

Graph Algorithms

Thành viên:

- Nguyễn Đình Bình An – 19521178
- Võ Đăng Châu - 19521282



Graph Algorithms

1. Giới thiệu
2. Các giải thuật và ứng dụng
3. Tổng kết



Graph Algorithms

1. Giới thiệu

- Đồ thị
- Graph Algorithm

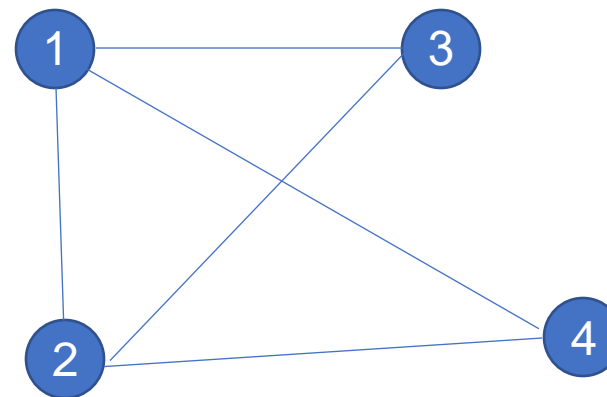


Graph Algorithms là gì nào?

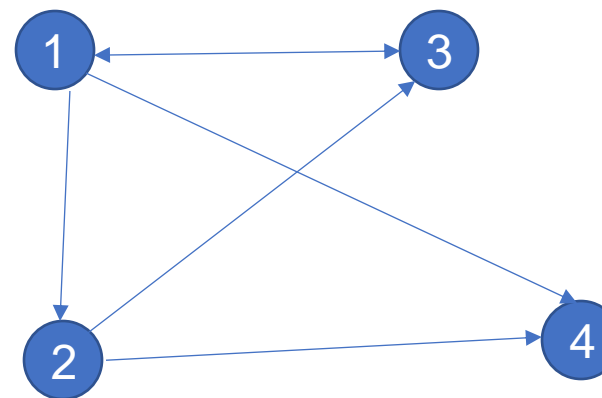
- Là một **tập** các **thuật toán** giải quyết các **vấn đề** liên quan đến **đồ thị**

Đồ thị

- Đồ thị bao gồm
 - ✓ Một tập hữu hạn các đỉnh (vertex)
 - ✓ Một tập hữu hạn các cạnh (edge) đi từ 1 đỉnh đến 1 đỉnh. Các cạnh trong đồ thị có thể được thêm trọng số để biểu diễn chi phí cần bỏ ra để đi qua cạnh đó.
- Có 2 loại đồ thị: đồ thị có hướng và đồ thị vô hướng
- Có 2 cách để biểu diễn đồ thị:
Ma trận đỉnh kề hoặc Danh sách đỉnh kề

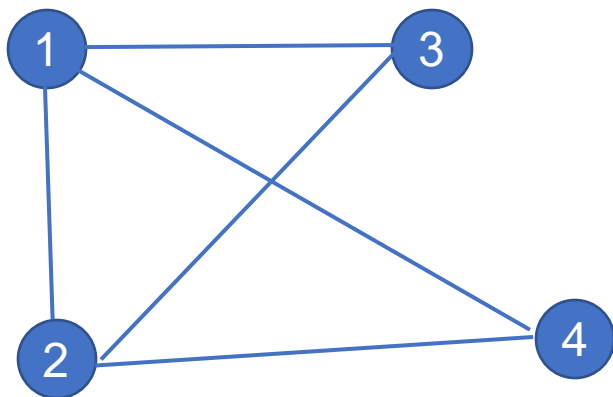


Đồ thị vô hướng



Đồ thị có hướng

Đồ thị



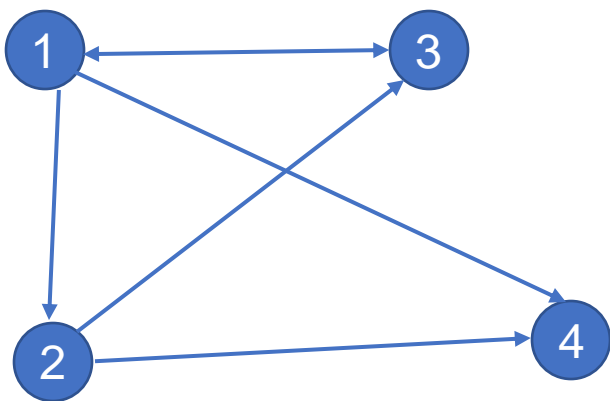
	1	2	3	4
1	0	1	1	1
2	1	0	1	1
3	1	1	0	0
4	1	1	0	0

Ma trận đỉnh kề

1	2, 3, 4
2	1, 3, 4
3	1, 2
4	1, 2

Danh sách đỉnh kề

Đồ thị



	1	2	3	4
1	0	1	1	1
2	0	0	1	1
3	1	0	0	0
4	0	0	0	0

Ma trận đỉnh kề

1	2, 3, 4
2	3, 4
3	1
4	

Danh sách đỉnh kề

Graph

Cách khởi tạo 1 class Graph:

```
from collections import defaultdict
class Graph:
    #Thêm thư viện defaultdict để tạo và quản lí đồ thị

    def __init__(self):
        #Constructor
        self.graph = defaultdict(list)
        #Thư viện mặc định để lưu đồ thị

    def addEdge(self,u,v):
        self.graph[u].append(v)
        #Hàm để thêm 1 cạnh vào đồ thị
```


Graph Algorithms

2. Các giải thuật và ứng dụng

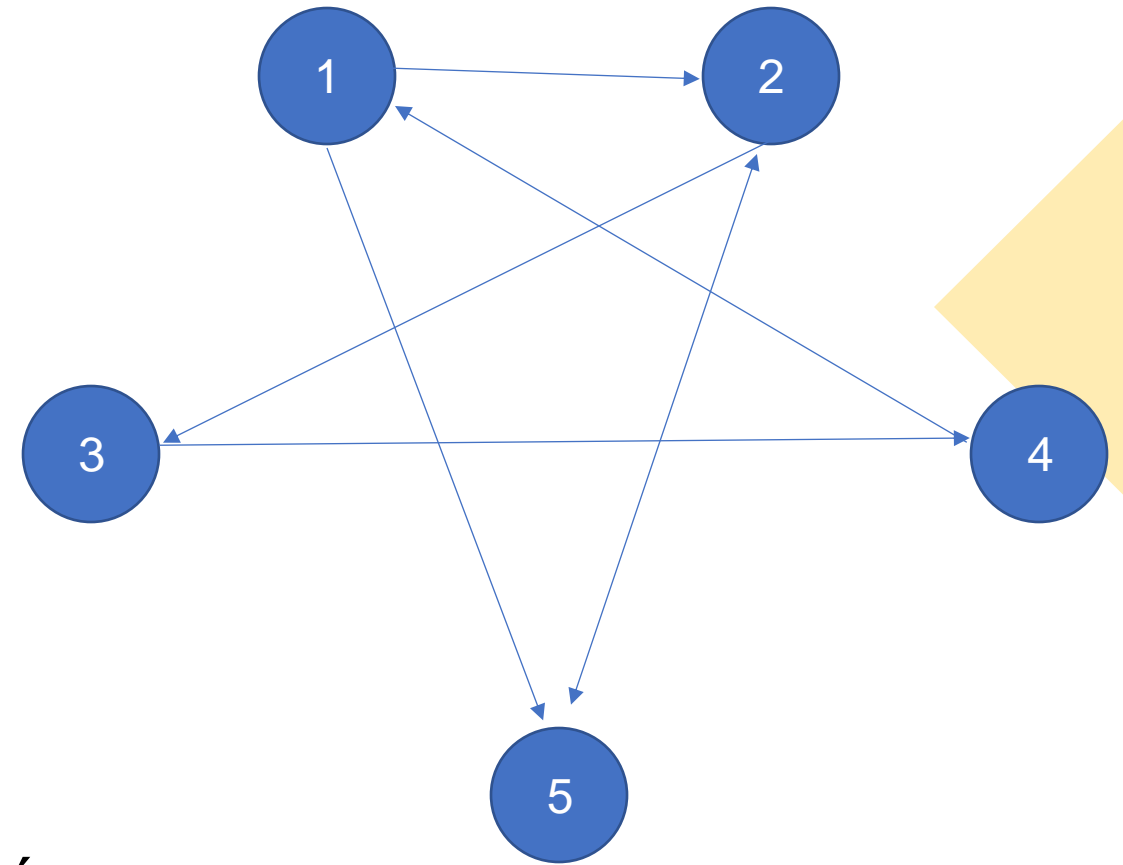
- Depth First Traversal
 - Detect cycle in directed graph
 - Detect cycle in undirected graph
- Breadth First Traversal
 - Unweighted shortest path
 - Weighted shortest path
 - Weighted shortest path with negative weights



Depth First Traversal

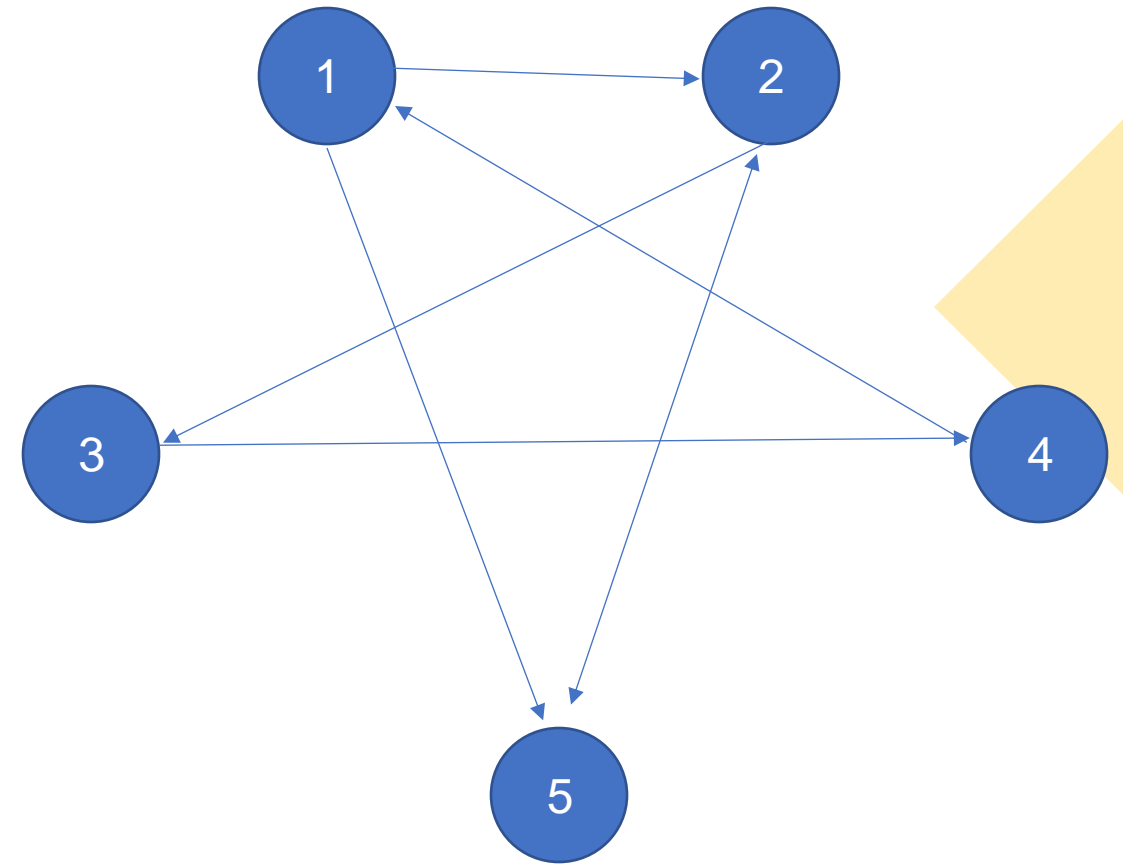
Depth First Traversal là **một thuật toán để duyệt các đỉnh** trong cây tìm kiếm hoặc đồ thị.

Thuật toán bắt đầu từ một node tùy ý (trong trường hợp đồ thị) hoặc node gốc (trong trường hợp cây) và **duyet một nhánh xa nhất có thể trước khi quay lui về node trước đó.**



Depth First Traversal

1. Tạo ra một hàm đệ quy với các tham số là thông số của node và dãy đánh dấu là đã đi qua (visited)
2. Đánh dấu node hiện hành là visited và in ra
3. Duyệt tất cả các node liền kề và chưa đi qua và gọi lại hàm đệ quy với node liền kề.



Depth First Traversal

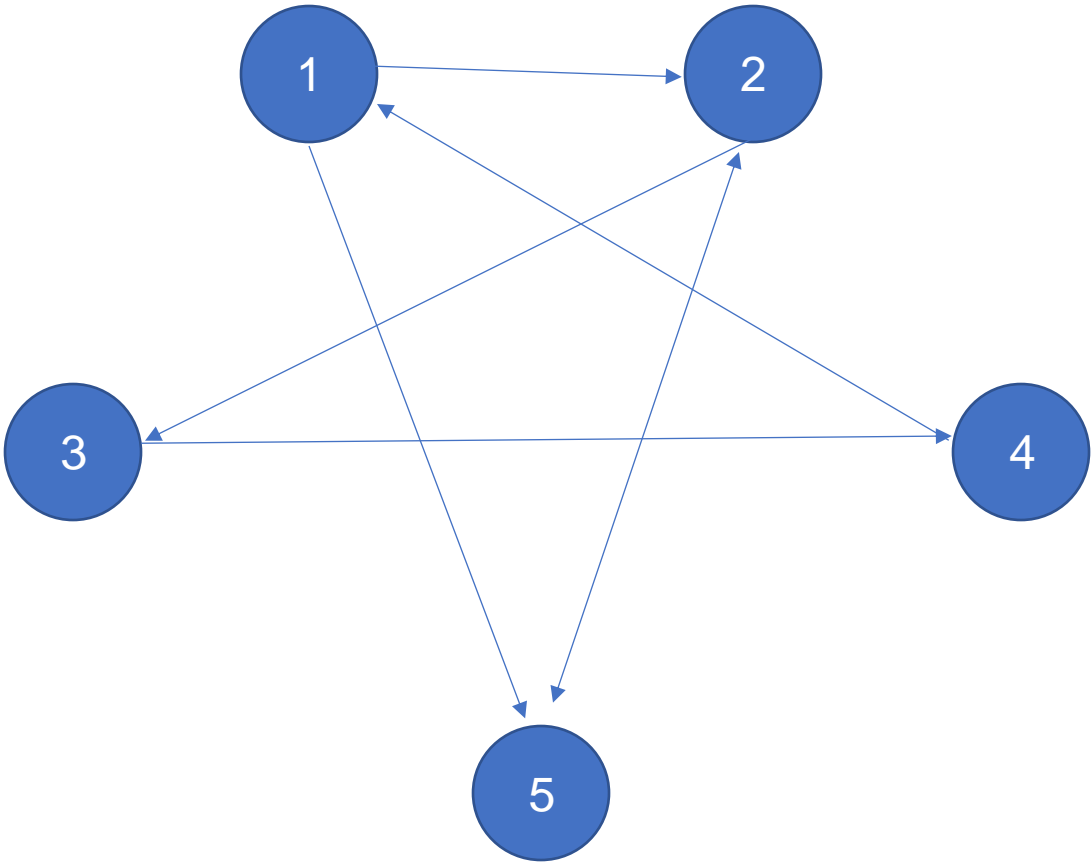
Visited

1	2	3	4	5
0	0	0	0	0

Stack

--	--	--	--	--

Print:



Depth First Traversal

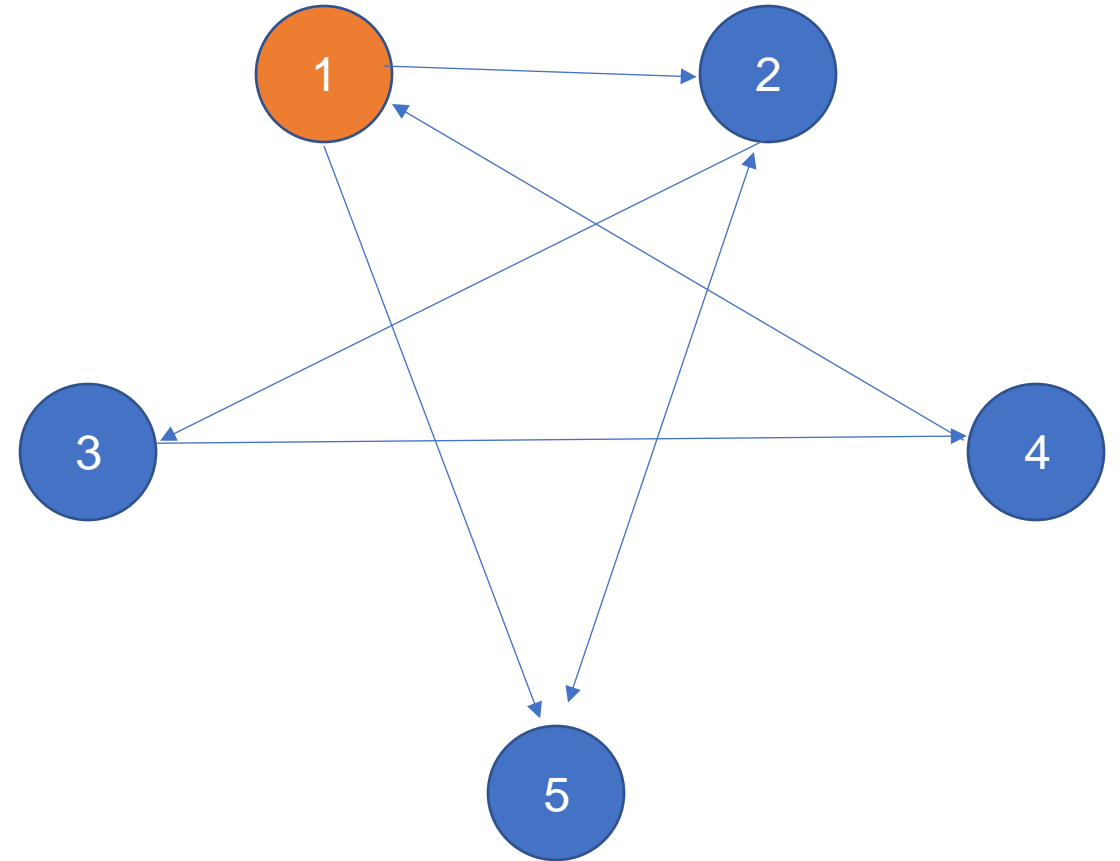
Visited

1	2	3	4	5
T	F	F	F	F

Stack

1				
---	--	--	--	--

Print: 1



Depth First Traversal

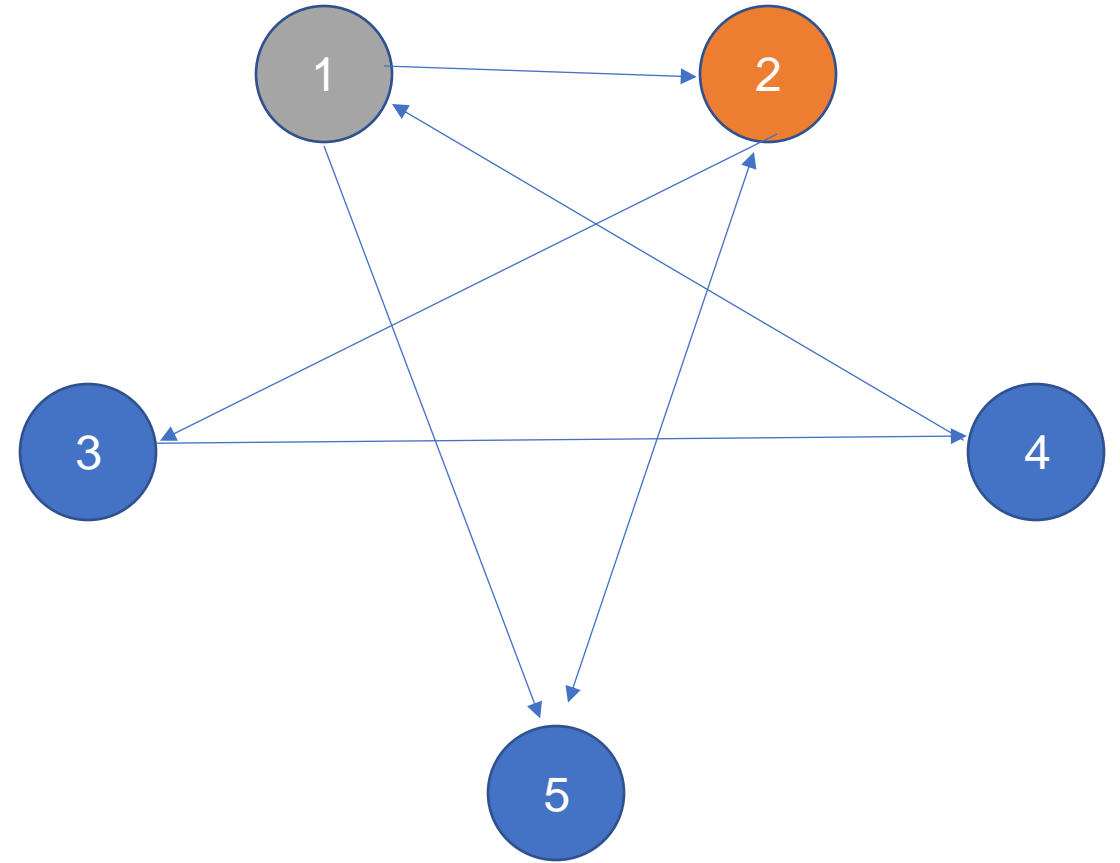
Visited

1	2	3	4	5
T	T	F	F	F

Stack

1	2			
---	---	--	--	--

Print: 1 2



Depth First Traversal

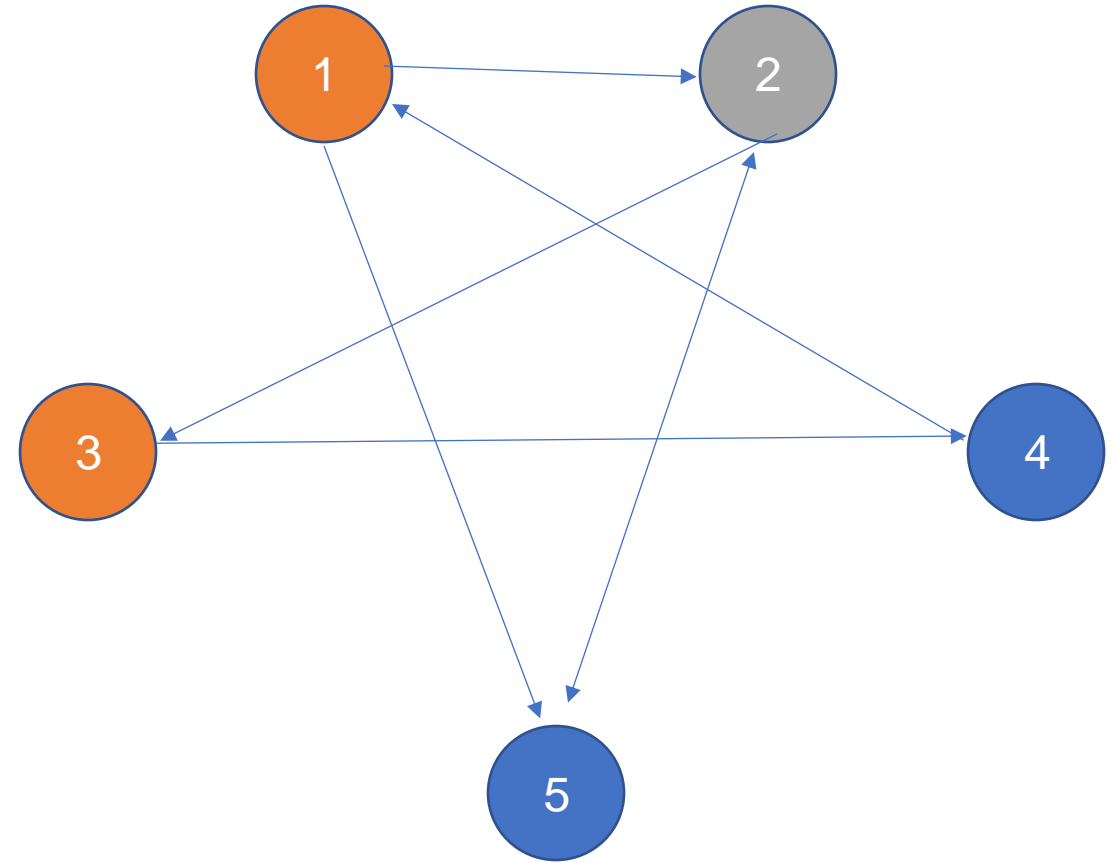
Visited

1	2	3	4	5
T	T	T	F	F

Stack

1	2	3		
---	---	---	--	--

Print: 1 2 3



Depth First Traversal

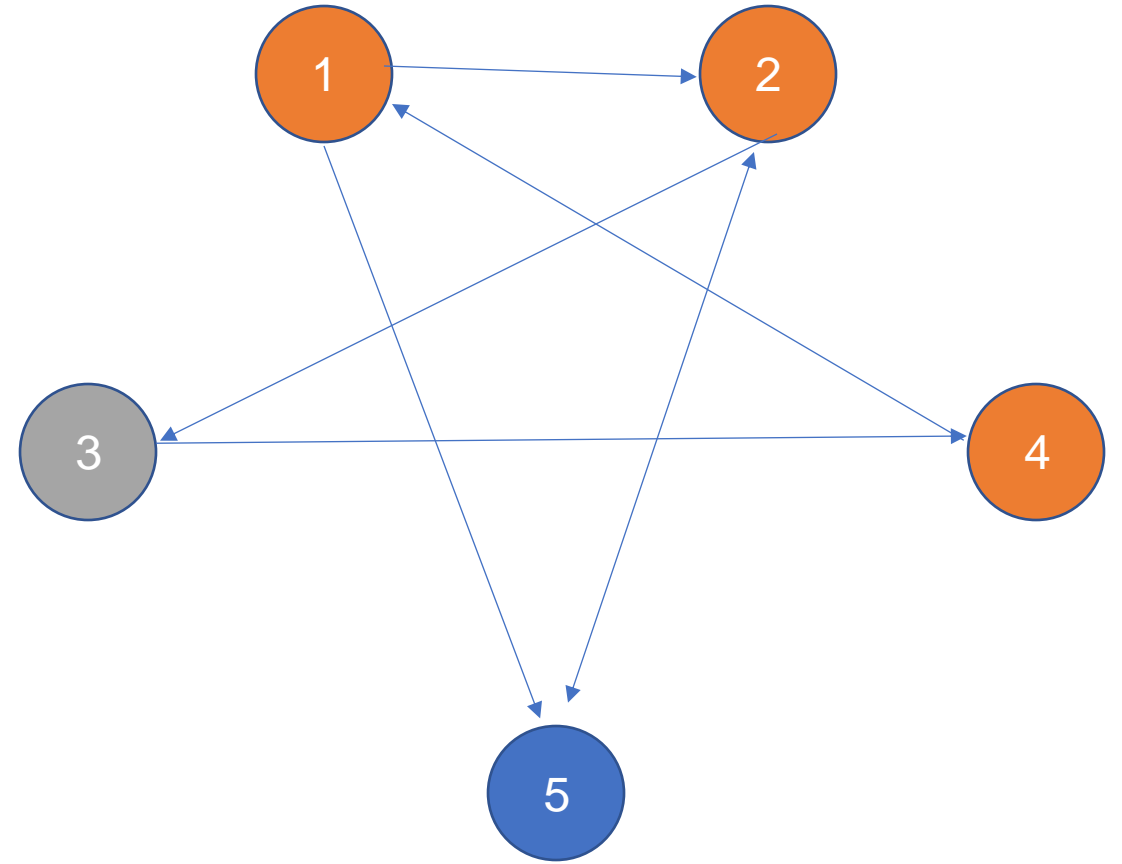
Visited

1	2	3	4	5
T	T	T	T	F

Stack

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

Print: 1 2 3 4



Depth First Traversal

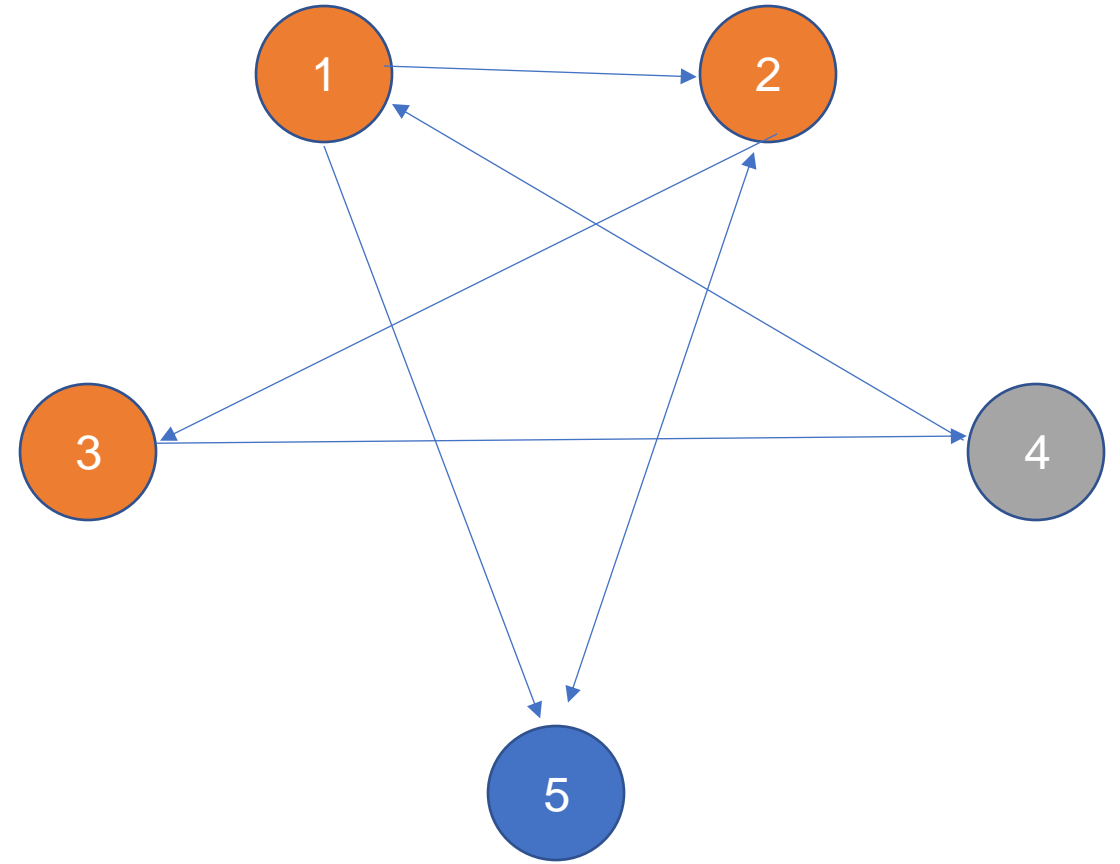
Visited

1	2	3	4	5
T	T	T	T	F

Stack

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

Print: 1 2 3 4



Depth First Traversal

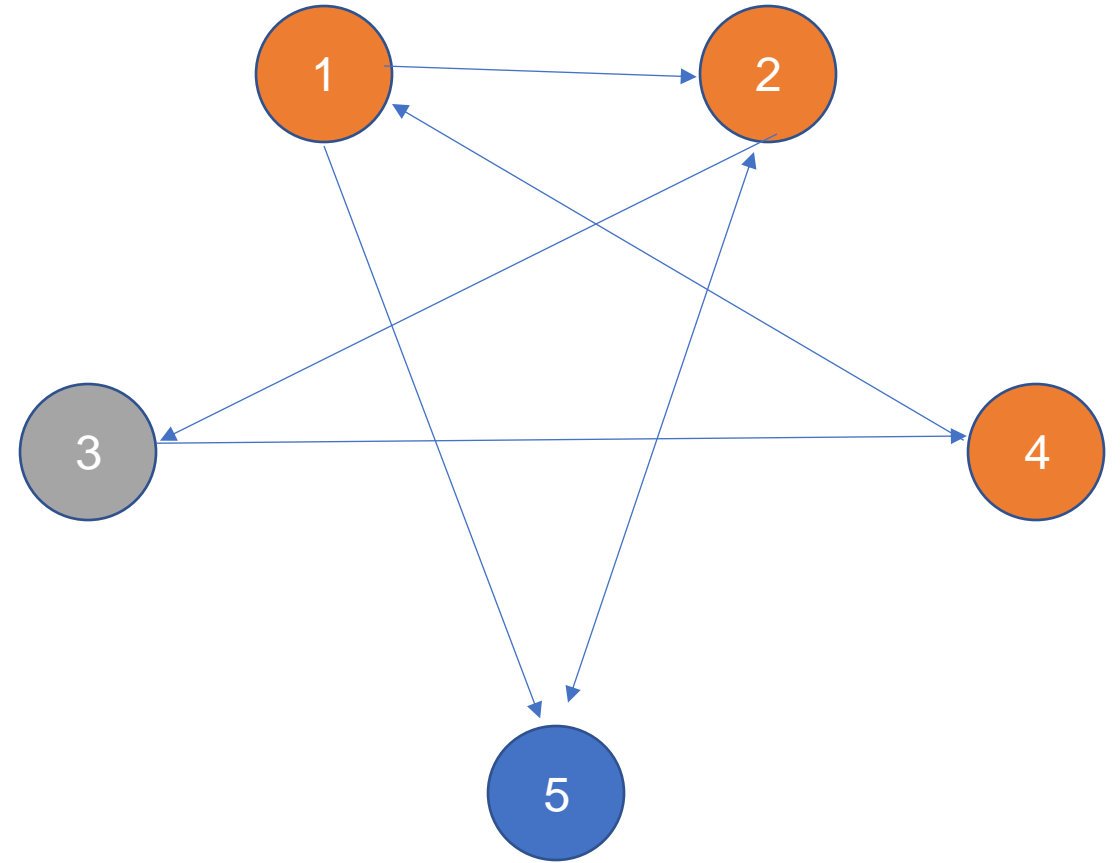
Visited

1	2	3	4	5
T	T	T	T	F

Stack

1	2	3		
---	---	---	--	--

Print: 1 2 3 4



Depth First Traversal

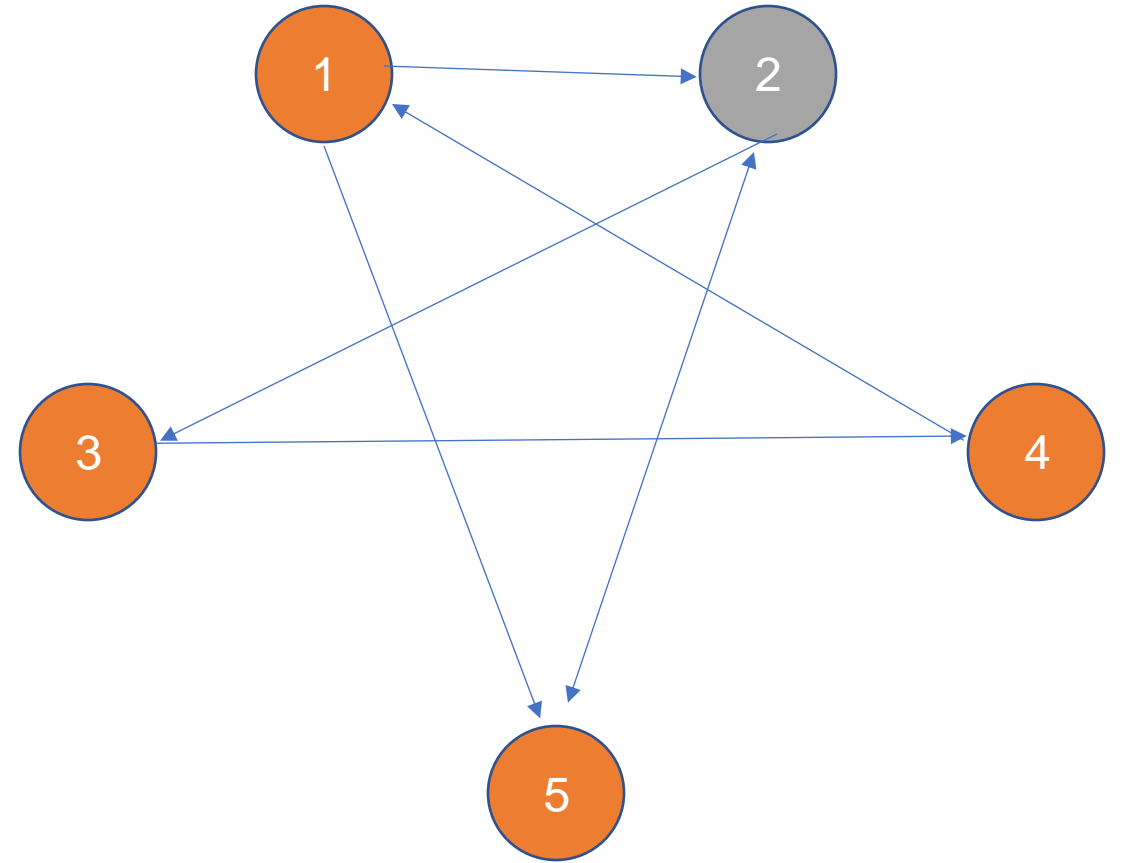
Visited

1	2	3	4	5
T	T	T	T	T

Stack

1	2	5		
---	---	---	--	--

Print: 1 2 3 4 5



Depth First Traversal

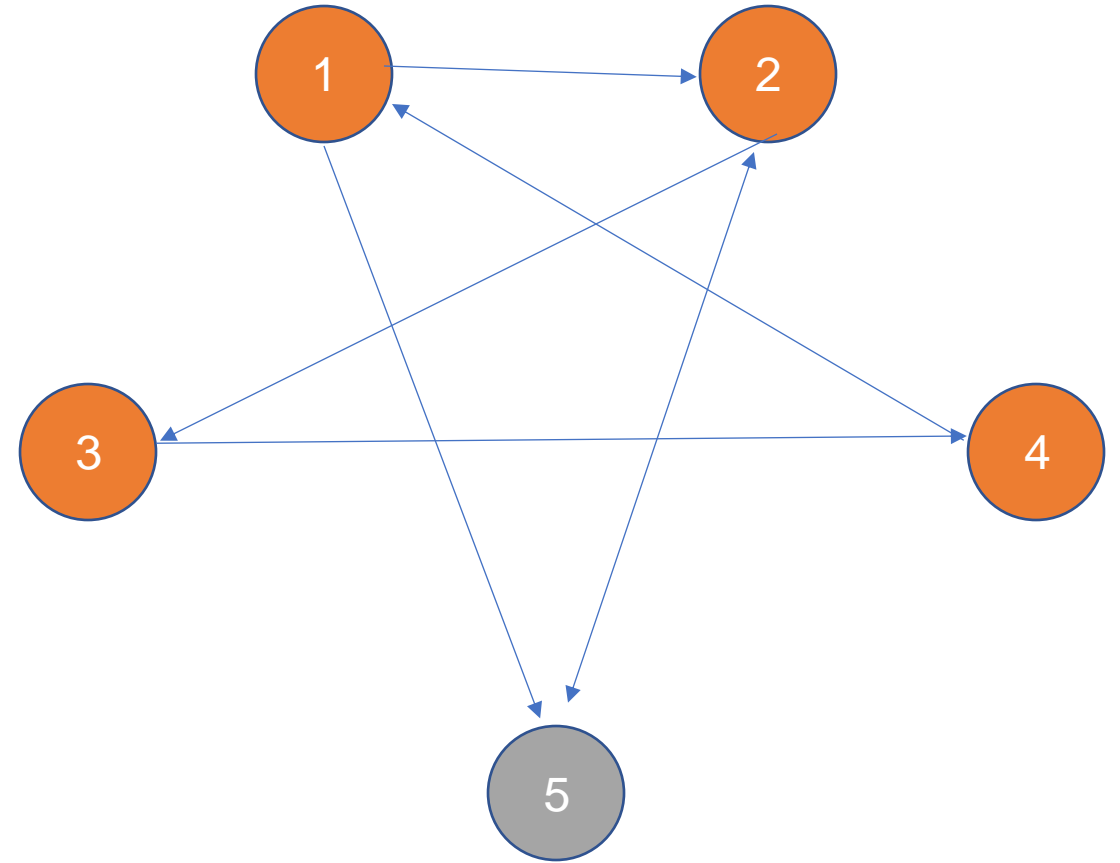
Visited

1	2	3	4	5
T	T	T	T	T

Stack

1	2	5		
---	---	---	--	--

Print: 1 2 3 4 5



Depth First Traversal

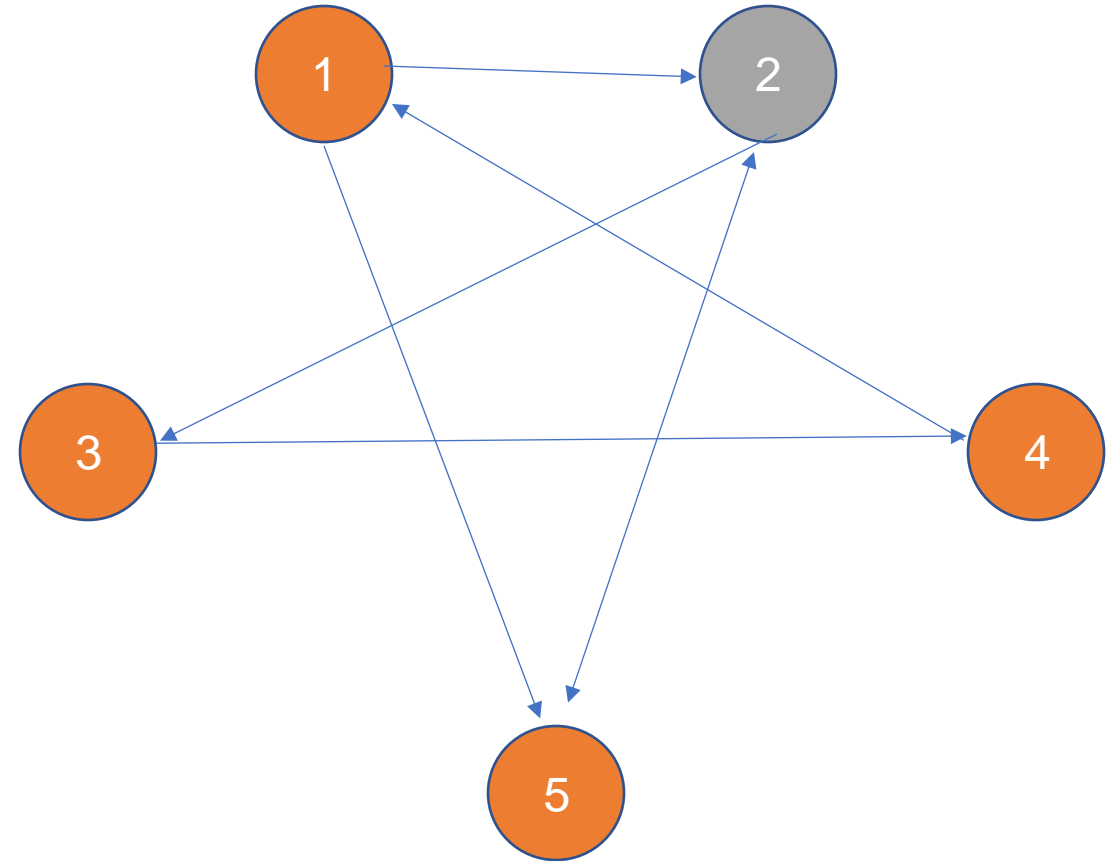
Visited

1	2	3	4	5
T	T	T	T	T

Stack

1	2			
---	---	--	--	--

Print: 1 2 3 4 5



Depth First Traversal

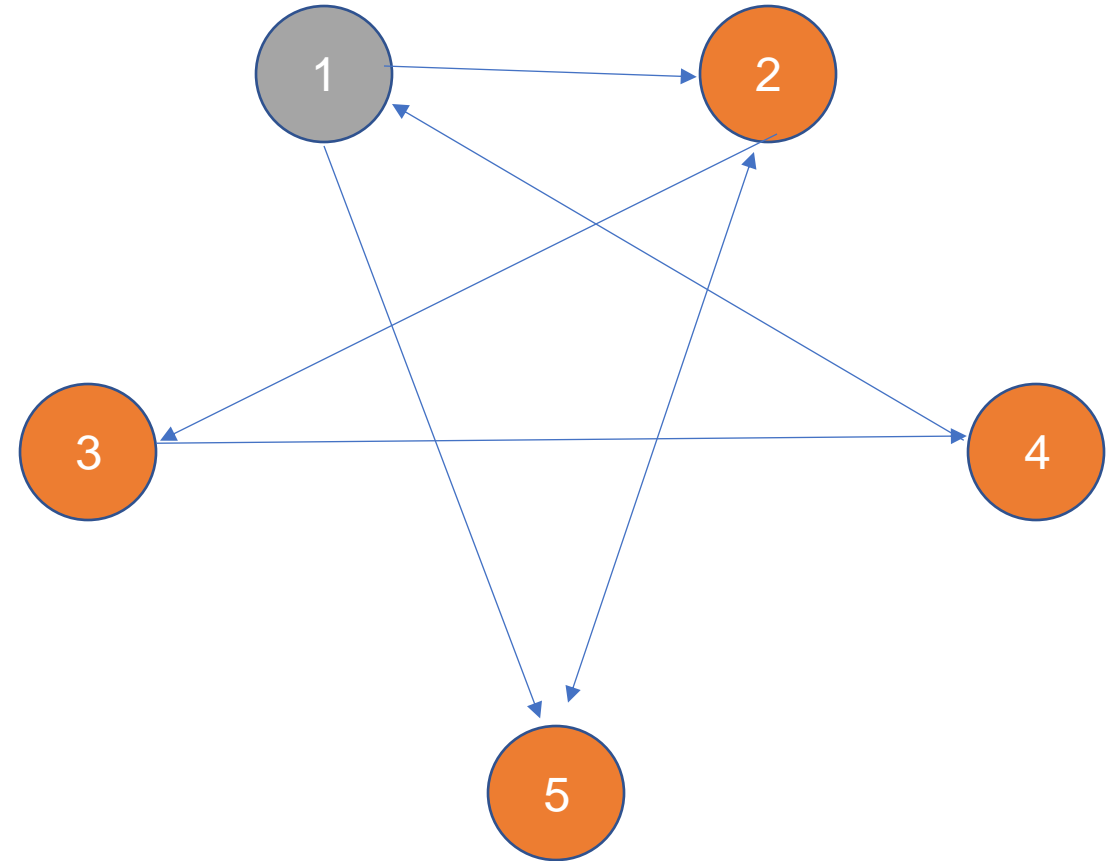
Visited

1	2	3	4	5
T	T	T	T	T

Stack

1				
---	--	--	--	--

Print: 1 2 3 4 5



Depth First Traversal

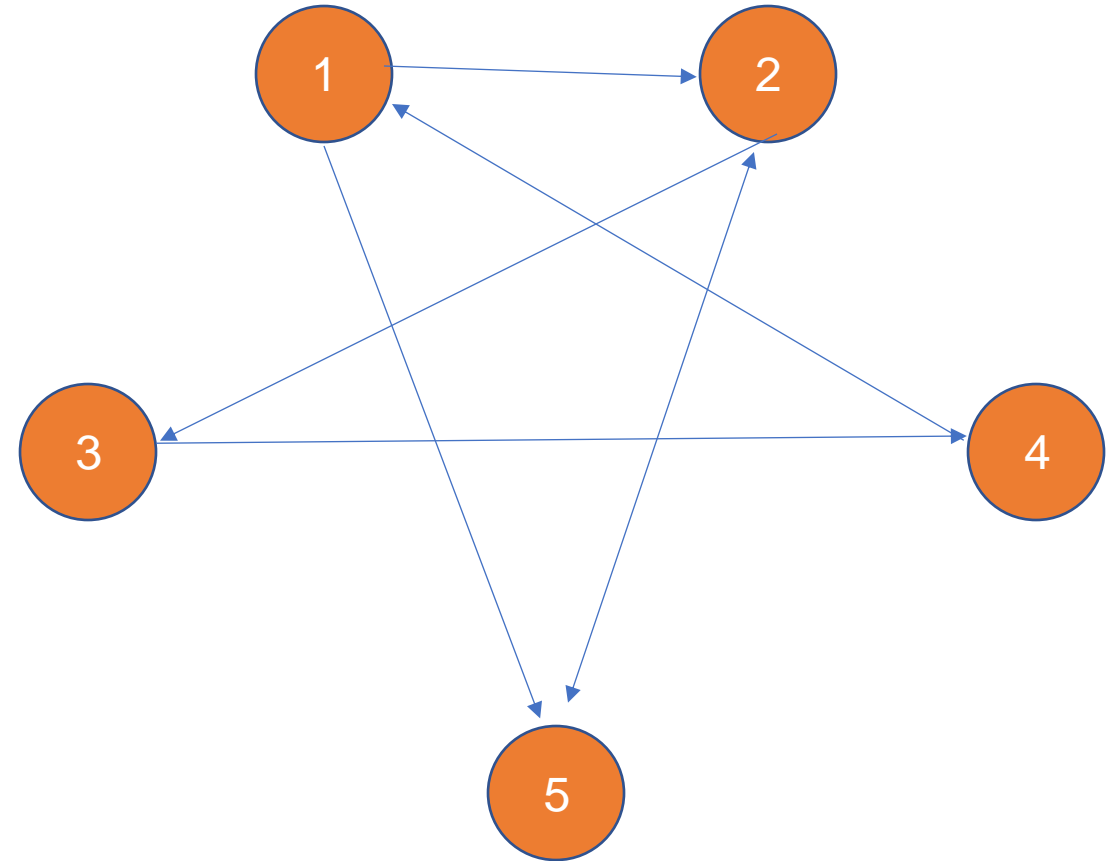
Visited

1	2	3	4	5
T	T	T	T	T

Stack

--	--	--	--	--

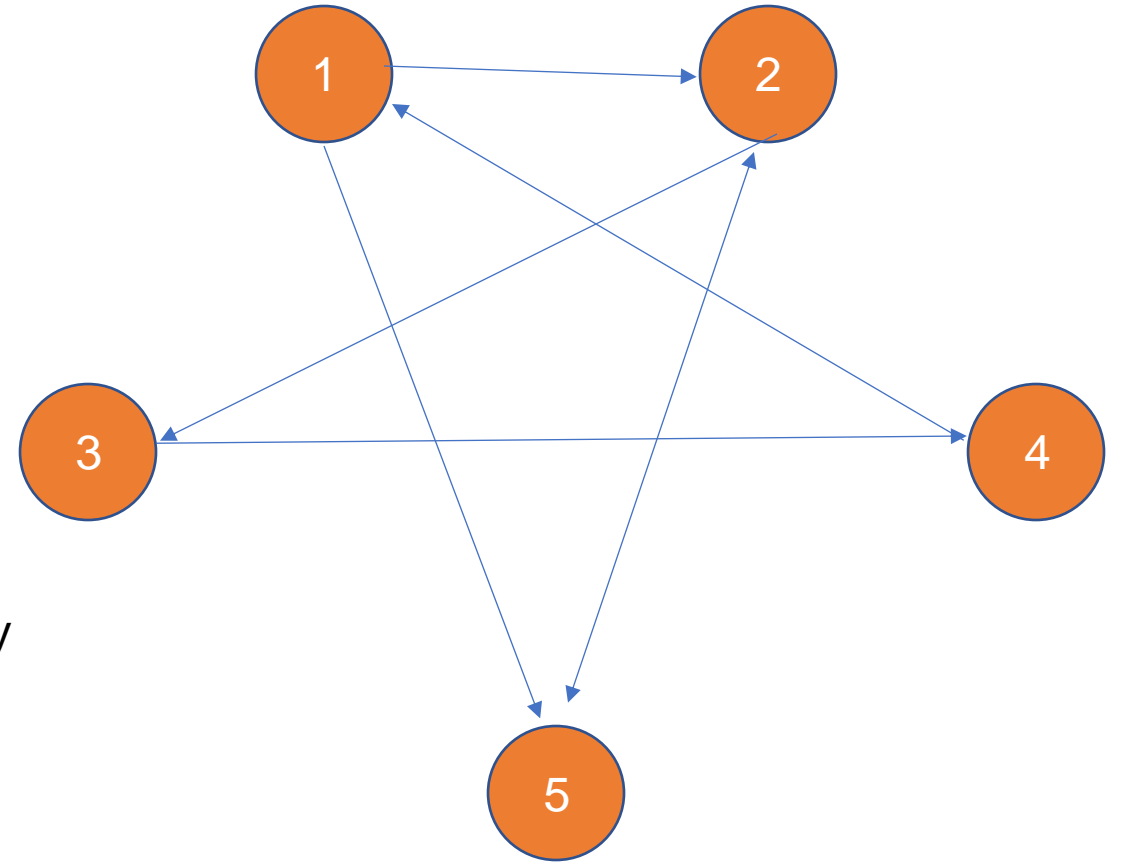
Print: 1 2 3 4 5



Depth First Traversal

Độ phức tạp của thuật toán:

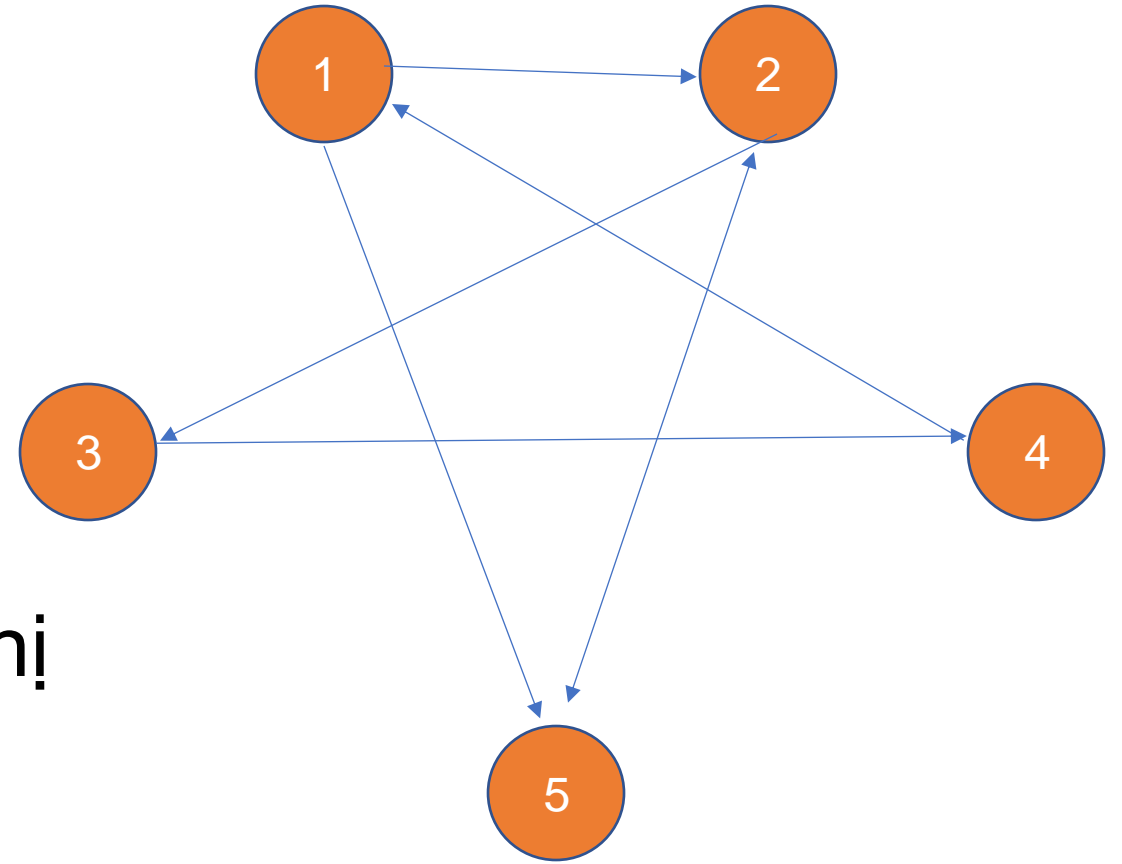
- ✓ Thời gian: $O(V+E)$ với V là số đỉnh và E là số cạnh
- ✓ Không gian: $O(V)$ vì cần thêm không gian để lưu dãy visited có V phần tử



Depth First Traversal

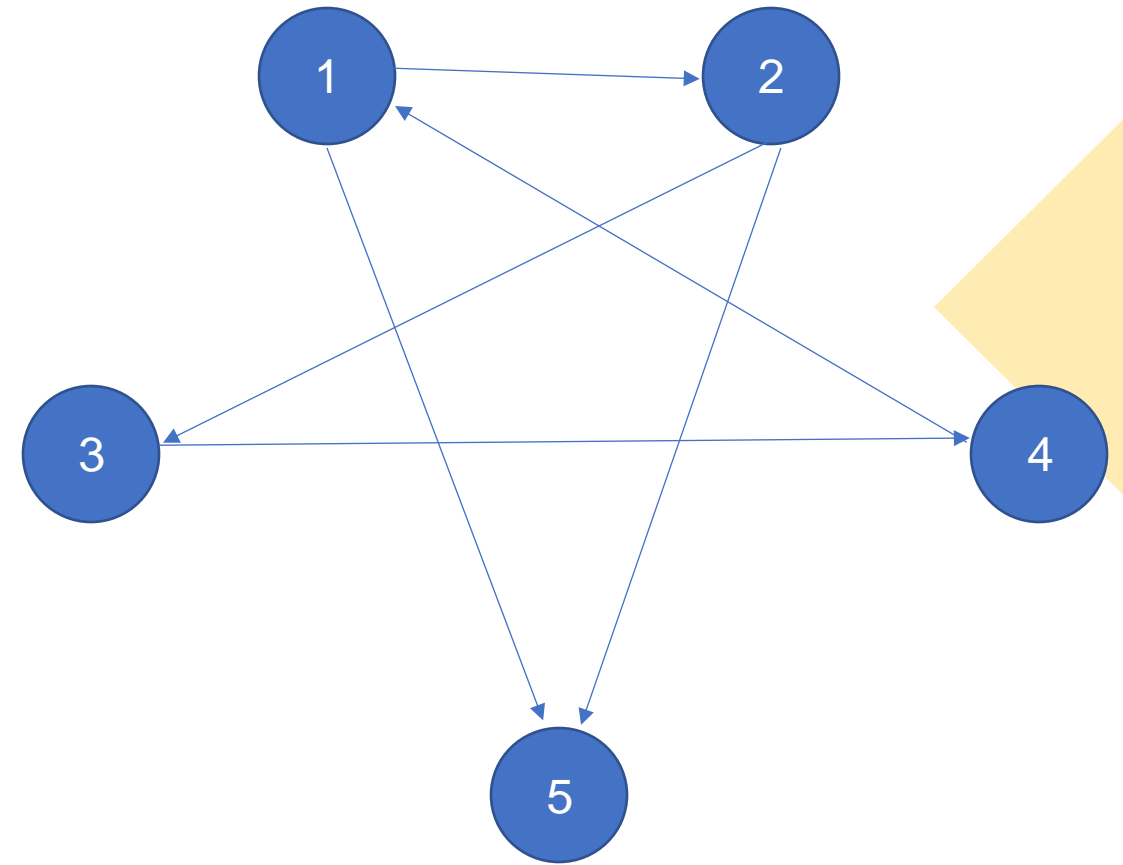
Ứng dụng:

- ✓ Phát hiện chu trình trong đồ thị
- ✓ Tìm đường đi
- ✓ Sắp xếp Topo
- ✓ Tạo mê cung ngẫu nhiên



Detect cycle in directed graph

Một bảng đăng kí môn học có các môn học được đánh mã từ 1 đến 5. Ngoại trừ môn học có mã 1, các môn học khác đều có điều kiện tiên quyết của mình. Hãy kiểm tra bảng đăng kí này có hợp lệ hay không.

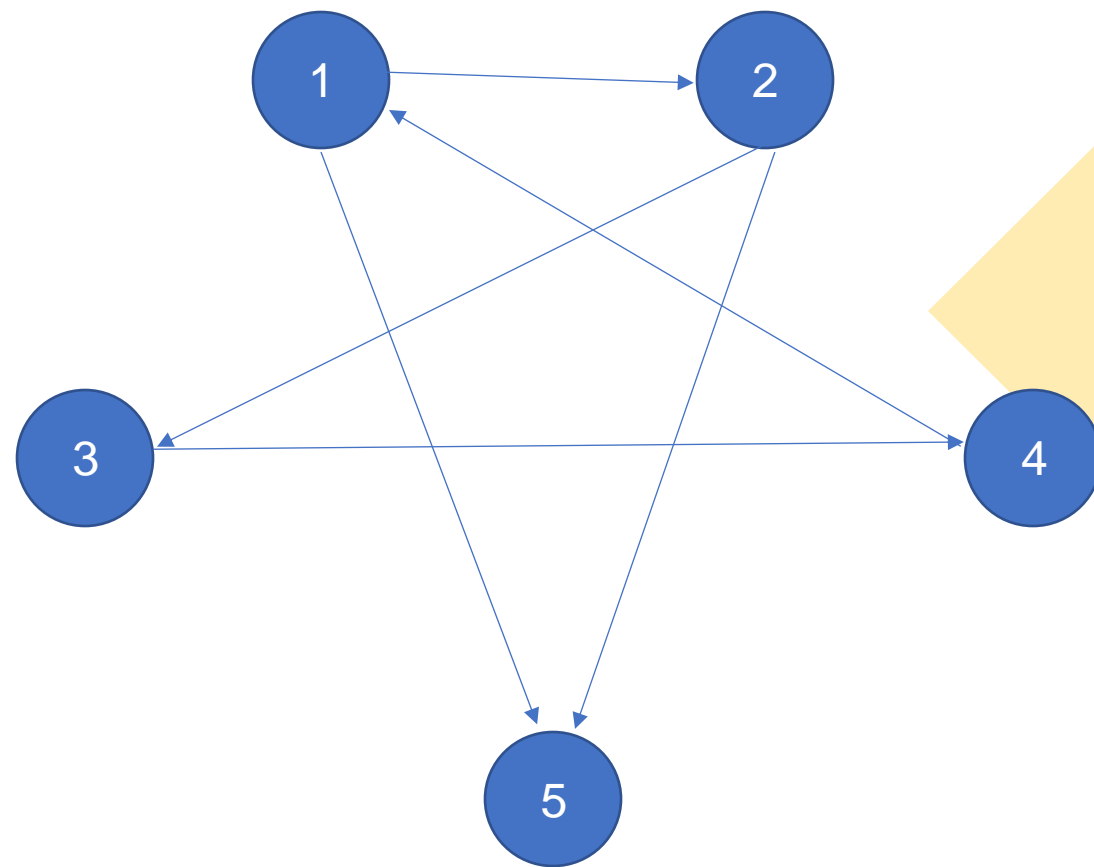


Detect cycle in directed graph

1. Tạo ra một hàm đệ quy với các tham số là thông số của node và dãy đánh dấu là đã đi qua (visited)

2. Đánh dấu node hiện hành là visited. Nếu node hiện hành đã được đánh dấu là visited thì return true.

3. Gọi hàm đệ quy với tham số là thông số của node tiếp theo.



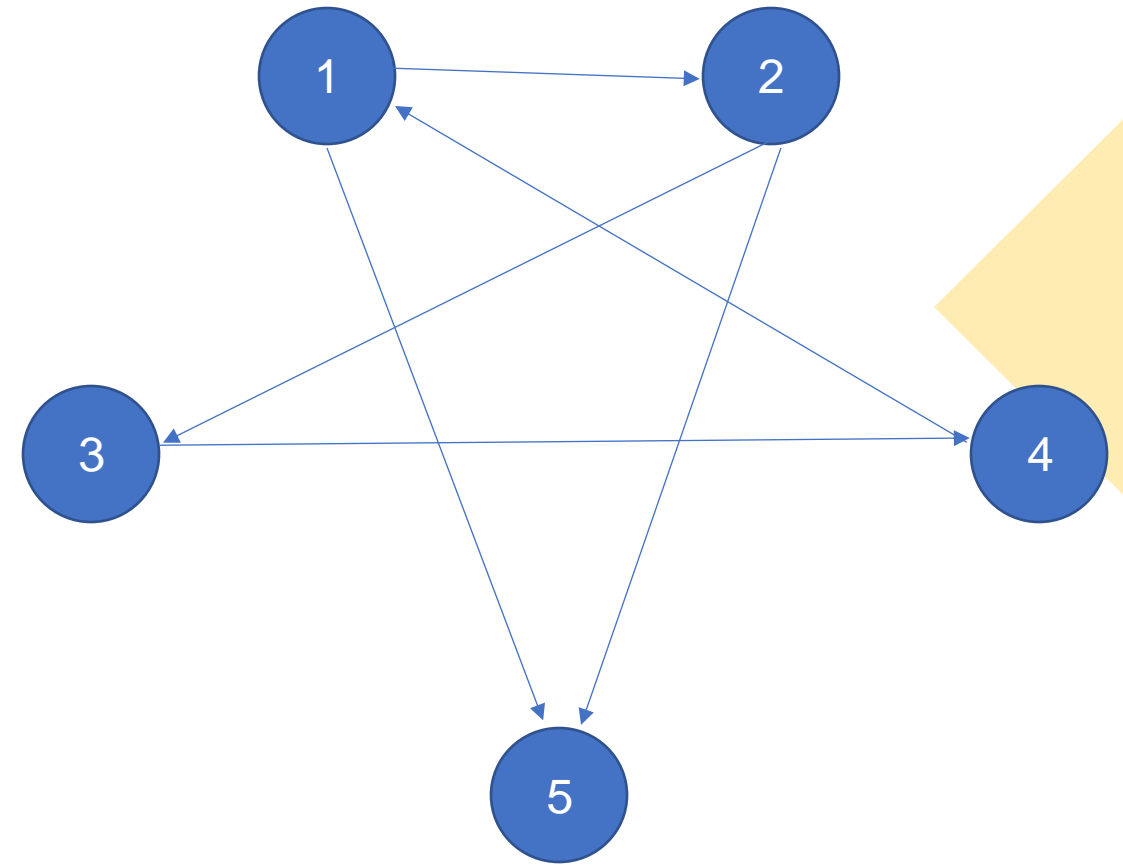
Detect cycle in directed graph

Visited

1	2	3	4	5
F	F	F	F	F

Stack

--	--	--	--	--



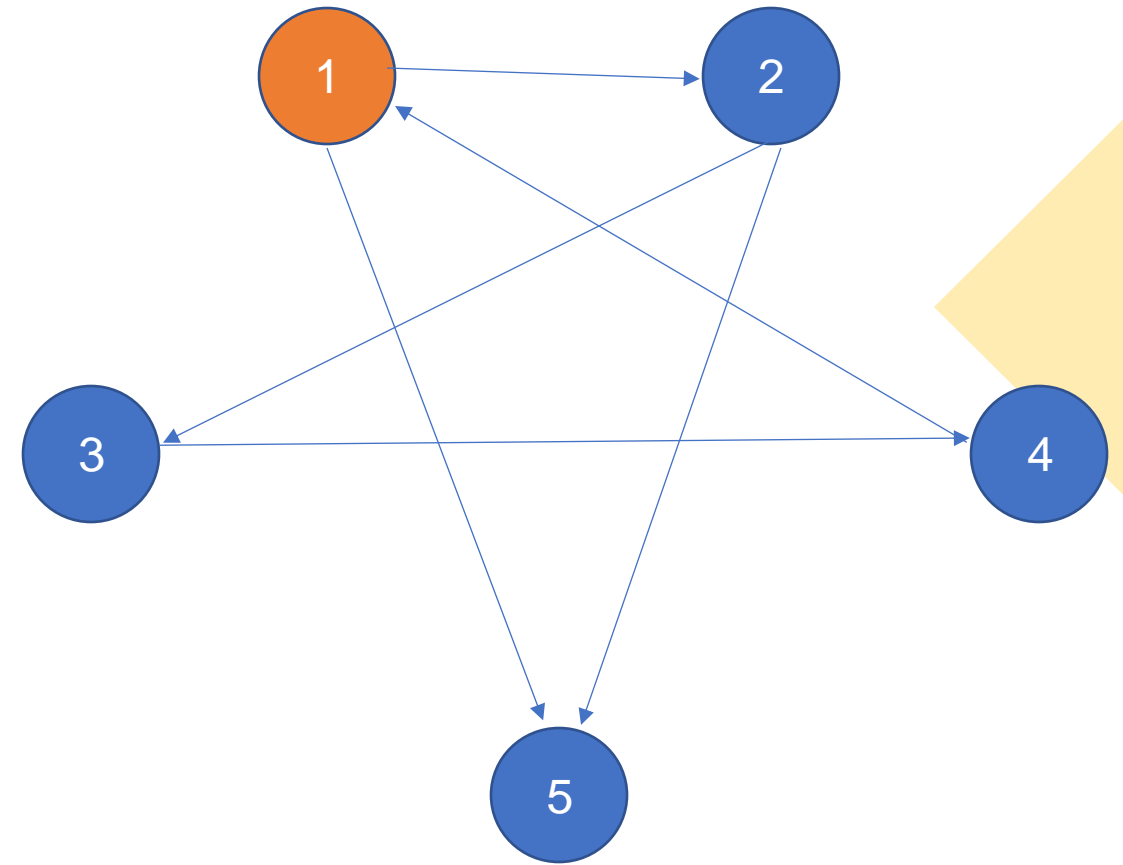
Detect cycle in directed graph

Visited

1	2	3	4	5
T	F	F	F	F

Stack

1				
---	--	--	--	--



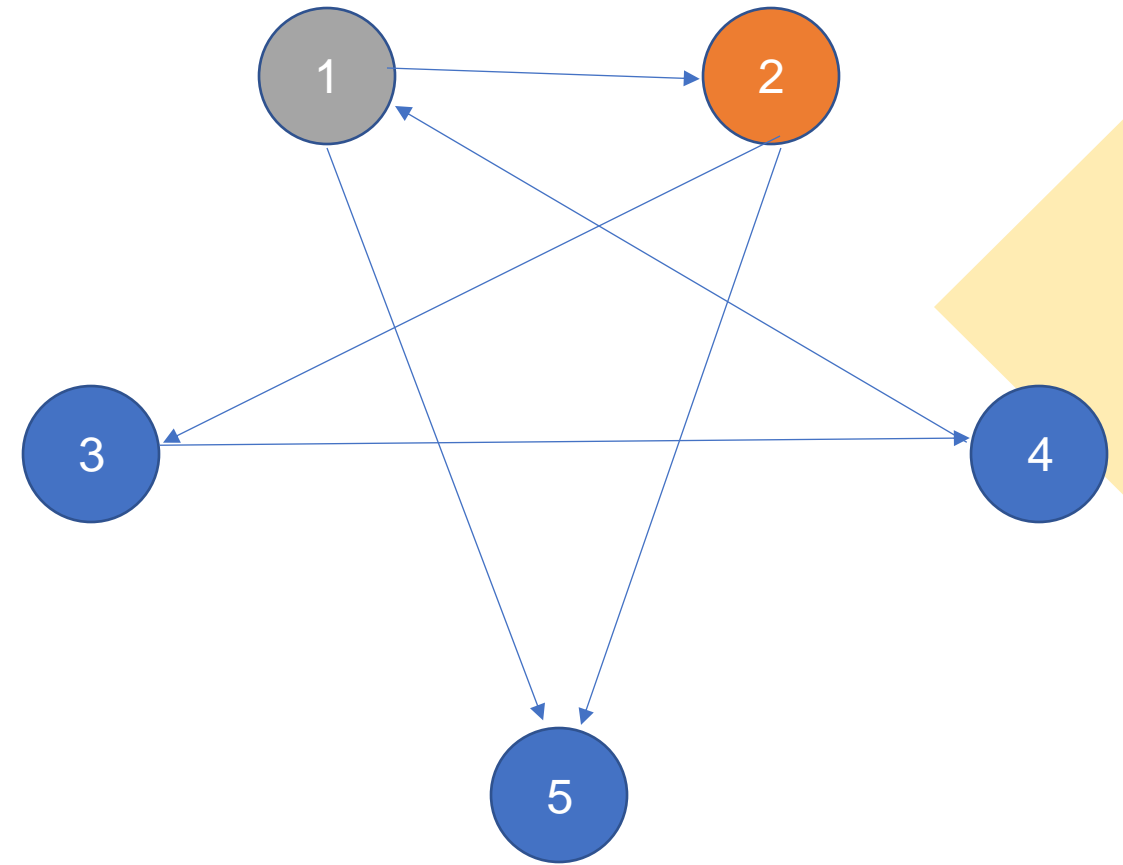
Detect cycle in directed graph

Visited

1	2	3	4	5
T	T	F	F	F

Stack

1	2			
---	---	--	--	--



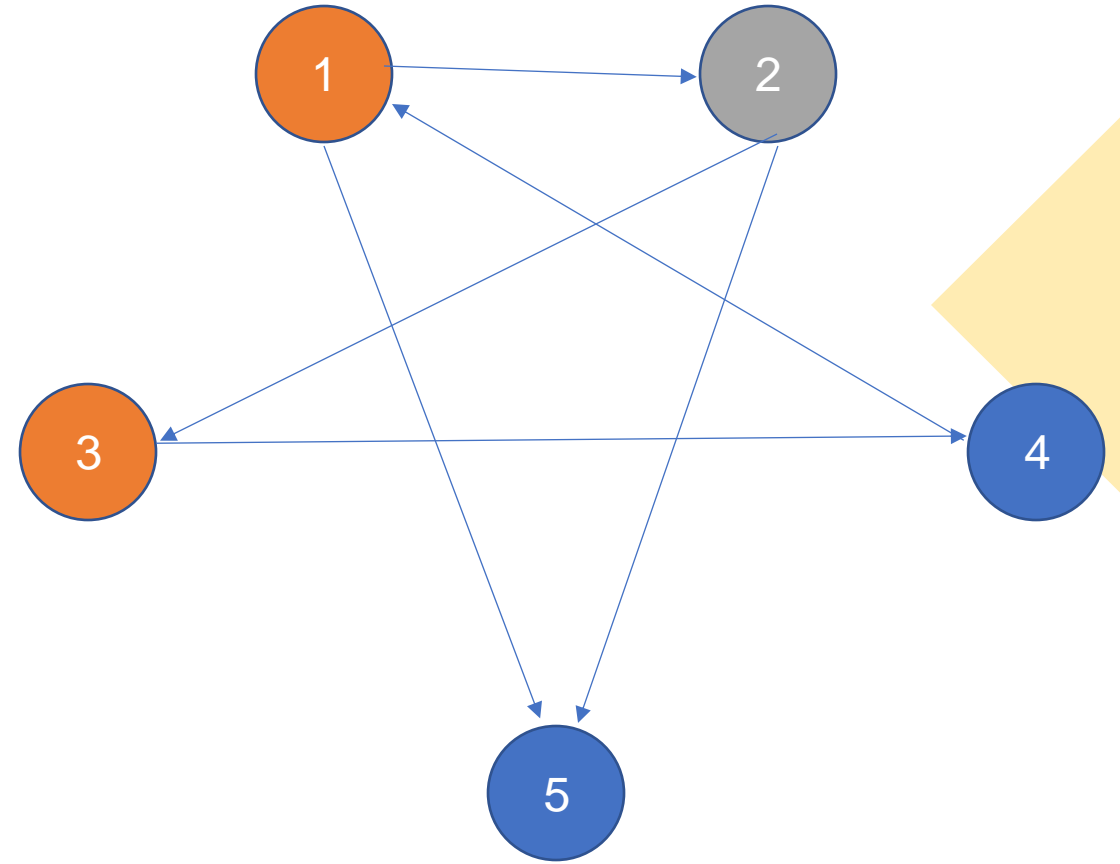
Detect cycle in directed graph

Visited

1	2	3	4	5
T	T	T	F	F

Stack

1	2	3		
---	---	---	--	--



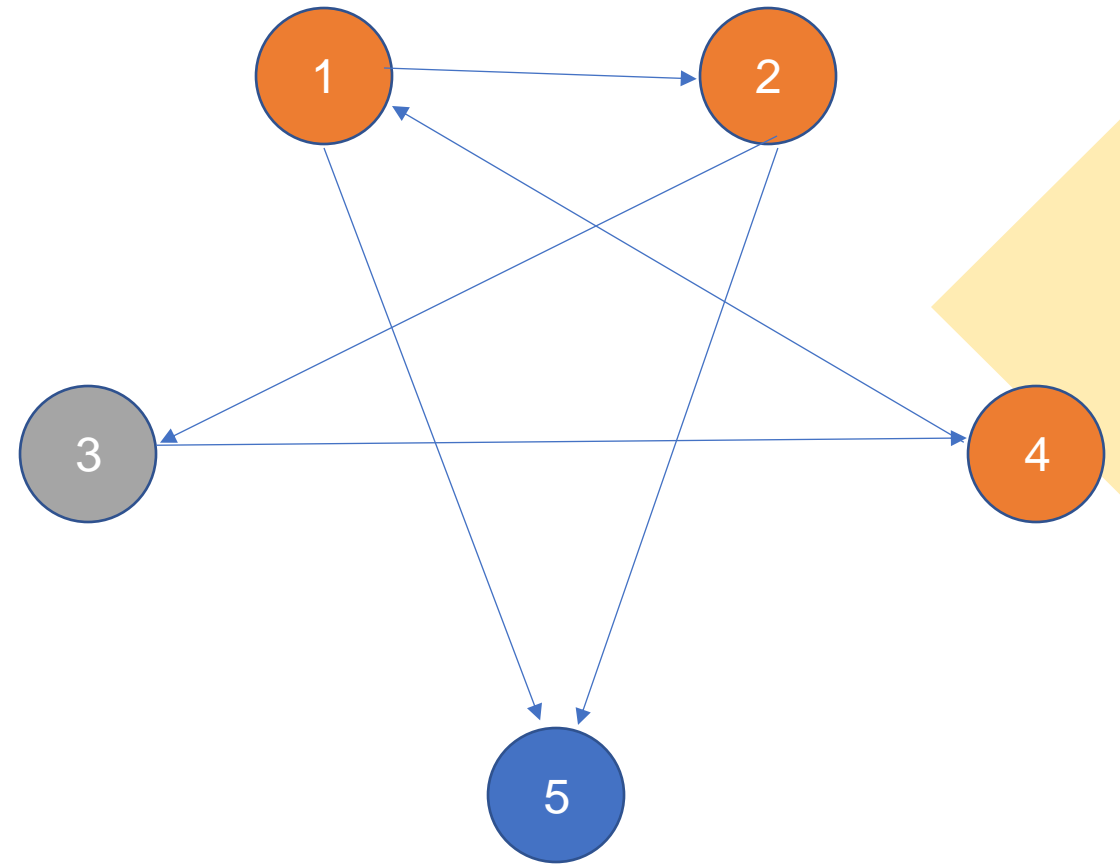
Detect cycle in directed graph

Visited

1	2	3	4	5
T	T	T	T	F

Stack

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



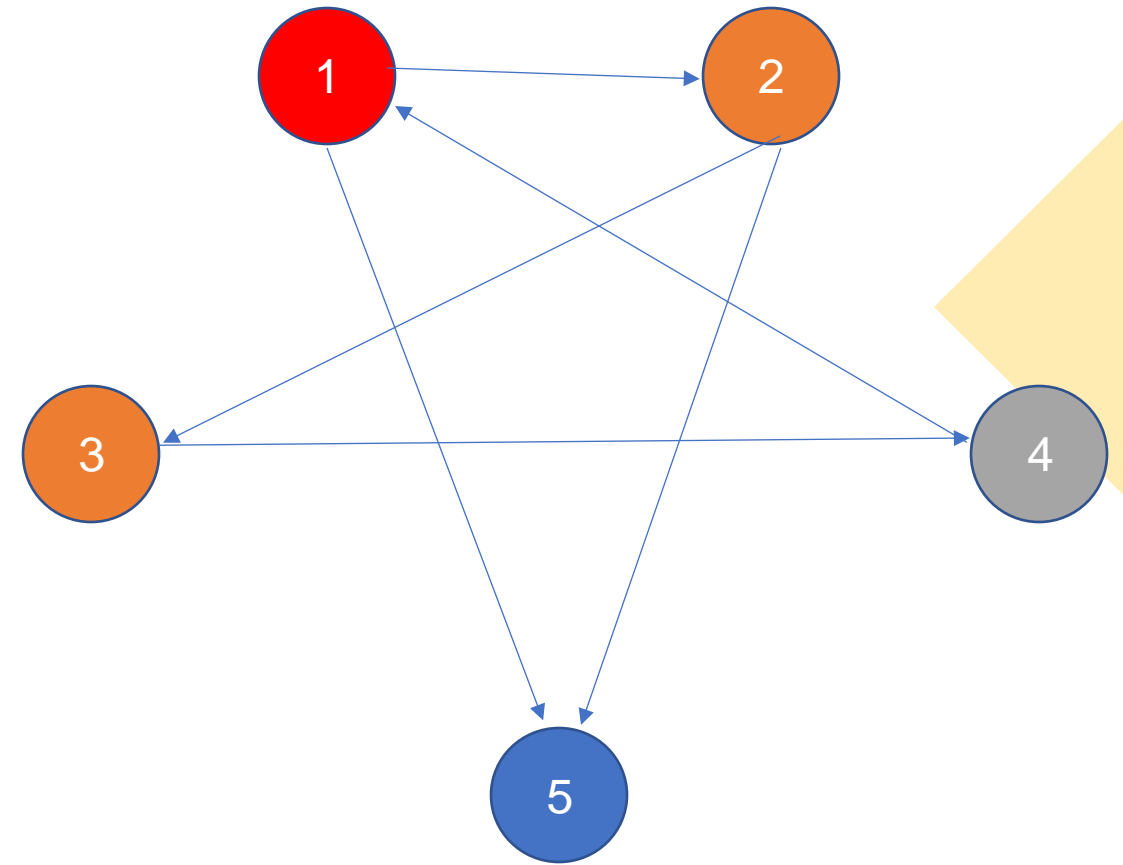
Detect cycle in directed graph

Visited

1	2	3	4	5
T	T	T	T	F

Stack

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

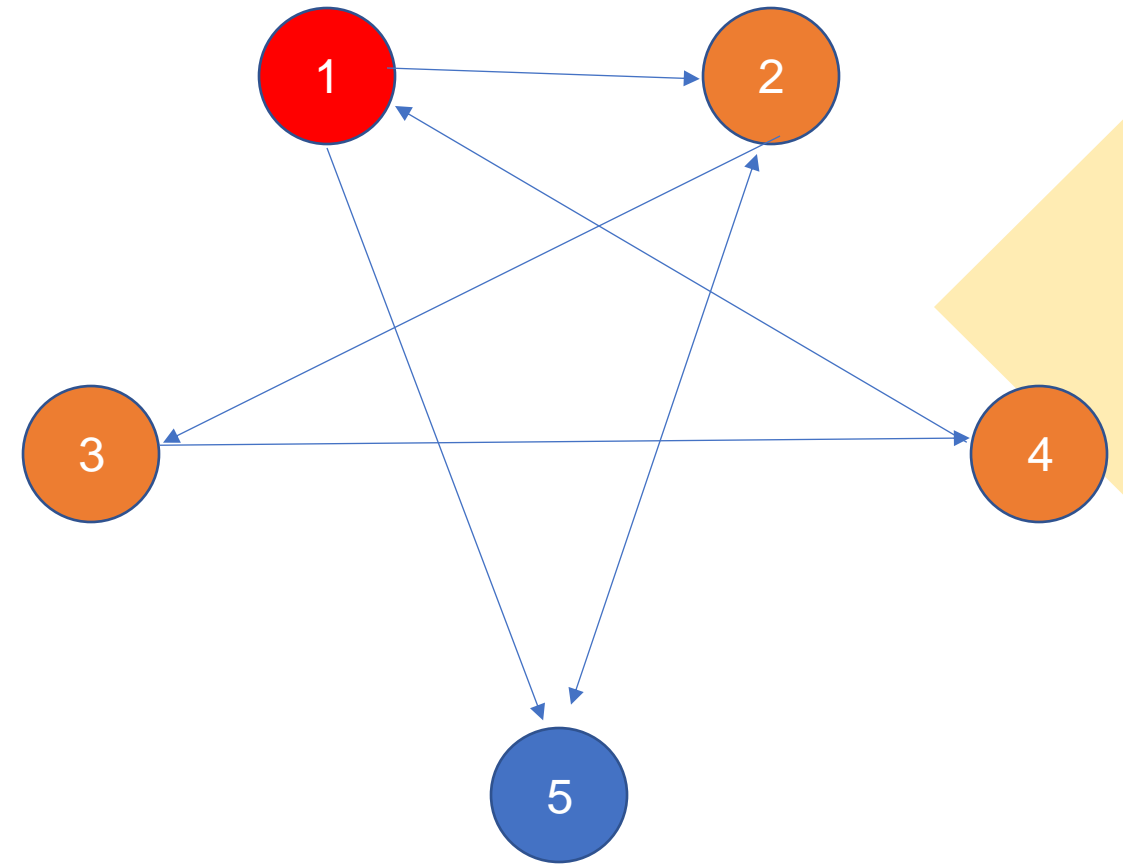


Return True

Detect cycle in directed graph

Độ phức tạp thuật toán

- ✓ Thời gian: $O(V+E)$
- ✓ Không gian: $O(V)$



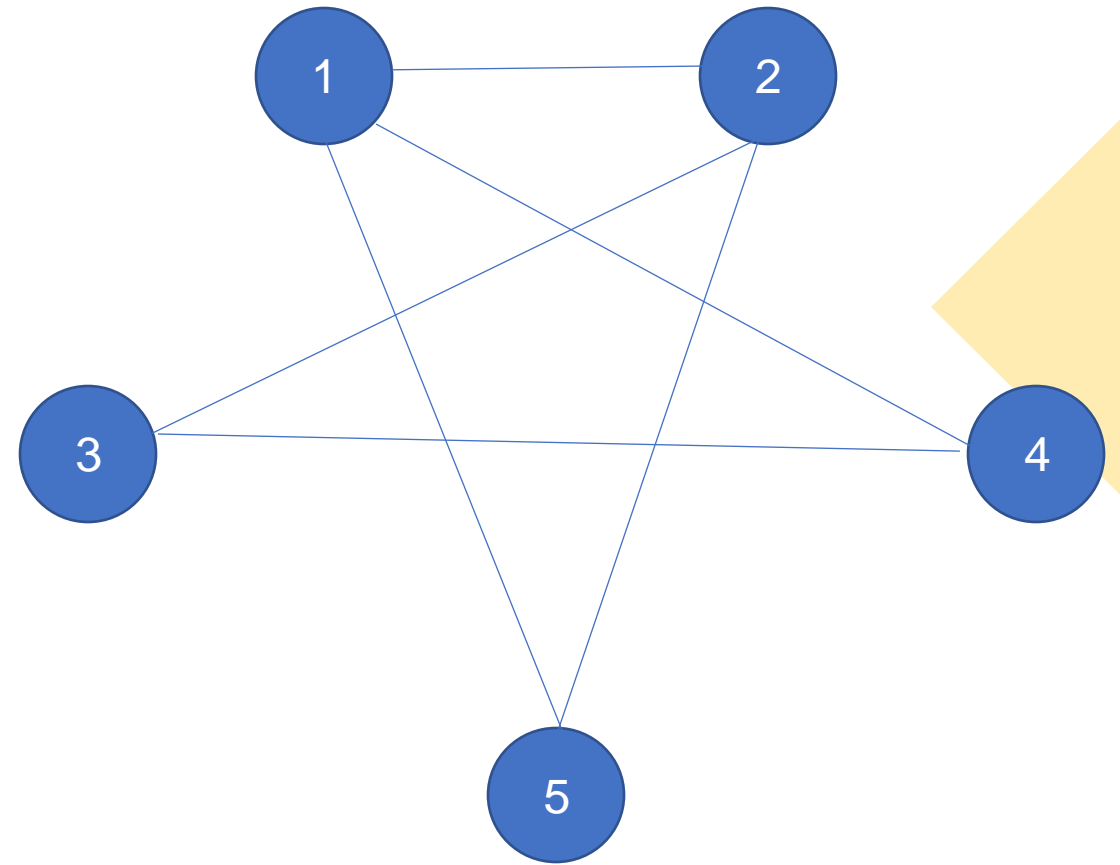
Return True

Detect cycle in undirected graph

Cho một đồ thị vô hướng. Xác định có chu trình nào không?

```
for v in graph[u]:  
    if v == parent(u):  
        continue  
    if v not in visited:  
        .....//Duyệt node tiếp  
    Else:  
        return True
```

Tạo thêm một biến parent để lưu lại giá trị của node trước của node hiện hành.



Detect cycle in undirected graph

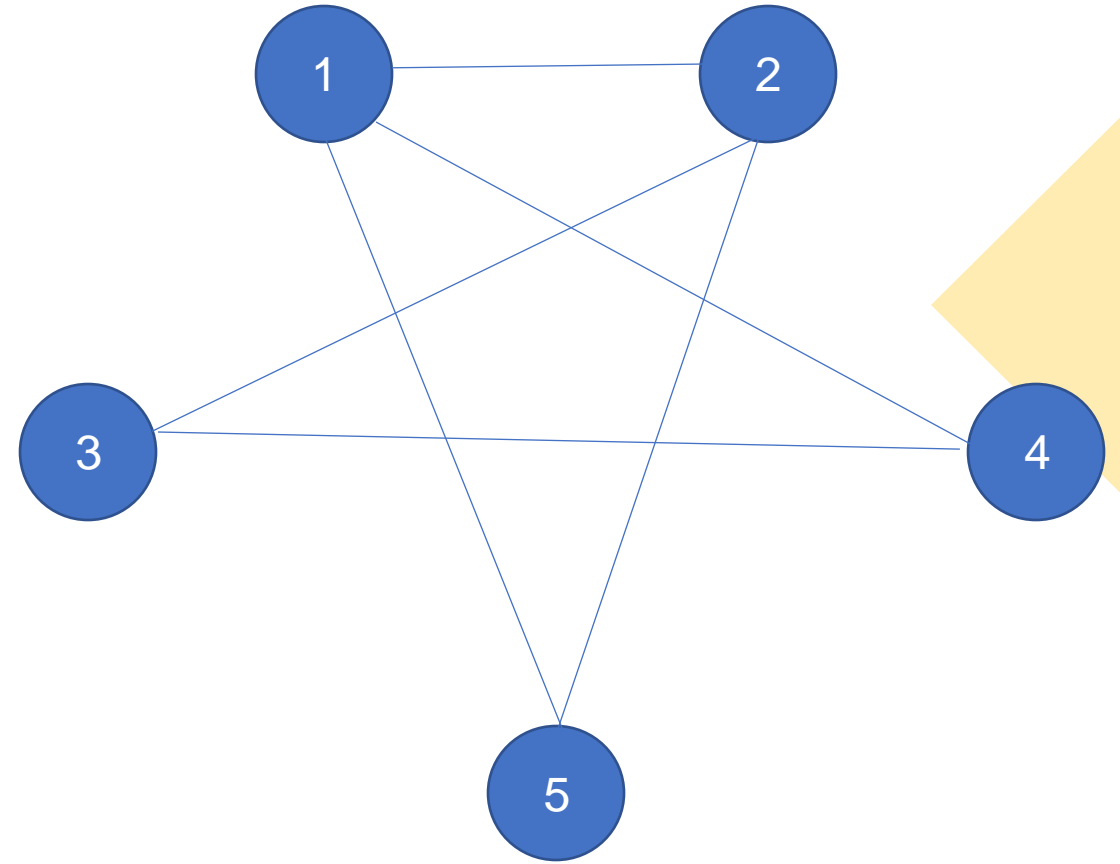
Visited

1	2	3	4	5
F	F	F	F	F

Stack

--	--	--	--	--

Parent:



Detect cycle in undirected graph

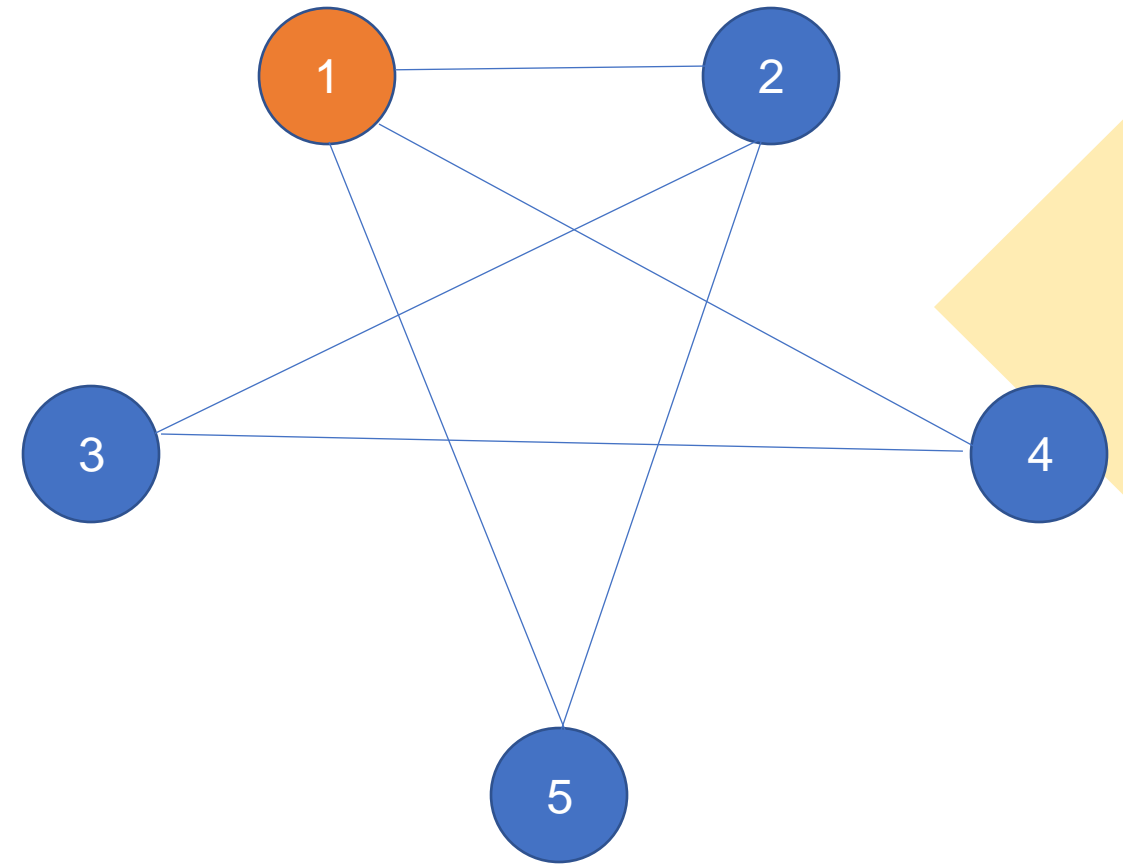
Visited

1	2	3	4	5
T	F	F	F	F

Stack

1				
---	--	--	--	--

Parent:



Detect cycle in undirected graph

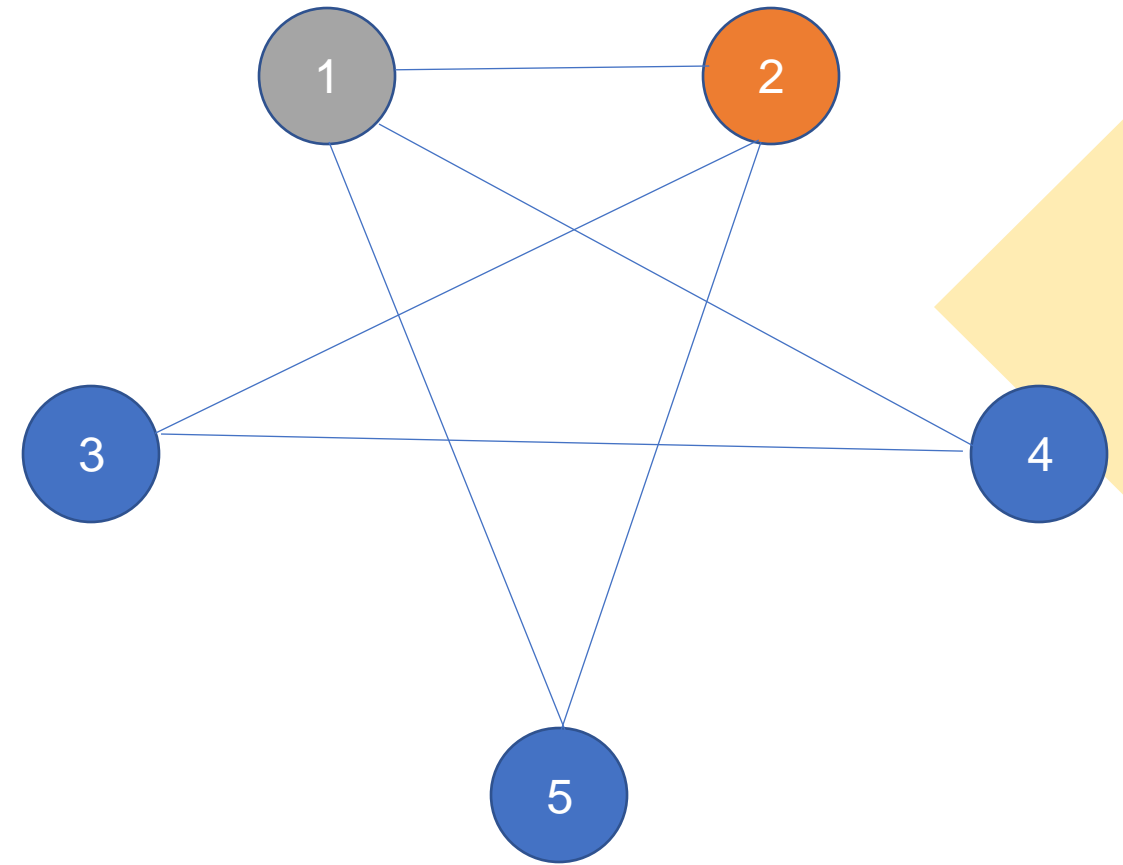
Visited

1	2	3	4	5
T	T	F	F	F

Stack

1	2			
---	---	--	--	--

Parent: -1



Detect cycle in undirected graph

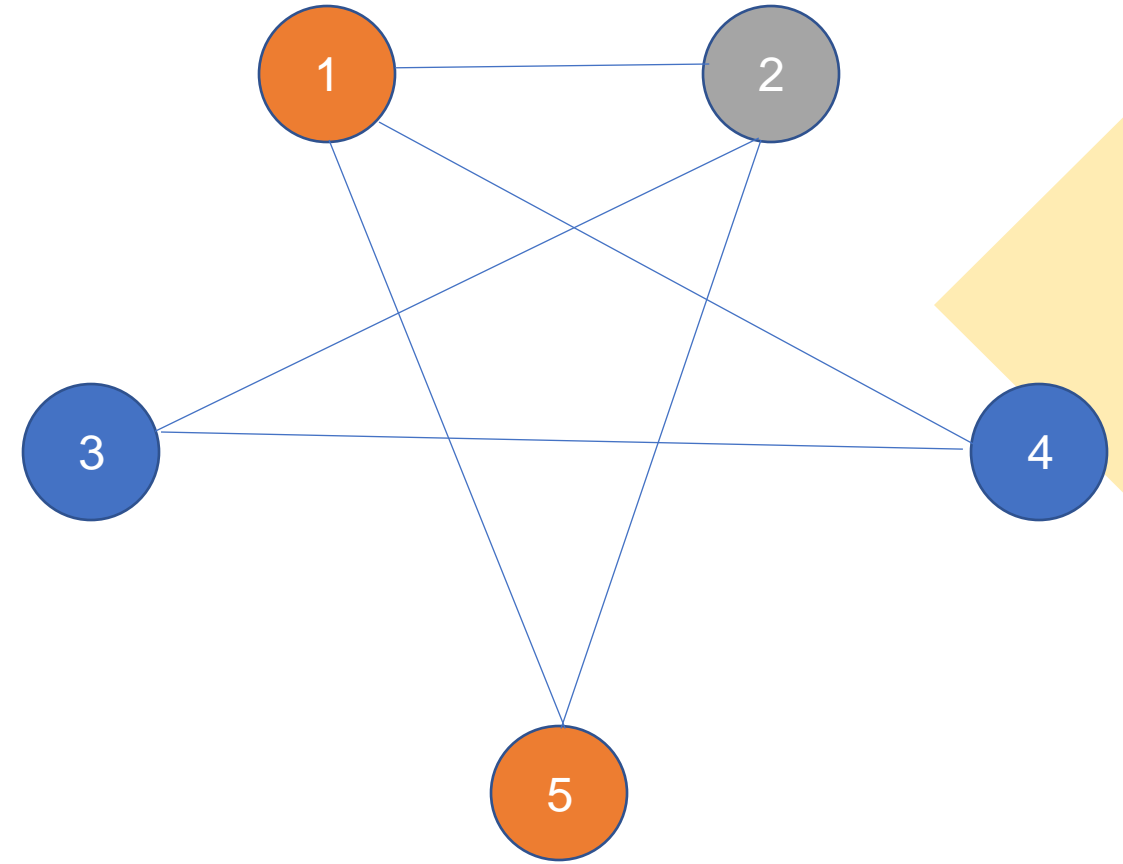
Visited

1	2	3	4	5
T	T	F	F	T

Stack

1	2	5		
---	---	---	--	--

Parent: 1



Detect cycle in undirected graph

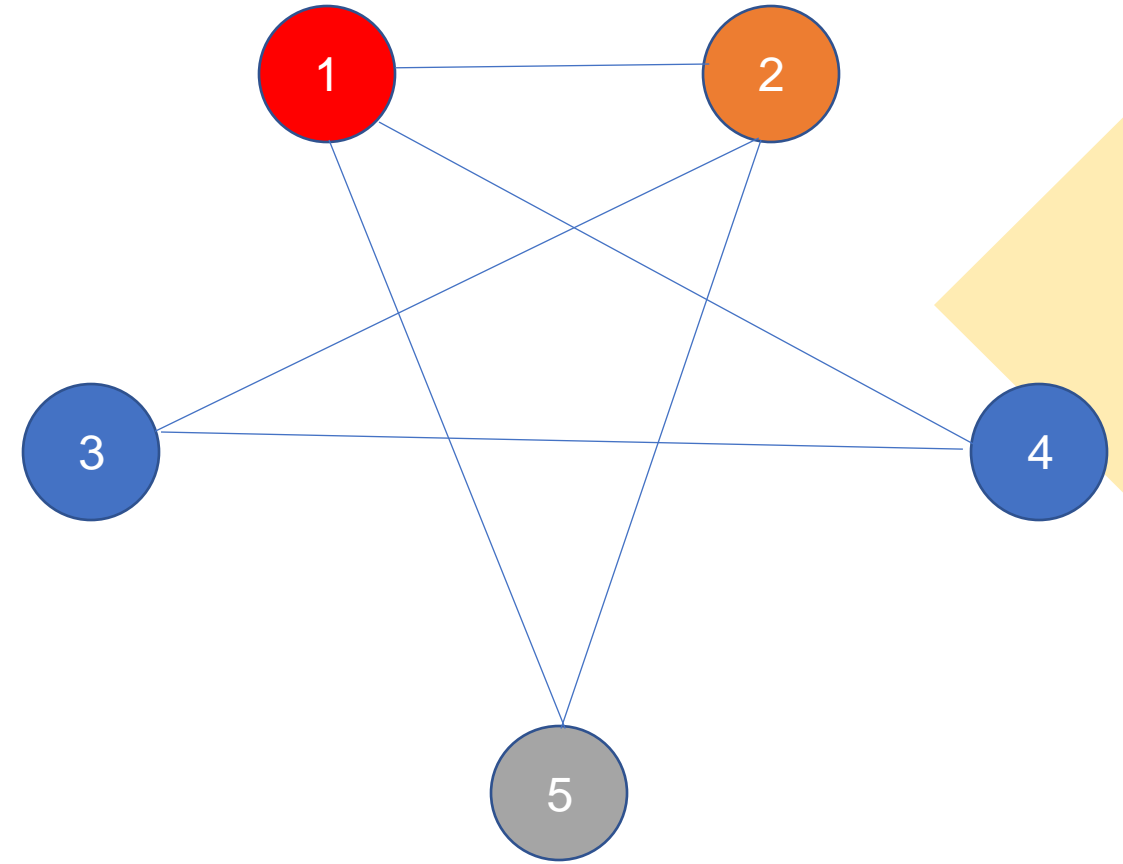
Visited

1	2	3	4	5
T	T	F	F	T

Stack

1	2	5		
---	---	---	--	--

Parent: 2

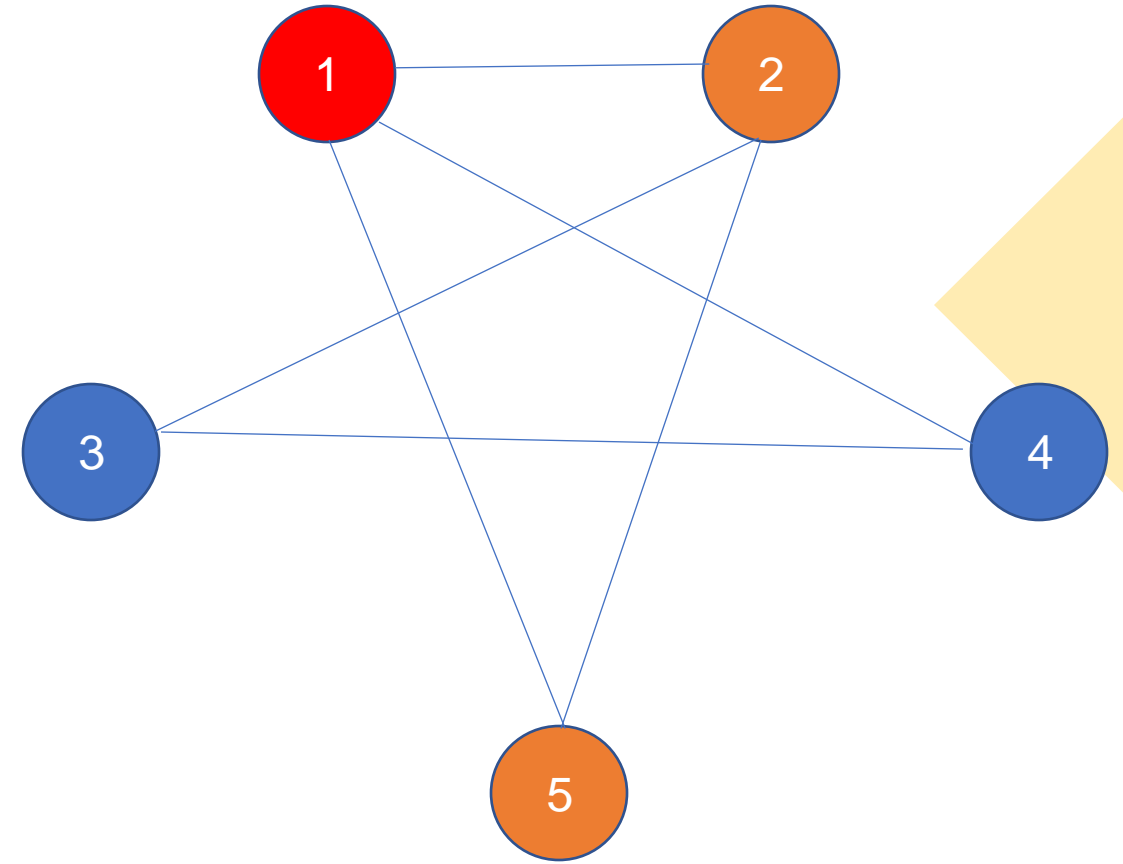


1 đã được đánh dấu là visited và không
là parent của 5
=> return true

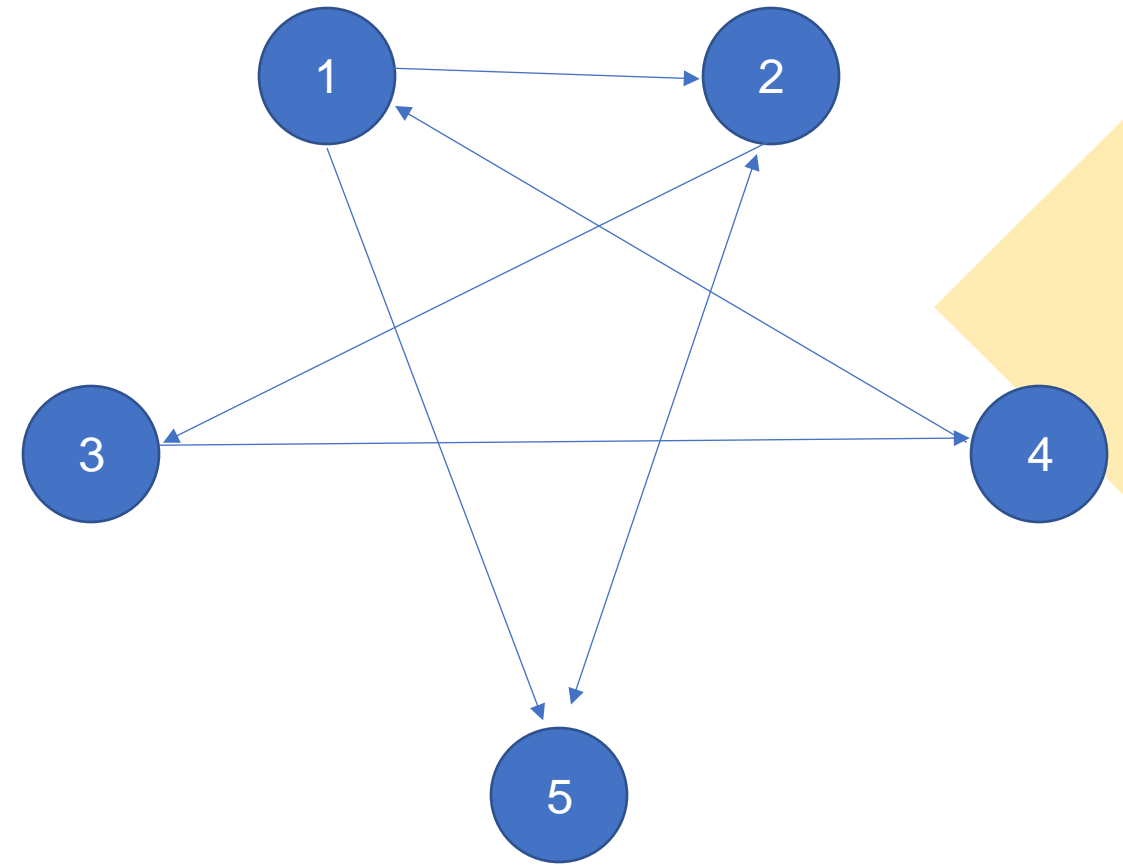
Detect cycle in undirected graph

Độ phức tạp thuật toán:

- ✓ Thời gian: $O(V+E)$ vì thuật toán chỉ thực hiện DFS đơn giản
- ✓ Không gian: $O(V)$ vì cần lưu trữ thêm 1 mảng visited

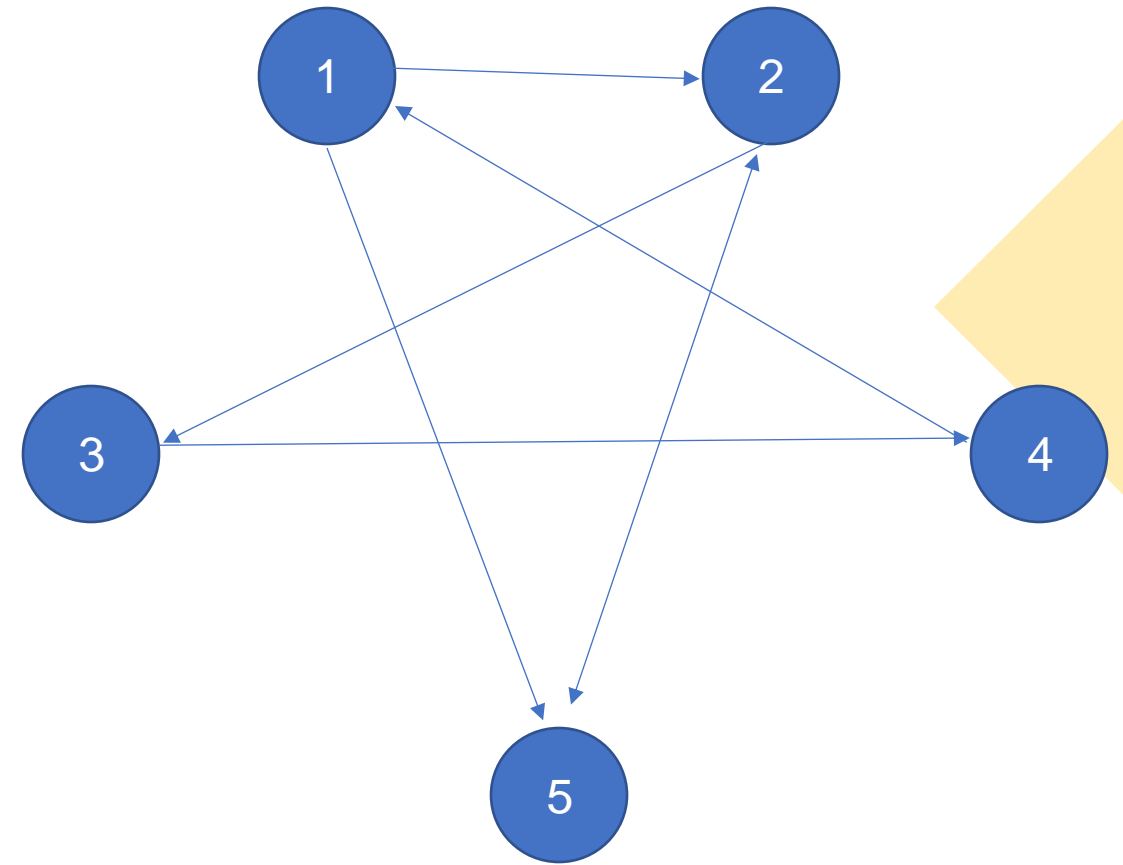


Breadth First Traversal



Breadth First Traversal

Tương tự như Depth First Search nhưng thay vì duyệt đến nhánh xa nhất thì chúng ta duyệt tất cả các node mà 1 node gốc chỉ đến



Breadth First Traversal

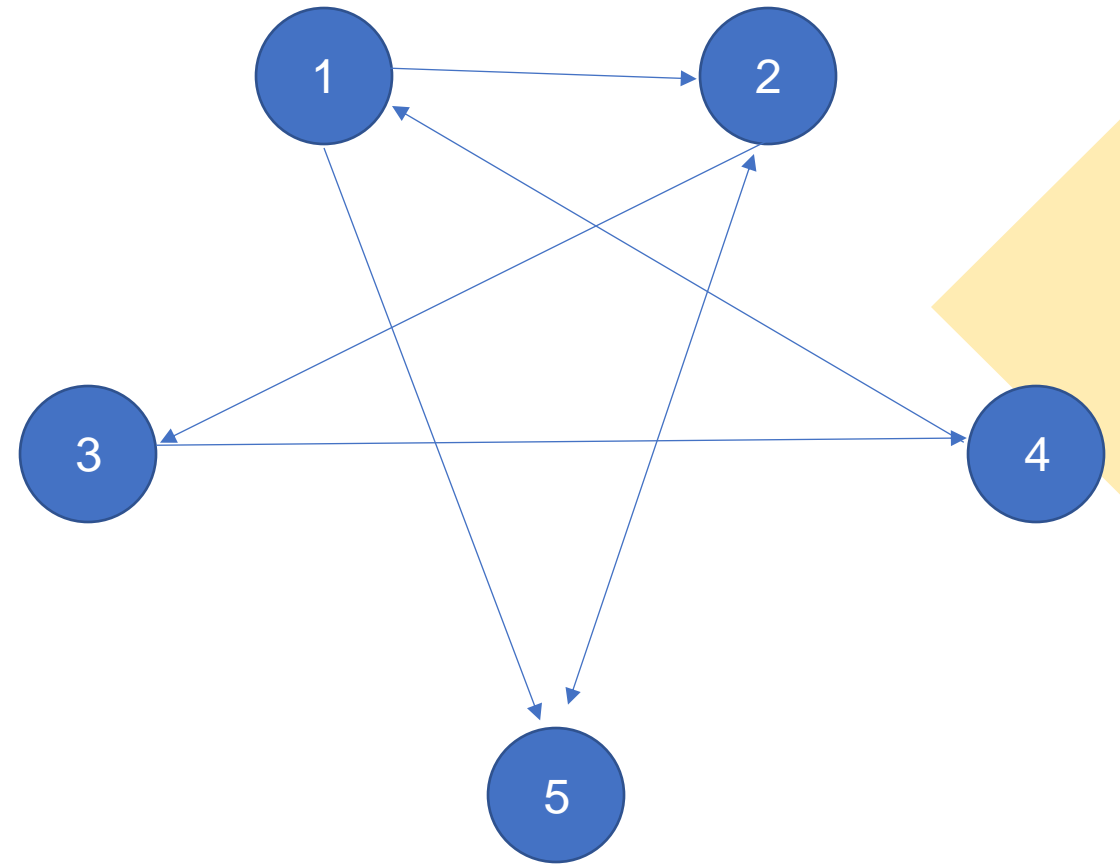
Visited

1	2	3	4	5
F	F	F	F	F

Queue

--	--	--	--	--

Print:



Breadth First Traversal

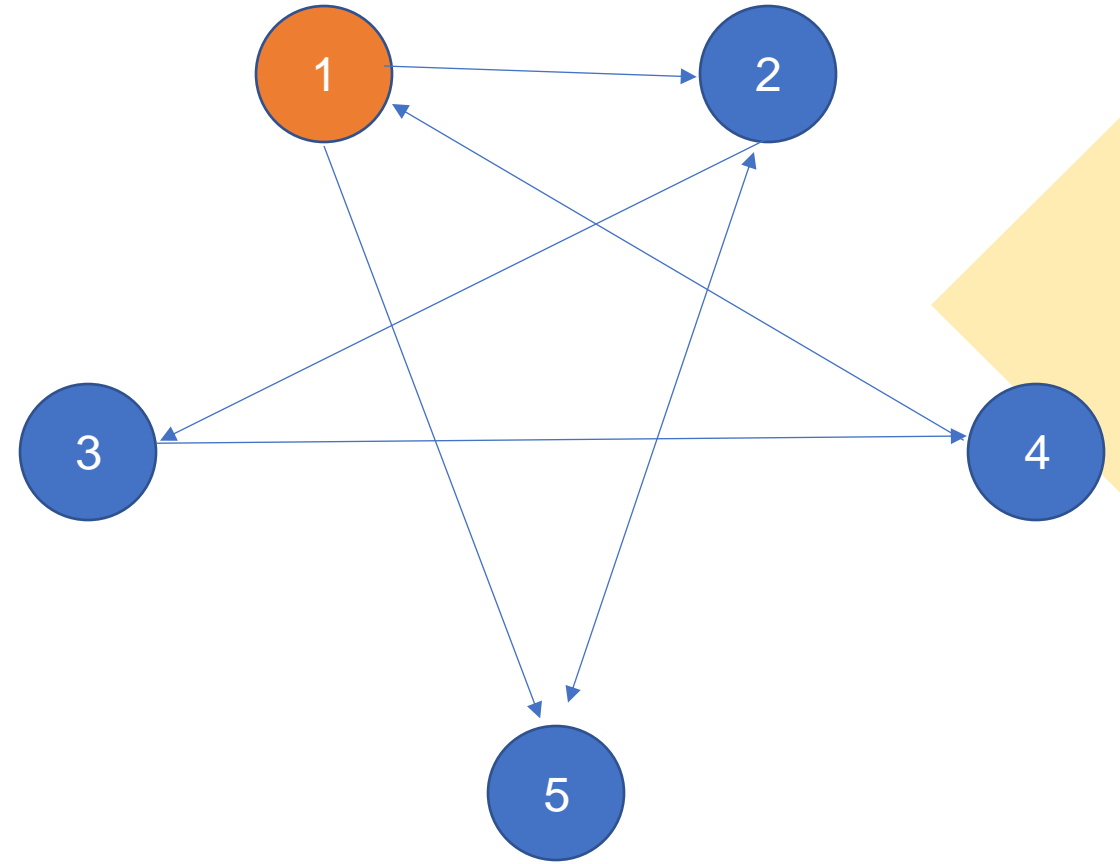
Visited

1	2	3	4	5
T	F	F	F	F

Queue

1				
---	--	--	--	--

Print:



Breadth First Traversal

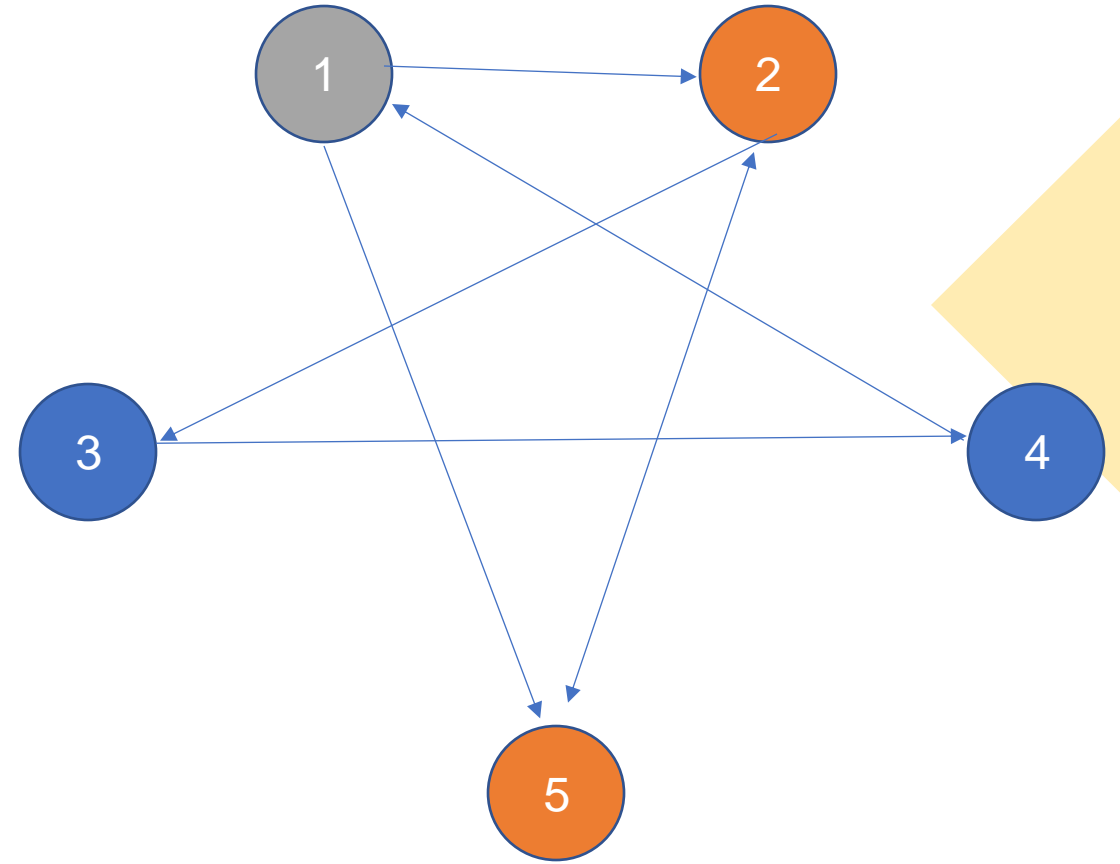
Visited

1	2	3	4	5
T	T	F	F	T

Queue

2	5			
---	---	--	--	--

Print: 1



Breadth First Traversal

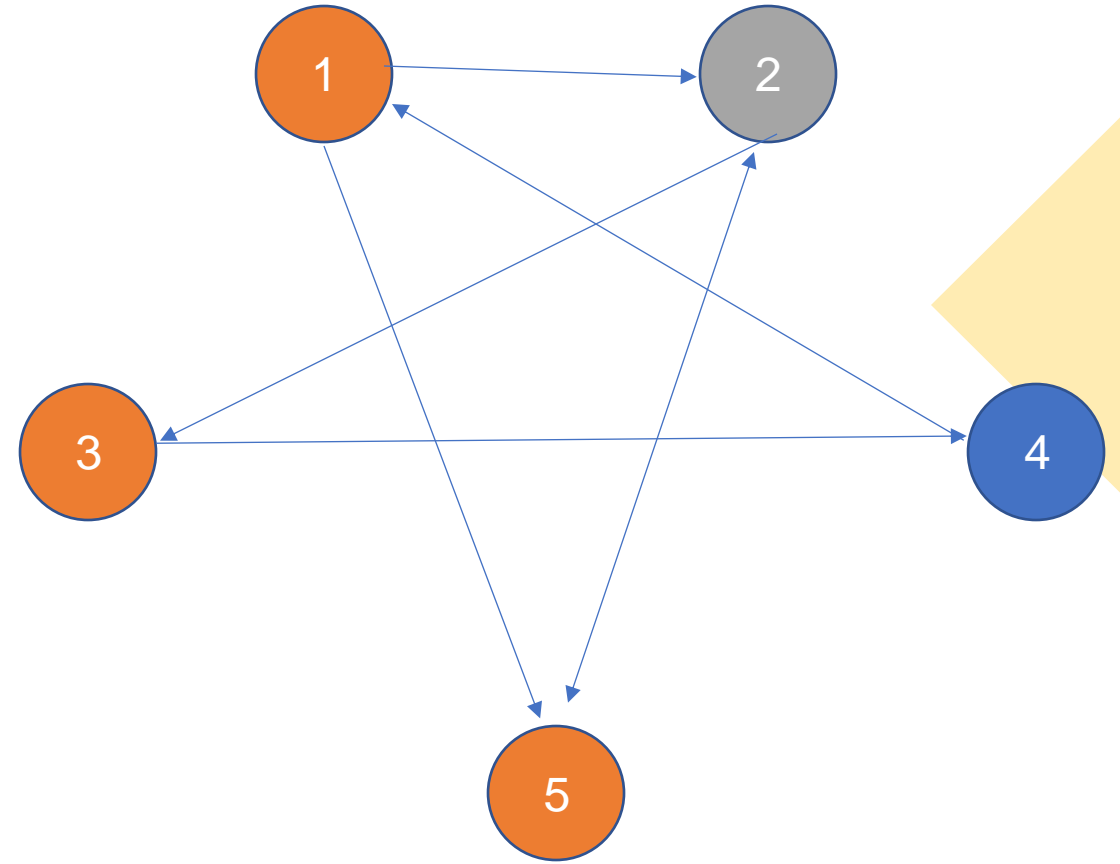
Visited

1	2	3	4	5
T	T	T	F	T

Queue

5	3			
---	---	--	--	--

Print: 1 2



Breadth First Traversal

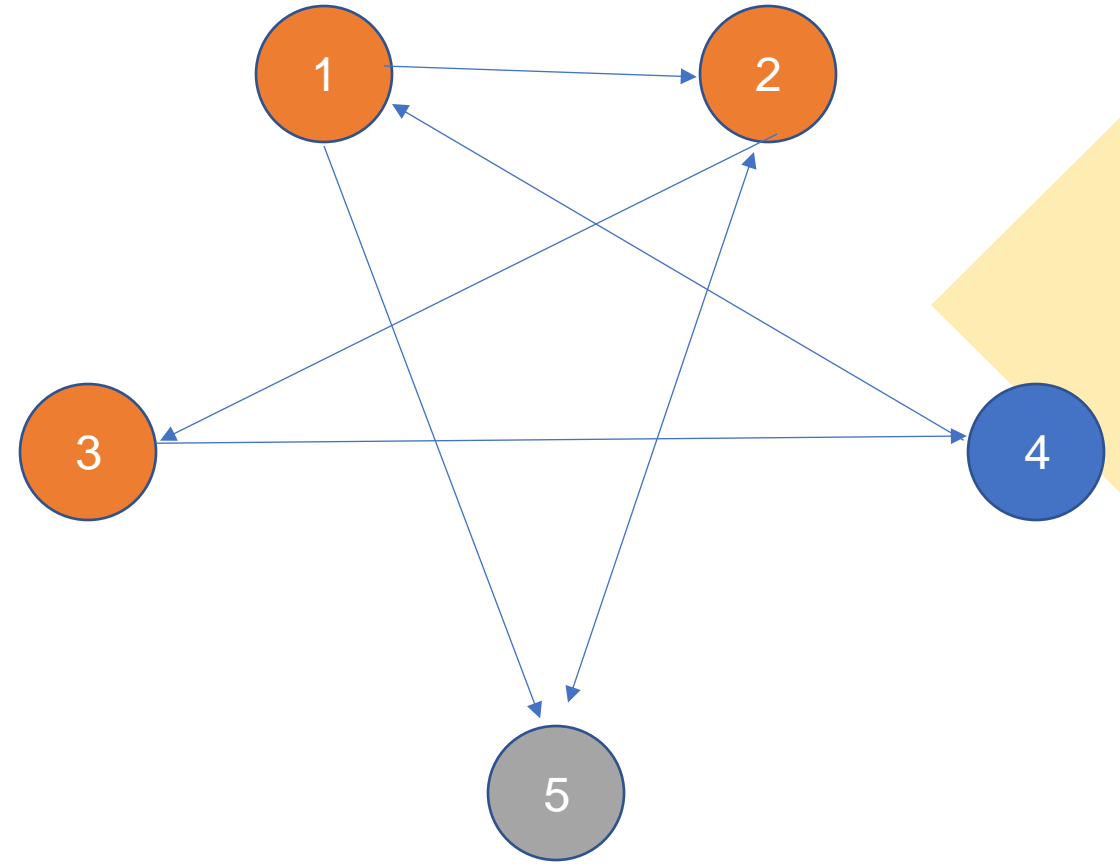
Visited

1	2	3	4	5
T	T	T	F	T

Queue

3				
---	--	--	--	--

Print: 1 2 5



Breadth First Traversal

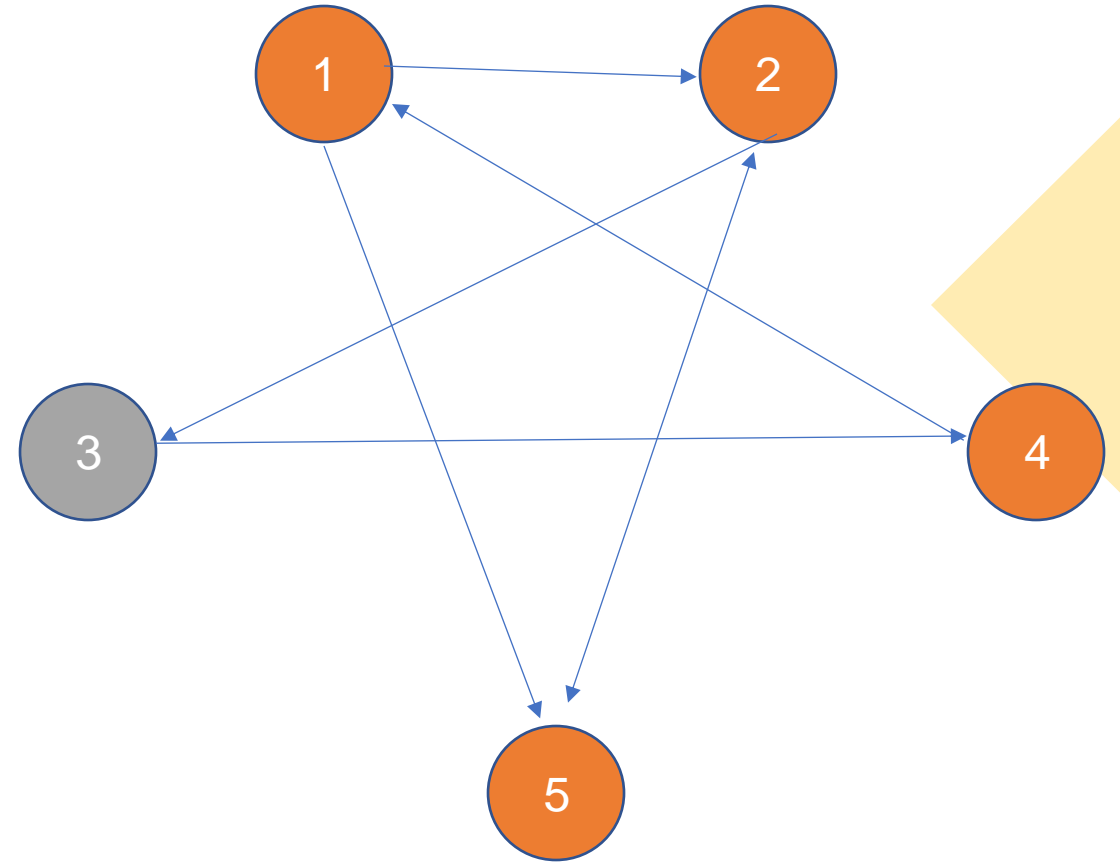
Visited

1	2	3	4	5
T	T	T	T	T

Queue

4				
---	--	--	--	--

Print: 1 2 5 3 4



Breadth First Traversal

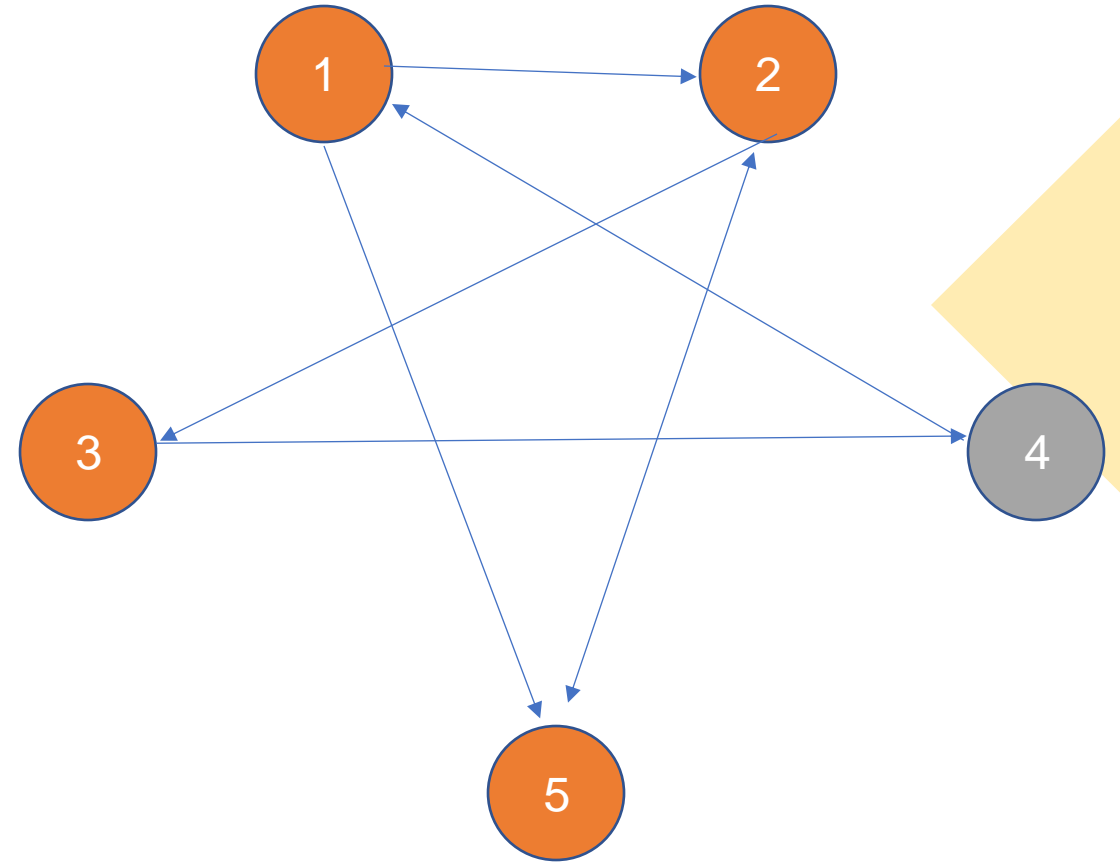
Visited

1	2	3	4	5
T	T	T	T	T

Queue

--	--	--	--	--

Print: 1 2 5 3 4



Breadth First Traversal

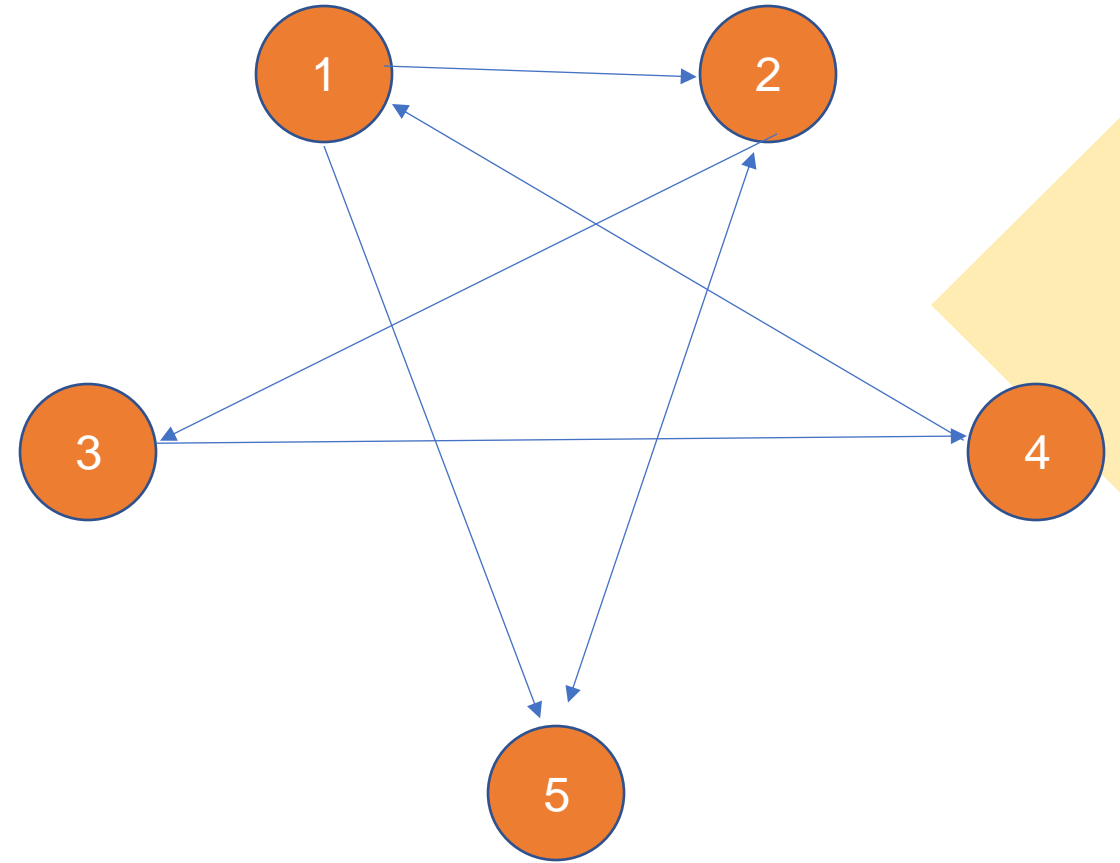
Visited

1	2	3	4	5
T	T	T	T	T

Queue

--	--	--	--	--

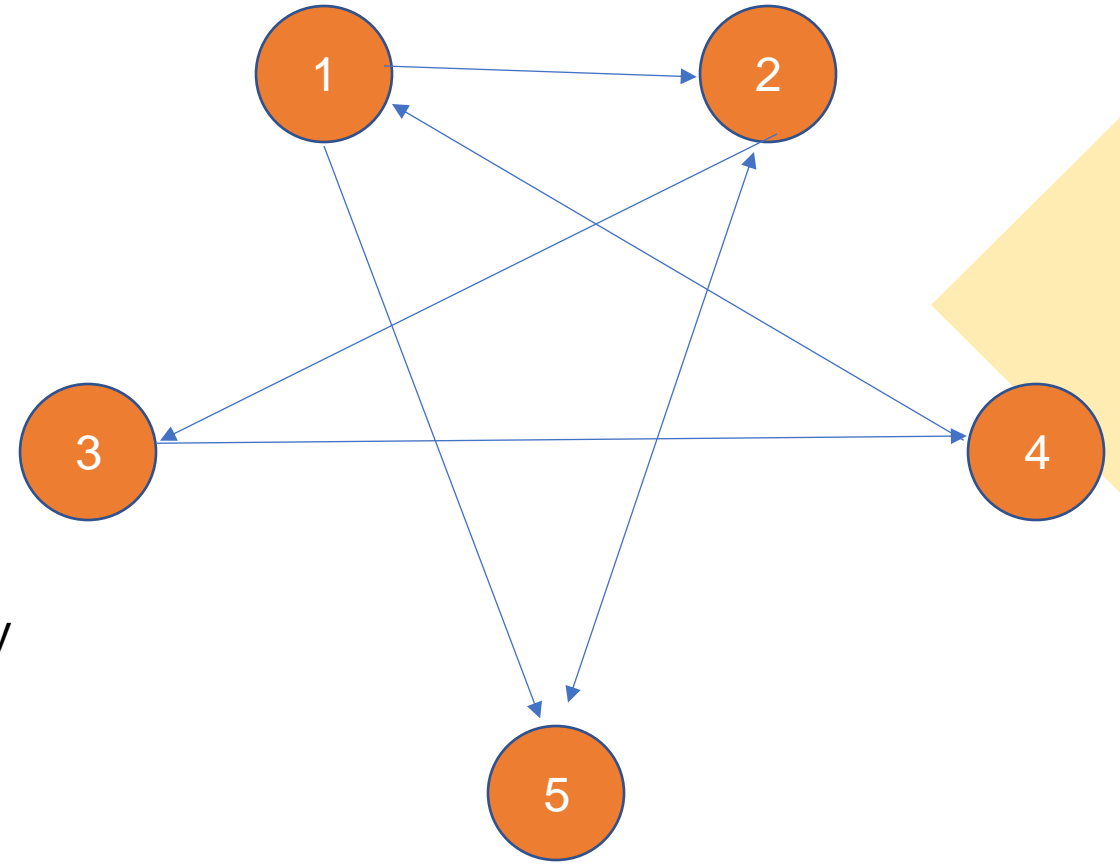
Print: 1 2 5 3 4



Breadth First Traversal

Độ phức tạp của thuật toán:

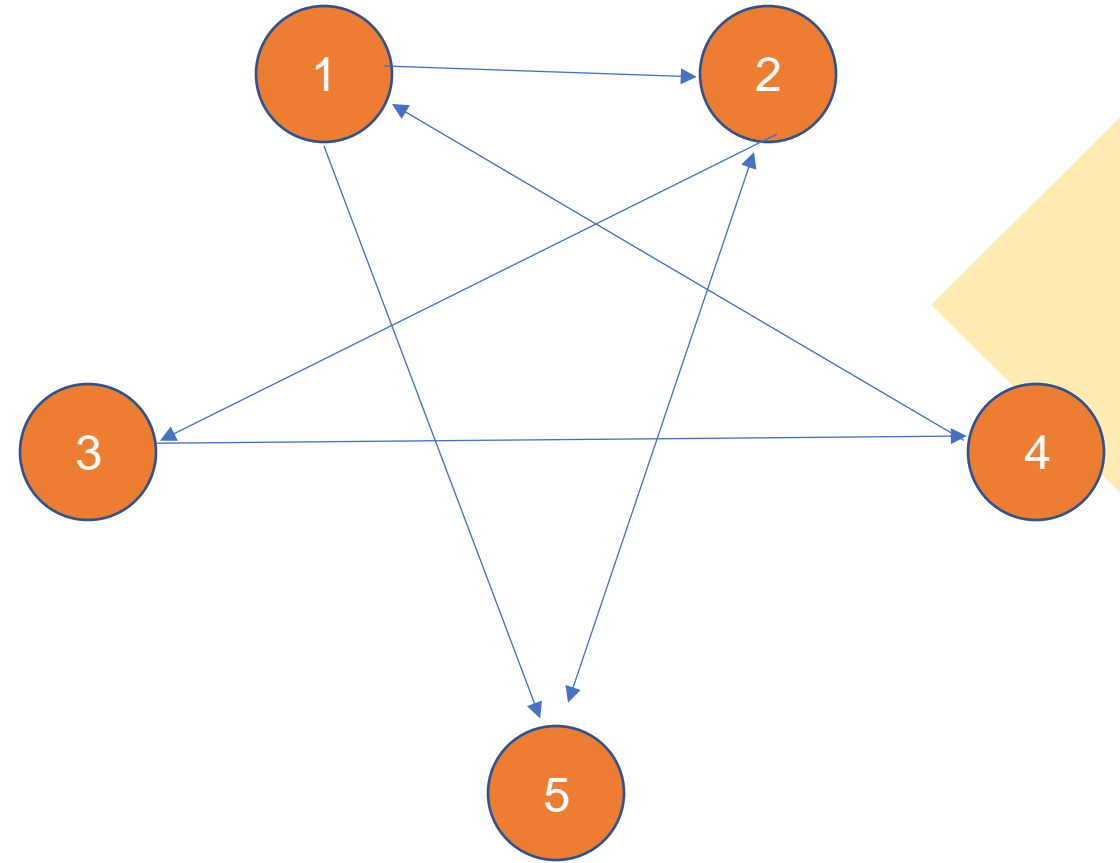
- ✓ Thời gian: $O(V+E)$ với V là số đỉnh và E là số cạnh
- ✓ Không gian: $O(V)$ vì cần thêm không gian để lưu dãy visited có V phần tử



Breadth First Traversal

Ứng dụng:

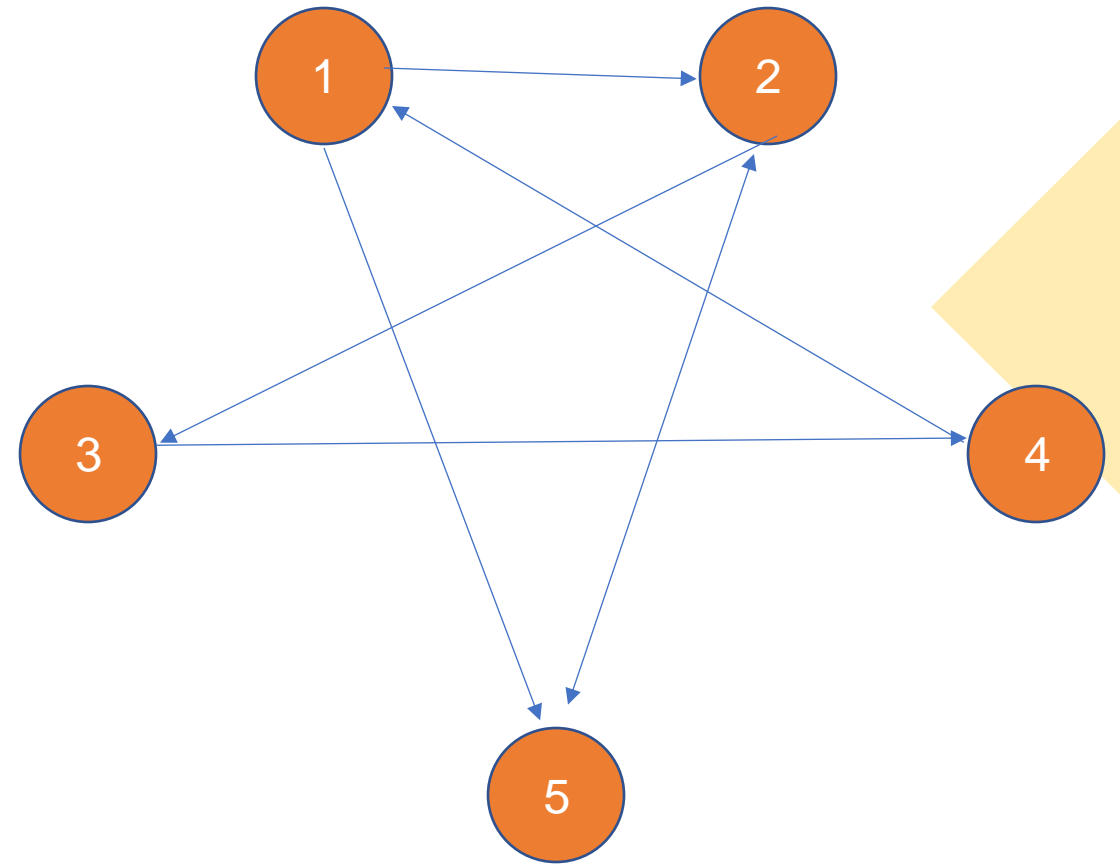
- ✓ Shortest Path and Minimum Spanning Tree
- ✓ Cycle detecting in undirected graph
- ✓ Crawlers in Search Engines
- ✓ GPS Navigation systems



Shortest Paths

Finding

Yêu cầu: Tìm đường đi ngắn nhất từ 1 đỉnh đến toàn bộ đỉnh



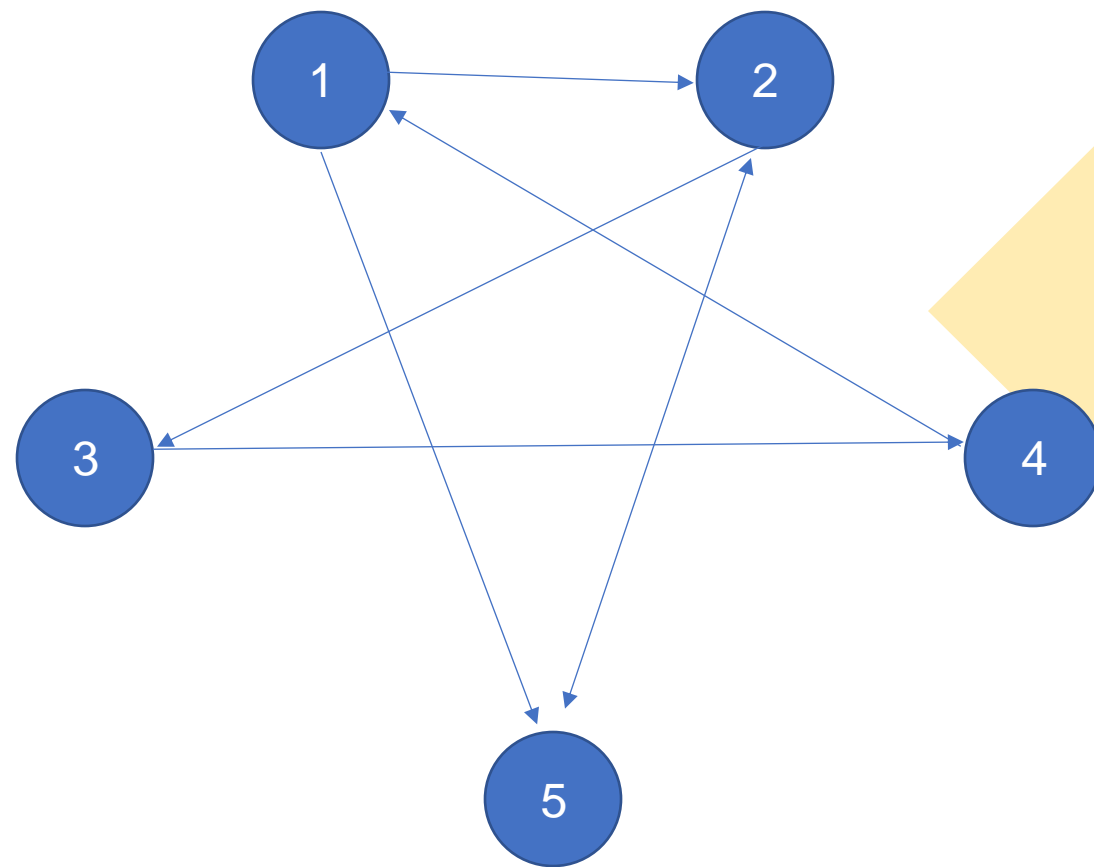
Các thuật toán:

- ❖ **BFS:** dùng cho Unweighted Graph
- ❖ **Dijkstra:** dùng cho Positive Weighted Graph
- ❖ **Bellman – Ford:** dùng cho Negative Weighted Graph

BFS

Phương hướng:

- Có thể tới được node nào thì chốt luôn nốt đấy
- Không được tiến hành duyệt theo chiều sâu để tránh việc đi vòng



BFS

Visited

1	2	3	4	5
F	F	F	F	F

Pre

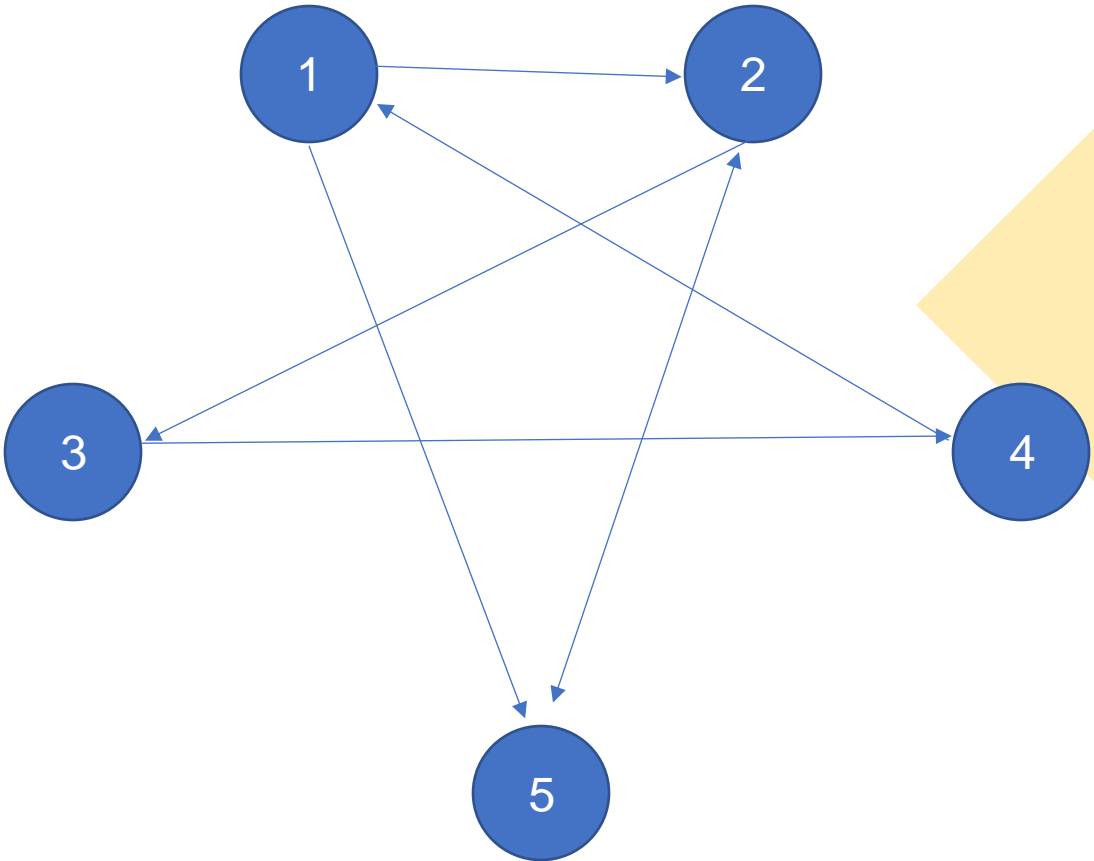
1	2	3	4	5
-1	-1	-1	-1	-1

Queue

--	--	--	--	--

Distance

1	2	3	4	5
-1	-1	-1	-1	-1



BFS

Visited

1	2	3	4	5
T	F	F	F	F

Parent

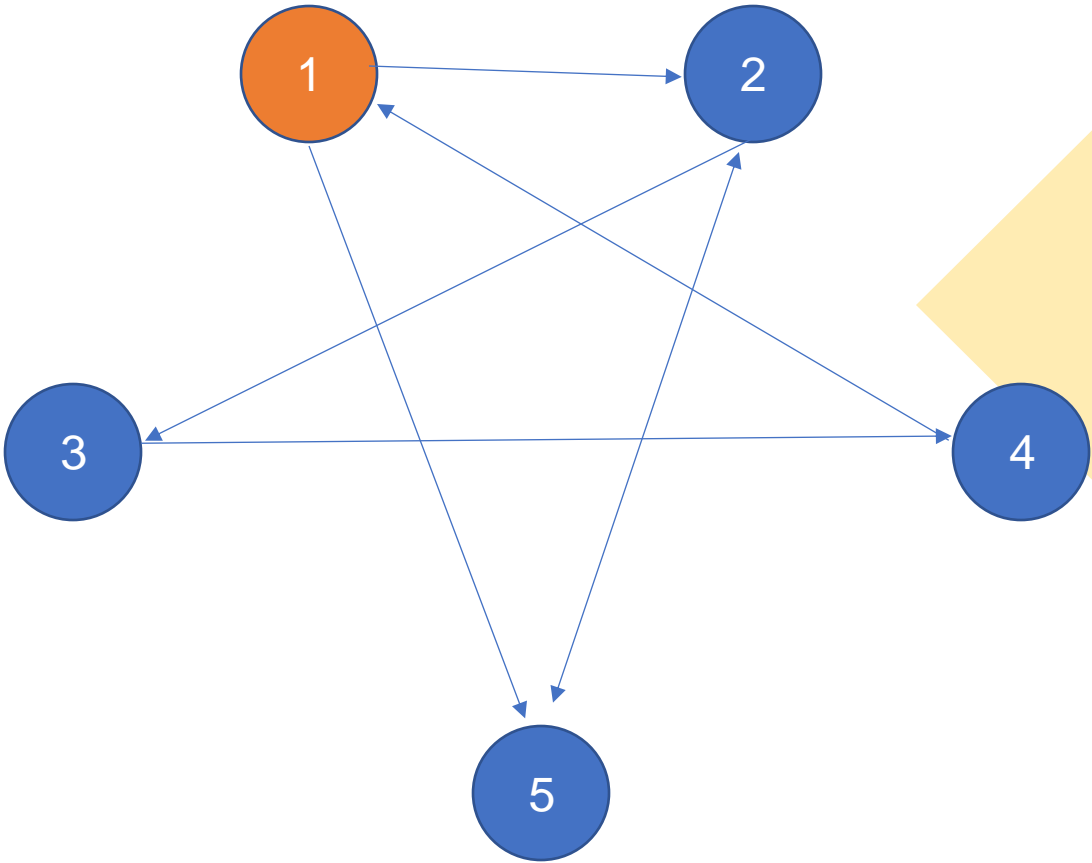
1	2	3	4	5
-1	-1	-1	-1	-1

Queue

1				
---	--	--	--	--

Distance

1	2	3	4	5
0	-1	-1	-1	-1



BFS

Visited

1	2	3	4	5
T	T	T	F	F

Pre

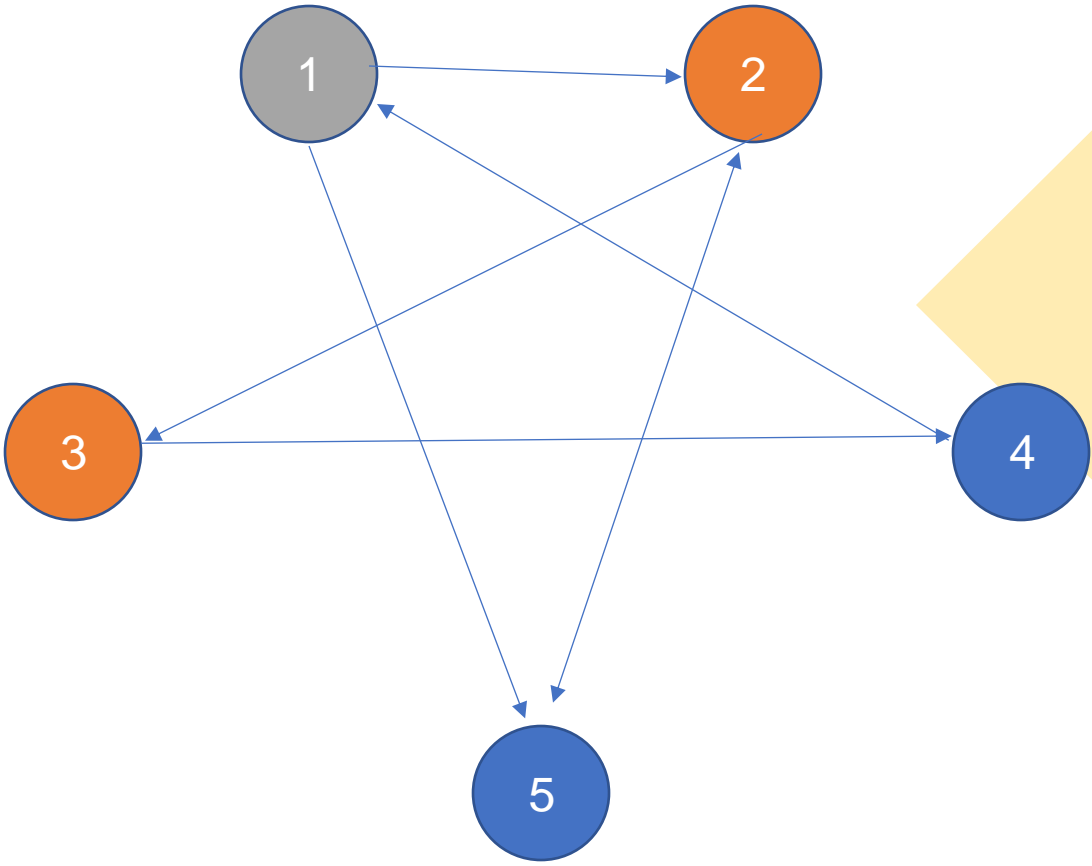
1	2	3	4	5
-1	1	1	-1	-1

Queue

2	3			
---	---	--	--	--

Distance

1	2	3	4	5
0	1	1	-1	-1



BFS

Visited

1	2	3	4	5
T	T	T	F	T

Pre

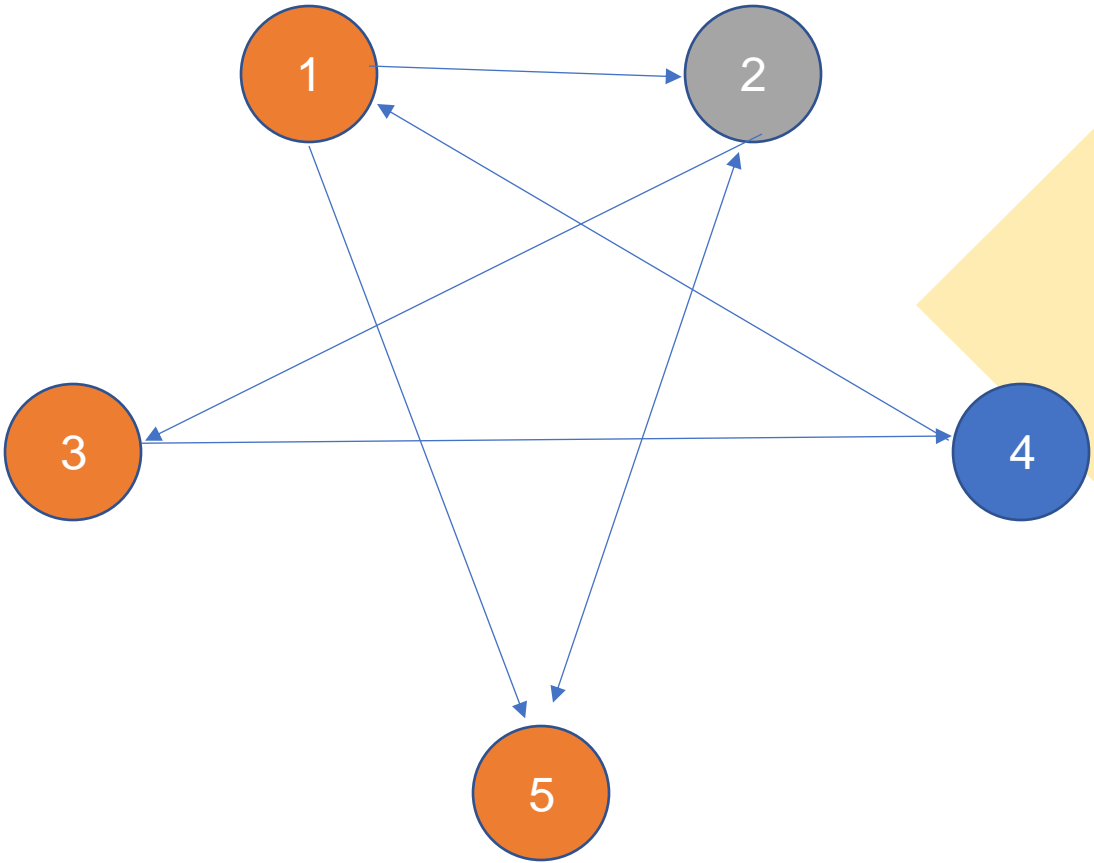
1	2	3	4	5
-1	1	1	-1	2

Queue

3	5			
---	---	--	--	--

Distance

1	2	3	4	5
0	1	1	-1	2



BFS

Visited

1	2	3	4	5
T	T	T	T	T

Pre

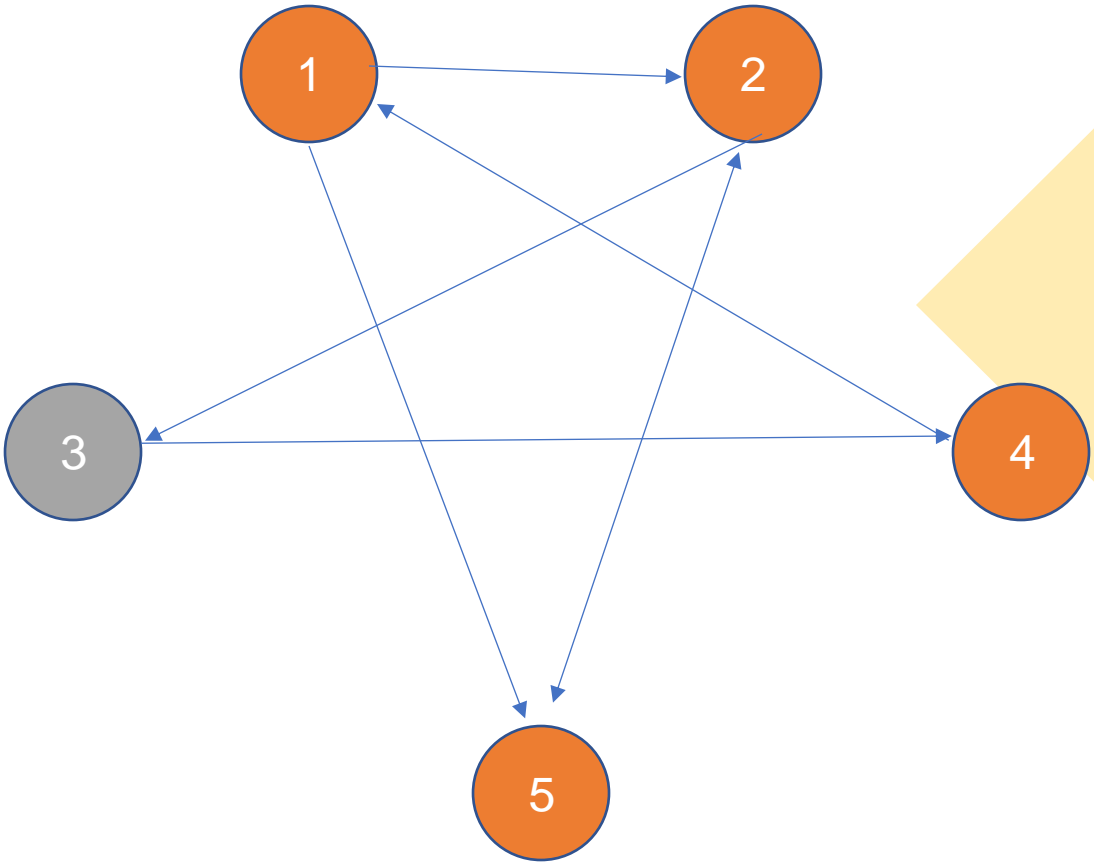
1	2	3	4	5
-1	1	1	3	2

Queue

5	4			
---	---	--	--	--

Distance

1	2	3	4	5
0	1	1	2	2



BFS

Visited

1	2	3	4	5
T	T	T	T	T

Pre

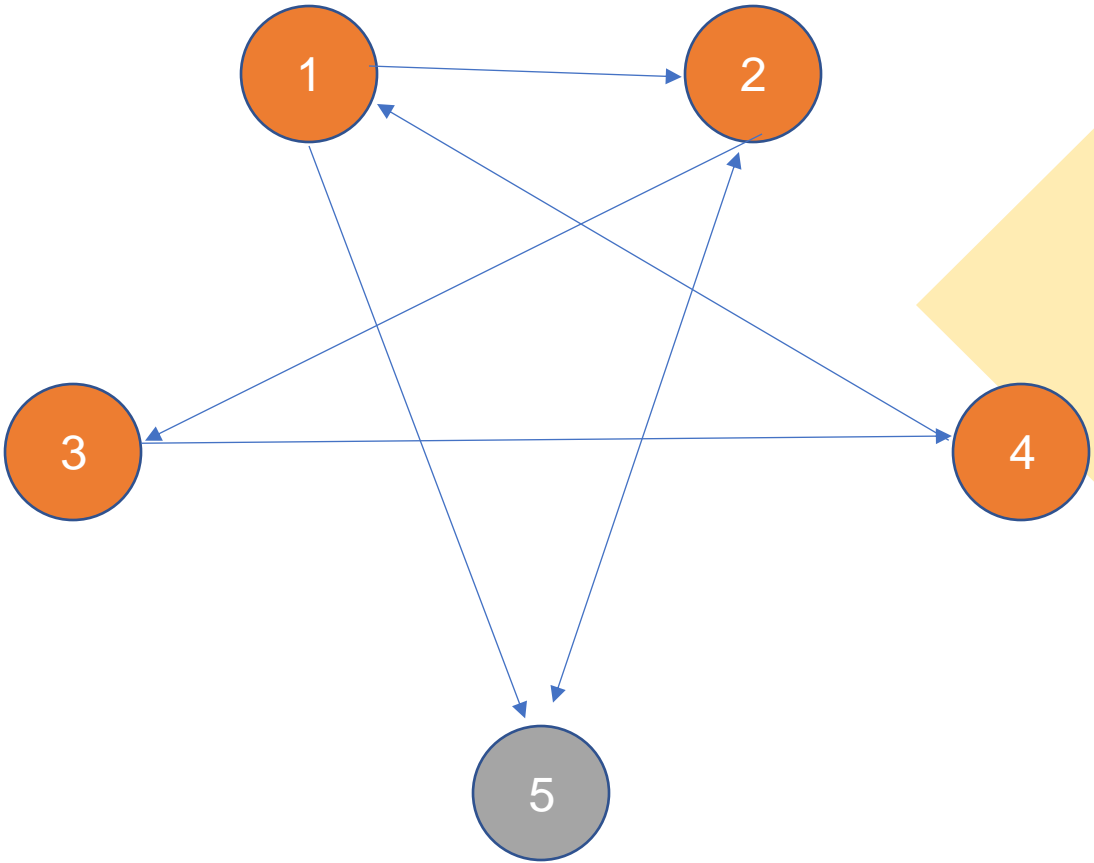
1	2	3	4	5
-1	1	1	3	2

Queue

4				
---	--	--	--	--

Distance

1	2	3	4	5
0	1	1	2	2



BFS

Visited

1	2	3	4	5
T	T	T	T	T

Pre

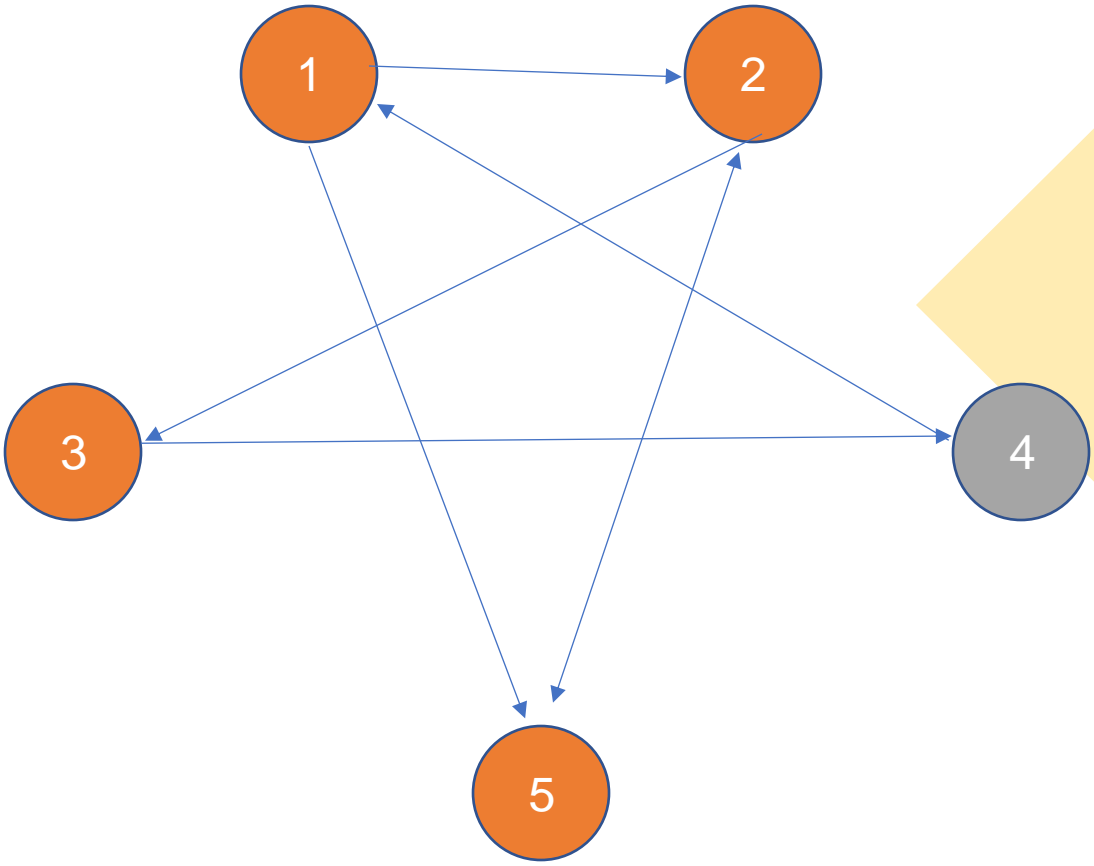
1	2	3	4	5
-1	1	1	3	2

Queue

--	--	--	--	--

Distance

1	2	3	4	5
0	1	1	2	2



BFS

Visited

1	2	3	4	5
T	T	T	T	T

Pre

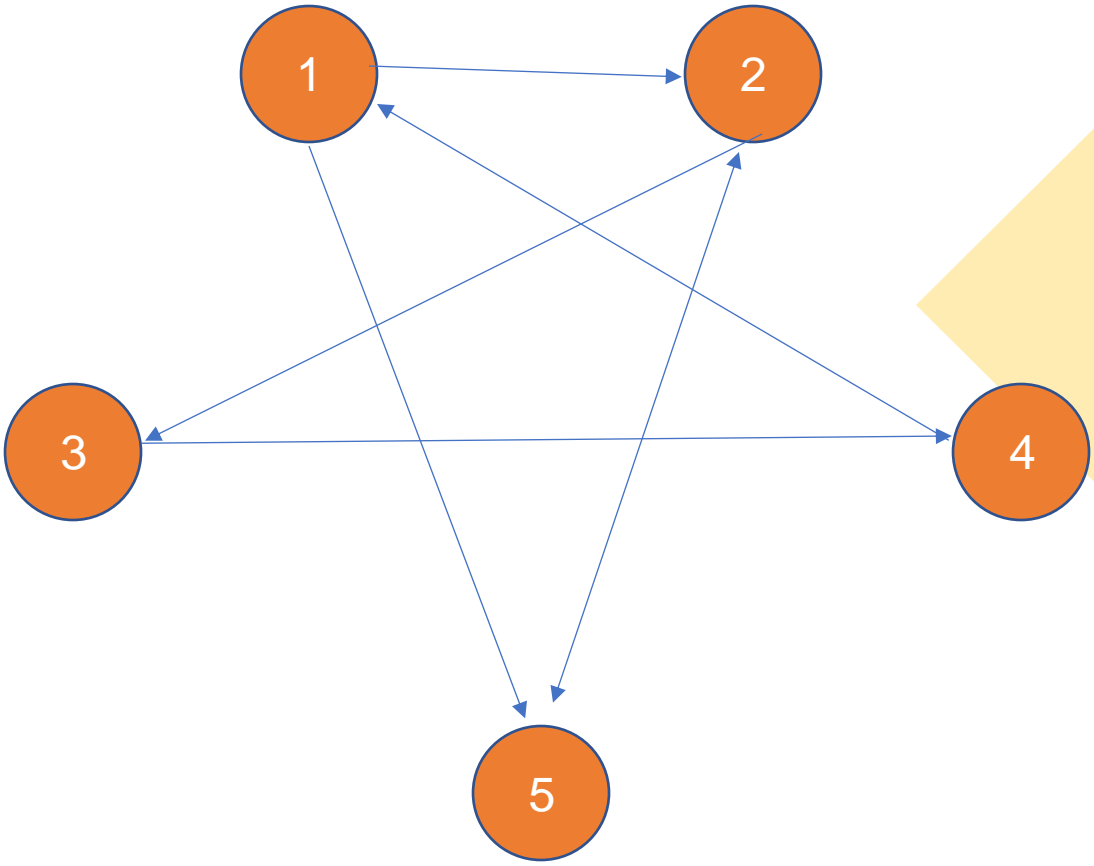
1	2	3	4	5
-1	1	1	3	2

Queue

--	--	--	--	--

Distance

1	2	3	4	5
0	1	1	2	2



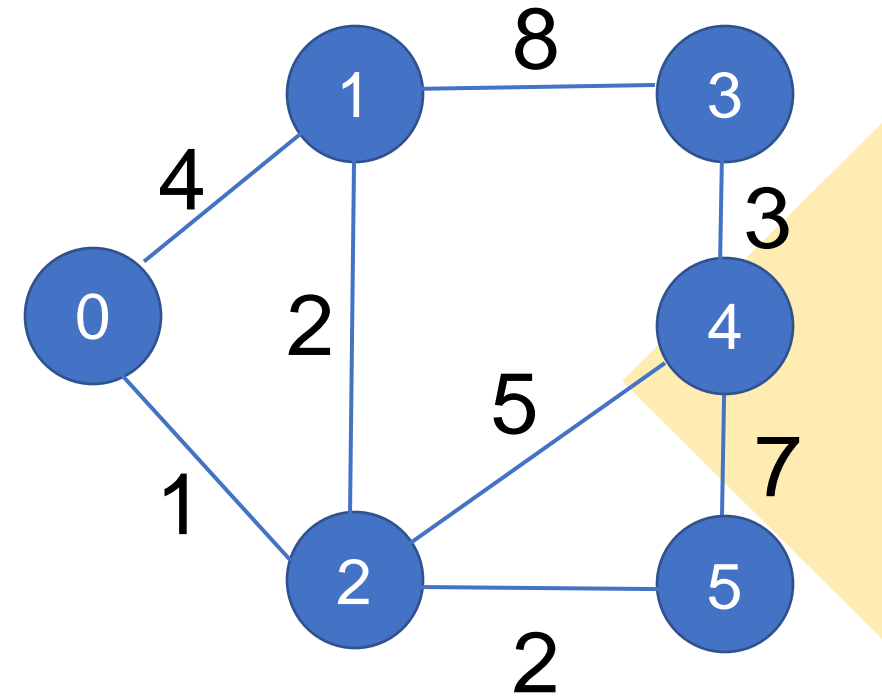
Dijkstra

Vấn đề:

- Đi đường vòng có thể distance ngắn hơn
- Không thể chốt cả 2 node (1) và (2) ngay lập tức được

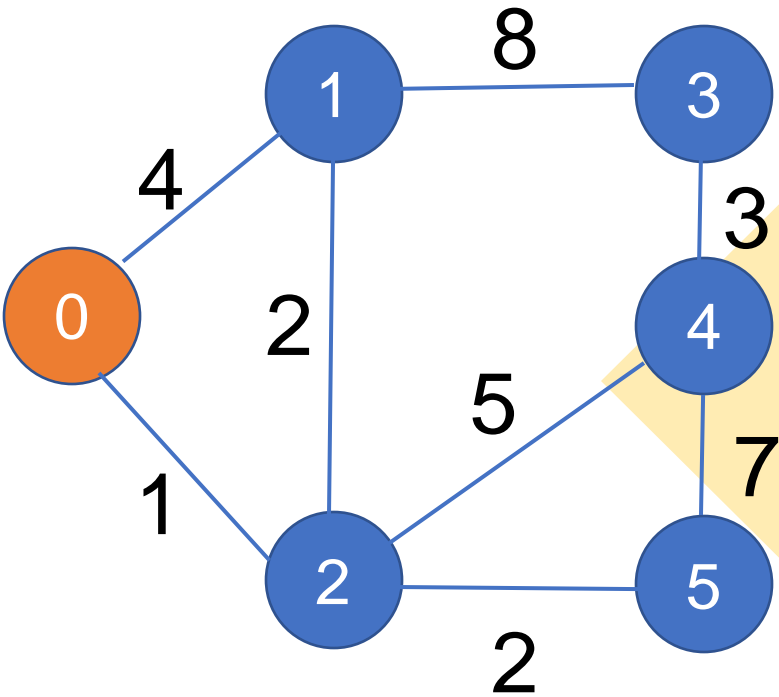
Giải quyết:

- Cập nhật distance
- Chốt node có distance nhỏ nhất
- Lấy node vừa chốt để tiếp tục tìm kiếm
- Không hướng tầm nhìn đến những node đã chốt
- Thực hiện $|V|$ lần lặp



Dijkstra

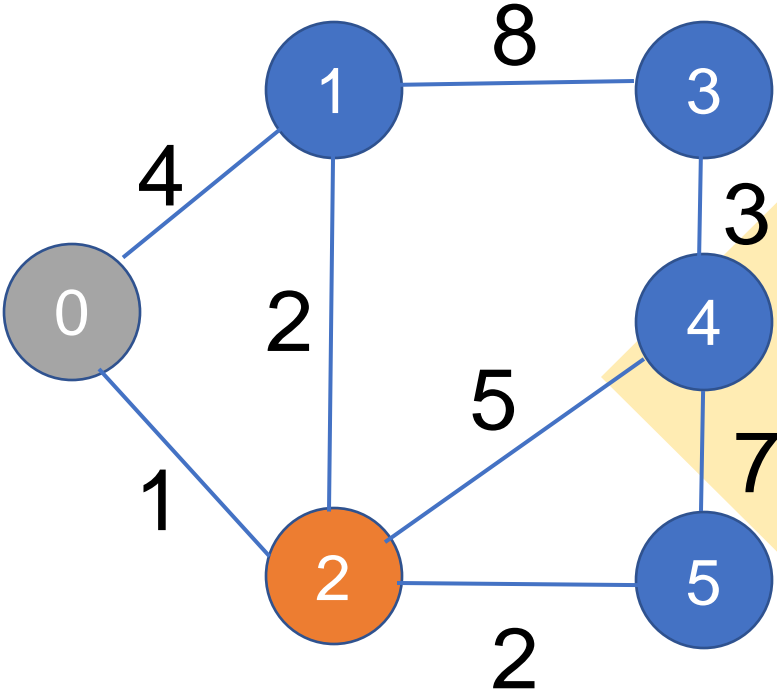
	0	1	2	3	4	5
	0	inf	inf	inf	inf	inf
(0): 0						



detected	0					
----------	---	--	--	--	--	--

Dijkstra

	0	1	2	3	4	5
	0	inf	inf	inf	inf	inf
(0): 0	-	4	1	inf	inf	inf
(2): 1						

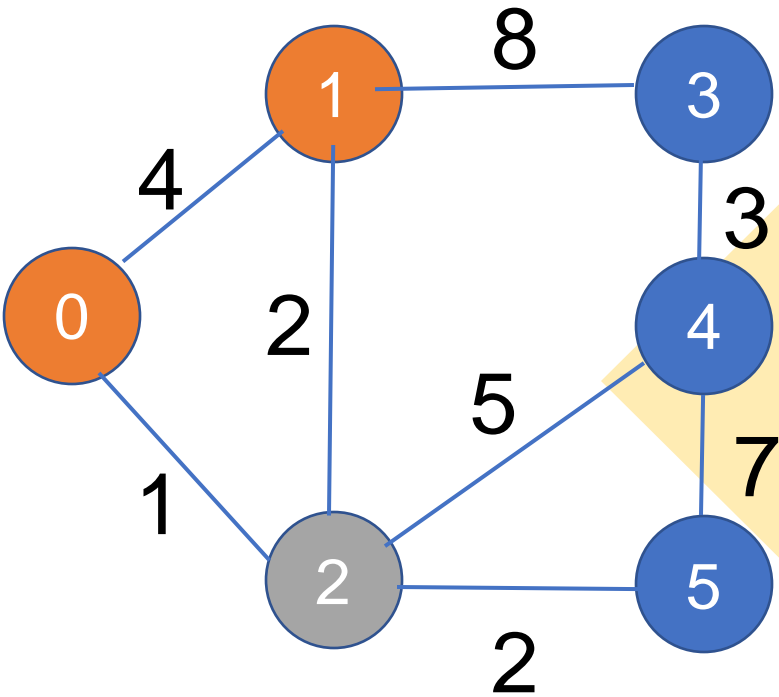


detected

0	2				
---	---	--	--	--	--

Dijkstra

	0	1	2	3	4	5
	0	inf	inf	inf	inf	inf
(0): 0	-	4	1	inf	inf	inf
(2): 1	-	3	-	inf	6	3
(1): 3						

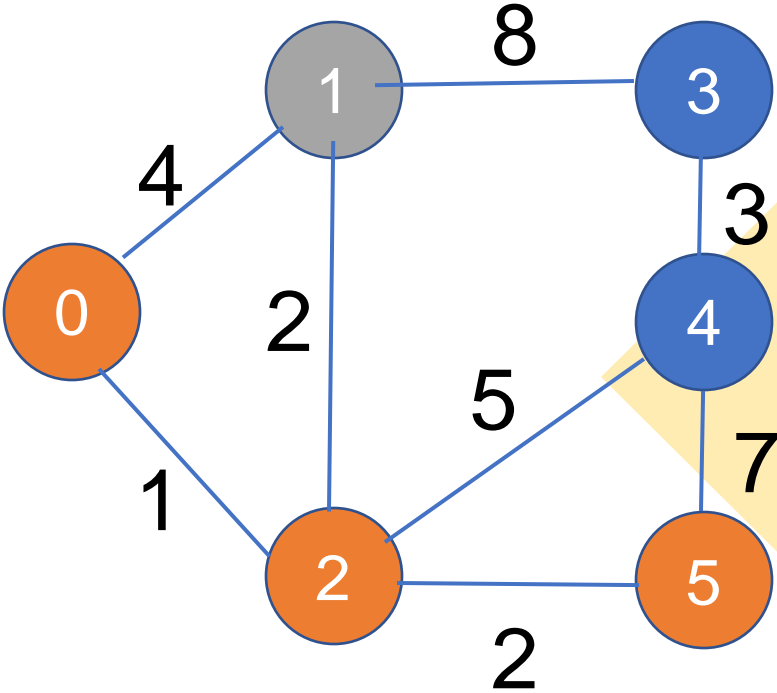


detected

0	2	1			
---	---	---	--	--	--

Dijkstra

	0	1	2	3	4	5
	0	inf	inf	inf	inf	inf
(0): 0	-	4	1	inf	inf	inf
(2): 1	-	3	-	inf	6	3
(1): 3	-	-	-	12	6	3
(5): 3						

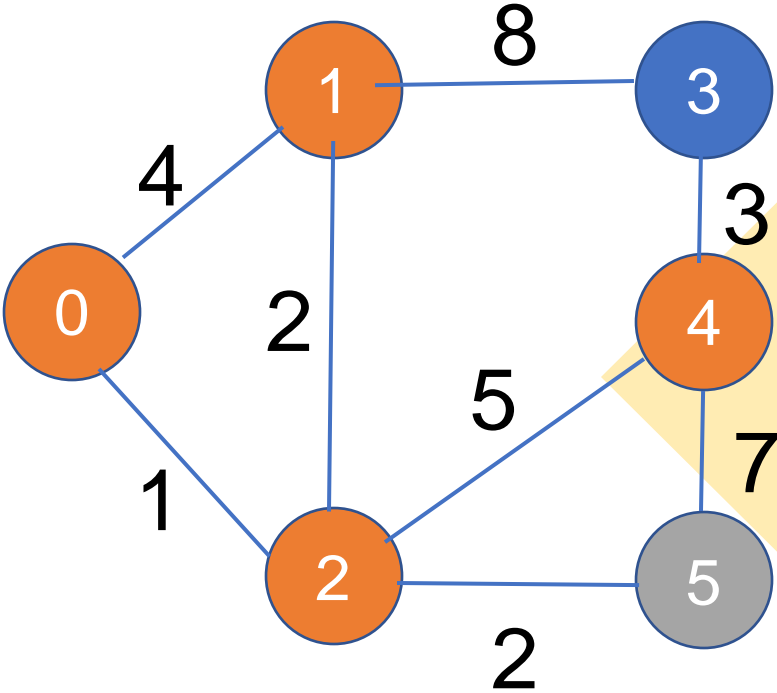


detected

0	2	1	5		
---	---	---	---	--	--

Dijkstra

	0	1	2	3	4	5
	0	inf	inf	inf	inf	inf
(0): 0	-	4	1	inf	inf	inf
(2): 1	-	3	-	inf	6	3
(1): 3	-	-	-	12	6	3
(5): 3	-	-	-	12	6	-
(4): 6						



detected

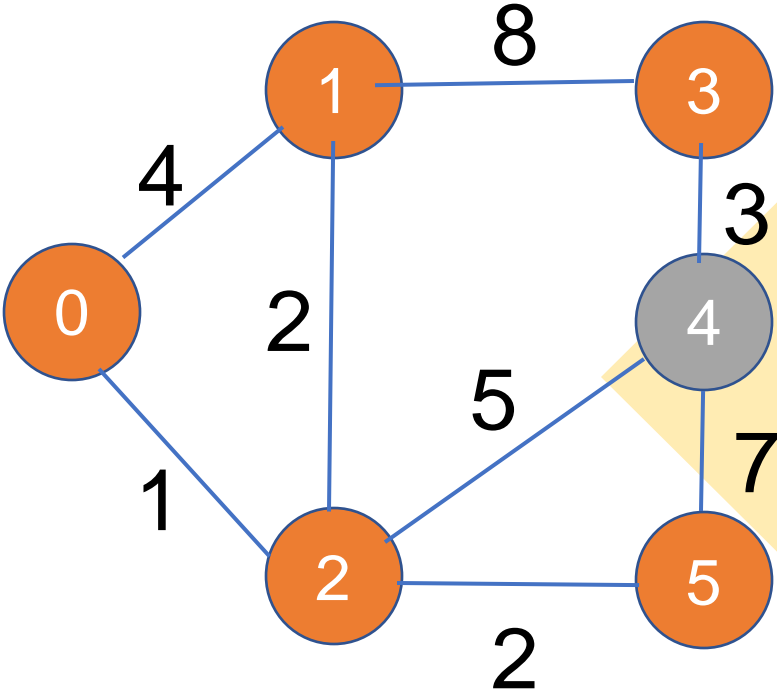
0	2	1	5		
---	---	---	---	--	--

Dijkstra

	0	1	2	3	4	5
	0	inf	inf	inf	inf	inf
(0): 0	-	4	1	inf	inf	inf
(2): 1	-	3	-	inf	6	3
(1): 3	-	-	-	12	6	3
(5): 3	-	-	-	12	6	-
(4): 6	-	-	-	9	-	-
(3): 9						

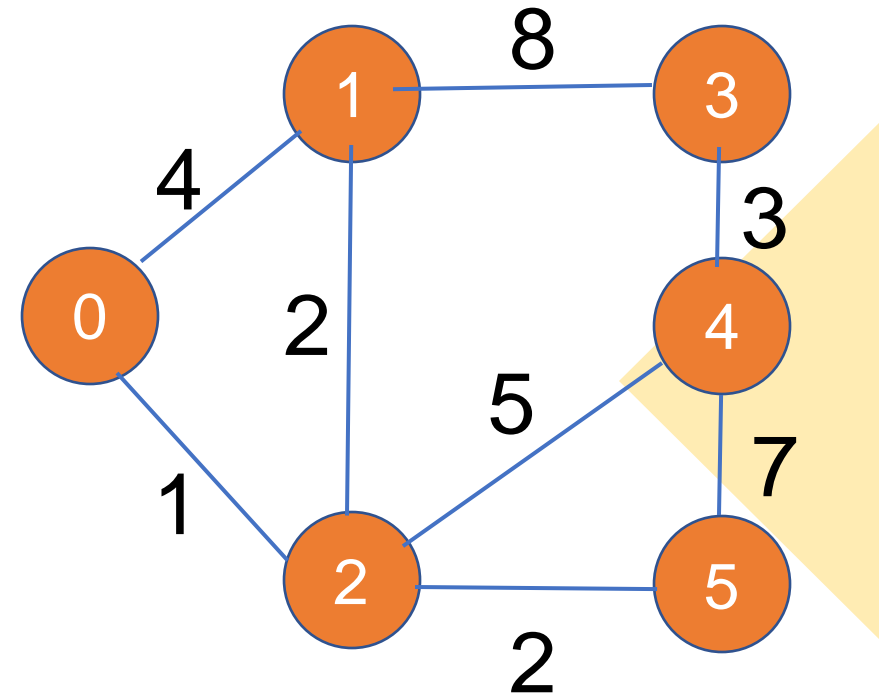
detected

0	2	1	5	4	3
---	---	---	---	---	---



Dijkstra

	0	1	2	3	4	5
	0	inf	inf	inf	inf	inf
(0): 0	-	4	1	inf	inf	inf
(2): 1	-	3	-	inf	6	3
(1): 3	-	-	-	12	6	3
(5): 3	-	-	-	12	6	-
(4): 6	-	-	-	9	-	-
(3): 9	-	-	-	-	-	-



detected

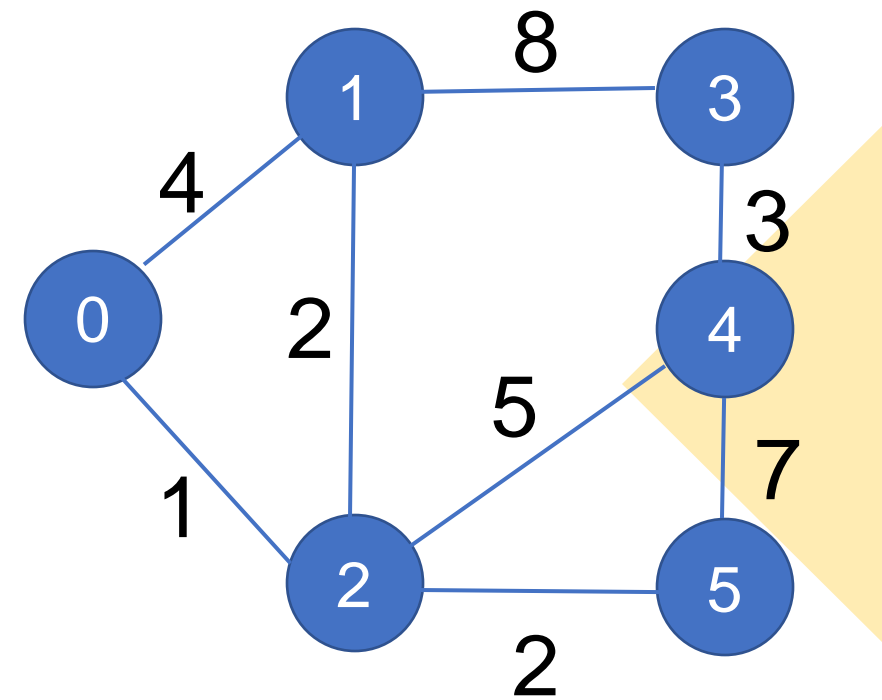
0 2 1 5 4 3

Dijkstra

Dùng cho đồ thị có trọng số dương

Độ phức tạp thời gian:
 $O(E + V^2)$

Có thể dùng binary heap để tìm min dễ dàng hơn.
 $O(E + V \cdot \log V)$

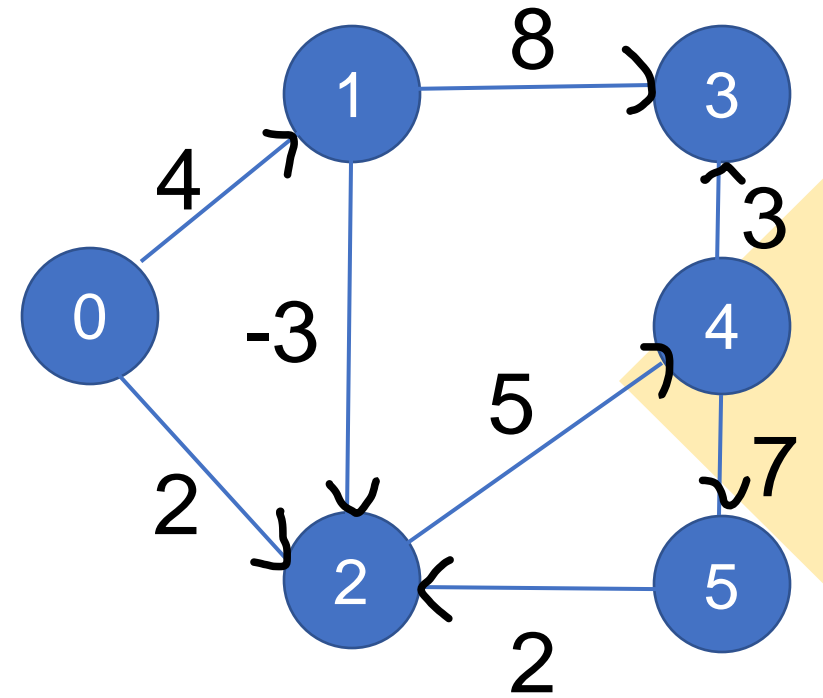


Tham lam

Bellman-Ford

Vấn đề:

- Min của distance tức thời chưa chắc là shortest



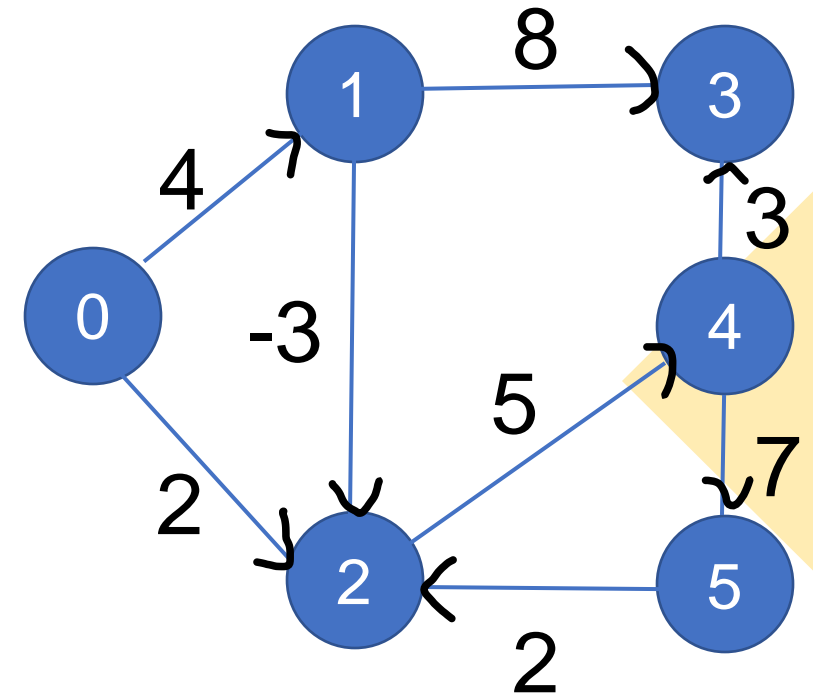
Bellman-Ford

Thuật toán khác với Dijkstra:

Không tìm min của
 $dst[]$ của mỗi vòng lặp

Những node được
cập nhật distance
đều được đưa vào
queue

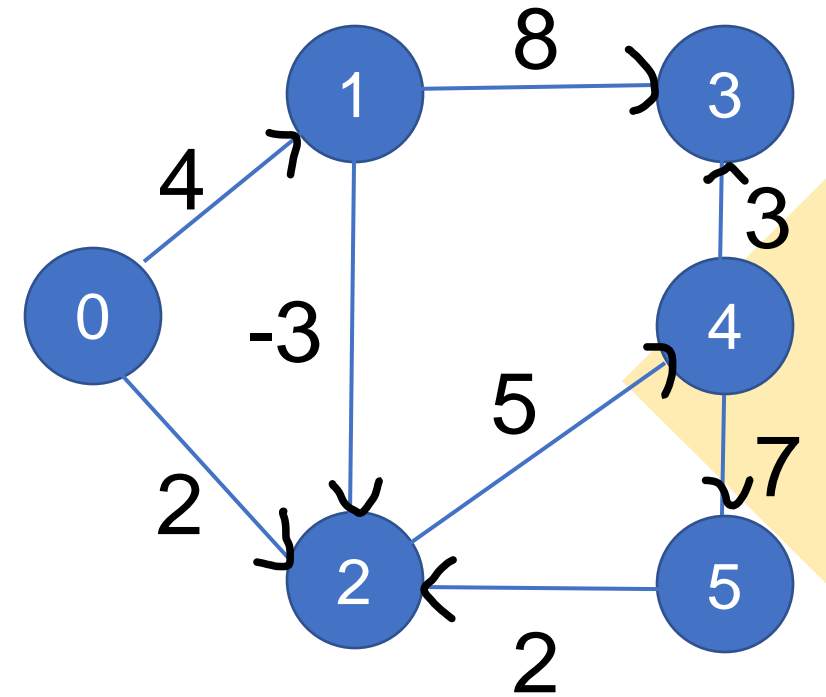
Chỉ khi kết thúc tất cả
vòng lặp mới chốt tất
cả distance



Bellman-Ford

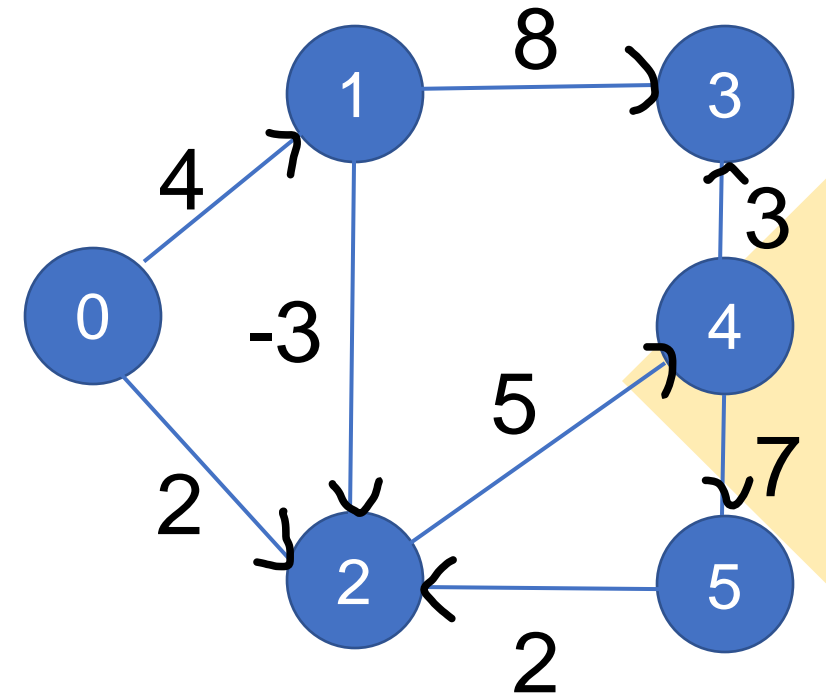
Hướng giải quyết:

- Cập nhật distance
- Điểm nào vừa được cập nhật cho vào queue
- Thực hiện tìm kiếm từ phần tử lấy ra từ queue
- Đưa hướng nhìn thoải mái, không điều kiện gì
- Thực hiện vòng lặp cho đến khi queue hết



Bellman-Ford

0	1	2	3	4	5
0	inf	inf	inf	inf	inf

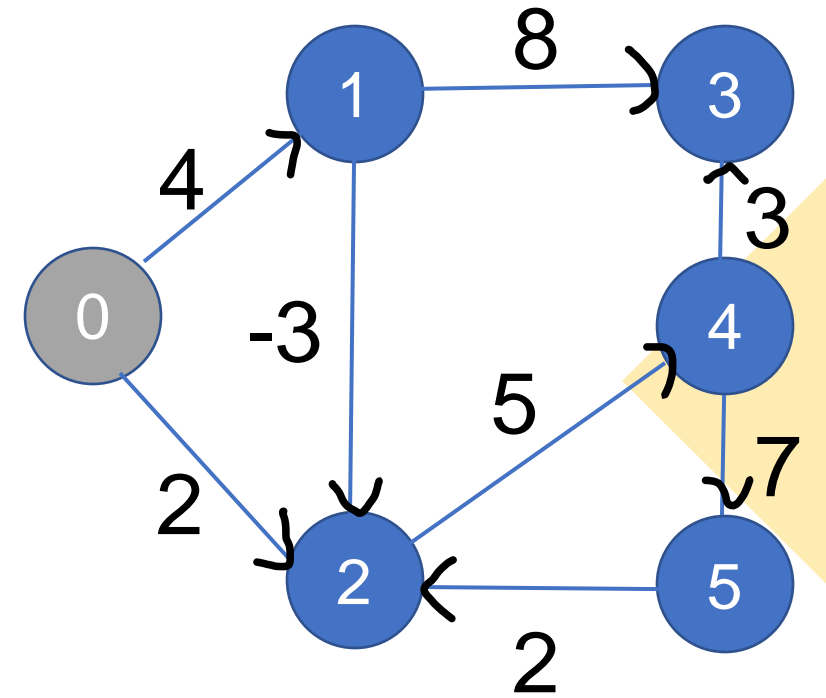


queue



Bellman-Ford

	0	1	2	3	4	5
0		inf	inf	inf	inf	inf
(0)		4	2			

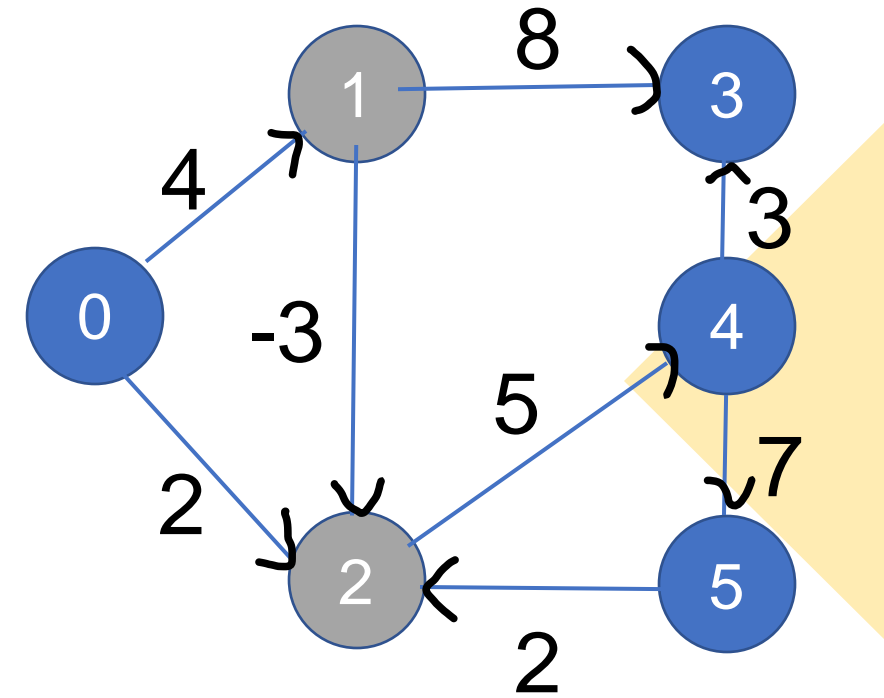


queue

1	2				
---	---	--	--	--	--

Bellman-Ford

		0	1	2	3	4	5
	0	0	inf	inf	inf	inf	inf
(0)	(1)		4	2			
(1)	(2)			1	12	7	



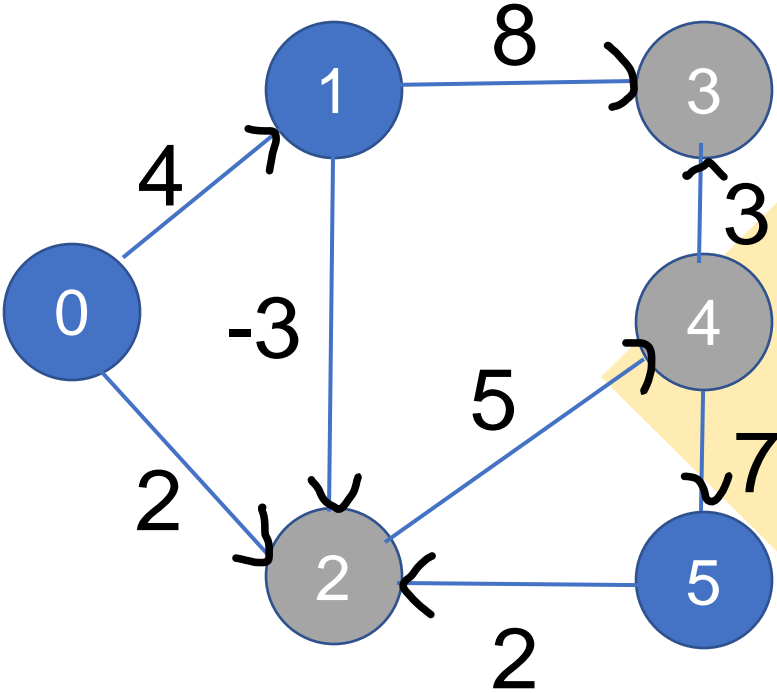
queue

2	3	4			
---	---	---	--	--	--

Bellman-Ford

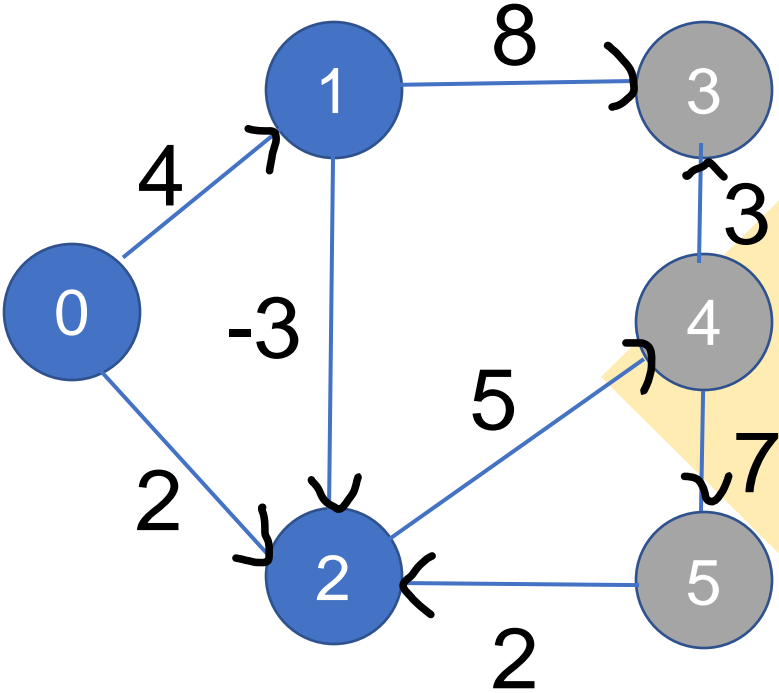
		0	1	2	3	4	5
	0	0	inf	inf	inf	inf	inf
(0)			4	2			
(1)	(2)			1	12	7	
(2)	(3)	(4)			10	6	14

queue 3 4 5



Bellman-Ford

			0	1	2	3	4	5
			0	inf	inf	inf	inf	inf
		(0)		4	2			
	(1)	(2)			1	12	7	
(2)	(3)	(4)				10	6	14
(4)	(3)	(5)				9		13



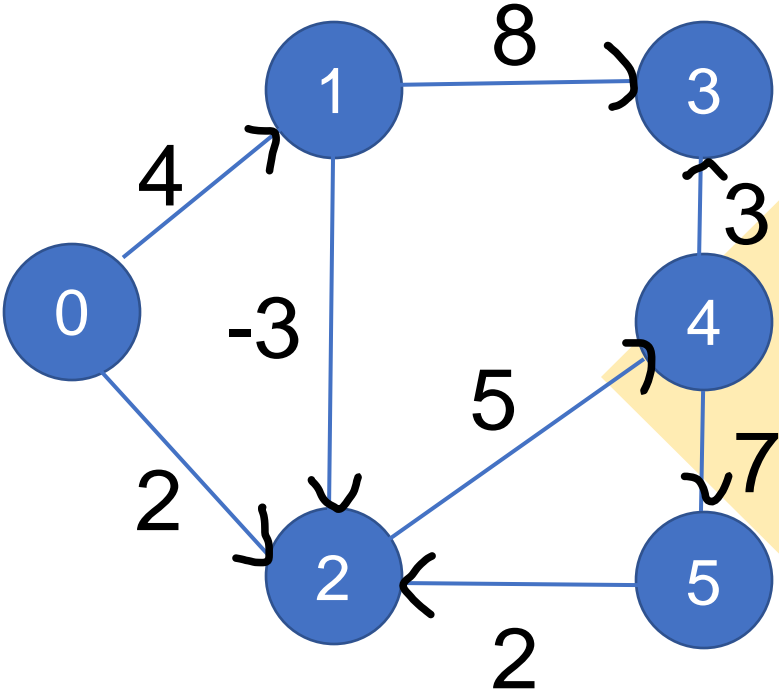
queue

3	5				
---	---	--	--	--	--

Bellman-Ford

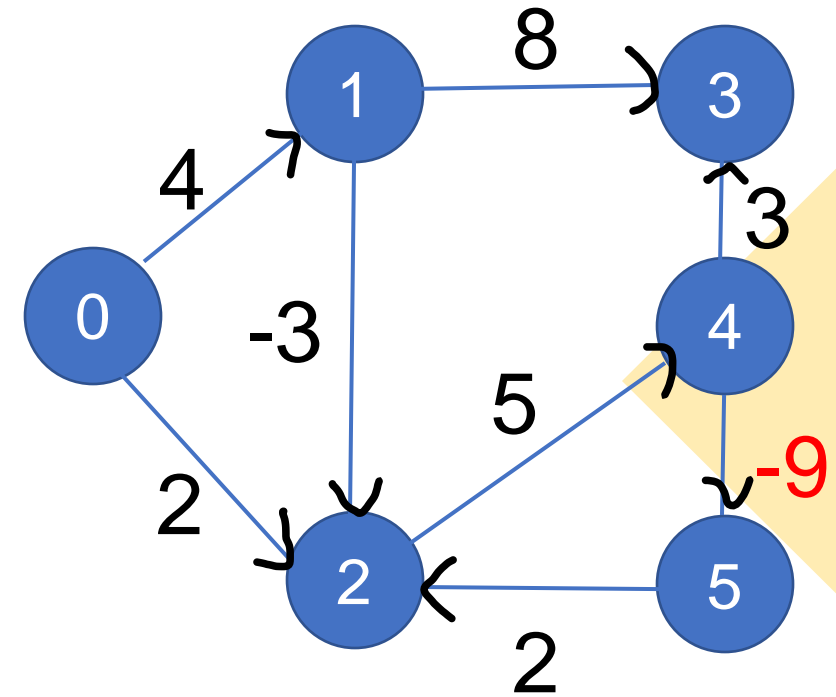
			0	1	2	3	4	5
			0	inf	inf	inf	inf	inf
(0)				4	2			
(1)	(2)				1	12	7	
(2)	(3)	(4)				10	6	14
(4)	(3)	(5)				9		13
(3)	(5)							
			0	4	1	9	6	13

queue



Bellman-Ford

*Làm sao để vòng lặp
dừng lại khi có chu trình
âm?*



cUpdate

0	1	2	3	4	5
0	0	0	0	0	0

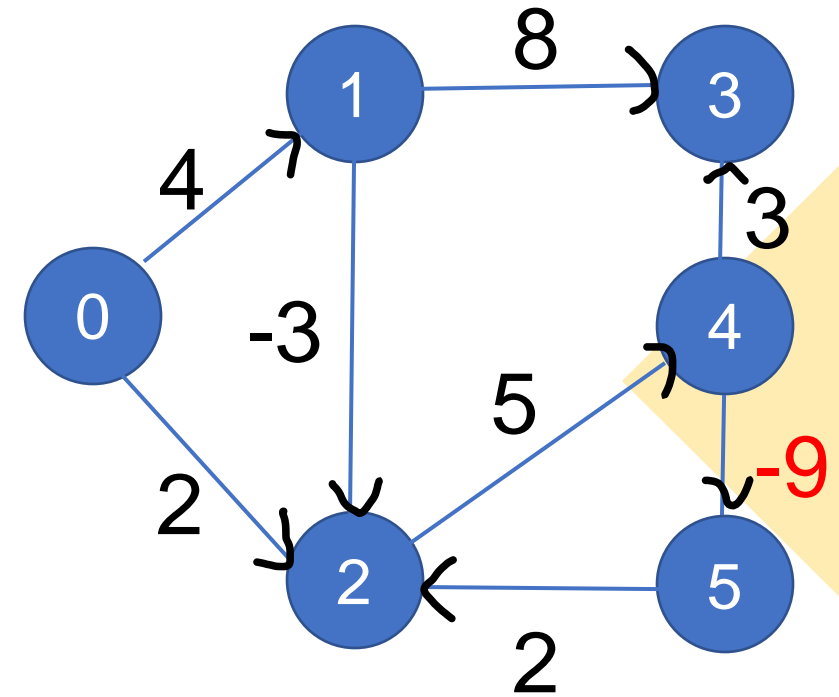
Nếu $\text{cUpdate}[u] > |V|$ thì return
"Có chu trình âm"

Bellman-Ford

Dùng cho đồ thị có trọng số âm. Nhưng:

- Không phải là đồ thị có chu trình âm
- Không phải là đồ thị vô hướng có trọng số âm

*Độ phức tạp thời gian: $O(V^*E)$*



**Quy hoạch
động**

Graph Algorithms

3. Tổng kết

- Khi nào dùng đồ thị
- So sánh DFS với BFS



Khi nào dùng đồ thị

- ❖ Khi input đề là các đối tượng và có kết nối giữa các đối tượng đó, theo 1 chiều hoặc 2 chiều

So sánh DFS, BFS

DFS

BFS

