

Data structure and Algorithms

Graph – Đồ thị

Thanh-Hai Tran

Electronics and Computer Engineering
School of Electronics and Telecommunications

Hanoi University of Science and Technology
1 Dai Co Viet - Hanoi - Vietnam

Nội dung bài học

- **Các khái niệm cơ bản**
- **Biểu diễn đồ thị**
 - ◆ Ma trận lân cận
 - ◆ Danh sách lân cận
- **Duyệt đồ thị**
- **Bài toán áp dụng**
 - ◆ Tìm cây khung cực tiểu
 - ◆ Tìm đường đi ngắn nhất
- **Bài tập**
- **Tài liệu tham khảo**

Giới thiệu

- **Đồ thị (graph):** là một cấu trúc dữ liệu trừu tượng (abstract) được sử dụng để cài đặt đồ thị trong toán học
- **Đồ thị:** có thể coi như khái quát của cấu trúc cây, trong đó chỉ có quan hệ giữa cha và con
- **Tại sao cấu trúc đồ thị lại hữu ích:** Graph được sử dụng để mô hình hóa bất kỳ tình huống nào trong đó các thực thể hoặc đối tượng có quan hệ từng cặp với nhau

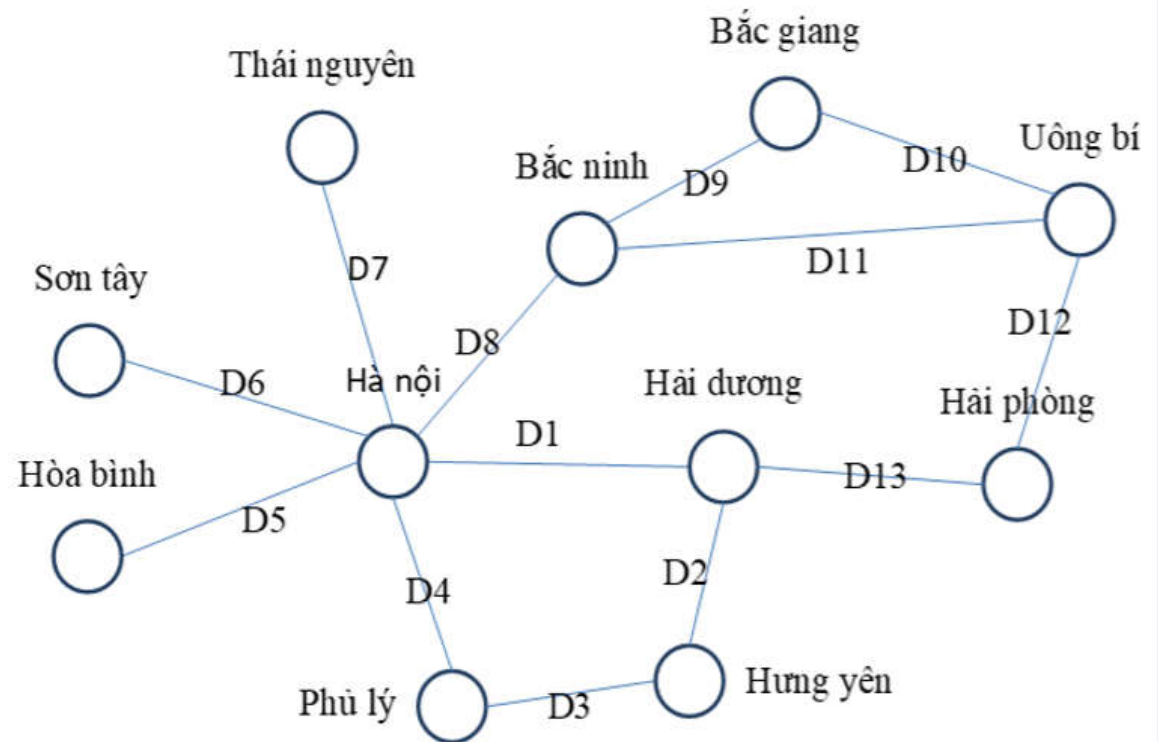
Các khái niệm cơ bản

■ Một đồ thị $G = (V, E)$

- ◆ V : tập các đỉnh (vertices)
- ◆ E : tập các cung (edges) nối các đỉnh trong V
- ◆ Một cung $e = (u, v)$ là một cặp đỉnh

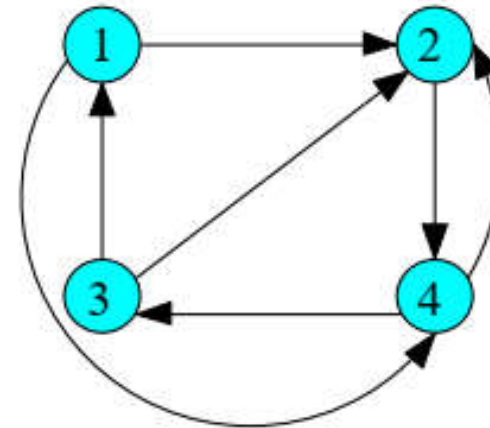
■ Ví dụ:

- ◆ Bản đồ giao thông là một đồ thị, trong đó các thành phố biểu diễn các đỉnh, còn các đường giao thông nối giữa chúng biểu diễn các cạnh



Các khái niệm liên quan

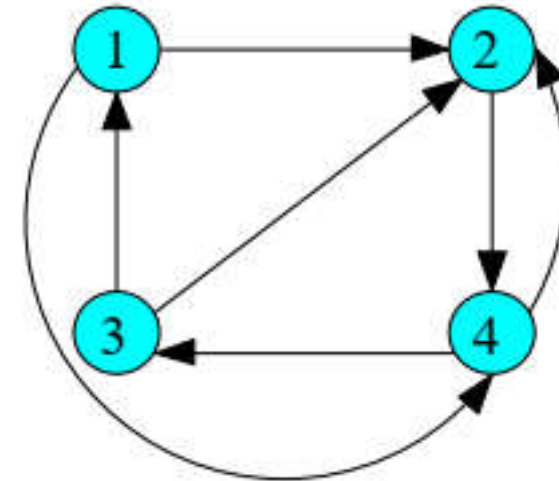
- **Bậc của một đỉnh (degree):** là số cung kề với đỉnh
- **Nếu là đồ thị có hướng:** một đỉnh có thể có
 - ◆ Bậc trong (in-degree)
 - ◆ Bậc ngoài (out-degree)



- **Ví dụ:**
 - ◆ Đỉnh 1 có bậc 3
 - ◆ Đỉnh 1 có bậc trong là 1 và bậc ngoài là 2
- **Một đỉnh có bậc bằng 0:** nghĩa là không có cạnh nào nối đến nó (đứng biệt lập)

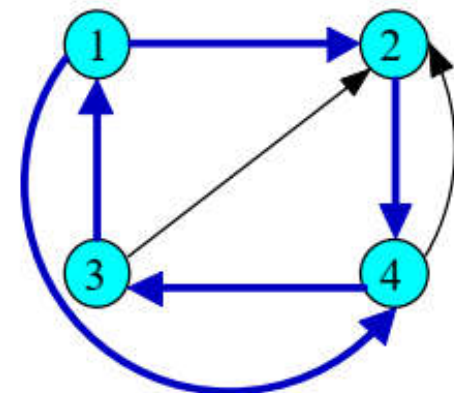
Các khái niệm liên quan

- **Đỉnh lân cận (Adjacent vertices) hay hàng xóm (neighbours):** với mỗi cạnh $e = (u, v)$, hai đỉnh u và v là kề cận và là hàng xóm của nhau
- **Trong đồ thị:**
 - ◆ 1, 2 là lân cận của nhau
 - ◆ 1, 3 là lân cận của nhau
 - ◆
- **Cung kề (Incident edges):**
Nếu có cung (u, v) thì cung này là cung kề của hai đỉnh u và v



Các khái niệm liên quan

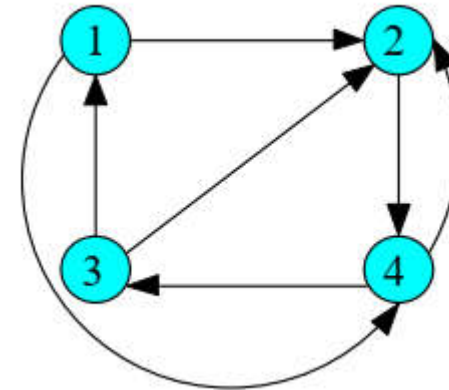
- **Đường đi:** Dãy các đỉnh v_1, v_2, \dots, v_k mà tồn tại cung (v_i, v_{i+1}) trong đồ thị ($i = 1 \dots k-1$)
- **Đường đi đơn:** Đường đi với các đỉnh không lặp lại
- **Chu trình:** Đường đi đơn với đỉnh đầu và cuối trùng nhau
- **Độ dài đường đi:** số cung trên đường đi
- **Kích thước:** là số cạnh trên đồ thị



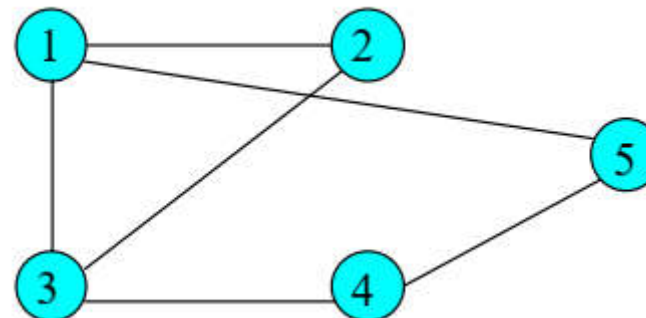
Path : 1, 2, 4, 3, 1, 4

Các loại đồ thị

- **Đồ thị có hướng (directed graph):** Trong một cung, thứ tự của các đỉnh là quan trọng. Cung (u,v) khác với cung (v,u) .

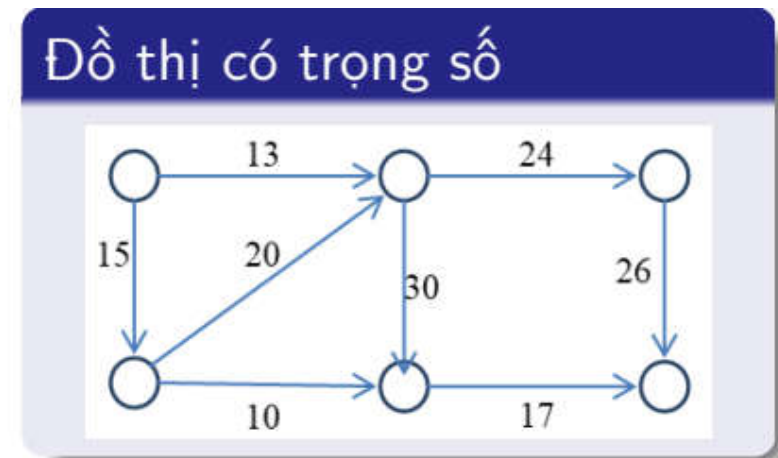
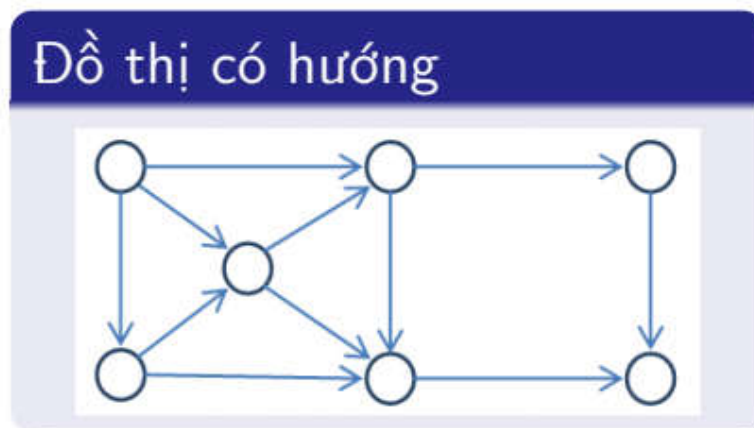


- **Đồ thị vô hướng (undirected graph):** Trong một cung, thứ tự của các đỉnh là không quan trọng. Cung (u,v) cũng giống như cung (v,u) .



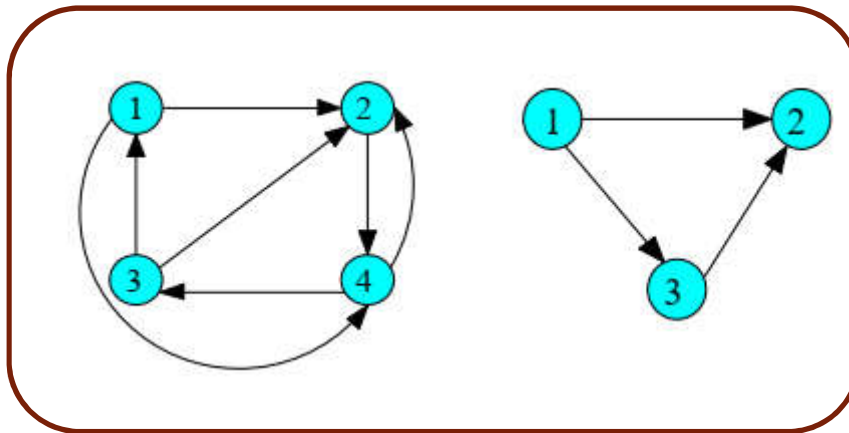
Các loại đồ thị

- **Đồ thị có trọng số (weighted graph):** Là đồ thị mà mỗi cạnh mang một giá trị nào đó mà được gọi là trọng số của cạnh đó
- Ngược lại, đồ thị không có trọng số nếu các cạnh không mang trọng số nào.

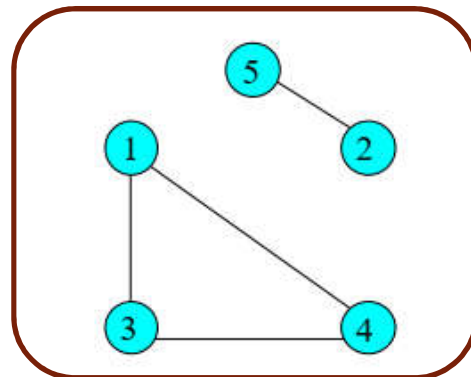


Các loại đồ thị

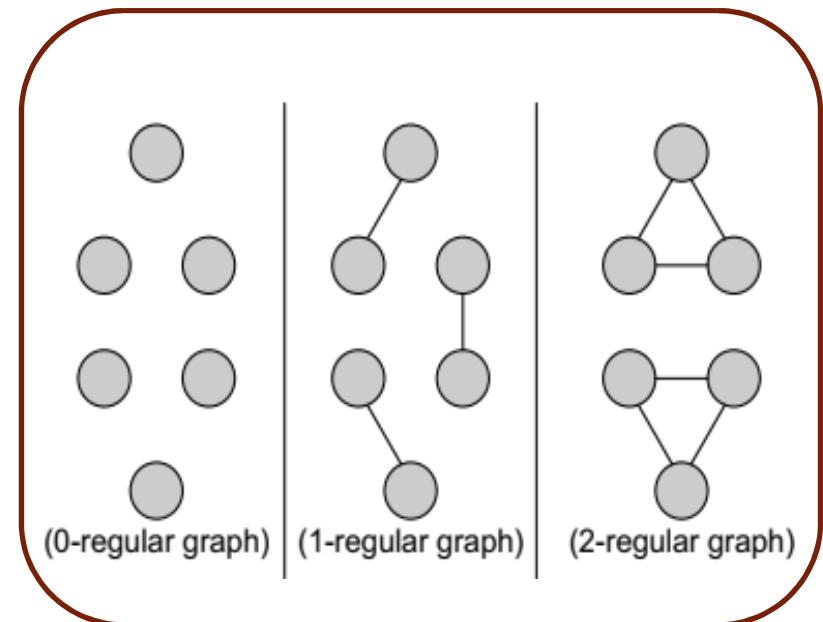
- **Đồ thị liên thông (connected graph)**



- **Đồ thị không liên thông)**



- **Regular graph:** mỗi đỉnh có số hàng xóm bằng nhau (độ của các đỉnh là giống nhau)



Các loại đồ thị

- **Complete graph:** tất cả các nút đều nối với nhau. Nghĩa là luôn có một đường đi từ một nút đến một nút bất kỳ trong đồ thị
- Với số nút là n , số cạnh của đồ thị là $n(n-1)/2$

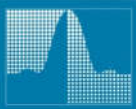
Kiểu dữ liệu trừu tượng đồ thị

■ Dữ liệu:

- ◆ Một tập không rỗng các đỉnh chứa các phần tử có kiểu nhất định,
- ◆ Một tập không rỗng các cung có thể biểu diễn các phần tử có kiểu nhất định

■ Các thao tác cơ bản:

- | | |
|-----------------------|-----------------------|
| ◆ Graph create() | ◆ areAdjacent(v, w) |
| ◆ insertVertex(o) | ◆ adjacentVertices(v) |
| ◆ insertEdge(u, v, o) | ◆ incidentEdges(v) |
| ◆ removeVertex(v) | ◆ vertices() |
| ◆ removeEdge(e) | ◆ edges() |
| ◆ endVertices(e) | ◆ numVertices() |
| ◆ opposite(v, e) | ◆ numEdges() |



Các thao tác cơ bản trên đồ thị

- **Duyệt đồ thị:** tương tự như duyệt cây, phép duyệt đồ thị cũng muốn truy cập vào tất cả các đỉnh của đồ thị, mỗi đỉnh đúng một lần.
- **Có hai phương pháp duyệt đồ thị:**
 - ◆ Duyệt theo chiều sâu
 - ◆ Duyệt theo chiều rộng

Các thao tác cơ bản trên đồ thị

- **Tìm kiếm một đỉnh hoặc một cạnh:** là thao tác muốn tìm một đỉnh hay một cạnh thỏa mãn một điều kiện nào đó,
- **Ví dụ:** tìm một đỉnh có giá trị cho trước, hay tìm cạnh có trọng số cho trước
- **Các giải thuật tìm kiếm thường dựa trên phương pháp duyệt đồ thị**

Các thao tác cơ bản trên đồ thị

- **Tìm đường đi ngắn nhất:**
 - ◆ Tìm đường đi ngắn nhất giữa hai đỉnh trên một đồ thị
 - ◆ Tìm đường đi ngắn nhất giữa một đỉnh và tất cả các đỉnh còn lại

Một số tính chất của đồ thị

- Nếu một đồ thị G có m cung thì tổng bậc của các đỉnh trong G sẽ là $2m$
- Nếu một đồ thị có hướng có m cung thì tổng bậc trong của các đỉnh, tổng bậc ngoài của các đỉnh đều là m
- Nếu đồ thị G là đồ thị đơn giản, G có n đỉnh và m cung thì
 - ◆ Nếu G là đồ thị vô hướng $m \leq n(n-1)/2$
 - ◆ Nếu G là đồ thị có hướng thì $m \leq n(n-1)$

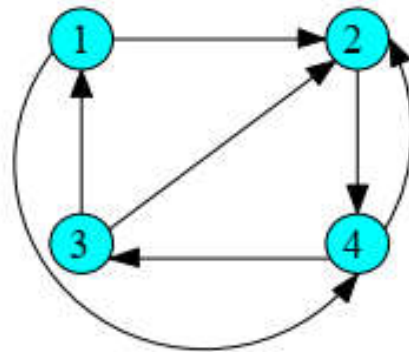
Nội dung bài học

- Các khái niệm cơ bản
- **Biểu diễn đồ thị**
 - ◆ Sử dụng cấu trúc lưu trữ tuần tự: Ma trận lân cận
 - ◆ Sử dụng cấu trúc lưu trữ móc nối: Danh sách lân cận
- Duyệt đồ thị
- Bài toán áp dụng
 - ◆ Tìm cây khung cực tiểu
 - ◆ Tìm đường đi ngắn nhất
- Bài tập
- Tài liệu tham khảo

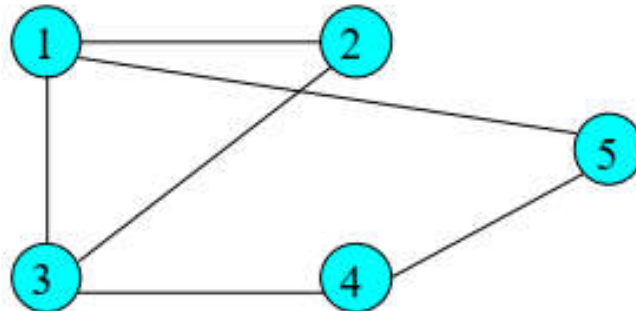
Biểu diễn đồ thị - ma trận kề

- **Biểu diễn bằng ma trận lân cận:**
 - ◆ Đánh số các đỉnh trong tập V từ 1 đến n
 - ◆ Ma trận biểu diễn đồ thị A ($n \times n$)
 - ★ $A_{ij} = 1$ nếu trong G tồn tại cung (i,j)
 - ★ $A_{ij} = 0$ nếu trong G không tồn tại cung đó
 - ◆ Với đồ thị vô hướng thì nếu $A_{ij} = 1$ thì $A_{ji} = 1$
 - ◆ A được gọi là ma trận lân cận của G

Biểu diễn đồ thị - ma trận cận kề



$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

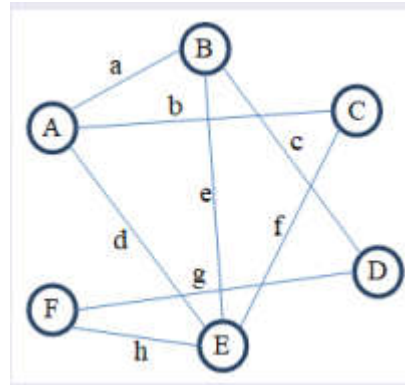


$$\begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Biểu diễn đồ thị - ma trận cận kề

- Cài đặt một đồ thị $G(V,E)$: $V = (v_0, v_1, \dots, v_{M-1})$, số cạnh là $E = (e_0, e_1, \dots, e_{N-1})$,
- Sử dụng 2 ma trận:
 - ◆ Ma trận $AV[M]$: lưu số đỉnh của đồ thị
 - ◆ Để lưu quan hệ giữa các cạnh:
 - ★ Sử dụng ma trận đỉnh cận kề: sử dụng mảng hai chiều $AE[M \times M]$ để lưu trữ các cạnh
 - ★ Sử dụng ma trận cạnh kề: cần hai cấu trúc mảng
 - $E[N]$: lưu các cạnh
 - $AE[M \times N]$: mảng logic để lưu trạng thái các cạnh kề

Biểu diễn đồ thị - ma trận cận kề



Ma trận đỉnh kề

	A	B	C	D	E	F
A		T	T		T	
B	T			T	T	
C	T				T	
D		T				T
E	T	T	T			T
F				T	T	

Ma trận cạnh kề

	a	b	c	d	e	f	g	h
A	T	T		T				
B	T		T		T			
C		T				T		
D			T				T	
E				T	T	T		T
F							T	T

Biểu diễn đồ thị - danh sách móc nối

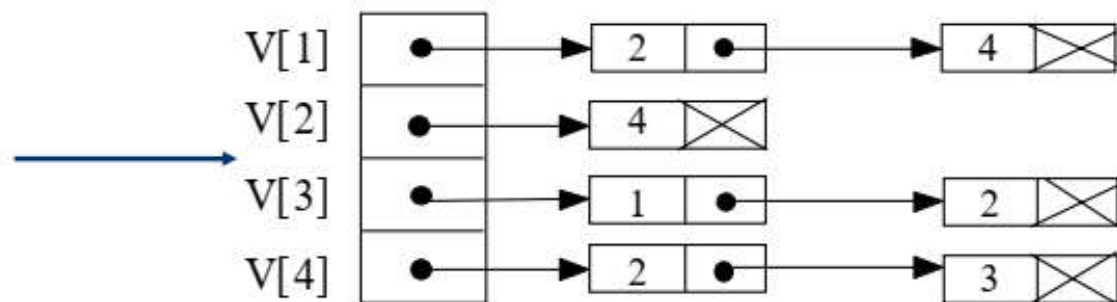
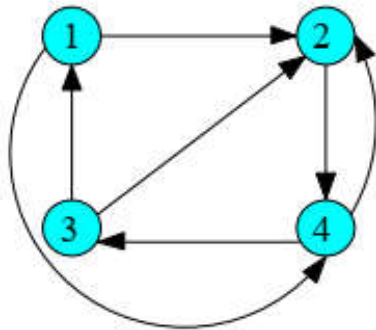
- Mỗi đỉnh trong đồ thị sẽ ứng với một danh sách móc nối chứa các đỉnh lân cận của nó
- Mỗi nút trong danh sách có quy cách

VERTEX	LINK
--------	------

- ◆ VERTEX chứa giá trị tương ứng với số thứ tự của đỉnh lân cận
- ◆ LINK chứa con trỏ trỏ tới nút tiếp theo trong danh sách
- Mỗi danh sách như vậy có một nút đầu danh sách
- Các nút đầu này là các phần tử của một vector V có kích thước n . Phần tử $V[i]$ ứng với danh sách lân cận của nút thứ i

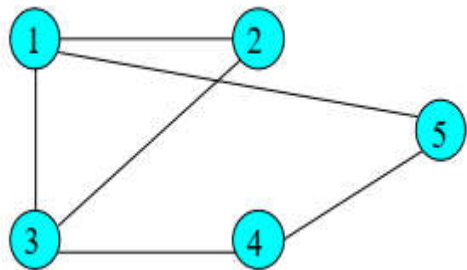
Biểu diễn đồ thị - danh sách móc nối

• Ví dụ

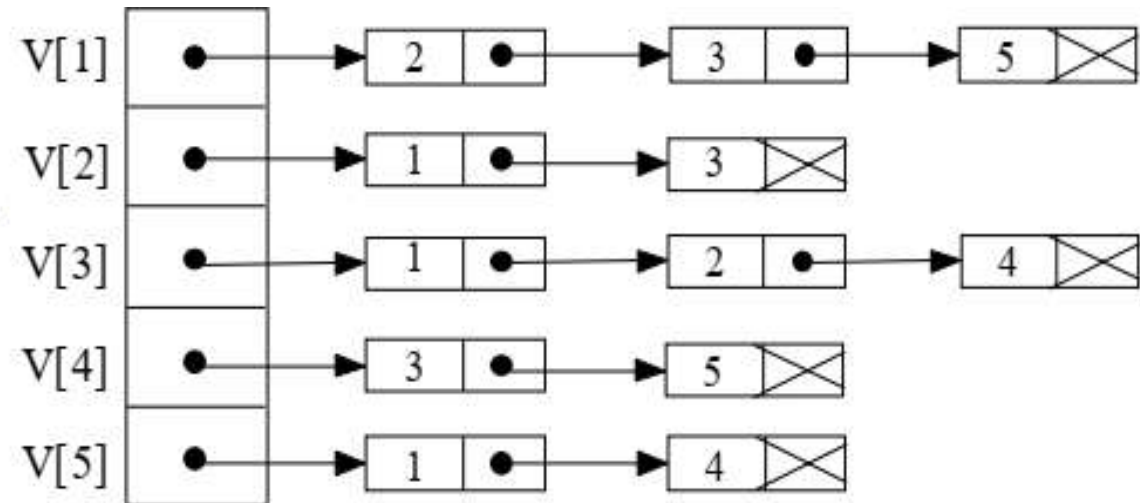


- 1: (2,4)
- 2: (4)
- 3: (1, 2)
- 4: (2, 3)

Biểu diễn đồ thị - danh sách móc nối



1: (2,3,5)
2: (1,3)
3: (1,2,4)
4: (3,5)
5: (1,4)








Nội dung bài học

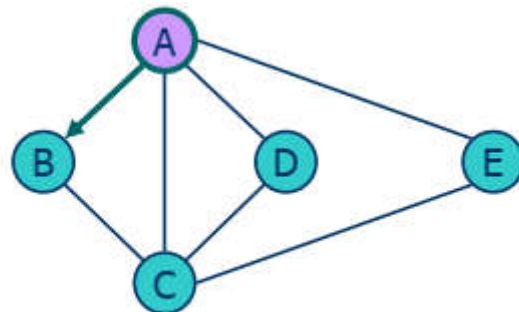
- Các khái niệm cơ bản
- Biểu diễn đồ thị
 - ◆ Sử dụng cấu trúc lưu trữ tuần tự: Ma trận lân cận
 - ◆ Sử dụng cấu trúc lưu trữ móc nối: Danh sách lân cận
- **Duyệt đồ thị**
- Bài toán áp dụng
 - ◆ Tìm cây khung cực tiểu
 - ◆ Tìm đường đi ngắn nhất
- Bài tập
- Tài liệu tham khảo

Duyệt đồ thị

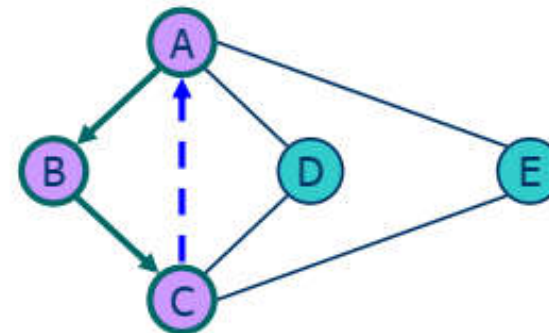
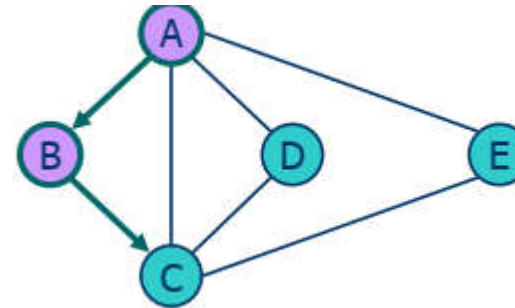
- **Duyệt:** Cho một đồ thị $G(V,E)$ và một đỉnh v thuộc V . Duyệt đồ thị là thăm mọi đỉnh liên thông với v
- **Có 2 phương pháp:**
 - ◆ Phương pháp duyệt theo chiều sâu (Depth First Search)
 - ◆ Phương pháp duyệt theo chiều rộng (Breadth First Search)

Duyệt đồ thị theo chiều sâu

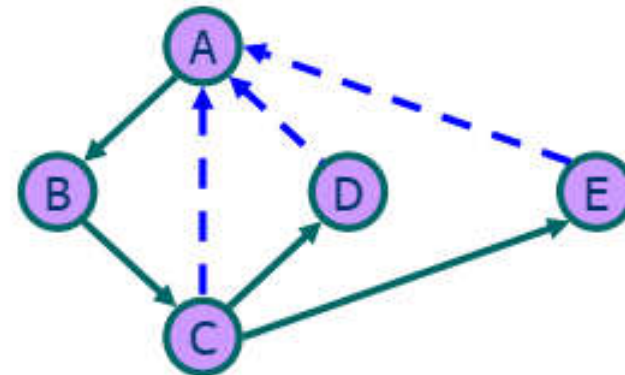
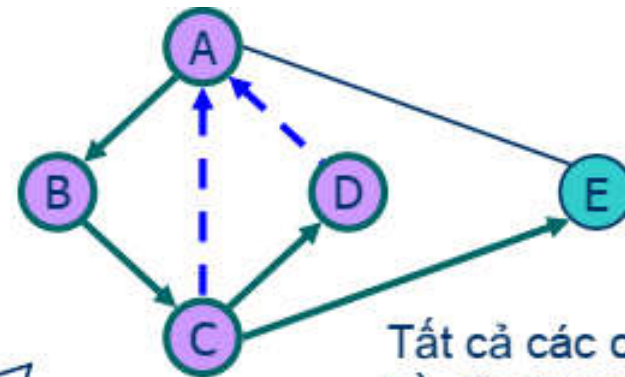
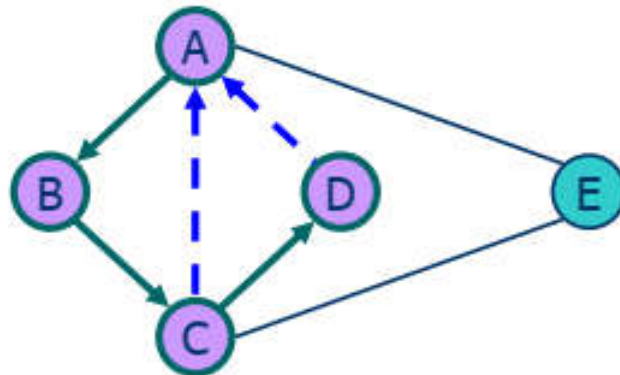
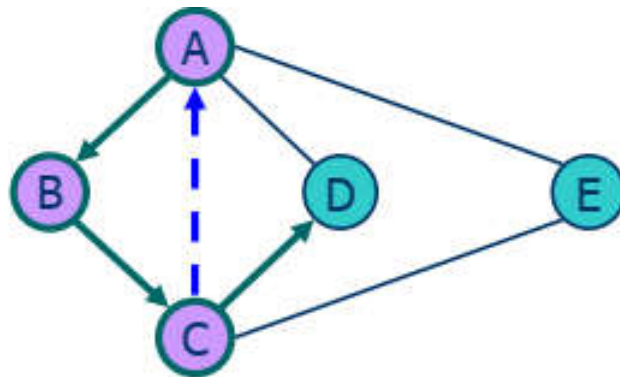
-  Đỉnh chưa thăm
-  Đỉnh đã thăm
-  Cung chưa thăm
-  Cung khám phá
-  Cung quay lui



Bắt đầu xuất phát từ A



Duyệt đồ thị theo chiều sâu



Tất cả các cung
kề của D đã duyệt
Xét tiếp các cung
kề của đỉnh C

Duyệt đồ thị theo chiều sâu

Ý tưởng giải thuật

Đây là một giải thuật đệ quy có các bước cơ bản như sau:

- 1 Đặt $v = v_s$.
- 2 Thăm nút v .
- 3 Với mỗi nút v_a là kề của v mà chưa được thăm, lặp lại bước 2 cho v_a .
- 4 (Điểm dừng): giải thuật kết thúc khi đồ thị không còn nút nào cần phải thăm nữa.

Duyệt đồ thị theo chiều sâu

Thủ tục

Gọi thủ tục cài đặt cho giải thuật là *FullDepthSearch*(*G*, *vs*), với *vs* là đỉnh bắt đầu mà thuộc tập các đỉnh *V* của đồ thị *G*.

```
void FullDepthSearch(G, vs)
```

```
{  
    v = vs;  
    DepthFirstSearch(G, v);  
}
```

DepthFirstSearch

DepthFirstSearch(G, v) là thủ tục đệ quy duyệt theo chiều sâu có dạng như sau:

```
void DepthFirstSearch( $G, v$ )
{
    if ( $v == \text{NULL}$ ) return; //Điểm dừng
    visit( $v$ );
    for each unvisited adjacent  $va$  to  $v$ 
        DepthFirstSearch( $G, va$ );
}
```

Duyệt đồ thị theo chiều sâu

- **Đồ thị có tồn tại chu trình:** cần cập nhật đúng trạng thái đã thăm / chưa thăm của từng nút trong quá trình duyệt để tránh rơi vào vòng lặp vô hạn
- **Đồ thị có các nút cô lập:** các nút này có thể bị bỏ quên khi duyệt, nên trong giải thuật cần cân nhắc để đưa thêm các nút này vào

Duyệt đồ thị theo chiều rộng

Bài toán duyệt đồ thị

Cho trước đồ thị $G(V, E)$ và nút xuất phát $v_s \in V$. Tìm cách duyệt tất cả các đỉnh của G , mỗi đỉnh đúng một lần, bắt đầu từ nút v_s .

Ý tưởng của giải thuật duyệt theo chiều rộng

- 1 Thăm nút v_s .
- 2 Tìm VA là tập các nút kề của v_s mà chưa được thăm.
- 3 Thăm lần lượt các nút trong VA .
- 4 Lặp lại giải thuật cho từng nút trong VA .

Duyệt đồ thị theo chiều rộng

- Để cài đặt giải thuật trên, ta cần một cấu trúc dữ liệu trung gian để lưu các nút kế sau này sẽ lần lượt được thăm.
- Có thể thấy danh sách kiểu hàng đợi là cấu trúc phù hợp vì các nút được lưu trước sẽ được thăm trước
- Gọi hàng đợi cần dùng là Q , khi đó gọi thủ tục duyệt theo chiều rộng là **BreathFirstSearch(G, v_s)** mà sẽ được cài đặt như sau

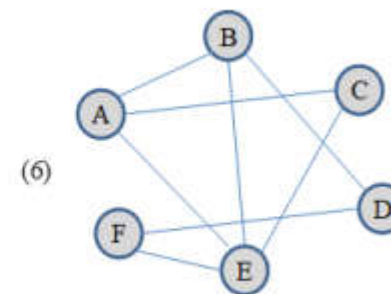
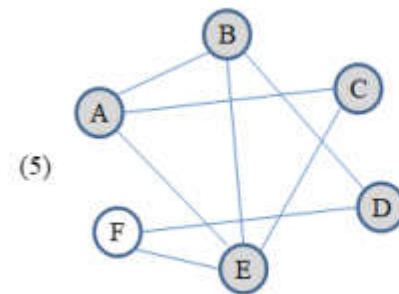
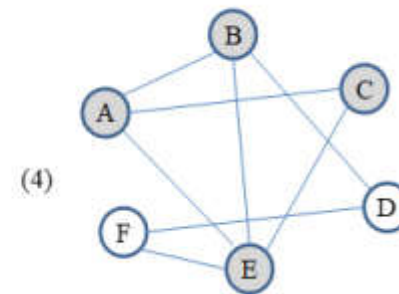
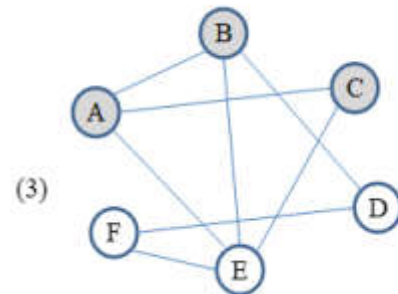
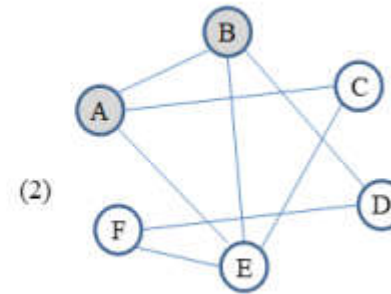
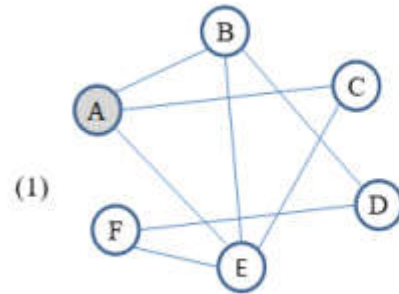
Duyệt đồ thị theo chiều rộng

```
void BreathFirstSearch(G, vs)
{
    insert(vs, Q);
    while (!isEmpty(Q))
    {
        u = remove(Q);
        visit(u);
        for each  $u_a$ : (unvisited) and
                     (adjacent to u) and ( $u_a \notin Q$ )
            insert( $u_a$ , Q);
    }
}
```

Duyệt đồ thị theo chiều rộng

- Độ phức tạp của thủ tục trên phụ thuộc vào thao tác tìm kiếm nút kề cận u_a
- Có một kỹ thuật để giảm độ phức tạp là bổ sung thêm một trạng thái nữa cho các nút, thay vì chỉ có hai trạng thái là **được thăm / chưa thăm**
- Ba trạng thái này được biểu diễn bởi các màu
 - ◆ Nút trắng: nút chưa được thăm và chưa nằm trong hàng đợi
 - ◆ Nút xám: chưa được thăm nhưng đang nằm trong hàng đợi
 - ◆ Nút đen: là nút đã được thăm

Duyệt đồ thị theo chiều rộng



Tài liệu tham khảo

- Bài giảng – Nguyễn Thanh Bình - ĐTVT
- Bài giảng – Đỗ Bích Diệp
- Chapter 13 - Graph– Data structure in C – Rema Thareja - 2014