CHƯƠNG 6

TÌM KIẾM TRÊN ĐỒ THỊ - ĐỒ THỊ EULER VÀ ĐỒ THỊ HAMILTON

Bài toán tìm kiếm trên đồ thị (1/2)

Tìm đường đi từ một đỉnh đến một đỉnh khác

- Kiểm tra tính liên thông
- Kiểm tra tính liên thông mạnh
- Xác định các thành phần liên thông của đồ thị

Duyệt qua các đỉnh của đồ thị

- Để cập nhật, xử lý dữ liệu tại các đỉnh của đồ thị

Bài toán tìm kiếm trên đồ thị (2/2)

Tìm đường đi:

- Từ đỉnh xuất phát s
- Đến đỉnh đích t

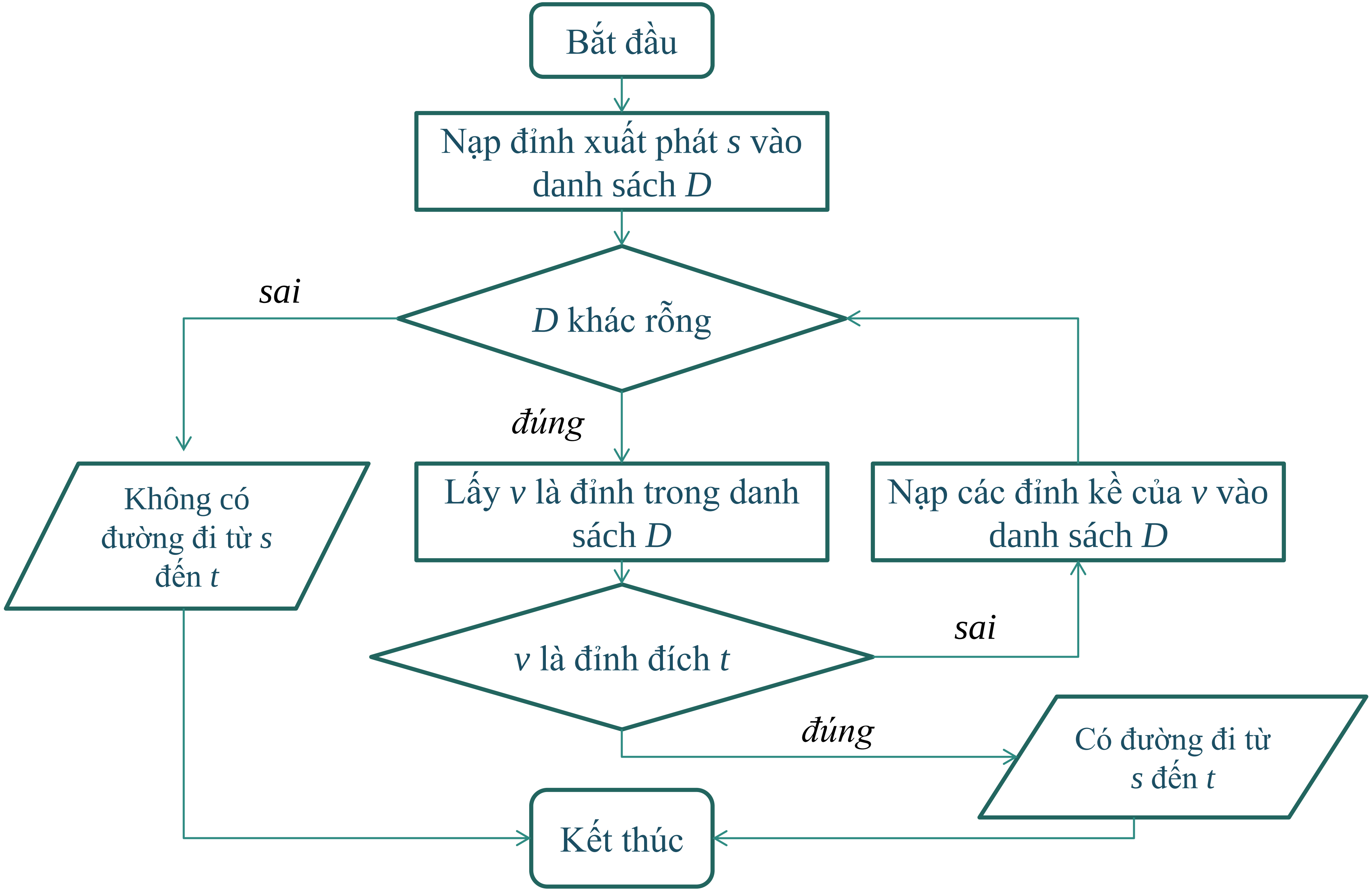
Yêu cầu 1:

- Tồn tại đường đi
- Hay không tồn tại đường đi

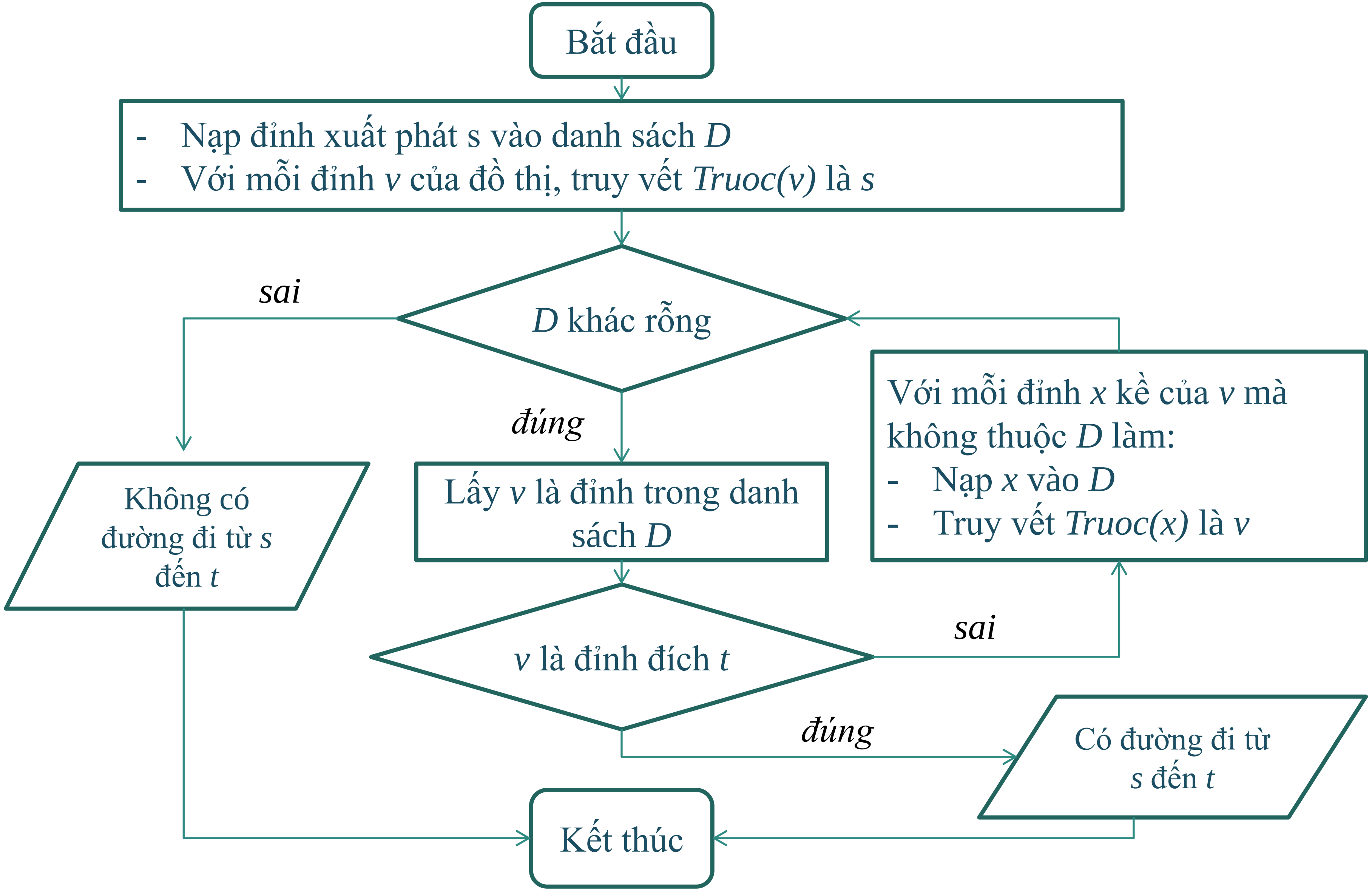
Yêu cầu 2:

- Nếu tồn tại đường đi từ s → t thì đi như thế nào?

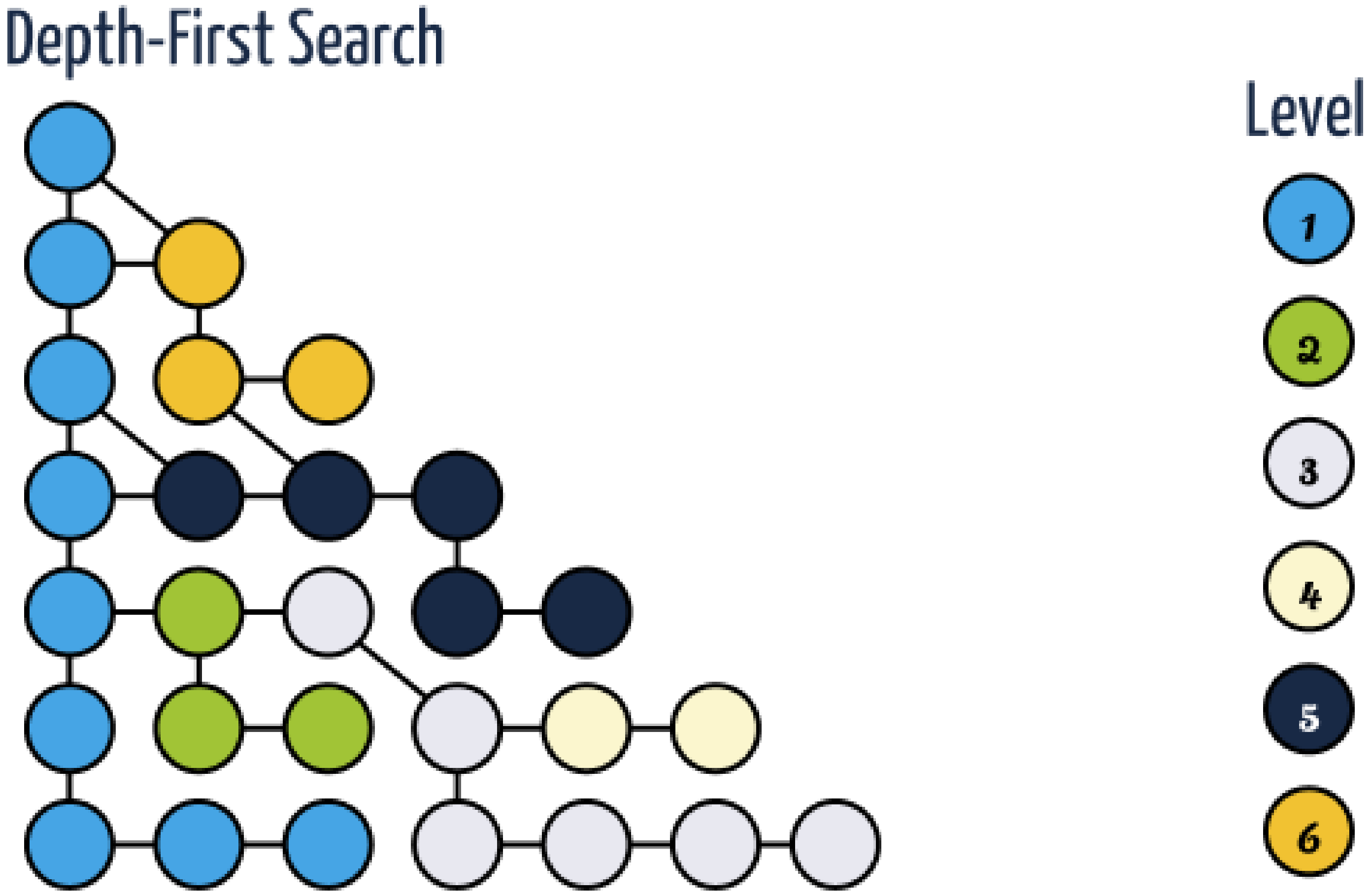
Thuật toán cho yêu cầu 1



Thuật toán cho yêu cầu 2



Thuật toán Depth First Search



Thuật toán Depth First Search

❖ Trong quá trình tìm kiếm DFS tổ chức lưu trữ danh sách các đỉnh theo kiểu LIFO - Last In First Out.

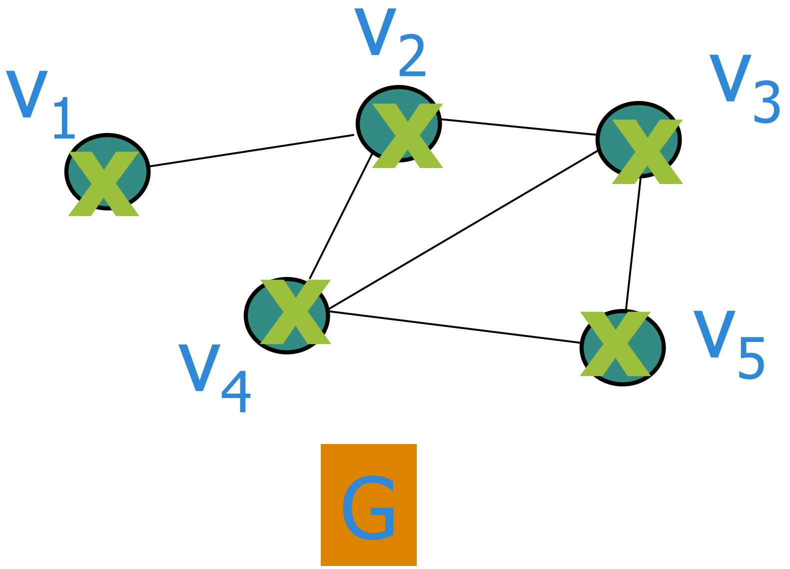
- Thuật toán tìm kiếm theo chiều sâu (DFS):
  - **Input**: G(V, E) và đỉnh s $\in V$
  - **Output**: Dãy đỉnh được viếng thăm
  - Bước 1: Viếng thăm đỉnh s
  - Bước 2: Tìm đỉnh u: u kề với s và u chưa được viếng thăm
    - Nếu có đỉnh u thì đặt s=u và quay về Bước 1
    - Nếu không có đỉnh u thì quay về đỉnh trước đỉnh s, và tìm hướng đi khác.
    - Thuật toán dừng khi không thể quay về đỉnh trước.

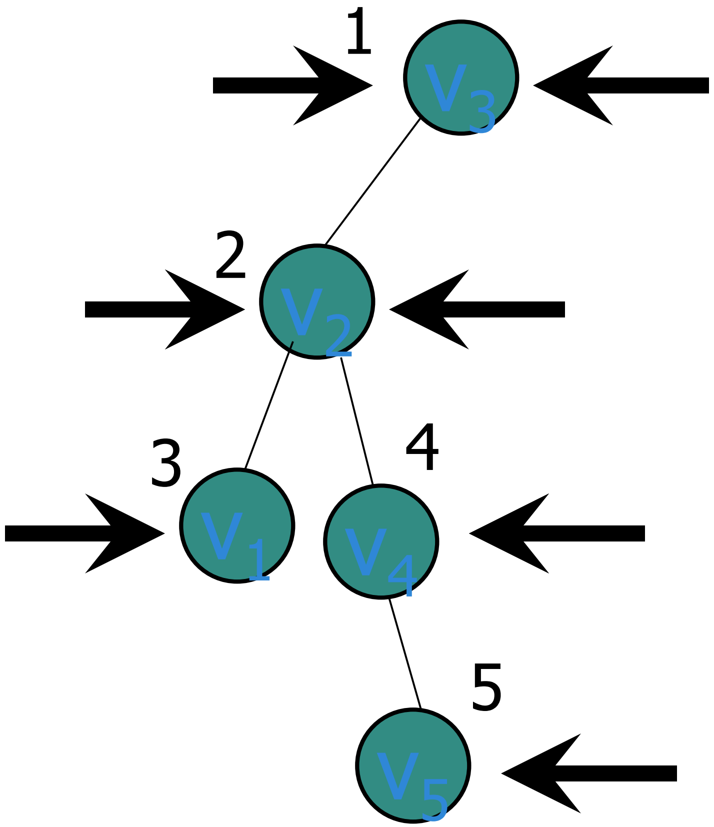
Tìm kiếm trên đồ thị theo chiều sâu
Cài đặt bằng Đệ quy

```
void DFS(int s) {
      if (visited[s] == true) return;
      // Bước 1
      visited[s] = true;

      // process node s:…

      // Bước 2
      foreach (int u in v[s])
            DFS(u);
}
```

DFS example

- Start from v_3



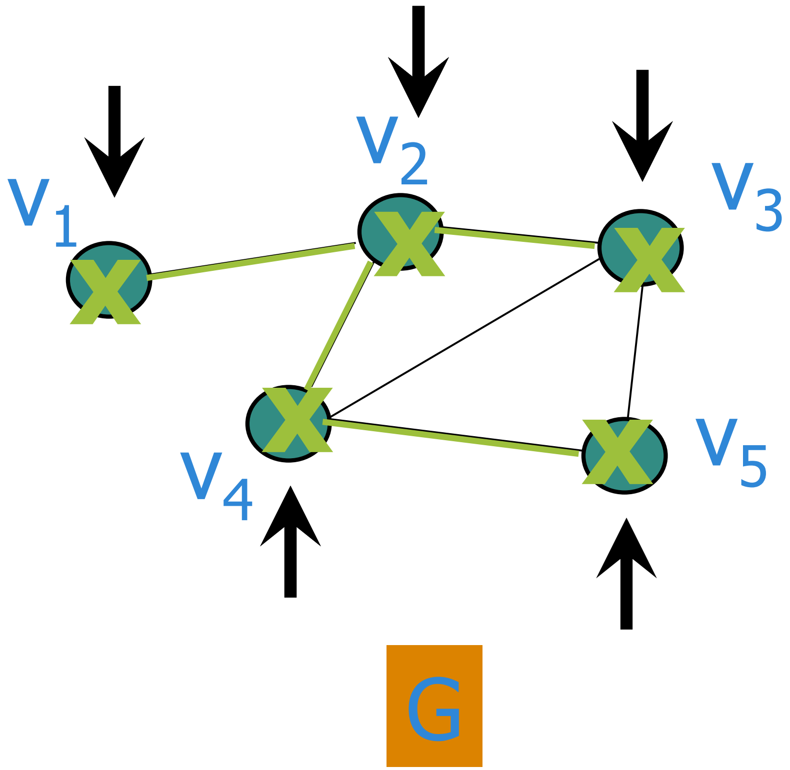


Tìm kiếm trên đồ thị theo chiều sâu
Cài đặt bằng Stack

```
// Dùng stack
void DFS(int s)
{
      1. Đánh dấu s đã viếng thăm
      2. Đưa s vào stack
      3. while stack chưa rỗng
      {
            <Lấy 1 đỉnh u từ stack>
            <Tìm 1 đỉnh v kề u và chưa được viếng thăm>
            {
                  Đánh dấu v đã viếng thăm
                  Đẩy u vào lại stack
                  Đẩy v vào stack
            }
      }
}
```

Non-recursive DFS example

visit	stack
v_3	v_3
v_2	v_3, v_2
v_1	v_3, v_2, v_1
backtrack	v_3, v_2
v_4	v_3, v_2, v_4
v_5	v_3, v_2, v_4 , v_5
backtrack	v_3, v_2, v_4
backtrack	v_3, v_2
backtrack	v_3
backtrack	empty



Thuật toán Breadth First Search

❖ Trong quá trình tìm kiếm BFS tổ chức lưu trữ danh sách các đỉnh theo kiểu FIFO – First In First Out.

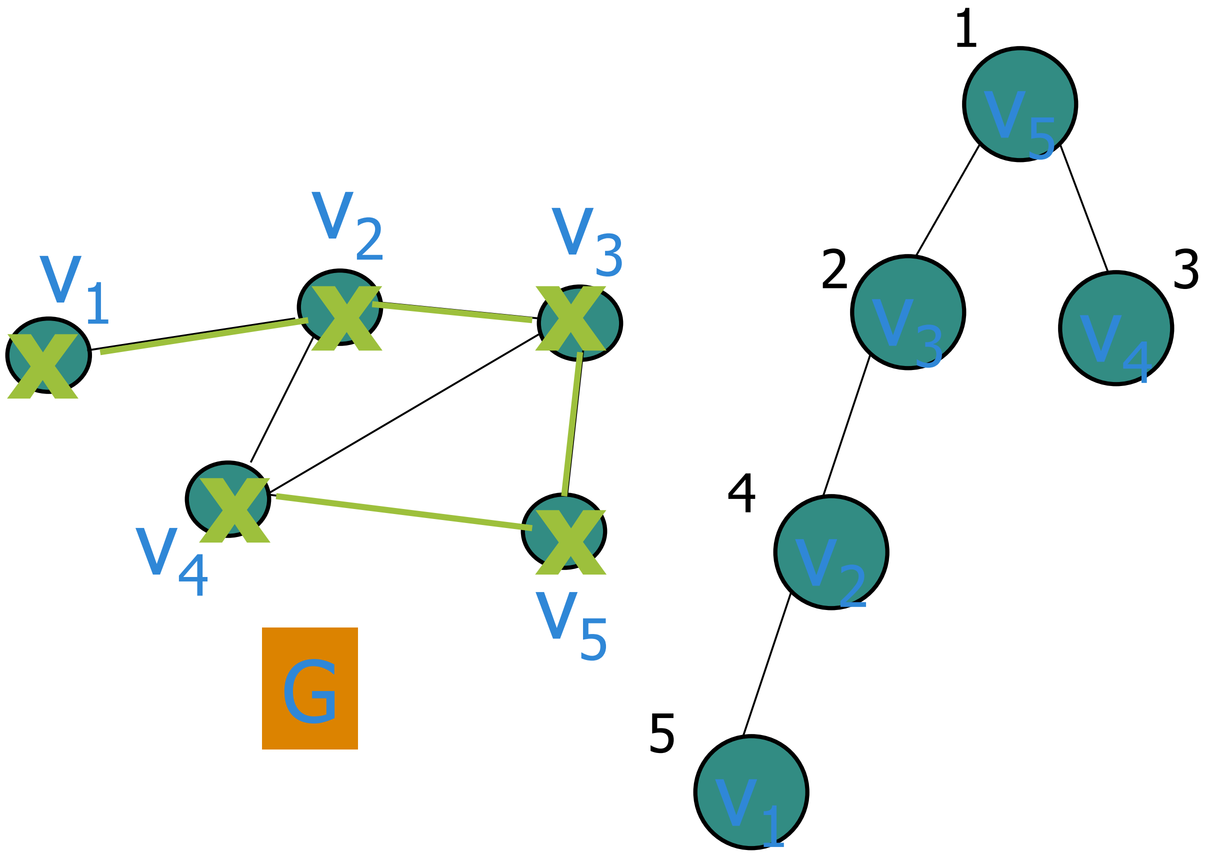
- Thuật toán tìm kiếm theo chiều rộng (BFS):
  - **Input**: G(V, E) và đỉnh $s \in V$
  - **Output**: Dãy đỉnh được viếng thăm
  - Bước 1: Gọi $S_1=\{s\}$, viếng thăm các đỉnh trong S_1
  - Bước 2: Tìm các đỉnh (gọi là S_2) kề với một trong những đỉnh S_1, và chưa được viếng thăm, rồi viếng thăm các đỉnh trong S_2
  - Bước 3: Tìm các đỉnh (gọi là S_3) kề với một trong những đỉnh S_2 và chưa được viếng thăm, rồi viếng thăm các đỉnh trong S_3
  - ... Cho đến khi không thể tìm thêm các đỉnh để viếng thăm

Tìm kiếm trên đồ thị theo chiều rộng
Cài đặt bằng hàng đợi

```
void BFS(int s)
{
  visited[s]=true;
  q.push(s);
  // Process node s

  while(q.Count!=0) {
    s = q.front();
    q.pop()

    foreach (int u in v[s]) {
      if (visited[u]) continue;
      visited[u]=true;
      q.push(u);

      // Process node u
    }
  }
}
```

BFS example

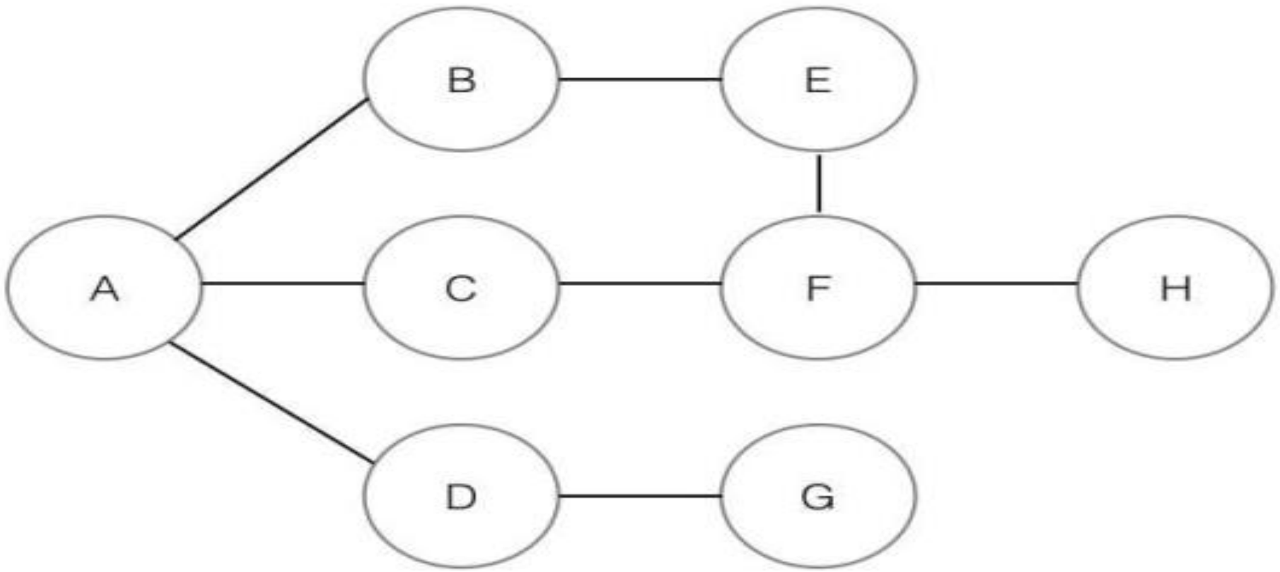
- Start from v_5



Visit	Queue (front to back)
v_5	v_5
	empty
v_3	v_3
v_4	v_3, v_4
	v_4
v_2	v_4, v_2
	v_2
	empty
v_1	v_1
	empty

DFS & BFS (từ A)

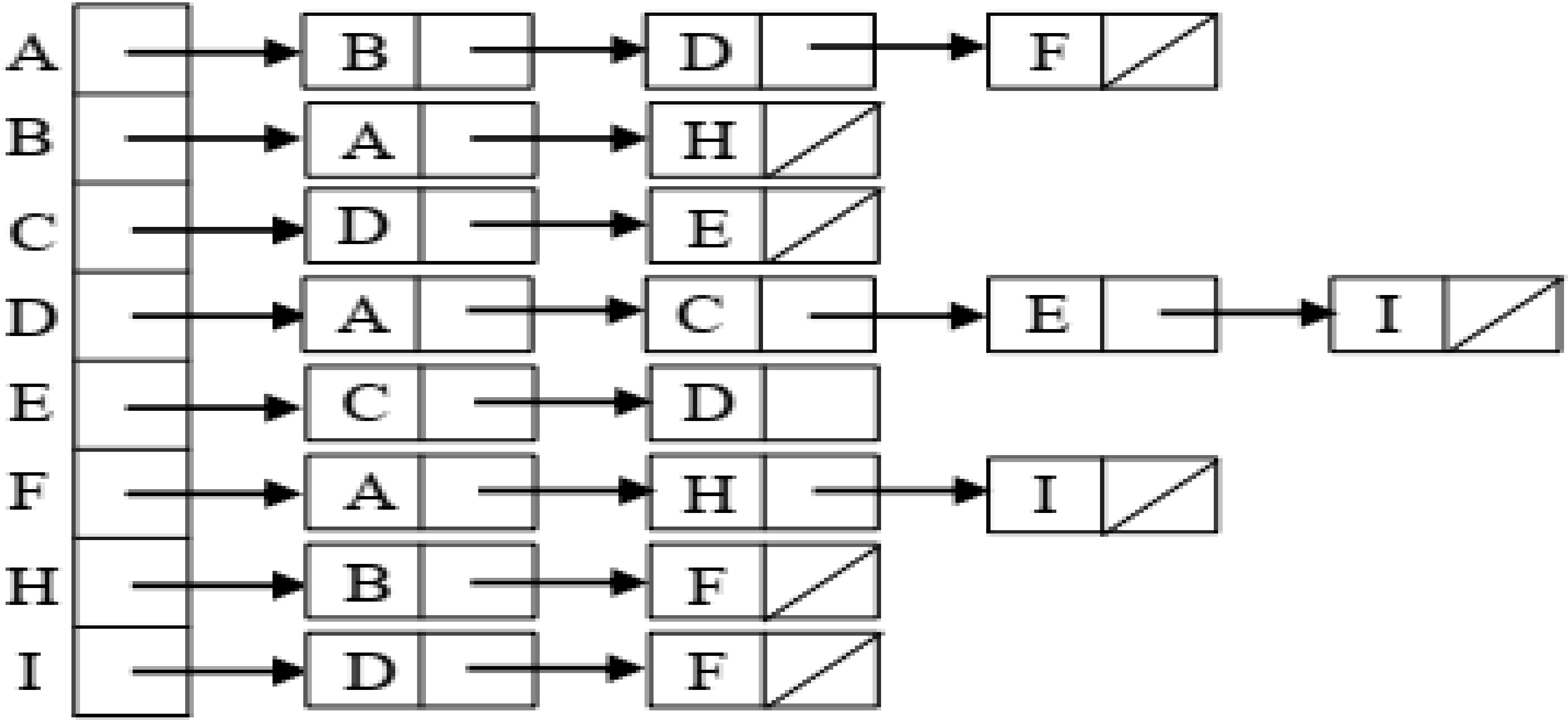
	A	B	C	D	E	F	G	H
A	0	1	1	1	0	0	0	0
B	1	0	0	0	1	0	0	0
C	1	0	0	0	0	1	0	0
D	1	0	0	0	0	0	1	0
E	0	1	0	0	0	1	0	0
F	0	0	1	0	1	0	0	1
G	0	0	0	1	0	0	0	0
H	0	0	0	0	0	1	0	0



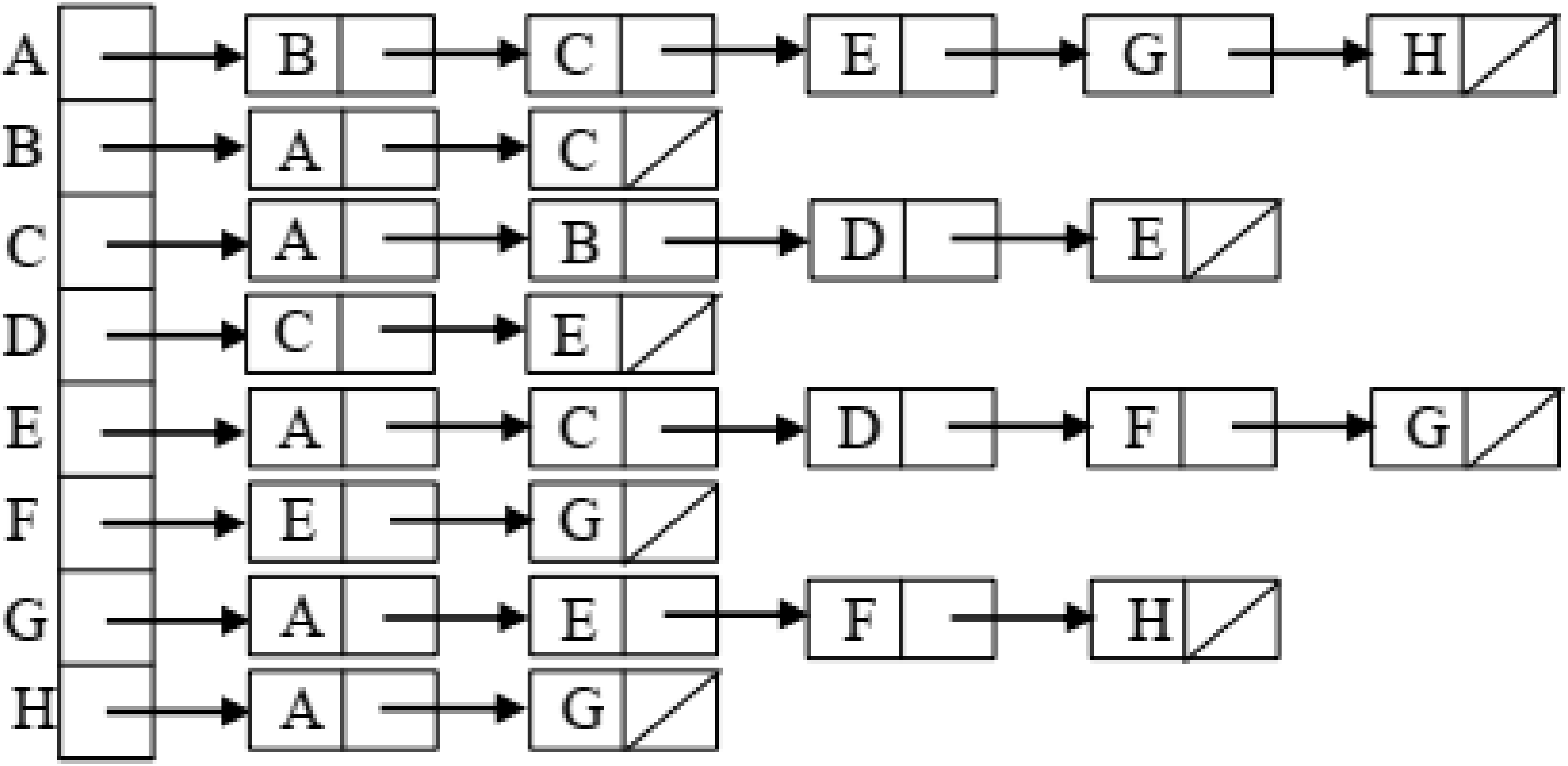
A: B, C, D
B: A, E
C: A, F
D: A, G
E: B, F
F: C, E, H
G: D
H: F

DFS (stack)	BFS (queue)
$S = [A]$	$Q = [A]$
$S = [D, C, B]$	$Q = [B, C, D]$
$S = [G, C, B]$	$Q = [C, D, E]$
$S = [C, B]$	$Q = [D, E, F]$
$S = [F, B]$	$Q = [E, F, G]$
$S = [H, E, B]$	$Q = [F, G]$
$S = [E, B]$	$Q = [G, H]$
$S = [B]$	$Q = [H]$
$S = []$	$Q = []$

Ví dụ 1 (tự làm): DFS & BFS (từ I)

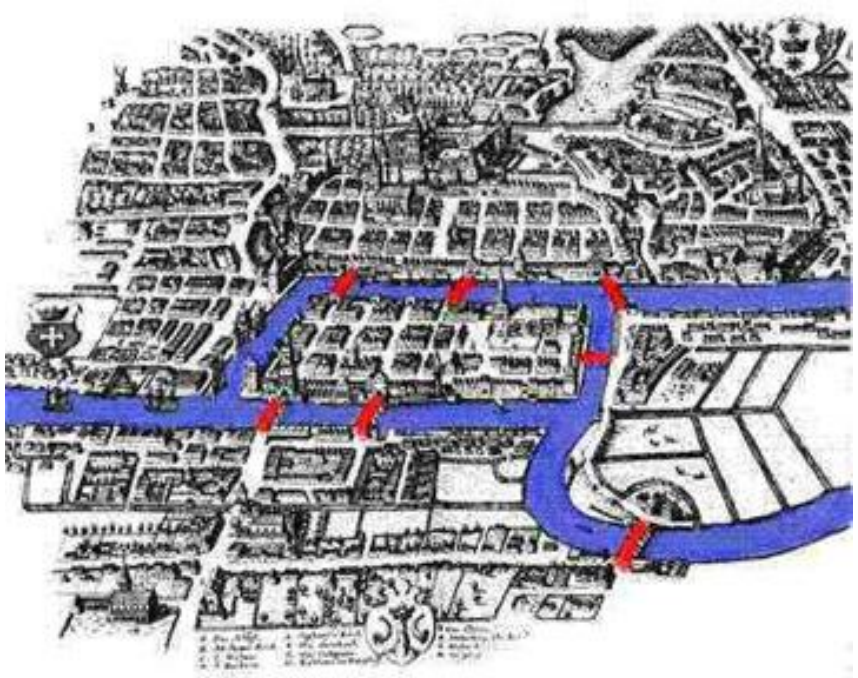


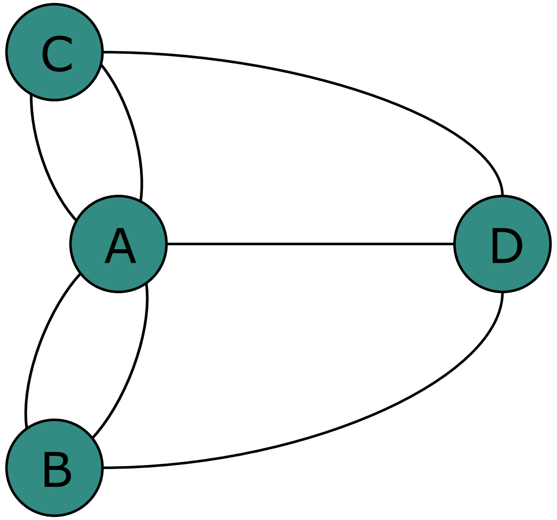
Ví dụ 2 (tự làm): DFS & BFS (từ A)



Đồ thị Euler

- **Bài toán Tìm đường đi qua 7 cái cầu trong thành phố Konigsberg:**
  - Làm sao xuất phát từ 1 vị trí, di chuyển qua tất cả các cầu (mỗi cầu qua 1 lần) và trở về vị trí xuất phát





Mô hình đồ thị

Đồ thị Euler

- Bài toán đã làm say mê cư dân của thành phố. Họ háo hức đi thử nhưng không thành công.
- Năm 1736, Leonhard Euler (nhà toán học Thụy Sĩ) đã chứng minh rằng bài toán không giải được.
- Từ bài toán này dẫn đến các khái niệm về đường, chu trình Euler và đồ thị Euler.

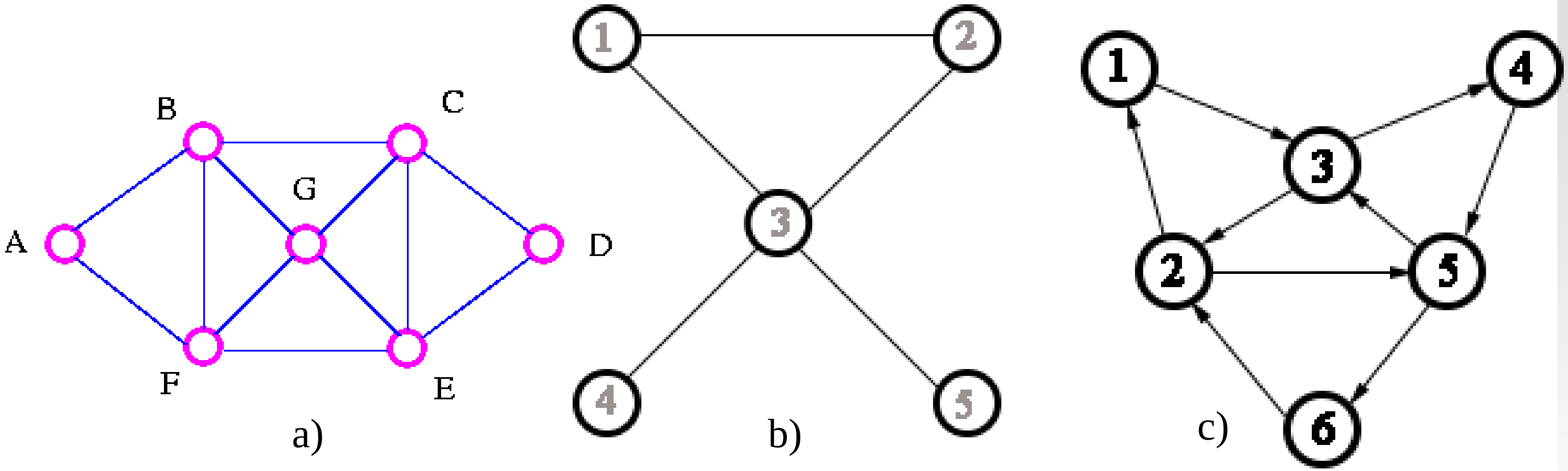
Đồ thị Euler

❖ **Một số định nghĩa:**

- *Đường Euler* là đường đi qua mỗi cạnh của đồ thị đúng một lần.
- *Chu trình Euler* là chu trình đi qua mỗi cạnh của đồ thị đúng một lần.
- *Đồ thị Euler* là đồ thị có chu trình Euler.
- *Đồ thị nửa Euler* là đồ thị có đường đi Euler.

Đồ thị Euler

❖ Ví dụ



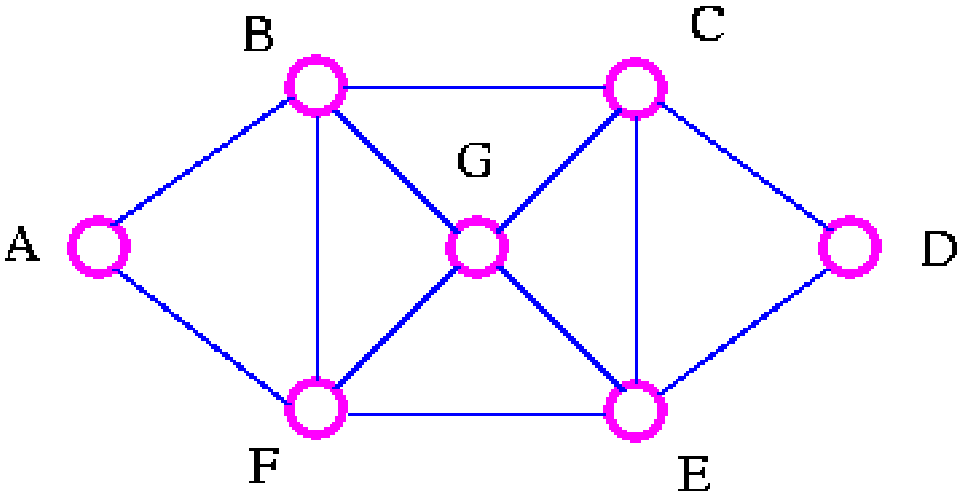
a) b) c)

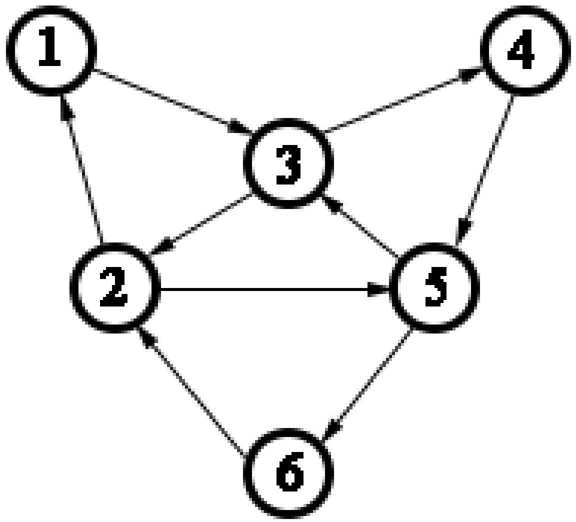
1. Hãy cho biết đâu là đồ thị Euler, nửa Euler? Vì sao?
2. Chỉ ra đường đi Euler và chu trình Euler

Đồ thị Euler

❖ **Định lý**

- Đồ thị vô hướng liên thông là Euler khi và chỉ khi mọi đỉnh đều có bậc chẵn.
- Đồ thị có hướng liên thông là Euler khi và chỉ khi với mọi đỉnh tổng bán bậc vào bằng tổng bán bậc ra của nó (tức là mọi đỉnh đều cân bằng).

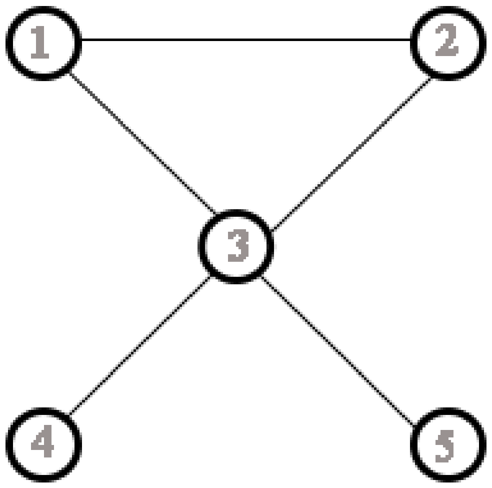


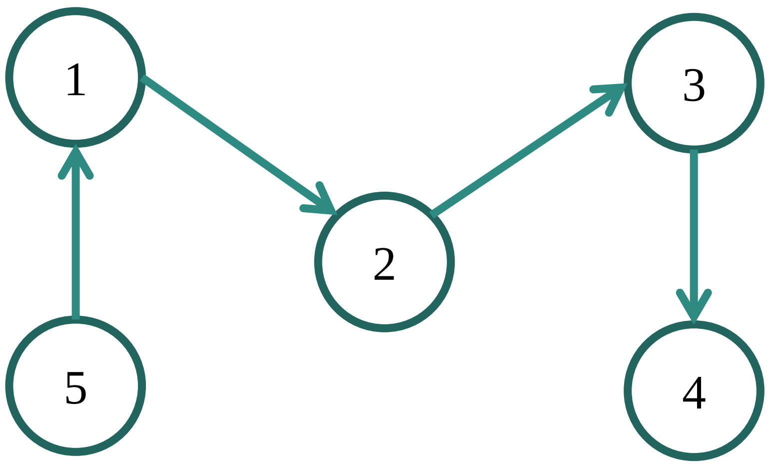


Đồ thị Euler

❖ Hệ quả

- Đồ thị vô hướng liên thông là nửa Euler khi và chỉ khi nó chứa không quá 2 đỉnh bậc lẻ.
- Đồ thị có hướng liên thông là nửa Euler khi và chỉ khi nó chứa 2 đỉnh a, b thoả mãn: indeg(a) = outdeg(a) – 1 và indeg(b) = outdeg(b) + 1, còn các đỉnh khác đều cân bằng





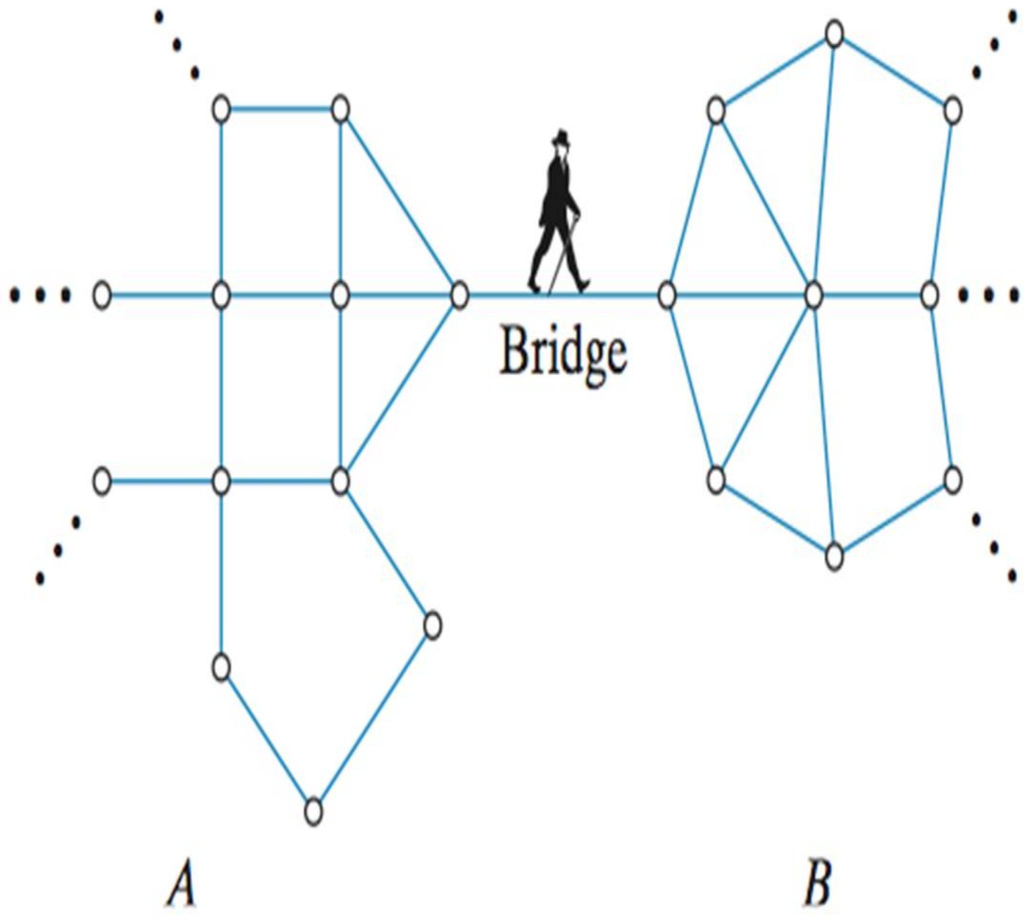
Đồ thị Euler

❖ Thuật toán tìm chu trình Euler :

Thuật toán Fleury

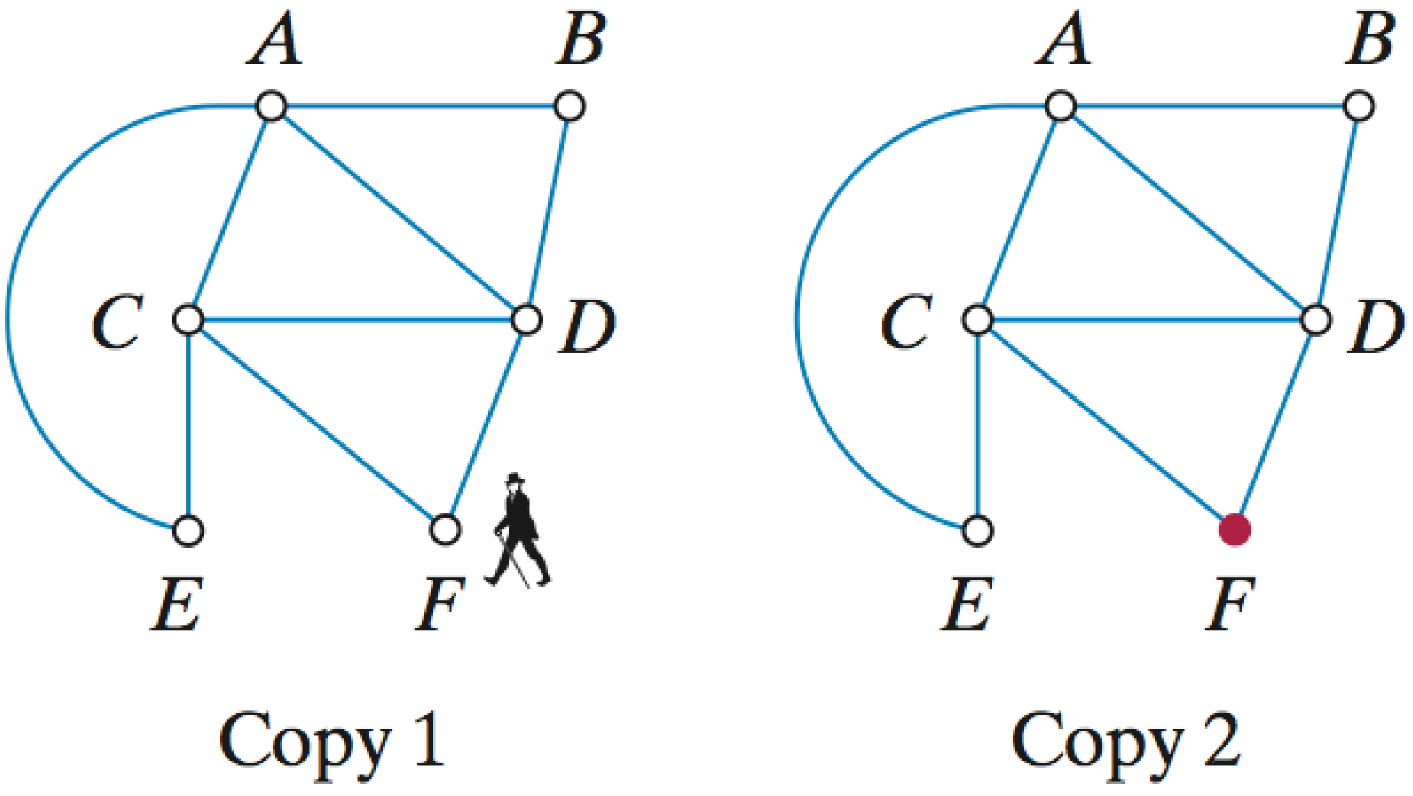
Bắt đầu từ một đỉnh bất kỳ, đi theo các cạnh của đồ thị theo quy tắc sau:

- *Qui tắc 1: Xóa các cạnh đã đi qua và các đỉnh cô lập nếu có*
- *Qui tắc 2: Tại mỗi đỉnh, ta chỉ đi qua cầu nếu không còn đường nào khác.*



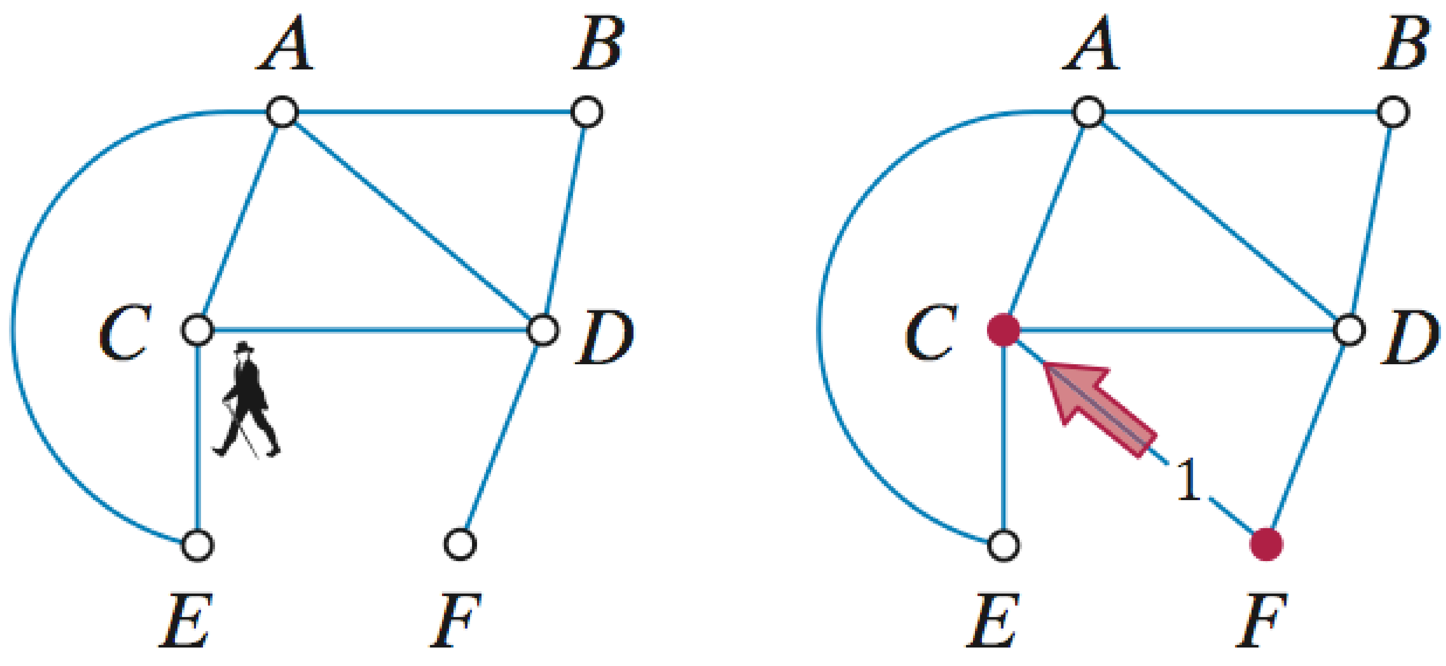
Implementing Fleury's Algorithm

Start: We can pick any starting point we want. Let's say we start at *F*.



Implementing Fleury's Algorithm

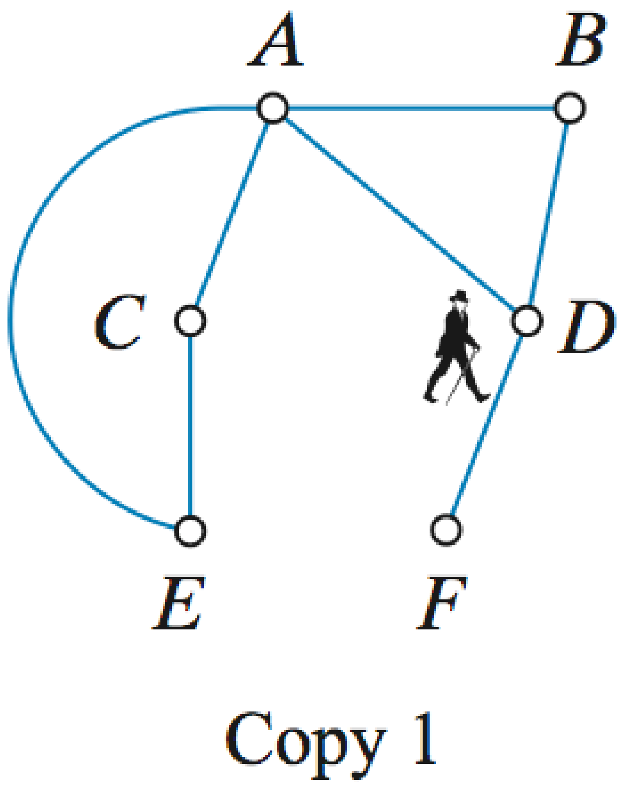
Step 1: Travel from *F* to *C*. (Could have also gone from *F* to *D*.)

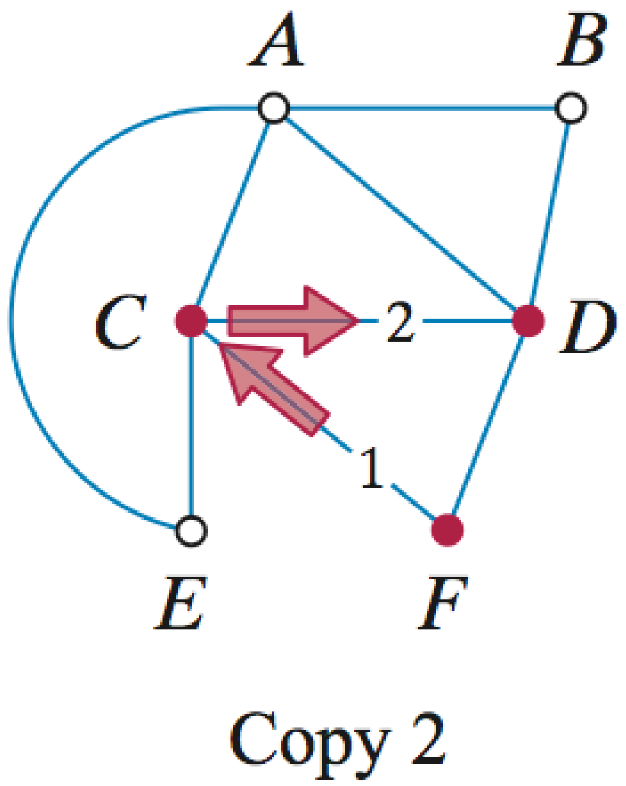


Copy 1 Copy 2

Implementing Fleury's Algorithm

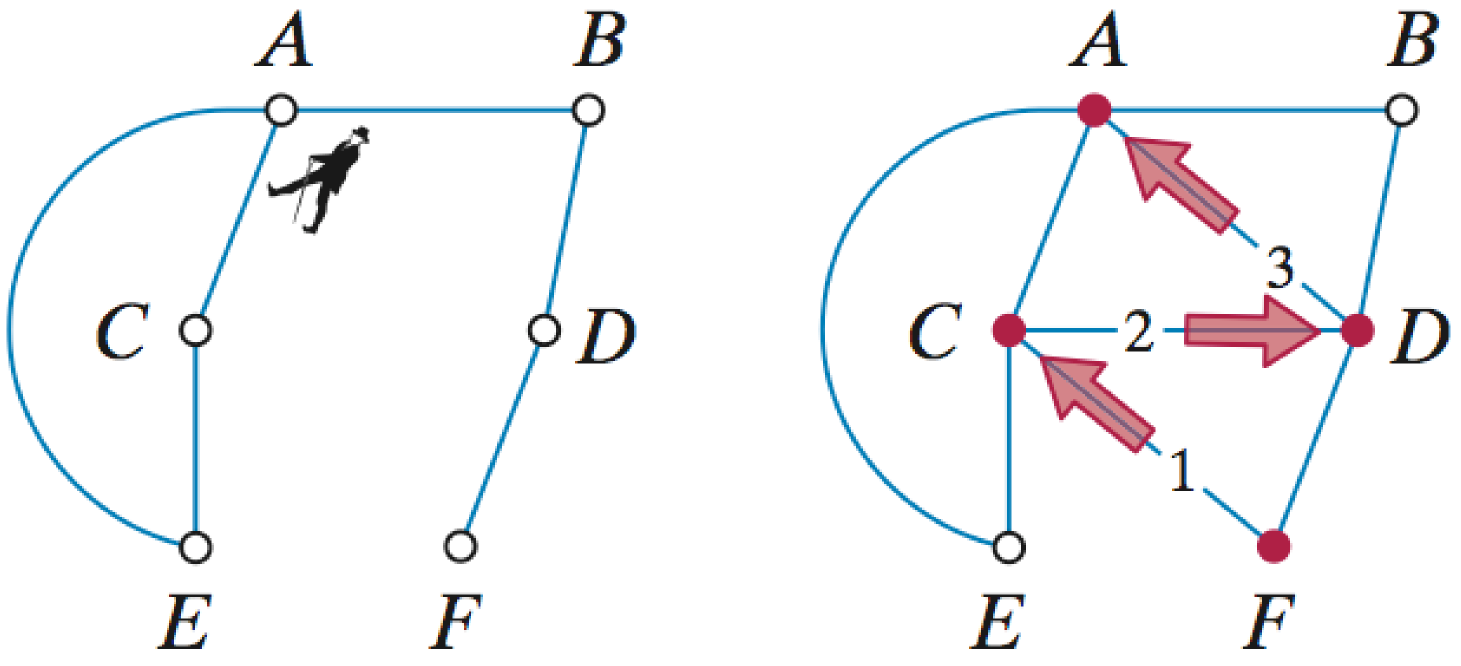
Step 2: Travel from *C* to *D*. (Could have also gone to *A* or to *E*.)





Implementing Fleury's Algorithm

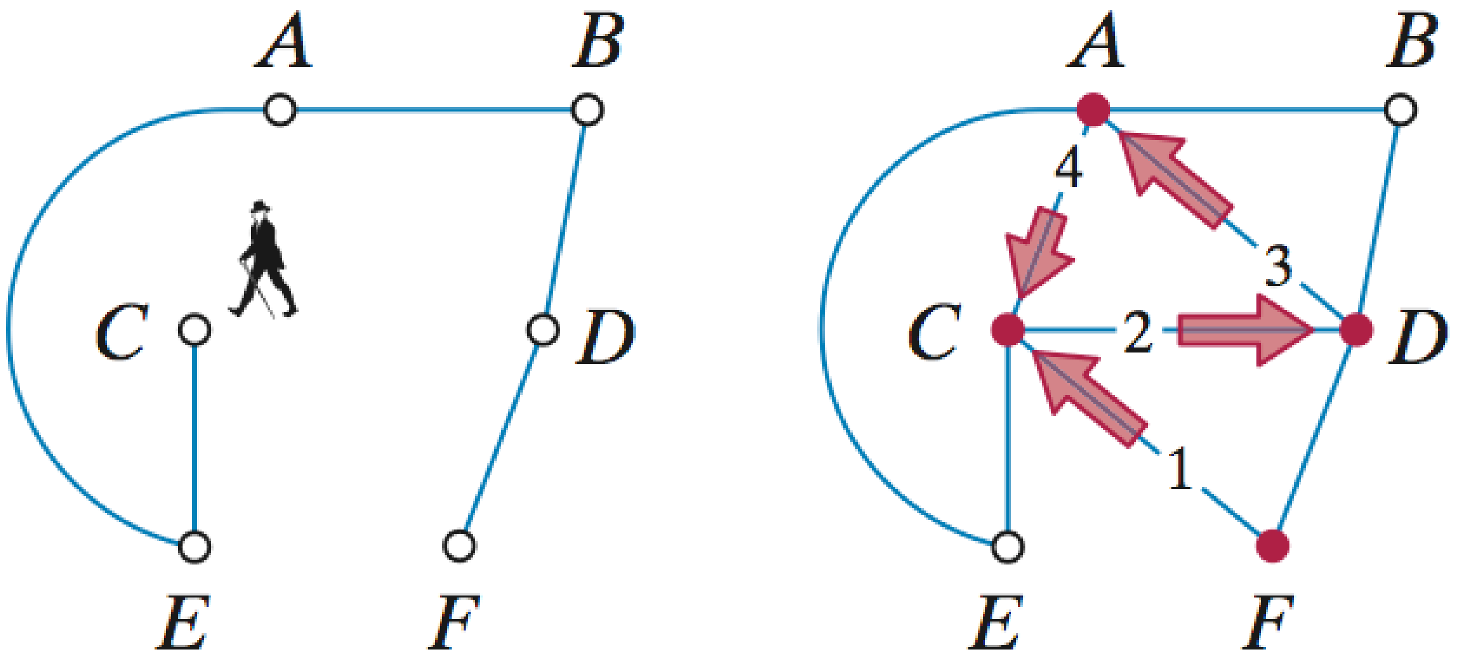
Step 3: Travel from *D* to *A*. (Could have also gone to *B* but not to *F* – *DF* is a bridge!)



Copy 1 Copy 2

Implementing Fleury’s Algorithm

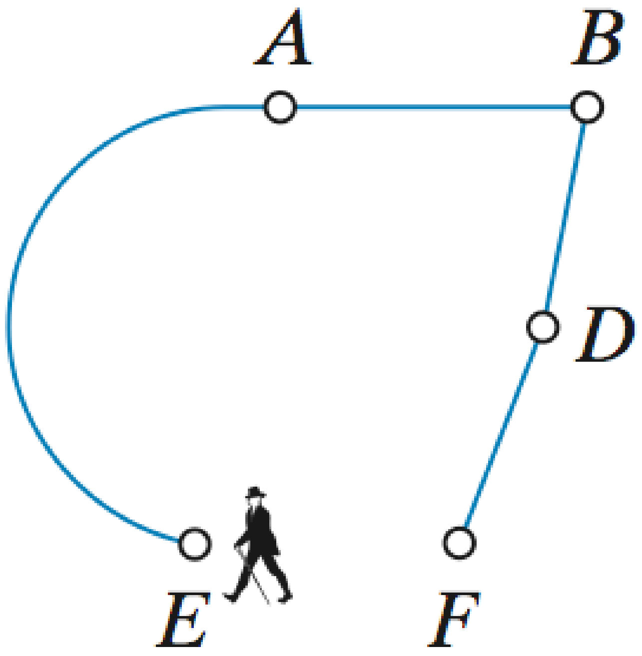
Step 4: Travel from *A* to *C*. (Could have also gone to *E* but not to *B* – *AB* is a bridge!)



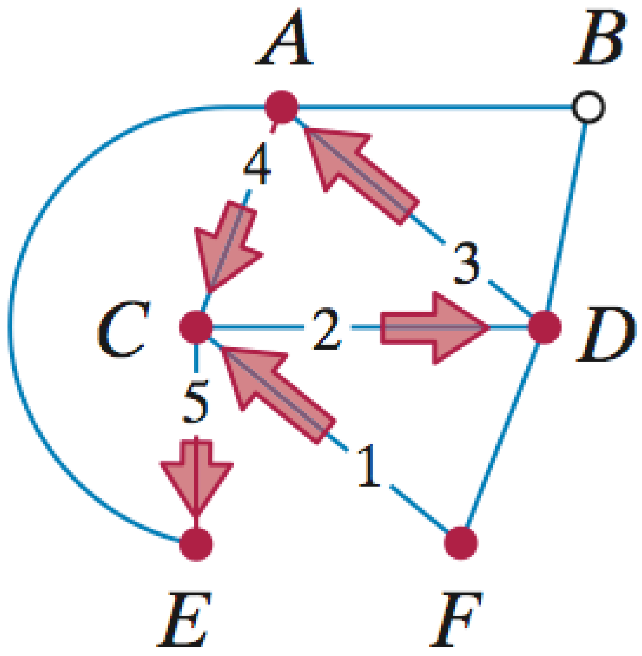
Copy 1 Copy 2

Implementing Fleury’s Algorithm

Step 5: Travel from *C* to *E*. (There is no choice!)



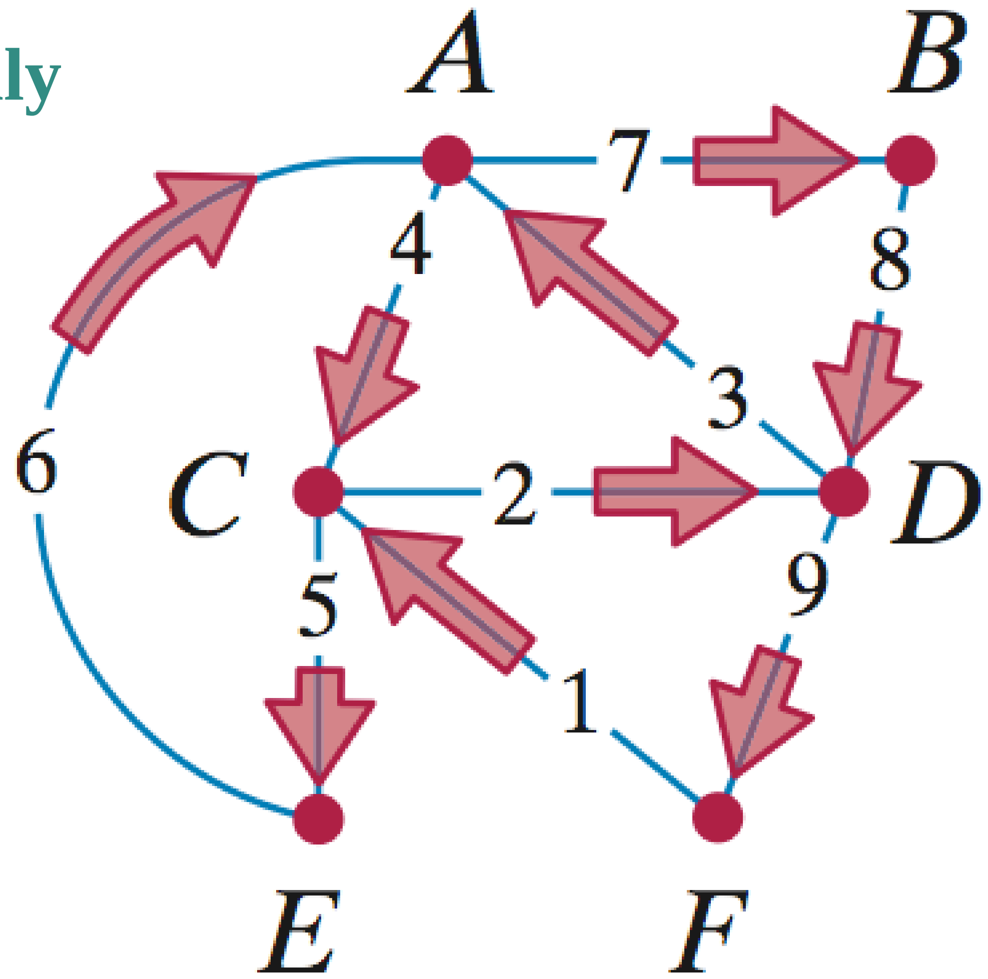
Copy 1



Copy 2

Implementing Fleury's Algorithm

Steps 6, 7, 8, and 9: Only one way to go at each step.



Đồ thị Euler

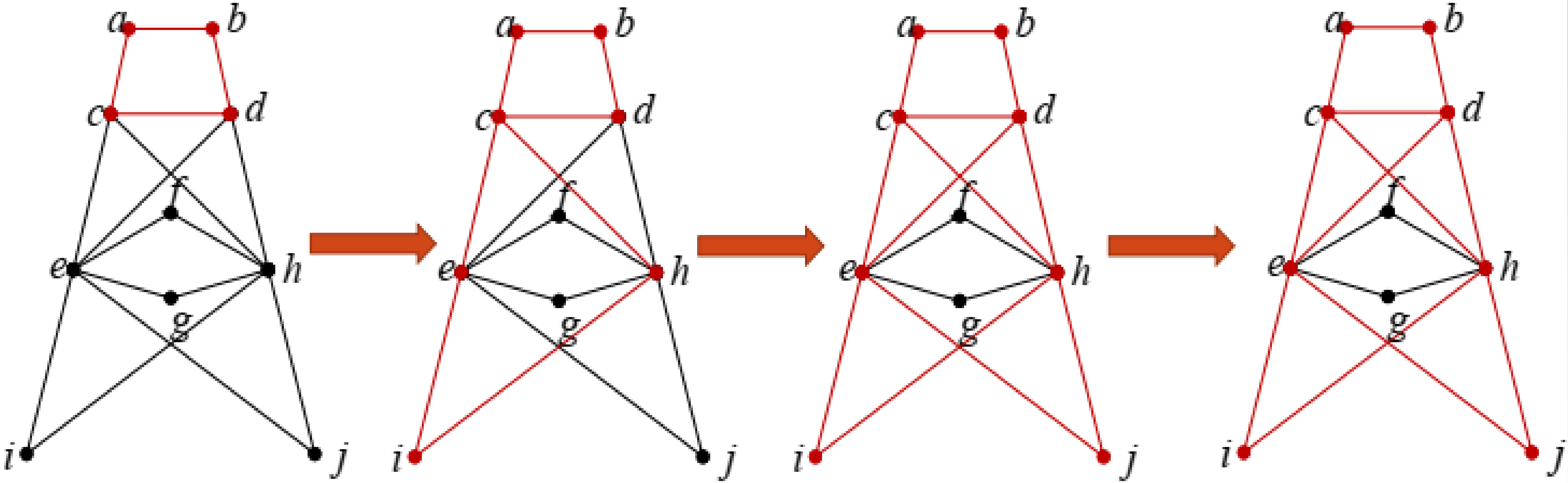
❖ **Thuật toán tìm chu trình Euler :**

Thuật toán Hierholzer

- B1: Xác định 1 chu trình đơn của G là R_1; i = 1
- B2: Nếu R_i chứa toàn bộ : kết thúc; R_i là kết quả
- B3: Nếu R_i không chứa toàn bộ G

  Xét đỉnh $v_i \in R_i$ là đỉnh của cạnh e_j không thuộc R_i
- B4: Xác định chu trình đơn Q_i bắt đầu từ v_i, đi qua e_j
- B5: Tạo R_{i+1} bằng cách thay v_i trong R_i bằng Q_i
- B6: Tăng i lên 1, quay lại bước 2.

Thuật toán Hierholzer



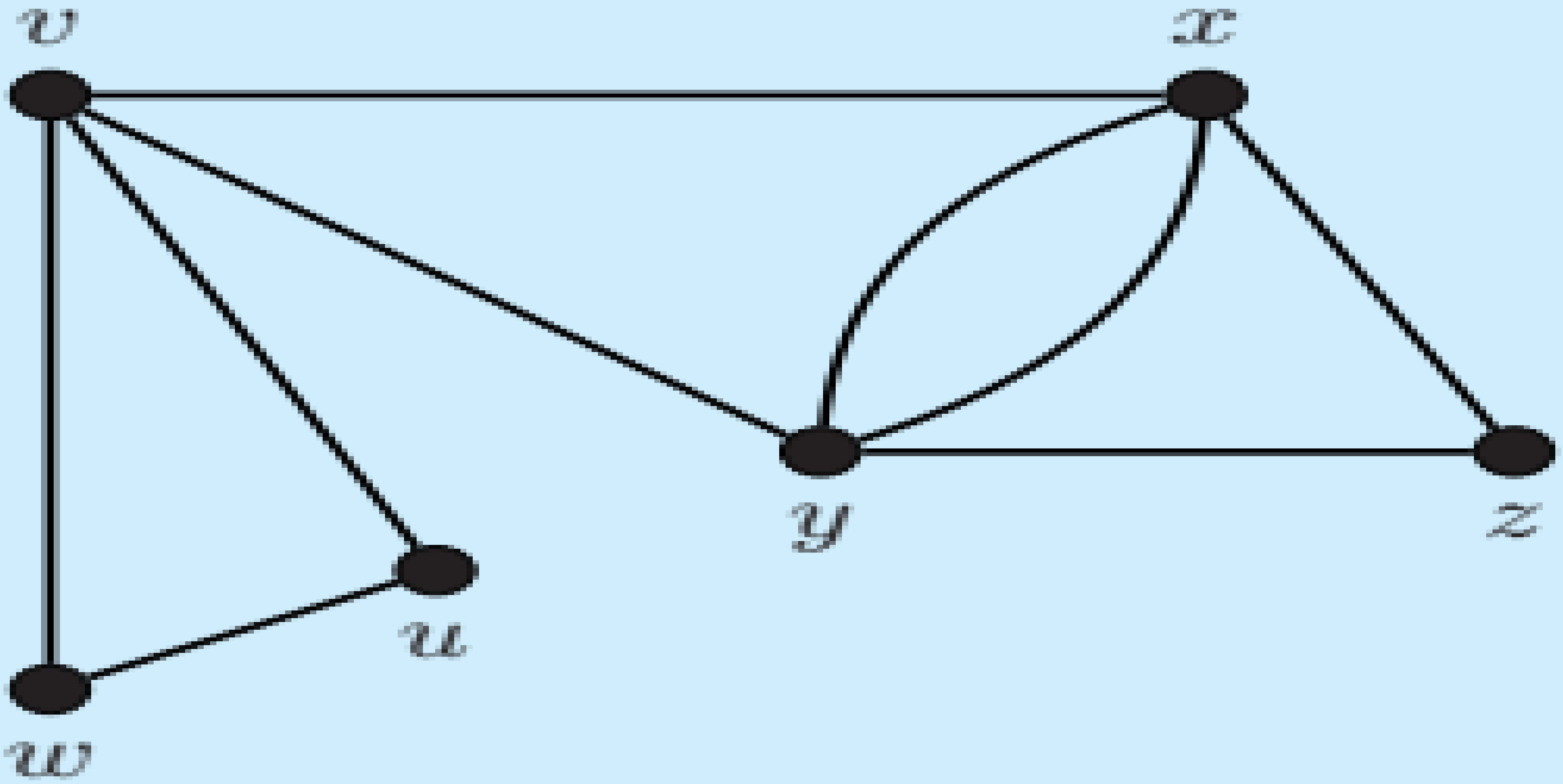
R_1 = a, b, d, c, a
Q_1 = c, e, i, h, c

R_2 = a, b, d, **c, e, i, h, c**, a
Q_2 = e, d, h, j, e

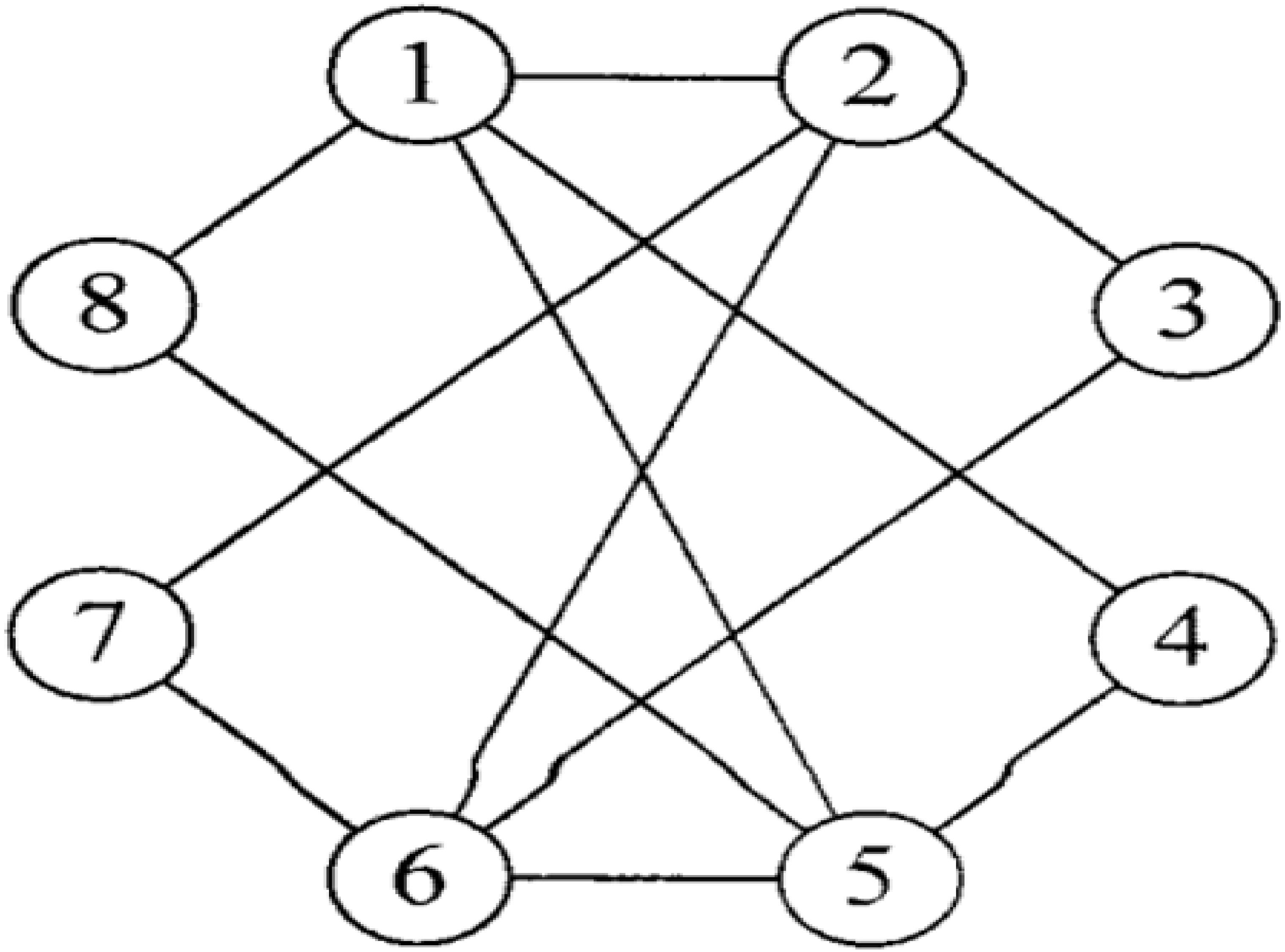
R_3 = a, b, d, c, **e, d, h, j, e**, i, h, c, a
Q_3 = e, f, h, g, e

R_4 = a, b, d, c, **e, f, h, g, e**, d, h, j e, i, h, c, a
=> Xong

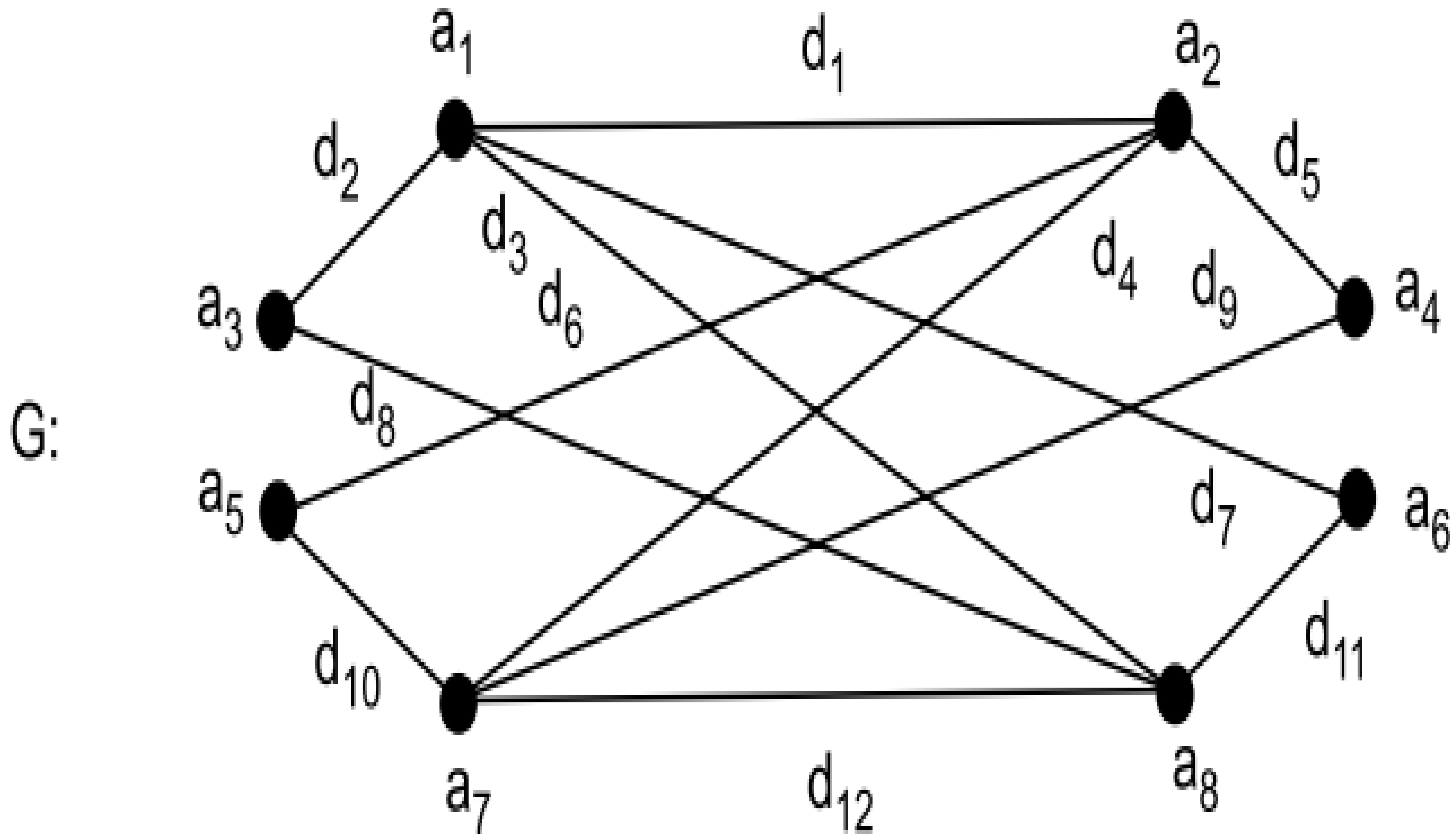
Ví dụ 3 (tự làm): Chạy tay thuật toán Fleury và Hierholzer để tìm chu trình Euler cho đồ thị sau



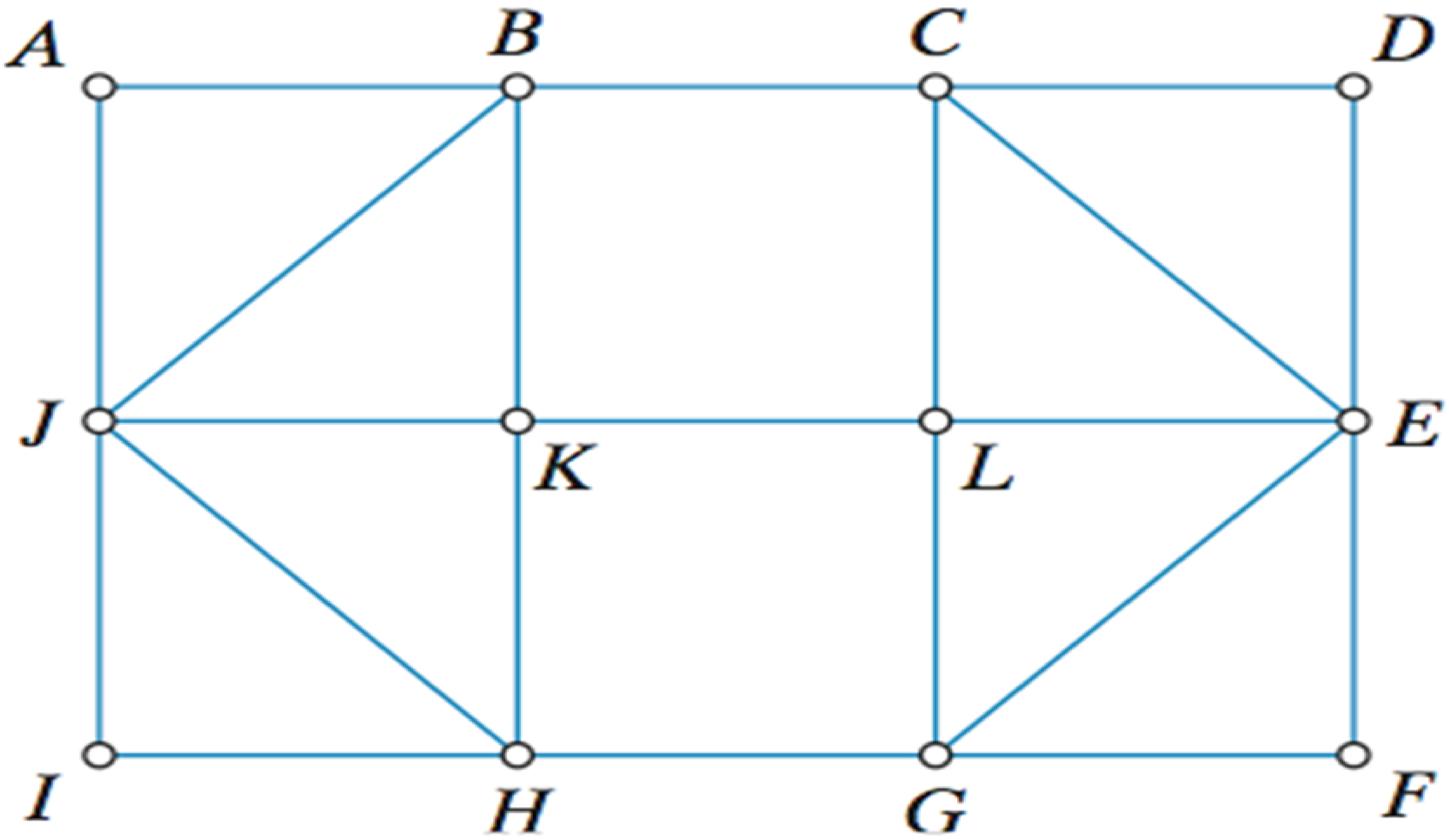
Ví dụ 4 (tự làm): Chạy tay thuật toán Fleury và Hierholzer để tìm chu trình Euler cho đồ thị sau



Ví dụ 5 (tự làm): Chạy tay thuật toán Fleury và Hierholzer để tìm chu trình Euler cho đồ thị sau



Ví dụ 5 (tự làm): Chạy tay thuật toán Fleury để tìm đường đi Euler cho đồ thị sau



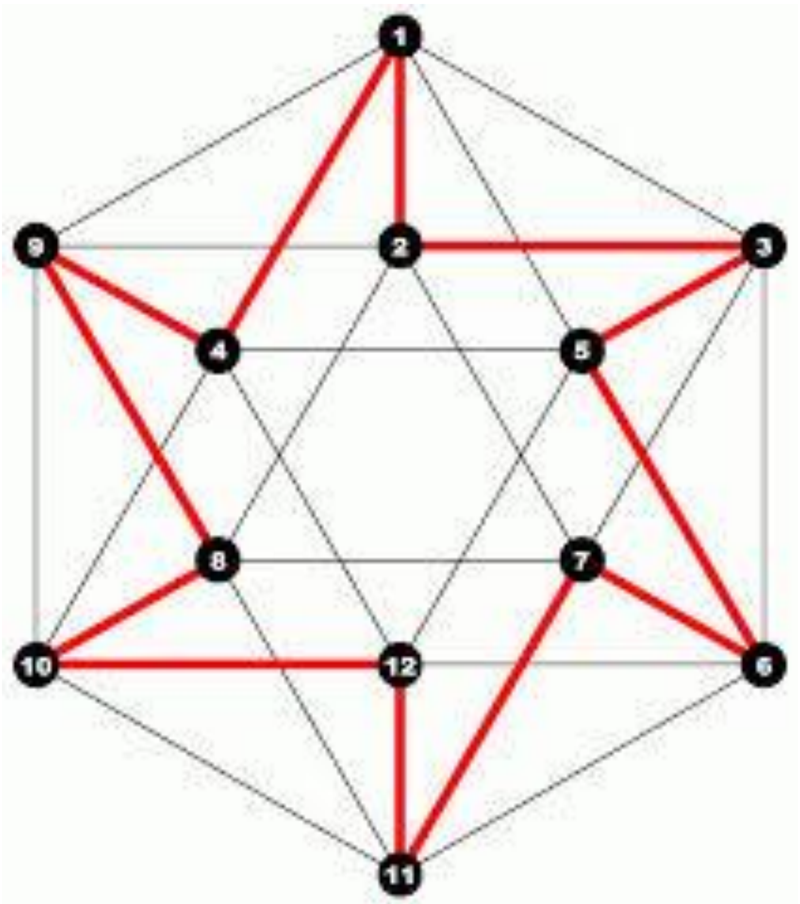
Đồ thị Hamilton (1/4)

Các định nghĩa

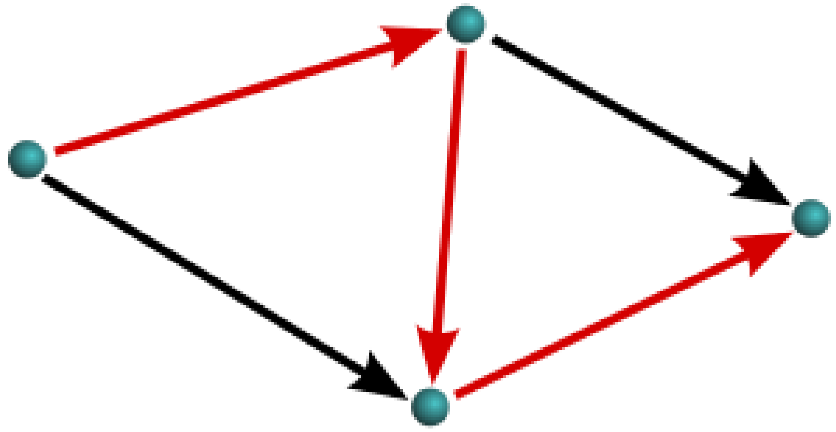
- *Đường Hamilton* là đường đi qua mỗi đỉnh của đồ thị đúng một lần.
- *Chu trình Hamilton* là chu trình đi qua mỗi đỉnh của đồ thị đúng một lần.
- *Đồ thị Hamilton* là đồ thị có chu trình Hamilton.
- *Đồ thị nửa Hamilton* là đồ thị có đường đi Hamilton.

Đồ thị Hamilton (2/4)

❖ Các ví dụ (1/2)



Đồ thị Hamilton



Đồ thị nửa Hamilton

Đồ thị Hamilton (3/4)

❖ Các ví dụ (2/2)

- Tổ chức tour du lịch sao cho người du lịch thăm quan mỗi thắng cảnh trong thành phố đúng một lần
- *Bài toán mã đi tuần*: cho con mã đi trên bàn cờ vua sao cho nó đi qua mỗi ô đúng một lần.

1	**2**	**3**	**4**
5	**6**	**7**	**8**
9	**10**	**11**	**12**

Đường Hamilton biểu diễn nước đi của con mã trên bàn cờ 3x4:

H = [8, 10, 1, 7, 9, 2, 11, 5, 3, 12, 6, 4]

Đồ thị Hamilton (4/4)

❖ Định lý Dirac

Nếu $\forall\ a \in V,\ deg(a) \geq (n/2)$ thì đồ thị vô hướng G(V,E) có chu trình Hamilton.

❖ Nhận xét

1. Đồ thị có đỉnh bậc ≤ 1 thì không có chu trình Hamilton.
2. Nếu đồ thị có các đỉnh đều có bậc ≥ 2 và có một đỉnh bậc 2 thì mọi chu trình Hamilton (nếu có) phải đi qua 2 cạnh kề của đỉnh này.
3. Nếu trong đồ thị có một đỉnh kề với 3 đỉnh bậc 2 thì không có chu trình Hamilton.

Thảo luận & bài tập (1/1)

- Cài đặt thuật toán BFS và DFS.
- Cài đặt thuật toán tìm chu trình Euler.
- Xây dựng & cài đặt thuật toán liệt kê tất cả các chu trình Hamilton.