

# LÝ THUYẾT ĐỒ THỊ GRAPH THEORY

LÊ THỊ PHƯƠNG DUNG

# NỘI DUNG

- 1. ĐẠI CƯƠNG VỀ ĐỒ THỊ**
- 2. TÍNH LIÊN THÔNG CỦA ĐỒ THỊ**
- 3. ĐƯỜNG ĐI NGẮN NHẤT TRÊN ĐỒ THỊ**
- 4. XẾP HẠNG ĐỒ THỊ**
- 5. CÂY VÀ CÂY CÓ HƯỚNG**
- 6. LUỒNG CỤC ĐẠI TRONG MẠNG**

# TÀI LIỆU THAM KHẢO

1. TOÁN RỜI RẠC – *NGUYỄN TÔ THÀNH, NGUYỄN  
ĐỨC NGHĨA*
  2. LÝ THUYẾT ĐỒ THỊ VÀ ỨNG DỤNG – *NGUYỄN  
TUẤN ANH*
- .....

## CHƯƠNG 2

# TÍNH LIÊN THÔNG CỦA ĐỒ THỊ

### NỘI DUNG:

1. ĐƯỜNG ĐI VÀ CHU TRÌNH
2. ĐỒ THỊ LIÊN THÔNG
3. DUYỆT ĐỒ THỊ & ỨNG DỤNG

# ĐƯỜNG ĐI VÀ CHU TRÌNH

**Đường đi** độ dài n (cạnh) xuất phát từ đỉnh  $x_0$  đến đỉnh  $x_n$  trên đồ thị  $G=(X,E)$  là dãy các đỉnh  $x_0, x_1, \dots, x_n$ , trong đó  $(x_i, x_{i+1}) \in E$  với  $i=0, 1, \dots, n-1$

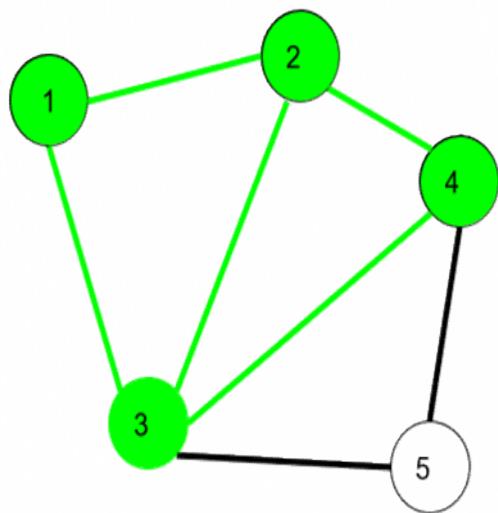
Đường đi trên còn có thể biểu diễn dưới dạng dãy các cạnh  $e_1, e_2, \dots, e_n$  trong đó  $e_i = (x_{i-1}, x_i)$ , với  $i=1, 2, \dots, n$

**Chu trình** là đường đi có đỉnh xuất phát trùng với đỉnh đến

Đường đi/chu trình không có cạnh bị lặp lại được gọi là đường đi/chu trình **đơn**

Đường đi/chu trình không có đỉnh bị lặp lại được gọi là đường đi/chu trình **sơ cấp**

# ĐƯỜNG ĐI VÀ CHU TRÌNH



Đường đi:

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4$

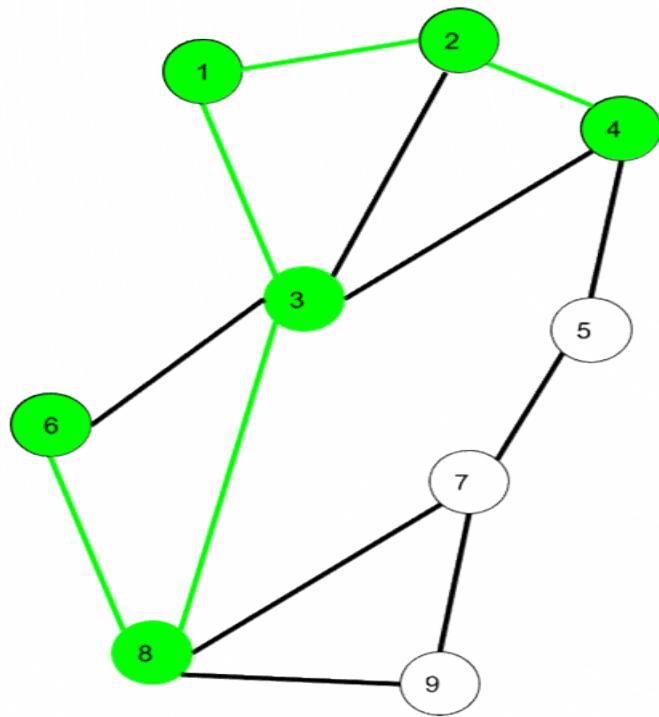
$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow 3$

Chu trình:

$2 \rightarrow 3 \rightarrow 4 \rightarrow 2$

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 1$

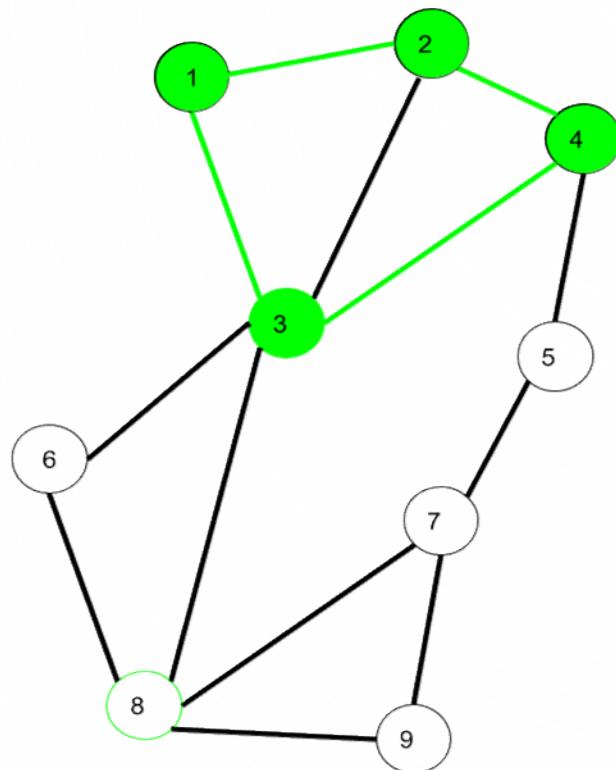
# ĐƯỜNG ĐI VÀ CHU TRÌNH



Đường đi đơn:

6->8->3->1->2->4

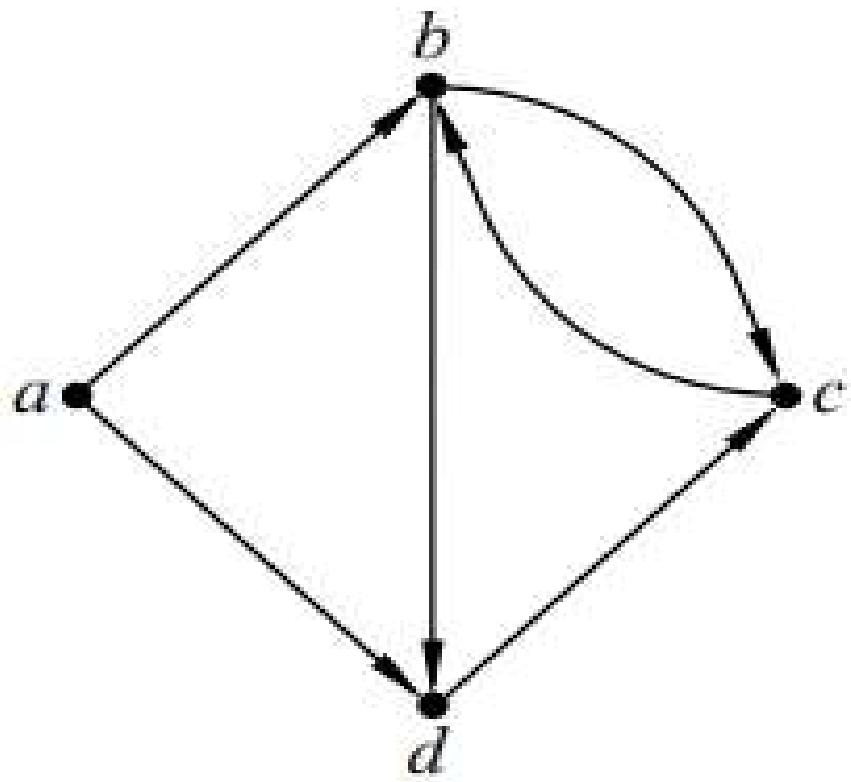
# ĐƯỜNG ĐI VÀ CHU TRÌNH



Chu trình đơn:

1->2->4->3->1

# ĐƯỜNG ĐI VÀ CHU TRÌNH



Đường đi:  $a \rightarrow b \rightarrow c \rightarrow b \rightarrow d$

Đường đi đơn:  $a \rightarrow b \rightarrow d$

Chu trình:  $d \rightarrow c \rightarrow b \rightarrow c \rightarrow b \rightarrow d$

Chu trình đơn:  $b \rightarrow d \rightarrow c \rightarrow b$

# ĐỒ THỊ LIÊN THÔNG

Cho đồ thị  $G=(X,E)$  ta định nghĩa quan hệ liên kết  $\sim$  như sau:

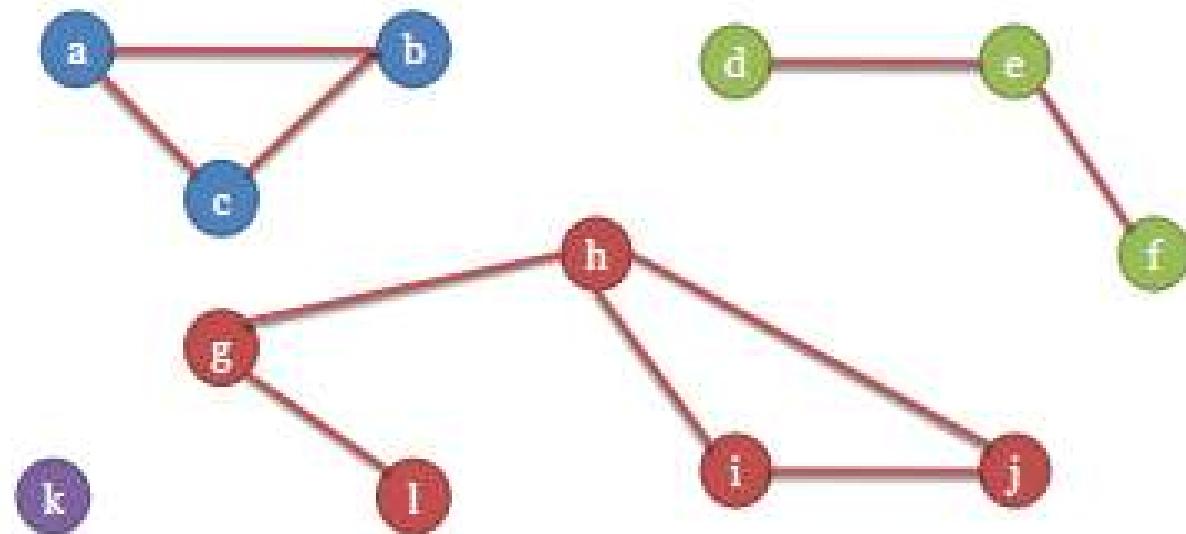
$$\forall i, j \in X, i \sim j \Leftrightarrow (i = j \text{ hoặc có đường đi từ } i \text{ đến } j)$$

Quan hệ này là quan hệ tương đương (tự chứng minh như bài tập nhỏ)  
Do đó, tập đỉnh  $X$  được phân hoạch thành các lớp tương đương  
(không giao nhau)

**Trong đồ thị vô hướng  $G$ :**

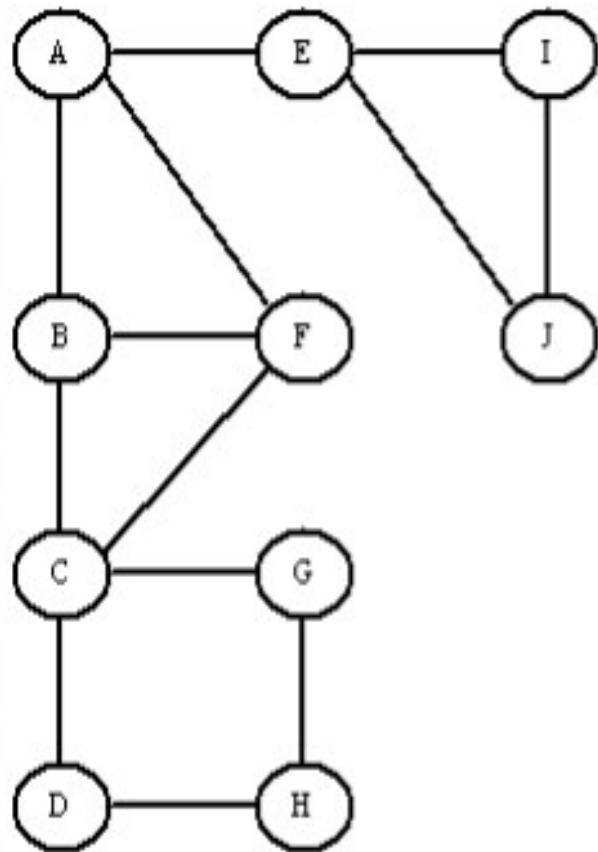
- Mỗi lớp tương đương được gọi là một **thành phần liên thông**
- Số lớp tương đương là **số thành phần liên thông**
- **Đồ thị liên thông** là đồ thị chỉ có một thành phần liên thông
- Nếu  $G$  có  $p$  thành phần liên thông  $G_1, G_2, \dots, G_p$  thì mỗi  $G_i$  là một đồ thị con của  $G$ , bậc của  $x$  trong  $G_i$  bằng với bậc của  $x$  trong  $G$

# ĐỒ THỊ LIÊN THÔNG



Đồ thị vô hướng  $G$  có 4 thành phần liên thông  
→ Đồ thị  $G$  không liên thông

# ĐỒ THỊ LIÊN THÔNG



Đồ thị vô hướng H có 1 thành  
phần liên thông  
→ Đồ thị H liên thông

# ĐỒ THỊ LIÊN THÔNG

Cho đồ thị  $G=(X,E)$  ta định nghĩa quan hệ liên kết  $\sim$  như sau:

$$\forall i, j \in X, i \sim j \Leftrightarrow ((i \equiv j) \text{ hoặc } (\text{đường đi từ } i \text{ đến } j \text{ và đường đi từ } j \text{ đến } i))$$

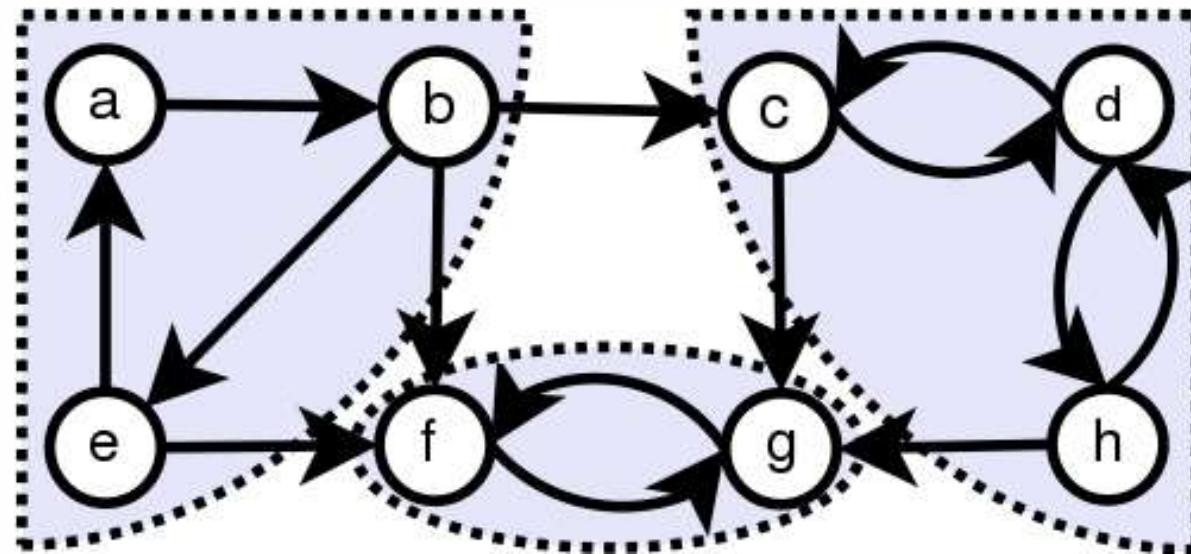
Quan hệ này là quan hệ tương đương (tự chứng minh như bài tập nhỏ)

Do đó, tập đỉnh  $X$  được phân hoạch thành các lớp tương đương (không giao nhau)

**Trong đồ thị có hướng G:**

- Mỗi lớp tương đương được gọi là một **thành phần liên thông mạnh**
- Số lớp tương đương là **số thành phần liên thông mạnh**
- **Đồ thị liên thông mạnh** là đồ thị **chỉ có một thành phần liên thông mạnh**
- Đồ thị có hướng  $G$  được gọi là **liên thông (yếu)** nếu đồ thị vô hướng tương ứng của  $G$  liên thông

# ĐỒ THỊ LIÊN THÔNG

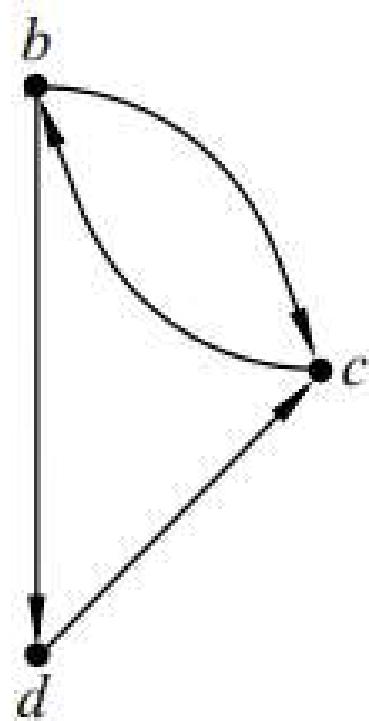


Đồ thị có hướng G liên thông yếu

Đồ thị có hướng G có 3 thành phần liên thông mạnh

→ Đồ thị có hướng G không liên thông mạnh

# ĐỒ THỊ LIÊN THÔNG



Đồ thị có hướng H có 1 thành phần liên thông mạnh  
→ H là đồ thị liên thông mạnh

# DUYỆT ĐỒ THỊ

- Khi giải quyết nhiều bài toán lý thuyết đồ thị, ta luôn phải xét qua tất cả các đỉnh của đồ thị đó. Do đó, cần có thuật toán duyệt toàn bộ các đỉnh của đồ thị này. Gọi chung là thuật toán duyệt đồ thị.
- Ứng dụng: Tìm các thành phần liên thông của đồ thị, tìm đường đi ngắn nhất, tìm luồng cực đại trong mạng...
- Các giải thuật duyệt đồ thị:
  - **Duyệt chiềу rộng**
  - **Duyệt chiềу sâu**

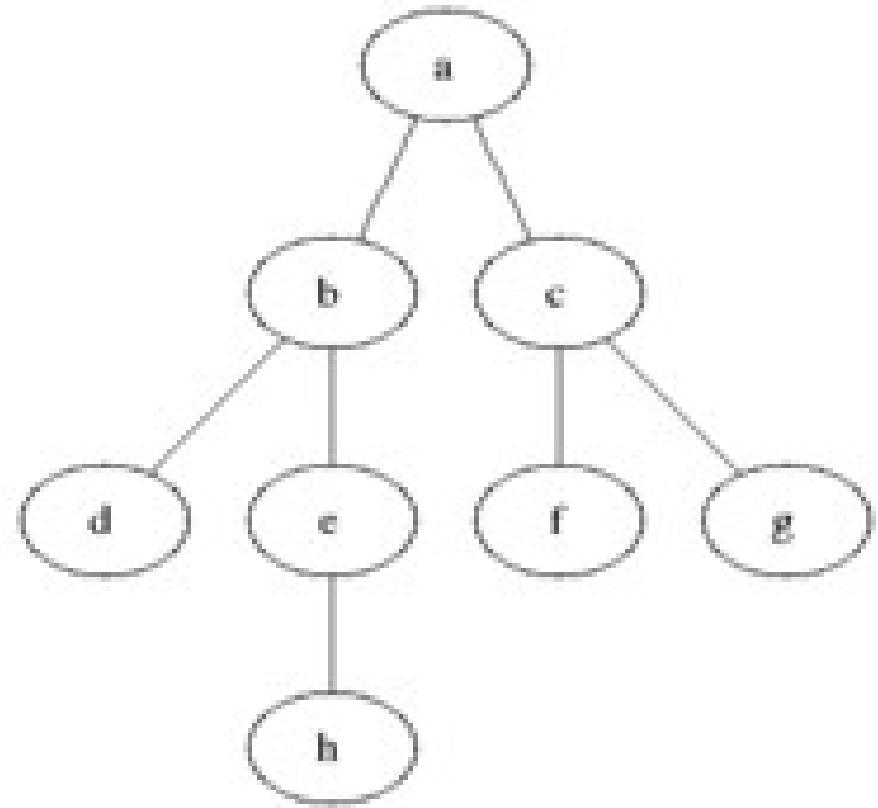
# DUYỆT ĐỘ THỊ

## Duyệt theo chiều rộng

(Breadth first search - **BFS**):

- B0: Từ đỉnh v nào đó chưa thăm, đưa v vào **hàng đợi (FIFO)**
- B1: Lấy từ hàng đợi một đỉnh v, thăm v, đưa các đỉnh u chưa thăm kèm với v vào hàng đợi
- B2: Lặp lại B1 cho tới khi hàng đợi rỗng.

Queue	
BFS	a b c d e f g h



Thứ tự duyệt bắt đầu từ a:

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

# DUYỆT ĐỒ THỊ

Duyệt theo chiều rộng (Breadth first search - BFS):

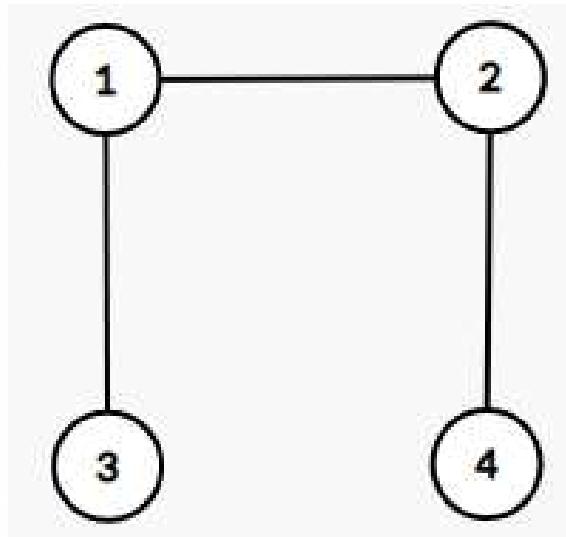
Thuật toán duyệt một thành phần liên thông:

Mỗi lần duyệt xong một đỉnh ta có một thành phần liên thông

```
1 procedure BFS1( $G, root$ ) is
2     let  $Q$  be a queue
3     label  $root$  as discovered
4      $Q.enqueue(root)$ 
5     while  $Q$  is not empty do
6          $v := Q.dequeue()$ 
7         for all edges from  $v$  to  $w$  in  $G.adjacentEdges(v)$  do
8             if  $w$  is not labeled as discovered then
9                 label  $w$  as discovered
10                parent( $w$ ) :=  $v$            //Optional
11                 $Q.enqueue(w)$ 
```

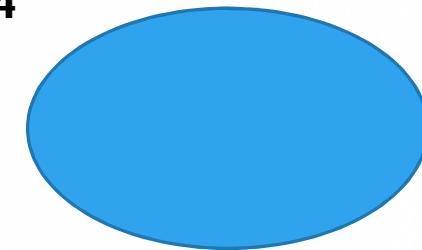
# DUYỆT ĐỒ THỊ

Duyệt theo chiều rộng (Breadth first search - **BFS**):



Thứ tự duyệt bắt đầu từ đỉnh 1:

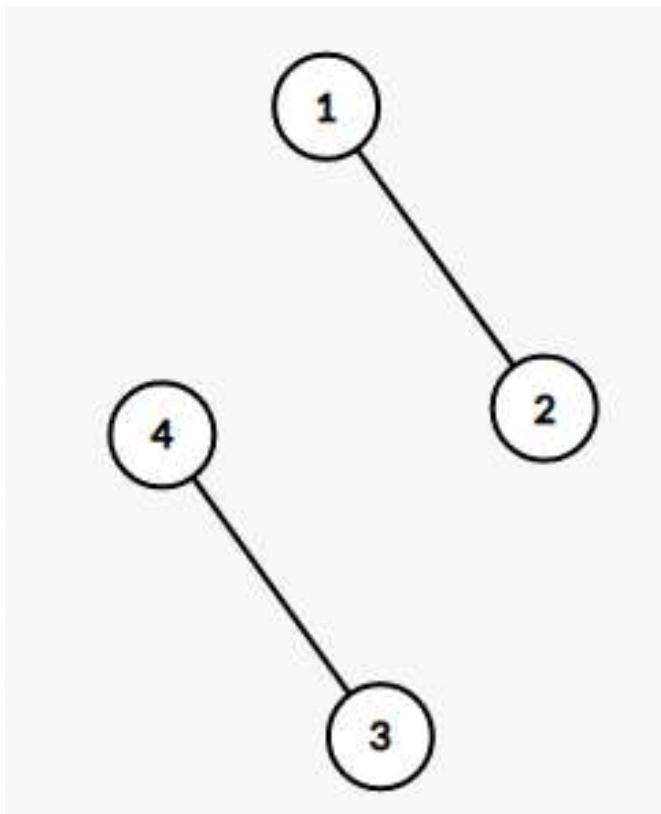
1 2 3 4



Queue	
BFS	1 2 3 4

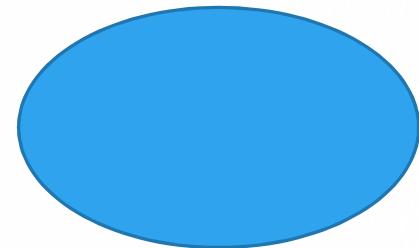
# DUYỆT ĐỒ THỊ

Duyệt theo chiều rộng (Breadth first search - BFS):



Thứ tự duyệt bắt đầu từ đỉnh 1:

1 2 3 4



Queue	
BFS	1 2 3 4

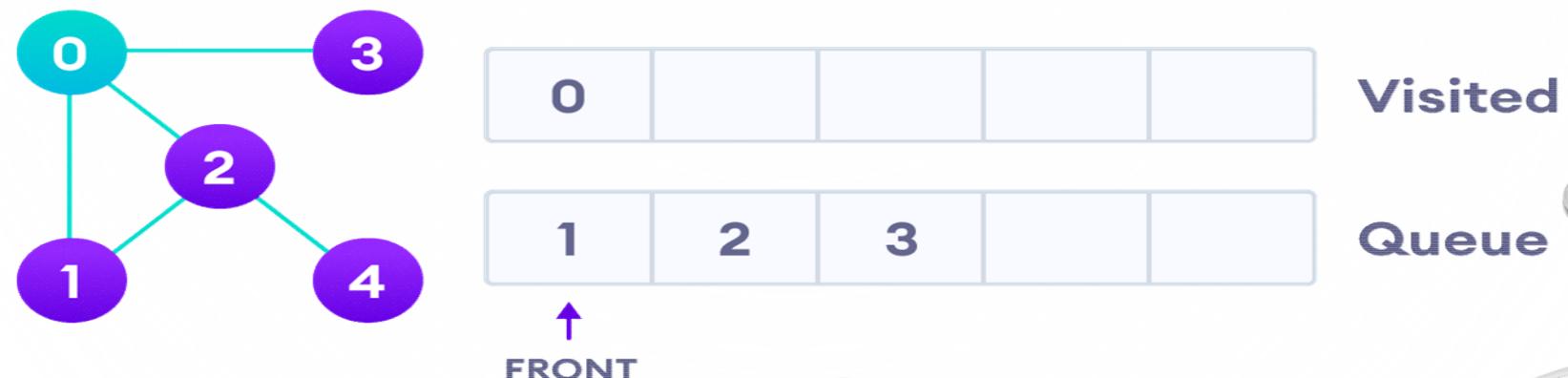
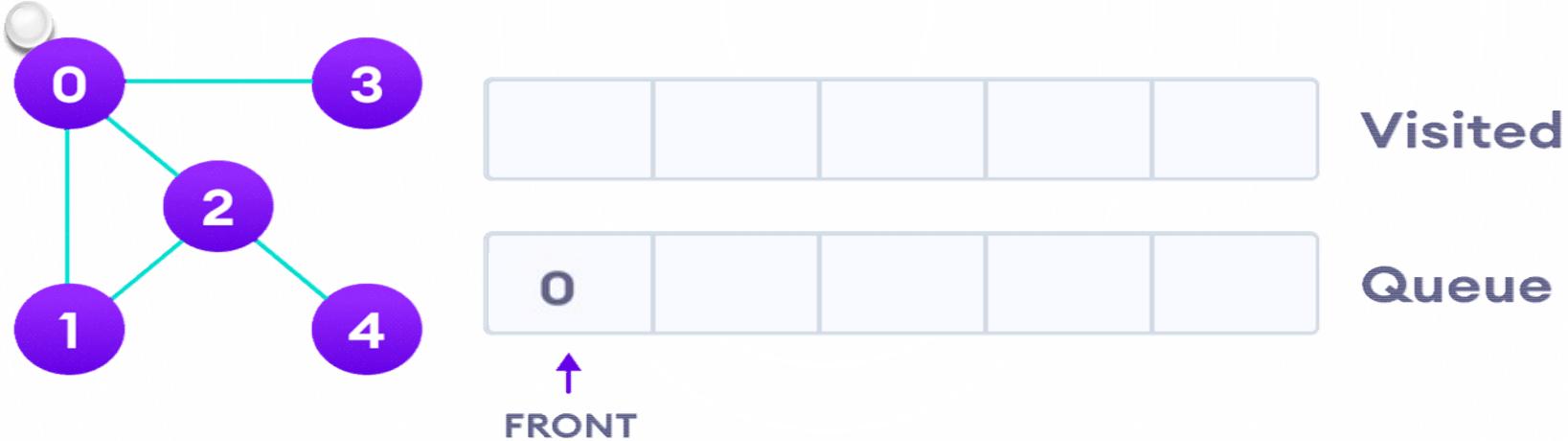
# DUYỆT ĐỘ THỊ

Duyệt theo chiều rộng (Breadth first search - **BFS**):  
Thuật toán duyệt tất cả thành phần liên thông:

```
1 procedure BFS( $G$ ) is
2     for each vertex  $v$  in  $V$  do
3         label  $v$  as not discovered
4     for each  $v$  in  $V$  do
5         if  $v$  is not labeled as discovered then
6             BFS1( $G, v$ )
```

# DUYỆT ĐỘ THỊ

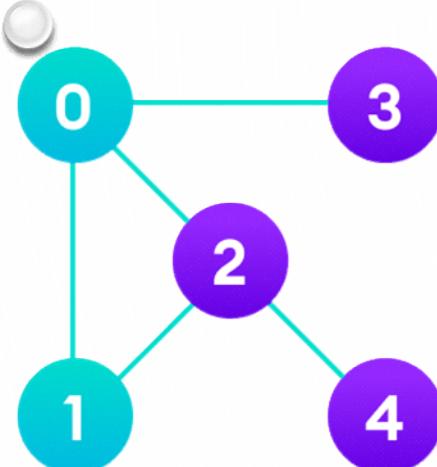
Duyệt theo chiều rộng (Breadth first search - **BFS**):



Visit start vertex and add its adjacent vertices to queue

# DUYỆT ĐỒ THỊ

Duyệt theo chiều rộng (Breadth first search - **BFS**):

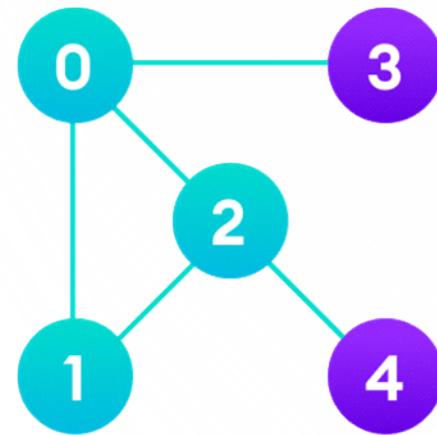


Visited



Queue

FRONT      Visit the first neighbour of start node 0, which is 1



Visited



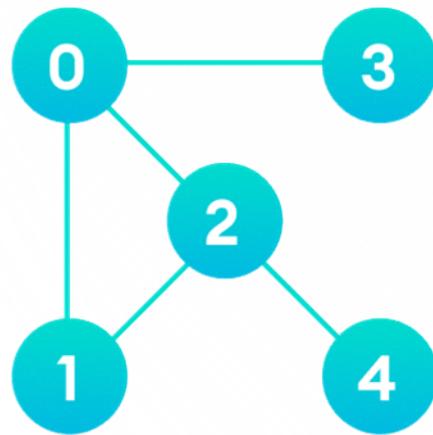
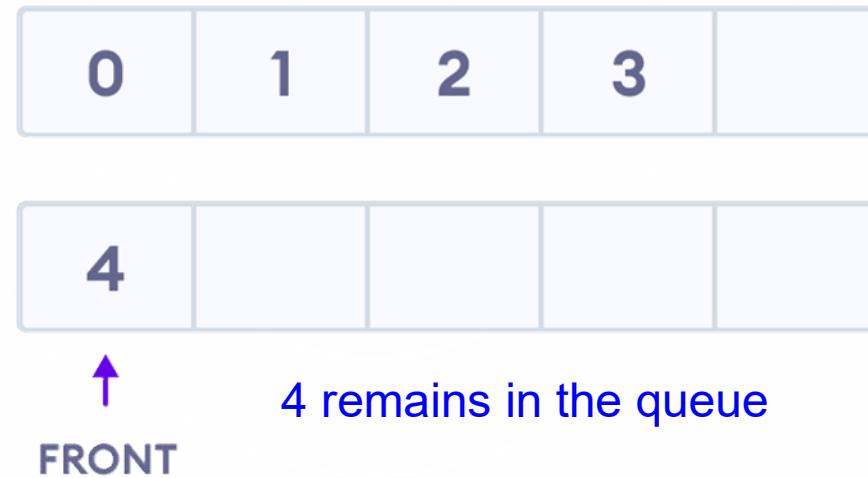
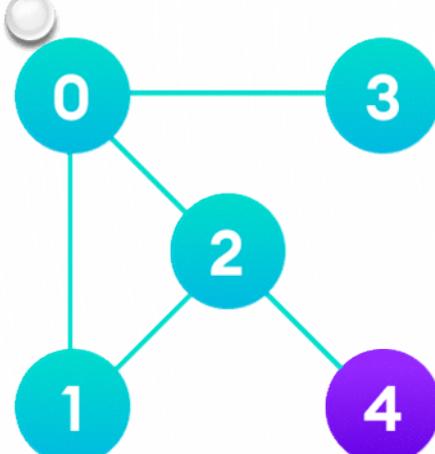
Queue

FRONT

Visit 2 which was added to queue earlier to add its neighbours

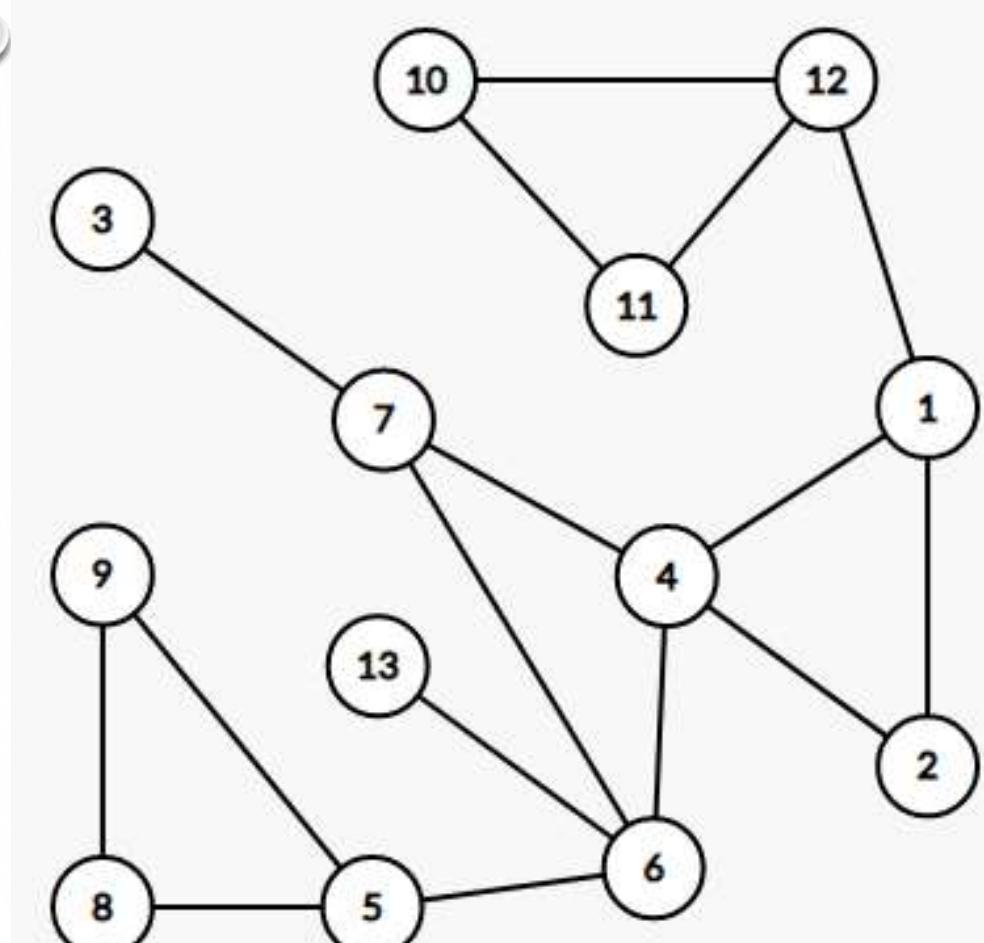
# DUYỆT ĐỘ THỊ

Duyệt theo chiều rộng (Breadth first search - **BFS**):

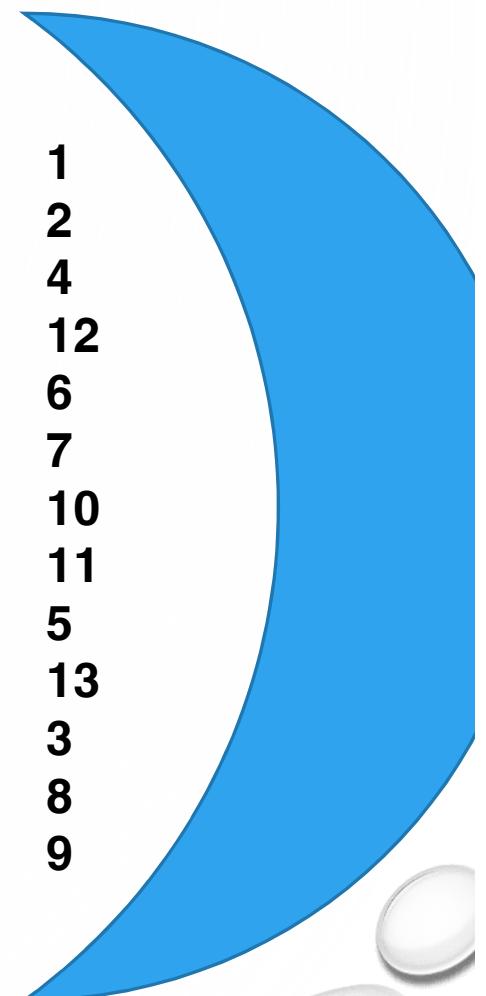


# DUYỆT ĐỒ THỊ

Duyệt theo chiều rộng (Breadth first search - **BFS**):

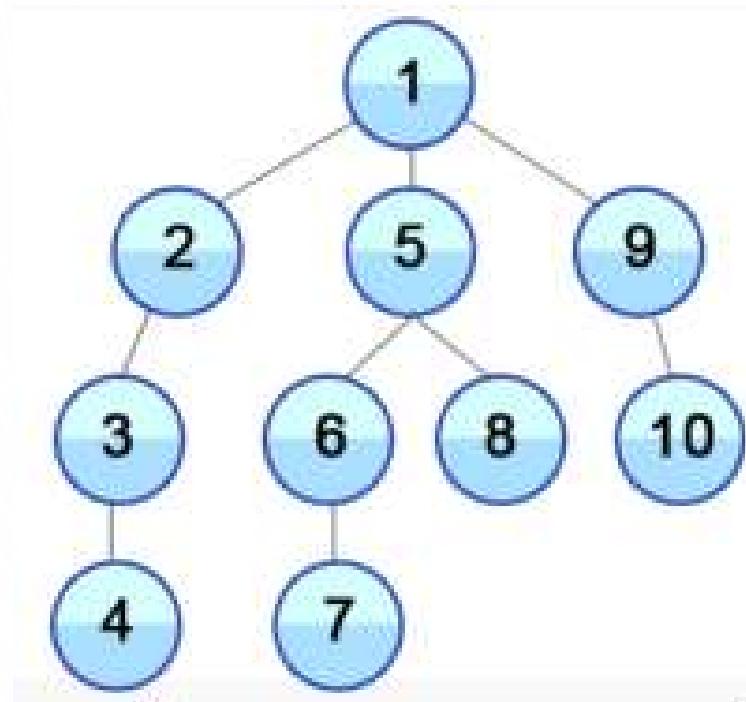


Queue	
BFS	1 2 4 12 6 7 10 11 5 13 3 8 9



# DUYỆT ĐỘ THỊ

- Duyệt theo chiều sâu (Depth first search - DFS):
  - Tương tự BFS, nhưng sử dụng cấu trúc **ngăn xếp** (LIFO)



Thứ tự duyệt bắt đầu từ 1:

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

Stack	
DFS	1 2 3 4 5 6 7 8 9 10

# DUYỆT ĐỒ THỊ

Duyệt theo chiều sâu (Depth first search - **DFS**):

Thuật toán duyệt một bộ phận liên thông của đồ thị (stack)

```
1 procedure DFS1(G, root) is
2     let S be a stack
3     S.push(root)
4     while S is not empty do
5         v := S.pop()
6         if v is not labeled as discovered then
7             label v as discovered
8             for all edges from v to w in G.adjacentEdges(v) do
9                 parent(w) := v      //Optional
10                S.push(w)
```

# DUYỆT ĐỒ THỊ

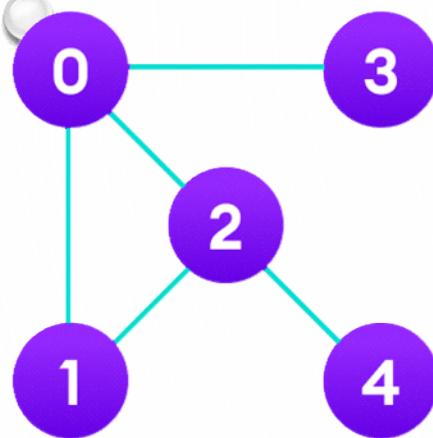
Duyệt theo chiều sâu (Depth first search - DFS):

Thuật toán duyệt một bộ phận liên thông của đồ thị (dệ quy)

```
1 procedure DFS1(G, root) is
2     if label v as discovered then
3         return v
4     label v as discovered
5     for all edges from v to w in G.adjacentEdges(v) do
6         if vertex w is not labeled as discovered then
7             recursively call DFS(G,w)
```

# DUYỆT ĐỘ THỊ

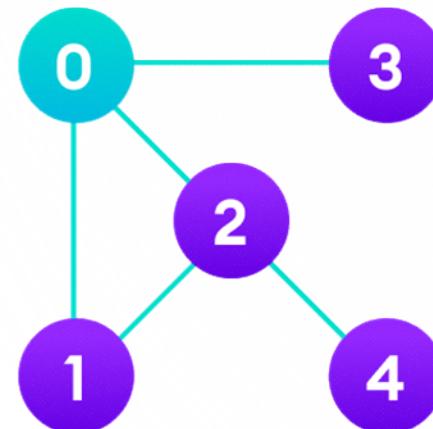
Duyệt theo chiều sâu (Depth first search - DFS):



Visited



Stack



Visited

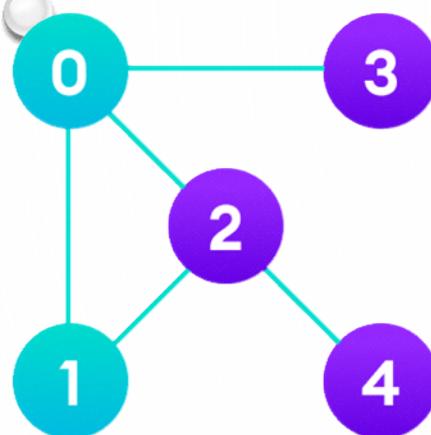


Stack

Visit start vertex and add its adjacent vertices to stack

# DUYỆT ĐỘ THỊ

Duyệt theo chiều sâu (Depth first search - DFS):

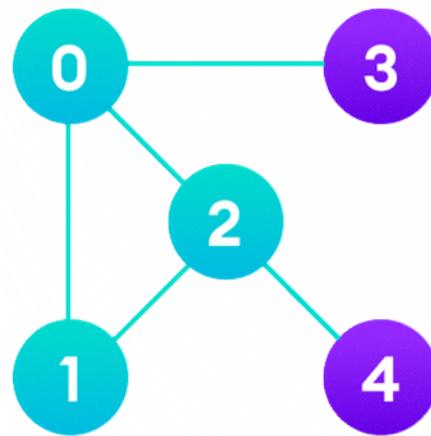


Visited



Stack

Visit the element at the top of stack



Visited

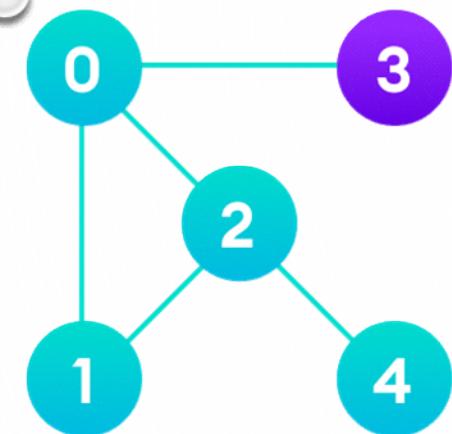


Stack

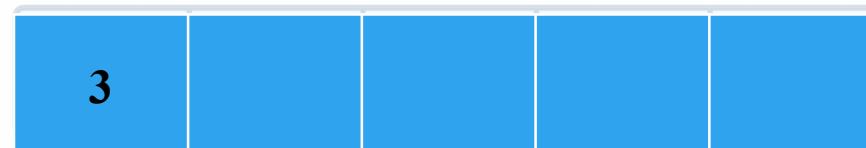
Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.

# DUYỆT ĐỒ THỊ

Duyệt theo chiều sâu (Depth first search - DFS):

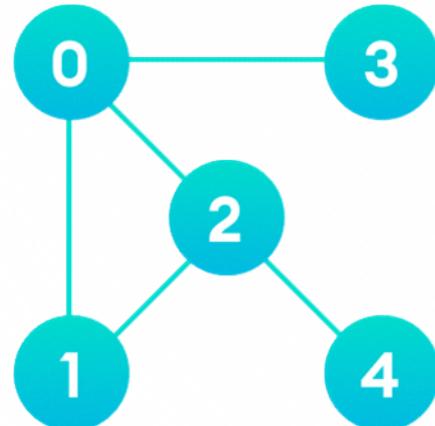


Visited

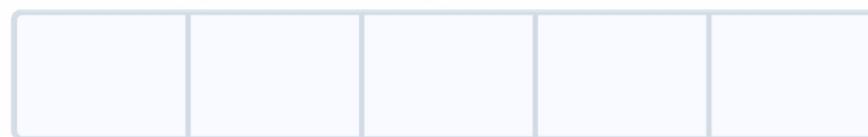


Stack

Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.



Visited

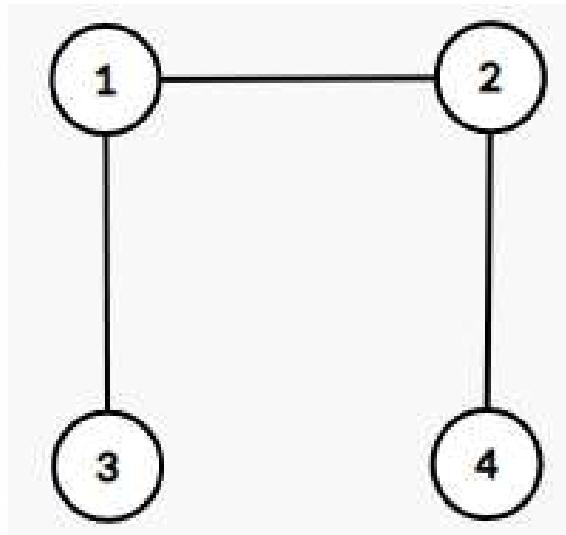


Stack

After we visit the last element 3, it doesn't have any unvisited adjacent nodes

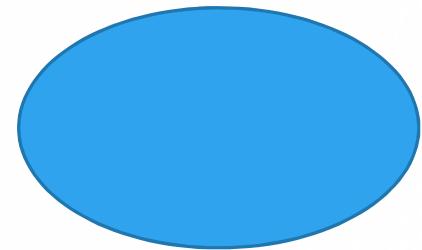
# DUYỆT ĐỒ THỊ

Duyệt theo chiều sâu (DFS):



Thứ tự duyệt bắt đầu từ đỉnh 1:

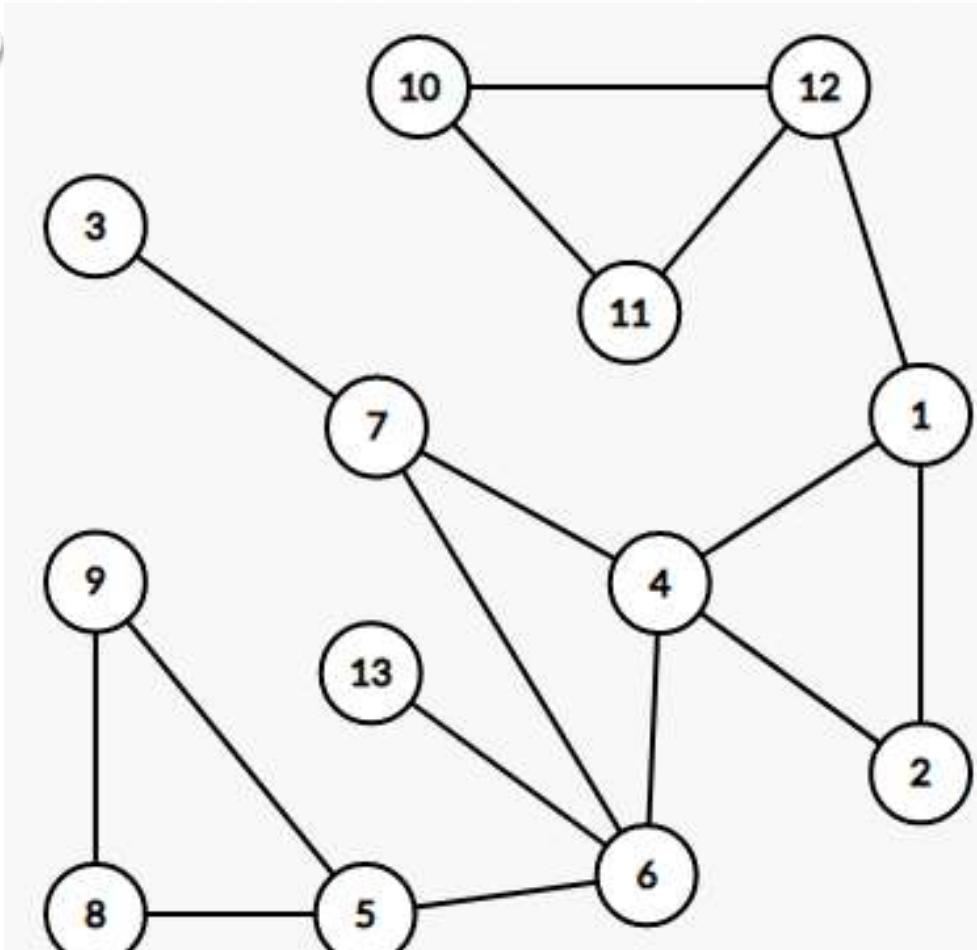
1 2 4 3



Stack	
DFS	1 2 4 3

# DUYỆT ĐỒ THỊ

## Duyệt theo chiều sâu (DFS):



Stack

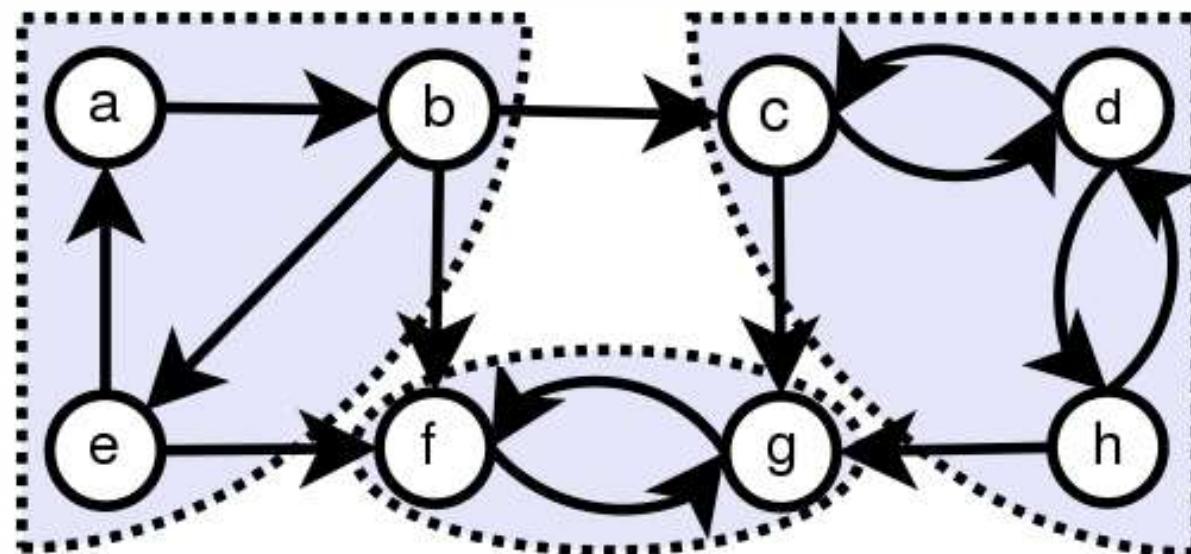
DFS

1  
12  
11  
10  
4  
7  
6  
13  
5  
9  
8  
3  
2

# TÌM THÀNH PHẦN LIÊN THÔNG MẠNH

Bộ phận **liên thông mạnh**: Luôn tồn tại đường đi **có hướng** giữa hai đỉnh bất kỳ

Đồ thị có hướng liên thông mạnh khi nó chỉ có một bộ phận liên thông mạnh



# TÌM THÀNH PHẦN LIÊN THÔNG MẠNH

## Thuật toán Tarjan

**Robert Endre Tarjan** (born April 30, 1948) is an American computer scientist and mathematician



Born	April 30, 1948 (age 72) <a href="#">Pomona, California</a>
Citizenship	American
Alma mater	<a href="#">California Institute of Technology (BS)</a> <a href="#">Stanford University (MS, PhD)</a>
Known for	Algorithms and data structures
Awards	<a href="#">Paris Kanellakis Award</a> (1999) <a href="#">Turing Award</a> (1986) <a href="#">Nevanlinna Prize</a> (1982)
Scientific career	
Fields	<a href="#">Computer science</a>

# TÌM THÀNH PHẦN LIÊN THÔNG MẠNH

## Thuật toán Tarjan

- Áp dụng thuật toán **duyệt theo chiều sâu** để đánh chỉ số các đỉnh  $v.index$
- Với mỗi đỉnh  $v$ , ta lưu thêm  $v.lowlink$  là chỉ số nhỏ nhất trong các đỉnh có thể đến được từ  $v$ . Trong quá trình duyệt,  $v.lowlink$  được cập nhật
- Khi duyệt xong đỉnh  $v$ , nếu  $v.lowlink=v.index$ , tức là không có đỉnh nào có chỉ số nhỏ hơn mà  $v$  có thể đến, thì  $v$  là đỉnh bắt đầu (đầu) của **bộ phận liên thông mạnh** (strongly connected component - SCC)

# TÌM THÀNH PHẦN LIÊN THÔNG MẠNH

## Thuật toán Tarjan

```
function strongconnect(v)
    // Set the depth index for v to the smallest unused index
    v.index := index  v.lowlink := index          index := index + 1
    S.push(v)           v.onStack := true
    for each (v, w) in E do // Consider successors of v
        if w.index is undefined then
            // Successor w has not yet been visited; recurse on it
            strongconnect(w)
            v.lowlink := min(v.lowlink, w.lowlink)
        else if w.onStack then
            v.lowlink := min(v.lowlink, w.index) //from the original paper
    // If v is a root node, pop the stack and generate an SCC
    if v.lowlink = v.index then
        start a new strongly connected component
        do
            w := S.pop()
            w.onStack := false
            add w to current strongly connected component
        while w ≠ v
        output the current strongly connected component end if end
```

# TÌM THÀNH PHẦN LIÊN THÔNG MẠNH

## Thuật toán Tarjan

**input:** graph  $G = (V, E)$

**output:** set of strongly connected components (sets of vertices)

*index* := 0

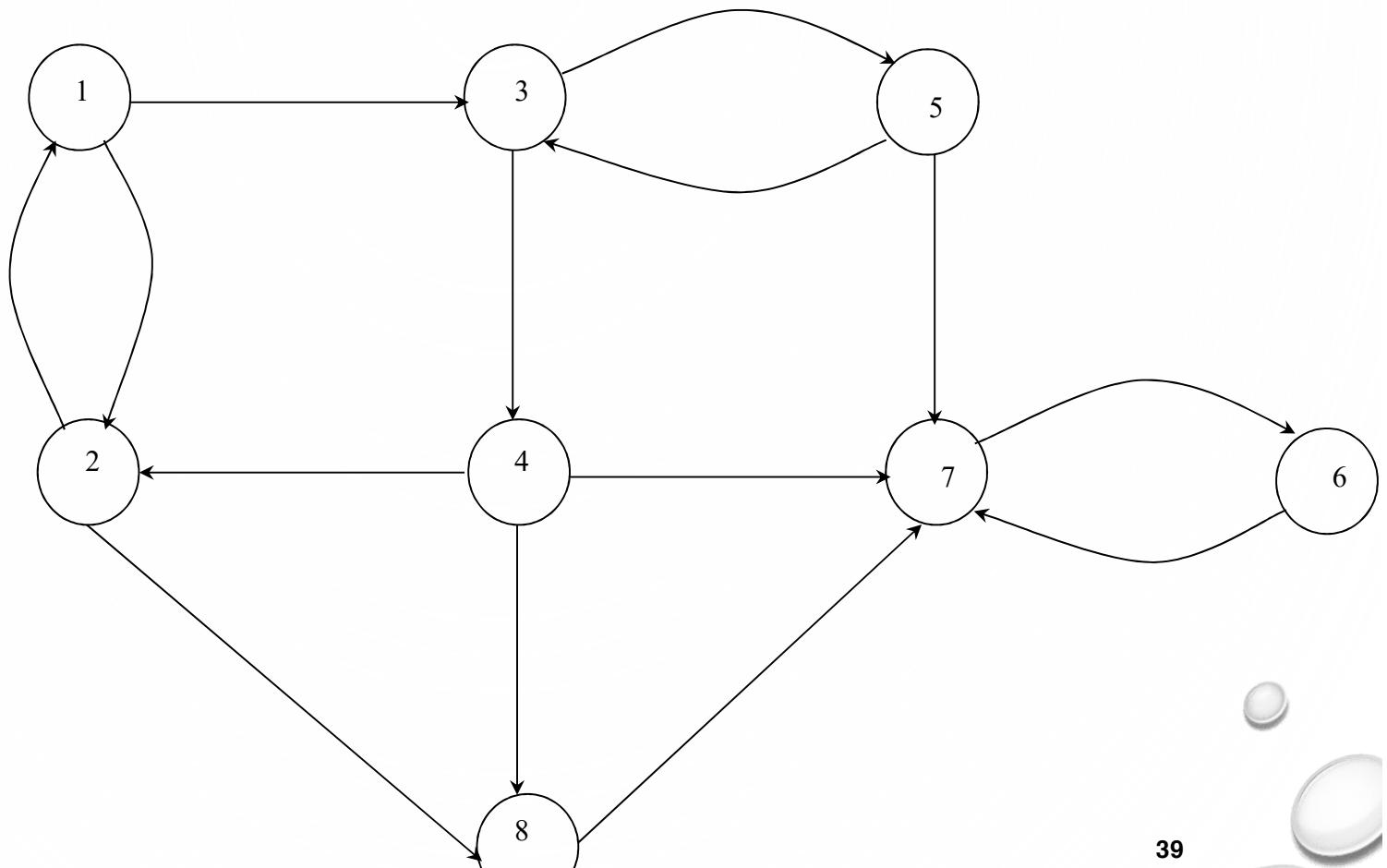
*S* := empty stack

**for each**  $v$  in  $V$  **do**

**if**  $v.\text{index}$  is undefined **then**

        strongconnect( $v$ )

# TÌM THÀNH PHẦN LIÊN THÔNG MẠNH

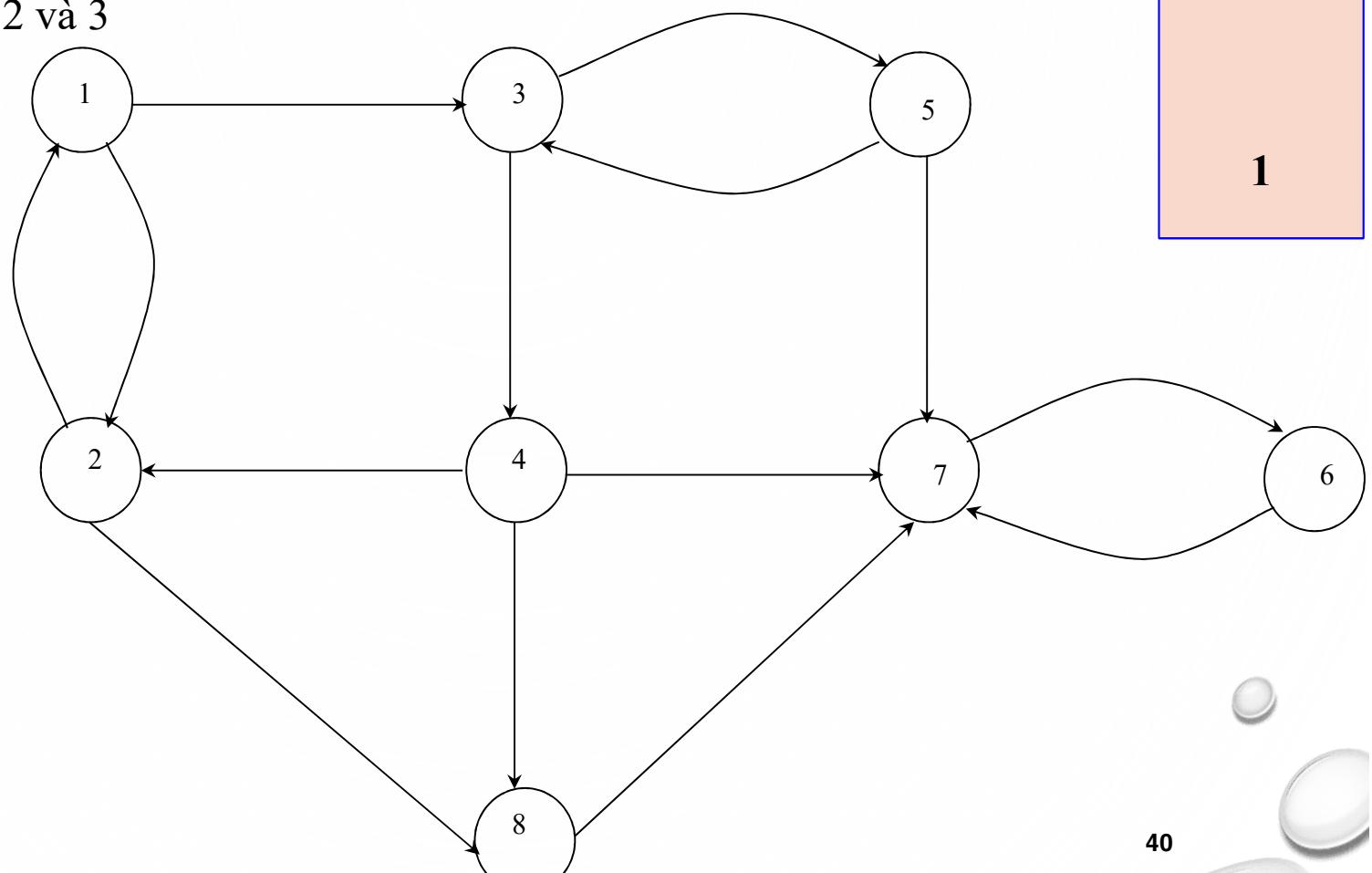


SC(1): 1 -> stack

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3



1

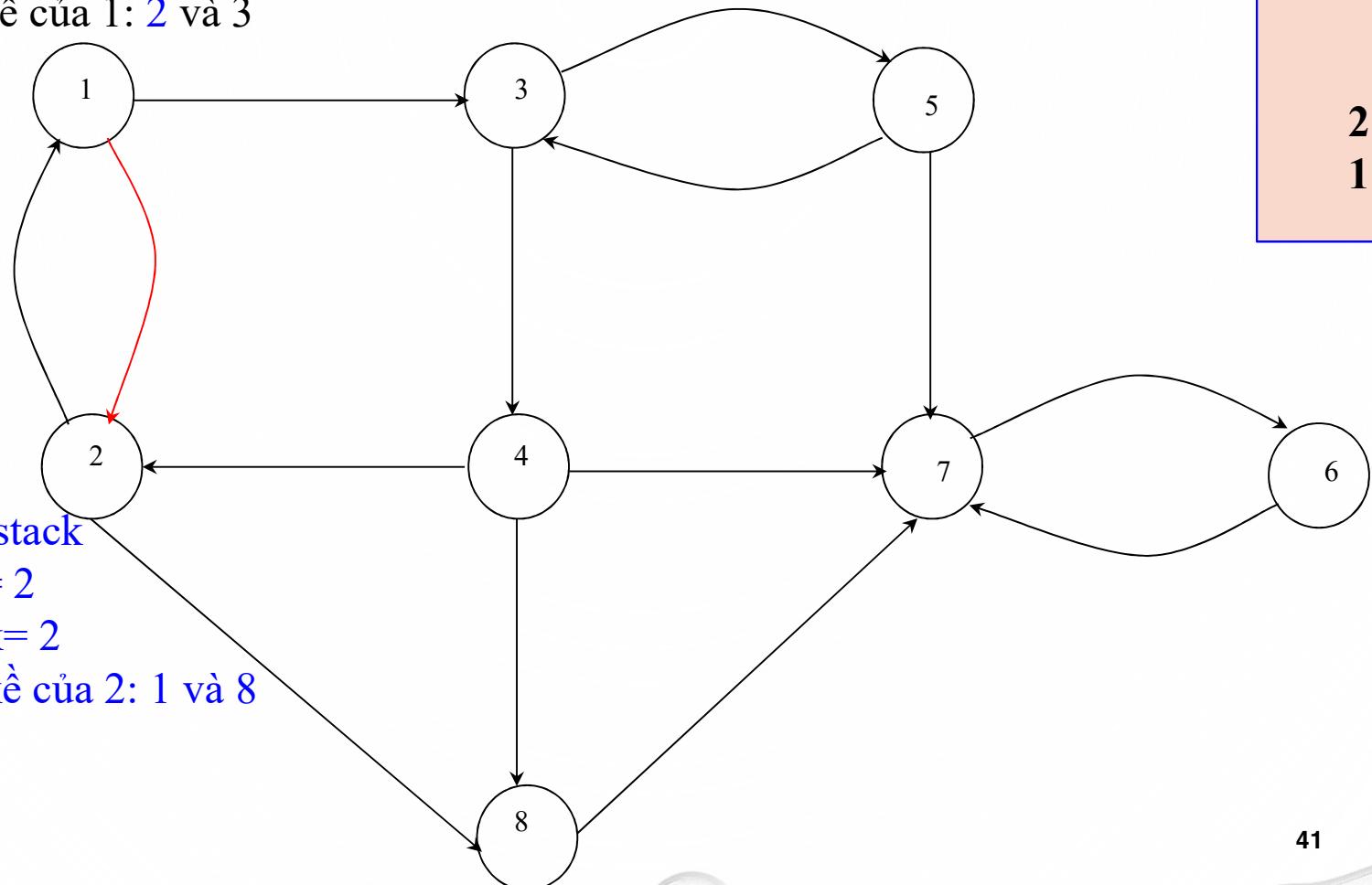
40

SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: **2** và 3



SC(2): 2 -> stack

2.index = 2

2.lowlink= 2

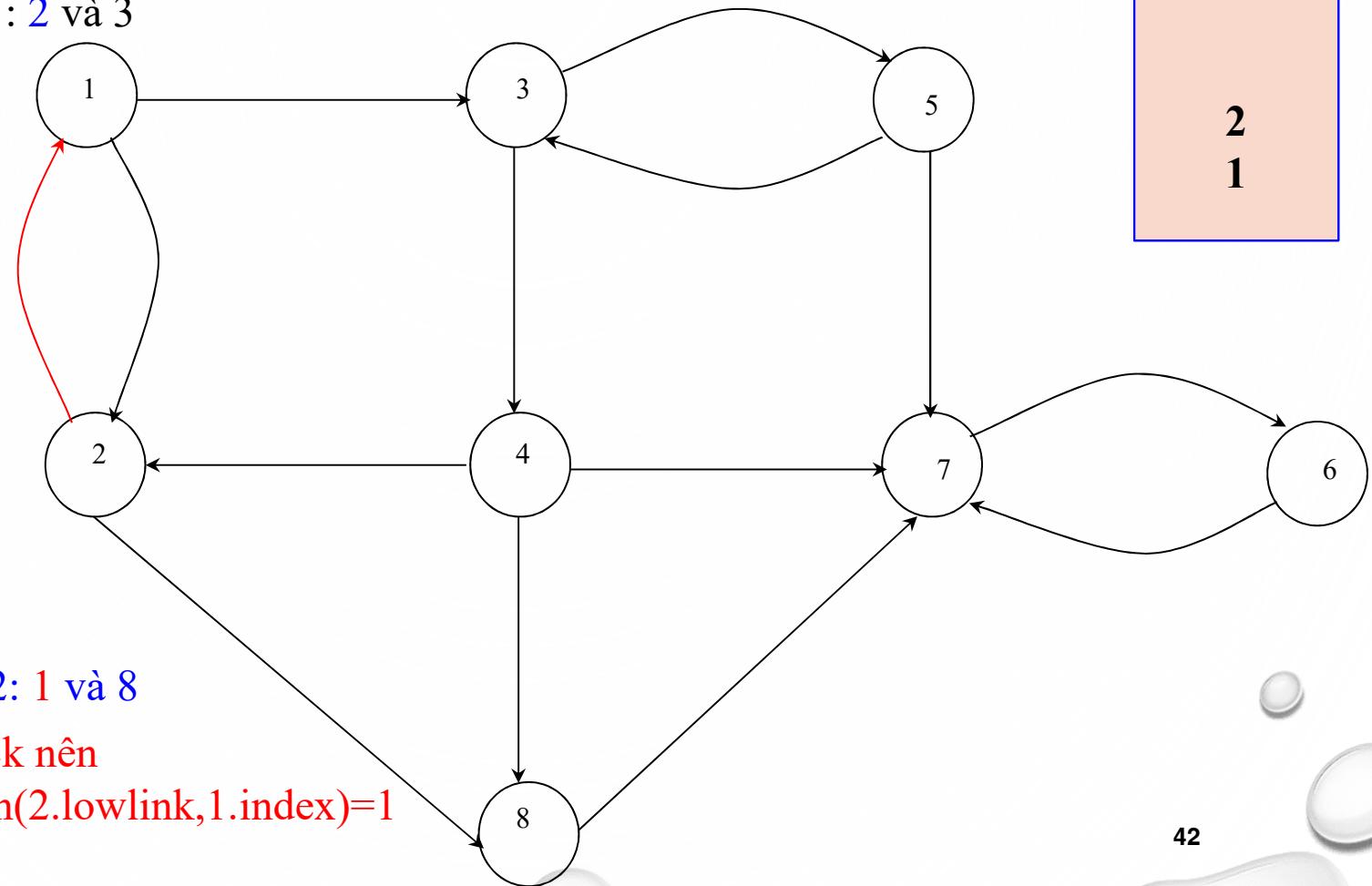
Xét các kè của 2: 1 và 8

SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: **2** và 3



SC(2):

2.index = 2

2.lowlink= 2

Xét các kè của 2: **1** và 8

Vì 1 trong stack nên

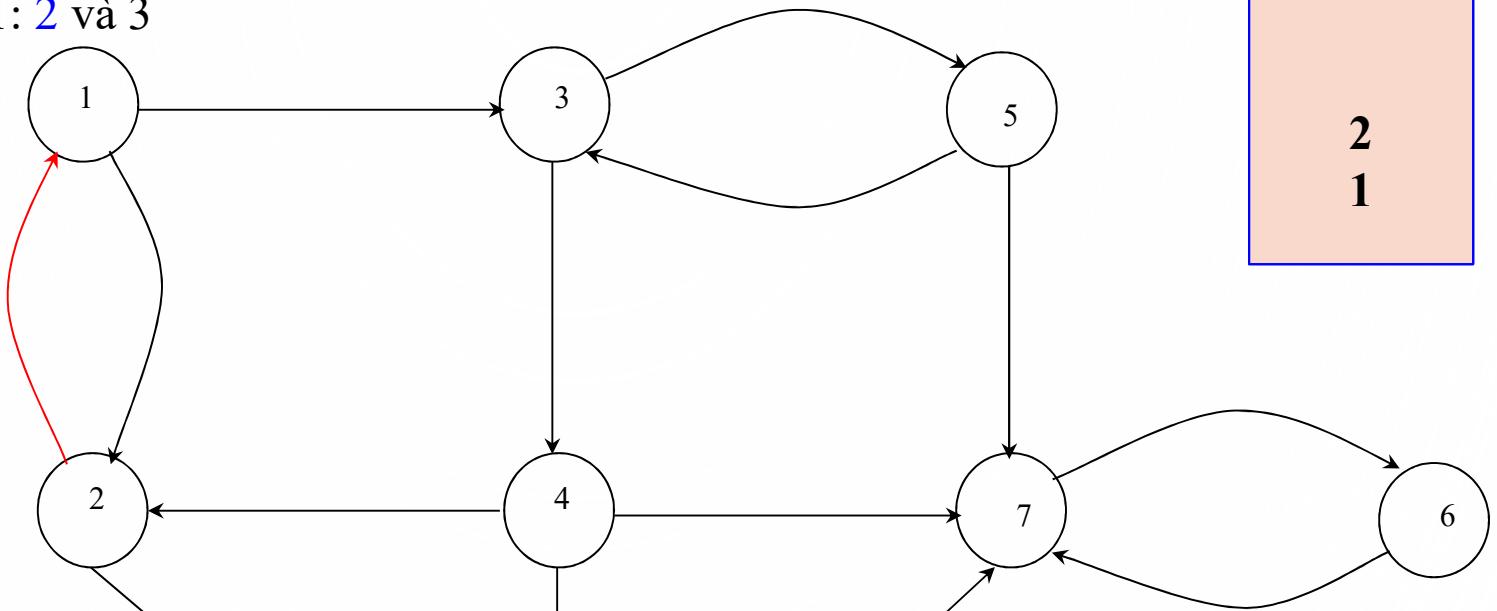
2.lowlink=min(2.lowlink,1.index)=1

SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: **2** và 3



SC(2):

2.index = 2

2.lowlink= 1

Xét các kè của 2: **1** và 8

Vì 1 trong stack nên

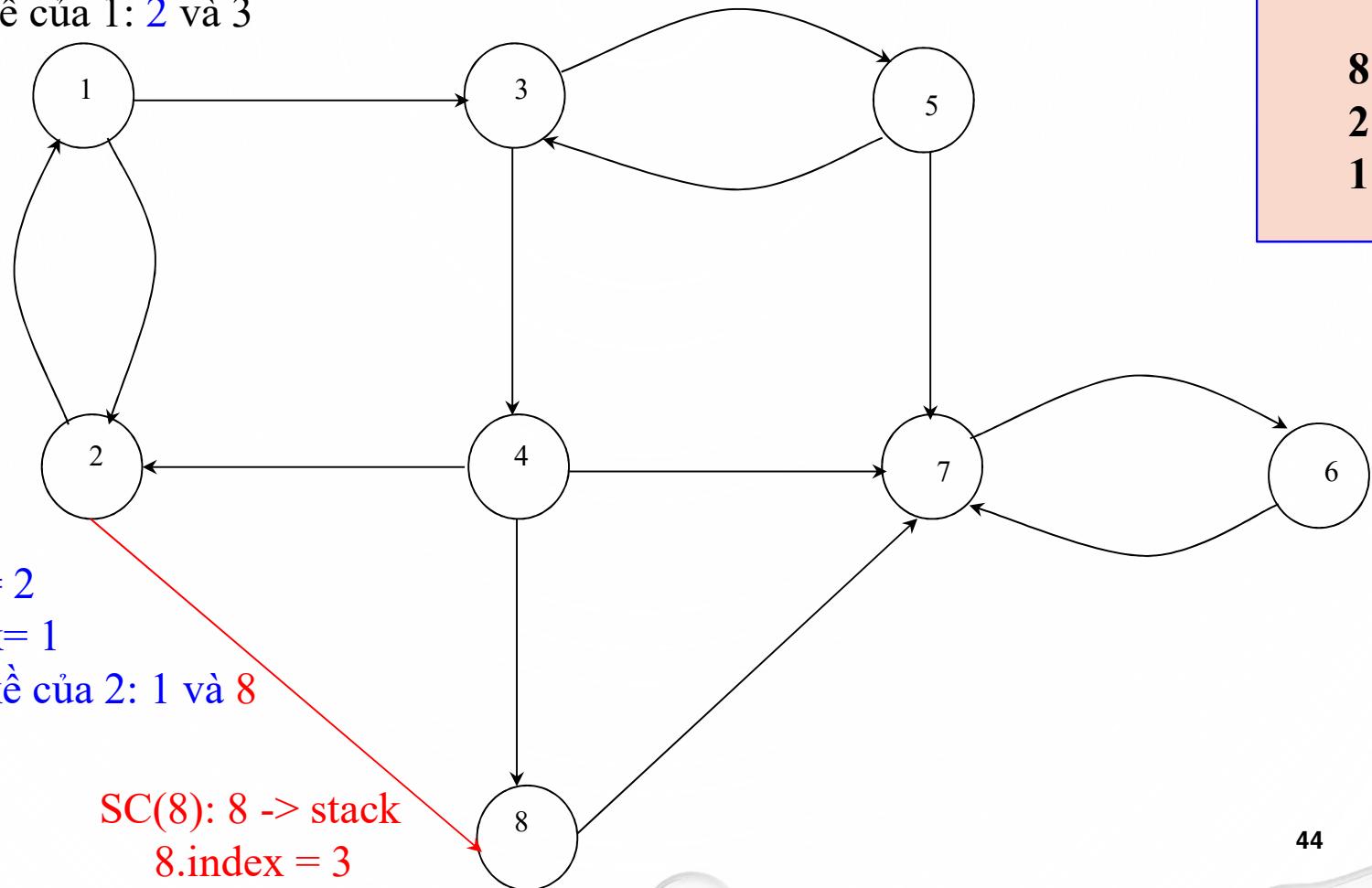
2.lowlink=min(2.lowlink,1.index)=1

SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3

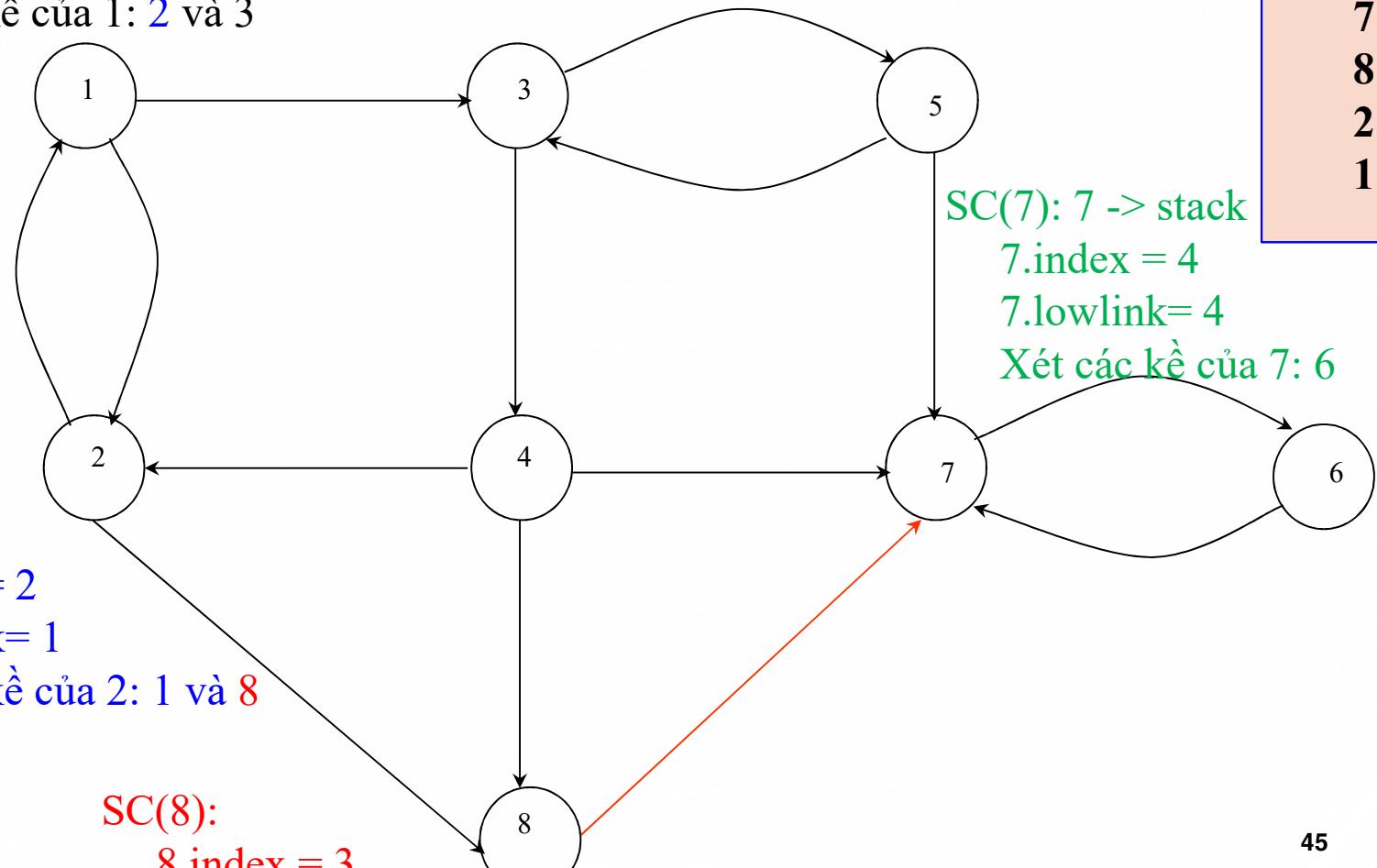


SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3



SC(2):

2.index = 2

2.lowlink= 1

Xét các kè của 2: 1 và 8

SC(8):

8.index = 3

8.lowlink= 3

Xét các kè của 8: 7

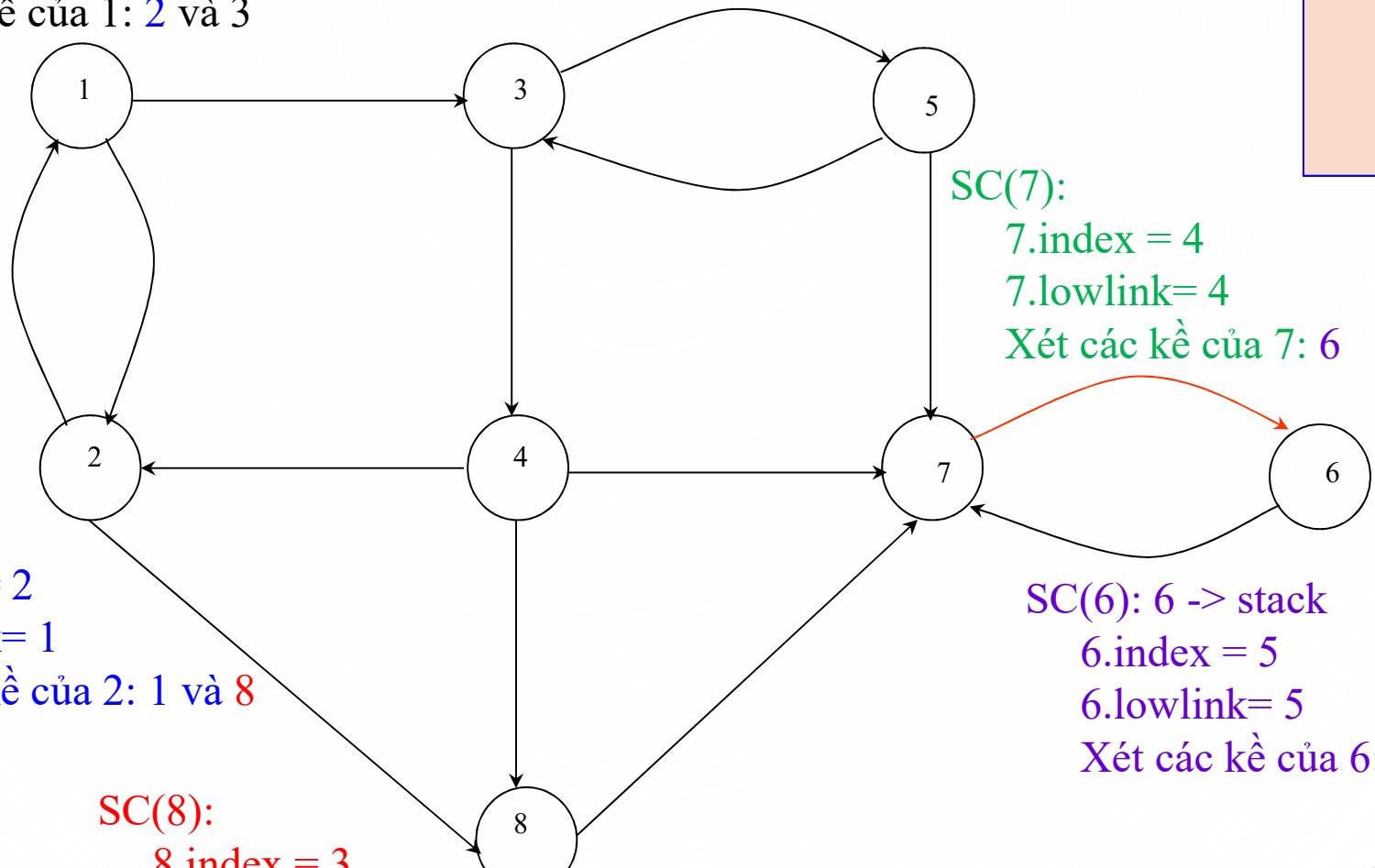
6
7
8
2
1

SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3



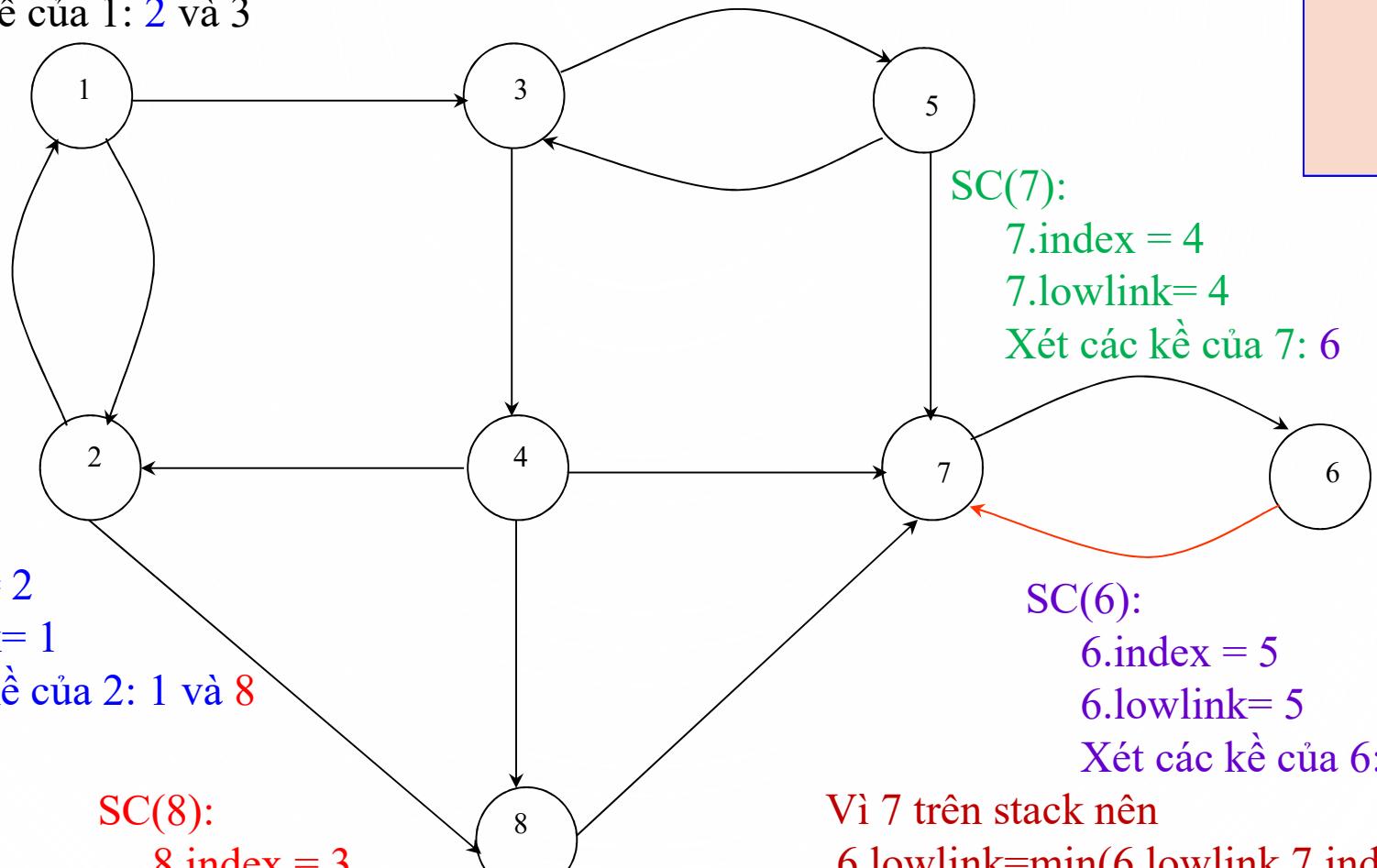
6
7
8
2
1

SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3



SC(2):

2.index = 2

2.lowlink= 1

Xét các kè của 2: 1 và 8

SC(8):

8.index = 3

8.lowlink= 3

Xét các kè của 8: 7

SC(7):

7.index = 4

7.lowlink= 4

Xét các kè của 7: 6

SC(6):

6.index = 5

6.lowlink= 5

Xét các kè của 6: 7

Vì 7 trên stack nên

6.lowlink=min(6.lowlink,7.index)=4

6  
7  
8  
2  
1

SC(1):

1.index = 1

1.lowlink= 1

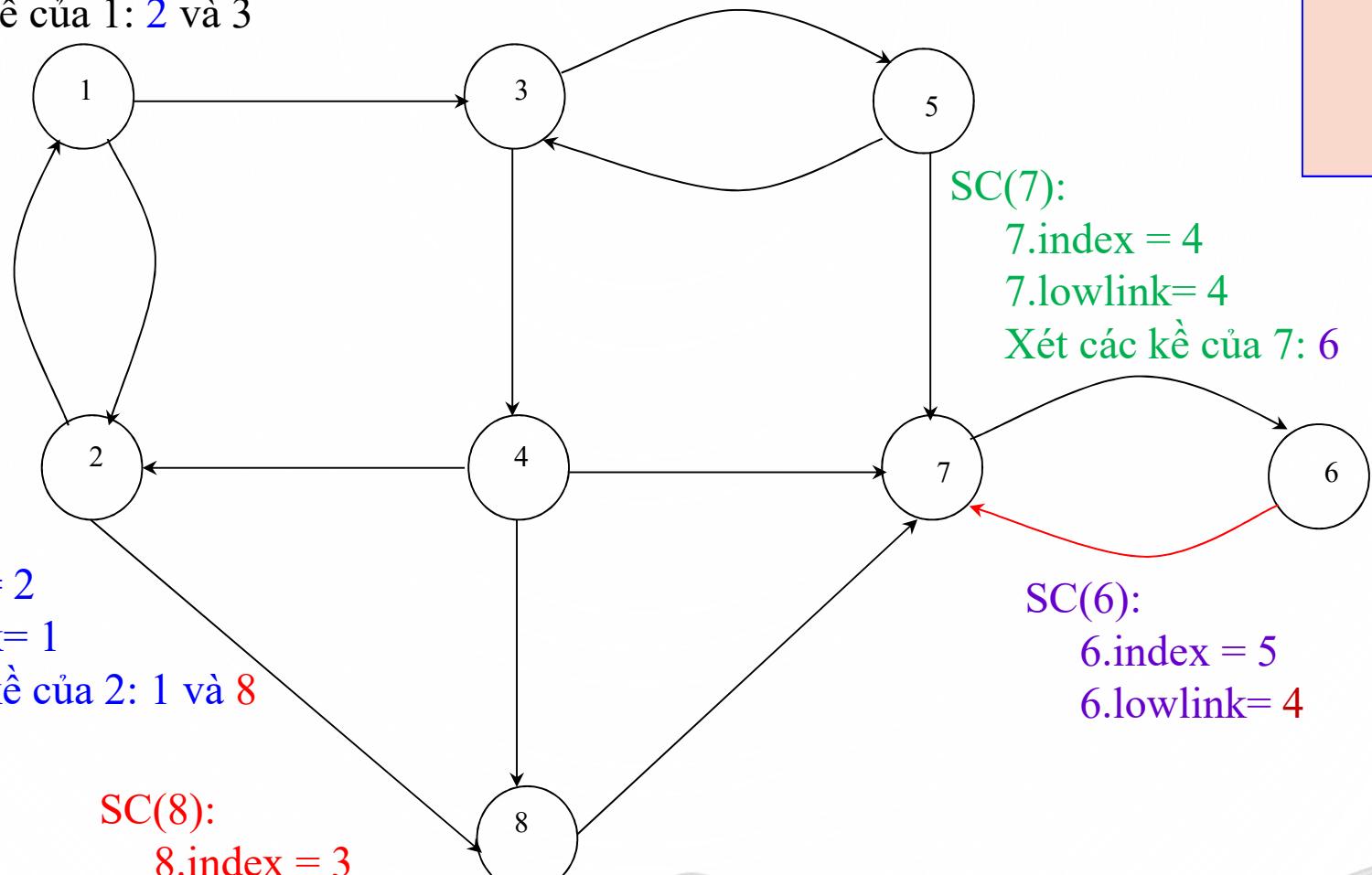
Xét các kè của 1: 2 và 3

SC(2):

2.index = 2

2.lowlink= 1

Xét các kè của 2: 1 và 8



SC(7):

7.index = 4

7.lowlink= 4

Xét các kè của 7: 6

SC(6):

6.index = 5

6.lowlink= 4

SC(8):

8.index = 3

8.lowlink= 3

Xét các kè của 8: 7

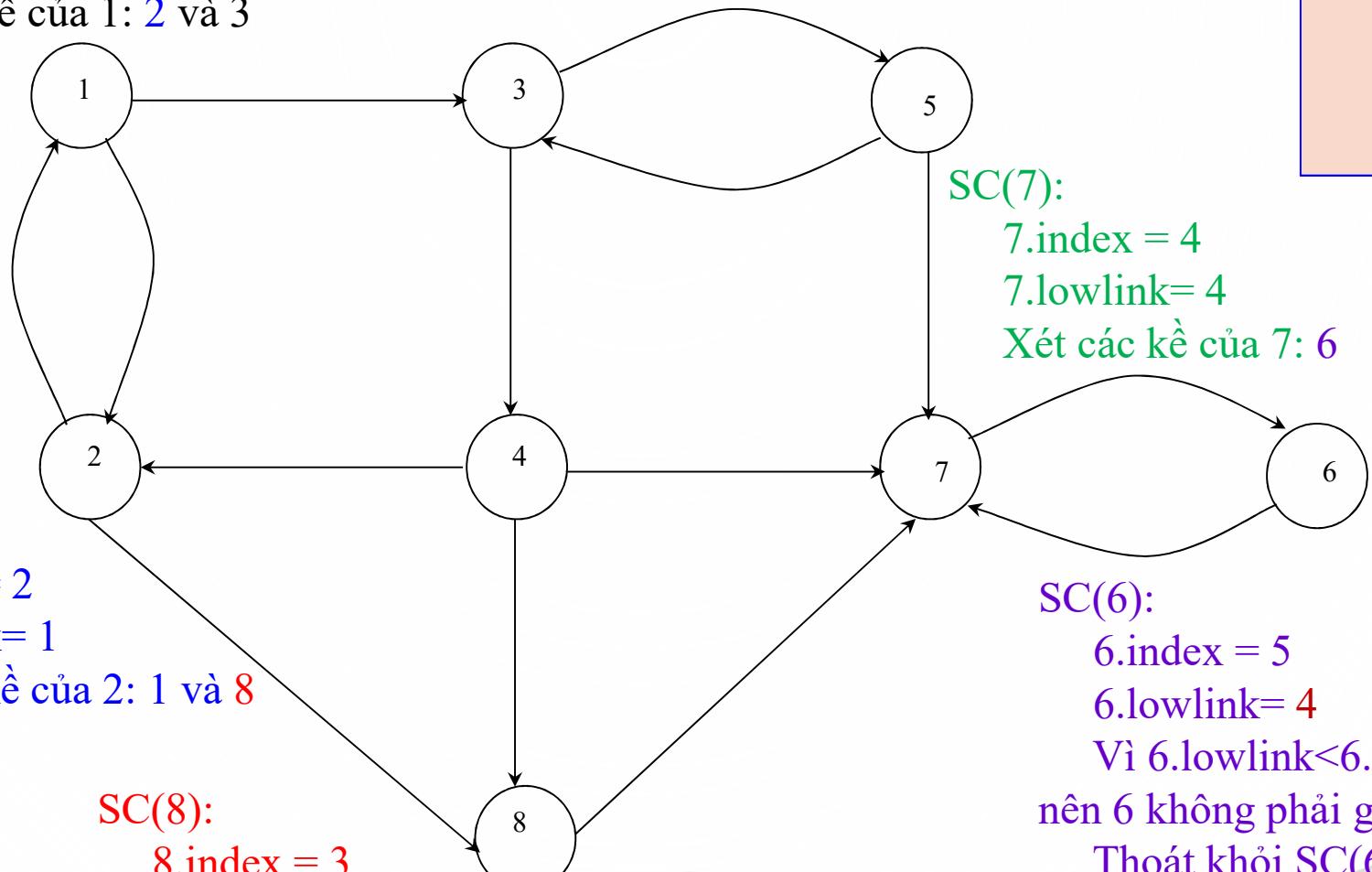
6
7
8
2
1

SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3



SC(2):

2.index = 2

2.lowlink= 1

Xét các kè của 2: 1 và 8

SC(8):

8.index = 3

8.lowlink= 3

Xét các kè của 8: 7

SC(7):

7.index = 4

7.lowlink= 4

Xét các kè của 7: 6

SC(6):

6.index = 5

6.lowlink= 4

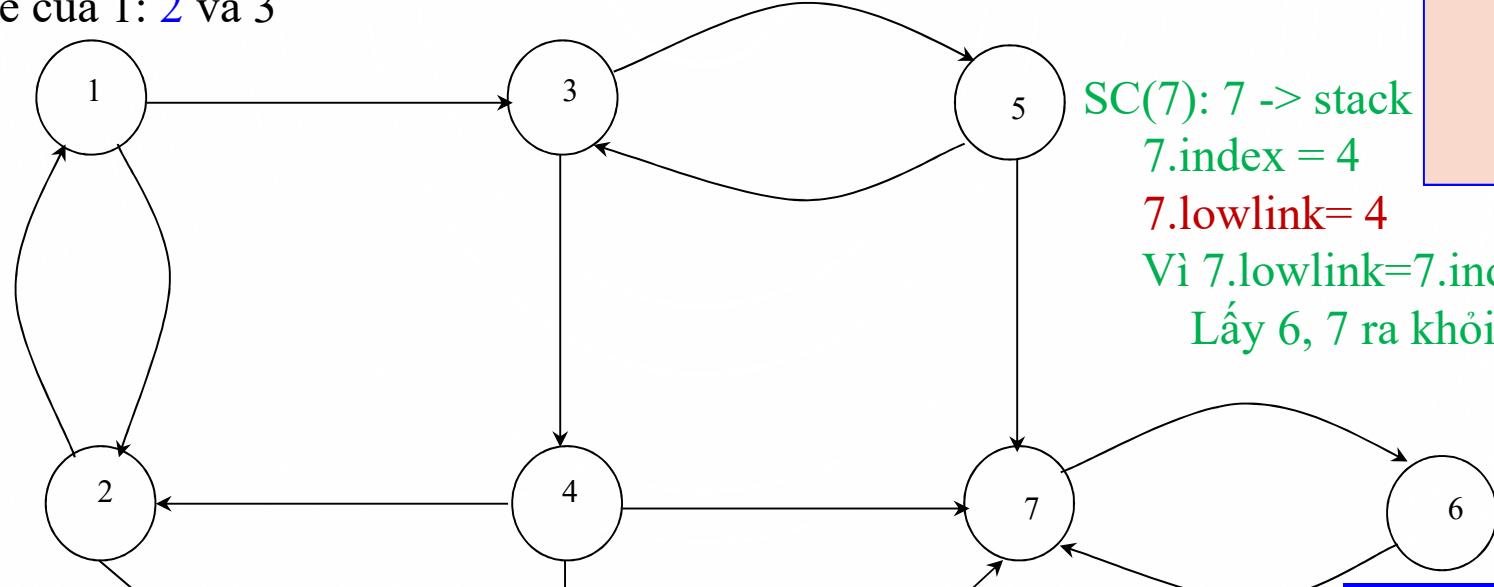
Vì 6.lowlink<6.index  
nên 6 không phải gốc  
Thoát khỏi SC(6)

SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3



SC(2):

2.index = 2

2.lowlink= 1

Xét các kè của 2: 1 và 8

SC(8):

8.index = 3

8.lowlink= 3

Xét các kè của 8: 7

SC(7): 7 -> stack

7.index = 4

7.lowlink= 4

Vì 7.lowlink=7.index nên:  
Lấy 6, 7 ra khỏi stack

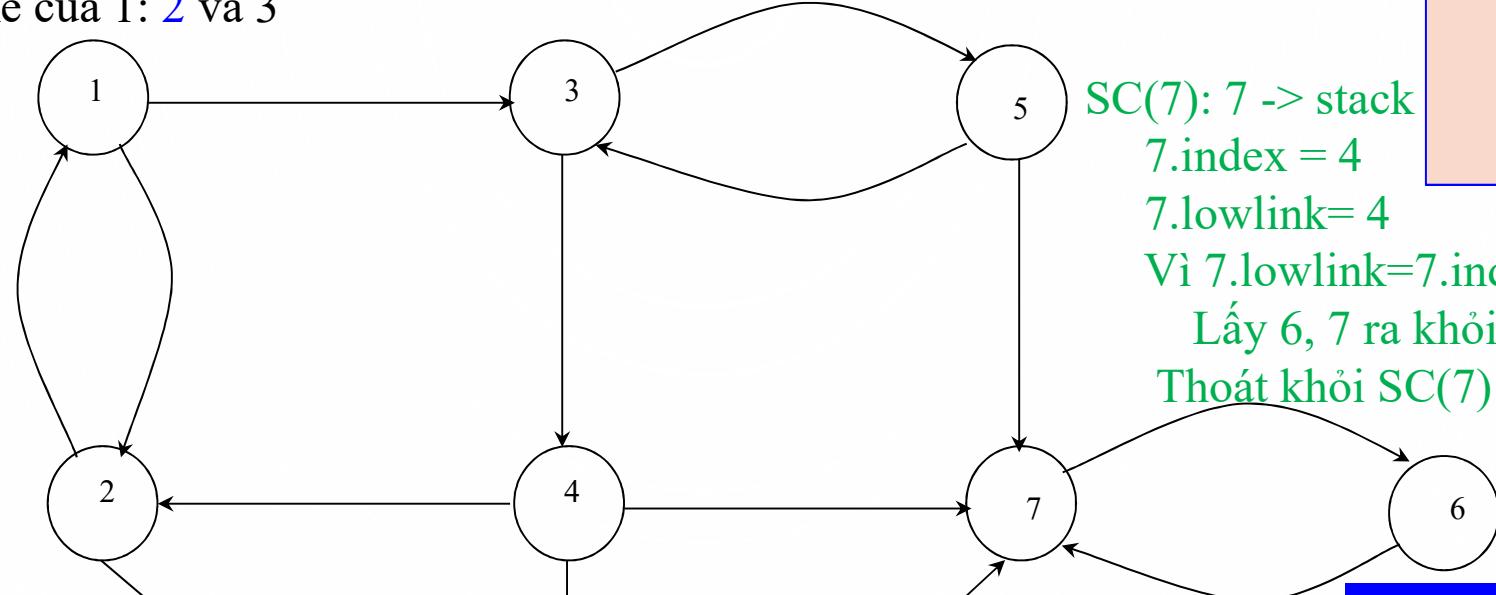
6.index = 5  
6.lowlink= 4

SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3



SC(2):

2.index = 2

2.lowlink= 1

Xét các kè của 2: 1 và 8

SC(8):

8.index = 3

8.lowlink= 3

Xét các kè của 8: 7

SC(7): 7 -> stack

7.index = 4

7.lowlink= 4

Vì 7.lowlink=7.index nên:

Lấy 6, 7 ra khỏi stack

Thoát khỏi SC(7)

6.index = 5

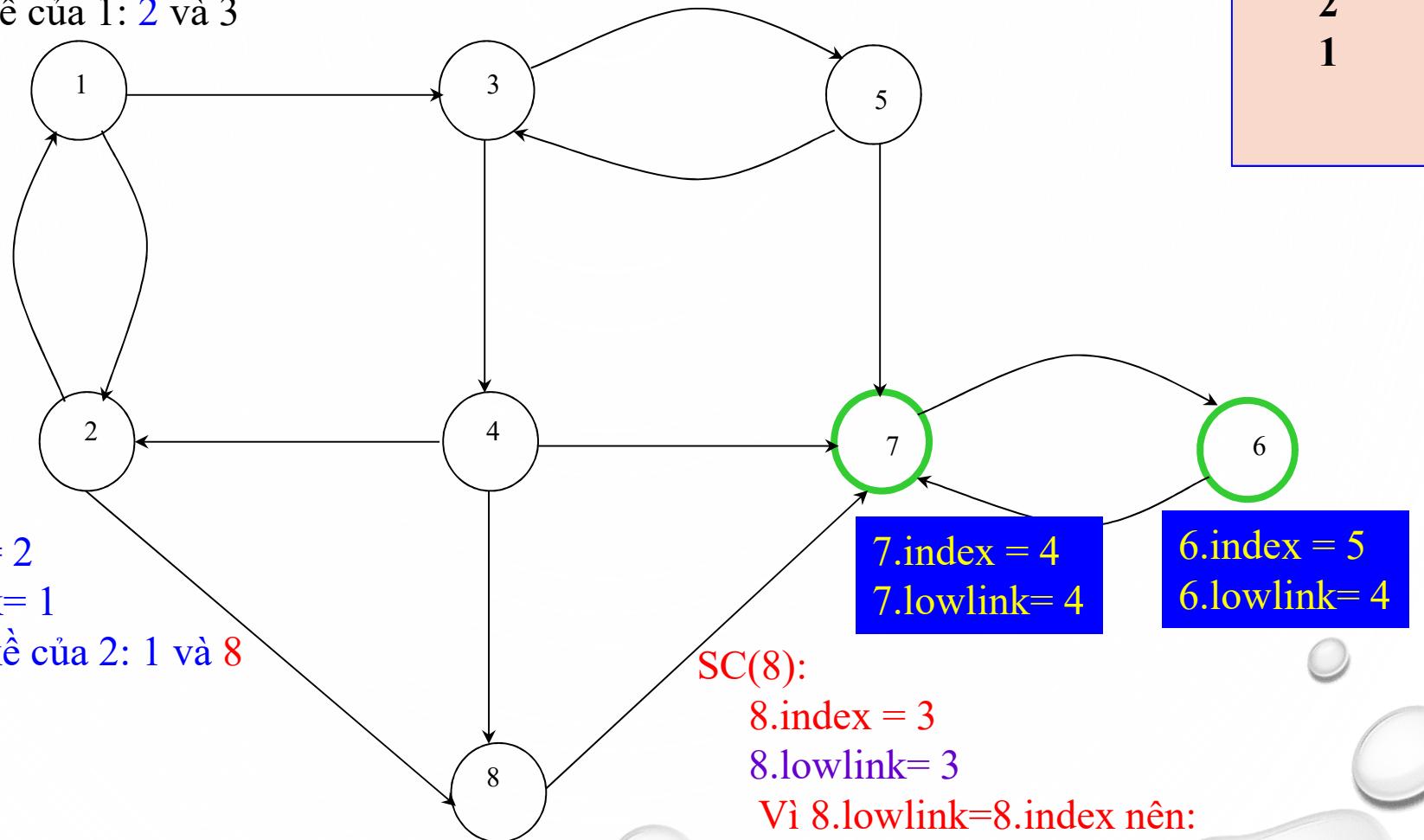
6.lowlink= 4

SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3

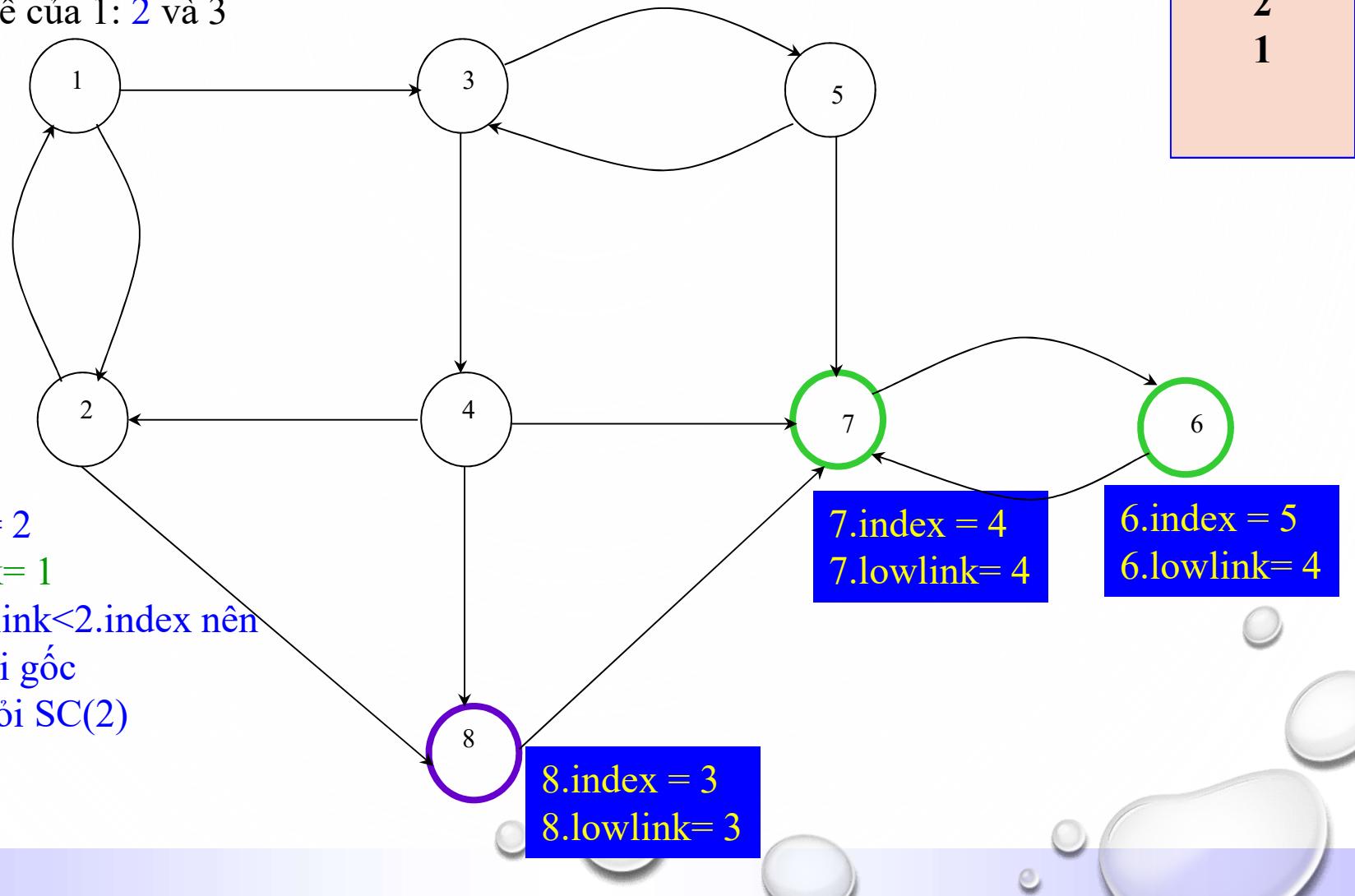


SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3



SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3

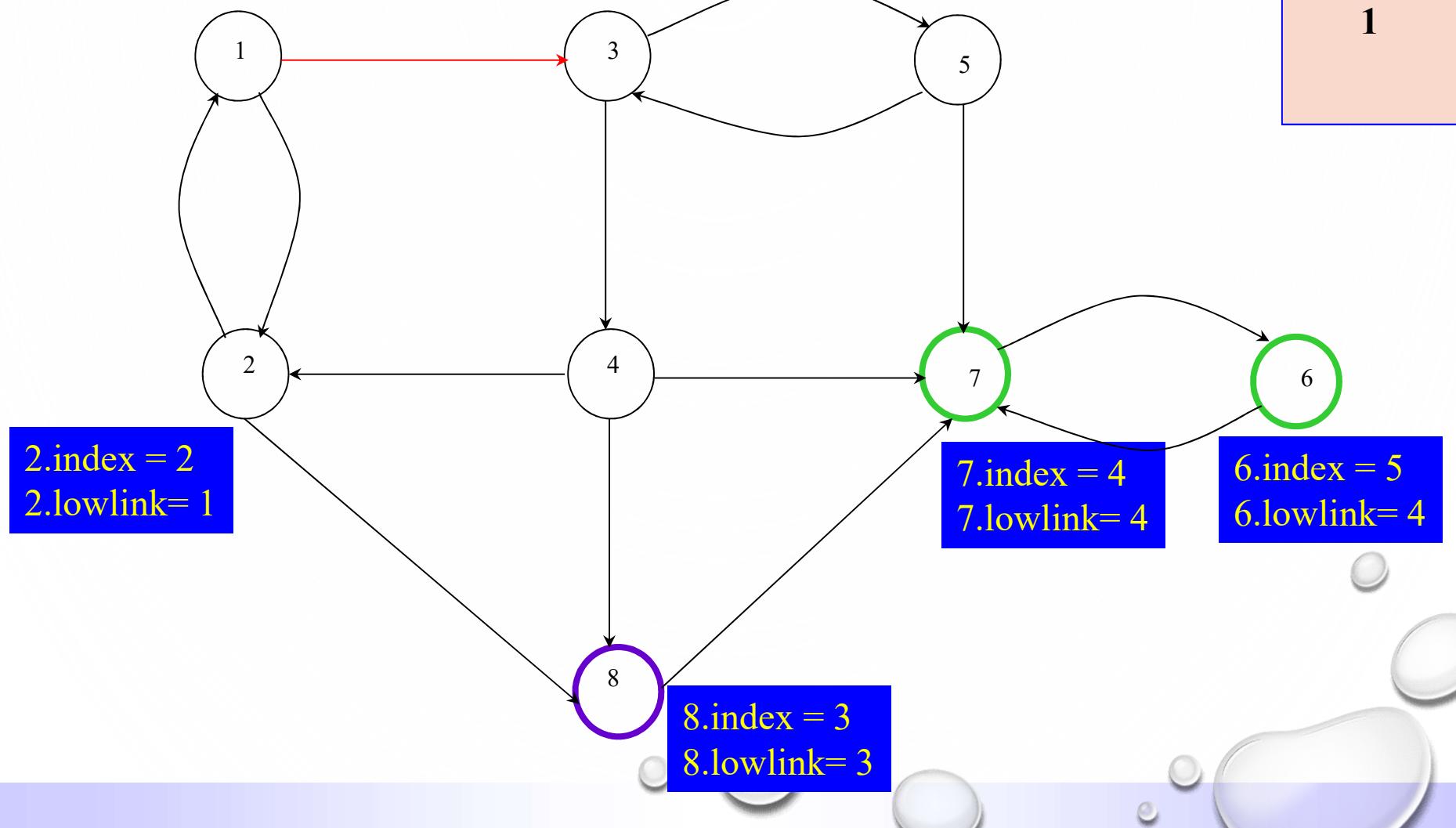
SC(3): 3->stack

3.index = 6

3.lowlink= 6

Xét các kè của 3: 4 và 5

3  
2  
1



SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3

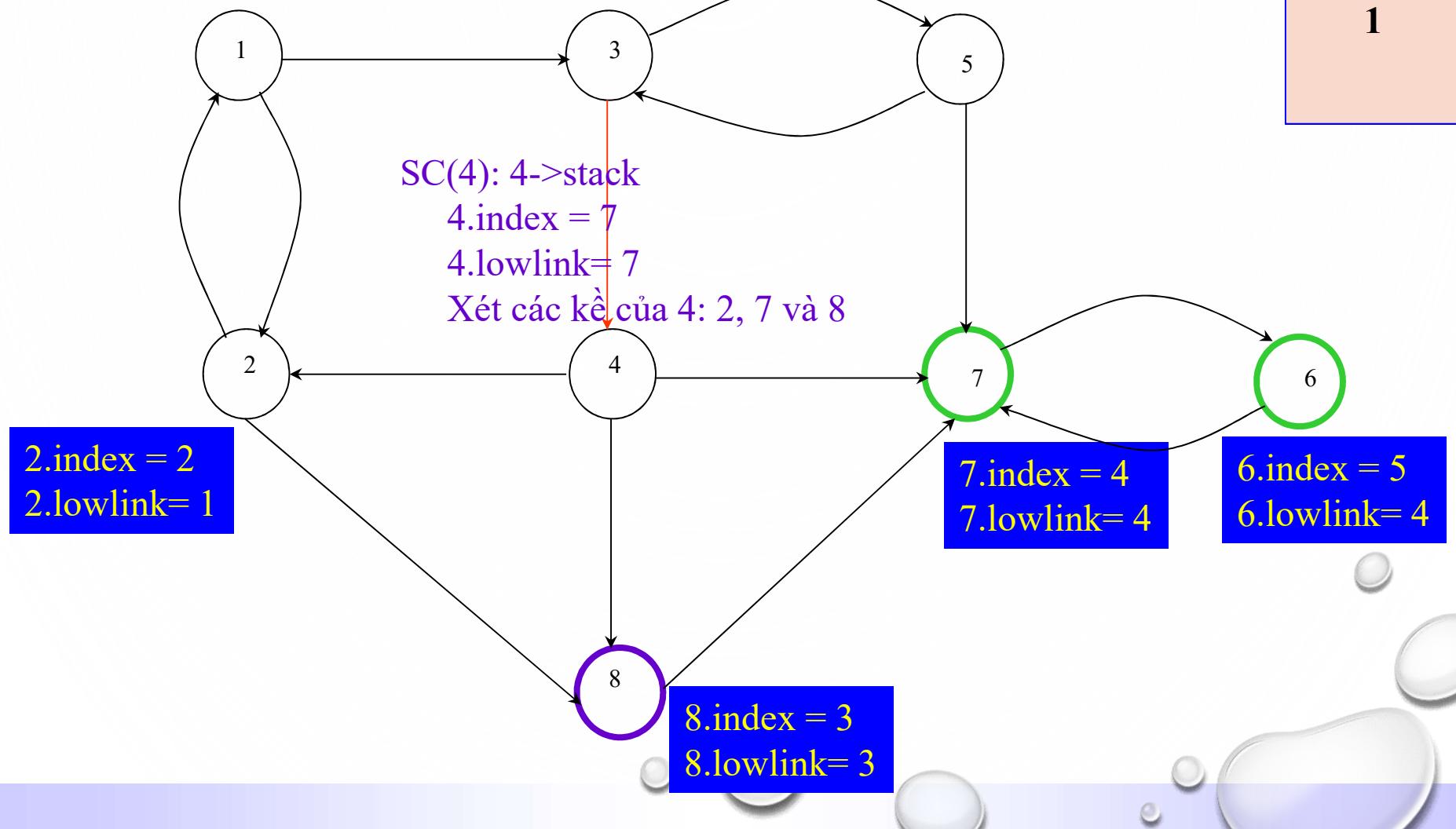
SC(3):

3.index = 6

3.lowlink= 6

Xét các kè của 3: 4 và 5

4  
3  
2  
1



4
3
2
1

SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3

SC(3):

3.index = 6

3.lowlink= 6

Xét các kè của 3: 4 và 5

SC(4):

4.index = 7

4.lowlink= 7

Xét các kè của 4: 2, 7 và 8

2.index = 2

2.lowlink= 1

Vì 2 trên stack nên

4.lowlink=min(4.lowlink,2.index)=2

7.index = 4

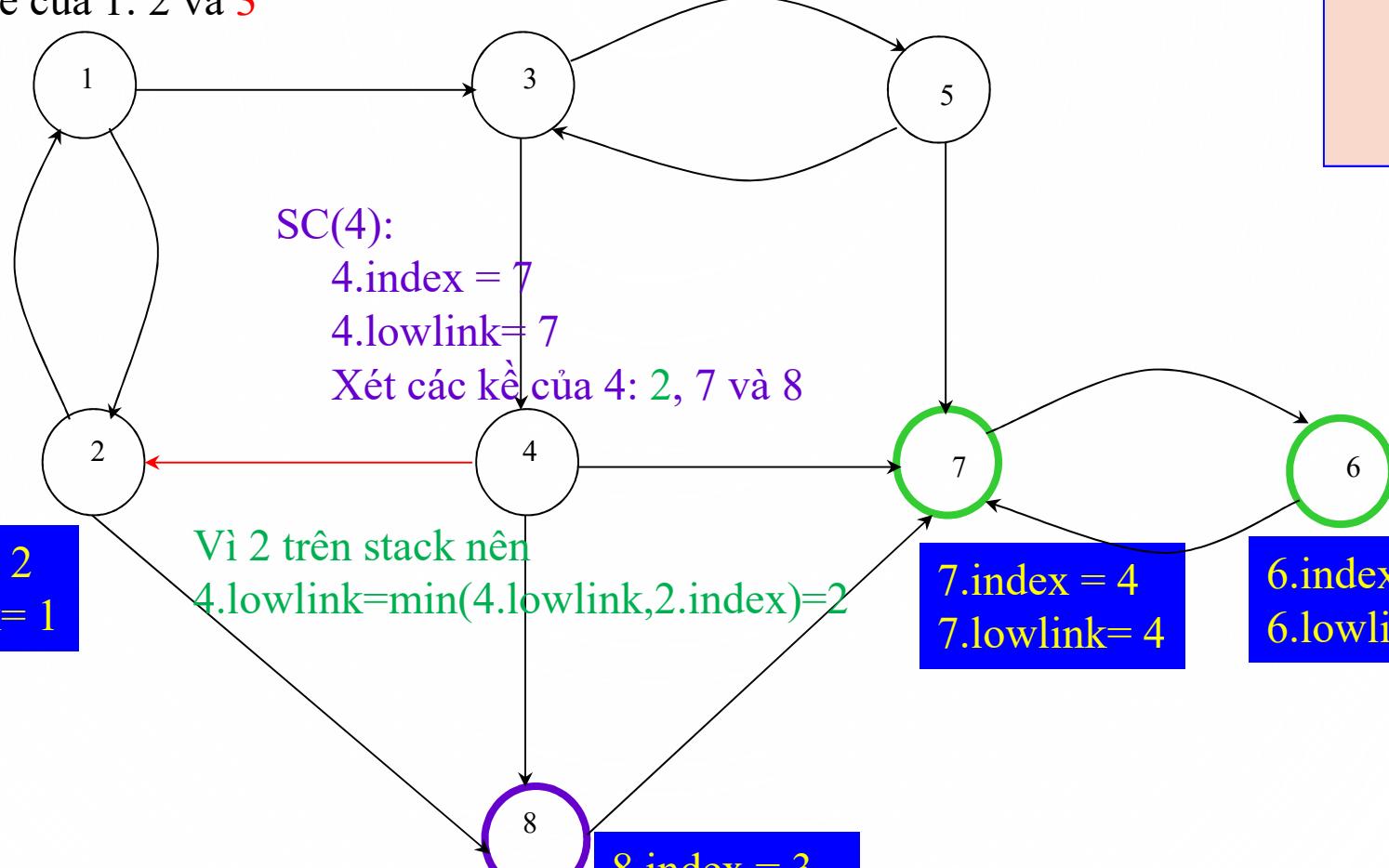
7.lowlink= 4

6.index = 5

6.lowlink= 4

8.index = 3

8.lowlink= 3



SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3

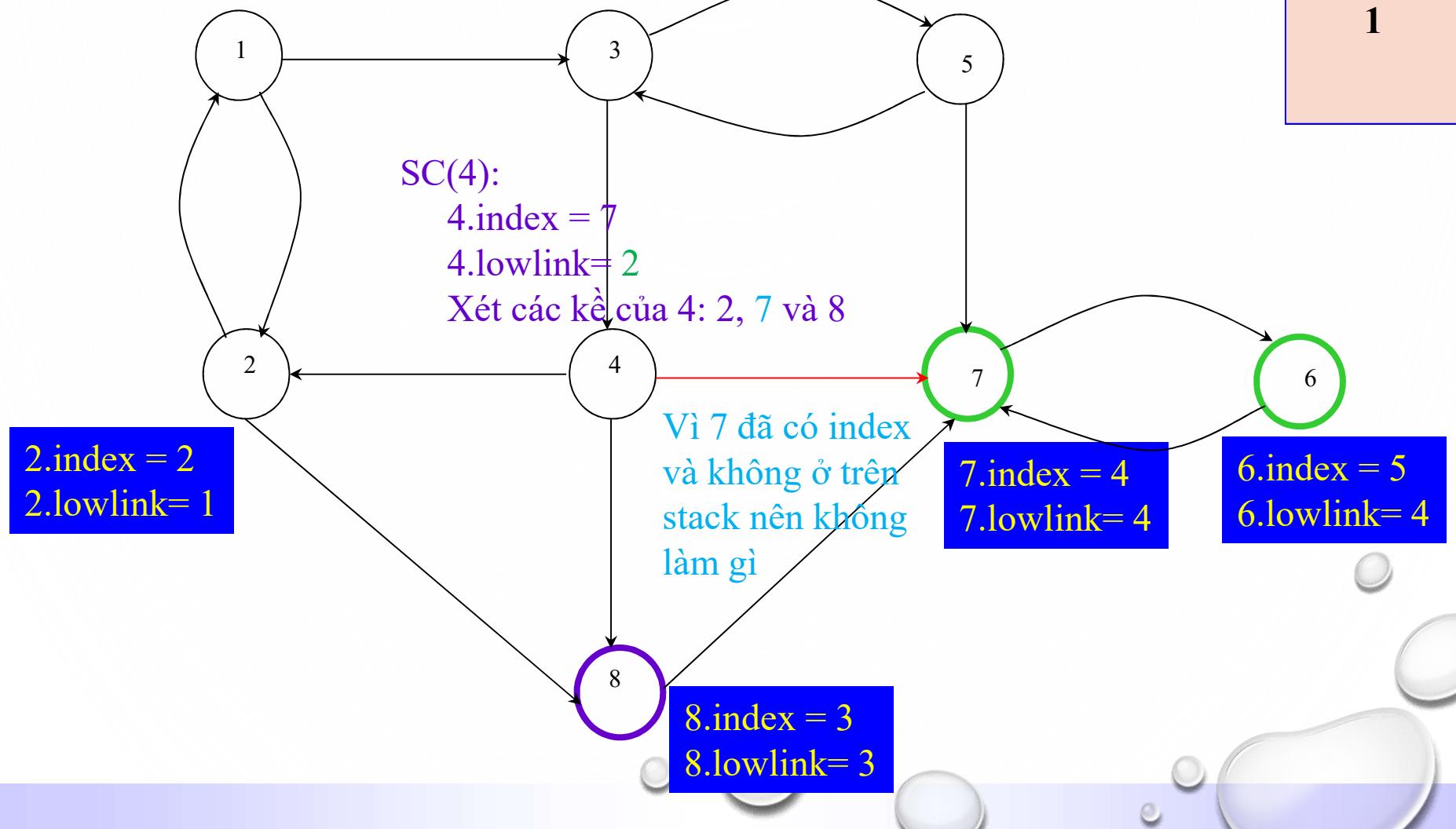
SC(3):

3.index = 6

3.lowlink= 6

Xét các kè của 3: 4 và 5

4  
3  
2  
1



SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3

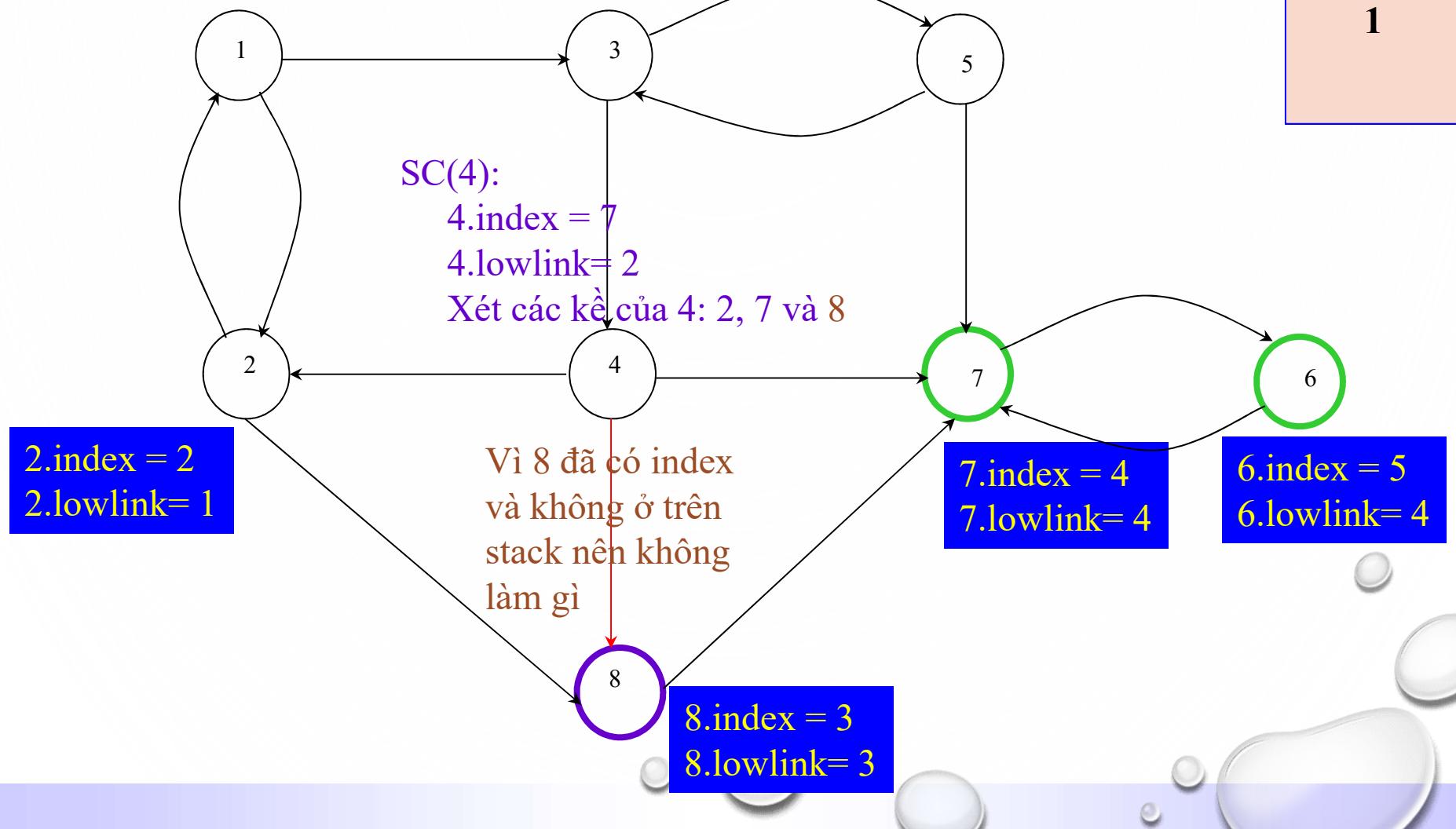
SC(3):

3.index = 6

3.lowlink= 6

Xét các kè của 3: 4 và 5

4  
3  
2  
1



4
3
2
1

SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3

SC(3):

3.index = 6

3.lowlink= 6

Xét các kè của 3: 4 và 5

SC(4):

4.index = 7

4.lowlink= 2

Vì 4.lowlink < 4.index nên

4 không là gốc

Thoát SC(4)

2.index = 2

2.lowlink= 1

7.index = 4

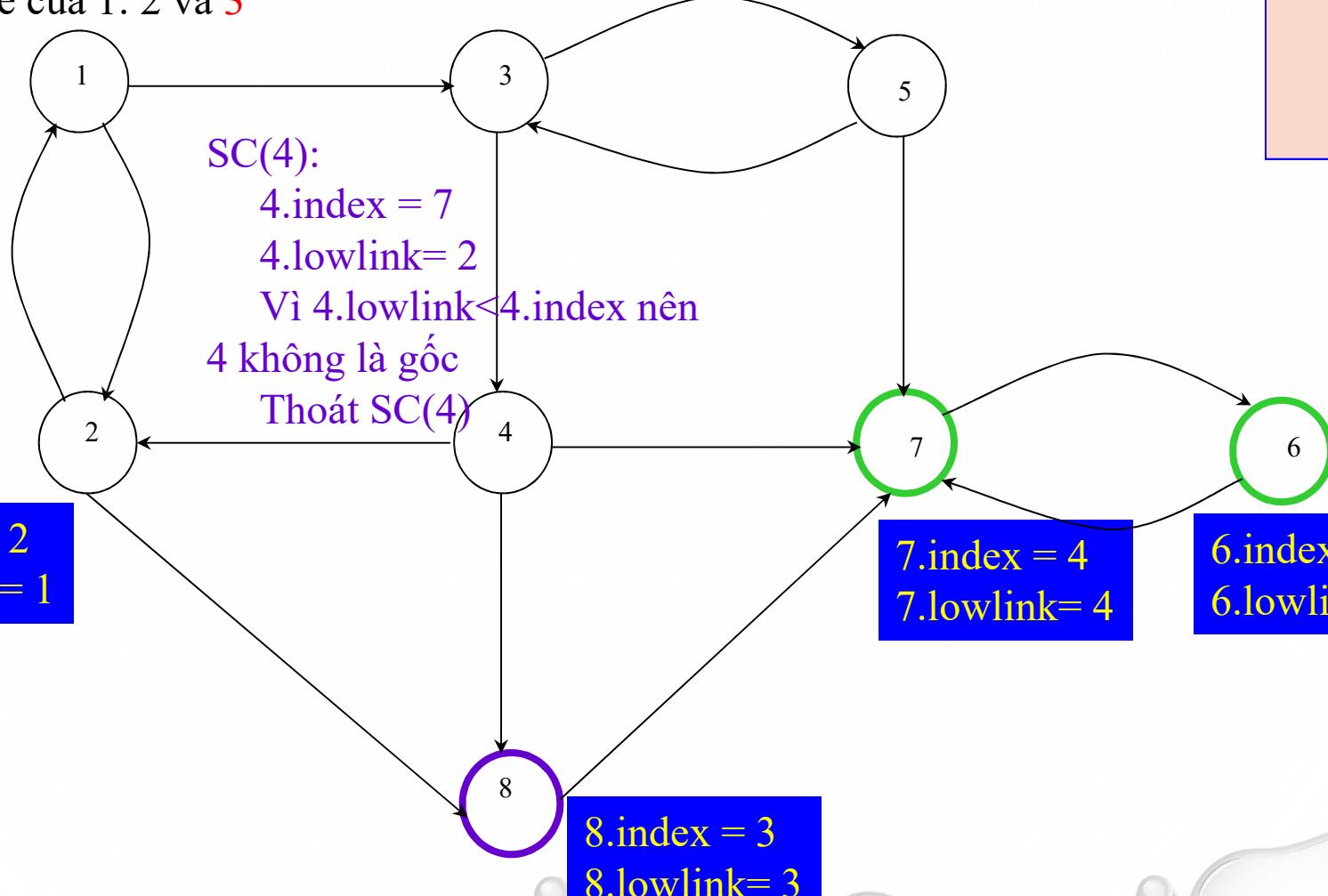
7.lowlink= 4

6.index = 5

6.lowlink= 4

8.index = 3

8.lowlink= 3



SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3

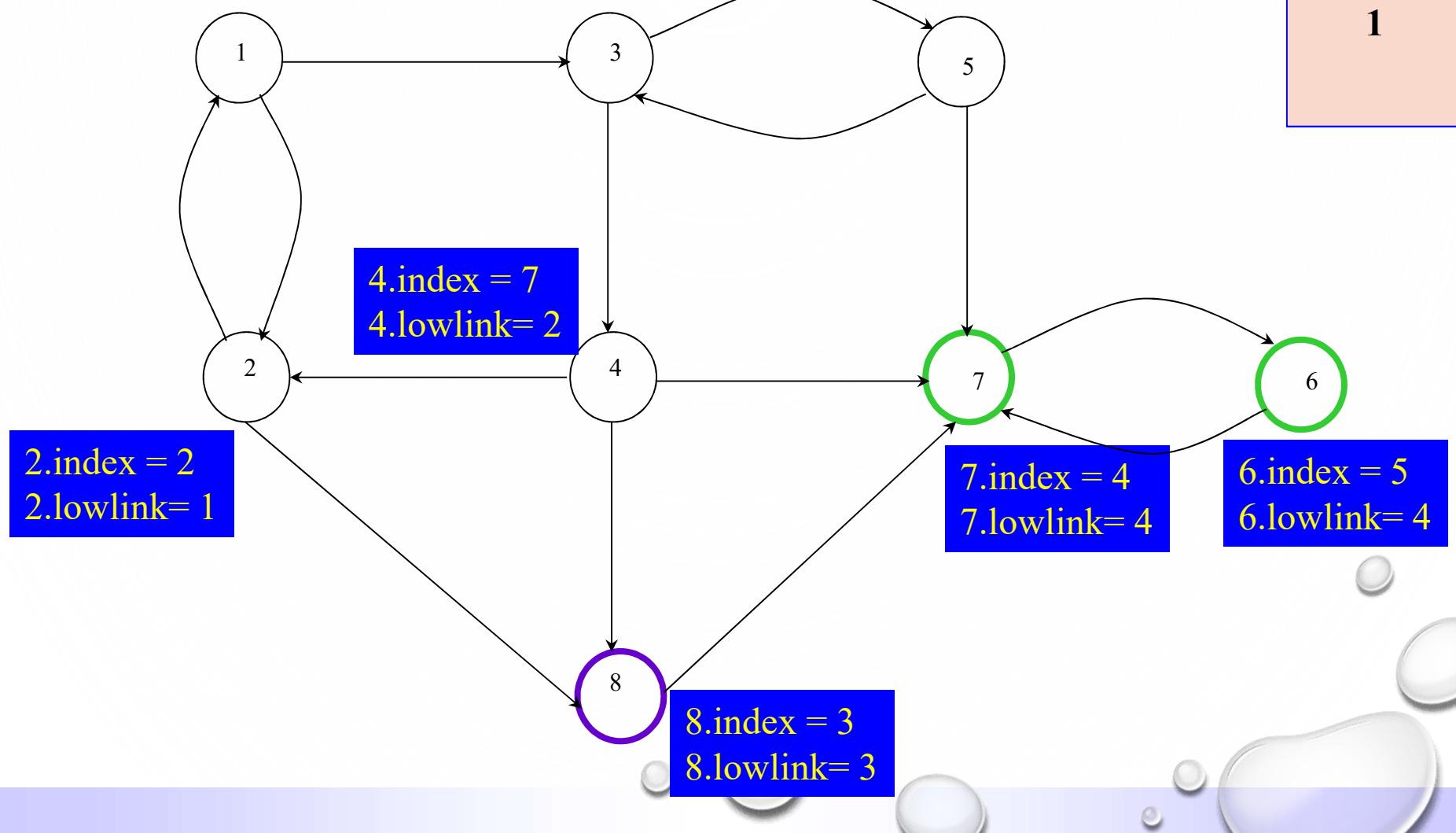
SC(3):

3.index = 6

3.lowlink=min(3.lowlink,4.lowlink)=2

Xét các kè của 3: 4 và 5

4  
3  
2  
1



SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3

SC(3):

3.index = 6

3.lowlink= 2

Xét các kè của 3: 4 và 5

SC(5): 5->stack

5.index = 8

5.lowlink= 8

Xét các kè của 5: 3 và 7

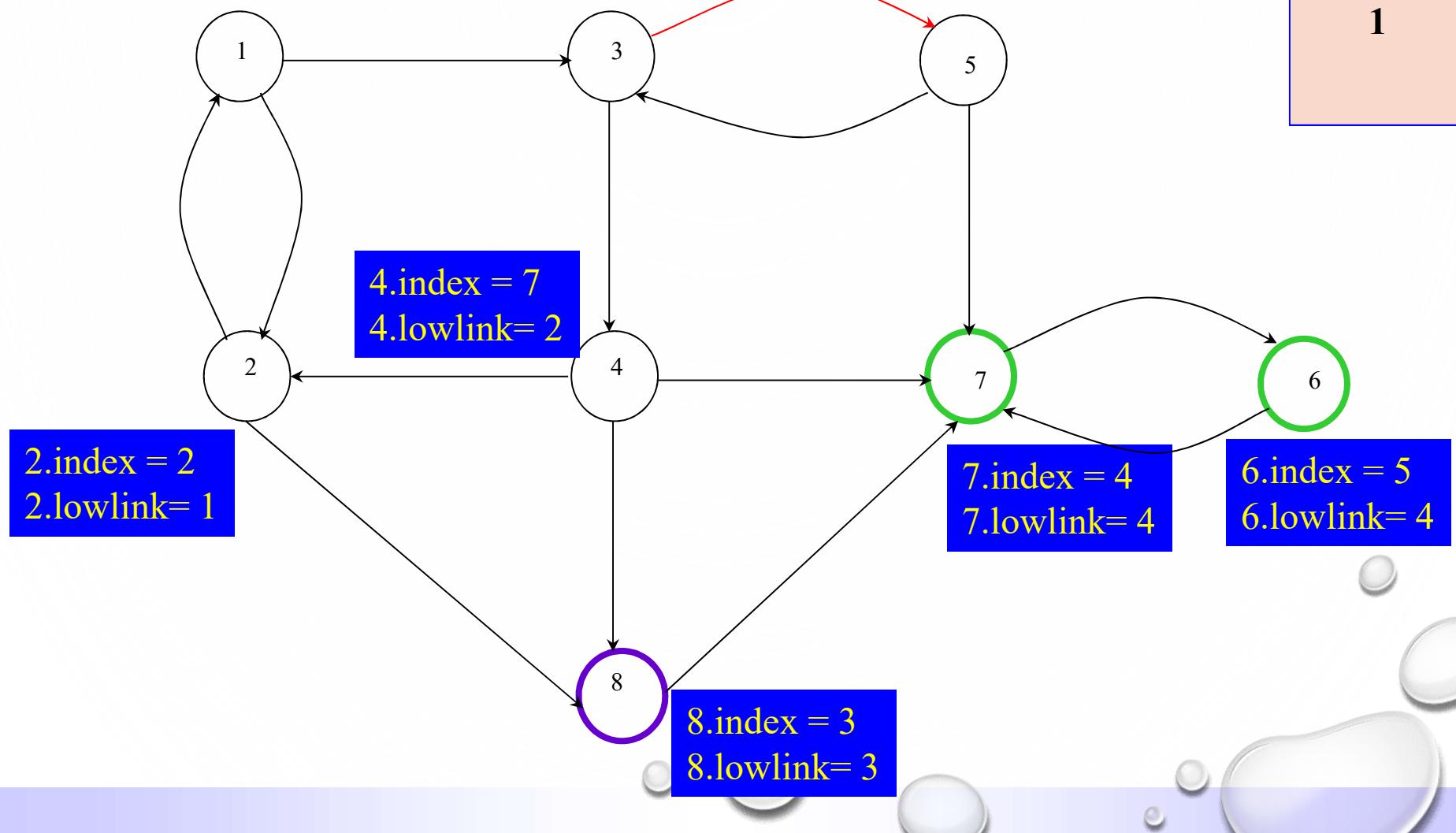
5

4

3

2

1



SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3

SC(3):

3.index = 6

3.lowlink= 2

Xét các kè của 3: 4 và 5

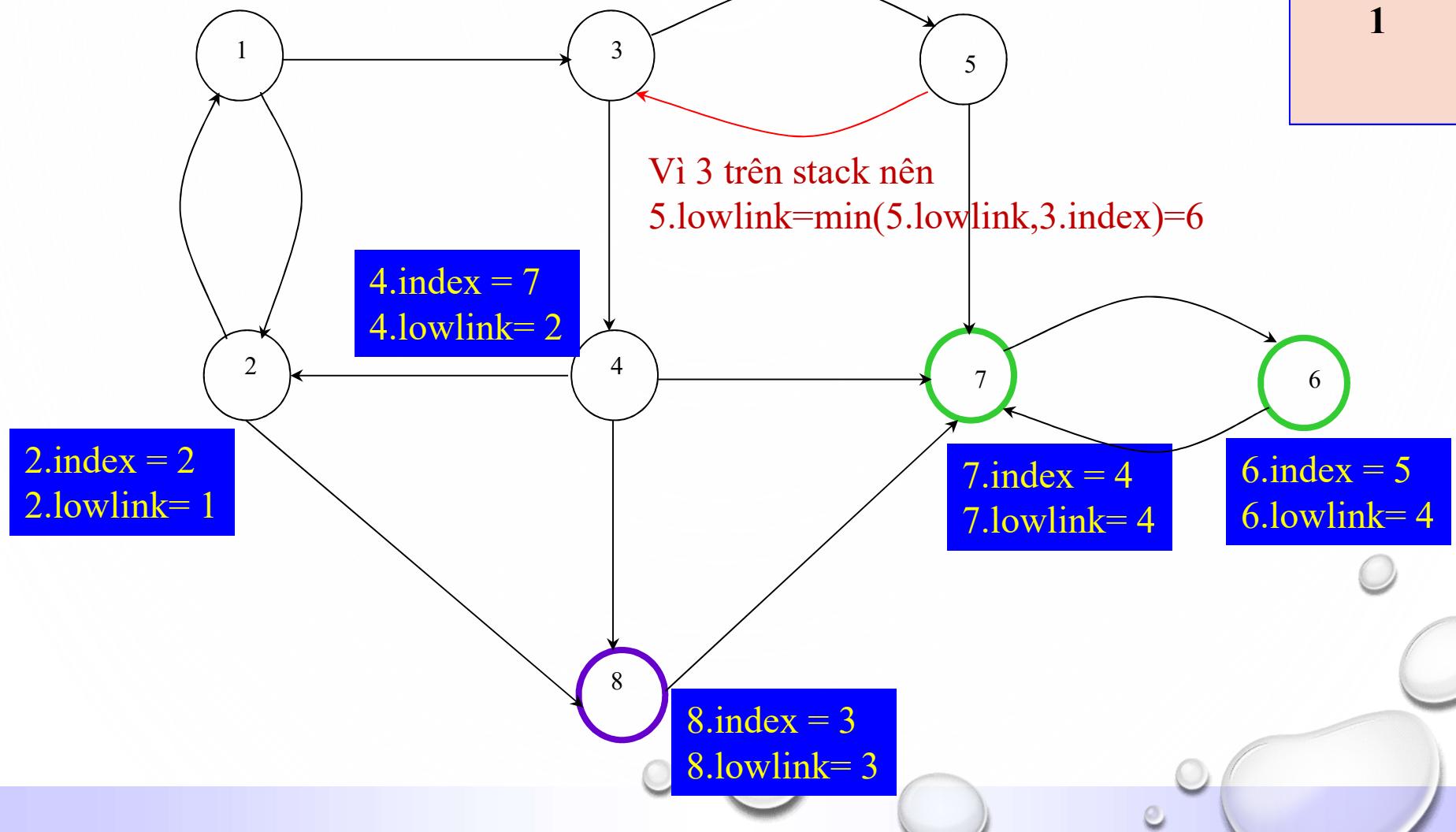
SC(5):

5.index = 8

5.lowlink= 8

Xét các kè của 5: 3 và 7

5  
4  
3  
2  
1



SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3

SC(3):

3.index = 6

3.lowlink= 2

Xét các kè của 3: 4 và 5

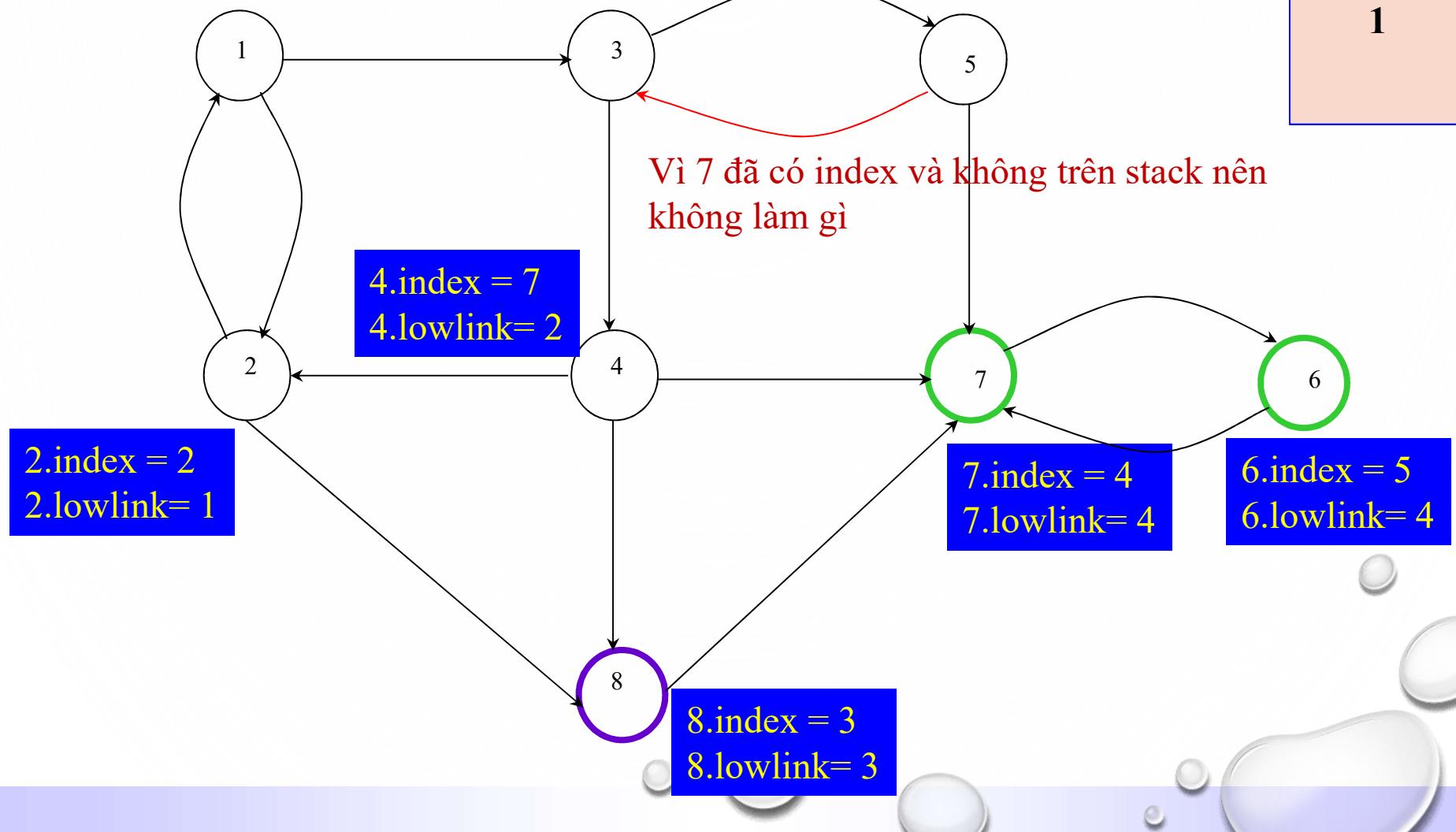
SC(5):

5.index = 8

5.lowlink= 6

Xét các kè của 5: 3 và 7

5  
4  
3  
2  
1



SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3

SC(3):

3.index = 6

3.lowlink= 2

Xét các kè của 3: 4 và 5

SC(5):

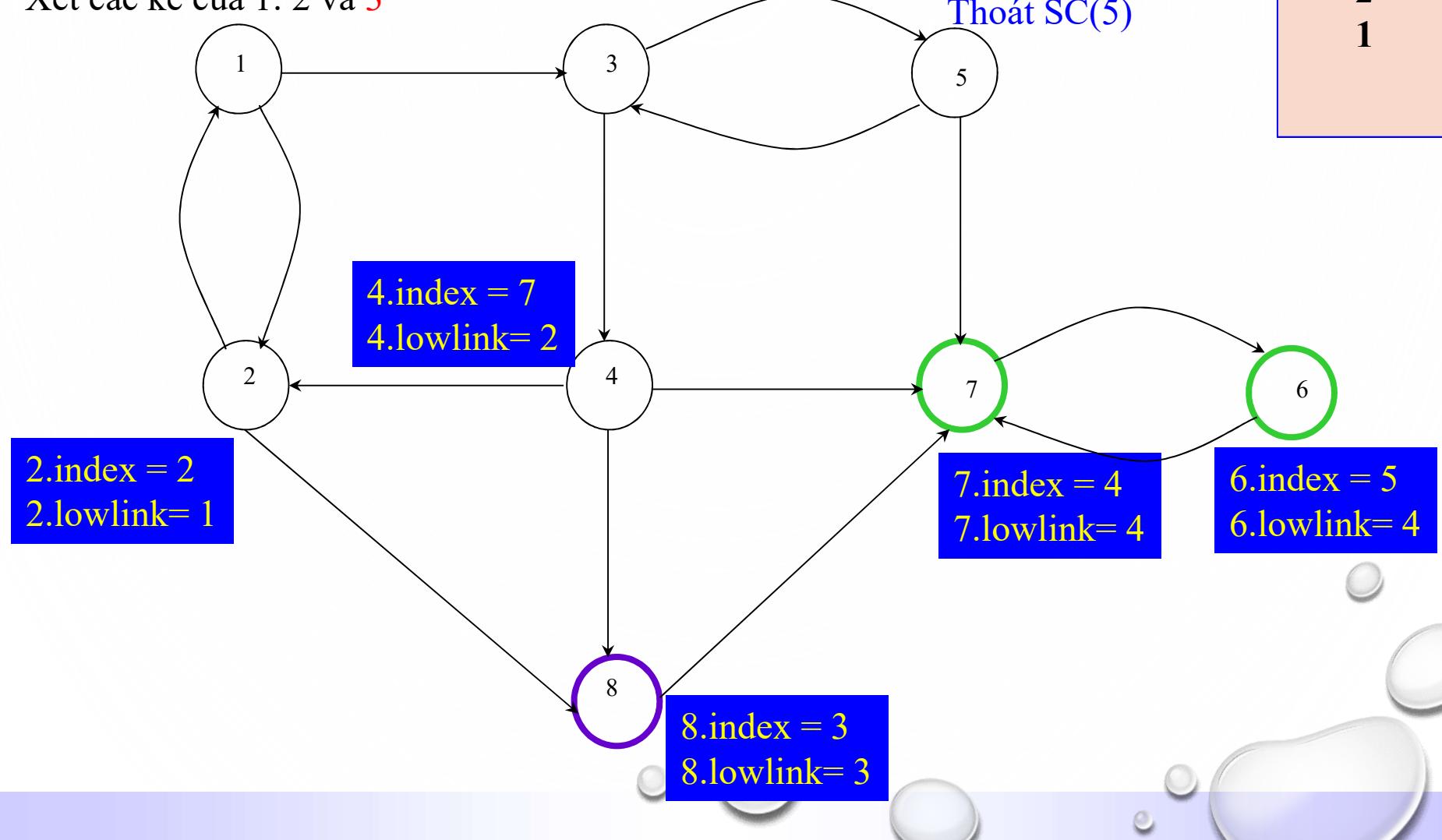
5.index = 8

5.lowlink= 6

Vì 5.lowlink<5.index  
nên 5 không là gốc

Thoát SC(5)

5  
4  
3  
2  
1



SC(1):

1.index = 1

1.lowlink= 1

Xét các kè của 1: 2 và 3

SC(3):

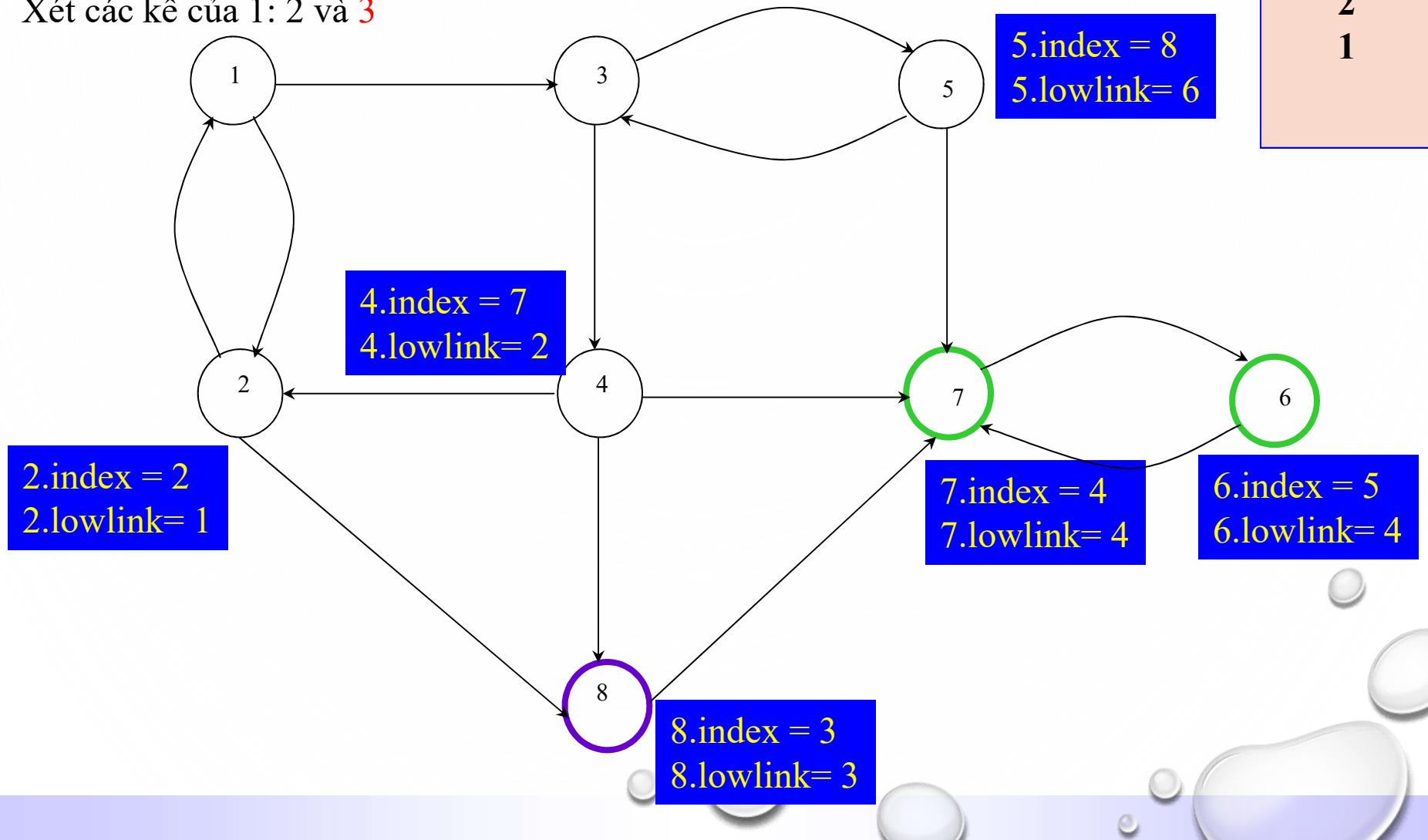
3.index = 6

3.lowlink= 2

Vì 3.lowlink<3.index nên 3 không là gốc

Thoát SC(3)

5  
4  
3  
2  
1



SC(1):

1.index = 1

1.lowlink= 1

Vì 1.lowlink=1.index nên 1 là gốc

-> lấy 5,4,3,2,1 khỏi stack

Thoát khỏi SC(1)

