HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

THUẬT TOÁN ỨNG DỤNG

Duyệt theo chiều sâu (DFS) và Ứng dụng

ONE LOVE. ONE FUTURE.

NỘI DUNG

- **Tìm kiếm theo chiều sâu (DFS)**
- Cây DFS và cấu trúc Num, Low
- Tìm cạnh cầu (bridges)
- Tìm đỉnh khớp (articulation)
- Tìm thành phần liên thông mạnh (strongly connected component)

Tìm kiếm theo chiều sâu

- Tìm kiếm theo chiều sâu (Depth First Search) là một kỹ thuật duyệt đồ thị cơ bản (thăm mỗi đỉnh và mỗi cạnh của đồ thị).
  - Thuật toán có thể trả lời câu hỏi, có tồn tại đường đi từ đỉnh u tới đỉnh v trên đồ thị G hay không, nếu có chỉ ra một đường đi.
  - Thuật toán không chỉ trả lời có đường đi từ u tới v, mà nó có thể trả lời từ u có thể đi được đến những đỉnh nào khác trên đồ thị G.
- Thứ tự duyệt trong DFS theo cơ chế ***vào sau ra trước*** (LIFO – Last In First Out), và bắt đầu từ đỉnh xuất phát u nào đó.
  - Có thể sử dụng CTDL Đệ quy quy lui hoặc Ngăn xếp để hỗ trợ duyệt
- Độ phức tạp tính toán: $O(|V| + |E|)$, trong đó V là tập đỉnh và E là tập cạnh của đồ thị G, vì mỗi đỉnh và mỗi cạnh của G được thăm một lần.

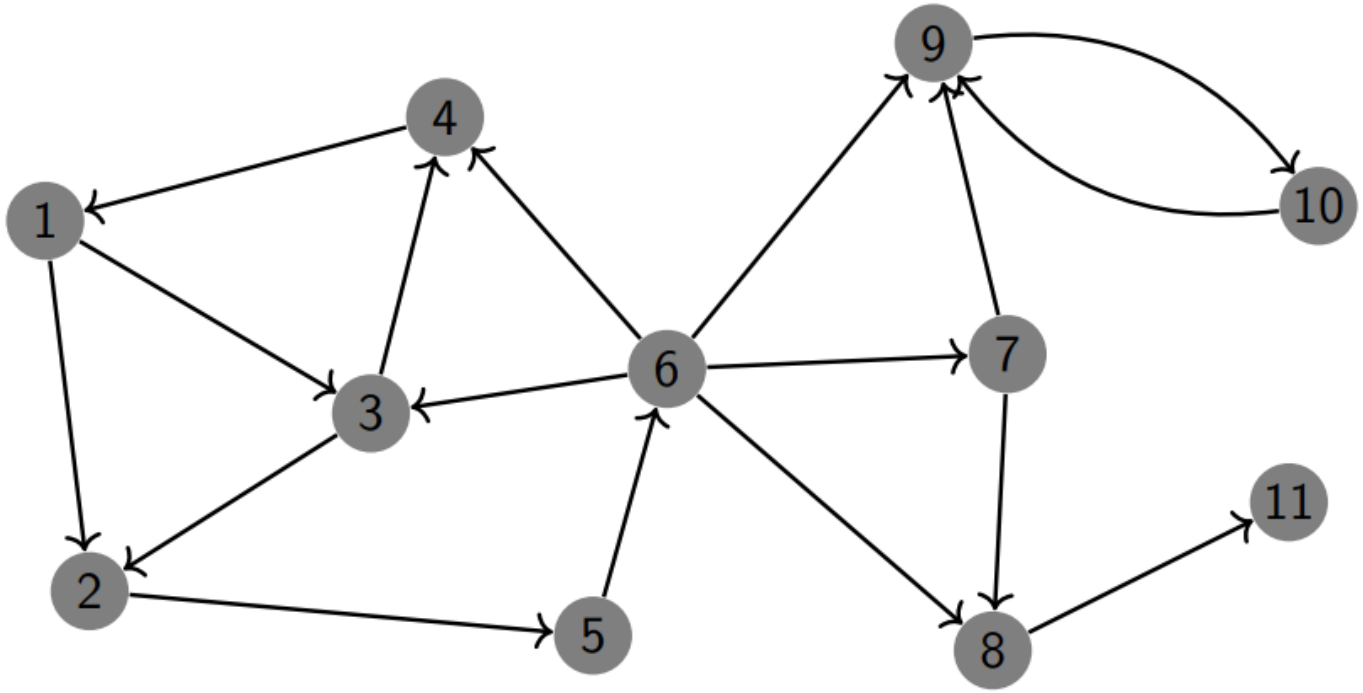
Ý tưởng cài đặt

```
//Main Program
for s ∈ V
        visited[s] ← false
        for s ∈ V
            if (visited[s] == false)
                        DFS(s)
```

```
//DFS using recursive approach
DFS(s)
        visited[s] ← true  // Visist s
        for each v ∈ Adj[s]
            if (visited[v] == false)
                        DFS(v)
```

```
//DFS using Stack
DFS(s)
        S.push(s) // Insert s into the stack S
        While (S is not empty)
            u = S.top()
            S.pop()
            visited[u] ← true  // Visist u
            for each v ∈ Adj[u]
                        if (visited[v] == false)
                                    S.push(v)
```

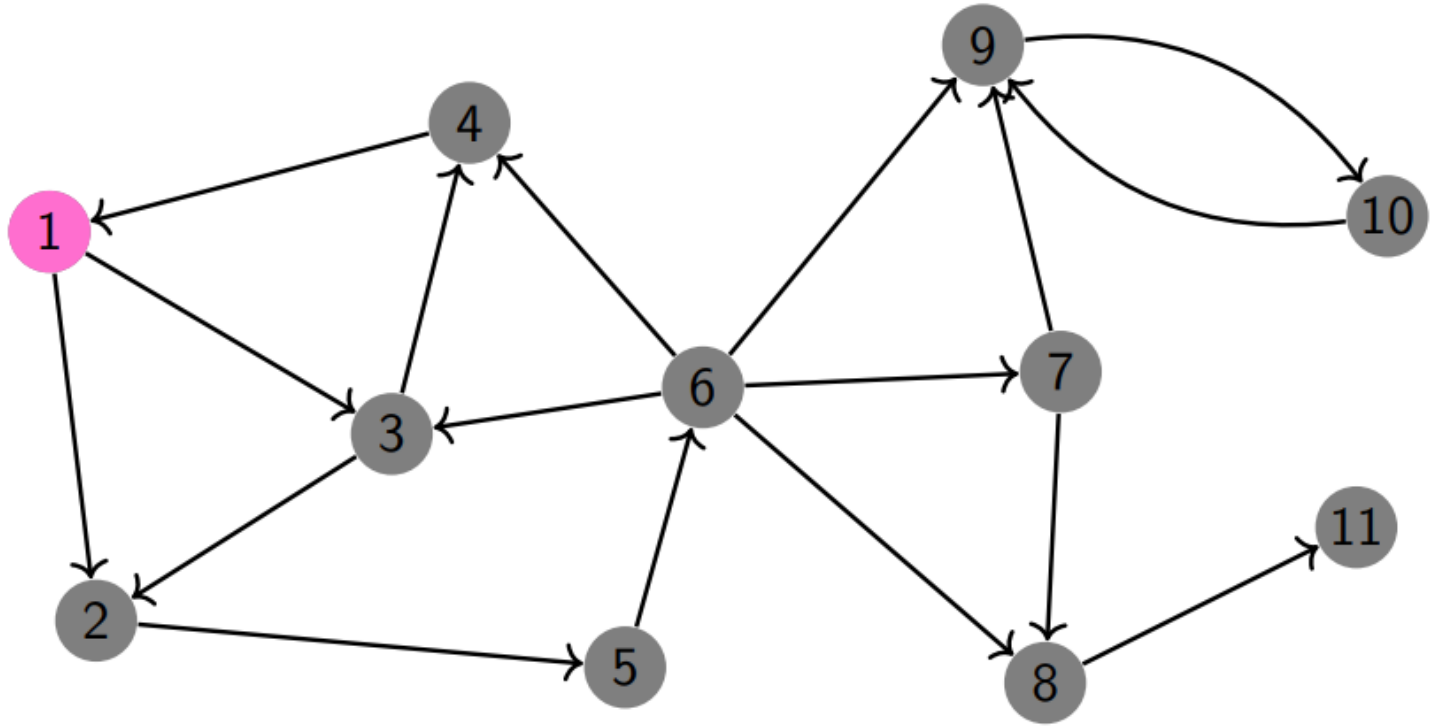
Ví dụ



Stack: |

	1	2	3	4	5	6	7	8	9	10	11
Visited	F	F	F	F	F	F	F	F	F	F	F

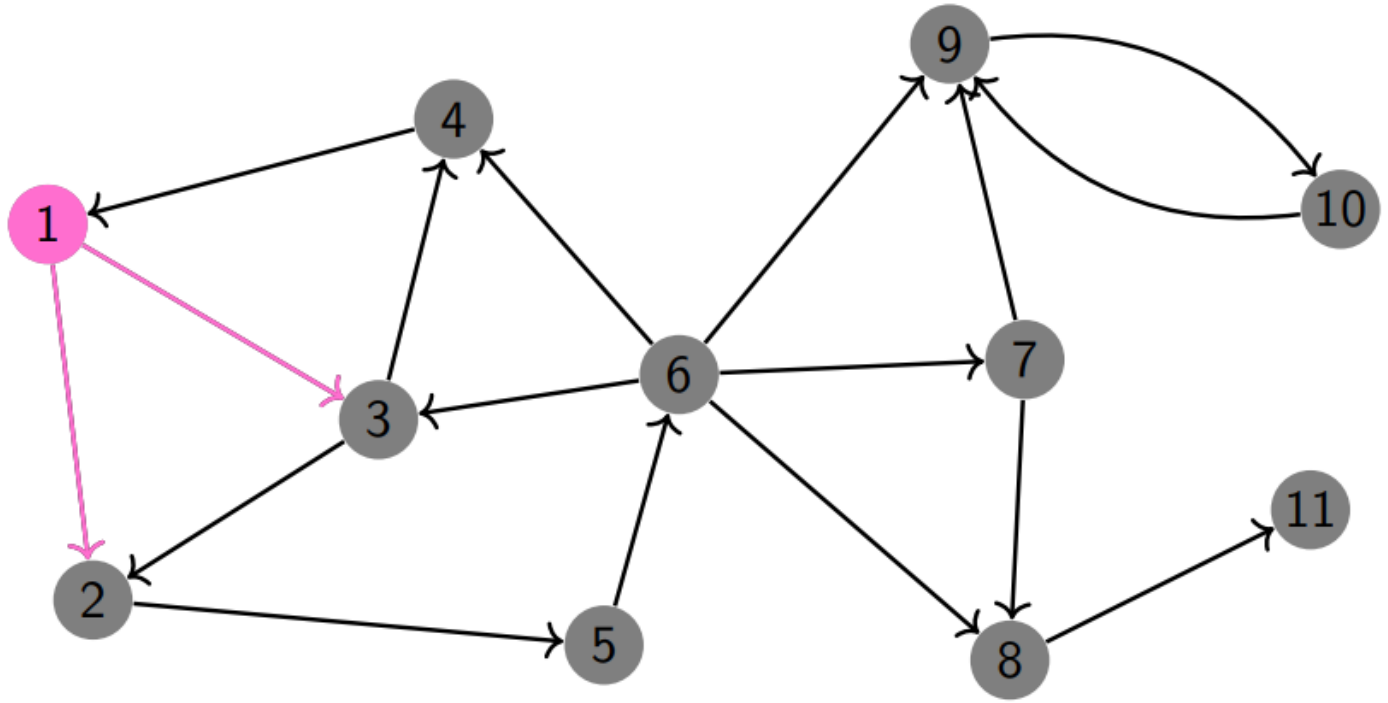
Ví dụ



Stack: 1 |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	F	F	F	F	F	F	F	F	F	F

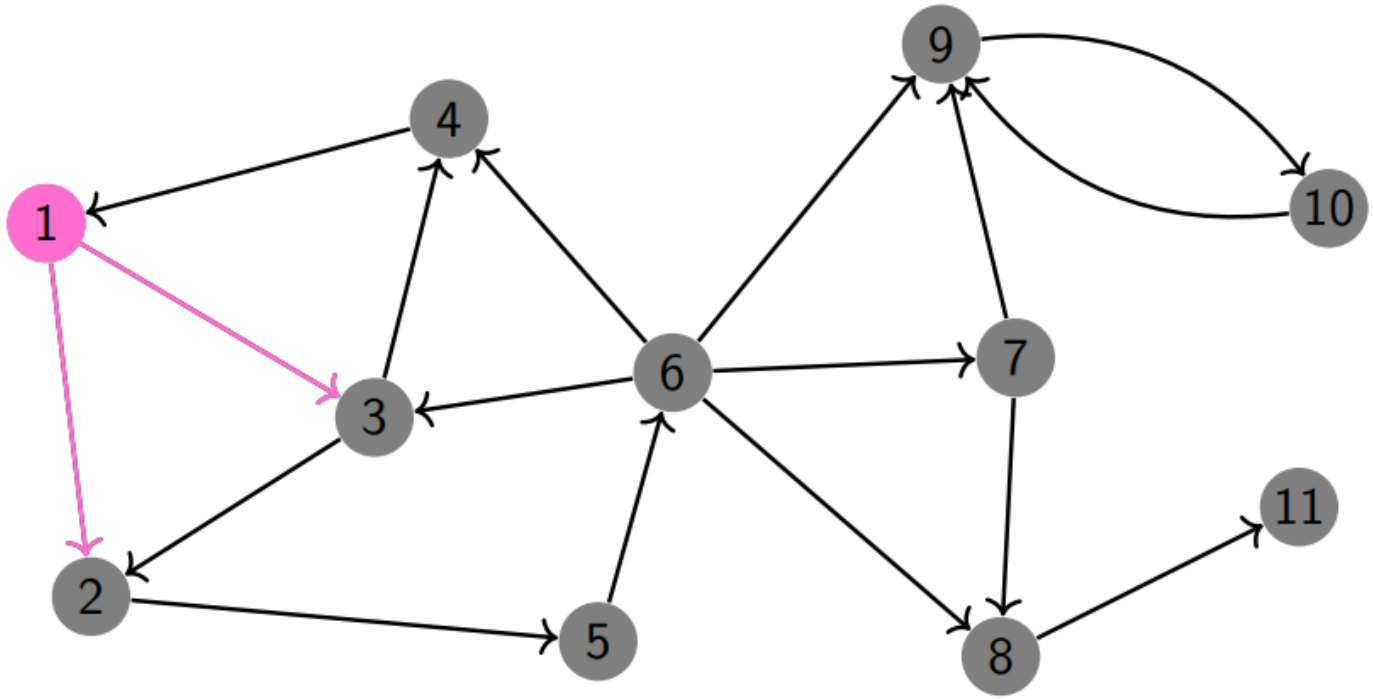
Ví dụ



Stack: 1 |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	F	F	F	F	F	F	F	F	F	F

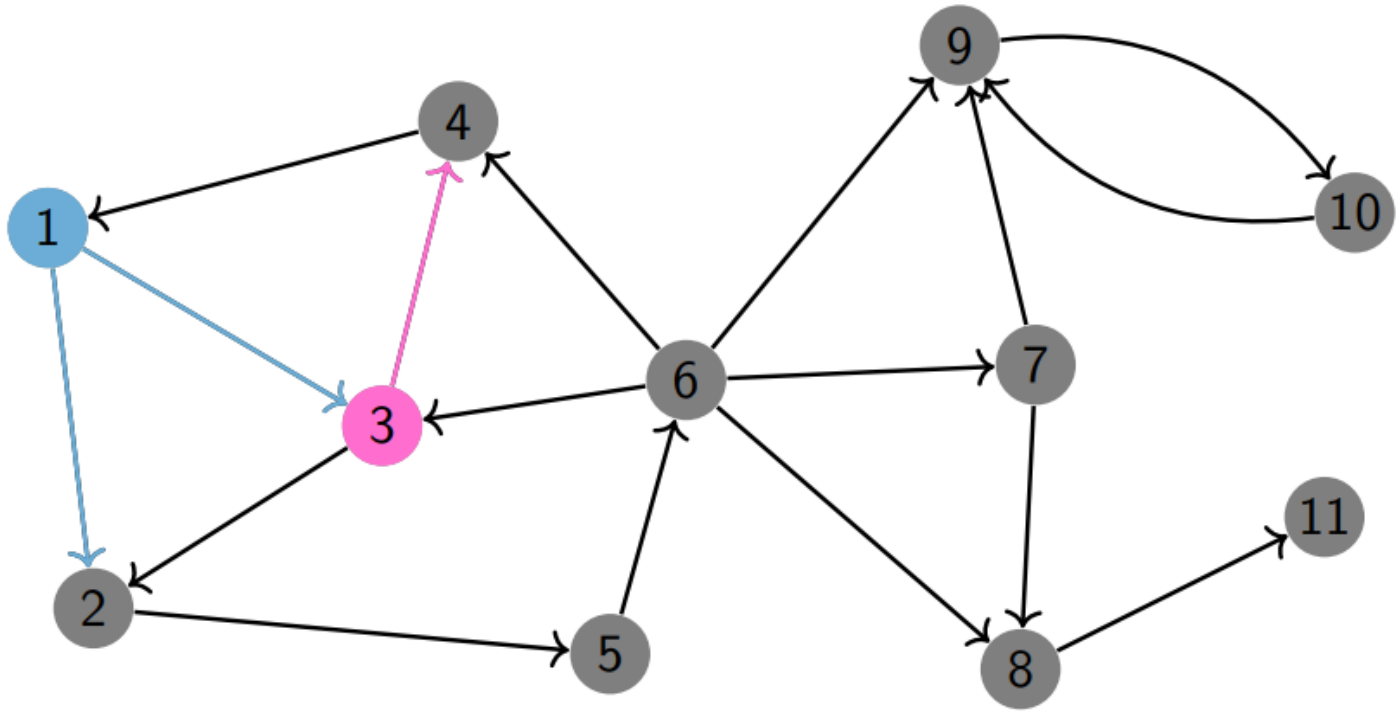
Ví dụ



Stack: 1 | 3 2

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	F	F	F	F	F	F	F	F

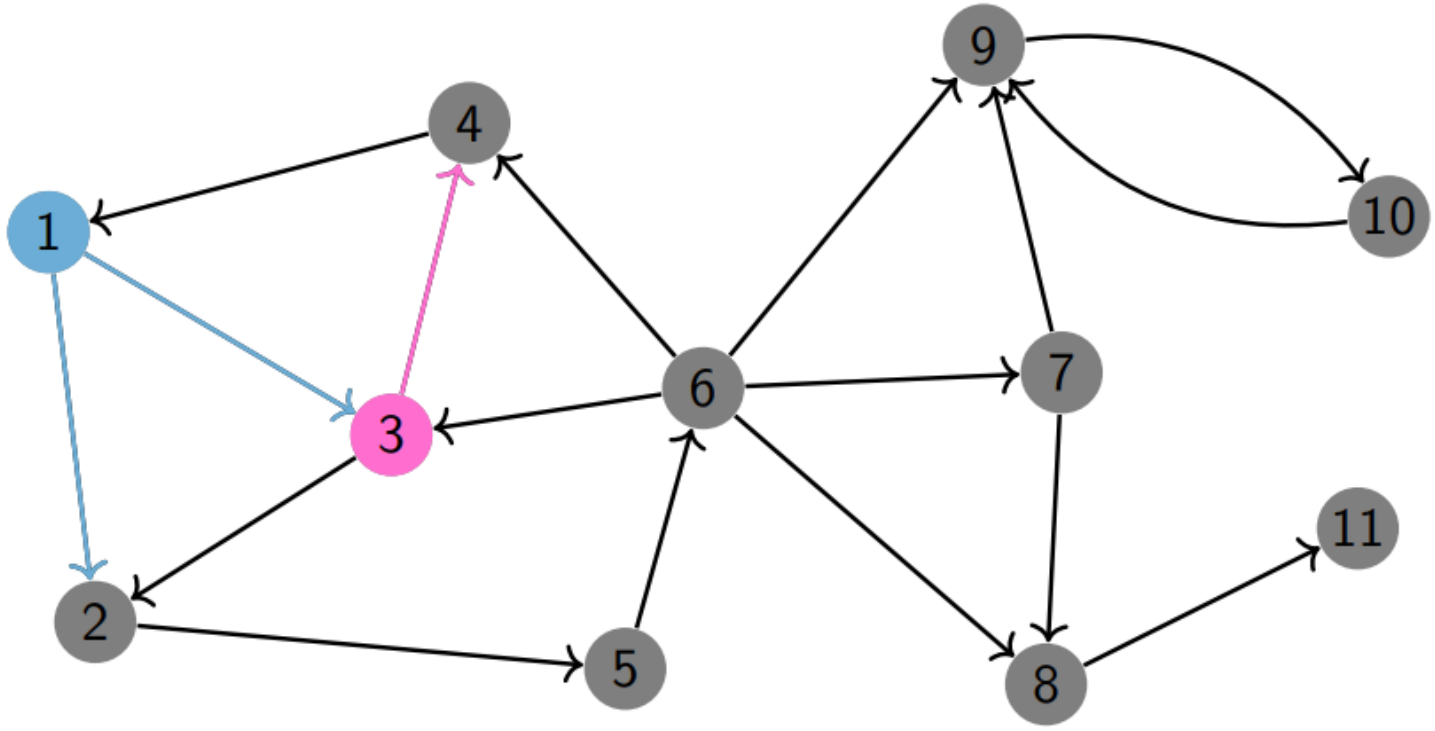
Ví dụ



Stack: 3 | 2

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	F	F	F	F	F	F	F	F

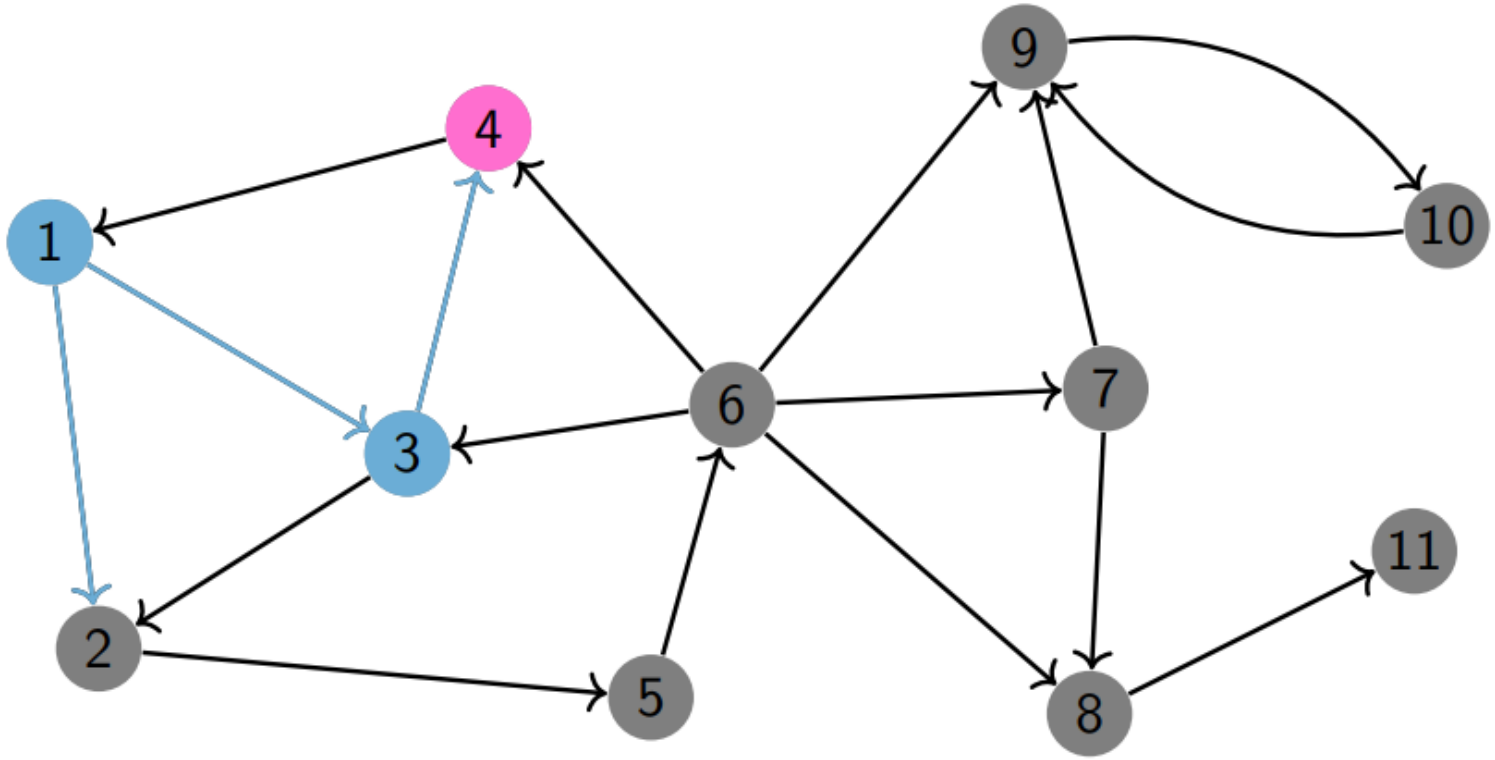
Ví dụ



Stack: 3 | 4 2

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	F	F	F	F	F	F	F

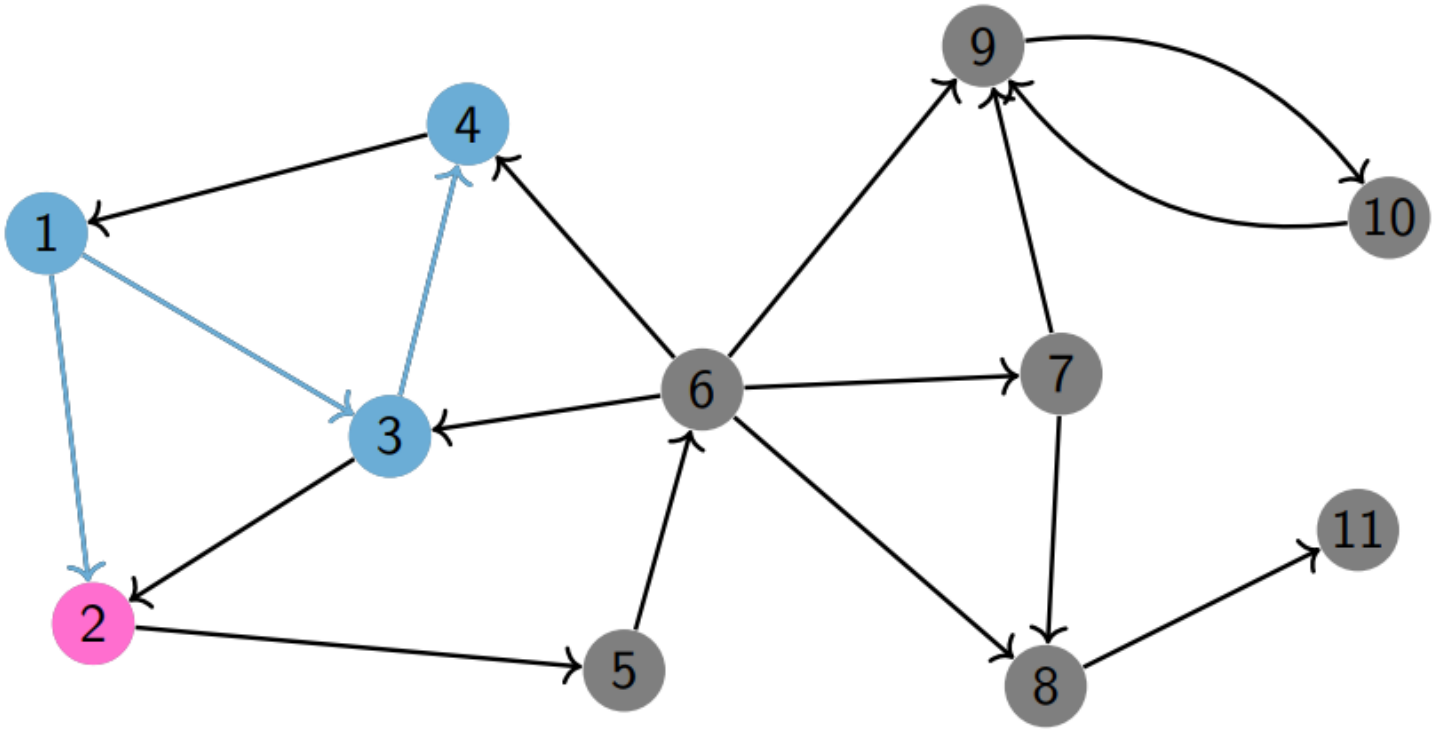
Ví dụ



Stack: 4 | 2

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	F	F	F	F	F	F	F

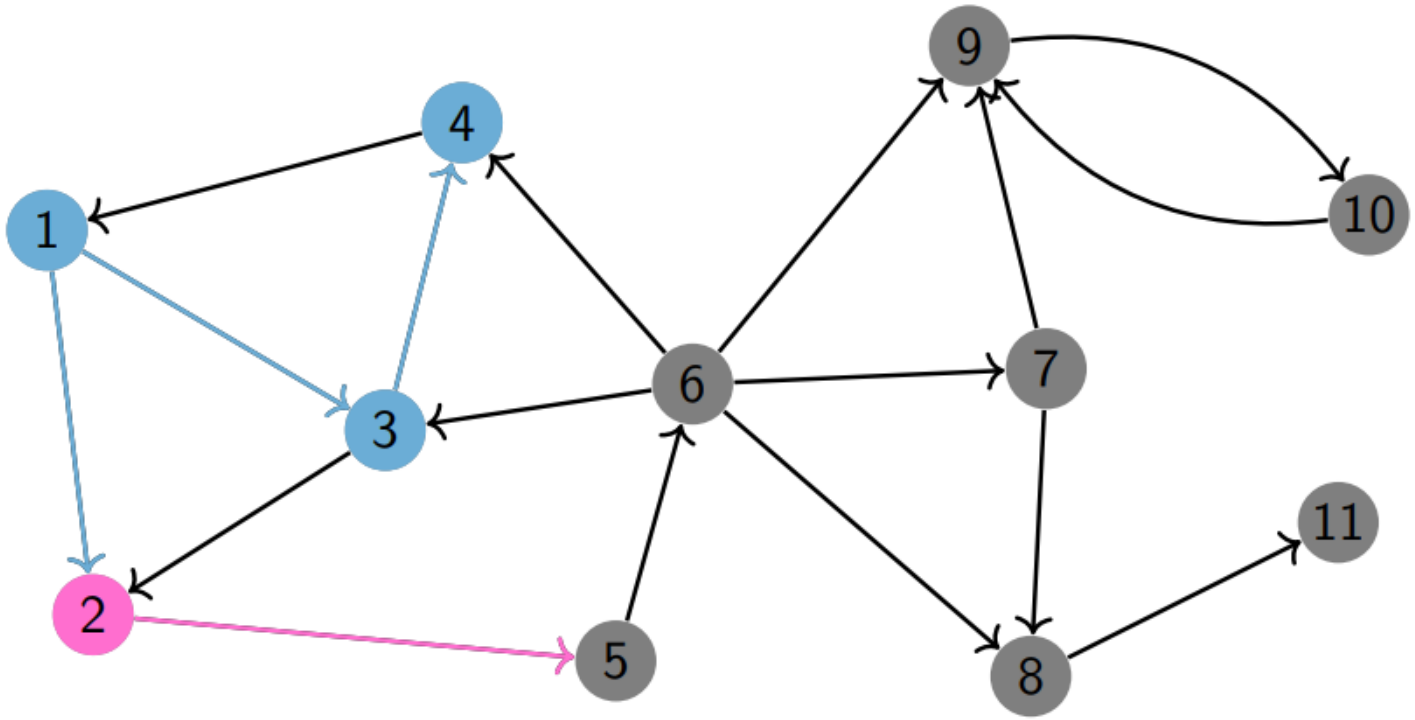
Ví dụ



Stack: 2 |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	F	F	F	F	F	F	F

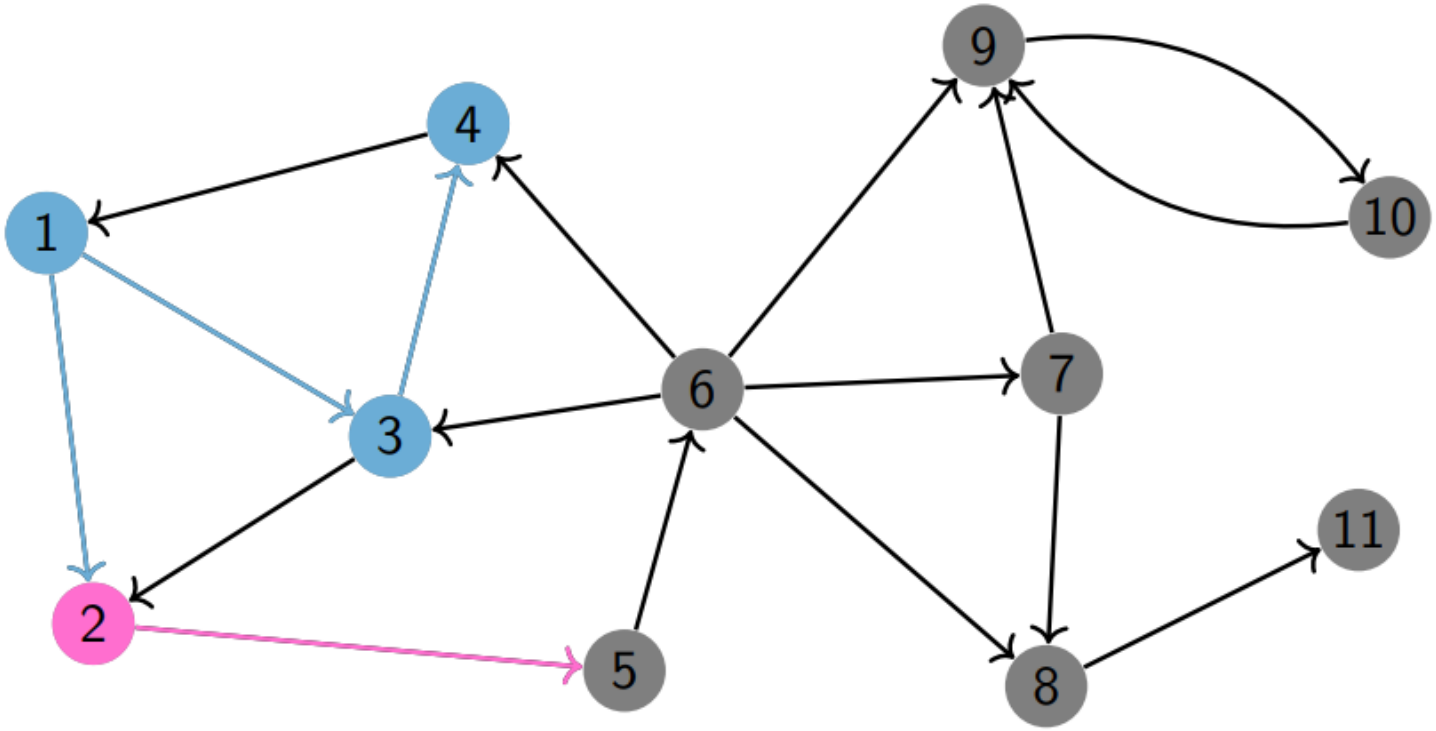
Ví dụ



Stack: 2 |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	F	F	F	F	F	F	F

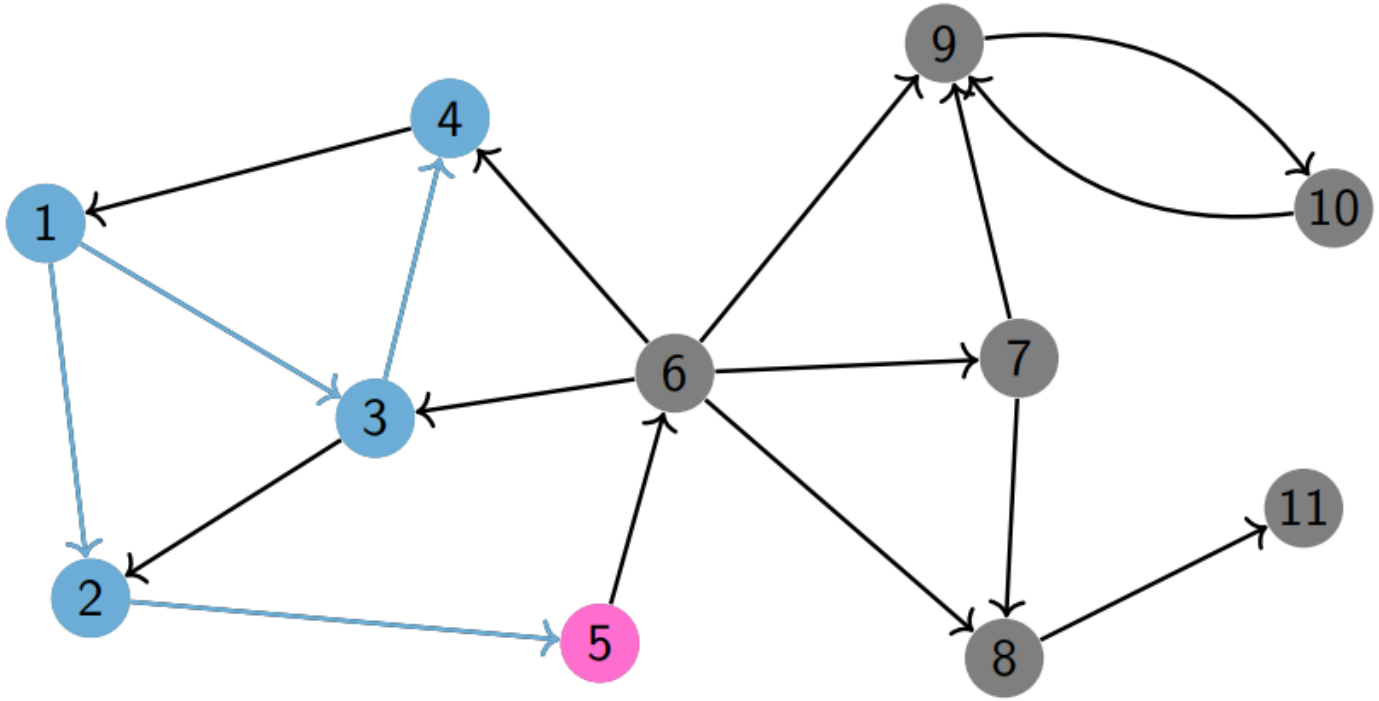
Ví dụ



Stack: 2 | 5

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	F	F	F	F	F	F

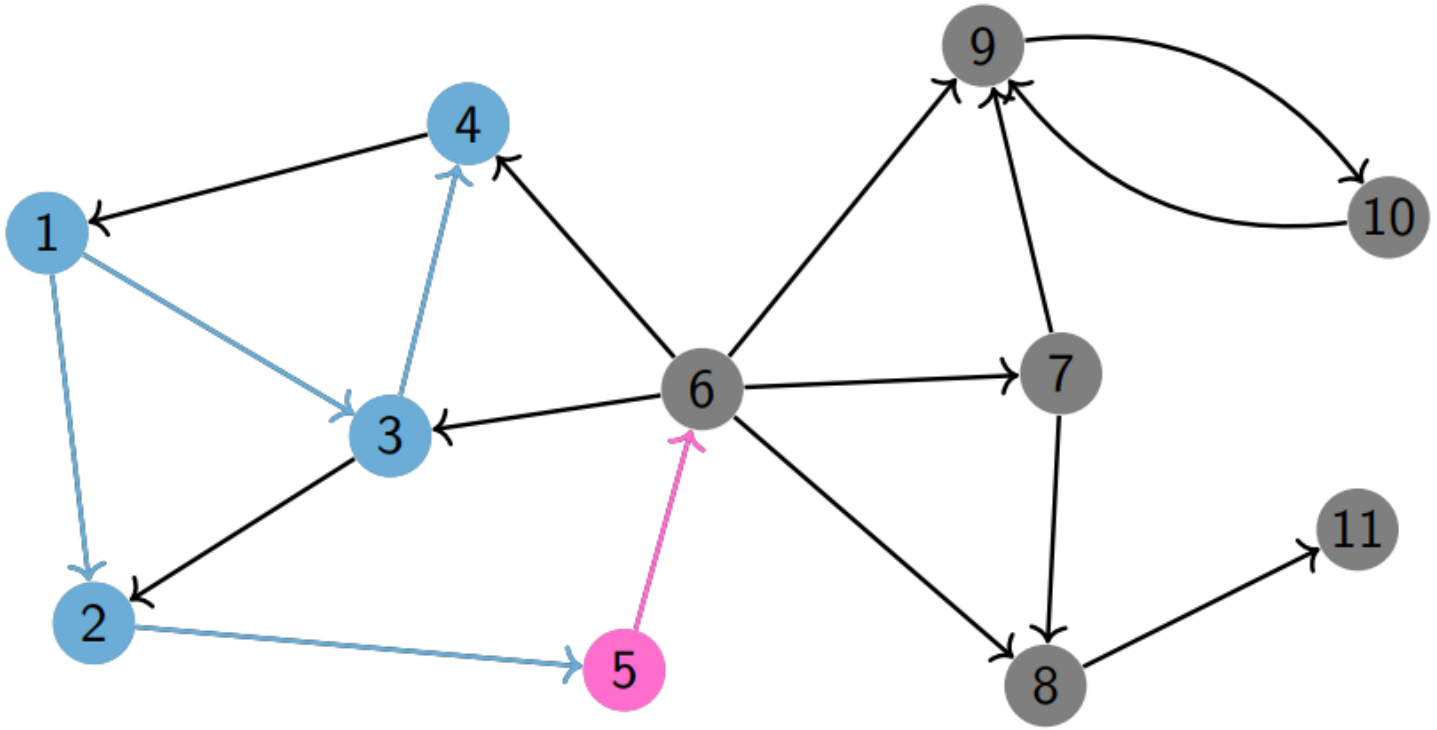
Ví dụ



Stack: 5 |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	F	F	F	F	F	F

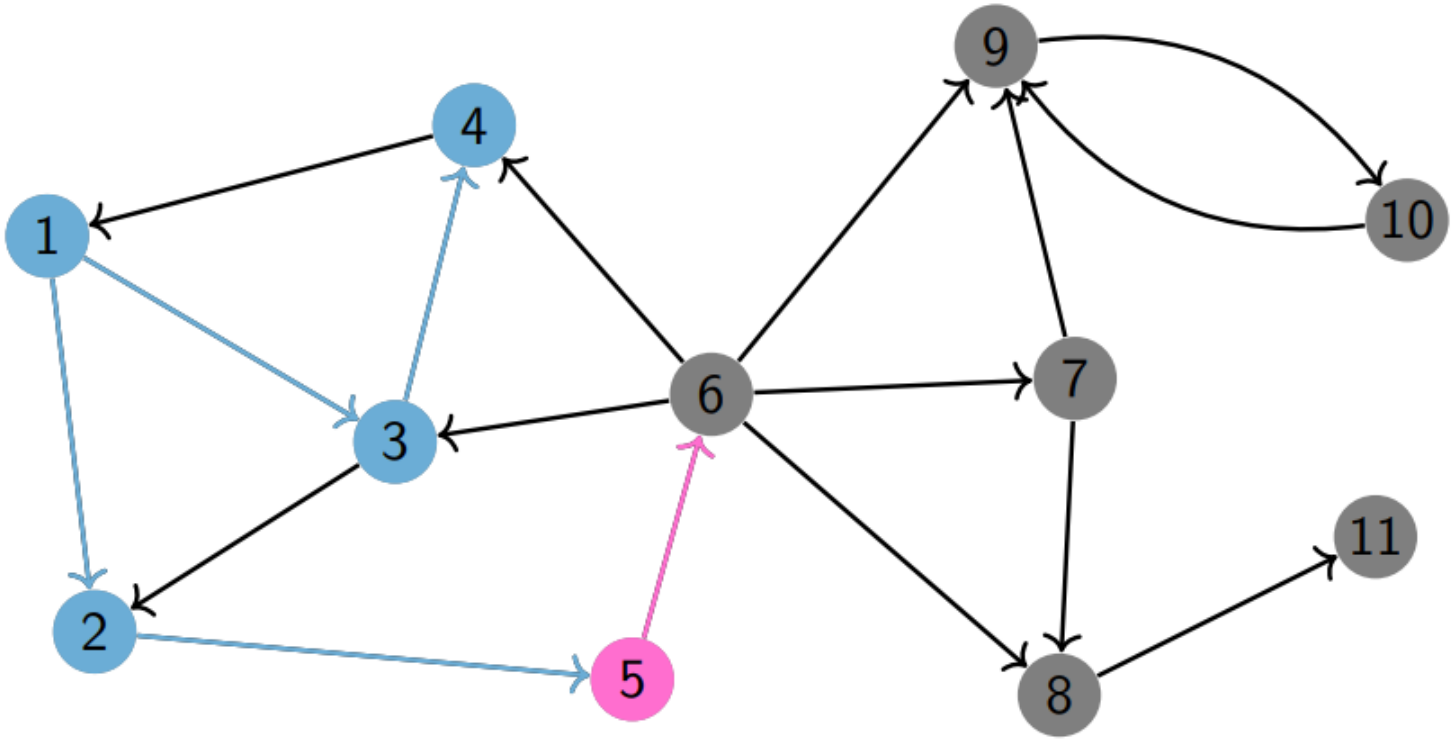
Ví dụ



Stack: 5 |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	F	F	F	F	F	F

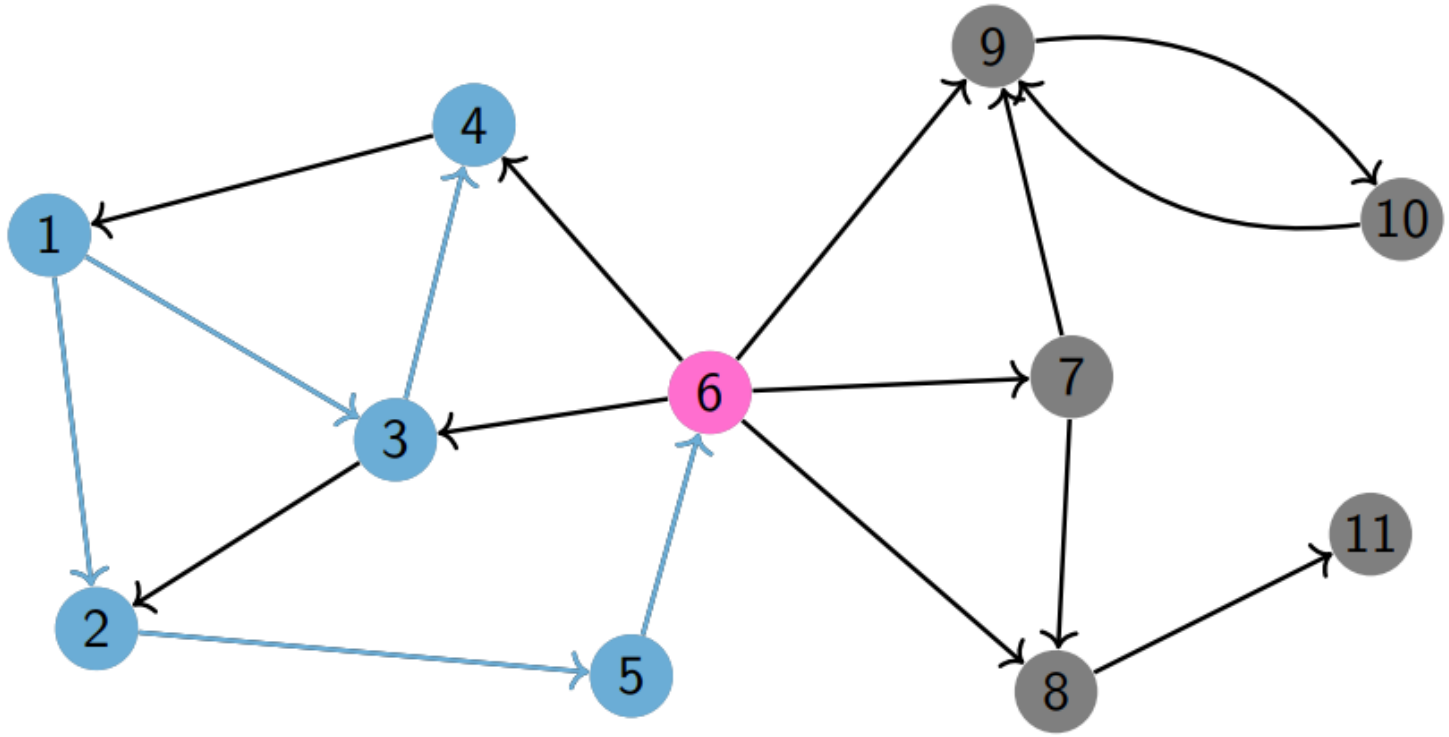
Ví dụ



Stack: 5 | 6

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	F	F	F	F	F

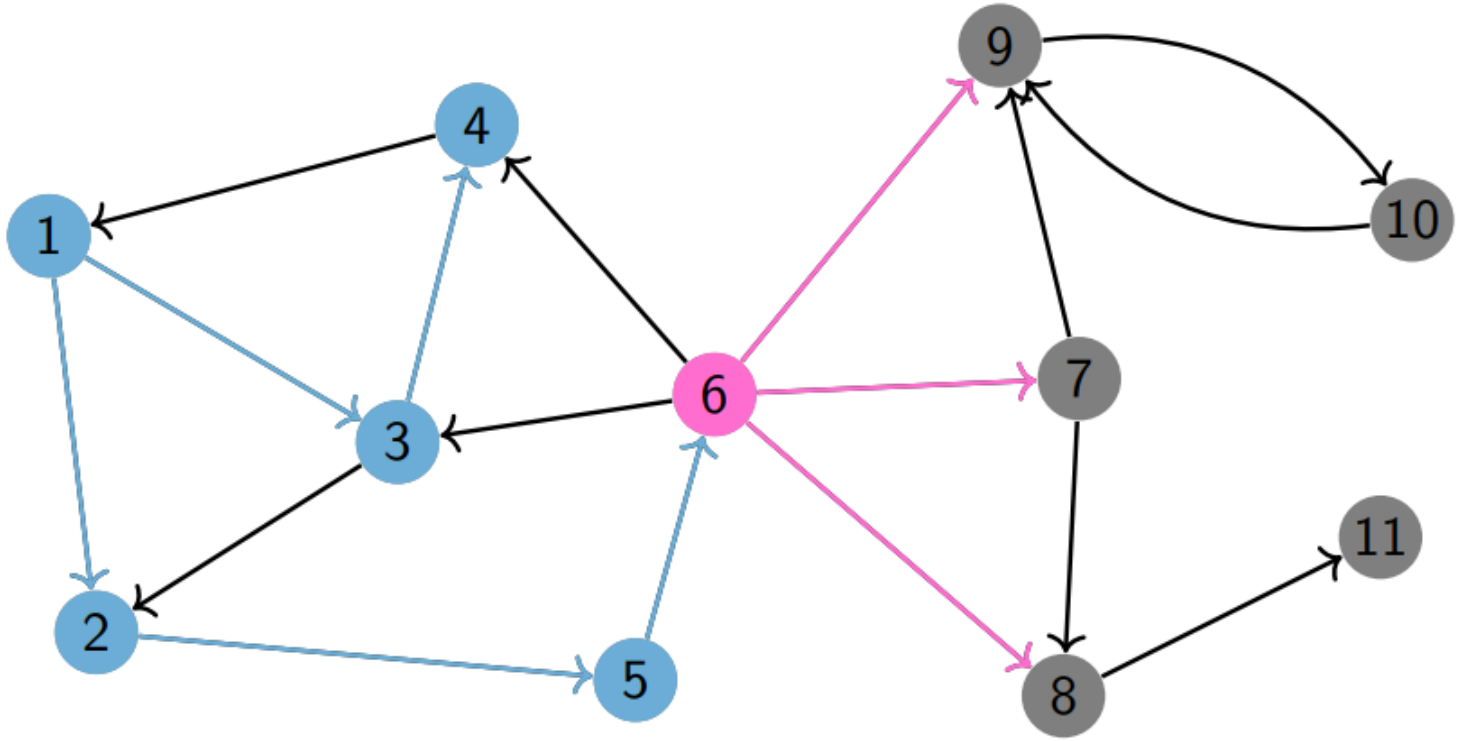
Ví dụ



Stack: 6 |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	F	F	F	F	F

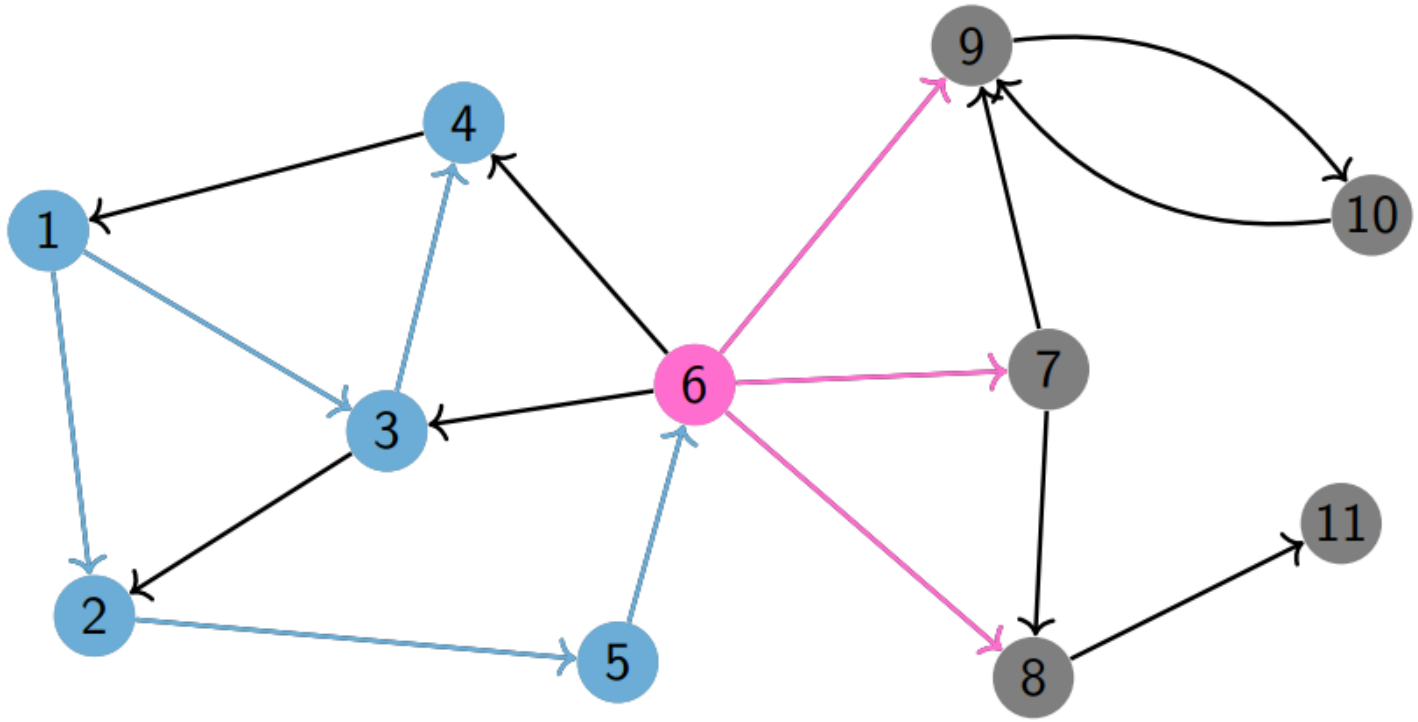
Ví dụ



Stack: 6 |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	F	F	F	F	F

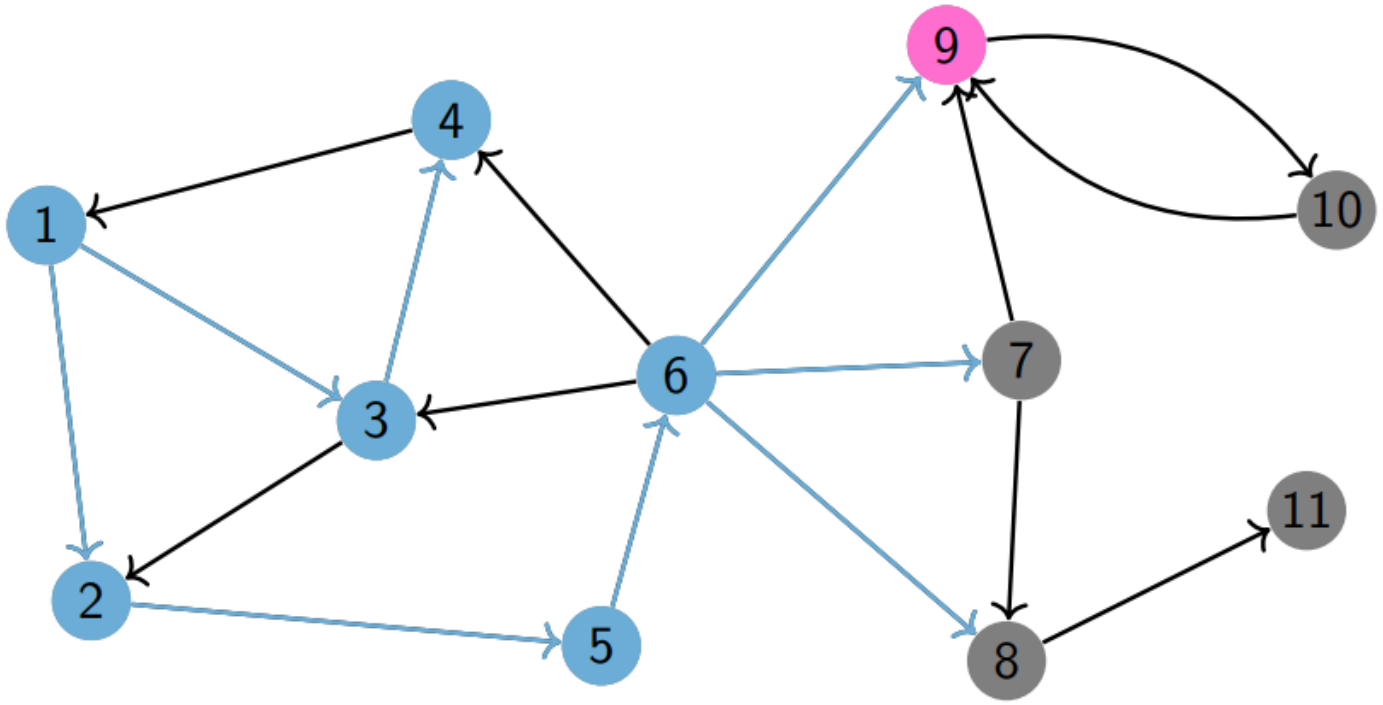
Ví dụ



Stack: 6 | 9 7 8

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	F	F

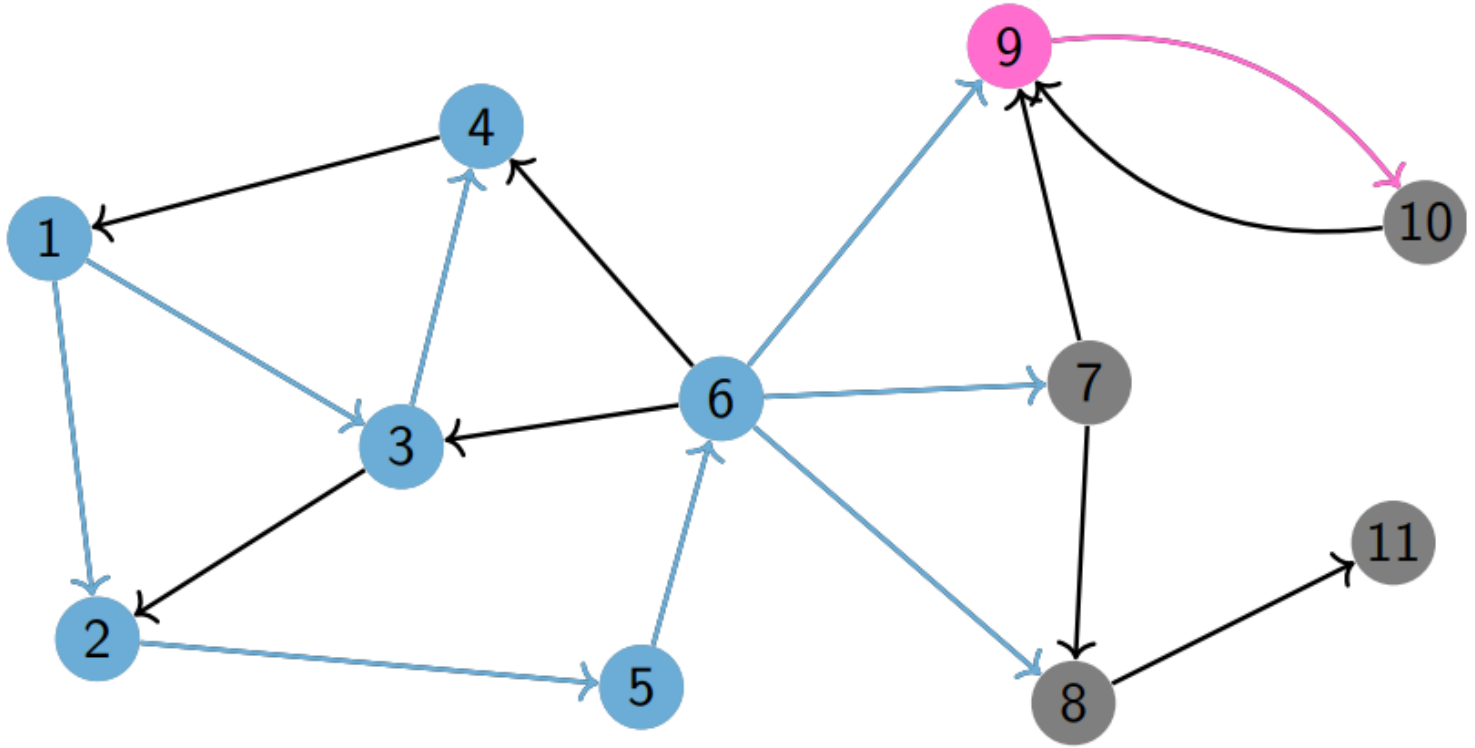
Ví dụ



Stack: 9 | 7 8

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	F	F

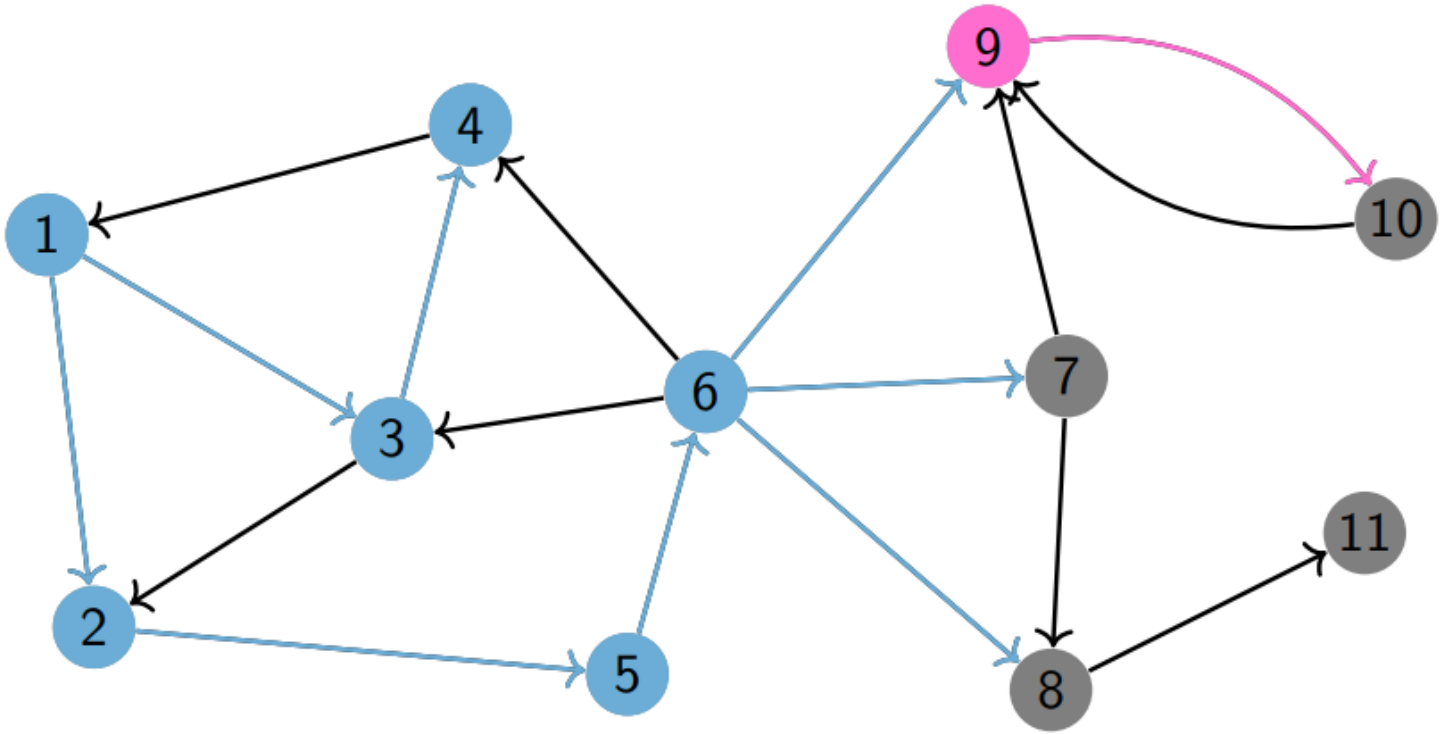
Ví dụ



Stack: 9 | 7 8

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	F	F

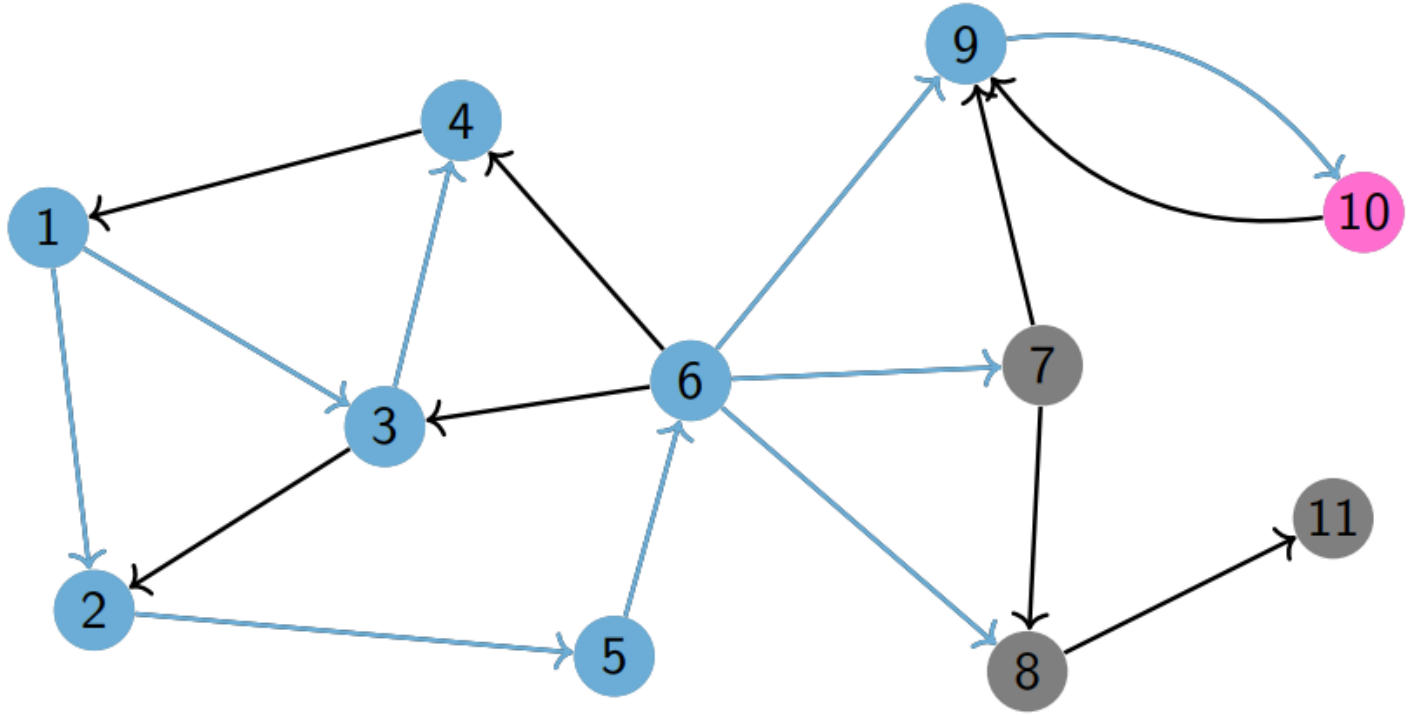
Ví dụ



Stack: 9 | 10 7 8

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	T	F

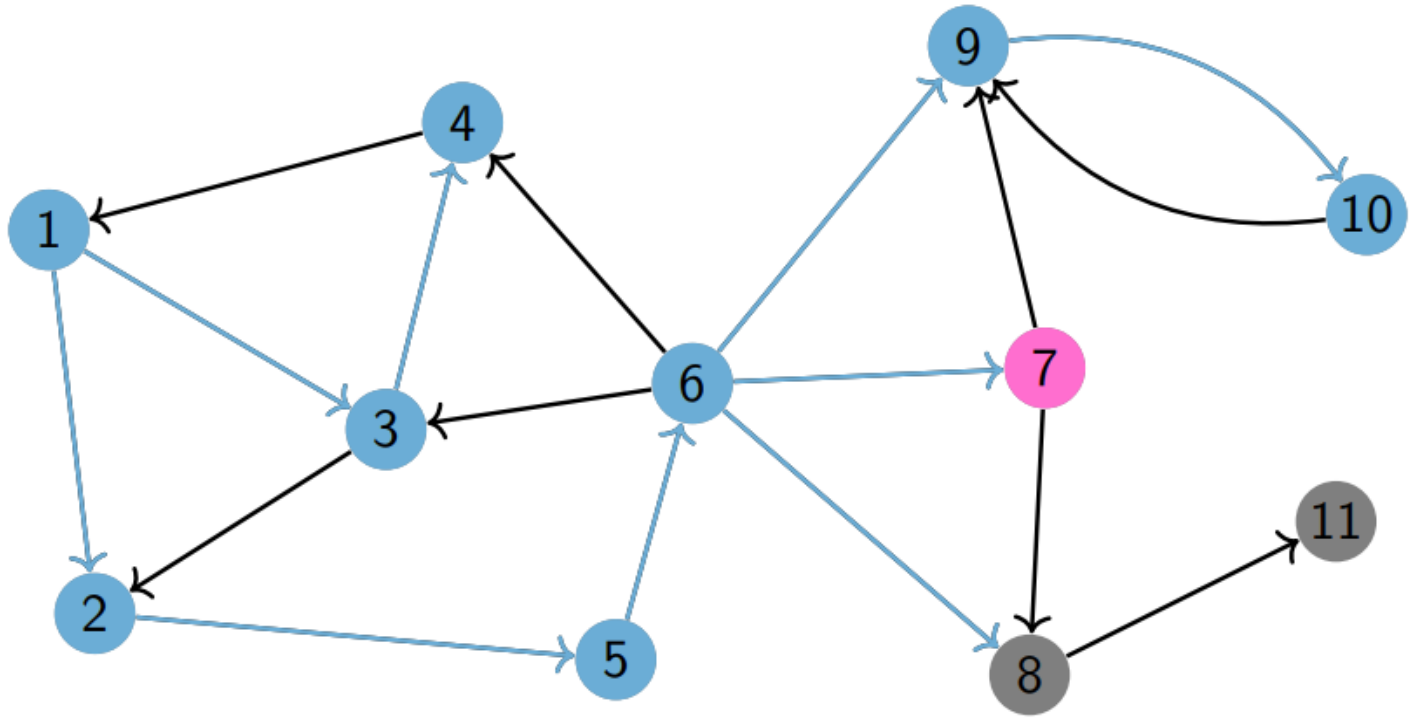
Ví dụ



Stack: 10 | 7 8

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	T	F

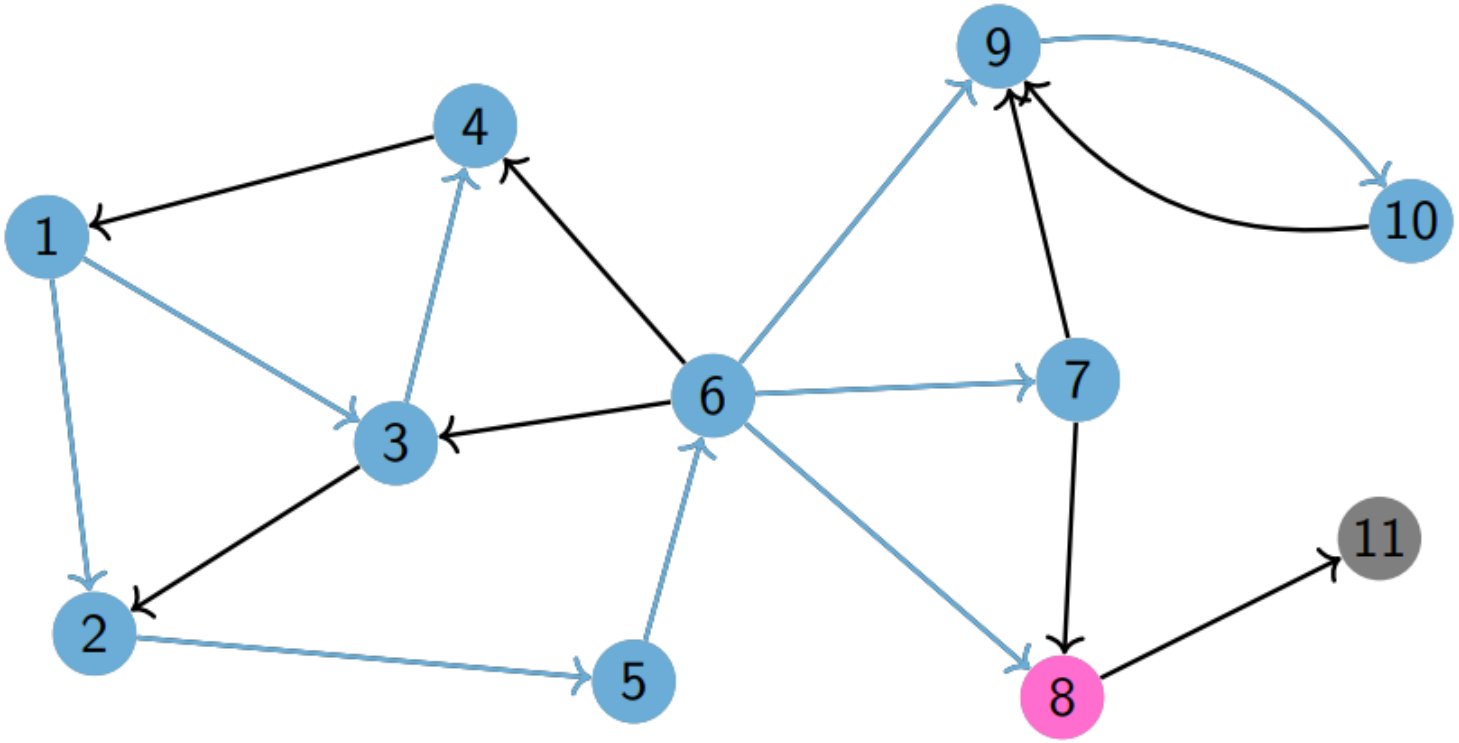
Ví dụ



Stack: 7 | 8

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	T	F

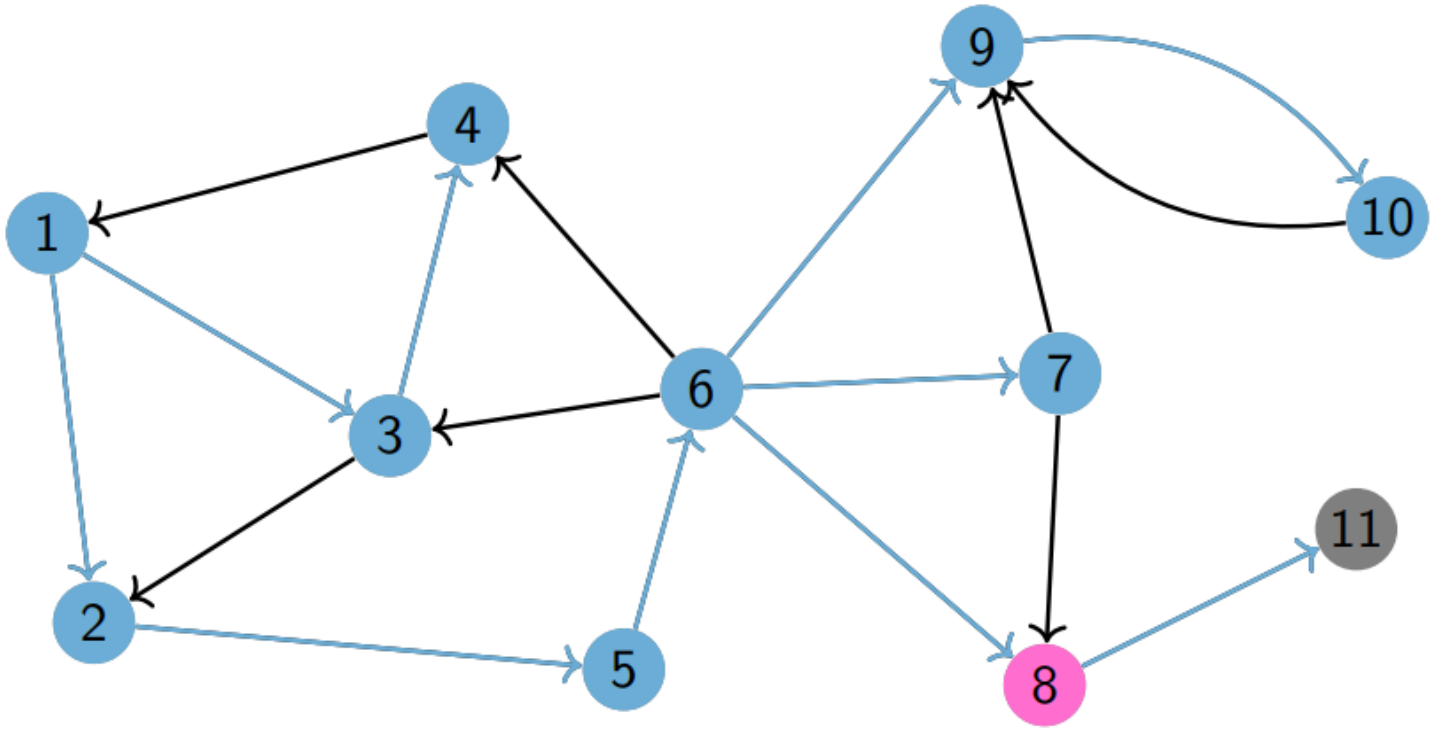
Ví dụ



Stack: 8 |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	T	F

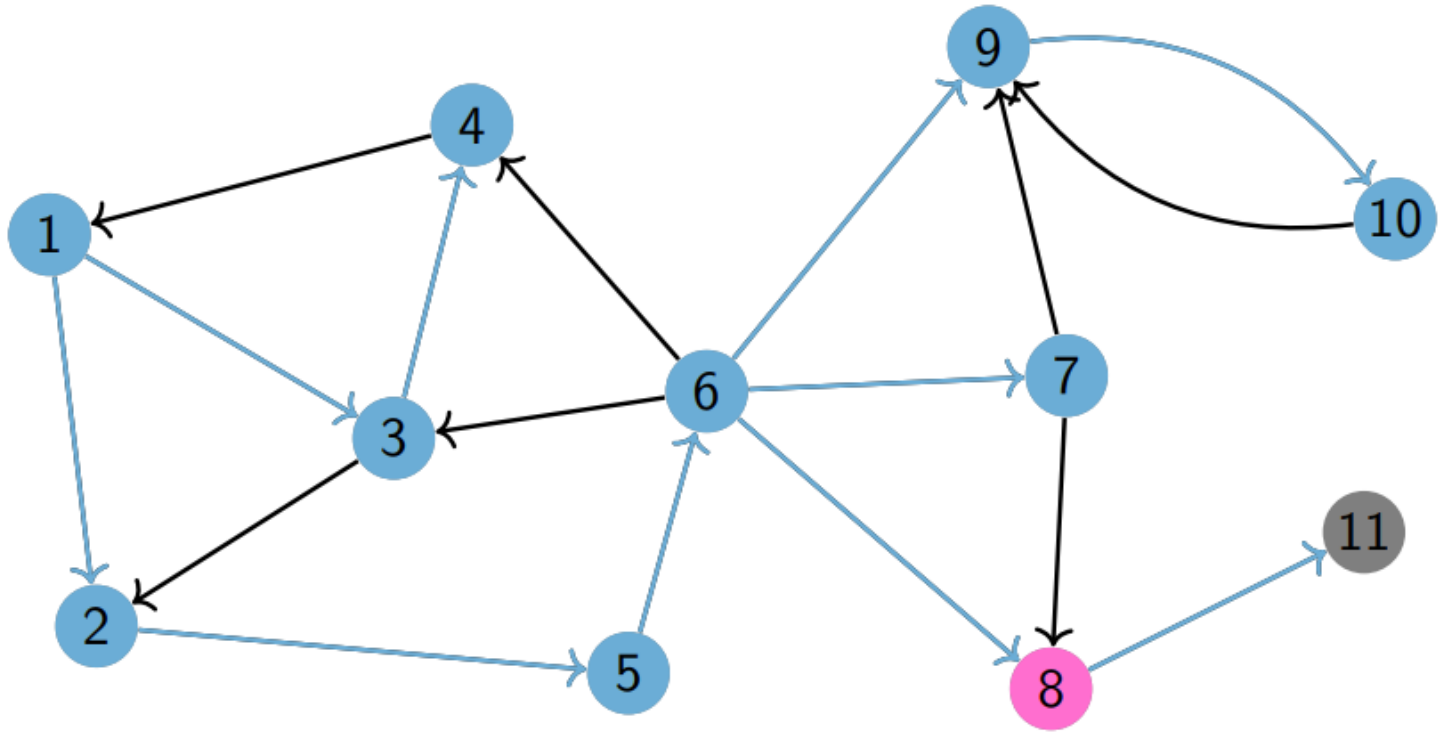
Ví dụ



Stack: 8 |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	T	F

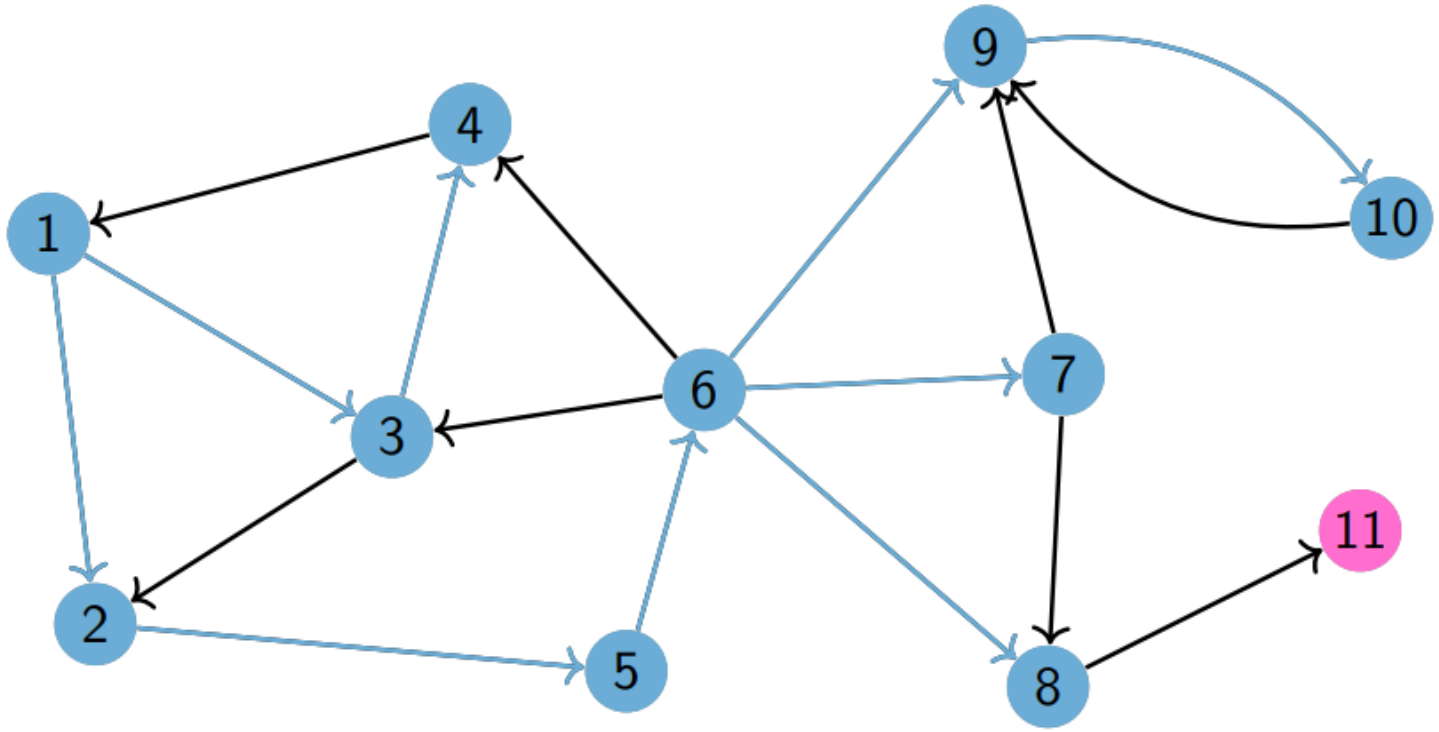
Ví dụ



Stack: 8 | 11

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	T	T

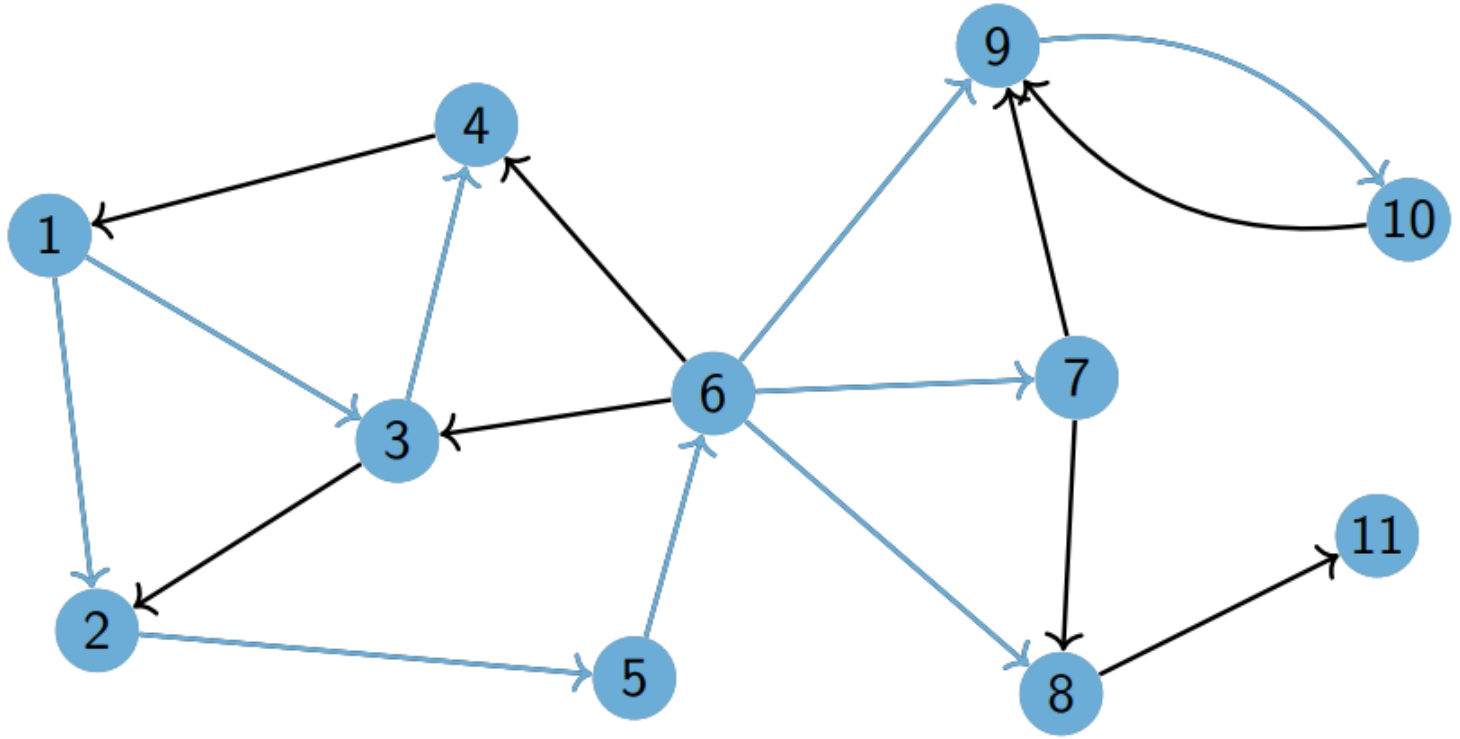
Ví dụ



Stack: 11 |

	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	T	T

Ví dụ



Stack: |

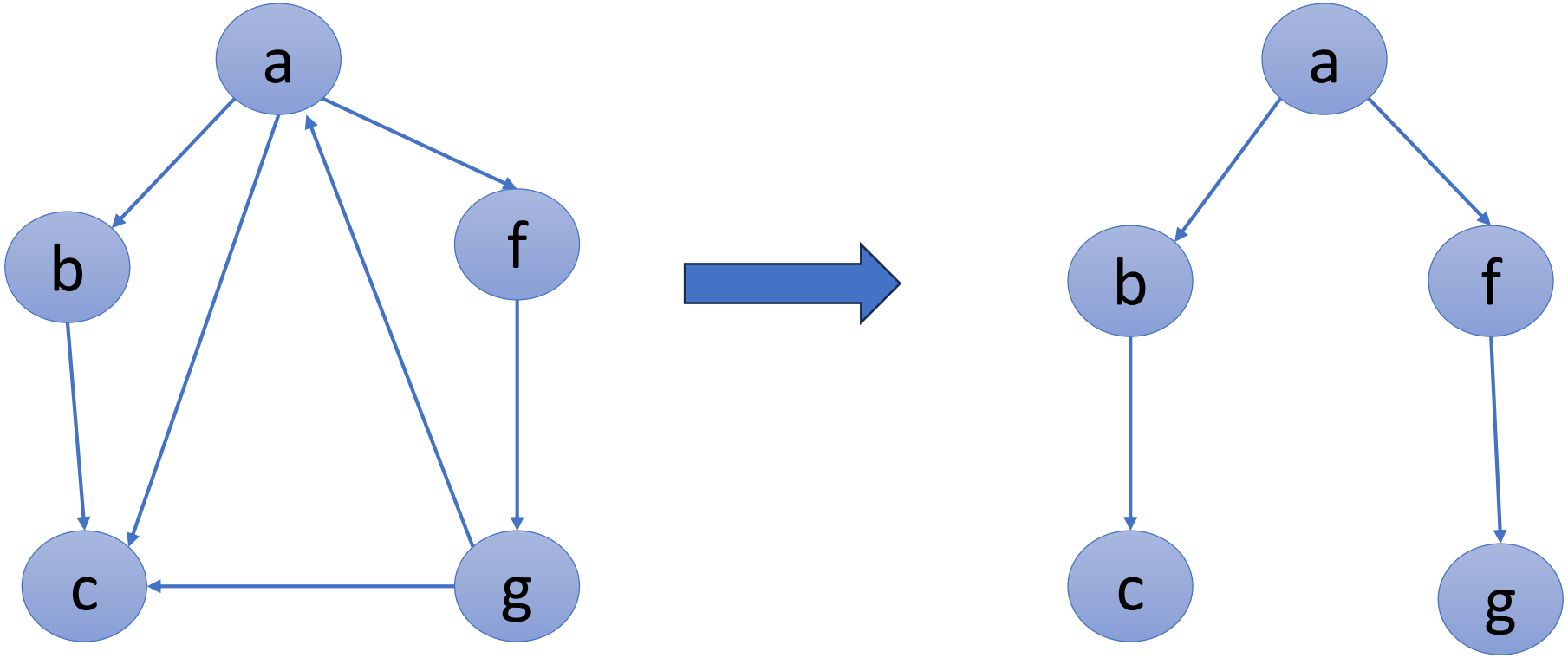
	1	2	3	4	5	6	7	8	9	10	11
Visited	T	T	T	T	T	T	T	T	T	T	T

NỘI DUNG

- Tìm kiếm theo chiều sâu (DFS)
- **Cây DFS và cấu trúc Num, Low**
- Tìm cạnh cầu (bridges)
- Tìm đỉnh khớp (articulation)
- Tìm thành phần liên thông mạnh (strongly connected component)

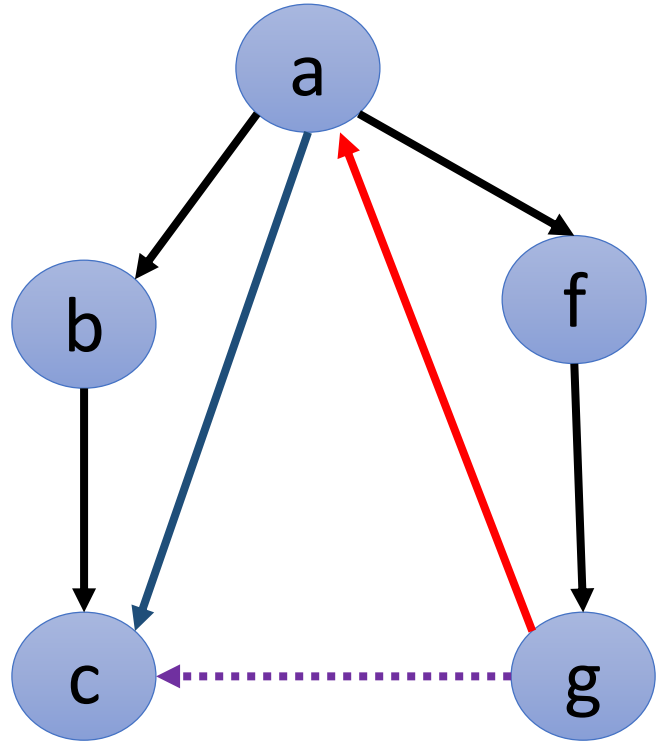
Cây DFS

- Vết của quá trình tìm kiếm theo chiều sâu sẽ tạo thành một cây



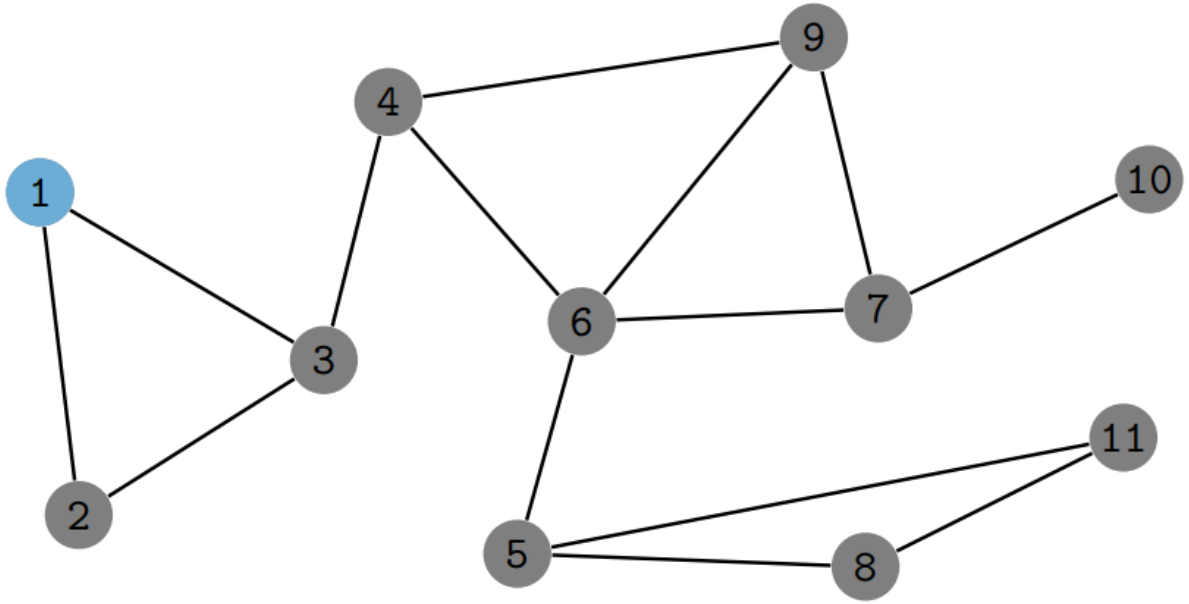
Cây DFS

- Vết của quá trình tìm kiếm theo chiều sâu sẽ tạo thành một cây
- Một số loại cạnh trong quá trình tìm kiếm
  - Cạnh cây (Tree Edge): Cạnh theo đó từ một đỉnh thăm được đỉnh mới, *ví dụ cạnh màu đen trong hình bên*
  - Cạnh ngược (Back Edge): Cạnh đi từ con cháu đến tổ tiên, *ví dụ cạnh màu đỏ (g,a) trong hình bên*
  - Cạnh xuôi (Forward Edge): Cạnh đi từ tổ tiên đến con cháu, *ví dụ cạnh màu xanh (a,c) trong hình bên*
  - Cạnh vòng (Crossing Edge): Cạnh nối giữa 2 đỉnh không có liên quan, *ví dụ cạnh màu tím nét đứt (c,g) trong hình bên*



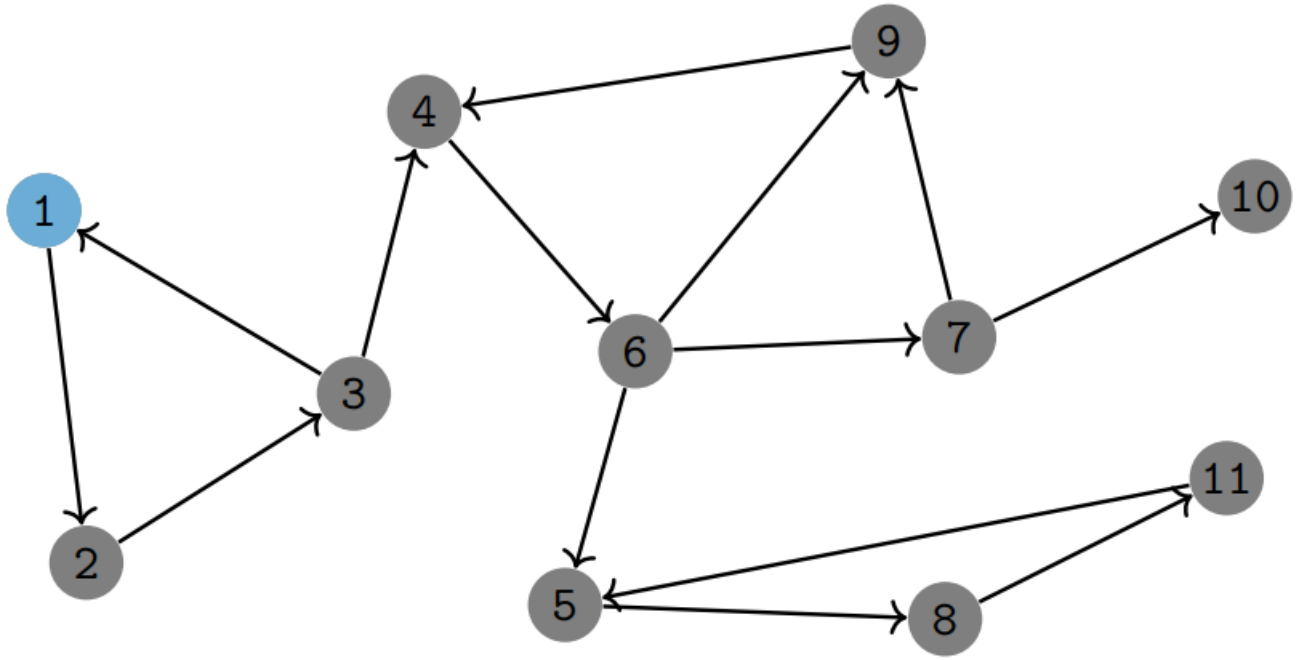
Cây DFS

- Cấu trúc Num và Low gắn với mỗi đỉnh của cây DFS.
- $Num[u]$: thứ tự thăm của đỉnh u trong DFS
- $Low[u]$: Giá trị nhỏ nhất trong những giá trị sau:
  - $Num[v]$ nếu (v, u) là một cạnh ngược
  - $Low[v]$ nếu v là con của u trong cây DFS
  - $Num[u]$



i	1	2	3	4	5	6	7	8	9	10	11
Num[i]	1	2	3	4	6	5	9	7	10	11	8
Low[i]	1	1	1	4	6	4	4	6	4	11	6

Ví dụ



i	1	2	3	4	5	6	7	8	9	10	11
Num[i]	1	2	3	4	6	5	9	7	10	11	8
Low[i]	1	1	1	4	6	4	4	6	4	11	6

Ý tưởng cài đặt

$cur\_num = 0$

for each u in V, $Num[u] = -1$

Analyze(v, pv)

$Low[v] = Num[v] = cur\_num++$

for each u in $Adj[v]$

if $u == pv$ continue

if $Num[u] == -1$

Analyze(u, v)

$Low[u] = \min(Low[u], Low[v])$

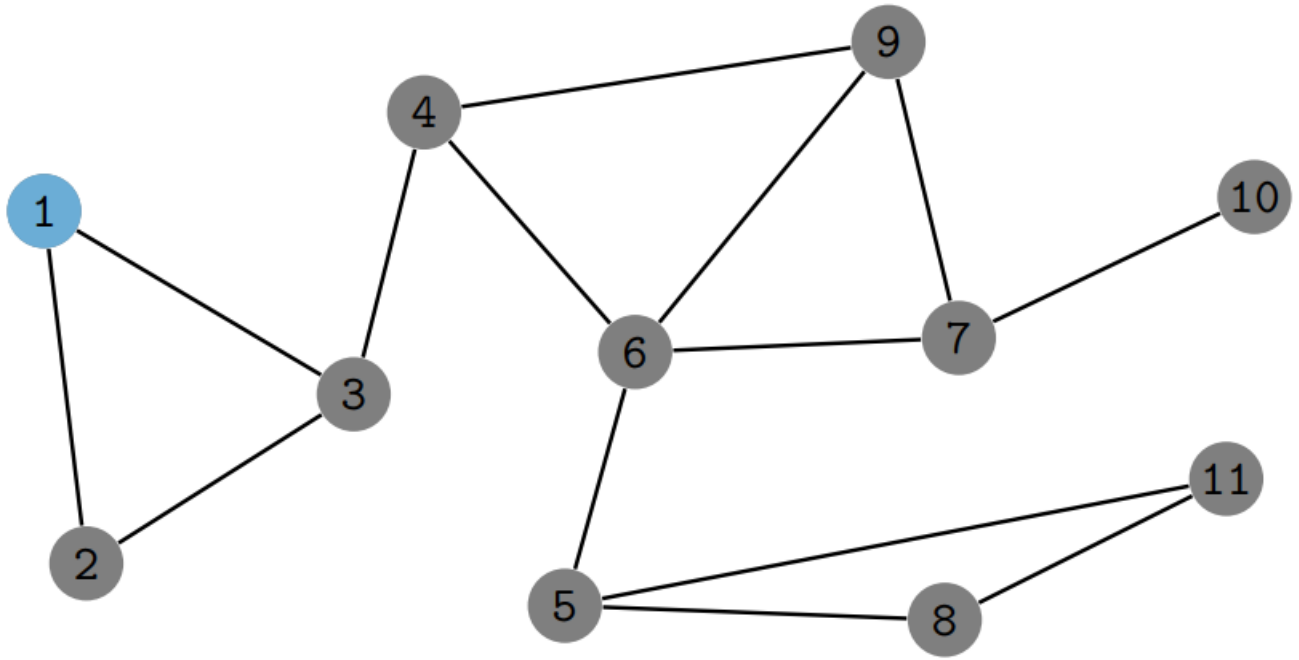
else $Low[u] = \min(Low[u], Low[v])$

NỘI DUNG

- Tìm kiếm theo chiều sâu (DFS)
- Cây DFS và cấu trúc Num, Low
- **Tìm cạnh cầu (bridges)**
- Tìm đỉnh khớp (articulation)
- Tìm thành phần liên thông mạnh (strongly connected component)

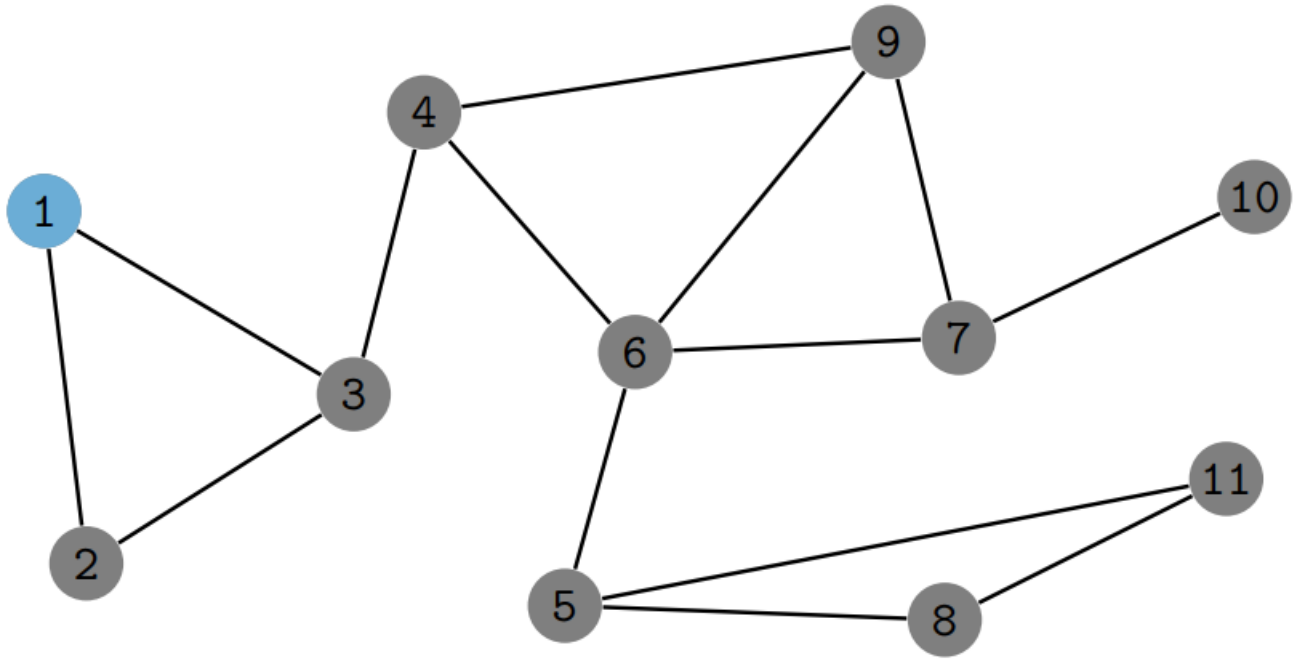
Tìm cầu (bridge) trong đồ thị

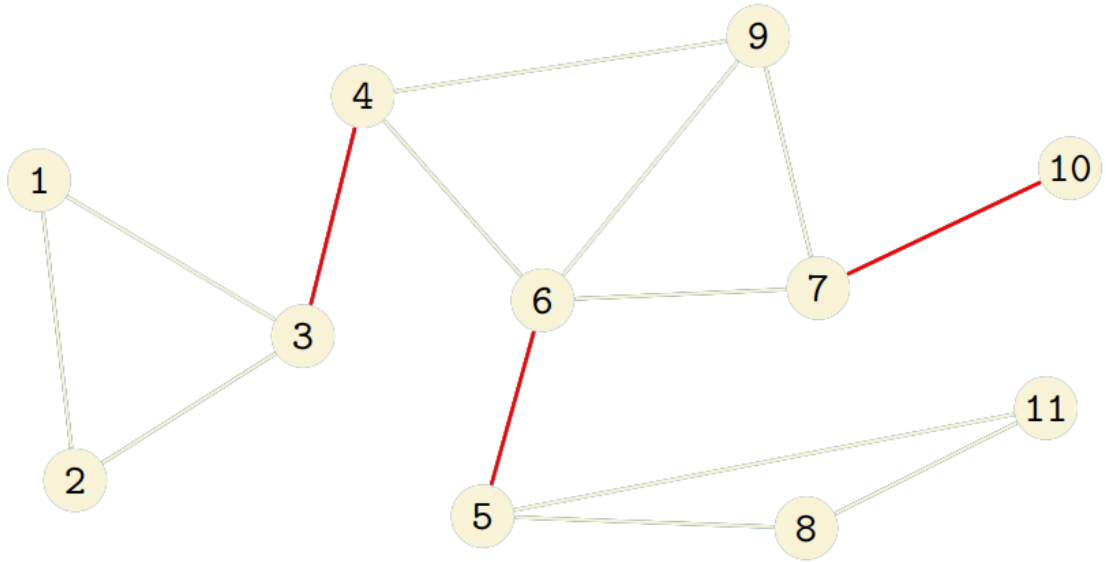
- **Định nghĩa**: Cầu là một cạnh của đồ thị vô hướng, nếu loại bỏ cạnh này khỏi đồ thị sẽ làm tăng số thành phần liên thông.
- **Nhận xét**: Một cạnh xuôi (u, v) là cầu khi và chỉ khi $Low[v] > Num[u]$



Tìm cầu (bridge) trong đồ thị

- **Định nghĩa**: Cầu là một cạnh của đồ thị, nếu loại bỏ cạnh này khỏi đồ thị sẽ làm tăng số thành phần liên thông.
- **Nhận xét**: Một cạnh xuôi (u, v) là cầu khi và chỉ khi $Low[v] > Num[u]$





i	1	2	3	4	5	6	7	8	9	10	11
Num[i]	1	2	3	4	6	5	9	7	10	11	8
Low[i]	1	1	1	4	6	4	4	6	4	11	6

Ý tưởng cài đặt

$cur\_num = 0$

for each u in V, $Num[u] = -1$

Vector<pair<int, int>> $bridgeLst$;

findBridges(v, pv)

$Low[v] = Num[v] = cur\_num++$

for each u in $Adj[v]$

if $u == pv$ continue

if $Num[u] == -1$

findBridges(u, v)

$Low[u] = \min(Low[u], Low[v])$

else $Low[u] = \min(Low[u], Low[v])$

if $Low[v] > Num[u]$

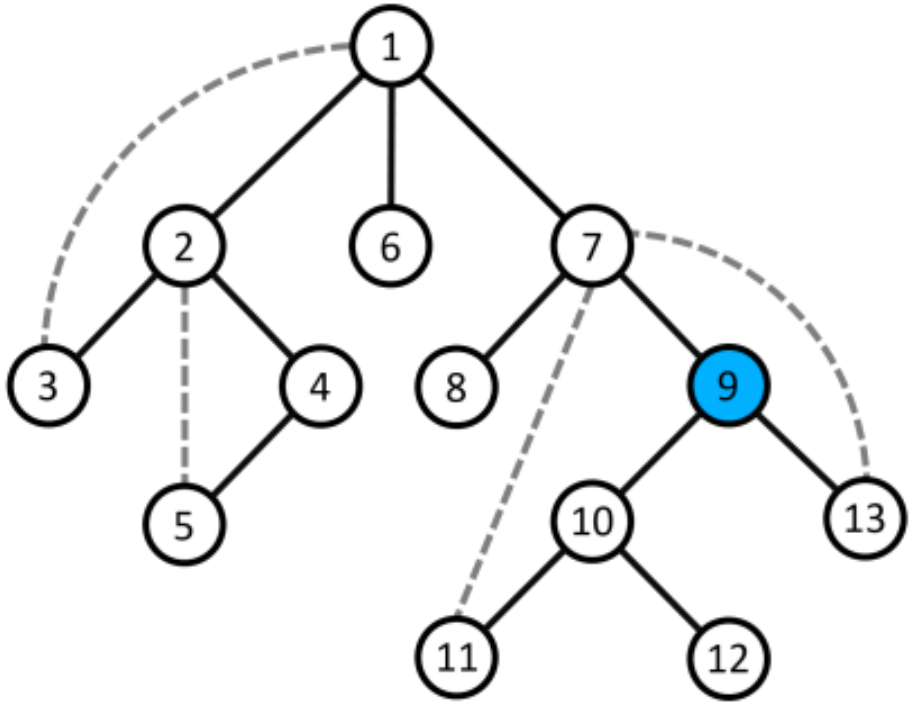
$bridgeLst$.push(make_pair(u, v))

NỘI DUNG

- Tìm kiếm theo chiều sâu (DFS)
- Cây DFS và cấu trúc Num, Low
- Tìm cạnh cầu (bridges)
- **Tìm đỉnh khớp (articulation)**
- Tìm thành phần liên thông mạnh (strongly connected component)

Tìm đỉnh khớp

- **Định nghĩa**: Trong đồ thị vô hướng, một đỉnh được gọi là khớp nếu như loại bỏ đỉnh này và các cạnh liên thuộc với nó ra khỏi đồ thị thì số thành phần liên thông của đồ thị tăng lên.
- **Nhận xét**: Đỉnh u được gọi là khớp nếu:
  - Đỉnh u không phải là gốc của cây DFS và $Low[v] > Num[u]$ (với v là một con trực tiếp bất kỳ của u trong cây DFS)
  - Hoặc đỉnh u là gốc của cây DFS và có ít nhất 2 con trực tiếp;



Ý tưởng cài đặt

$cur\_num = 0$

for each u in V, $Num[u] = -1$

bool $joint$[MAX];

findArticulation (v, pv)

Child = 0

$Low[v] = Num[v] = cur\_num++$

for each u in $Adj[v]$

if $u == pv$ continue

if $Num[u] == -1$

findArticulation(u, v)

child++

$Low[u] = \min(Low[u], Low[v])$

if $u == pv$ and $child > 1$

$joint[u]$ = true

else if $Low \geq Num[u]$

$joint[u]$ = true

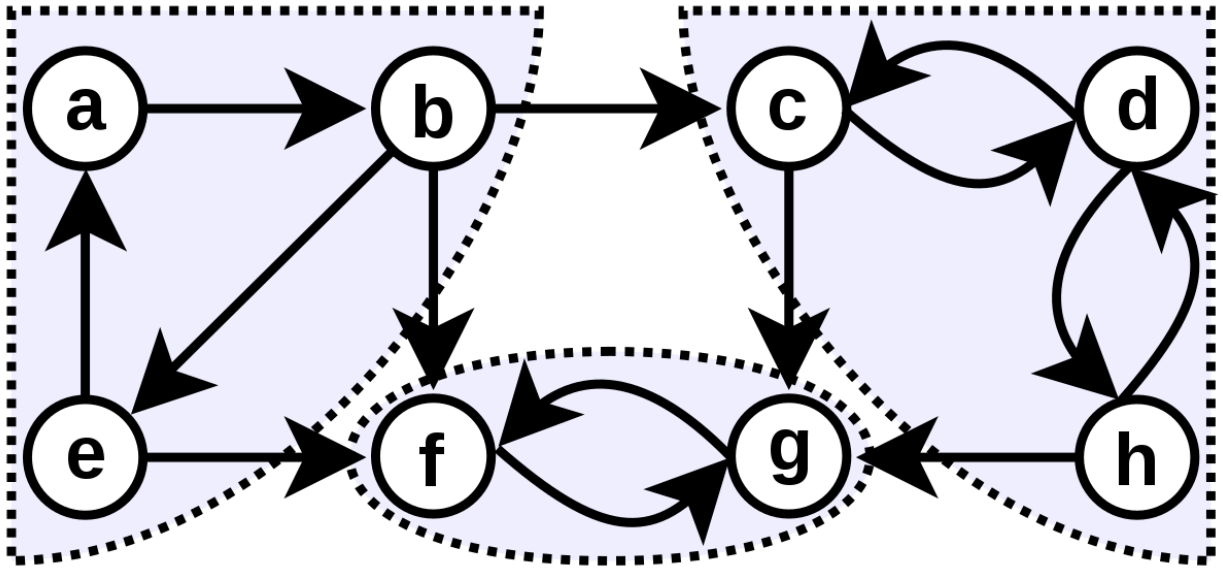
else $Low[u] = \min(Low[u], Low[v])$

NỘI DUNG

- Tìm kiếm theo chiều sâu (DFS)
- Cây DFS và cấu trúc Num, Low
- Tìm cạnh cầu (bridges)
- Tìm đỉnh khớp (articulation)
- **Tìm thành phần liên thông mạnh (strongly connected component)**

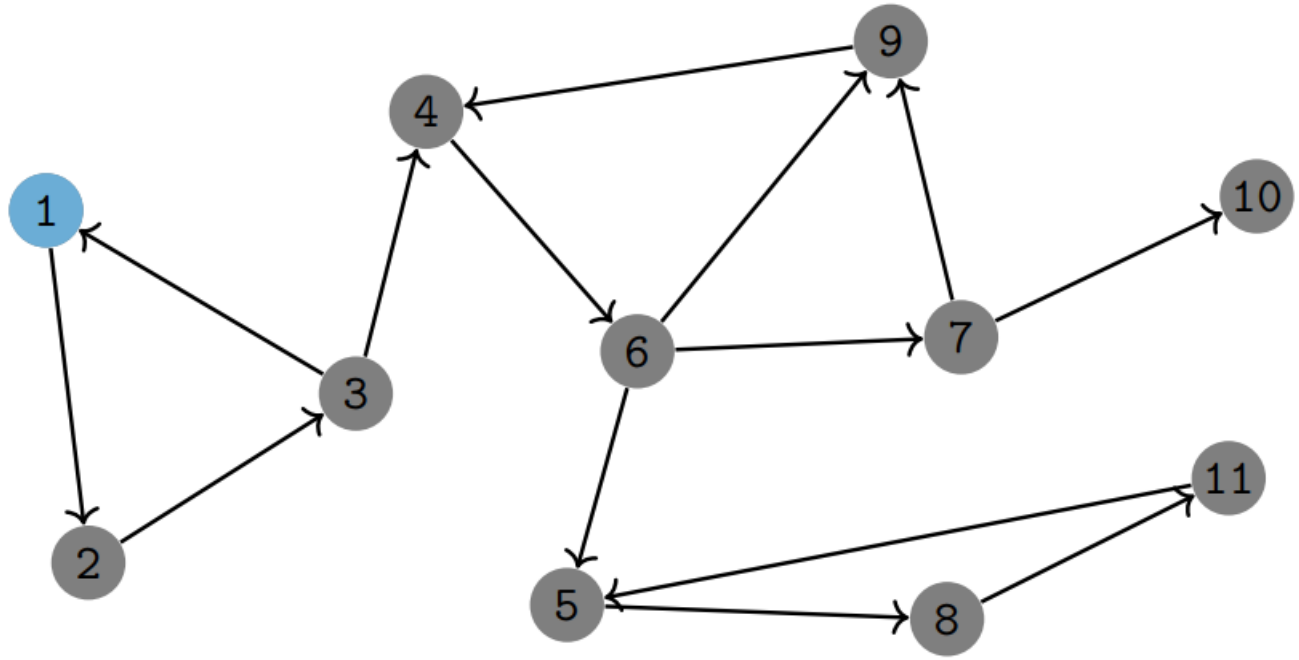
Thành phần liên thông mạnh (Strongly Connected Components)

- Duyệt đồ thị theo chiều rộng (BFS) và chiều sâu (DFS) có thể dễ dàng tìm ra tất cả các thành phần liên trong trong đồ thị vô hướng. Tuy nhiên, trong đồ thi có việc, việc tìm tất cả các thành phần liên thông mạnh không đơn giản.
  - Chú ý: Thành phần liên thông mạnh là một tập con tối đa các đỉnh sao cho giữa 2 đỉnh bất kỳ luôn có đường đi từ đỉnh này đến đỉnh kia và ngược lại.
- Có thể sử dụng cây tìm kiếm DFS để tìm tất cả các thành phần liên thông mạnh?



Thuật toán Tarjan

- **Nhận xét**: Sau khi phân tích cây DFS, nếu tại một đỉnh $u, Num[u] = Low[u]$ thì ta có một thành phân liên thông mạnh theo quá trình duyệt cây từ u.
- Sử dụng Stack để liệt kê các đỉnh trong một thành phần liên thông mạnh
- Độ phức tạp tính toán: $O(|V| + |E|)$



i	1	2	3	4	5	6	7	8	9	10	11
Num[i]	1	2	3	4	6	5	9	7	10	11	8
Low[i]	1	1	1	4	6	4	4	6	4	11	6

Ý tưởng cài đặt

$cur\_num = 0$

for each u in V, $Num[u] = -1$

Stack S

SCC(v, pv)

$Low[v] = Num[v] = cur_{num}++$

S.push(v)

for each u in $Adj[v]$

if $u == pv$ continue

if $Num[u] == -1$

SCC(u, v)

$Low[u] = \min(Low[u], Low[v])$

else $Low[u] = \min(Low[u], Low[v])$

if $Num[u] == Low[u]$

while(true)

v = S.top(), S.pop()

cout << v << " ";

if $v == u$

break

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !