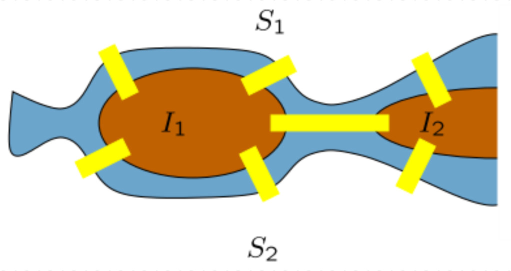
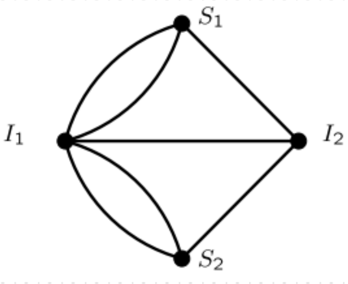
LÝ THUYẾT ĐỒ THỊ
Tính Liên thông

Phạm Nguyên Hoàng
BM. Khoa học máy tính, CNTT
pnhoang@cit.ctu.edu.vn





Cần Thơ, 2019

Tính liên thông của đồ thị

- **Đường đi** (walk): đường đi chiều dài k đi từ đỉnh $u = v_0$ đến đỉnh $v_k = v$ là danh sách các đỉnh và cung xen kẽ nhau:
  - $v_0, e_1, v_1, e_2, v_2, e_3, \ldots, e_k, v_k$
  - Cung e_i có đỉnh đầu là v_{i-1} và đỉnh cuối v_i

- **Đường đi đơn cung** (trail): là đường đi các cung đều khác nhau.
- **Đường đi đơn đỉnh** (path): là đường đi các đỉnh đều khác nhau.
- Đường đi đơn cung (trail) có đỉnh đầu và đỉnh cuối trùng nhau gọi là một **đường vòng** (circuit)
- Một đường vòng không có đỉnh nào lặp lại gọi là **chu trình** (cycle).

- Chiều dài của walk, trail, path, circuit hay cycle là số cung của nó.

Tính liên thông của đồ thị

- Định nghĩa:
  - Đồ thị vô hướng G được gọi là liên thông nếu và chỉ nếu với mọi cặp đỉnh u, v $\in$ V luôn tồn tại đường đi (walk) từ u → v. Ngược lại, G được gọi là không liên thông.
  - Đỉnh u được gọi là liên thông với đỉnh v ⇔ tồn tại đường đi từ u → v. Quan hệ liên thông trên G là tập các cặp có thứ tự (u, v) sao cho u liên thông với v.

Tính liên thông của đồ thị

- Định lý:
  - Quan hệ liên thông là một quan hệ tương đương
  - C/M: xem như bài tập

- Bổ đề:
  - Mỗi đường đi (walk) đi từ u đến v luôn chứa 1 một đường đi đơn đỉnh (path) đi từ u đến v.
  - C/M: xem như bài tập

Bộ phận liên thông

- Các bộ phận liên thông của đồ thị G là tập các đồ thị con liên thông lớn nhất của G (là các lớp tương đương của quan hệ liên thông).

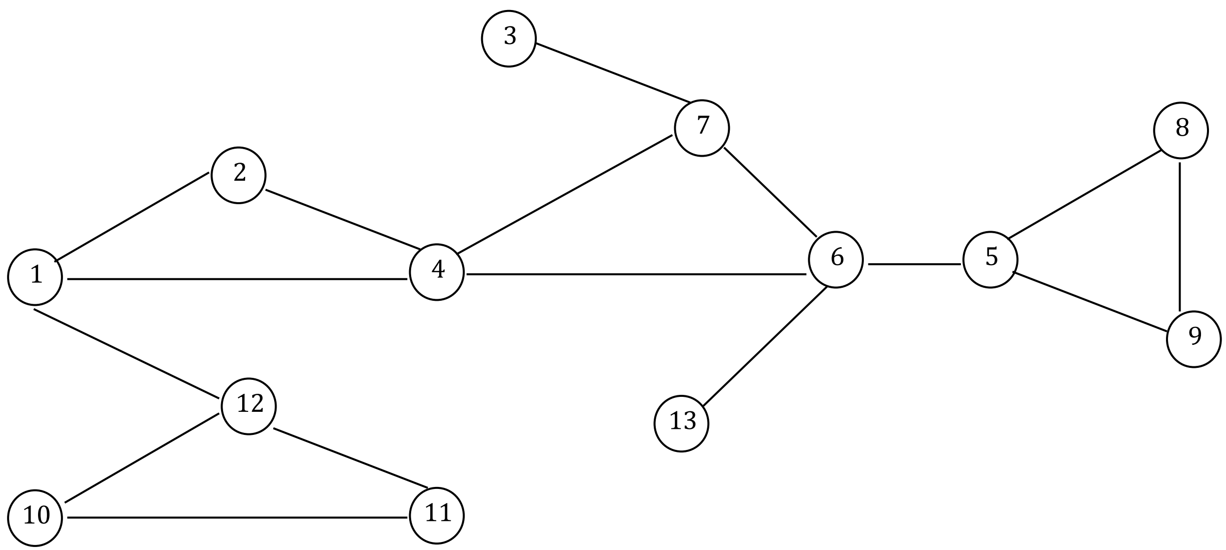
- Đỉnh cô lập (có bậc bằng 0) cũng là một bộ phân liên thông chỉ gồm chính nó và được gọi là bộ phận liên thông tầm thường (trivial connected component).

Khoảng cách giữa 2 đỉnh

- **Định nghĩa:** Trên một đồ thị vô hướng, khoảng cách từ đỉnh u đến đỉnh v, ký hiệu d(u,v) được định nghĩa bằng:
  - 0, nếu u $\equiv$ v
  - Chiều dài đường đi ngắn nhất từ u đến v, nếu tồn tại đường đi từ u đến v.
  - ∞, nếu không có đường đi từ u đến v.
- **Đường kính** của đồ thị vô hướng là khoảng cách lớn nhất giữa 2 đỉnh trên đồ thị.
- Bài tập:
  - Chứng minh bất đẳng thức tam giác:
    - $d(u, v) \leq d(u, x) + d(x, v)$

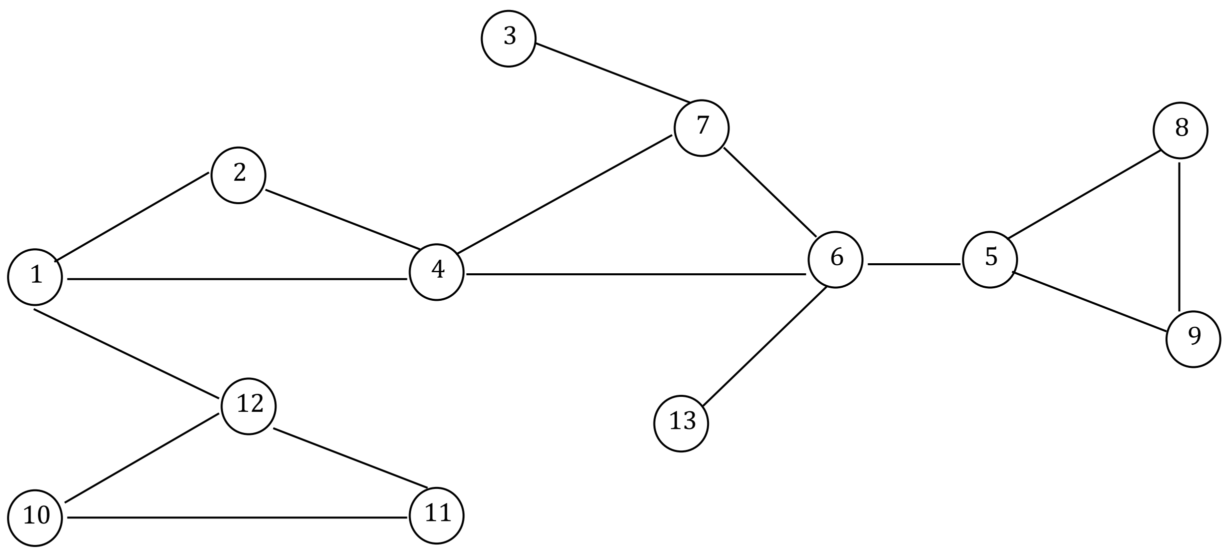
Giải thuật kiểm tra tính liên thông của đồ thị – Thực hành (1/2)

- Áp dụng giải thuật duyệt đồ thị để đánh số/đánh dấu (gán nhãn) các đỉnh
- Nếu sau khi duyệt tất cả các đỉnh đều có nhãn => liên thông, ngược lại không liên thông.



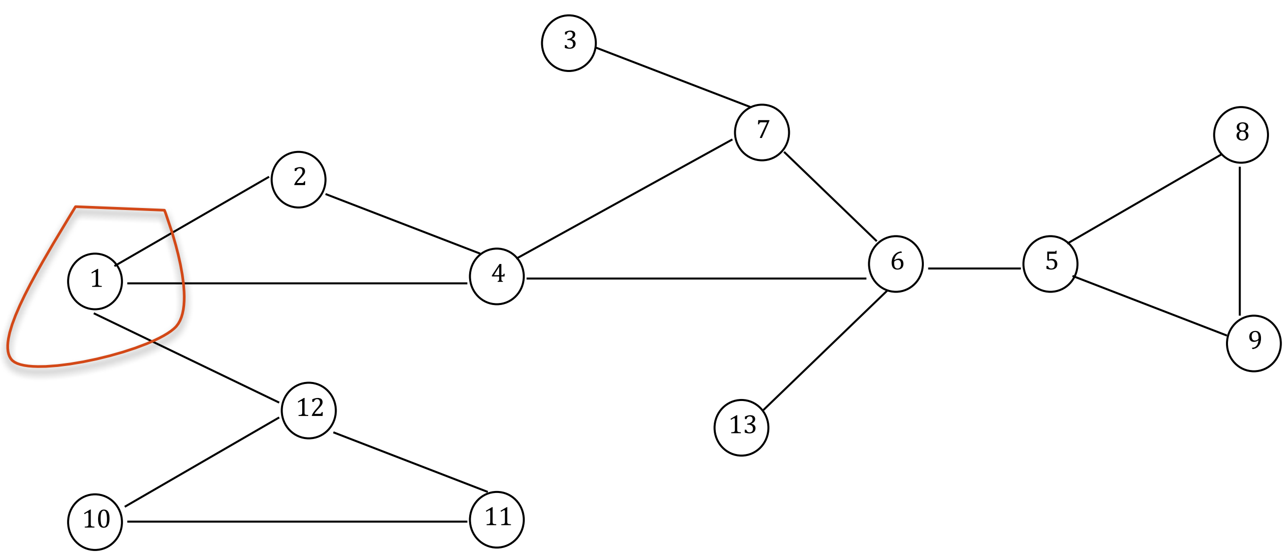
Duyệt đồ thị (nhắc lại)

- Lần lượt xem xét từng đỉnh của đồ thị (mỗi đỉnh chỉ xét lần)
- Bắt đầu từ 1 đỉnh bất kỳ, xét đỉnh các đỉnh kề của nó, xét các đỉnh kề của các đỉnh kề, …



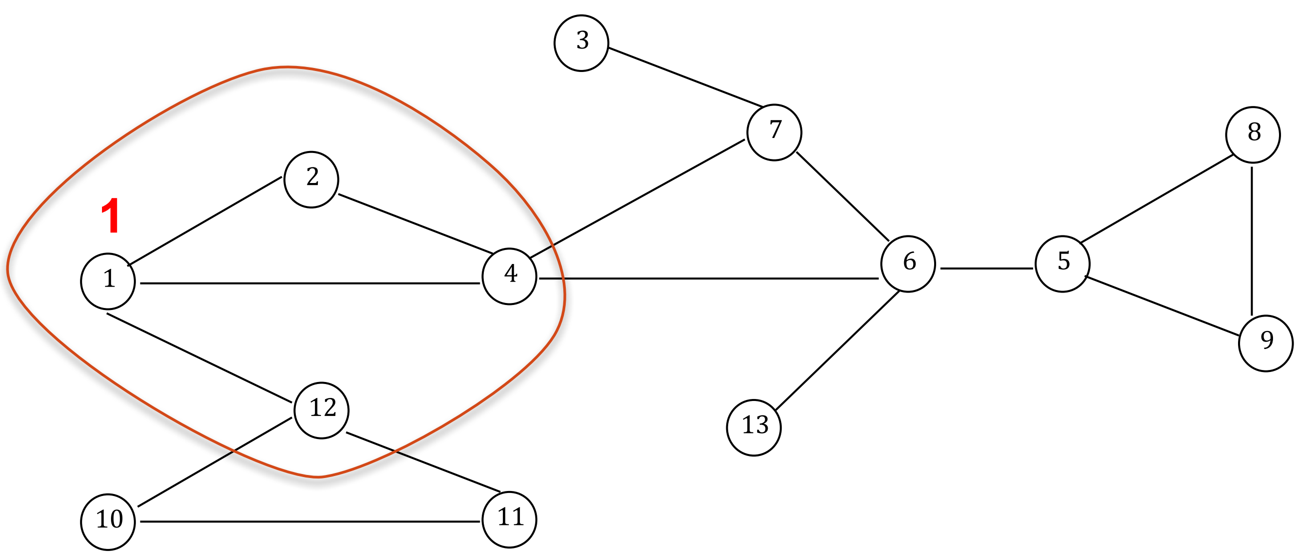
Duyệt đồ thị (nhắc lại)

- Lần lượt xem xét từng đỉnh của đồ thị (mỗi đỉnh chỉ xét lần)
- Bắt đầu từ 1 đỉnh bất kỳ, xét đỉnh các đỉnh kề của nó, xét các đỉnh kề của các đỉnh kề, ...



Duyệt đồ thị (nhắc lại)

- Lần lượt xem xét từng đỉnh của đồ thị (mỗi đỉnh chỉ xét lần)
- Bắt đầu từ 1 đỉnh bất kỳ, xét đỉnh các đỉnh kề của nó, xét các đỉnh kề của các đỉnh kề, …



Giải thuật kiểm tra tính liên thông của đồ thị – Thực hành (2/2)

- Giải thuật Trémeaux (1882) tìm bộ phận liên thông chứa một đỉnh cho trước:
  - Áp dụng giải thuật duyệt đồ thị theo chiều sâu để đánh số các đỉnh (giải thuật đệ quy).
  - Khởi tạo tất cả các đỉnh có num[x] = -1 và k = 1;

```
void Traversal(Graph* G, int x) {
   if (num[x] > 0) return; //x đã duyệt rồi
   num[x] = k; k++;
   for (các đỉnh kề y của x)
        Traversal(G, y);
}
```

  - Ví dụ: gọi Traversal(1); Khi giải thuật kết thúc, các đỉnh được đánh số (num[x] > 0) chính là bộ phận liên thông chứa đỉnh 1.

Giải thuật tìm tất cả các bộ phận liên thông (thực hành)

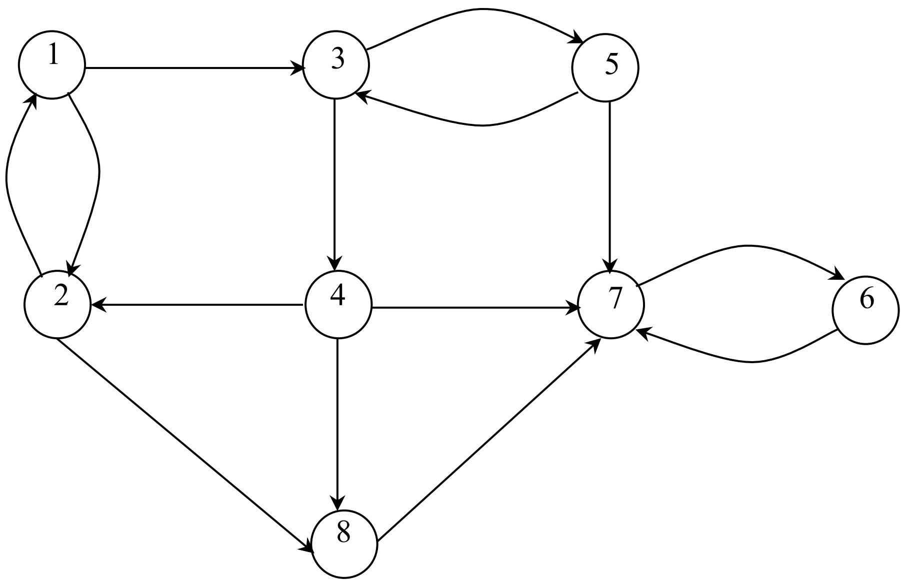
- Sử dụng giải thuật duyệt đồ thị (chiều rộng hoặc chiều sâu đều được)
- Khởi tạo tất cả các đỉnh có num[x] = -1 (chưa được đánh số)
- Xét qua các đỉnh, nếu x chưa được đánh dấu => gọi Traversal(x) để duyệt nó.

```
for (x = 1; x <= n; x++)
    if (num[x] < 0) {
        Traversal(G, x);
        //Tìm được 1 bộ phận liên thông chứa x.
    }
```

- Mỗi lần duyệt xong 1 đỉnh x, ta sẽ tìm được 1 bộ phận liên thông chứa x.

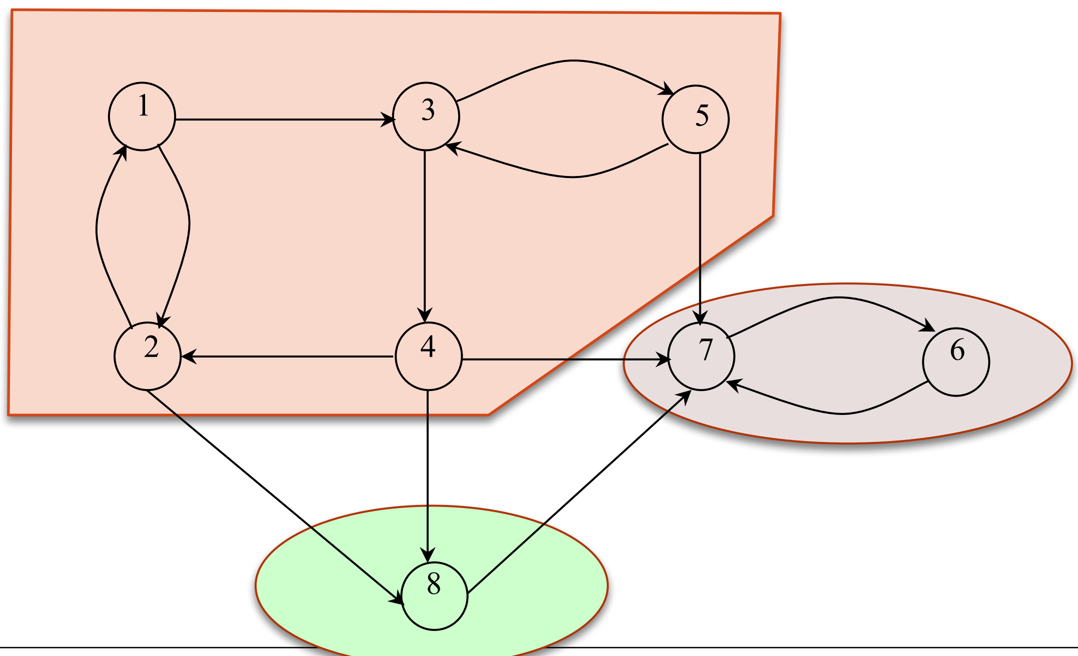
Tính liên thông của đồ thị có hướng

- Cho đồ thị có hướng G = <V, E>
  - G được gọi là **liên thông yếu** ⇔ đồ thị vô hướng nền của nó liên thông (xem tính liên thông của đồ thị vô hướng)
  - G được gọi là **liên thông mạnh** ⇔ giữa hai đỉnh x, y bất kỳ, luôn có đường đi từ x đến y.



Tính liên thông của đồ thị có hướng

- Bộ phận liên thông mạnh
  - Đồ thị con liên thông mạnh: có đường đi giữa hai đỉnh bất kỳ.
  - Đồ thị có hướng không liên thông mạnh bao gồm nhiều bộ phân liên thông mạnh



Tìm các bộ phận liên thông mạnh

- Nhận xét:
  - Các đỉnh trong một chu trình liên thông với nhau

- Giải thuật tìm các bộ phận liên thông mạnh
  - Tìm các chu trình (lớn nhất có thể) của một đồ thị

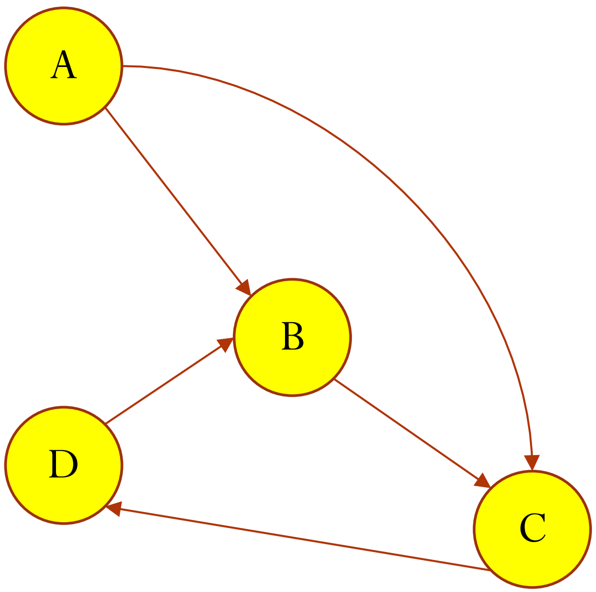
Tìm các bộ phận liên thông mạnh

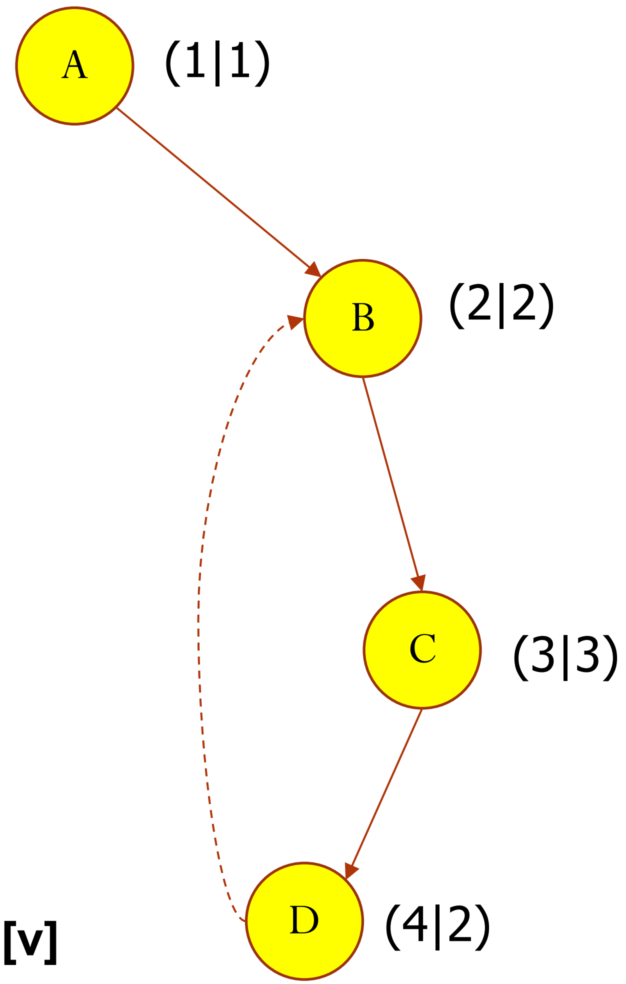
- Giải thuật Tarjan (1972)
  - Áp dụng **duyệt theo chiều sâu** (đệ quy hoặc không đệ quy) để đánh số các đỉnh.
  - Để tìm được chu trình, với mỗi đỉnh v, ngoài num[v], ta lưu thêm min_num[v] (là chỉ số nhỏ nhất trong các đỉnh có thể đi đến được từ v). Trong quá trình duyệt, min_num[v] sẽ được cập nhật.
  - Khi duyệt xong 1 đỉnh, nếu num[v] = min_num[v] thì v là đỉnh bắt đầu (đỉnh gốc/đỉnh khớp) của bộ phận liên thông mạnh.

Tìm các bộ phận liên thông mạnh

- Các biến hỗ trợ:
  - **S**: stack lưu các đỉnh chưa tìm được BPLT mạnh
  - **on_stack[v]**: kiểm tra v còn trên stack không
  - **num[v]**: chỉ số của đỉnh v trong quá trình duyệt
  - **min_num[v]**: chỉ số nhỏ nhất trong các chỉ số của các đỉnh trong stack S mà v đi đến nó được.
  - **idx**: chỉ số dùng để gán cho num của các đỉnh (tăng dần)

Tìm các bộ phận liên thông mạnh





num[v] và **min_num[v]**

Tìm các bộ phận liên thông mạnh

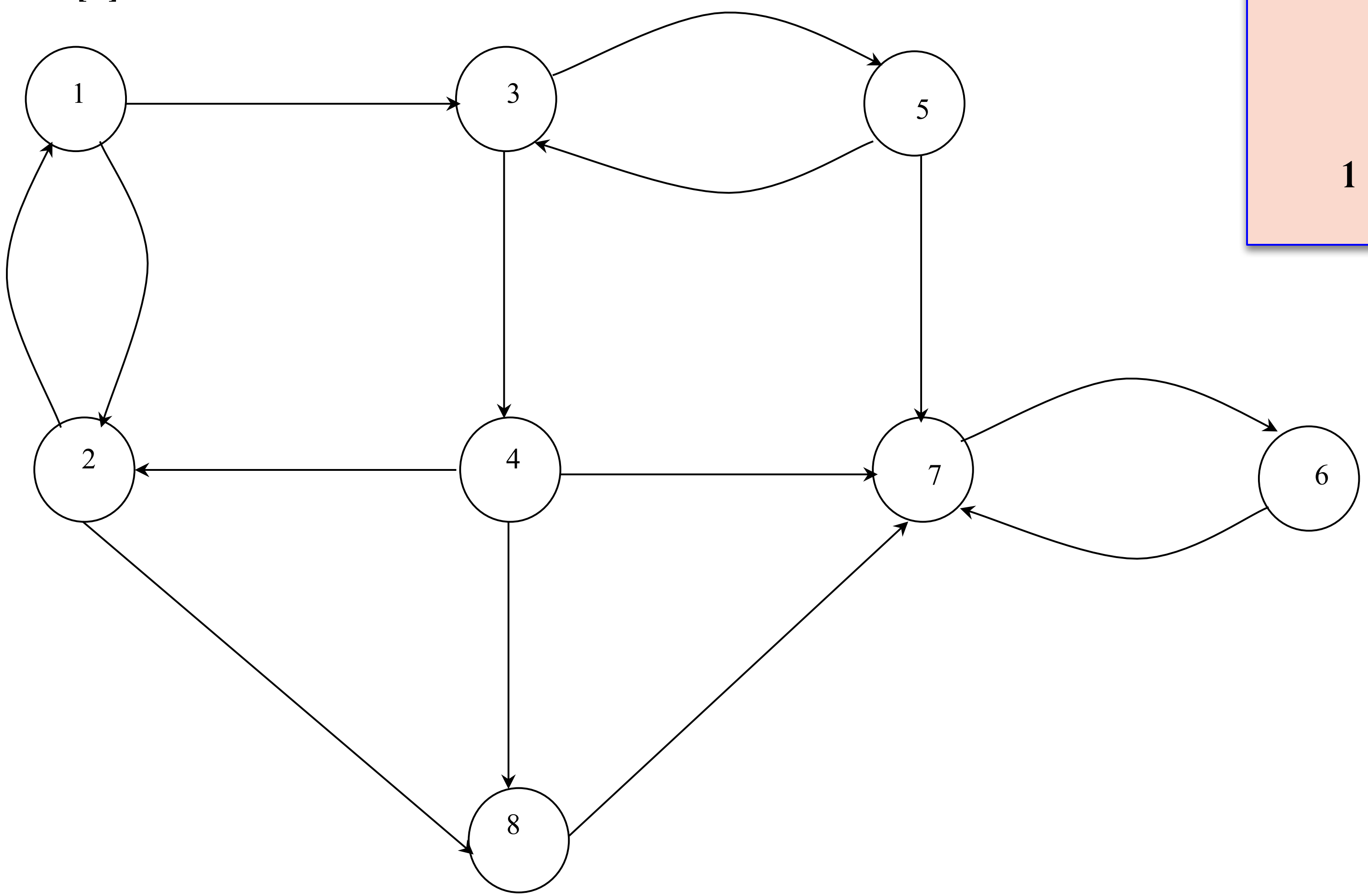
```
void strong_connect(Graph* G, int x) {
        num[x] = min_num[x] = idx; idx++;
        push(&S, x);                    /* Đưa x vào stack */
        on_stack[x] = 1; /* x đang ở trên stack */
        /* Lấy các đỉnh kề và duyệt nó */
        List list = neighbors(G, x);
        for (j = 1; j <= list.size; j++) {
                int y = element_at(&list, j);
                if (num[y] == -1) {
                        strong_connect(G, y);
                        min_num [x] = min(min_num[x], min_num[y]);
                } else if (on_stack[y])
                        min_num[x] = min(min_num[x], num[y]);
        }
        /* Kiểm tra nếu num[x] == min_num[x] */
}
```

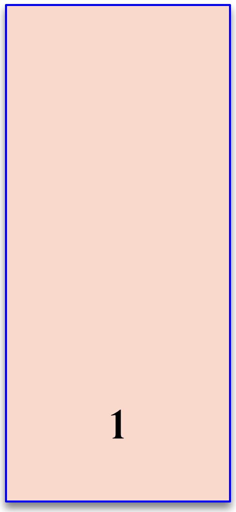
Tìm các bộ phận liên thông mạnh

```
void strong_connect(Graph* G, int x) {
        /* đánh dấu x */
        /* Lấy các đỉnh kề và duyệt nó */
        /* Kiểm tra nếu num[x] == min_num[x] */
        if (num[x] == min_num[x]) {
                /* Loại bỏ các đỉnh ra khỏi stack */
                int w;
                do {
                        w = top(&S); pop(&S);
                        on_stack[w] = 0;
                        /* làm gì đó trên w, vd: in ra màn hình
*/
                } while (w != x);
        }
}
```

num[1] = 1

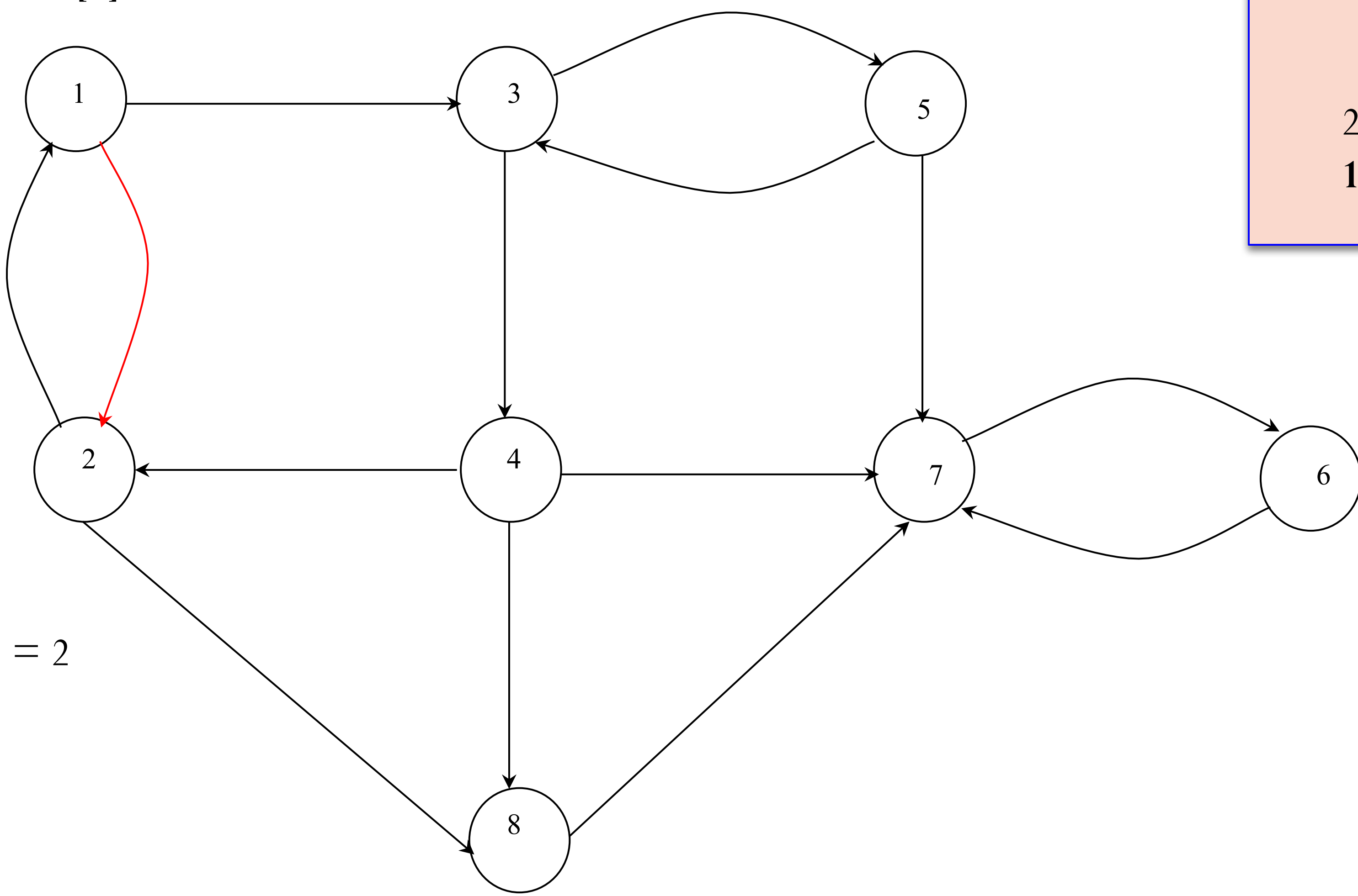
min_num[1] = 1





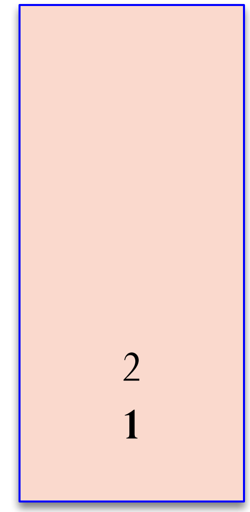
$num[1] = 1$

$min\_num[1] = 1$



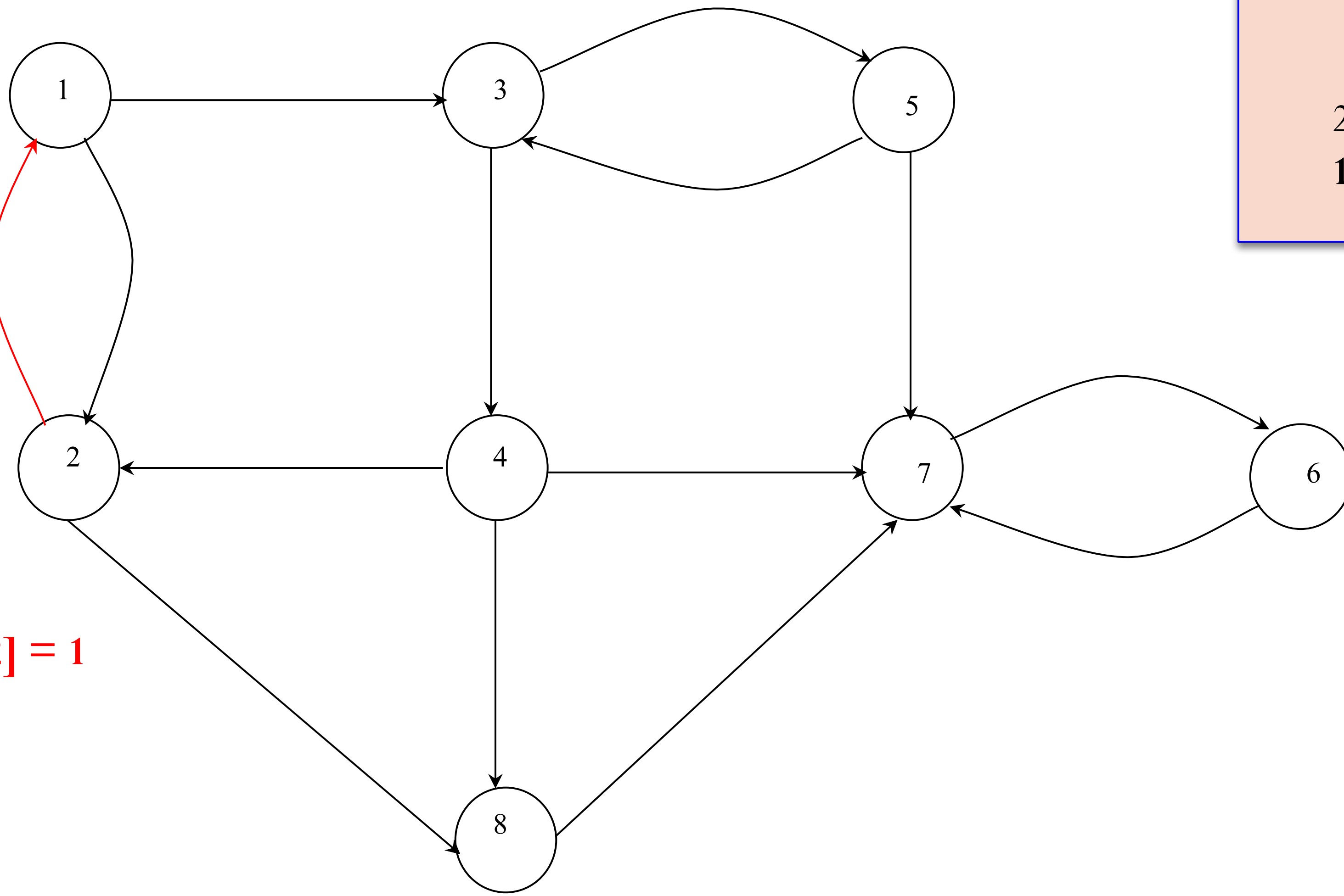
$num[2] = 2$

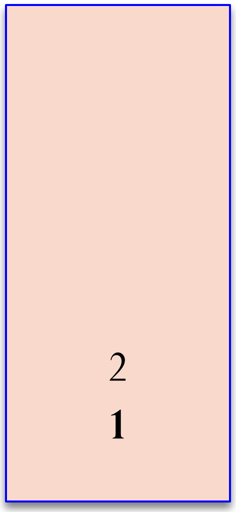
$min\_num[2] = 2$



num[1] = 1

min_num[1] = 1



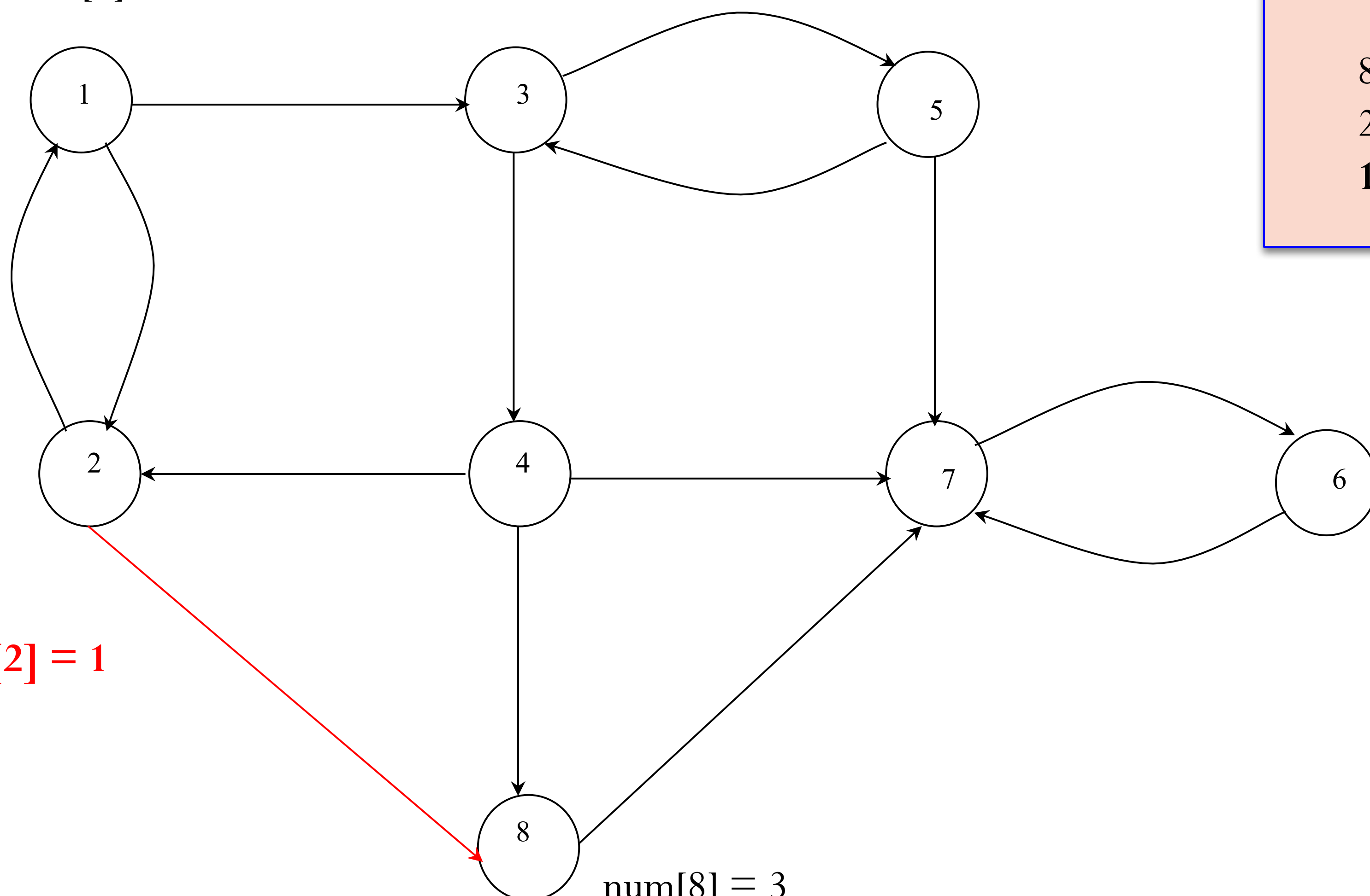


num[2] = 2

min_num[2] = 1

num[1] = 1

min_num[1] = 1



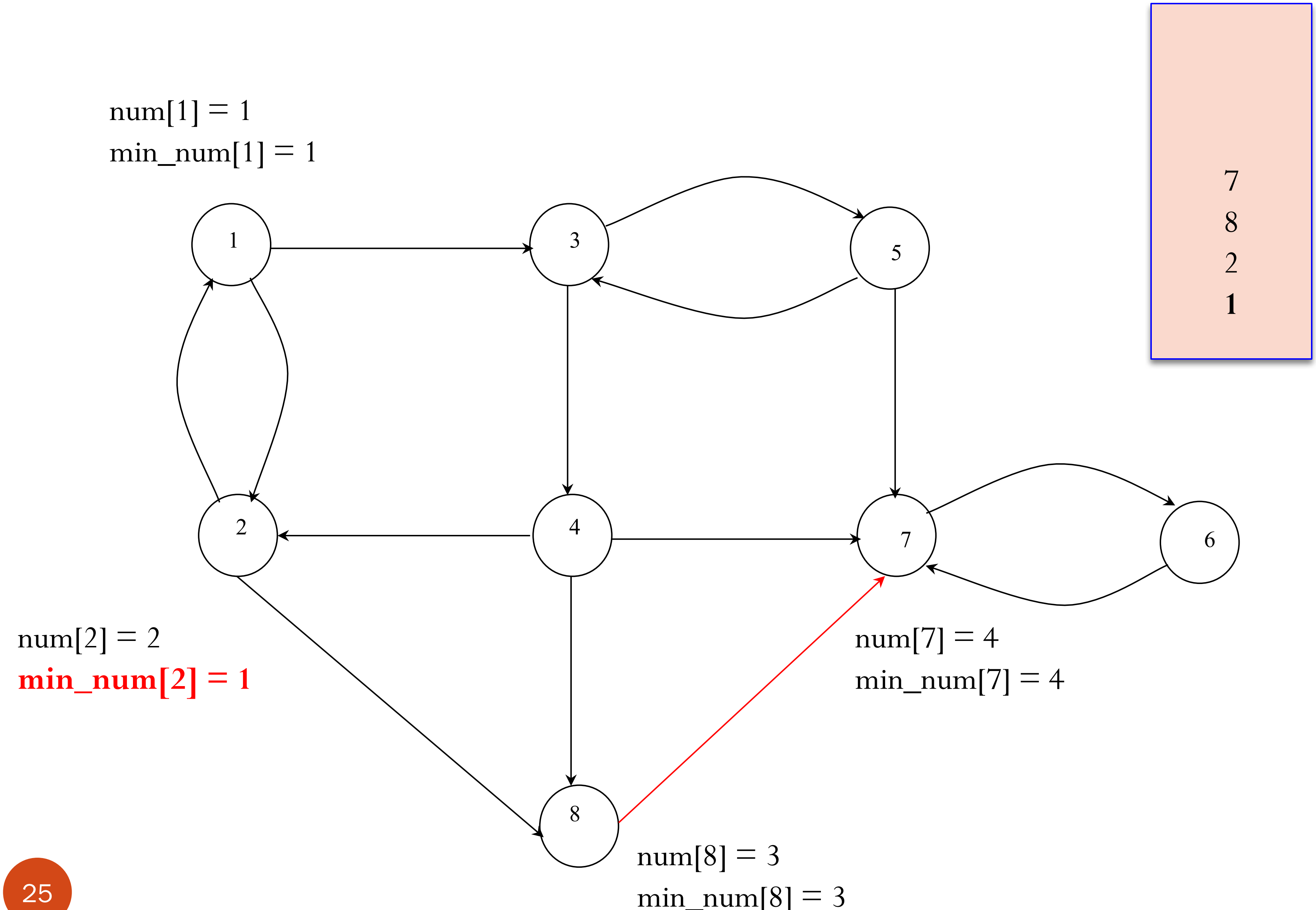
8
2
1

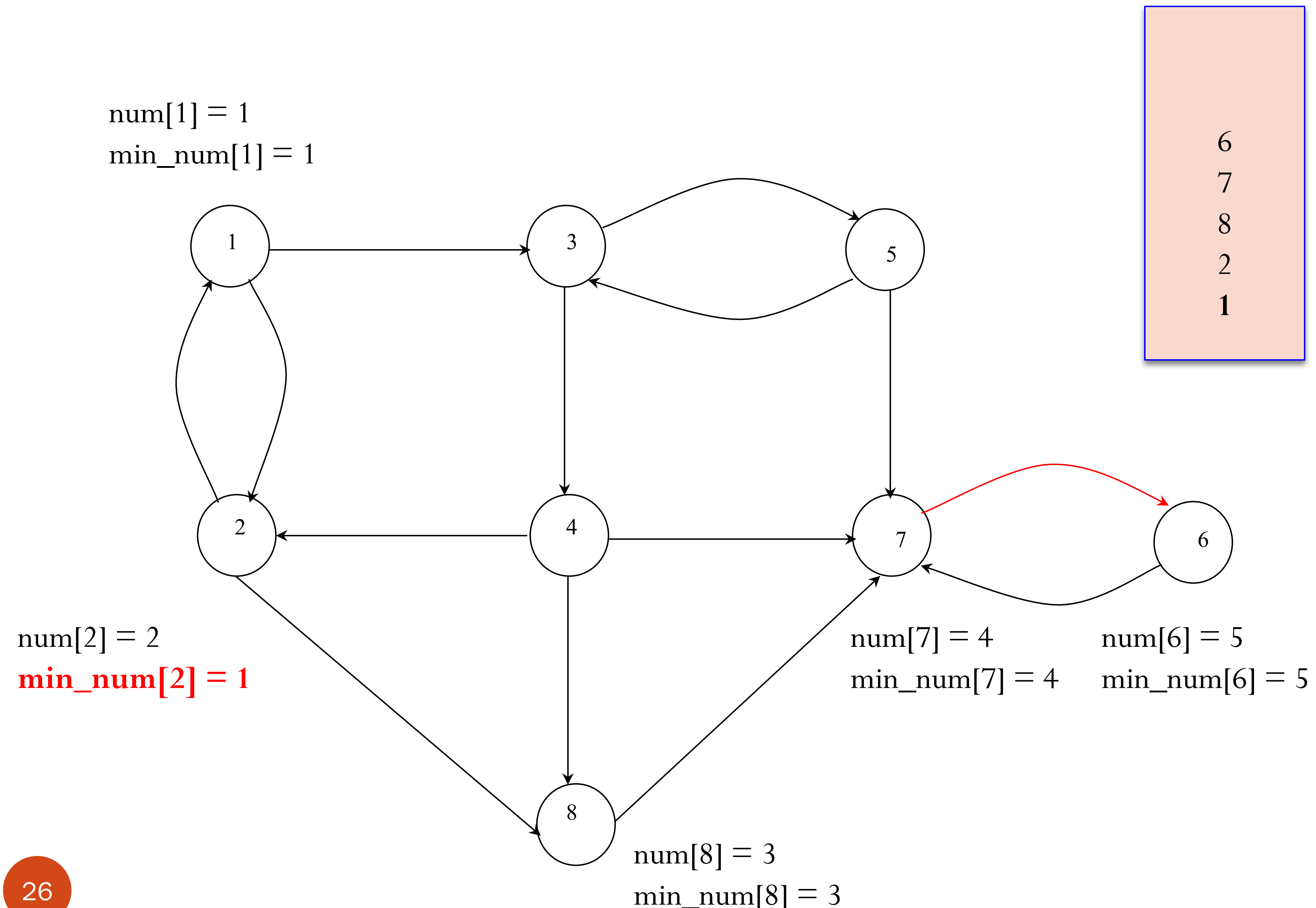
num[2] = 2

min_num[2] = 1

num[8] = 3

min_num[8] = 3

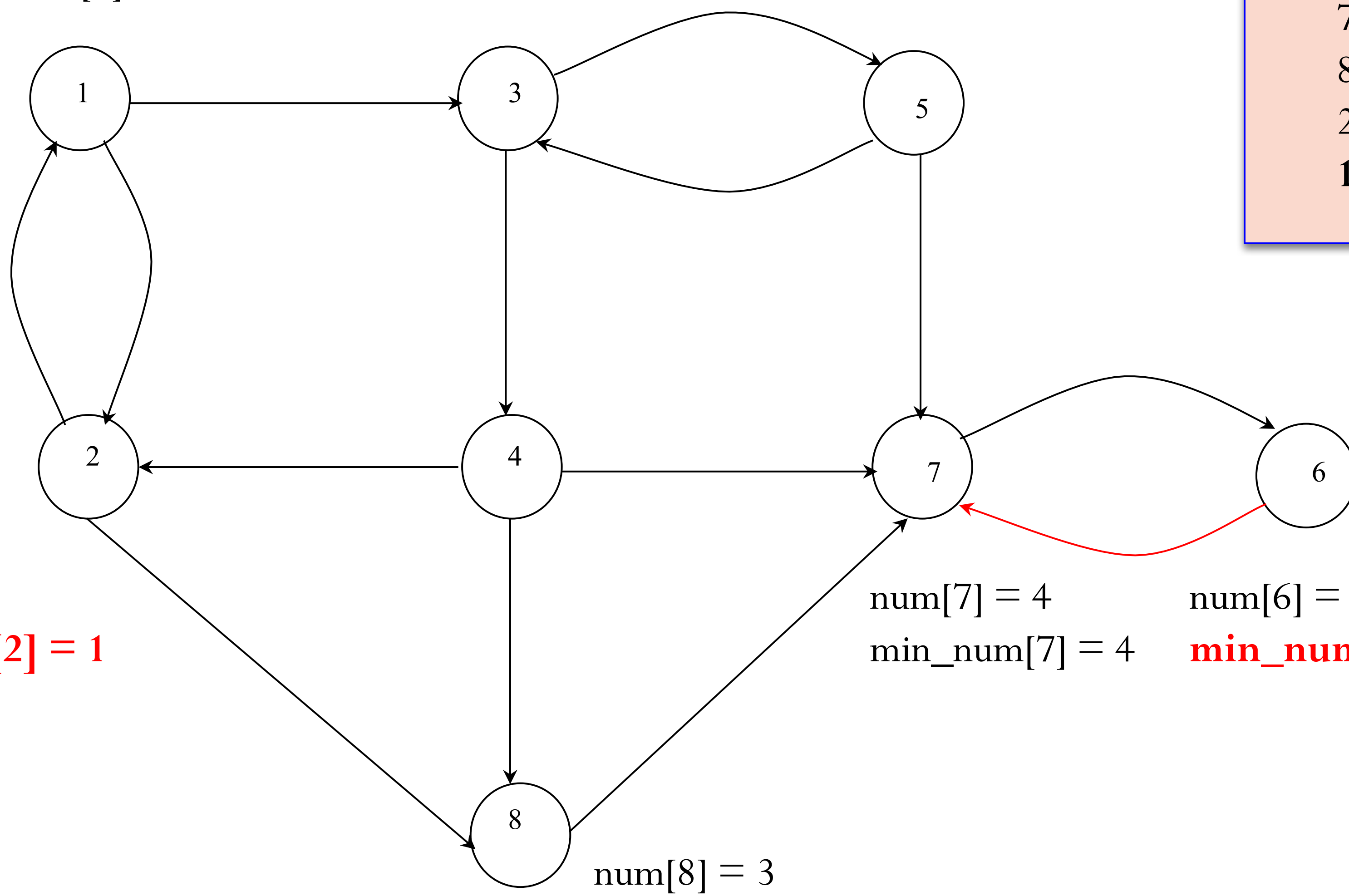

num[1] = 1
min_num[1] = 1
1
3
5
2
4
7
6
8
num[2] = 2
min_num[2] = 1
num[7] = 4
min_num[7] = 4
num[8] = 3
min_num[8] = 3
7
8
2
1



num[1] = 1
min_num[1] = 1
1
3
5
2
4
7
6
8
num[2] = 2
min_num[2] = 1
num[7] = 4
min_num[7] = 4
num[6] = 5
min_num[6] = 5
num[8] = 3
min_num[8] = 3
6
7
8
2
1

num[1] = 1
min_num[1] = 1

6
7
8
2
1



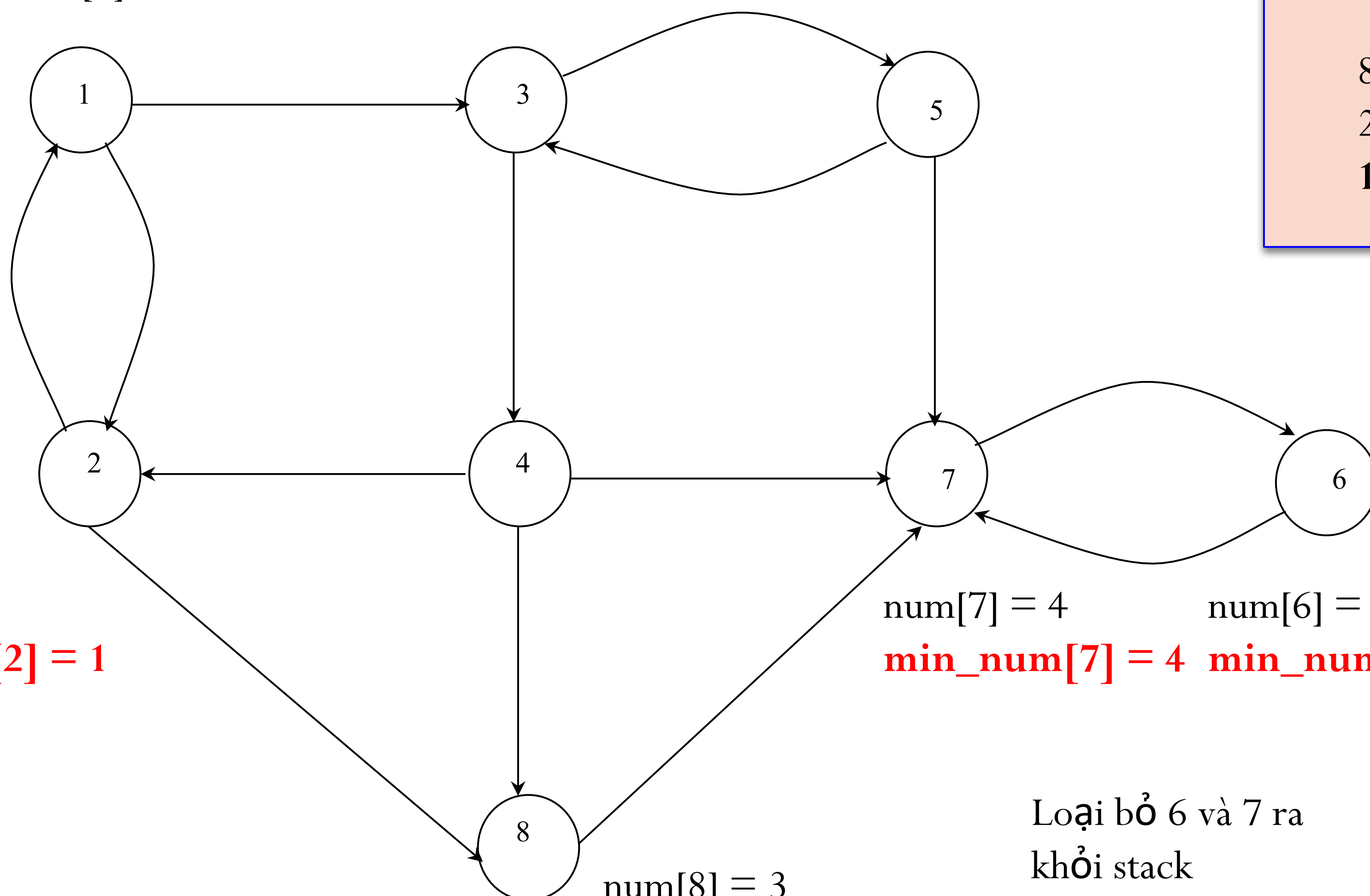
num[2] = 2
min_num[2] = 1

num[7] = 4
min_num[7] = 4

num[6] = 5
min_num[6] = 4

num[8] = 3
min_num[8] = 3

num[1] = 1
min_num[1] = 1



num[2] = 2
min_num[2] = 1

num[7] = 4
min_num[7] = 4

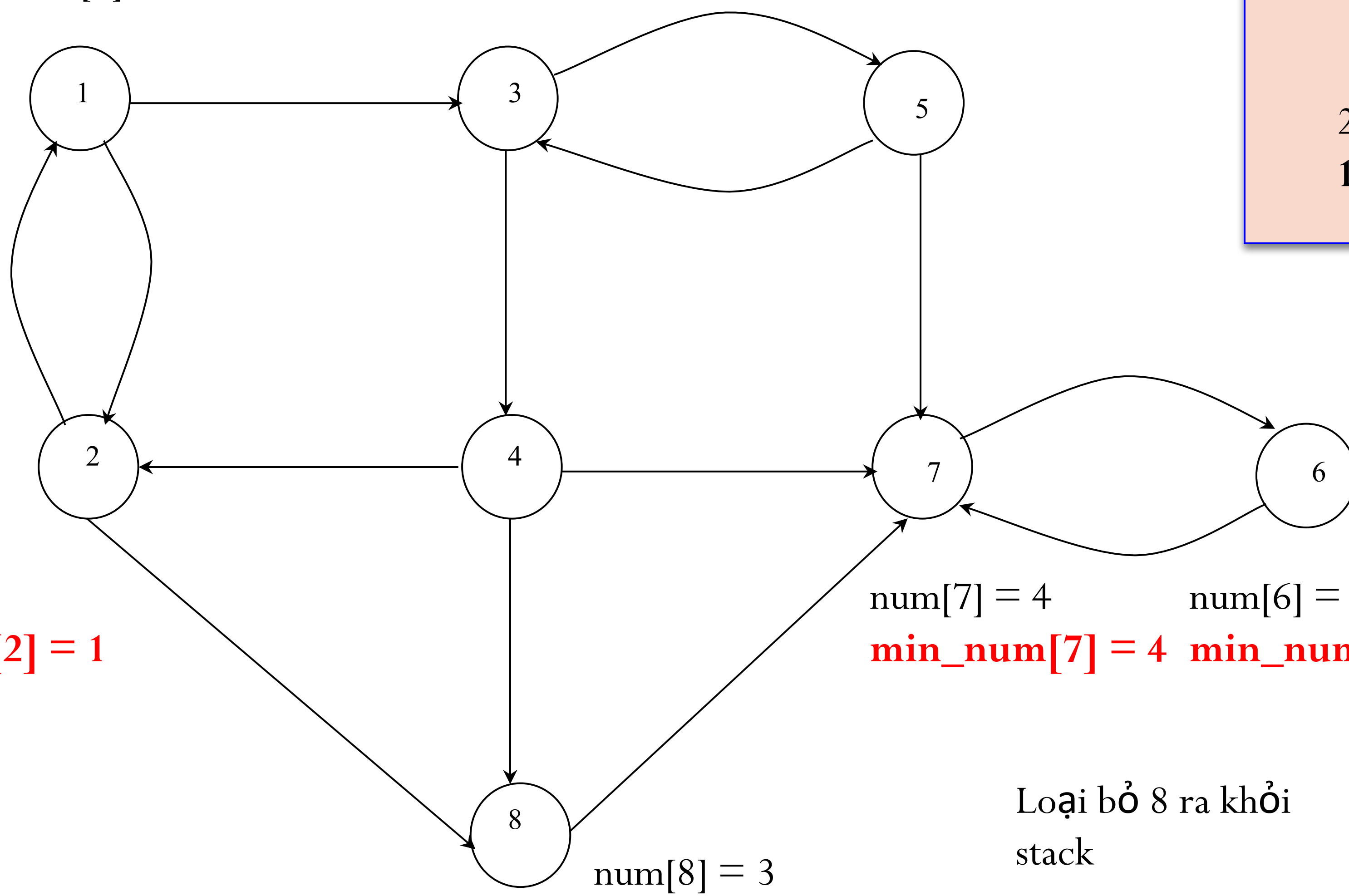
num[6] = 5
min_num[6] = 4

num[8] = 3
min_num[8] = 3

8
2
1

Loại bỏ 6 và 7 ra khỏi stack

num[1] = 1
min_num[1] = 1



2
1

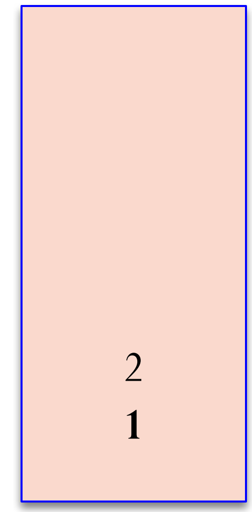
num[2] = 2
min_num[2] = 1

num[7] = 4
min_num[7] = 4

num[6] = 5
min_num[6] = 4

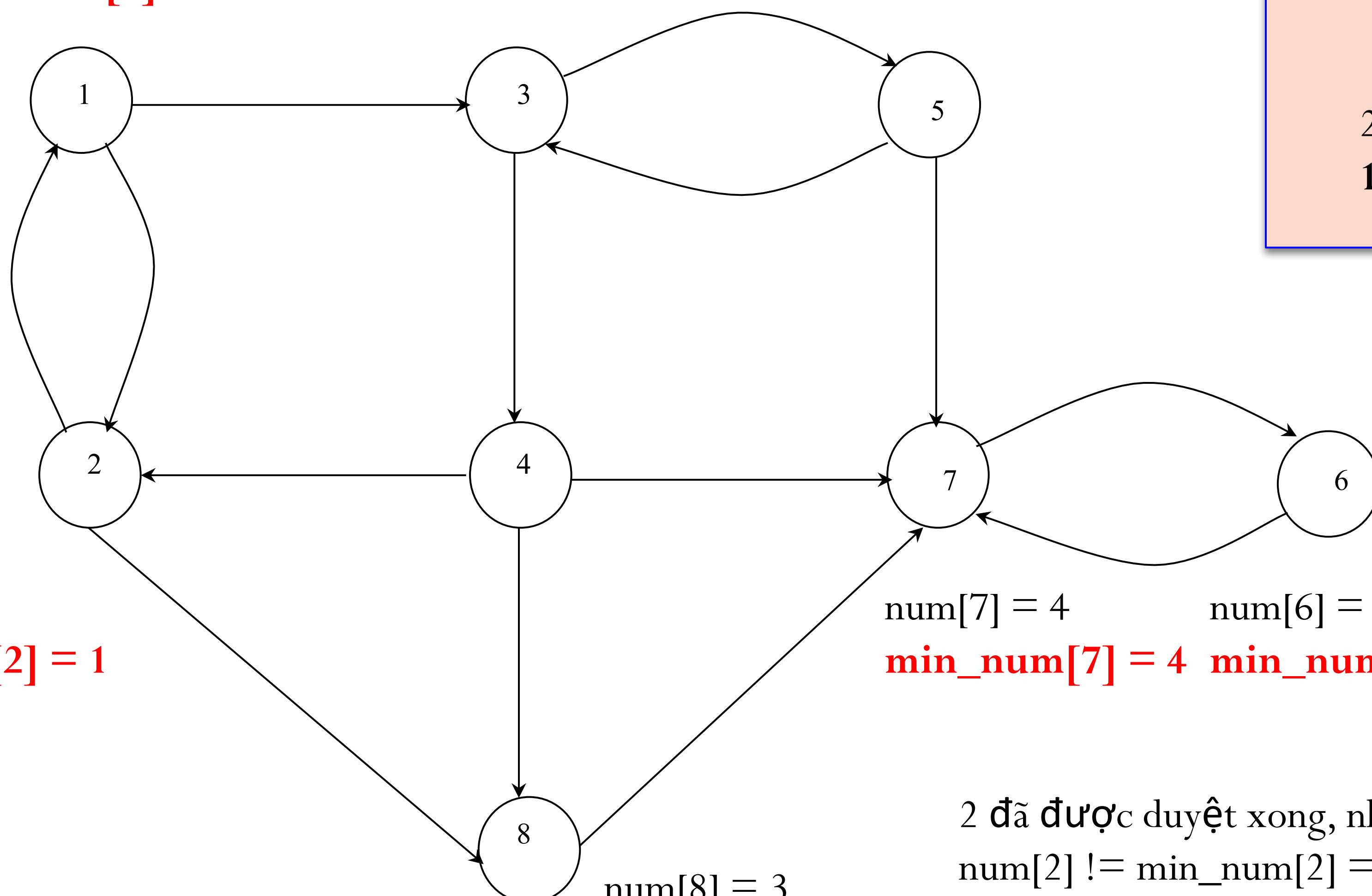
num[8] = 3
min_num[8] = 3

Loại bỏ 8 ra khỏi stack



num[1] = 1

min_num[1] = 1



num[2] = 2

min_num[2] = 1

num[7] = 4

min_num[7] = 4

num[6] = 5

min_num[6] = 4

num[8] = 3

min_num[8] = 3

2 đã được duyệt xong, nhưng num[2] != min_num[2] => giữ 2 lại trong stack

num[1] = 1

min_num[1] = 1

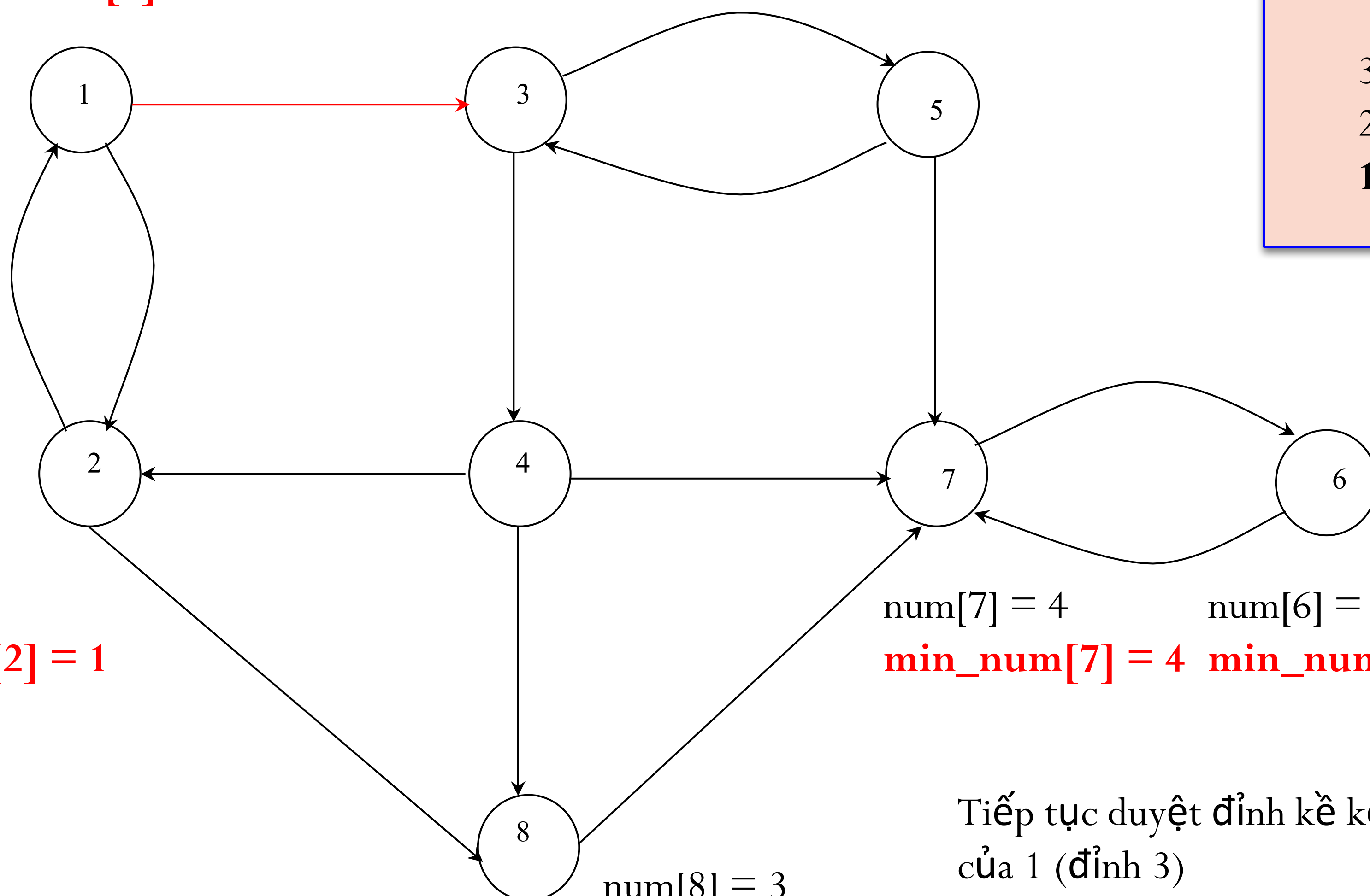
num[3] = 6

min_num[3] = 6

3

2

1



num[2] = 2

min_num[2] = 1

num[7] = 4

min_num[7] = 4

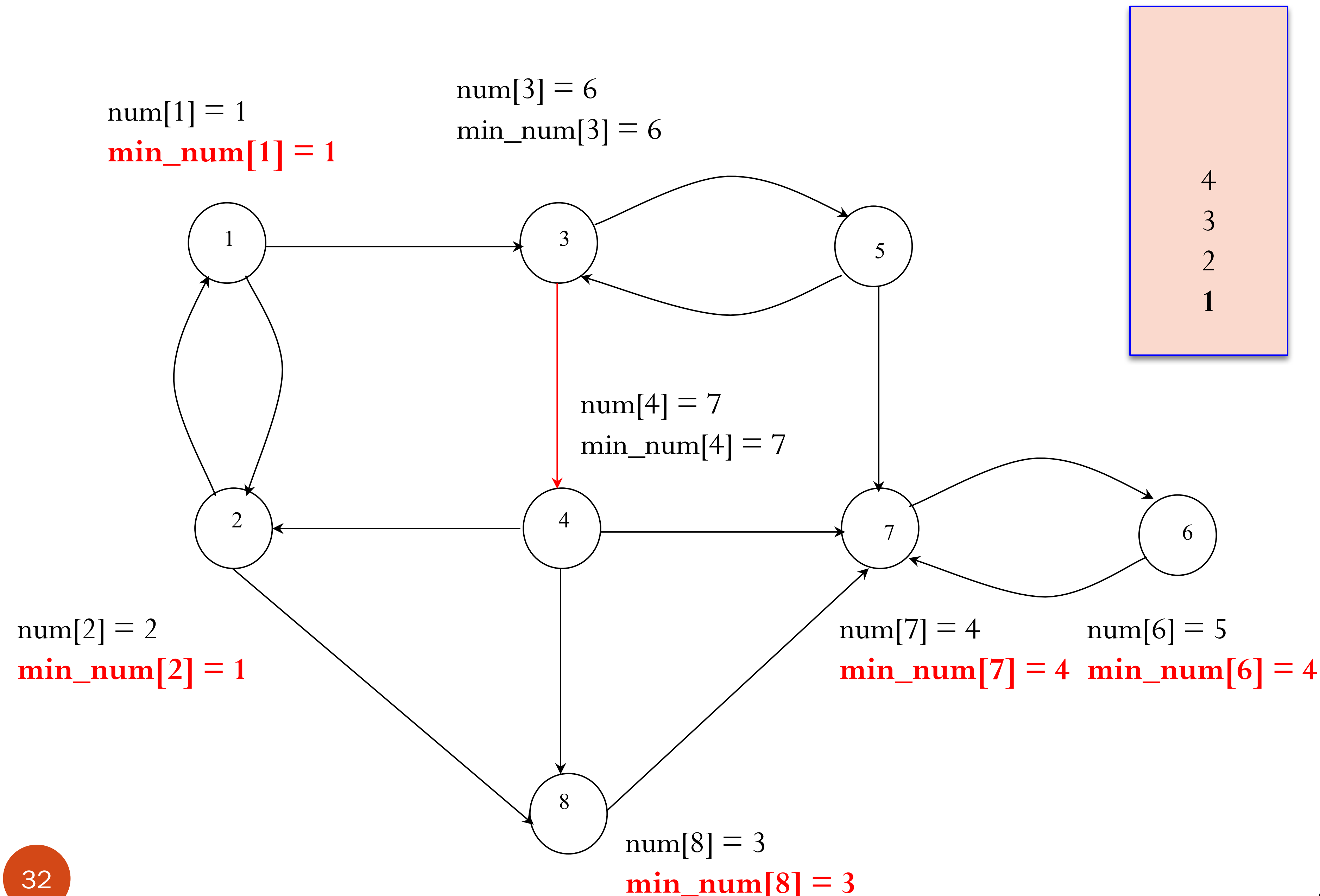
num[6] = 5

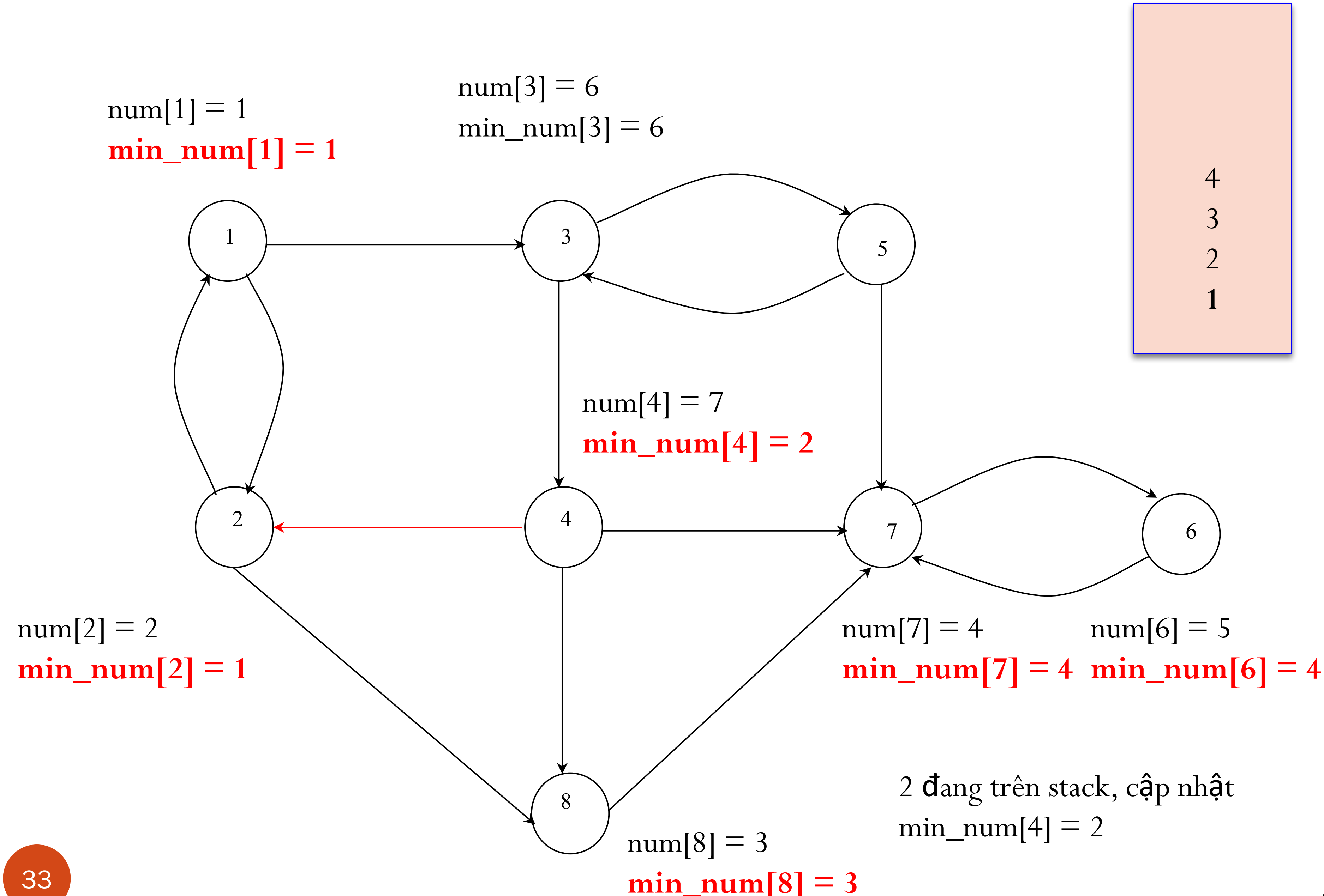
min_num[6] = 4

num[8] = 3

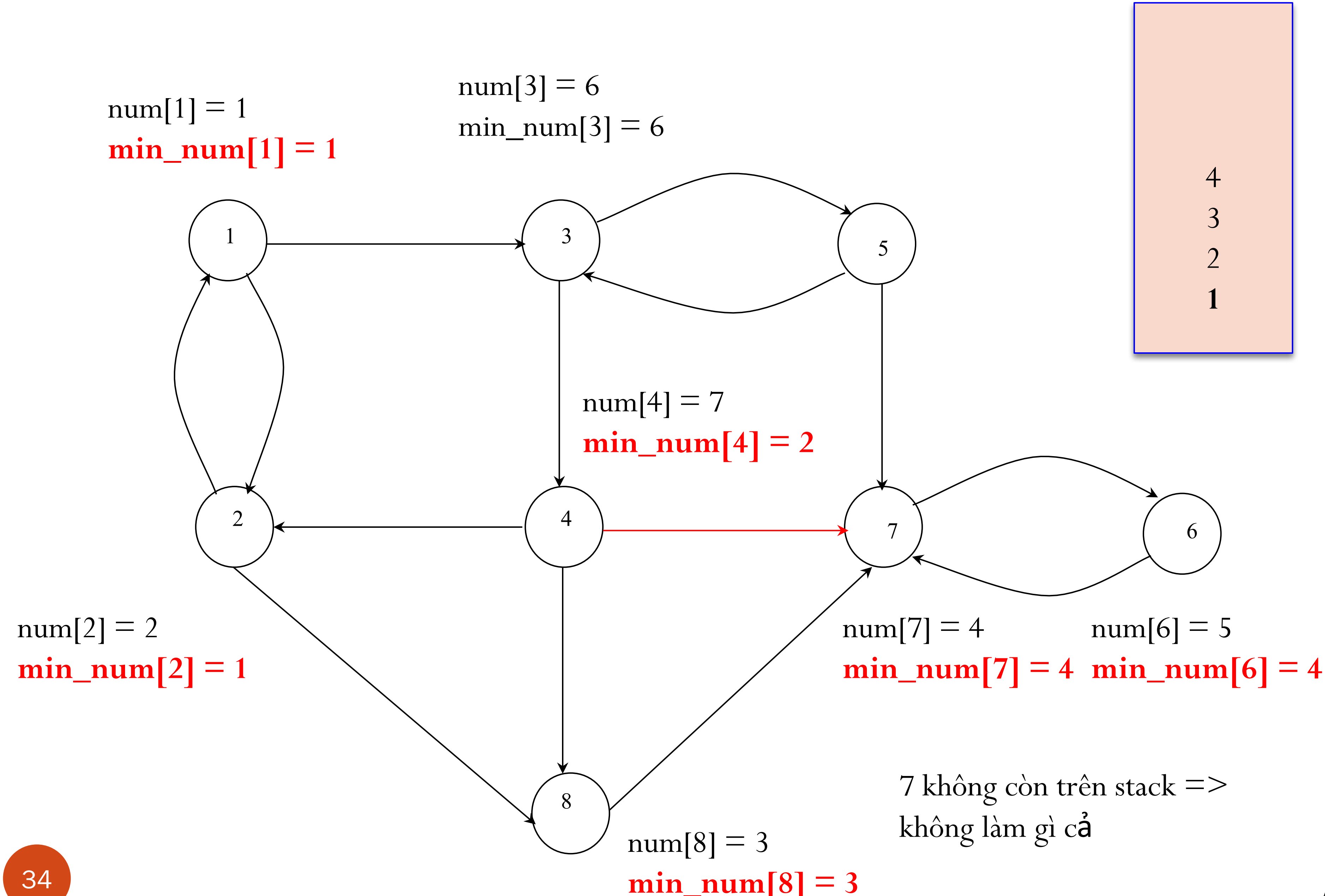
min_num[8] = 3

Tiếp tục duyệt đỉnh kề kế tiếp của 1 (đỉnh 3)

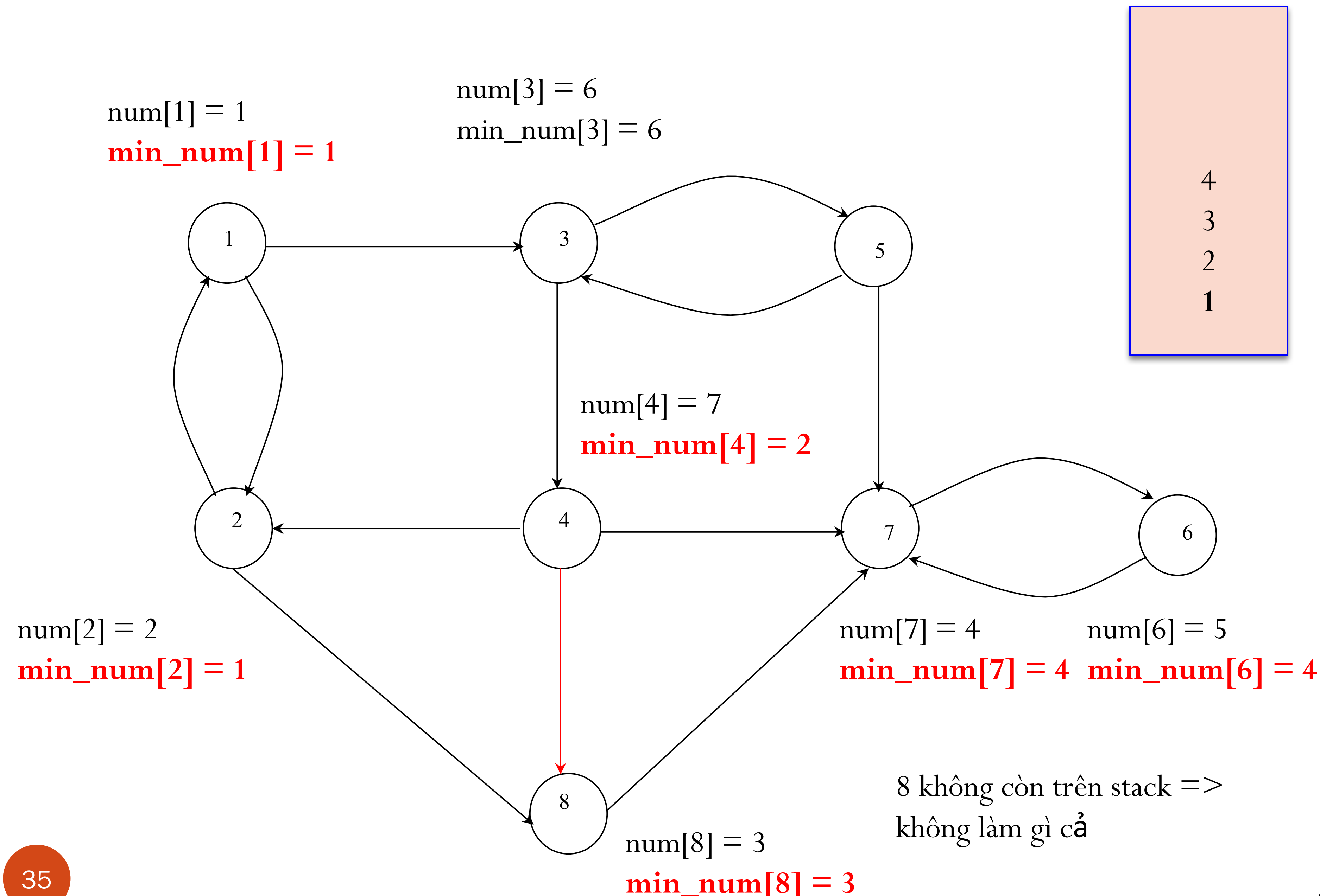

num[1] = 1
min_num[1] = 1
num[3] = 6
min_num[3] = 6
1
3
5
num[4] = 7
min_num[4] = 7
2
4
7
6
num[2] = 2
min_num[2] = 1
num[7] = 4
min_num[7] = 4
num[6] = 5
min_num[6] = 4
8
num[8] = 3
min_num[8] = 3
4
3
2
1



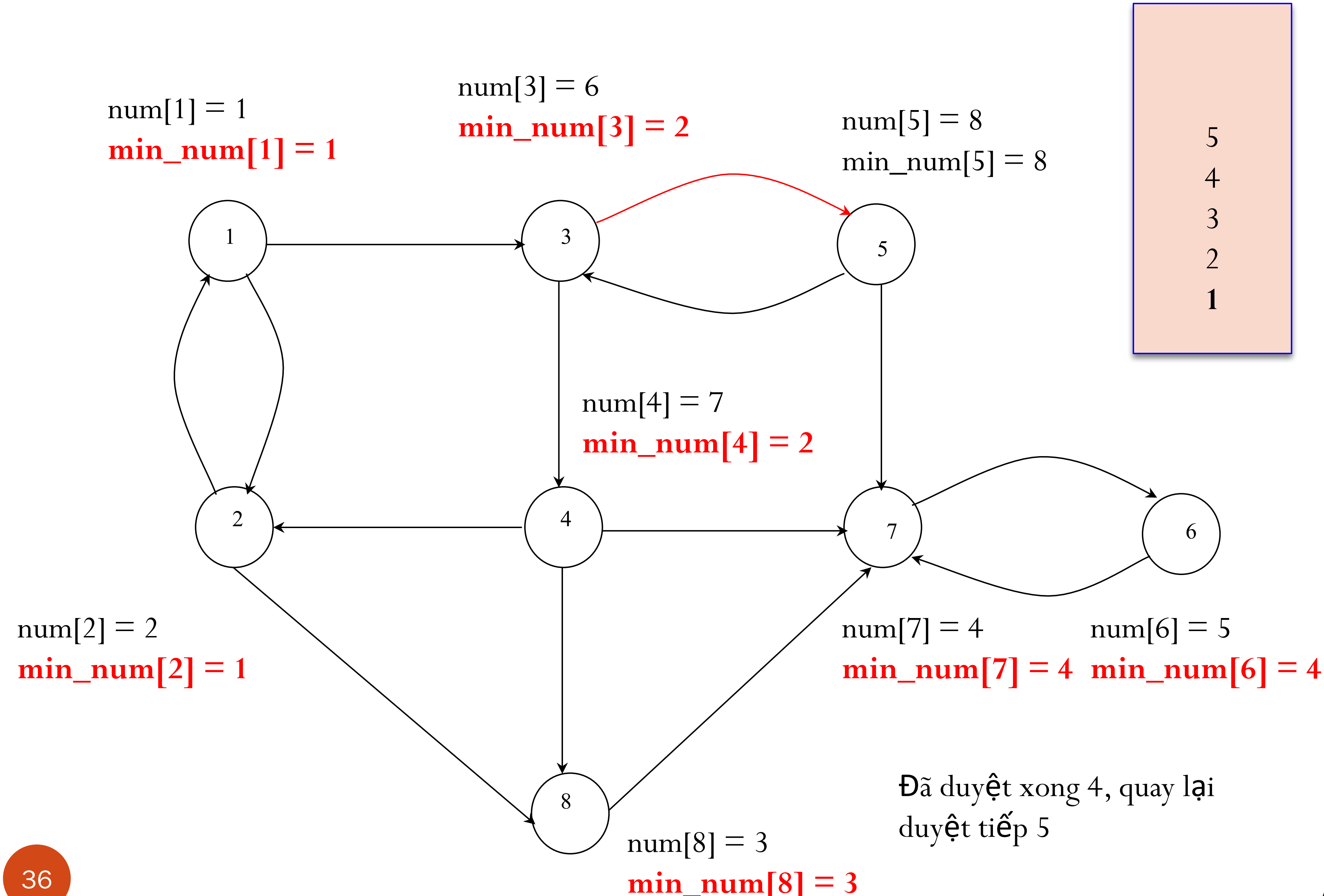
2 đang trên stack, cập nhật
min_num[4] = 2



7 không còn trên stack =>
không làm gì cả



8 không còn trên stack => không làm gì cả

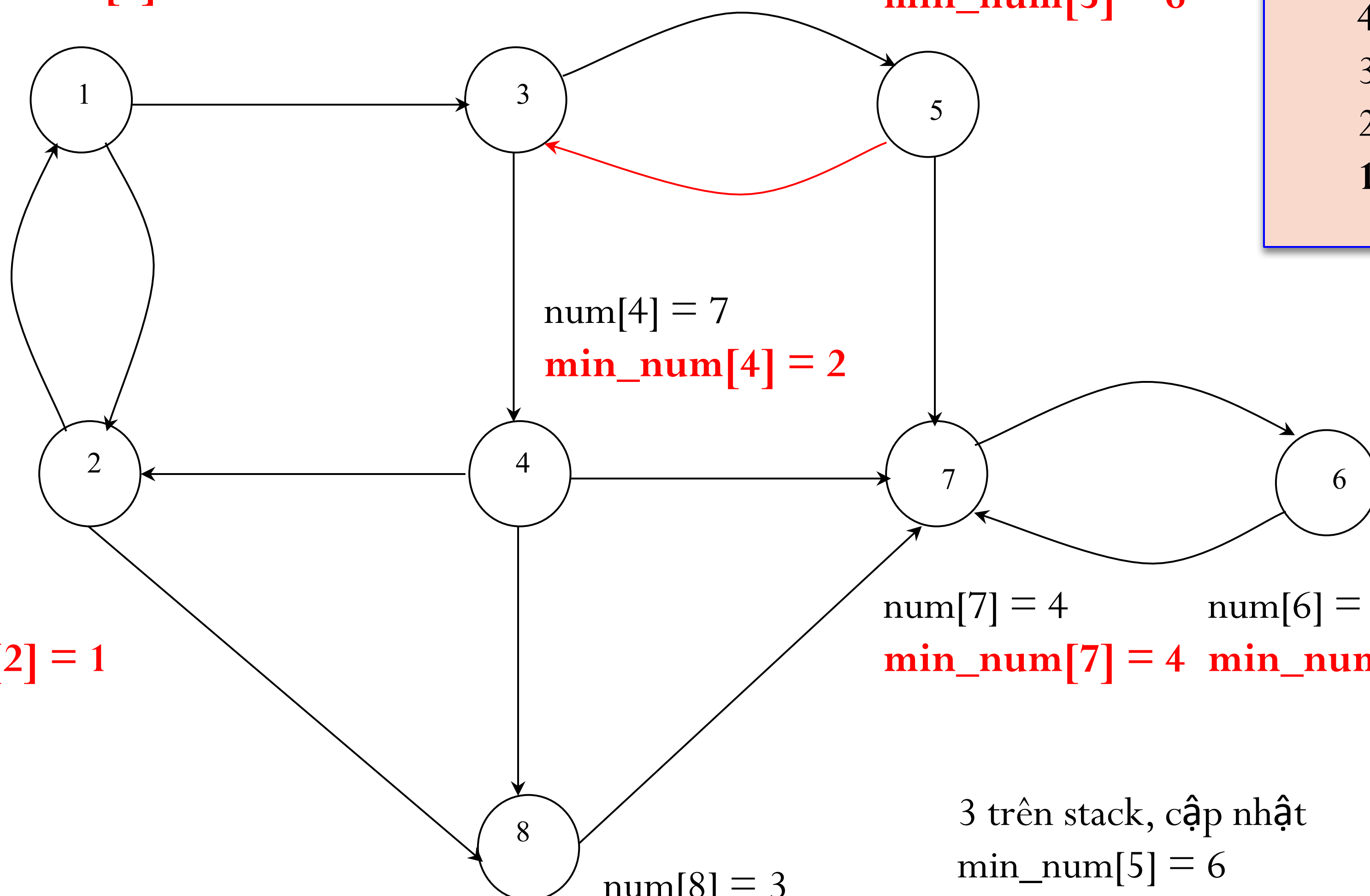


Đã duyệt xong 4, quay lại duyệt tiếp 5

num[1] = 1
min_num[1] = 1

num[3] = 6
min_num[3] = 2

num[5] = 8
min_num[5] = 6



num[4] = 7
min_num[4] = 2

num[2] = 2
min_num[2] = 1

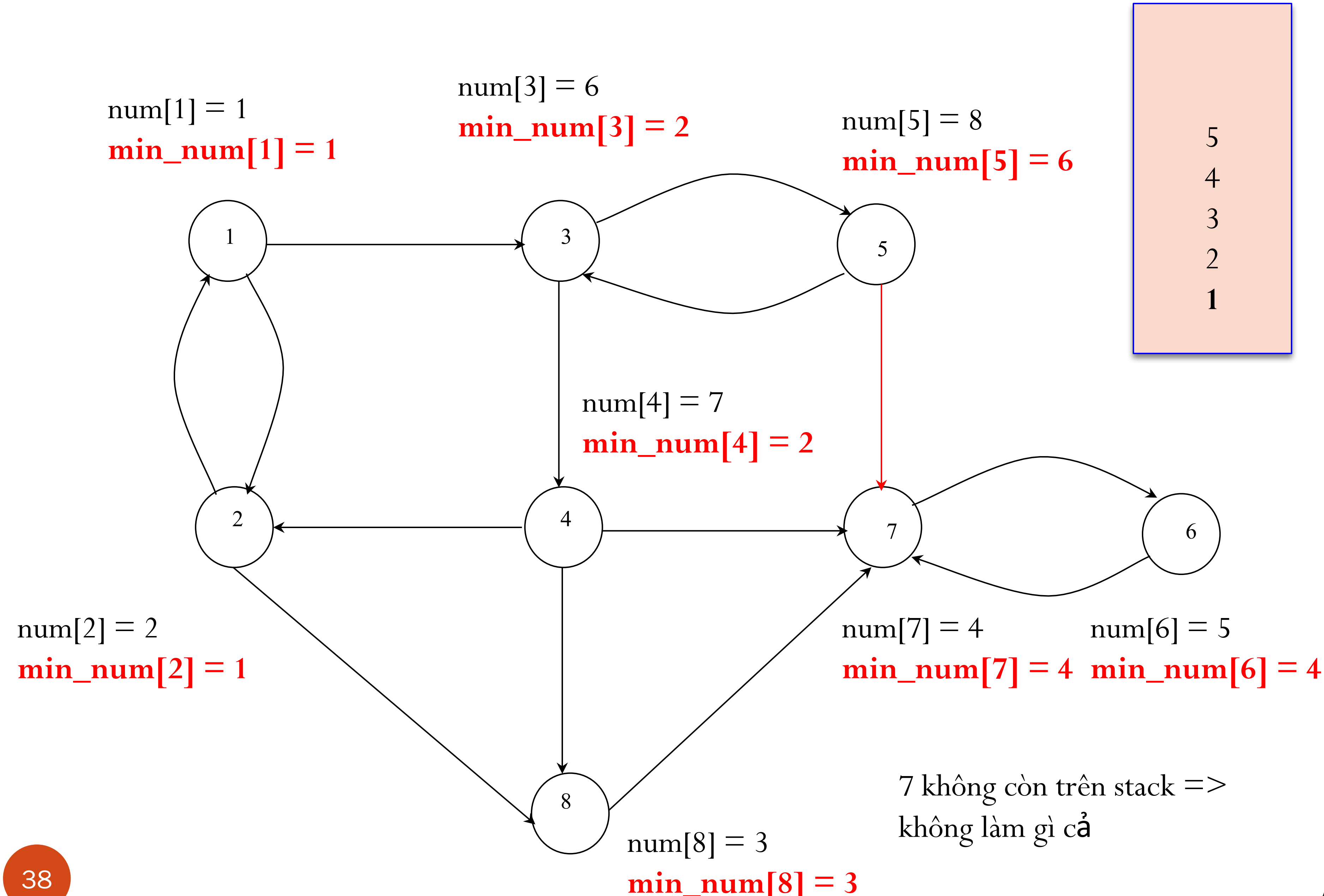
num[7] = 4
min_num[7] = 4

num[6] = 5
min_num[6] = 4

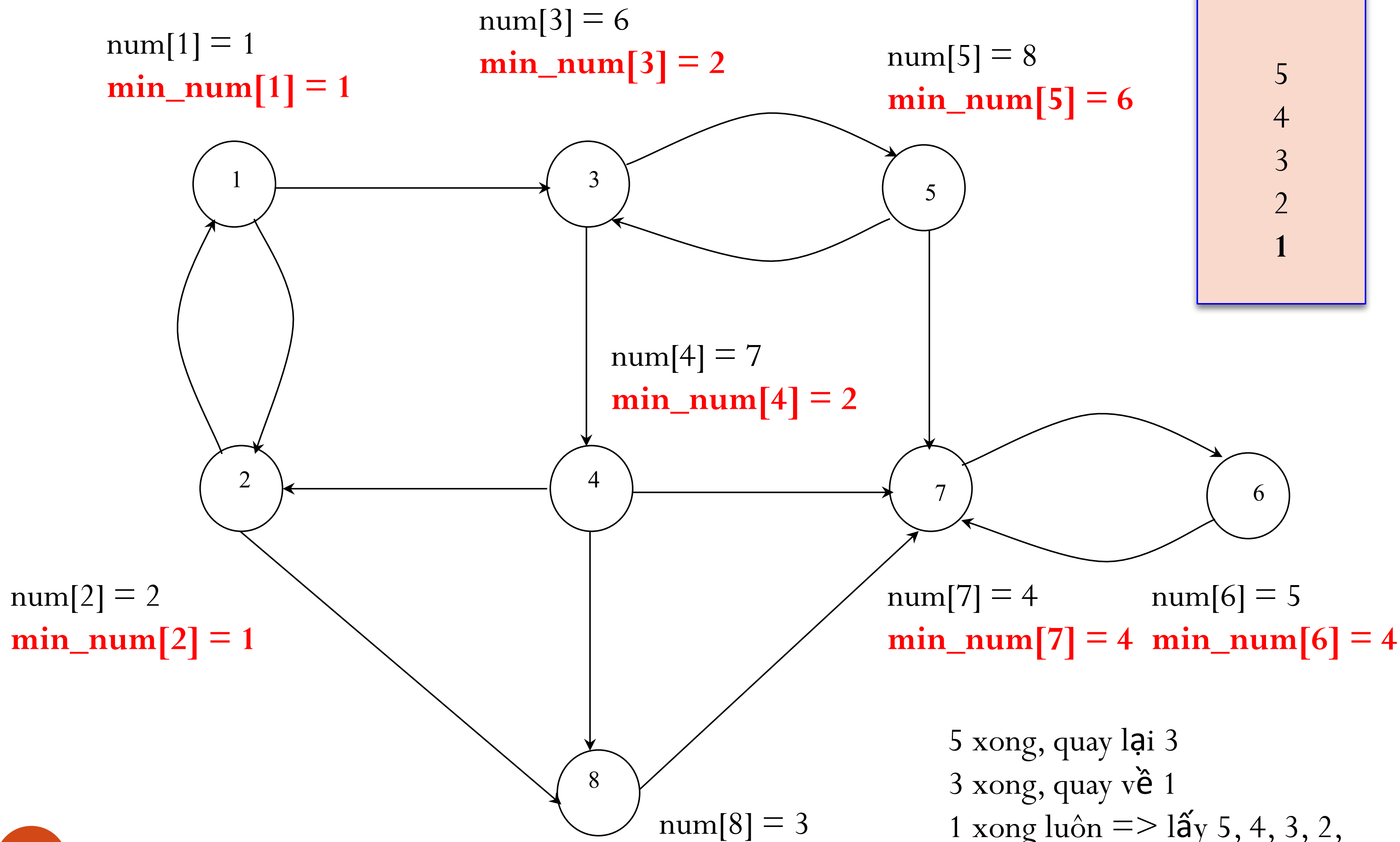
num[8] = 3
min_num[8] = 3

5
4
3
2
1

3 trên stack, cập nhật
min_num[5] = 6



7 không còn trên stack =>
không làm gì cả

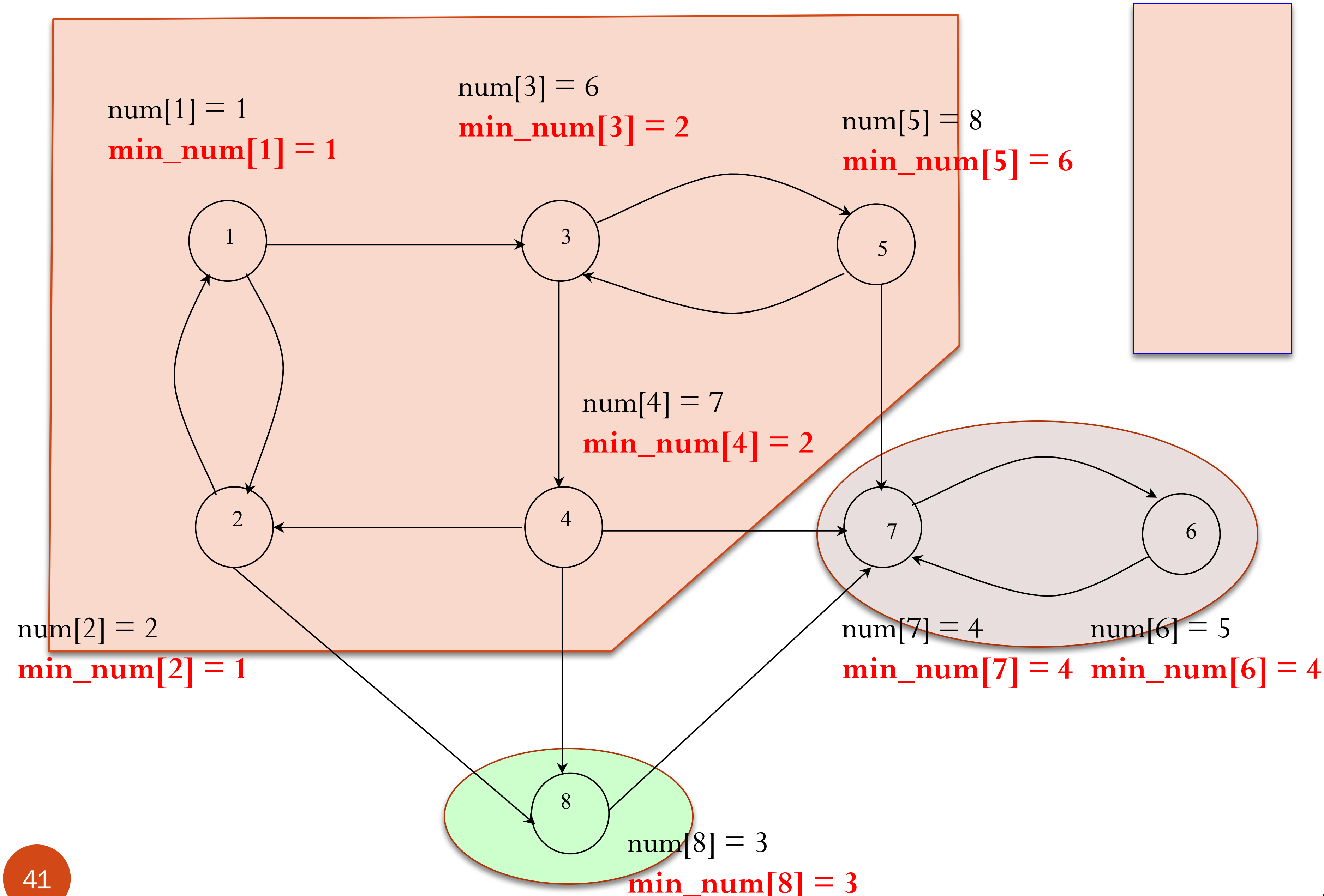


5
4
3
2
1

5 xong, quay lại 3
3 xong, quay về 1
1 xong luôn => lấy 5, 4, 3, 2,
1 ra khỏi stack

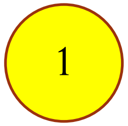
Lưu ý:

- Thứ tự các đỉnh kề của 1 đỉnh được sắp xếp từ bé đến lớn.
- Khi duyệt xong 1 đỉnh y, quay về đỉnh cha x (đỉnh trước), cập nhật lại min_num[x] (so với **min_num[y]**)
- Khi xét 1 đỉnh kề y của x mà y đang có mặt trong stack => cập nhật lại min_num[x] (so với **num[y]**).

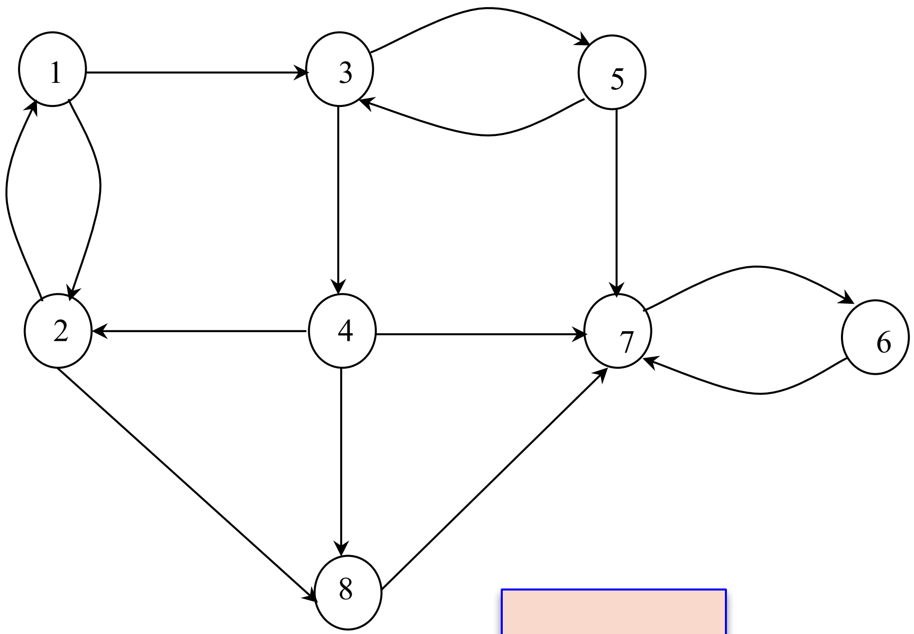

num[1] = 1
min_num[1] = 1
num[3] = 6
min_num[3] = 2
num[5] = 8
min_num[5] = 6
num[4] = 7
min_num[4] = 2
num[2] = 2
min_num[2] = 1
num[7] = 4
min_num[7] = 4
num[6] = 5
min_num[6] = 4
num[8] = 3
min_num[8] = 3
1
3
5
2
4
7
6
8

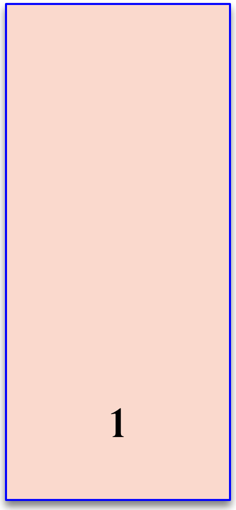
Cây duyệt theo chiều sâu

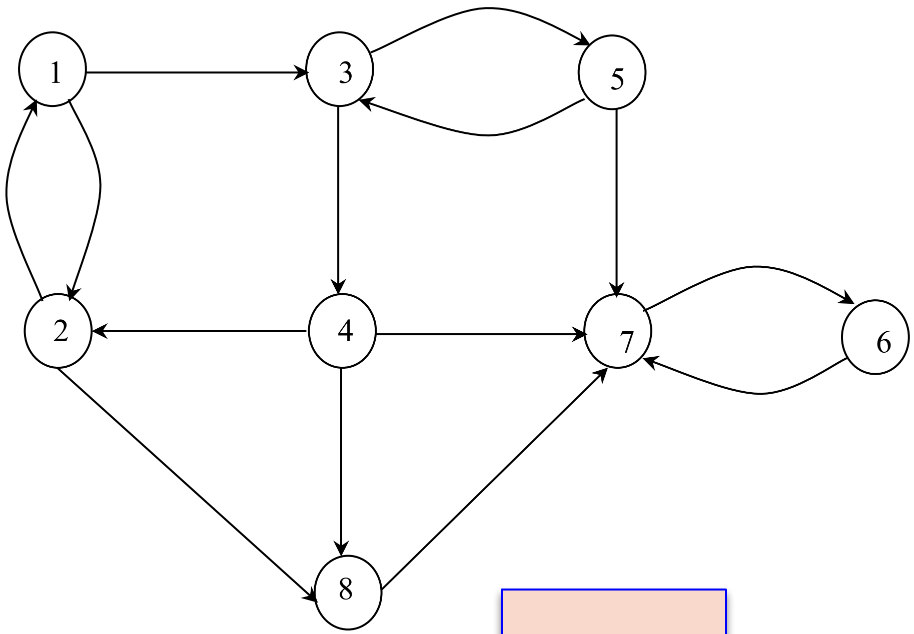
- Quá trình duyệt cây kết hợp với đánh số (num, min_num) tạo ra cây "duyệt đồ thị"
- Phần tiếp theo minh hoạ quá trình chạy giải thuật và cây "duyệt đồ thị tương ứng"
- Cung liền nét (cung thuận)
- Cung không liền nét: cung quay lui (minh hoạ việc cập nhật min_num)

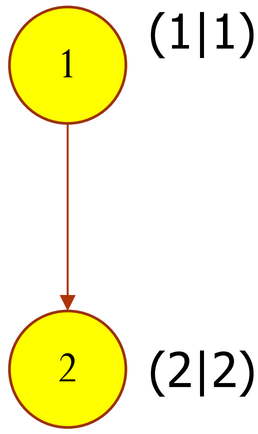


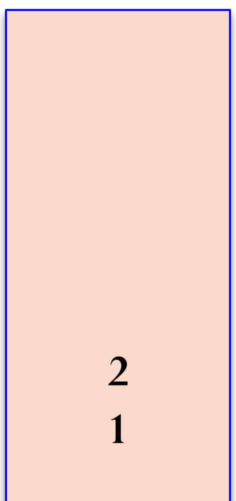
(1|1)

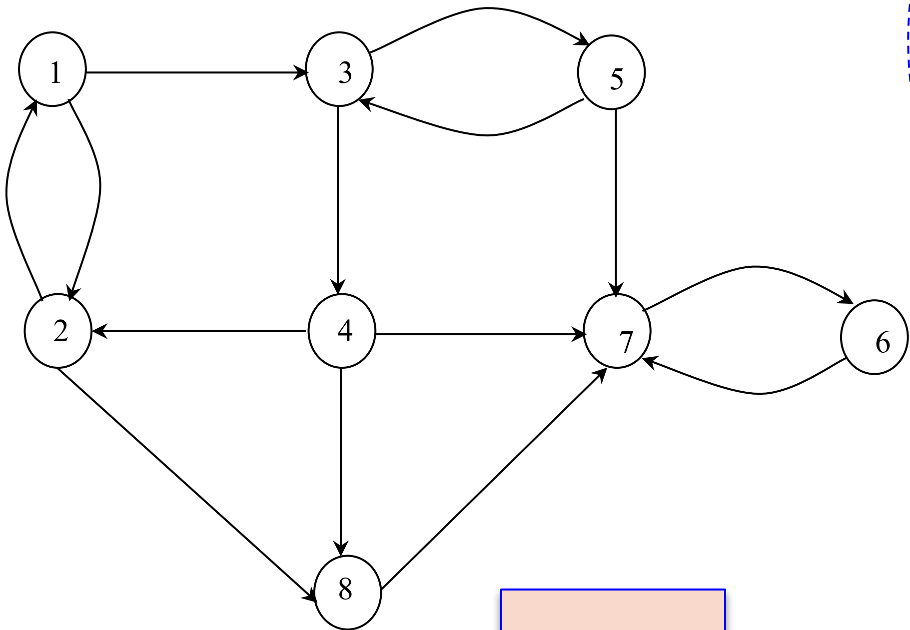


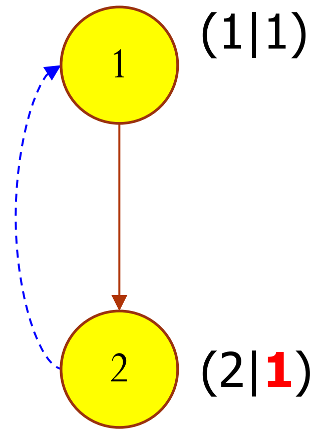



1
3
5
2
4
7
6
8

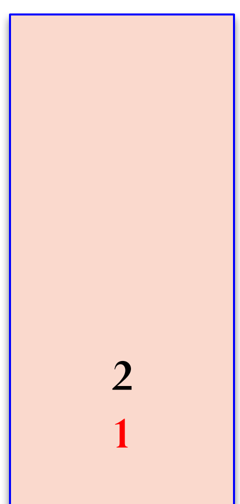

1
(1|1)
2
(2|2)

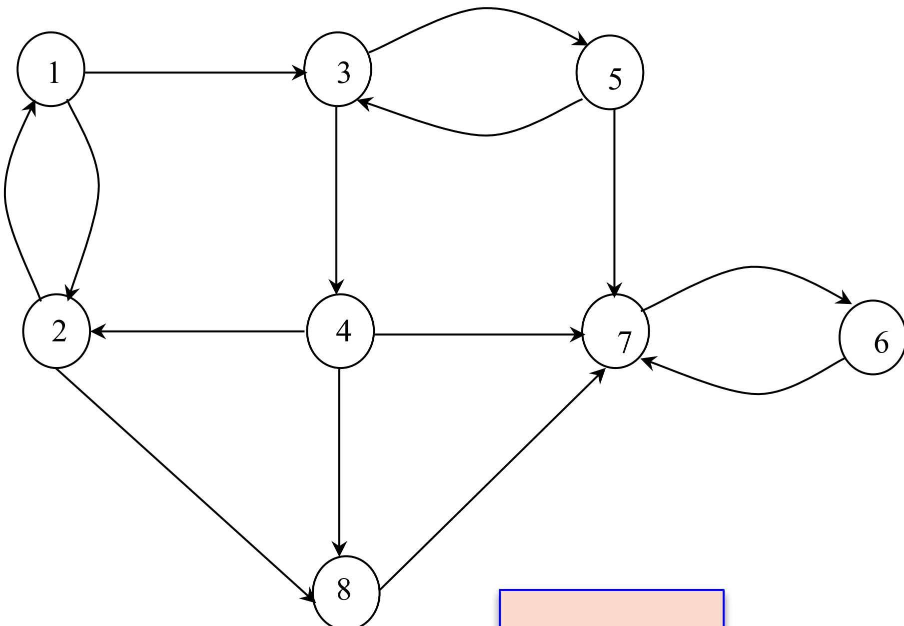

2
1

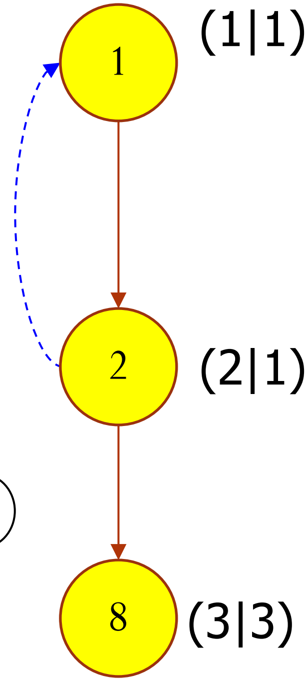




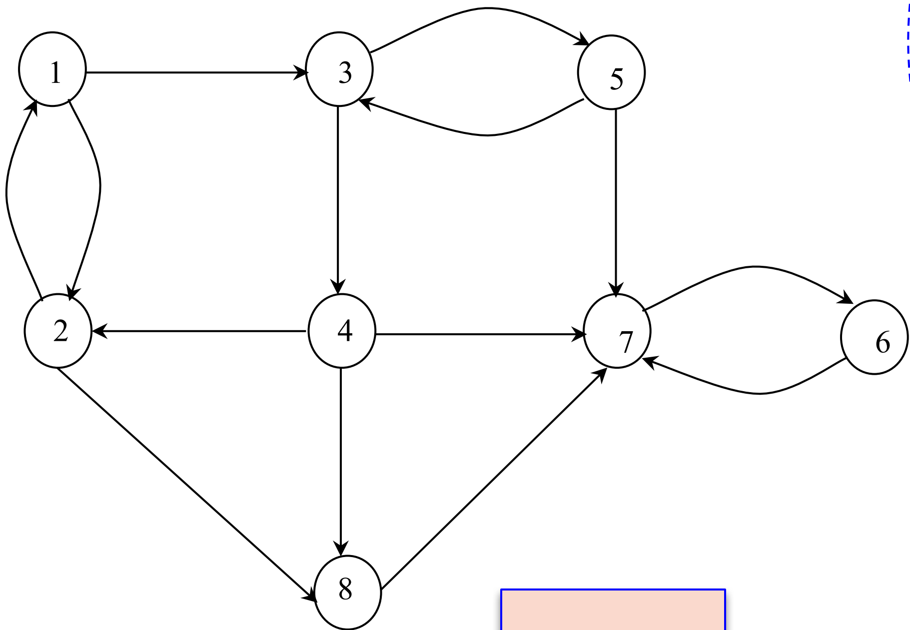
Cập nhật
min_num[2] = 1

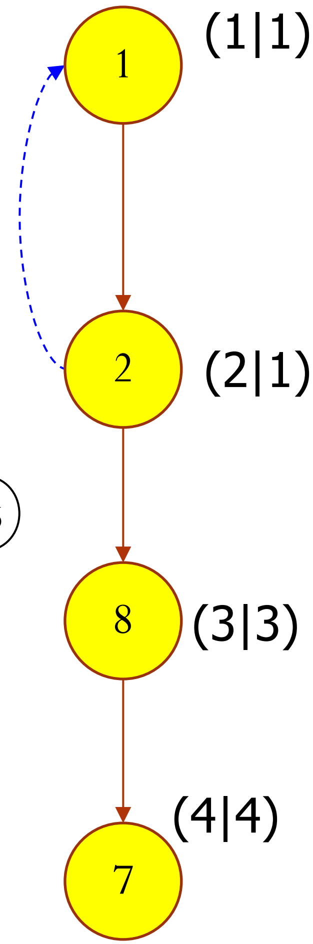




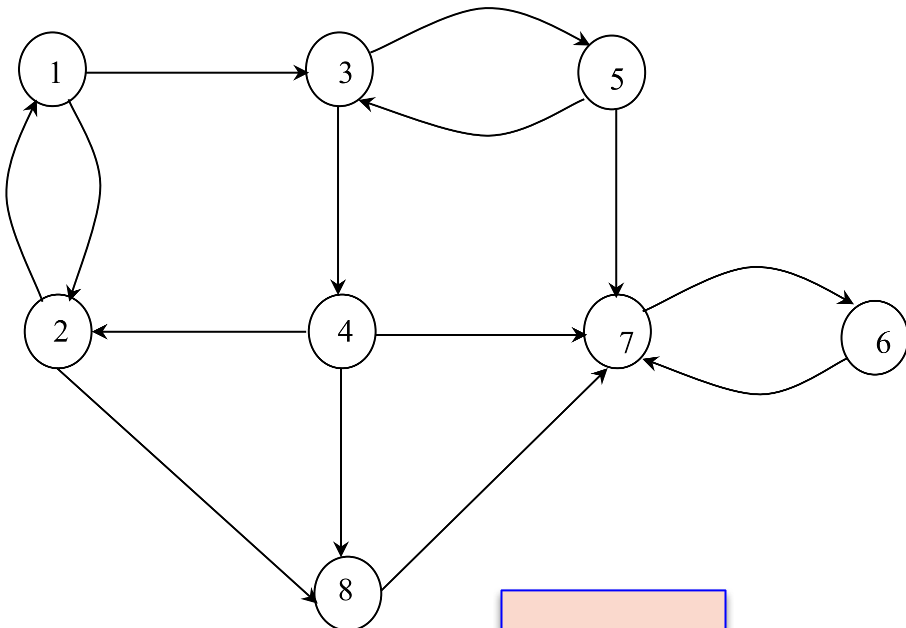


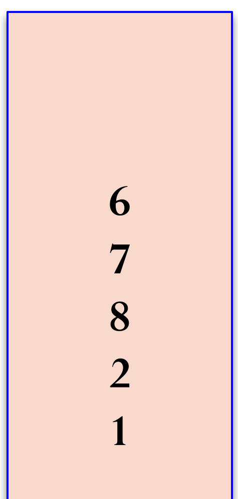
8
2
1

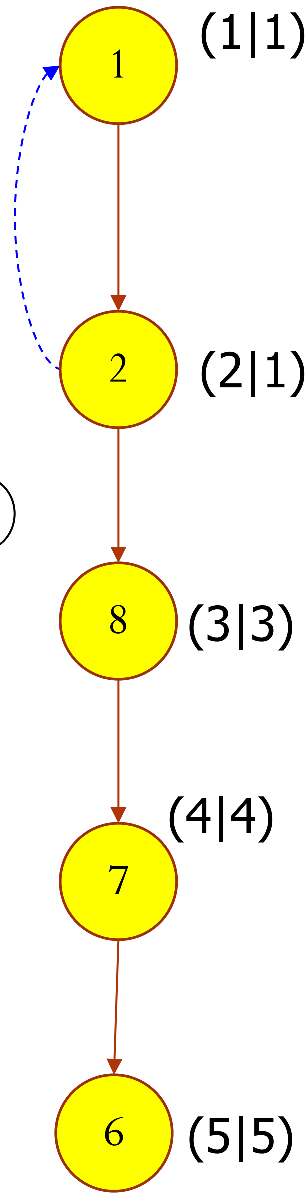


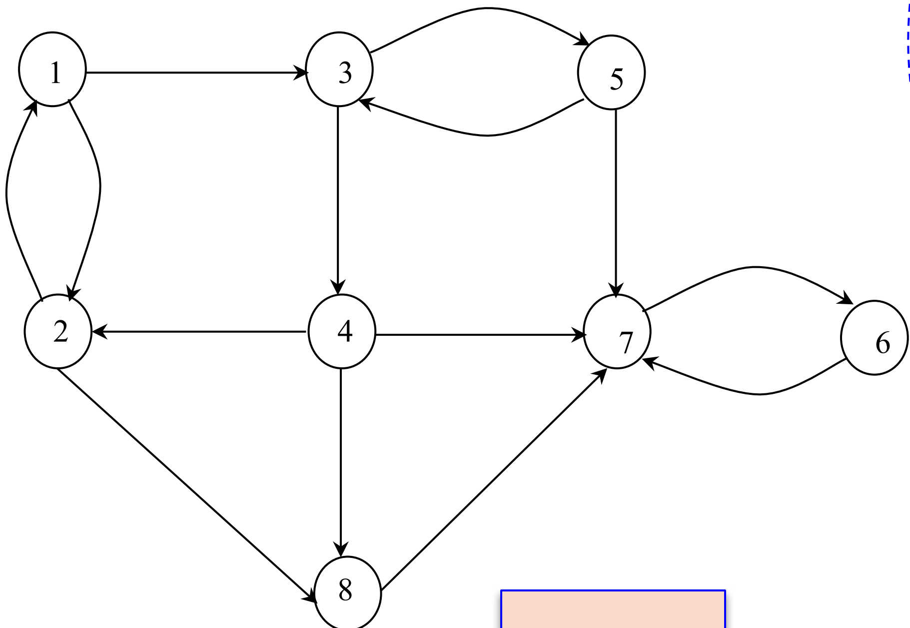


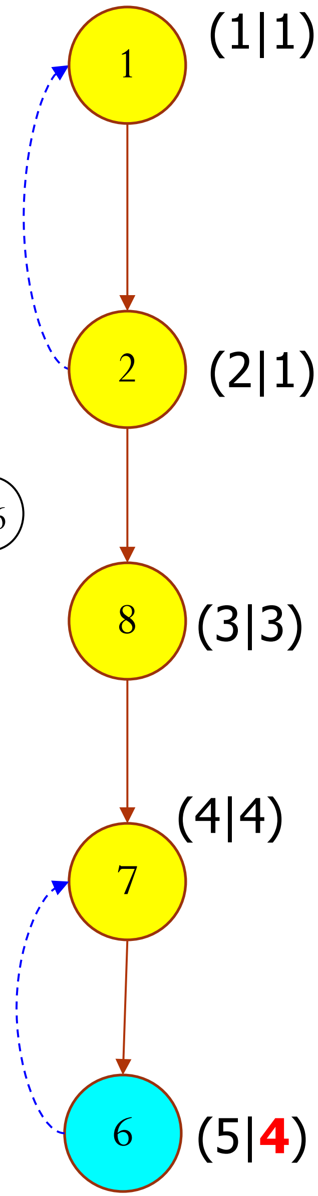
7
8
2
1

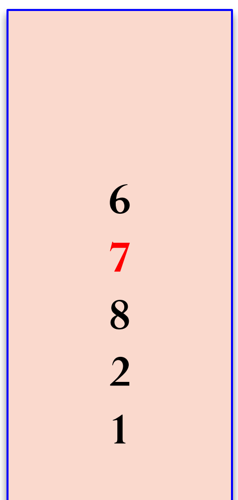

1
3
5
2
4
7
6
8


6
7
8
2
1

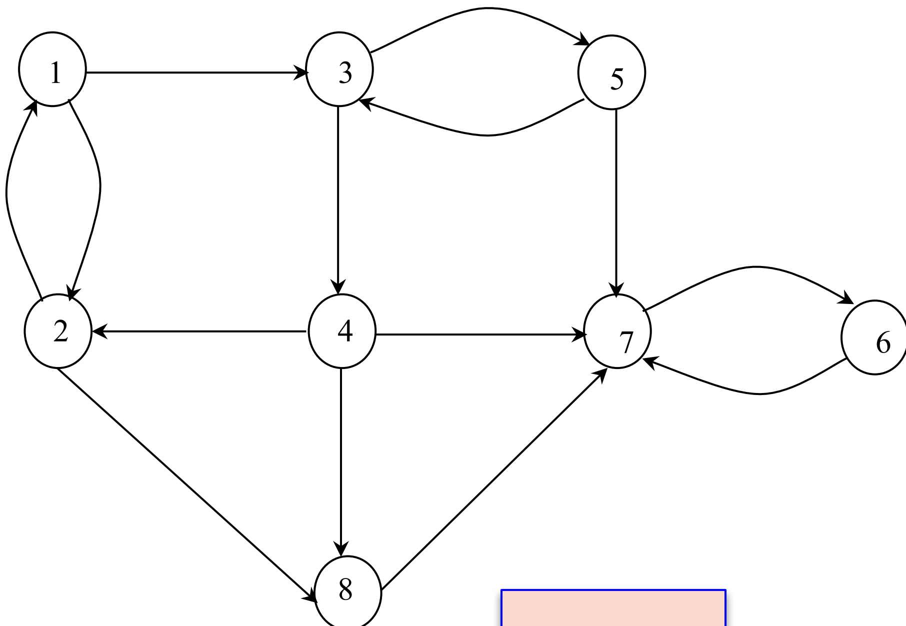

1
(1|1)
2
(2|1)
8
(3|3)
(4|4)
7
6
(5|5)



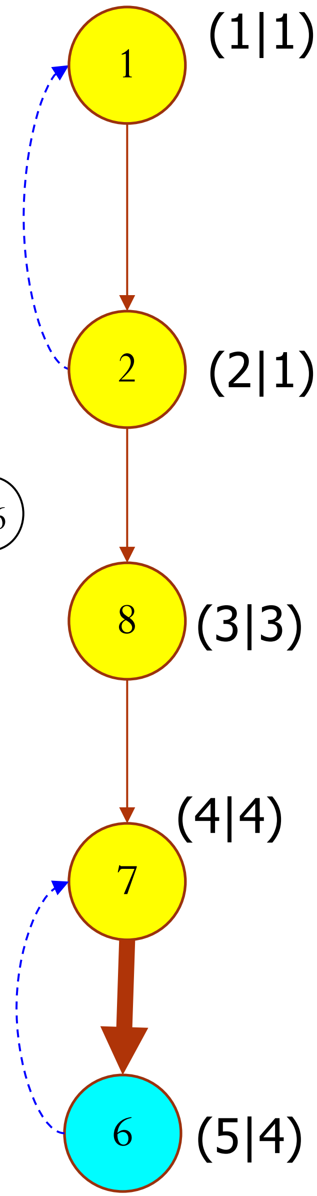




6 đã được duyệt xong, nhưng 5 != 4

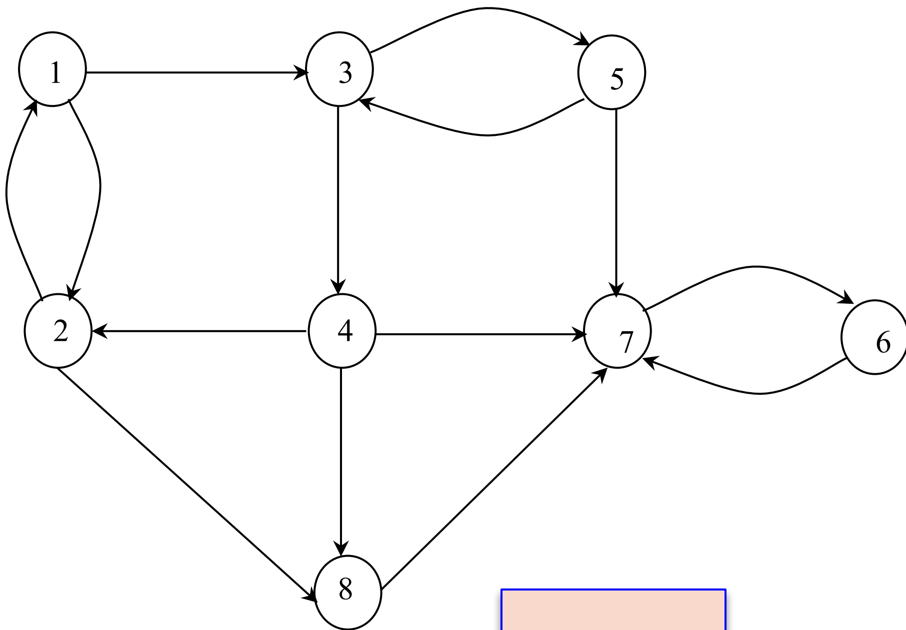


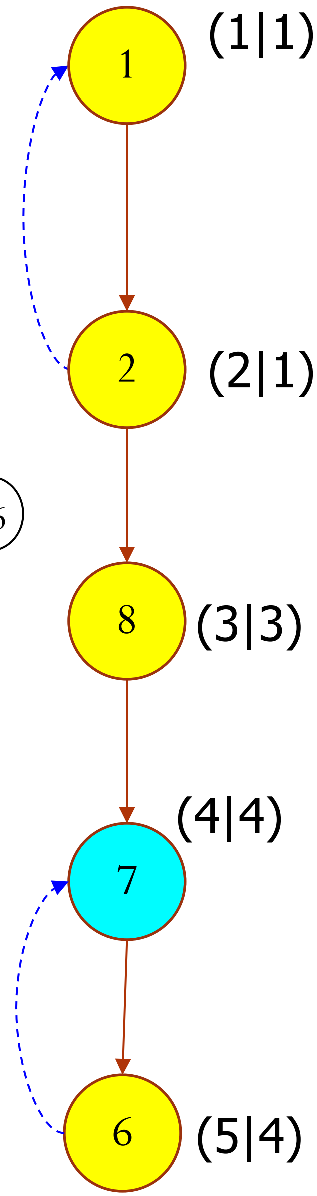
6
7
8
2
1



6 đã được duyệt xong, nhưng 5 != 4

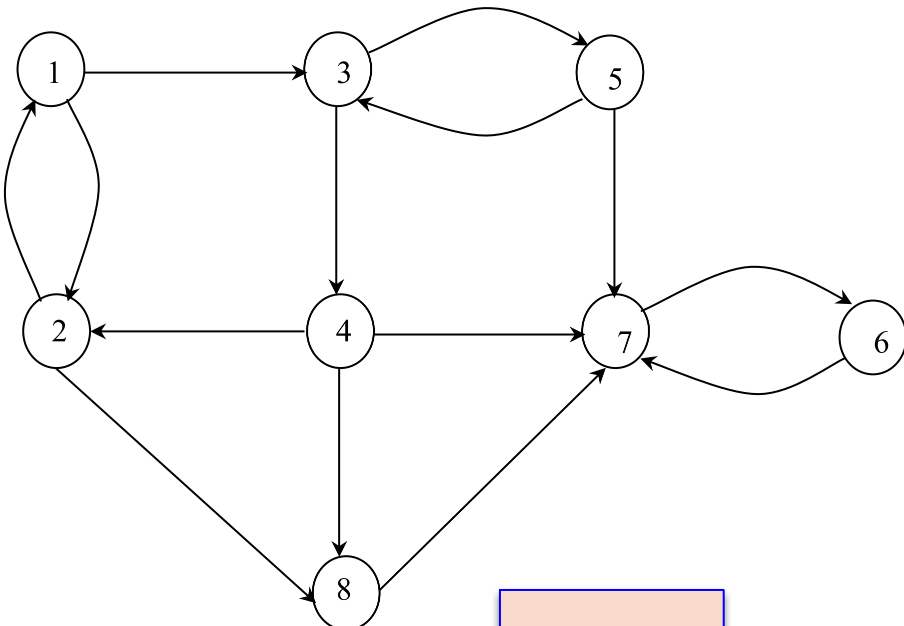
Kiểm tra cập nhật lại min_num[7] (vì lúc nãy 7 gọi duyệt 6) => **Không cần cập nhật vì 4 <= 4**

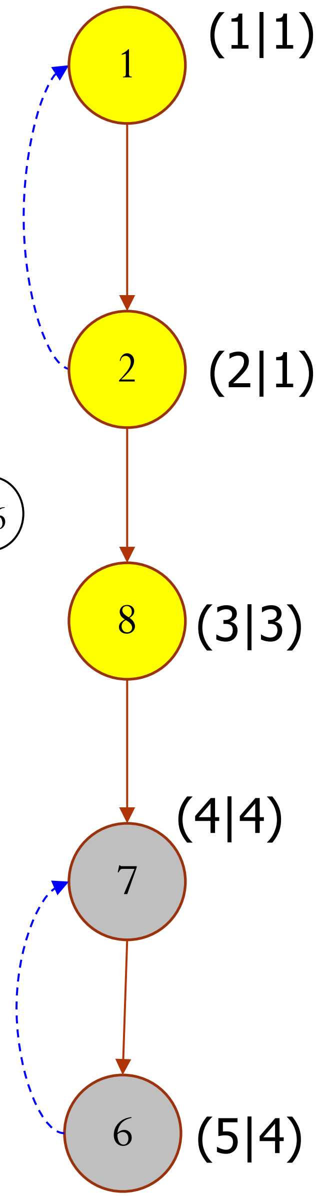




7 đã được duyệt xong và có 4 == 4

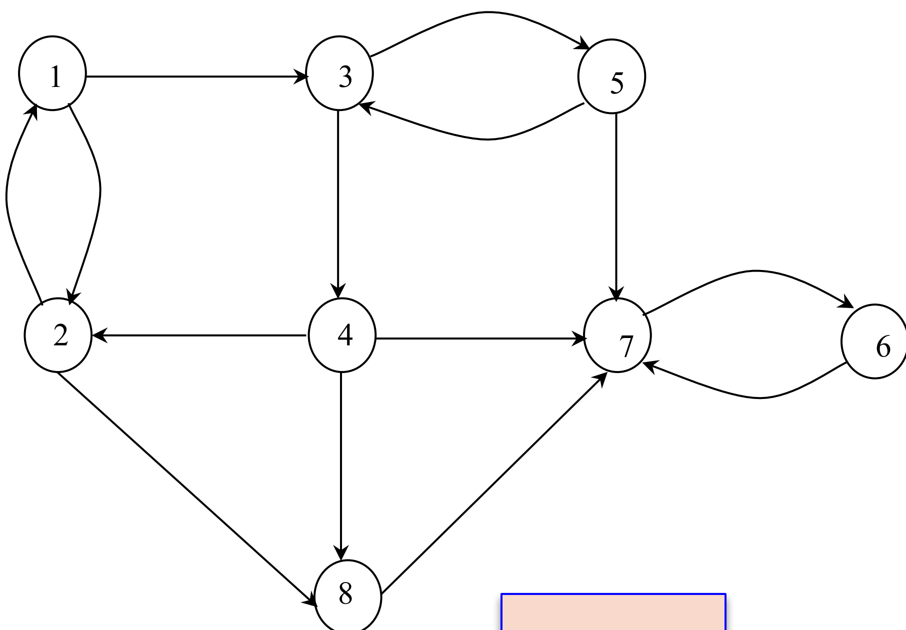
6
7
8
2
1

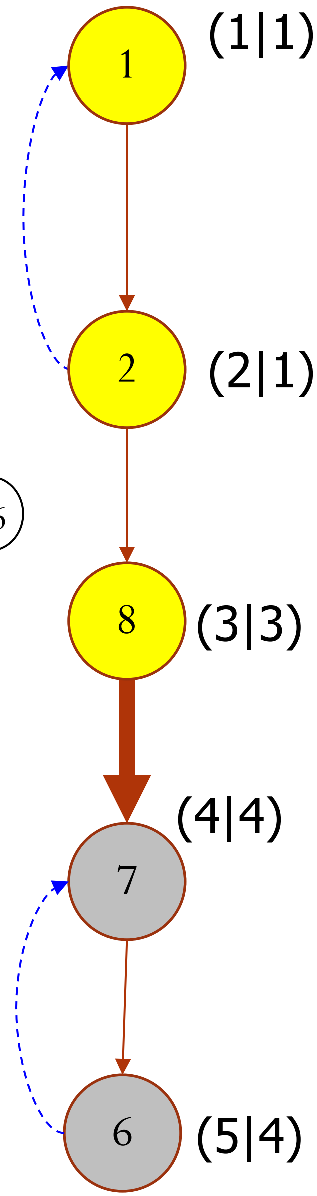




7 đã được duyệt xong và có 4 == 4
Loại bỏ 6, 7 ra
=> tìm được BPLT (6, 7)

~~6~~
~~7~~
8
2
1

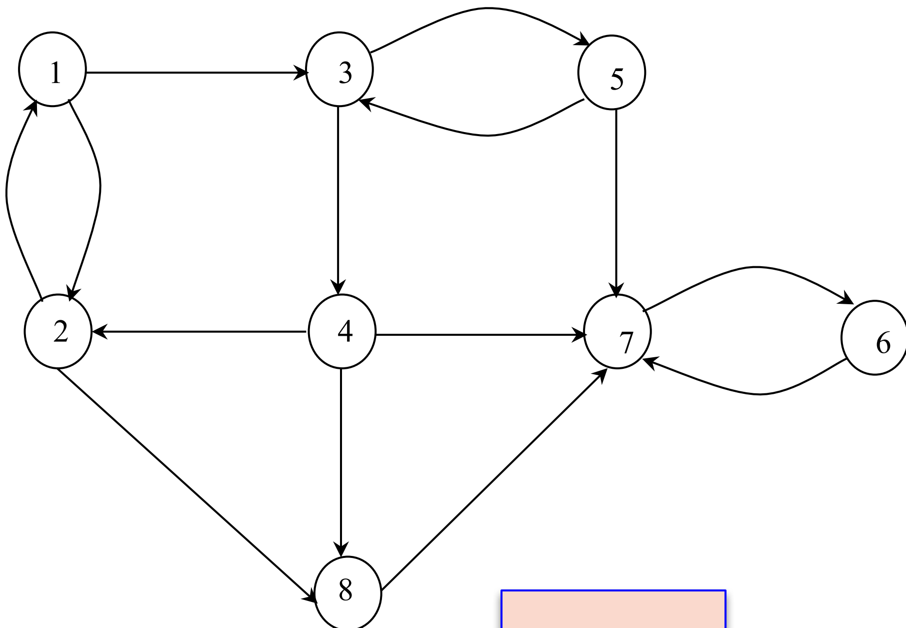


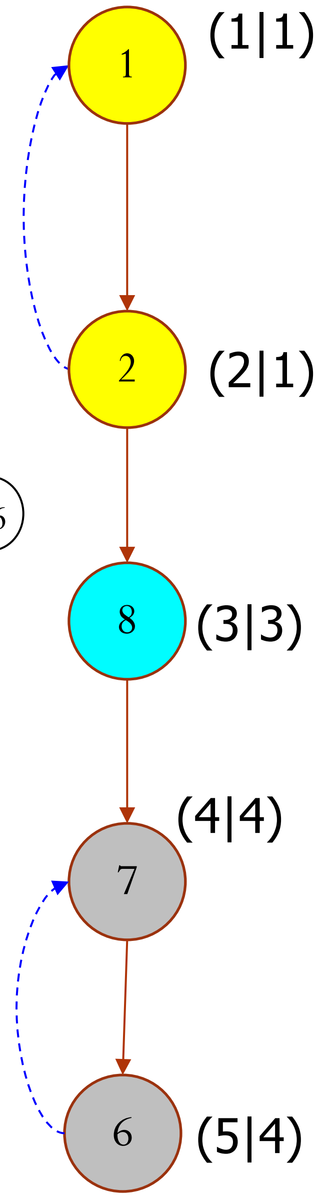


7 đã được duyệt xong và có 4 == 4
Loại bỏ 6, 7 ra
=> tìm được BPLT (6, 7)

~~6~~
~~7~~
8
2
1

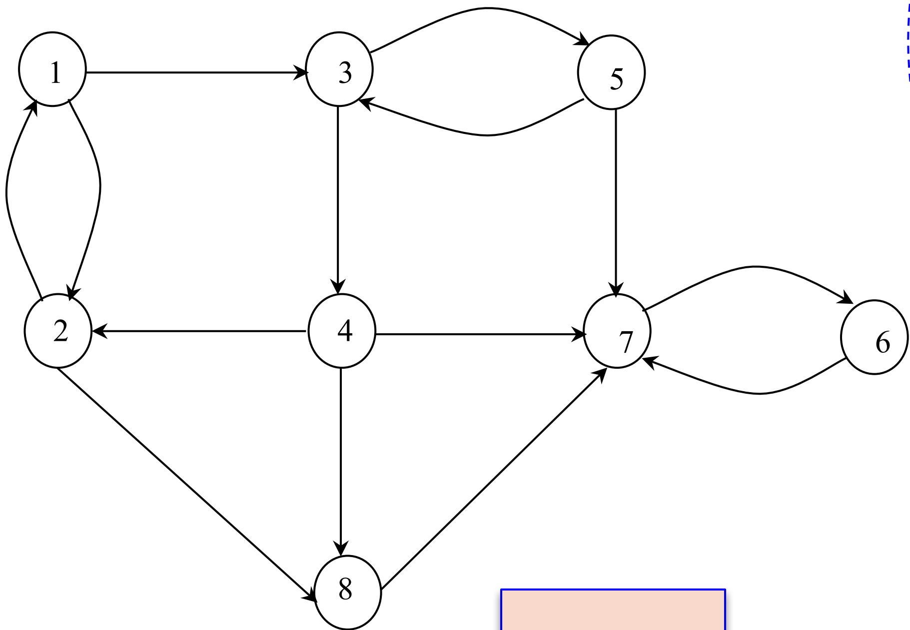
Kiểm tra cập nhật lại min_num[8] (vì lúc nãy 8 gọi duyệt 7) => **Không cần cập nhật vì 3 <= 4**

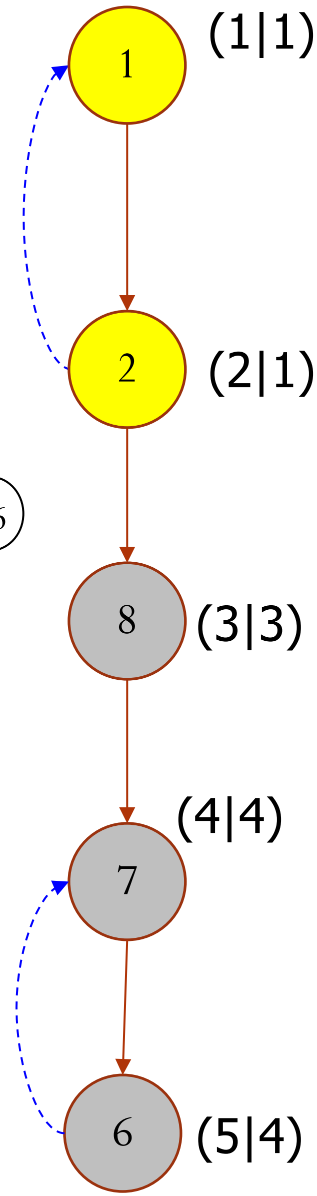




8 đã được duyệt
xong và có 3 == 3

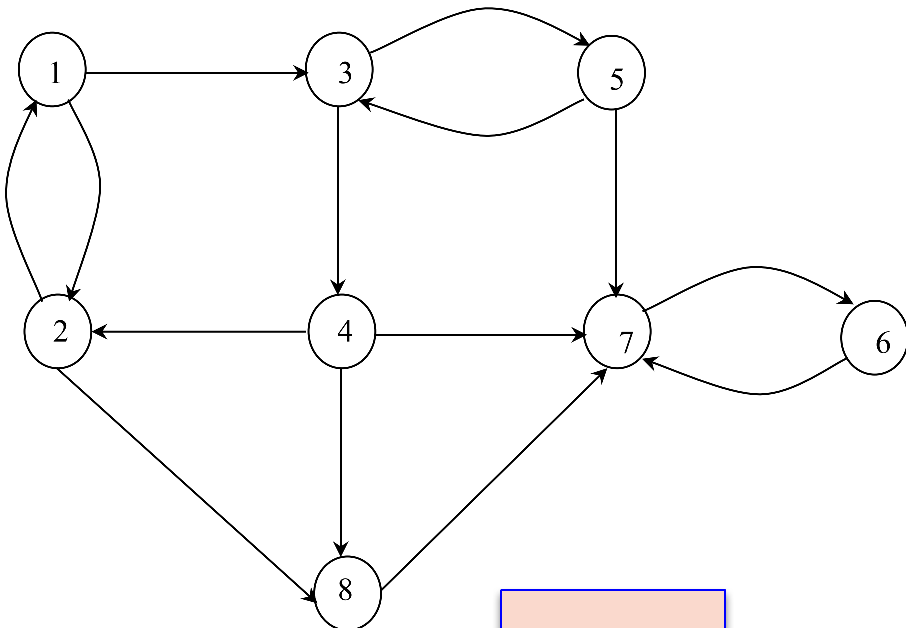
8
2
1

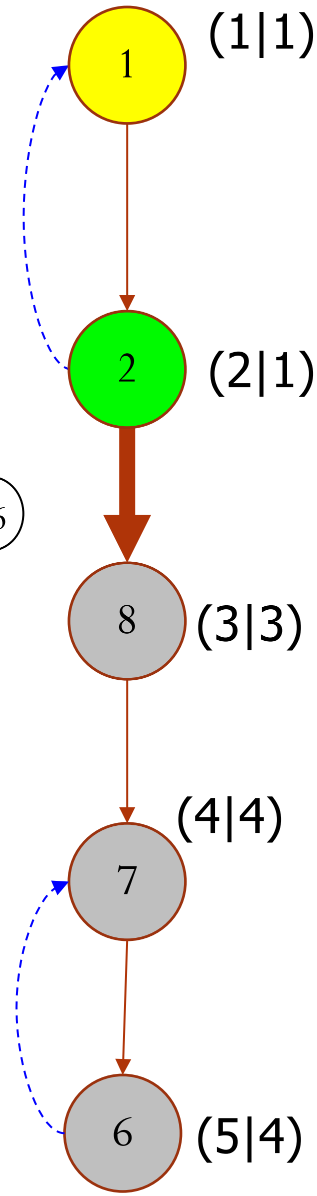




8 đã được duyệt xong và có 3 == 3
Loại bỏ 8 ra
=> tìm được BPLT (8)

~~8~~
2
1

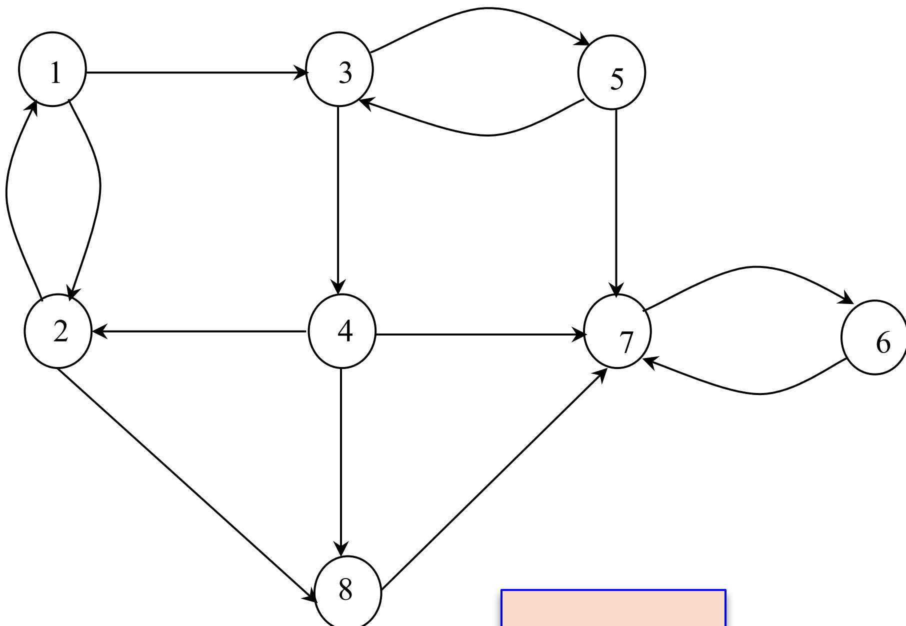


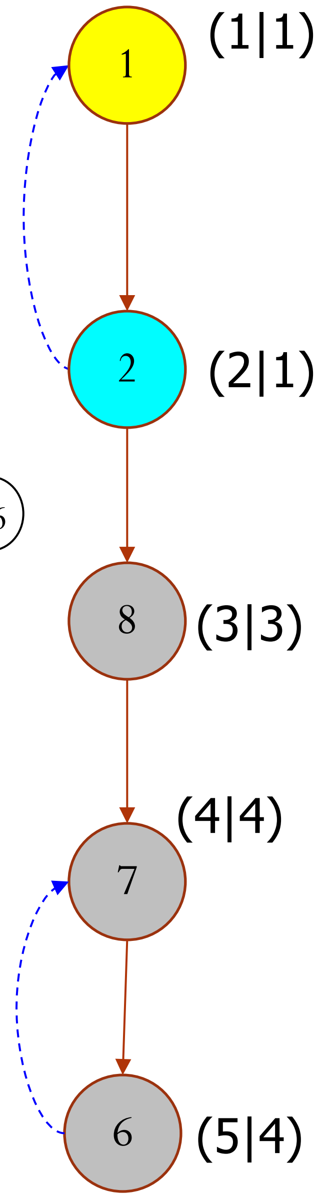


8 đã được duyệt xong và có 3 == 3
Loại bỏ 8 ra
=> tìm được BPLT (8)

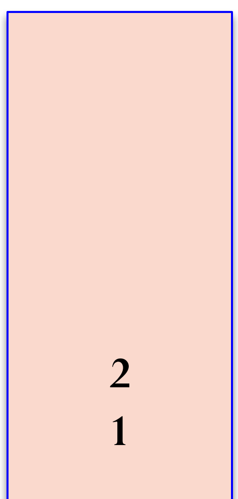
~~8~~
2
1

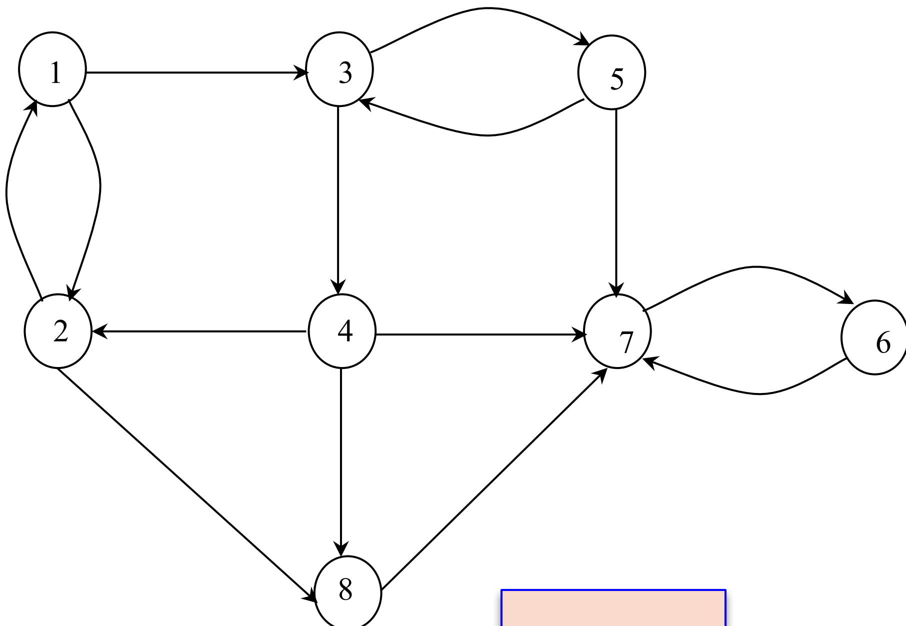
Kiểm tra cập nhật lại min_num[2] (vì lúc nãy 2 gọi duyệt 8) => **Không cần cập nhật vì 1 <= 3**



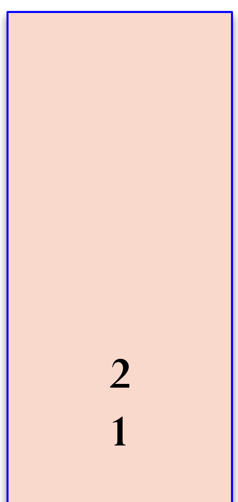


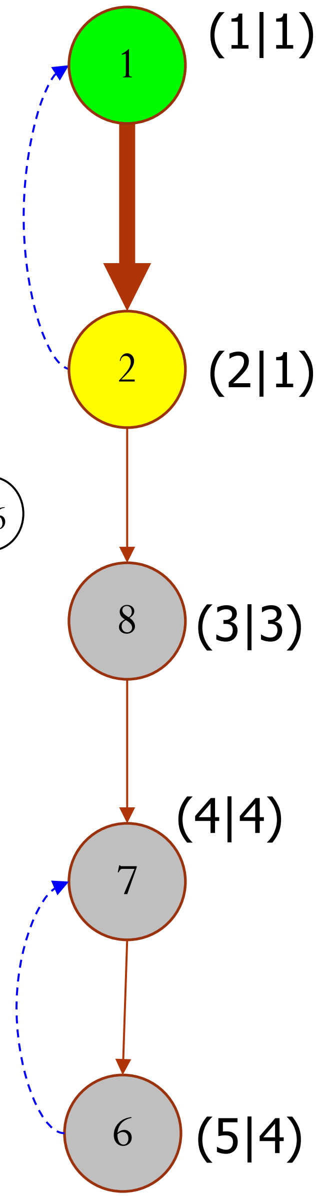
2 đã được duyệt
xong và có 2 != 1



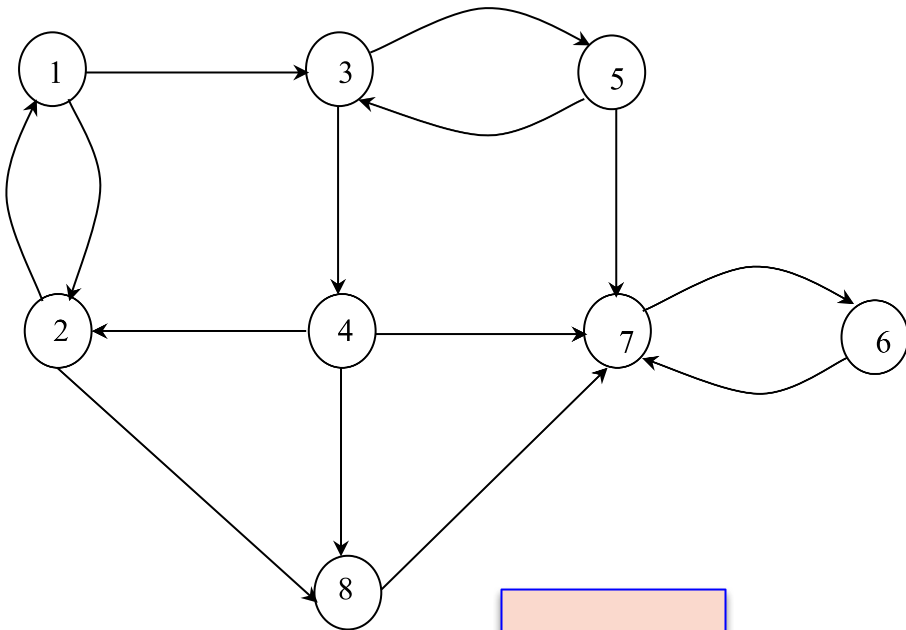


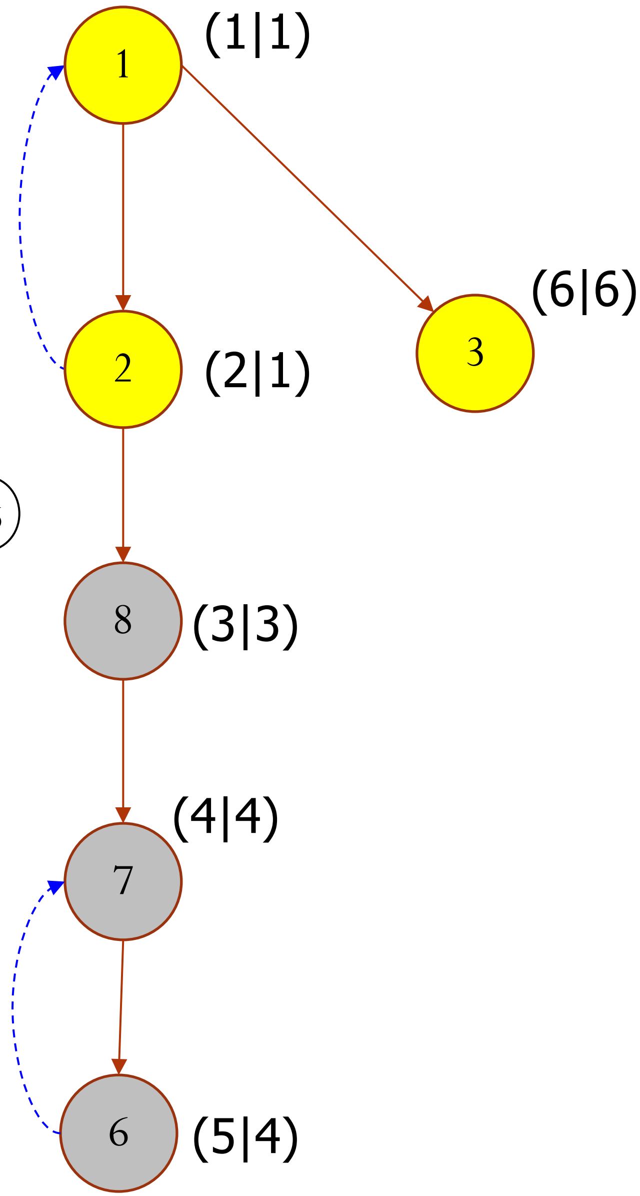
2 đã được duyệt
xong và có 2 != 1



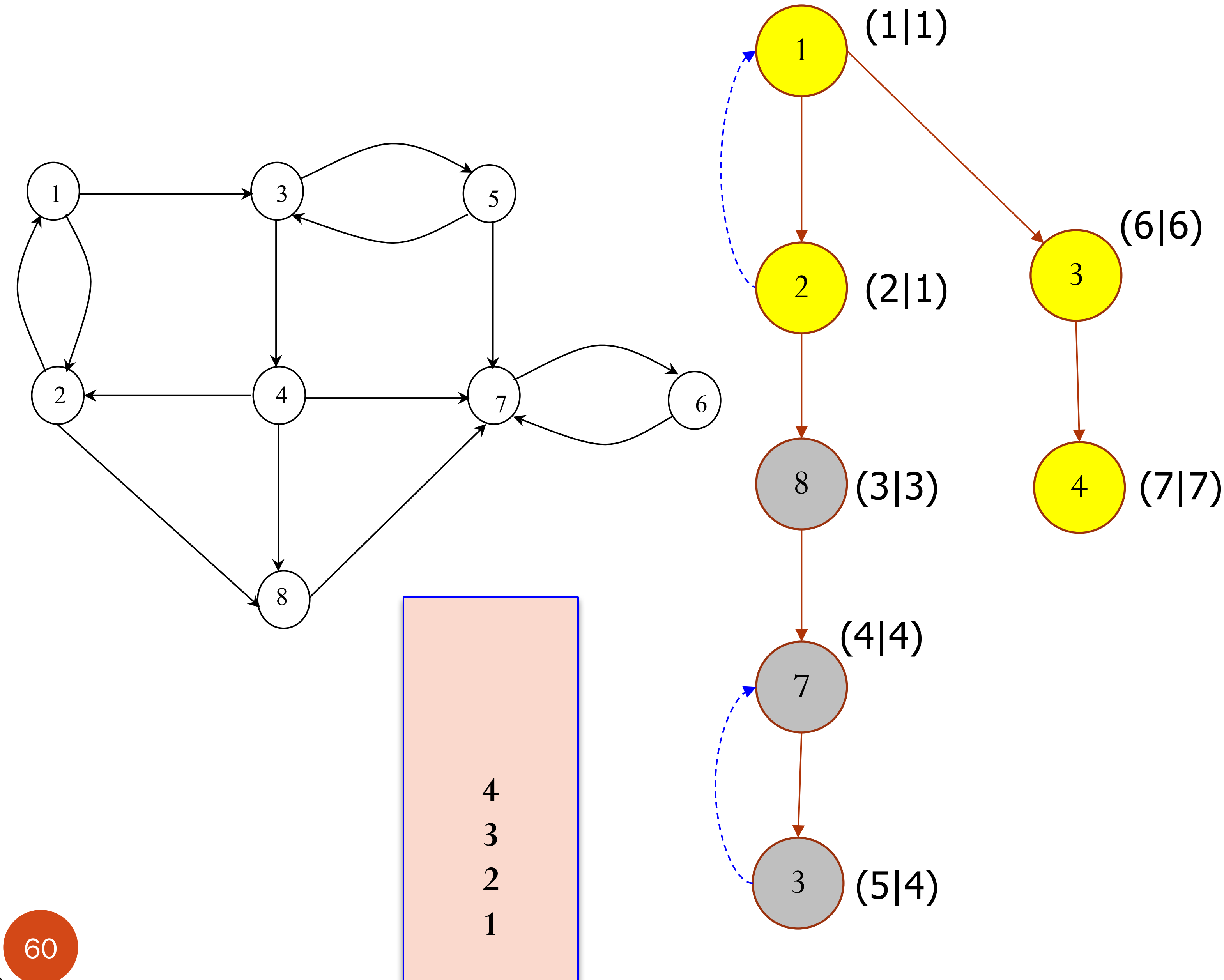


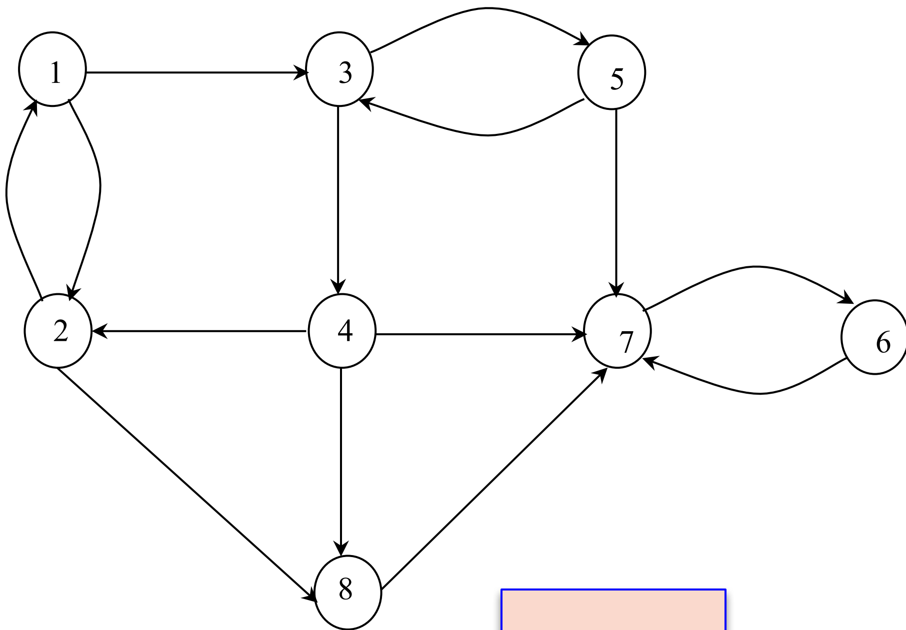
Kiểm tra cập nhật lại min_num[1] (vì lúc nãy 1 gọi duyệt 2) => **Không cần cập nhật vì 1 <= 1**

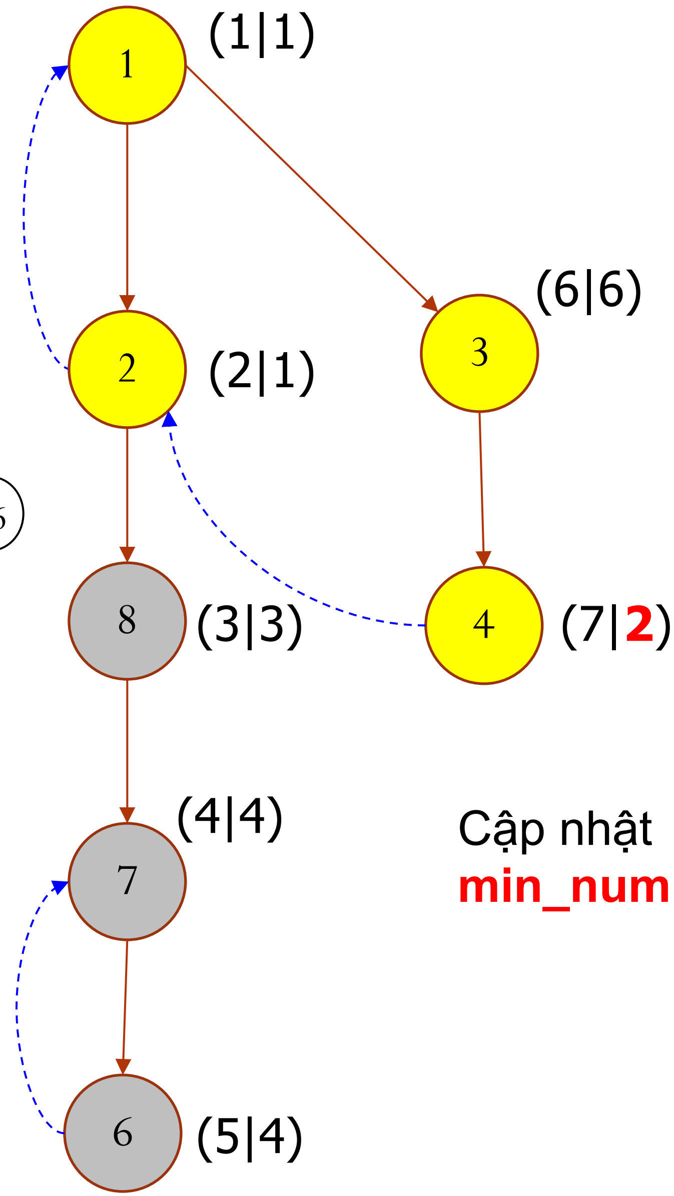




3
2
1

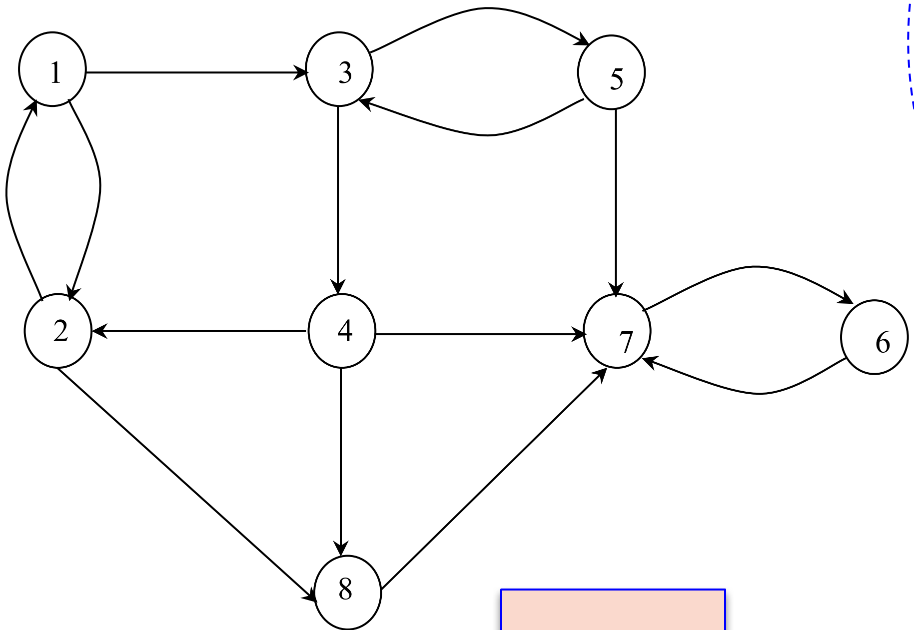

1
3
5
2
4
7
6
8
4
3
2
1
(1|1)
(2|1)
(6|6)
(3|3)
(7|7)
(4|4)
(5|4)



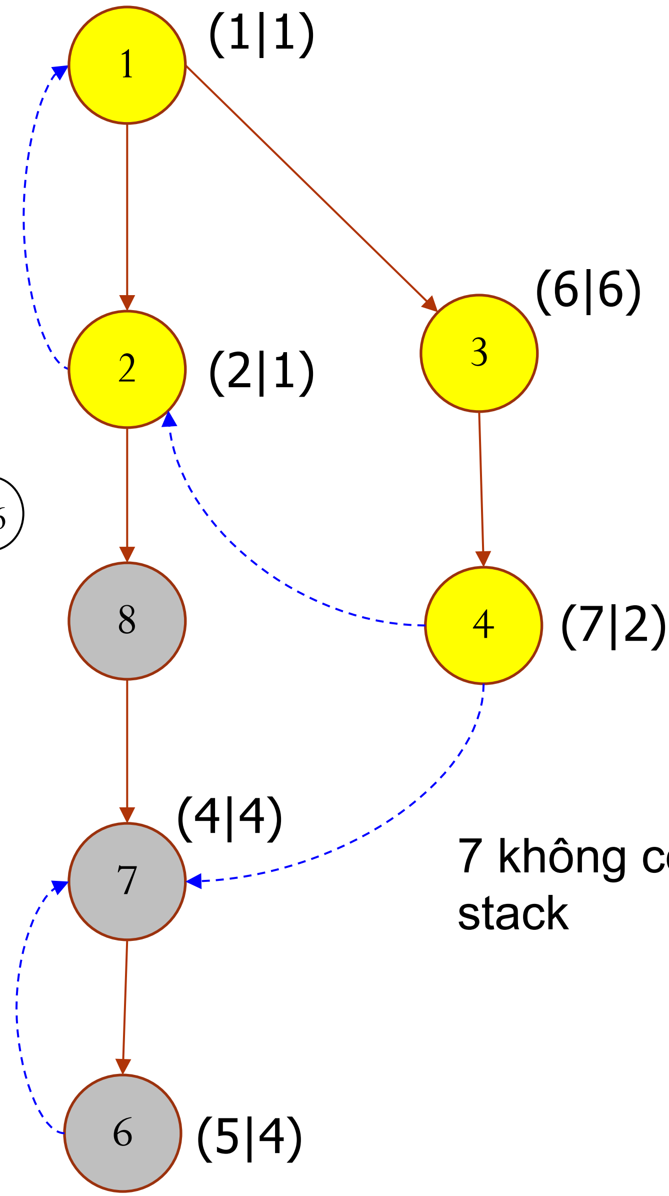


4
3
2
1

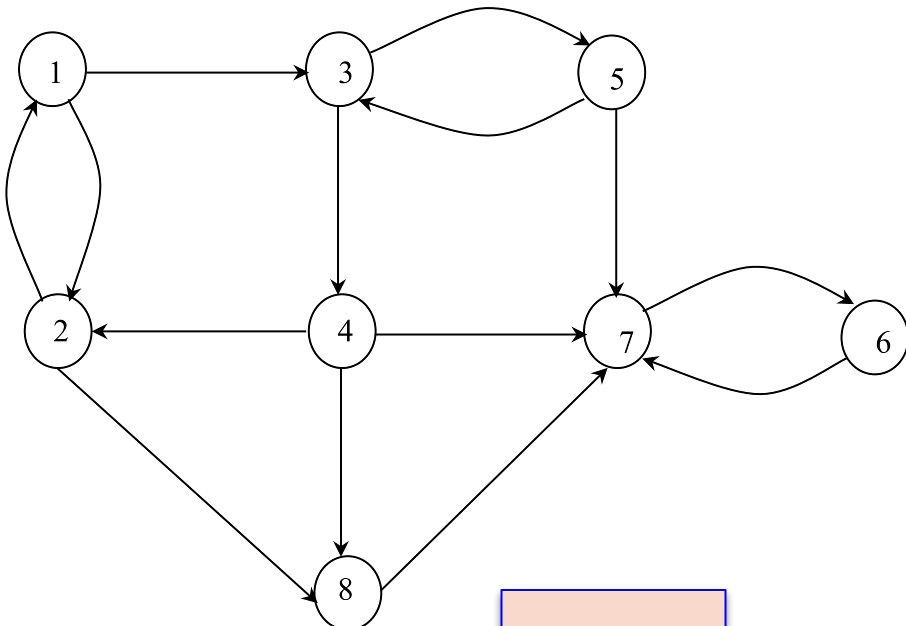
Cập nhật
min_num[4] = 2

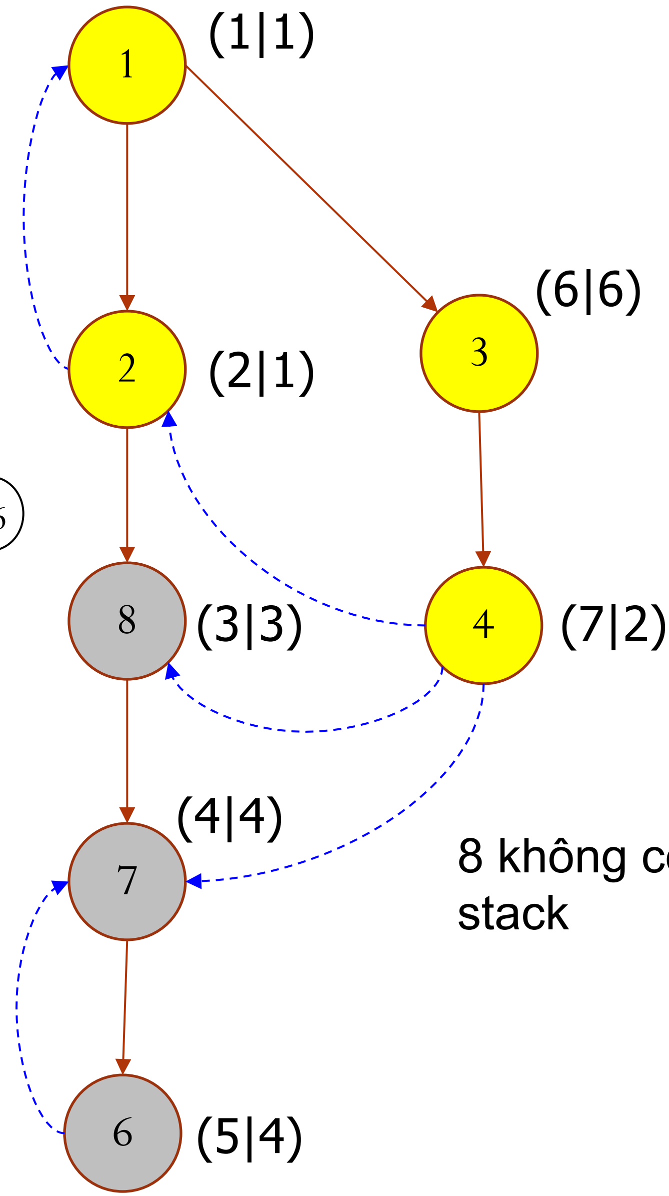


4
3
2
1



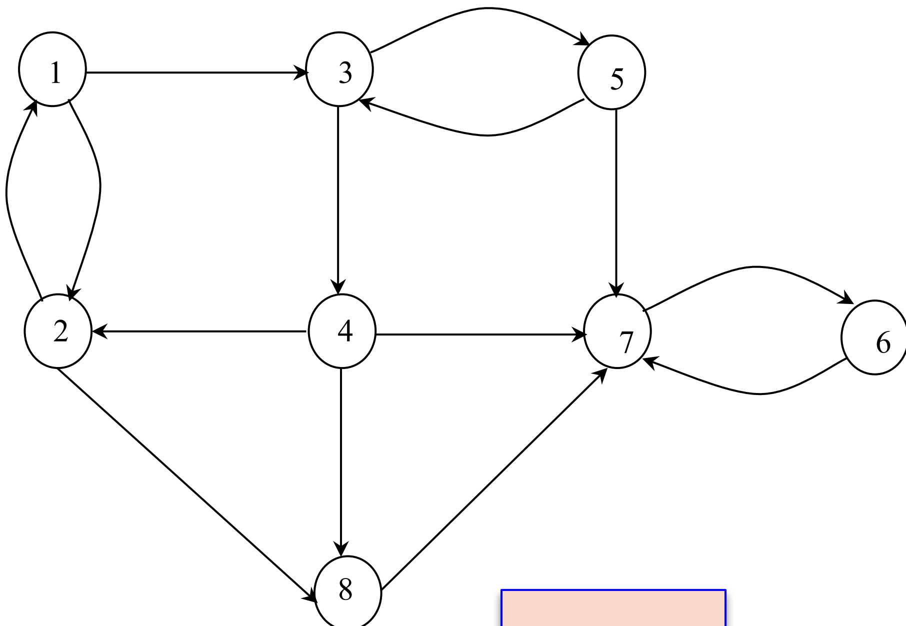
7 không còn trên stack

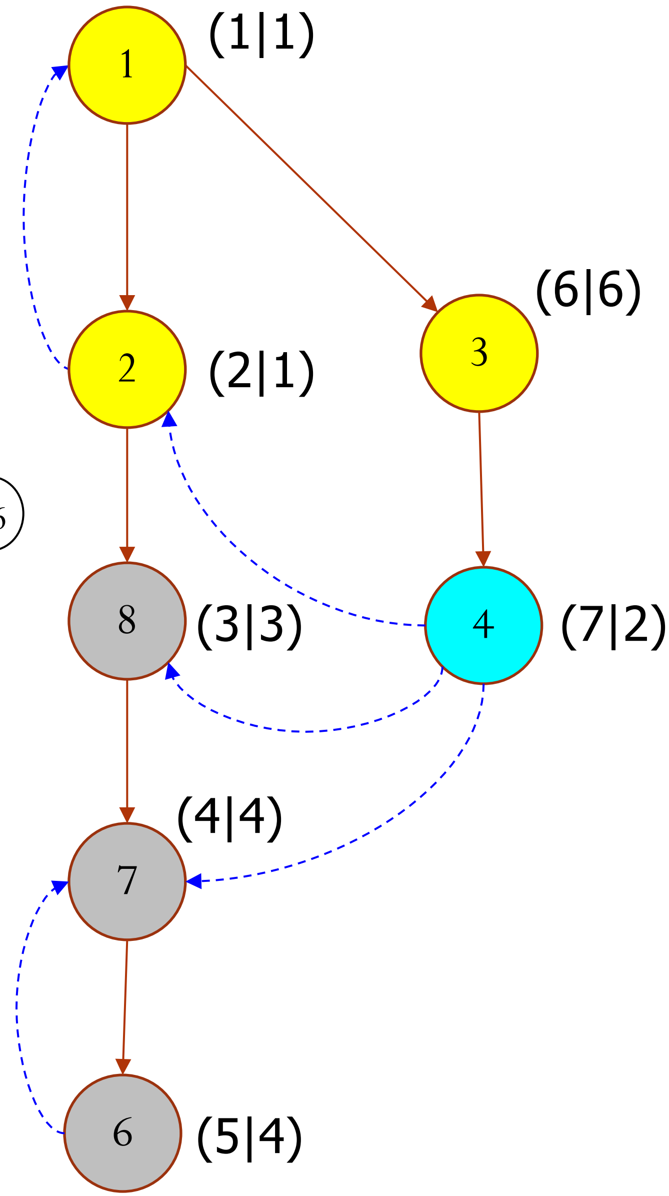




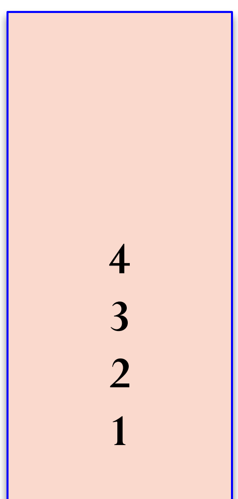
4
3
2
1

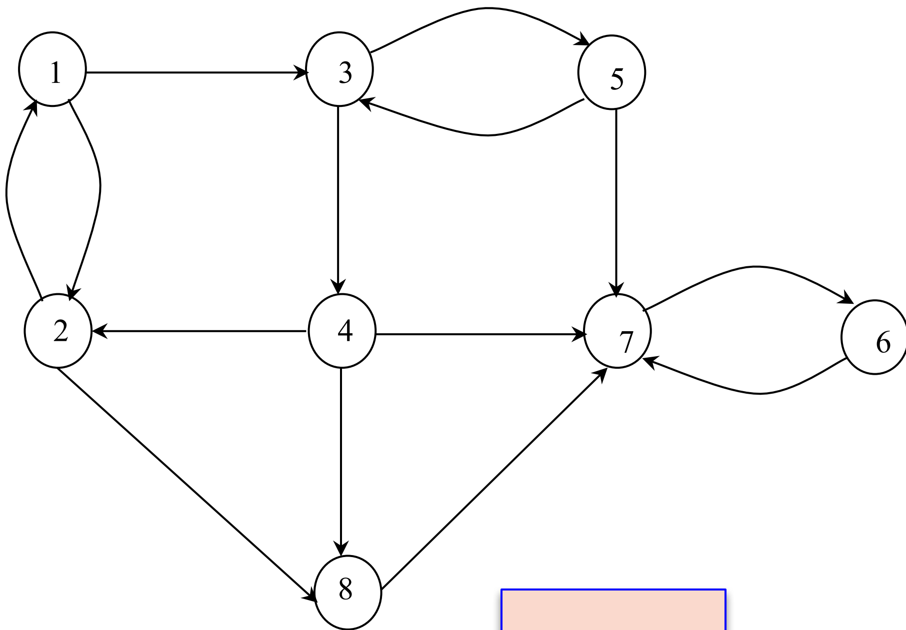
8 không còn trên stack

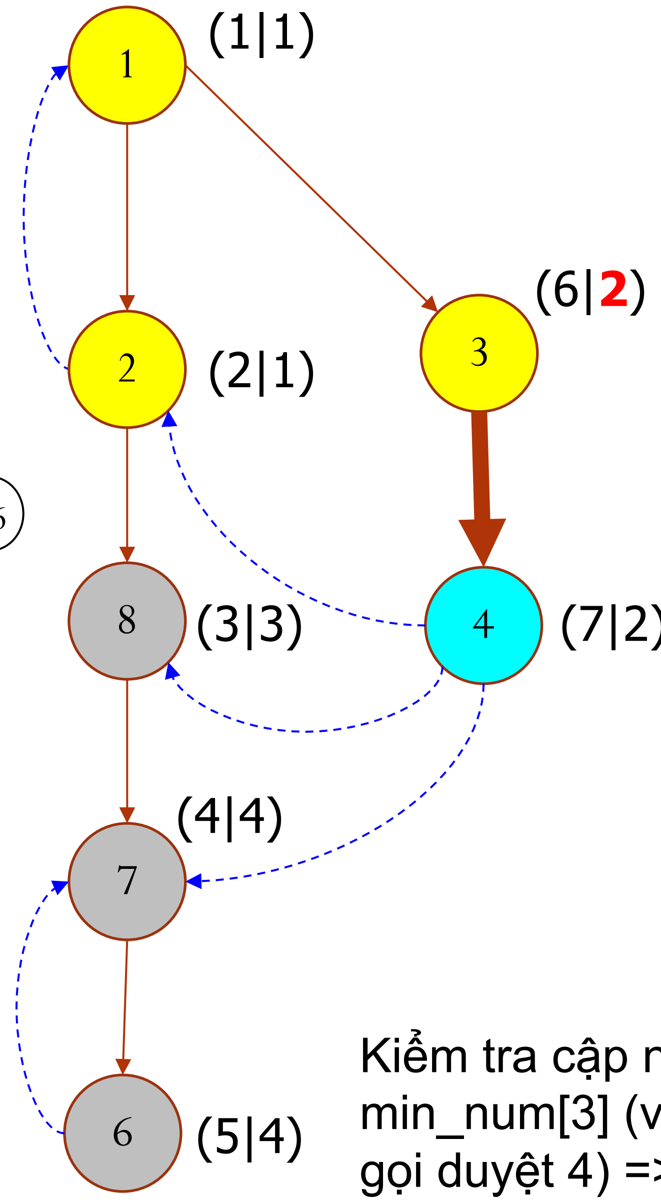




4 đã được duyệt xong và có 7 != 2



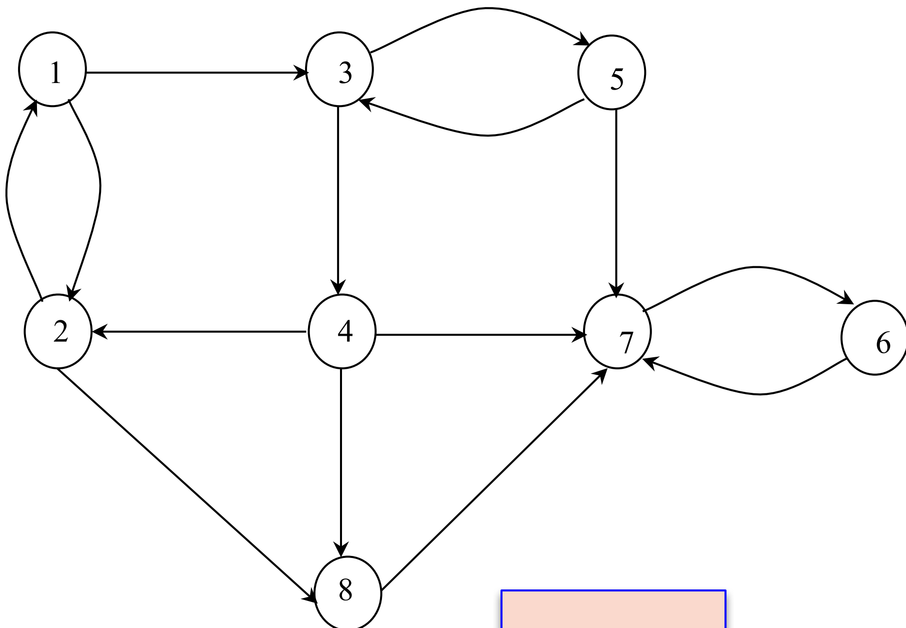


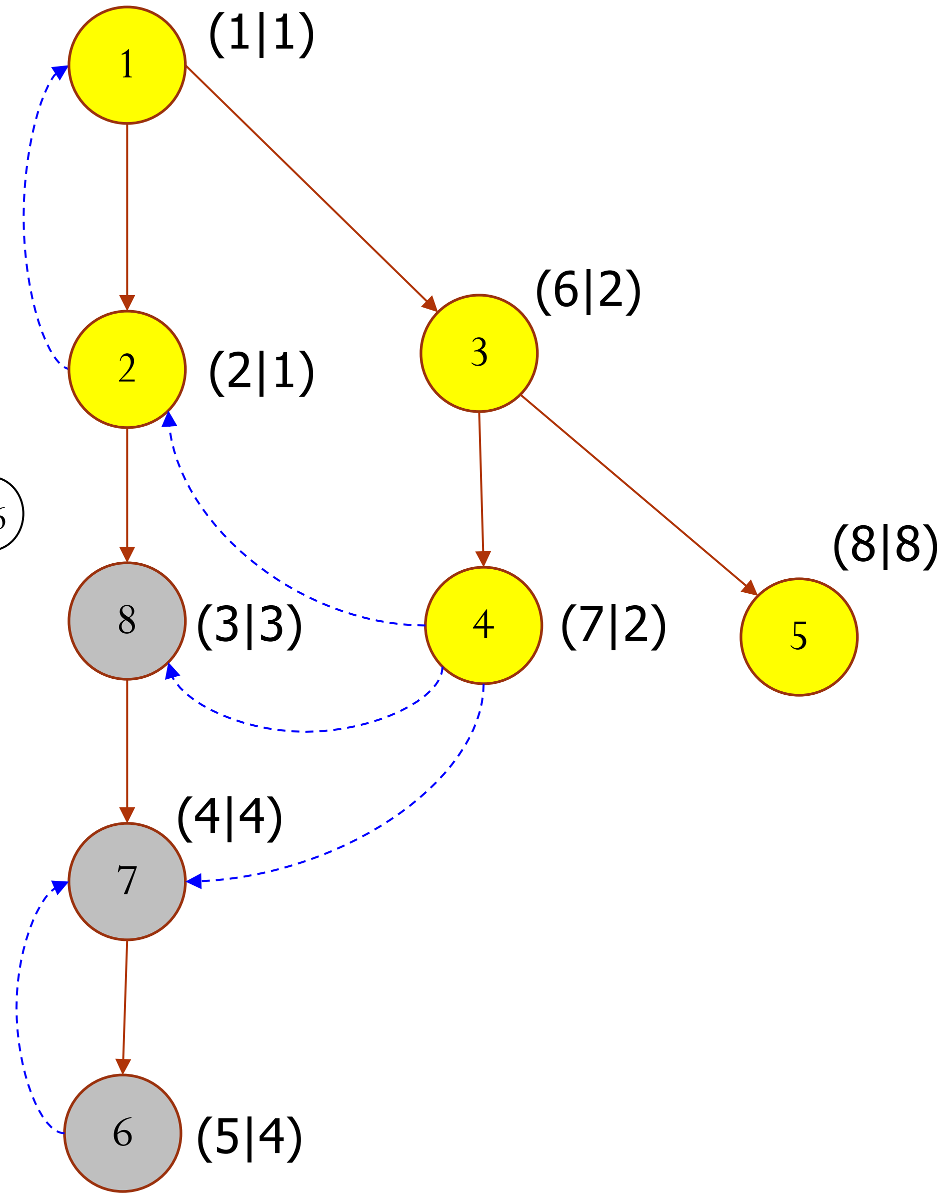


4 đã được duyệt xong và có 7 != 2

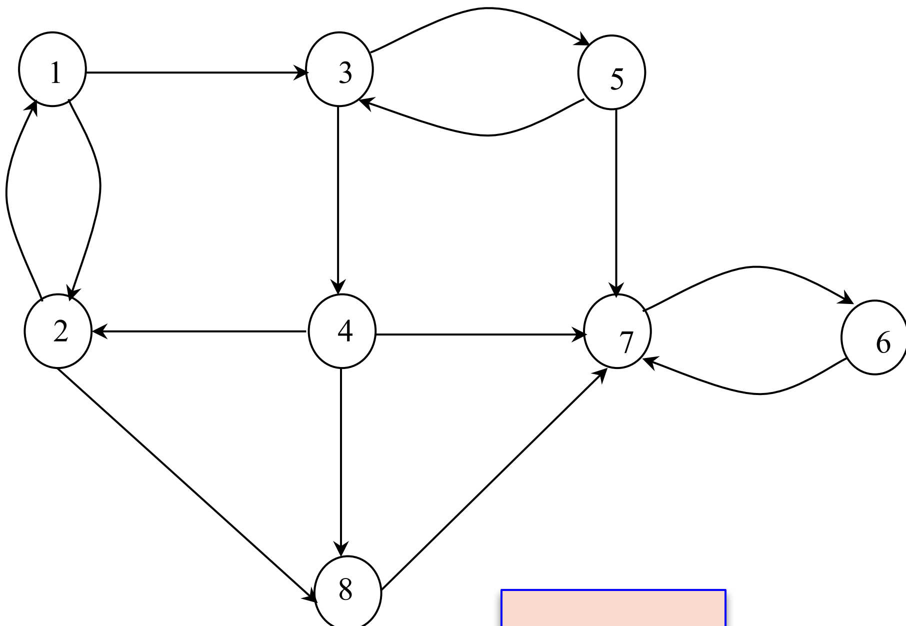
4
3
2
1

Kiểm tra cập nhật lại min_num[3] (vì lúc nãy 3 gọi duyệt 4) => **Cập nhật lại min_num[3] = 2**

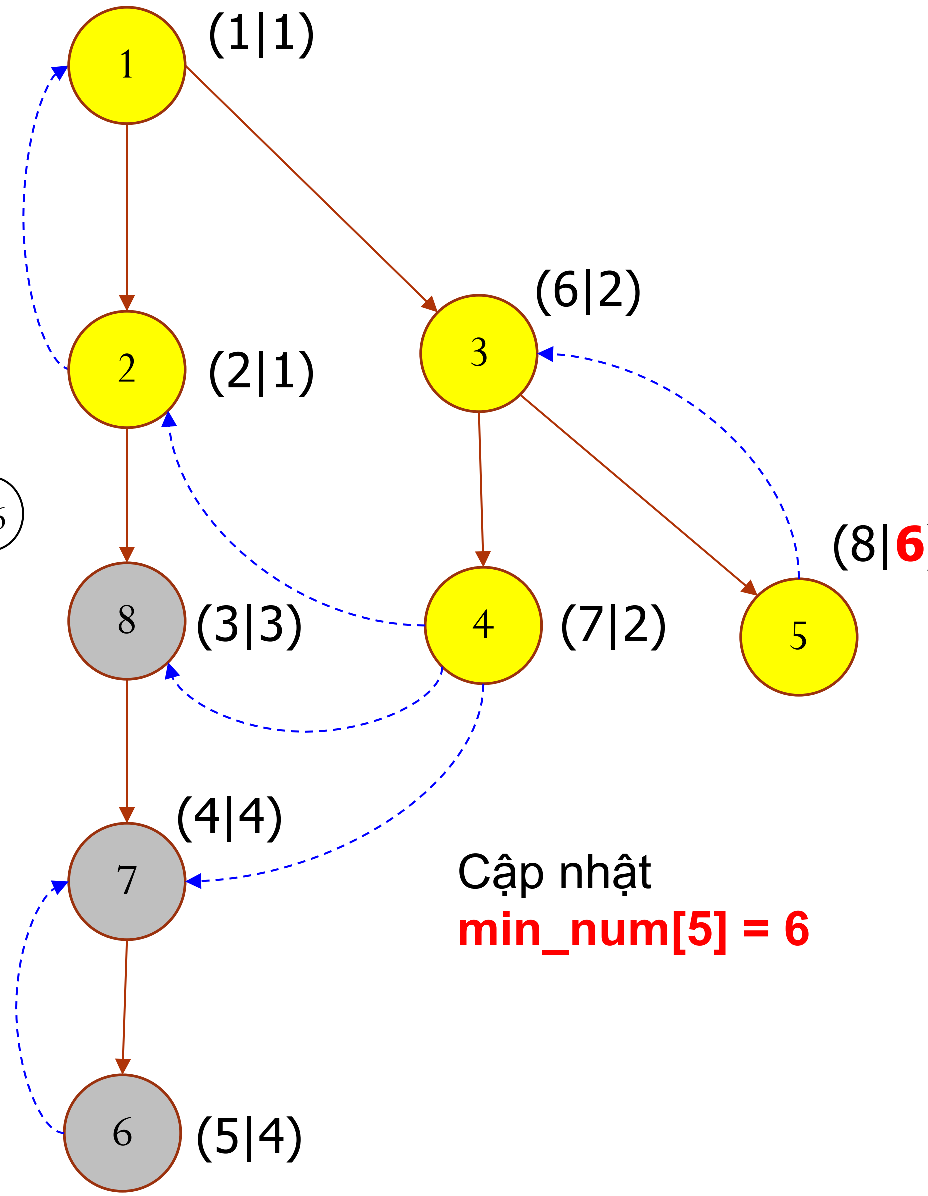




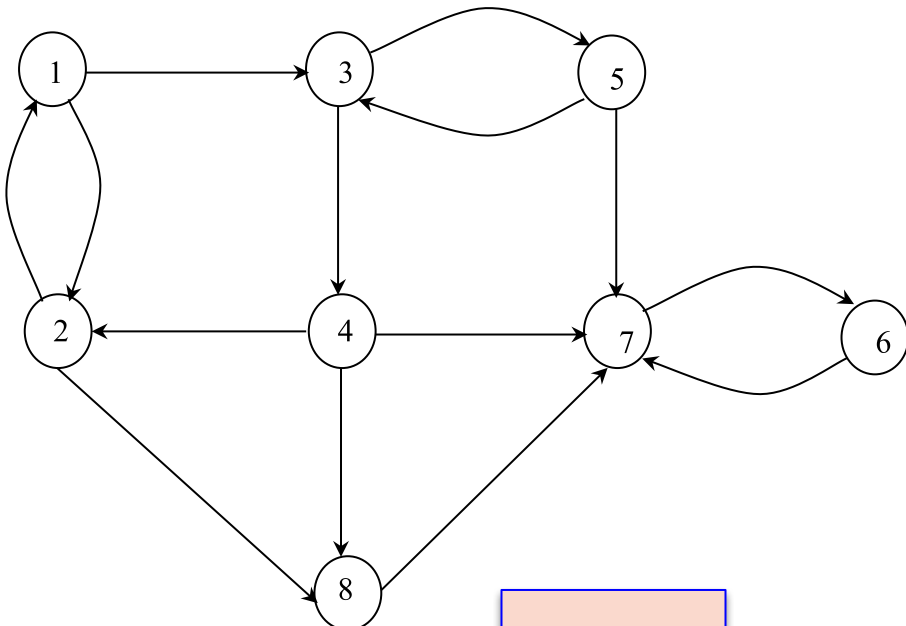
5
4
3
2
1



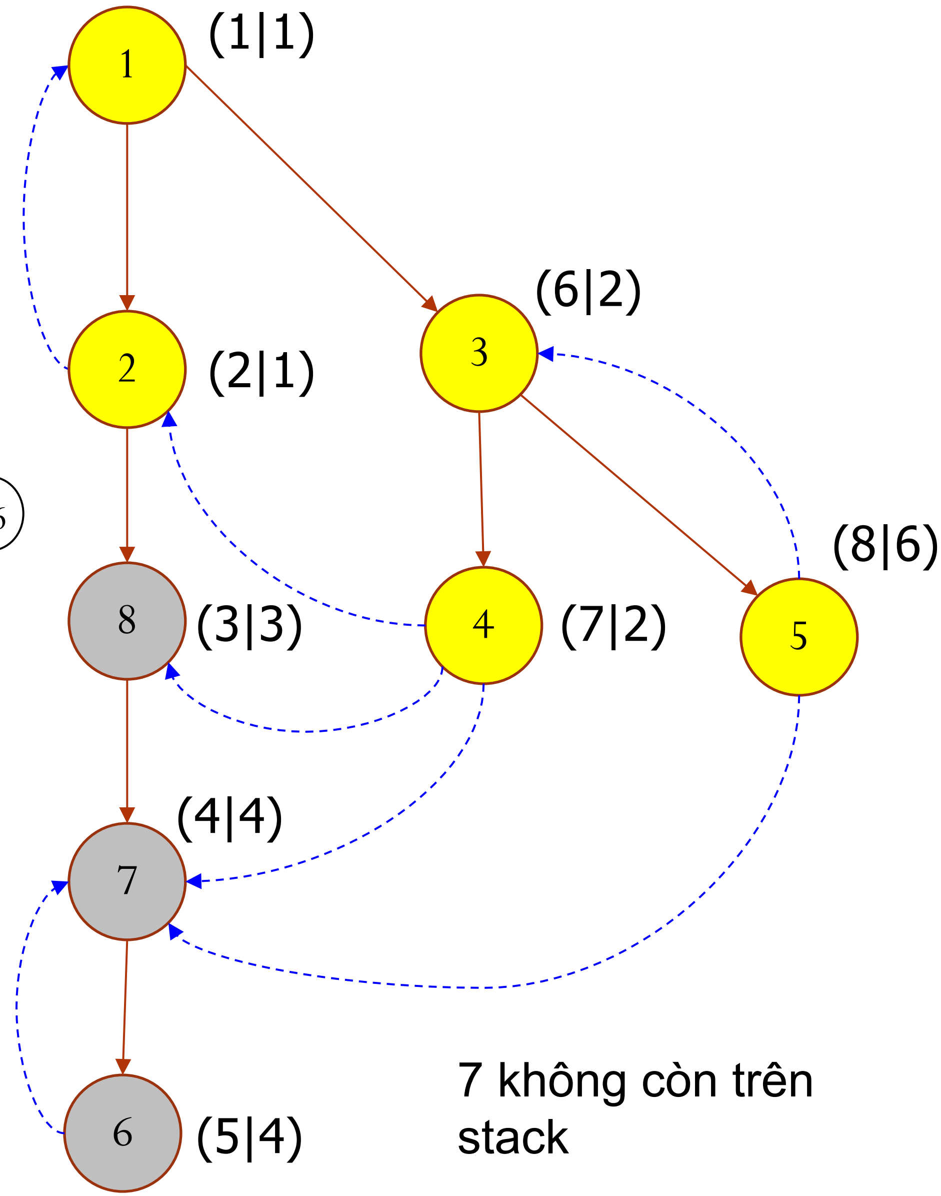
5
4
3
2
1



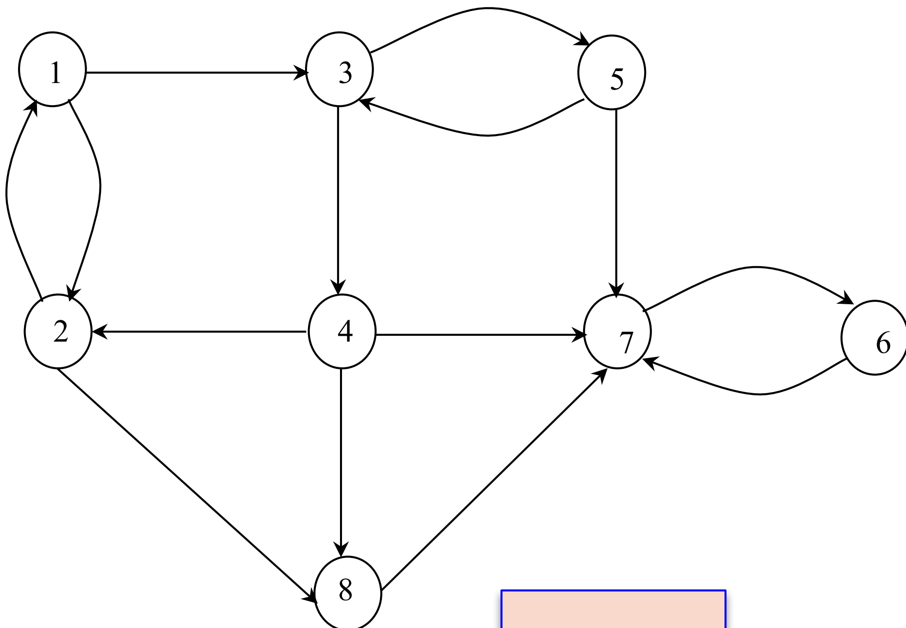
Cập nhật
min_num[5] = 6

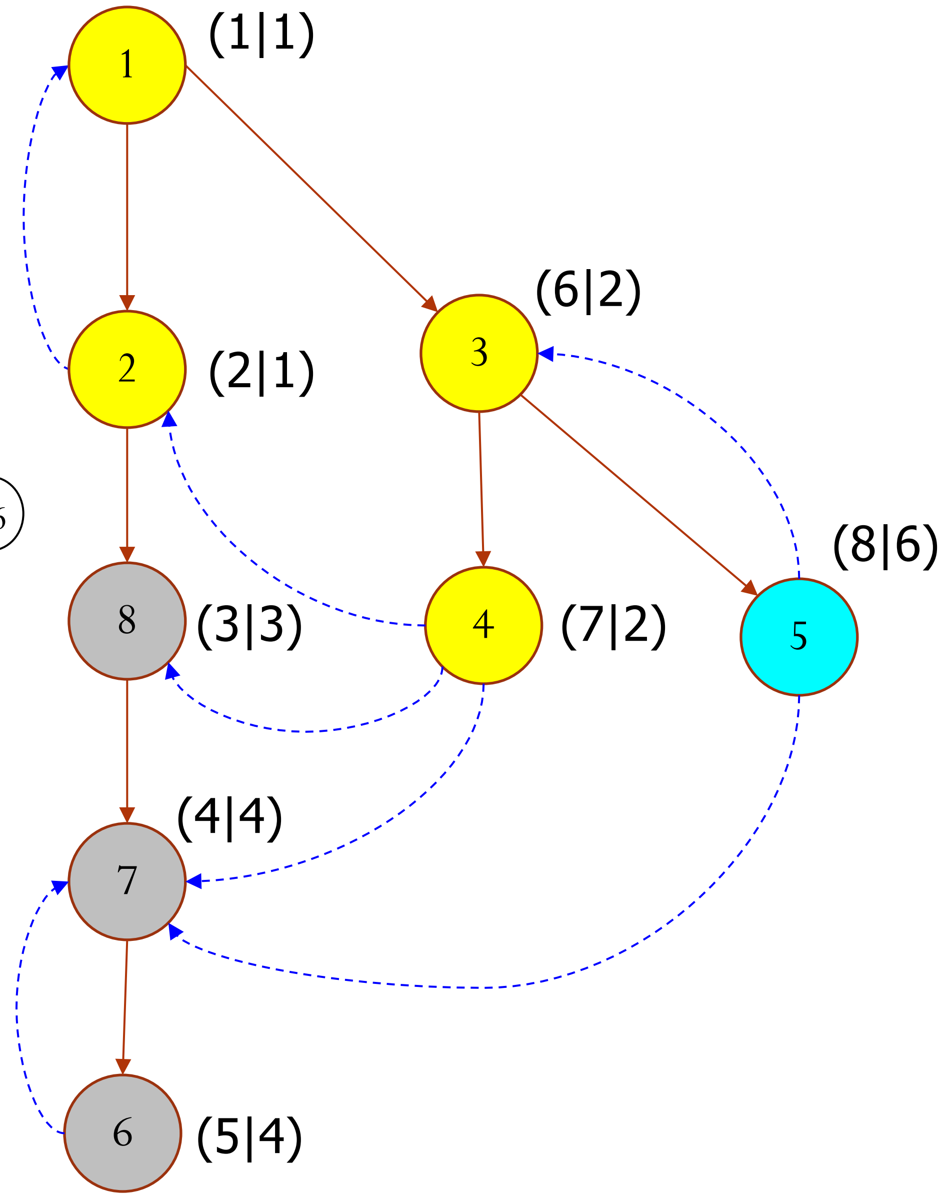


5
4
3
2
1



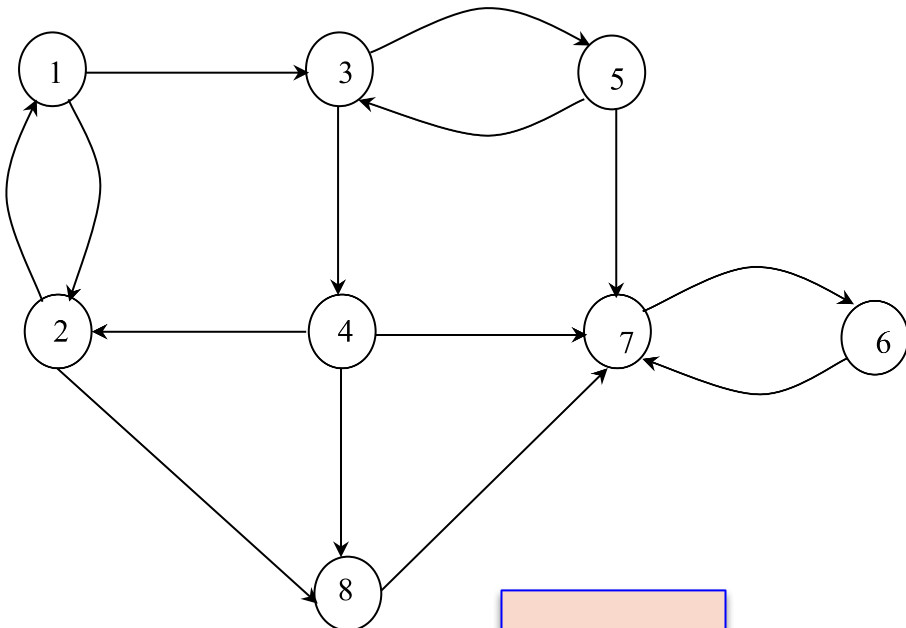
7 không còn trên stack





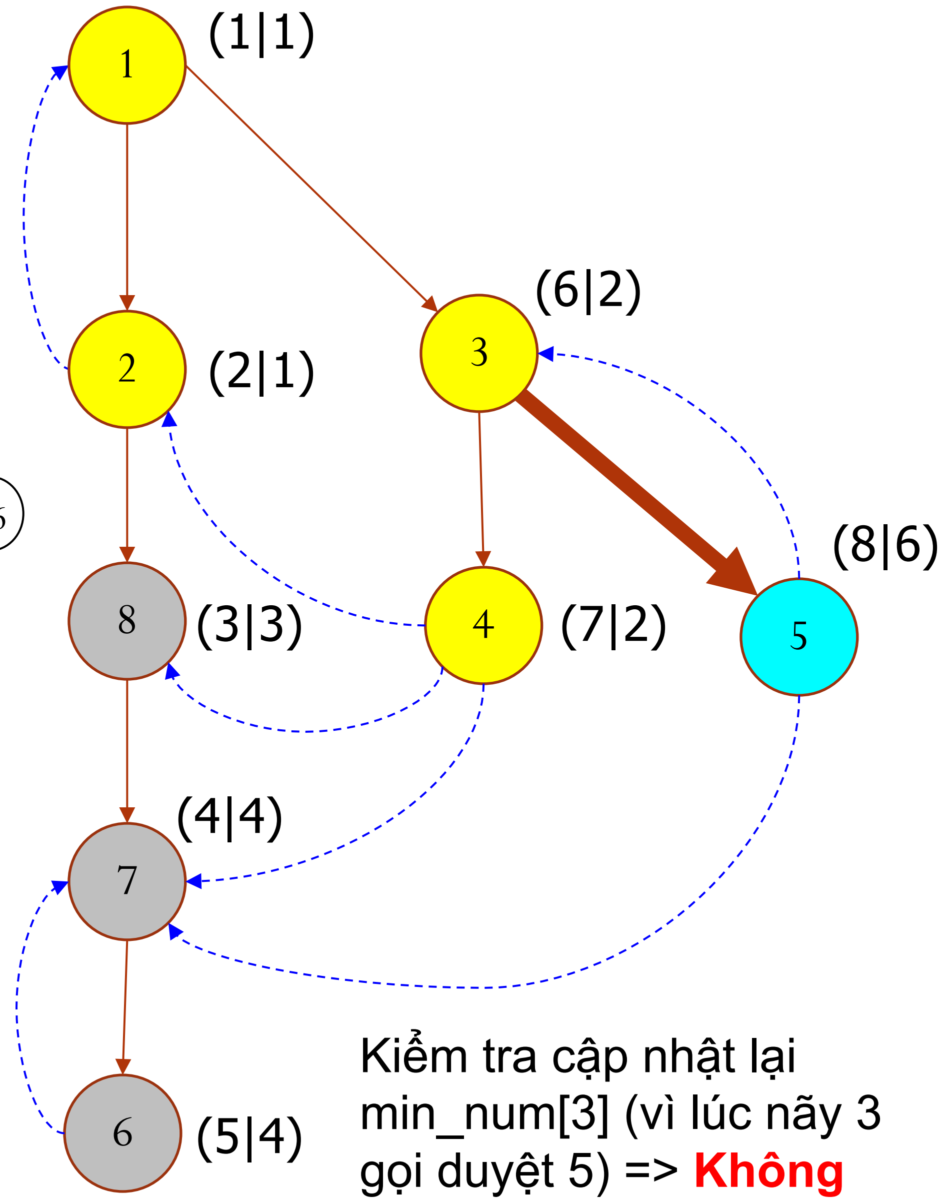
5 đã được duyệt xong và có 8 != 6

5
4
3
2
1

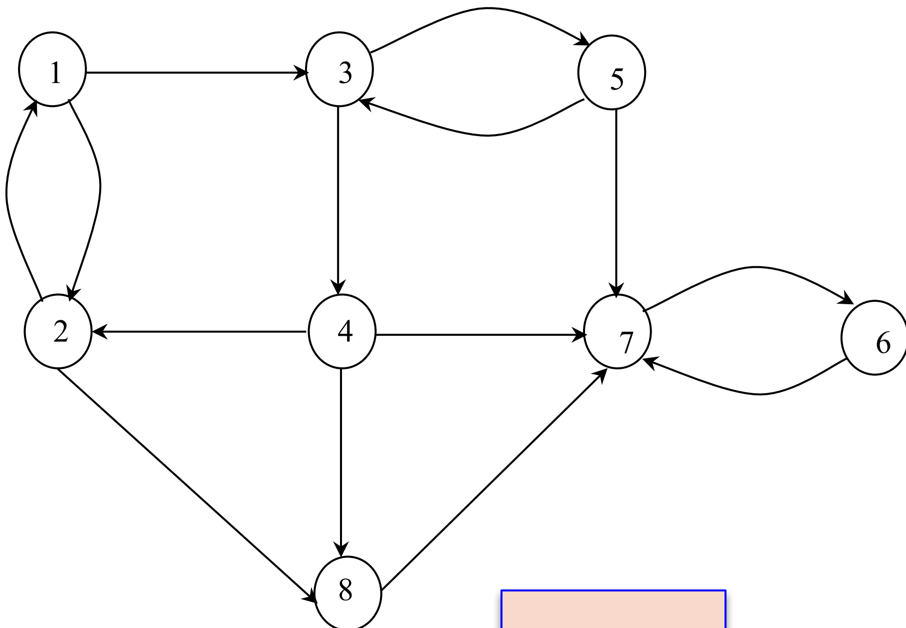


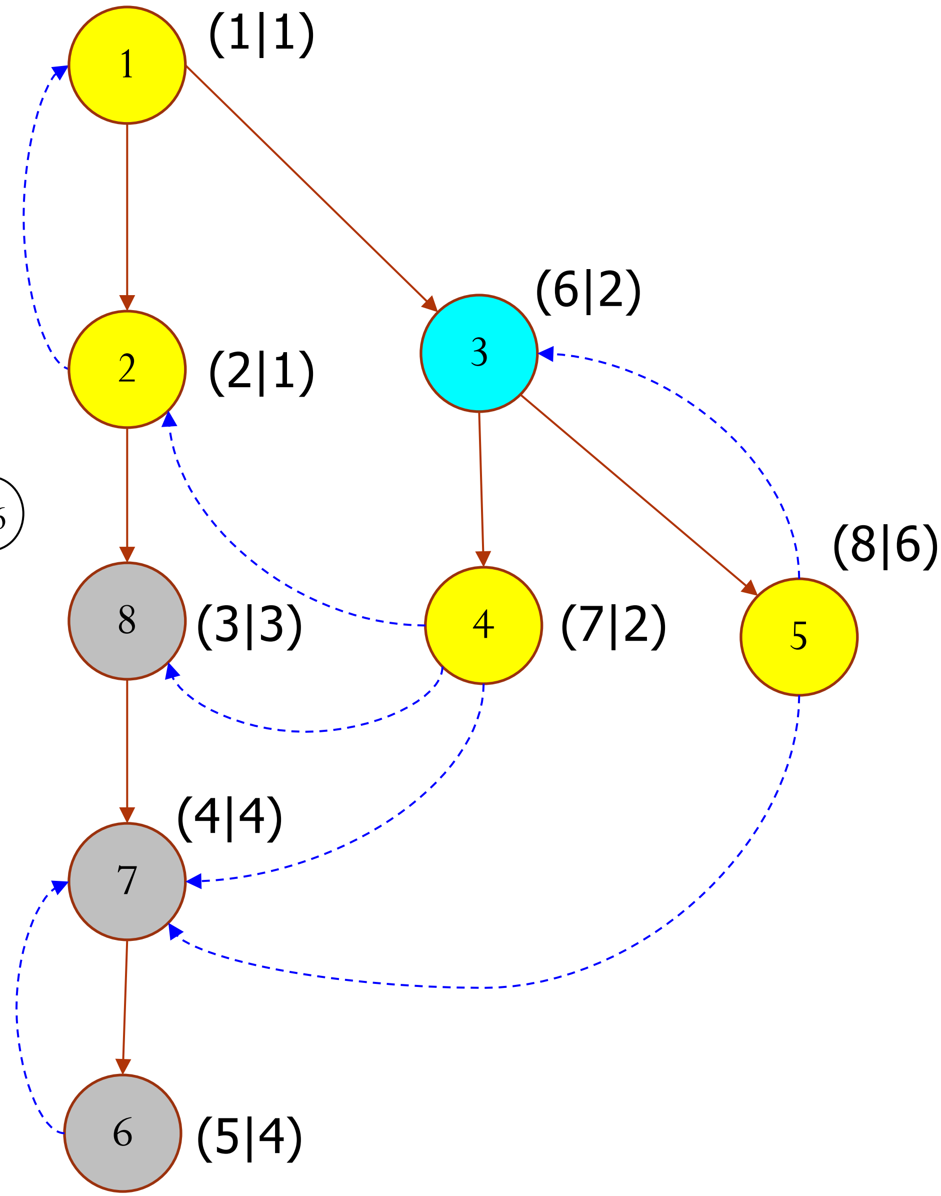
5 đã được duyệt xong và có 8 != 6

5
4
3
2
1



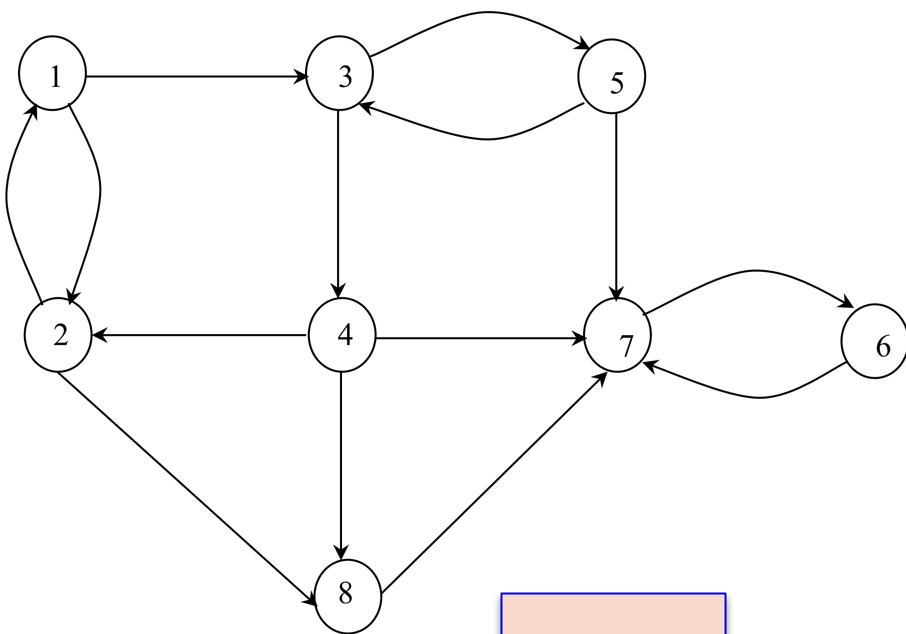
Kiểm tra cập nhật lại min_num[3] (vì lúc nãy 3 gọi duyệt 5) => **Không cần CN vì 2 <= 6**

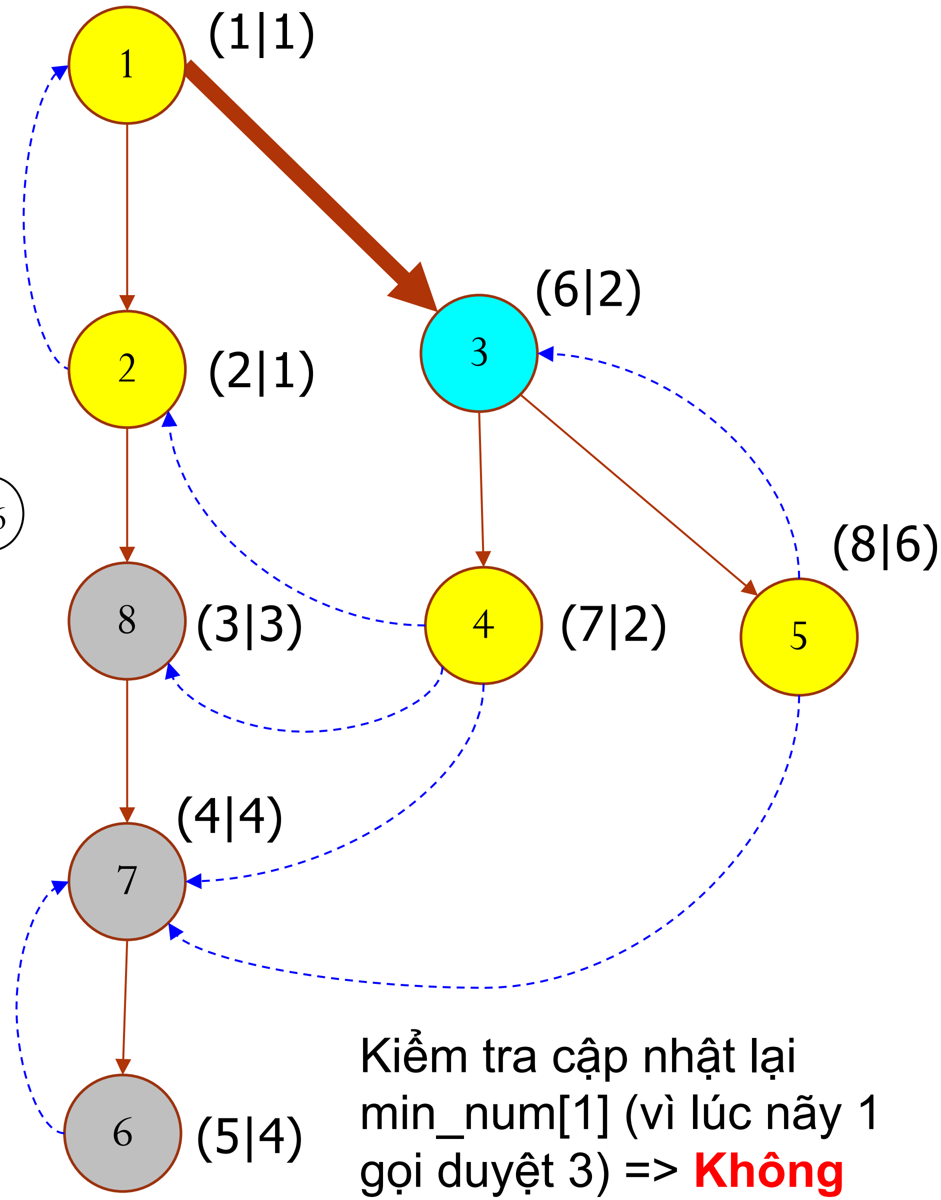




3 đã được duyệt
xong và có 6 != 2

5
4
3
2
1

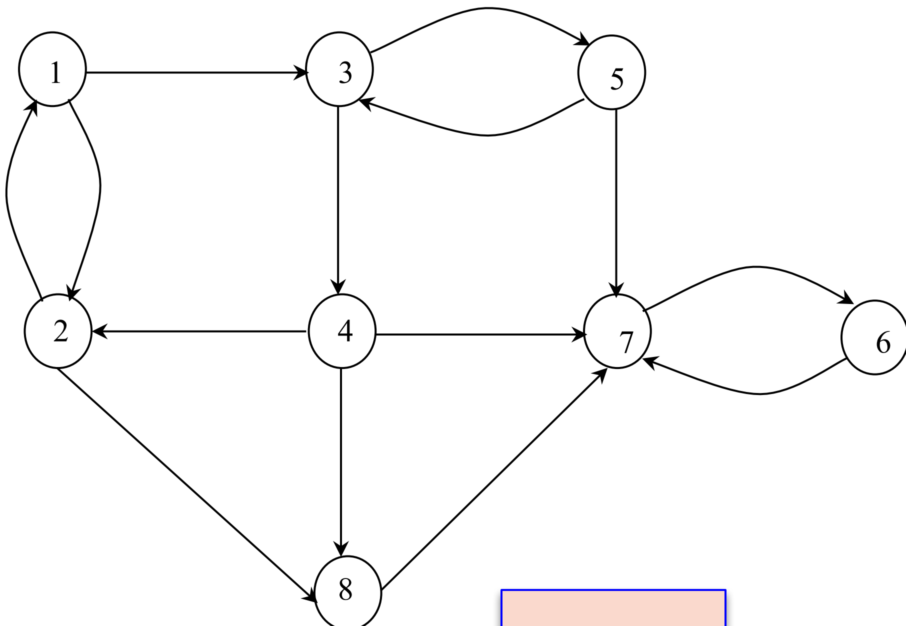


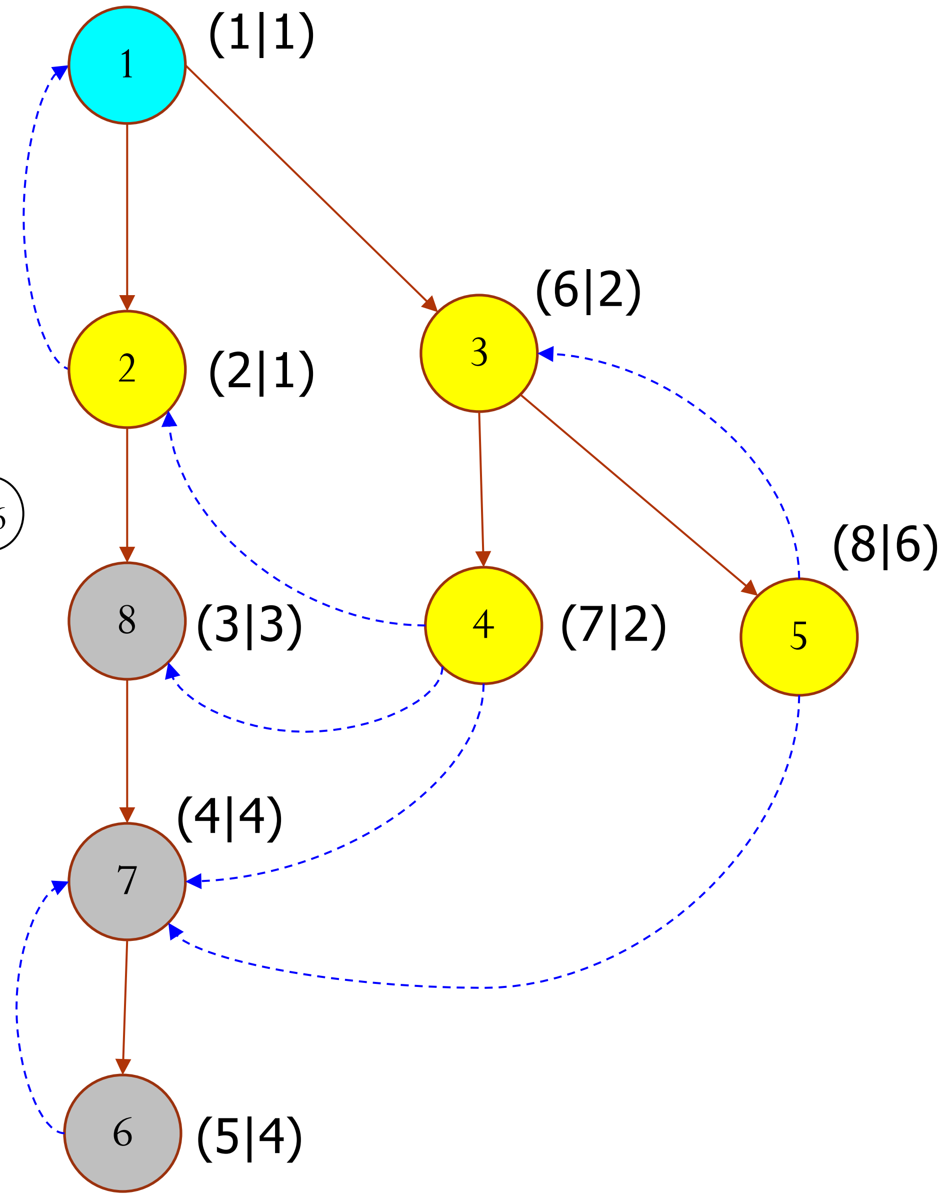


3 đã được duyệt xong và có 6 != 2

5
4
3
2
1

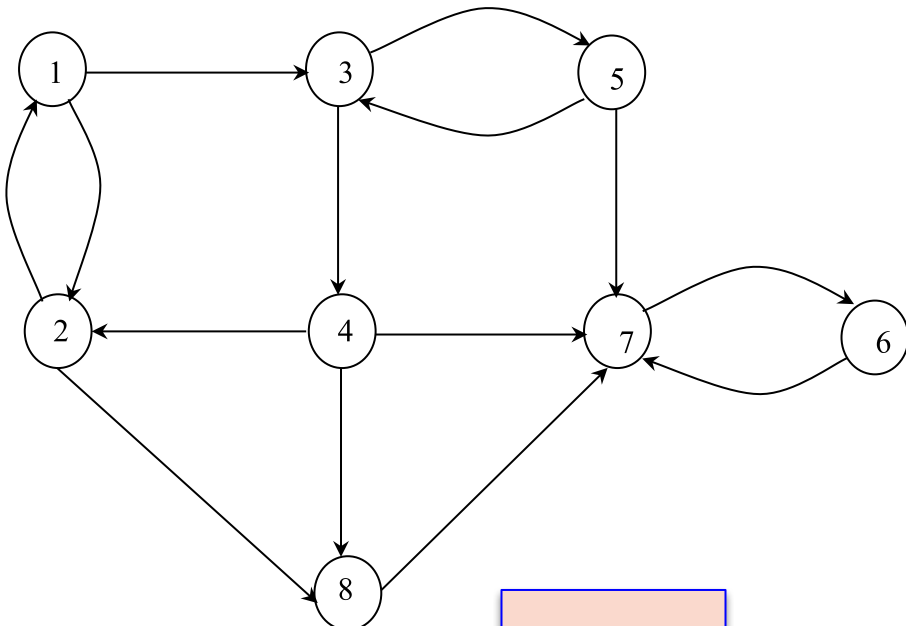
Kiểm tra cập nhật lại min_num[1] (vì lúc nãy 1 gọi duyệt 3) => **Không cần CN vì 1 <= 2**

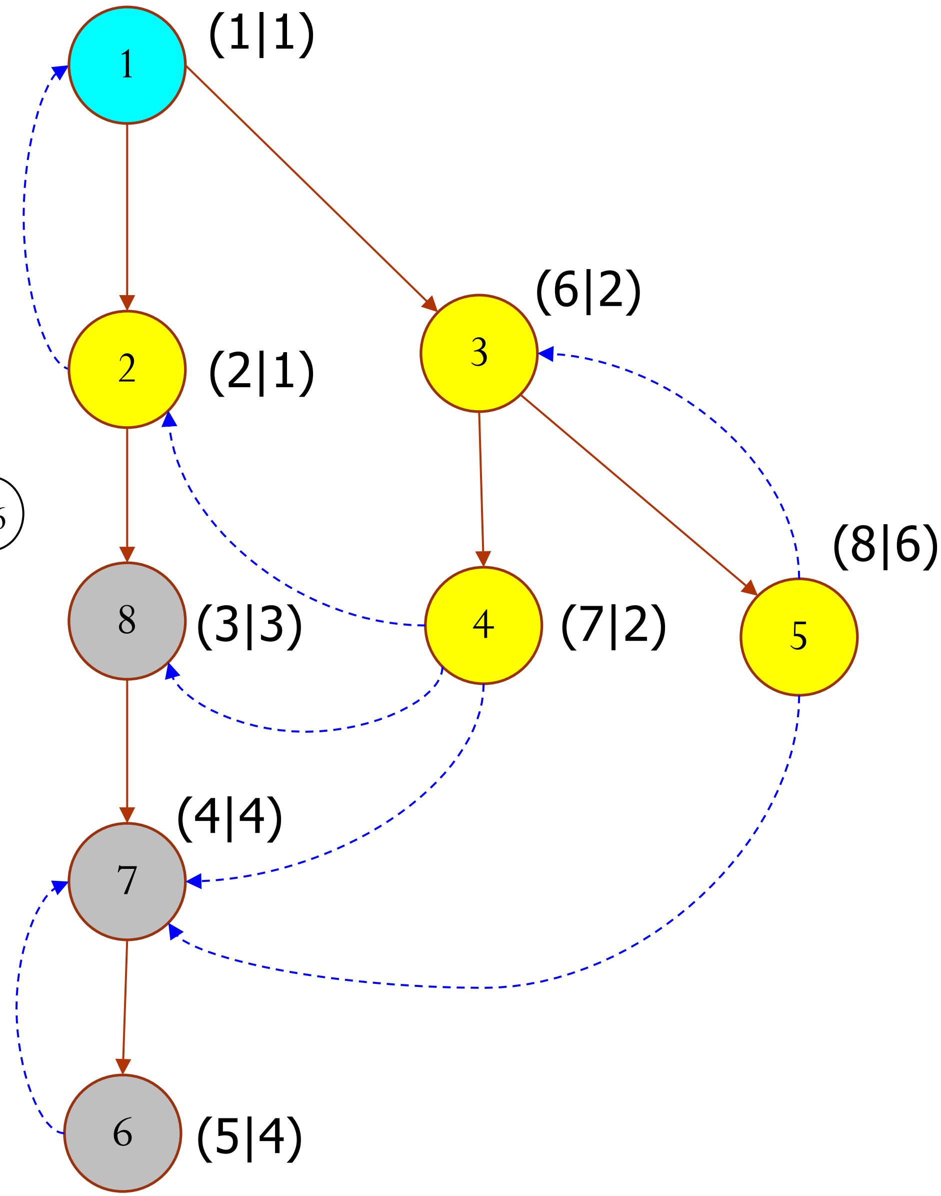




1 đã được duyệt xong và có 1 = 1

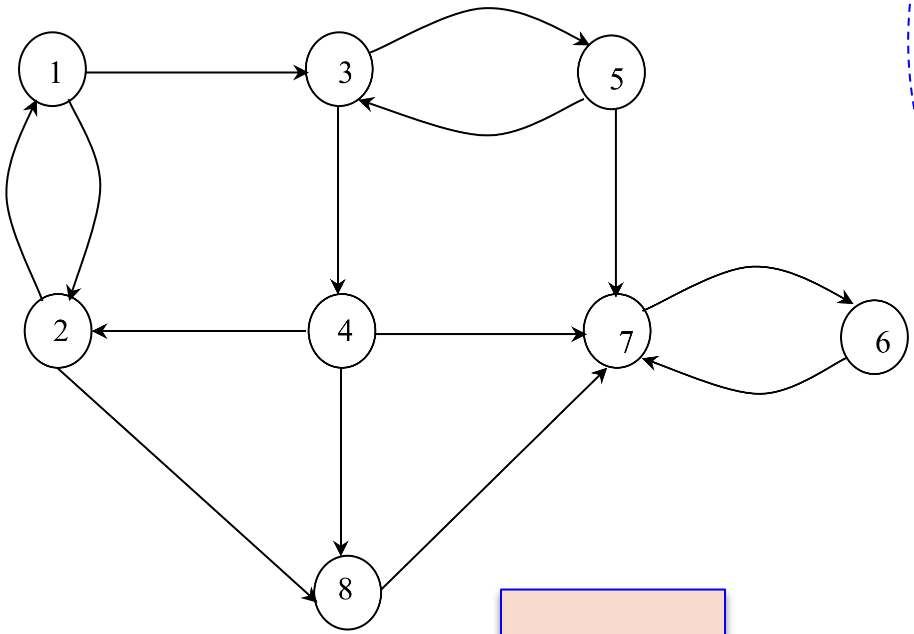
5
4
3
2
1

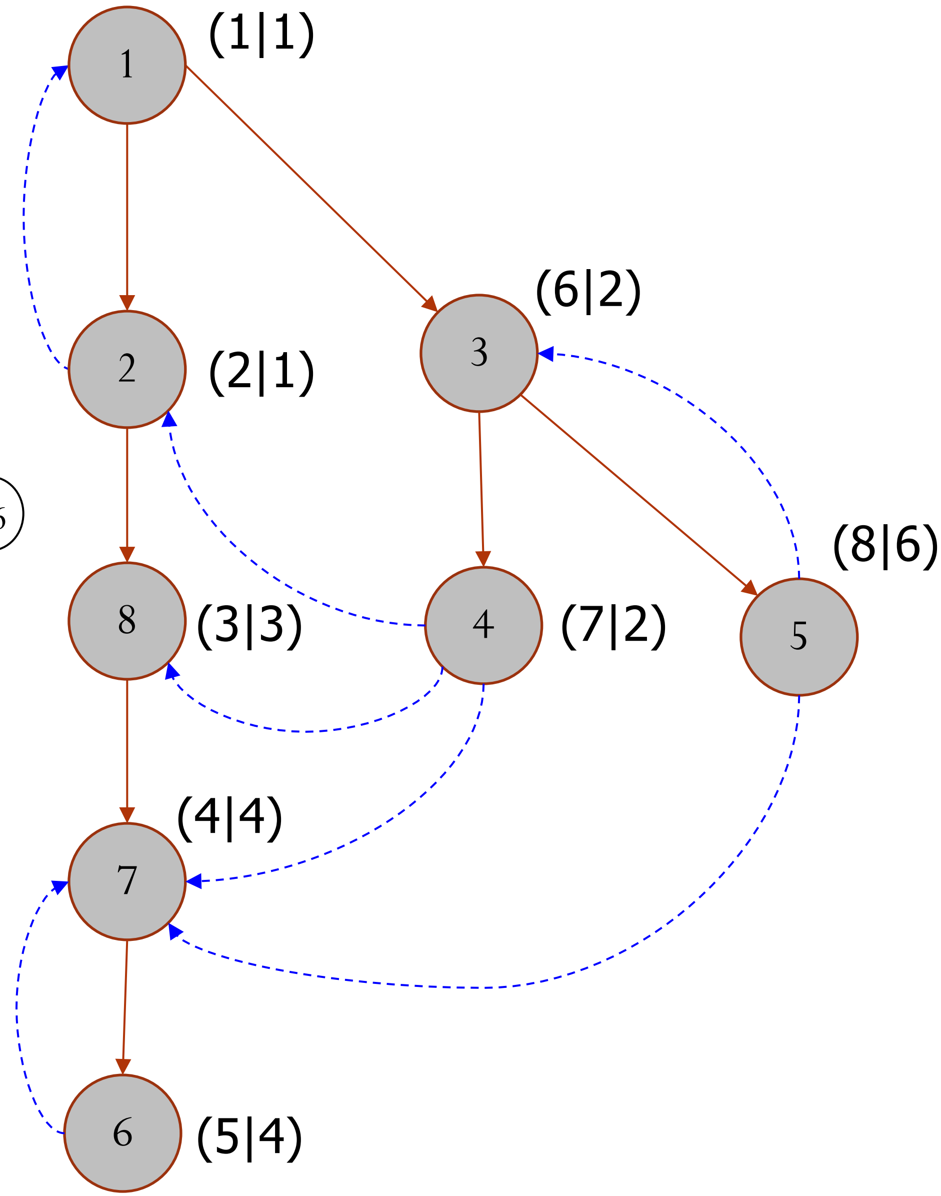


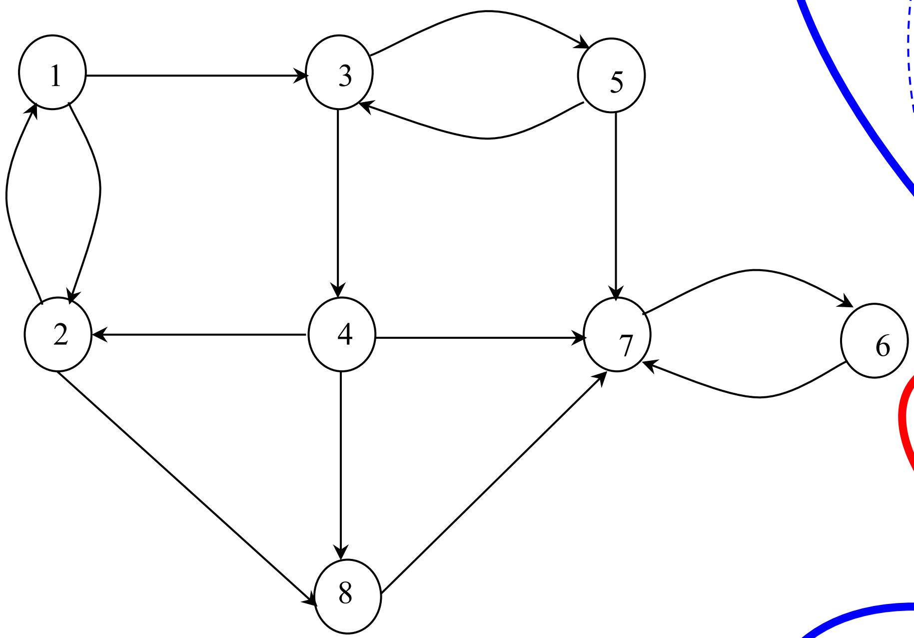


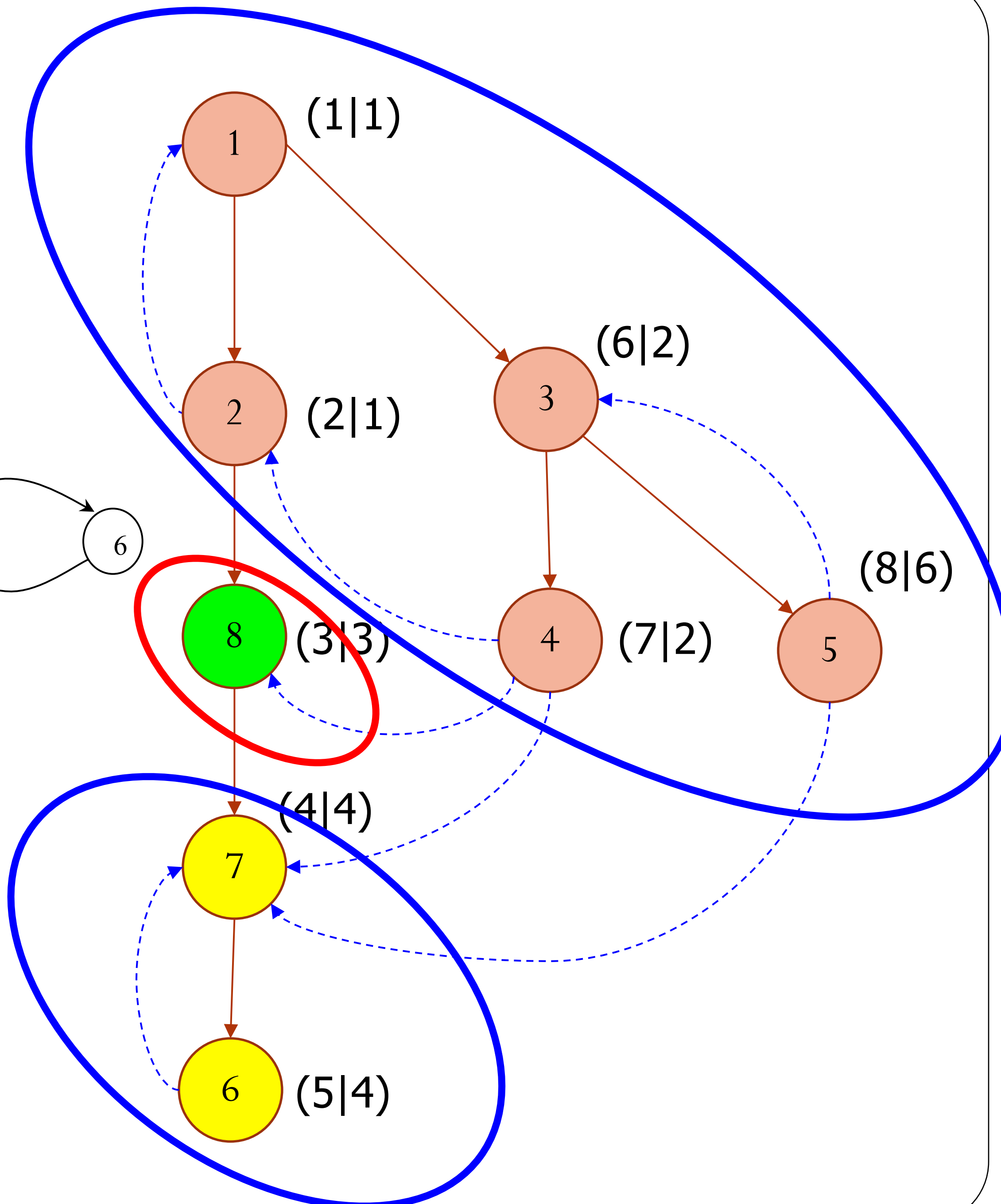
1 đã được duyệt xong và có 1 = 1 Loại bỏ 5, 4, 3, 2, 1 ra => tìm được BPLT (1,2,3,4,5)

~~5~~
~~4~~
~~3~~
~~2~~
~~1~~


1
3
5
2
4
7
6
8


1
(1|1)
2
(2|1)
3
(6|2)
8
(3|3)
4
(7|2)
5
(8|6)
7
(4|4)
6
(5|4)


1
3
5
2
4
7
6
8


1
(1|1)
2
(2|1)
3
(6|2)
8
(3|3)
4
(7|2)
5
(8|6)
7
(4|4)
6
(5|4)