Toán rời rạc

Phần thứ hai

LÝ THUYẾT ĐỒ THỊ

Graph Theory

Nội dung phần 2: Lý thuyết đồ thị

Chương 1. Các khái niệm cơ bản

Chương 2. Biểu diễn đồ thị

Chương 3. Duyệt đồ thị

Chương 4. Cây và cây khung của đồ thị

Chương 5. Bài toán đường đi ngắn nhất

Chương 6. Bài toán luồng cực đại trong mạng

Các thuật toán duyệt đồ thị

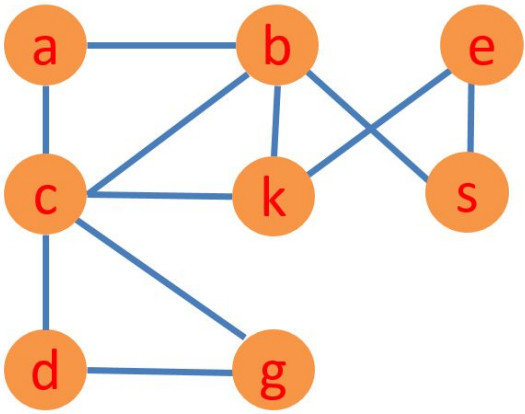
- Duyệt đồ thị: Graph Searching hoặc Graph Traversal
  - Duyệt qua mỗi đỉnh và mỗi cạnh của đồ thị
- Ứng dụng:
  - Cần để khảo sát các tính chất của đồ thị
  - Là thành phần cơ bản của nhiều thuật toán trên đồ thị
- Hai thuật toán duyệt cơ bản:
  - Tìm kiếm theo chiều rộng (Breadth First Search – BFS)
  - Tìm kiếm theo chiều sâu (Depth First Search – DFS)

Tìm kiếm theo chiều rộng

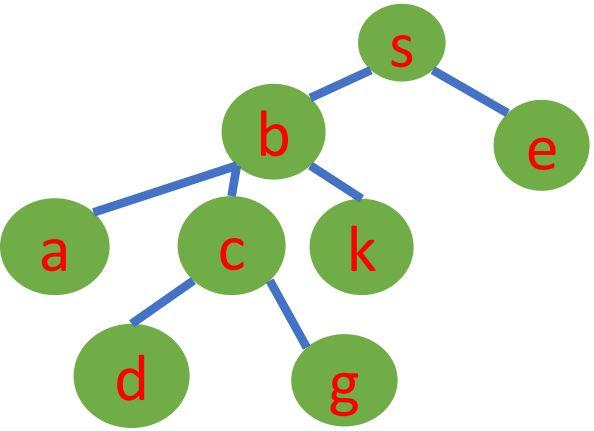
Breadth-first Search (BFS)

Breadth First Search

- Cho đồ thị G=(V,E) – tập đỉnh V, tập cạnh E và 1 đỉnh nguồn s
- Thuật toán tìm kiếm theo chiều rộng sẽ đi qua các cạnh của G để thăm tất cả các đỉnh đạt được từ s.
- Với mỗi đỉnh *v* đạt được từ *s*, đường đi trên cây BFS tương ứng với đường đi ngắn nhất từ *s* đến *v* trên đồ thị G.
- Thuật toán làm việc trên đồ thị vô hướng và có hướng.



G=(V, E)



BFS(s) tree

BFS tạo nên cây BFS chứa đỉnh s là gốc và các đỉnh đạt được từ s

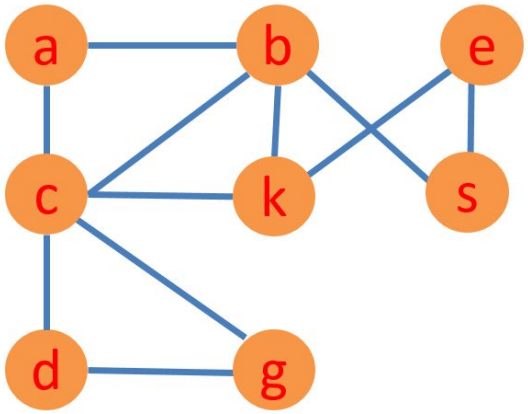
Danh sách kề của s

- Từ s: đến được b và e. Thăm chúng và cho
- Dequeue (Q): lấy b ra khỏi Q, lúc này Q = {e}
  - Từ b: đến được a, c, k, s. Nhưng s đã thăm, nên ta chỉ thăm a, c, k; và cho chúng vào queue: Q = {e, a, c, k}
- Dequeue(Q): lấy e khỏi Q, lúc này Q = {a, c, k}
  - Từ e: đến được k, s. Nhưng tất cả các đỉnh này đã được thăm.
- Dequeue(Q): lấy a khỏi Q, lúc này Q = {c, k}
  - Từ a: đến được b, c. Nhưng tất cả các đỉnh này đã được thăm.
- Dequeue(Q): lấy c khỏi Q, lúc này Q = {k}.
  - Từ c: đến được a, b, d, g, k. Nhưng a, b, k đã thăm, nên ta chỉ thăm d, g; và cho chúng vào queue: Q = {k, d, g}
- Dequeue(Q): lấy k khỏi Q, lúc này Q = {d, g}.
  - Từ k: đến được b, c, e. Nhưng tất cả các đỉnh này đã được thăm.
- Dequeue (Q): lấy d khỏi Q, lúc này Q = {g}
  - Từ d: đến được c, g. Nhưng tất cả các đỉnh này đã được thăm.
- Dequeue(Q): lấy g khỏi Q, lúc này Q = empty
  - Từ g: đến được d, c. Nhưng tất cả các đỉnh này đã được thăm.
- Q lúc này rỗng. Tất cả các đỉnh của đồ thị đã được thăm. Thuật toán kết thúc.

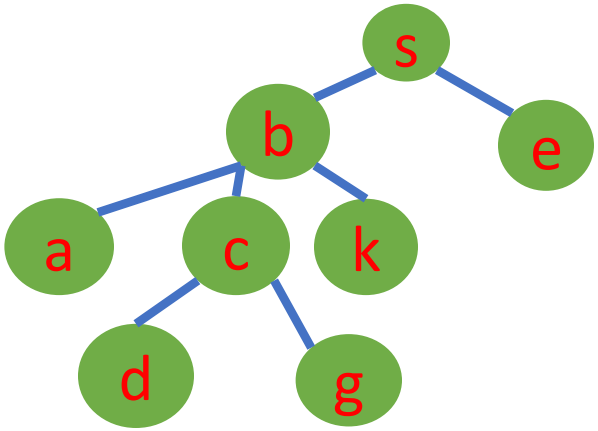
Tìm kiếm theo chiều rộng (Breadth-first Search)

```
BFS(s) {
// Tìm kiếm theo chiều rộng bắt đầu từ đỉnh s
        visited[s] ← 1; //đã thăm
        Q ← ∅; enqueue(Q,s); // Nạp s vào Q
        while (Q ≠ ∅)
        {
             u ← dequeue(Q);  // Lấy u khỏi Q
             for  v ∈Adj[u]
                  if (visited[v] == 0) //chưa thăm
                  {
                       visited[v] ← 1; //đã thăm
                       enqueue(Q,v)   // Nạp v vào Q
                  }
        }
}
main ()
{
        for  s  ∈ V  // Khởi tạo
              visited[s] ← 0;

        for  s  ∈ V
          if (visited[s]==0)  BFS(s);
}
```



G=(V, E)



Phân tích độ phức tạp tính toán của BFS

```
BFS(s) {
// Tìm kiếm theo chiều rộng bắt đầu từ đỉnh s
        visited[s] ← 1; //đã thăm
        Q ← ∅; enqueue(Q,s); // Nạp s vào Q
        while (Q ≠ ∅)
        {
            u ← dequeue(Q);  // Lấy u khỏi Q
            for  v ∈Adj[u]
                if (visited[v] == 0) //chưa thăm
                {
                    visited[v] ← 1; //đã thăm
                    enqueue(Q,v)   // Nạp v vào Q
                }
        }
}
main ()
{
        for  s  ∈ V  // Khởi tạo
             visited[s] ← 0;

        for  s  ∈ V
          if (visited[s]==0)  BFS(s);
}
```

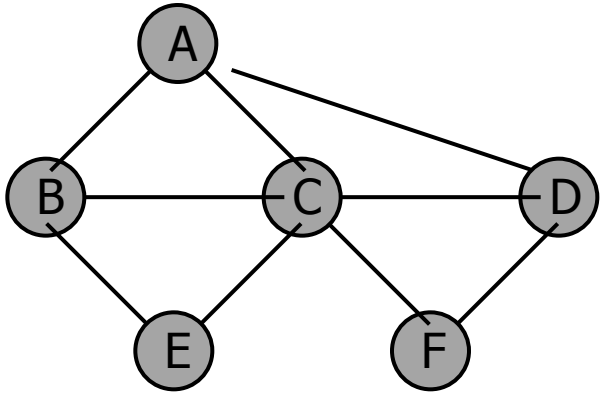
- Việc khởi tạo đòi hỏi $O(|V|)$.
- Vòng lặp duyệt
  - Mỗi đỉnh được nạp vào và loại ra khỏi hàng đợi một lần, mỗi thao tác đòi hỏi thời gian $O(1)$. Như vậy tổng thời gian làm việc với hàng đợi là $O(|V|)$.
  - Danh sách kề của mỗi đỉnh được duyệt qua đúng một lần. Tổng độ dài của tất cả các danh sách kề là $O(|E|)$.
- Tổng cộng ta có thời gian tính của BFS(s) là $O(|V|+|E|)$, là tuyến tính theo kích thước của danh sách kề biểu diễn đồ thị.

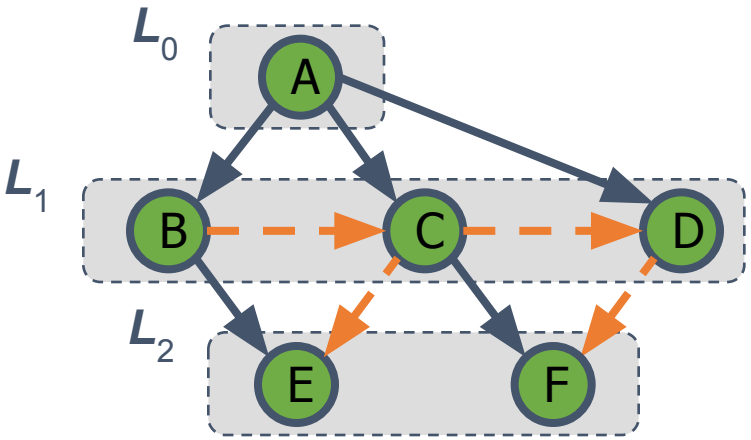
Cây BFS(s)

- Đối với đồ thị $G = (V, E)$ và đỉnh s. Sau khi thực hiện BFS(s), thu được đồ thị con $G_\pi = (V_\pi, E_\pi)$ trong đó
  - $V_\pi = \{v \in V : \text{truoc}[v] \neq \text{NULL}\} \cup \{s\}$
  - $E_\pi = \{(\text{truoc}[v], v) \in E : v \in V_\pi \setminus \{s\}\}$
- $G_\pi = (V_\pi, E_\pi)$ là cây và được gọi là cây BFS(s)
- Các cạnh trong E_π được gọi là cạnh của cây. $|E_\pi| = |V_\pi| - 1$.
- BFS(s) cho phép đến thăm tất cả các đỉnh đạt tới được từ s.
- Trình tự thăm các đỉnh khi thực hiện BFS(s): Đầu tiên đến thăm các đỉnh đạt được từ s bởi đường đi qua 1 cạnh, sau đó là thăm các đỉnh đạt được từ s bởi đường đi qua 2 cạnh, …Do đó nếu đỉnh t được thăm trong BFS(s) thì nó sẽ được thăm theo đường đi ngắn nhất theo số cạnh.

BFS – Loang trên đồ thị

- Thứ tự thăm đỉnh nhờ thực hiện BFS(A)





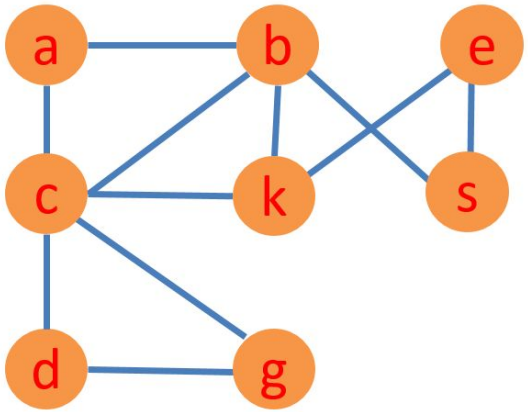
Tìm kiếm theo chiều sâu

Depth-first Search (DFS)

Tìm kiếm theo chiều sâu (Depth-First Search)

```
(* Main Program*)
1. for s ∈ V
2.     visited[s] ← false
3. for s ∈ V
4.     if (visited[s] == false)
5.         DFS(s)
```

```
DFS(s)
1.     visited[s] ← true  // Thăm đỉnh s
2.     for each v ∈ Adj[s]
3.         if (visited[v] == false)
4.             DFS(v)
```



G=(V, E)

Phân tích độ phức tạp tính toán của BFS

```
(* Main Program*)
1.  for s ∈ V
2.      visited[s] ← false
3   for s ∈ V
4.      if (visited[s] == false)
5.          DFS(s)
```

```
DFS(s)
1.      visited[s] ← true  // Thăm đỉnh s
2.      for each v ∈ Adj[s]
3.          if (visited[v] == false)
4.              DFS(v)
```

- Ở `main`: vòng lặp trên các dòng 1-2 và 3-5 đòi hỏi thời gian $O(|V|)$, chưa tính thời gian thực hiện lệnh `DFS(s)`.
- Vòng lặp trong `DFS(s)` thực hiện việc duyệt cạnh của đồ thị:
  - Các dòng 3-4 của `DFS(s)` sẽ thực hiện |Adj[s]| lần
  - Mỗi cạnh được duyệt qua đúng một lần nếu đồ thị là có hướng và 2 lần nếu đồ thị là vô hướng

Vậy thời gian tổng cộng của `DFS(s)` trong chương trình chính là $\sum_{s \in V} |\text{Adj}[s]| = O(|E|)$

- Do đó, thời gian của DFS là $O(|V|+|E|)$.

DFS: Nếu muốn đưa ra đường đi từ s đến các đỉnh còn lại của đồ thị

```
(* Main Program*)
1. for s ∈ V
2.     visited[s] ← false
3. for s ∈ V
4.     if (visited[s] == false)
5.         DFS(s)
```

```
DFS(s)
1.     visited[s] ← true  // Thăm đỉnh s
2.     for each v ∈ Adj[s]
3.         if (visited[v] == false)
4.             DFS(v)
```

```
void main()
1.     for each s ∈ V
2.         truoc[s] = NULL;
3.         visited[s] = false;
4.     DFS(s);
```

```
DFS(s)
1.     visited[s]= true; //Thăm đỉnh s
2.     for each v ∈ Adj[s]
3.         if (visited[v] == false) {
4.             truoc[v] ← s;
5.             DFS(v);
6.         }
```

DFS: Phân loại cạnh

```
void main()
1.      for each s ∈ V
2.              truoc[s] = NULL;
3.              visited[s] = false;
4.      time = 0
5.      for each s ∈ V
6.          if (visited[s] == false) DFS(s);
```

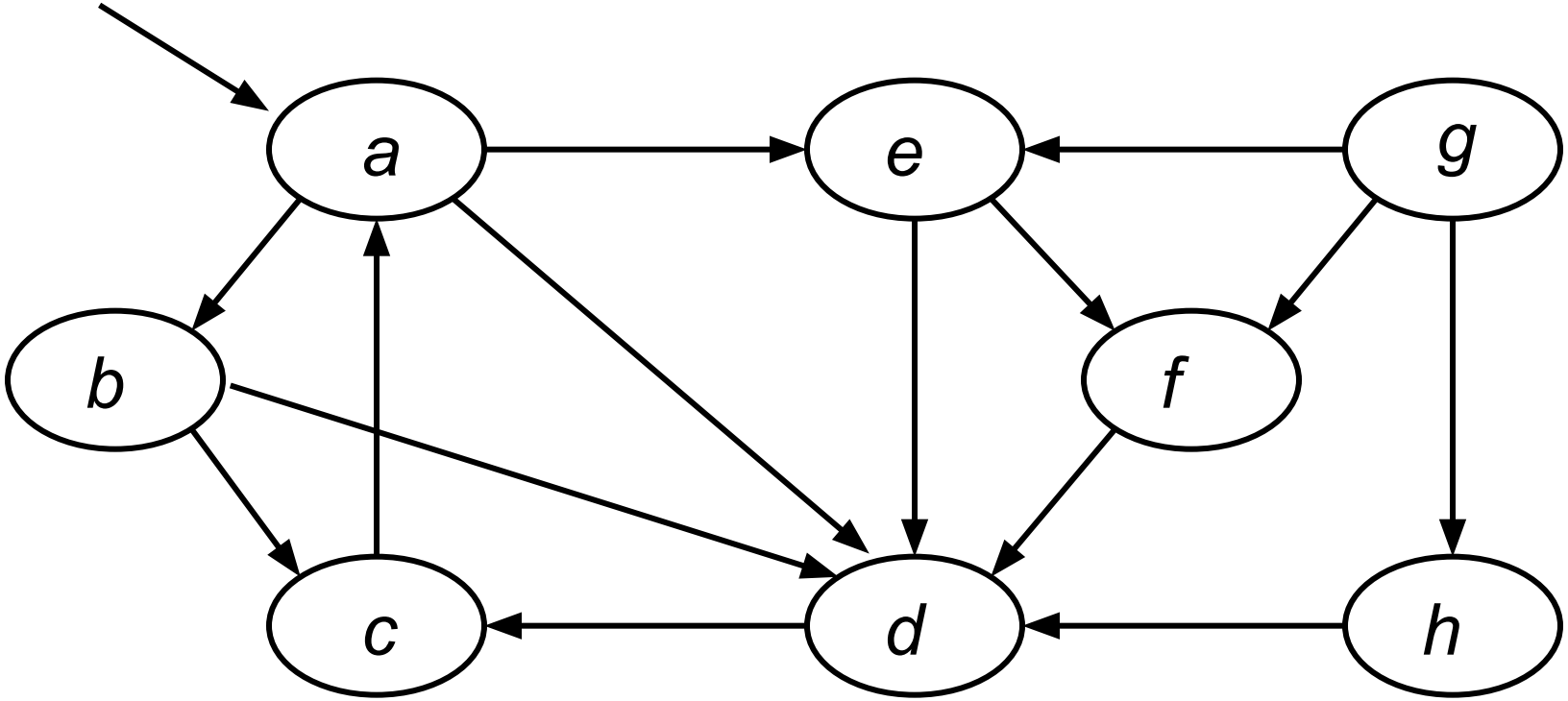
```
DFS(s)
1.      visited[s]= true; //Thăm đỉnh s
2.      time = time +1
3.      d[s] = time
4.      for each v ∈ Adj[s]
5.              if (visited[v] == false) {
6.                      truoc[v] ← s;
7.                      DFS(v);
8.              }
9.      time = time +1
10.     f[s] = time
```

Với mỗi đỉnh s của đồ thị:

- `d[s]` : thời điểm bắt đầu thăm đỉnh s
- `f[s]` : thời điểm kết thúc thăm đỉnh s

Ví dụ: DFS

Đỉnh xuất phát tìm kiếm
(Source vertex)

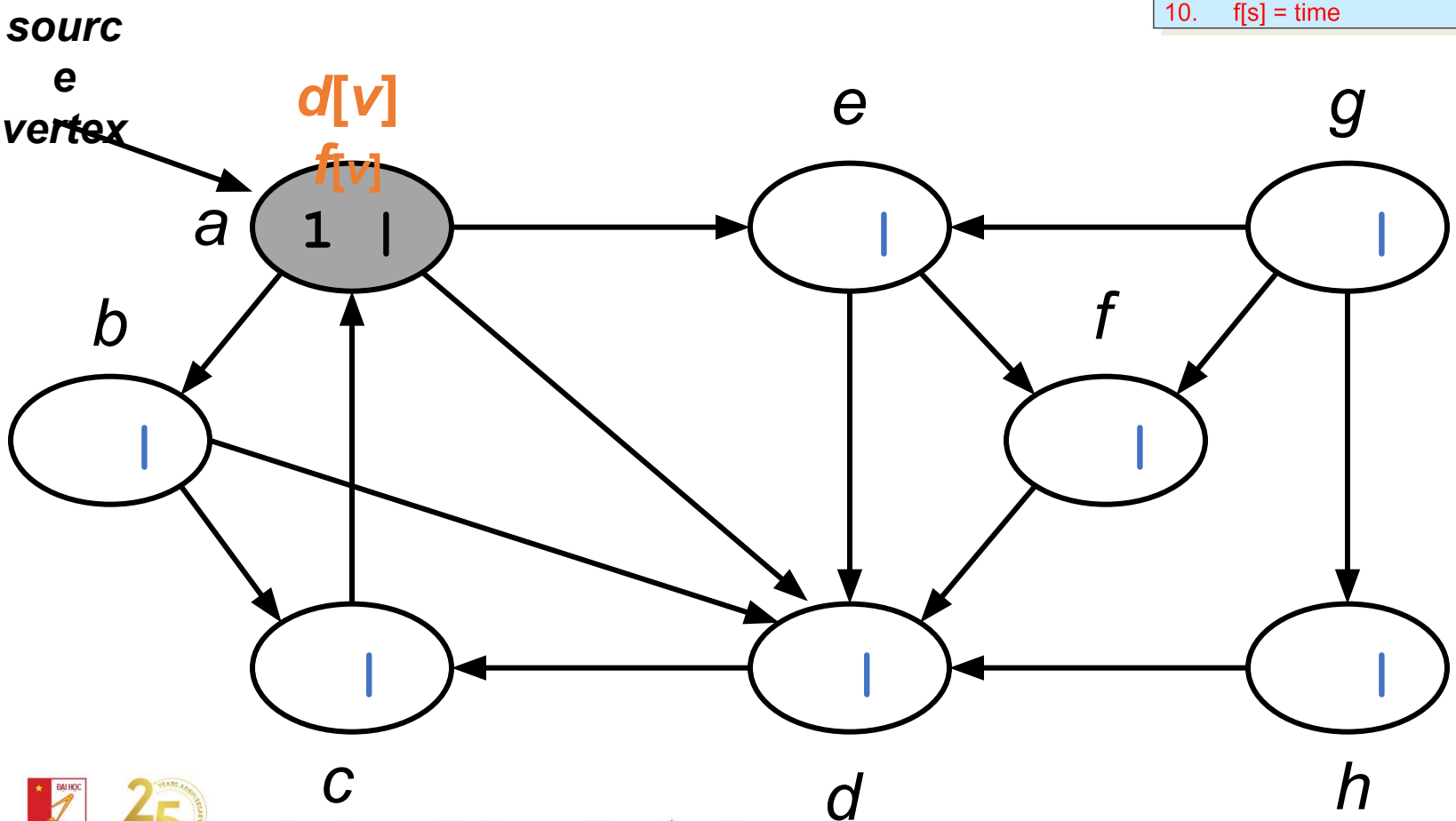


Để hoạt động của thuật toán là xác định, giả thiết rằng ta duyệt các đỉnh trong danh sách kề của một đỉnh theo thứ tự từ điển

Ví dụ: DFS

```
void main()
1.    for each s ∈ V
2.          truoc[s] = NULL;
3.          visited[s] = false;
4.    time = 0
5.    for each s ∈ V
6.       if (visited[s] == false) DFS(s);
```

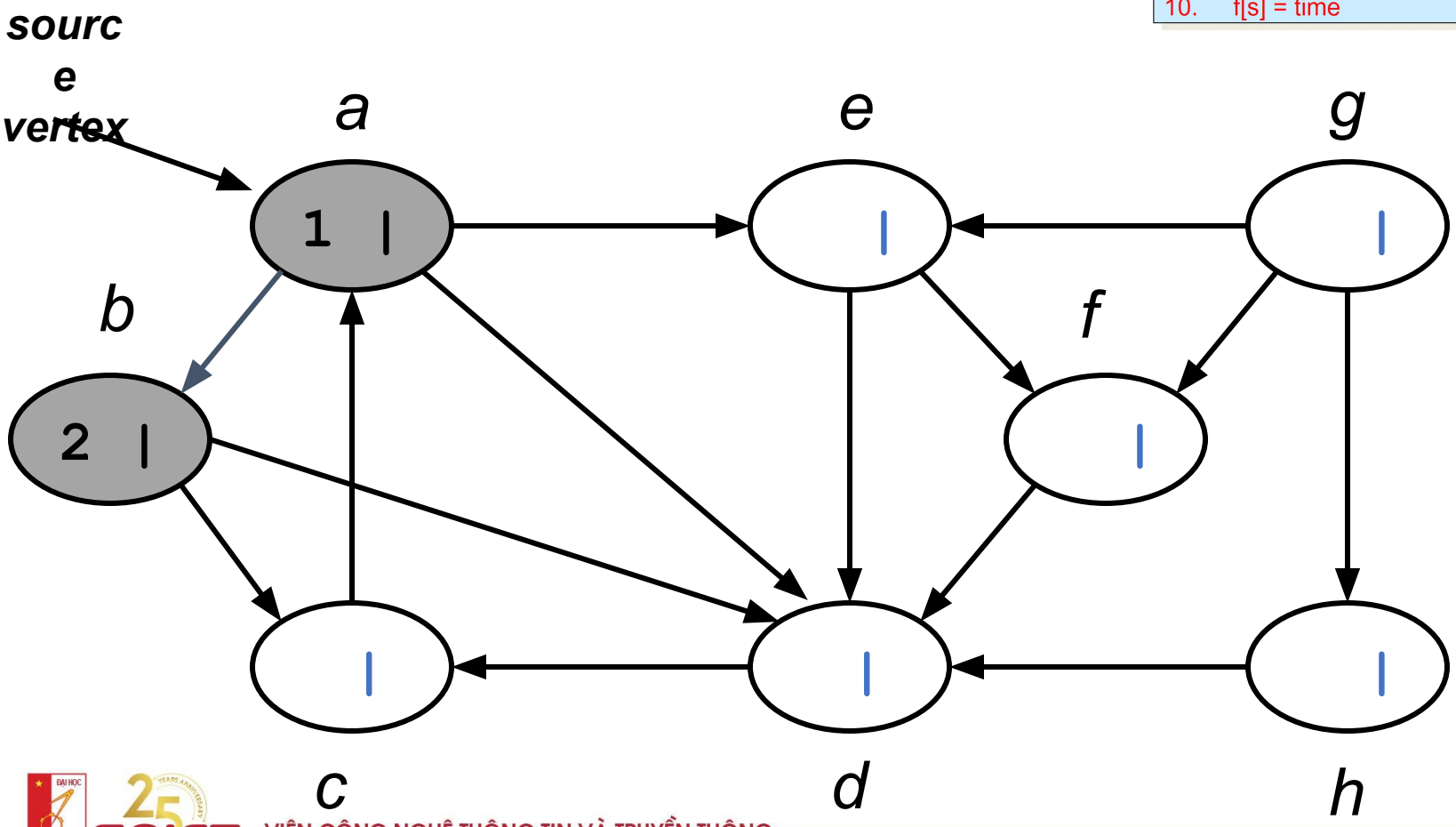
```
DFS(s)
1.    visited[s]= true; //Thăm đỉnh s
2.    time = time +1
3.    d[s] = time
4.    for each v ∈ Adj[s]
5.          if (visited[v] == false) {
6.                truoc[v] ← s;
7.                DFS(v);
8.          }
9.    time = time +1
10.   f[s] = time
```



Ví dụ: DFS

```
void main()
1.    for each s ∈ V
2.          truoc[s] = NULL;
3.          visited[s] = false;
4.    time = 0
5.    for each s ∈ V
6.       if (visited[s] == false) DFS(s);
```

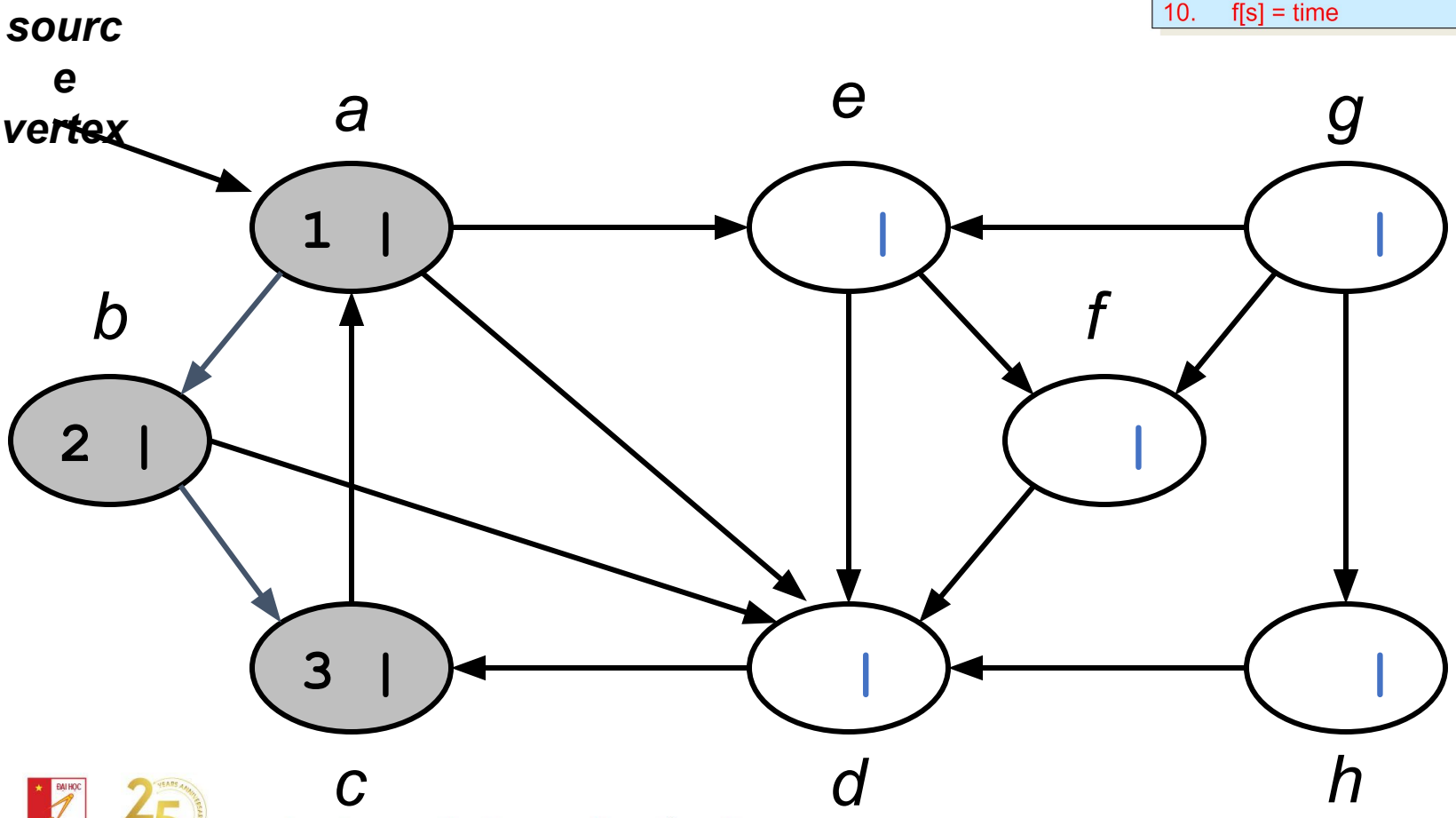
```
DFS(s)
1.    visited[s]= true; //Thăm đỉnh s
2.    time = time +1
3.    d[s] = time
4.    for each v ∈ Adj[s]
5.          if (visited[v] == false) {
6.                truoc[v] ← s;
7.                DFS(v);
8.          }
9.    time = time +1
10.   f[s] = time
```



Ví dụ: DFS

```
void main()
1.    for each s ∈ V
2.          truoc[s] = NULL;
3.          visited[s] = false;
4.    time = 0
5.    for each s ∈ V
6.       if (visited[s] == false) DFS(s);
```

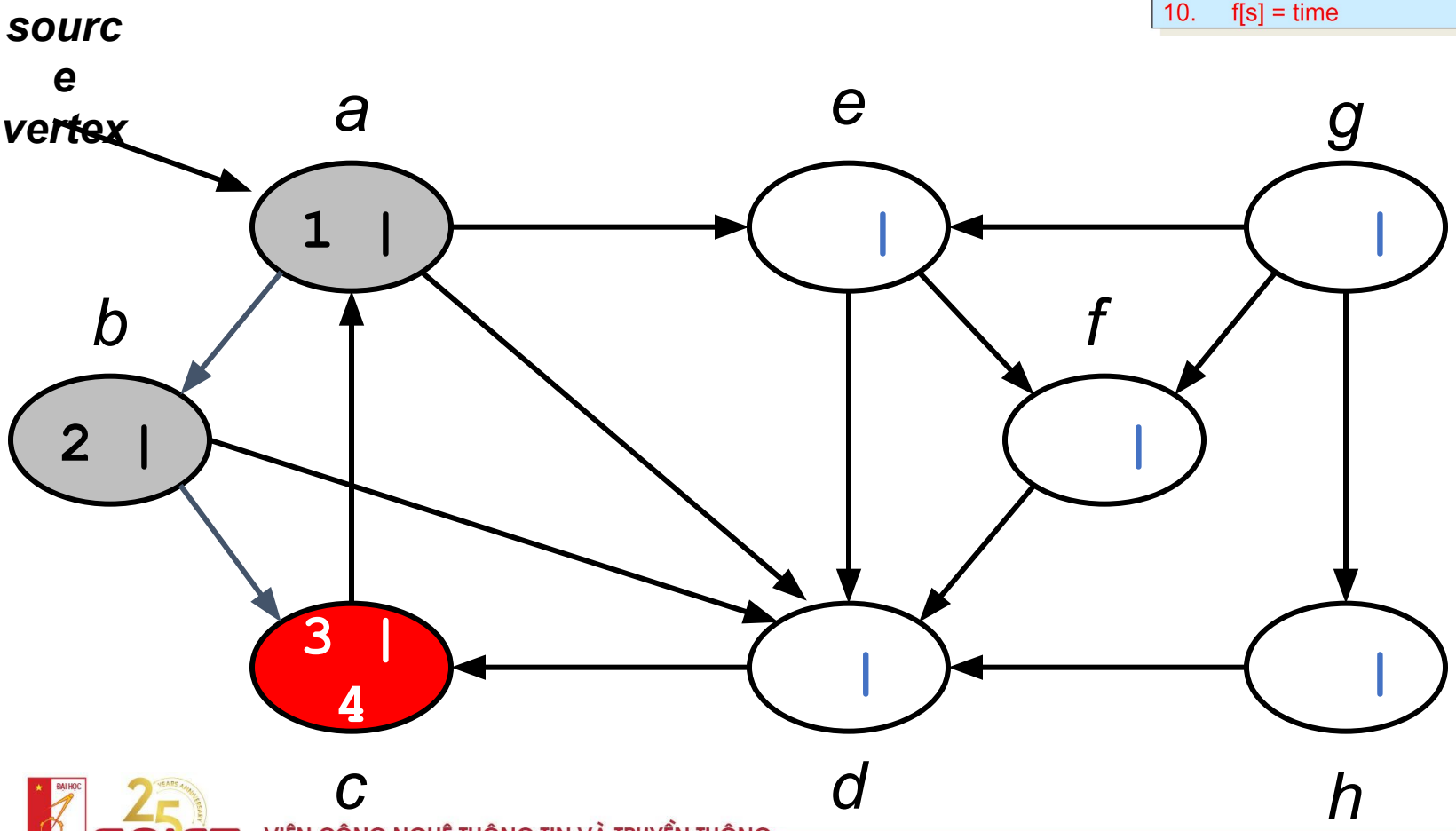
```
DFS(s)
1.    visited[s]= true; //Thăm đỉnh s
2.    time = time +1
3.    d[s] = time
4.    for each v ∈ Adj[s]
5.          if (visited[v] == false) {
6.                truoc[v] ← s;
7.                DFS(v);
8.          }
9.    time = time +1
10.   f[s] = time
```



Ví dụ: DFS

```
void main()
1.    for each s ∈ V
2.          truoc[s] = NULL;
3.          visited[s] = false;
4.    time = 0
5.    for each s ∈ V
6.       if (visited[s] == false) DFS(s);
```

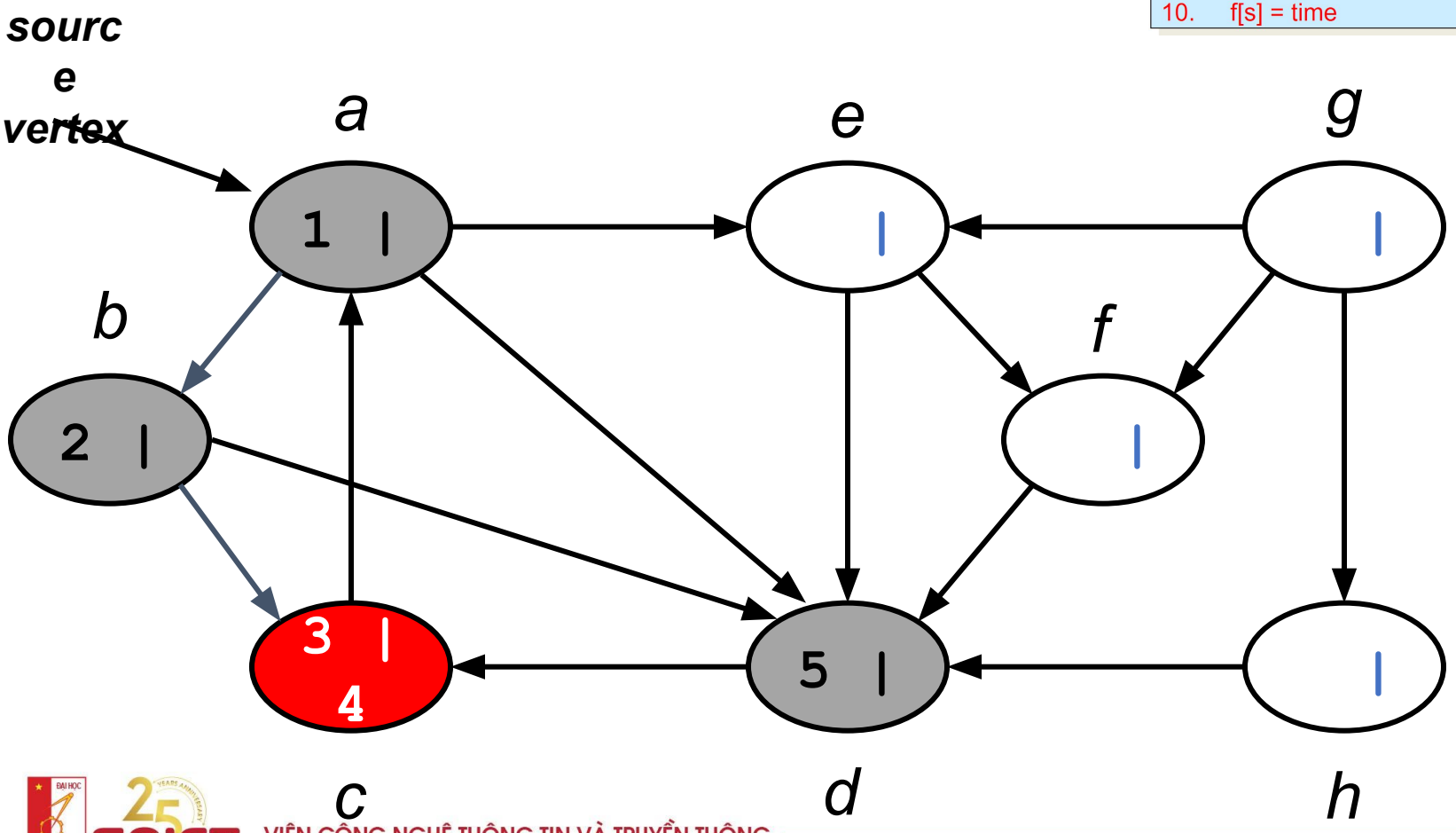
```
DFS(s)
1.    visited[s]= true; //Thăm đỉnh s
2.    time = time +1
3.    d[s] = time
4.    for each v ∈ Adj[s]
5.          if (visited[v] == false) {
6.                truoc[v] ← s;
7.                DFS(v);
8.          }
9.    time = time +1
10.   f[s] = time
```



Ví dụ: DFS

```
void main()
1.    for each s ∈ V
2.          truoc[s] = NULL;
3.          visited[s] = false;
4.    time = 0
5.    for each s ∈ V
6.       if (visited[s] == false) DFS(s);
```

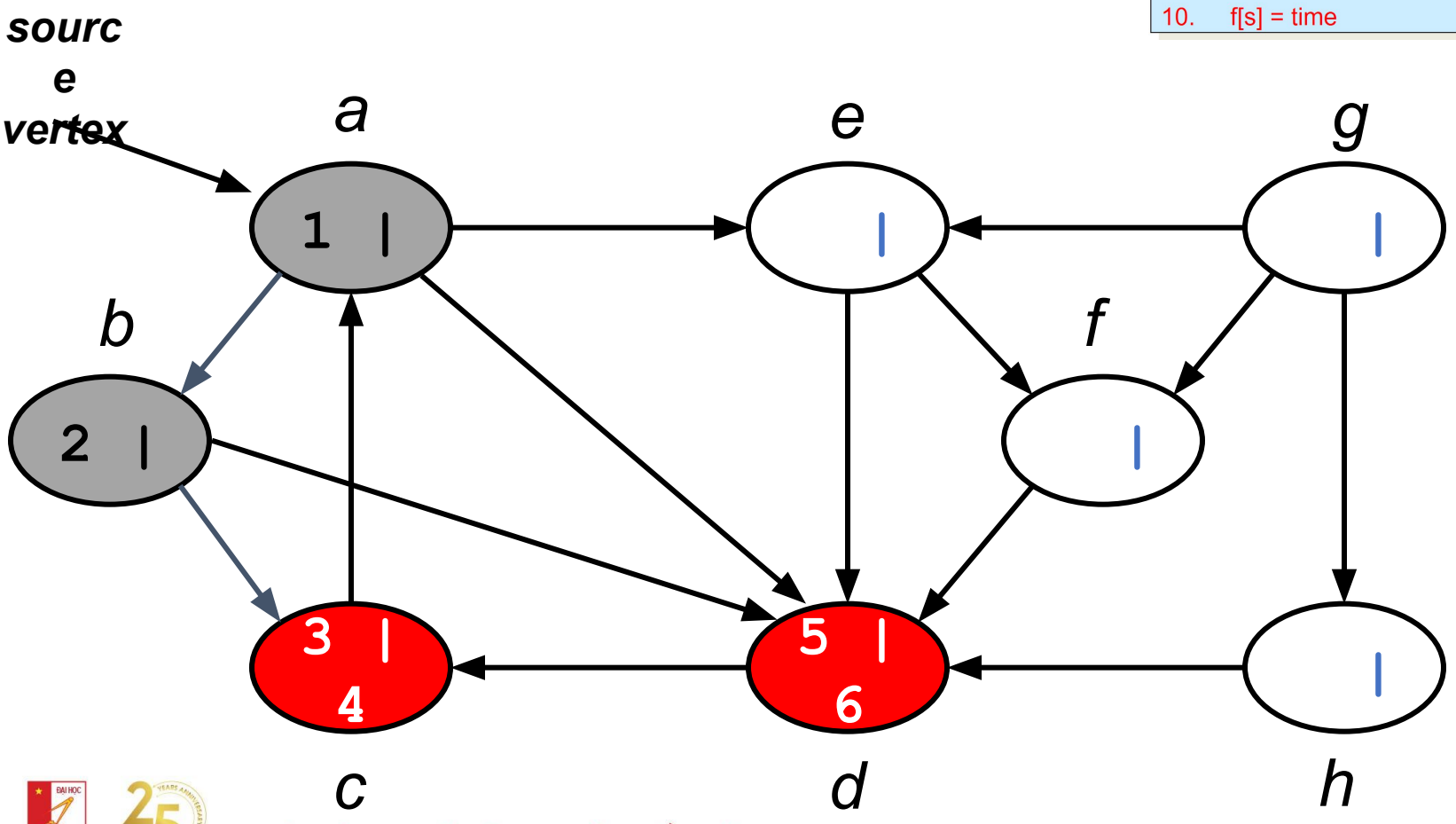
```
DFS(s)
1.    visited[s]= true; //Thăm đỉnh s
2.    time = time +1
3.    d[s] = time
4.    for each v ∈ Adj[s]
5.          if (visited[v] == false) {
6.                truoc[v] ← s;
7.                DFS(v);
8.          }
9.    time = time +1
10.   f[s] = time
```



Ví dụ: DFS

```
void main()
1.    for each s ∈ V
2.          truoc[s] = NULL;
3.          visited[s] = false;
4.    time = 0
5.    for each s ∈ V
6.       if (visited[s] == false) DFS(s);
```

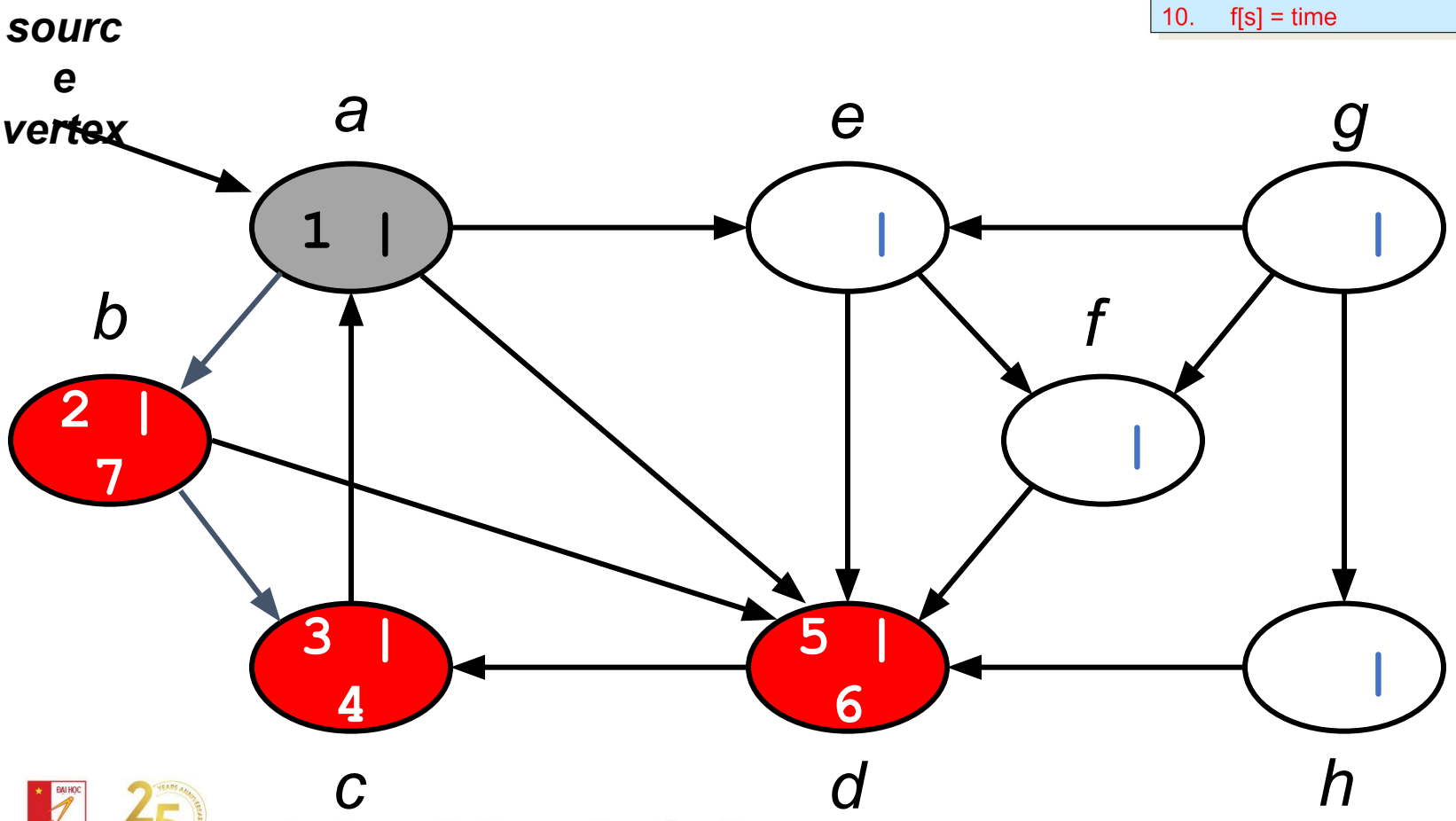
```
DFS(s)
1.    visited[s]= true; //Thăm đỉnh s
2.    time = time +1
3.    d[s] = time
4.    for each v ∈ Adj[s]
5.          if (visited[v] == false) {
6.                truoc[v] ← s;
7.                DFS(v);
8.          }
9.    time = time +1
10.   f[s] = time
```



Ví dụ: DFS

```
void main()
1.    for each s ∈ V
2.          truoc[s] = NULL;
3.          visited[s] = false;
4.    time = 0
5.    for each s ∈ V
6.       if (visited[s] == false) DFS(s);
```

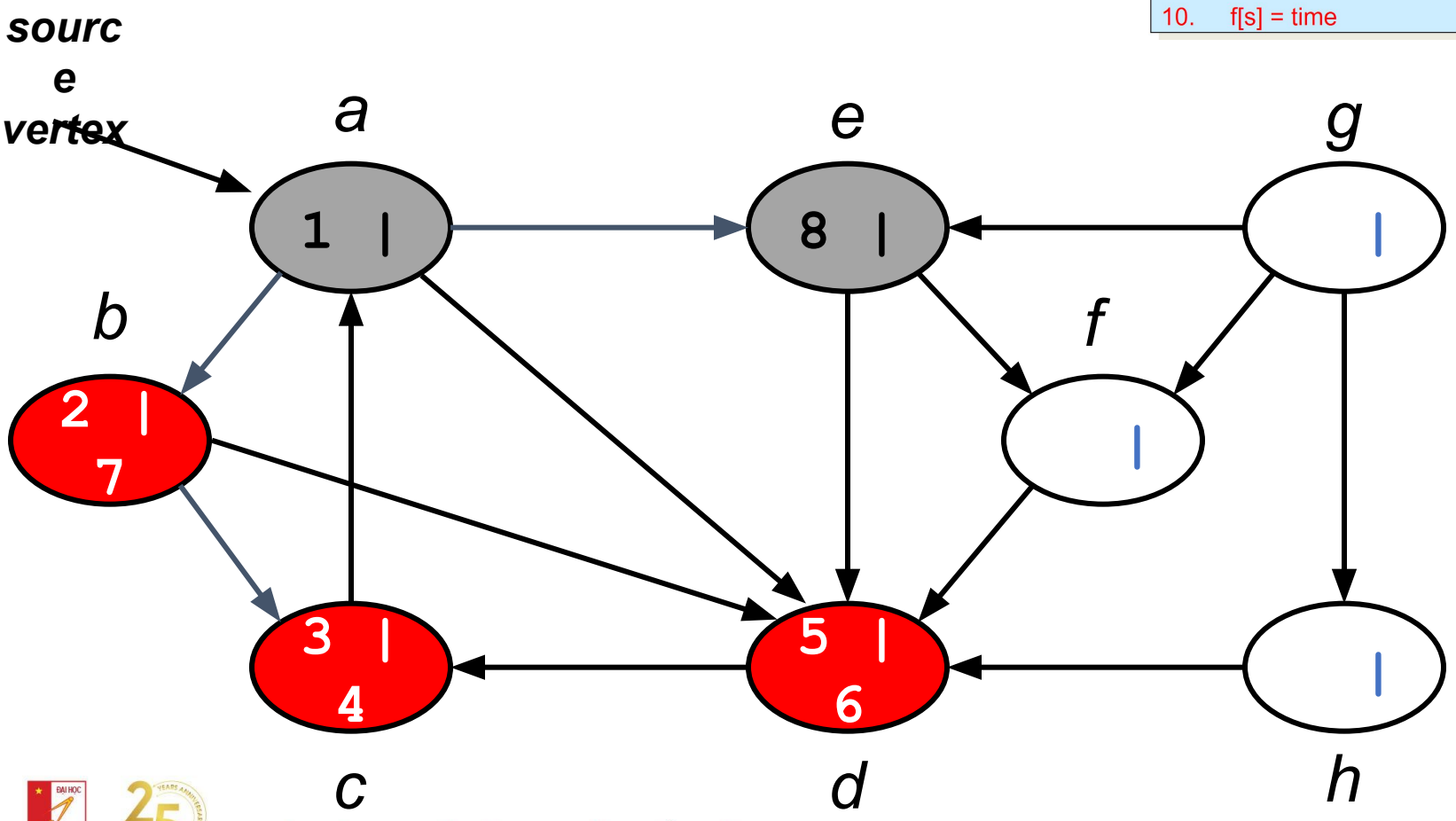
```
DFS(s)
1.    visited[s]= true; //Thăm đỉnh s
2.    time = time +1
3.    d[s] = time
4.    for each v ∈ Adj[s]
5.          if (visited[v] == false) {
6.                truoc[v] ← s;
7.                DFS(v);
8.          }
9.    time = time +1
10.   f[s] = time
```



Ví dụ: DFS

```
void main()
1.    for each s ∈ V
2.          truoc[s] = NULL;
3.          visited[s] = false;
4.    time = 0
5.    for each s ∈ V
6.       if (visited[s] == false) DFS(s);
```

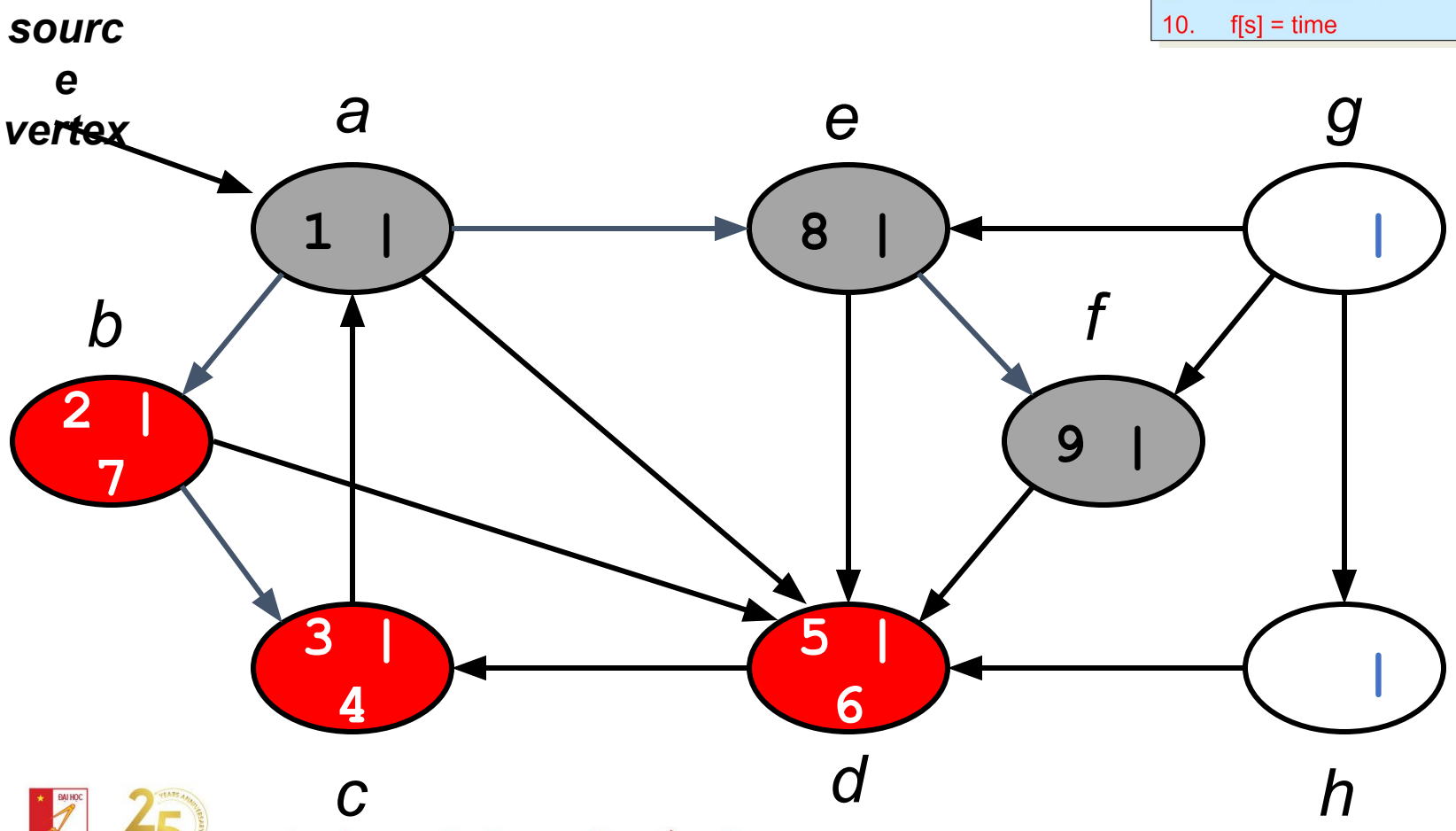
```
DFS(s)
1.    visited[s]= true; //Thăm đỉnh s
2.    time = time +1
3.    d[s] = time
4.    for each v ∈ Adj[s]
5.          if (visited[v] == false) {
6.                truoc[v] ← s;
7.                DFS(v);
8.          }
9.    time = time +1
10.   f[s] = time
```



Ví dụ: DFS

```
void main()
1.    for each s ∈ V
2.          truoc[s] = NULL;
3.          visited[s] = false;
4.    time = 0
5.    for each s ∈ V
6.       if (visited[s] == false) DFS(s);
```

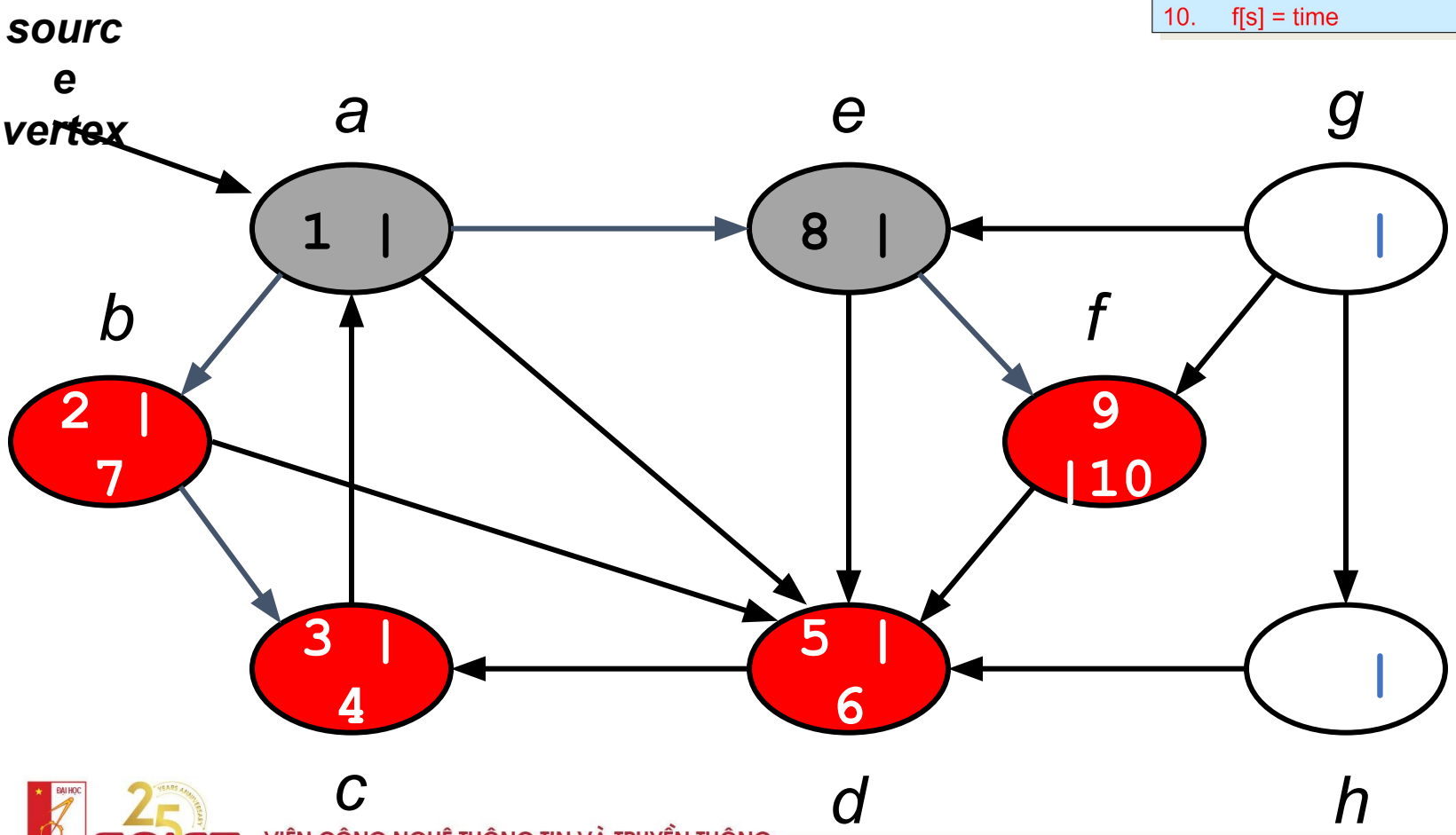
```
DFS(s)
1.    visited[s]= true; //Thăm đỉnh s
2.    time = time +1
3.    d[s] = time
4.    for each v ∈ Adj[s]
5.          if (visited[v] == false) {
6.                truoc[v] ← s;
7.                DFS(v);
8.          }
9.    time = time +1
10.   f[s] = time
```



Ví dụ: DFS

```
void main()
1.    for each s ∈ V
2.          truoc[s] = NULL;
3.          visited[s] = false;
4.    time = 0
5.    for each s ∈ V
6.       if (visited[s] == false) DFS(s);
```

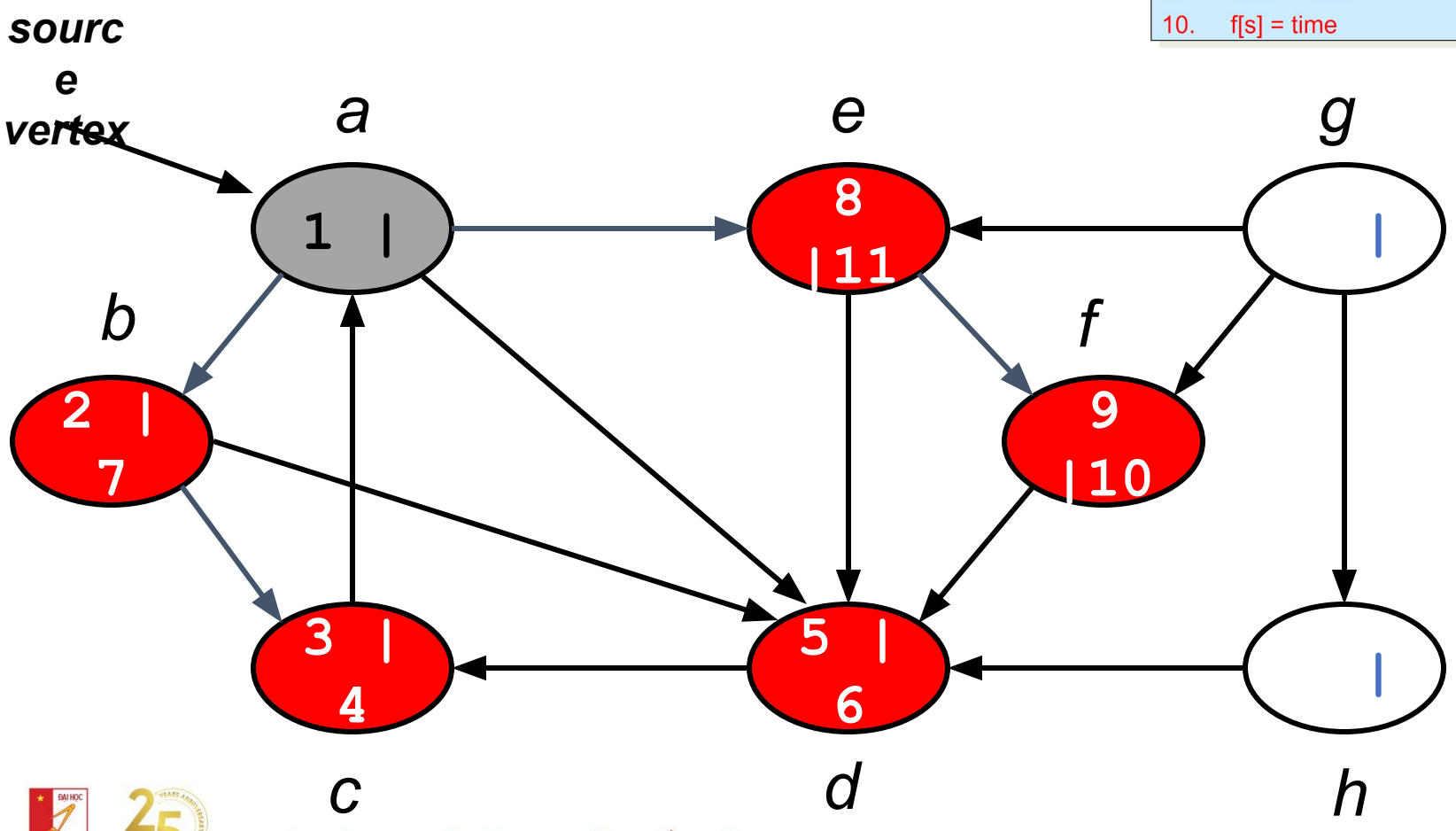
```
DFS(s)
1.    visited[s]= true; //Thăm đỉnh s
2.    time = time +1
3.    d[s] = time
4.    for each v ∈ Adj[s]
5.          if (visited[v] == false) {
6.                truoc[v] ← s;
7.                DFS(v);
8.          }
9.    time = time +1
10.   f[s] = time
```



Ví dụ: DFS

```
void main()
1.    for each s ∈ V
2.          truoc[s] = NULL;
3.          visited[s] = false;
4.    time = 0
5.    for each s ∈ V
6.       if (visited[s] == false) DFS(s);
```

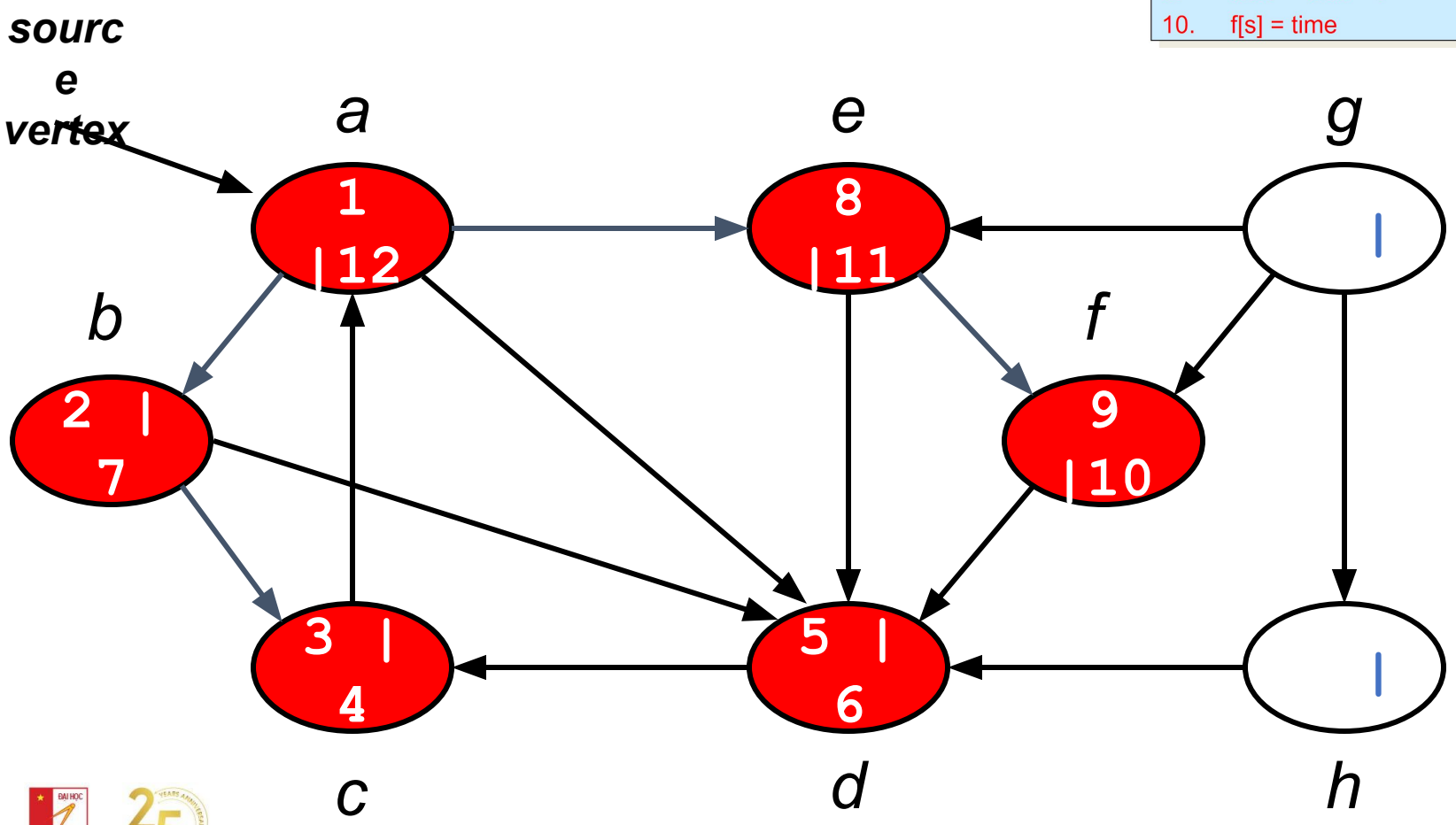
```
DFS(s)
1.    visited[s]= true; //Thăm đỉnh s
2.    time = time +1
3.    d[s] = time
4.    for each v ∈ Adj[s]
5.          if (visited[v] == false) {
6.                truoc[v] ← s;
7.                DFS(v);
8.          }
9.    time = time +1
10.   f[s] = time
```



Ví dụ: DFS

```
void main()
1.    for each s ∈ V
2.          truoc[s] = NULL;
3.          visited[s] = false;
4.    time = 0
5.    for each s ∈ V
6.       if (visited[s] == false) DFS(s);
```

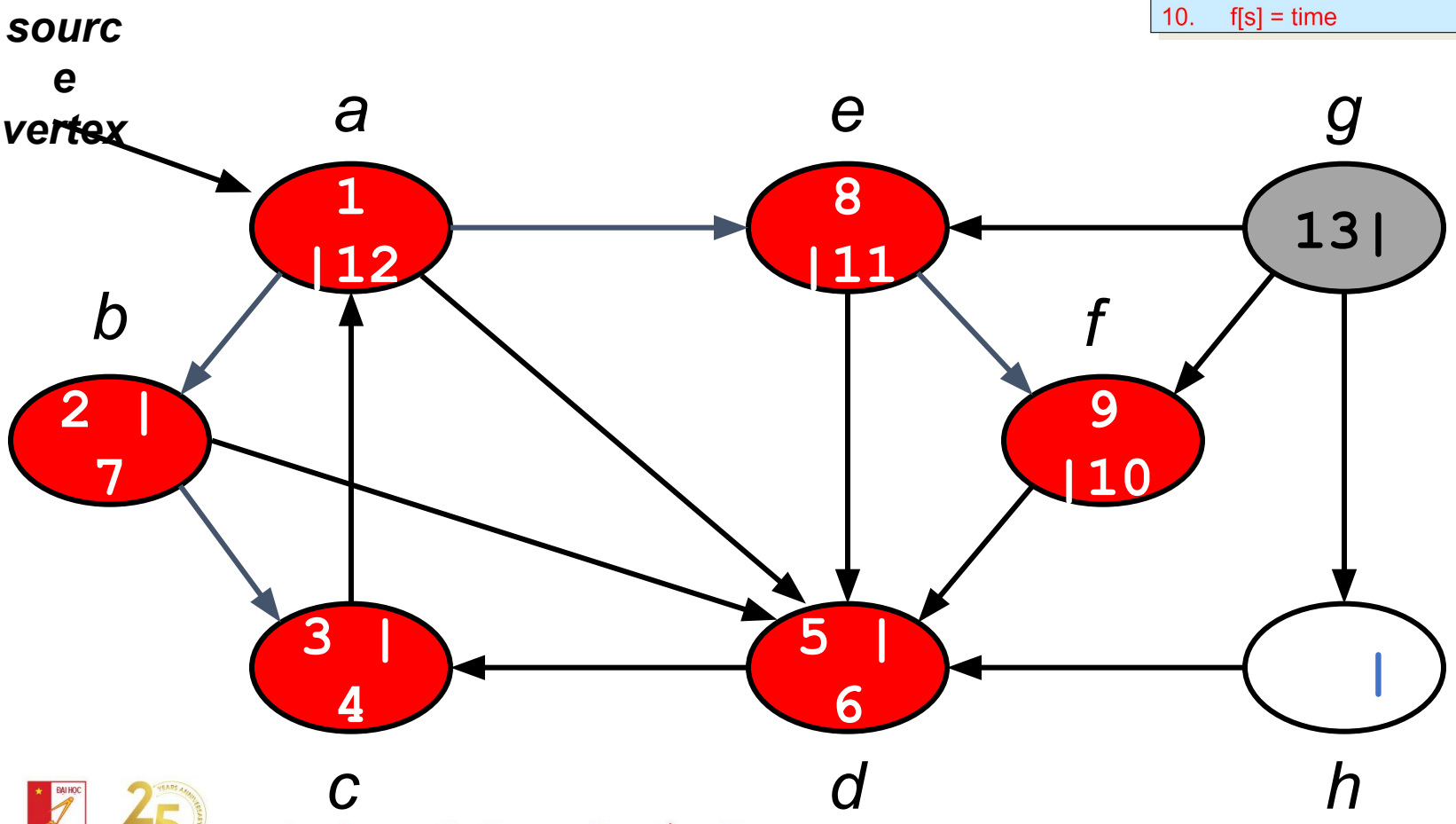
```
DFS(s)
1.    visited[s]= true; //Thăm đỉnh s
2.    time = time +1
3.    d[s] = time
4.    for each v ∈ Adj[s]
5.          if (visited[v] == false) {
6.                truoc[v] ← s;
7.                DFS(v);
8.          }
9.    time = time +1
10.   f[s] = time
```



Ví dụ: DFS

```
void main()
1.    for each s ∈ V
2.          truoc[s] = NULL;
3.          visited[s] = false;
4.    time = 0
5.    for each s ∈ V
6.       if (visited[s] == false) DFS(s);
```

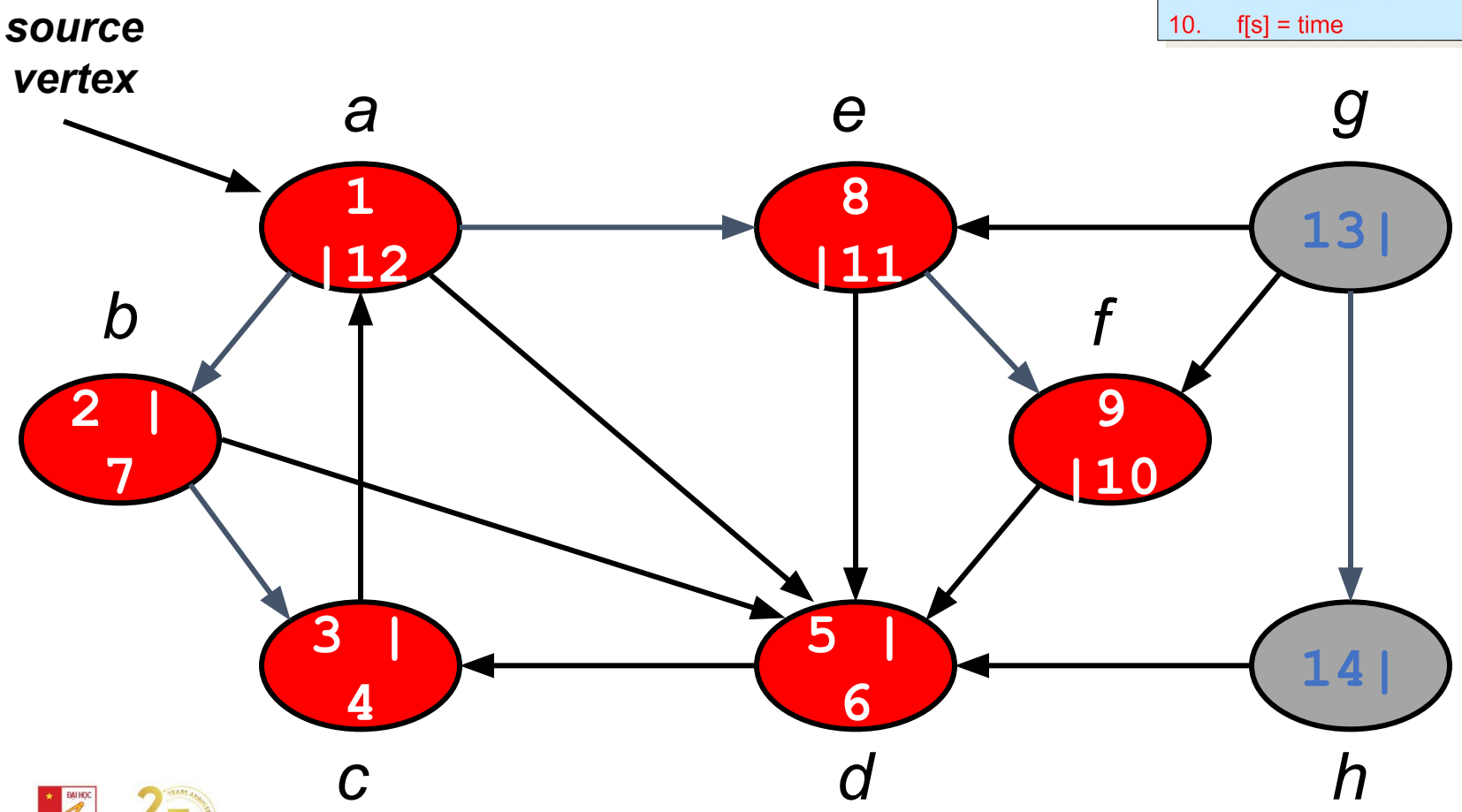
```
DFS(s)
1.    visited[s]= true; //Thăm đỉnh s
2.    time = time +1
3.    d[s] = time
4.    for each v ∈ Adj[s]
5.          if (visited[v] == false) {
6.                truoc[v] ← s;
7.                DFS(v);
8.          }
9.    time = time +1
10.   f[s] = time
```



Ví dụ: DFS

```
void main()
1.    for each s ∈ V
2.          truoc[s] = NULL;
3.          visited[s] = false;
4.    time = 0
5.    for each s ∈ V
6.       if (visited[s] == false) DFS(s);
```

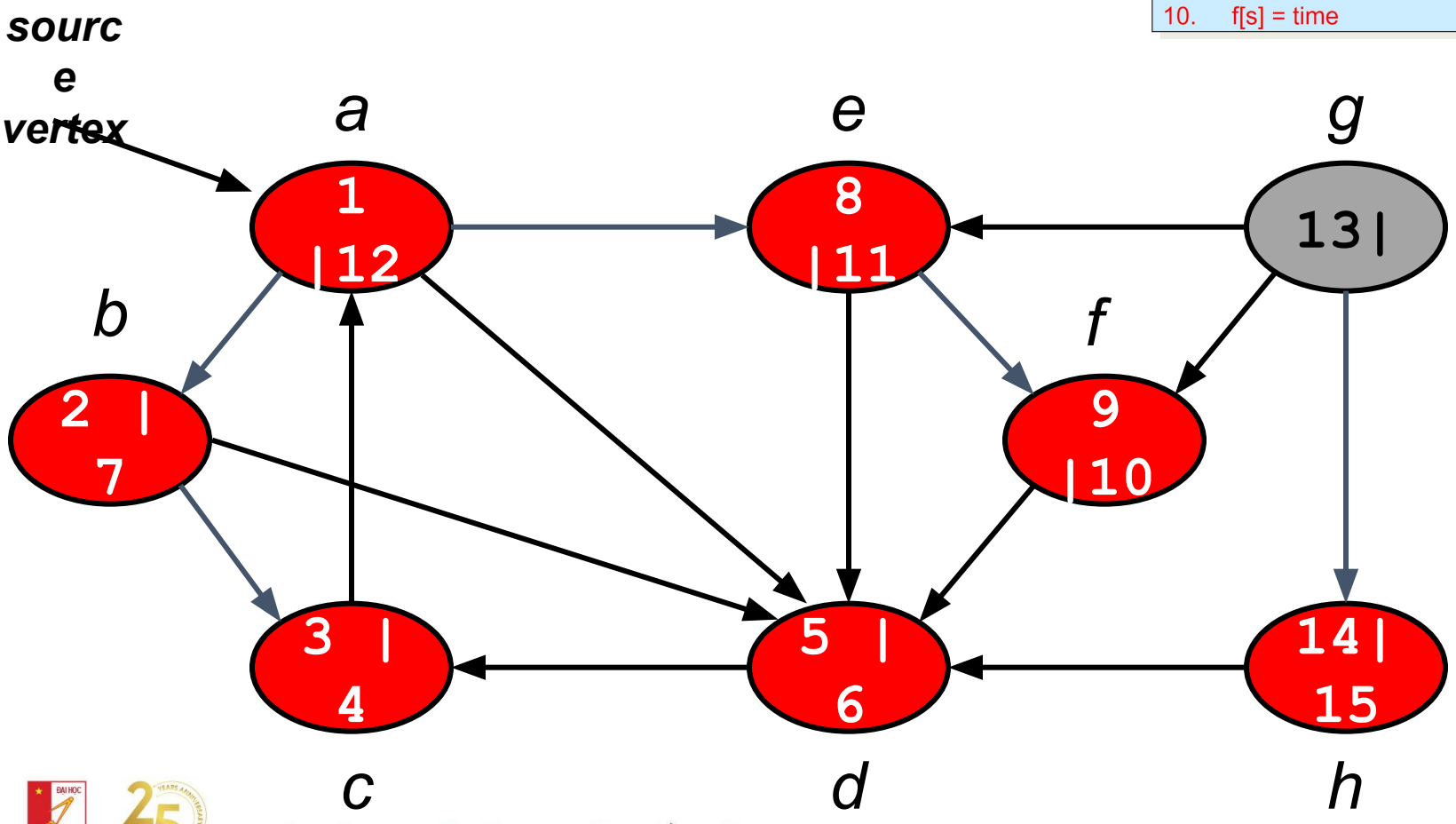
```
DFS(s)
1.    visited[s]= true; //Thăm đỉnh s
2.    time = time +1
3.    d[s] = time
4.    for each v ∈ Adj[s]
5.          if (visited[v] == false) {
6.                truoc[v] ← s;
7.                DFS(v);
8.          }
9.    time = time +1
10.   f[s] = time
```



Ví dụ: DFS

```
void main()
1.    for each s ∈ V
2.          truoc[s] = NULL;
3.          visited[s] = false;
4.    time = 0
5.    for each s ∈ V
6.       if (visited[s] == false) DFS(s);
```

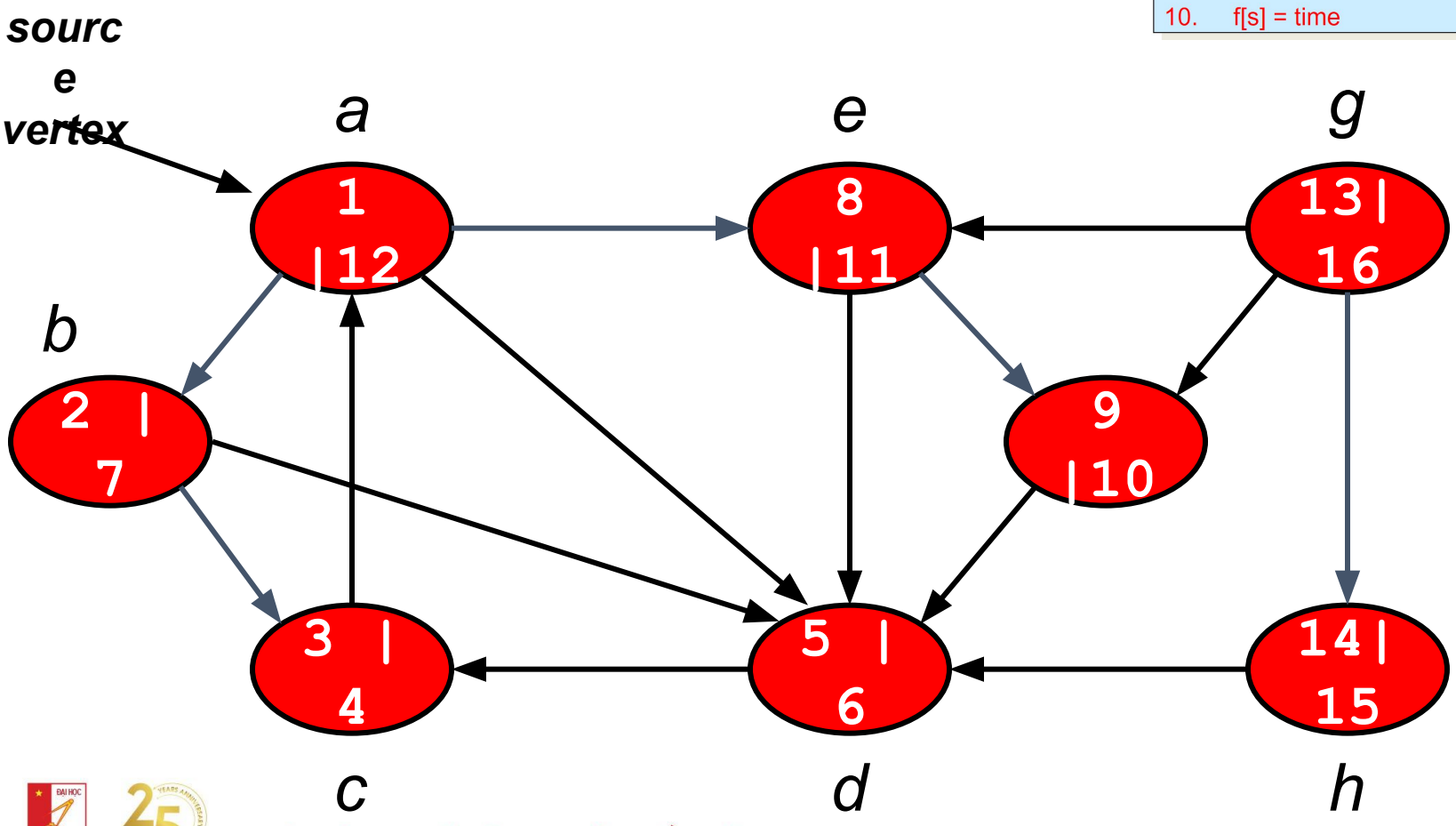
```
DFS(s)
1.    visited[s]= true; //Thăm đỉnh s
2.    time = time +1
3.    d[s] = time
4.    for each v ∈ Adj[s]
5.          if (visited[v] == false) {
6.                truoc[v] ← s;
7.                DFS(v);
8.          }
9.    time = time +1
10.   f[s] = time
```



Ví dụ: DFS

```
void main()
1.    for each s ∈ V
2.          truoc[s] = NULL;
3.          visited[s] = false;
4.    time = 0
5.    for each s ∈ V
6.       if (visited[s] == false) DFS(s);
```

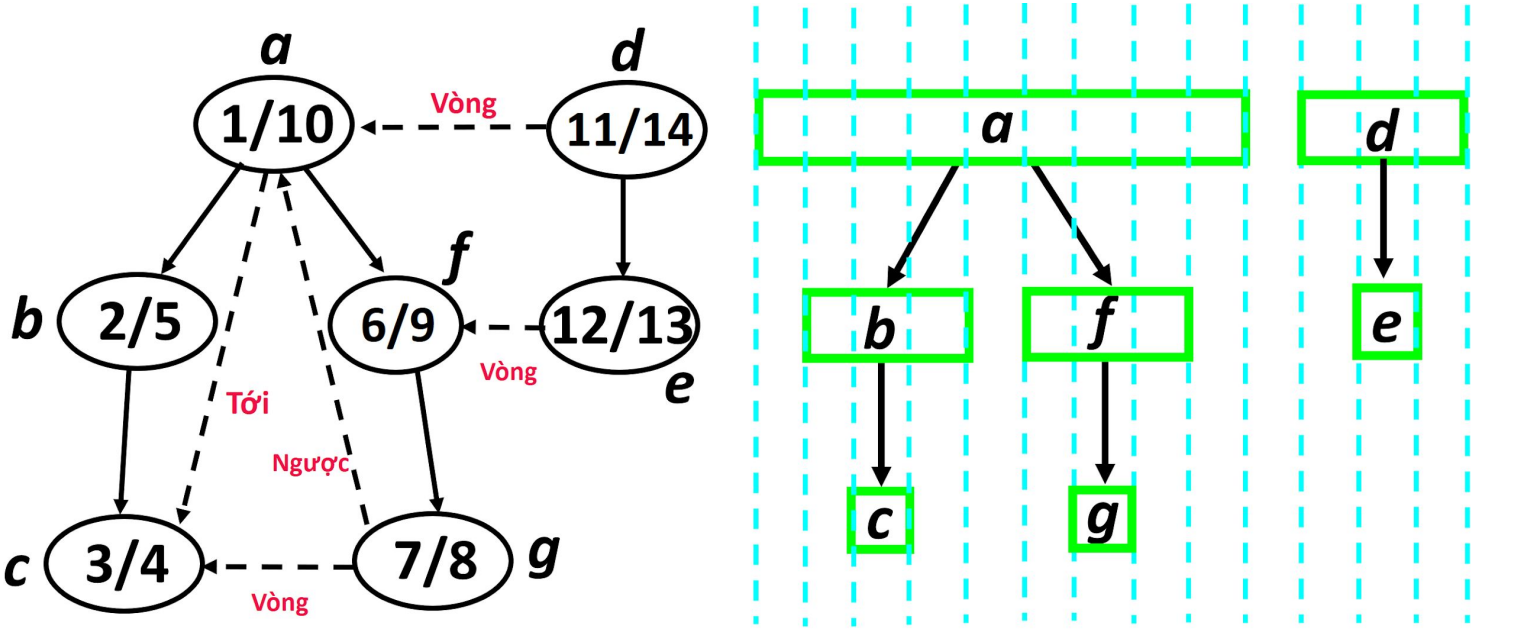
```
DFS(s)
1.    visited[s]= true; //Thăm đỉnh s
2.    time = time +1
3.    d[s] = time
4.    for each v ∈ Adj[s]
5.          if (visited[v] == false) {
6.                truoc[v] ← s;
7.                DFS(v);
8.          }
9.    time = time +1
10.   f[s] = time
```



Bổ đề về các khoảng lồng nhau

Cho đồ thị có hướng $G = (V, E)$, và cây DFS bất kỳ của G và hai đỉnh u, v tuỳ ý của nó. Khi đó

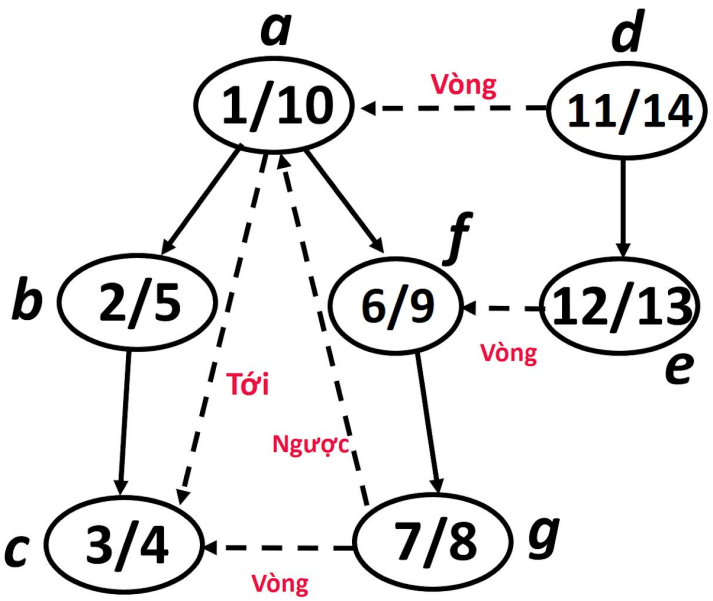
- u là con cháu của v khi và chỉ khi $[d[u], f[u]] \subseteq [d[v], f[v]]$
- u là tổ tiên của v khi và chỉ khi $[d[u], f[u]] \supseteq [d[v], f[v]]$
- u và v không có quan hệ họ hàng khi và chỉ khi $[d[u], f[u]]$ và $[d[v], f[v]]$ là không giao nhau.

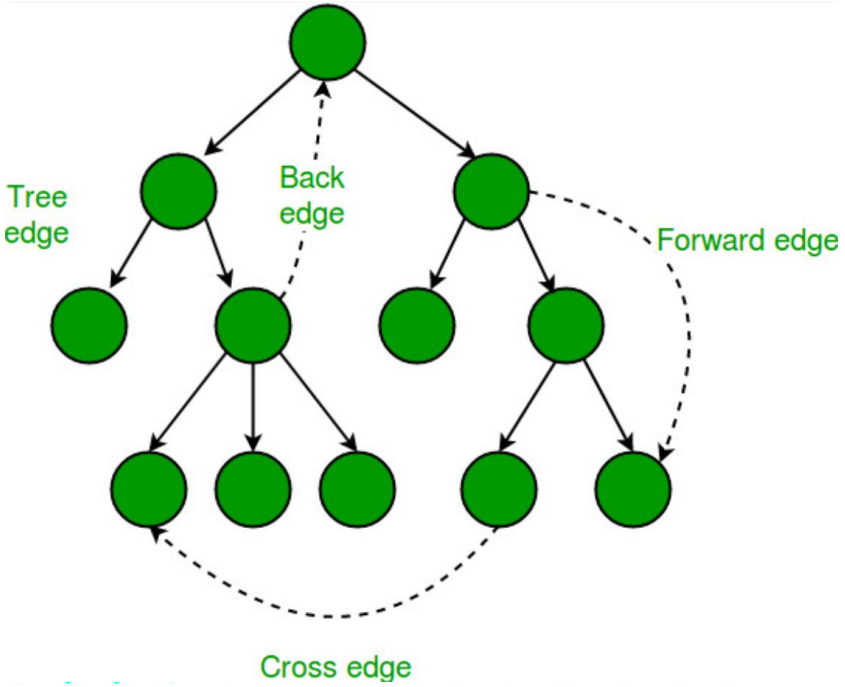


Bổ đề về các khoảng lồng nhau

Cho đồ thị có hướng $G = (V, E)$, và cây DFS bất kỳ của G và hai đỉnh u, v tuỳ ý của nó. Khi đó

- u là con cháu của v khi và chỉ khi $[d[u], f[u]] \subseteq [d[v], f[v]]$
- u là tổ tiên của v khi và chỉ khi $[d[u], f[u]] \supseteq [d[v], f[v]]$
- u và v không có quan hệ họ hàng khi và chỉ khi $[d[u], f[u]]$ và $[d[v], f[v]]$ là không giao nhau.





Các ứng dụng của DFS

1. Tính liên thông của đồ thị

2. Tìm đường đi từ s đến t

3. Phát hiện chu trình

4. Kiểm tra tính liên thông mạnh

5. Định hướng đồ thị

Bài toán về tính liên thông

- **Bài toán:** Cho đồ thị vô hướng $G = (V,E)$. Hỏi đồ thị gồm bao nhiêu thành phần liên thông, và từng thành phần liên thông gồm các đỉnh nào?
- Giải: Sử dụng DFS (BFS) :
  - Mỗi lần gọi đến DFS (BFS) ở trong chương trình chính sẽ sinh ra một thành phần liên thông

DFS giải bài toán liên thông

```
(* Main Program*)
1.  for s ∈ V
2.      visited[s] ← false
3.  for s ∈ V
4.      if (visited[s] == false)
5.          DFS(s)
```

```
DFS(s)
1.      visited[s] ← true  // Thăm đỉnh s
2.      for each v ∈ Adj[s]
3.          if (visited[v] == false)
4.              DFS(v)
```

```
(* Main Program*)
1.  for u ∈ V
2.      id[u] ← 0;
3.  cnt ← 0;  //cnt – số lượng tplt
4.  for u ∈ V
5.      if (id[u] == 0){
6.          cnt ← cnt +1;
7.          DFS(u) ;
8.      }
```

```
DFS(u)
1.      id[u] ← cnt;
2.      for each v ∈ Adj[u]
3.          if (id[v] == 0)
4.              DFS(v);
```

BFS giải bài toán liên thông

```
void BFS(s) {
// Tìm kiếm theo chiều rộng bắt đầu từ đỉnh s
        visited[s] ← 1; //đã thăm
        Q ← ∅; enqueue(Q,s); // Nạp s vào Q
        while (Q ≠ ∅)
        {
            u ← dequeue(Q);  // Lấy u khỏi Q
            for  v ∈Adj[u]
                if (visited[v] == 0) //chưa thăm
                {
                    visited[v] ← 1; //đã thăm
                    enqueue(Q,v)   // Nạp v vào Q
                }
        }
}
void main ()
{
        for  s  ∈ V  // Khởi tạo
             visited[s] ← 0;

        for  s  ∈ V
          if (visited[s]==0)  BFS(s);
}
```

```
BFS(s)
1.      id[s] ← cnt
2.      Q ← ∅; enqueue(Q,s); // Nạp s vào Q
3.      while (Q ≠ ∅)
4.      {
5.          u ← dequeue(Q);  // Lấy u khỏi Q
6.          for  v ∈Adj[u]
7.              if (id[v] == 0) //v chưa thăm
8.              {
9.                  id[v] ← cnt;
10.                 enqueue(Q,v)   // Nạp v vào Q
11.             }
12.     }
```

```
(* Main Program*)
1.  for s ∈ V
2.      id[s] ← 0
3.  cnt ← 0  //cnt – số lượng tplt
4.  for s ∈ V
5.      if (id[s] == 0){
6.              cnt  ← cnt +1
7.              BFS(s)
8.      }
```

Các ứng dụng của DFS

1. Tính liên thông của đồ thị
2. **Tìm đường đi từ s đến t**
3. Phát hiện chu trình
4. Kiểm tra tính liên thông mạnh
5. Định hướng đồ thị

Tìm đường đi

Bài toán tìm đường đi

- **Input:** Đồ thị $G = (V,E)$ xác định bởi danh sách kề và hai đỉnh s, t.
- **Output:** Đường đi từ đỉnh s đến đỉnh t, hoặc khẳng định không tồn tại đường đi từ s đến t.

Thuật toán: Thực hiện DFS(s) (hoặc BFS(s)).

- Nếu truoc[t] == NULL thì không có đường đi, trái lại ta có đường đi

$$t \leftarrow truoc[t] \leftarrow truoc[truoc[t]] \leftarrow \ldots \leftarrow s$$

DFS giải bài toán đường đi

```
(* Main Program*)
1. for u ∈ V {
2.       visited[u] ← false
3.       truoc[u] ← NULL }
4. DFS(s)
```

```
DFS(s)
1.       visited[s] ← true //Thăm đỉnh s
2.       for each v ∈ Adj[s]
3.             if (visited[v] == false) {
4.                   truoc[v] ← s
5.                   DFS(v)
6.             }
```

BFS giải bài toán đường đi

```
(* Main Program*)
1. for u ∈ V {
2.       visited[u] ← false
3.       truoc[u] ← NULL }
4. BFS(s)
```

```
BFS(s)
1.      visited[s] ← true;  //Thăm đỉnh s
2.      truoc[s] ← null;
3.      Q ← ∅; enqueue(Q,s); // Nạp s vào Q
4.      while (Q ≠ ∅)
5.      {
6.            u ← dequeue(Q);  // Lấy u khỏi Q
7.            for  v ∈Adj[u]
8.               if (visited[v] == false)   //chưa thăm
9.               {
10.                    visited[v] ← true;  //đã thăm
11.                    truoc[v] ← u;
12.                    enqueue(Q,v)   //Nạp v vào Q
13.              }
14.     }
```

Các ứng dụng của DFS

1. Tính liên thông của đồ thị
2. Tìm đường đi từ s đến t
3. **Phát hiện chu trình**
4. Kiểm tra tính liên thông mạnh
5. Định hướng đồ thị

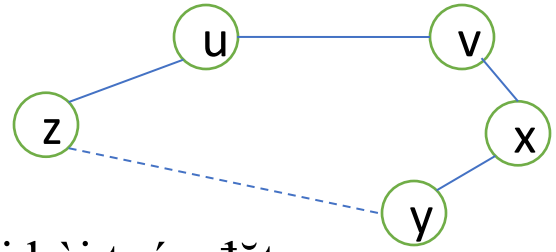
3. Bài toán chu trình: sử dụng DFS

Bài toán: Cho đồ thị $G=(V,E)$. Hỏi G có chứa chu trình hay không?

- ***Định lý:*** *Đồ thị G là không chứa chu trình khi và chỉ khi trong quá trình thực hiện DFS ta không phát hiện ra cạnh ngược.*

Chứng minh:

- $\Rightarrow$) Nếu G không chứa chu trình thì không thể có cạnh ngược. Hiển nhiên: bởi vì sự tồn tại cạnh ngược kéo theo sự tồn tại chu trình.
- $\Leftarrow$) Ta phải chứng minh: Nếu không có cạnh ngược thì *G không chứa* chu trình. Ta chứng minh bằng lập luận phản đề: G có chu trình $\Rightarrow \exists$ cạnh ngược. Gọi v là đỉnh trên chu trình được thăm đầu tiên trong quá trình thực hiện DFS, và u là đỉnh đi trước v trên chu trình. Khi v được thăm, các đỉnh khác trên chu trình đều chưa được thăm. Ta phải thăm được tất cả các đỉnh đạt được từ v trước khi quay trở lại từ DFS(v). Vì thế cạnh $u \rightarrow v$ được duyệt từ đỉnh u về tổ tiên v của nó, vì thế (u, v) là cạnh ngược.



Như vậy DFS có thể áp dụng để giải bài toán đặt ra.

Các ứng dụng của DFS

1. Tính liên thông của đồ thị
2. Tìm đường đi từ s đến t
3. Phát hiện chu trình
4. **Kiểm tra tính liên thông mạnh**
5. Định hướng đồ thị

Kiểm tra tính liên thông mạnh

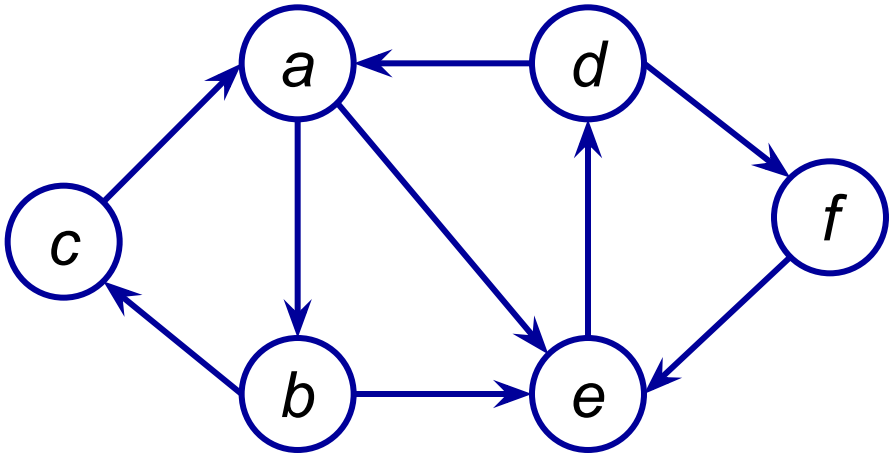
Bài toán: Cho đồ thị có hướng $G=(V,E)$. Hãy kiểm tra xem đồ thị G có phải liên thông mạnh hay không?

Mệnh đề: *Đồ thị có hướng $G=(V,E)$ là liên thông mạnh khi và chỉ khi luôn tìm được đường đi từ một đỉnh v đến tất cả các đỉnh còn lại và luôn tìm được đường đi từ tất cả các đỉnh thuộc $V \setminus \{v\}$ đến v.*

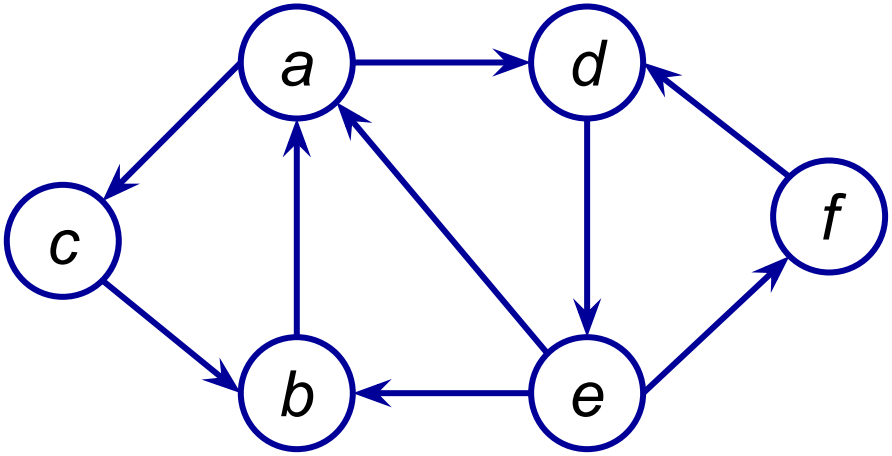
Thuật toán kiểm tra tính liên thông mạnh

- Chọn $v \in V$ là một đỉnh tuỳ ý.
- Thực hiện DFS(v) trên G. Nếu tồn tại đỉnh u không được thăm thì G không liên thông mạnh và thuật toán kết thúc. Trái lại thực hiện tiếp
- Thực hiện DFS(v) trên $G^T = (V, E^T)$, với E^T thu được từ E bởi việc đảo ngược hướng các cung. Nếu tồn tại đỉnh u không được thăm thì G không liên thông mạnh, nếu trái lại G là liên thông mạnh.
- Thời gian tính: $O(|V|+|E|)$

Thuật toán kiểm tra tính liên thông mạnh



Đồ thị G



Đồ thị G^T

Các ứng dụng của DFS

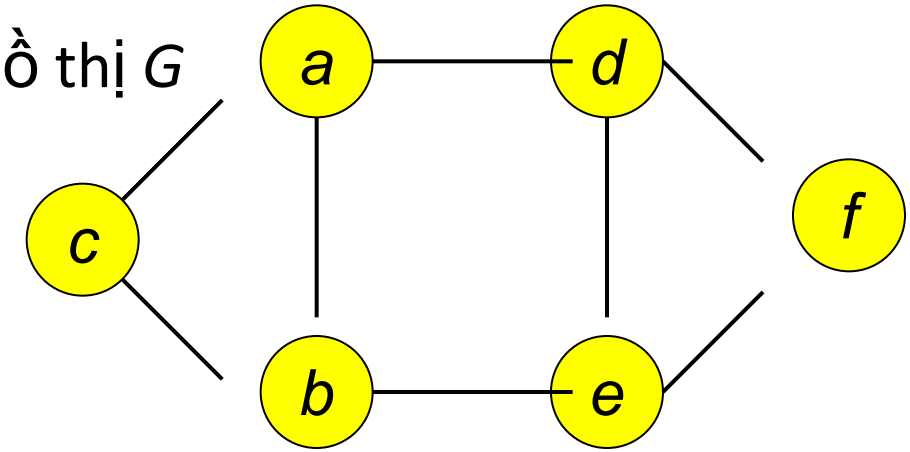
1. Tính liên thông của đồ thị
2. Tìm đường đi từ s đến t
3. Phát hiện chu trình
4. Kiểm tra tính liên thông mạnh
5. **Định hướng đồ thị**

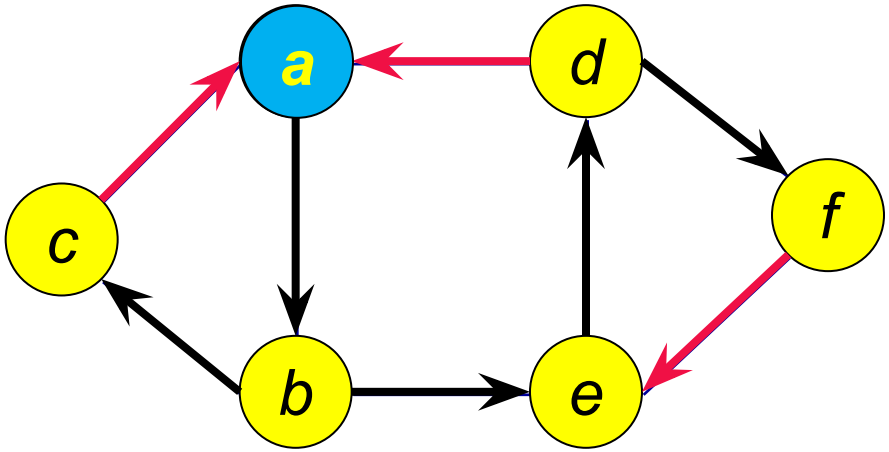
Định hướng đồ thị

- **Bài toán:** Cho đồ thị vô hướng liên thông $G = (V, E)$. Hãy tìm cách định hướng các cạnh của nó để thu được đồ thị có hướng liên thông mạnh hoặc trả lời G là không định hướng được.
- **Thuật toán định hướng δ:** Trong quá trình thực hiện DFS(G) định hướng: (1) các cạnh của cây DFS theo chiều từ tổ tiên đến con cháu, (2) các cạnh ngược theo hướng từ con cháu đến tổ tiên. Ký hiệu đồ thị thu được là $G(\delta)$
- **Bổ đề. G là định hướng được khi và chỉ khi $G(\delta)$ là liên thông mạnh.**

Ví dụ: Định hướng đồ thị

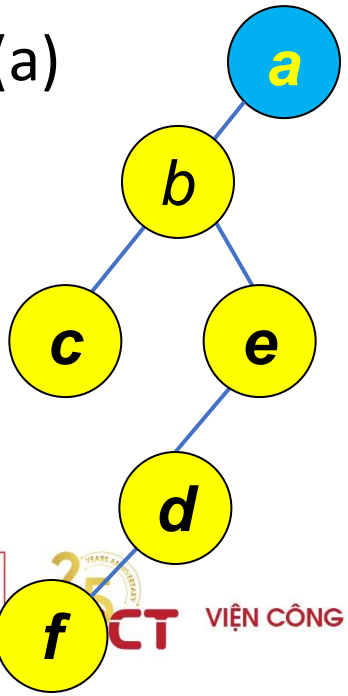
Đồ thị *G*





Đồ thị *G*(δ)

DFS(a)



Thuật toán định hướng δ: Trong quá trình thực hiện DFS(G) định hướng: (1) các cạnh của cây DFS theo chiều từ tổ tiên đến con cháu, (2) các cạnh ngược theo hướng từ con cháu đến tổ tiên. Ký hiệu đồ thị thu được là *G*(δ)