tìm dãy con cực đại của dãy $mid + 1, \dots, n$
 - Bài toán con 3: Tìm dãy con cực đại có chứa phần tử mid và không biết thành phần đầu tiên và cuối cùng
- **Giải bài toán con:** Dùng đệ quy và duyệt mảng độ phức tạp $O(n)$
- **Tổng hợp:** Lời giải là lời giải tốt nhất trong 3 lời giải của 3 bài toán con
- **Độ phức tạp thuật toán:** $O(\log(n))$



Ví dụ: Tính nhanh a^n

- **Phát biểu:** Tính a^n với a và n là các số nguyên dương
- **Phân tích:** Phương pháp trực tiếp, thực hiện n phép nhân, có độ phức tạp tính toán $O(n)$. Chúng ta có thể làm tốt hơn với Thuật toán chia để trị?
- **Thiết kế thuật toán chia để trị:**
 - **Bài toán con:** Tính $a^k, k < n$
 - **Chia:** Ý tưởng chia đôi, 2 bài toán con giống hệt nhau, chỉ cần giải một bài toán con
 - Nếu n chẵn, ta có $a^n = \left(a^{\frac{n}{2}}\right) \times \left(a^{\frac{n}{2}}\right)$
 - Nếu n lẻ, ta có $a^n = a \times a^{n-1}$
 - **Xử lý:** Sử dụng đệ quy để giải
 - **Tổng hợp:** Đơn giản, chỉ là một phép tính toán nhân
 - **Độ phức tạp thuật toán:** $O(\log(n))$



Ví dụ: Tính nhanh a^n

- Code minh họa

```
int Pow(int x, int n) {  
    if (n == 0) return 1;  
    if (n % 2 != 0) return x * pow(x, n - 1);  
    int res = Pow(x, n/2);  
    return res * res;  
}
```



- **Ý tưởng:** Cài đặt thuật toán Sắp xếp trộn (Merge sort)
- **Đề bài:** Sắp xếp mảng gồm n số nguyên sử dụng thuật toán chia để trị. Gợi ý, bài toán cha chia thành 2 bài toán con bằng nhau.
- **Hình thức:** Làm tại nhà và nộp lại trên hệ thống chấm code

A large, faint watermark of the HUST logo is visible across the entire slide, consisting of a grid of red dots.

HUST

THANK YOU !

HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.



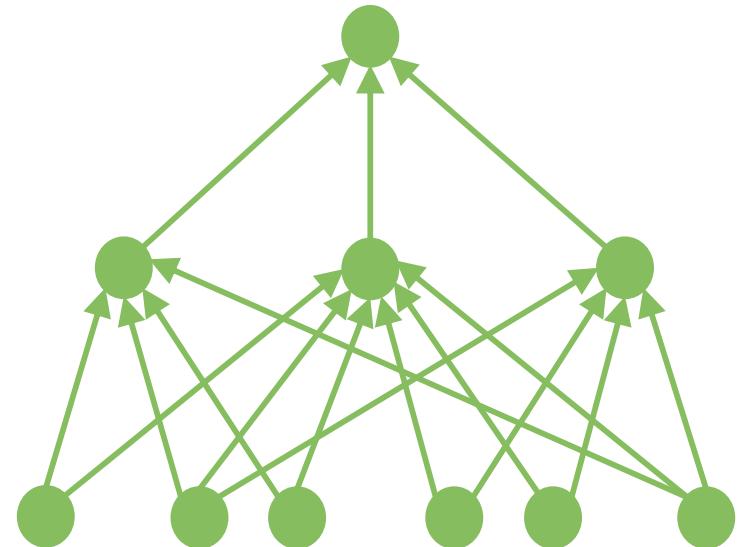
ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

Sơ đồ thuật toán quy hoạch động

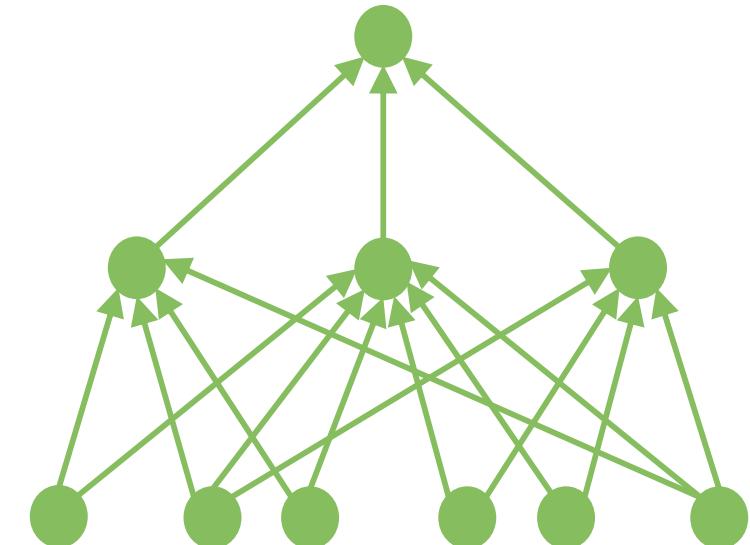
ONE LOVE. ONE FUTURE.

- Thuật toán quy hoạch động (Dynamic programming algorithm) được phát minh bởi Bellman trong thế chiến thứ 2. Tên đầu tiên của thuật toán này là multi-stage decision process (ra quyết định qua nhiều giai đoạn)
- **Thuật** toán quy hoạch động là một kỹ thuật mạnh để giải các bài toán tối ưu bằng cách chia chúng thành các bài toán nhỏ hơn và giải các bài toán con một lần duy nhất
- Thuật toán quy hoạch động có nhiều điểm giống với Thuật toán quay lui (backtracking) và Thuật toán chia để trị (divide conquer)



Sơ đồ thuật toán

- **CHIA** bài toán xuất phát thành các bài toán con không nhất thiết độc lập với nhau
- **GIẢI** các bài toán con từ nhỏ đến lớn, lời giải được lưu trữ lại vào bộ nhớ (để đảm bảo mỗi bài toán chỉ giải đúng 1 lần)
 - Bài toán con nhỏ nhất phải được giải một cách trực tiếp, đơn giản
- **KẾT HỢP** lời giải của bài toán lớn hơn từ lời giải đã có của các bài toán con nhỏ hơn (cần sử dụng công thức truy hồi)
 - Số lượng bài toán con cần được bị chặn bởi một hàm đa thức của kích thước dữ liệu đầu vào



Sơ đồ thuật toán

```
map<problem , value> Memory ;  
  
value DP(problem P) {  
    if (is_base_case(P))  
        return base_case_value(P);  
  
    if (Memory .find(P) != Memory .end())  
        return Memory [P];  
  
    value result = some value;  
    for (problem Q in subproblems(P))  
        result = Combine(result , DP(Q));  
  
    Memory [P] = result;  
    return result;  
}
```

Bộ nhớ, ánh xạ bài toán và lời giải

Bài toán con nhỏ nhất, tương ứng với bước cơ sở của thuật toán đệ quy

Luôn kiểm tra bộ nhớ xem bài toán con đã giải chưa, đã giải rồi không giải lại, nếu chưa giải thì thực hiện giải.

Giải bài toán con

Giải tất cả bài toán con và Kết hợp những lời giải tìm được để hình thành lời giải bài toán lớn hơn

Lưu kết quả bài toán con vào trong bộ nhớ



Sơ đồ thuật toán

```
map<problem, value> Memory;

value DP(problem P) {
    if (is_base_case(P))
        return base_case_value(P);

    if (Memory.find(P) != Memory.end())
        return Memory[P];

    value result = some value;
    for (problem Q in subproblems(P))
        result = Combine(result, DP(Q));

    Memory[P] = result;
    return result;
}
```



Sơ đồ thuật toán

```
map<problem , value> Memory ;
```

Bộ nhớ, ánh xạ bài toán và lời giải

```
value DP(problem P) {  
    if (is_base_case(P))  
        return base_case_value(P);
```

Bài toán con nhỏ nhất, tương ứng với bước cơ sở của thuật toán đệ quy

```
if (Memory .find(P) != Memory .end())  
    return Memory [P];
```

Luôn kiểm tra bộ nhớ xem bài toán con đã giải chưa, đã giải rồi không giải lại, nếu chưa giải thì thực hiện giải.

```
value result = some value;
```

Giải một bài toán con

```
for (problem Q in subproblems(P))  
    result = Combine(result , DP(Q));
```

Giải tất cả bài toán con và Kết hợp những lời giải tìm được để hình thành lời giải bài toán lớn hơn

```
Memory [P] = result;  
return result;
```

Lưu kết quả bài toán con vào trong bộ nhớ

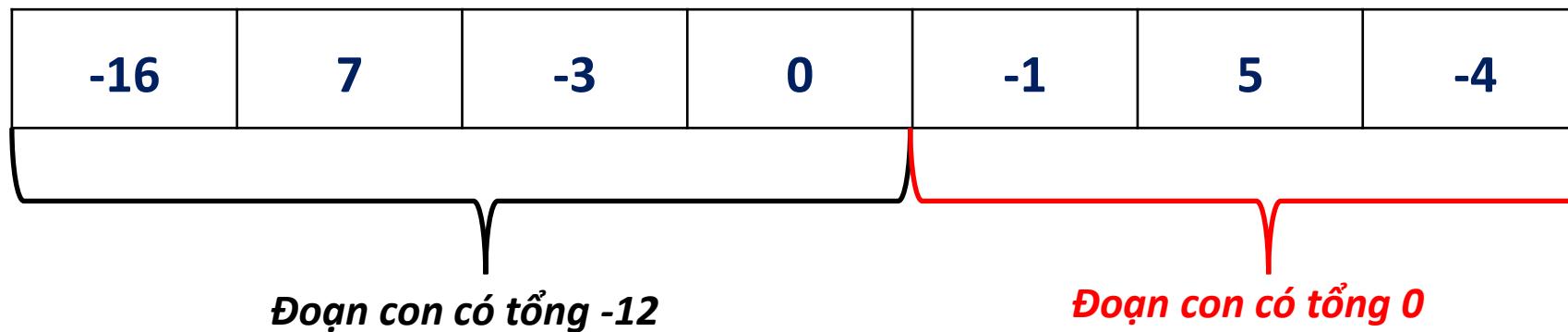
Đặc điểm

- So với thuật toán chia để trị, thuật toán quy hoạch động cũng có 3 bước là Chia, Giải bài toán con và Kết hợp. Tuy nhiên, trong chia để trị, các bài toán con là độc lập; còn trong quy hoạch động, các bài toán con gối nhau hay chồng chéo lên nhau.
- Khó khăn lớn nhất trong đề xuất thuật toán quy hoạch động là Công thức truy hồi (tên khác là Công thức quy hoạch động, Công thức đệ quy)
- Có 2 cách tiếp cận: Top-Down và Bottom-Up, trong đó Top-Down tự nhiên và dễ hiểu, dễ cài đặt
- Thiết kế bộ nhớ ảnh hưởng lớn tới tốc độ của thuật toán
- Bộ nhớ còn dung để truy vết, tìm ra tường minh lời giải tối ưu



Ví dụ: Đoạn con có tổng lớn nhất

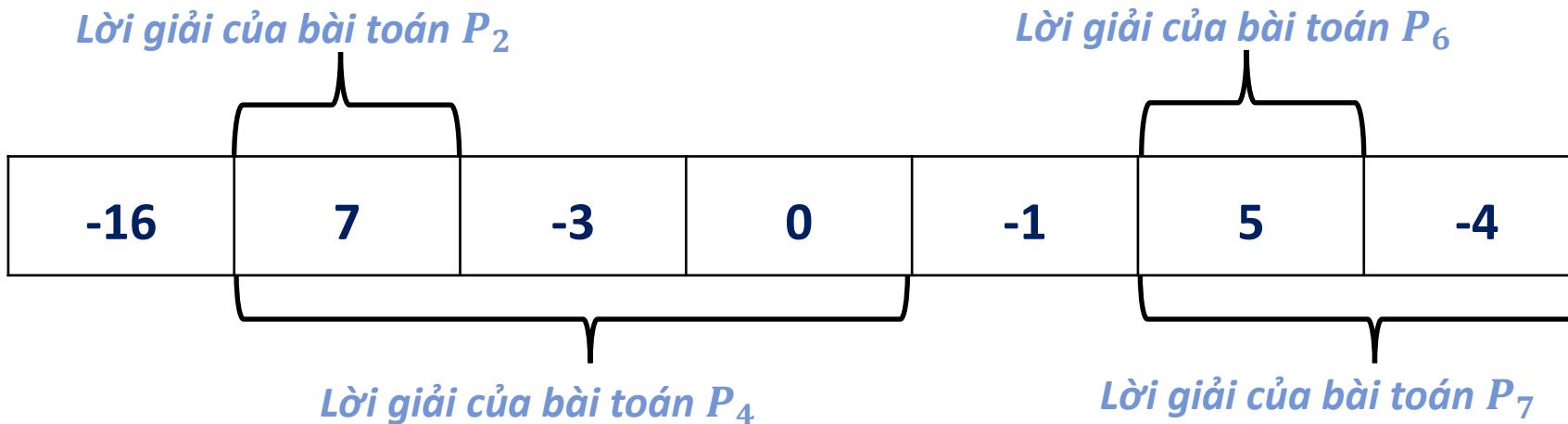
- **Phát biểu:** Cho dãy n số nguyên (a_1, a_2, \dots, a_n) , hãy tìm đoạn con bao gồm các phần tử liên tiếp của dãy sao cho tổng các phần tử được chọn là cực đại.
- **Ví dụ:** Cho dãy 7 số nguyên sau



- *Lời giải tối ưu (dãy con có tổng cực đại bằng 8): 7, -3, 0, -1, 5*
- Thuật toán chia để trị có độ phức tạp $O(n \log(n))$, chúng ta có thể làm tốt hơn với thuật toán quy hoạch động hay không?

Ví dụ: Đoạn con có tổng lớn nhất

- **Phát biểu:** Cho dãy n số nguyên (a_1, a_2, \dots, a_n) , hãy tìm tổng của các phần tử của đoạn con bao gồm các phần tử liên tiếp của dãy sao cho tổng các phần tử được chọn là cực đại.
- **Xác định bài toán con:** P_i là bài toán tìm đoạn con bao gồm các phần tử liên tiếp có tổng lớn nhất mà phần tử cuối cùng là a_i , với mọi $i = 1, \dots, n$.



Ví dụ: Đoạn con có tổng lớn nhất

- **Phát biểu:** Cho dãy n số nguyên (a_1, a_2, \dots, a_n) , hãy tìm tổng các phần tử của đoạn con bao gồm các phần tử liên tiếp của dãy sao cho tổng các phần tử được chọn là cực đại.
- **Xác định bài toán con:** P_i là bài toán tìm đoạn con bao gồm các phần tử liên tiếp có tổng lớn nhất mà phần tử cuối cùng là a_i , với mọi $i = 1, \dots, n$.
- **Công thức quy hoạch động** (công thức kết hợp lời giải các bài toán con để thu được lời giải bài toán cha): Gọi S_i là tổng các phần tử của lời giải của P_i , $\forall i = 1, \dots, n$.

Ta có: $S_1 = a_1$,

$$S_i = \begin{cases} S_{i-1} + a_i & \text{nếu } S_{i-1} > 0 \\ a_i & \text{nếu } S_{i-1} \leq 0 \end{cases}$$



Ví dụ: Đoạn con có tổng lớn nhất

- Ví dụ minh họa:

-16	7	-3	0	-1	5	-4
-----	---	----	---	----	---	----

$$S_1 = -16, S_2 = a_2 = 7, S_3 = S_2 + a_3 = 4, S_4 = S_3 + 0 = 4,$$

$$S_5 = S_4 + (-1) = 3, S_6 = S_5 + 5 = 8, S_7 = S_6 + (-4) = 4$$

Ví dụ: Đoạn con có tổng lớn nhất

- **Công thức quy hoạch động** (công thức kết hợp lời giải các bài toán con để thu được lời giải bài toán cha): Gọi S_i là tổng các phần tử của lời giải của P_i , $\forall i = 1, \dots, n$. Ta có: $S_1 = a_1$,

$$S_i = \begin{cases} S_{i-1} + a_i & \text{nếu } s_{i-1} > 0 \\ a_i & \text{nếu } s_{i-1} \leq 0 \end{cases}$$

- **Lời giải:** Tổng của các phần tử của đoạn con bao gồm các phần tử liên tiếp của dãy có tổng các phần tử được chọn lớn nhất là:

$$\max(S_1, S_2, \dots, S_n)$$



Ví dụ: Đoạn con có tổng lớn nhất

- **Lời giải:** Tổng của các phần tử của đoạn con bao gồm các phần tử liên tiếp của dãy có tổng các phần tử được chọn lớn nhất là:

$$\max(S_1, S_2, \dots, S_n)$$

Ví dụ minh họa:

-16	7	-3	0	-1	5	-4
-----	---	----	---	----	---	----

$$S_1 = -16, S_2 = a_2 = 7, S_3 = S_2 + a_3 = 4, S_4 = S_3 + 0 = 4,$$

$$S_5 = S_4 + (-1) = 3, S_6 = S_5 + 5 = 8, S_7 = S_6 + (-4) = 4$$

Lời giải: $\max(S_1, S_2, \dots, S_7) = 8$

Ví dụ: Đoạn con có tổng lớn nhất

- **Công thức quy hoạch động** (công thức kết hợp lời giải các bài toán con để thu được lời giải bài toán cha): Gọi S_i là tổng các phần tử của lời giải của P_i , $\forall i = 1, \dots, n$. Ta có: $S_1 = a_1$,

$$S_i = \begin{cases} S_{i-1} + a_i & \text{nếu } s_{i-1} > 0 \\ a_i & \text{nếu } s_{i-1} \leq 0 \end{cases}$$

- **Lời giải:** Tổng của các phần tử của đoạn con bao gồm các phần tử liên tiếp của dãy có tổng các phần tử được chọn lớn nhất là:

$$\max(S_1, S_2, \dots, S_n)$$

- **Độ phức tạp thuật toán:** $O(n)$



Bài tập: Dãy con tăng chặt dài nhất

- **Đề bài:** Cho dãy số nguyên $A = (a_1, a_2, \dots, a_n)$ thỏa mãn điều kiện các phần tử đôi một khác nhau ($a_i \neq a_j, \forall i \neq j$). Một dãy con của A là một dãy thu được bằng cách xóa đi một số phần tử trong A . Một dãy con $B = (b_1, \dots, b_k)$ được gọi là tăng chặt khi $b_i < b_{i+1}, \forall i \in \{1, \dots, k - 1\}$. Tìm độ dài của dãy con tăng chặt của A có độ dài lớn nhất.
- **Hình thức:** Làm tại nhà và nộp lại trên hệ thống chấm code



Ví dụ: Dãy con chung dài nhất

- **Phát biểu:** Cho 2 dãy ký tự $X = (x_1, x_2, \dots, x_n)$ và $Y = (y_1, \dots, y_m)$. Một dãy con của A là một dãy thu được bằng cách xóa đi một số phần tử trong A . Hãy tìm độ dài của dãy con chung dài nhất của 2 X và Y .
- **Ví dụ:**
 - $X = "abcb"$ và $Y = "bdcab"$
 - Dãy con chung dài nhất của 2 dãy là dãy " bcb " với độ dài 3
- **Nhận xét:** Thuật toán vét cạn, so sánh tất cả các dãy con của X và Y sẽ có độ phức tạp $O(2^n \times 2^m \times \max(m, n))$. Chúng ta có thể giải bài toán này nhanh hơn với một thuật toán quy hoạch động hay không?



Ví dụ: Dãy con chung dài nhất

- **Phát biểu:** Cho 2 dãy ký tự $X = (x_1, x_2, \dots, x_n)$ và $Y = (y_1, \dots, y_m)$. Một dãy con của A là một dãy thu được bằng cách xóa đi một số phần tử trong A . Hãy tìm độ dài của dãy con chung dài nhất của 2 X và Y .
- **Xác định bài toán con:** Gọi $S(i, j)$ là độ dài của dãy con chung dài nhất của 2 dãy, dãy con của X là $X_i = (x_1, \dots, x_i)$ với $i \in \{1, \dots, n\}$ và dãy con của Y là $Y_j = (y_1, \dots, y_j)$ với $j \in \{1, \dots, m\}$.
- **Bài toán cơ sở (bài toán con nhỏ nhất):**

$$S(i, 0) = 0, \forall i \in \{1, \dots, n\}$$

$$S(0, j) = 0, \forall j \in \{1, \dots, m\}$$



Ví dụ: Dãy con chung dài nhất

- **Xác định bài toán con:** Gọi $S(i, j)$ là độ dài của dãy con chung dài nhất của 2 dãy, dãy con của X là $X_i = (x_1, \dots, x_i)$ với $i \in \{1, \dots, n\}$ và dãy con của Y là $Y_j = (y_1, \dots, y_j)$ với $j \in \{1, \dots, m\}$.

- **Bài toán cơ sở** (bài toán con nhỏ nhất):

$$S(i, 0) = 0, \forall i \in \{1, \dots, n\}$$

$$S(0, j) = 0, \forall j \in \{1, \dots, m\}$$

- **Công thức quy hoạch động:**

$$S(i, j) = \max \begin{cases} S(i - 1, j - 1) & \text{nếu } x_i = y_j \\ S(i - 1, j) \\ S(i, j - 1) \end{cases}$$

Ví dụ: Dãy con chung dài nhất

- Công thức quy hoạch động

$$S(i, j) = \max \begin{cases} S(i - 1, j - 1) & \text{nếu } x_i = y_j \\ S(i - 1, j) \\ S(i, j - 1) \end{cases}$$

X

3	7	2	5	1	4	9
---	---	---	---	---	---	---

Y

4	3	2	3	6	1	5	4	9	7
---	---	---	---	---	---	---	---	---	---

1 2 3 4 5 6 7 8 9 10

1	0	1	1	1	1	1	1	1	1	1
2	0	1	1	1	1	1	1	1	1	2
3	0	1	2	2	2	2	2	2	2	2
4	0	1	2	2	2	2	3	3	3	3
5	0	1	2	2	2	3	3	3	3	3
6	1	1	2	2	2	3	3	4	4	4
7	1	1	2	2	2	3	3	4	5	5

Ví dụ: Dãy con chung dài nhất

- **Xác định bài toán con:** Gọi $S(i, j)$ là độ dài của dãy con chung dài nhất của 2 dãy, dãy con của X là $X_i = (x_1, \dots, x_i)$ với $i \in \{1, \dots, n\}$ và dãy con của Y là $Y_j = (y_1, \dots, y_j)$ với $j \in \{1, \dots, m\}$.

- **Bài toán cơ sở** (bài toán con nhỏ nhất):

$$S(i, 0) = 0, \forall i \in \{1, \dots, n\}$$

$$S(0, j) = 0, \forall j \in \{1, \dots, m\}$$

- **Công thức quy hoạch động:**

$$S(i, j) = \max \begin{cases} S(i - 1, j - 1) & \text{nếu } x_i = y_j \\ S(i - 1, j) \\ S(i, j - 1) \end{cases}$$

- **Độ phức thuật toán:** $O(n \times m)$

Bài tập: Dãy con tăng chặt dài nhất

- **Đề bài:** Cho dãy $A = (a_1, a_2, \dots, a_n)$. Một dãy con của dãy A là một dãy thu được bằng cách loại bỏ một số phần tử khỏi A . Tìm độ dài của dãy con của A là một cấp số cộng với bước nhảy bằng 1 và có độ dài lớn nhất.
- **Hình thức:** Làm tại nhà và nộp lại trên hệ thống chấm code





HUST

THANK YOU !

HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT



ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

TUẦN 4 : DANH SÁCH LIÊN KẾT

ONE LOVE. ONE FUTURE.

NỘI DUNG

1. Kiến thức cơ sở
2. Danh sách liên kết đơn
3. Thao tác trên danh sách liên kết



MỤC TIÊU

Sau bài học này, người học có thể:

1. Hiểu về cấu trúc dữ liệu **danh sách liên kết đơn**;
2. Xây dựng hai thao tác cơ bản trên cấu trúc dữ liệu danh sách liên kết đơn: Duyệt và Tìm kiếm



NỘI DUNG TIẾP THEO

1. Kiến thức cơ sở

1.1. Con trỏ

1.2. Cấu trúc

2. Danh sách liên kết đơn

3. Thao tác trên danh sách liên kết

3.1 Duyệt danh sách

3.2 Tìm kiếm



1. KIẾN THỨC CƠ SỞ

1.1. Con trỏ

- Con trỏ (Pointer) là khái niệm cơ bản trong ngôn ngữ lập trình C, dùng để làm việc với địa chỉ bộ nhớ.
- Biến con trỏ cũng là một biến, cũng cần khai báo, khởi tạo và dùng để lưu trữ dữ liệu.
- Biến con trỏ có địa chỉ riêng.
- Biến con trỏ không lưu giá trị như biến cơ bản, nó trả tới một địa chỉ khác, tức mang giá trị là một địa chỉ trong RAM.

Địa chỉ trong ô nhớ 0x13AB

0x56CD

Con trỏ

Địa chỉ trong ô nhớ 0x56CD

100

Biến cơ bản



1. KIẾN THỨC CƠ SỞ

1.1. Con trỏ

- Con trỏ (Pointer) là khái niệm cơ bản trong ngôn ngữ lập trình C, dùng để làm việc với địa chỉ bộ nhớ.
- Kiểu dữ liệu của con trỏ trùng với kiểu dữ liệu tại vùng nhớ mà nó trỏ đến.
- Giá trị của con trỏ chứa địa chỉ vùng nhớ mà con trỏ trỏ đến.
- Địa chỉ của con trỏ là địa chỉ của bản thân biến con trỏ đó trong RAM.

Địa chỉ trong ô nhớ 0x13AB

0x56CD

Con trỏ

Địa chỉ trong ô nhớ 0x56CD

100

Biến cơ bản



1. KIẾN THỨC CƠ SỞ

1.1. Con trỏ

- Khai báo: **int *p;**
 - Khai báo con trỏ để trỏ tới biến kiểu nguyên
 - Giá trị của **p** xác định địa chỉ của biến
- Gán giá trị: **int a; int* p = &a;**
 - **p** trỏ vào biến **a**

Địa chỉ trong ô nhớ 0x13AB

0x56CD

Con trỏ

Địa chỉ trong ô nhớ 0x56CD

100

Biến cơ bản



1. KIẾN THỨC CƠ SỞ

1.1. Con trỏ

■ Ví dụ

In ra cùng một địa chỉ bộ nhớ

```
#include <stdio.h>
int main() {
    int* pc, c;

    c = 22;
    printf("Address of c: %p\n", &c);
    printf("Value of c: %d\n\n", c);

    pc = &c;
    printf("Address of pointer pc: %p\n", pc);
    printf("Content of pointer pc: %d\n\n", *pc);

    c = 11;
    printf("Address of pointer pc: %p\n", pc);
    printf("Content of pointer pc: %d\n\n", *pc);

    *pc = 2;
    printf("Address of c: %p\n", &c);
    printf("Value of c: %d\n\n", c);
    return 0;
}
```

In ra giá trị: 22

In ra giá trị: 22

In ra giá trị: 11

In ra giá trị: 2



1. KIẾN THỨC CƠ SỞ

1.2. Cấu trúc

- Cấu trúc (struct) là một kiểu dữ liệu người dùng tự định nghĩa (user defined datatype)
- Cấu trúc là một tập hợp các biến, những biến này có thể có kiểu dữ liệu khác nhau

```
struct structureName {  
    dataType member1;  
    dataType member2;  
    ...  
};
```

```
typedef struct TNode{  
    int a;  
    double b;  
    char* s;  
}TNode;
```



1. KIẾN THỨC CƠ SỞ

1.2. Cấu trúc

- Cấu trúc (struct) là một kiểu dữ liệu người dùng tự định nghĩa, bao gồm một tập hợp các biến, những biến này có thể có kiểu dữ liệu khác nhau

```
typedef struct TNode{  
    int a;  
    double b;  
    char* s;  
}TNode;
```

- TNode* q:** q là con trỏ trỏ đến 1 biến có kiểu TNode
- Q→a:** truy nhập đến thành phần a của kiểu cấu trúc
- q = (TNode*)malloc(sizeof(TNode)):** cấp phát bộ nhớ cho 1 kiểu TNode
~~và q trỏ vào vùng nhớ được cấp phát~~

NỘI DUNG TIẾP THEO

1. Kiến thức cơ sở

1.1. Con trỏ

1.2. Cấu trúc

2. Danh sách liên kết đơn

3. Thao tác trên danh sách liên kết

3.1 Duyệt danh sách

3.2 Tìm kiếm



2. DANH SÁCH LIÊN KẾT ĐƠN

2.1. Giới thiệu

- Danh sách liên kết đơn (Singly linked list) là một danh sách có thứ tự các phần tử; các phần tử được kết nối với nhau thông qua một liên kết (link)
- Danh sách liên kết và Mảng:***

	Danh sách liên kết	Mảng
Kiểu dữ liệu	Không cần đồng nhất	Đồng nhất
Cấp phát bộ nhớ	Phân tán	Liên tục, cạnh nhau



2. DANH SÁCH LIÊN KẾT ĐƠN

2.1. Giới thiệu

▪ Đặc điểm:

- Mỗi phần tử của danh sách gồm 2 phần: Dữ liệu và Con trỏ lưu trữ địa chỉ của phần tử kế tiếp trong danh sách;
- Trong danh sách liên kết đơn, mỗi phần tử chỉ trỏ tới một phần tử kế tiếp;
- Một danh sách liên kết có một phần tử đầu tiên – head và phần tử cuối, phần con trỏ của phần tử cuối cùng luôn null.

```
struct Node{  
    int value;  
    Node* next;  
};  
Node* head;
```

head



NỘI DUNG TIẾP THEO

1. Kiến thức cơ sở

1.1. Con trỏ

1.2. Cấu trúc

2. Danh sách liên kết đơn

3. Thao tác trên danh sách liên kết

3.1 Duyệt danh sách

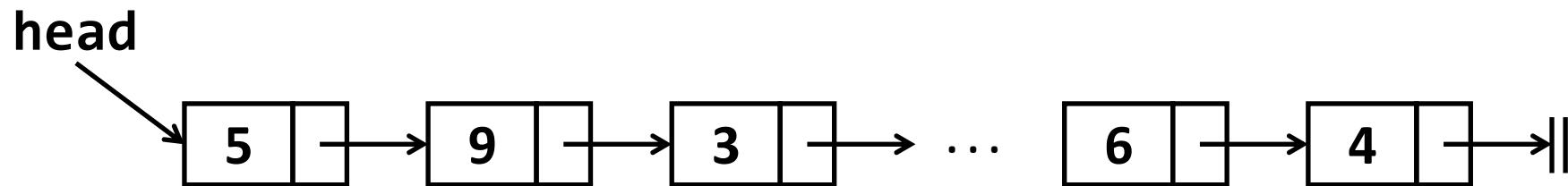
3.2 Tìm kiếm



3. THAO TÁC TRÊN DANH SÁCH LIÊN KẾT ĐƠN

3.1. Duyệt danh sách

- **Nhiệm vụ:** Thăm mỗi phần tử của danh sách đúng một lần
- **Ý tưởng:** Dùng con trỏ **next** để truy cập đến phần tử tiếp theo



3. THAO TÁC TRÊN DANH SÁCH LIÊN KẾT ĐƠN

3.1. Duyệt danh sách

- Nhiệm vụ: Thăm mỗi phần tử của danh sách đúng một lần

```
#include <stdio.h>
```

```
typedef struct Node{  
    int value;  
    struct Node* next;  
}Node;
```

```
//Create a physical node  
Node*makeNode (int v) {  
    Node* p = (Node*)malloc (sizeof (Node)) ;  
    p->value = v;  
    p->next = NULL;  
    return p;  
}
```

```
//Print a list  
void printList(Node* h) {  
    Node* p = h;  
    while (p != NULL) {  
        printf ("%d ", p->value);  
        p = p->next;  
    }  
}
```

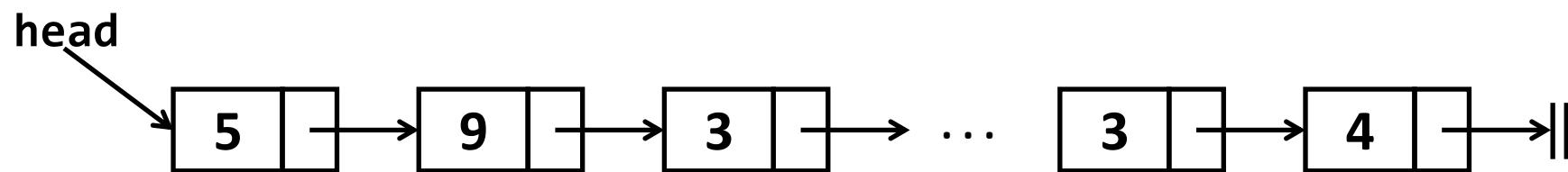
```
int main() {  
    Node* head, *node1, *node2;  
    head = makeNode (10);  
    node1 = makeNode (20);  
    node2 = makeNode (30);  
  
    head->next = node1;  
    node1->next = node2;  
  
    printList (head);  
    return 0;  
}
```



3. THAO TÁC TRÊN DANH SÁCH LIÊN KẾT ĐƠN

3.2. Tìm kiếm

- **Nhiệm vụ:** Tìm phần tử đầu tiên của danh sách có giá trị bằng giá trị đầu vào
- **Ý tưởng:** Dùng con trỏ **next** để truy cập đến phần tử tiếp theo
 - Ví dụ tìm phần tử đầu tiên của danh sách có giá trị 3



3. THAO TÁC TRÊN DANH SÁCH LIÊN KẾT ĐƠN

3.1. Duyệt danh sách

- Nhiệm vụ: Thăm mỗi phần tử của danh sách đúng một lần

```
#include <stdio.h>

typedef struct Node{
    int value;
    struct Node* next;
}Node;

//Create a physical node
Node*makeNode(int v){
    Node* p = (Node*)malloc(sizeof(Node));
    p->value = v;
    p->next = NULL;
    return p;
}

//Find a node with given value
Node * findFirst(Node * head, int val){
    Node* p = head;
    while(p != NULL){
        if(p->value == val)
            return p;
        p = p->next;
    }
    return NULL;
}
```

```
int main() {
    Node* head, *node1, *node2;
    head = makeNode(10);
    node1 = makeNode(20);
    node2 = makeNode(30);

    head->next = node1;
    node1->next = node2;

    Node * res = findFirst(head, 20);
    if(res != NULL){
        printf("Found");
    }else{
        printf("Not Found");
    }

    return 0;
}
```



TỔNG KẾT VÀ GỢI MỞ

1. Tổng kết:

Bài học đã giới thiệu **danh sách liên kết đơn** và 2 thao tác cơ bản trên danh sách đơn là **duyệt và tìm kiếm**

2. Gợi mở:

Thiết kế và cài đặt các thao tác khác trên danh sách



NỘI DUNG

1. Chèn một phần tử vào đầu danh sách
2. Chèn một phần tử vào cuối danh sách
3. Chèn một phần tử vào trước một phần tử của danh sách



MỤC TIÊU

Sau bài học này, người học có thể:

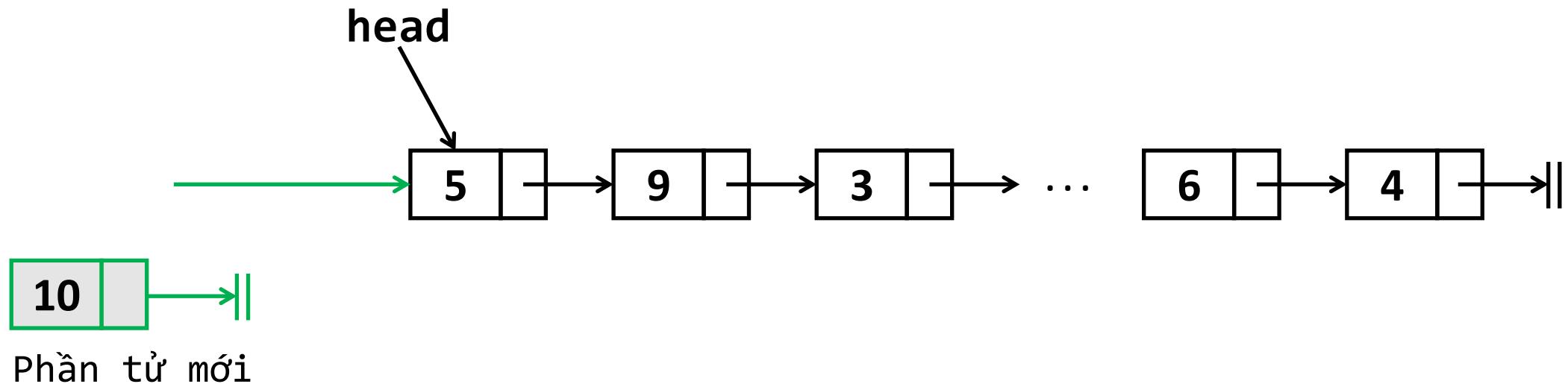
1. Hiểu thuật toán và cài đặt thành công ba thao tác cơ bản trên danh sách liên kết đơn: **chèn một phần tử vào đầu, cuối và trước một phần tử** trong danh sách liên kết đơn.



- 1. Chèn một phần tử vào đầu danh sách**
2. Chèn một phần tử vào cuối danh sách
3. Chèn một phần tử vào trước một phần tử của danh sách

1. CHÈN MỘT PHẦN TỬ VÀO ĐẦU DANH SÁCH

1.1. Ý tưởng



Bước 1: Tạo phần tử mới

Bước 2: Cập nhật head trở về phần tử mới

Bước 3: Cập nhật next của phần tử mới về đầu danh sách cũ, để biến phần tử mới thành phần tử đầu danh sách

1. CHÈN MỘT PHẦN TỬ VÀO ĐẦU DANH SÁCH

1.2. Cài đặt

```
Node* insertFirst(Node * head, int v){  
    Node * new_node = makeNode(v);  
  
    if(head == NULL) return new_node;  
    else{  
        new_node->next = head;  
        head = new_node;  
        return head;  
    }  
}
```

Tạo phần tử mới

Nếu danh sách hiện tại rỗng, trả về phần tử mới

- (1) Cập nhật head trở về phần tử mới
- (2) Cập nhật next của phần tử mới về đầu danh sách cũ, để biến phần tử mới thành phần tử đầu danh sách



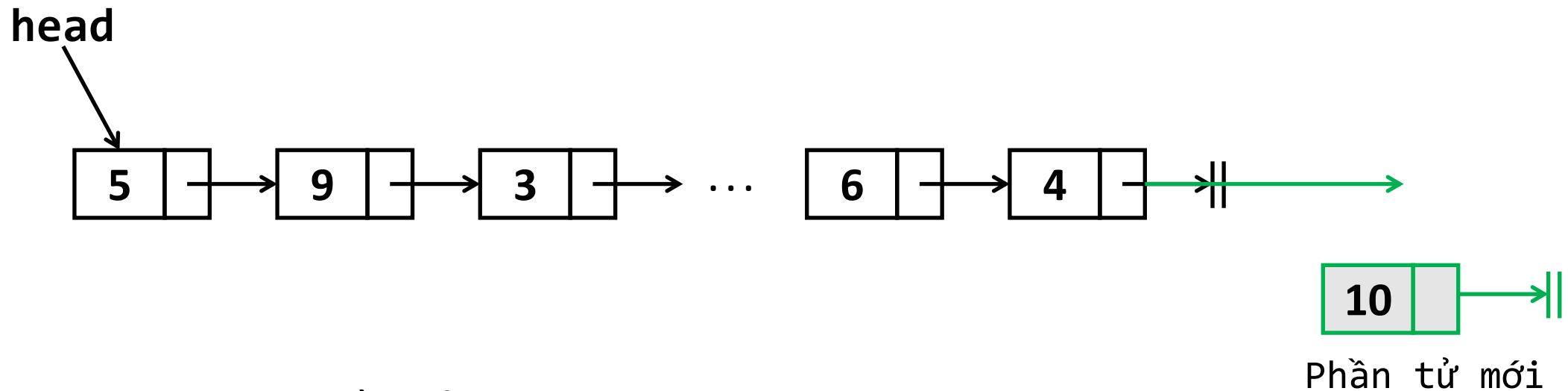
NỘI DUNG TIẾP THEO

1. Chèn một phần tử vào đầu danh sách
- 2. Chèn một phần tử vào cuối danh sách**
3. Chèn một phần tử vào trước một phần tử của danh sách



2. CHÈN MỘT PHẦN TỬ VÀO ĐẦU DANH SÁCH

2.1. Ý tưởng



Bước 1: Tạo phần tử mới

Bước 2: Tìm phần tử cuối cùng của danh sách

Bước 3: Cập nhật next của phần tử cuối cùng trở tới phần tử mới

2. CHÈN MỘT PHẦN TỬ VÀO ĐẦU DANH SÁCH

2.2. Cài đặt

```
Node * findLastNode (Node * head) {  
    Node* p = head;  
    while (p != NULL) {  
        if (p->next == NULL) return p;  
        p = p->next;  
    }  
    return NULL;  
}
```

Dùng vòng lặp để tìm phần tử cuối cùng của Danh sách

```
Node * insertLast (Node* head, int v) {  
    Node * new_node = makeNode (v);  
    if (head == NULL) return new_node;  
    else {  
        Node * lastNode = findLastNode (head);  
        lastNode->next = new_node;  
        return head;  
    }  
}
```

(1) Tạo phần tử mới
(2) Chèn phần tử mới vào sau phần tử cuối cùng của danh sách



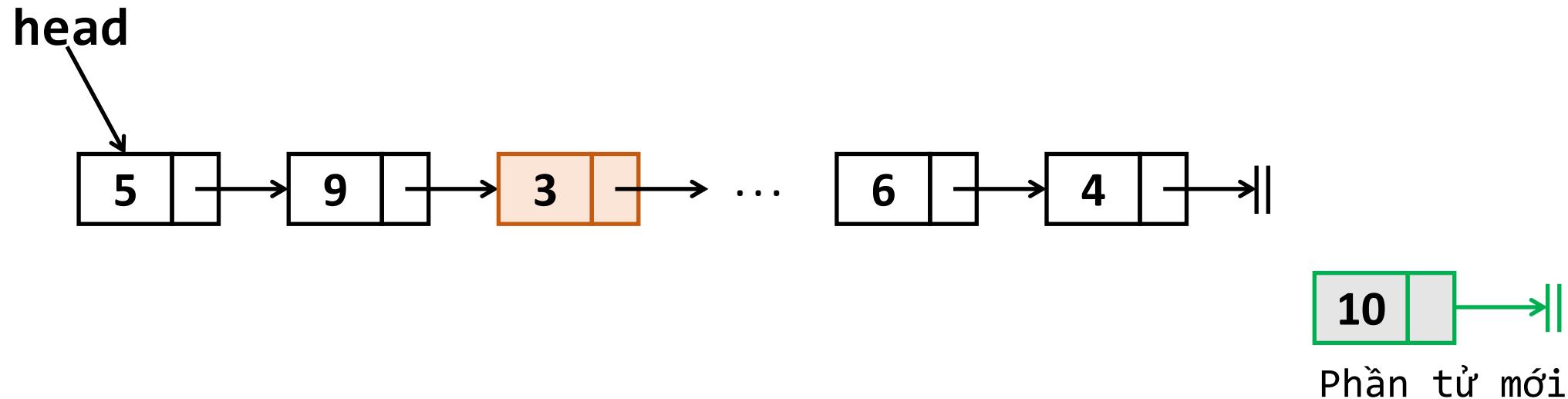
NỘI DUNG TIẾP THEO

1. Chèn một phần tử vào đầu danh sách
2. Chèn một phần tử vào cuối danh sách
- 3. Chèn một phần tử vào trước một phần tử của danh sách**



3. CHÈN MỘT PHẦN TỬ VÀO TRƯỚC MỘT PHẦN TỬ

3.1. Ý tưởng



Bước 1: Tạo phần tử mới

Bước 2: Cập nhật next của phần tử mới là phần tử bị chèn trước

Bước 3: Cập nhật next của phần tử ngay trước phần tử bị chèn là phần tử mới

3. CHÈN MỘT PHẦN TỬ VÀO TRƯỚC MỘT PHẦN TỬ

3.1. Ý tưởng

```
Node* prevNode (Node* head, Node* p) {  
    Node* q = head;  
    while (q != NULL) {  
        if (q->next == p) return q;  
        q = q->next;  
    }  
    return NULL;  
}
```

Tìm phần tử đứng trước một node cho trước

```
Node * insertBeforeNode (Node* head, Node* p, int v) {  
    Node* pp = prevNode (head, p);  
    if (pp == NULL && p != NULL) return head;  
    Node* q = makeNode (v);  
    if (pp == NULL) {  
        if (head == NULL)  
            return q;  
        q->next = head;  
        return q;  
    }  
    q->next = p;  
    pp->next = q;  
    return head;  
}
```

Thực hiện chèn

TỔNG KẾT VÀ GỢI MỞ

1. Tổng kết:

Cài đặt 3 thao tác chèn một phần tử mới vào một danh sách liên kết đơn: **chèn một phần tử vào đầu, cuối và trước một phần tử** của danh sách.

2. Gợi mở

Thiết kế và cài đặt các thao tác khác trên danh sách



NỘI DUNG

1. Xóa một phần tử của danh sách
2. Đảo ngược thứ tự các phần tử của danh sách



MỤC TIÊU

Sau bài học này, người học có thể:

Hiểu thuật toán và cài đặt thành công hai thao tác cơ bản trên danh sách liên kết đơn:

- xóa một phần tử khỏi danh sách
- đảo ngược thứ tự các phần tử trong danh sách



NỘI DUNG TIẾP THEO

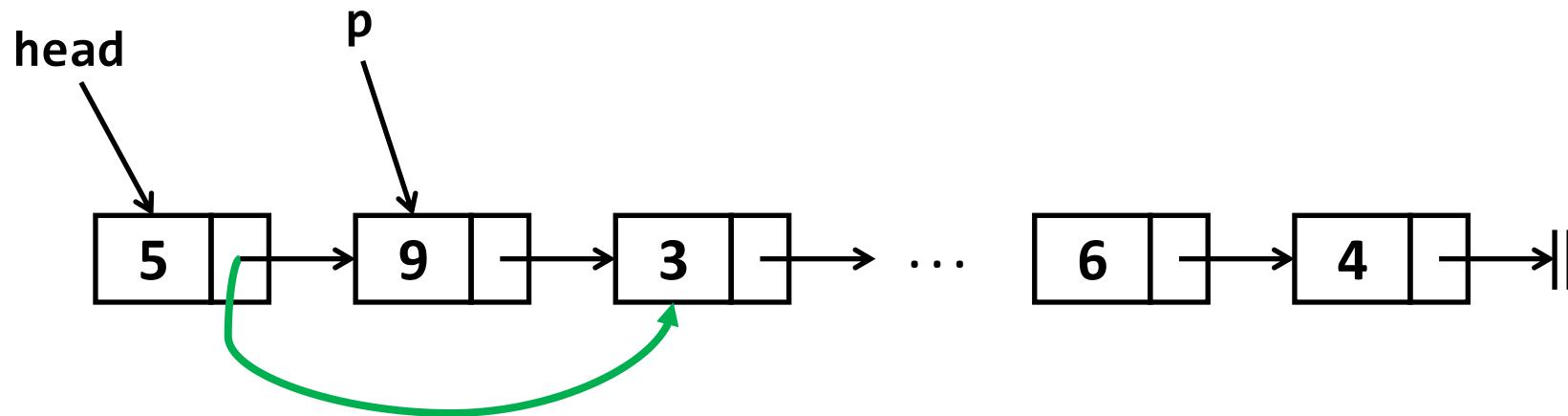
1. Xóa một phần tử của danh sách

2. Đảo ngược thứ tự các phần tử của danh sách



1. XÓA MỘT PHẦN TỬ RA KHỎI DANH SÁCH

1.1. Ý tưởng



- Kiểm tra danh sách và phần tử cần xóa p có NULL không?
- Nếu phần tử cần xóa là đầu danh sách → Đơn giản
- Dùng đệ quy để xóa

1. XÓA MỘT PHẦN TỬ RA KHỎI DANH SÁCH

1.2. Cài đặt

```
//Remove  
Node* removeNode (Node* head, Node * p) {  
    if(head == NULL || p == NULL) return head;  
    if(head == p) {  
        head = head->next;  
        free(p);  
        return head;  
    } else {  
        head->next = removeNode (head->next, p);  
        return head;  
    }  
}
```

Nếu danh sách hoặc phần tử cần xóa NULL, trả về đầu danh sách

Nếu phần tử cần xóa là phần tử đầu danh sách, đổi phần tử đầu và xóa phần tử cần xóa

Áp dụng kỹ thuật đệ quy để xóa



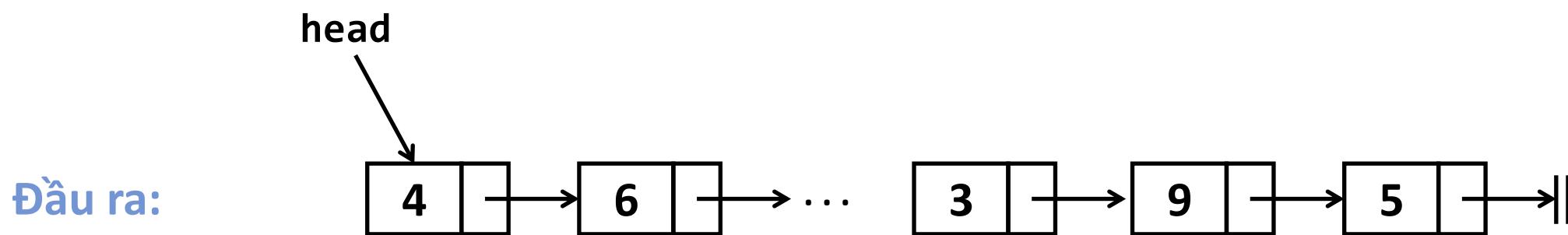
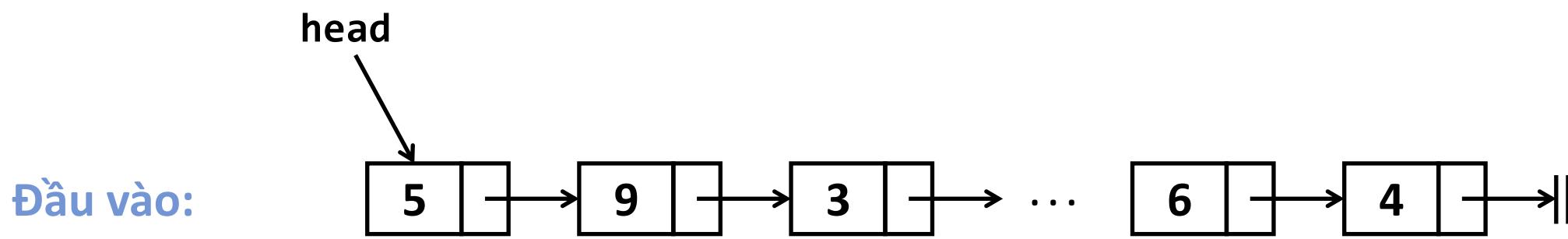
NỘI DUNG TIẾP THEO

1. Xóa một phần tử của danh sách
2. Đảo ngược thứ tự các phần tử của danh sách



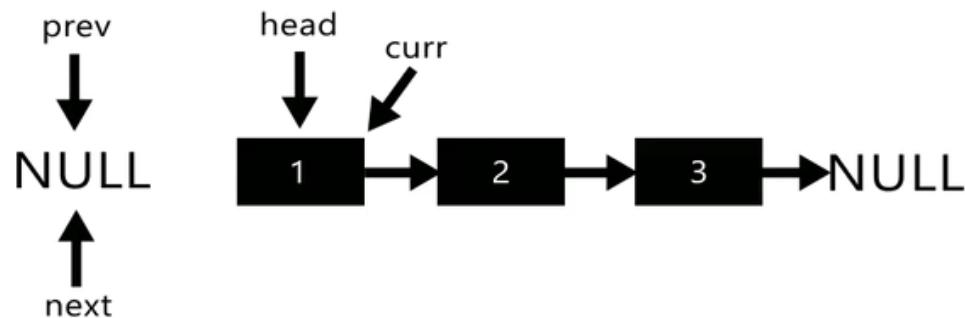
2. ĐẢO NGƯỢC THỨ TỰ CÁC PHẦN TỬ CỦA DANH SÁCH

2.1. Bài toán



2. ĐẢO NGƯỢC THỨ TỰ CÁC PHẦN TỬ CỦA DANH SÁCH

2.2. Ý tưởng



```
while (current != NULL)
{
    next = current->next;
    current->next = prev;
    prev = current;
    current = next;
}
*head_ref = prev;
```

2. ĐẢO NGƯỢC THỨ TỰ CÁC PHẦN TỬ CỦA DANH SÁCH

2.3. Cài đặt

```
//Reverse
Node* reverse(Node* head) {
    Node* cur = head;
    Node* next, *pre;
    pre = NULL;
    next = NULL;

    while(cur != NULL) {
        //Get next
        next = cur->next;
        //Reverse
        cur->next = pre;
        //Move points ahead
        pre = cur;
        cur = next;
    }
    head = pre;
    return head;
}
```



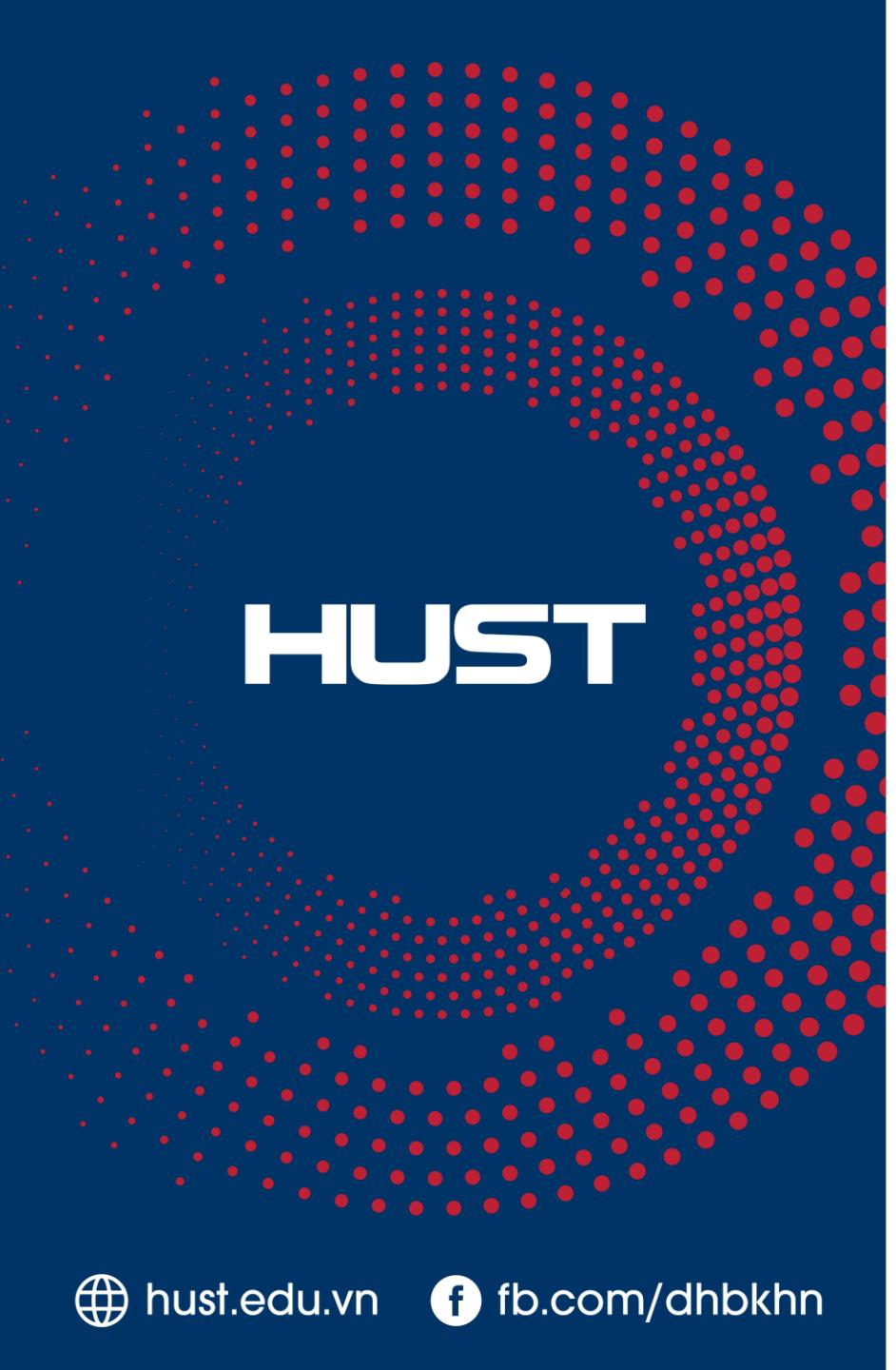
TỔNG KẾT VÀ GỢI MỞ

1. **Tổng kết:** Cài đặt 2 thao tác quan trọng trên danh sách liên kết đơn: **loại bỏ một phần tử ra khỏi danh sách và đảo ngược thứ tự các phần tử của danh sách.**

2. **Gợi mở:**

- Danh sách liên kết đơn chỉ có có 1 liên kết giữa 2 phần tử liên tiếp trong danh sách, nếu có 2 liên kết thì thao tác trên danh sách có dễ dàng hơn không?



A large, faint watermark of the HUST logo is visible across the entire slide, consisting of a grid of red dots.

HUST

THANK YOU !

HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.



ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

Ngăn xếp, hàng đợi

ONE LOVE. ONE FUTURE.

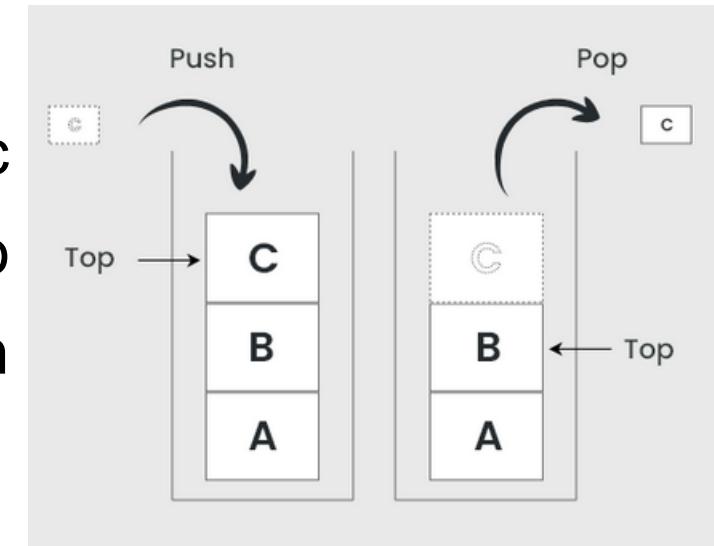
Nội dung

- Ngăn xếp
- Bài tập: Kiểm tra tính cân xứng của dãy ngoặc
- Hàng đợi
- Bài tập: Tìm đường đi nhanh nhất thoát khỏi mê cung



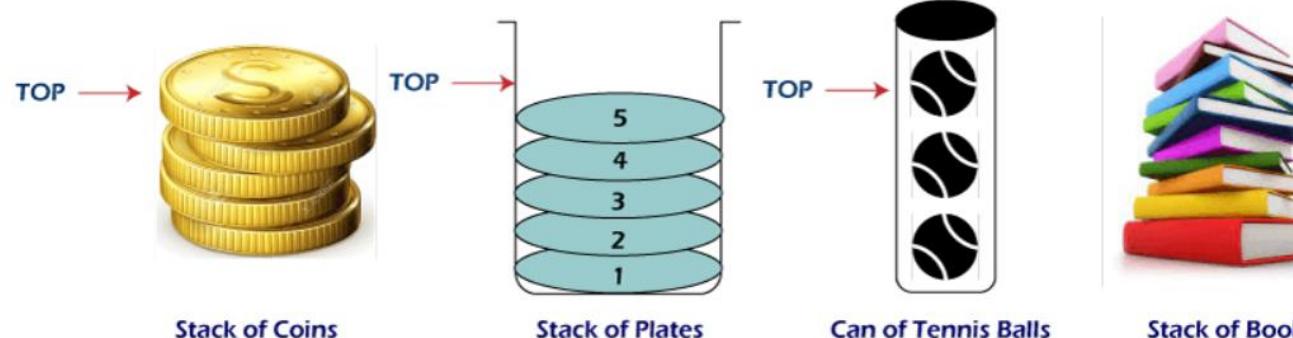
Giới thiệu ngăn xếp

- Ngăn xếp là một cấu trúc dữ liệu tuyến tính (Linear Data Structure);
- Thao tác thêm mới và loại bỏ phần tử trên danh sách được thực hiện ở 1 đầu (đỉnh hay **top**) của danh sách theo nguyên tắc Last In First Out (phần tử cuối cùng được thêm vào ngăn xếp là phần tử đầu tiên bị loại bỏ).

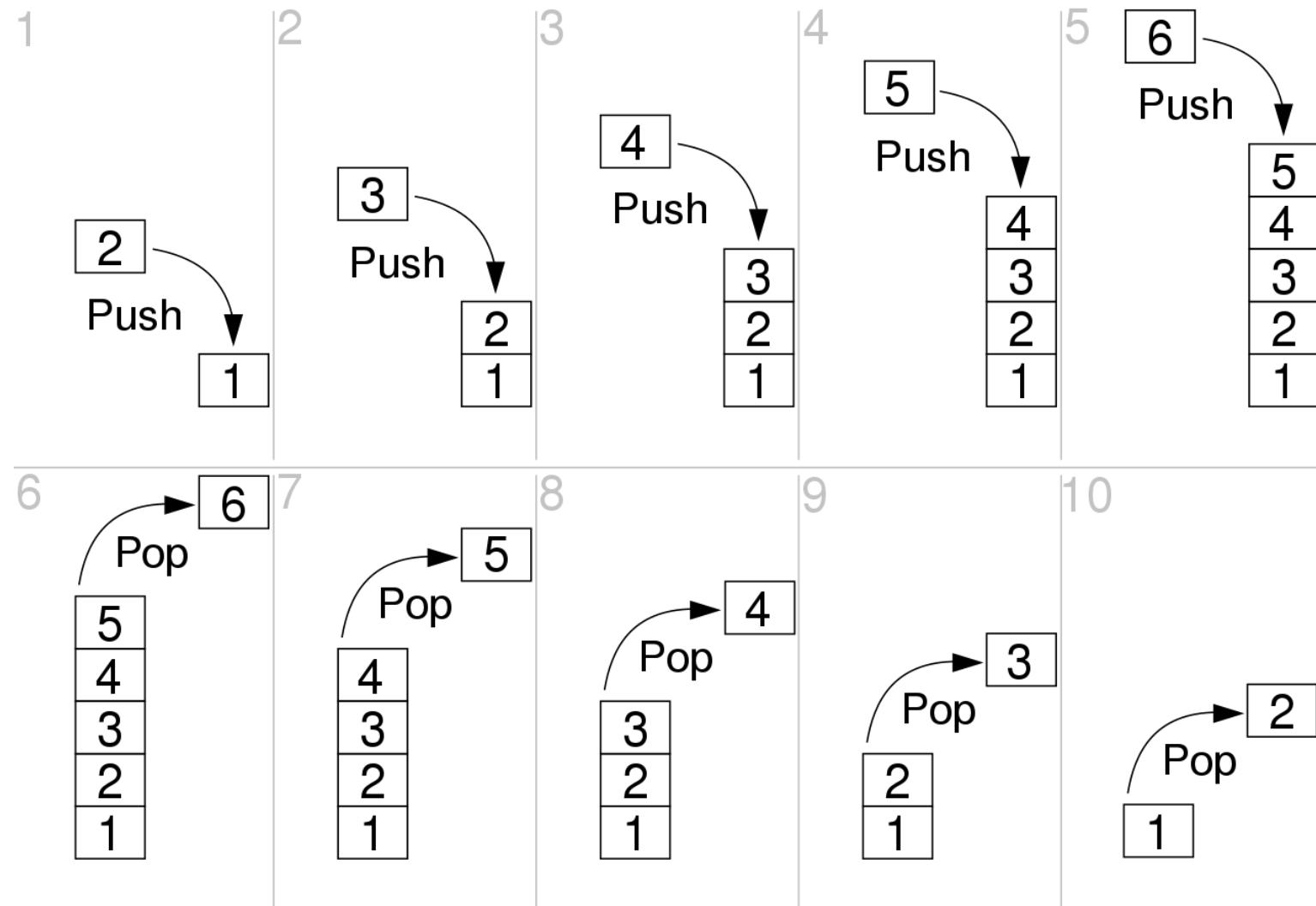


- Thao tác cơ bản trên ngăn xếp S :

- $Push(x, S)$: chèn 1 phần tử x vào ngăn xếp
- $Pop(S)$: lấy ra 1 phần tử đầu ra khỏi ngăn xếp
- $Top(S)$: truy cập phần tử ở đỉnh của ngăn xếp
- $isEmpty(S)$: trả về true nếu ngăn xếp rỗng



Giới thiệu ngăn xếp



Nguồn: https://en.wikipedia.org/wiki/Stack_%28abstract_data_type%29

Cài đặt ngăn xếp sử dụng danh sách liên kết đơn

- Cấu trúc và hàm khởi tạo thành phần

```
typedef struct Node{
    char value;
    struct Node* next;
}Node;

Node*makeNode (char v) {
    Node* p = (Node*)malloc(sizeof(Node));
    p->value = v;
    p->next = NULL;
    return p;
}
```

Khai báo struct cho một phần tử của ngăn xếp

Tạo một phần tử (node) của ngăn xếp

Cài đặt ngăn xếp sử dụng danh sách liên kết đơn

- Thao tác Push

```
Node* push(Node * head, char v) {
    Node * new_node = makeNode(v);
    if(head == NULL) return new_node;
    else {
        new_node->next = head;
        head = new_node;
        return head;
    }
}
```

Tạo một node mới

Nếu ngăn xếp rỗng, trả về phần tử mới tạo

Nếu ngăn xếp không rỗng, biến node mới tạo thành phần tử đầu danh sách

Cài đặt ngăn xếp sử dụng danh sách liên kết đơn

- Thao tác Pop

```
Node* pop (Node* head) {  
    if (head == NULL) return head;  
    Node* p = head;  
    head = head->next;  
    free (p);  
    return head;  
}
```

Nếu ngăn xếp rỗng, trả về rỗng

Nếu ngăn xếp không rỗng, xóa phần tử đầu của ngăn xếp và trả về phần tử đầu của ngăn xếp

Cài đặt ngăn xếp sử dụng danh sách liên kết đơn

- Thao tác Top và isEmpty

```
char top(Node* head) {
    return head->value;
}

bool isEmpty(Node* head) {
    if(head == NULL) return true;

    return false;
}
```



Một số ứng dụng ngăn xếp

- **Ngăn xếp lời gọi hàm:** Ngăn xếp được sử dụng rộng rãi để quản lý cuộc gọi hàm và biến cục bộ trong ngôn ngữ lập trình. Khi một hàm được gọi, ngữ cảnh của nó (bao gồm đối số và biến cục bộ) được đẩy vào ngăn xếp. Khi hàm trả về, ngữ cảnh của nó được đẩy ra khỏi ngăn xếp, cho phép lồng ghép hàm một cách chính xác.
- **Cơ chế quay lại (Undo):** Ngăn xếp được sử dụng trong các ứng dụng yêu cầu tính năng Hoàn tác, như trình soạn thảo văn bản, phần mềm đồ họa hoặc hệ thống quản lý phiên bản. Mỗi hành động của người dùng có thể được đẩy vào ngăn xếp, và việc hoàn tác một hành động liên quan đến việc lấy nó ra khỏi ngăn xếp để hoàn ngược thay đổi.



Một số ứng dụng ngăn xếp

- **Thuật toán Backtracking:** Trong các thuật toán như tìm kiếm theo chiều sâu (DFS) và các thuật toán backtracking (Ví dụ: giải quyết các câu đố như vấn đề N-Queens hoặc Sudoku), một ngăn xếp có thể được sử dụng để theo dõi các nút hoặc trạng thái đã được thăm. Điều này cho phép dễ dàng quay lại để khám phá các lựa chọn thay thế khi cần.
- **Phân tích Biểu thức:** Ngăn xếp được sử dụng để phân tích và hiểu biểu thức trong các trình biên dịch và trình thông dịch. Chúng giúp duy trì thứ tự ưu tiên của các toán tử và đánh giá biểu thức một cách chính xác.



Một số ứng dụng ngăn xếp

- **Quản lý Bộ nhớ:** Ngăn xếp đóng vai trò quan trọng trong quản lý bộ nhớ trong các hệ thống máy tính. Chúng được sử dụng để quản lý ngăn xếp gọi hàm, nơi lưu trữ thông tin về cuộc gọi hàm và biến cục bộ. Ngăn xếp giúp phân bổ bộ nhớ cho cuộc gọi hàm và giải phóng nó khi các hàm trả về, ngăn chặn rò rỉ bộ nhớ.
- **Phân tích Biểu thức:** Ngăn xếp được sử dụng để phân tích và hiểu biểu thức trong các trình biên dịch và trình thông dịch. Chúng giúp duy trì thứ tự ưu tiên của các toán tử và đánh giá biểu thức một cách chính xác.

Nội dung

- Ngăn xếp
- Bài tập: Kiểm tra tính cân xứng của dãy ngoặc
- Hàng đợi
- Bài tập: Tìm đường đi nhanh nhất thoát khỏi mê cung



Bài toán kiểm tra tính cân xứng của dãy ngoặc

- Cho dãy dấu ngoặc E mà mỗi phần tử là một dấu ngoặc thuộc một trong các loại: (,), [], {}, {}. Viết chương trình kiểm tra xem dãy ngoặc đó có cân xứng (balanced) hay không?
- Ví dụ:
 - ()[{}([])]: cân xứng
 - ()[{}([])]: không cân xứng



Bài toán kiểm tra tính cân xứng của dãy ngoặc

- Dữ liệu đầu vào:

- Một dòng duy nhất chứa xâu ký tự thể hiện dãy dấu ngoặc

- Kết quả đầu ra:

- Ghi 1 nếu dãy dấu ngoặc là cân xứng và ghi 0, nếu dãy dấu ngoặc không cân xứng

stdin	stdout
() [{ } ([])]	1

stdin	stdout
() [{ } ([])]	0

Bài toán kiểm tra tính cân xứng của dãy ngoặc

▪ Thuật toán

- Khởi tạo một ngăn xếp rỗng S
- Duyệt dãy dấu ngoặc từ trái qua phải
 - Nếu gặp ngoặc mở A thì đưa ngoặc mở đó vào S
 - Nếu gặp ngoặc đóng B
 - Nếu S rỗng thì kết luận dãy ngoặc E không cân xứng
 - Nếu S không rỗng
 - Lấy một ngoặc mở A khỏi ngăn xếp S
 - Nếu A và B không cân xứng (ngoặc mở và đóng khác loại) thì kết luận E không cân xứng
- Kết thúc duyệt, nếu S không rỗng thì kết luận E không cân xứng, ngược lại thì E là cân xứng



Minh họa

- **Minh họa số 1: dãy ngoặc () [({ })]**

- Khởi tạo ngăn xếp rỗng, thực hiện duyệt dãy ngoặc từ trái qua phải



Minh họa

- **Minh họa số 1:** dãy ngoặc ()[({})]

- Bước 1: Xét ngoặc tiếp theo là ngoặc mở “(“ → đưa ngoặc mở vào ngăn xếp



Minh họa

- **Minh họa số 1:** dãy ngoặc ()[({})]

- Bước 1: Xét ngoặc tiếp theo là ngoặc mở “(“ → đưa ngoặc mở vào ngăn xếp
- Bước 2: Xét ngoặc tiếp theo là ngoặc đóng “)” → lấy ngoặc mở ra khỏi ngăn xếp và so khớp “(“ ”)” → kết quả là đúng nên ta tiếp tục duyệt



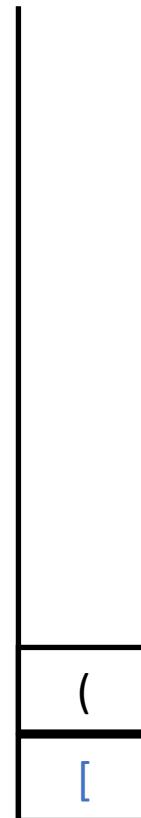
■ Minh họa số 1: dãy ngoặc () [({ })]

- Bước 1: Xét ngoặc tiếp theo là ngoặc mở “(“ → đưa ngoặc mở vào ngăn xếp
- Bước 2: Xét ngoặc tiếp theo là ngoặc đóng “)” → lấy ngoặc mở ra khỏi ngăn xếp và so khớp “(“ “)” → kết quả là đúng nên ta tiếp tục duyệt
- Bước 3: Gặp ngoặc mở “[“ → đưa ngoặc mở này vào ngăn xếp



- **Minh họa số 1:** dãy ngoặc () [({ })]

- Bước 1: Xét ngoặc tiếp theo là ngoặc mở “(” → đưa ngoặc mở vào ngăn xếp
- Bước 2: Xét ngoặc tiếp theo là ngoặc đóng “)” → lấy ngoặc mở ra khỏi ngăn xếp và so khớp “(” “)” → kết quả là đúng nên ta tiếp tục duyệt
- Bước 3: Gặp ngoặc mở “[” → đưa ngoặc mở này vào ngăn xếp
- Bước 4: Gặp ngoặc mở “(” → đưa ngoặc mở này vào ngăn xếp



Minh họa

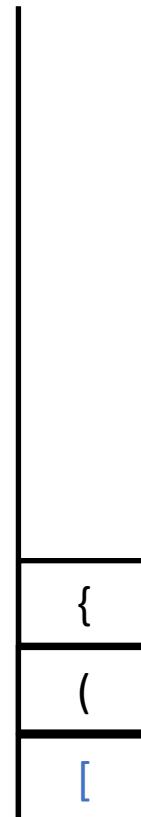
▪ Minh họa số 1: dãy ngoặc () [({ })]

- Bước 1: Xét ngoặc tiếp theo là ngoặc mở "(" → đưa ngoặc mở vào ngăn xếp
- Bước 2: Xét ngoặc tiếp theo là ngoặc đóng ")" → lấy ngoặc mở ra khỏi ngăn xếp và so khớp "(" ")" → kết quả là đúng nên ta tiếp tục duyệt
- Bước 3: Gặp ngoặc mở "[" → đưa ngoặc mở này vào ngăn xếp
- Bước 4: Gặp ngoặc mở "(" → đưa ngoặc mở này vào ngăn xếp
- Bước 5: Gặp ngoặc mở "{" → đưa ngoặc mở này vào ngăn xếp



▪ Minh họa số 1: dãy ngoặc () [({ })]

- Bước 1: Xét ngoặc tiếp theo là ngoặc mở "(" → đưa ngoặc mở vào ngăn xếp
- Bước 2: Xét ngoặc tiếp theo là ngoặc đóng ")" → lấy ngoặc mở ra khỏi ngăn xếp và so khớp "(" ")" → kết quả là đúng nên ta tiếp tục duyệt
- Bước 3: Gặp ngoặc mở "[" → đưa ngoặc mở này vào ngăn xếp
- Bước 4: Gặp ngoặc mở "(" → đưa ngoặc mở này vào ngăn xếp
- Bước 5: Gặp ngoặc mở "{" → đưa ngoặc mở này vào ngăn xếp
- Bước 6: Gặp ngoặc đóng "}" → lấy 1 ngoặc mở là "{" ra khỏi ngăn xếp



Minh họa

▪ Minh họa số 1: dãy ngoặc () [({ })]

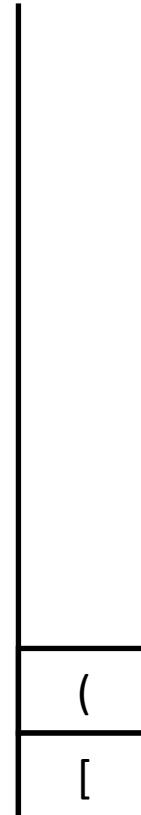
- Bước 5: Gặp ngoặc mở “{“ → đưa ngoặc mở này vào ngăn xếp
- Bước 6: Gặp ngoặc đóng “}” → lấy 1 ngoặc mở là “{“ ra khỏi ngăn xếp
- So khớp “{“ “}” → kết quả là đúng nên ta tiếp tục duyệt



Minh họa

- **Minh họa số 1:** dãy ngoặc () [({ })]

- Bước 5: Gặp ngoặc mở “{“ → đưa ngoặc mở này vào ngăn xếp
- Bước 6: Gặp ngoặc đóng “}” → lấy 1 ngoặc mở là “{“ ra khỏi ngăn xếp
 - So khớp “{“ “}” → kết quả là đúng nên ta tiếp tục duyệt
- Bước 7: Gặp ngoặc đóng “)” → lấy 1 ngoặc mở là “(“ ra khỏi ngăn xếp



Minh họa

- **Minh họa số 1: dãy ngoặc () [({ })]**

- Bước 5: Gặp ngoặc mở “{“ → đưa ngoặc mở này vào ngăn xếp
- Bước 6: Gặp ngoặc đóng “}” → lấy 1 ngoặc mở là “{“ ra khỏi ngăn xếp
 - So khớp “{“ “}” → kết quả là đúng nên ta tiếp tục duyệt
- Bước 7: Gặp ngoặc đóng “)” → lấy 1 ngoặc mở là “(“ ra khỏi ngăn xếp
 - So khớp “(“ “)” → kết quả là đúng nên ta tiếp tục duyệt

[



- **Minh họa số 1: dãy ngoặc () [({ })]**

- Bước 5: Gặp ngoặc mở “{“ → đưa ngoặc mở này vào ngăn xếp
- Bước 6: Gặp ngoặc đóng “}” → lấy 1 ngoặc mở là “{“ ra khỏi ngăn xếp
 - So khớp “{“ “}” → kết quả là đúng nên ta tiếp tục duyệt
- Bước 7: Gặp ngoặc đóng “)” → lấy 1 ngoặc mở là “(“ ra khỏi ngăn xếp
 - So khớp “(“ “)” → kết quả là đúng nên ta tiếp tục duyệt
- Bước 8: Gặp ngoặc đóng “]” → lấy 1 ngoặc mở là “[“ ra khỏi ngăn xếp

[

Minh họa

- **Minh họa số 1:** dãy ngoặc () [({ })]
 - Bước 5: Gặp ngoặc mở “{” → đưa ngoặc mở này vào ngăn xếp
 - Bước 6: Gặp ngoặc đóng “}” → lấy 1 ngoặc mở là “{“ ra khỏi ngăn xếp
 - So khớp “{“ “}” → kết quả là đúng nên ta tiếp tục duyệt
 - Bước 7: Gặp ngoặc đóng “)” → lấy 1 ngoặc mở là “(“ ra khỏi ngăn xếp
 - So khớp “(“ “)” → kết quả là đúng nên ta tiếp tục duyệt
 - Bước 8: Gặp ngoặc đóng “[” → lấy 1 ngoặc mở là “[“ ra khỏi ngăn xếp
 - So khớp “[“ “[”” → lúc này dãy ngoặc đã duyệt xong và ngăn xếp rỗng, nên kết quả dãy ngoặc này là cân xứng



Minh họa

- **Minh họa số 2:** dãy ngoặc () [({ })]
 - Khởi tạo ngăn xếp rỗng và duyệt dãy ngoặc từ trái qua phải



- **Minh họa số 2:** dãy ngoặc () [({ })]

- Bước 1: Gặp ngoặc tiếp theo là ngoặc mở “(“ → đưa ngoặc mở này vào ngăn xếp



- **Minh họa số 2:** dãy ngoặc () [({ })]

- Bước 1: Gặp ngoặc tiếp theo là ngoặc mở “(“ → đưa ngoặc mở này vào ngăn xếp
- Bước 2: Gặp ngoặc tiếp theo là ngoặc đóng “)” → lấy 1 ngoặc mở là “(“ ra khỏi ngăn xếp



- **Minh họa số 2:** dãy ngoặc () [({ })]

- Bước 1: Gặp ngoặc tiếp theo là ngoặc mở “(“ → đưa ngoặc mở này vào ngăn xếp
- Bước 2: Gặp ngoặc tiếp theo là ngoặc đóng “)” → lấy 1 ngoặc mở là “(“ ra khỏi ngăn xếp
 - So khớp “(“ “)” → kết quả đúng nên ta duyệt tiếp



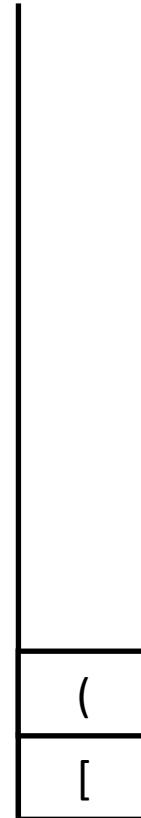
- **Minh họa số 2:** dãy ngoặc () [({ })]

- Bước 1: Gặp ngoặc tiếp theo là ngoặc mở “(“ → đưa ngoặc mở này vào ngăn xếp
- Bước 2: Gặp ngoặc tiếp theo là ngoặc đóng “)” → lấy 1 ngoặc mở là “(“ ra khỏi ngăn xếp
 - So khớp “(“ “)” → kết quả đúng nên ta duyệt tiếp
- Bước 3: Gặp ngoặc tiếp theo là ngoặc mở “[“ → đưa ngoặc mở này vào ngăn xếp

[

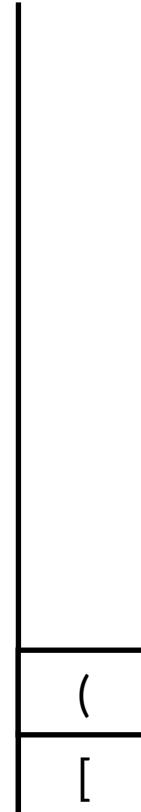
▪ Minh họa số 2: dãy ngoặc () [({ })]

- Bước 1: Gặp ngoặc tiếp theo là ngoặc mở “(“ → đưa ngoặc mở này vào ngăn xếp
- Bước 2: Gặp ngoặc tiếp theo là ngoặc đóng “)” → lấy 1 ngoặc mở là “(“ ra khỏi ngăn xếp
 - So khớp “(“ “)” → kết quả đúng nên ta duyệt tiếp
- Bước 3: Gặp ngoặc tiếp theo là ngoặc mở “[“ → đưa ngoặc mở này vào ngăn xếp



▪ Minh họa số 2: dãy ngoặc () [({})]

- Bước 1: Gặp ngoặc tiếp theo là ngoặc mở “(“ → đưa ngoặc mở này vào ngăn xếp
- Bước 2: Gặp ngoặc tiếp theo là ngoặc đóng “)” → lấy 1 ngoặc mở là “(“ ra khỏi ngăn xếp
 - So khớp “(“ “)” → kết quả đúng nên ta duyệt tiếp
- Bước 3: Gặp ngoặc tiếp theo là ngoặc mở “[“ → đưa ngoặc mở này vào ngăn xếp
- Bước 4: Gặp ngoặc tiếp theo là ngoặc mở “(“ → đưa ngoặc mở này vào ngăn xếp
- Bước 5: Gặp ngoặc tiếp theo là ngoặc đóng “}” → lấy 1 ngoặc mở là ngoặc “(“ ra



Minh họa

- Bước 4: Gặp ngoặc tiếp theo là ngoặc mở “(“ → đưa ngoặc mở này vào ngăn xếp
- Bước 5: Gặp ngoặc tiếp theo là ngoặc đóng “}” → lấy 1 ngoặc mở là ngoặc “(“ ra
 - So khớp “(“ “}” → kết quả so khớp là sai nên ta kết luận dãy ngoặc đã cho không cân xứng

[



Cài đặt thuật toán

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef struct Node{
    char c;
    struct Node* next;
}Node;
Node* top;
char s[1000001];
Node* makeNode(char c){
    Node* p=(Node*)malloc(sizeof(Node));
    p->c = c; p->next = NULL; return p;
}
```



Cài đặt thuật toán

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef struct Node{
    char c;
    struct Node* next;
}Node;
Node* top;
char s[1000001];
Node* makeNode(char c){
    Node* p=(Node*)malloc(sizeof(Node));
    p->c = c; p->next = NULL; return p;
}
```

```
void push(char c){
    Node* p = makeNode(c);
    p->next = top; top = p;
}
char pop(){
    if(top == NULL) return ' ';
    Node* tmp = top; top = top->next;
    char res = tmp->c;
    free(tmp);
    return res;
}
```



Cài đặt thuật toán

```
int match(char a, int b){  
    if(a == '(' && b == ')') return 1;  
    if(a == '{' && b == '}') return 1;  
    if(a == '[' && b == ']') return 1;  
    return 0;  
}
```



Cài đặt thuật toán

```
int match(char a, int b){  
    if(a == '(' && b == ')') return 1;  
    if(a == '{' && b == '}') return 1;  
    if(a == '[' && b == ']') return 1;  
    return 0;  
}
```

```
int check(char* s){  
    for(int i = 0; i < strlen(s); i++){  
        if(s[i] == '(' || s[i] == '{' || s[i] == '[')  
            push(s[i]);  
        else{  
            if(top==NULL) return 0;  
            char o = pop();  
            if(!match(o,s[i])) return 0;  
        }  
    }  
    return top == NULL;  
}
```



Cài đặt thuật toán

```
int match(char a, int b){  
    if(a == '(' && b == ')') return 1;  
    if(a == '{' && b == '}') return 1;  
    if(a == '[' && b == ']') return 1;  
    return 0;  
}
```

```
int main(){  
    scanf("%s",s);  
    int res = check(s);  
    printf("%d",res);  
    return 0;  
}
```

```
int check(char* s){  
    for(int i = 0; i < strlen(s); i++){  
        if(s[i] == '(' || s[i] == '{' || s[i] == '[')  
            push(s[i]);  
        else{  
            if(top==NULL) return 0;  
            char o = pop();  
            if(!match(o,s[i])) return 0;  
        }  
    }  
    return top == NULL;  
}
```



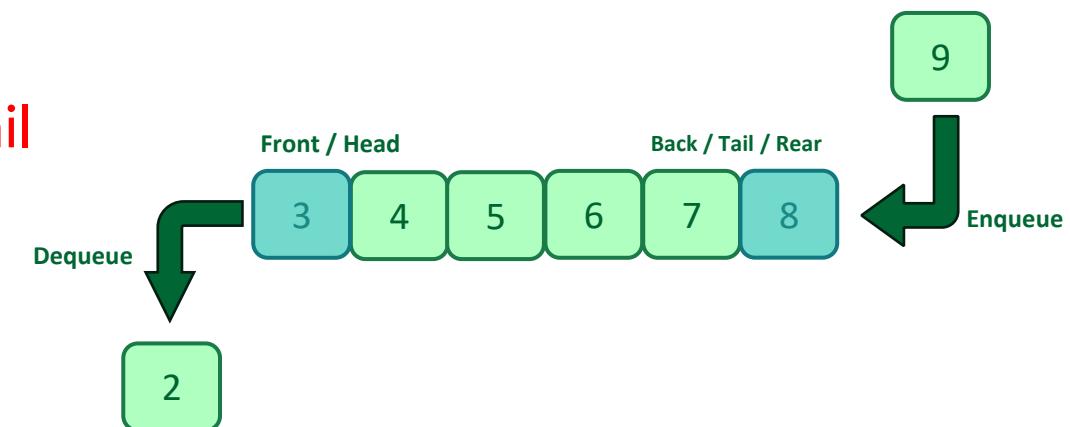
Nội dung

- Ngăn xếp
- Bài tập: Kiểm tra tính cân xứng của dãy ngoặc
- Hàng đợi
- Bài tập: Tìm đường đi nhanh nhất thoát khỏi mê cung



Giới thiệu hàng đợi

- Hàng đợi là một cấu trúc dữ liệu tuyến tính có hai đầu, **head** và **tail**
- Thao tác thêm mới phần tử được thực hiện ở **tail**
Thao tác loại bỏ phần tử được thực hiện ở **head**
- Nguyên tắc hoạt động: FIFO – First In First Out
- Thao tác cơ bản trên hàng đợi Q:
 - enqueue(x, Q): chèn 1 phần tử x vào hàng đợi
 - dequeue(Q): lấy ra 1 phần tử khỏi hàng đợi
 - isEmpty(Q): trả về true nếu hàng đợi rỗng



Ý tưởng hoạt động của hàng đợi



Minh họa hoạt động của hàng đợi

Thao tác	Trạng thái hàng đợi	Phần tử trả về nếu có
Create an empty queue	Empty	-
Enqueue 1	1	-
Enqueue 2	1, 2	-
Enqueue 3	1, 2, 3	-
Dequeue	2, 3	1
Enqueue 4	2, 3, 4	-
Enqueue 5	2, 3, 4, 5	-
Dequeue	3, 4, 5	2
Dequeue	4, 5	3
Check if empty	4, 5	0



Cài đặt hàng đợi sử dụng danh sách liên kết đơn

- Cấu trúc và hàm khởi tạo thành phần

```
typedef struct Node {  
    char value;  
    struct Node* next;  
} Node;
```

Khai báo struct cho một phần tử
của hàng đợi

```
Node* makeNode (char v) {  
    Node* p = (Node*) malloc (sizeof (Node) ) ;  
    p->value = v;  
    p->next = NULL;  
    return p;  
}
```

Tạo một phần tử (node) của hàng
đợi



Cài đặt hàng đợi sử dụng danh sách liên kết đơn

- Thao tác Enqueue

```
Node* enqueue (Node * head, char v) {  
    Node * new_node = makeNode (v);  
    if (head == NULL) return new_node;  
    else {  
        new_node->next = head;  
        head = new_node;  
        return head;  
    }  
}
```

Tạo một node mới

Nếu hàng đợi rỗng, trả về phần tử mới tạo

Nếu hàng đợi không rỗng, biến node mới tạo thành phần tử đầu danh sách

Cài đặt hàng đợi sử dụng danh sách liên kết đơn

• Thao tác Dequeue

```
char dequeue (Node** head) {  
    if (*head == NULL) return NULL;  
    Node* p, *q;  
    p = *head;  
    char ans;
```

Nếu hàng đợi rỗng, trả về rỗng

```
if (p->next == NULL) {  
    ans = p->value;  
    free(p);  
    *head = NULL;  
    return ans;  
}
```

Nếu hàng đợi chỉ có một phần tử

```
while (p->next->next != NULL) {  
    p = p->next;  
}  
q = p->next;  
p->next = NULL;  
ans = q->value;  
free(q);  
  
return ans;
```

Nếu hàng đợi có nhiều hơn một phần tử, lấy phần tử cuối cùng của danh sách

Một số ứng dụng của hàng đợi

- **Tìm kiếm theo chiều rộng (BFS):** BFS là một thuật toán để duyệt hoặc tìm kiếm trong cấu trúc dữ liệu cây và đồ thị. Nó sử dụng một hàng đợi để khám phá các nút theo từng cấp độ, làm cho nó trở thành công cụ quan trọng để giải quyết các vấn đề liên quan đến đồ thị.
- **Lập lịch công việc (Task scheduling):** Hàng đợi được sử dụng trong các hệ điều hành để lên lịch cho các tác vụ hoặc tiến trình để thực thi. Các tác vụ được đặt vào hàng đợi và hệ điều hành thực hiện chúng theo thứ tự chúng được thêm vào, đảm bảo sự công bằng trong phân phối tài nguyên.

Một số ứng dụng của hàng đợi

- **Xử lý yêu cầu trên máy chủ web (Web server request handling):** Các máy chủ web sử dụng hàng đợi để quản lý các yêu cầu HTTP đến. Mỗi yêu cầu đến được đặt vào hàng đợi và xử lý bởi các luồng hoặc tiến trình làm việc, cho phép máy chủ xử lý nhiều yêu cầu cùng một lúc
- **Đệm trong các hoạt động I/O (Buffering in I/O operations):** Hàng đợi được sử dụng để đệm dữ liệu trong các hoạt động I/O. Ví dụ, khi dữ liệu được đọc từ tệp hoặc ổ đĩa mạng, thường được đặt vào hàng đợi trước khi xử lý để làm dịu sự biến đổi trong tốc độ nhận dữ liệu.

Một số ứng dụng của hàng đợi

- **Quản lý tác vụ trong ứng dụng đa luồng (Task management in multithreading):**
Trong các ứng dụng đa luồng, hàng đợi có thể được sử dụng để quản lý các tác vụ cần được thực thi đồng thời. Các luồng thực hiện bóc tác vụ từ hàng đợi và thực hiện công việc tương ứng.
- **Xử lý đơn đặt hàng trong kho (Order fulfillment in warehouses):**
Trong quản lý logistics và kho lưu trữ, hàng đợi có thể được sử dụng để quản lý quy trình thực hiện đơn đặt hàng. Đơn đặt hàng được đặt vào hàng đợi để lựa chọn, đóng gói và giao hàng để đảm bảo xử lý hiệu quả.

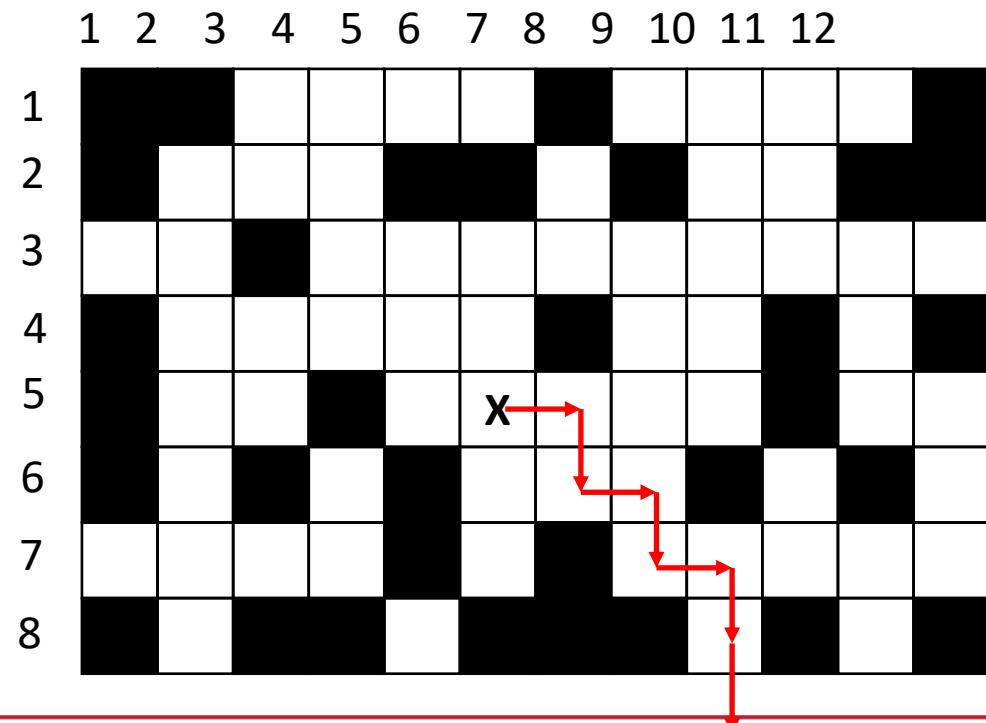
Nội dung

- Ngăn xếp
- Bài tập: Kiểm tra tính cân xứng của dãy ngoặc
- Hàng đợi
- Bài tập: Tìm đường đi nhanh nhất thoát khỏi mê cung



Đề bài

- **Đề bài:** Một mê cung hình chữ nhật được biểu diễn bởi 0-1 ma trận NxM trong đó $A[i,j] = 1$ thể hiện ô (i,j) là tường gạch và $A[i,j] = 0$ thể hiện ô (i,j) là ô trống, có thể di chuyển vào. Từ 1 ô trống, ta có thể di chuyển sang 1 trong 4 ô lân cận (lên trên, xuống dưới, sang trái, sang phải) nếu ô đó là ô trống.
 - Xuất phát từ 1 ô trống trong mê cung, hãy tìm đường ngắn nhất thoát ra khỏi mê cung



7 bước

Đề bài

- **Dữ liệu đầu vào:**

- Dòng 1: ghi 4 số nguyên dương n, m, r, c trong đó n và m tương ứng là số hàng và cột của ma trận A ($1 \leq n, m \leq 999$) và r, c tương ứng là chỉ số hàng, cột của ô xuất phát.
- Dòng $i+1$ ($i=1, \dots, n$): ghi dòng thứ i của ma trận A

- **Kết quả đầu ra:**

- Ghi ra số bước cần di chuyển ngắn nhất để thoát ra khỏi mê cung, hoặc ghi giá trị -1 nếu không tìm thấy đường đi nào thoát ra mê cung.



Đề bài

- Dữ liệu đầu vào:

- Dòng 1: ghi 4 số nguyên dương n, m, r, c trong đó n và m tương ứng là số hàng và cột của ma trận A ($1 \leq n,m \leq 999$) và r, c tương ứng là chỉ số hàng, cột của ô xuất phát.
- Dòng $i+1$ ($i=1,\dots,n$): ghi dòng thứ i của ma trận A

- Kết quả đầu ra:

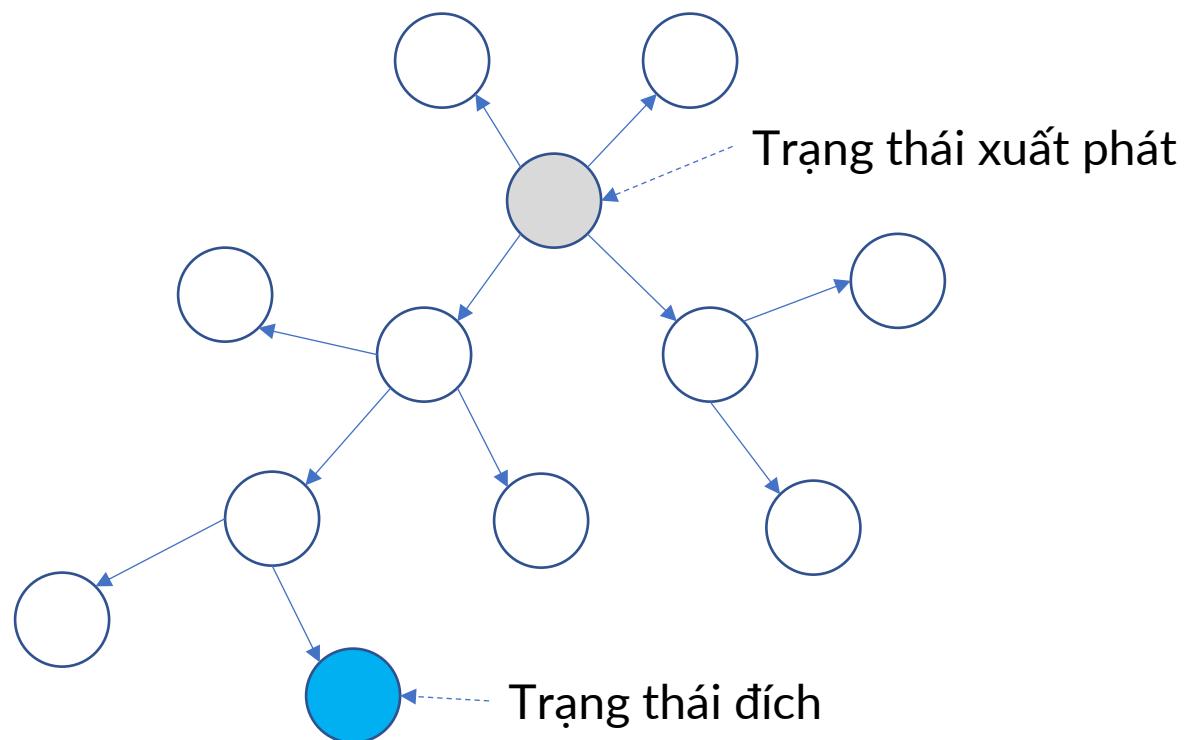
- Ghi giá số bước cần di chuyển ngắn nhất để thoát ra khỏi mê cung, hoặc ghi giá trị -1 nếu không tìm thấy đường đi nào thoát ra mê cung.

stdin	stdout
8 12 5 6 1 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 1 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 1 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 1 1 1 0 1 0 1	7



Thuật toán

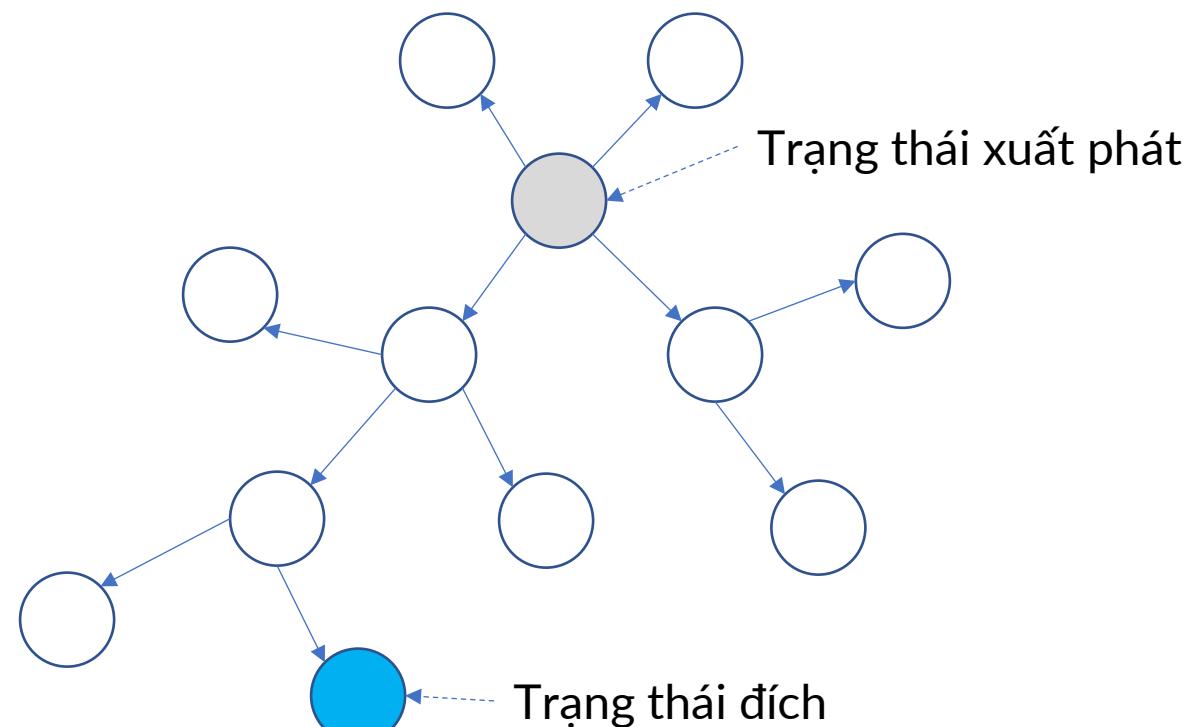
- Thuật toán tìm kiếm theo chiều rộng (loang) tìm đường đi ngắn nhất trên mô hình chuyển trạng thái:
 - Trạng thái đầu
 - Trạng thái đích
 - Mọi trạng thái s sẽ có 1 tập $N(s)$ các trạng thái lân cận



Thuật toán

- Thuật toán tìm kiếm theo chiều rộng (loang) tìm đường đi ngắn nhất trên mô hình chuyển trạng thái:

```
findPath(s0, N){  
    Khởi tạo hàng đợi Queue  
    Queue.PUSH(s0);  
    visited[s0] = true;  
    while Queue not empty do{  
        s = Queue.POP();  
        for x ∈ N(s) do{  
            if not visited[x] and check(x) then{  
                if target(x) then  
                    return x;  
                else{  
                    Queue.PUSH(x);  
                    visited[x] = true;  
                }  
            }  
        }  
    }  
}
```

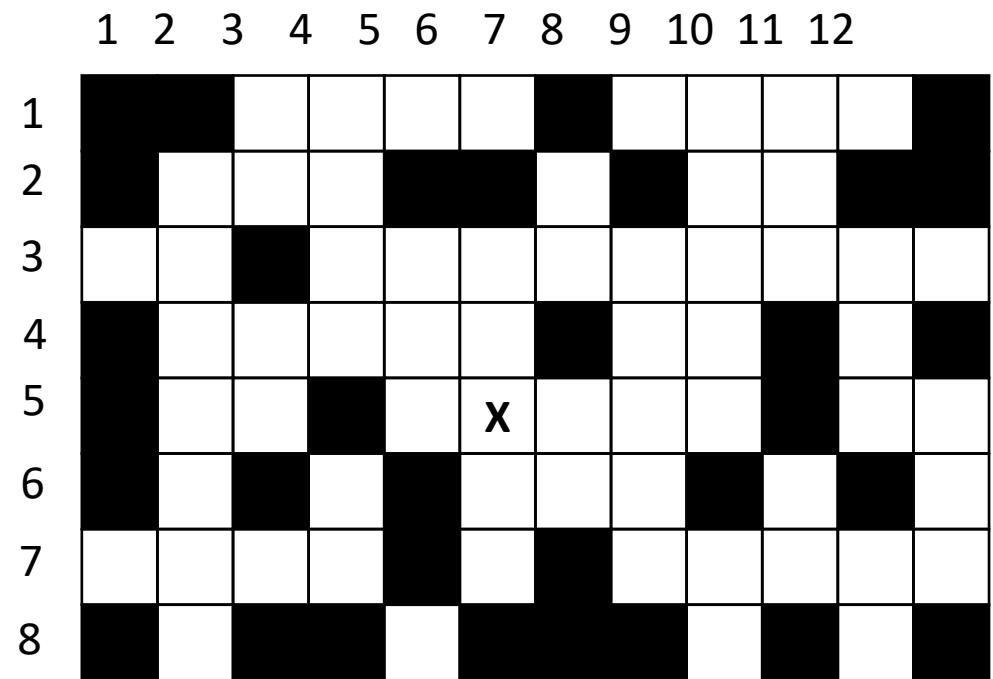


Minh họa

Khởi tạo hàng đợi Q rỗng

Đưa trạng thái (5,6) vào Q

	(5,6)	
--	-------	--



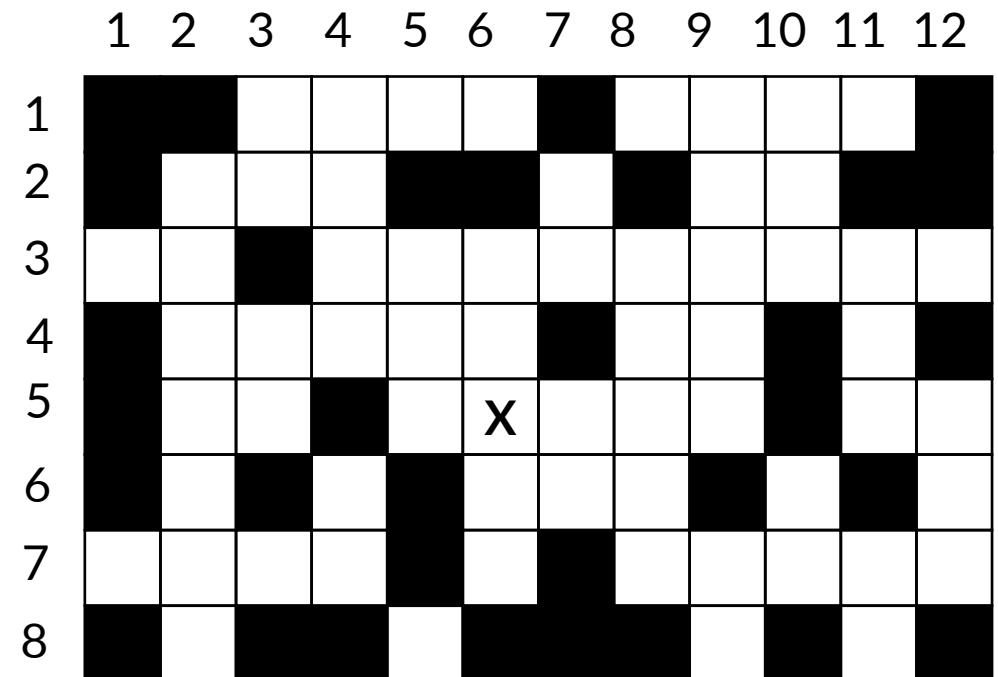
Minh họa

- Minh họa

Lấy (5,6) ra khỏi Q

Đưa trạng thái (5,7), (5, 5), (4, 6), (6,6) vào Q

	(5,6)	(5,7)	(5,5)	(4,6)	(6,6)							
--	-------	-------	-------	-------	-------	--	--	--	--	--	--	--



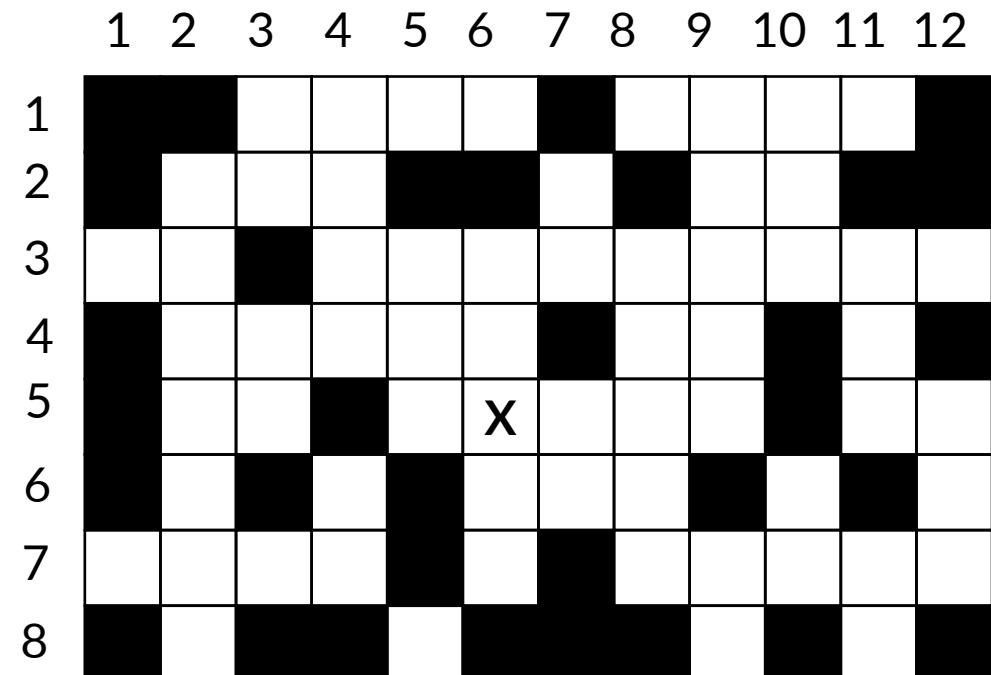
Minh họa

- Minh họa

Lấy (5,7) ra khỏi Q

Đưa trạng thái (6,7), (5, 8) vào Q

	(5,7)	(5,5)	(4,6)	(6,6)	(6,7)	(5,8)						
--	-------	-------	-------	-------	-------	-------	--	--	--	--	--	--

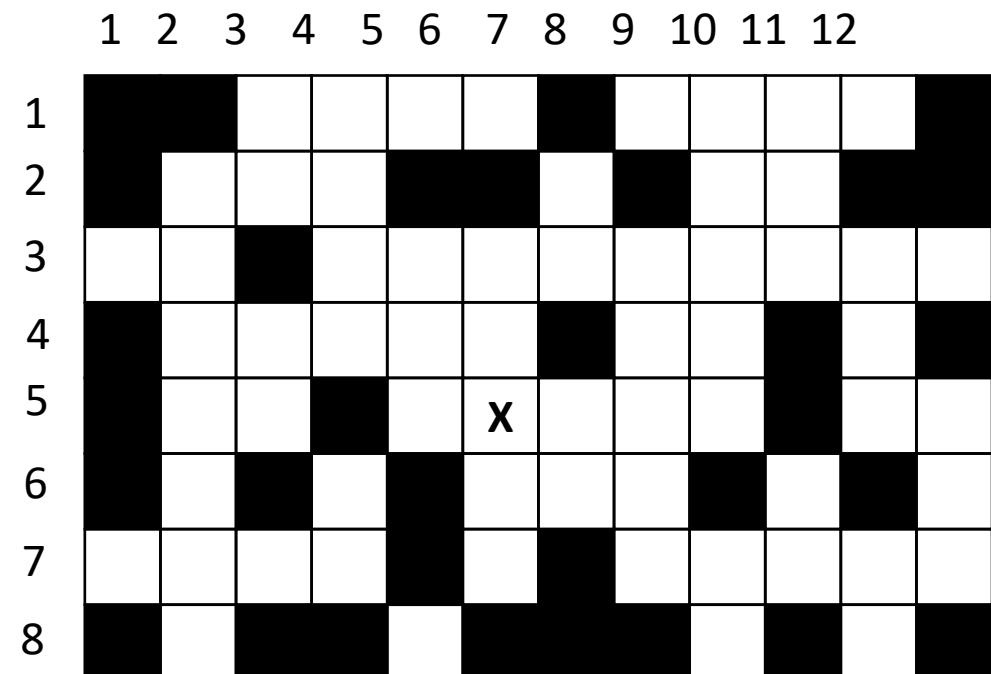
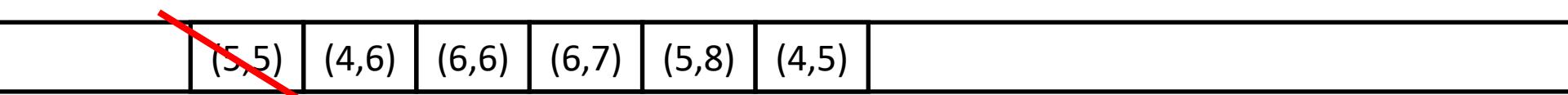


Minh họa

▪ Minh họa

Lấy (5,5) ra khỏi Q

Đưa trạng thái (4,5) vào Q

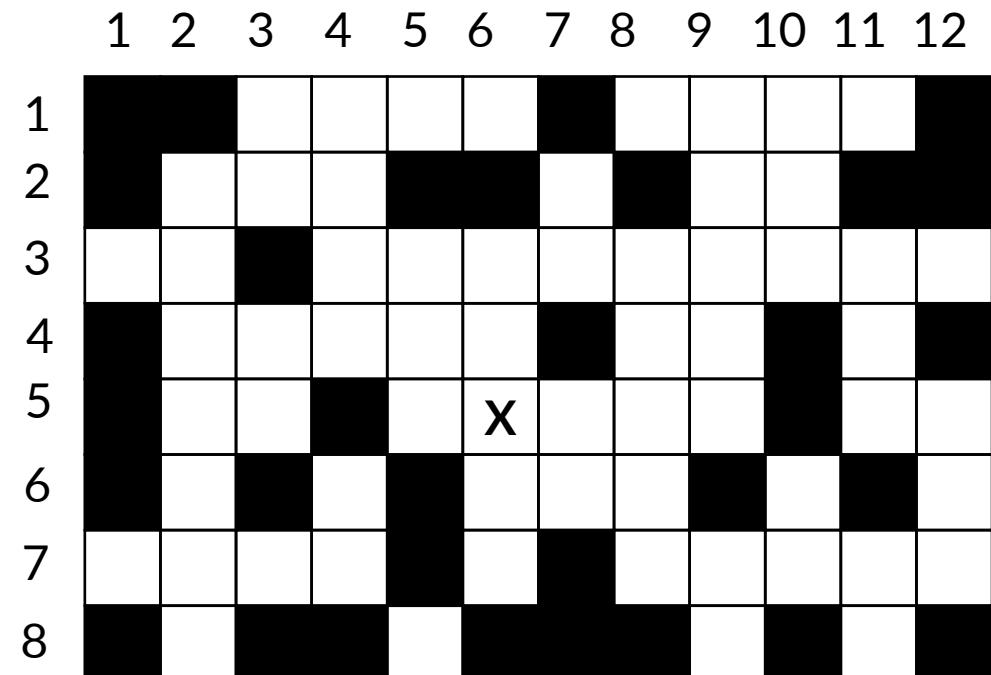
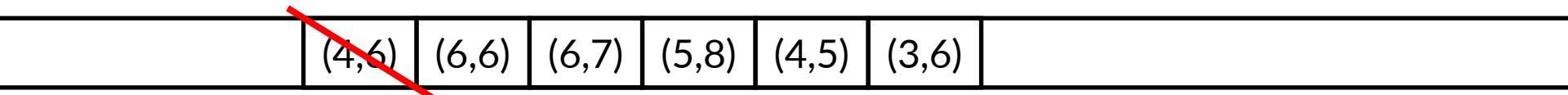


Minh họa

- Minh họa

Lấy (4,6) ra khỏi Q

Đưa trạng thái (3,6) vào Q

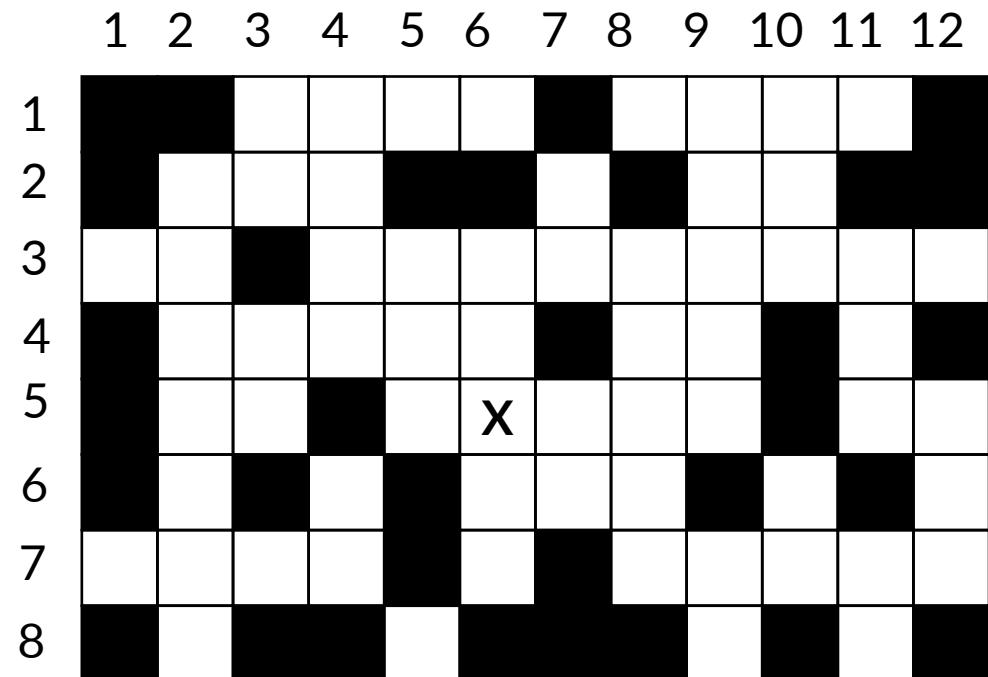


Minh họa

Lấy (6,6) ra khỏi Q

Đưa trạng thái (7,6) vào Q

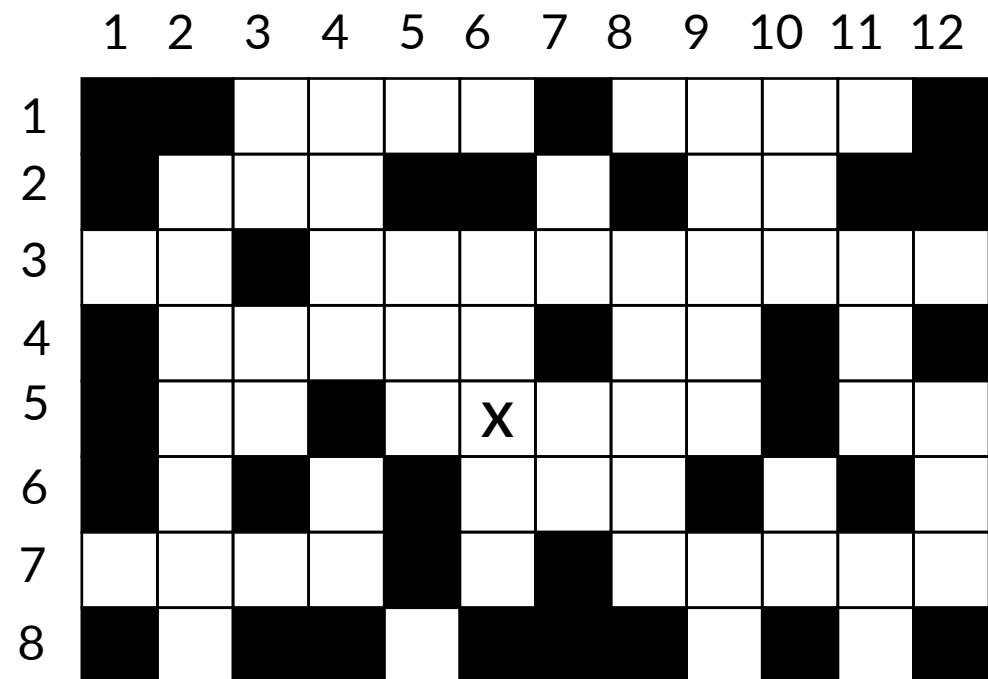
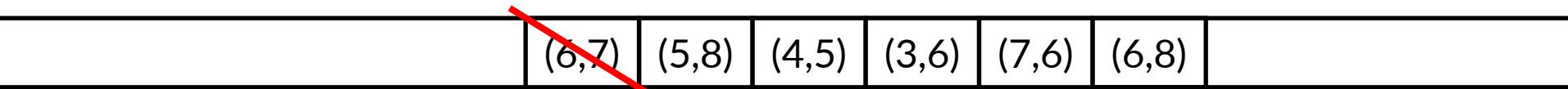
	(6,6)	(6,7)	(5,8)	(4,5)	(3,6)	(7,6)	
--	-------	-------	-------	-------	-------	-------	--



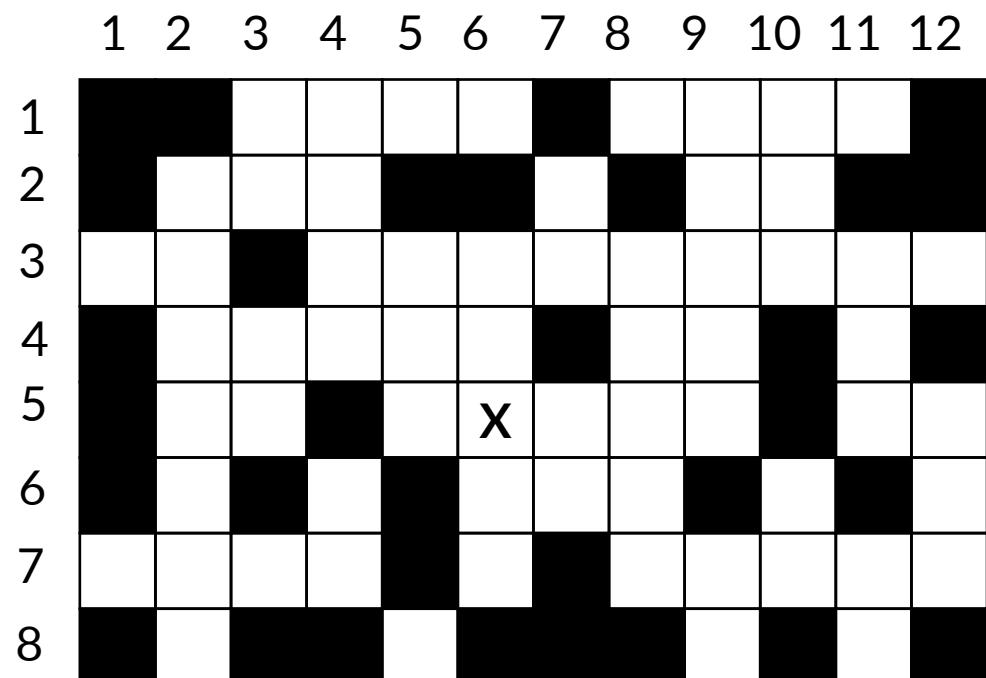
Minh họa

Lấy (6,7) ra khỏi Q

Đưa trạng thái (6,8) vào Q



Minh họa

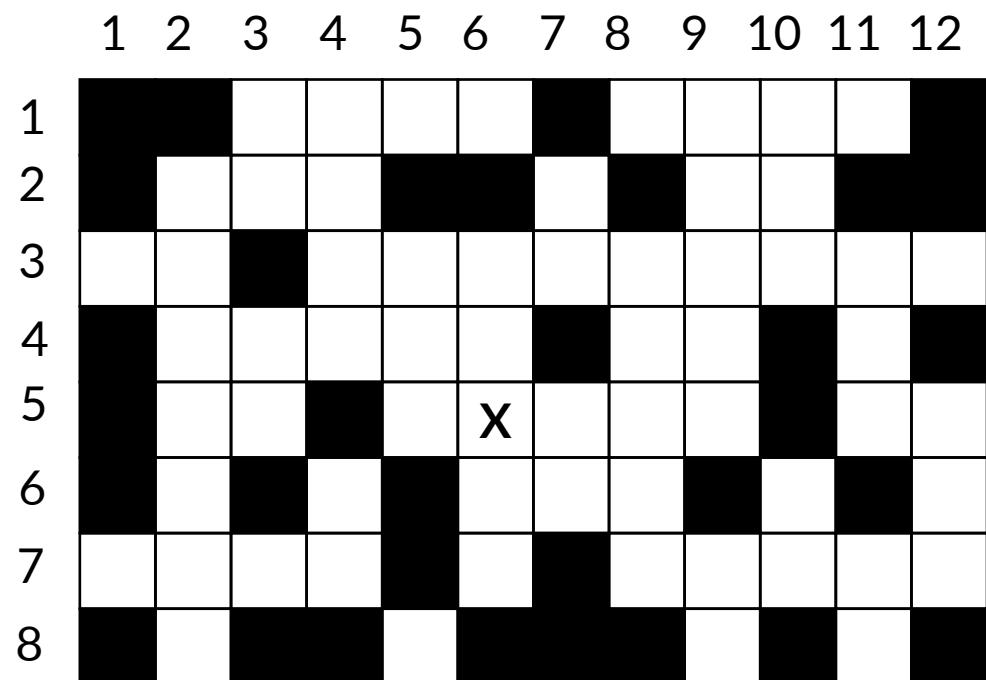


Lấy (5,8) ra khỏi Q

Đưa trạng thái (5,9) và (4,8) vào Q



Minh họa



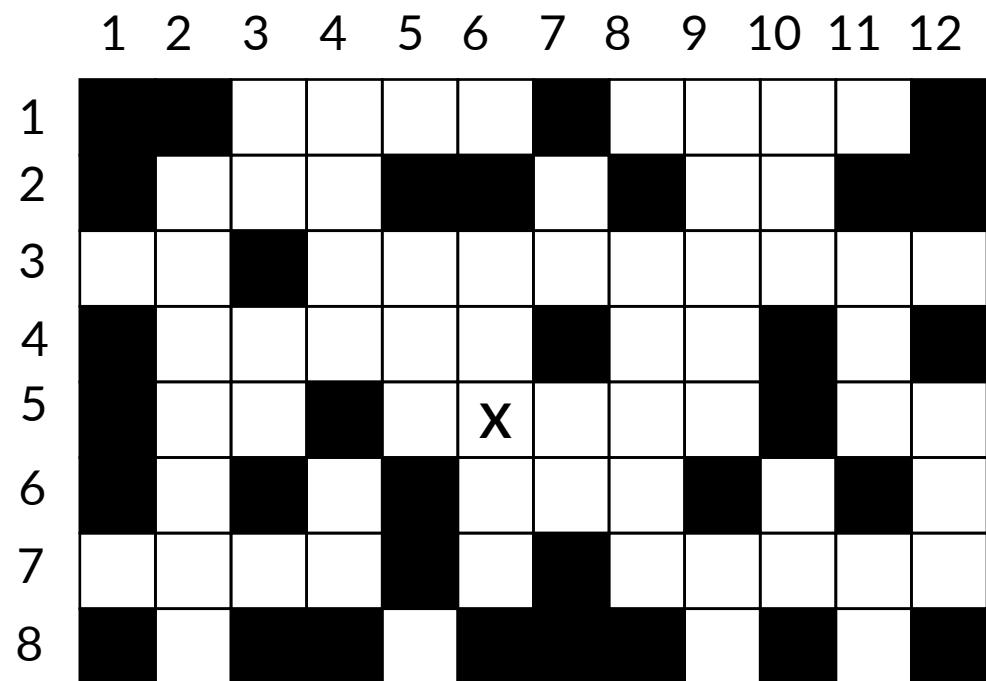
Lấy (4,5) ra khỏi Q

Đưa trạng thái (4,4) và (3,5) vào Q

	(4,5)	(3,6)	(7,6)	(6,8)	(5,9)	(4,8)	(4,4)	(3,5)				
--	-------	-------	-------	-------	-------	-------	-------	-------	--	--	--	--

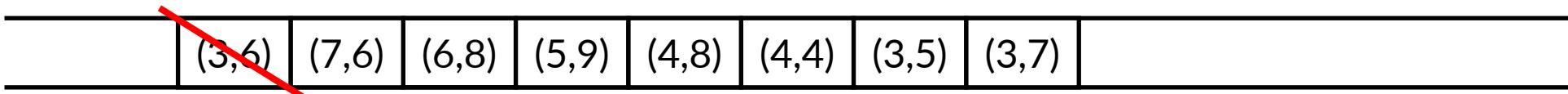


Minh họa

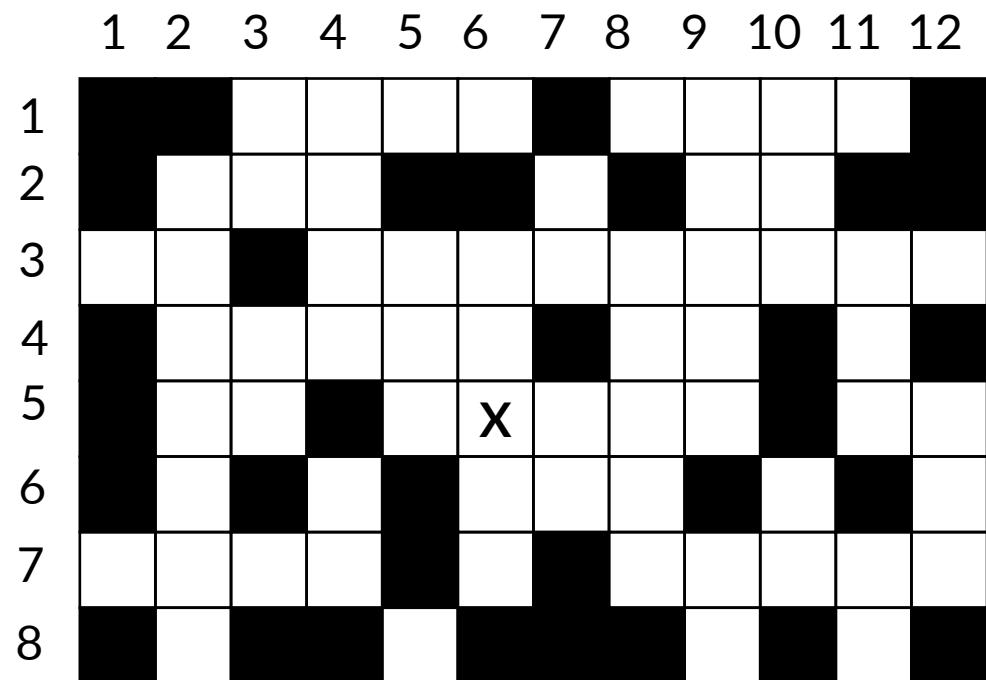


Lấy (3,6) ra khỏi Q

Đưa trạng thái (3,7) vào Q

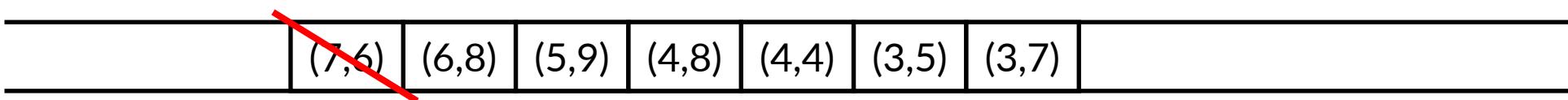


Minh họa



Lấy (7,6) ra khỏi Q

Không đưa được trạng thái mới nào vào Q

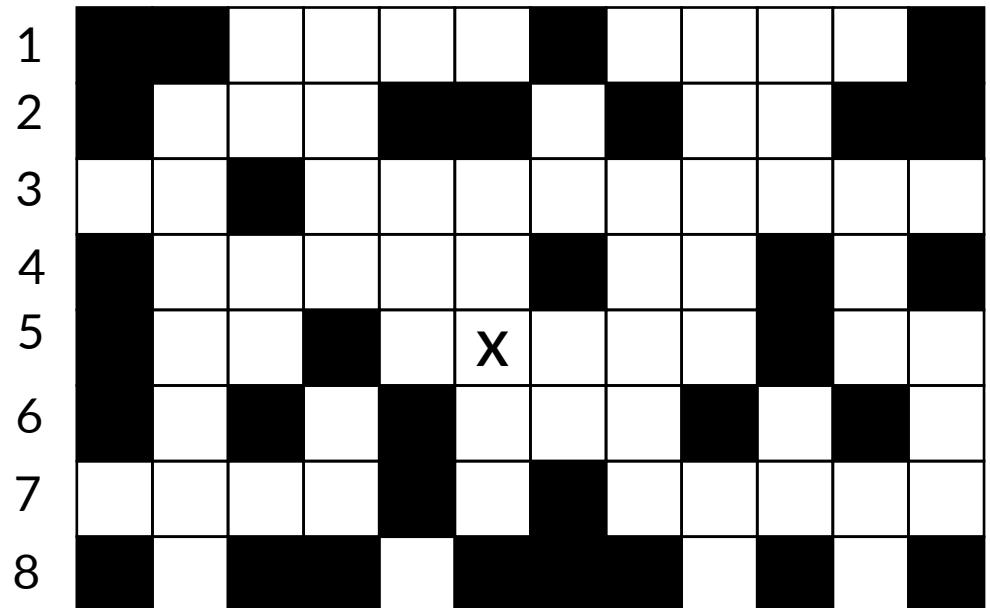


Minh họa

Lấy (6,8) ra khỏi Q

Đưa được trạng thái (7,8) vào Q

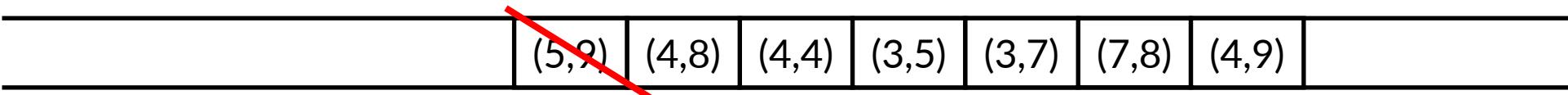
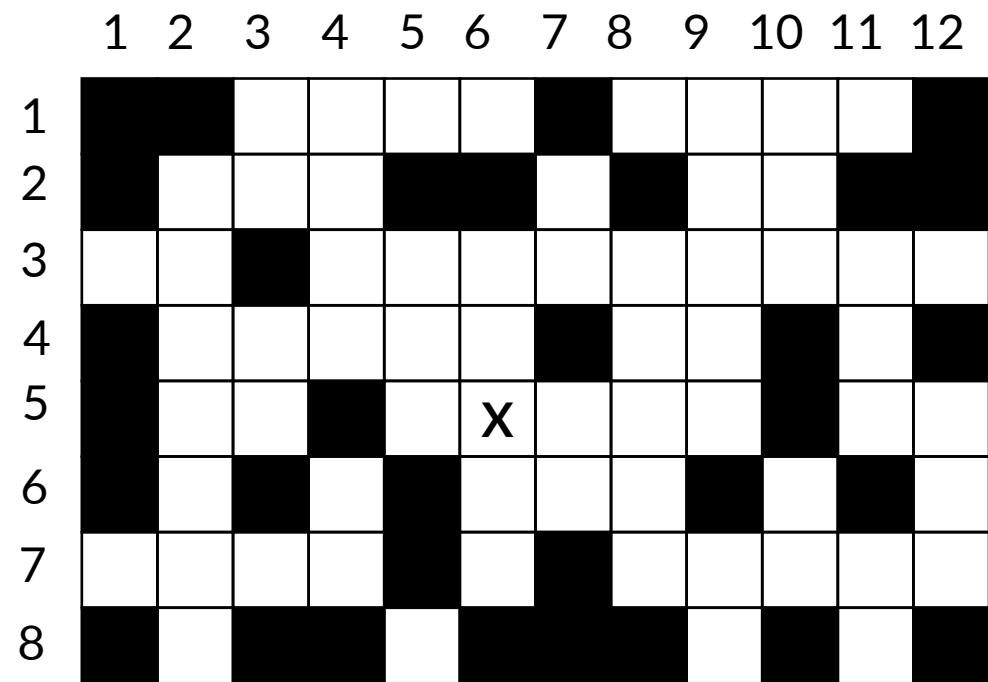
	(6,8)	(5,9)	(4,8)	(4,4)	(3,5)	(3,7)	(7,8)	
--	-------	-------	-------	-------	-------	-------	-------	--



Minh họa

Lấy $(5,9)$ ra khỏi Q

Đưa được trạng thái (4,9) vào Q

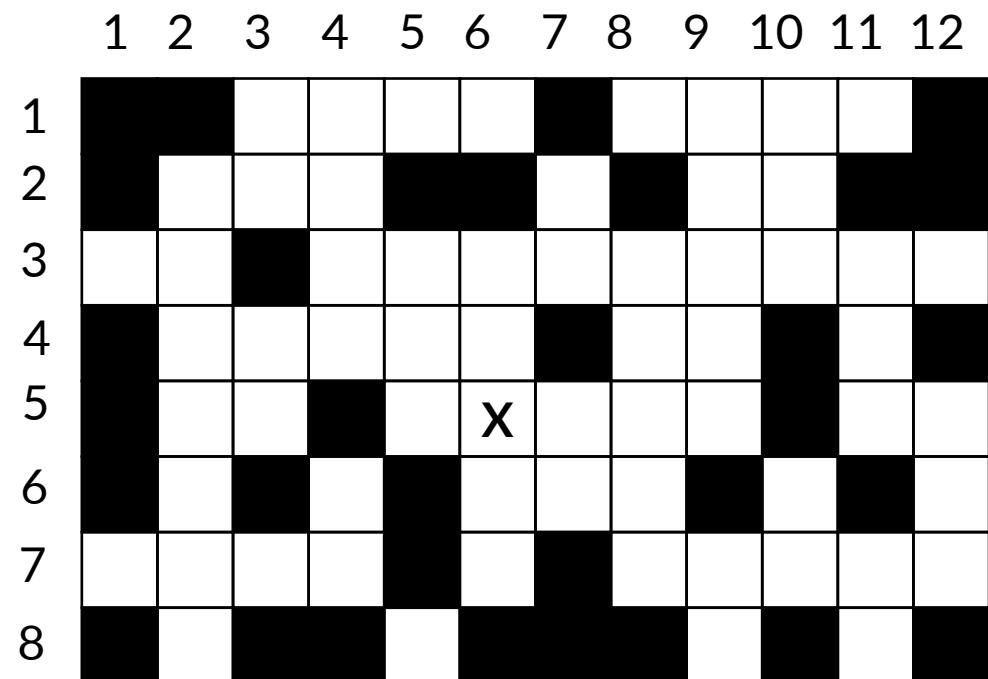


Minh họa

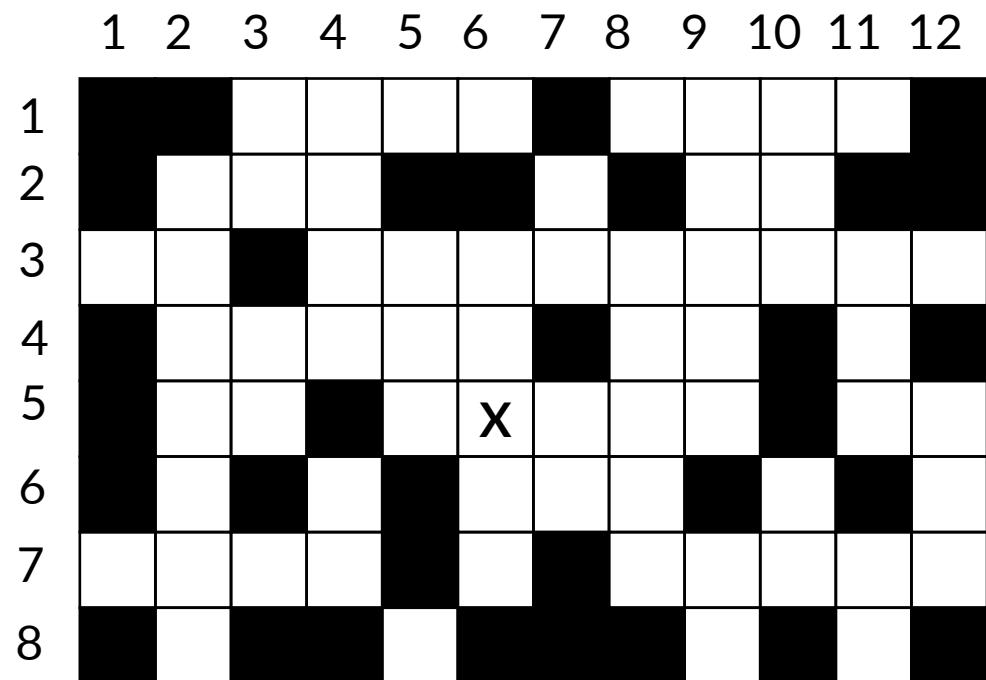
Lấy (4,8) ra khỏi Q

Đưa được trạng thái (3,8) vào Q

	(4,8)	(4,4)	(3,5)	(3,7)	(7,8)	(4,9)	(3,8)	
--	-------	-------	-------	-------	-------	-------	-------	--



Minh họa



Lấy (4,4) ra khỏi Q

Đưa được trạng thái (4,3), (3,4) vào Q

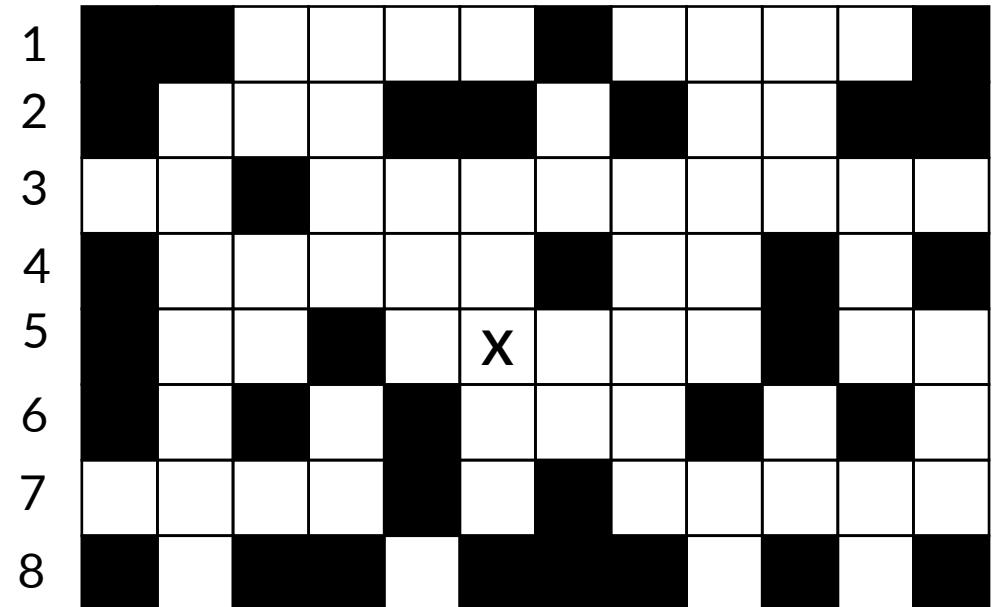
	(4,4)	(3,5)	(3,7)	(7,8)	(4,9)	(3,8)	(4,3)	(3,4)	
--	-------	-------	-------	-------	-------	-------	-------	-------	--

Minh họa

Lấy (3,5) ra khỏi Q

Không đưa được trạng thái mới nào vào Q

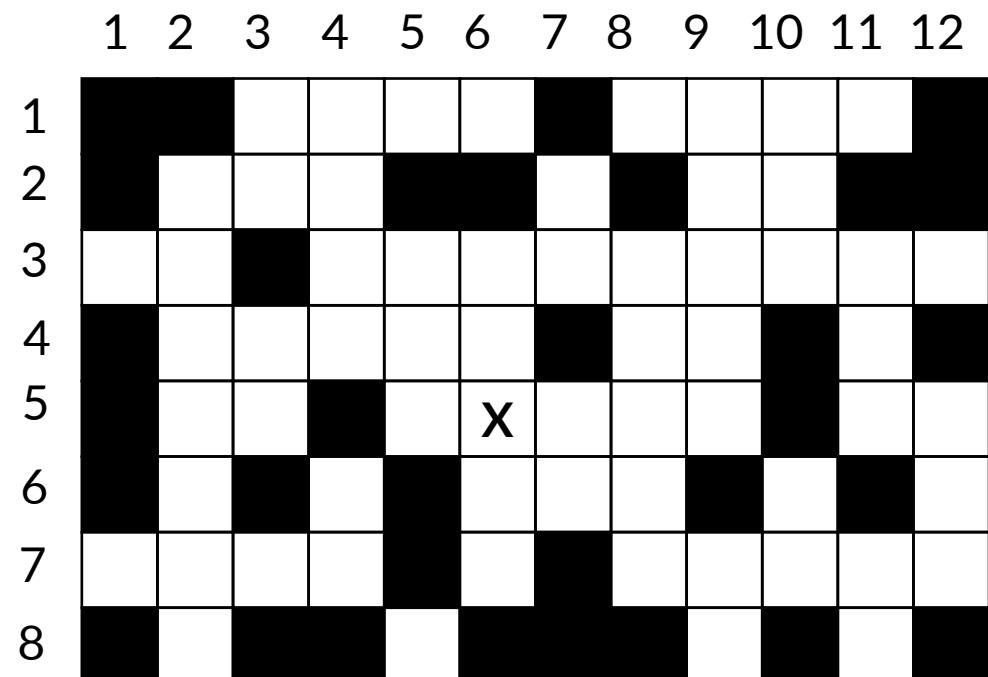
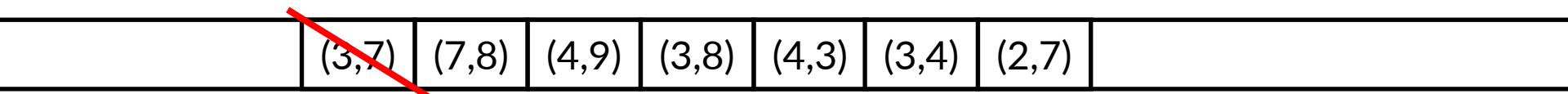
	(3,5)	(3,7)	(7,8)	(4,9)	(3,8)	(4,3)	(3,4)	
--	-------	-------	-------	-------	-------	-------	-------	--



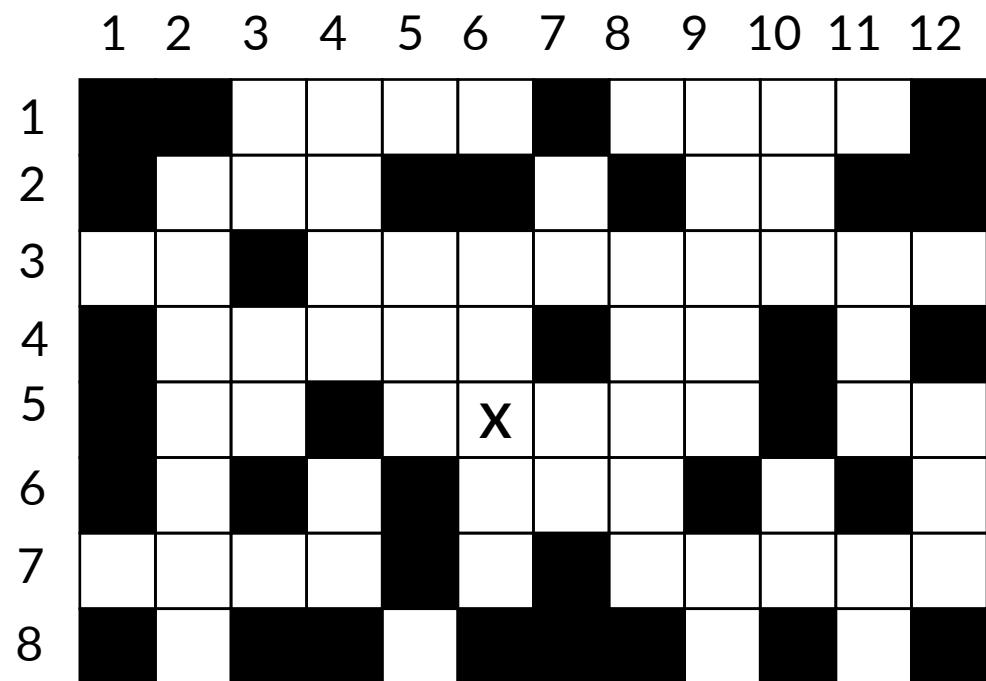
Minh họa

Lấy (3,7) ra khỏi Q

Đưa được trạng thái mới (2,7) vào Q



Minh họa

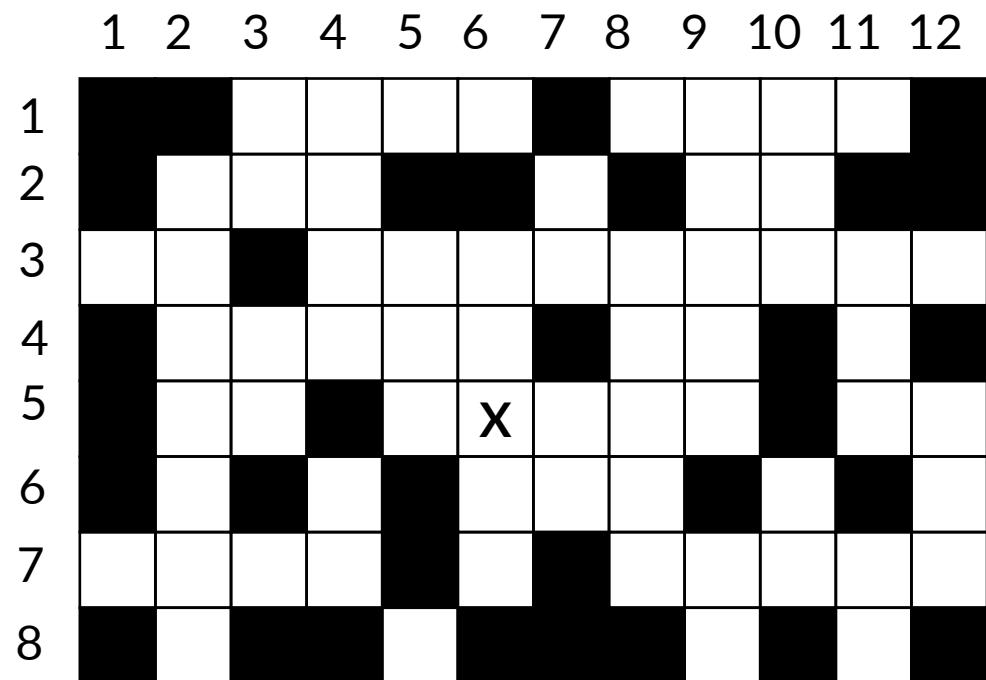


Lấy (7,8) ra khỏi Q

Đưa được trạng thái mới (7,9) vào Q

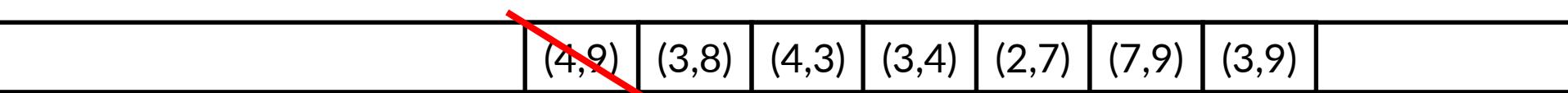
(7,8)

Minh họa

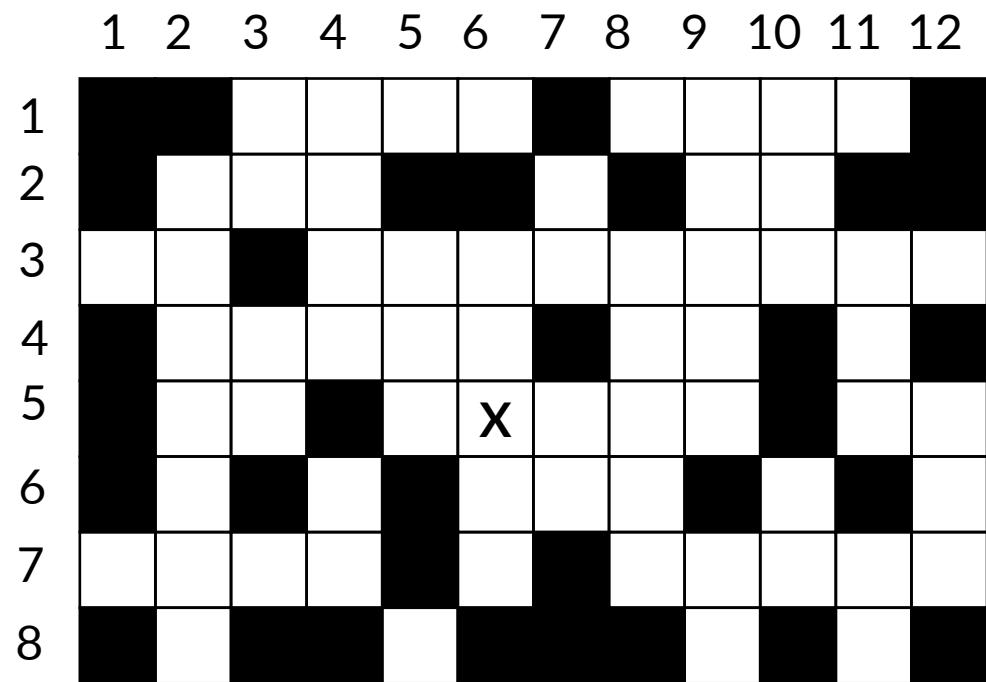


Lấy (4,9) ra khỏi Q

Đưa được trạng thái mới (3,9) vào Q

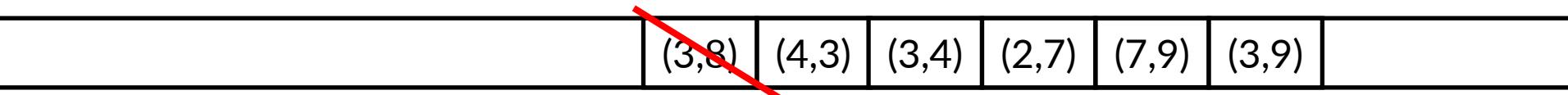


Minh họa



Lấy (3,8) ra khỏi Q

Không đưa được trạng thái mới nào vào Q



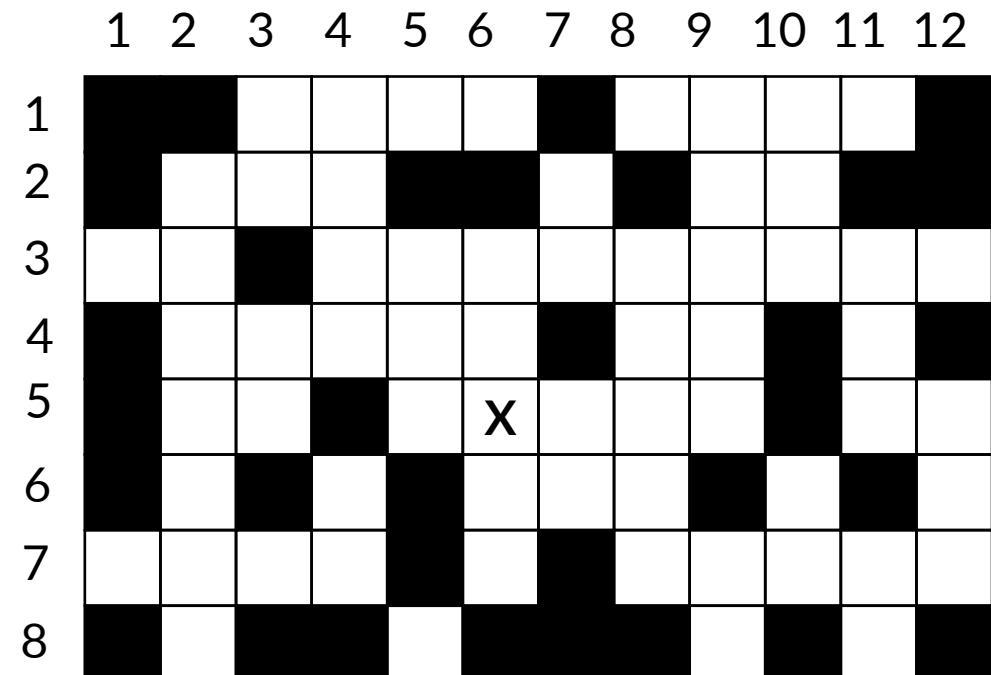
Minh họa

- Minh họa

Lấy (4,3) ra khỏi Q

Đưa được trạng thái mới (4,2), (5,3) vào Q

	(4,3)	(3,4)	(2,7)	(7,9)	(3,9)	(4,2)	(5,3)	
--	-------	-------	-------	-------	-------	-------	-------	--

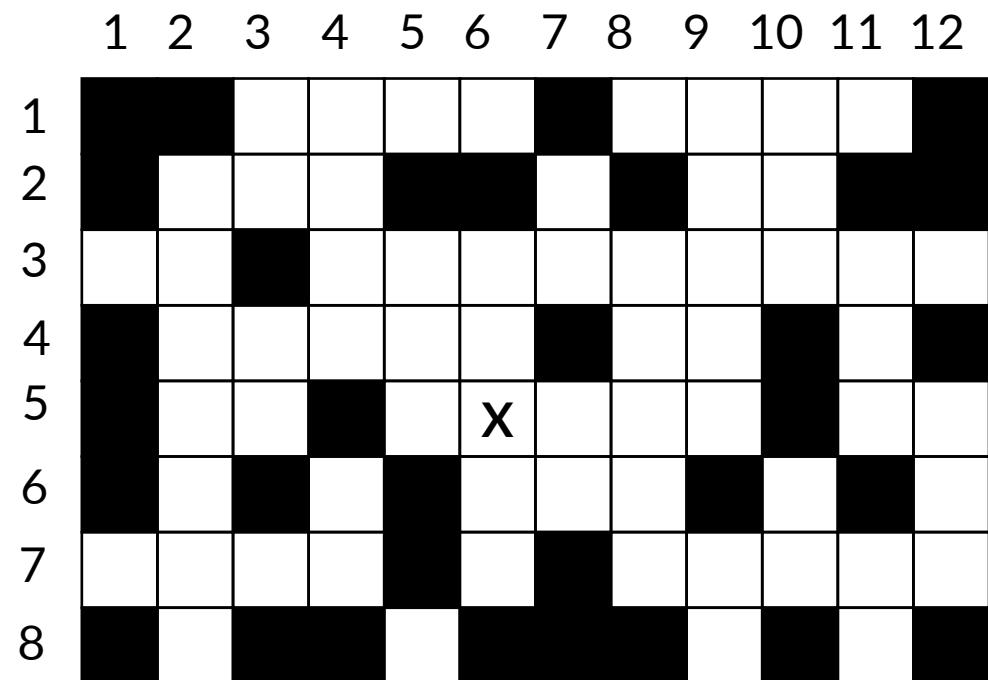


Minh họa

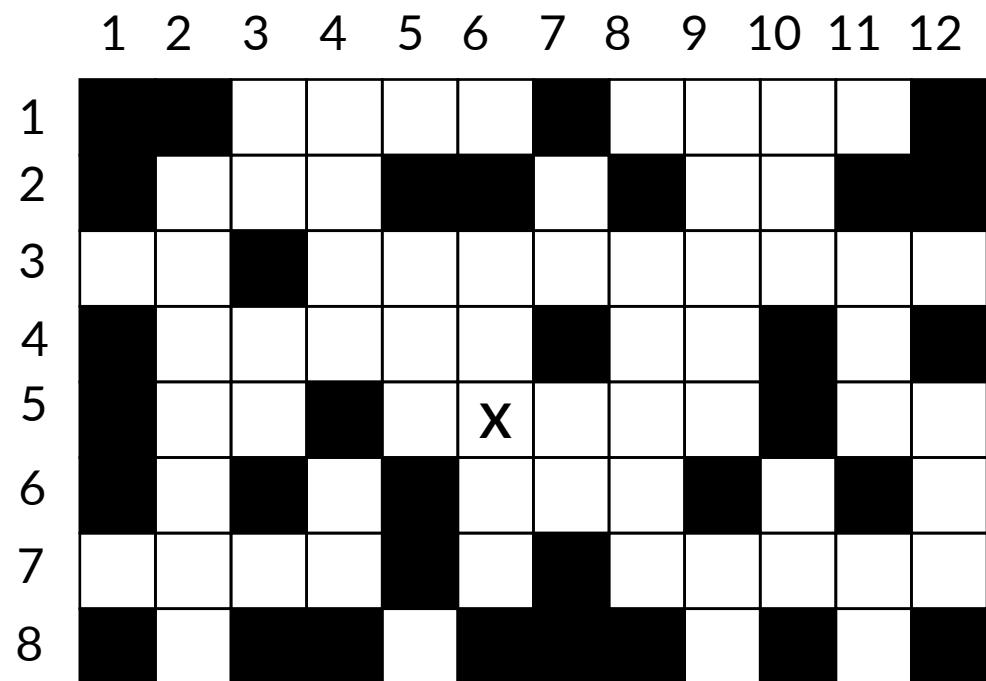
Lấy (3,4) ra khỏi Q

Đưa được trạng thái mới (2,4) vào Q

	(3,4)	(2,7)	(7,9)	(3,9)	(4,2)	(5,3)	(2,4)	
--	-------	-------	-------	-------	-------	-------	-------	--

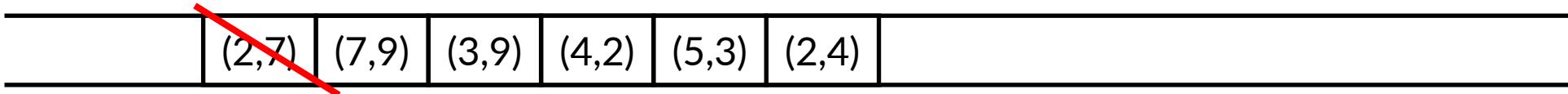


Minh họa

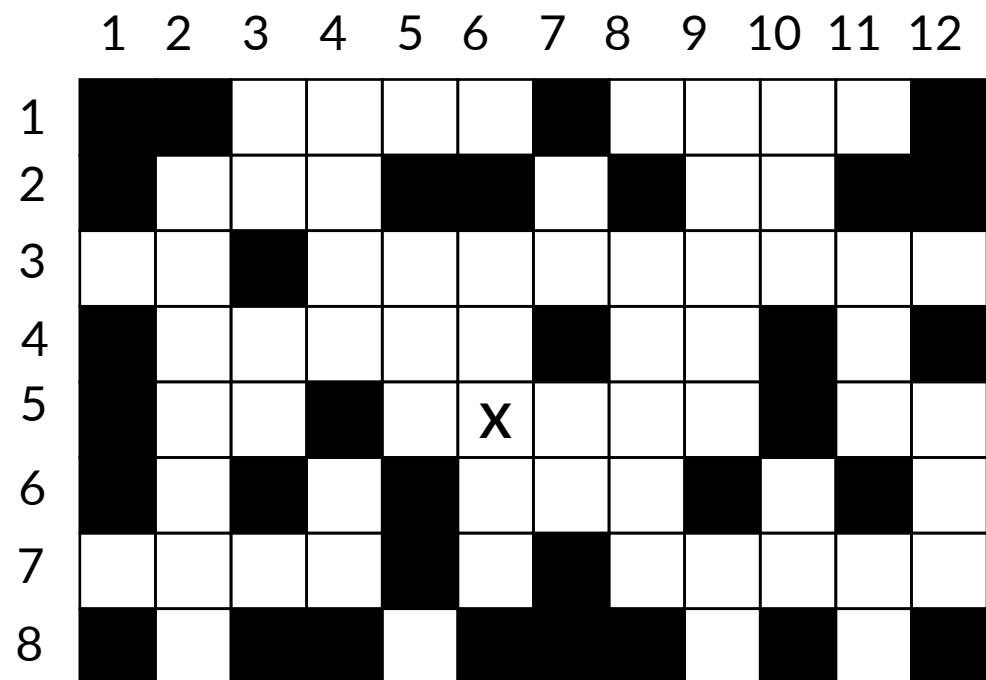


Lấy (2,7) ra khỏi Q

Không đưa được trạng thái mới nào vào Q

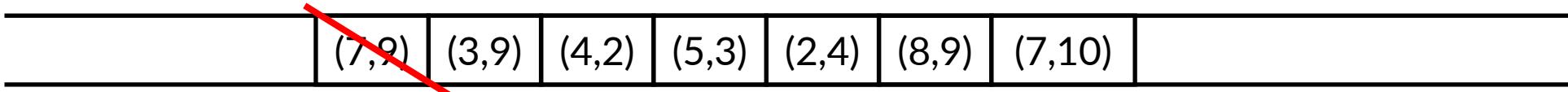


Minh họa

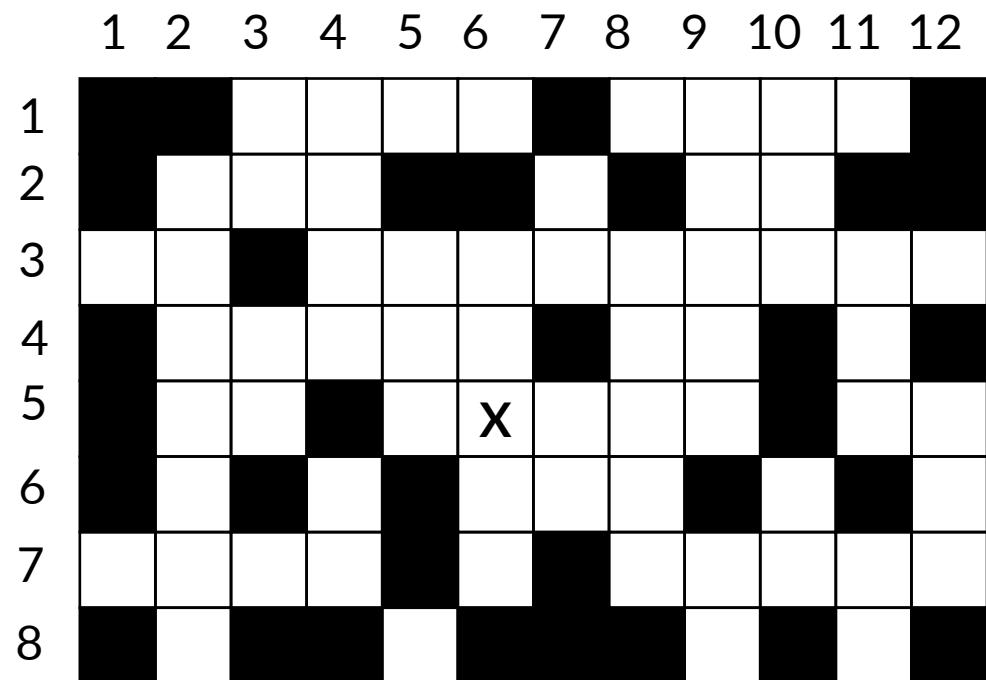


Lấy (7,9) ra khỏi Q

Đưa được trạng thái mới (8,9), (7,10) vào Q

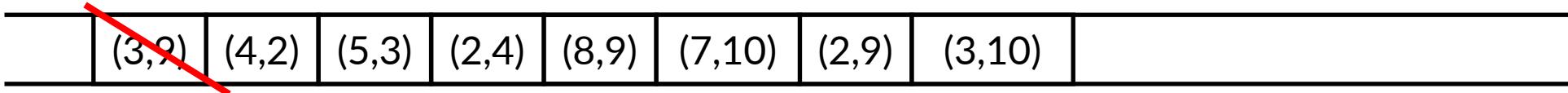


Minh họa

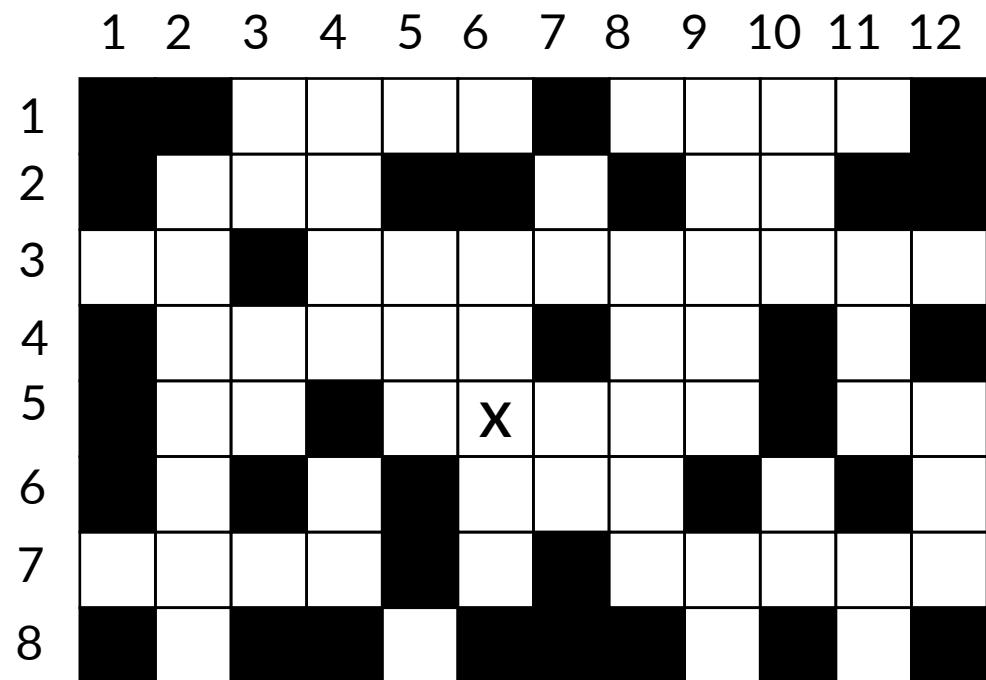


Lấy (3,9) ra khỏi Q

Đưa được trạng thái mới (2,9), (3,10) vào Q



Minh họa



Lấy (4,2) ra khỏi Q

Đưa được trạng thái mới (3,2), (5,2) vào Q

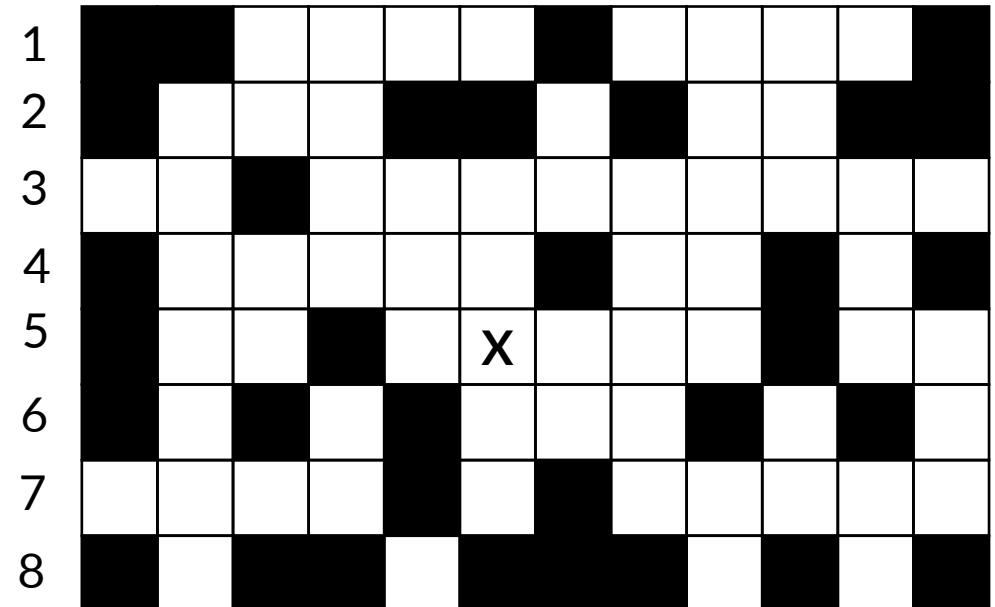
	(4,2)	(5,3)	(2,4)	(8,9)	(7,10)	(2,9)	(3,10)	(3,2)	(5,2)	
--	-------	-------	-------	-------	--------	-------	--------	-------	-------	--

Minh họa

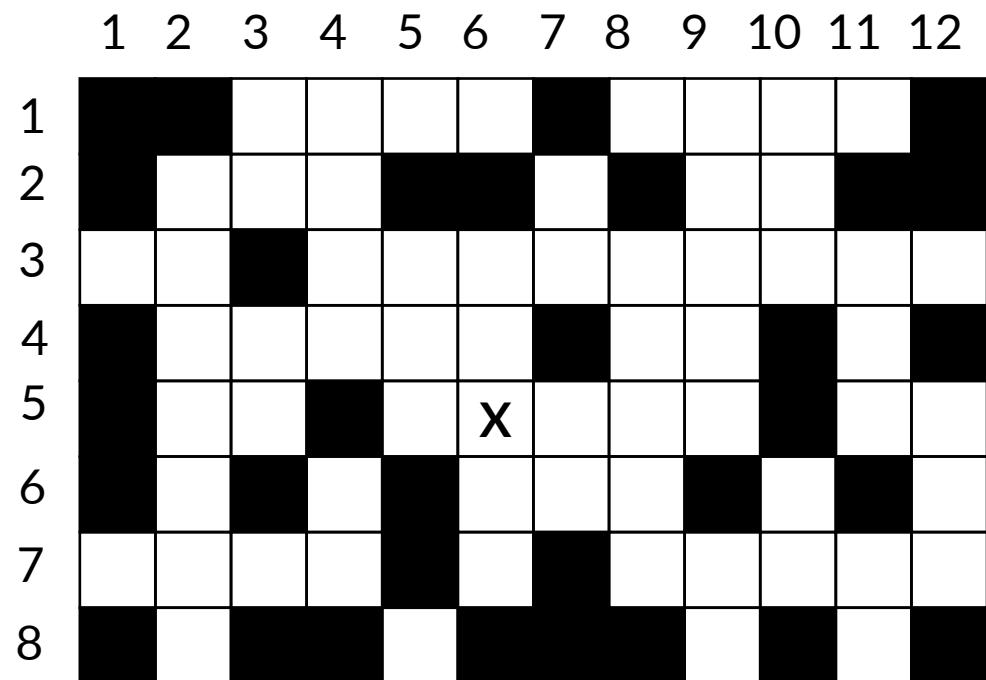
Lấy (5,3) ra khỏi Q

Không đưa được trạng thái mới nào vào Q

	(5,3)	(2,4)	(8,9)	(7,10)	(2,9)	(3,10)	(3,2)	(5,2)	
--	-------	-------	-------	--------	-------	--------	-------	-------	--

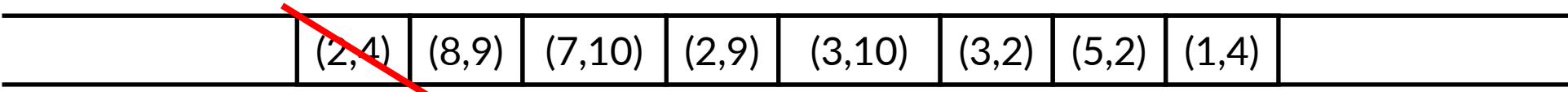


Minh họa



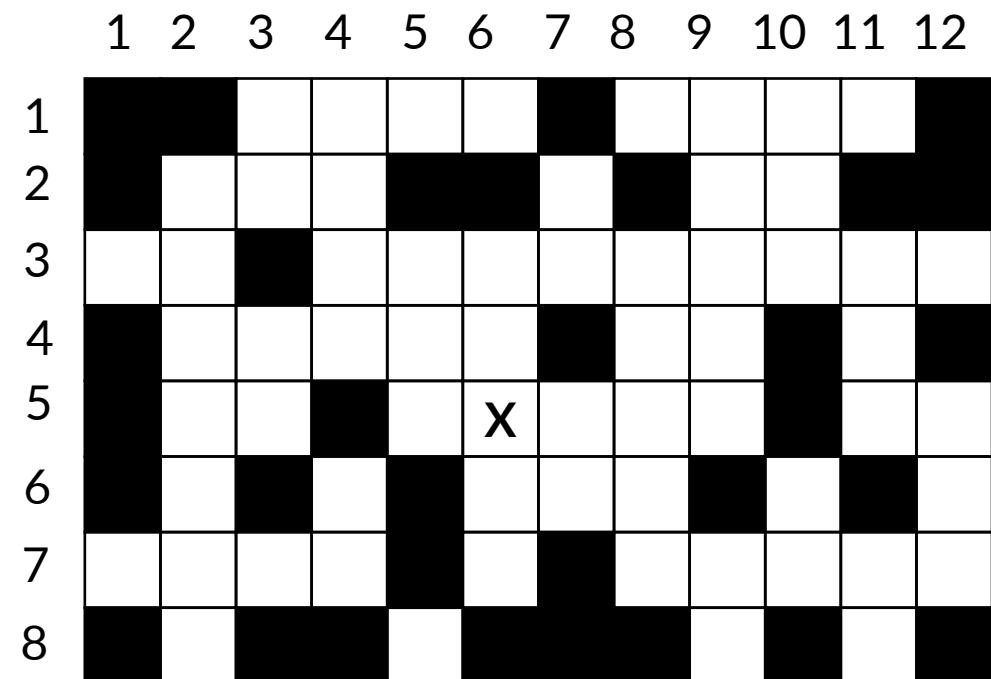
Lấy (2,4) ra khỏi Q

Đưa được trạng thái mới (1,4) vào Q



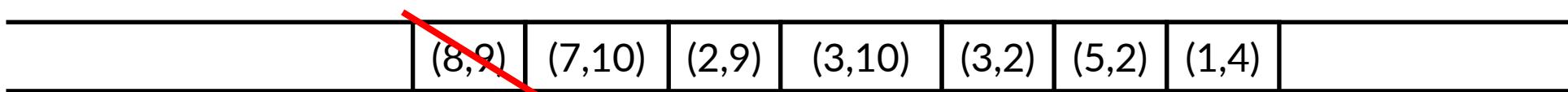
Minh họa

- Minh họa



Lấy (8,9) ra khỏi Q

Sinh ra được trạng thái đích (9,9) ứng với vị trí ngoài mê cung



Cài đặt thuật toán

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node{
    int row;
    int col;
    int step;
    struct Node* next;
}Node;
```



Cài đặt thuật toán

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node{
    int row;
    int col;
    int step;
    struct Node* next;
}Node;
```

```
int n,m;
int A[1000][1000];
int startRow, startCol;
int visited[1000][1000];
Node* first;
Node* last;
int dr[4] = {0,0,1,-1};
int dc[4] = {1,-1,0,0};
```



Cài đặt thuật toán

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node{
    int row;
    int col;
    int step;
    struct Node* next;
}Node;
```

```
int n,m;
int A[1000][1000];
int startRow, startCol;
int visited[1000][1000];
Node* first;
Node* last;
int dr[4] = {0,0,1,-1};
int dc[4] = {1,-1,0,0};

Node* makeNode(int r,int c, int step){
    Node* p = (Node*)malloc(sizeof(Node));
    p->row = r; p->col = c;
    p->step = step; p->next = NULL;
    return p;
}
```



Cài đặt thuật toán

```
int isEmpty(){
    return first == NULL && last == NULL;
}

void push(Node* p){
    if(isEmpty()){
        first = p; last = p; return;
    }
    last->next = p; last = p;
}

Node* pop(){
    if(isEmpty()) return NULL;
    Node* tmp = first; first = first->next;
    if(first == NULL) last = NULL;
    return tmp;
}
```



Cài đặt thuật toán

```
int isEmpty(){
    return first == NULL && last == NULL;
}

void push(Node* p){
    if(isEmpty()){
        first = p; last = p; return;
    }
    last->next = p; last = p;
}

Node* pop(){
    if(isEmpty()) return NULL;
    Node* tmp = first; first = first->next;
    if(first == NULL) last = NULL;
    return tmp;
}
```

```
void input(){
    scanf("%d %d %d %d",&n,&m,&sr,&sc);
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= m; j++)
            scanf("%d",&A[i][j]);
}
```



Cài đặt thuật toán

```
int isEmpty(){  
    return first == NULL && last == NULL;  
}  
  
void push(Node* p){  
    if(isEmpty()){  
        first = p; last = p; return;  
    }  
    last->next = p; last = p;  
}  
  
Node* pop(){  
    if(isEmpty()) return NULL;  
    Node* tmp = first; first = first->next;  
    if(first == NULL) last = NULL;  
    return tmp;  
}
```

```
void input(){  
    scanf("%d %d %d %d",&n,&m,&sr,&sc);  
    for(int i = 1; i <= n; i++)  
        for(int j = 1; j <= m; j++)  
            scanf("%d",&A[i][j]);  
}
```

```
int targetState(Node* s){  
    return (s->row < 1 ||  
            s->row > n || s->col < 1  
            || s->col > m);  
}
```



Cài đặt thuật toán

```
void init(){  
    first = NULL; last = NULL;  
    for(int i = 0; i <= 1000; i++)  
        for(int j = 0; j <= 1000; j++)  
            visited[i][j] = 0;  
    Node* startState = makeNode(sr,sc,0);  
    push(startState); visited[sr][sc] = 1;  
}
```



Cài đặt thuật toán

```
void init(){
    first = NULL; last = NULL;
    for(int i = 0; i <= 1000; i++)
        for(int j = 0; j <= 1000; j++)
            visited[i][j] = 0;
    Node* startState = makeNode(sr,sc,0);
    push(startState); visited[sr][sc] = 1;
}
```

```
int solve(){
    init();
    while(!isEmpty()){
        Node* s = pop();
        for(int k = 0; k < 4; k++){
            int nr = s->row + dr[k]; int nc = s->col + dc[k];
            if(visited[nr][nc]==0&& A[nr][nc]==0){
                Node* ns = makeNode(nr, nc, s->step + 1);
                push(ns); visited[nr][nc] = 1;
                if(targetState(ns)) return ns->step;
            }
        }
    }
    return -1;
}
```



Cài đặt thuật toán

```
void init(){

    first = NULL; last = NULL;

    for(int i = 0; i <= 1000; i++)
        for(int j = 0; j <= 1000; j++)
            visited[i][j] = 0;

    Node* startState = makeNode(sr,sc,0);
    push(startState); visited[sr][sc] = 1;
}
```

```
int main(){

    input();
    int res = solve();
    printf("%d",res);
    return 0;
}
```

```
int solve(){

    init();

    while(!isEmpty()){

        Node* s = pop();

        for(int k = 0; k < 4; k++){
            int nr = s->row + dr[k];int nc= s->col + dc[k];
            if(visited[nr][nc]==0&& A[nr][nc]==0){

                Node* ns = makeNode(nr, nc, s->step + 1);
                push(ns);visited[nr][nc] = 1;
                if(targetState(ns)) return ns->step;
            }
        }
    }
    return -1;
}
```

A large, faint watermark of the HUST logo is visible across the entire slide, consisting of a grid of red dots.

HUST

THANK YOU !



HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN



ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

Cây: định nghĩa và các khái niệm chung, cây nhị phân

ONE LOVE. ONE FUTURE.

MỤC TIÊU

Sau bài học này, người học có thể:

1. Hiểu được khái niệm **cấu trúc dữ liệu cây** và các khái niệm liên quan.
2. Cài đặt được cấu trúc dữ liệu cây.

NỘI DUNG TIẾP THEO

1. Tìm kiếm tuần tự

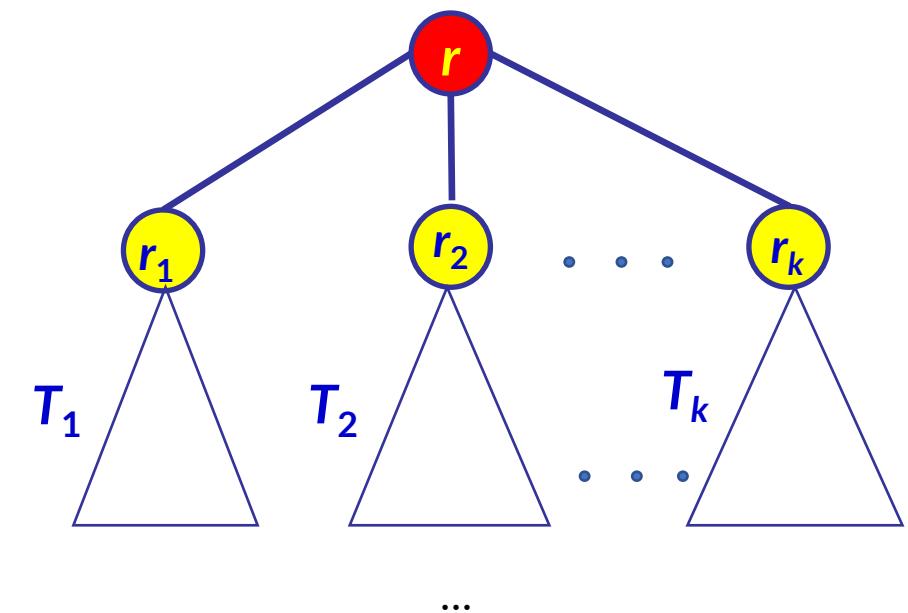
2. Cây nhị phân

1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.1. Định nghĩa cây
 - Cây bao gồm các nút, có một nút đặc biệt được gọi là gốc (root) và các cạnh nối các nút

Định nghĩa cây:

- **Bước cơ sở:** Một nút r là cây và r được gọi là gốc của cây này
- **Bước đệ quy:** Giả sử T_1, T_2, \dots, T_k là các cây với gốc là r_1, r_2, \dots, r_k
- Xây dựng cây mới bằng cách đặt r làm cha của các nút r_1, r_2, \dots, r_k
- Trong cây này r là gốc và T_1, T_2, \dots, T_k là các cây con của gốc r . Các nút r_1, r_2, \dots, r_k được gọi là con của nút r

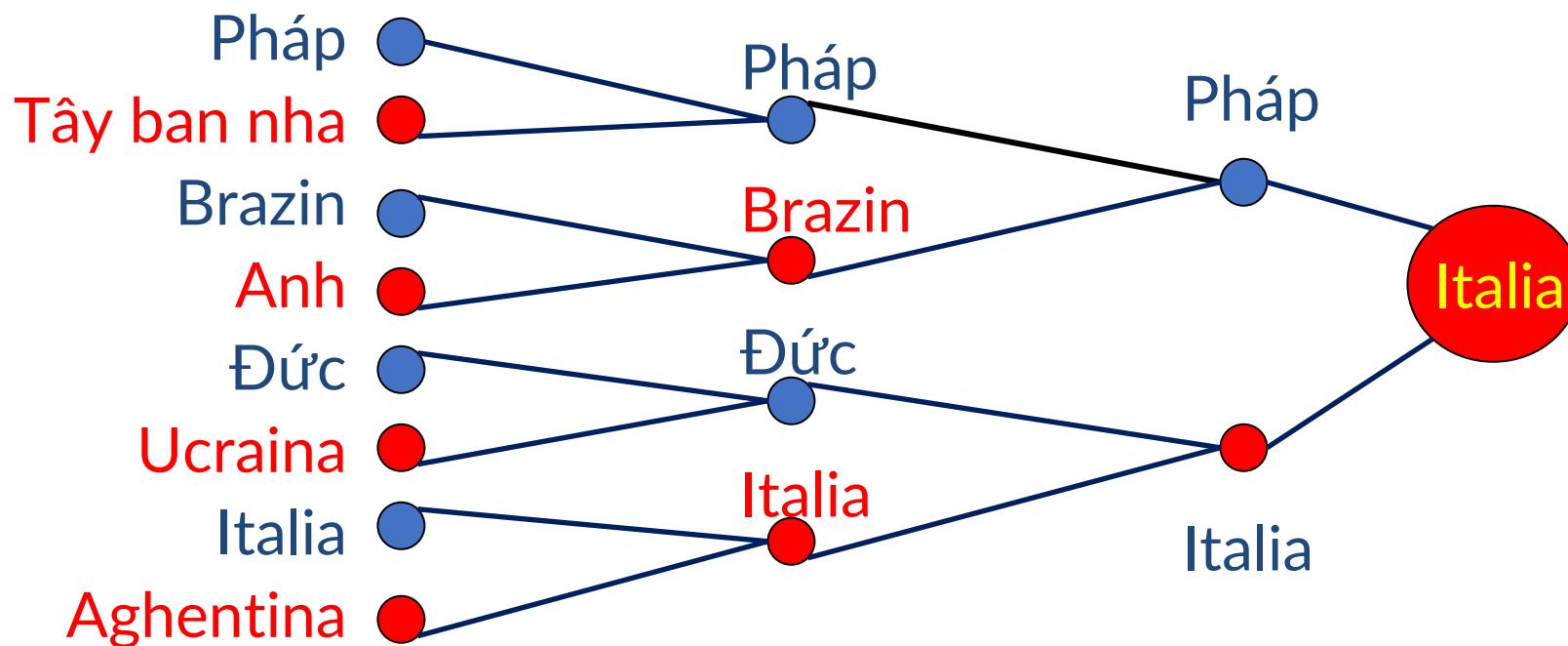


Định nghĩa cây

Chú ý: Cây rỗng (null tree) là cây không có nút nào cả

1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

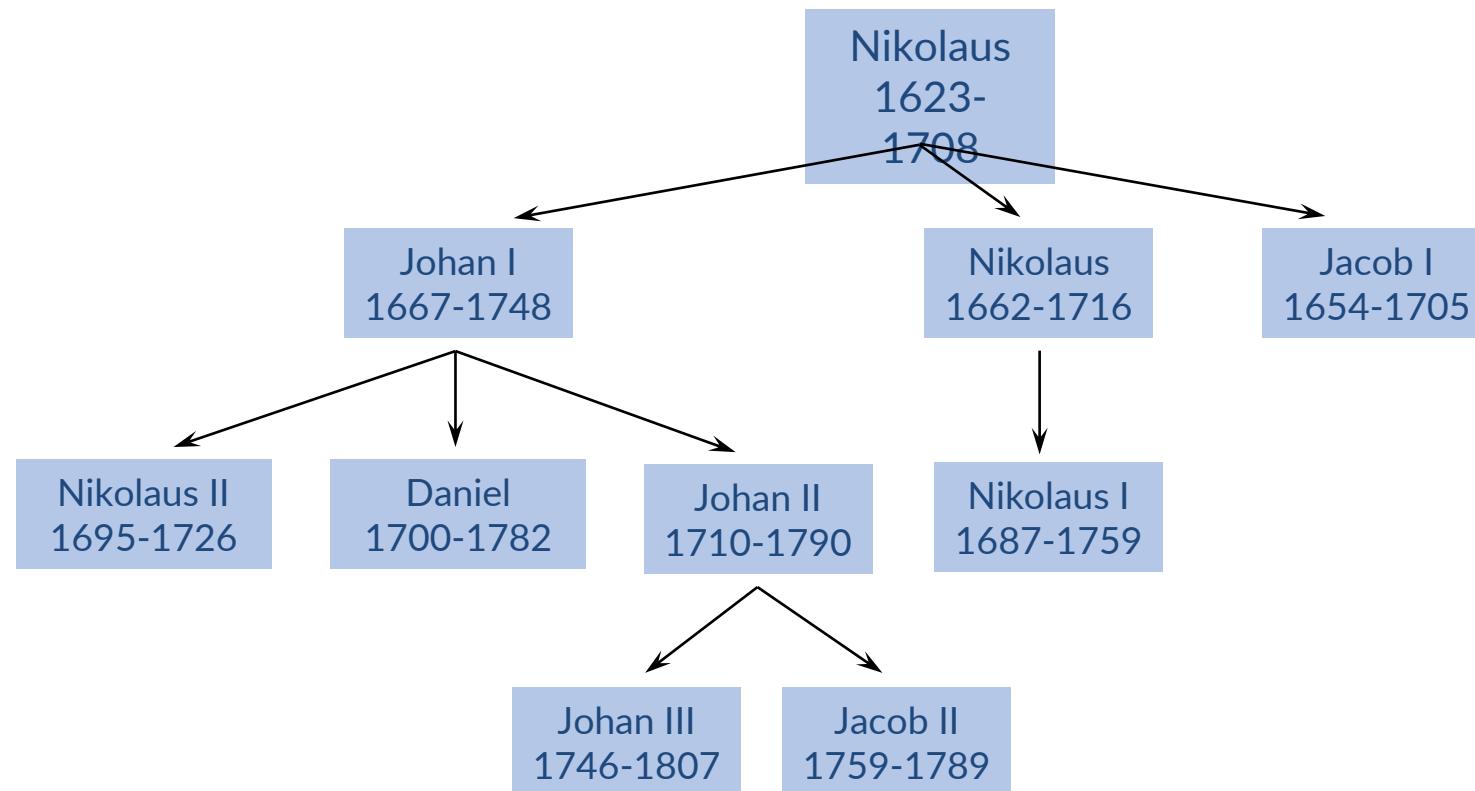
- 1.1. Định nghĩa cây
 - Ví dụ cây thực tế trong các ứng dụng



Diễn tả lịch thi đấu của các giải thể thao theo thể thức đấu loại trực tiếp, chẳng hạn vòng 2 của World Cup

1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

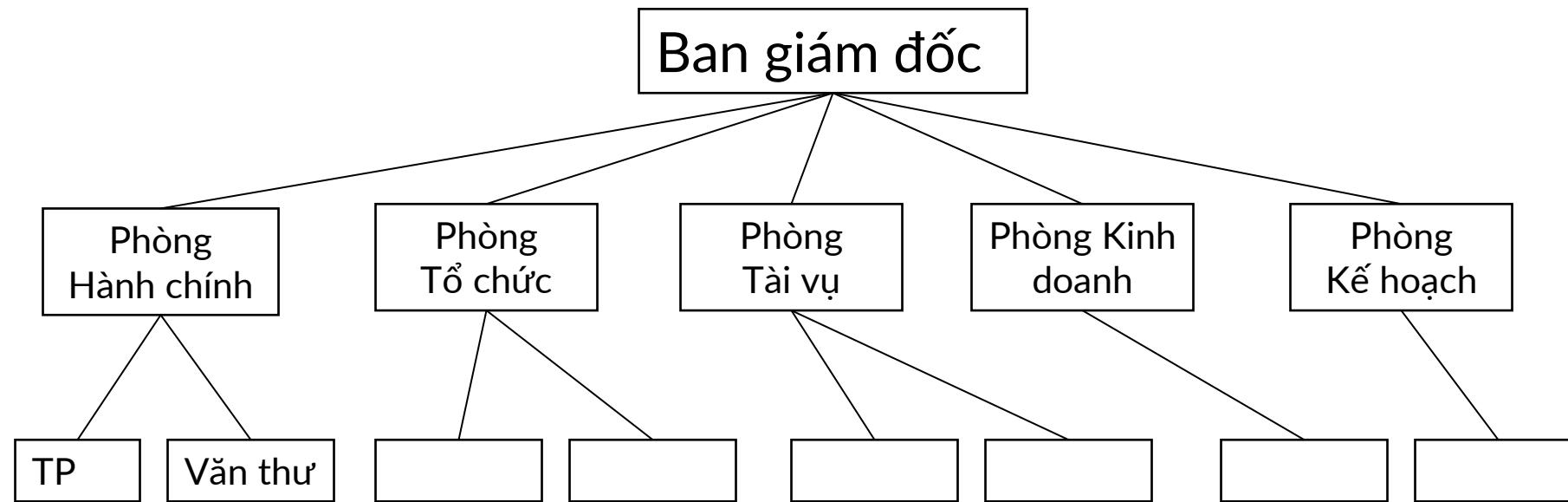
- 1.1. Định nghĩa cây
 - Ví dụ cây thực tế trong các ứng dụng



Cây gia phả của các nhà toán dòng họ Bernoulli

1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.1. Định nghĩa cây
 - Ví dụ cây thực tế trong các ứng dụng



Cây phân cấp quản lý hành chính

1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

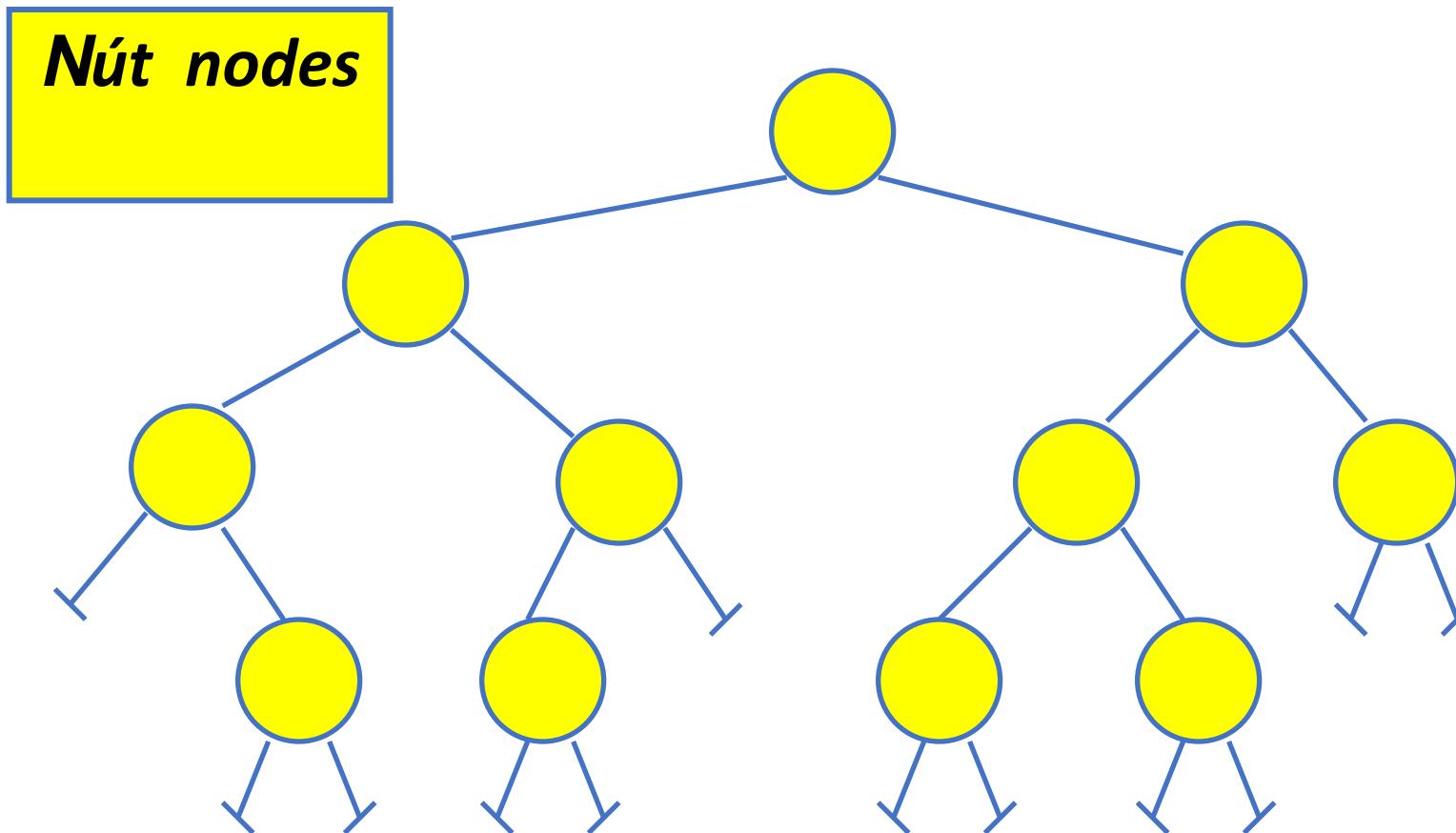
- 1.1. Định nghĩa cây
 - Ví dụ cây thực tế trong các ứng dụng



Cây thư mục

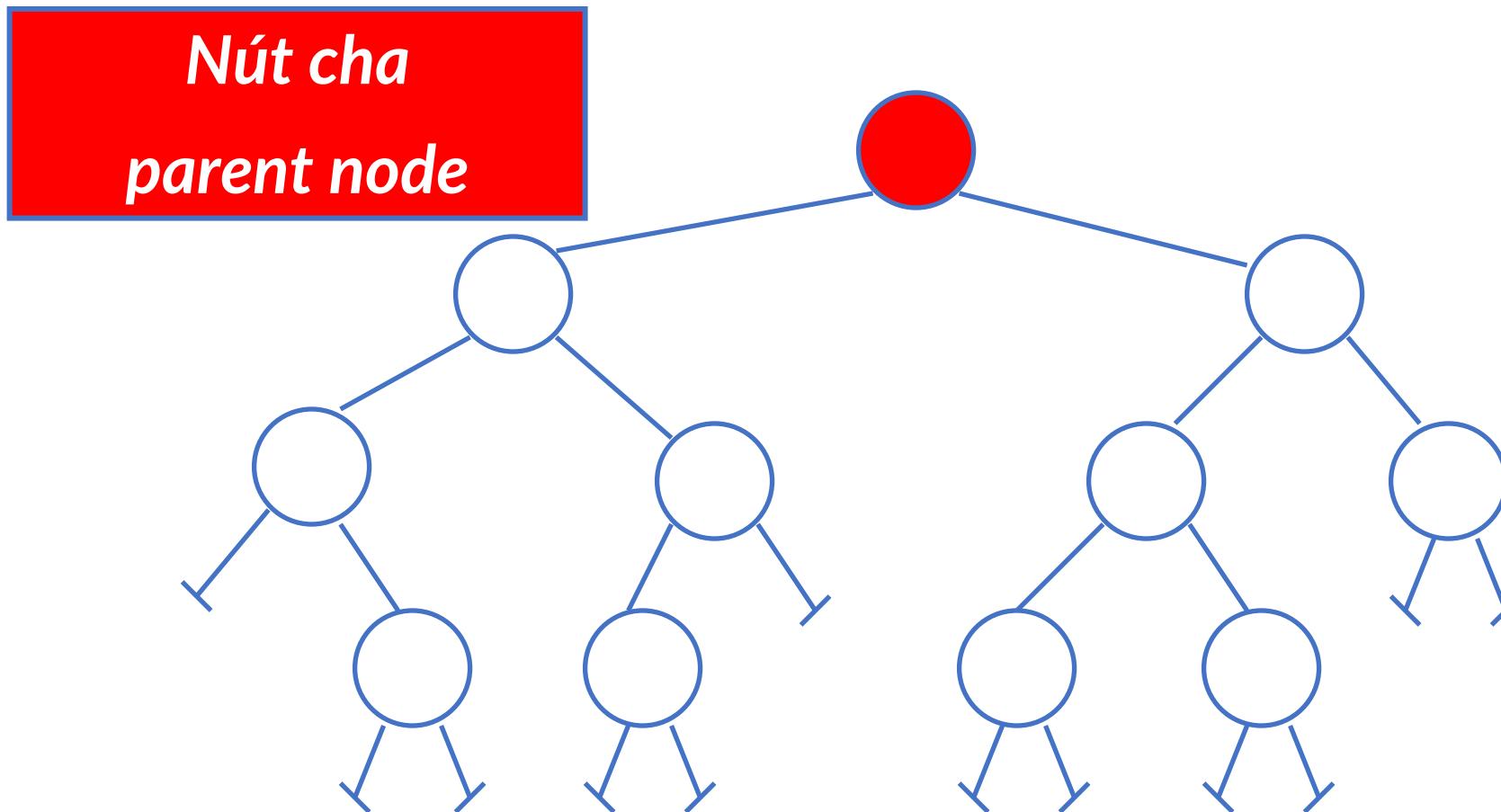
1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.2. Các thuật ngữ



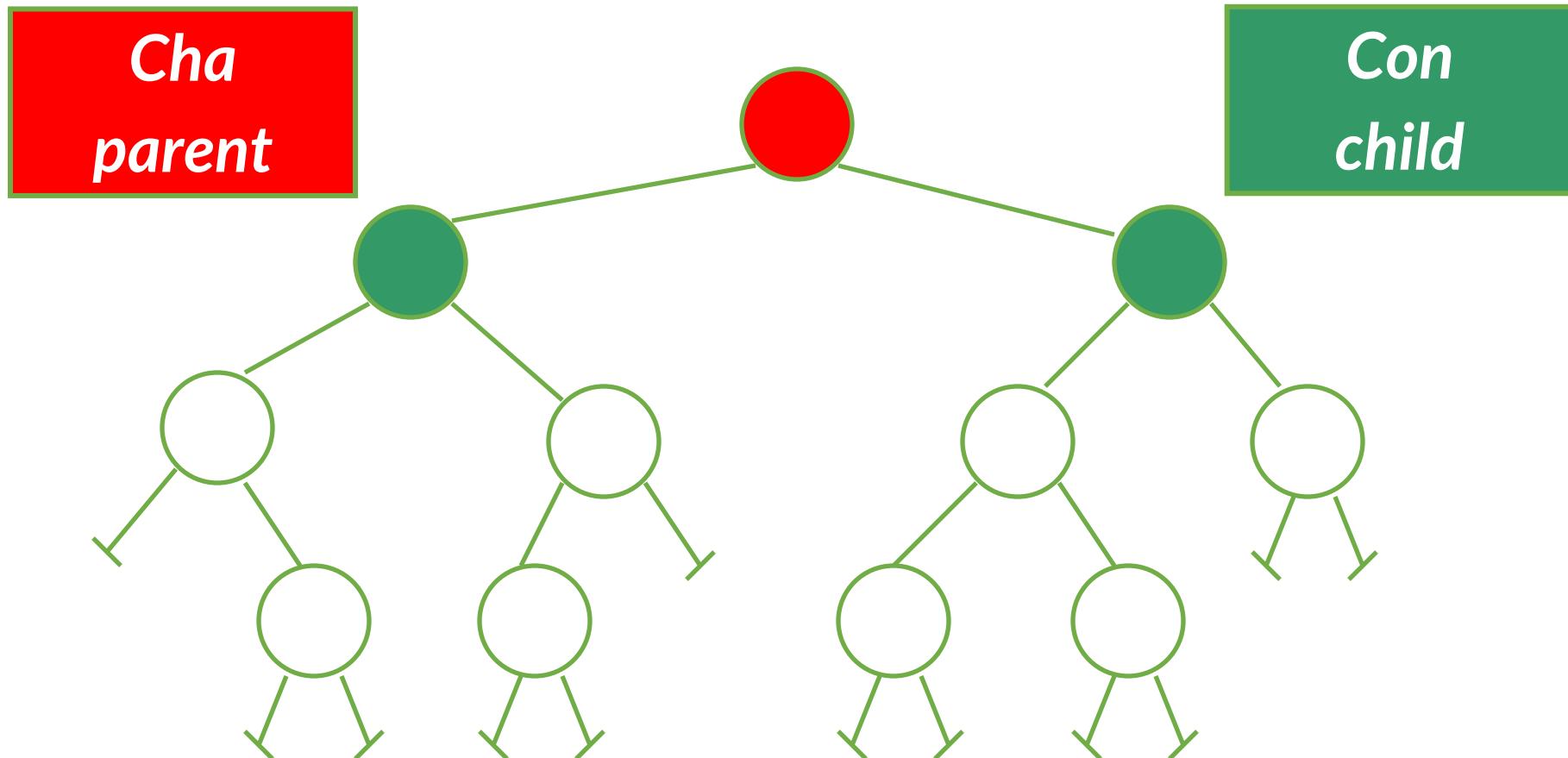
1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.2. Các thuật ngữ



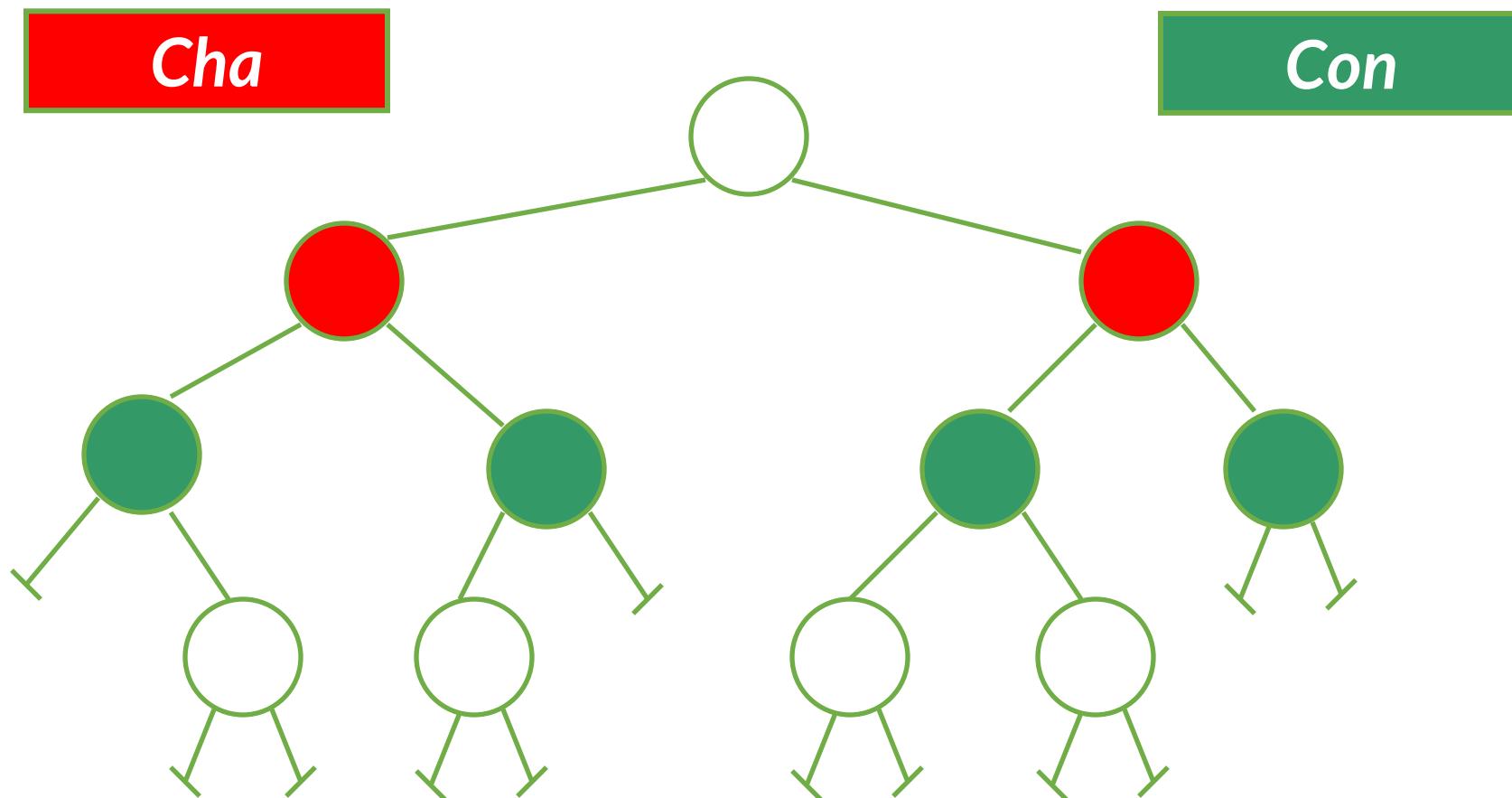
1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.2. Các thuật ngữ



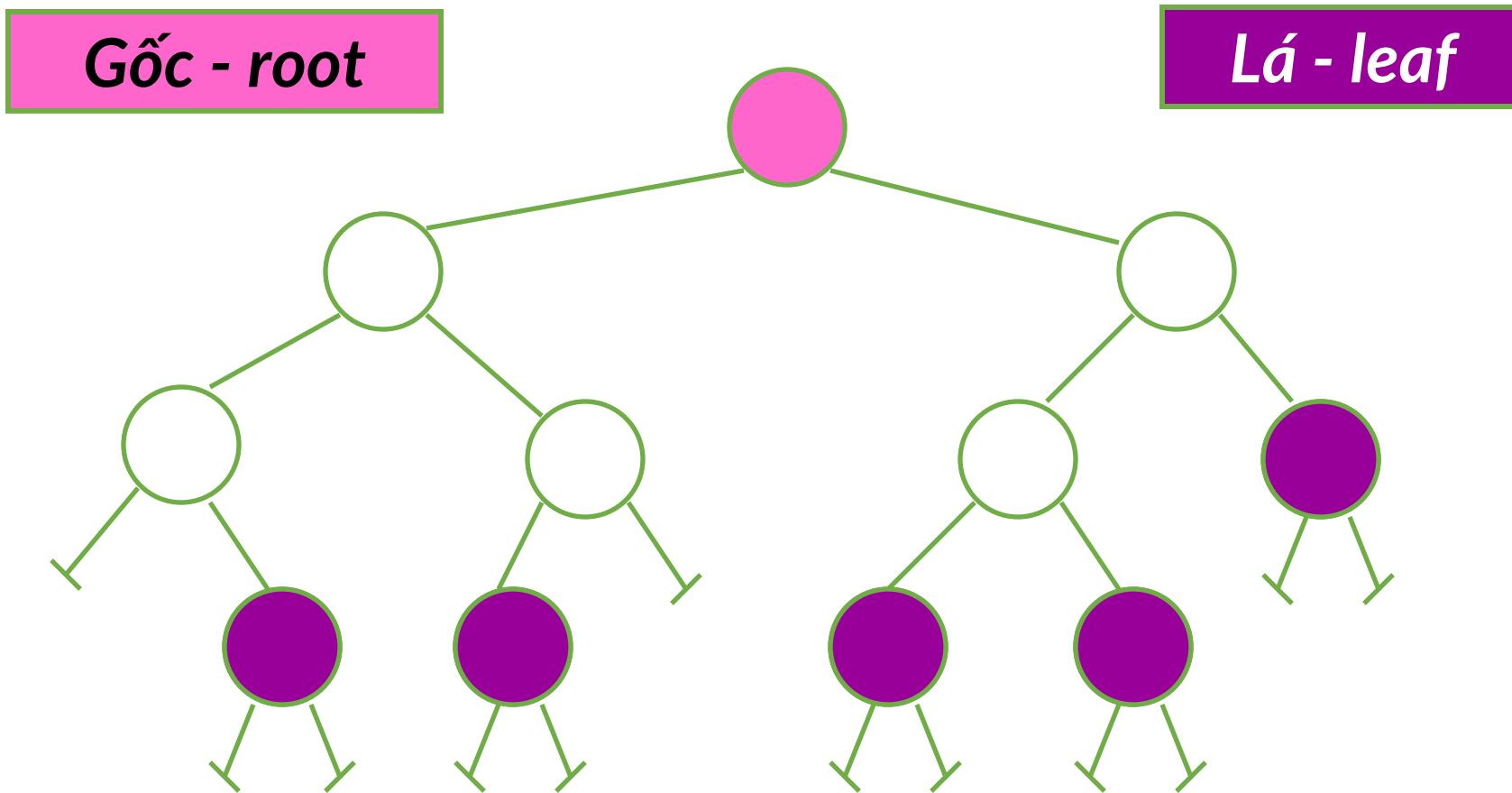
1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.2. Các thuật ngữ



1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

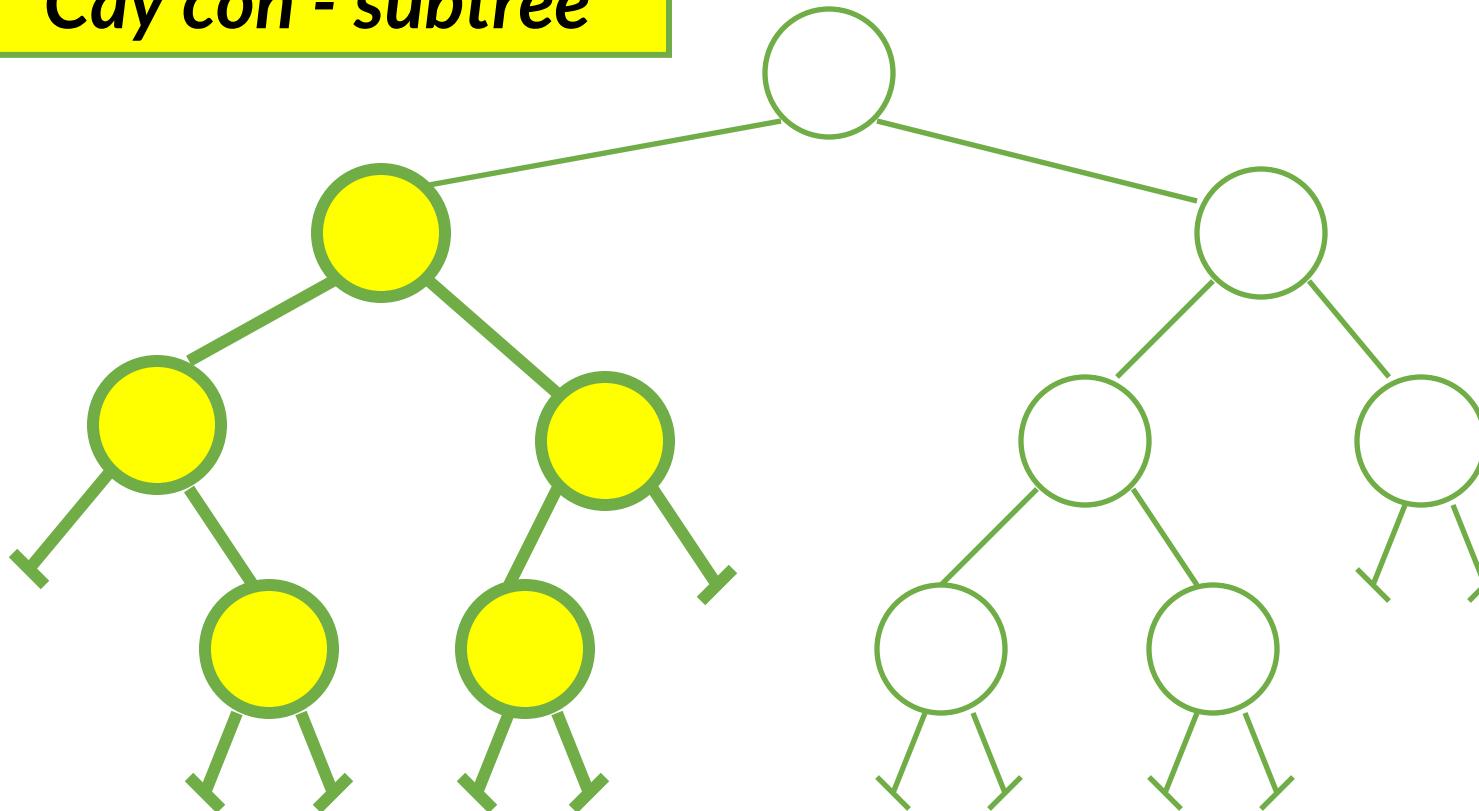
- 1.2. Các thuật ngữ



1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

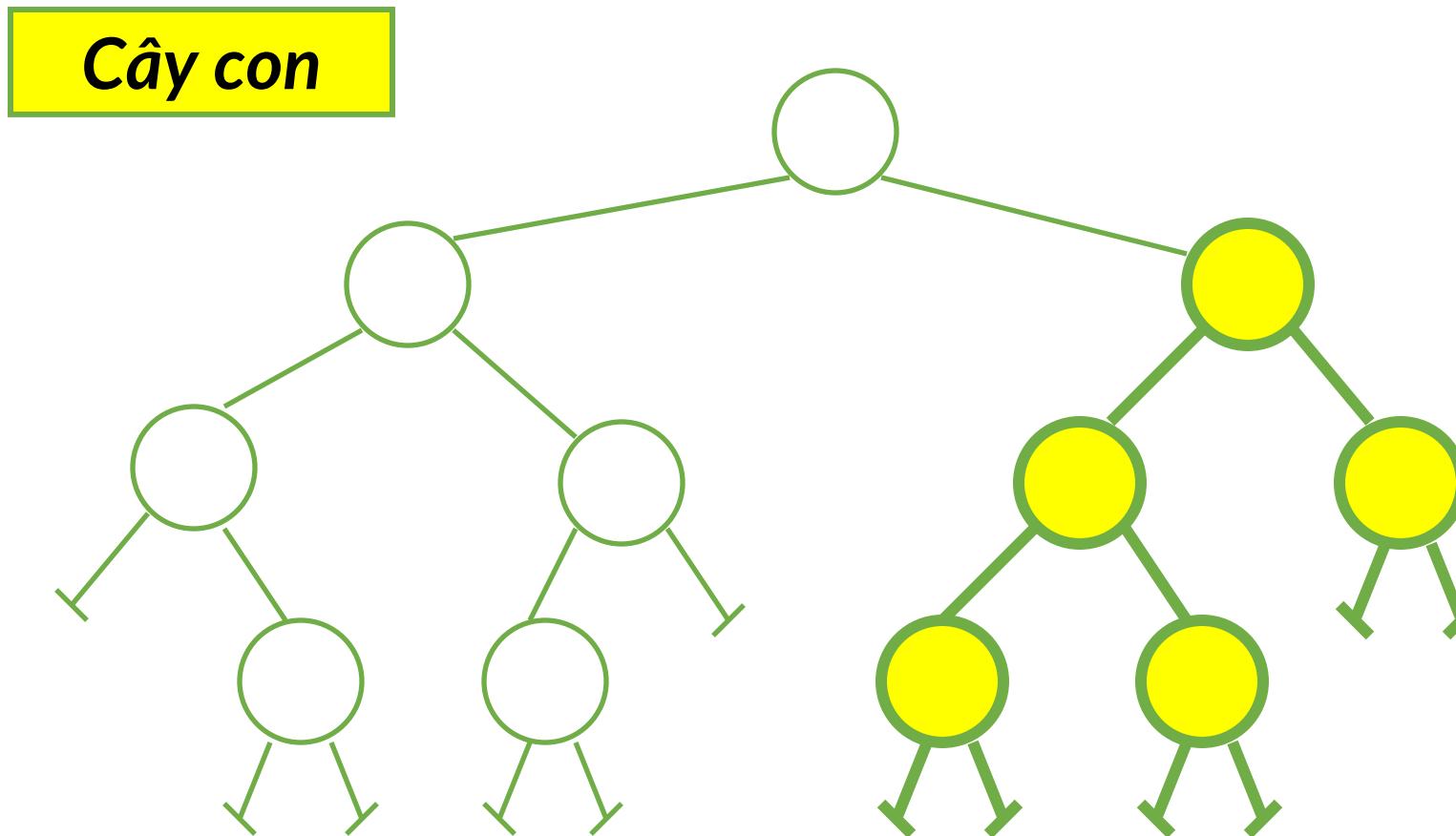
- 1.2. Các thuật ngữ

Cây con - subtree



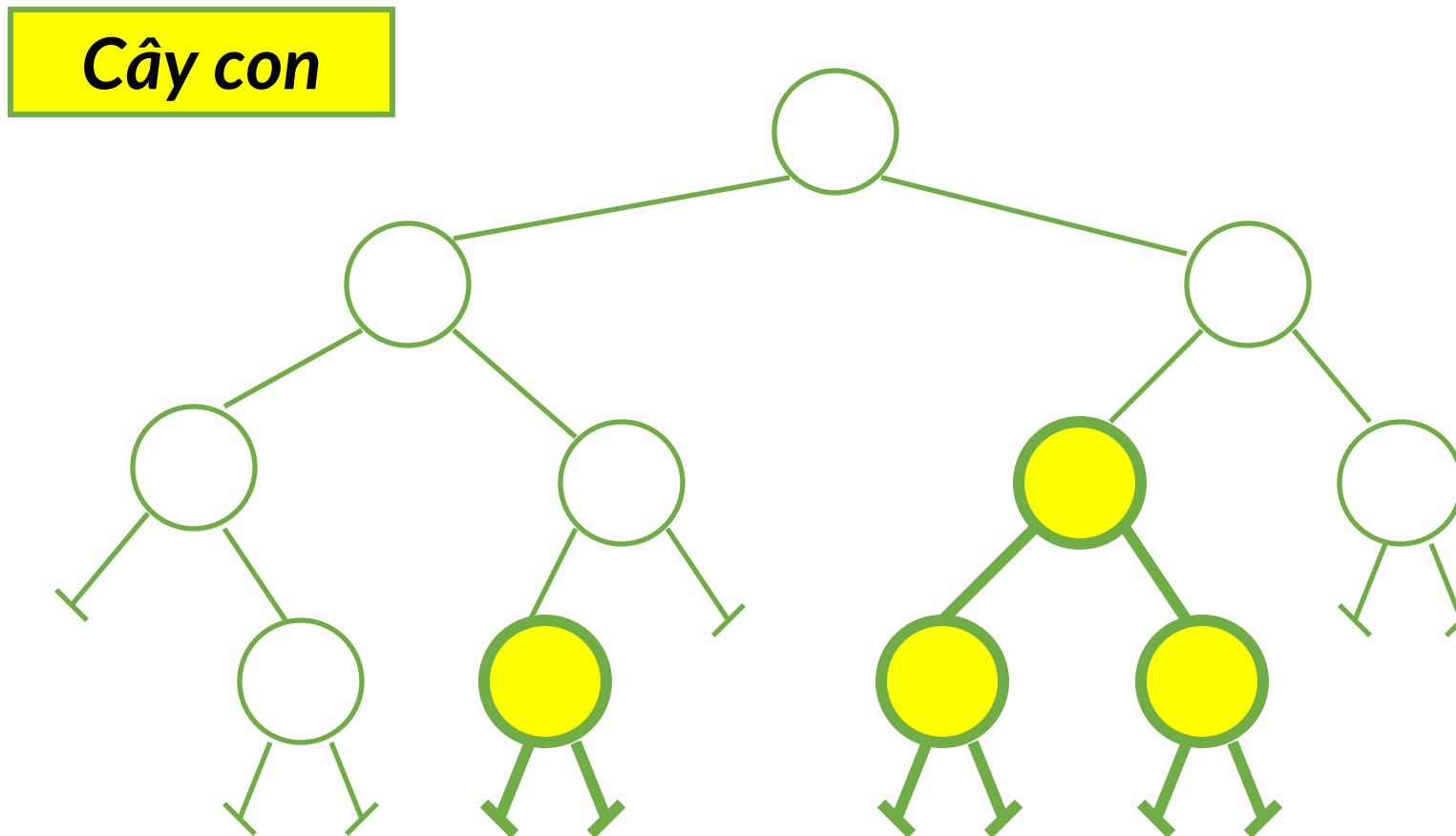
1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.2. Các thuật ngữ



1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

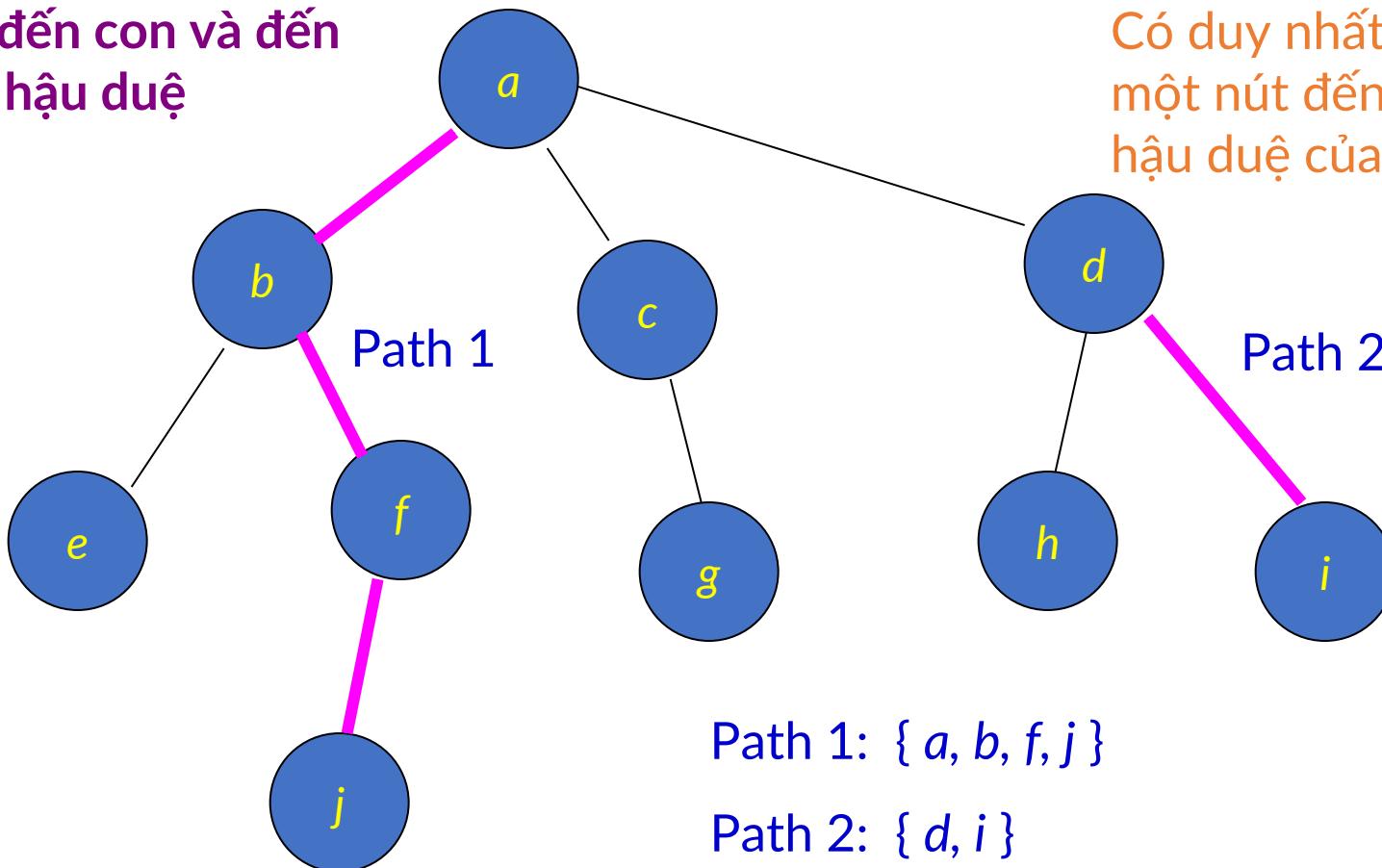
- 1.2. Các thuật ngữ



1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

• 1.2. Các thuật ngữ

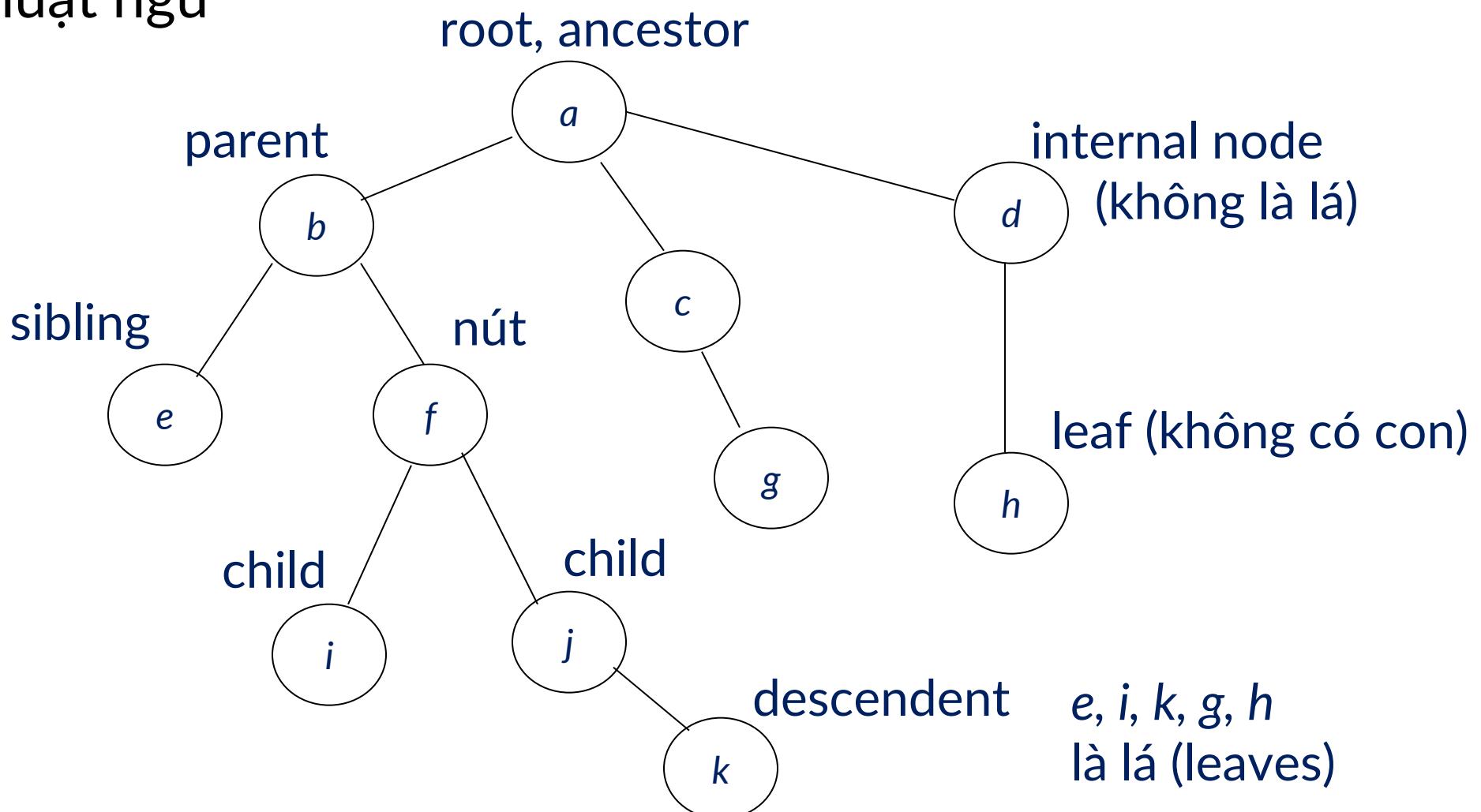
Từ cha đến con và đến các nút hậu duệ



Có duy nhất 1 đường đi từ một nút đến một nút là hậu duệ của nó

1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

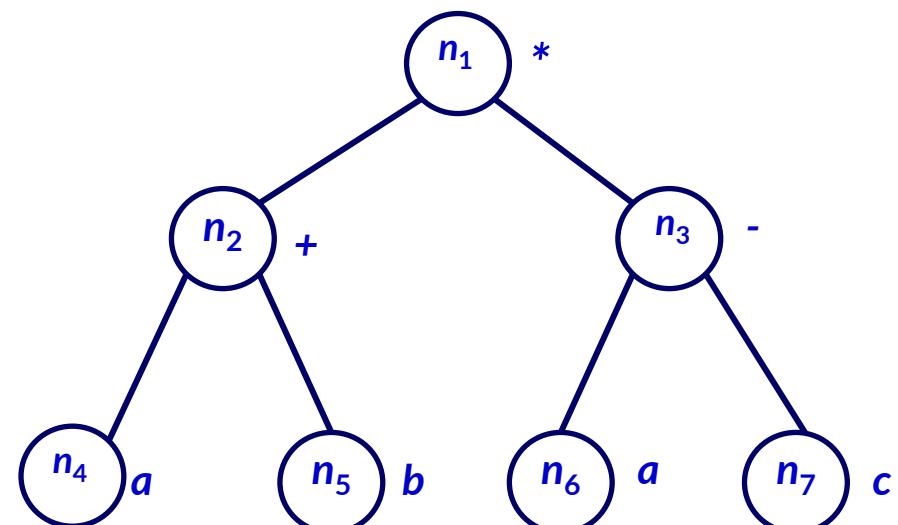
• 1.2. Các thuật ngữ



1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.2. Các thuật ngữ
Cây có nhãn (Labeled Tree)

- Mỗi nút của cây 1 nhãn (label) hoặc 1 giá trị
- Nhãn của nút không phải là tên gọi của nút mà là giá trị được cất giữ trong nó
- Ví dụ: Xét cây có 7 nút n1, ..., n7. Gán nhãn cho các nút:
 - Nút n1 có nhãn *;
 - Nút n2 có nhãn +;
 - Nút n3 có nhãn -;
 - Nút n4 có nhãn a;
 - Nút n5 có nhãn b;
 - Nút n6 có nhãn a;
 - Nút n7 có nhãn c.



Cây biểu thức $(a+b)^*(a-c)$

1. Tìm kiếm tuần tự

2. Cây nhị phân

2. CÂY NHỊ PHÂN

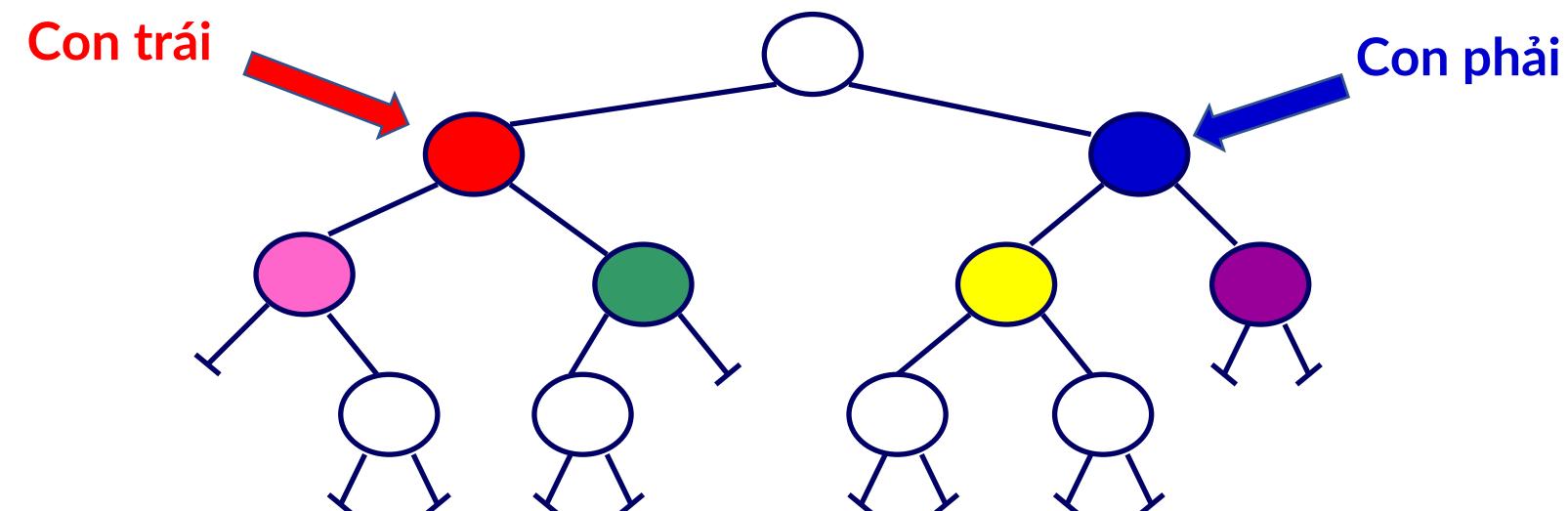
- 2.1. Cây nhị phân

- Cây nhị phân (Binary Tree): Cây mà mỗi nút có nhiều nhất là 2 con

- *Con trái* và *con phải*

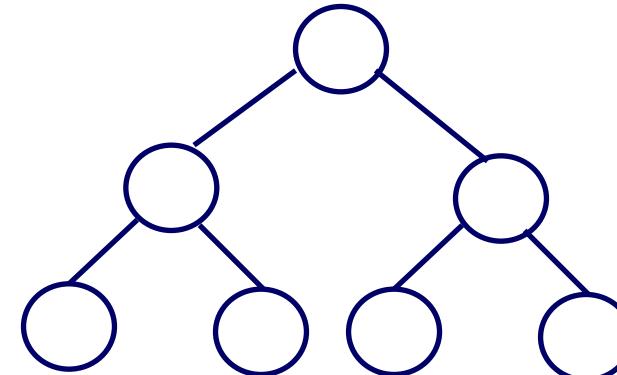
- Mỗi nút hoặc là:

- Không có con
- Chỉ có con trái
- Chỉ có con phải
- Con trái và con phải



2. CÂY NHỊ PHÂN

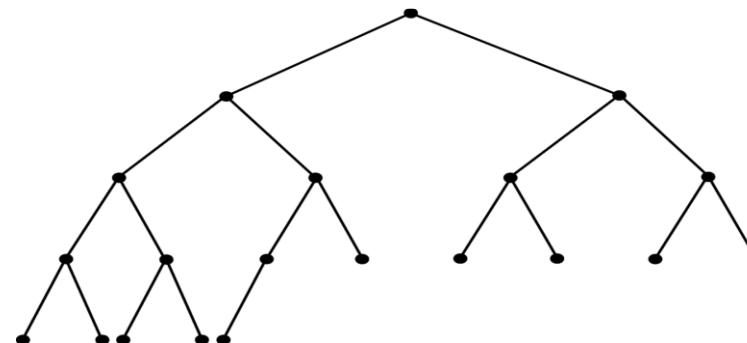
- 2.2. Cây nhị phân đầy đủ
 - **Cây nhị phân đầy đủ (Full Binary Trees):** Cây nhị phân thoả mãn
 - Mỗi nút lá đều có cùng độ sâu
 - Các nút trong có đúng 2 con



2. CÂY NHỊ PHÂN

• 2.3. Cây nhị phân hoàn chỉnh

- **Cây nhị phân hoàn chỉnh (Complete Binary Trees):** Cây nhị phân độ sâu n thoả mãn:
 - Là cây nhị phân đầy đủ nếu không tính đến các nút ở độ sâu n, và
 - Tất cả các nút ở độ sâu n là lệch sang trái nhất có thể được.
- Cây nhị phân hoàn chỉnh độ sâu n có số lượng nút nằm trong khoảng từ 2^{n-1} đến $2^n - 1$



Cây nhị phân hoàn chỉnh

2. CÂY NHỊ PHÂN

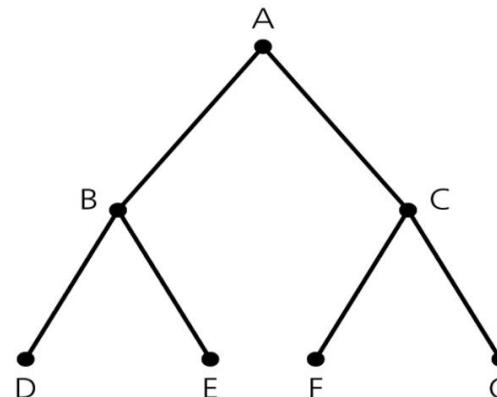
• 2.4. Cây nhị phân cân đối

- **Cây nhị phân được gọi là cân đối (balanced)** nếu chiều cao của cây con trái và chiều cao của cây con phải chênh lệch nhau không quá 1 đơn vị

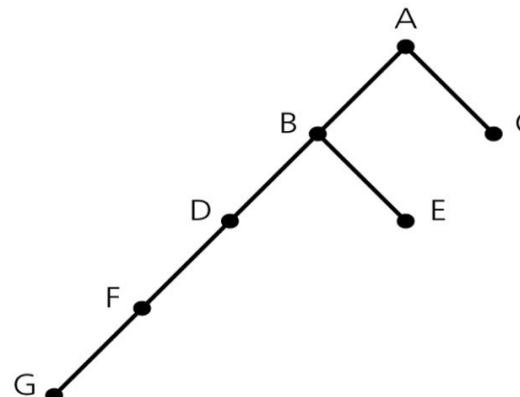
- Nhận xét:

- Nếu cây nhị phân là đầy đủ thì nó là hoàn chỉnh
- Nếu cây nhị phân là hoàn chỉnh thì nó là cân đối

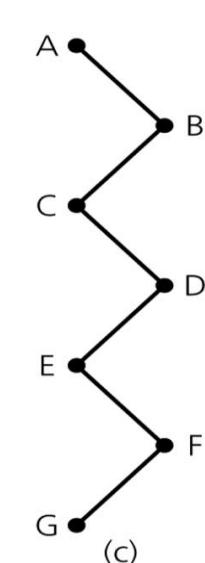
Ví dụ:



(a)



(b)



(c)

1. Cây nào là đầy đủ?
2. Cây nào là hoàn chỉnh?
3. Cây nào là cân đối?



Chương 6- Cây

Bài 2. Cấu trúc dữ liệu biểu diễn cây,
duyệt cây

ONE LOVE. ONE FUTURE.

MỤC TIÊU

Sau bài học này, người học có thể:

1. Hiểu được khái niệm duyệt cây theo thứ tự trước, sau và giữa
2. Cài đặt được cấu trúc dữ liệu biểu diễn cây

NỘI DUNG TIẾP THEO

1. Cấu trúc dữ liệu biểu diễn cây

- 1.1. Biểu diễn cây bằng mảng
- 1.2. Biểu diễn cây bằng danh sách các con
- 1.3. Biểu diễn cây bằng con trái và em kế cận phải

2. Duyệt cây

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.1. Biểu diễn cây bằng mảng
 - Giả sử T là cây với các nút đặt tên là $1, 2, \dots, n$.
 - Biểu diễn T :
 - danh sách tuyến tính A trong đó mỗi phần tử $A[i]$ chứa **con trỏ đến cha** của **nút i** .
 - Gốc của T có thể phân biệt bởi con trỏ rỗng.
 - Đặt $A[i] = j$ nếu nút j là cha của nút i ,
 $A[i] = 0$ nếu nút i là gốc.
 - thao tác ***parent*** trả về cha của một nút
 - Cách biểu diễn này dựa trên cơ sở là mỗi nút của cây (ngoại trừ gốc) đều có duy nhất một cha.

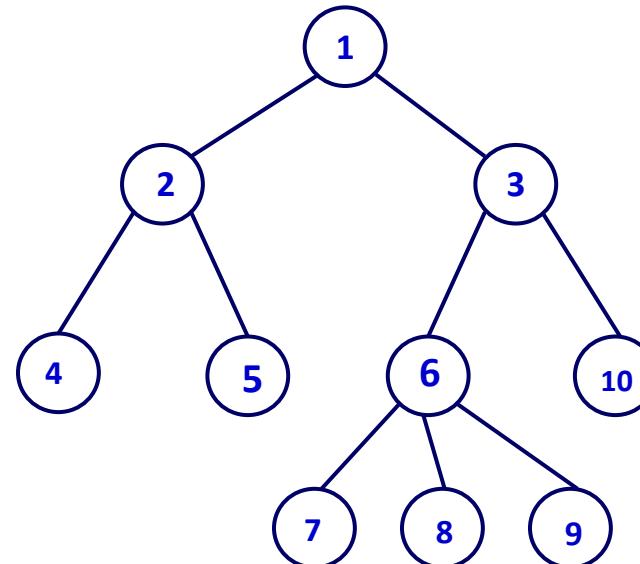
1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.1. Biểu diễn cây bằng mảng
 - Với cách biểu diễn này **cha** của 1 nút có thể xác định trong **thời gian hằng số**.
 - **Đường đi từ 1 nút đến tổ tiên** (kể cả đến gốc):
$$n \leftarrow \text{parent}(n) \leftarrow \text{parent}(\text{parent}(n)) \leftarrow \dots$$
 - Có thể dùng thêm mảng $L[i]$ để hỗ trợ việc ghi nhận nhãn cho các nút,
 - hoặc biến mỗi phần tử $A[i]$ thành bản ghi gồm **2 trường**:
 - biến nguyên ghi nhận cha
 - nhãn.

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.1. Biểu diễn cây bằng mảng

- Ví dụ



A	0	1	1	2	2	3	6	6	6	3
---	---	---	---	---	---	---	---	---	---	---

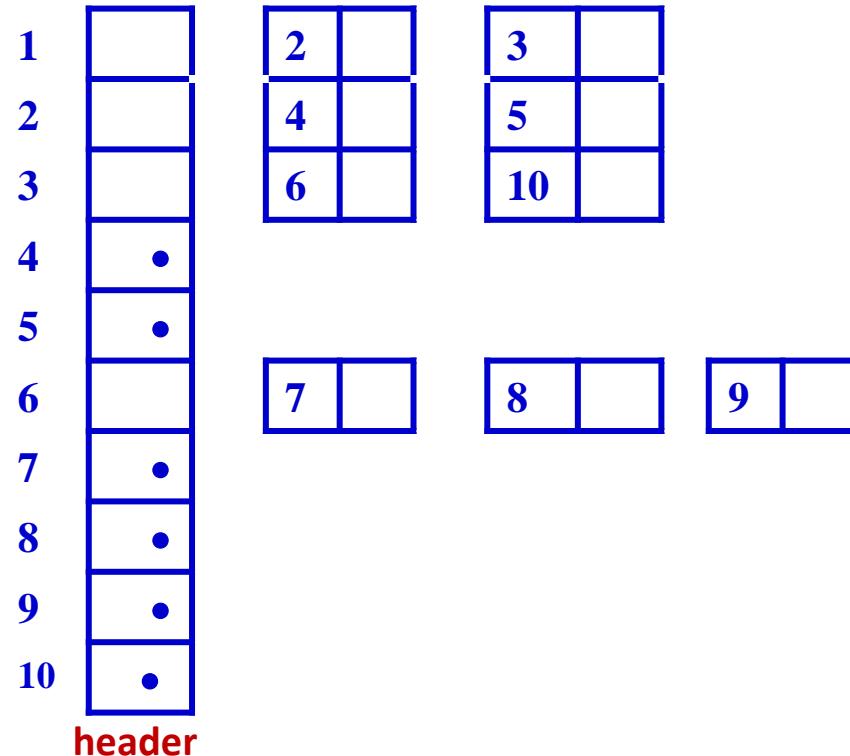
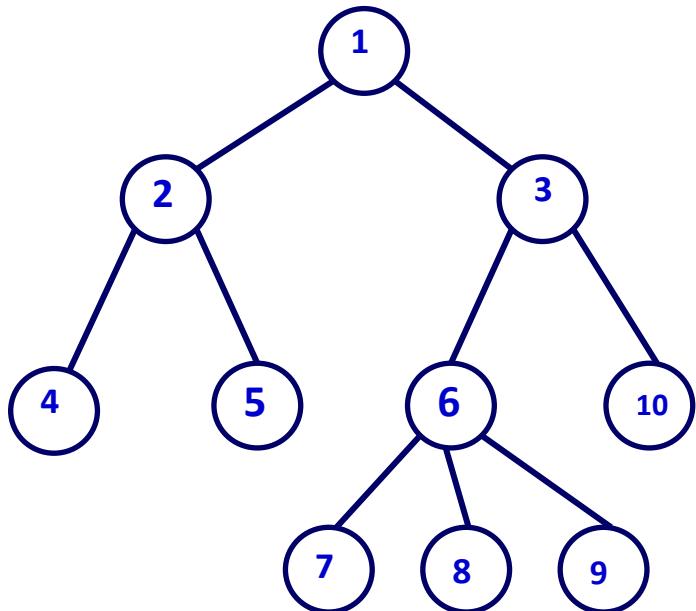
- Hạn chế:** Cách dùng con trỏ cha không thích hợp cho các thao tác với con.
- Cho nút n , mất **nhiều thời gian** để xác định các **con của n , chiều cao của n** .
- Không cho thứ tự của các nút con \rightarrow **leftmost_child** và **right_sibling** là không xác định
 \rightarrow cách biểu diễn này chỉ dùng trong một số trường hợp nhất định.

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.2. Biểu diễn cây bằng danh sách các con
 - Mỗi nút của cây ta cất giữ 1 danh sách các con của nó.
 - Danh sách con có thể biểu diễn bởi một trong những cách biểu diễn danh sách đã trình bày trong chương trước.
 - Số lượng con của các nút là rất khác nhau → danh sách móc nối thường là lựa chọn thích hợp nhất.

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.2. Biểu diễn cây bằng danh sách các con



- Có mảng con trỏ đến đầu các danh sách con của các nút $1, 2, \dots, 10$:
 $\text{header}[i]$ trỏ đến danh sách con của nút i .

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.2. Biểu diễn cây bằng danh sách các con
- **Ví dụ:** Có thể sử dụng mô tả sau đây để biểu diễn cây

```
typedef ? NodeT; /* dấu ? cần thay bởi định nghĩa kiểu phù hợp */
typedef ? ListT; /* dấu ? cần thay bởi định nghĩa kiểu danh sách phù hợp */
typedef ? position;
typedef struct
{
    ListT header[maxNodes];
    labeltype labels[maxNodes];
    NodeT root;
} TreeT;
```

- Giả thiết rằng gốc của cây được cất giữ trong trường *root* và 0 để thể hiện nút rỗng.

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.2. Biểu diễn cây bằng danh sách các con
 - Dưới đây là minh họa cài đặt phép toán **leftmost_child**. Việc cài đặt các phép toán còn lại được coi là bài tập.

```
NodeT leftmost_child (NodeT n, TreeT T)
/* trả lại con trái nhất của nút n trong cây T */
{
    ListT L; /* danh sách các con của n */
    L = T.header[n];
    if (empty(L)) /* n là lá */
        return(0);
    else return(retrive ( first(L), L));
}
```

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

• 1.3. Biểu diễn cây bằng con trái và em kế cận phải

Nhận xét:

- Mỗi một nút của cây hoặc là
 - không có con
 - có đúng 1 nút con cực trái
 - không có em kế cận phải,
 - có đúng 1 nút em kế cận phải
- Để biểu diễn cây: lưu trữ thông tin về con cực trái và em kế cận phải của mỗi nút.

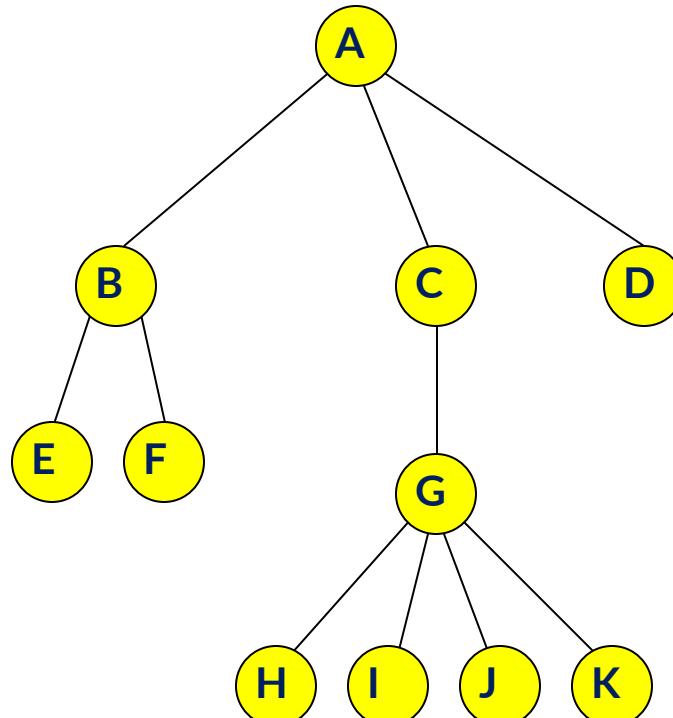
Sử dụng mô tả sau:

```
struct Tnode
{
    char word[20]; // Dữ liệu cất giữ ở nút
    struct Tnode *leftmost_child;
    struct Tnode *right_sibling;
};

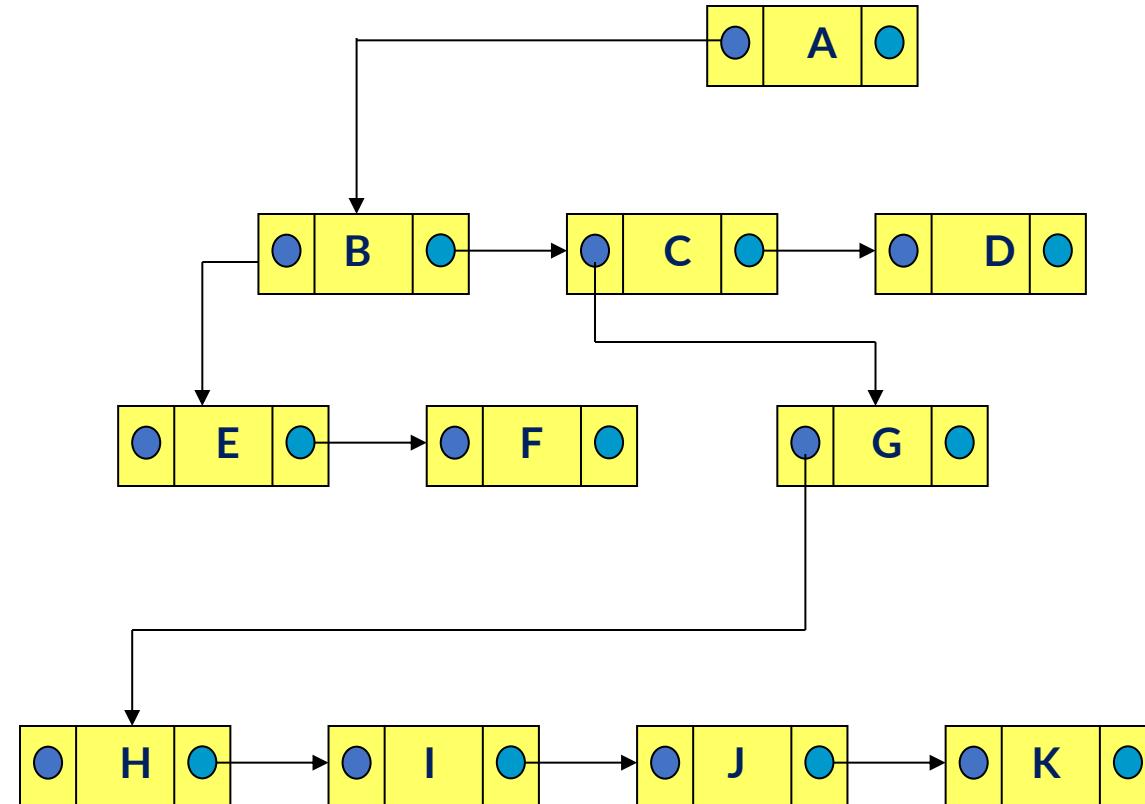
typedef struct Tnode treeNode;
treeNode Root;
```

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.3. Biểu diễn cây bằng con trái và em kế cận phải



Cây tổng quát



Biểu diễn cây

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.3. Biểu diễn cây bằng con trái và em kế cận phải

Nhận xét:

- Với cách biểu diễn này, các thao tác cơ bản dễ dàng cài đặt.
- Chỉ có thao tác **parent** là đòi hỏi phải duyệt danh sách nên không hiệu quả.
 - Trong trường hợp phép toán này phải dùng thường xuyên, bổ sung thêm một trường nữa vào bản ghi để lưu cha của nút.

NỘI DUNG TIẾP THEO

1. Cấu trúc dữ liệu biểu diễn cây

2. Duyệt cây

2.1. Duyệt theo thứ tự trước

2.2. Duyệt theo thứ tự sau

2.3. Duyệt theo thứ tự giữa

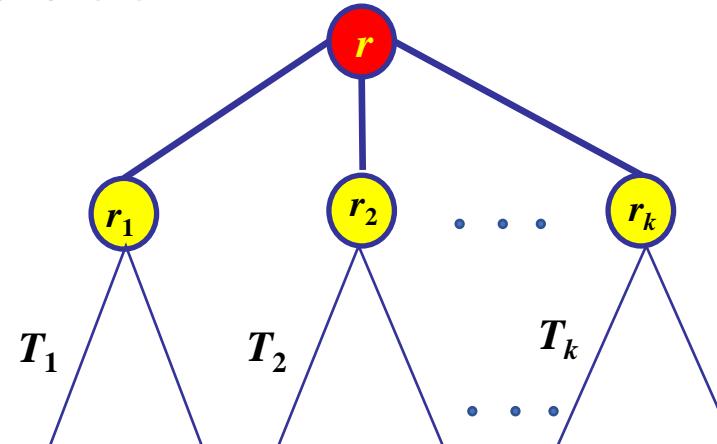
3. Cây nhị phân

2. DUYỆT CÂY

- Xếp thứ tự các nút
 - Thứ tự trước, Thứ tự sau và Thứ tự giữa (Preorder, Postorder và Inorder)
 - Các thứ tự này được định nghĩa một cách đệ qui như sau
 - Nếu cây T là rỗng, thì danh sách rỗng là danh sách theo thứ tự trước, thứ tự sau và thứ tự giữa của cây T .
 - Nếu cây T có 1 nút, thì nút đó chính là danh sách theo thứ tự trước, thứ tự sau và thứ tự giữa của cây T .
 - Trái lại, giả sử T là cây có gốc r với các cây con là T_1, T_2, \dots, T_k .

2. DUYỆT CÂY

• 2.1. Duyệt theo thứ tự trước



- **Thứ tự trước** (hay duyệt theo thứ tự trước - *preorder traversal*) của các nút của T là:
 - Gốc r của T ,
 - Tiếp đến là các nút của T_1 theo thứ tự trước,
 - Tiếp đến là các nút của T_2 theo thứ tự trước,
 - ...
 - Và cuối cùng là các nút của T_k theo thứ tự trước.

2. DUYỆT CÂY

• 2.1. Duyệt theo thứ tự trước

Thuật toán:

```
void PREORDER( nodeT r )
```

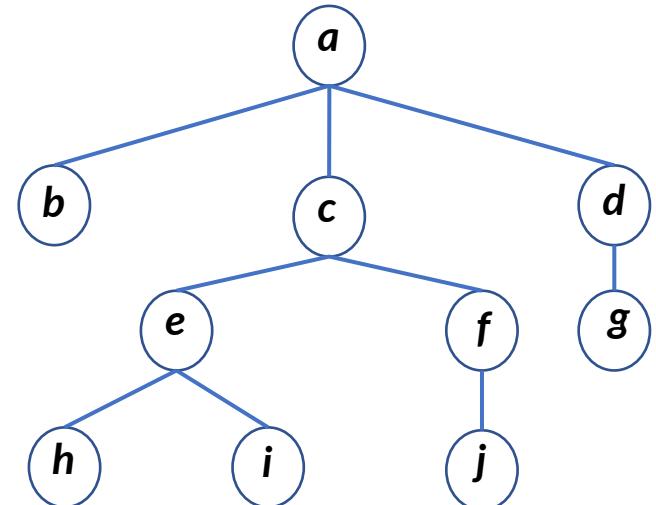
```
{
```

- (1) Đưa ra r ;
- (2) **for** (mỗi con c của r , nếu có, theo thứ tự từ trái sang) **do**

```
    PREORDER(c);
```

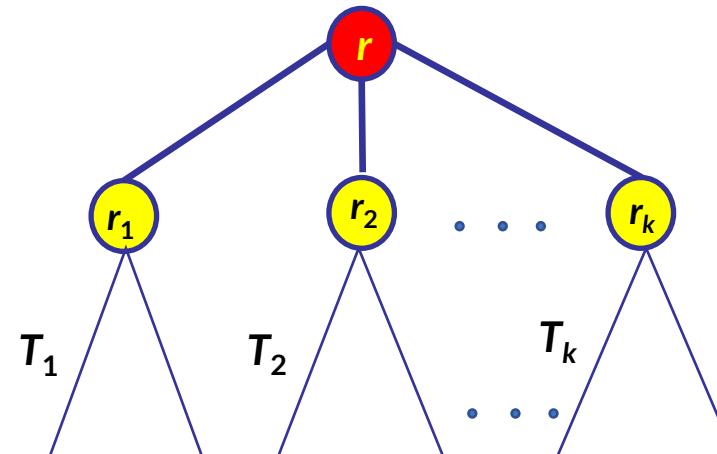
```
}
```

Ví dụ: Thứ tự trước của các đỉnh của cây trên hình vẽ là
 $a, b, c, e, h, i, f, j, d, g$



2. DUYỆT CÂY

• 2.2. Duyệt theo thứ tự sau



- **Thứ tự sau** của các nút của cây T là:
 - Các nút của T_1 theo thứ tự sau,
 - tiếp đến là các nút của T_2 theo thứ tự sau,
 - ...
 - các nút của T_k theo thứ tự sau,
 - sau cùng là nút gốc r .

2. DUYỆT CÂY

• 2.2. Duyệt theo thứ tự sau

Thuật toán:

```
void POSTORDER ( nodeT r )
```

```
{
```

```
    for (mỗi con c của r, nếu có, theo thứ tự từ trái sang)
```

```
    do
```

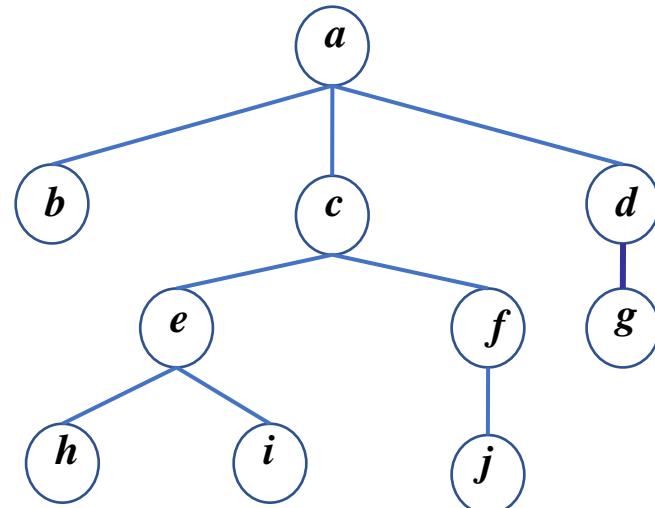
```
        POSTORDER(c)
```

```
    Đưa ra r;
```

```
}
```

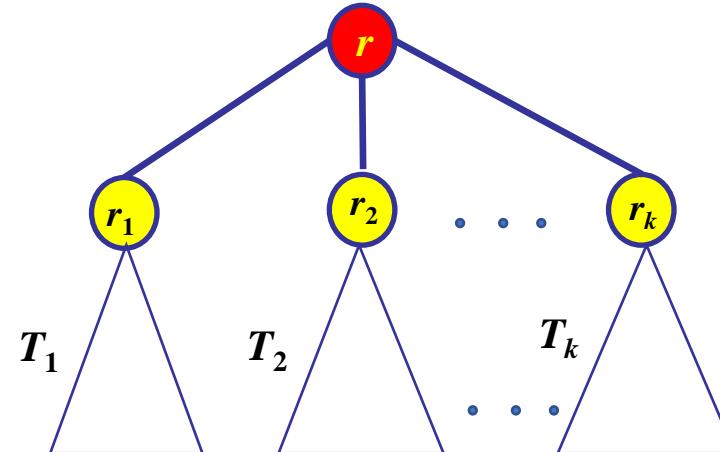
- **Ví dụ:** Dãy các đỉnh được liệt kê theo **thứ tự sau** của cây trong hình vẽ là:

b, h, i, e, j, f, c, g, d, a



2. DUYỆT CÂY

• 2.3. Duyệt theo thứ tự giữa



- **Thứ tự giữa** của các nút của cây T là:
 - Các nút của T_1 theo thứ tự giữa,
 - Tiếp đến là nút gốc r ,
 - Tiếp theo là các nút của T_2, \dots, T_k , mỗi nhóm nút được xếp theo thứ tự giữa.

2. DUYỆT CÂY

• 2.3. Duyệt theo thứ tự giữa

```
void INORDER (nodeT r )
```

```
{
```

```
    if ( r là lá ) Đưa ra r;
```

```
else
```

```
{
```

```
    INORDER(con trái nhất của r);
```

```
    Đưa ra r;
```

```
    for (mỗi con c của r, ngoại trừ con trái nhất, theo thứ tự từ trái sang) do
```

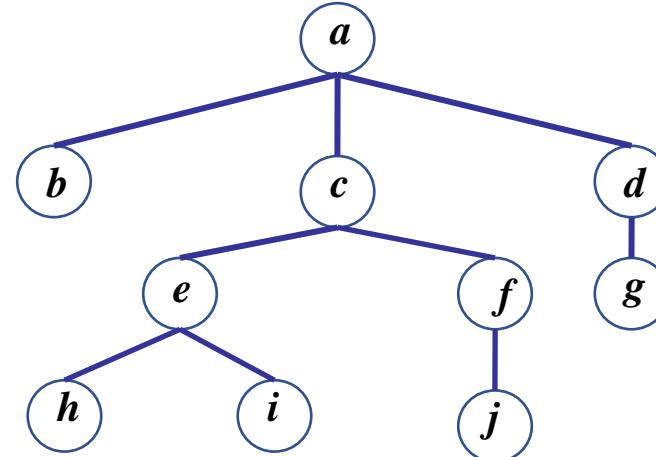
```
        INORDER(c);
```

```
}
```

```
}
```

- **Ví dụ:** Dãy các đỉnh của cây trong hình vẽ được liệt kê theo thứ tự giữa:

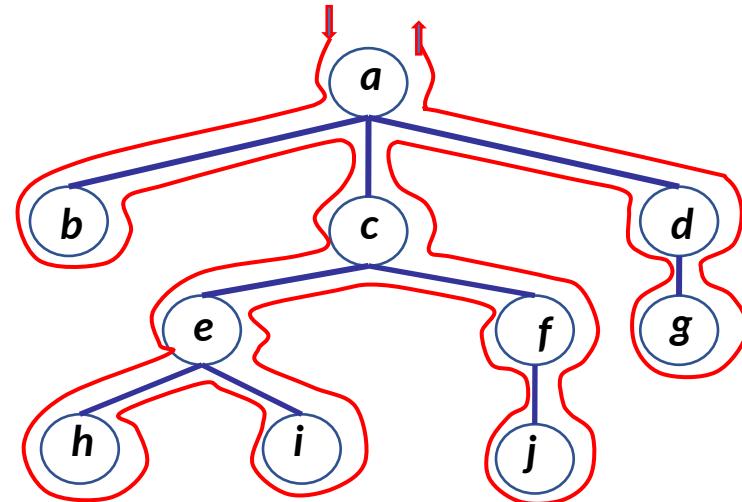
b, a, h, e, i, c, j, f, g, d



2. DUYỆT CÂY

▪ Xếp thứ tự các nút

Đi vòng quanh bên ngoài cây bắt đầu từ gốc, ngược chiều kim đồng hồ và sát theo cây nhất



- **Thứ tự trước:** đưa ra nút mỗi khi đi qua nó.
- **Thứ tự sau:** đưa ra nút khi qua nó ở lần cuối trước khi quay về cha của nó.
- **Thứ tự giữa:** đưa ra lá ngay khi đi qua nó, còn những nút trong được đưa ra khi lần thứ hai được đi qua.
- **Chú ý:** các lá được xếp thứ tự từ trái sang phải như nhau trong cả 3 cách sắp xếp.

NỘI DUNG TIẾP THEO

1. Cấu trúc dữ liệu biểu diễn cây

2. Duyệt cây

3. Cây nhị phân

3.1. Biểu diễn cây nhị phân

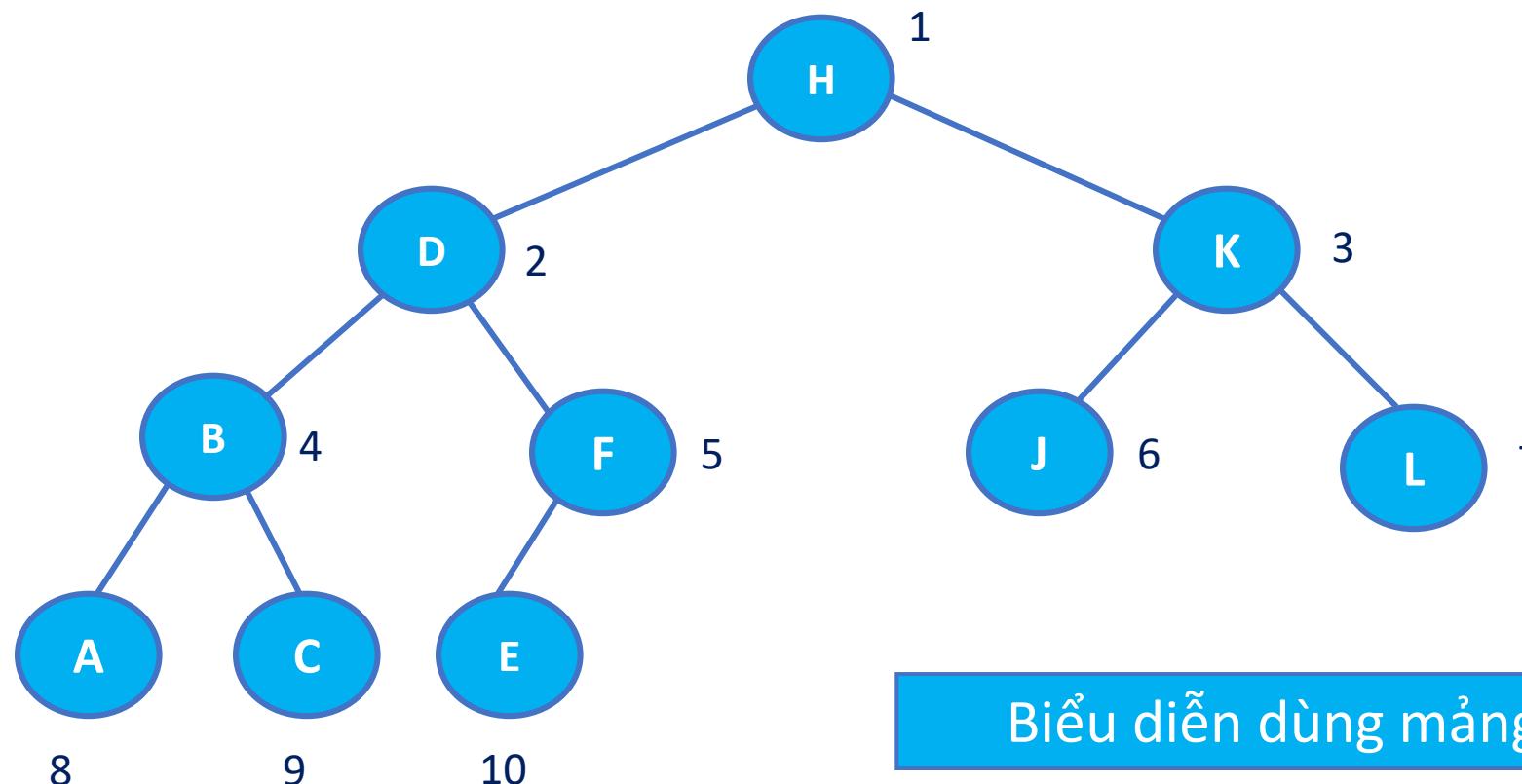
3.2. Duyệt cây nhị phân

3. CÂY NHỊ PHÂN

- 3.1. Biểu diễn cây nhị phân
 - **Biểu diễn sử dụng mảng**
 - Tương tự như trong cách biểu diễn cây tổng quát.
 - Trong trường hợp cây nhị phân hoàn chỉnh, có thể cài đặt nhiều phép toán với cây rất hiệu quả.
 - Xét cây nhị phân hoàn chỉnh T có n nút, trong đó mỗi nút chứa một giá trị.
 - Gán tên cho các nút của cây nhị phân hoàn chỉnh T từ trên xuống dưới và từ trái qua phải bằng các số $1, 2, \dots, n$.
 - Đặt tương ứng cây T với **mảng A** trong đó phần tử thứ i của A là giá trị cất giữ trong nút thứ i của cây T , $i = 1, 2, \dots, n$.

3. CÂY NHỊ PHÂN

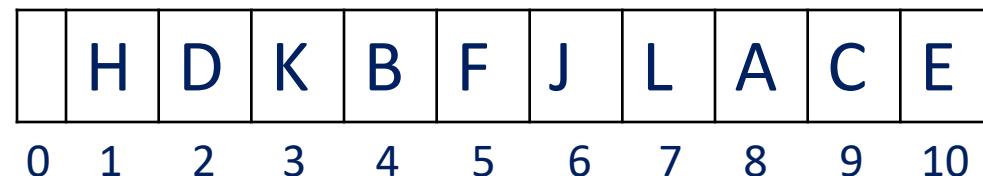
- 3.1. Biểu diễn cây nhị phân
 - Biểu diễn sử dụng mảng



	H	D	K	B	F	J	L	A	C	E
0	1	2	3	4	5	6	7	8	9	10

3. CÂY NHỊ PHÂN

- 3.1. Biểu diễn cây nhị phân
 - Biểu diễn sử dụng mảng

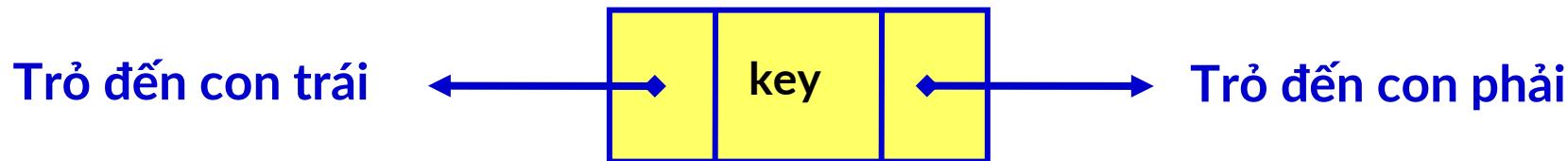


Để tìm	Sử dụng	Hạn chế
Con trái của $A[i]$	$A[2*i]$	$2*i \leq n$
Con phải của $A[i]$	$A[2*i + 1]$	$2*i + 1 \leq n$
Cha của $A[i]$	$A[i/2]$	$i > 1$
Gốc	$A[1]$	A khác rỗng
Thứ $A[i]$ là lá?	True	$2*i > n$

3. CÂY NHỊ PHÂN

- 3.1. Biểu diễn cây nhị phân
 - Biểu diễn sử dụng con trỏ

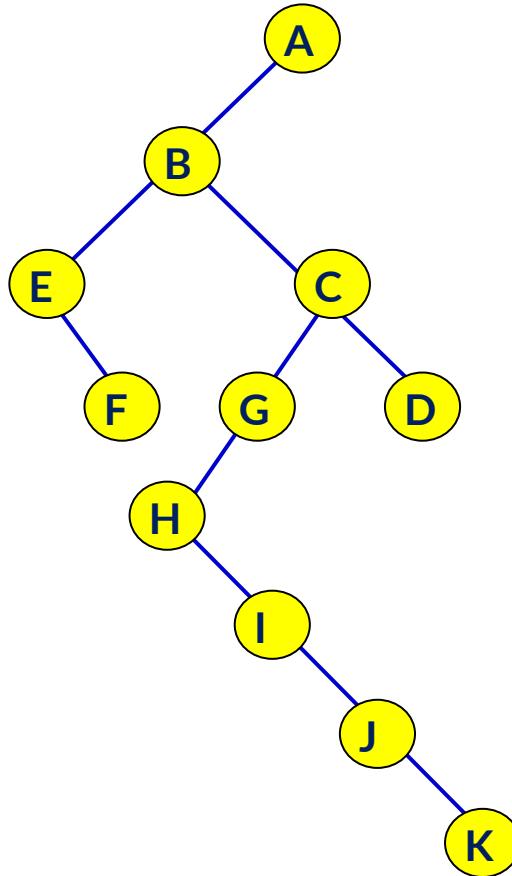
Mỗi nút của cây sẽ có con trỏ đến con trái và con trỏ đến con phải:



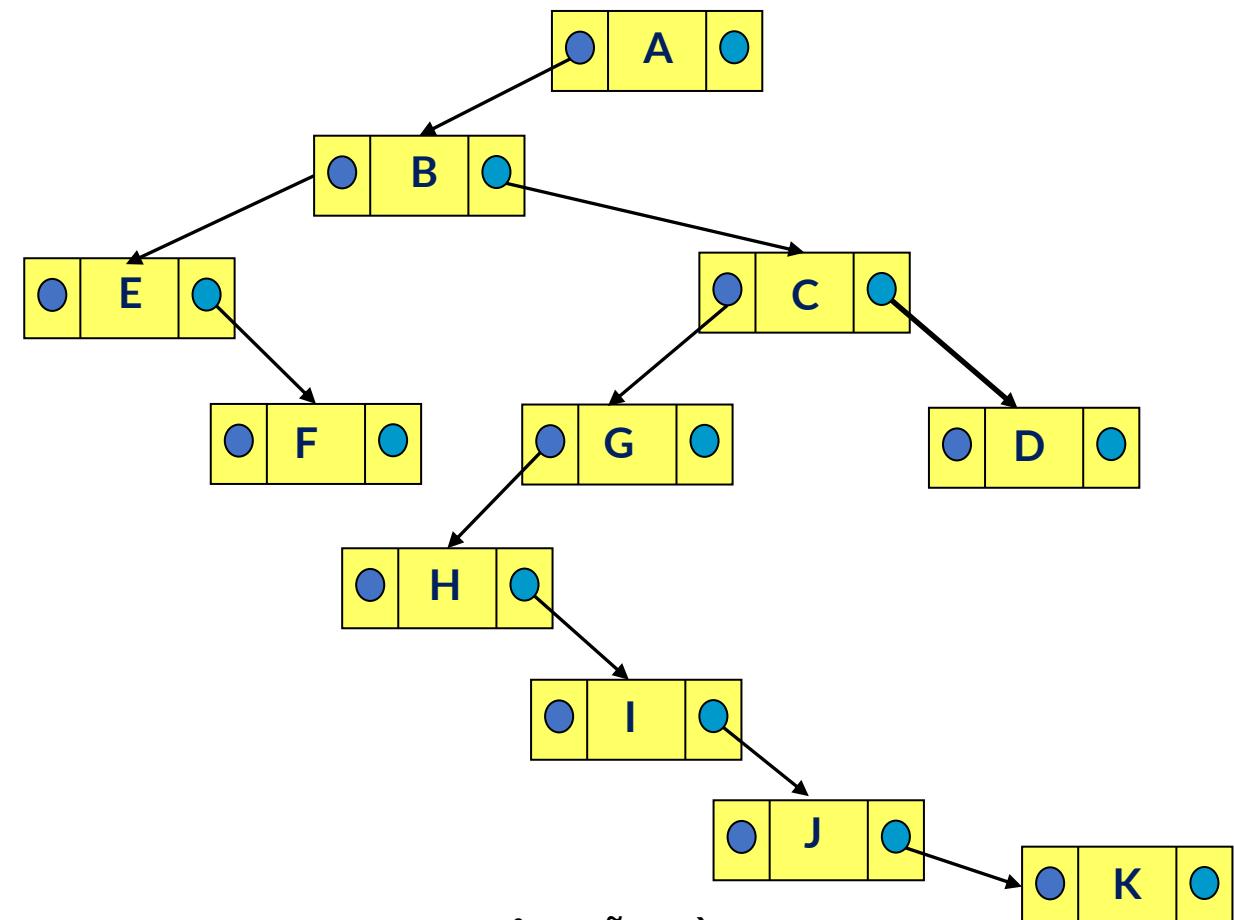
```
struct Tnode {  
    DataType Item; // DataType - kiểu dữ liệu của phần tử  
  
    struct Tnode *left;  
    struct Tnode *right;  
};  
typedef struct Tnode treeNode;
```

3. CÂY NHỊ PHÂN

- 3.1. Biểu diễn cây nhị phân
 - Biểu diễn sử dụng con trỏ



Cây nhị phân



Cây nhị phân biểu diễn bằng con trỏ

3. CÂY NHỊ PHÂN

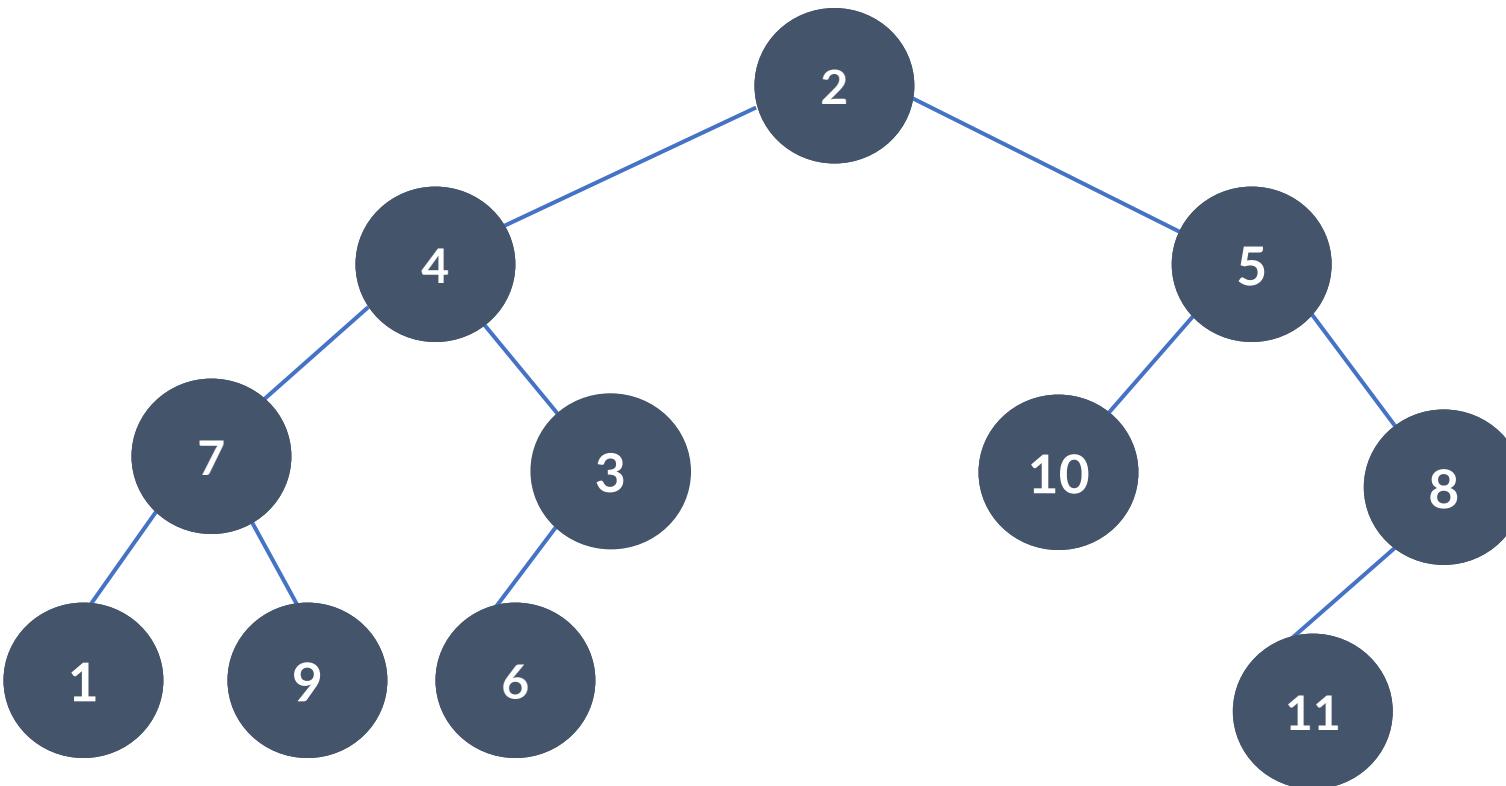
- 3.2. Duyệt cây nhị phân
 - Duyệt theo thứ tự trước

- Thăm nút
- Duyệt cây con trái
- Duyệt cây con phải

```
void printPreorder(treeNode *tree)
{
    if( tree != NULL )
    {
        printf("%s\n", tree->word);
        printPreorder(tree->left);
        printPreorder(tree->right);
    }
}
```

3. CÂY NHỊ PHÂN

- 3.2. Duyệt cây nhị phân
 - Duyệt theo thứ tự trước



2, 4, 7, 1, 9, 3, 6, 5, 10, 8, 11

3. CÂY NHỊ PHÂN

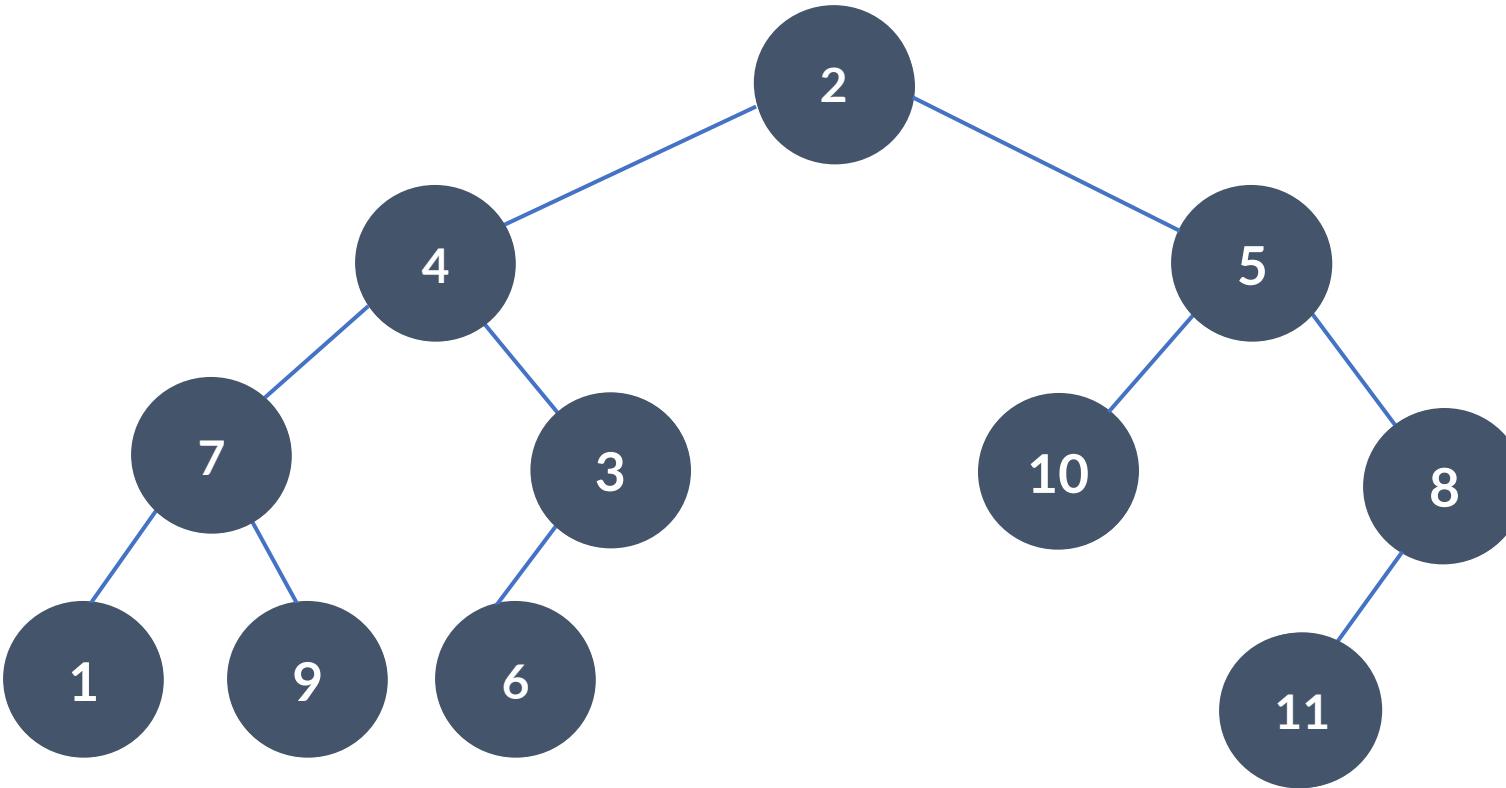
- 3.2. Duyệt cây nhị phân
 - Duyệt theo thứ tự giữa

- Duyệt cây con trái
- Thăm nút
- Duyệt cây con phải

```
void printInorder(treeNode *tree)
{
    if( tree != NULL )
    {
        printInorder(tree->left);
        printf("%s\n", tree->word);
        printInorder(tree->right);
    }
}
```

3. CÂY NHỊ PHÂN

- 3.2. Duyệt cây nhị phân
 - Duyệt theo thứ tự giữa



1, 7, 9, 4, 6, 3, 2, 10, 5, 11, 8

3. CÂY NHỊ PHÂN

- 3.2. Duyệt cây nhị phân

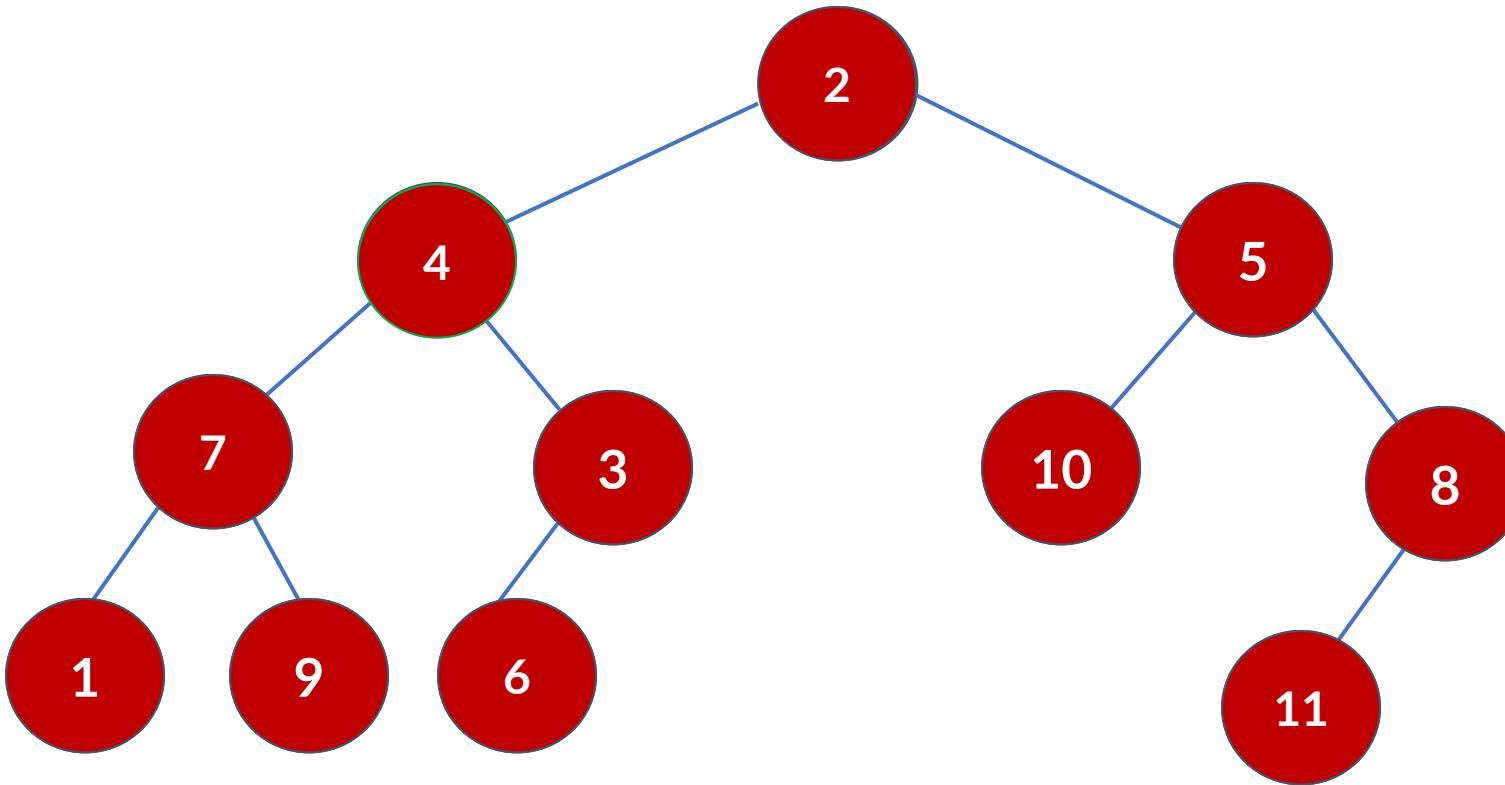
3.2.3. Duyệt theo thứ tự sau

- Duyệt cây con trái
- Thăm nút
- Duyệt cây con phải

```
void printPostorder(treeNode *tree)
{
    if( tree != NULL )
    {
        printPostorder(tree->left);
        printPostorder(tree->right);
        printf("%s\n", tree->word);
    }
}
```

3. CÂY NHỊ PHÂN

- 3.2. Duyệt cây nhị phân
 - Duyệt theo thứ tự sau



1, 9, 7, 6, 3, 4, 10, 11, 8, 5, 2



Chương 6- Cây

Bài 3. Tính chiều cao, độ sâu của một
nút trên cây

ONE LOVE. ONE FUTURE.

MỤC TIÊU

Sau bài học này, người học có thể:

1. Hiểu được khái niệm **chiều cao** và **độ sâu** của một nút trong cấu trúc dữ liệu cây
2. Cài đặt được thuật toán tìm chiều cao và độ sâu của nút trên cây

1. Định nghĩa chiều cao và độ sâu của nút

1.1. Định nghĩa

1.2. Ví dụ

2. Thuật toán tìm chiều cao và độ sâu

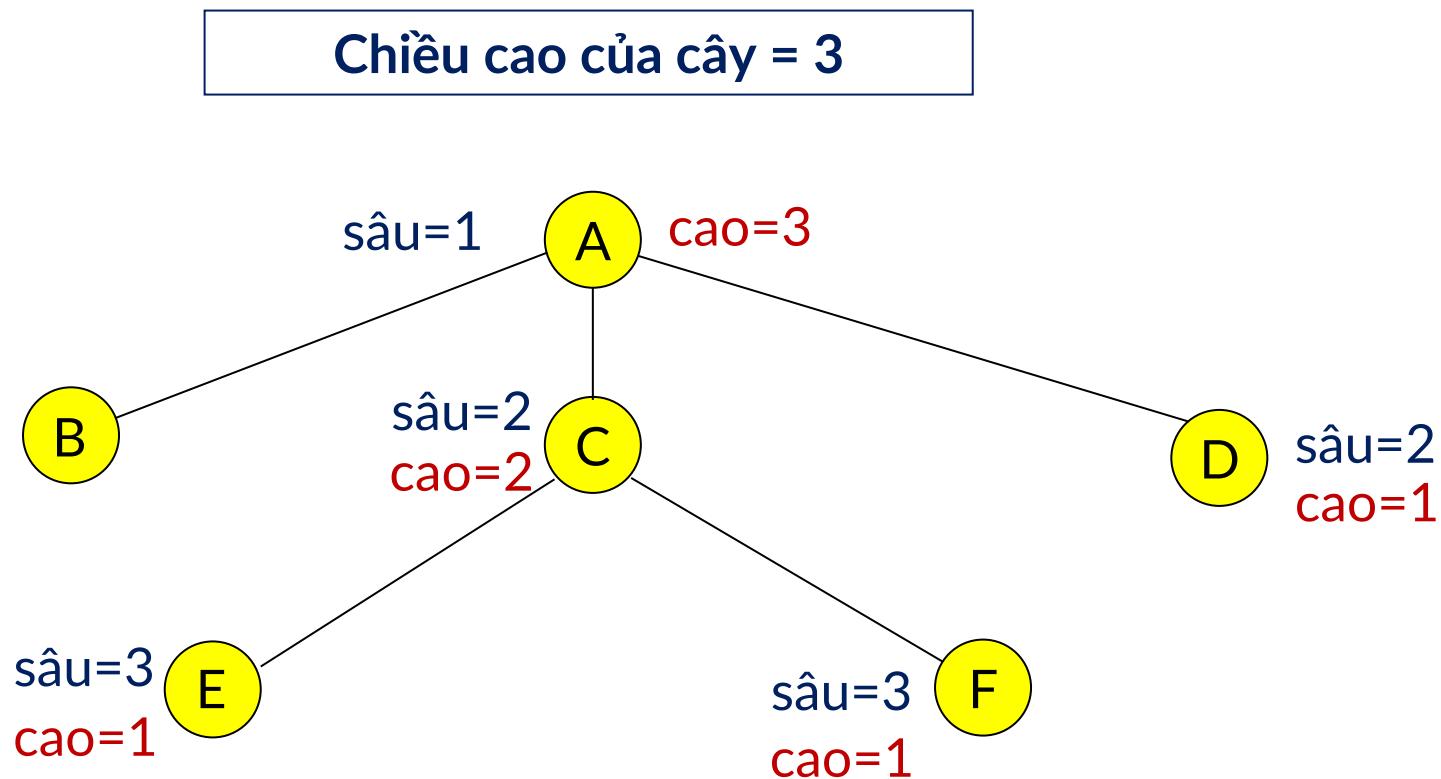
1. ĐỊNH NGHĨA CHIỀU CAO VÀ ĐỘ SÂU CỦA NÚT

- 1.1. Định nghĩa

- Chiều cao (*height*) của nút trên cây là bằng độ dài của đường đi dài nhất từ nút đó đến lá cộng 1.
 - Chiều cao của cây (*height of a tree*) là chiều cao của gốc.
- Độ sâu/mức (*depth/level*) của nút là bằng 1 cộng với độ dài của đường đi duy nhất từ gốc đến nó.

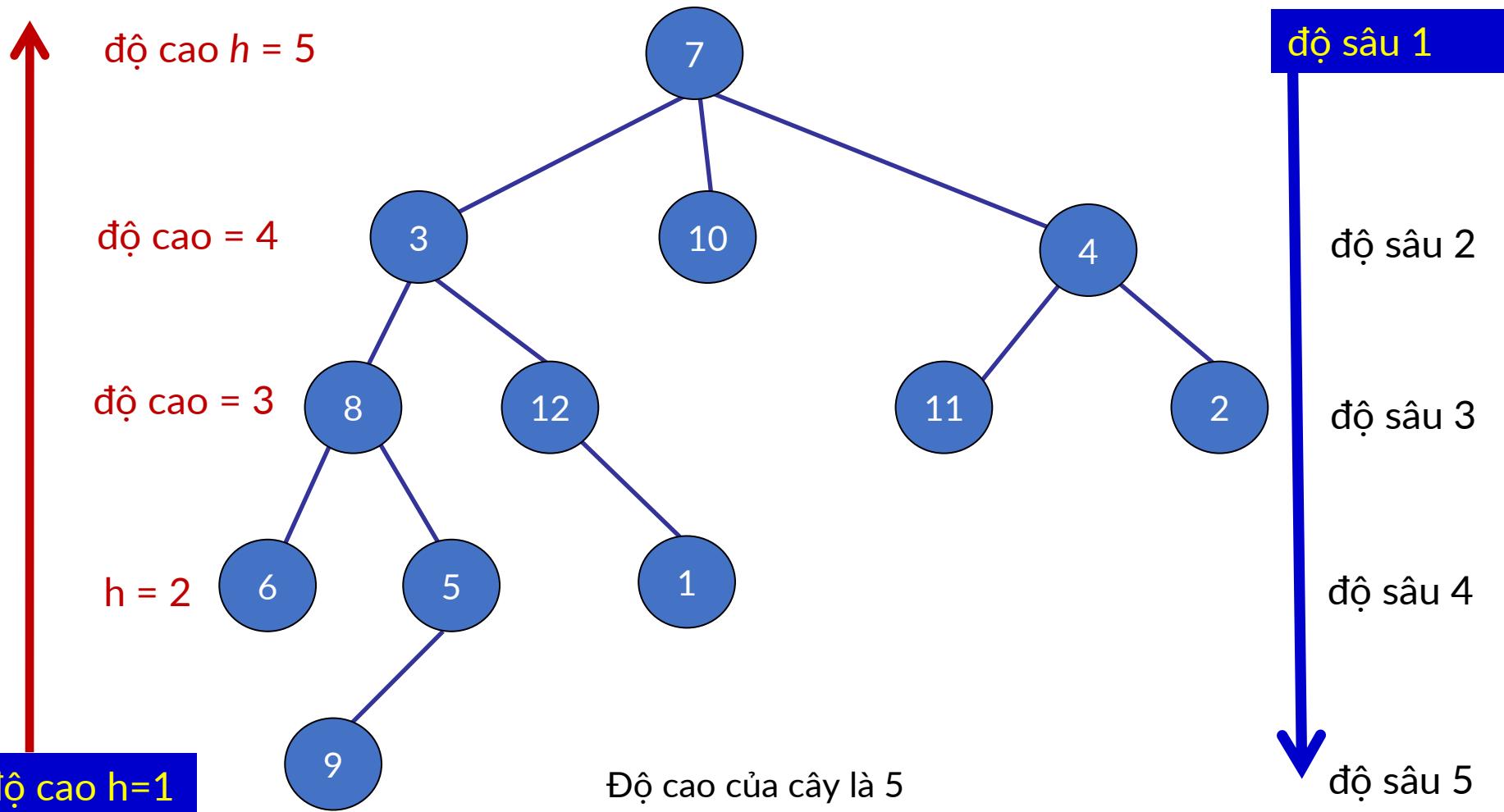
1. ĐỊNH NGHĨA CHIỀU CAO VÀ ĐỘ SÂU CỦA NÚT

- 1.2. Ví dụ



1. ĐỊNH NGHĨA CHIỀU CAO VÀ ĐỘ SÂU CỦA NÚT

• 1.2. Ví dụ



NỘI DUNG TIẾP THEO

1. Định nghĩa chiều cao và độ sâu của nút

2. Thuật toán tìm chiều cao và độ sâu

2.1 Tìm độ cao của một nút

2.2 Tìm độ sâu của một nút

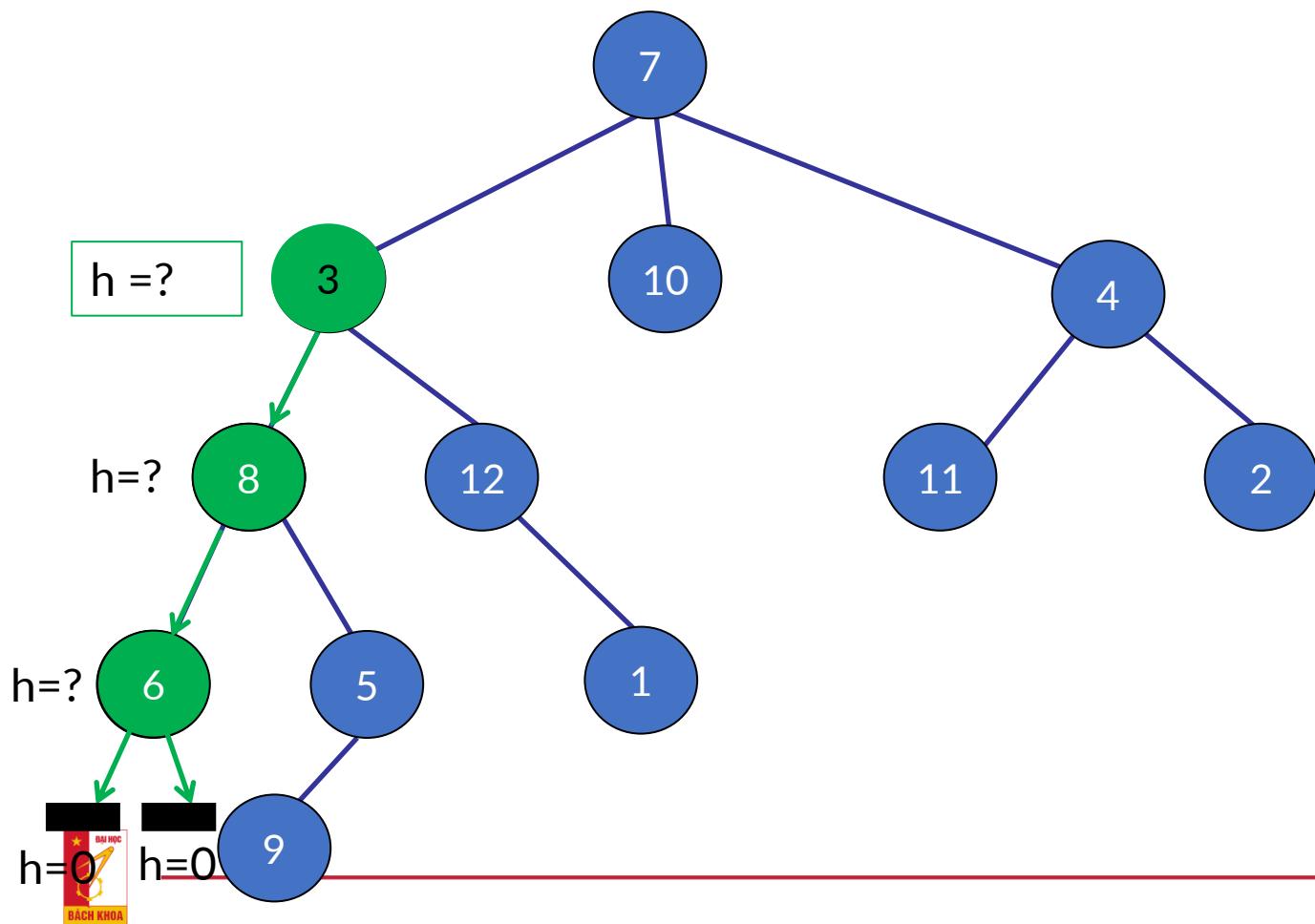
2. THUẬT TOÁN TÌM CHIỀU CAO VÀ ĐỘ SÂU

- Cấu trúc dữ liệu biểu diễn cây

```
struct Node{  
    int id;  
    Node* leftMostChild;  
    Node* rightSibling;  
};  
Node* root;
```

2. THUẬT TOÁN TÌM CHIỀU CAO VÀ ĐỘ SÂU

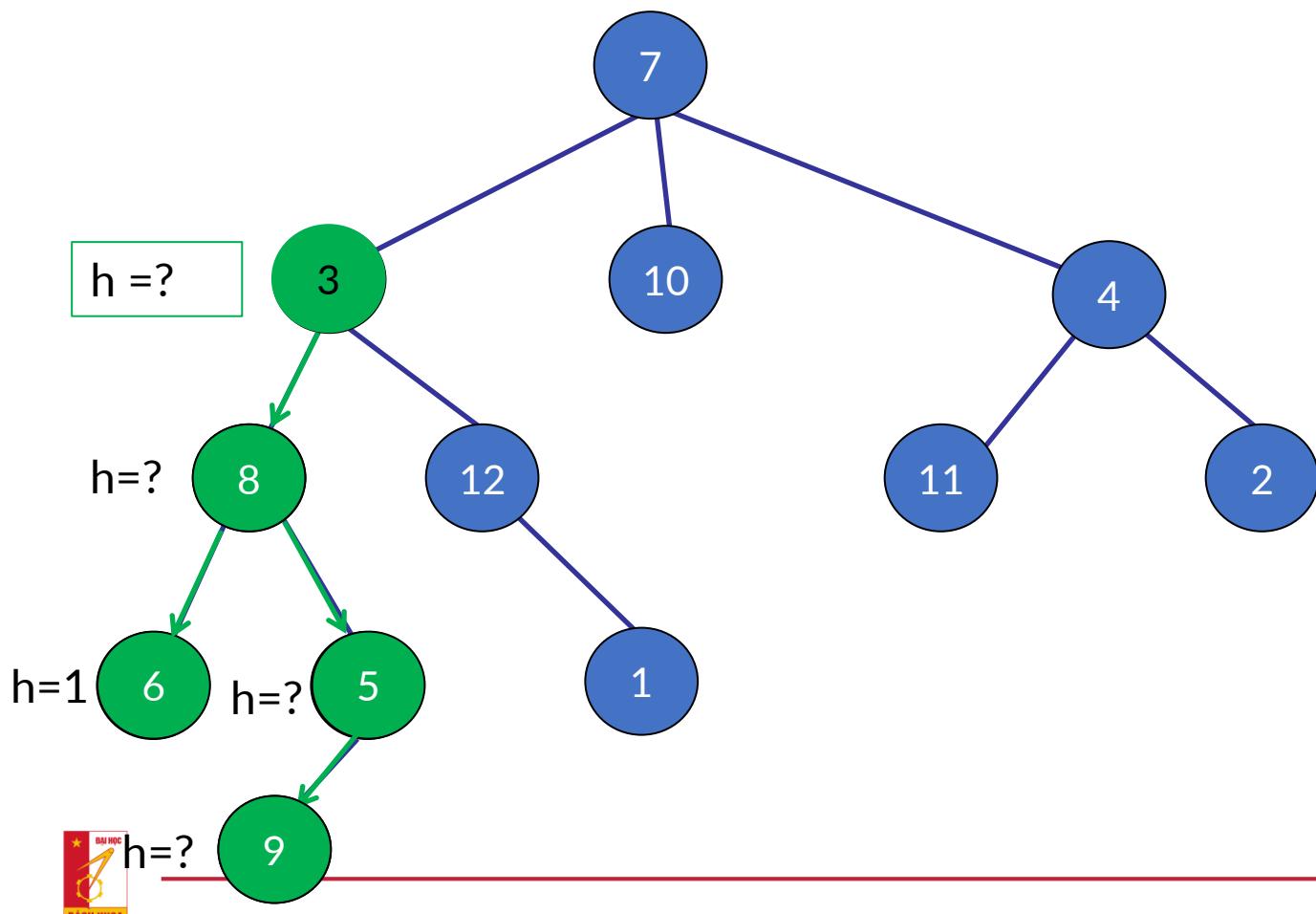
- 2.1. Tìm độ cao của một nút
 - Độ cao của một nút



```
int height(Node* p){  
    if(p == NULL) return 0;  
    int maxh = 0;  
    Node* q = p->leftMostChild;  
    while(q != NULL){  
        int h = height(q);  
        if(h > maxh) maxh = h;  
        q = q->rightSibling;  
    }  
    return maxh + 1;  
}
```

2. THUẬT TOÁN TÌM CHIỀU CAO VÀ ĐỘ SÂU

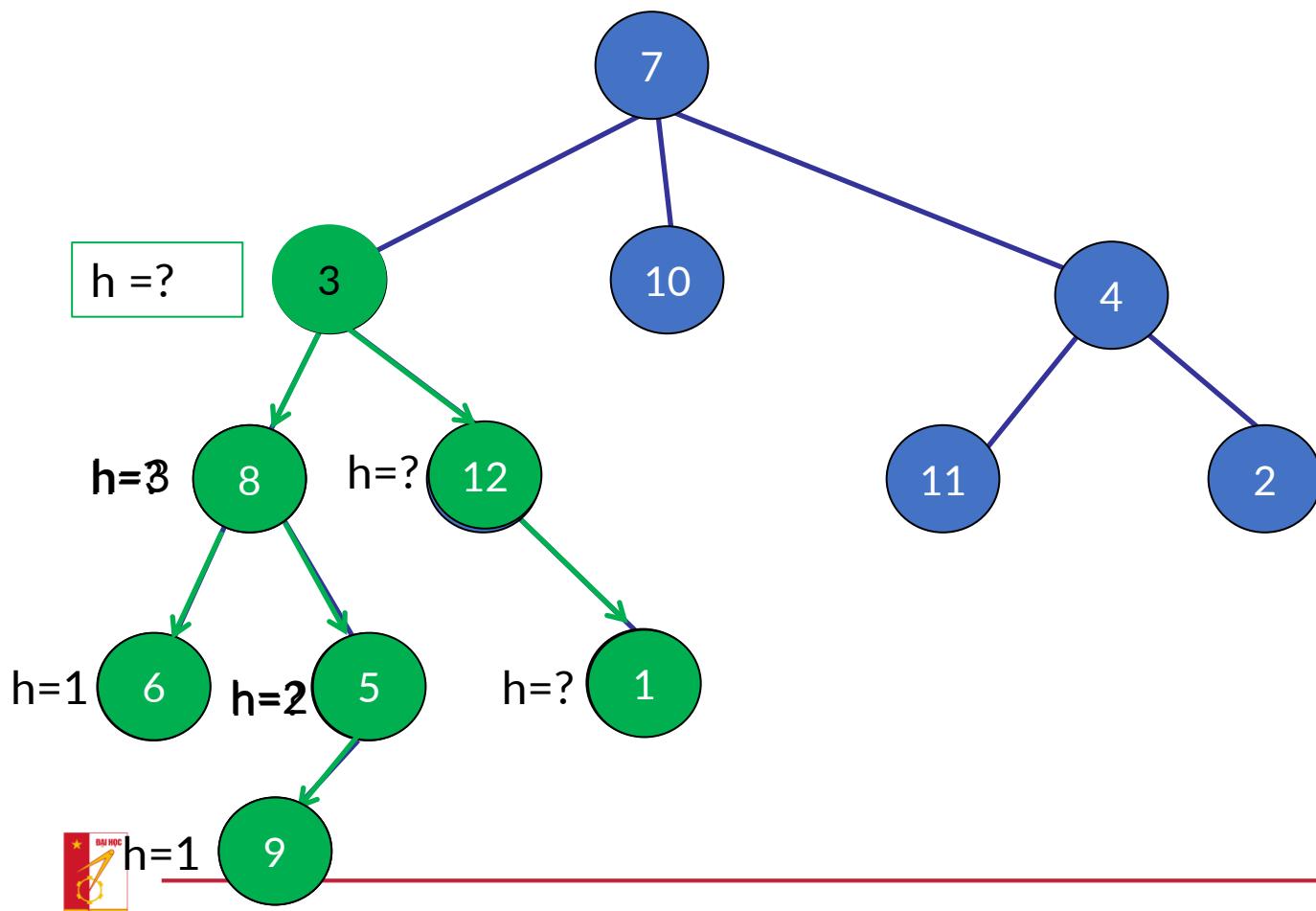
- 2.1. Tìm độ cao của một nút
 - Độ cao của một nút



```
int height(Node* p){  
    if(p == NULL) return 0;  
    int maxh = 0;  
    Node* q = p->leftMostChild;  
    while(q != NULL){  
        int h = height(q);  
        if(h > maxh) maxh = h;  
        q = q->rightSibling;  
    }  
    return maxh + 1;  
}
```

2. THUẬT TOÁN TÌM CHIỀU CAO VÀ ĐỘ SÂU

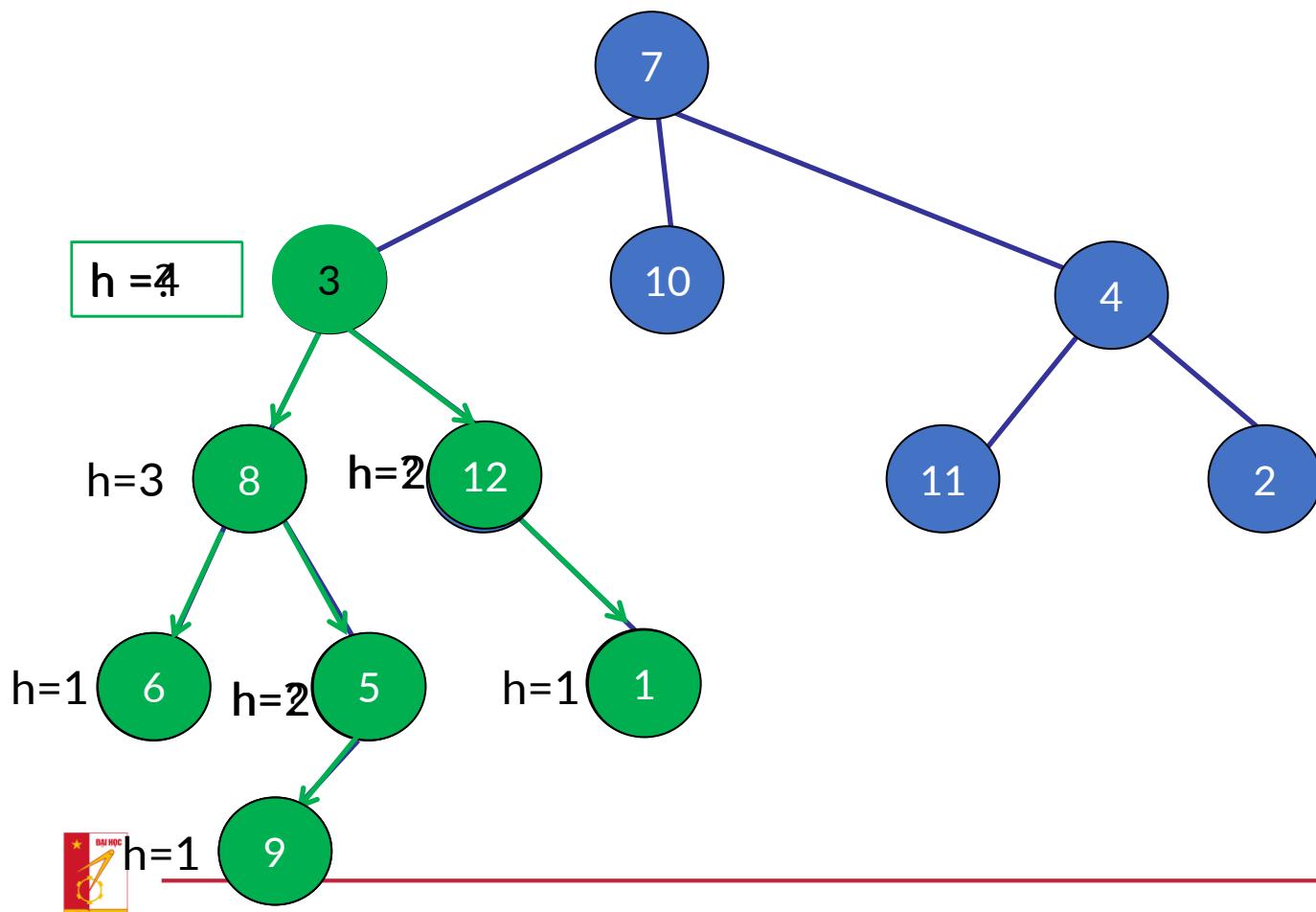
- 2.1. Tìm độ cao của một nút
 - Độ cao của một nút



```
int height(Node* p){  
    if(p == NULL) return 0;  
    int maxh = 0;  
    Node* q = p->leftMostChild;  
    while(q != NULL){  
        int h = height(q);  
        if(h > maxh) maxh = h;  
        q = q->rightSibling;  
    }  
    return maxh + 1;  
}
```

2. THUẬT TOÁN TÌM CHIỀU CAO VÀ ĐỘ SÂU

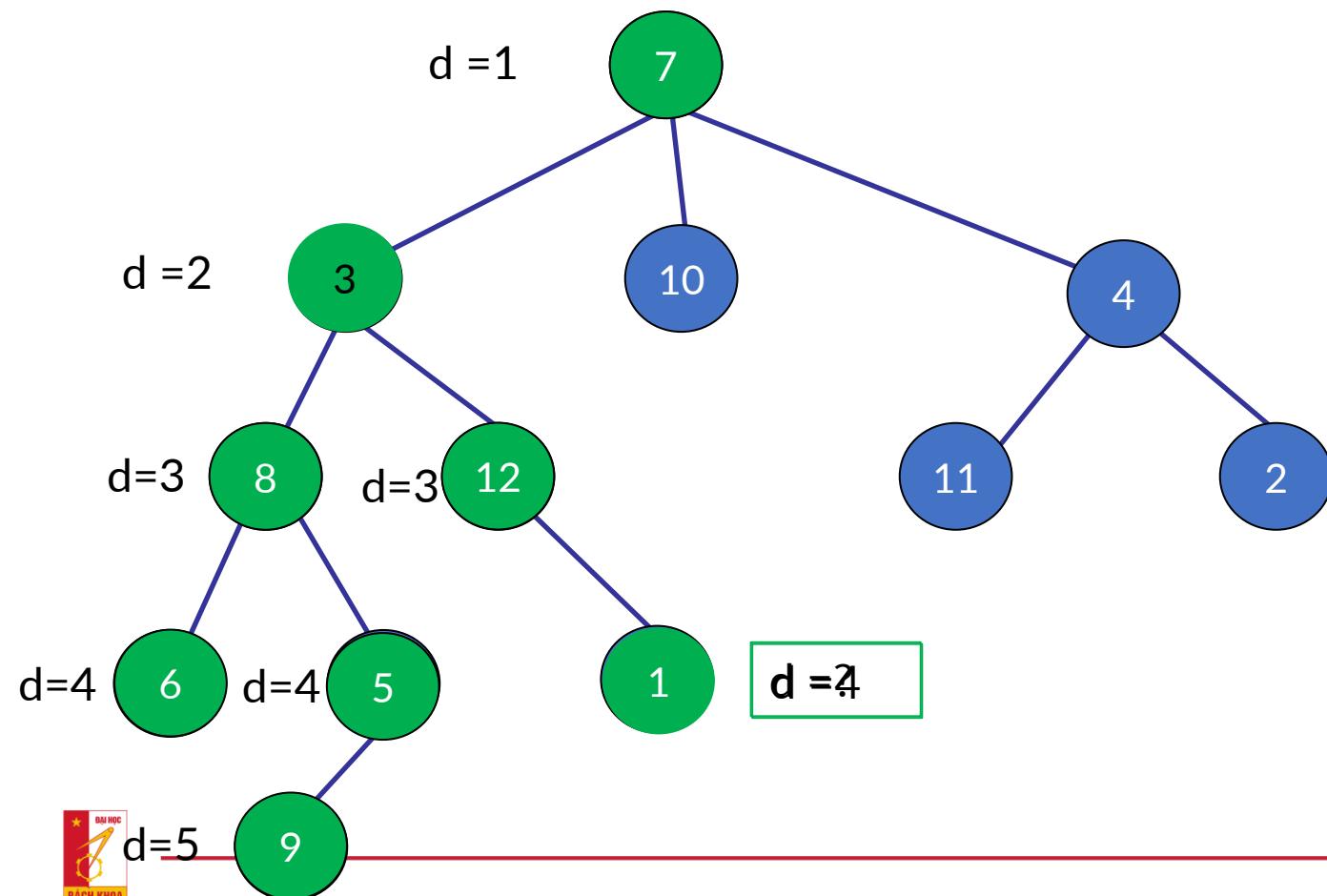
- 2.1. Tìm độ cao của một nút
 - Độ cao của một nút



```
int height(Node* p){  
    if(p == NULL) return 0;  
    int maxh = 0;  
    Node* q = p->leftMostChild;  
    while(q != NULL){  
        int h = height(q);  
        if(h > maxh) maxh = h;  
        q = q->rightSibling;  
    }  
    return maxh + 1;  
}
```

2. THUẬT TOÁN TÌM CHIỀU CAO VÀ ĐỘ SÂU

- 2.2. Tìm độ sâu của một nút
 - Độ sâu của một nút



```
int depth(Node* r, int v, int d){  
    // d la do sau cua nut r  
    if(r == NULL) return -1;  
    if(r->id == v) return d;  
    Node* p = r->leftMostChild;  
    while(p != NULL){  
        if(p->id == v) return d+1;  
        int dv = depth(p,v,d+1);  
        if(dv > 0) return dv;  
        p = p->rightSibling;  
    }  
    return -1;  
}  
int find_depth(Node* r, int v){  
    return depth(r,v,1);  
}
```



HUST

THANK YOU !