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

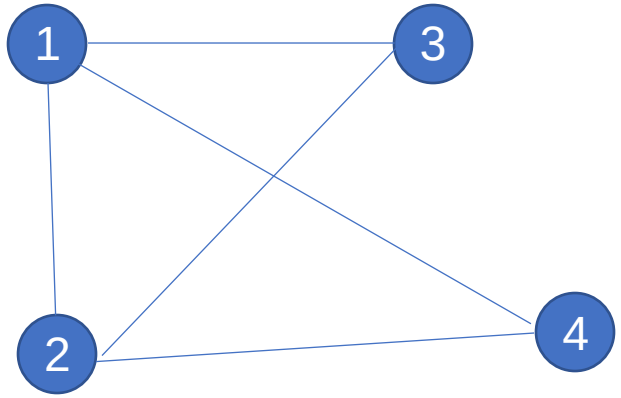
- Đồ thị
- Graph Algorithm

Graph Algorithms là gì nào?

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

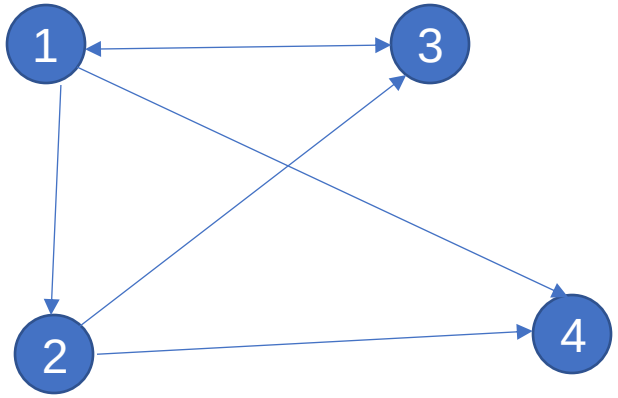
Đồ thị

- 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.



Đồ thị vô hướng

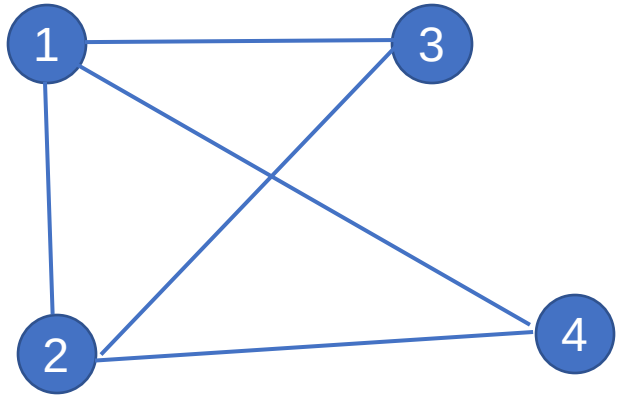
- Có 2 loại đồ thị: đồ thị có hướng và đồ thị vô hướng



Đồ thị có hướng

- Có 2 cách để biểu diễn đồ thị:
  Ma trận đỉnh kề hoặc Danh sách đỉnh kề

Đồ thị



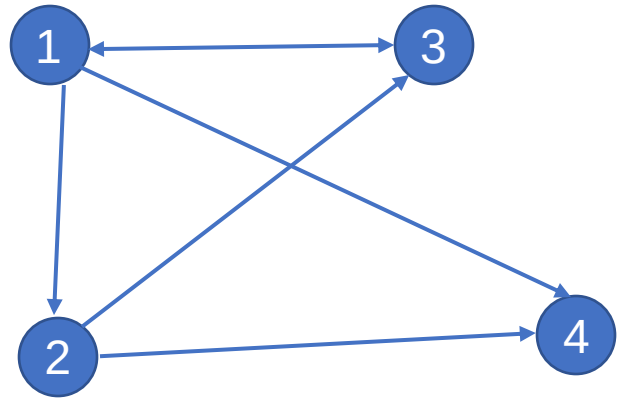
	1	2	3	4
1	0	1	1	1
2	1	0	1	1
3	1	1	0	0
4	1	1	0	0

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:

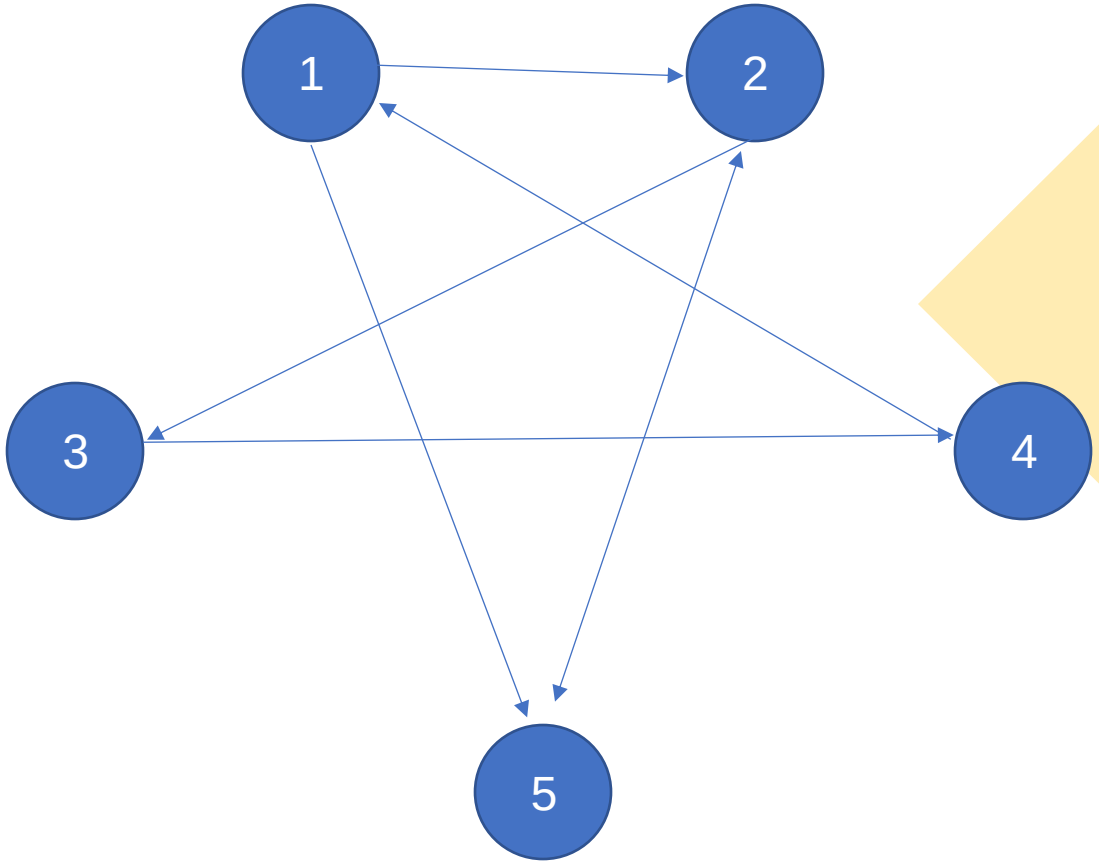
```
from collections import defaultdict
class Graph:
#Thêm thư viện defaultdict để tạo và quản lí đồ thị

    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 (Search)



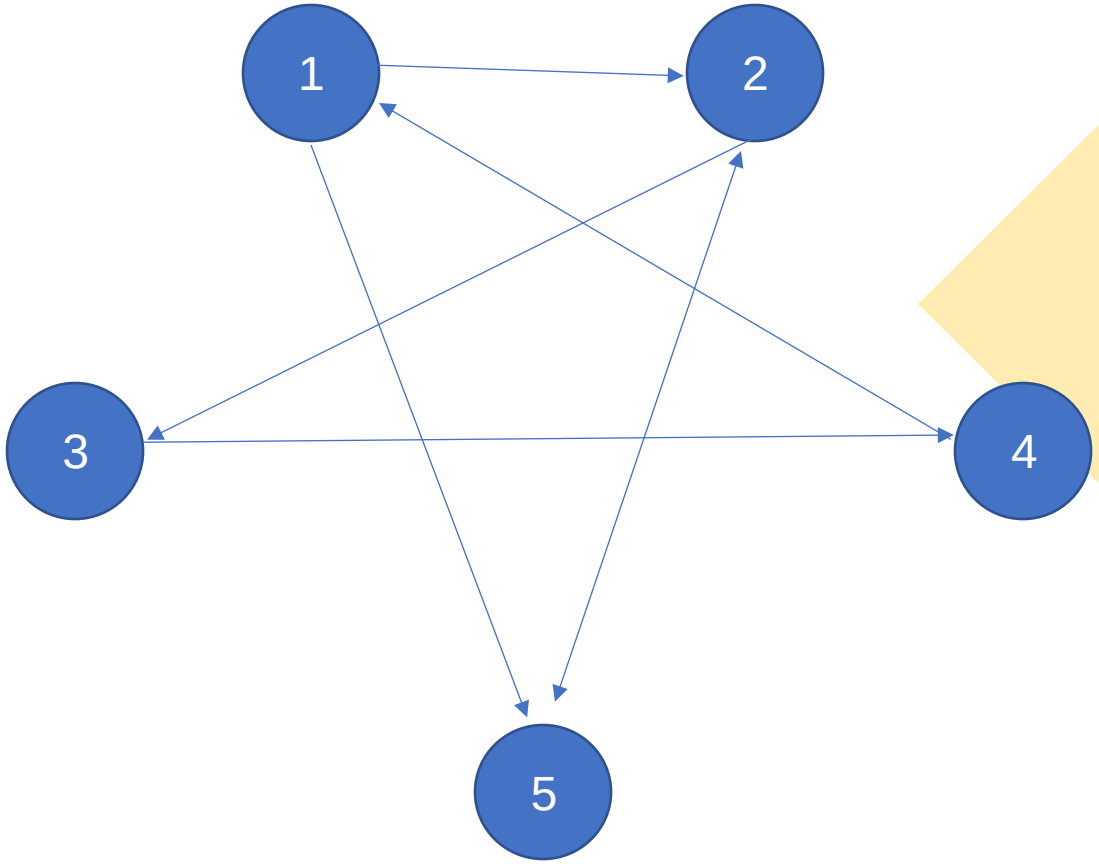
Depth First Traversal là **một thuật toán để duyệt các đỉnh** trong cây tìm kiếm hoặc đồ thị.

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

Depth First Traversal

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

2. Hướng tầm nhìn xung quanh, gặp đỉnh kề nào chưa visit thì (đánh dấu visit và gọi đệ quy) lên đỉnh đó



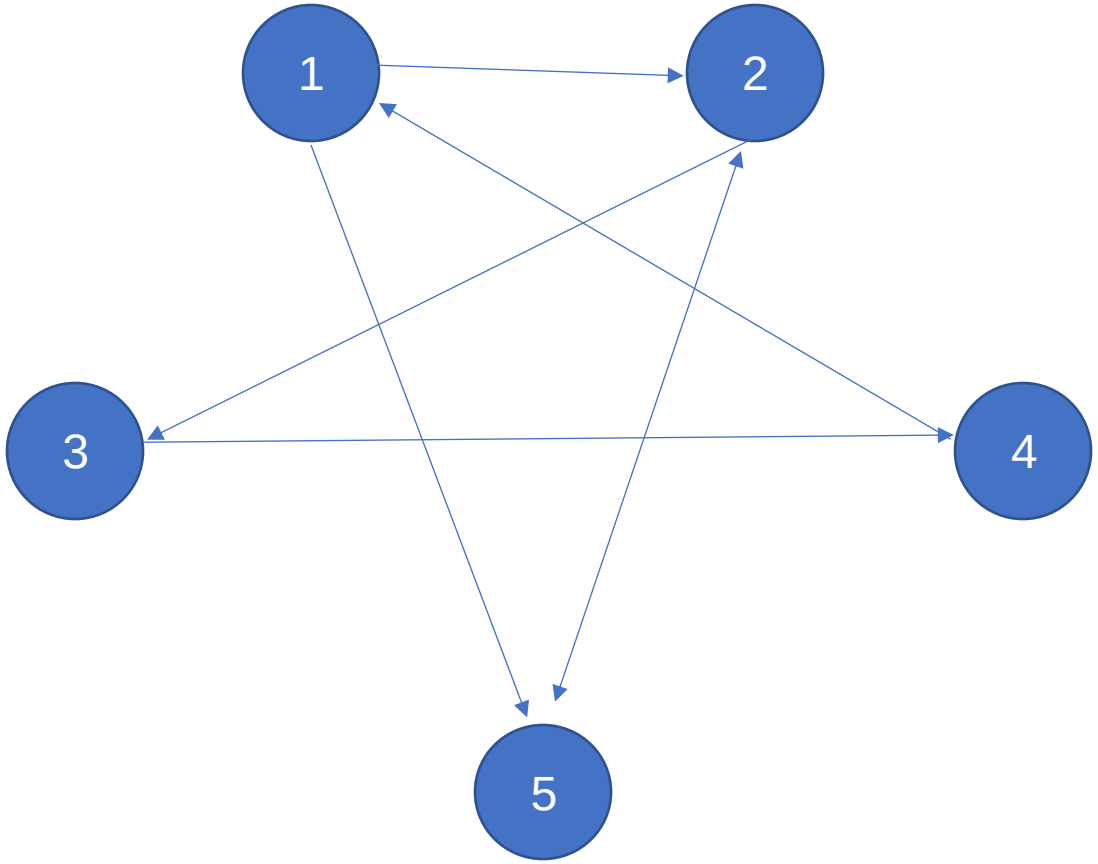
Depth First Traversal

Visited

1	2	3	4	5
0	0	0	0	0

Stack

Print:



Depth First Traversal

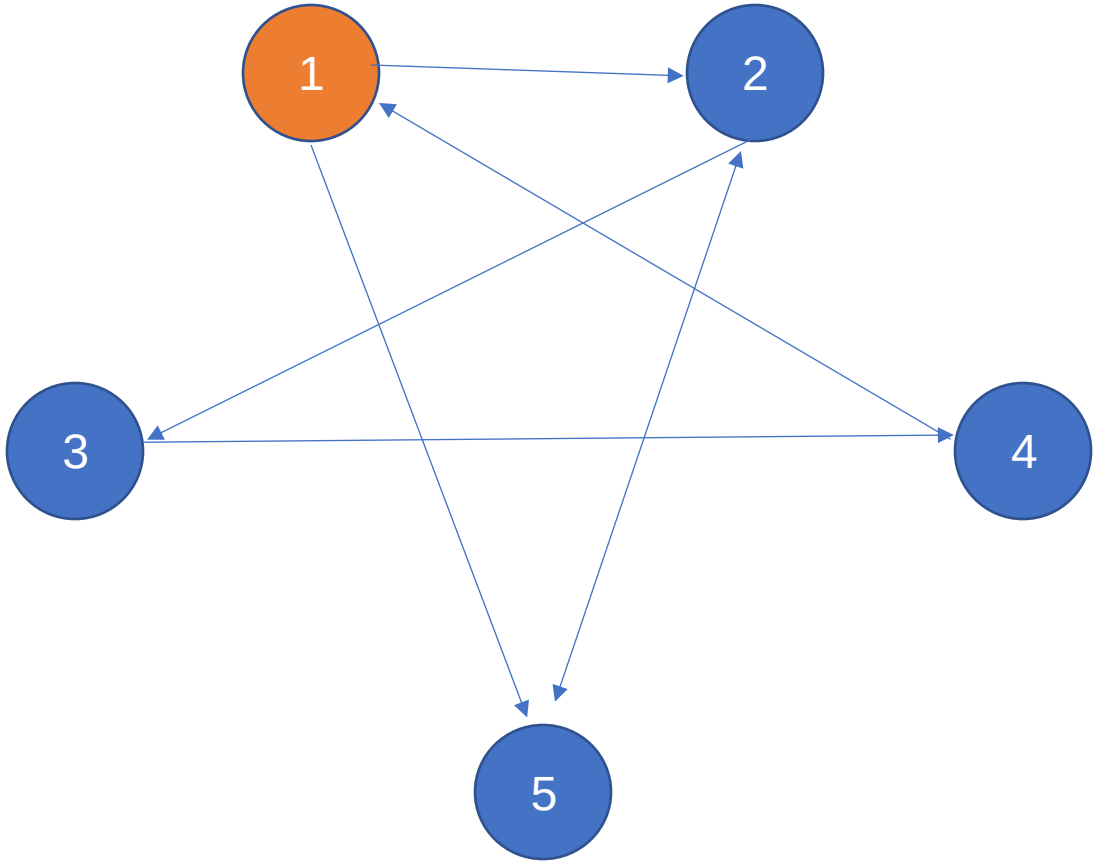
Visited

1	2	3	4	5
T	F	F	F	F

Stack

1				

Print: 1



Depth First Traversal

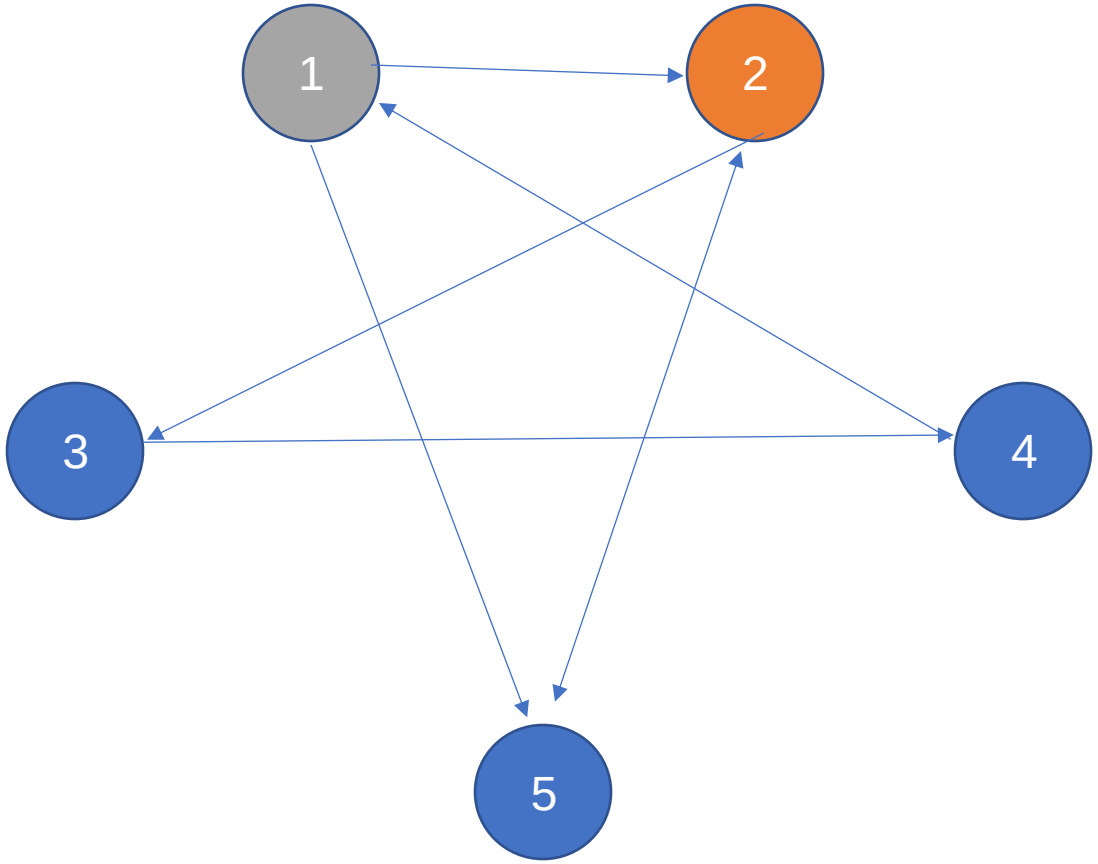
Visited

1	2	3	4	5
T	T	F	F	F

Stack

1	2			

Print: 1 2



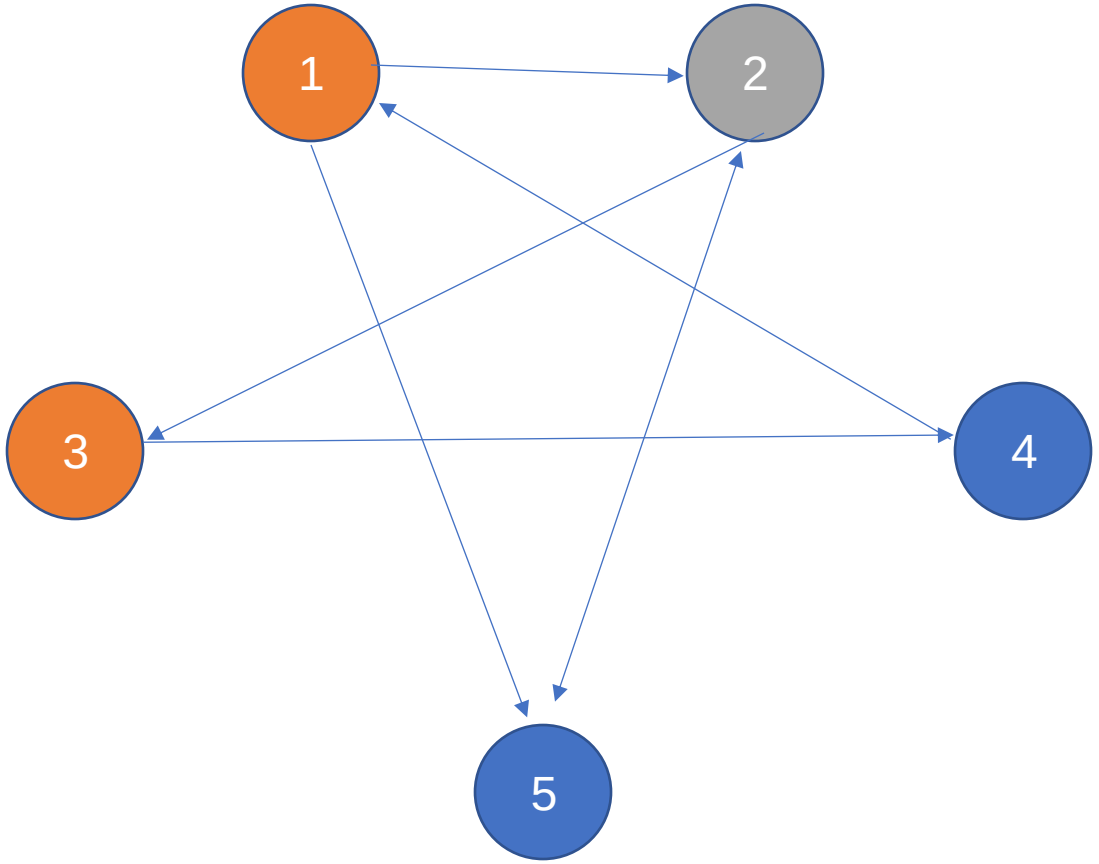
Depth First Traversal

Visited

1	2	3	4	5
T	T	T	F	F

Stack

1	2	3		



Print: 1 2 3

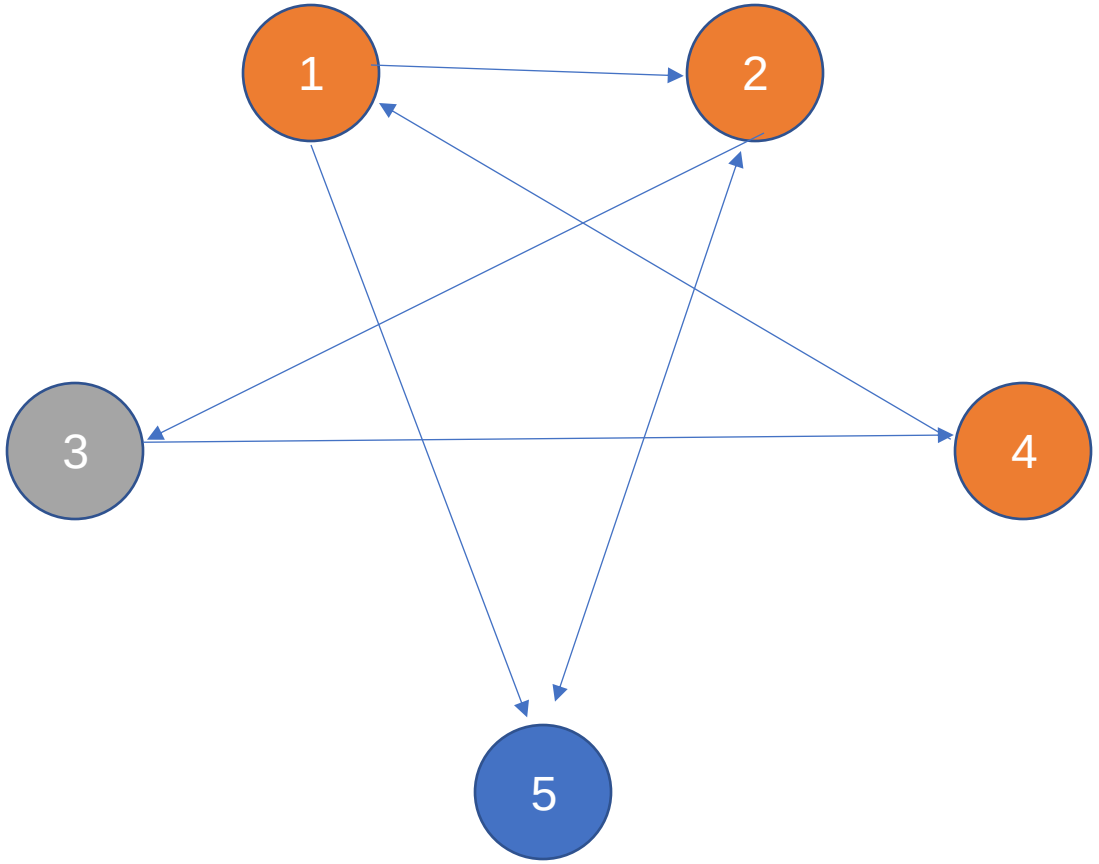
Depth First Traversal

Visited

1	2	3	4	5
T	T	T	T	F

Stack

1	2	3	4	



Print: 1 2 3 4

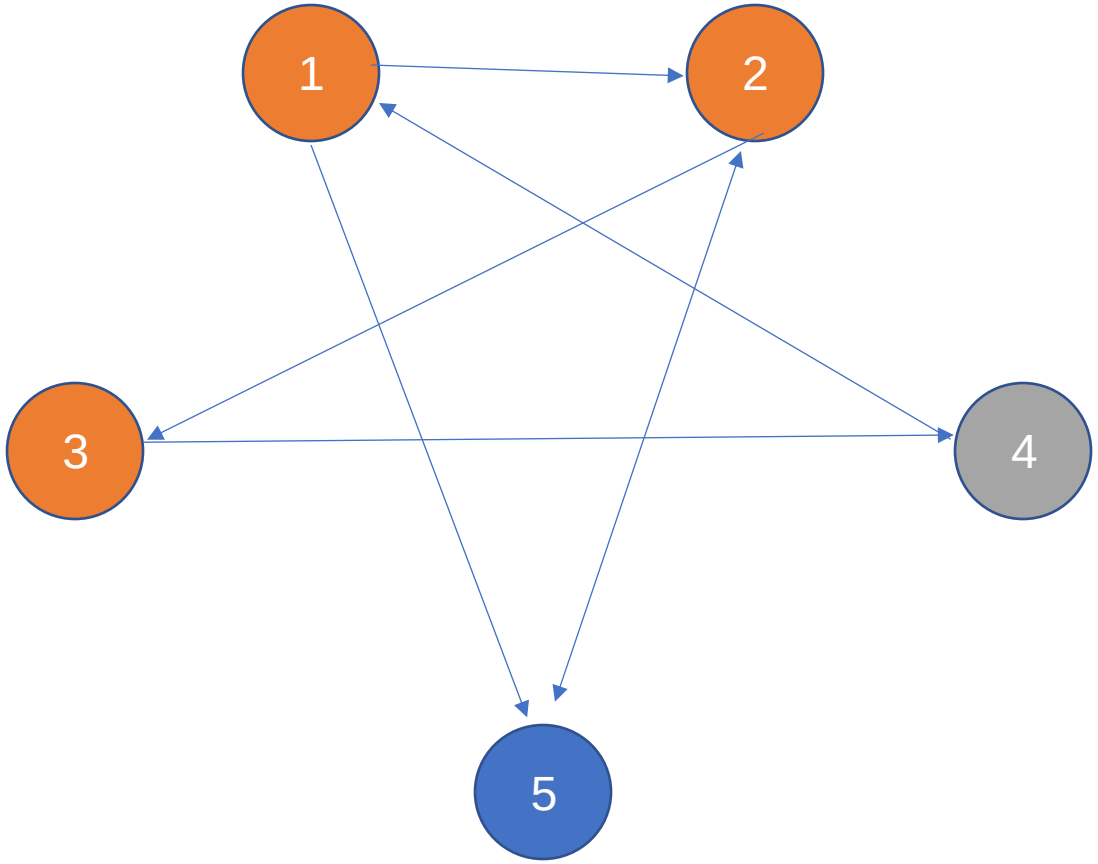
Depth First Traversal

Visited

1	2	3	4	5
T	T	T	T	F

Stack

1	2	3	4	



Print: 1 2 3 4

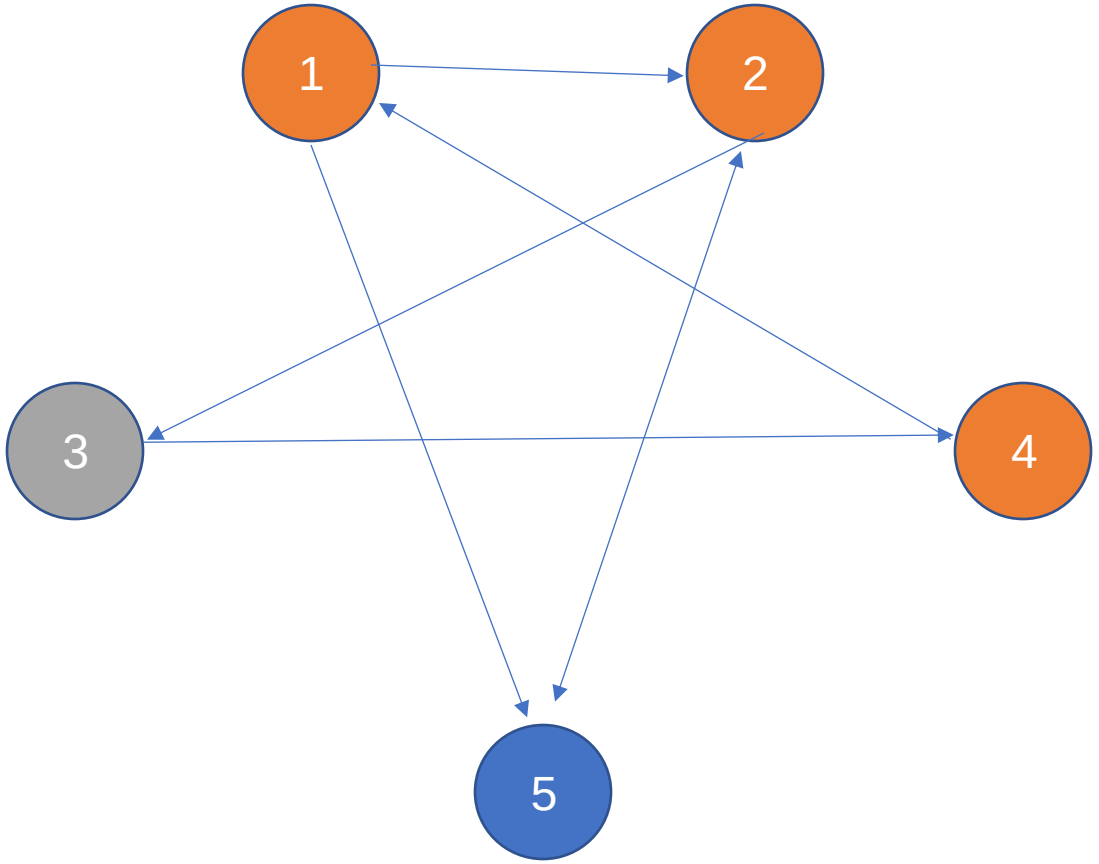
Depth First Traversal

Visited

1	2	3	4	5
T	T	T	T	F

Stack

1	2	3		



Print: 1 2 3 4

Depth First Traversal

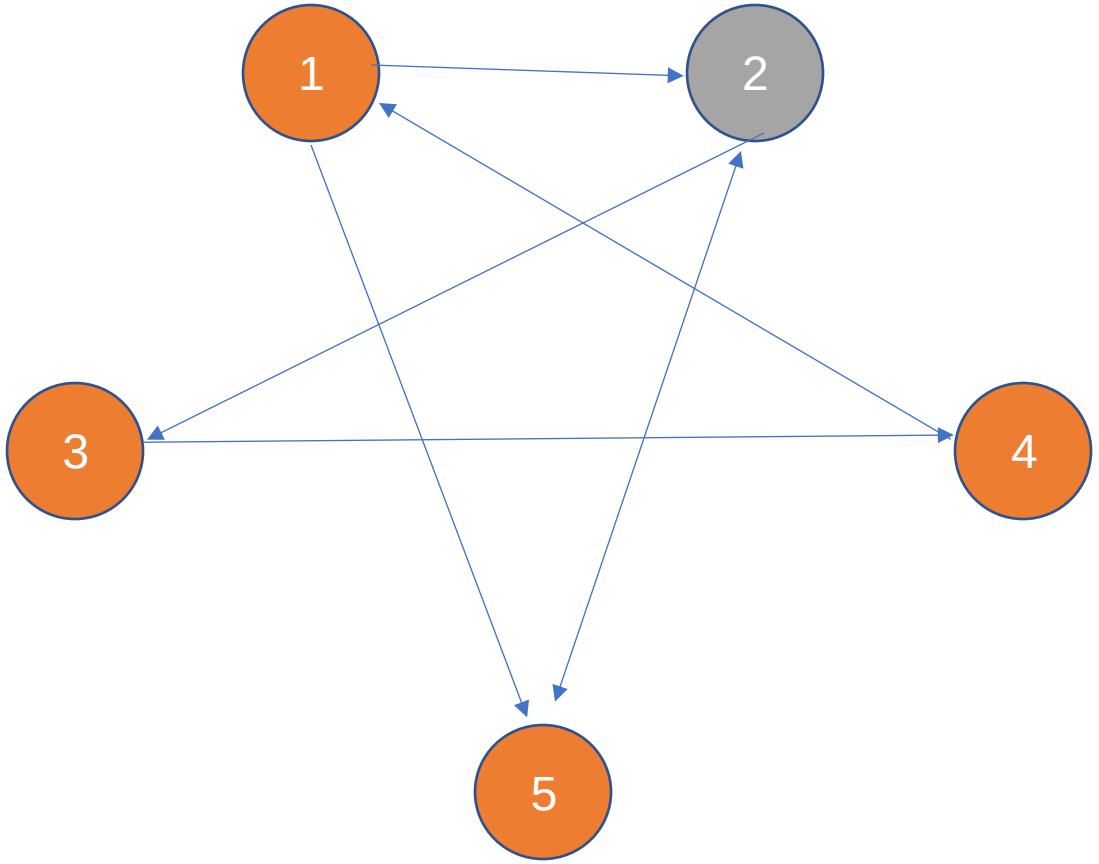
Visited

1	2	3	4	5
T	T	T	T	T

Stack

1	2	5		

Print: 1 2 3 4 5



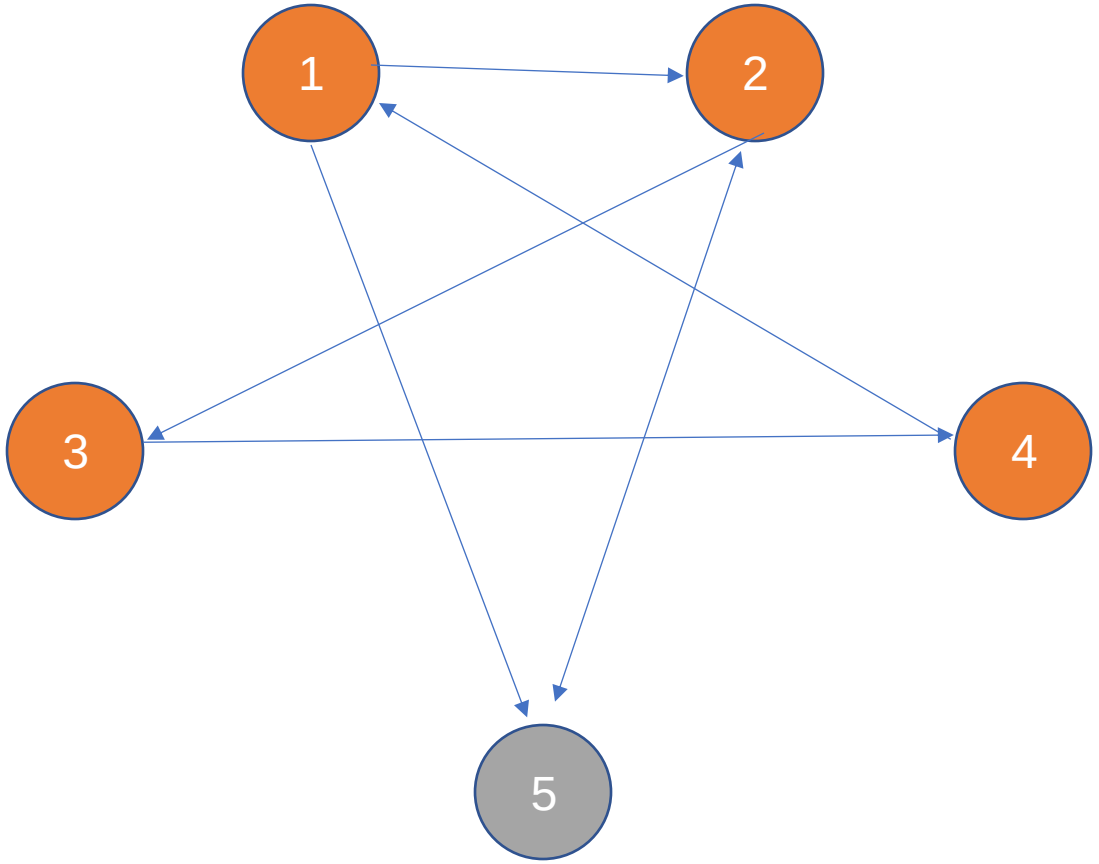
Depth First Traversal

Visited

1	2	3	4	5
T	T	T	T	T

Stack

1	2	5		



Print: 1 2 3 4 5

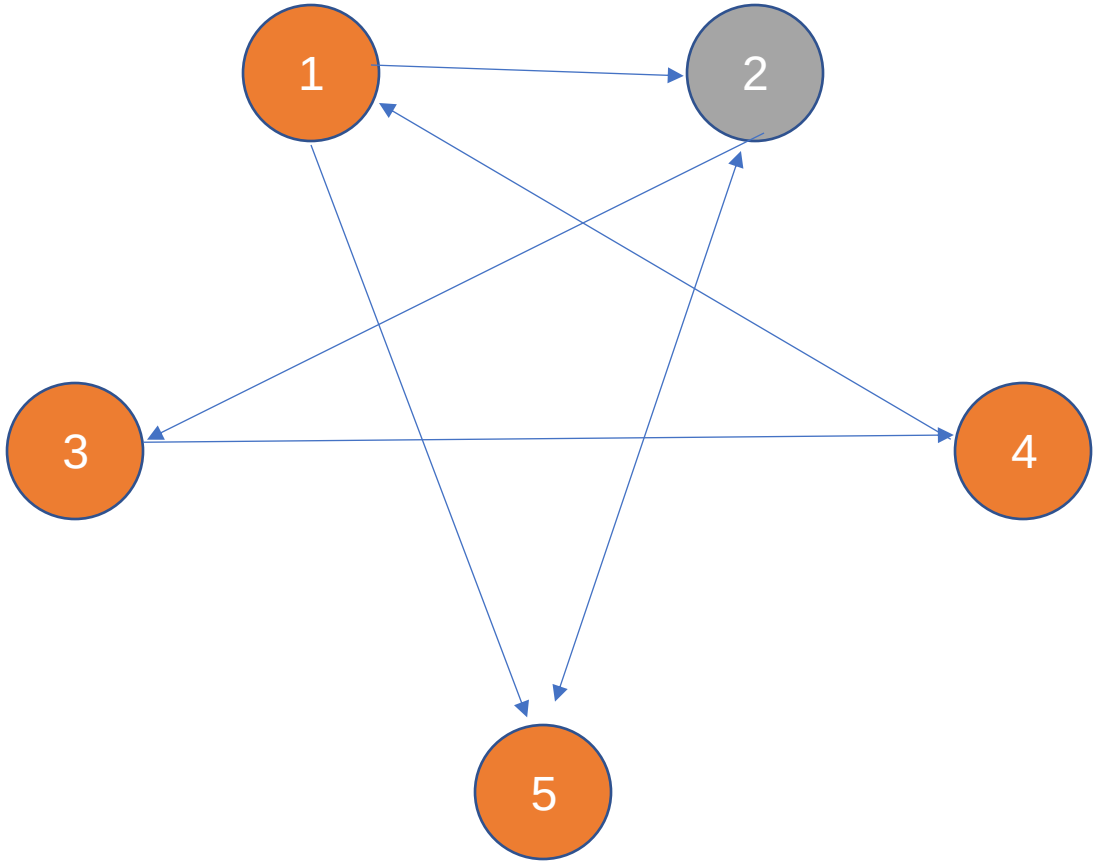
Depth First Traversal

Visited

1	2	3	4	5
T	T	T	T	T

Stack

1	2			



Print: 1 2 3 4 5

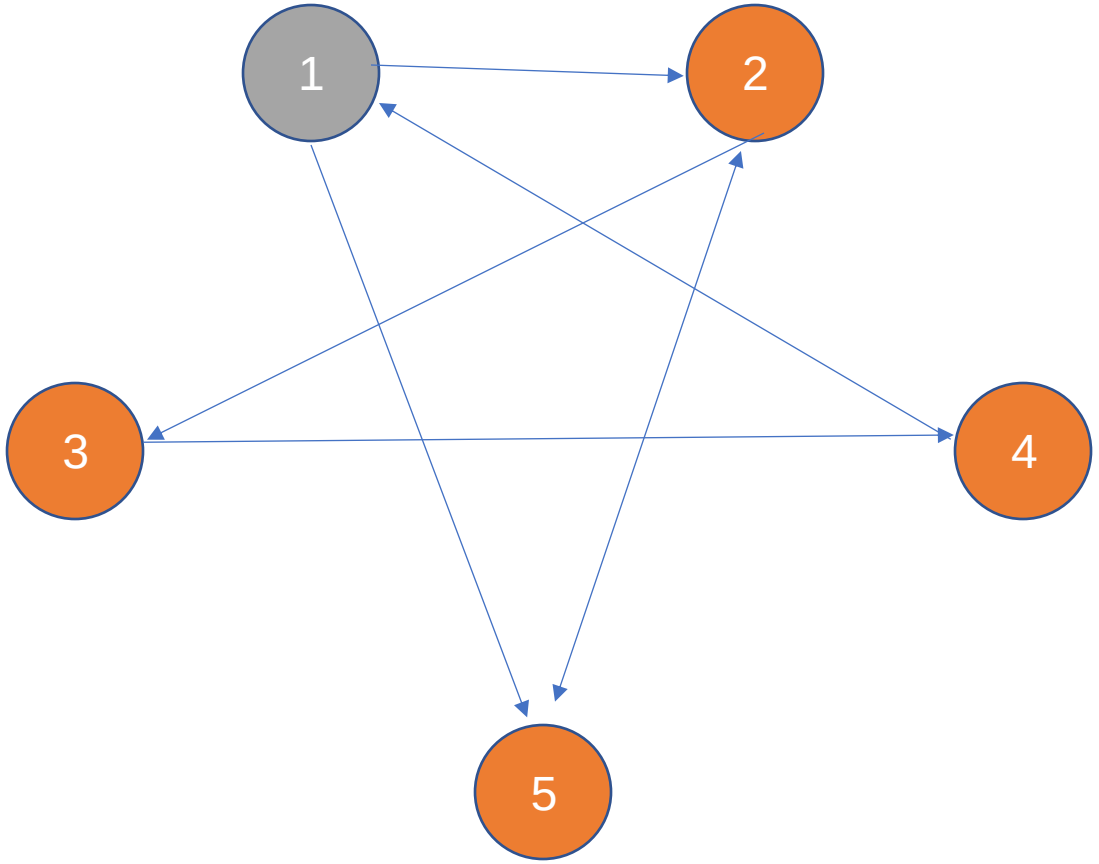
Depth First Traversal

Visited

1	2	3	4	5
T	T	T	T	T

Stack

1				



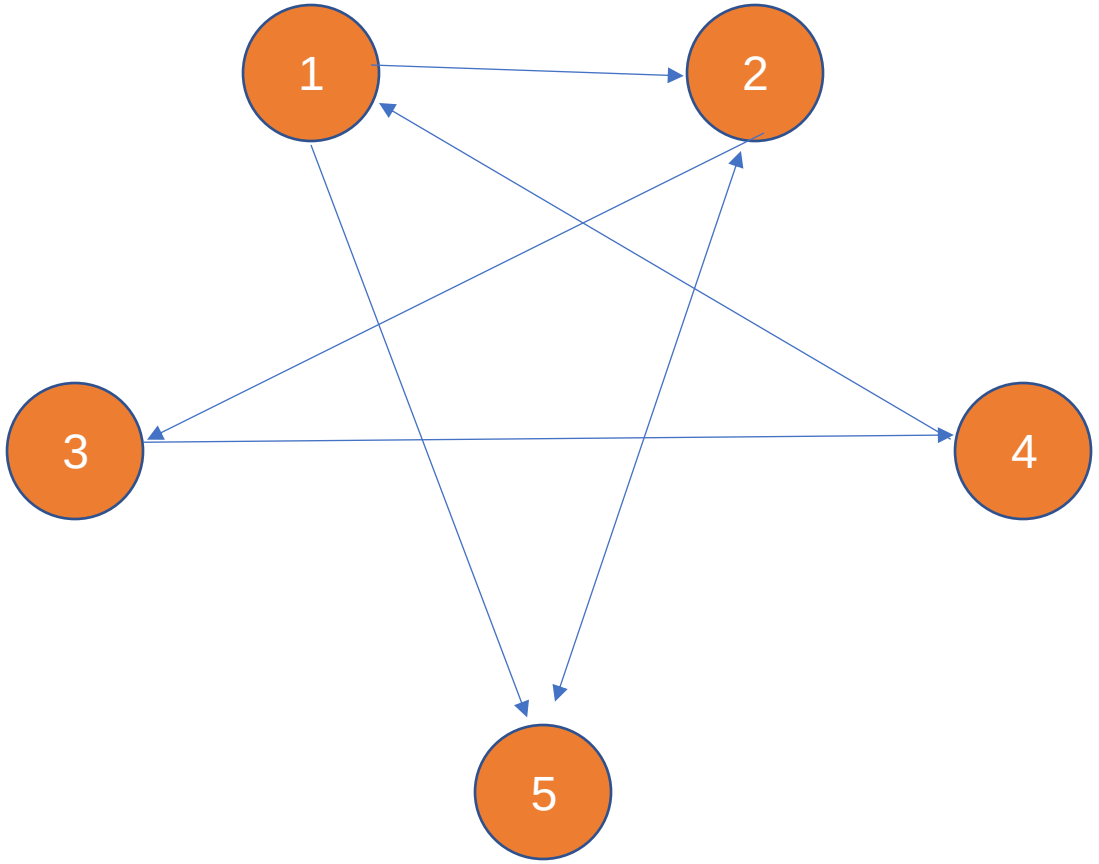
Print: 1 2 3 4 5

Depth First Traversal

Visited

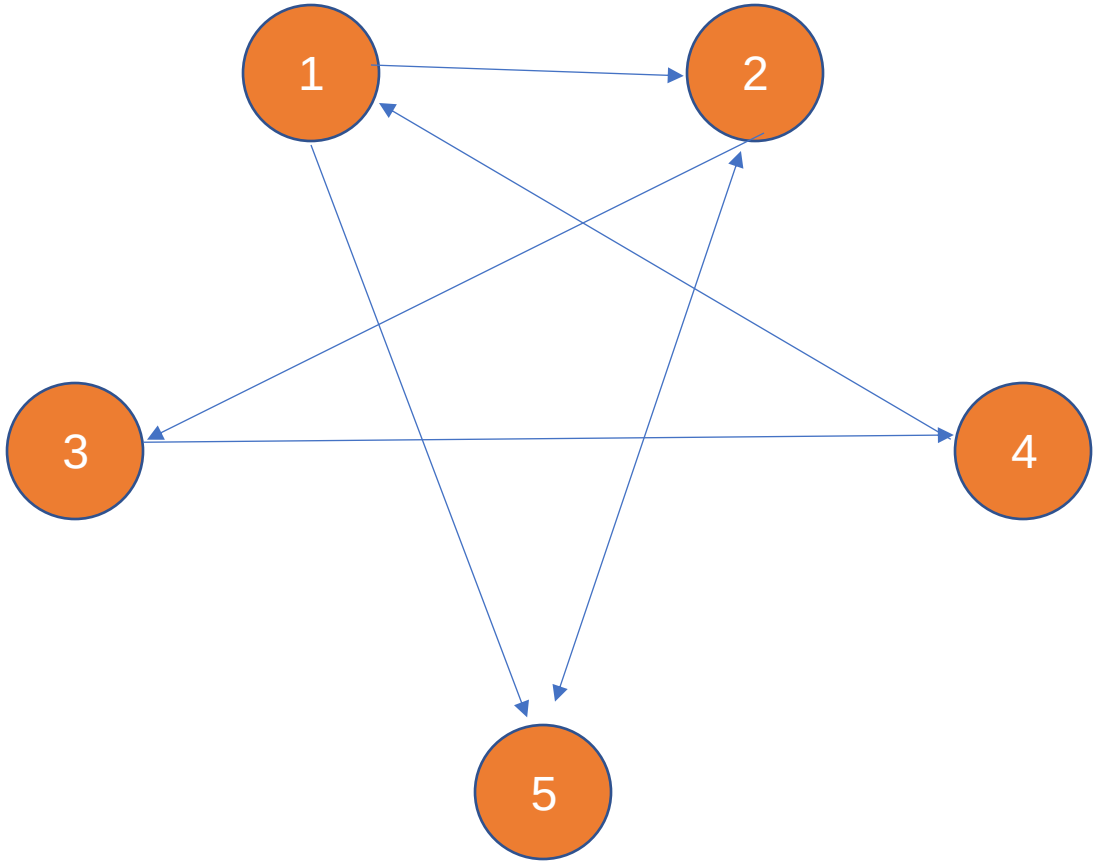
1	2	3	4	5
T	T	T	T	T

Stack



Print: 1 2 3 4 5

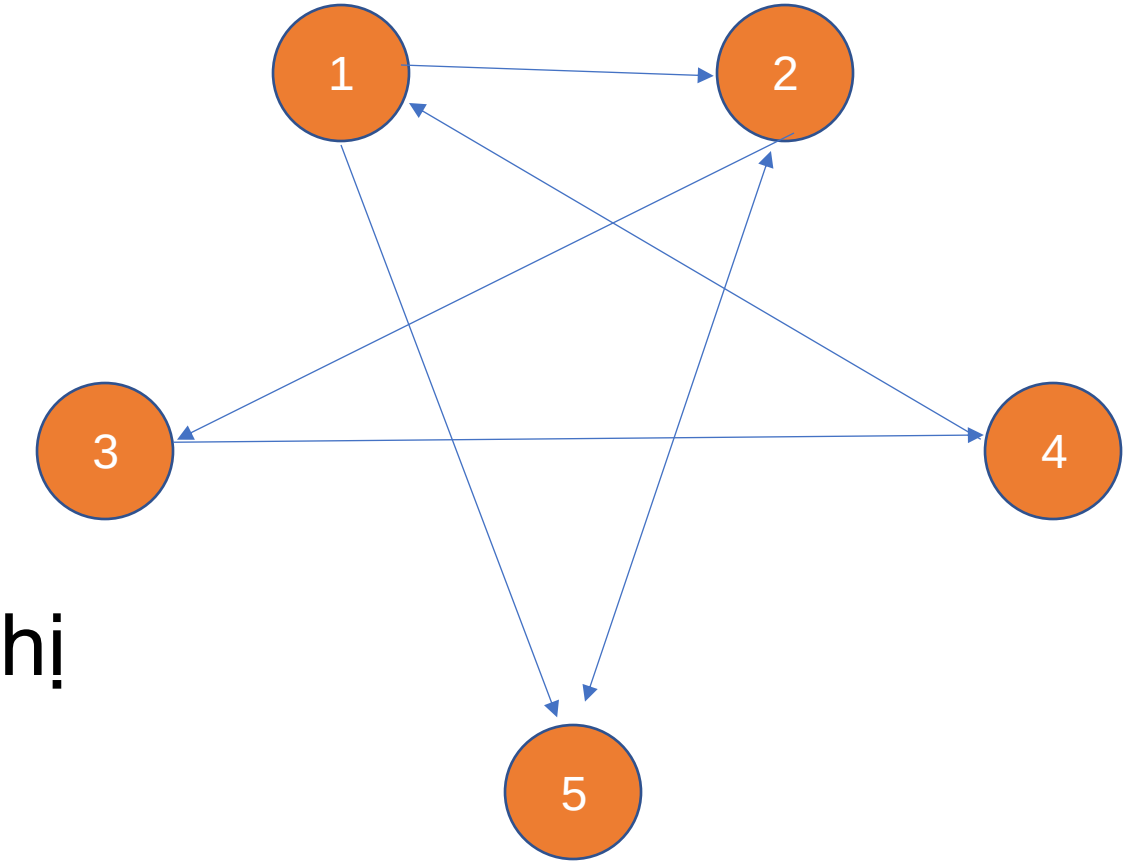
Depth First Traversal



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

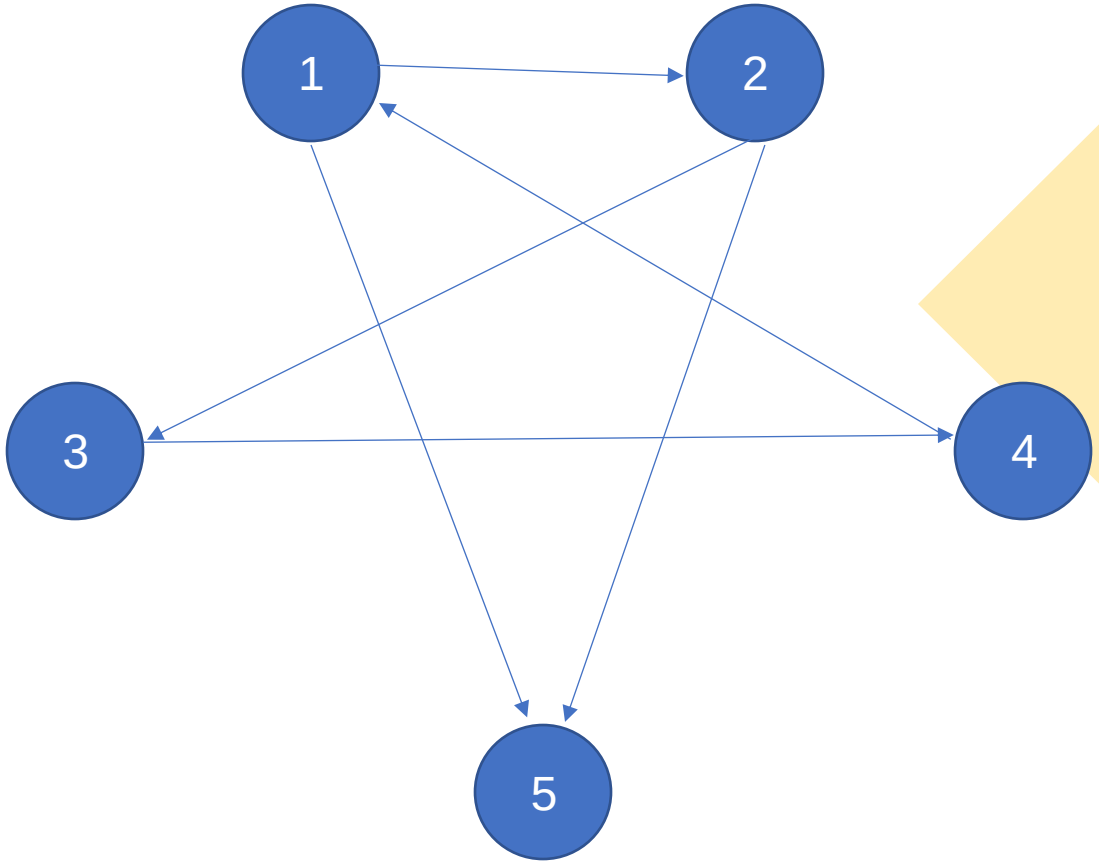


Ứng dụng:

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

Detect cycle in directed graph

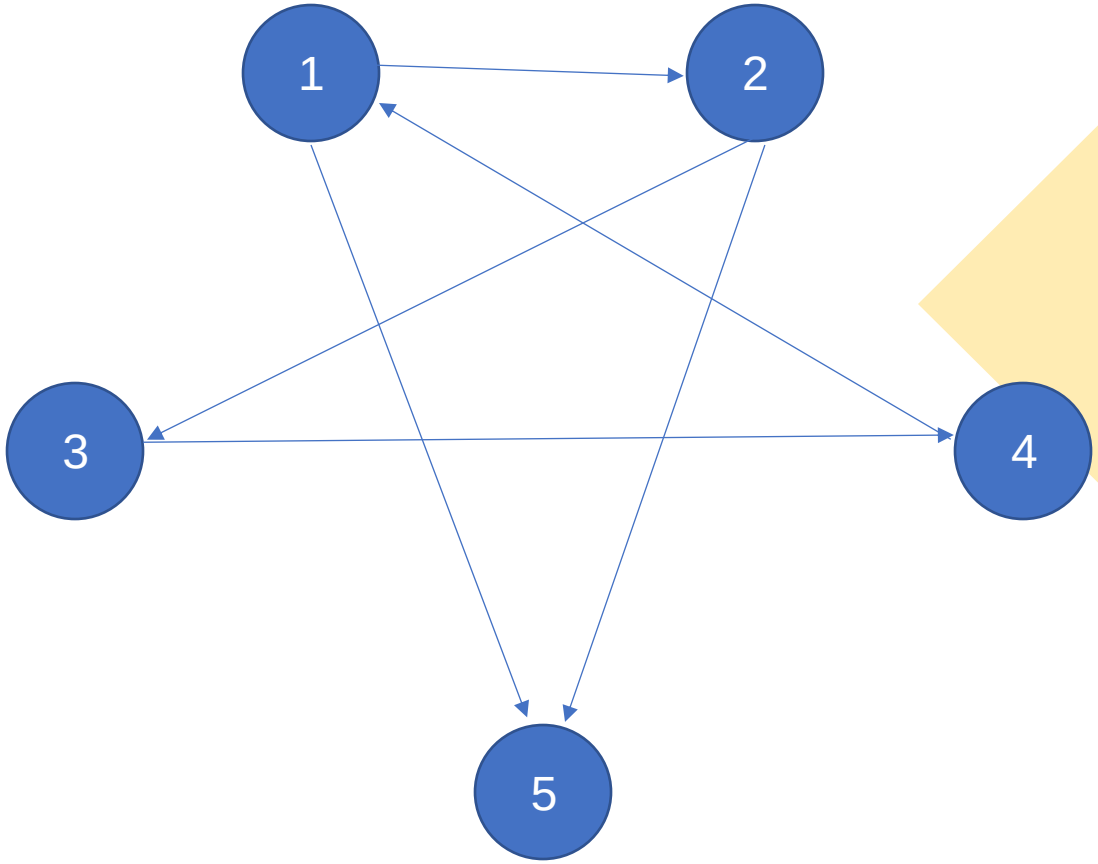
Một bảng đăng kí môn học có các môn học được đánh mã từ 1 đến 5. Ngoại trừ môn học có mã 1, các môn học khác đều có điều kiện tiên quyết của mình. Hãy kiểm tra bảng đăng kí này có hợp lệ hay không.



Detect cycle in directed graph

1. Cài đặt như DFS

2. Trong quá trình nhìn xung quanh, nếu tồn tại 1 đỉnh cận nằm trong stack, return True

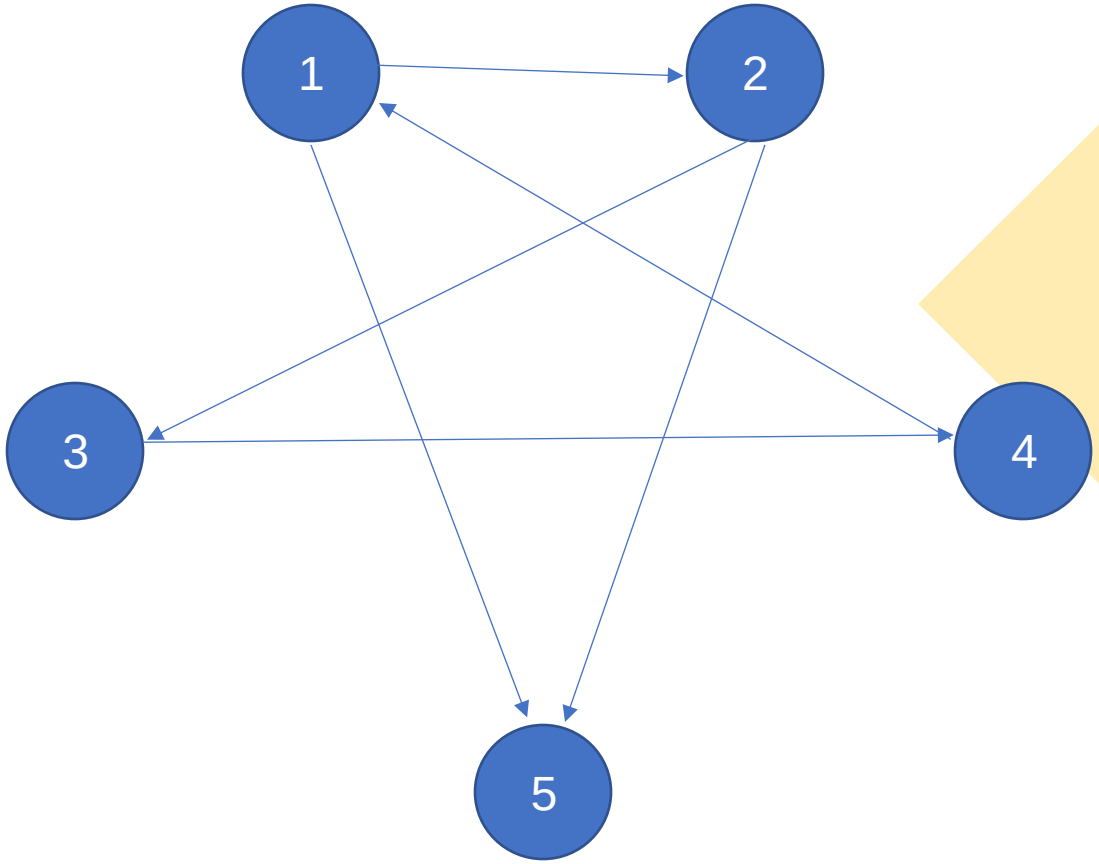


Detect cycle in directed graph

Visited

1	2	3	4	5
F	F	F	F	F

Stack



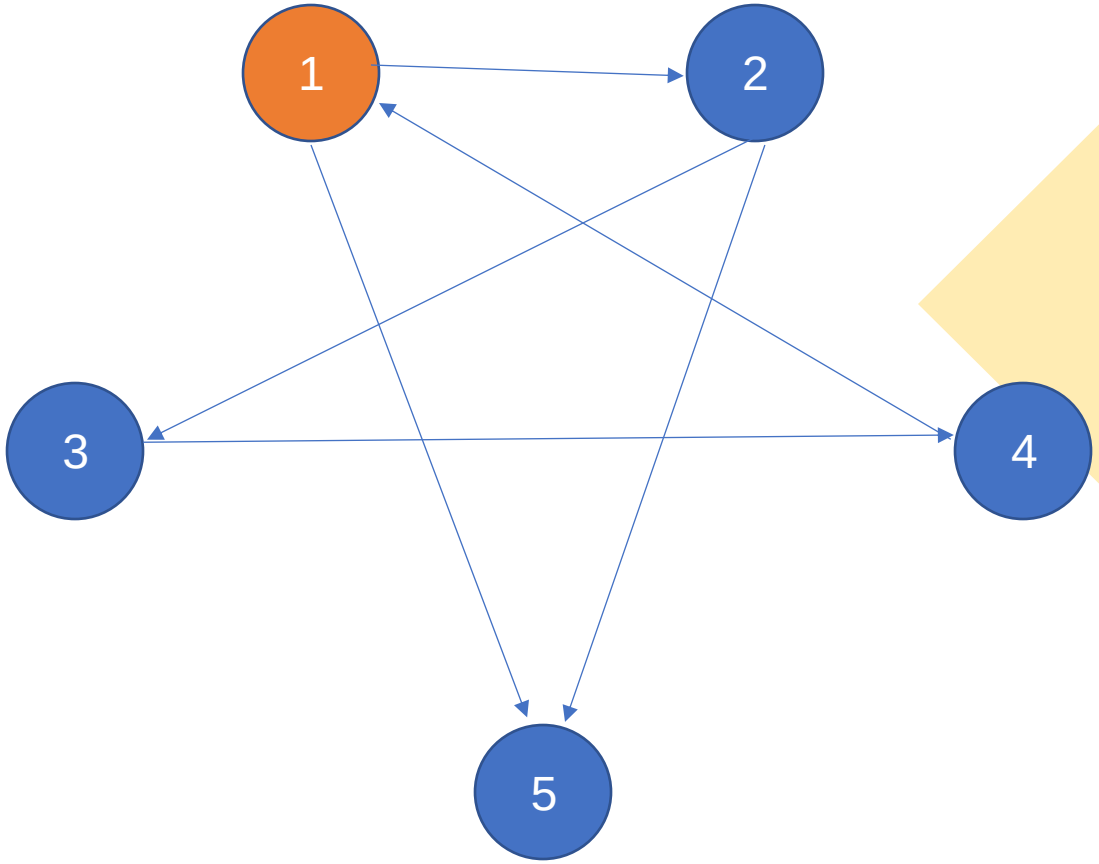
Detect cycle in directed graph

Visited

1	2	3	4	5
T	F	F	F	F

Stack

1				



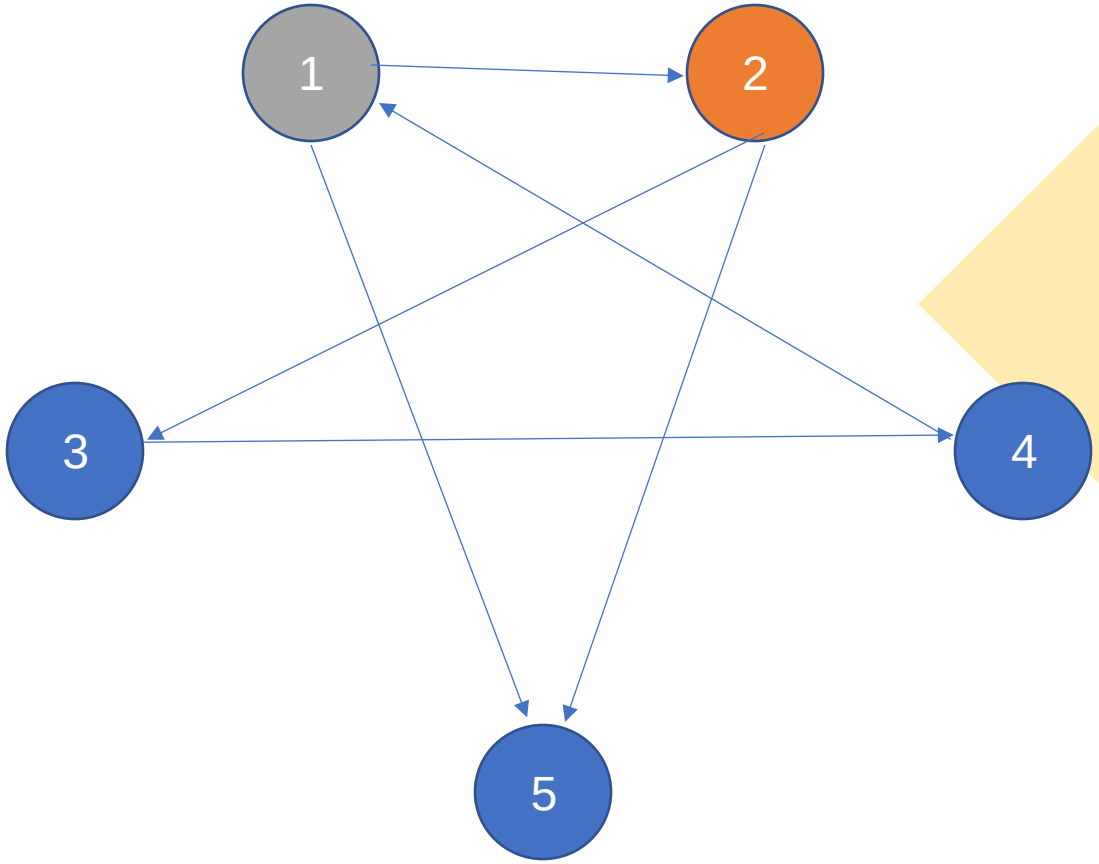
Detect cycle in directed graph

Visited

1	2	3	4	5
T	T	F	F	F

Stack

1	2			



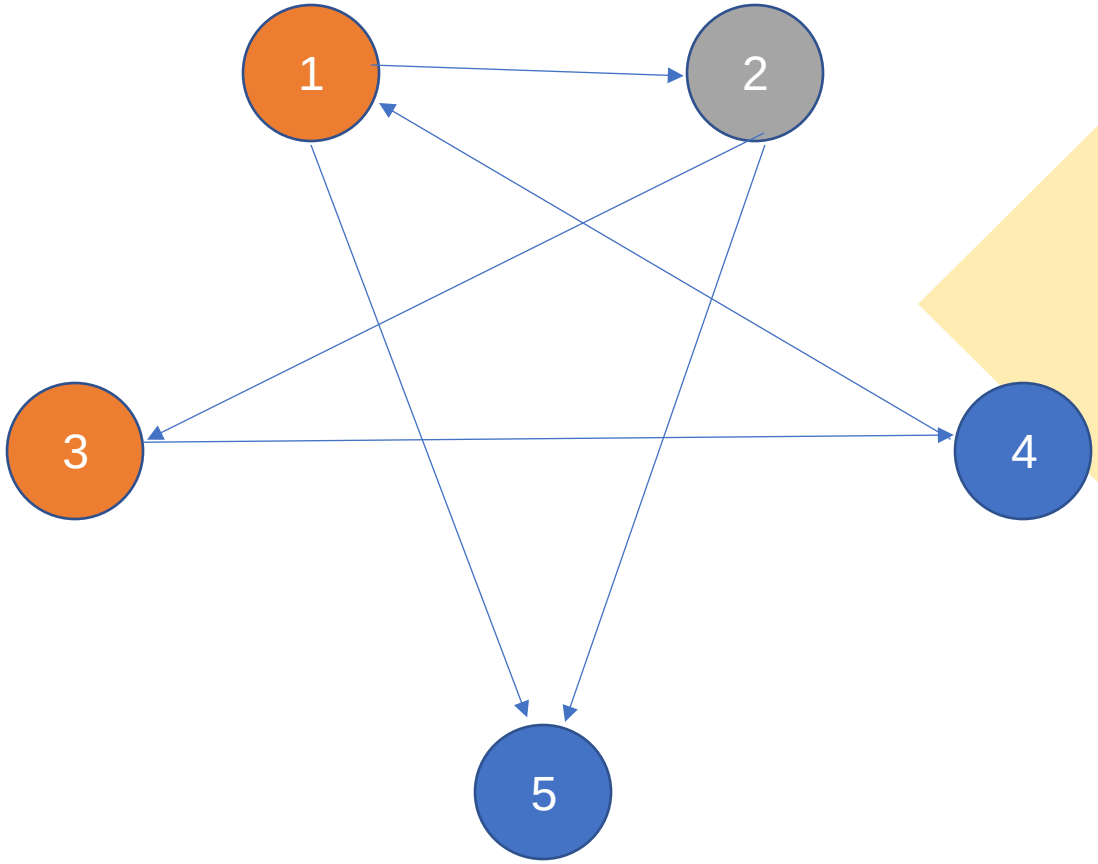
Detect cycle in directed graph

Visited

1	2	3	4	5
T	T	T	F	F

Stack

1	2	3		



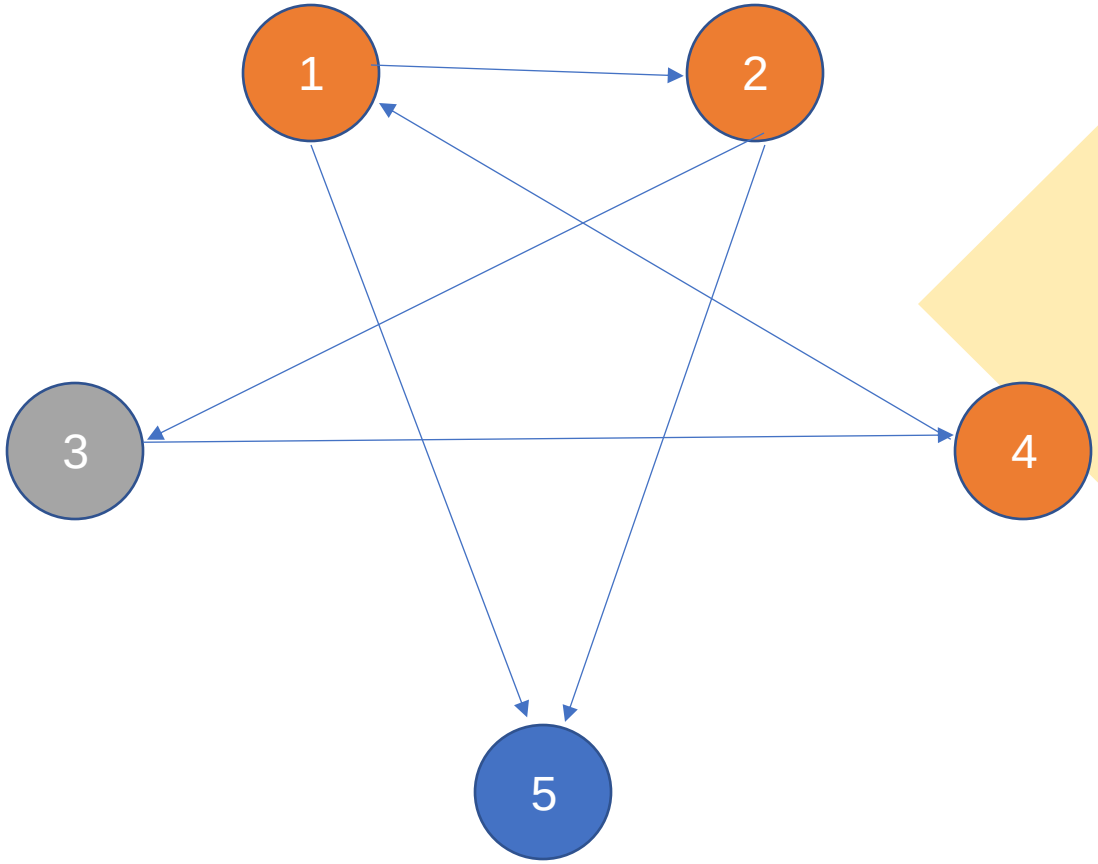
Detect cycle in directed graph

Visited

1	2	3	4	5
T	T	T	T	F

Stack

1	2	3	4	



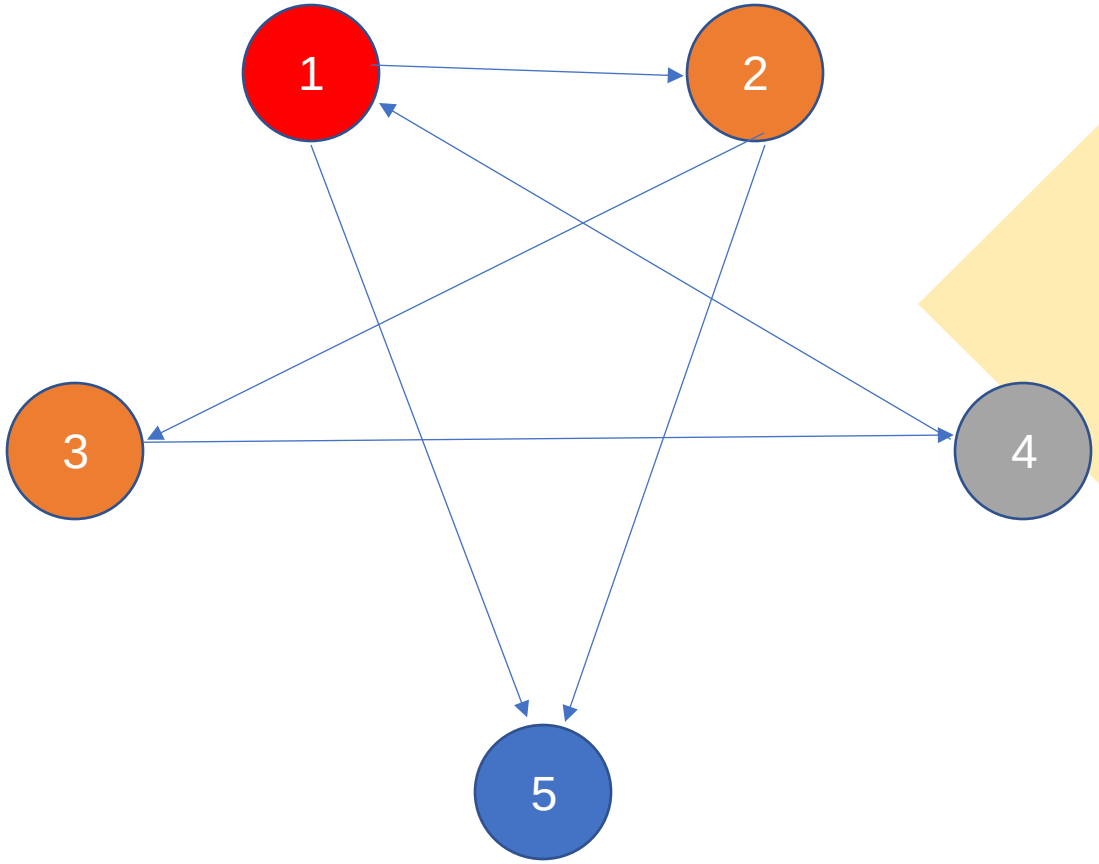
Detect cycle in directed graph

Visited

1	2	3	4	5
T	T	T	T	F

Stack

1	2	3	4	

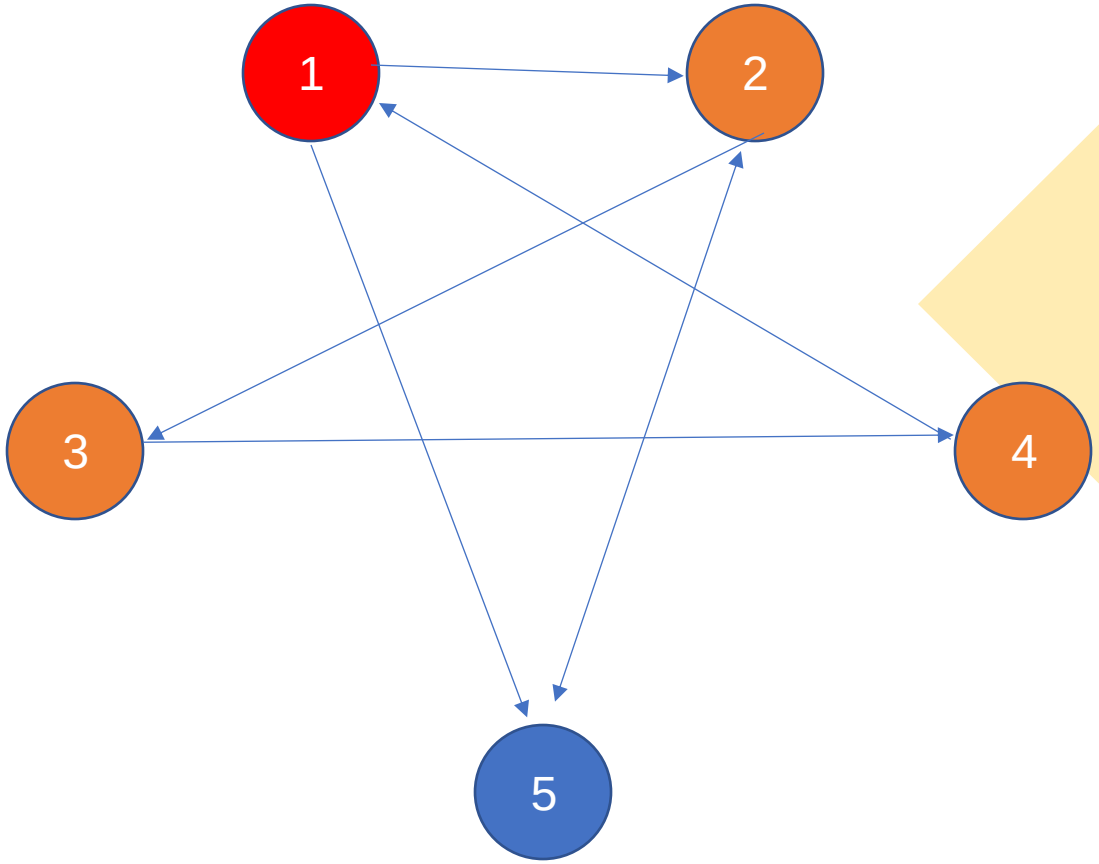


(1) đã nằm trong Stack:
return True

Detect cycle in directed 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



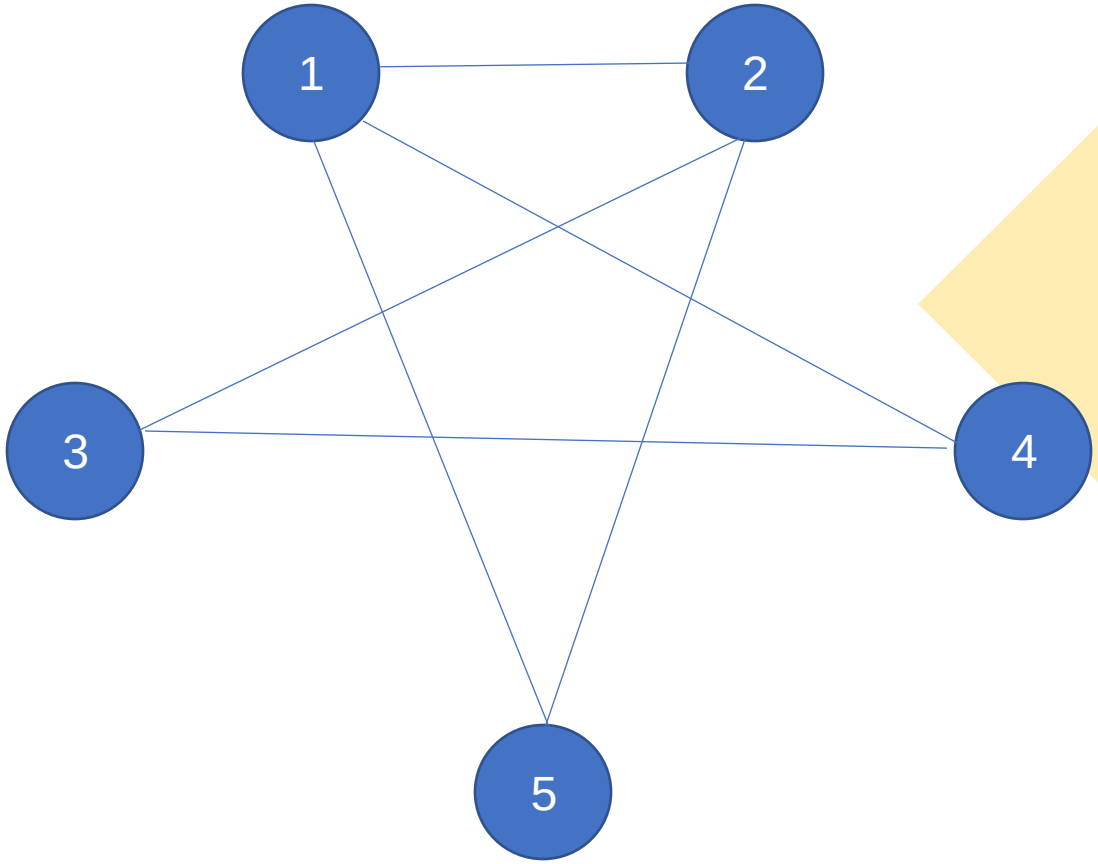
Return True

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.



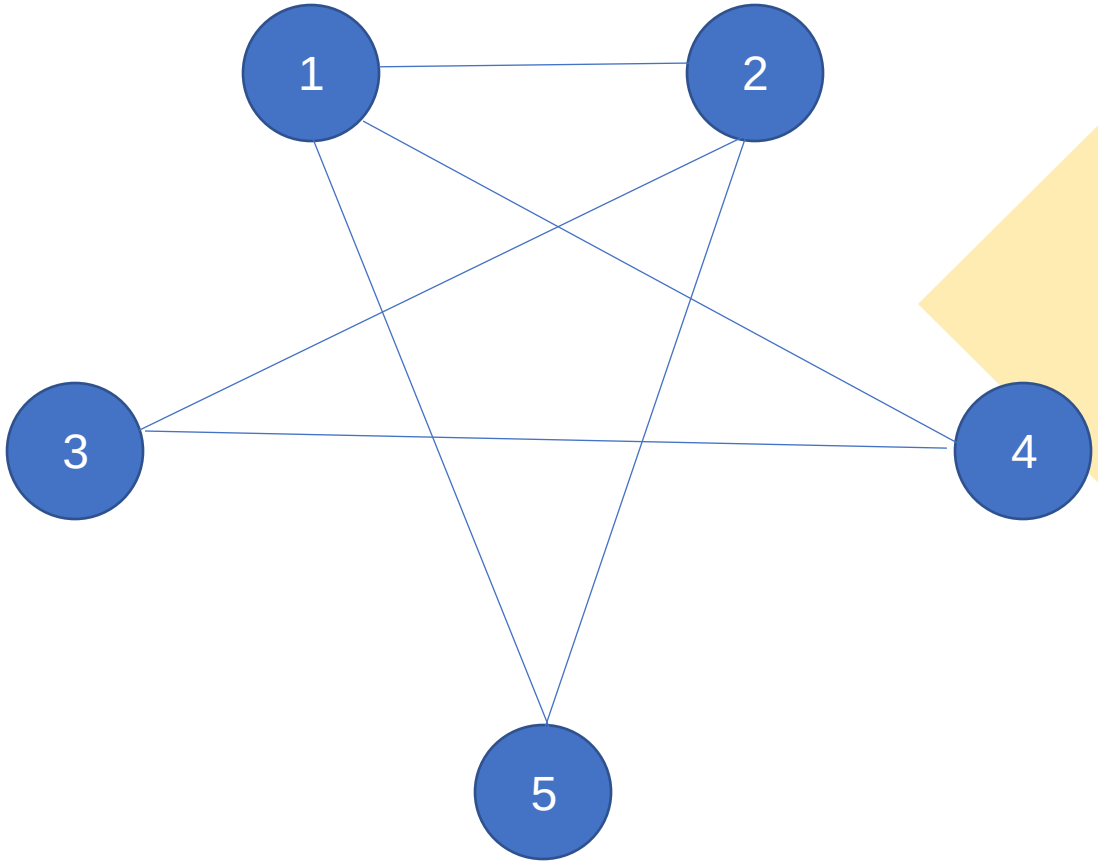
Detect cycle in undirected graph

Visited

1	2	3	4	5
F	F	F	F	F

Stack

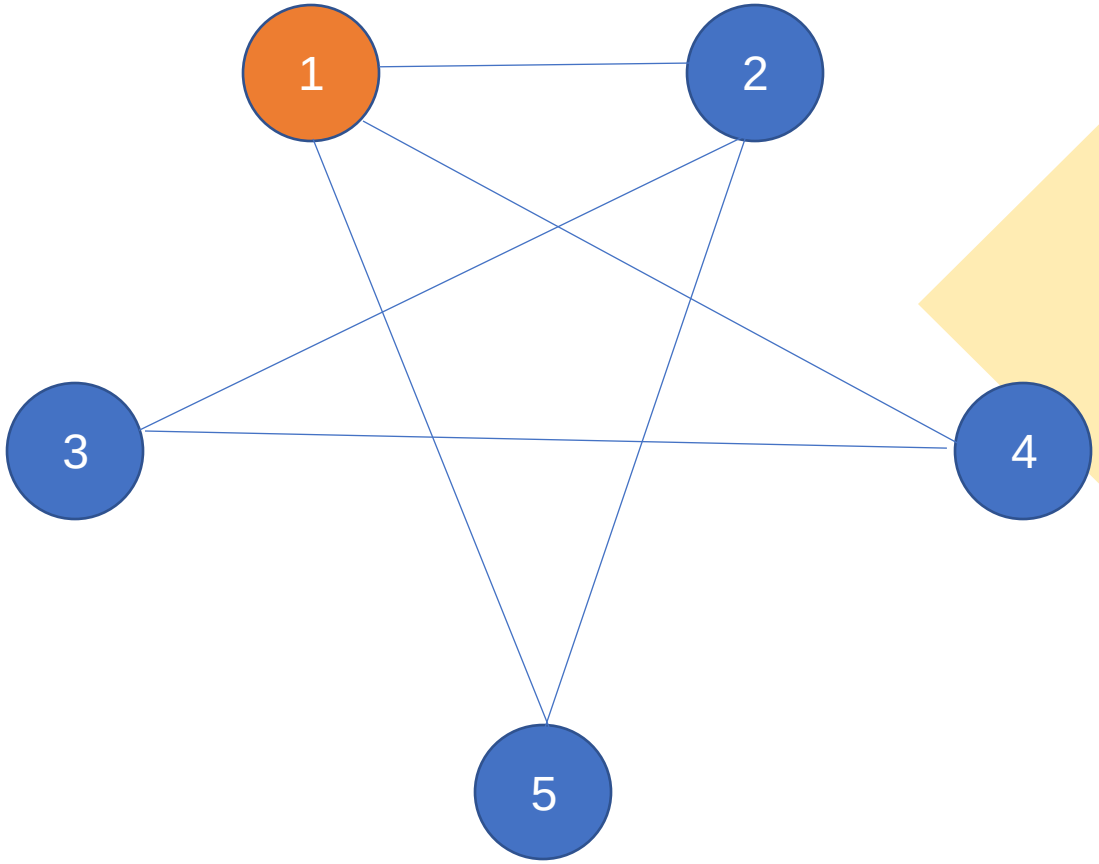
Parent:



Detect cycle in undirected graph

Visited

1	2	3	4	5
T	F	F	F	F



Stack

1				

Parent:

Detect cycle in undirected graph

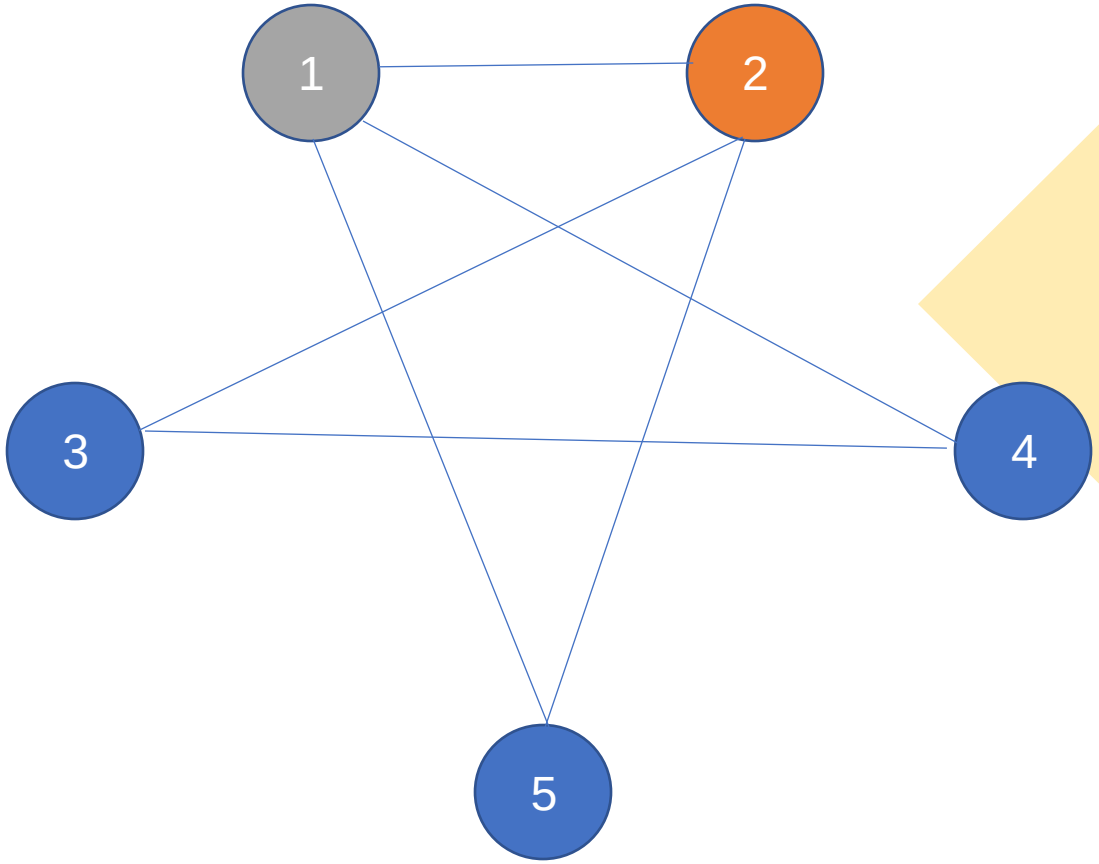
Visited

1	2	3	4	5
T	T	F	F	F

Stack

1	2			

Parent: -1



Detect cycle in undirected graph

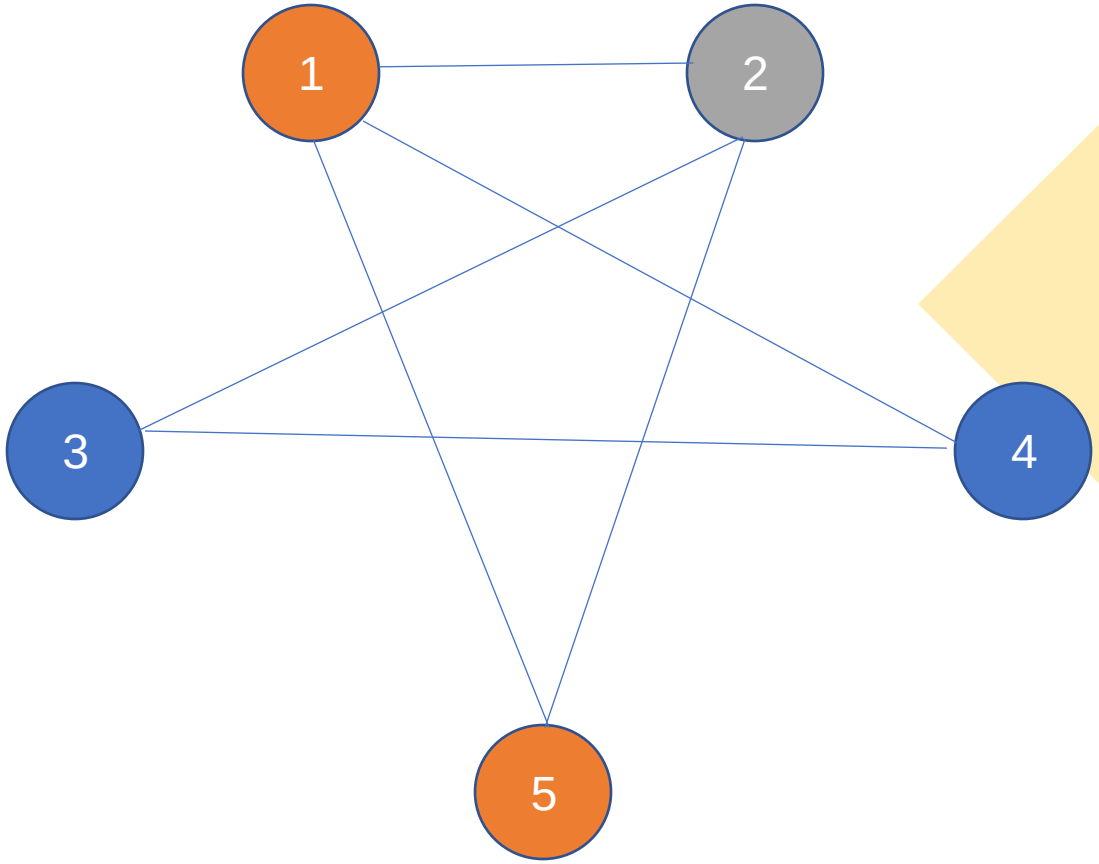
Visited

1	2	3	4	5
T	T	F	F	T

Stack

1	2	5		

Parent: 1



Detect cycle in undirected graph

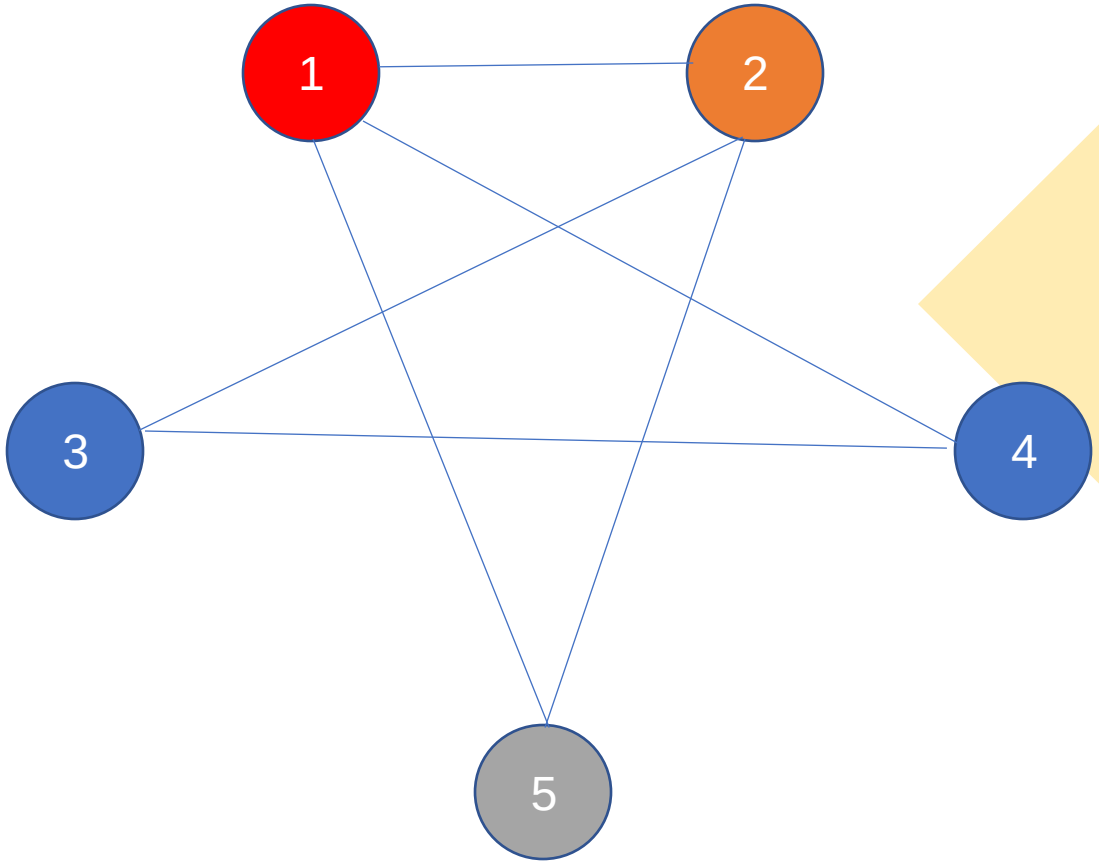
Visited

1	2	3	4	5
T	T	F	F	T

Stack

1	2	5		

Parent: 2

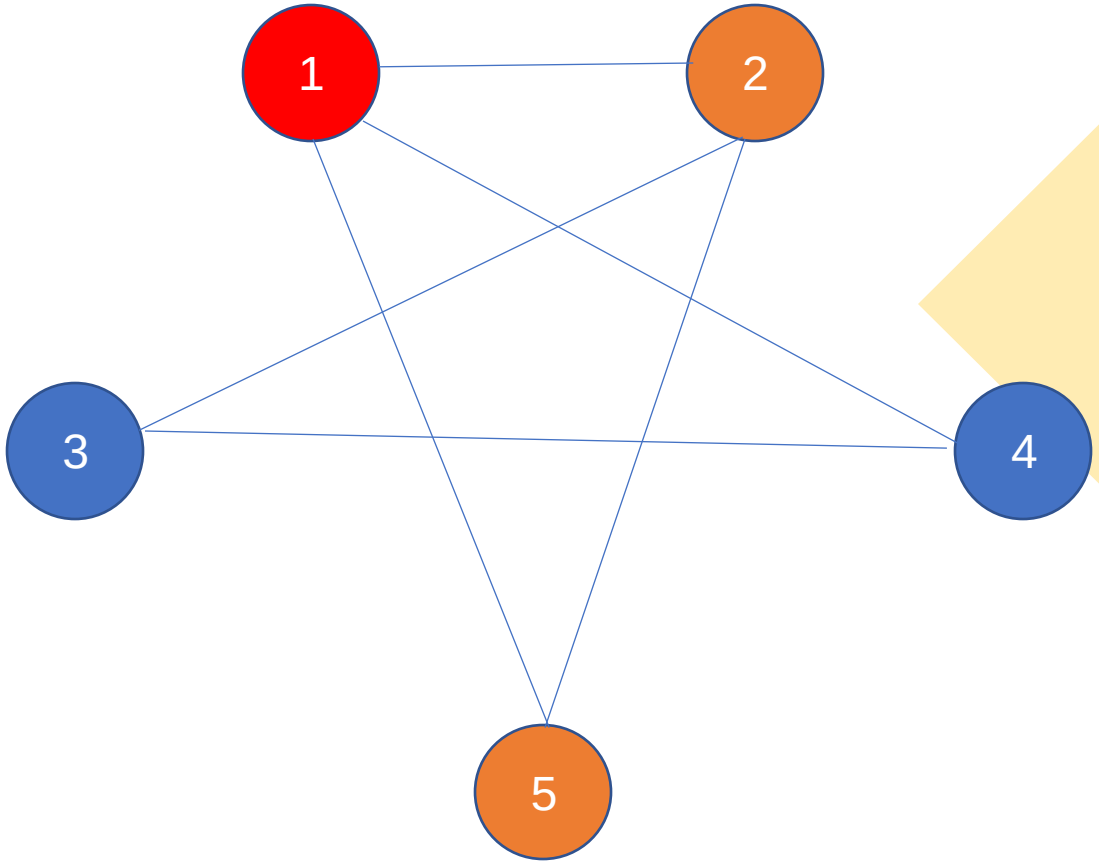


1 đã được đánh dấu là visited và không là parent của 5
=> return true

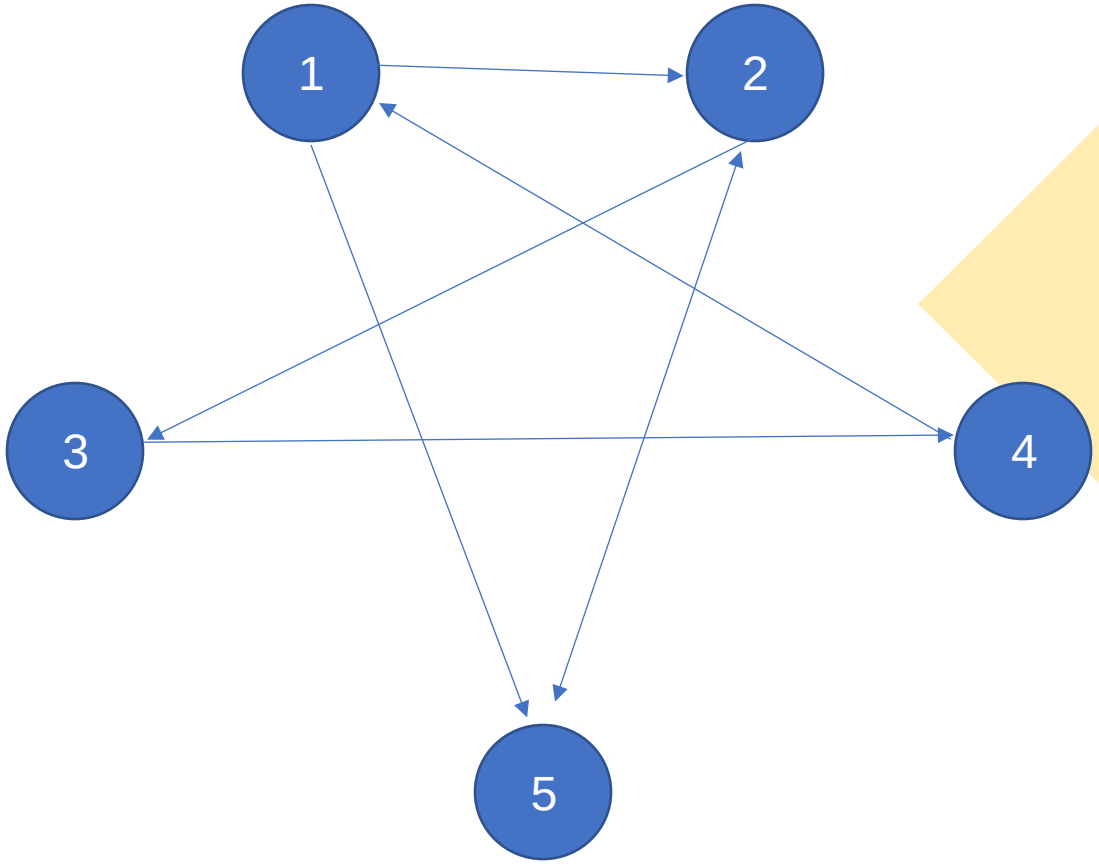
Detect cycle in undirected graph

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

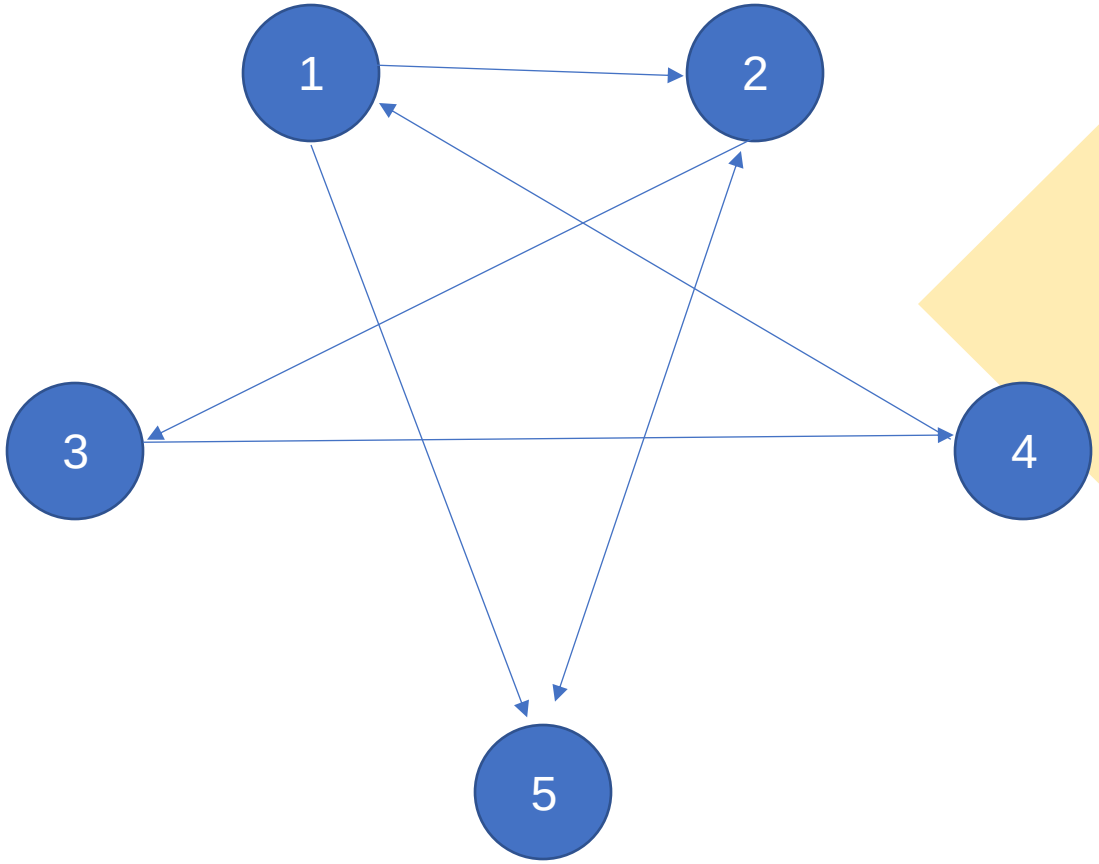


Breadth First Traversal (Search)



Breadth First Traversal

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



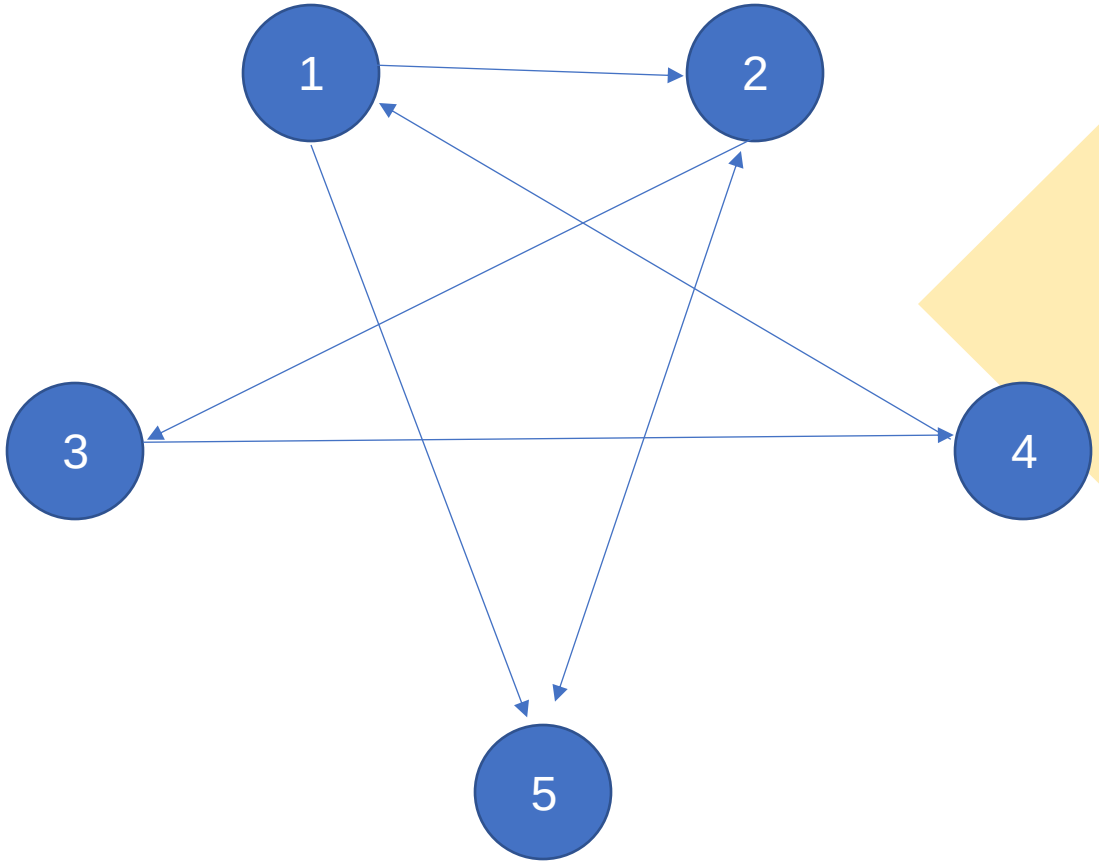
Breadth First Traversal

Visited

1	2	3	4	5
F	F	F	F	F

Queue

Print:



Breadth First Traversal

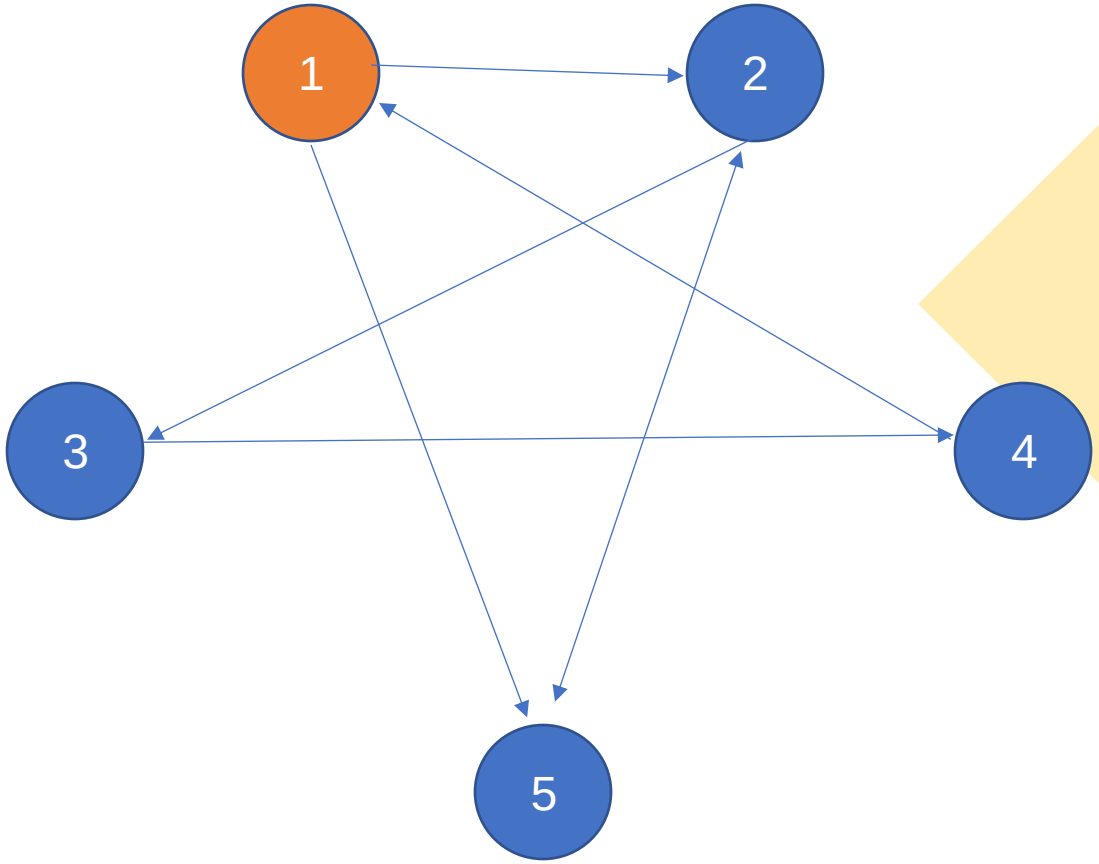
Visited

1	2	3	4	5
T	F	F	F	F

Queue

1				

Print:



Breadth First Traversal

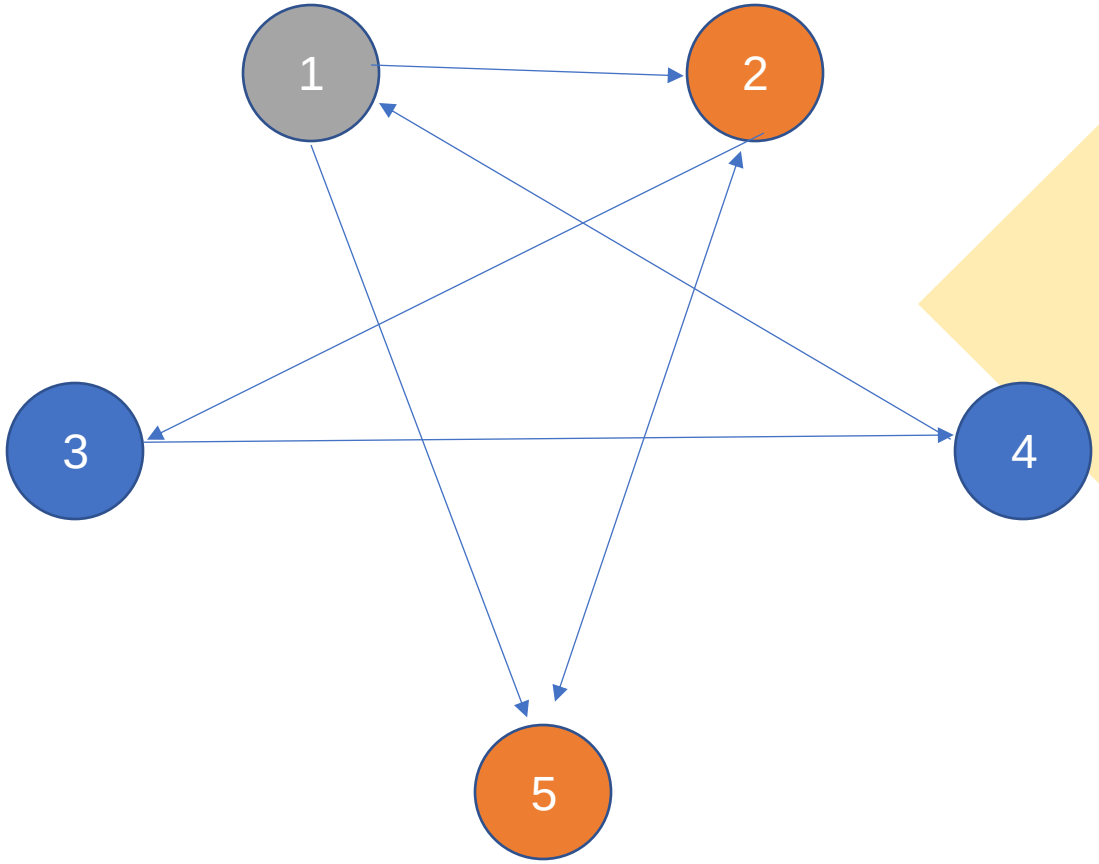
Visited

1	2	3	4	5
T	T	F	F	T

Queue

2	5			

Print: **1**



Breadth First Traversal

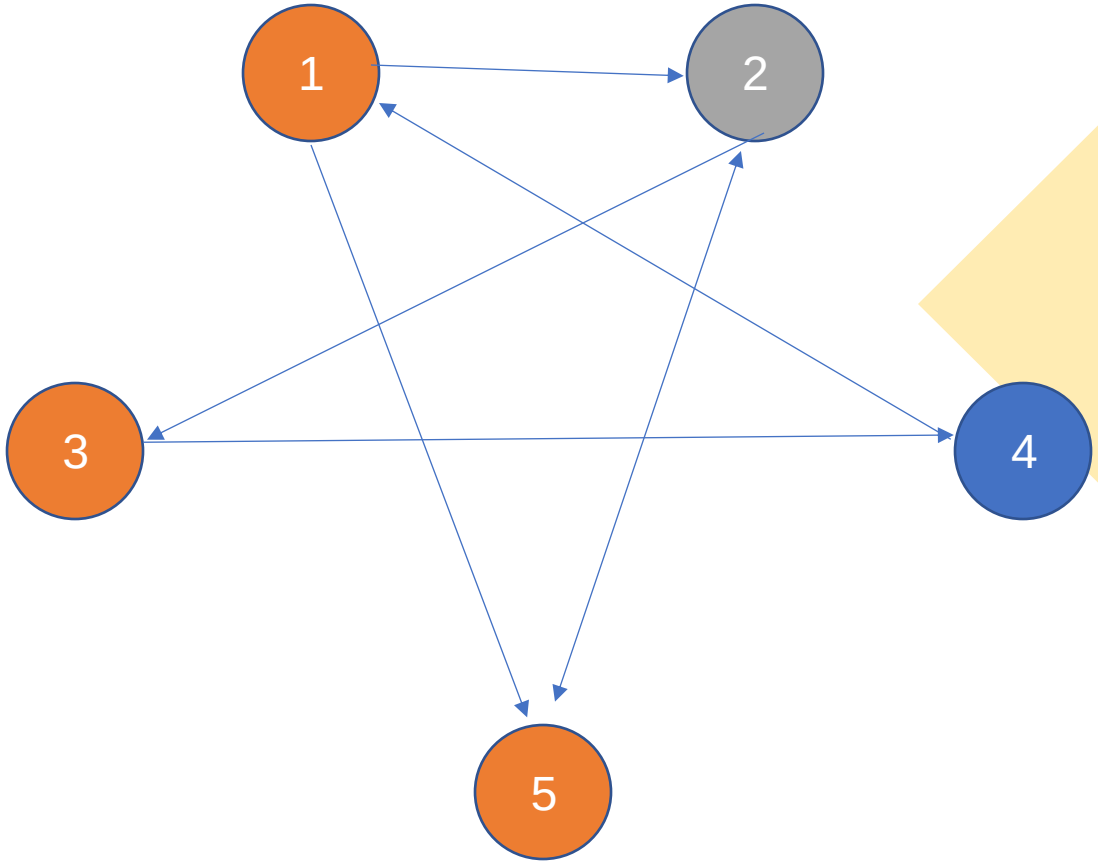
Visited

1	2	3	4	5
T	T	T	F	T

Queue

5	3			

Print: 1 **2**



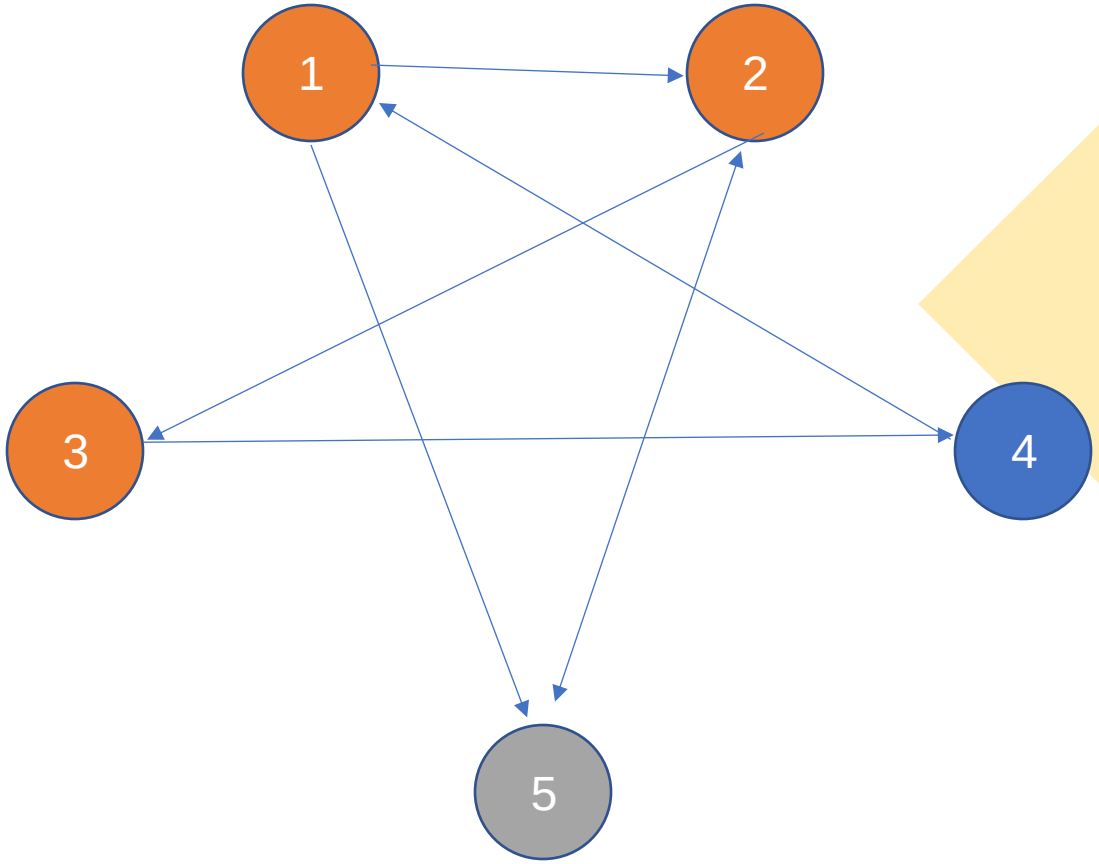
Breadth First Traversal

Visited

1	2	3	4	5
T	T	T	F	T

Queue

3				



Print: 1 2 **5**

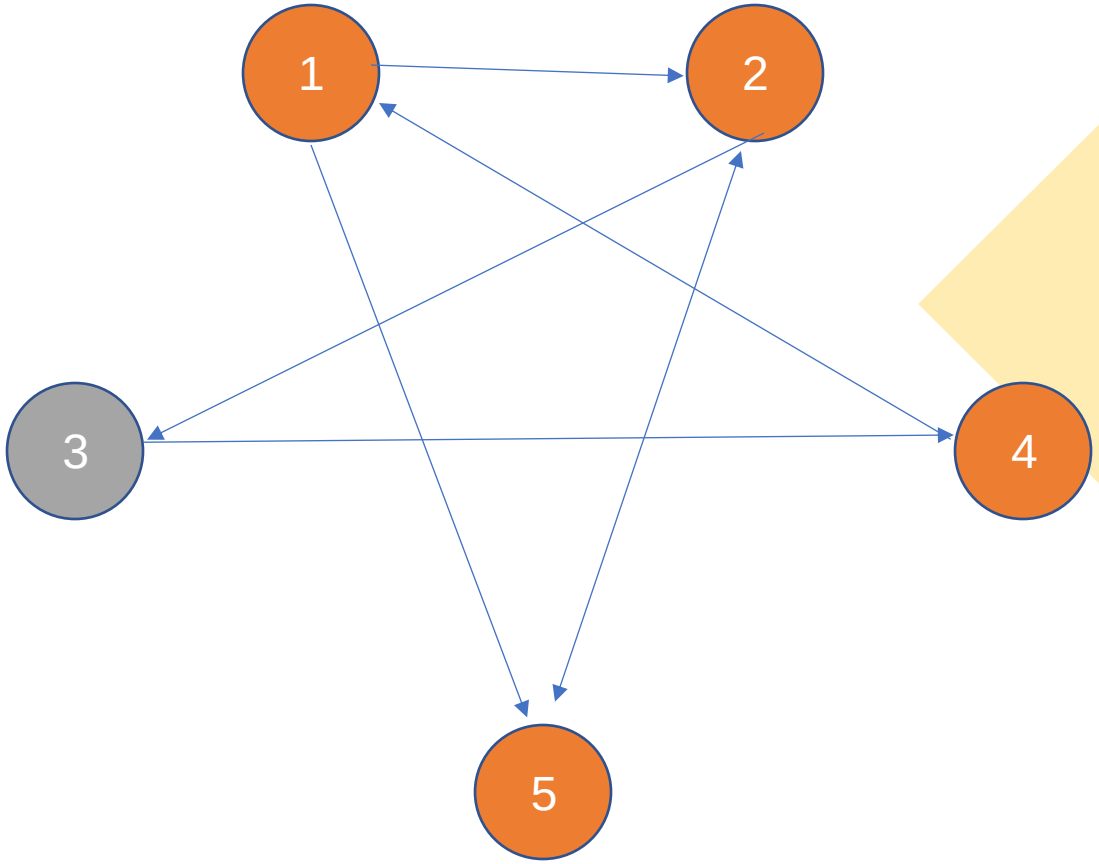
Breadth First Traversal

Visited

1	2	3	4	5
T	T	T	T	T

Queue

4				



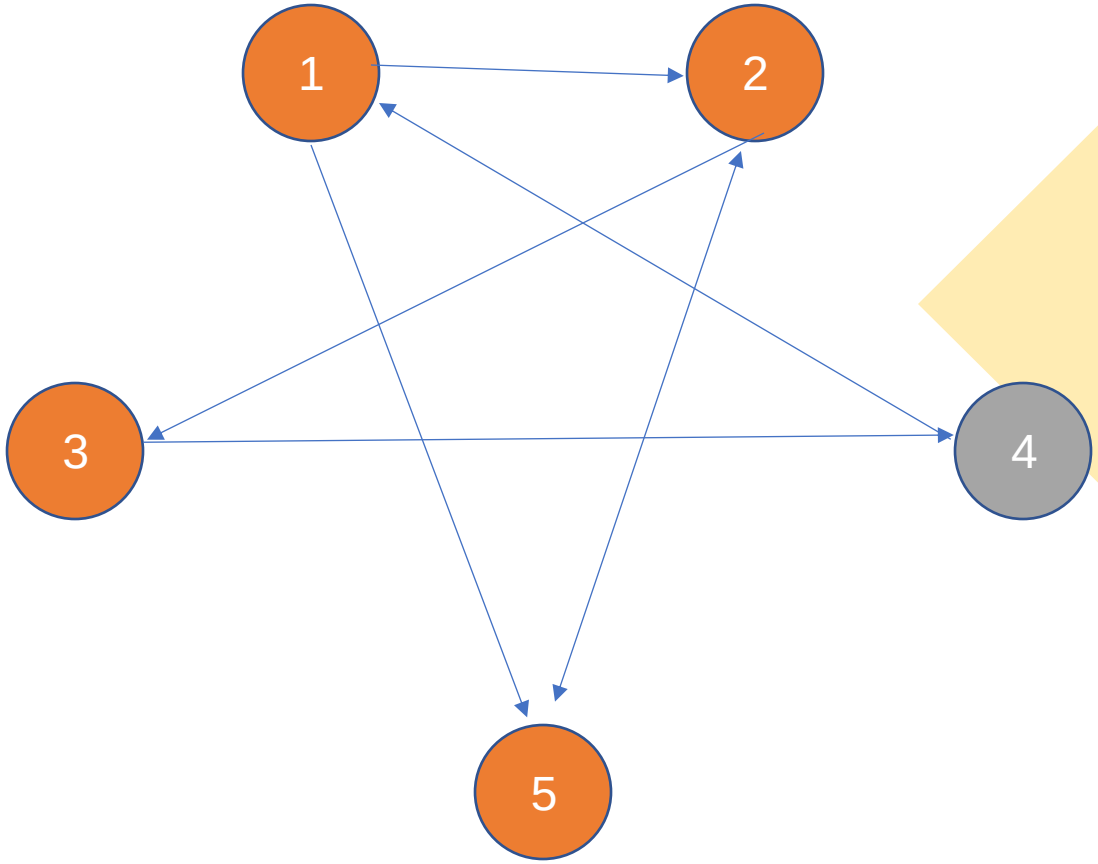
Print: 1 2 5 3 4

Breadth First Traversal

Visited

1	2	3	4	5
T	T	T	T	T

Queue



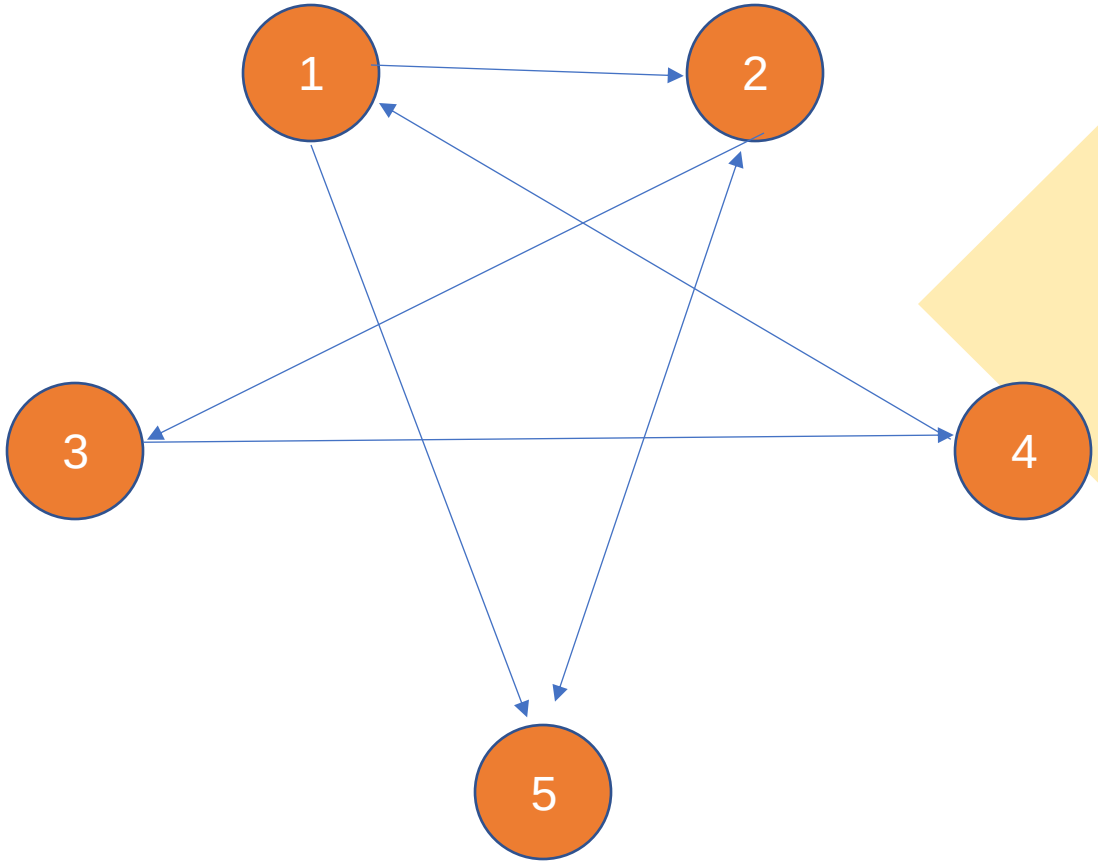
Print: 1 2 5 3 **4**

Breadth First Traversal

Visited

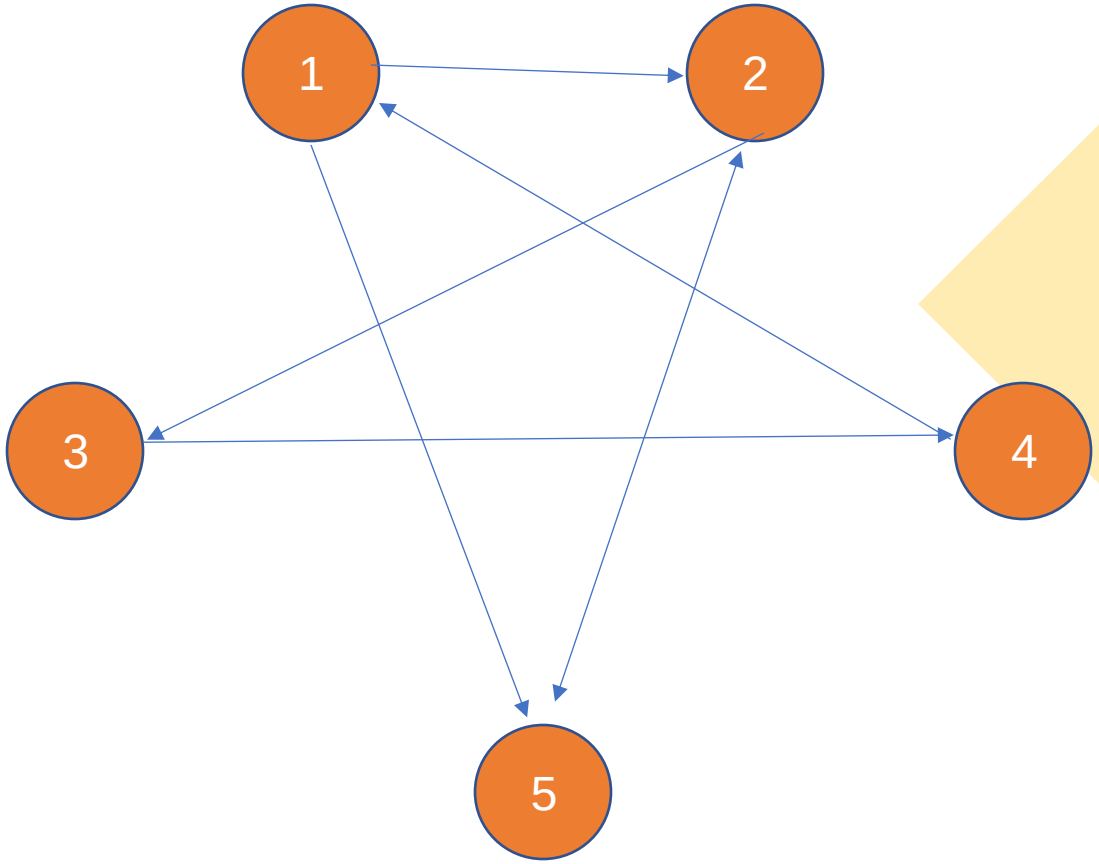
1	2	3	4	5
T	T	T	T	T

Queue



Print: 1 2 5 3 4

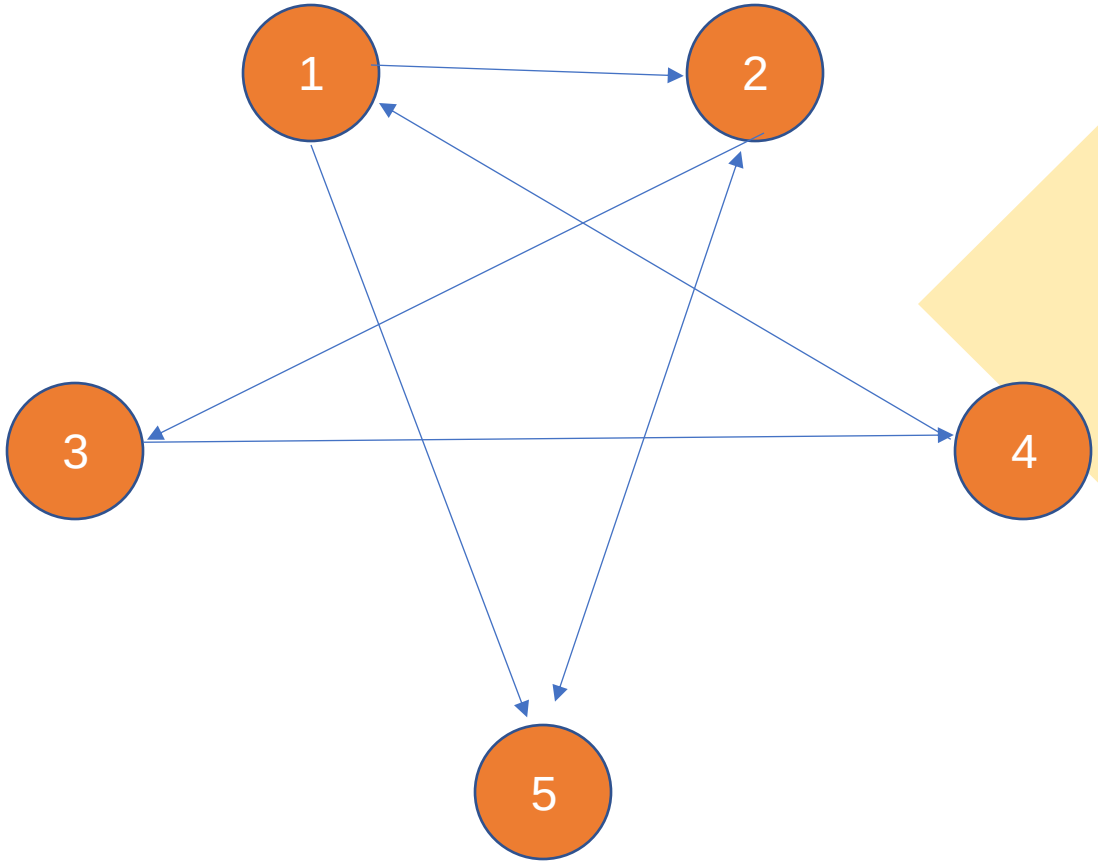
Breadth First Traversal



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

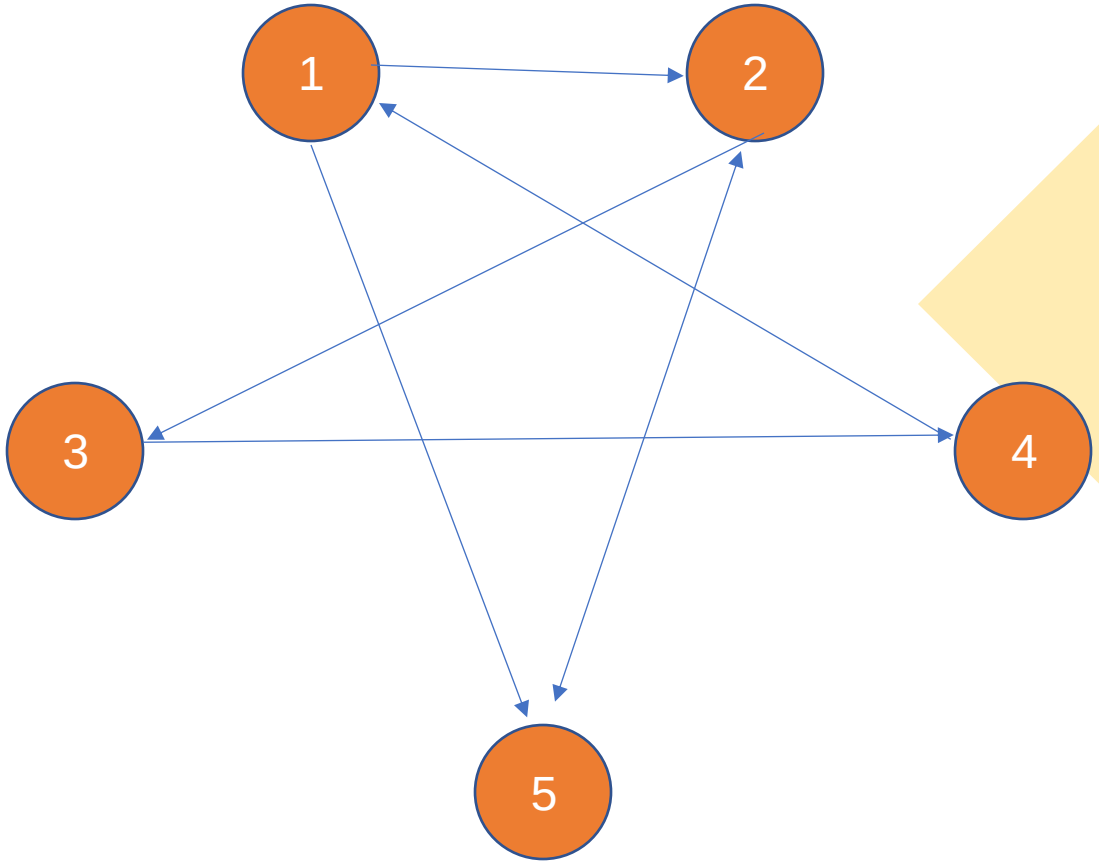


Ứng dụng:

- ✓ 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



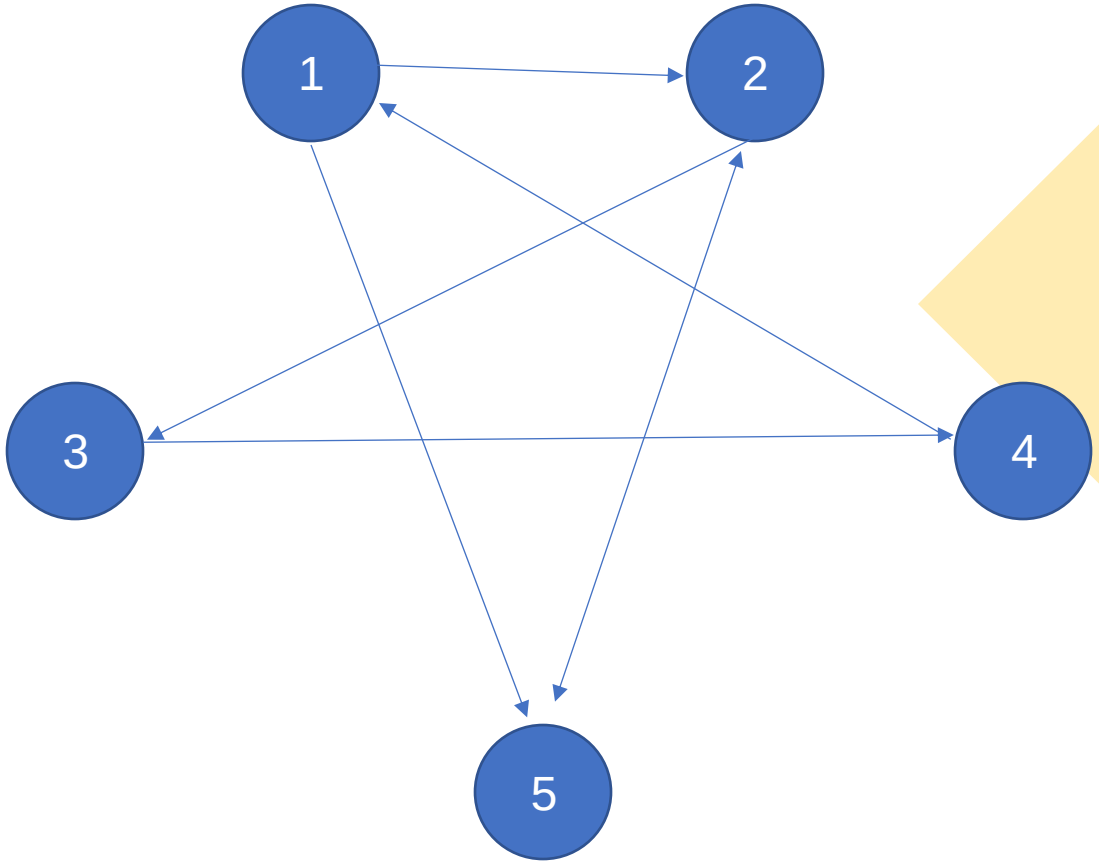
Các thuật toán:

- **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 đỉnh nào thì chốt luôn đỉnh 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

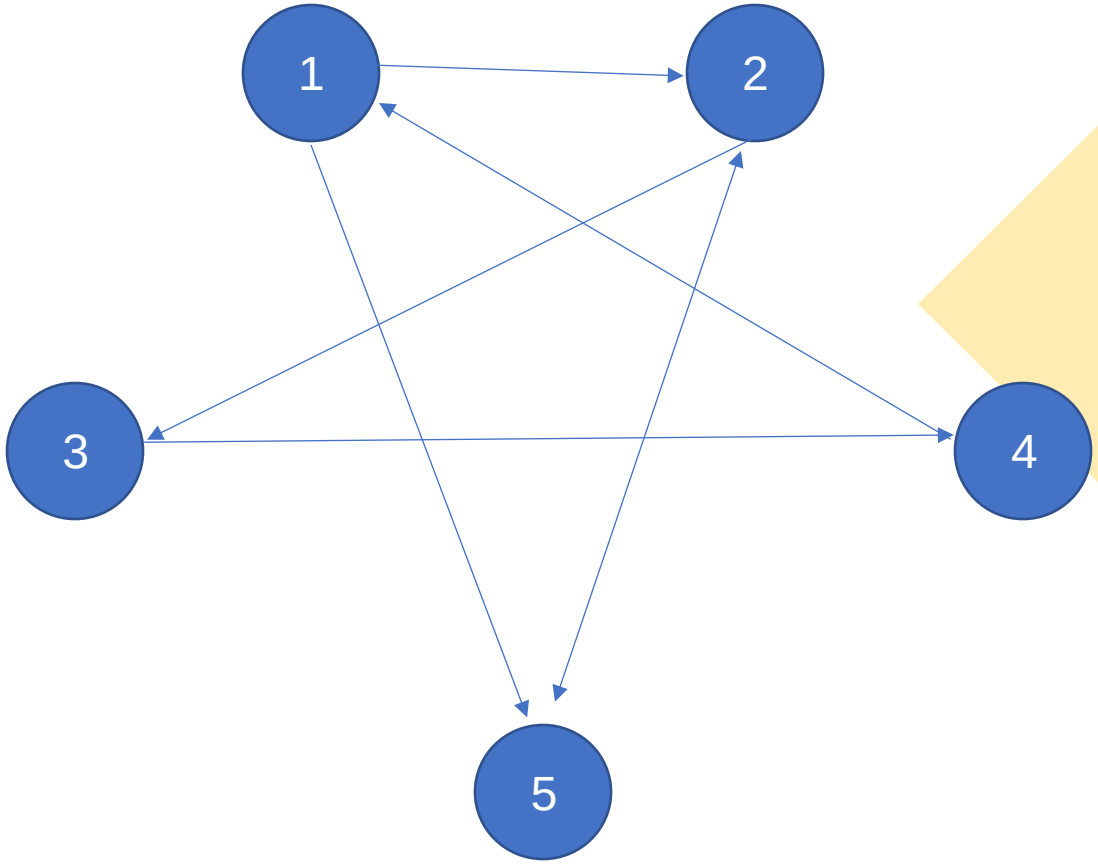
Parent

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

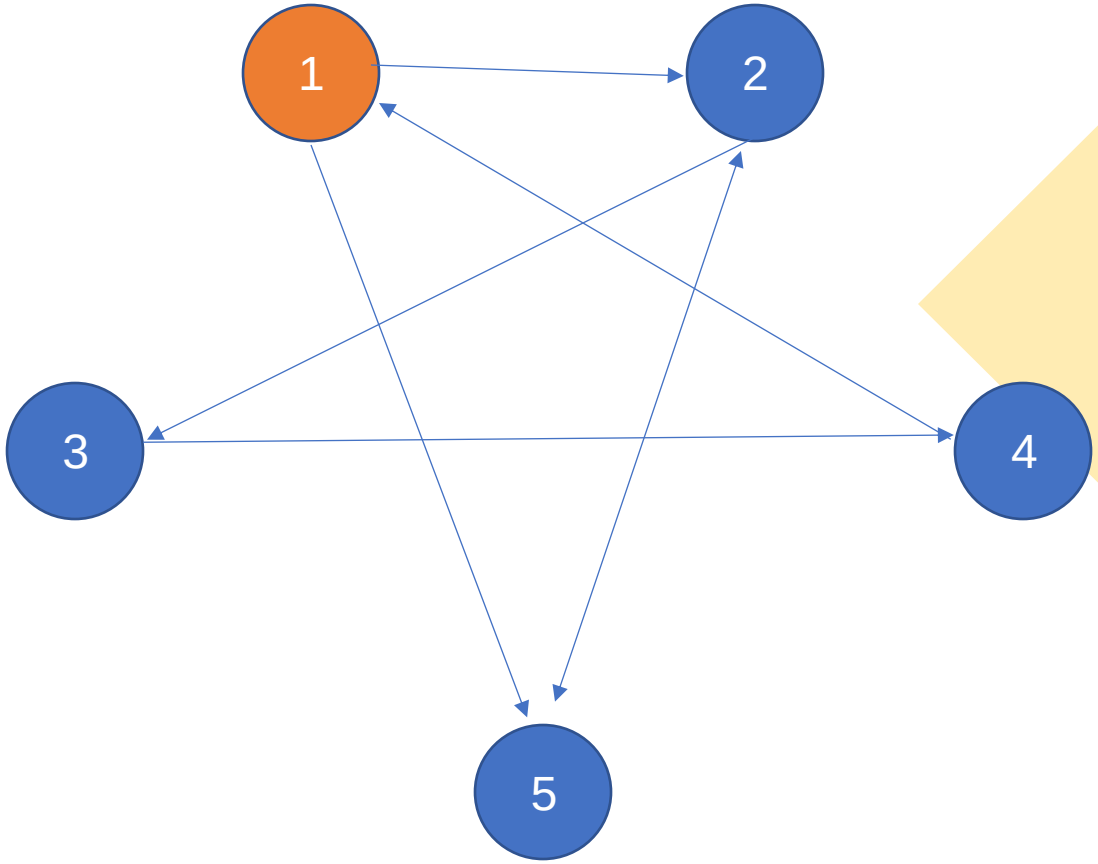
1	2	3	4	5
-1	-1	-1	-1	-1

Queue

1				

Distance

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



BFS

Visited

1	2	3	4	5
T	T	F	F	5

Parent

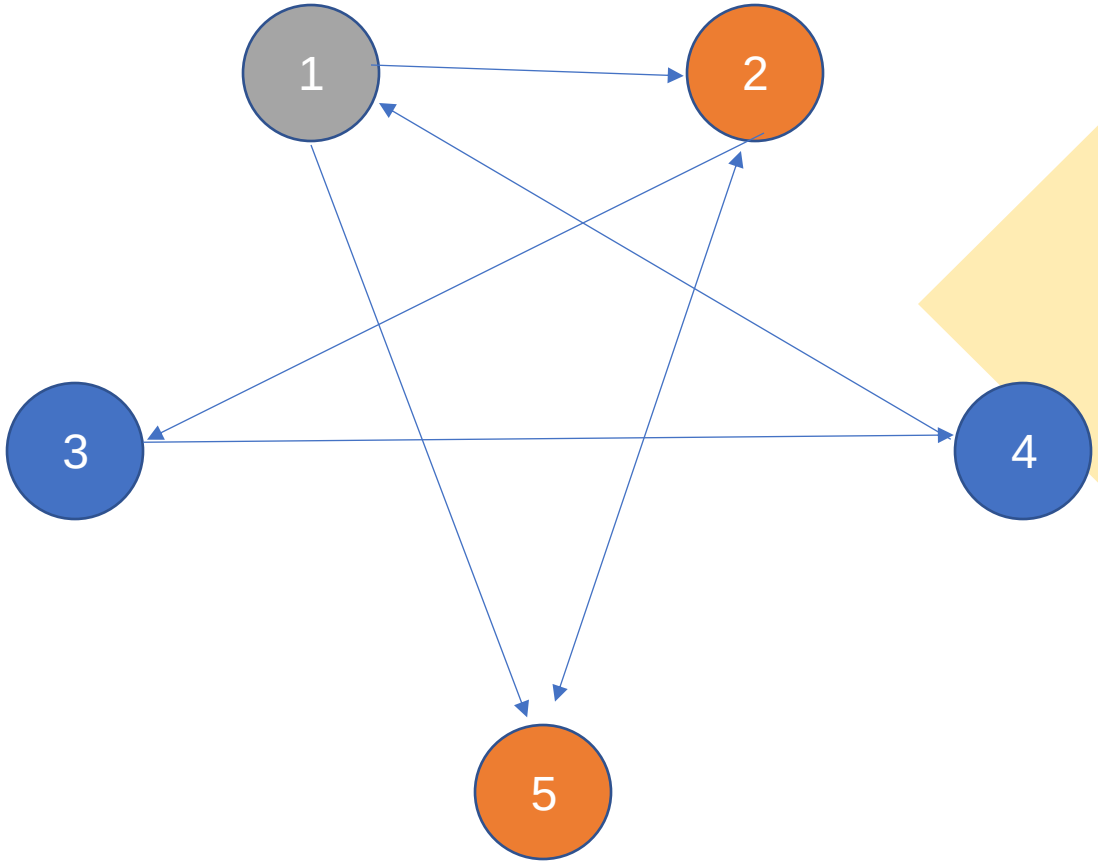
1	2	3	4	5
-1	1	-1	-1	1

Queue

2	5			

Distance

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



BFS

Visited

1	2	3	4	5
T	T	T	F	T

Parent

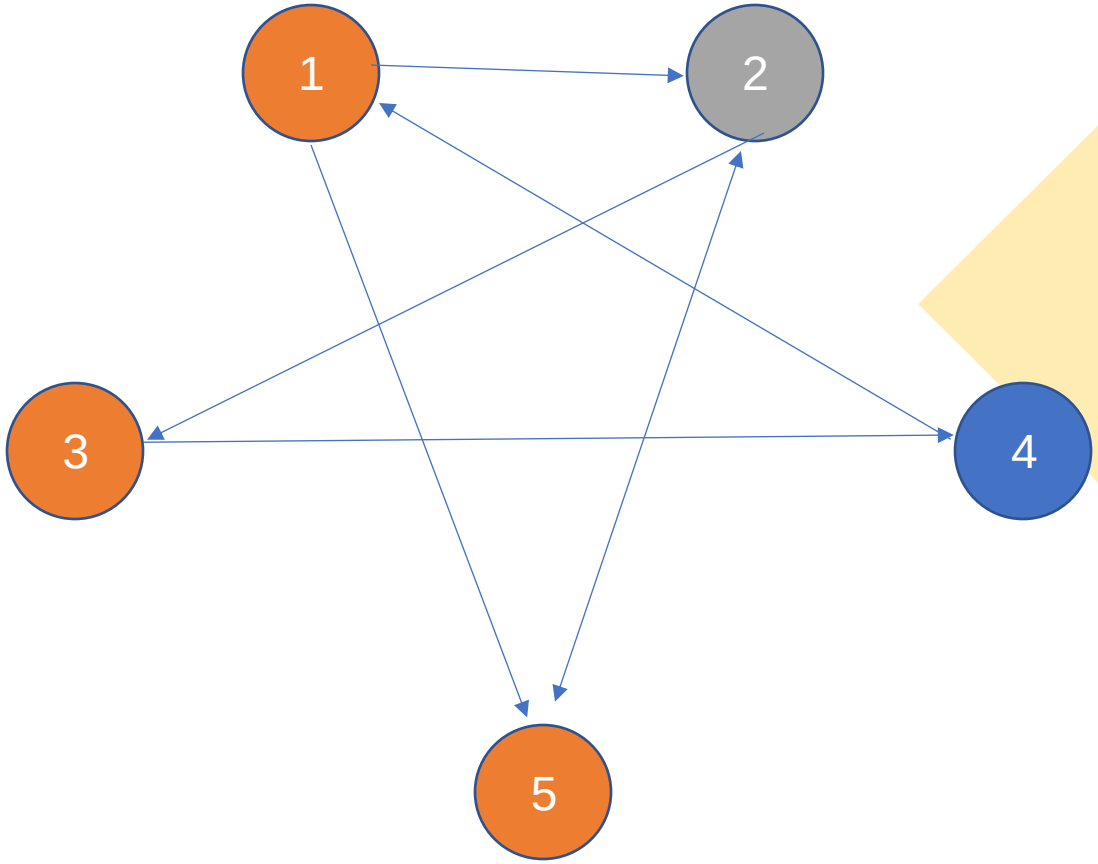
1	2	3	4	5
-1	1	2	-1	1

Queue

5	3			

Distance

1	2	3	4	5
0	1	2	-1	1



BFS

Visited

1	2	3	4	5
T	T	T	F	T

Parent

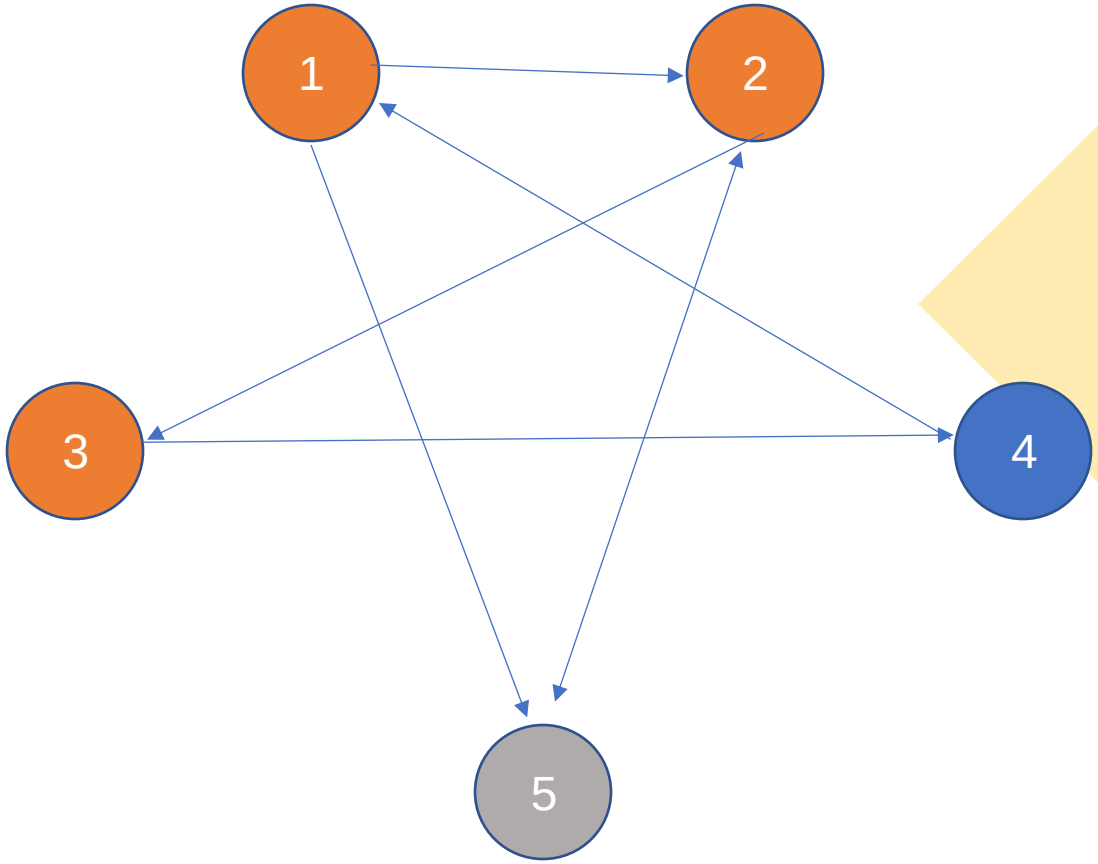
1	2	3	4	5
-1	1	2	-1	1

Queue

3				

Distance

1	2	3	4	5
0	1	2	-1	1



BFS

Visited

1	2	3	4	5
T	T	T	T	T

Parent

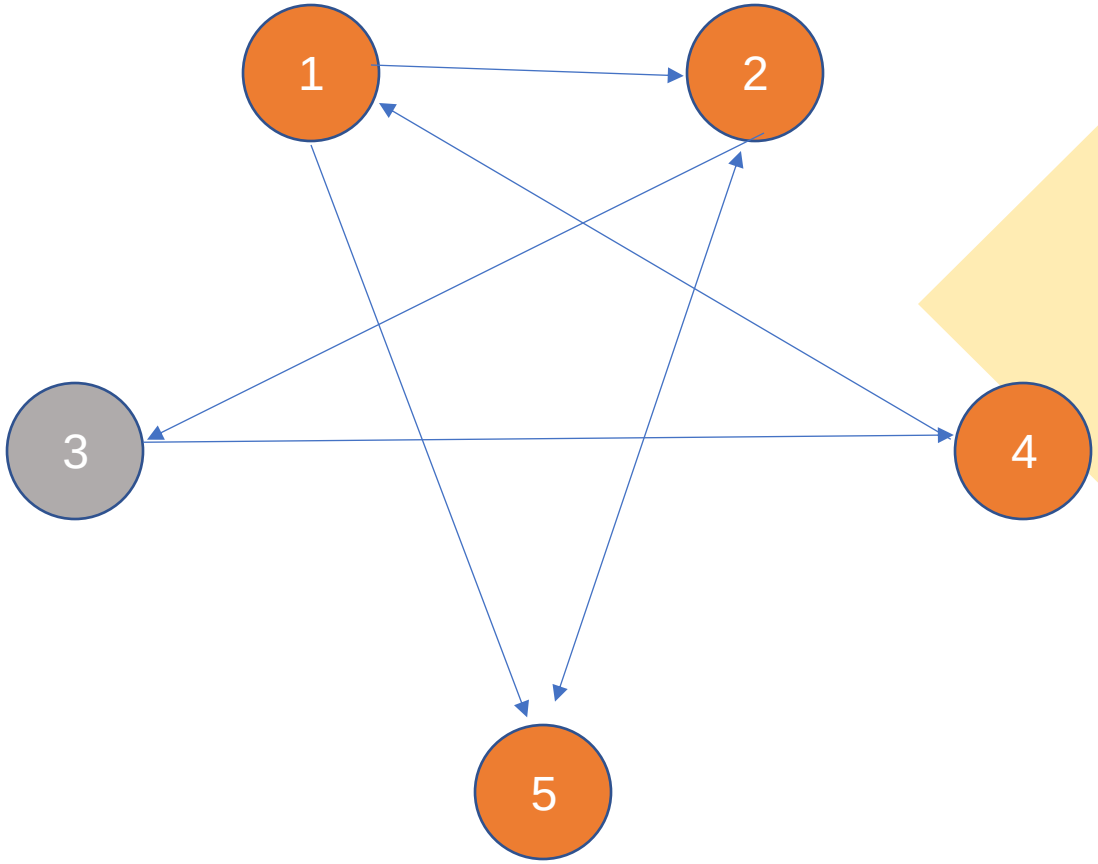
1	2	3	4	5
-1	1	2	3	1

Queue

4				

Distance

1	2	3	4	5
0	1	2	3	1



BFS

Visited

1	2	3	4	5
T	T	T	T	T

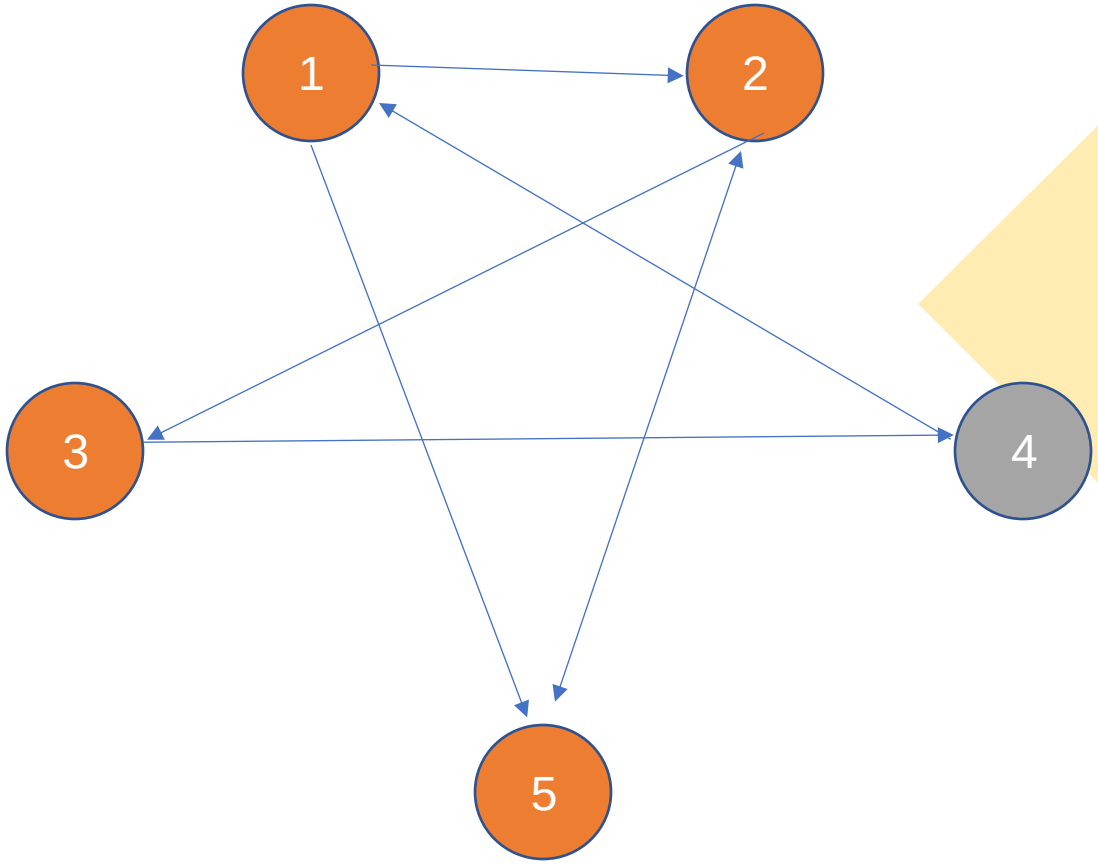
Parent

1	2	3	4	5
-1	1	2	3	1

Queue

Distance

1	2	3	4	5
0	1	2	3	1



BFS

Visited

1	2	3	4	5
T	T	T	T	T

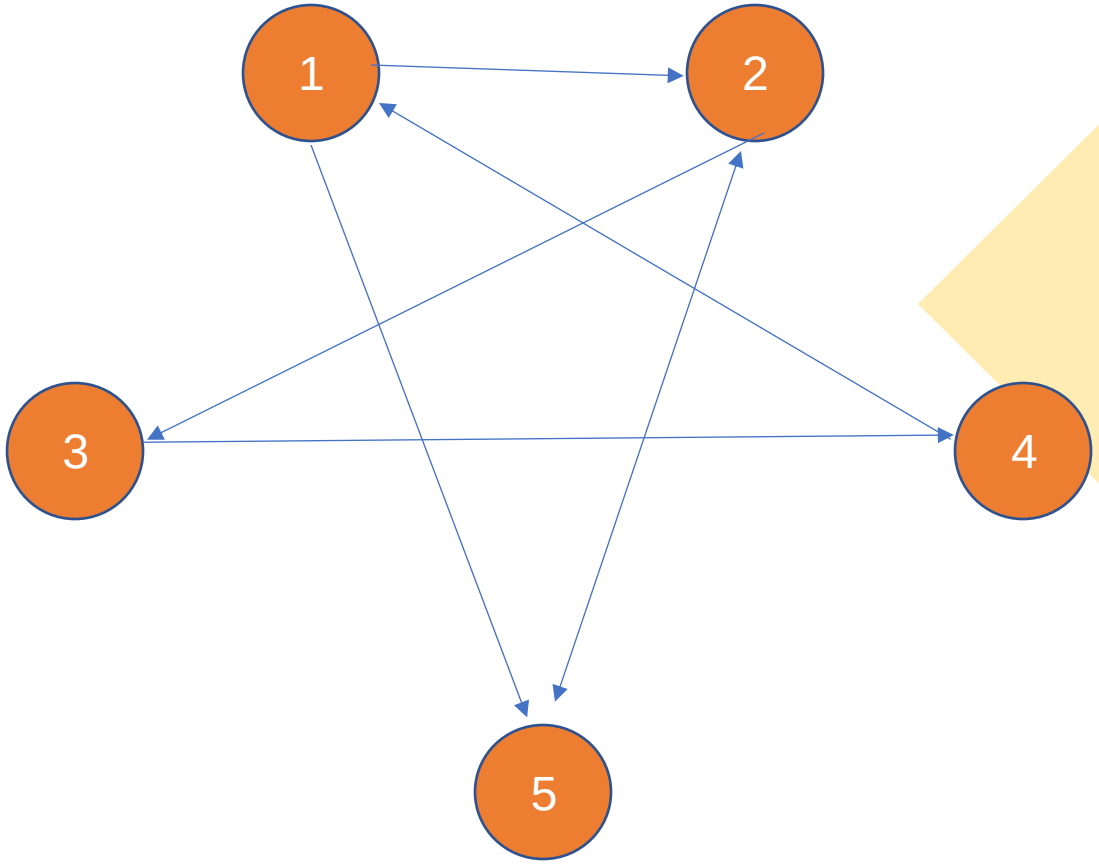
Parent

1	2	3	4	5
-1	1	1	3	2

Queue

Distance

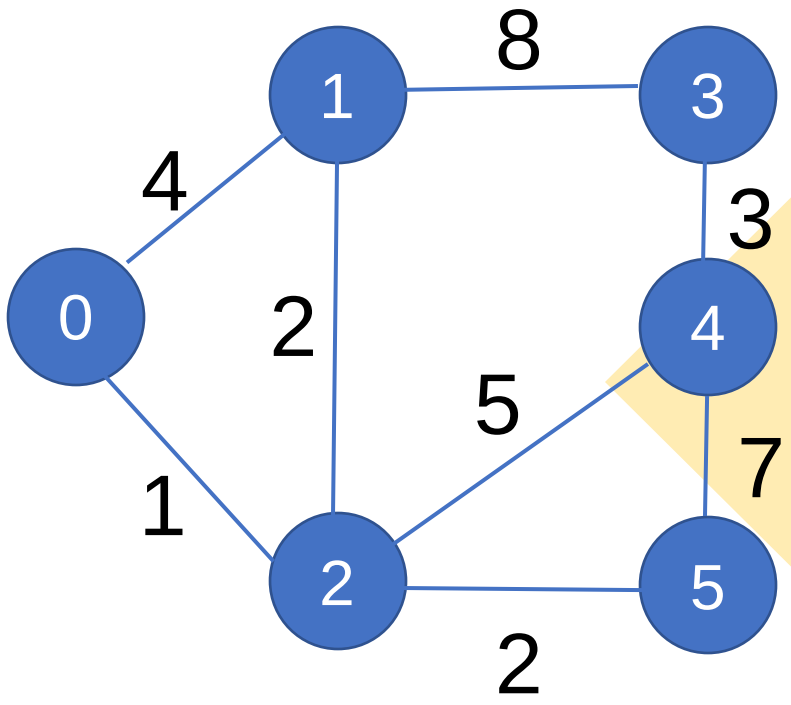
1	2	3	4	5
0	1	1	2	2



Dijkstra

Vấn đề:

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



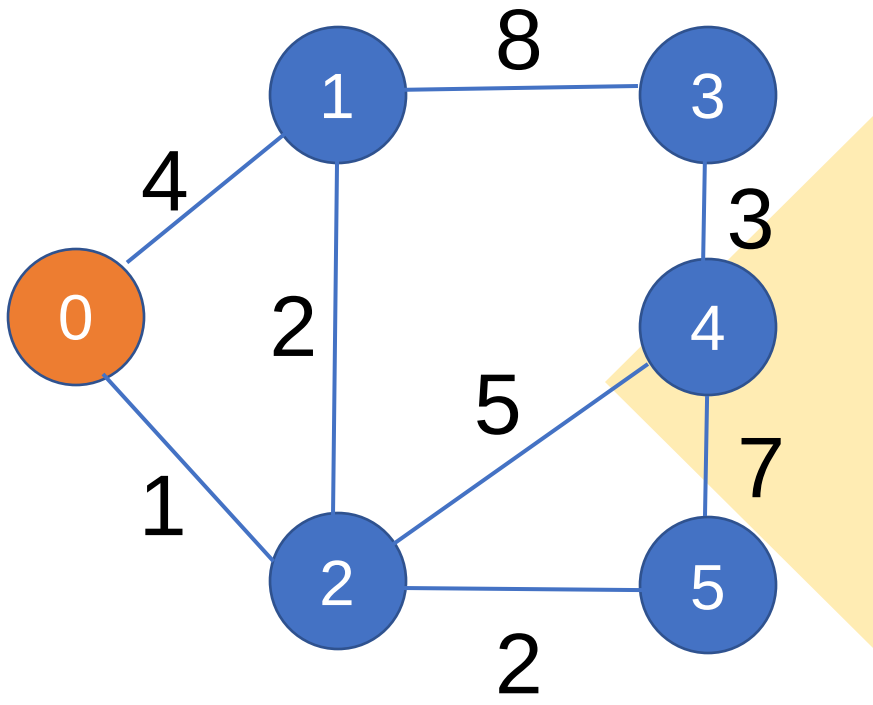
Giải quyết:

- Cập nhật distance
- Chốt (node chưa chốt) 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

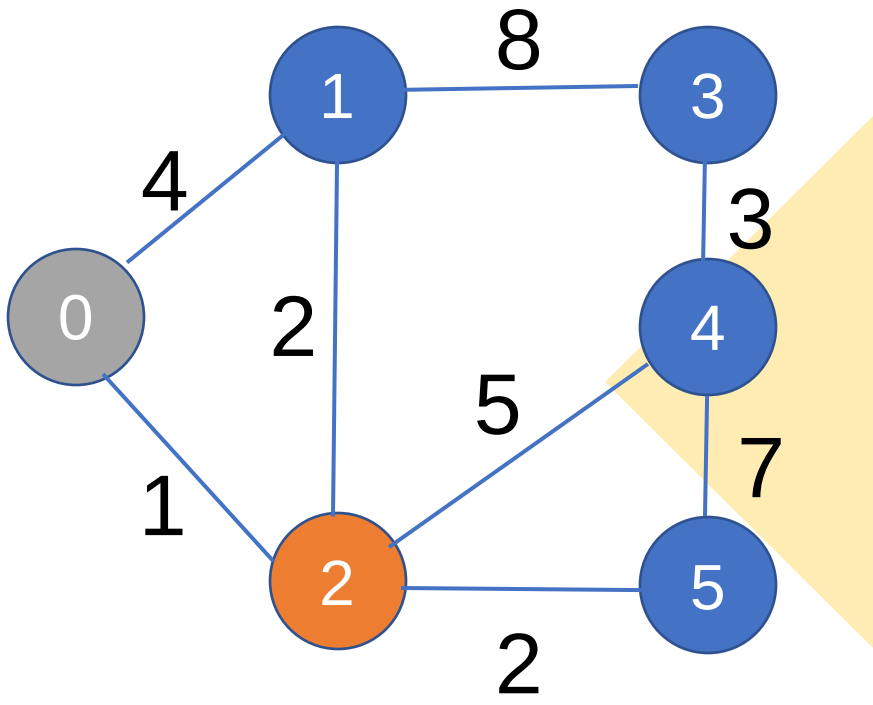


Confirmed

0					

Dijkstra

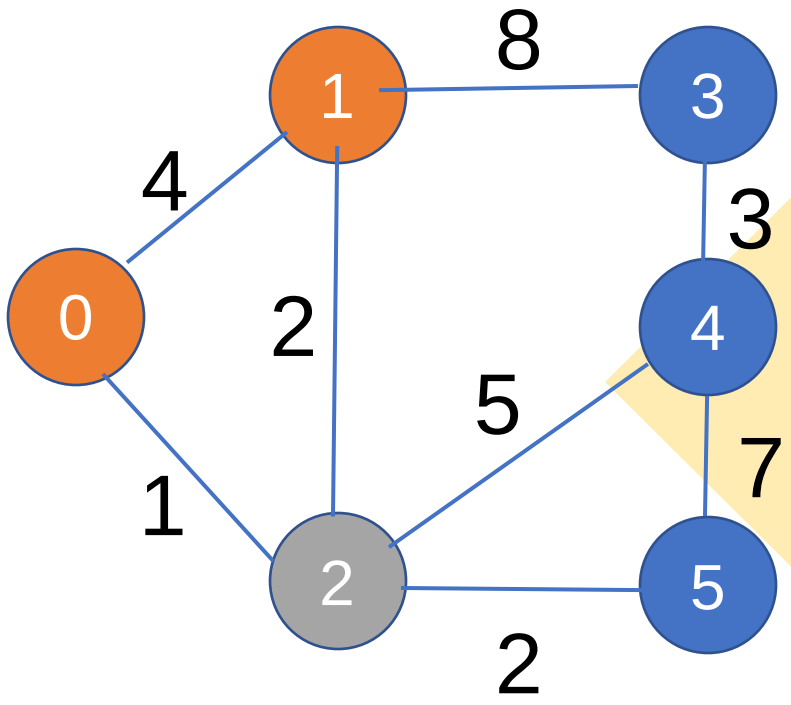
	0	1	2	3	4	5
	0	inf	inf	inf	inf	inf
(0): 0	-	4	1	inf	inf	inf
(2): 1						



Confirmed	0	2				

Dijkstra

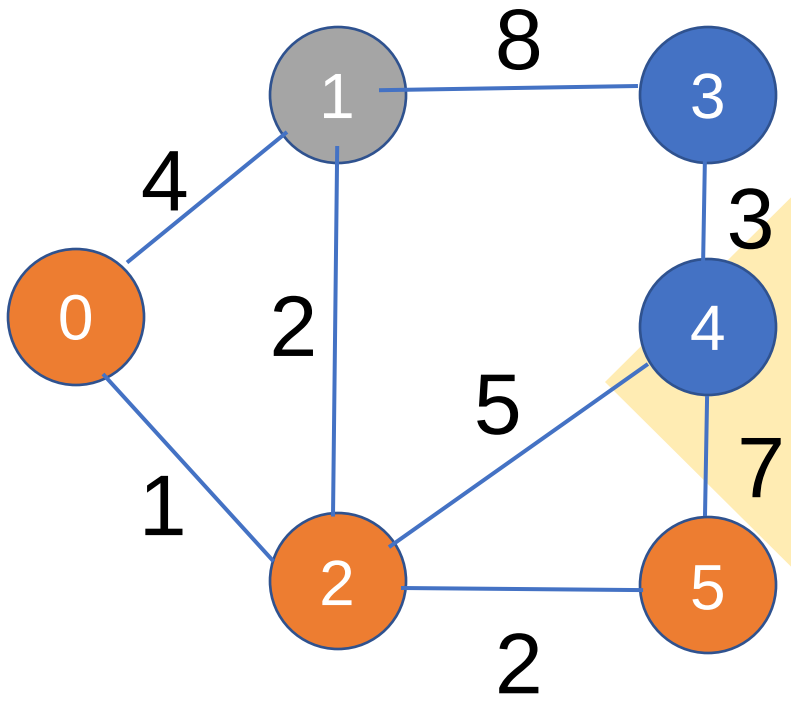
	0	1	2	3	4	5
	0	inf	inf	inf	inf	inf
(0): 0	-	4	1	inf	inf	inf
(2): 1	-	3	-	inf	6	3
(1): 3						



Confirmed	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						

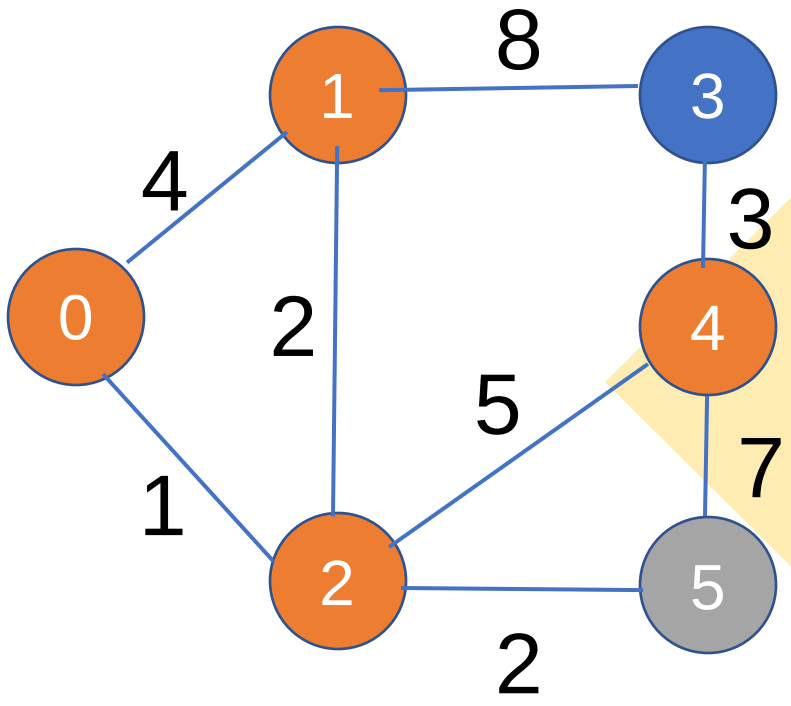


Confirmed

0	2	1	5		

Dijkstra

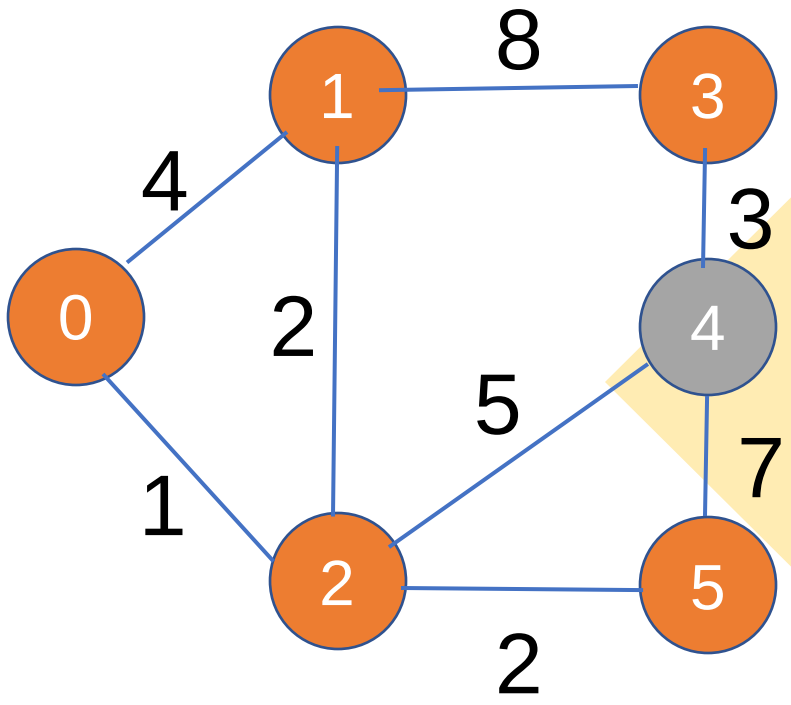
	0	1	2	3	4	5
	0	inf	inf	inf	inf	inf
(0): 0	-	4	1	inf	inf	inf
(2): 1	-	3	-	inf	6	3
(1): 3	-	-	-	12	6	3
(5): 3	-	-	-	12	6	-
(4): 6						



Confirmed	0	2	1	5		

Dijkstra

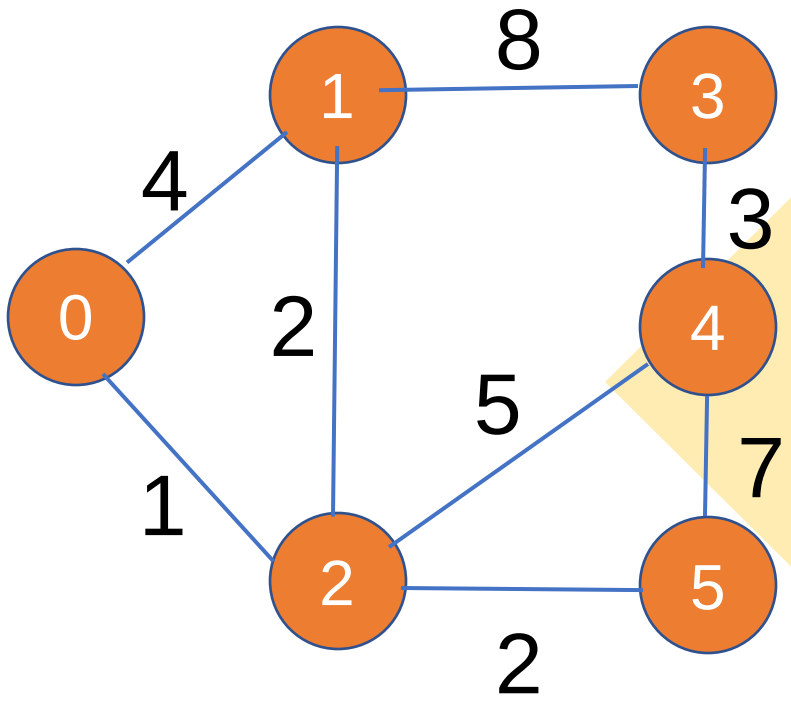
	0	1	2	3	4	5
	0	inf	inf	inf	inf	inf
(0): 0	-	4	1	inf	inf	inf
(2): 1	-	3	-	inf	6	3
(1): 3	-	-	-	12	6	3
(5): 3	-	-	-	12	6	-
(4): 6	-	-	-	9	-	-
(3): 9						



Confirmed	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	-	-	-	-	-	-



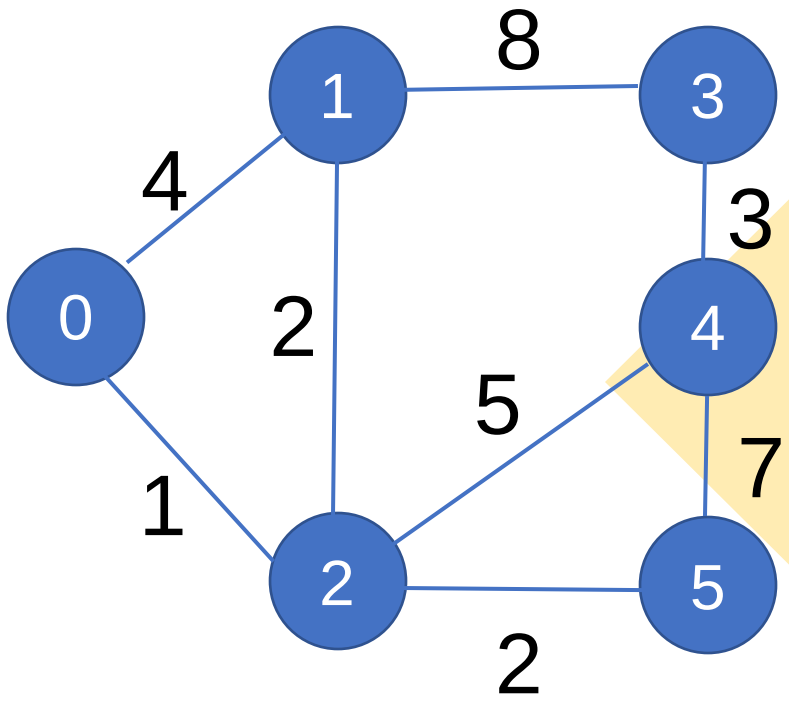
Confirmed	0	2	1	5	4	3

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 với độ phức tạp là O(E*logV)

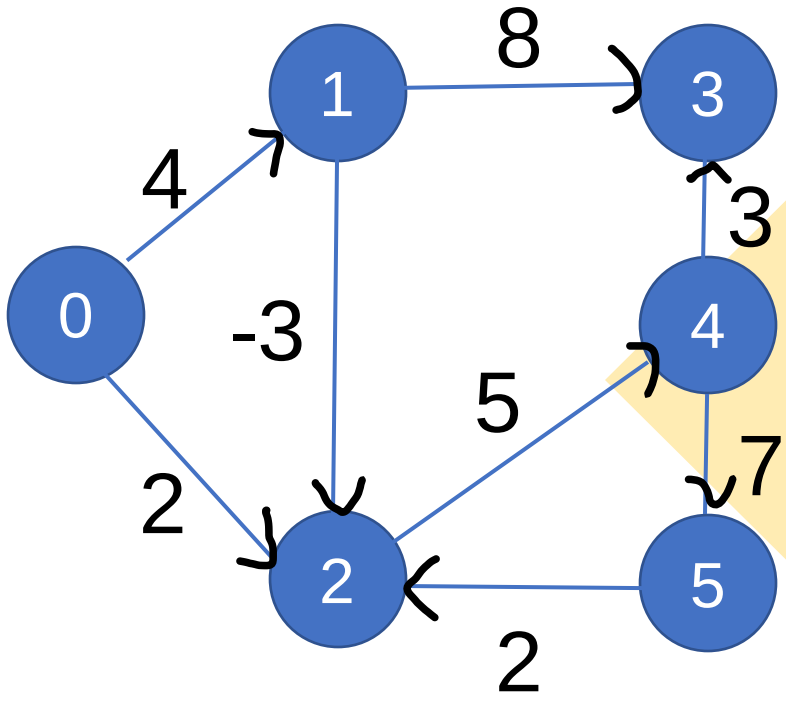


Tham lam

Bellman-Ford

Vấn đề:

- Min của distance tức thời chưa chắc là shortest

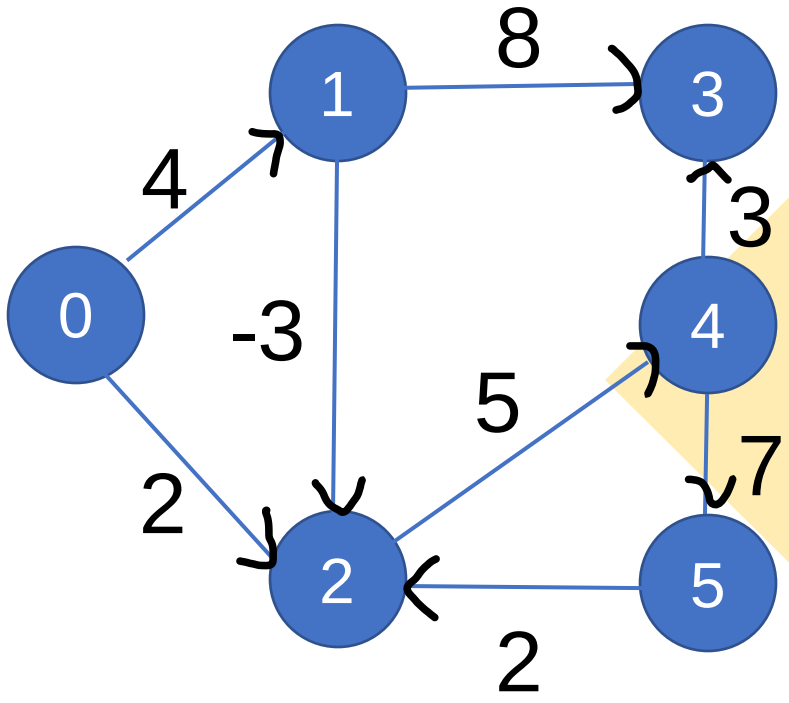


Bellman-Ford

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_tmp rồi đưa vào queue sau khi kết thúc vòng lặp

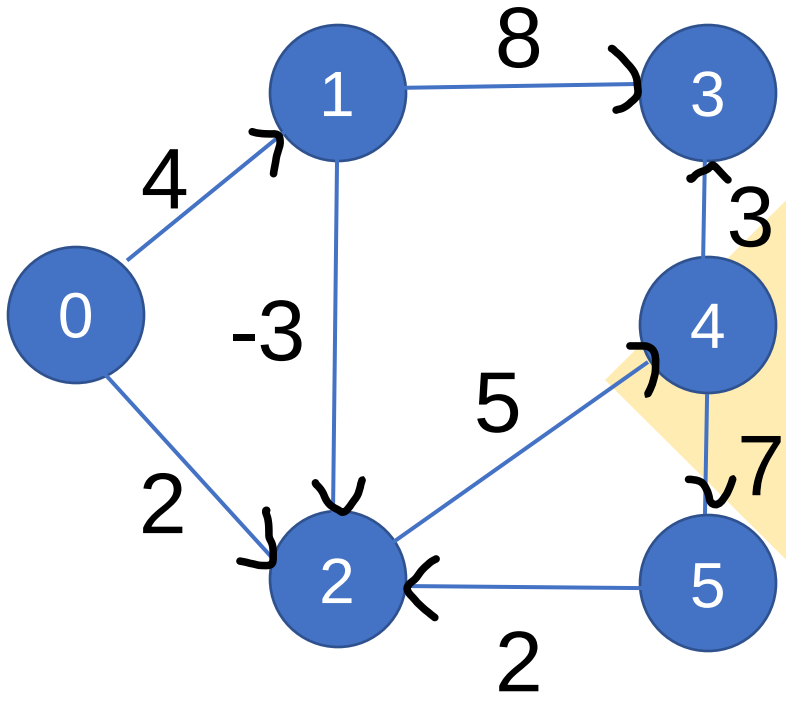


Chỉ khi kết thúc tất cả vòng lặp mới chốt tất cả distance

Bellman-Ford

Hướng giải quyết:

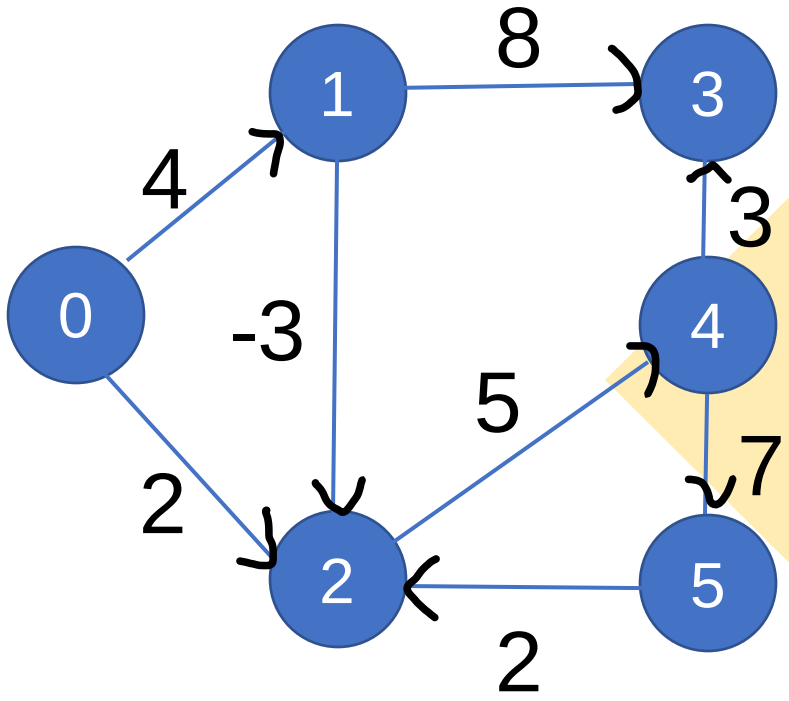
- Tạo dst_pre là bản copy của dst, queue_tmp rỗng
- Thực hiện cập nhật từ các đỉnh u trong queue:
  - Hướng nhìn không có điều kiện
  - Nếu có đỉnh v:
    - dst_new == dst_pre[u] + weight(u, v)
    - dst_new < dst_pre[v]
  - Thì:
    - cập nhật dst[v] = dst_new
    - Thêm v vào queue_tmp
- Thay queue = queue_tmp và dst_pre = dst



Thực hiện |V|-1 lần

Bellman-Ford

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

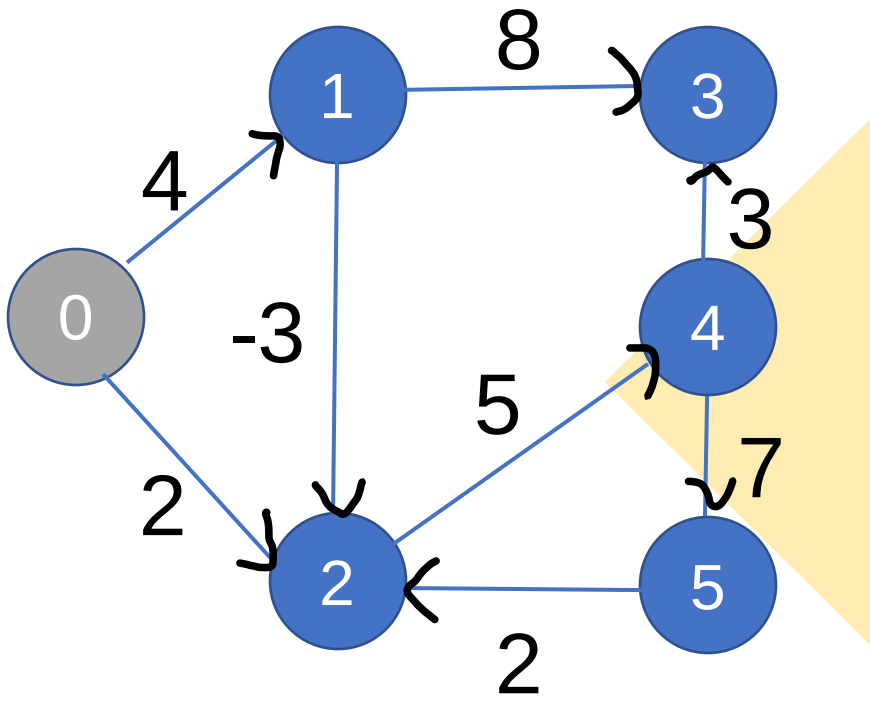


queue

0					

Bellman-Ford

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

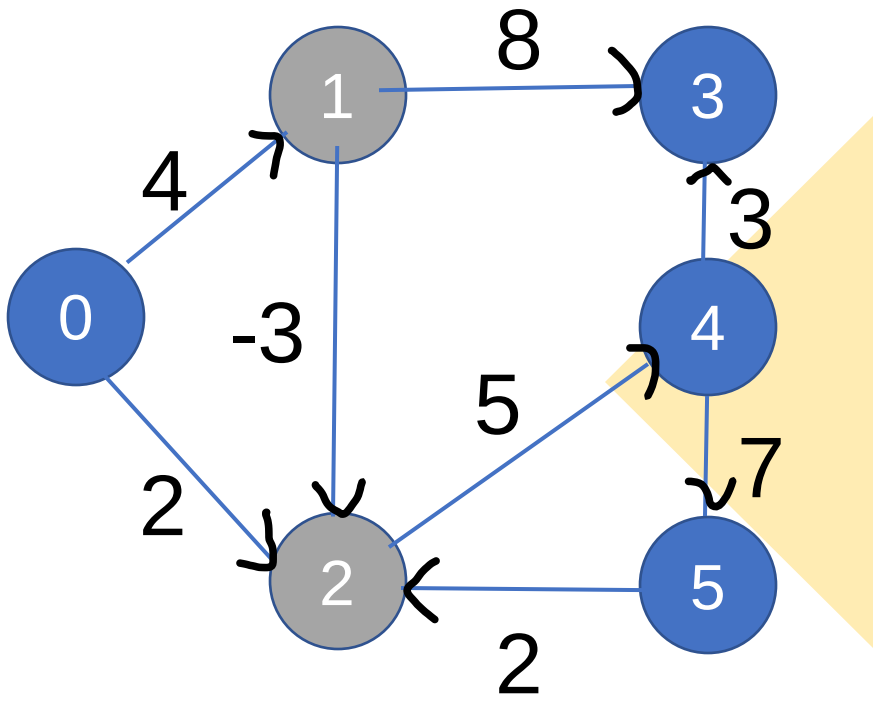


queue

1	2				

Bellman-Ford

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

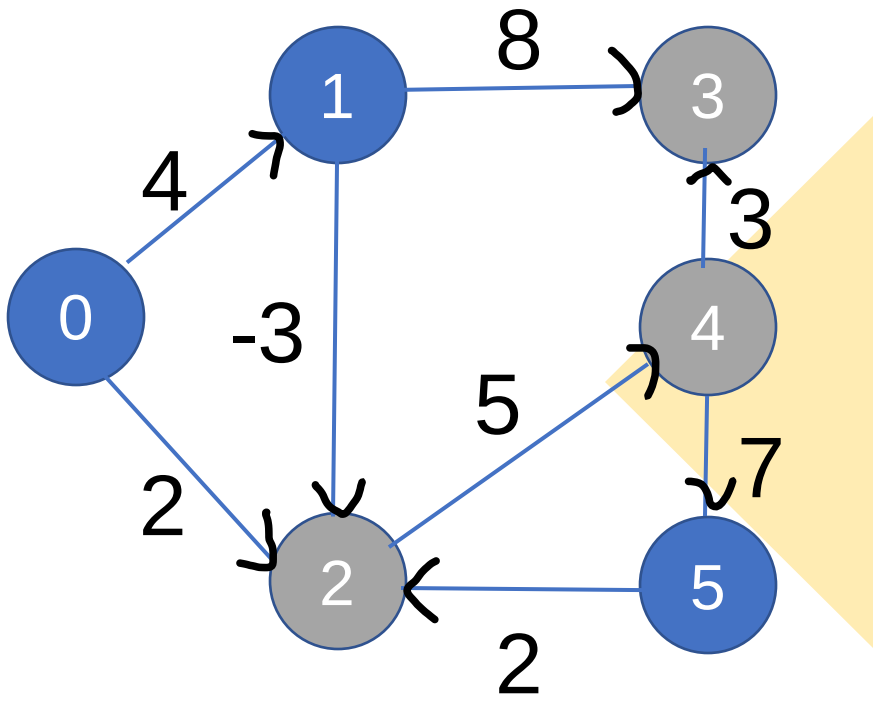


queue

2	3	4			

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

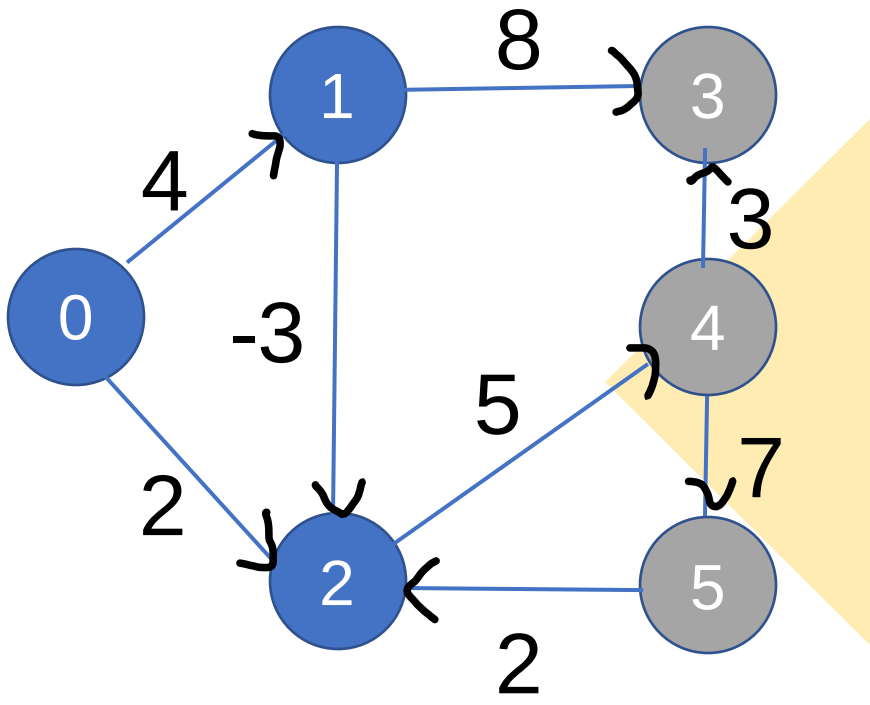


queue

4	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

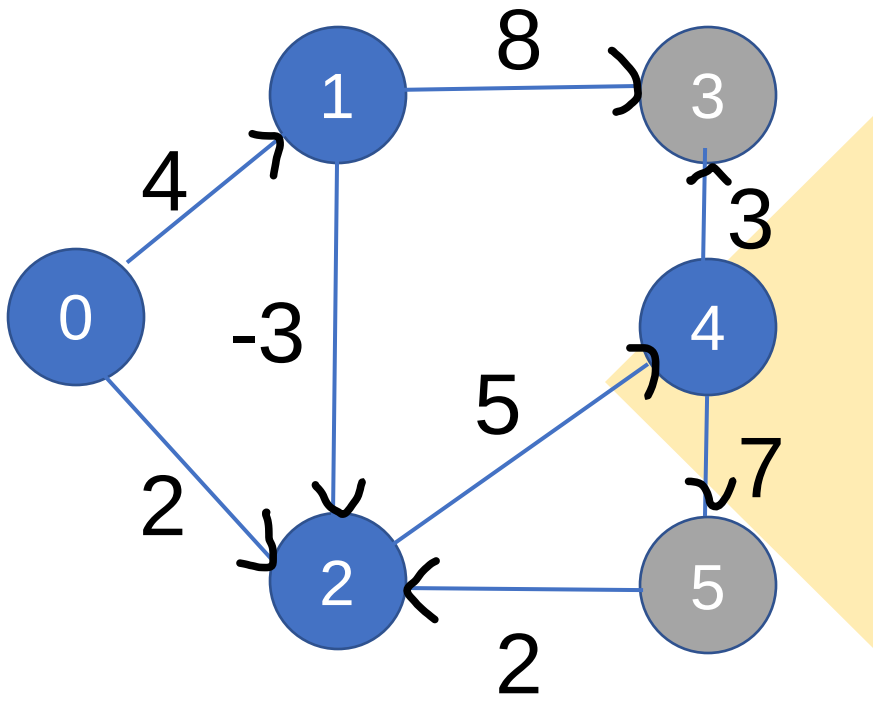


queue

3	5				

Bellman-Ford

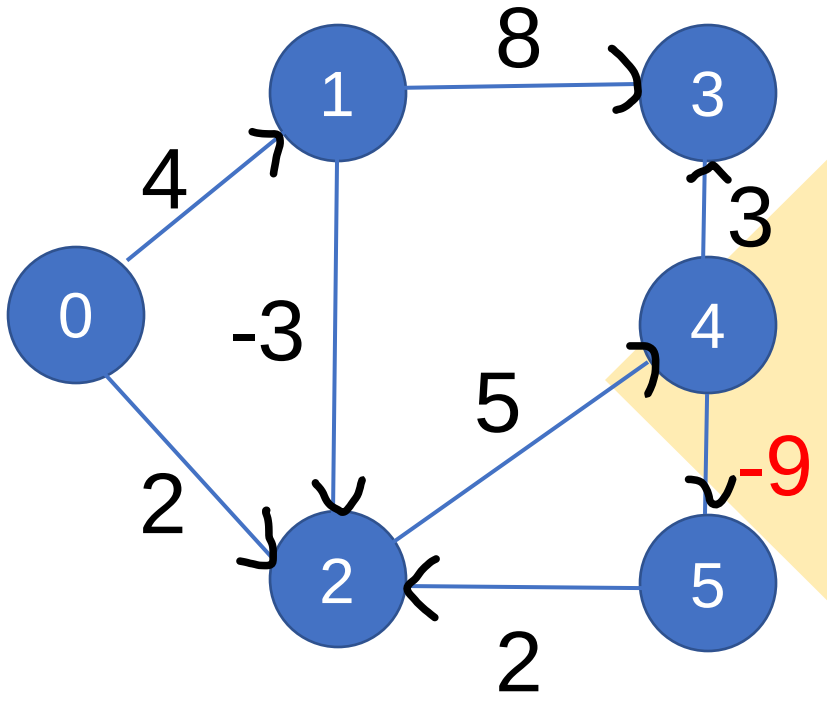
	0	1	2	3	4	5
	0	inf	inf	inf	inf	inf
(0)		4	2			
(1) (2)			1	12	7	
(2) (3) (4)				10	6	14
(4) (3) (5)				9		13
(3) (5)						
	0	4	1	9	6	13



queue

Bellman-Ford

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



Thực hiện |V|-1 lần

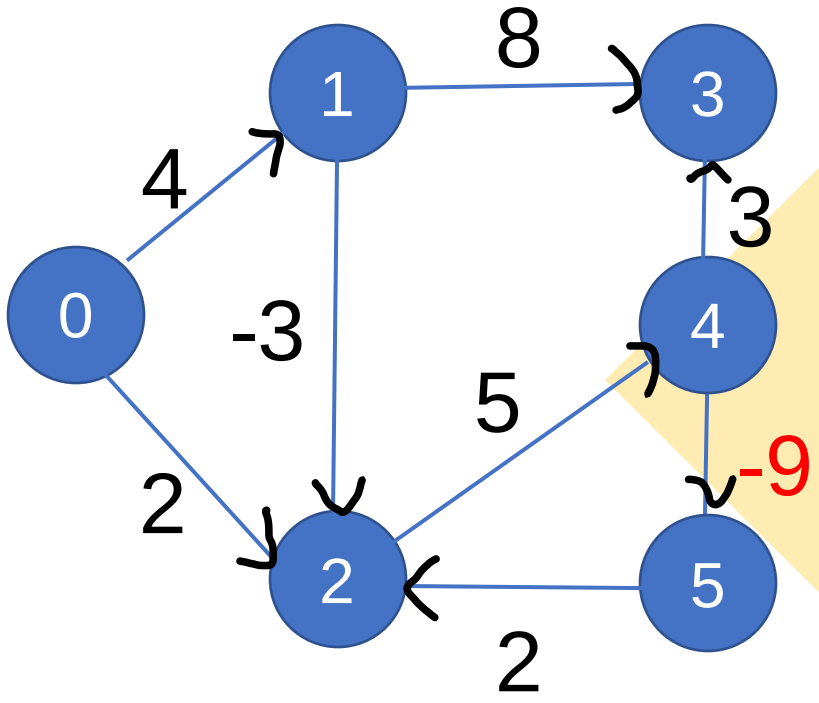
Bellman-Ford

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

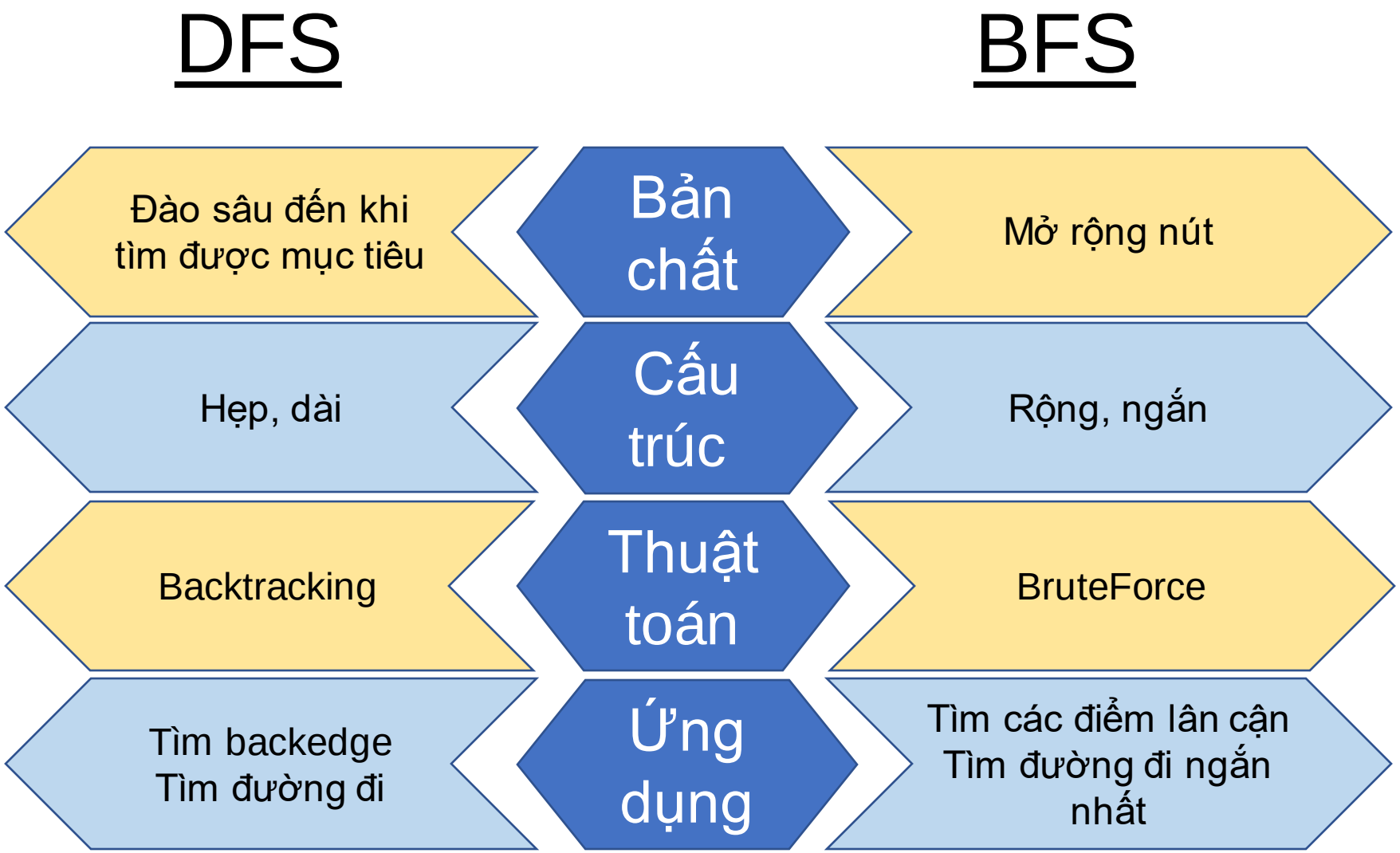
3. Tổng kết

- Khi nào dùng đồ thị
- So sánh DFS với BFS
- Ứng dụng của đồ thị

Khi nào dùng đồ thị

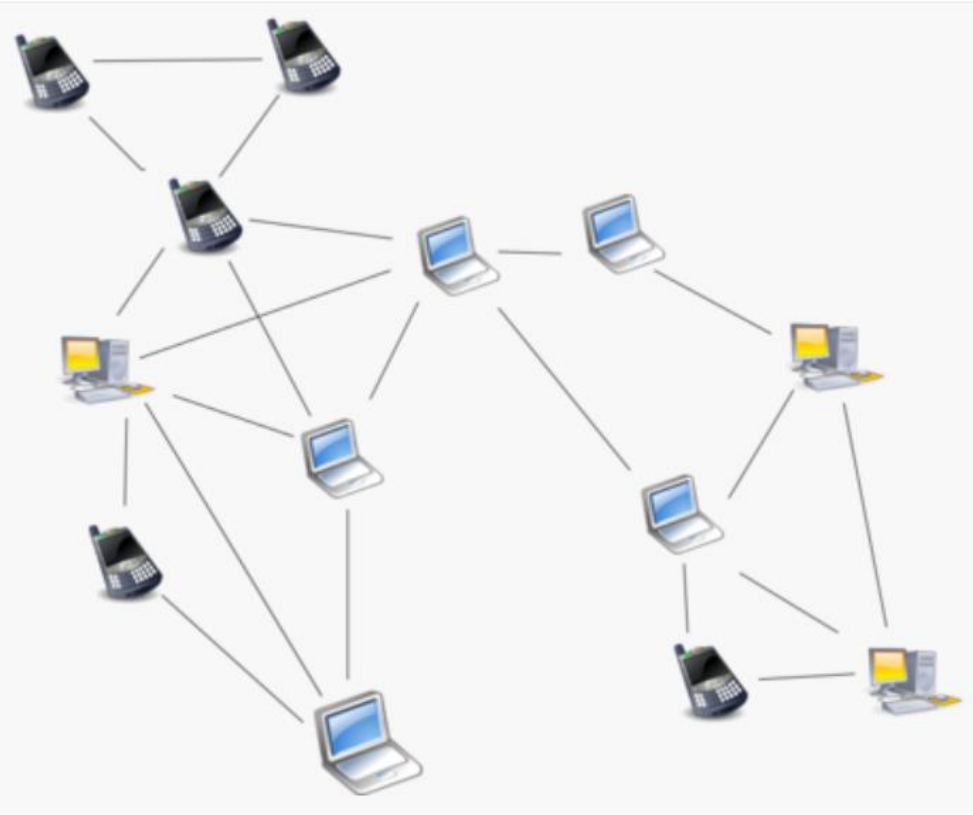
- Khi input đề là các đối tượng và phải có kết nối giữa các đối tượng đó, theo 1 chiều hoặc 2 chiều

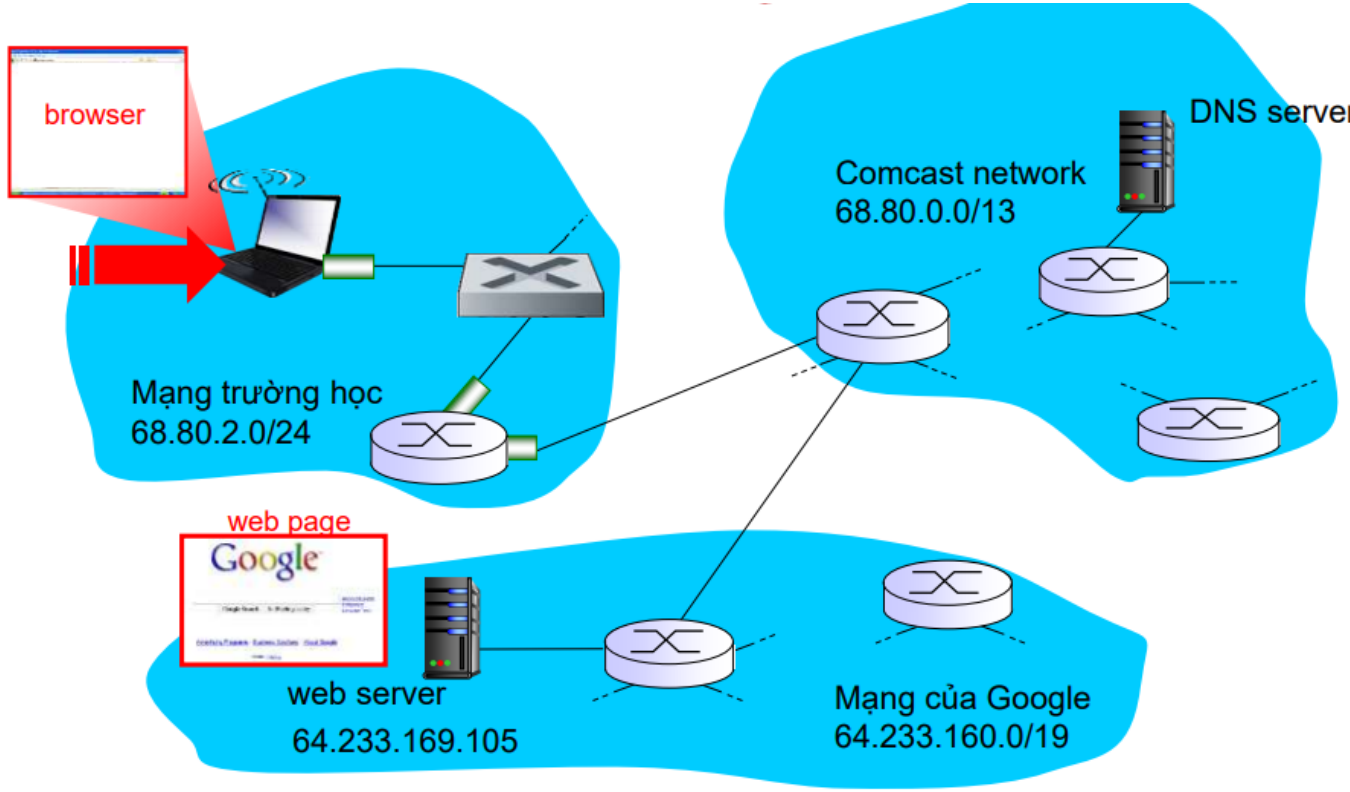
So sánh DFS, BFS



Ứng dụng của thuật toán đồ thị

Thuật toán đồ thị được áp dụng rộng rãi trong mạng máy tính nhằm tìm ra địa chỉ của các thiết bị được kết nối trong mạng. Xem các thiết bị là các đỉnh và đường liên kết giữa chúng là các cạnh, chúng ta sẽ có một đồ thị.



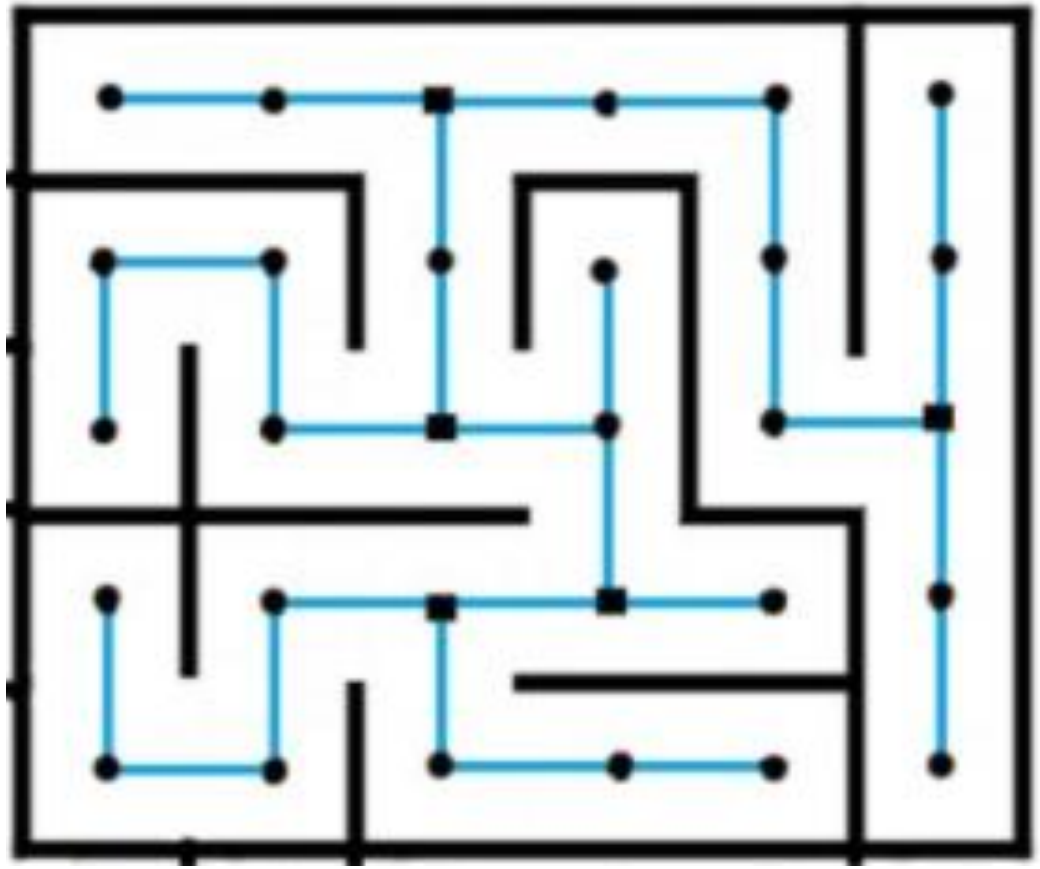


Ứng dụng của thuật toán đồ thị

Thuật toán BFS được sử dụng trong các ứng dụng định vị để tìm đường đi ngắn nhất. Trọng số âm có thể được thêm vào để thể hiện những con đường cấm đi.

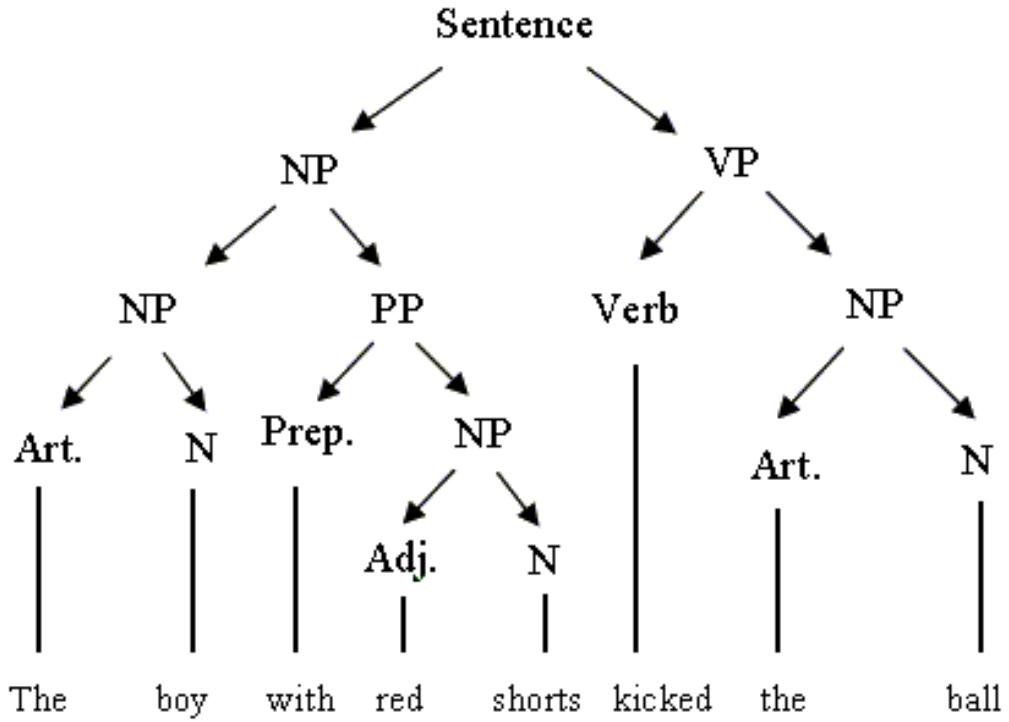
Ứng dụng của thuật toán đồ thị

Thuật toán DFS được sử dụng để giải mã các mê cung, hang động... bằng cách xem mỗi giao điểm là một đỉnh và đường đi giữa chúng là các cạnh



Ứng dụng của thuật toán đồ thị

Đồ thị được áp dụng trong Xử lí ngôn ngữ tự nhiên:



Ứng dụng của thuật toán đồ thị

Đồ thị được áp dụng trong các chuỗi phản ứng hóa học. Các trọng số có thể được dùng để biểu diễn mức năng lượng tạo ra với trọng số dương và mức năng lượng hấp thụ với trọng số âm.