Topological Sort

Ju-Won Seo
2019.08.21

Table of Contents

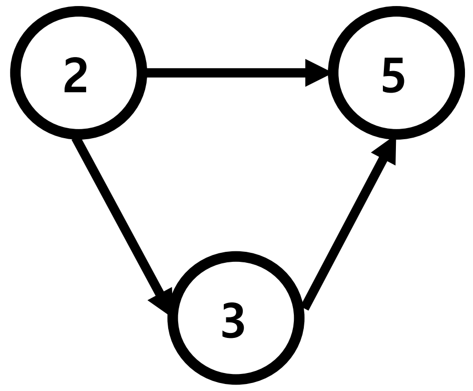
1. Introduction

What is Topological sort?

- Listing all vertices without violating the linear ordering of each vertex.

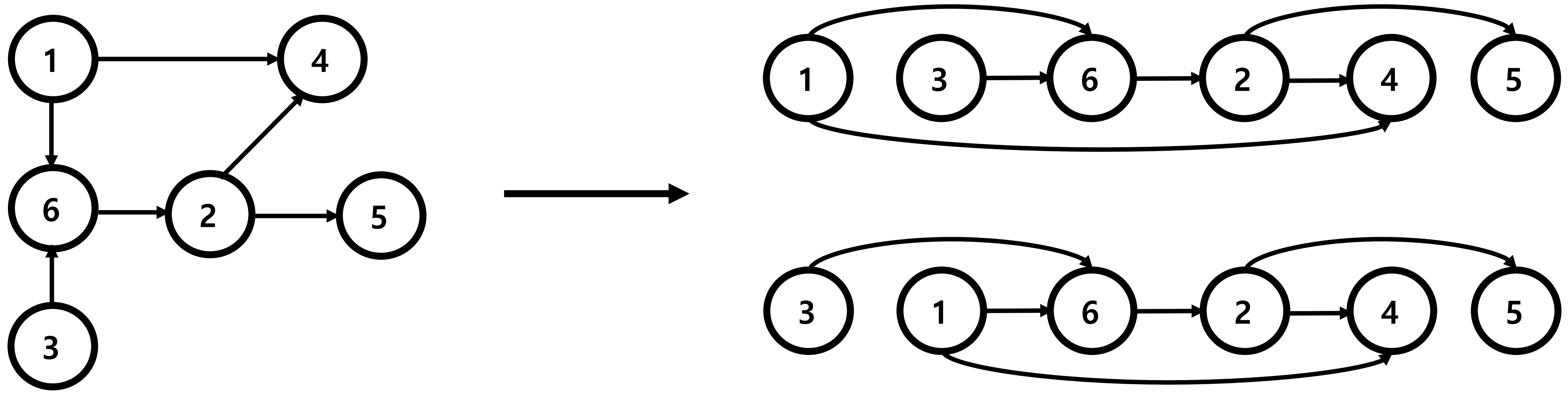
- The graph is a directed acyclic graph called 'DAG'.

- After sorting, we can view a horizontal line so that all directed edges go from left to right.



DAG

What is Topological sort?



Applications

- To indicate precedence among events.

1. A project that must be completed before each task can be completed.

2. Prerequisite subject.

2. DFS, BFS

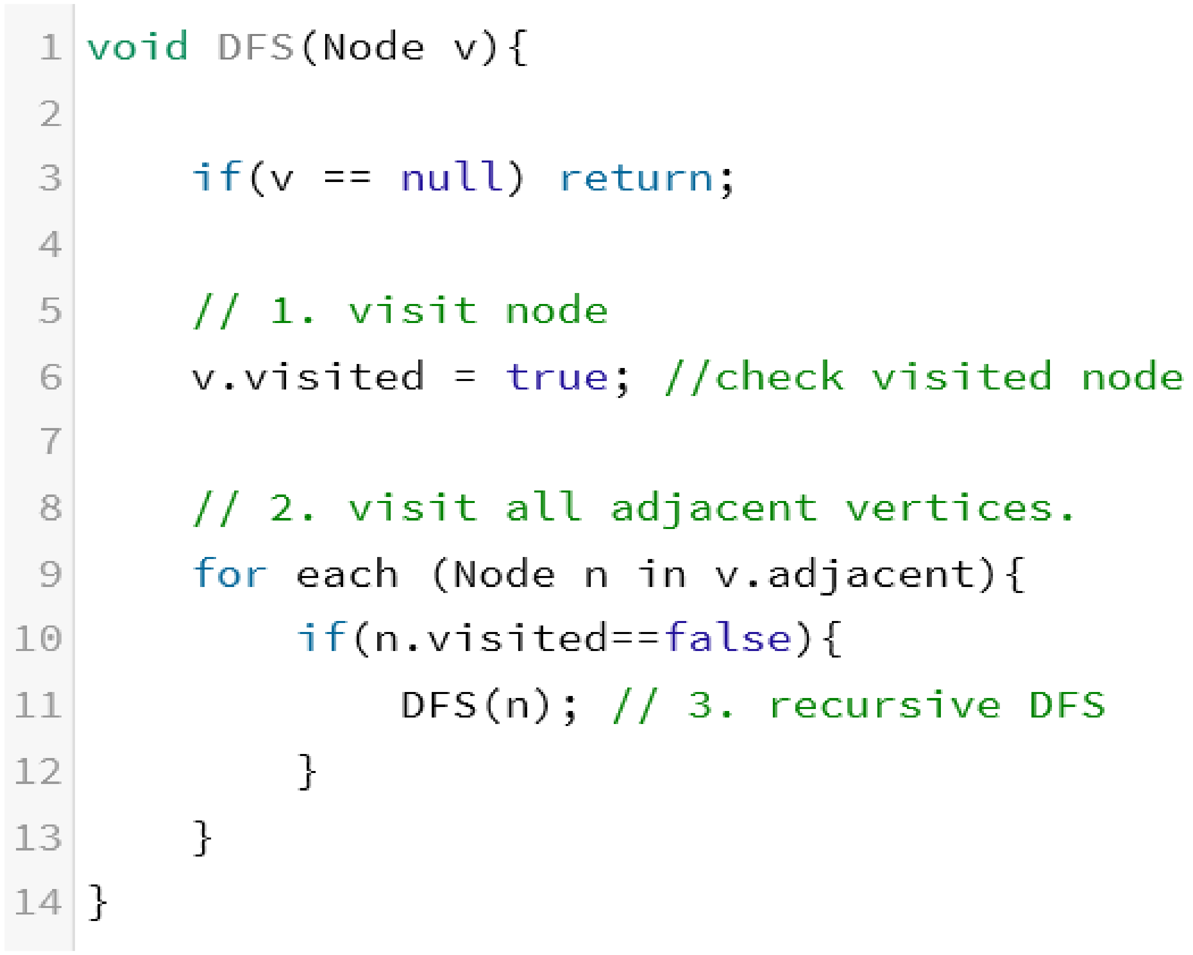
- DFS (Depth First Search)

- DFS has a form of recursive algorithm.

- If you want to visit all nodes, you can select this method.

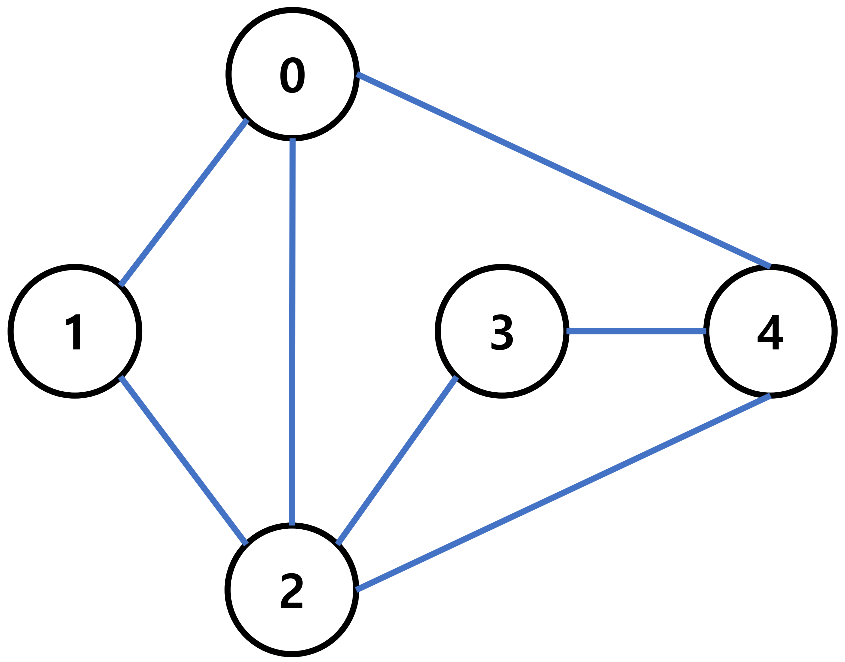
- Must be checked whether any nodes have been visited.
  If you not, there is a risk into an infinite loop.

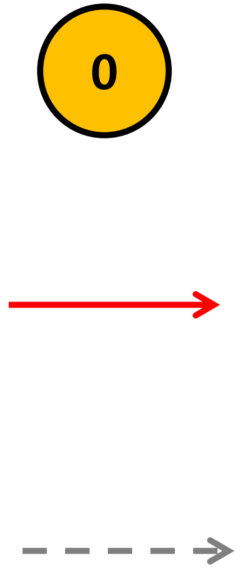
- Algorithm



```
void DFS(Node v){

    if(v == null) return;

    // 1. visit node
    v.visited = true; //check visited node

    // 2. visit all adjacent vertices.
    for each (Node n in v.adjacent){
        if(n.visited==false){
            DFS(n); // 3. recursive DFS
        }
    }
}
```

- Algorithm



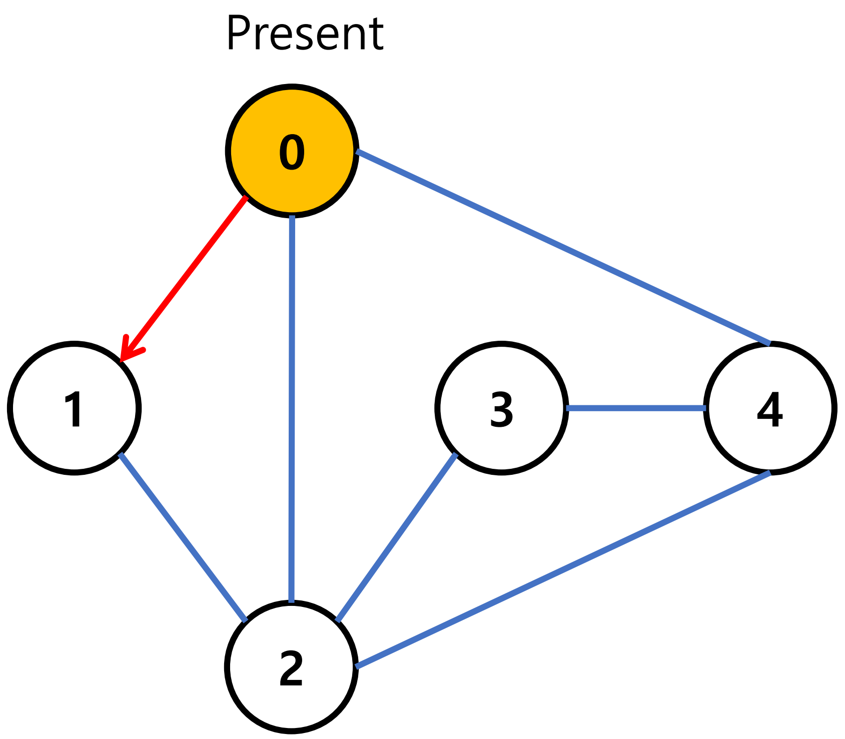


Visited vertex

Present edge

Visited edge

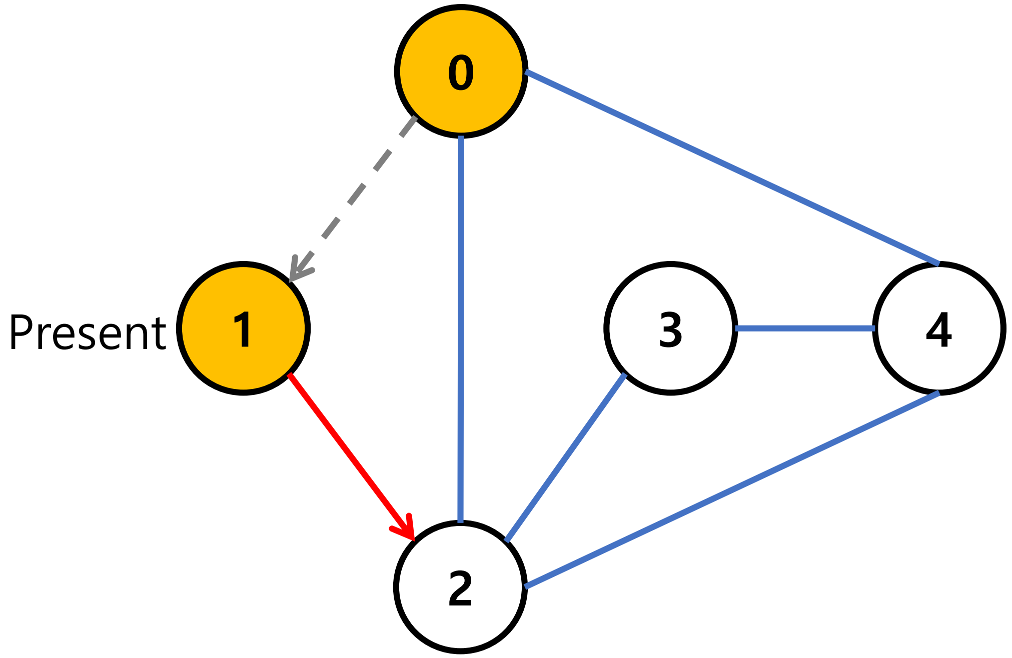
- Algorithm



Visited

0				

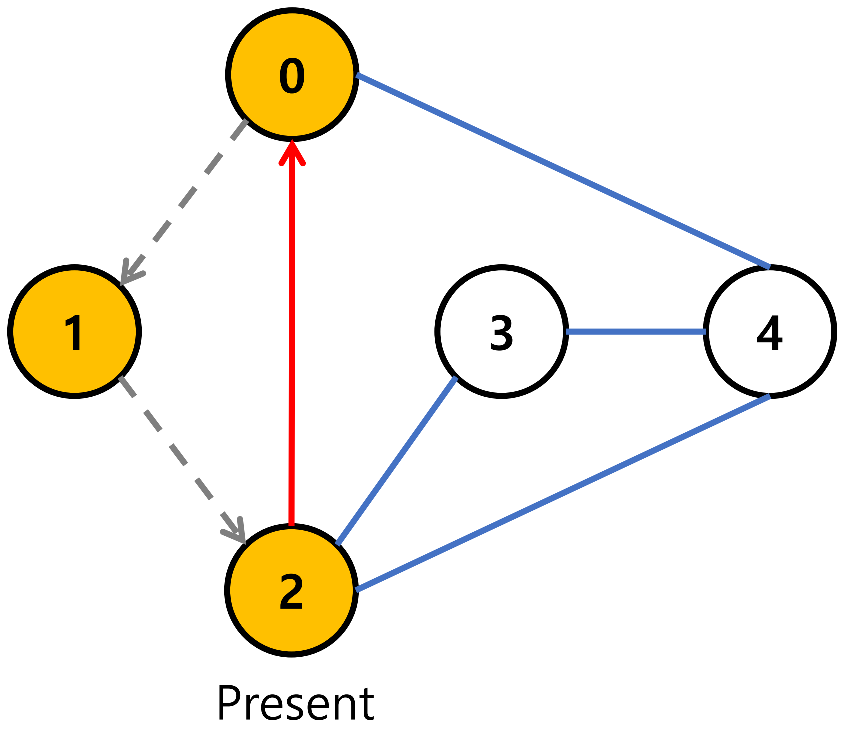
- Algorithm



Visited

0	1			

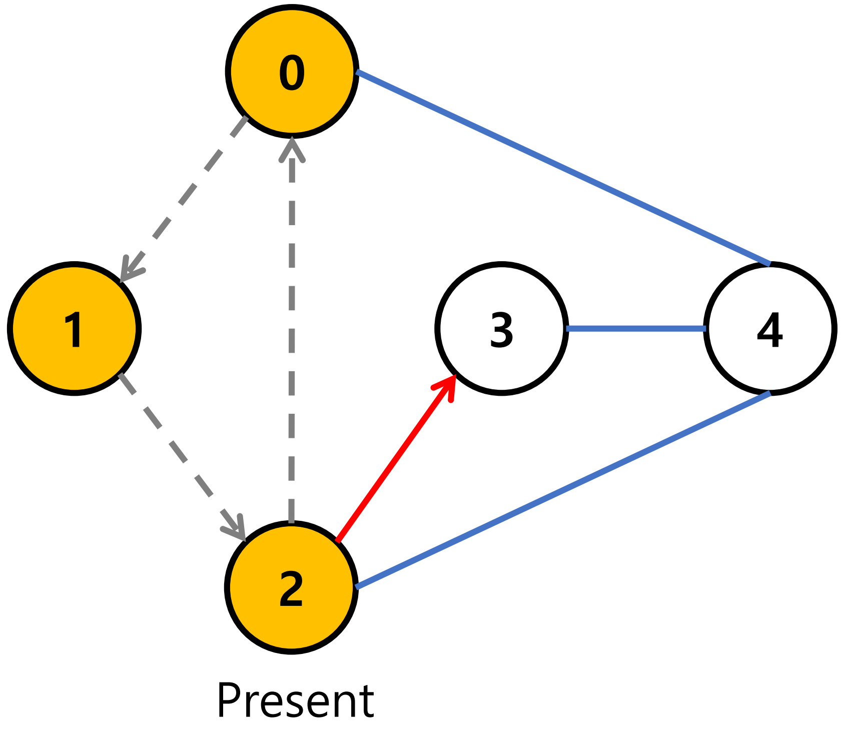
- Algorithm



Visited

0	1	2		

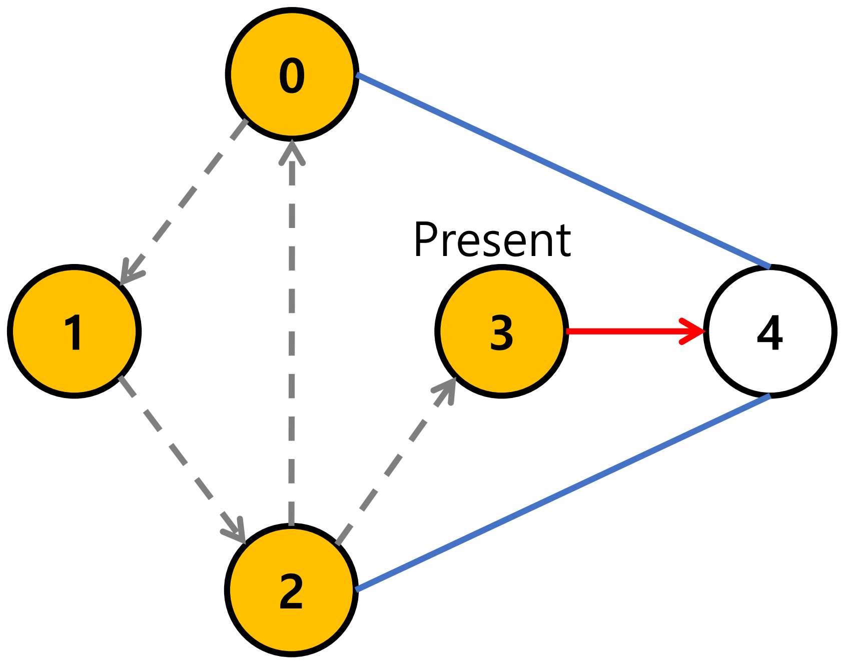
- Algorithm



Visited

0	1	2		

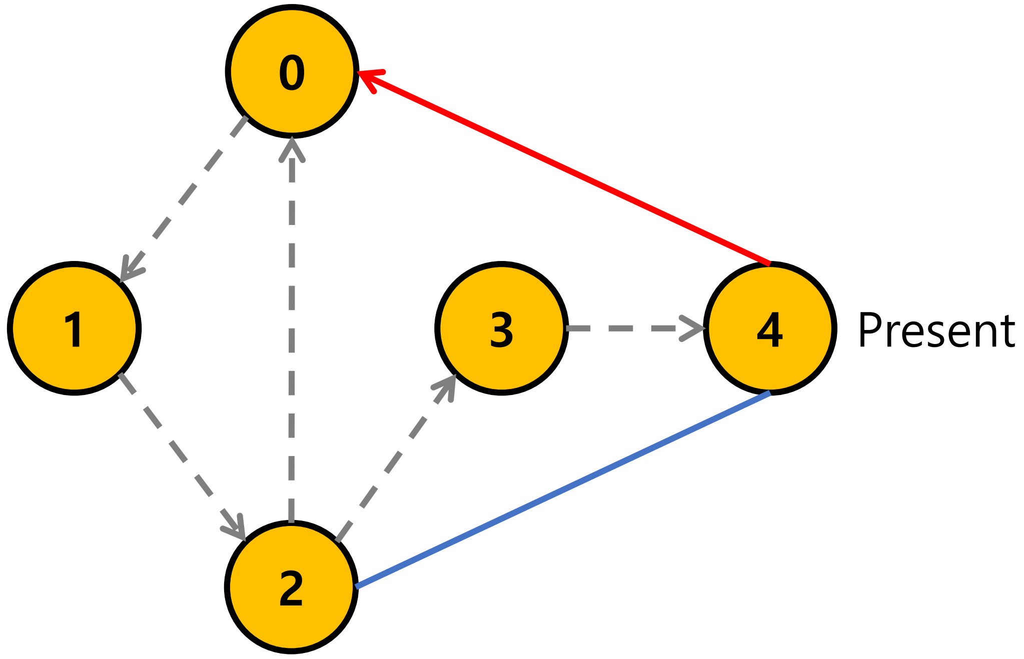
- Algorithm



Visited

0	1	2	3	

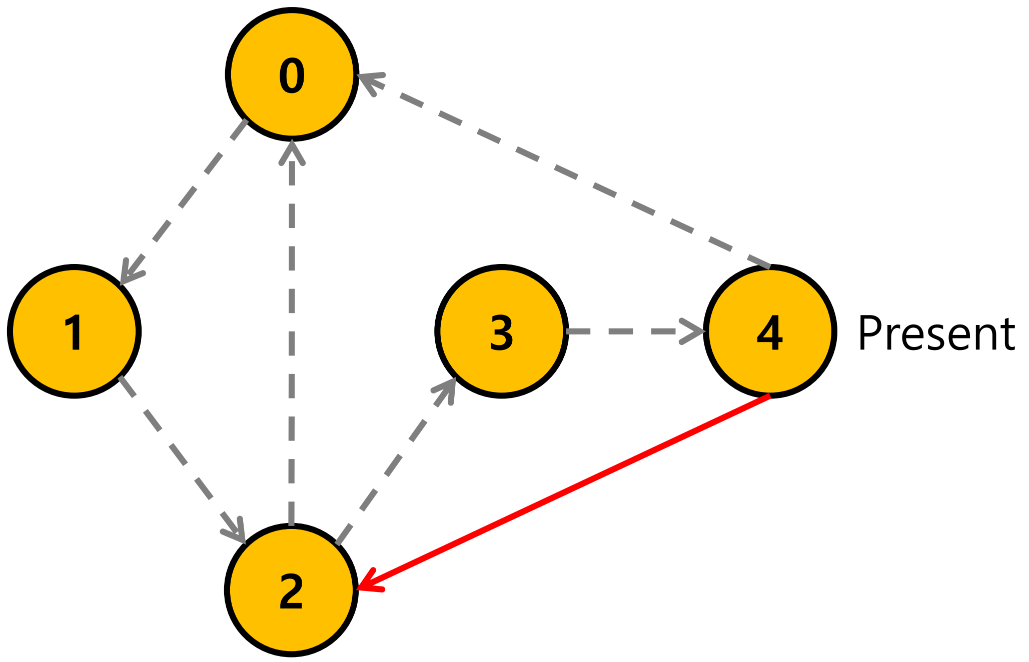
- Algorithm



Visited

0	1	2	3	4

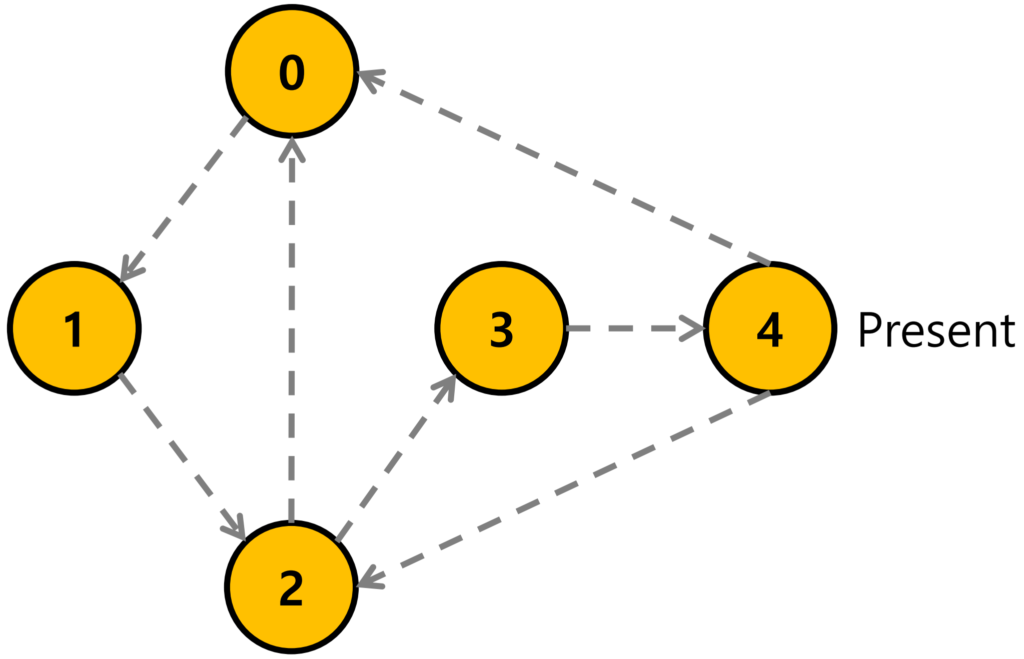
- Algorithm



Visited

0	1	2	3	4

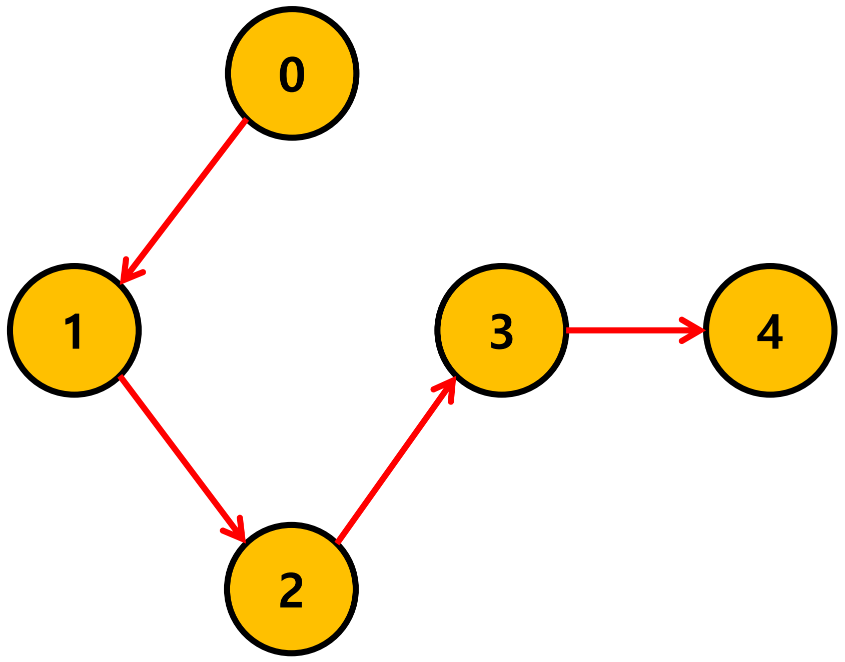
- Algorithm



Visited

0	1	2	3	4

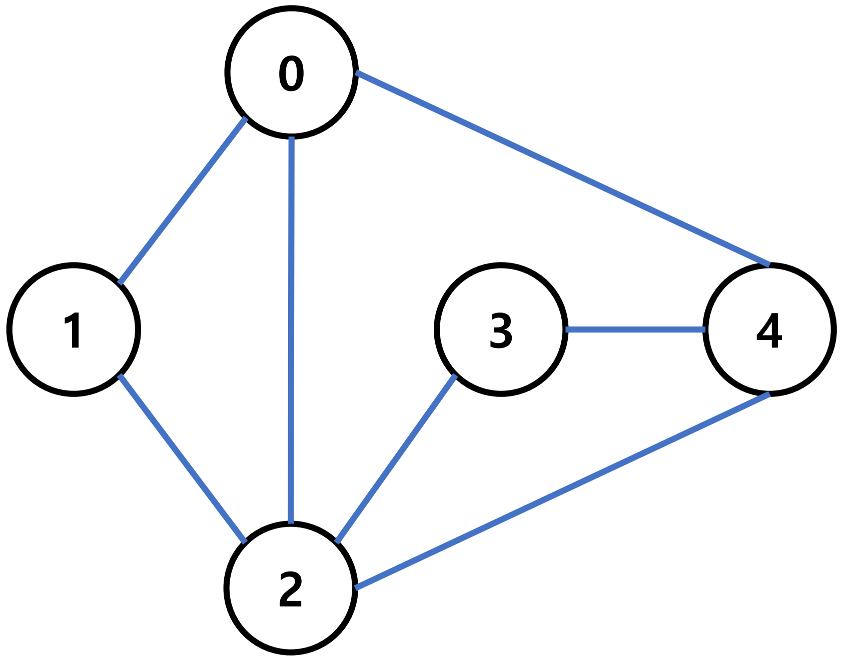
- Algorithm



Visited

0	1	2	3	4

- Analysis



Adjacent list : $\mathbf{O}(\boldsymbol{V}+\boldsymbol{E})$

$(\boldsymbol{V}\ \boldsymbol{is}\ \boldsymbol{vetex}\ \boldsymbol{and}\ \boldsymbol{E}\ \boldsymbol{is}\ \boldsymbol{edge}\ )$

- BFS (Breadth First Search)

- It visits vertices near the starting vertex first.

- When you want to find the shortest path, you can choose this method.

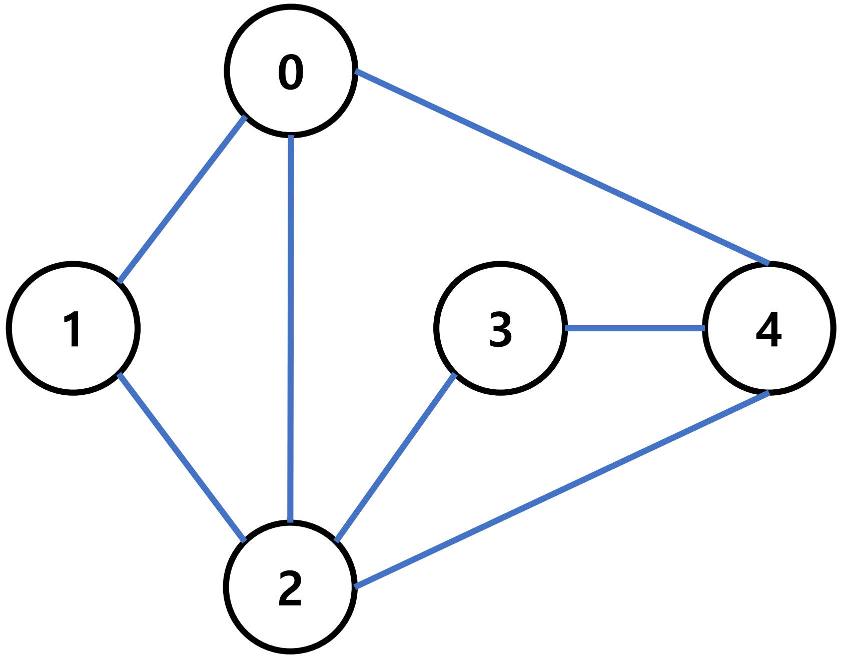
- It used data structure called ‘Queue’

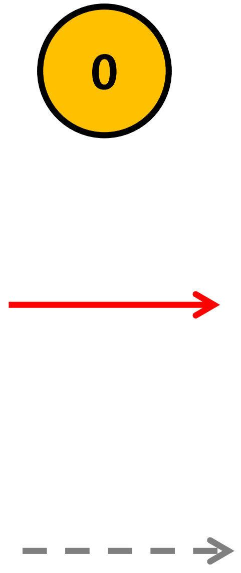
- Must be checked whether any nodes have been visited.
  If you not, there is a risk into an infinite loop.

- Algorithm

```
void BFS(Node* start){

    q.push(satart); // push the start vertex in queue

    //repeat until queue is empty
    while(!q.empty()){

        node* tmp = q.front(); // pop first vertex in queue
        q.pop();
        tmp.visited = true; // checked present vertex

        //visit all adjacent vertices of tmp vertex
        for each (node v in tmp.adjacent){
            if(v.visited == false){
                v.visited = true; // checked v vertex
                q.push(v); // push the v vertex in queue
            }
        }
    }
}
```

- Algorithm



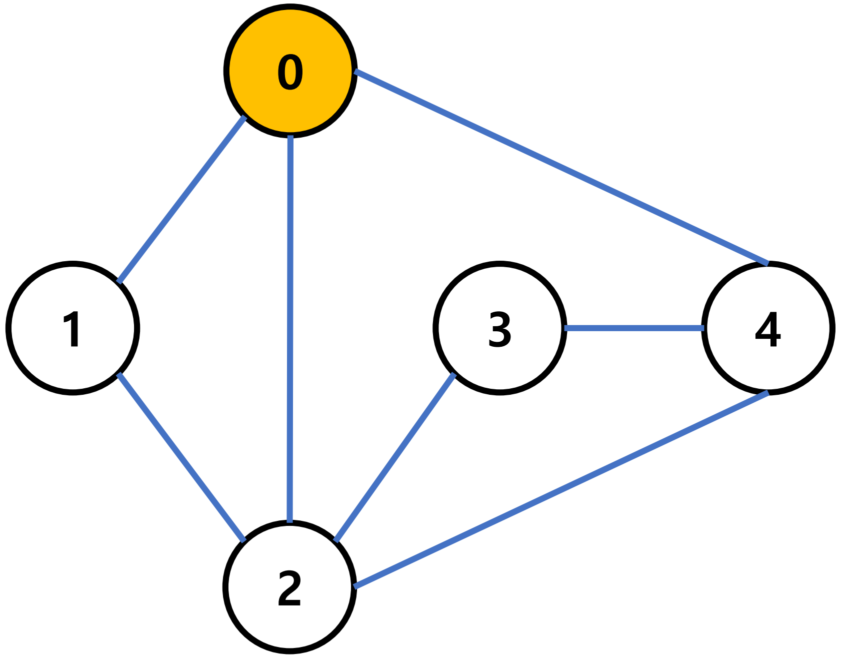


Visited vertex

Present edge

Visited edge

- Algorithm



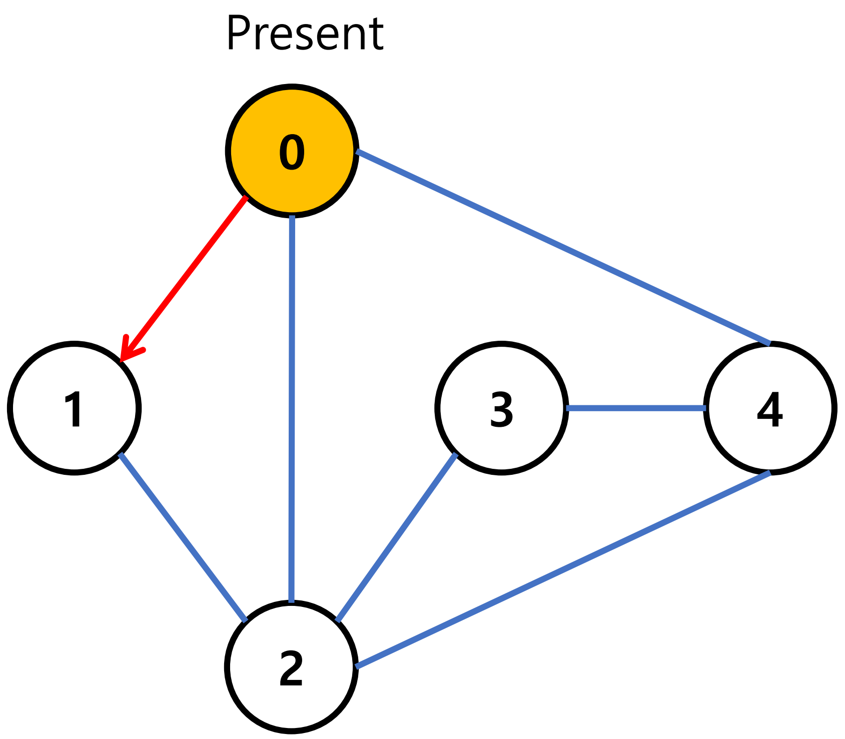
Queue

0			

Visited

0				

- Algorithm

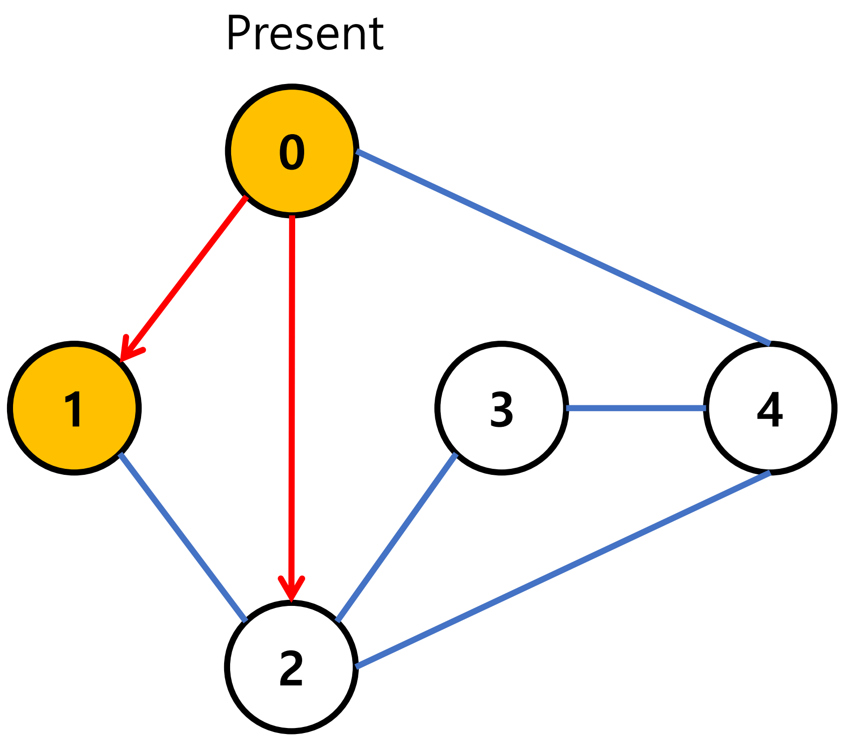


Queue

Visited

0				

- Algorithm



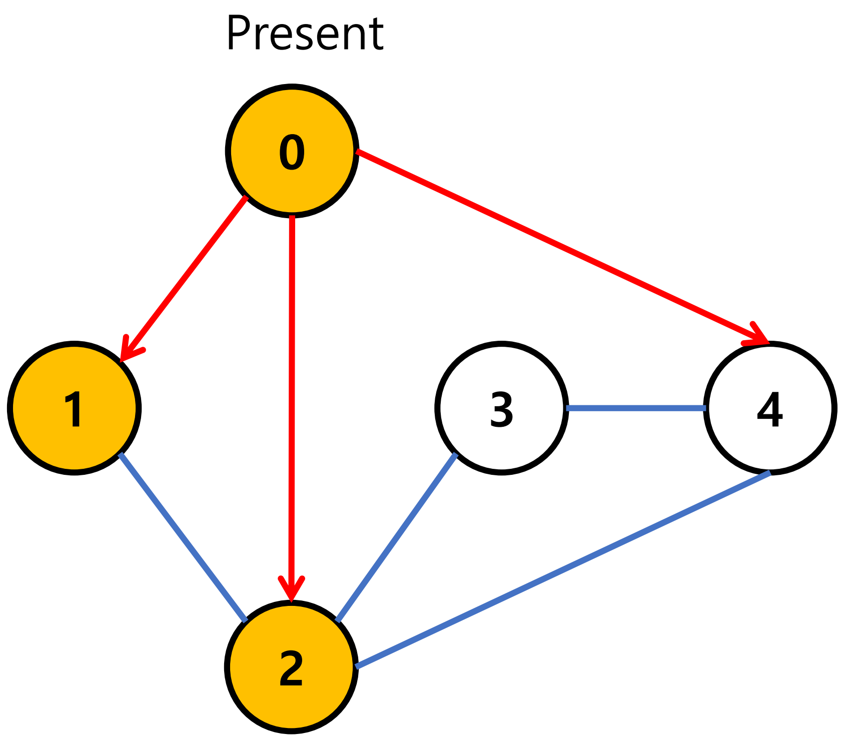
Queue

1			

Visited

0	1			

- Algorithm



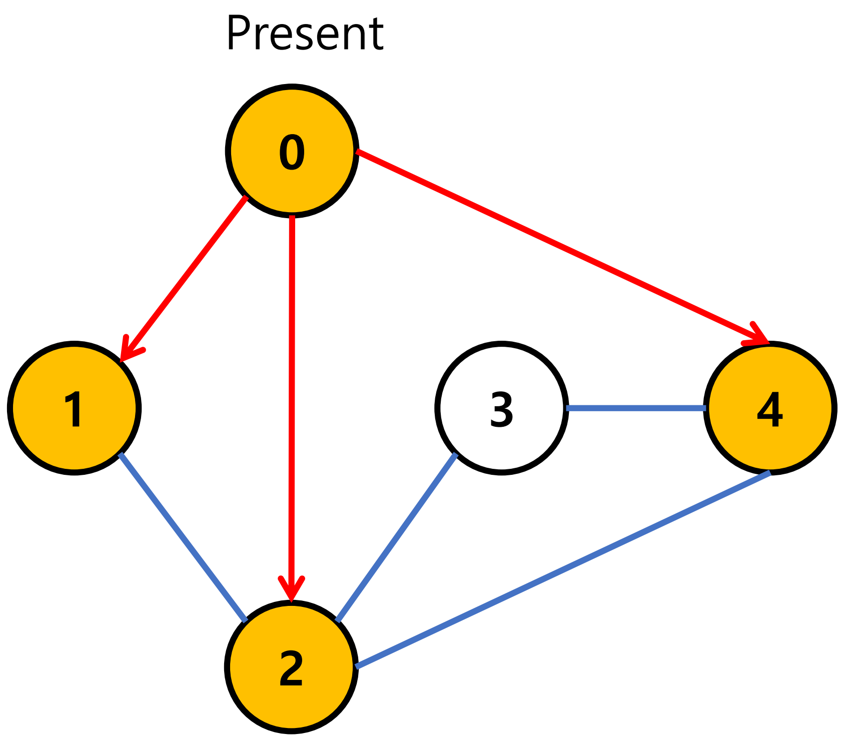
Queue

1	2		

Visited

0	1	2		

- Algorithm



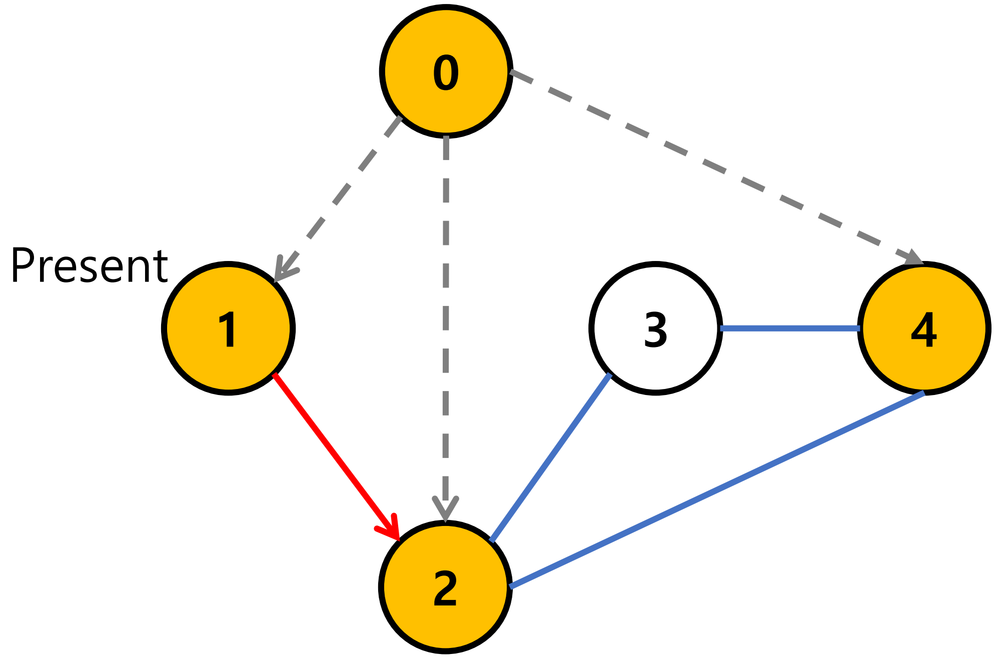
Queue

1	2	4	

Visited

0	1	2		4

- Algorithm



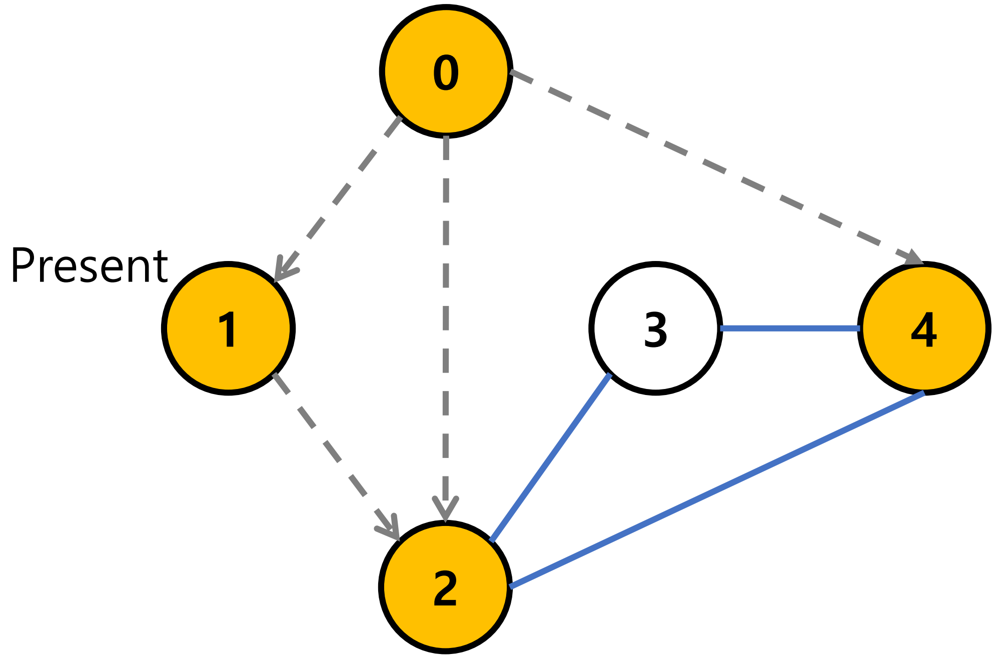
Queue

2	4		

Visited

0	1	2		4

- Algorithm



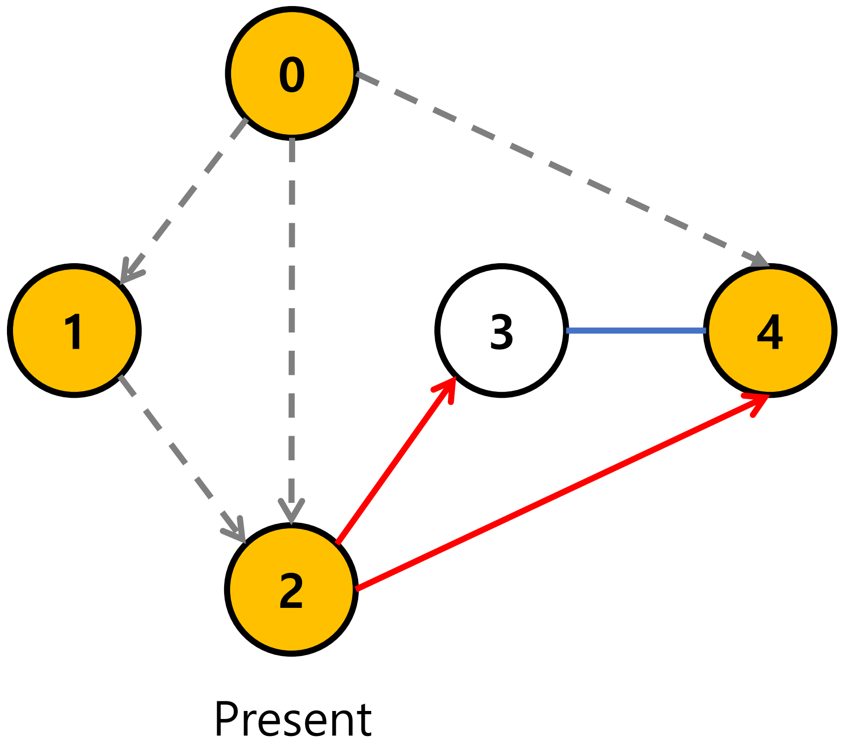
Queue

2	4		

Visited

0	1	2		4

- Algorithm



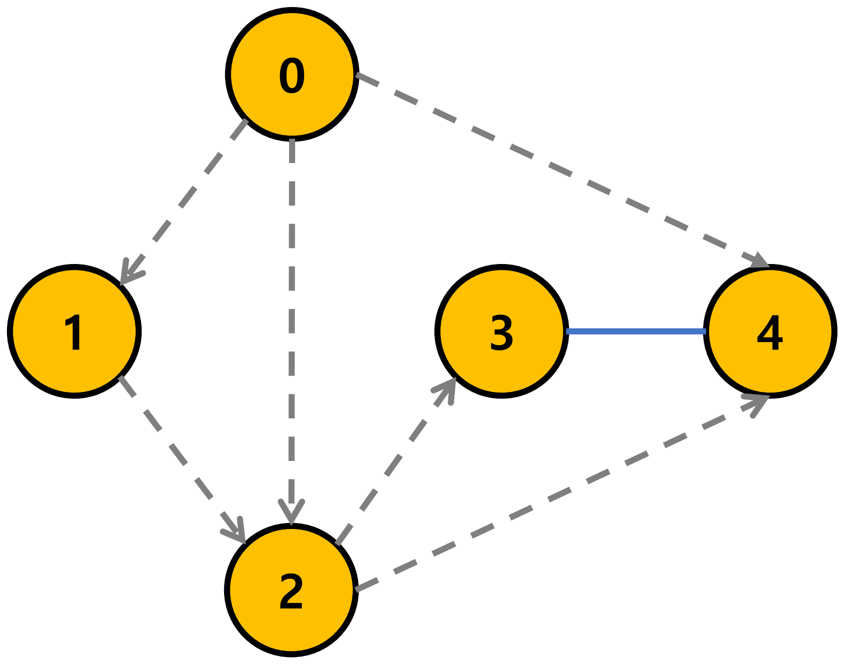
Queue

4			

Visited

0	1	2	3	4

- Algorithm



Present

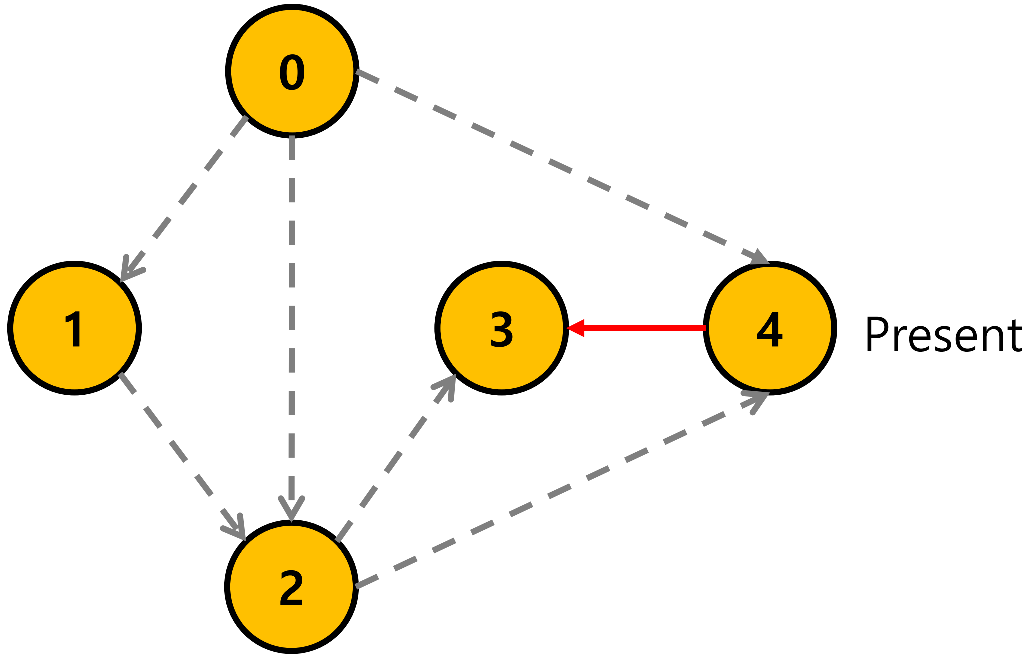
Queue

4	3		

Visited

0	1	2	3	4

- Algorithm



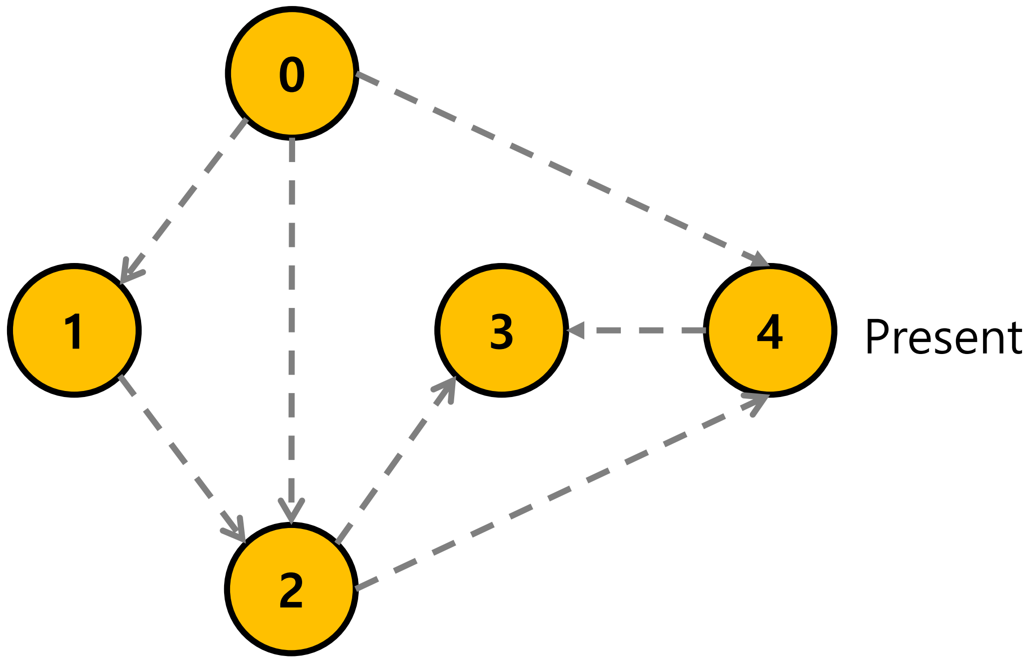
Queue

3			

Visited

0	1	2	3	4

- Algorithm



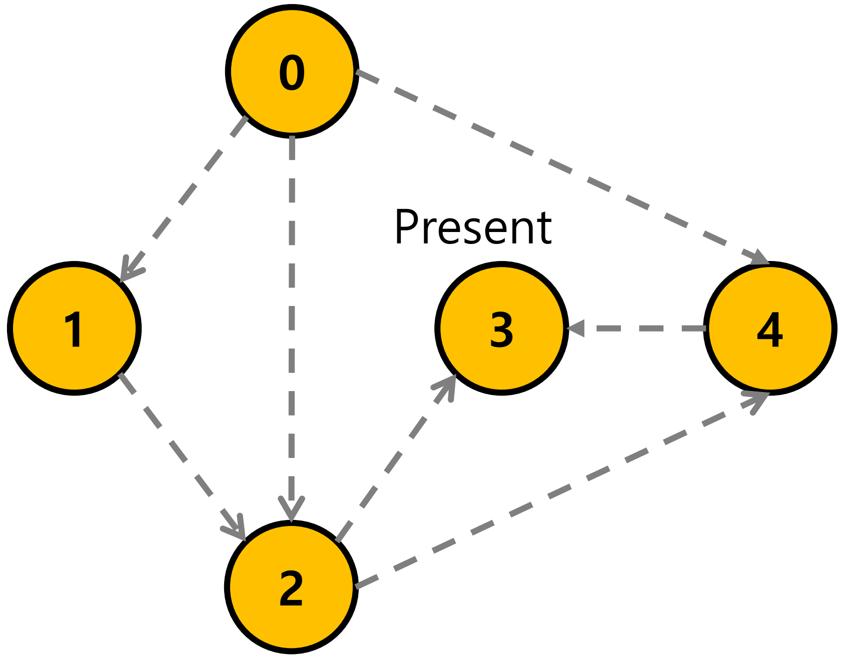
Queue

3			

Visited

0	1	2	3	4

- Algorithm

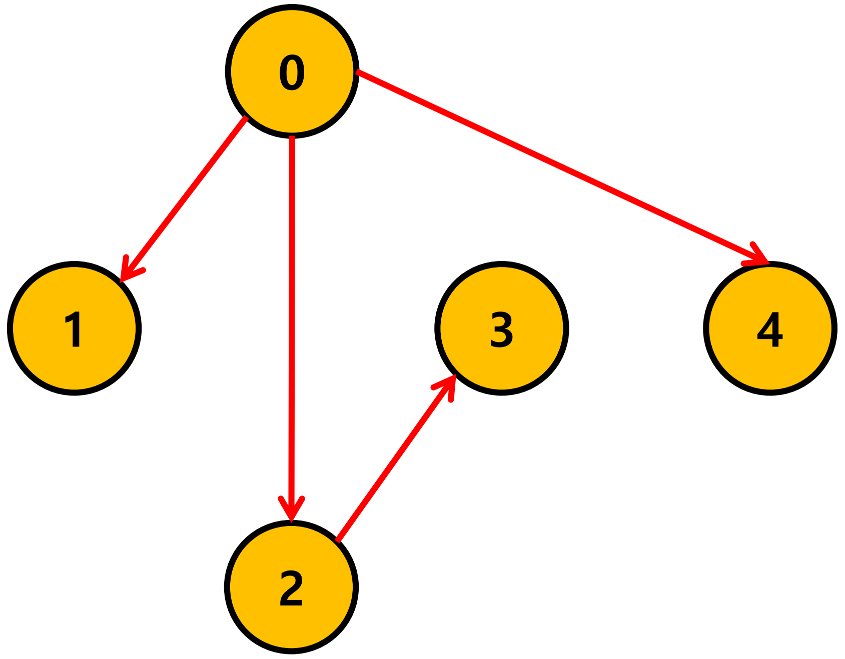


Queue

Visited

0	1	2	3	4

- Algorithm

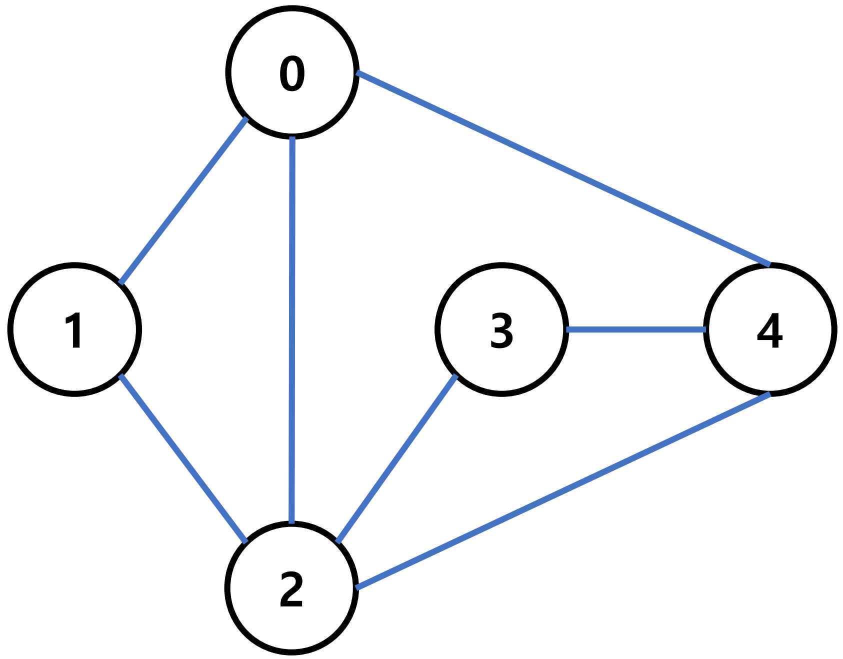


Queue

Visited

0	1	2	3	4

- Analysis

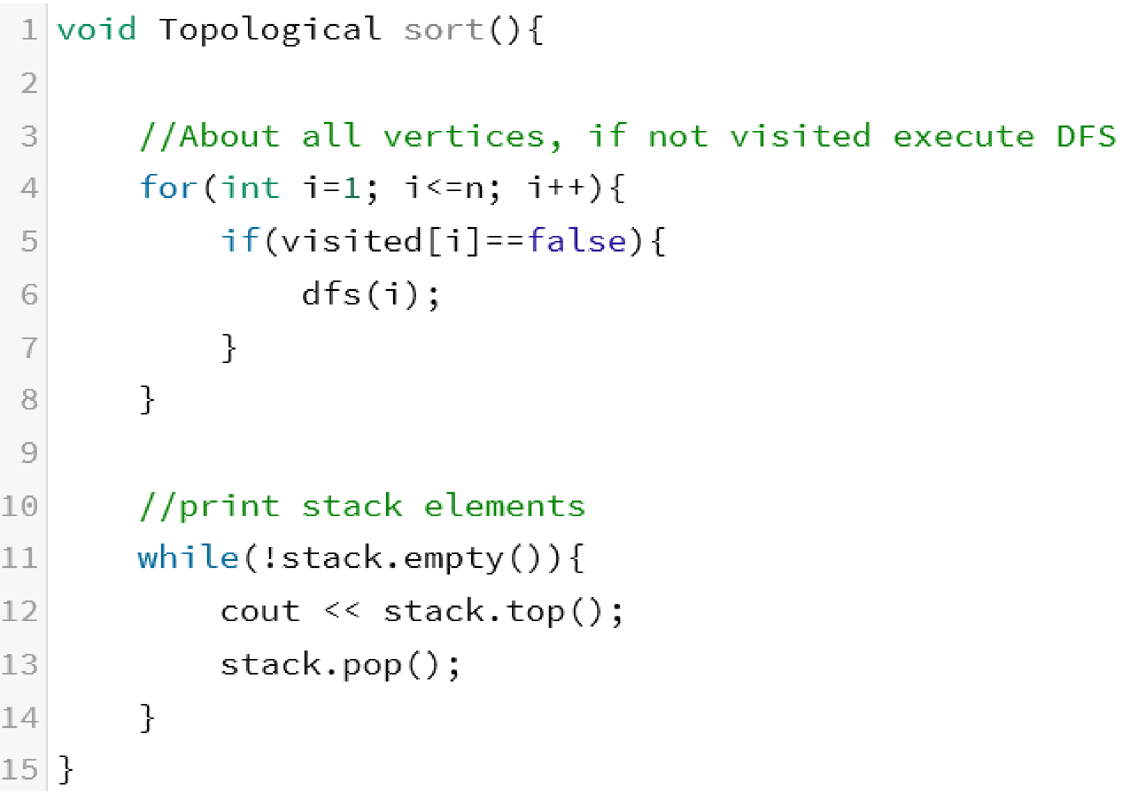


Adjacent list : $\mathbf{O}(\boldsymbol{V}+\boldsymbol{E})$

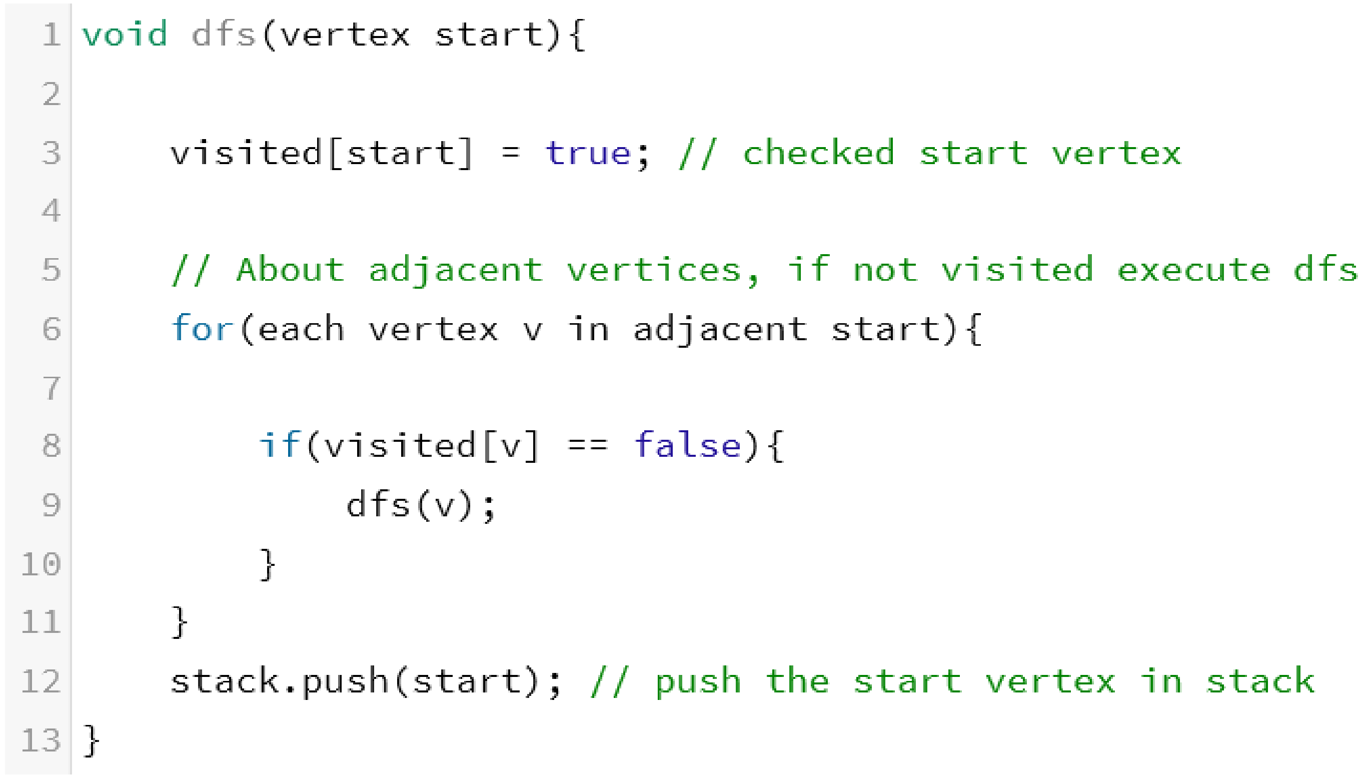
$(\boldsymbol{V}\ \boldsymbol{is}\ \boldsymbol{vetex}\ \boldsymbol{and}\ \boldsymbol{E}\ \boldsymbol{is}\ \boldsymbol{edge})$

3. Topological Sort

- Topological sort (DFS)

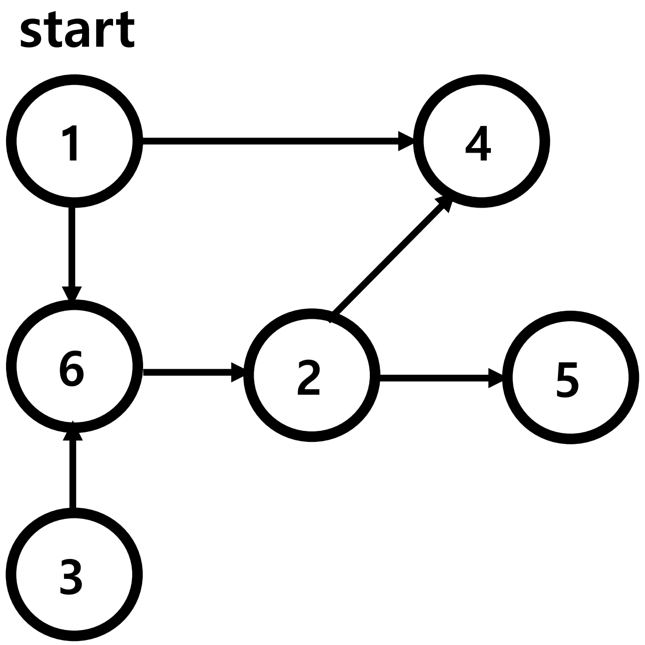


```
void Topological sort(){

    //About all vertices, if not visited execute DFS
    for(int i=1; i<=n; i++){
        if(visited[i]==false){
            dfs(i);
        }
    }

    //print stack elements
    while(!stack.empty()){
        cout << stack.top();
        stack.pop();
    }
}
```



```
void dfs(vertex start){

    visited[start] = true; // checked start vertex

    // About adjacent vertices, if not visited execute dfs
    for(each vertex v in adjacent start){

        if(visited[v] == false){
            dfs(v);
        }
    }
    stack.push(start); // push the start vertex in stack
}
```

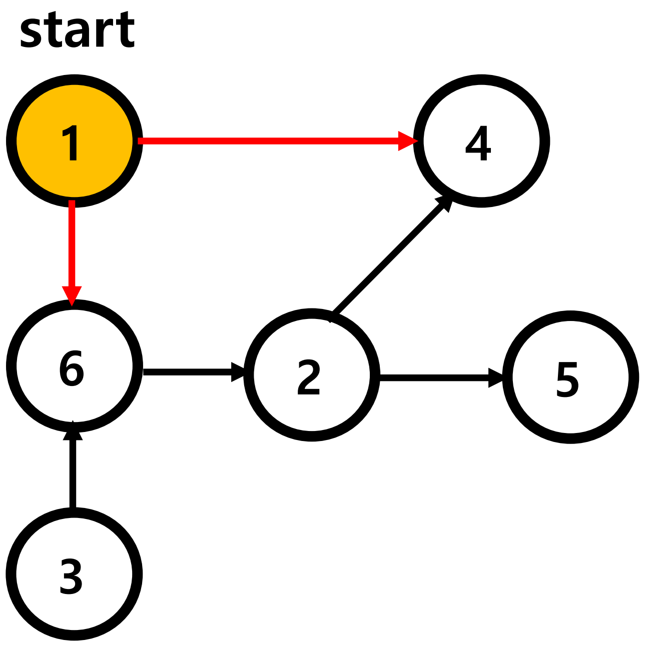
- Algorithm (Topological DFS)



Stack

Visited

- Algorithm (Topological DFS)

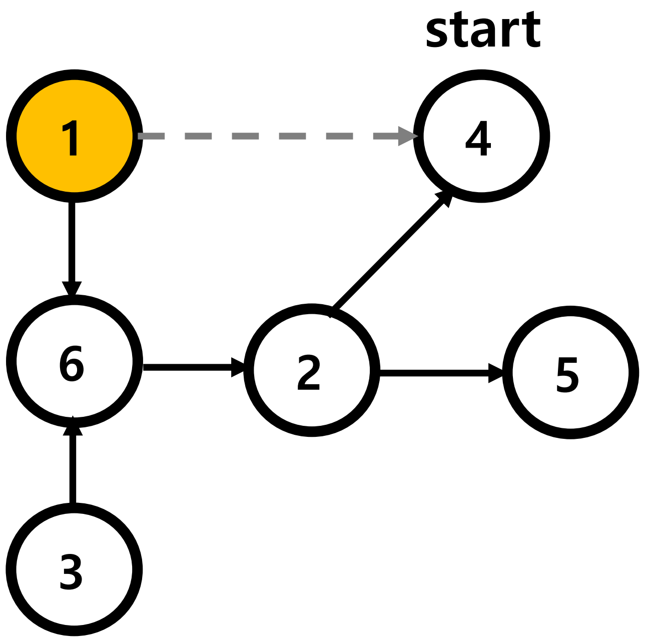


Stack

Visited

1					

- Algorithm (Topological DFS)

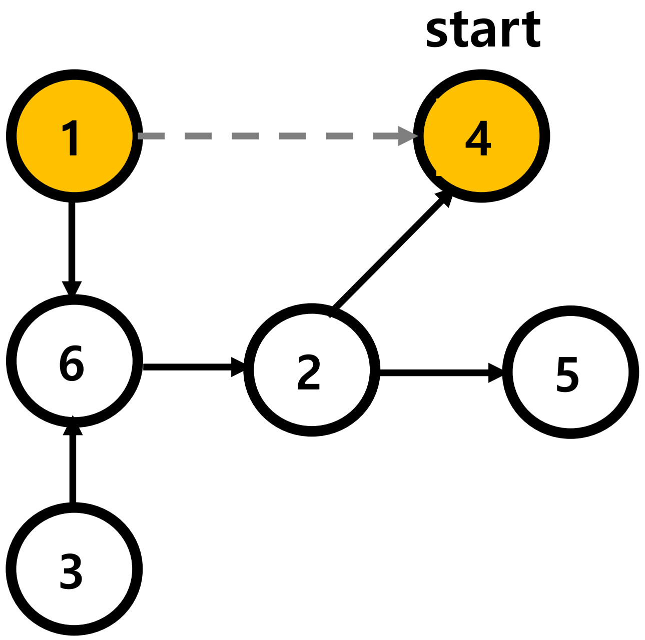


Stack

Visited

1					

- Algorithm (Topological DFS)



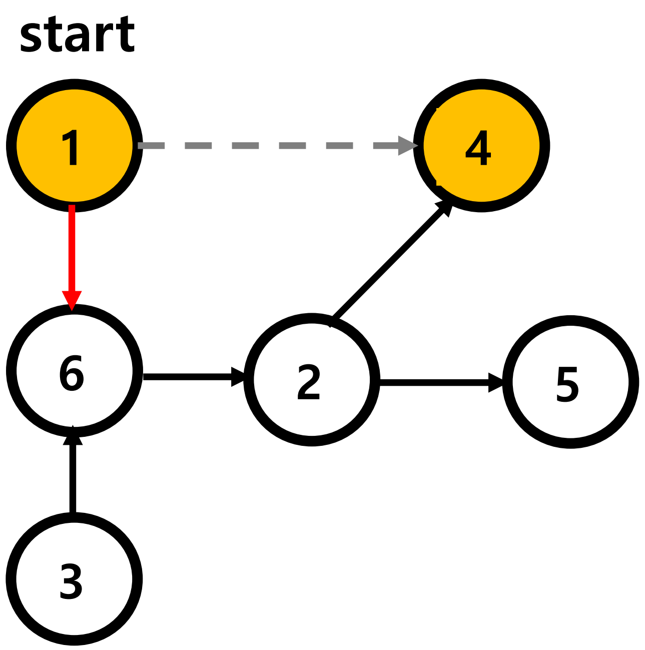
Stack

4					

Visited

1			4		

- Algorithm (Topological DFS)



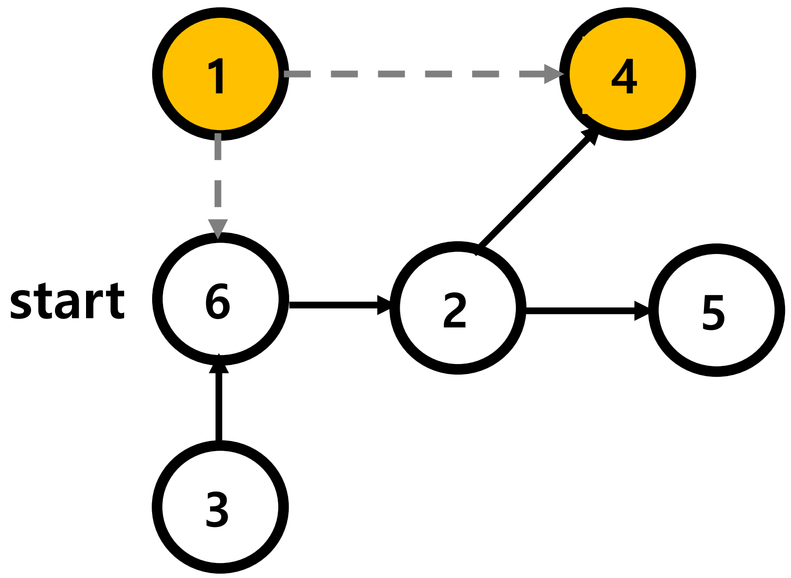
Stack

4					

Visited

1			4		

- Algorithm (Topological DFS)



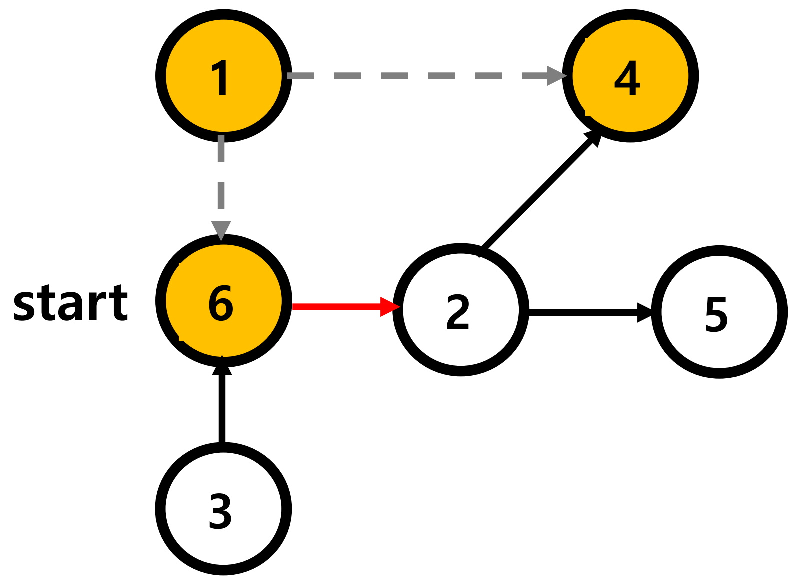
Stack

4					

Visited

1			4		

- Algorithm (Topological DFS)



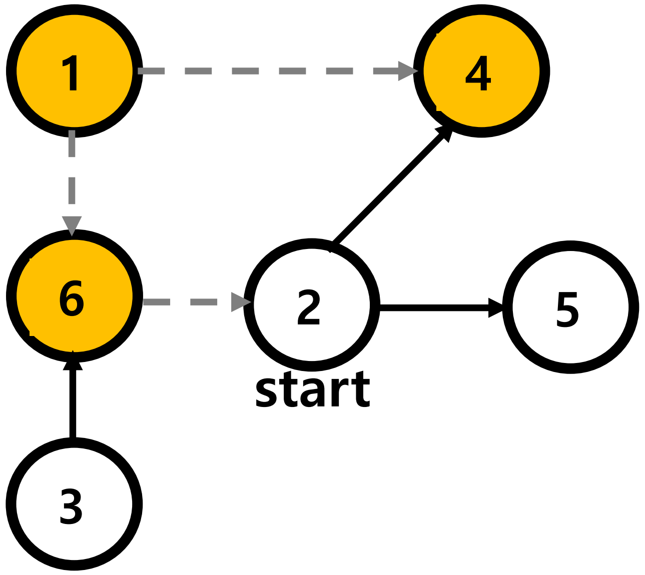
Stack

4					

Visited

1			4		6

- Algorithm (Topological DFS)



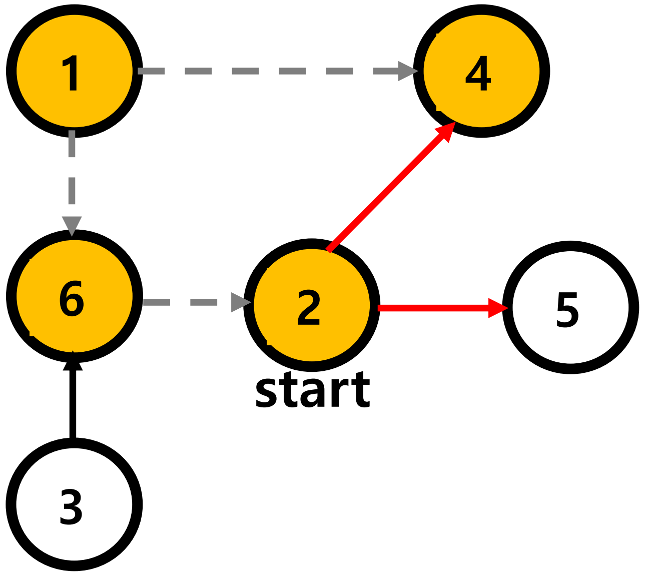
Stack

4					

Visited

1			4		6

- Algorithm (Topological DFS)



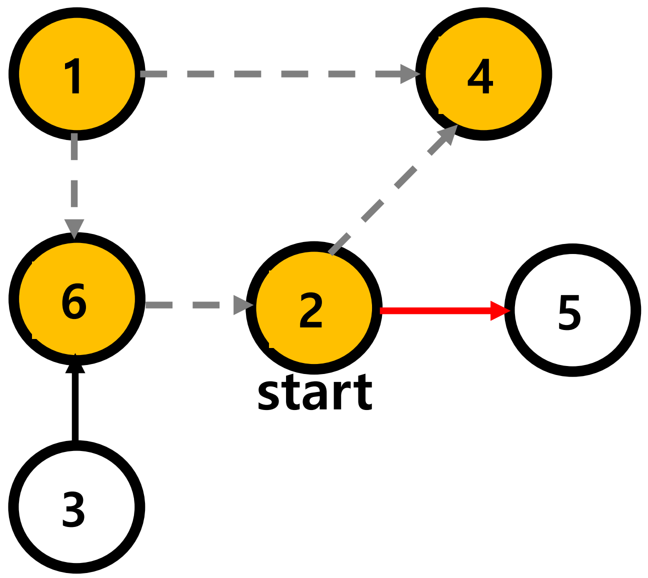
Stack

4					

Visited

1	2		4		6

- Algorithm (Topological DFS)



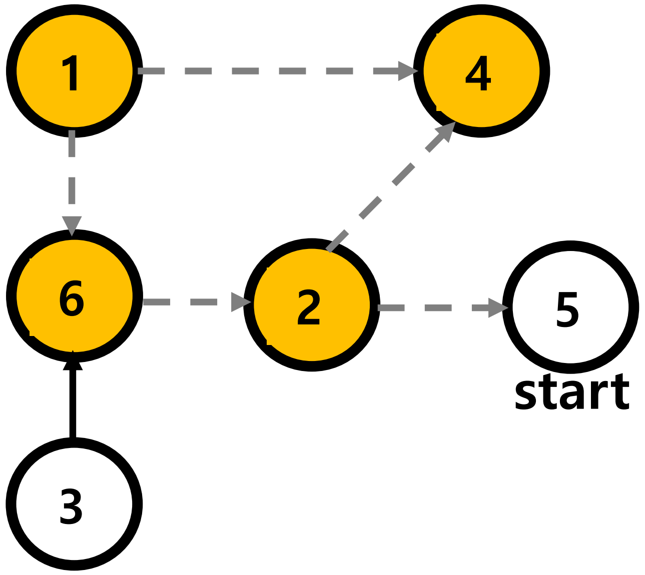
Stack

4					

Visited

1	2		4		6

- Algorithm (Topological DFS)



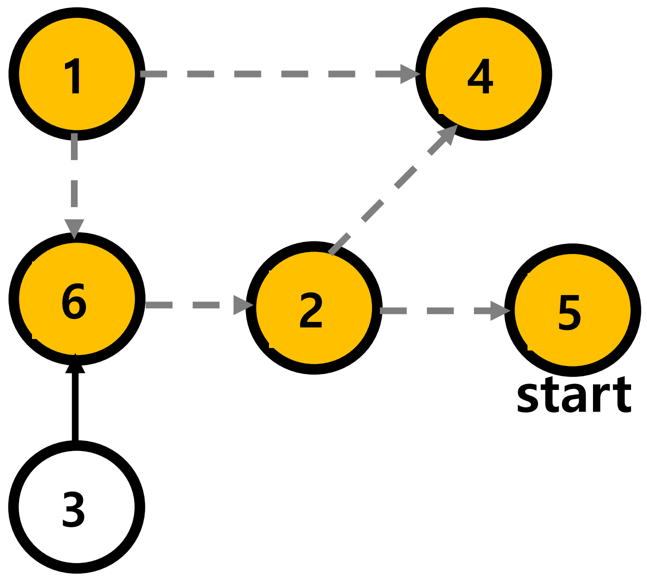
Stack

4					

Visited

1	2		4		6

- Algorithm (Topological DFS)



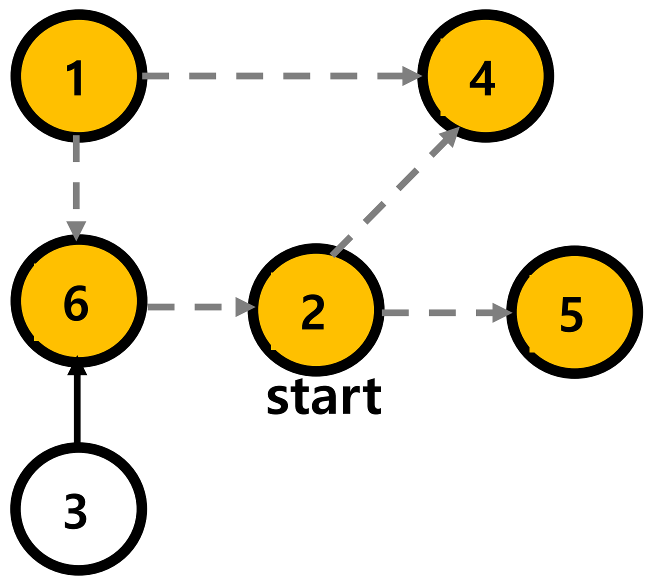
Stack

4					

Visited

1	2		4	5	6

- Algorithm (Topological DFS)



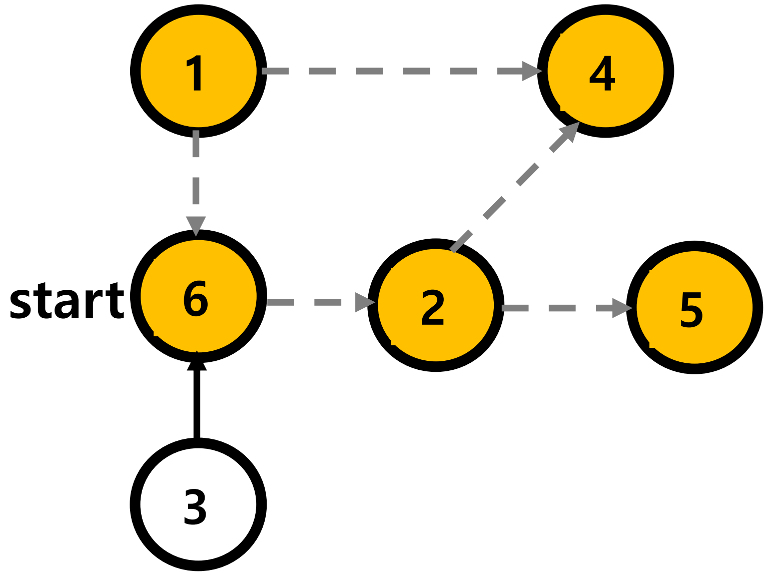
Stack

4	5				

Visited

1	2		4	5	6

- Algorithm (Topological DFS)



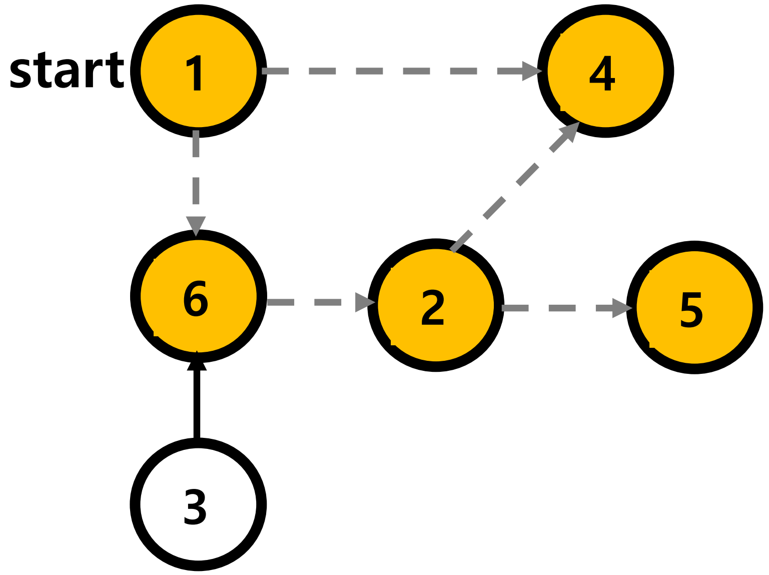
Stack

4	5	2			

Visited

1	2		4	5	6

- Algorithm (Topological DFS)



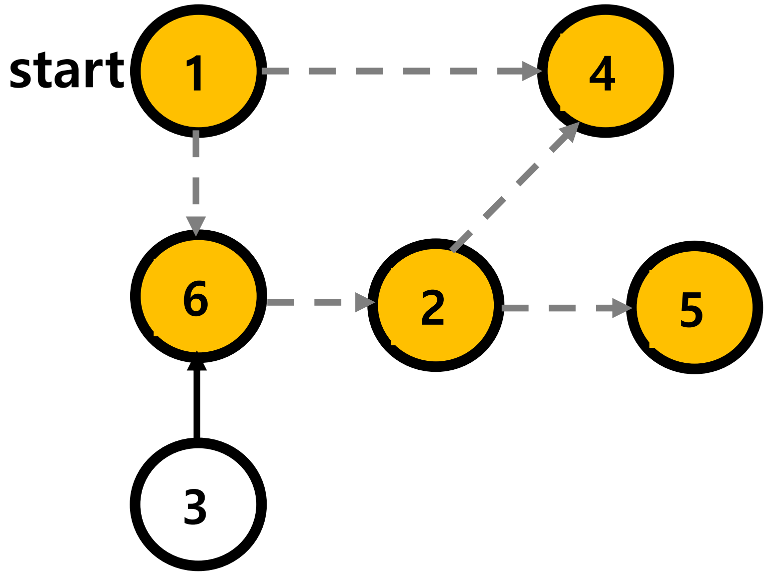
Stack

4	5	2	6		

Visited

1	2		4	5	6

- Algorithm (Topological DFS)



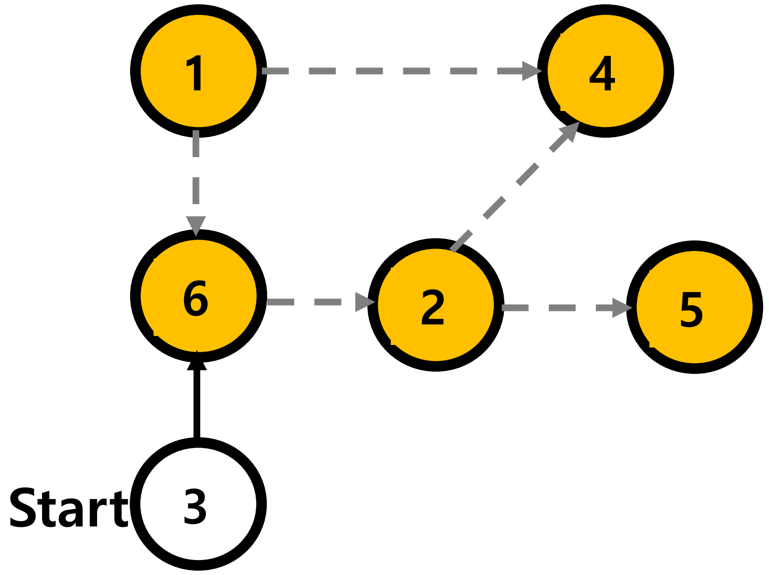
Stack

4	5	2	6	1	

Visited

1	2		4	5	6

- Algorithm (Topological DFS)



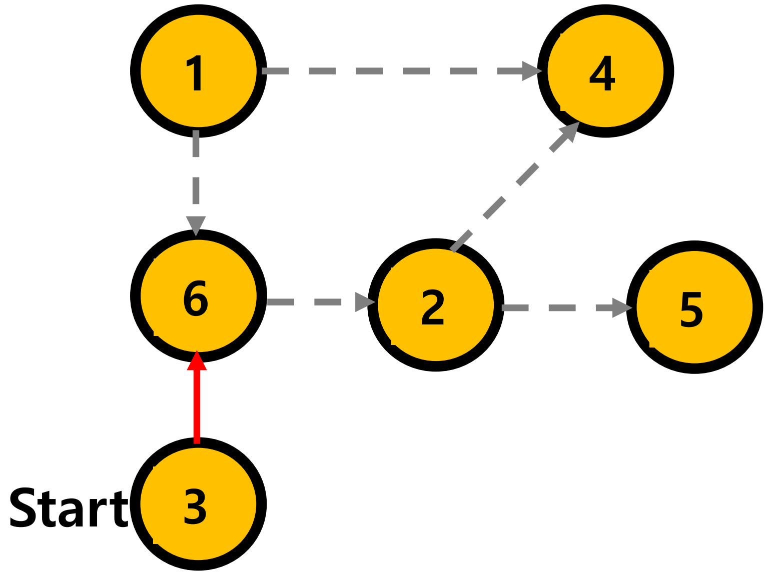
Stack

4	5	2	6	1	

Visited

1	2		4	5	6

- Algorithm (Topological DFS)



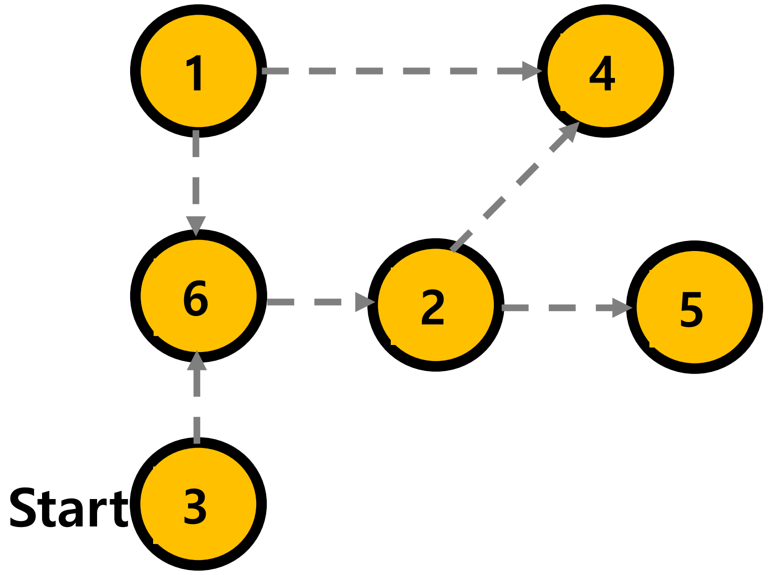
Stack

4	5	2	6	1	

Visited

1	2	3	4	5	6

- Algorithm (Topological DFS)



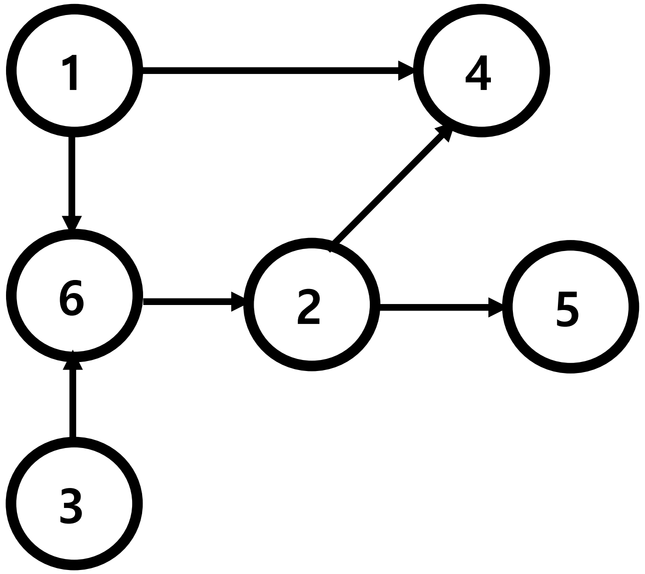
Stack

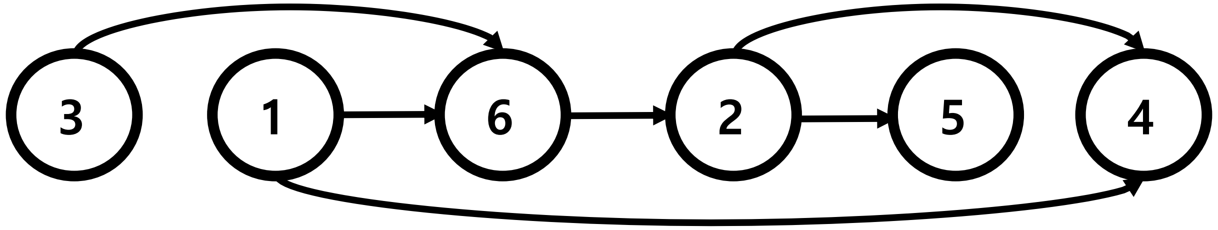
4	5	2	6	1	3

Visited

1	2	3	4	5	6

- Algorithm (Topological DFS)





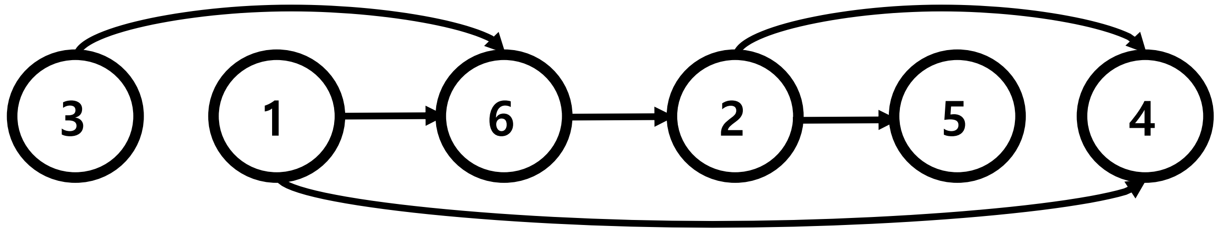
Stack

4	5	2	6	1	3

sorting

3	1	6	2	5	4

- Analysis



Adjacent list : $\mathbf{O(V+E)}$

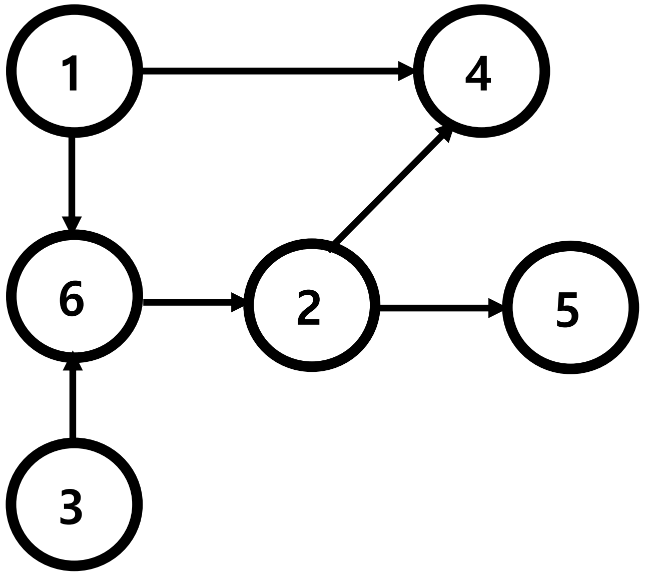
$\boldsymbol{(V\ is\ vetex\ and\ E\ is\ edge\ )}$

- Topological sort (BFS)

```
void Topological sort(){

    for(int i=1; i<=n; i++){
        if(indegree[i]==0){
            q.push(i);
        }
    }
    bfs();
}
```

```
void bfs(){

    while(!q.empty()){
        int now = q.front();
        visited[now] = true; //checked now vertex
        q.pop();
        cout << now << " ";
        for(each vertex v in adjacent now){
            indegree[v]--;
            if(visited[v]==false && indegree[v]==0){
                q.push(v);
            }
        }
    }
}
```

- Algorithm (Topological BFS)



Indegree

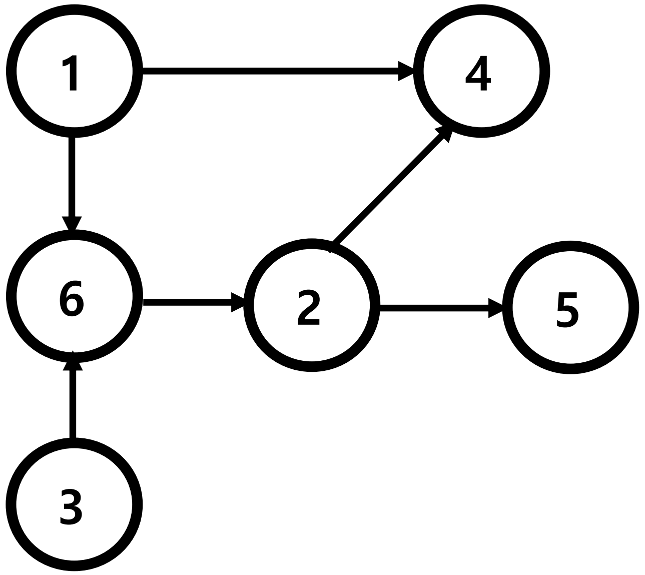
1	2	3	4	5	6
0	**1**	**0**	**2**	**1**	**2**

Queue

Visited

Sorting

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	1	0	2	1	2

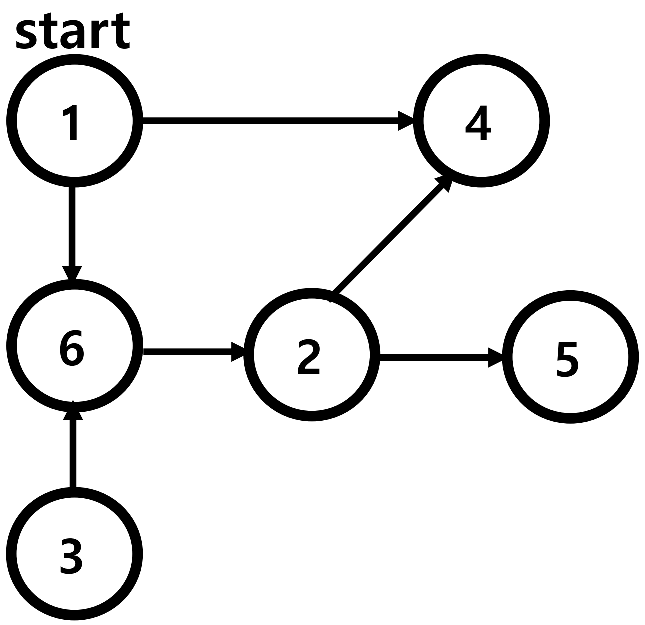
Queue

1	3				

Visited

Sorting

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	**1**	**0**	**2**	**1**	**2**

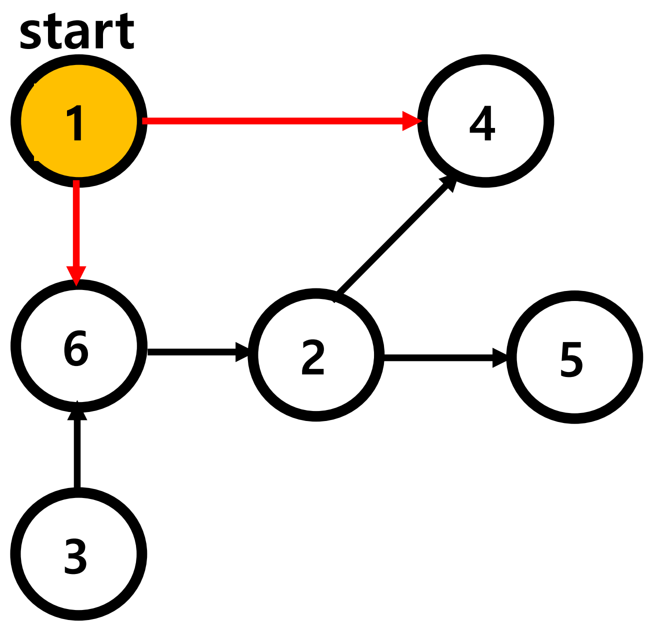
Queue

3					

Visited

Sorting

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	**1**	**0**	**2**	**1**	**2**

Queue

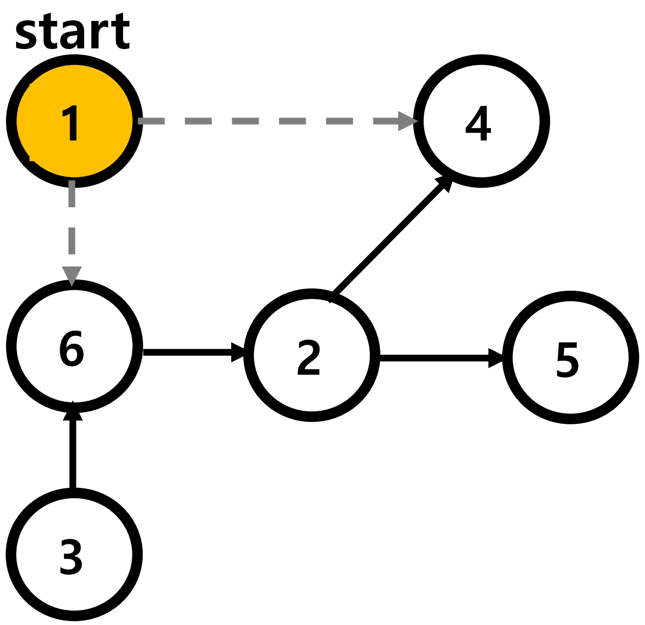
3					

Visited

1					

Sorting

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	**1**	**0**	**1**	**1**	**1**

Queue

3					

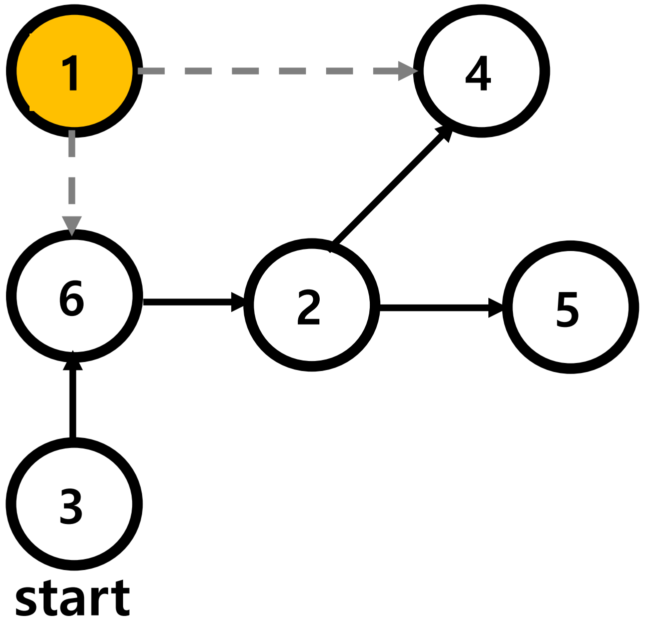
Visited

1					

Sorting

1					

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	**1**	**0**	**1**	**1**	**1**

Queue

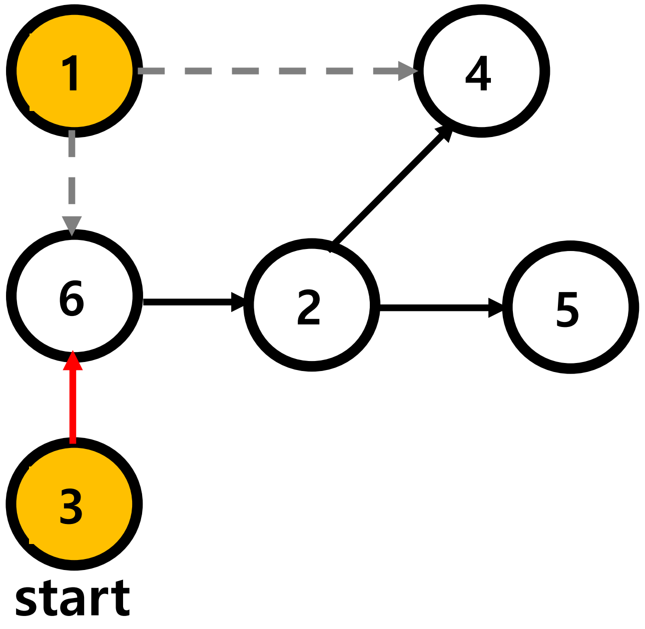
Visited

1					

Sorting

1					

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	**1**	**0**	**1**	**1**	**1**

Queue

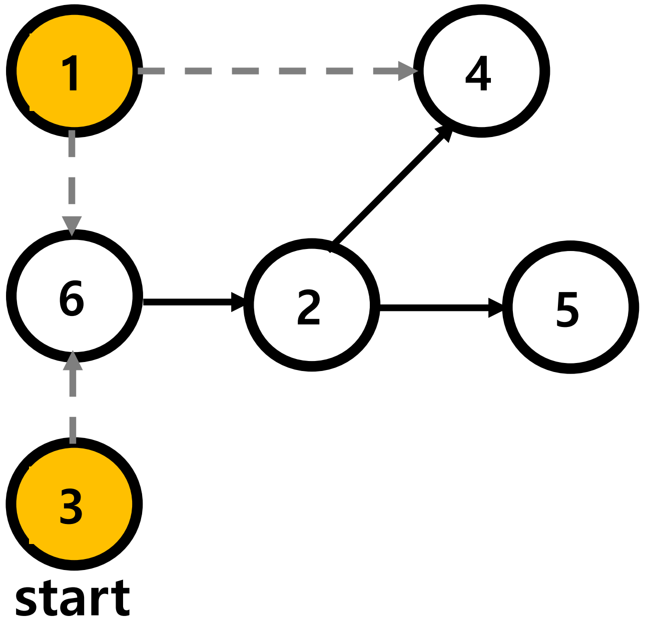
Visited

1		**3**			

Sorting

1					

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	1	0	1	1	0

Queue

6					

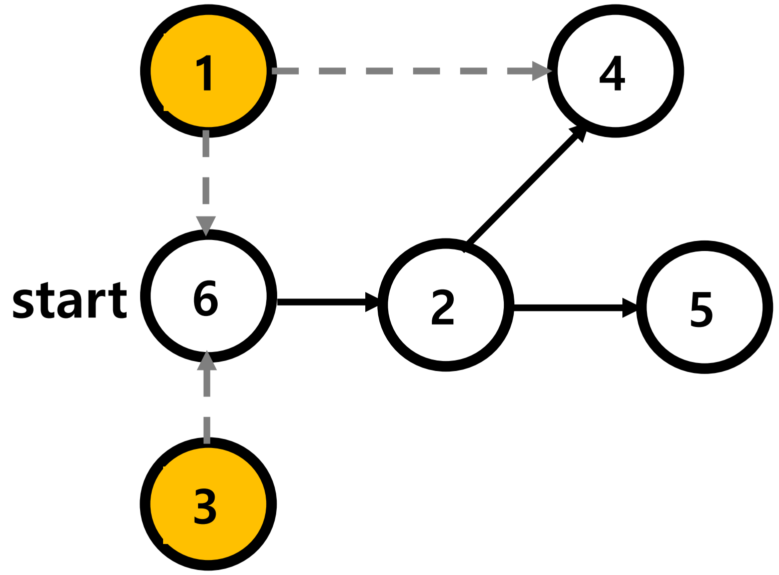
Visited

1		3			

Sorting

1	3				

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	1	0	1	1	0

Queue

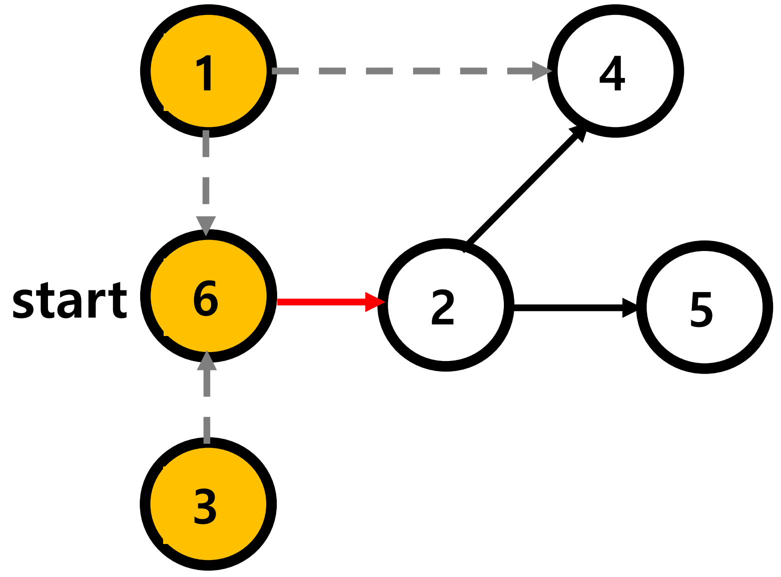
Visited

1		3			

Sorting

1	3				

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	**1**	**0**	**1**	**1**	**0**

Queue

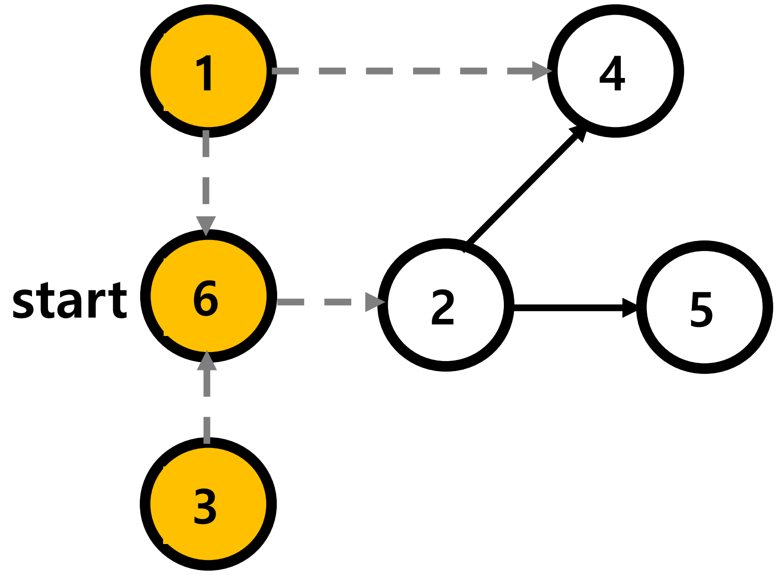
Visited

1		**3**			**6**

Sorting

1	**3**				

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	0	0	1	1	0

Queue

2					

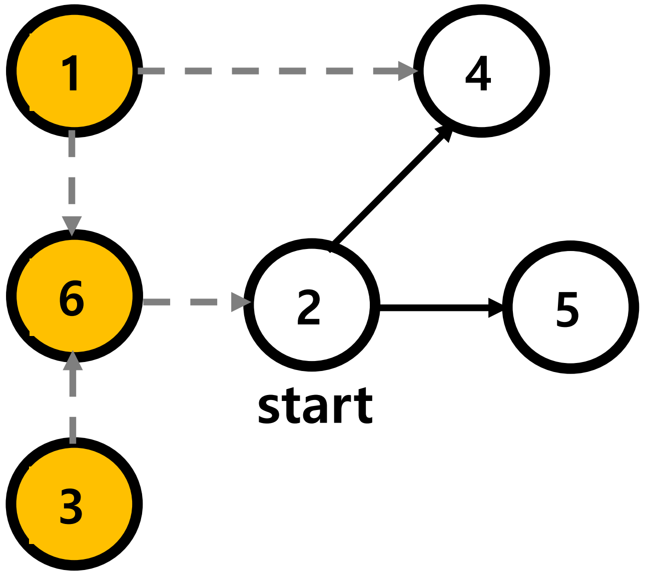
Visited

1		3			6

Sorting

1	3	6			

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	**0**	**0**	**1**	**1**	**0**

Queue

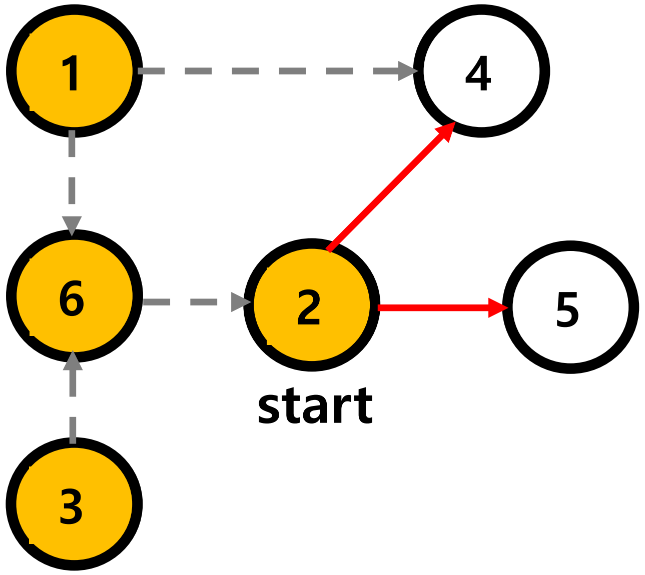
Visited

1		**3**			**6**

Sorting

1	**3**	**6**			

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	**0**	**0**	**1**	**1**	**0**

Queue

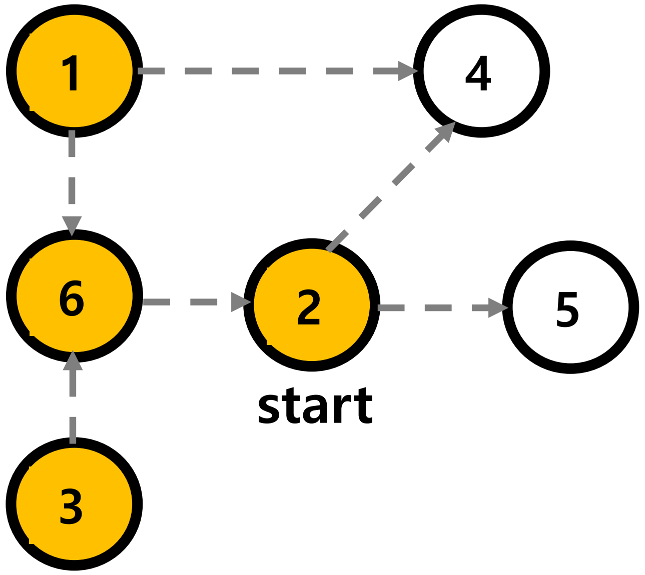
Visited

1	**2**	**3**			**6**

Sorting

1	**3**	**6**			

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	0	0	0	0	0

Queue

4	5				

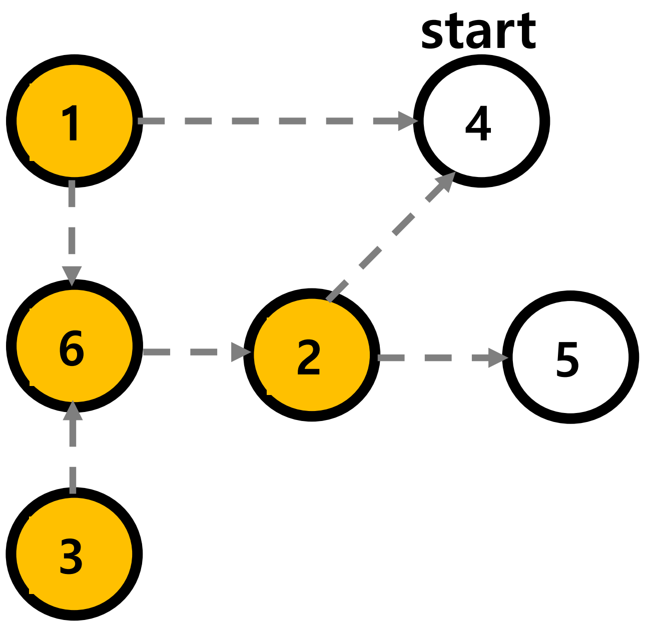
Visited

1	2	3			6

Sorting

1	3	6	2		

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	**0**	**0**	**0**	**0**	**0**

Queue

5					

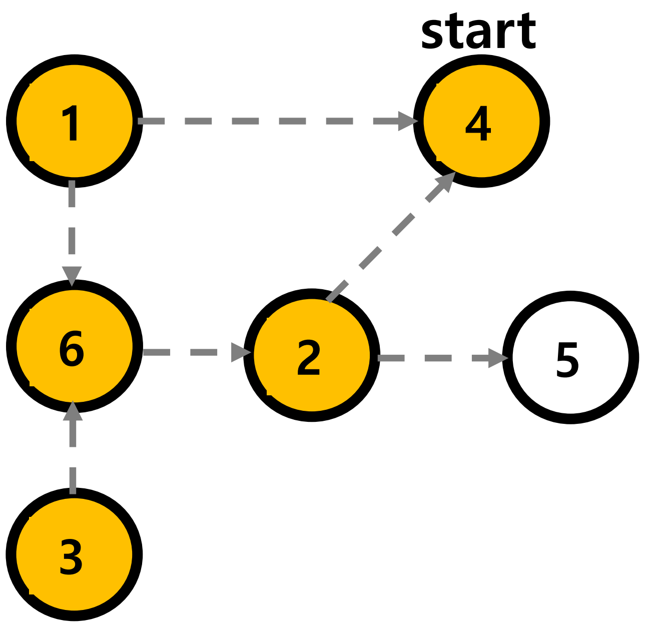
Visited

1	**2**	**3**			**6**

Sorting

1	**3**	**6**	**2**		

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	0	0	0	0	0

Queue

5					

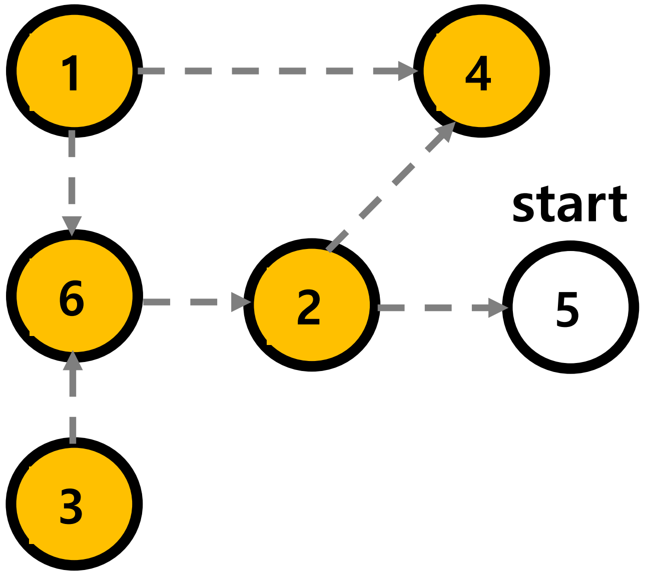
Visited

1	2	3	4		6

Sorting

1	3	6	2	4	

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	**0**	**0**	**0**	**0**	**0**

Queue

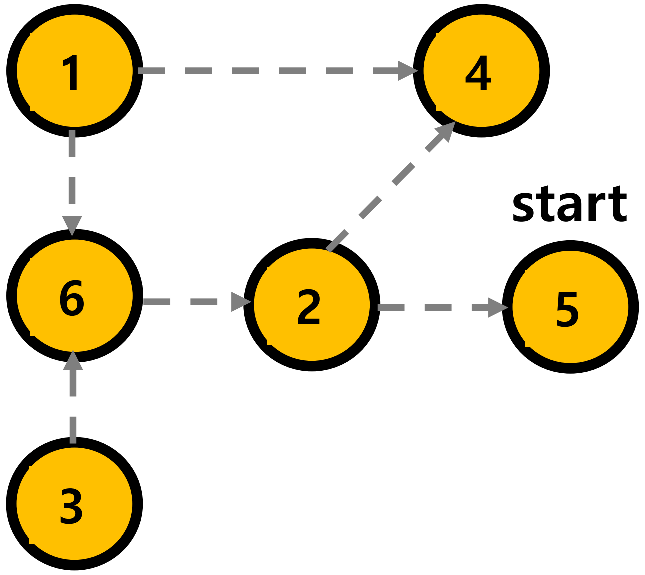
Visited

1	**2**	**3**	**4**		**6**

Sorting

1	**3**	**6**	**2**	**4**	

- Algorithm (Topological BFS)



Indegree

1	2	3	4	5	6
0	0	0	0	0	0

Queue

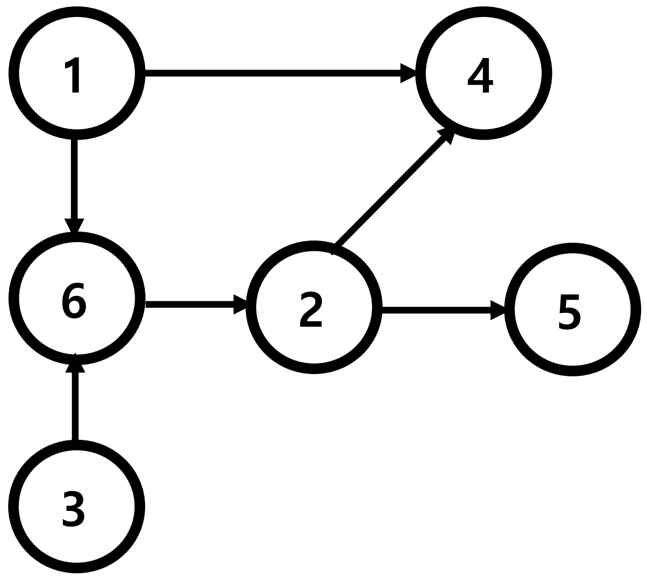
Visited

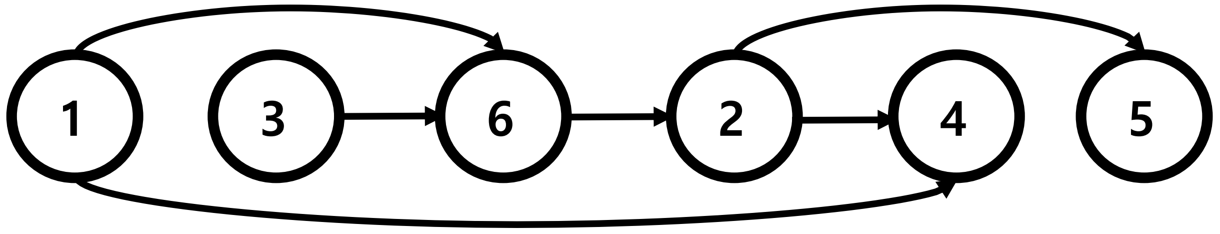
1	2	3	4	5	6

Sorting

1	3	6	2	4	5

- Algorithm (Topological BFS)

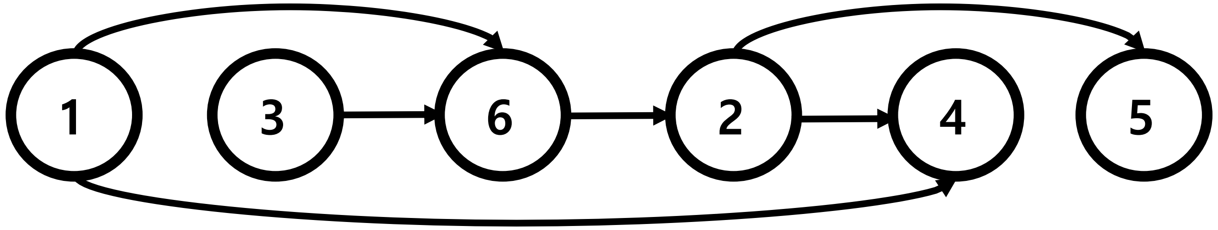




Sorting

1	3	6	2	4	5

- Analysis



Adjacent list : $\mathbf{O}(\boldsymbol{V}+\boldsymbol{E})$

$\boldsymbol{(V\ is\ vetex\ and\ E\ is\ edge\ )}$

4. Conclusion

Summary

1. If the graph is ‘DAG’, we can make topological sort by using DFS, BFS.

2. DFS and BFS requires O$(V + E)$ time.

3. The results can appear in many ways.

4. When execute topological sort, you may need some data structures such as stack or queue.

5. No matter what method you choose, the sorting runs O$(V + E)$ time.

Q & A

Thank you!