Graphs Theory
Shortest Path Problem(5)

University of M'sila
Faculty of Mathematics and informatics
Mathematics department

Speciality:

Applied Mathematics

L3

2024/2025

MOHAMED BAHACHE

Problematic

- Let G=(V,E) be a directed graph.

We consider a function $f : E \rightarrow R$

This function assigns a real value to each edge

$e_i \in E$

This real value is called cost.

Let the path p= <v1,v2,….vk>

The cost or distance of p is :

$$Cost(p) = \sum_{i=1}^{k} Cost(vi, vi+1)$$

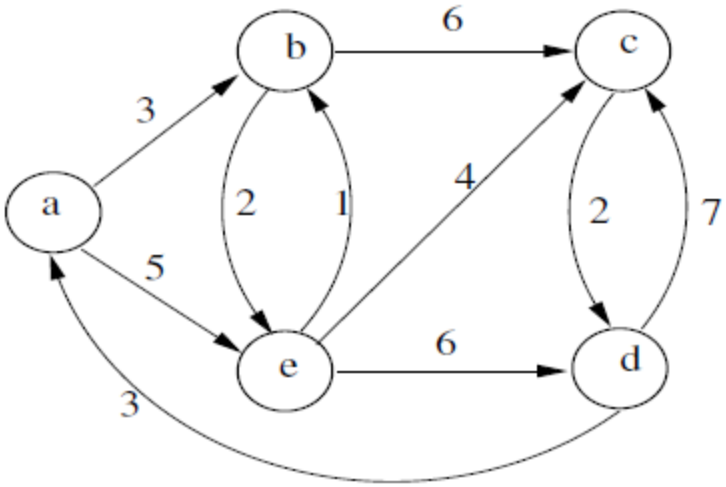
Problematic

- So we can define the weight of the shortest path between two vertices v1 and v2 as:

$$w(v1, v2) = \begin{cases} \min\{\cos t(p)\} \\ +\infty, \text{ if } \nexists \text{ path from v1 and v2} \end{cases}$$

Problematic

- Example : we consider the following directed graph:

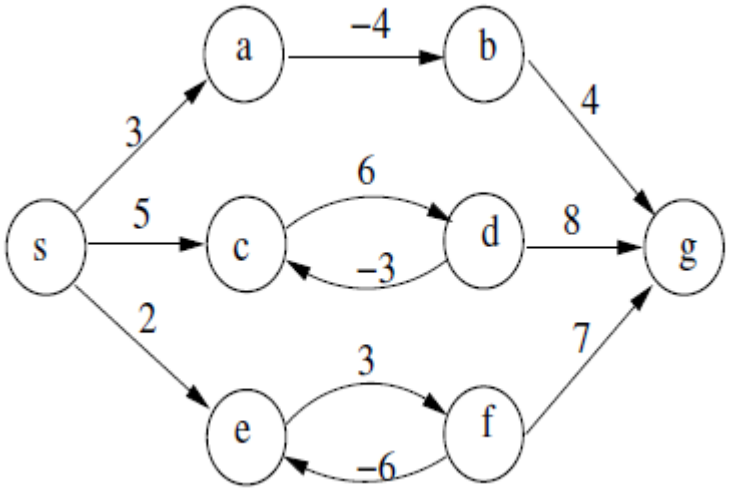


- w(a,b)=3,w(a,e)=5,w(a,c)=9

Problematic

❑Condition of existence : to search the shortest path in a directed graph , this graph must be without negative circuits, means Cost(circuits)<0, then w(shortest path)=-∞

The path < s; e; f; e; f; g >
contains the circuit < e; f; e >
has a negative cost -3.
In other words , each time
Pass through this circuit
we decrase by 3 the total cost



Thus w(s; g) =- ∞

Problematic

- Dijkstra's Algorithm
  - Use greedy search as strategy,
  - Well for graphs where the Wight are positives or nuls.
- Bellman-Ford's Algorithm
  - Use greedy search as strategy,
  - Solve the problem with negative, nul, or positive weights.
- Remark : With insurance that our graph does not contains a negative circuits.

Shortest Path

- Uses greedy search as strategy,
- The idea is the same attribute a maximum value to each vertex.
- Initially :
- $w(v_0) = 0 \quad w(v_i) = +\infty$ for each $v_i \neq v_0$

- And when we found a better choice we update, this what is called relaxing

Shortest Path

- Relaxing procedure

Procedure relaxing
begin
if $w(v_j) > w(v_i) + d(v_i, v_j)$ then
    $w(v_j) = w(v_i) + d(v_i, v_j)$
    $\pi(v_j) \leftarrow v_i$
endif
end

Dijkstra's algorithm

- With Dijkstra's algorithm: each edge is relaxed one time.
- But with Bellman-Ford's Algorithm, one edge may be relaxed several times.

- $\pi(v_i)$: for saving the predecessor vertex of the actual one.

Dijkstra's algorithm

- Let $G = (V, E)$ be an directed graph
- Let $v_0 \in V$

- Let E, F two sets such that:

- Initialization:

$$w(v_0) = 0, w(v_i) = +\infty (i \neq 0), \pi(v_i) = nil, F \leftarrow V, E \leftarrow \phi$$

Dijkstra's algorithm

- Dijkstra's algorithm

begin

while($F \neq \phi$)

let v_i the vertex has the minimum weigth

$F \leftarrow F - \{v_i\}$

$E \leftarrow E \cup \{v_i\}$

relax(v_i, v_j), v_j is a successor of v_i

endwhile

return π

end

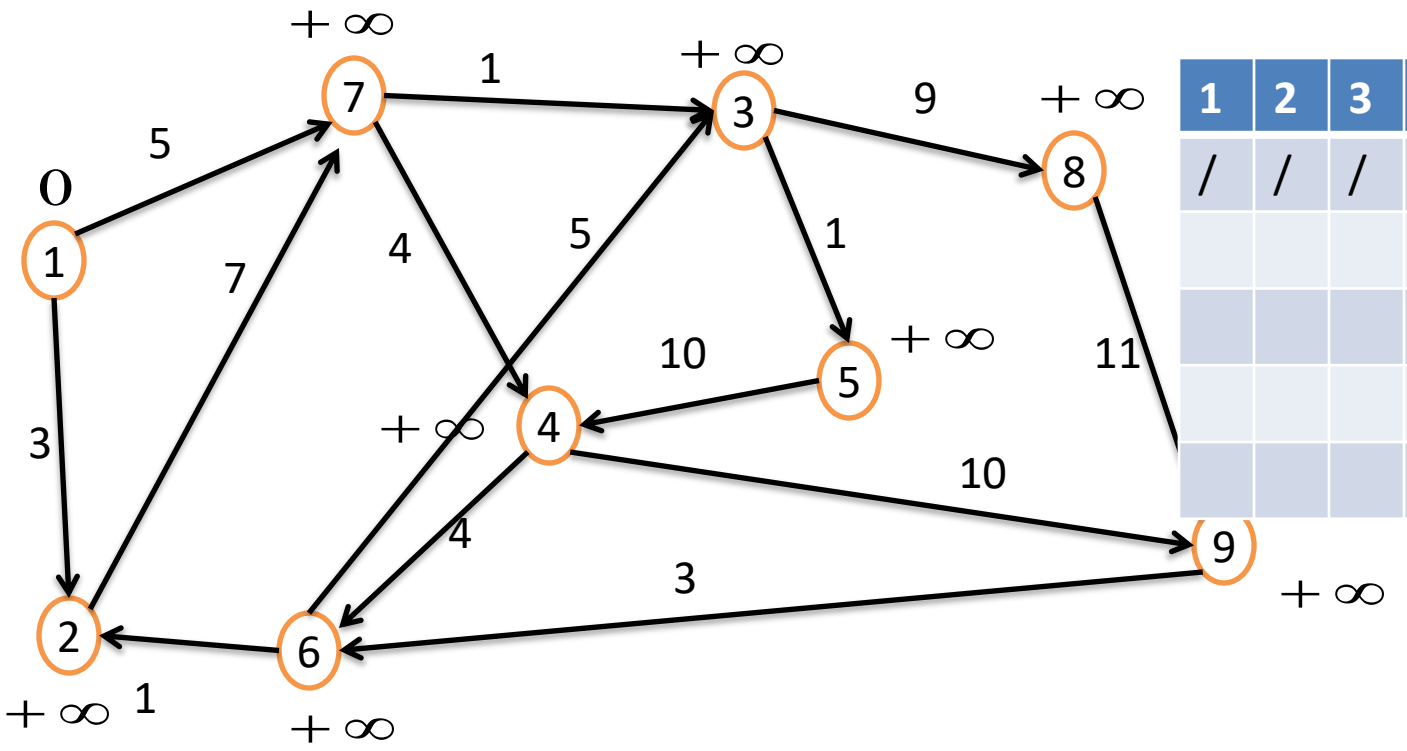
Dijkstra's algorithm

- Dijkstra's algorithm complexity

$$O(n^2), \text{ if represented by adjacency matrix}$$

Dijkstra's algorithm Example(initialization)

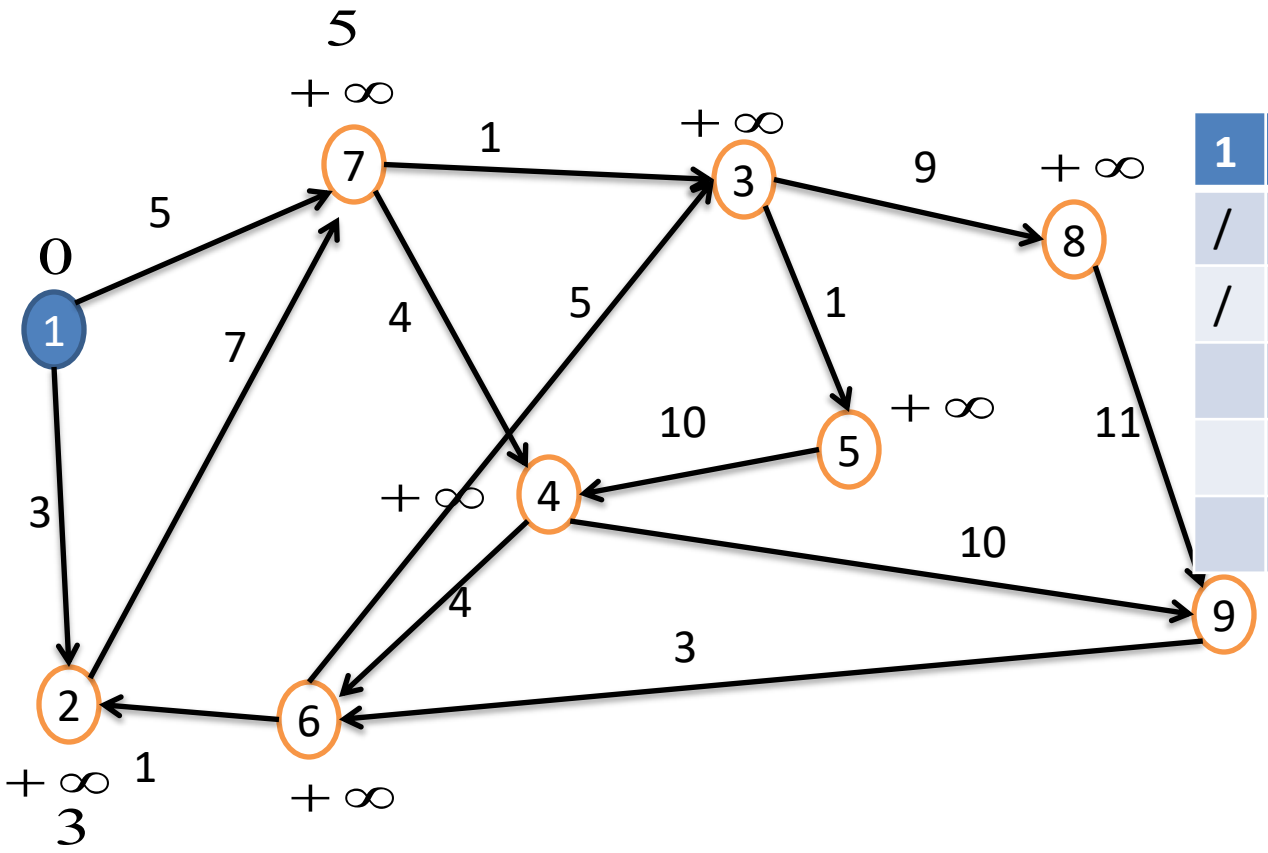
$$E = \phi, F = \{1,2,3,4,5,6,7,8,9\}$$



1	2	3	4	5	6	7	8	9
/	/	/	/	/	/	/	/	/

Dijkstra's algorithm Example(iteration1)

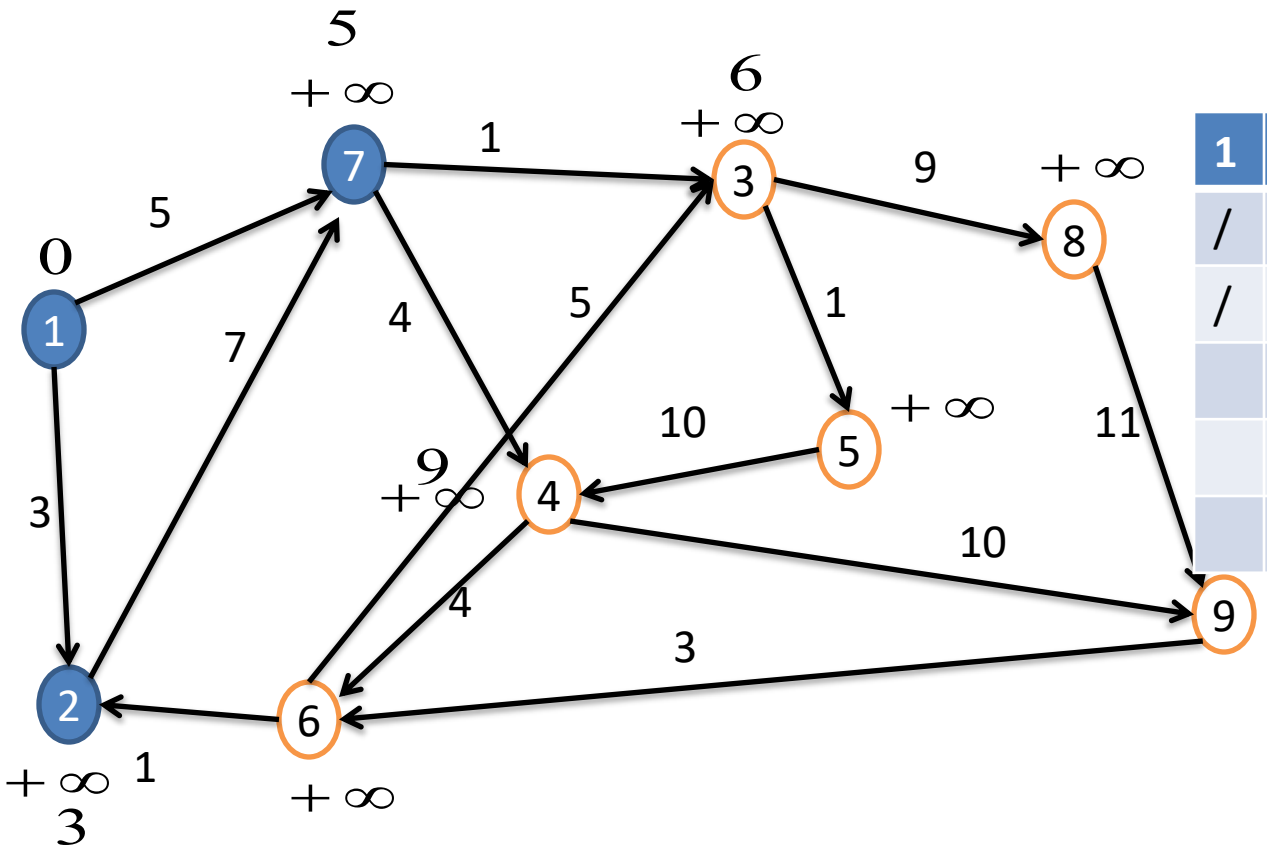
$$E = \{1\}, F = \{2,3,4,5,6,7,8,9\}$$



1	2	3	4	5	6	7	8	9
/	/	/	/	/	/	/	/	/
/	1					1		

Dijkstra's algorithm Example(iteration2+3)

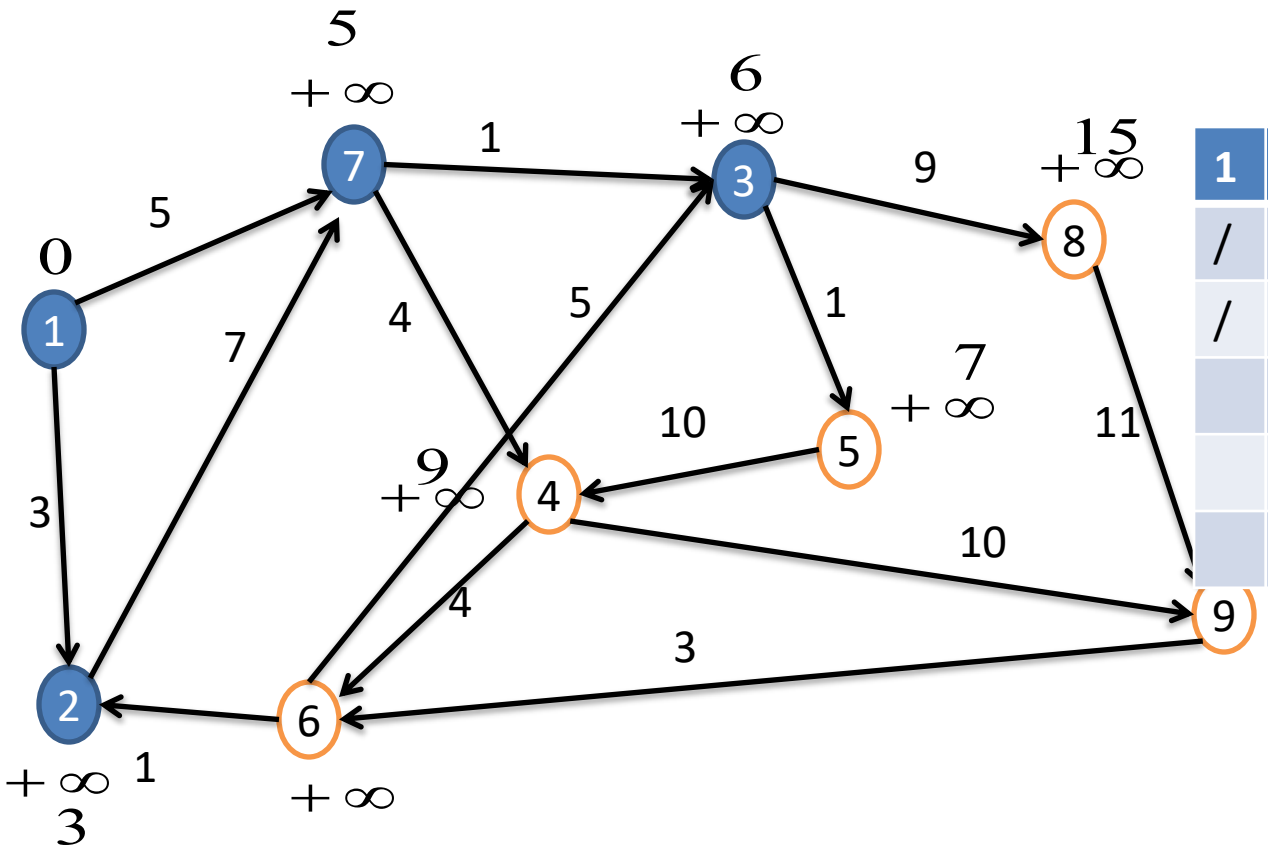
$$E = \{1,2,7\}, F = \{3,4,5,6,8,9\}$$



1	2	3	4	5	6	7	8	9
/	/	/	/	/	/	/	/	/
/	1					1		
		7	7					

Dijkstra's algorithm Example(iteration4)

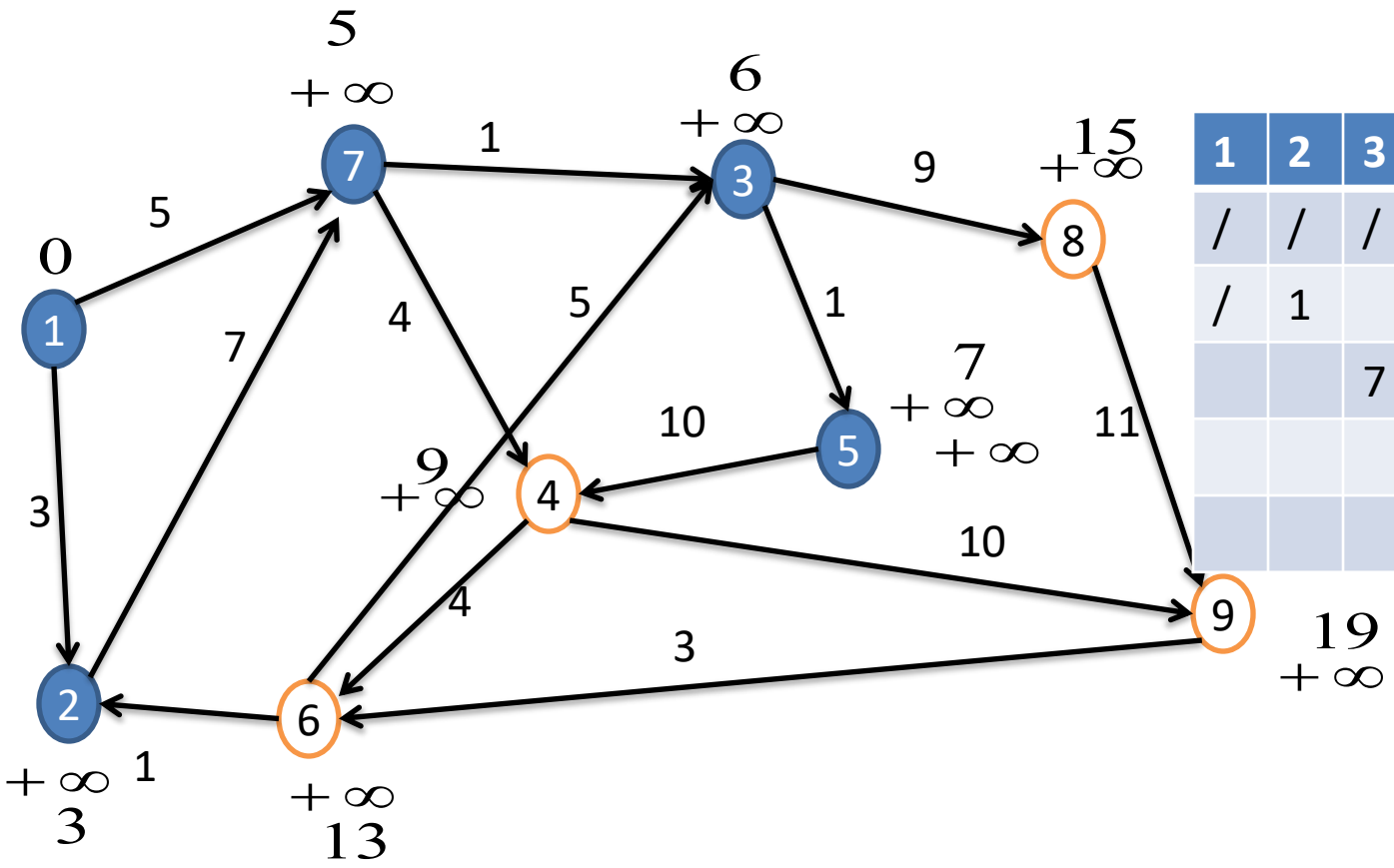
$$E = \{1,2,7,3\}, F = \{4,5,6,8,9\}$$



1	2	3	4	5	6	7	8	9
/	/	/	/	/	/	/	/	/
/	1					1		
		7	7					
				3			3	

Dijkstra's algorithm Example(iteration5)

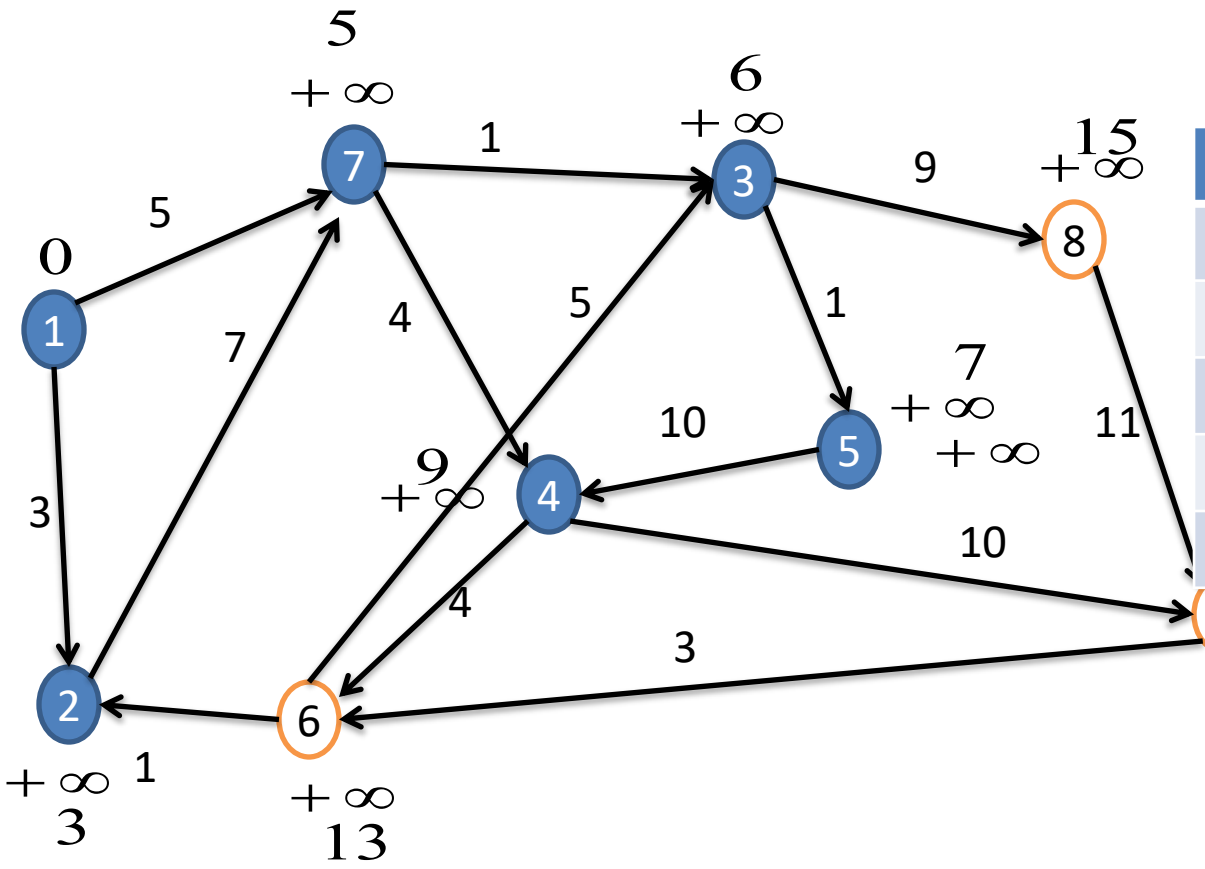
$$E = \{1,2,7,3,5\}, F = \{4,6,8,9\}$$



1	2	3	4	5	6	7	8	9
/	/	/	/	/	/	/	/	/
/	1					1		
		7	7					
				3			3	
					4			4

Dijkstra's algorithm Example(iteration6)

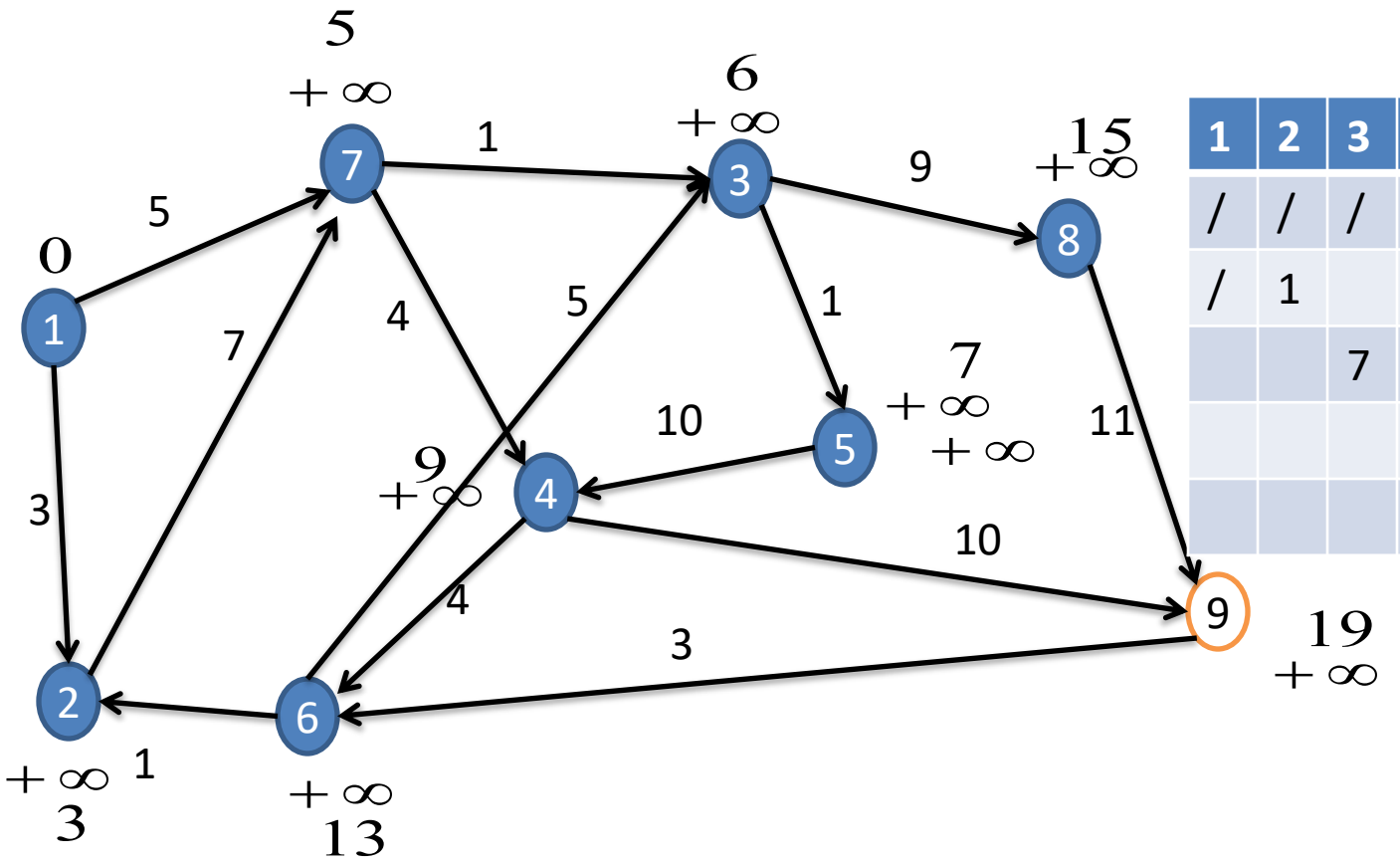
$$E = \{1,2,7,3,5,4\}, F = \{6,8,9\}$$



1	2	3	4	5	6	7	8	9
/	/	/	/	/	/	/	/	/
/	1					1		
		7	7					
				3			3	
					4			4

Dijkstra's algorithm Example (iteration7)

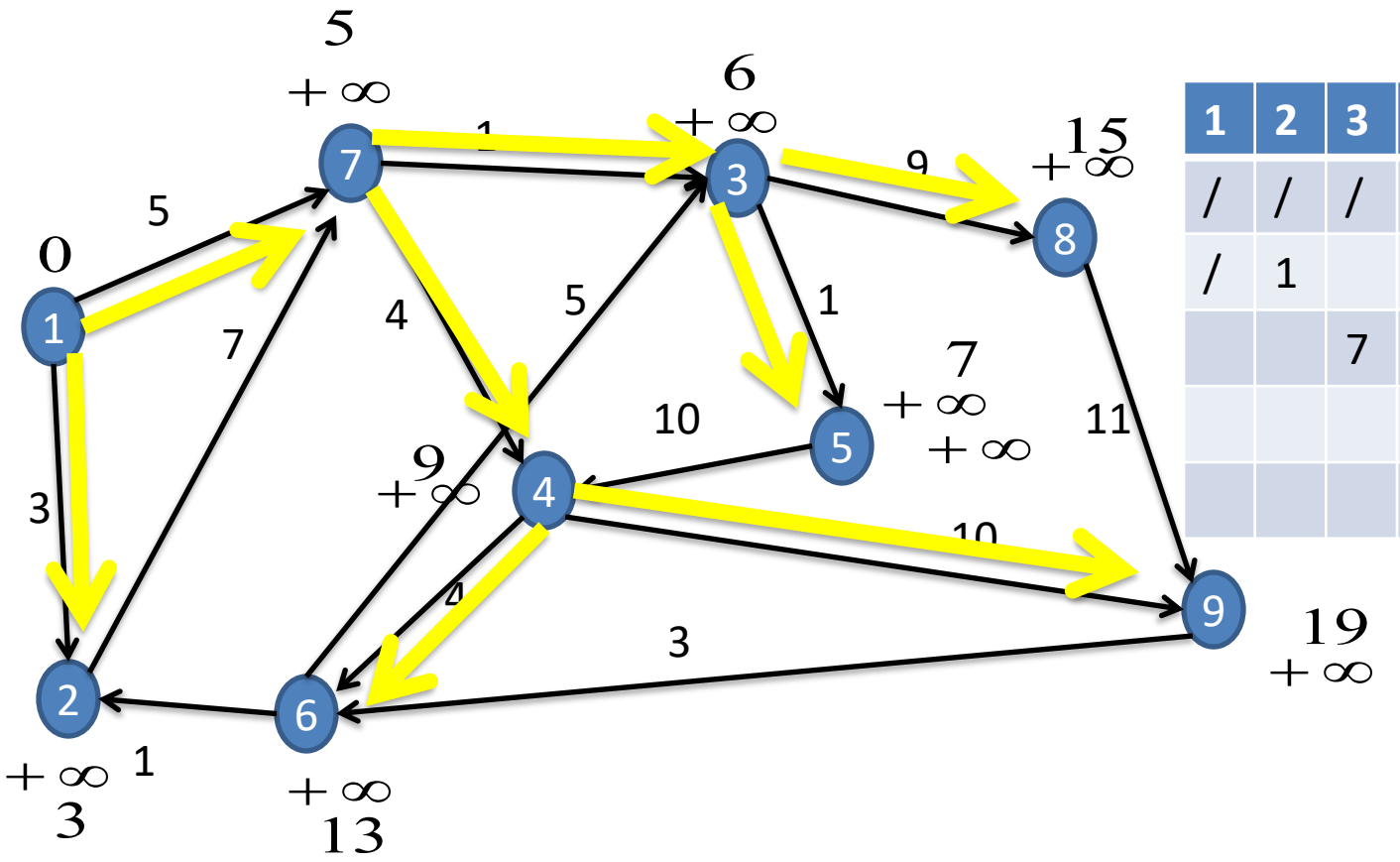
$$E = \{1,2,7,3,5,4,6,8\}, F = \{9\}$$



1	2	3	4	5	6	7	8	9
/	/	/	/	/	/	/	/	/
/	1					1		
		7	7					
				3			3	
					4			4

Dijkstra's algorithm Example(iteration8)

$$E = \{1,2,7,3,5,4,6,8,9\}, F = \phi$$



1	2	3	4	5	6	7	8	9
/	/	/	/	/	/	/	/	/
/	1					1		
		7	7					
				3			3	
					4			4

Bellman-Ford's algorithm

- Bellman-Ford's algorithm
- Unlike Dijkstra Algorithm, where each edge is relaxed one time,
- Here each edge is relaxed n-1 times ,
- Complexity

$$O(n^3), \text{if represented by adjacency matrix}$$

- Initialization

$$w(v_0) = 0, w(v_i) = +\infty (i \neq 0), \pi(v_i) = nil.$$

Bellman-Ford's algorithm

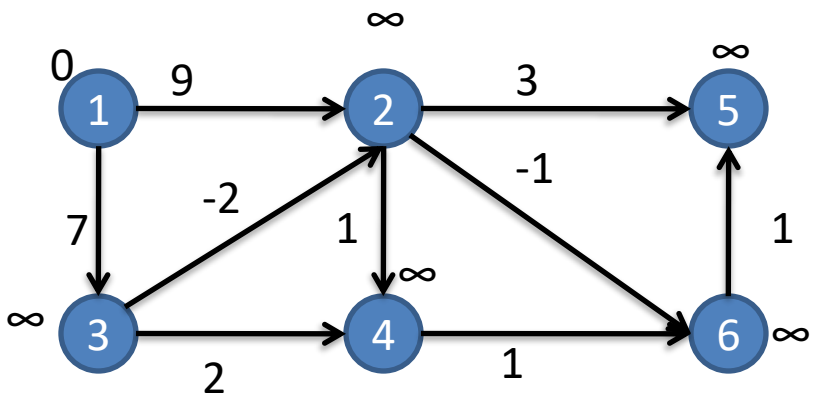
```
begin
For(k = 1 to |V| - 1)
  for each edge (u, v) ∈ E
   relax(u, v)
   endfor
endfor
  for  each edge (u, v) ∈ E
   if  w(v) > w(u) + w(u, v) then
     write("negative circuit") endif
 endfor
return π
end
```

Edges Relaxing

Negative Circuit existence check

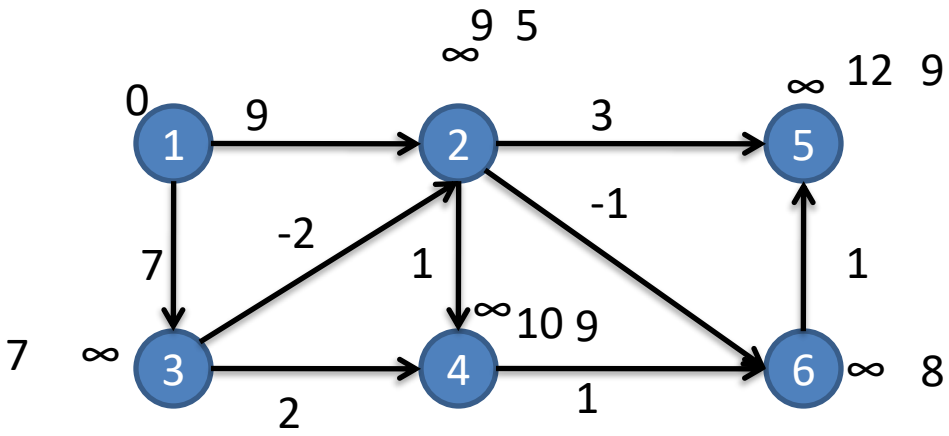
(1,2),(1,3),(2,4),(2,5),(2,6),(3,2),(3,4),(4,6),(6,5)

1	2	3	4	5	6
/	/	/	/	/	/



	1	2	3	4	5	6
initial	0	∞	∞	∞	∞	∞
It1						
It2						
It3						
It4						
It5						

(1,2),(1,3),(2,4),(2,5),(2,6),(3,2),(3,4),(4,6),(6,5)

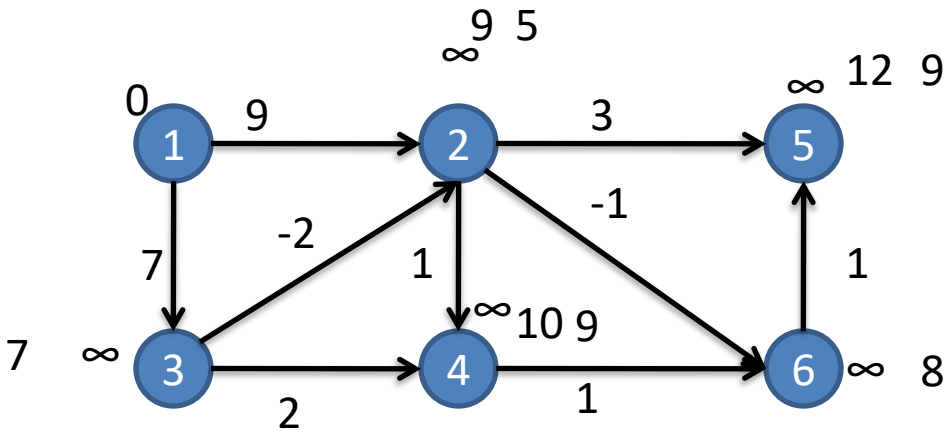


1	2	3	4	5	6
/	/	/	/	/	/
	3	1	3	6	2

	1	2	3	4	5	6
initial	0	∞	∞	∞	∞	∞
It1						
It2						
It3						
It4						
It5						

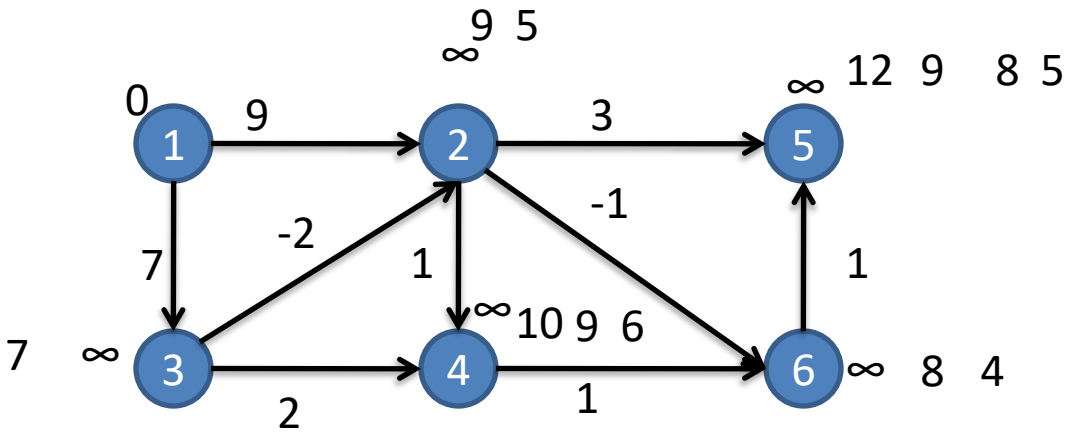
(1,2),(1,3),(2,4),(2,5),(2,6),(3,2),(3,4),(4,6),(6,5)

1	2	3	4	5	6
/	/	/	/	/	/
	3	1	3	6	2



	1	2	3	4	5	6
initial	0	∞	∞	∞	∞	∞
It1	0	5	7	9	9	8
It2						
It3						
It4						
It5						

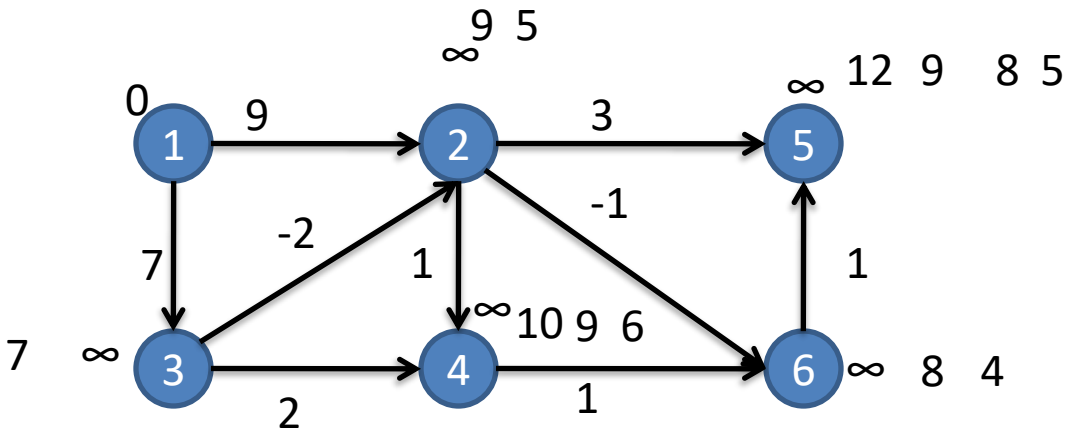
(1,2),(1,3),(2,4),(2,5),(2,6),(3,2),(3,4),(4,6),(6,5)



1	2	3	4	5	6
/	/	/	/	/	/
	3	1	2	6	2

	1	2	3	4	5	6
initial	0	∞	∞	∞	∞	∞
It1	0	5	7	9	9	8
It2						
It3						
It4						
It5						

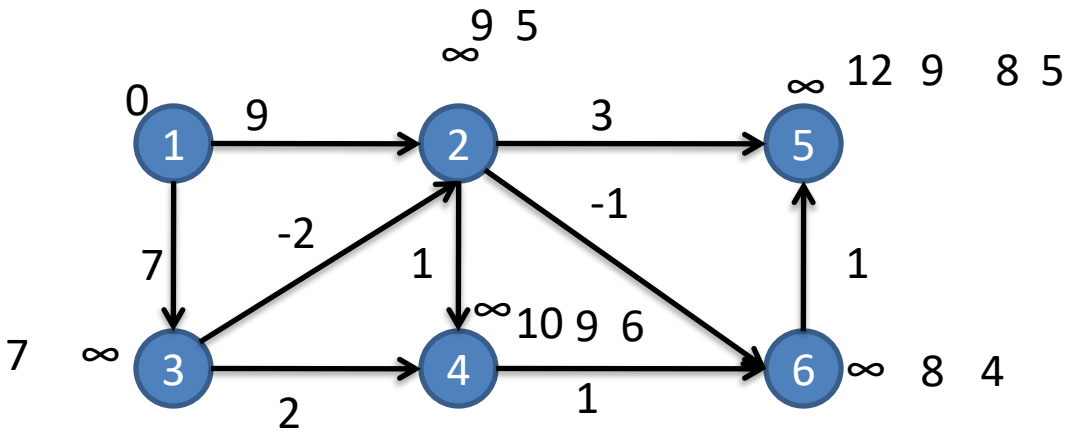
(1,2),(1,3),(2,4),(2,5),(2,6),(3,2),(3,4),(4,6),(6,5)



1	2	3	4	5	6
/	/	/	/	/	/
	3	1	2	6	2

	1	2	3	4	5	6
initial	0	∞	∞	∞	∞	∞
It1	0	5	7	9	9	8
It2	0	5	7	6	5	4
It3						
It4						
It5						

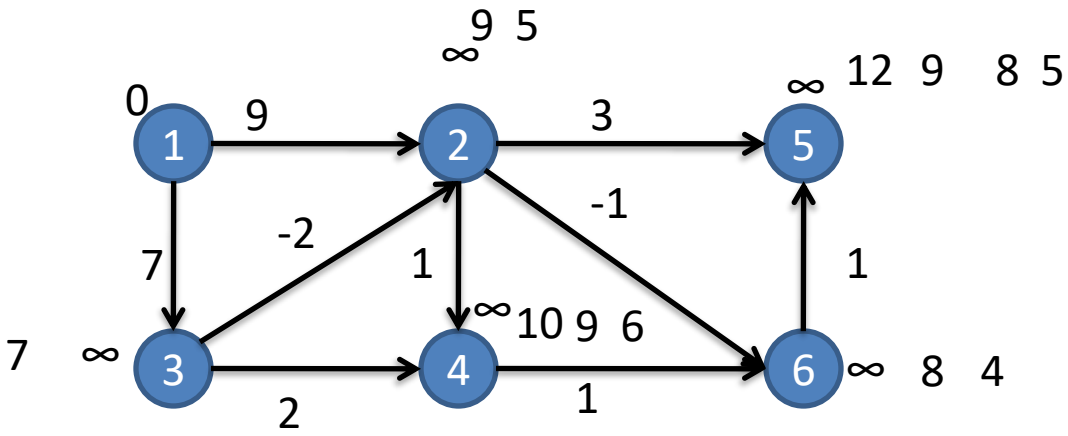
(1,2),(1,3),(2,4),(2,5),(2,6),(3,2),(3,4),(4,6),(6,5)



1	2	3	4	5	6
/	/	/	/	/	/
	3	1	2	6	2

	1	2	3	4	5	6
initial	0	∞	∞	∞	∞	∞
It1	0	5	7	9	9	8
It2	0	5	7	6	5	4
It3	0	5	7	6	5	4
It4						
It5						

(1,2),(1,3),(2,4),(2,5),(2,6),(3,2),(3,4),(4,6),(6,5)



1	2	3	4	5	6
/	/	/	/	/	/
	3	1	2	6	2

	1	2	3	4	5	6
initial	0	∞	∞	∞	∞	∞
It1	0	5	7	9	9	8
It2	0	5	7	6	5	4
It3	0	5	7	6	5	4
It4						
It5						

Breadth First Search (BFS)

Queue management:

- FIFO(First In First Out)

Application :

- Search of the vertices can be reached from the vertex v0
- Determine the connected components
- May be used to search the shortest path in function of the number of edges.

Complexity

- $O(n^2)$: With adjacency matrix
- $O(n+m)$: With adjacency list.

Breadth First Search (BFS)

Algorithm
Initialize_queue(Q)
For each vertex $v_i \in V$ do

$$\pi(v_i) \leftarrow nil$$

$$d[v_i] \leftarrow \infty$$

$$color[v_i] \leftarrow white$$

Endfor

$$d[v_0] \leftarrow 0$$

$$append\_end\_queue(Q, v_0)$$

$$color[v_0] \leftarrow gray$$

Breadth First Search (BFS)

While (empty(Q)=false do

$take\_begin\_queue(Q, v_i)$

For each $v_j \in succ(v_i)$ do

If $color[v_j] = white$ then

Append_end_queue(Q,vj)

$\pi(v_j) \leftarrow v_i$

$d[v_j] \leftarrow d[v_i] + 1$

$color[v_j] \leftarrow gray$

Endif

Endfor

$color[v_i] = Black$

Endwhile

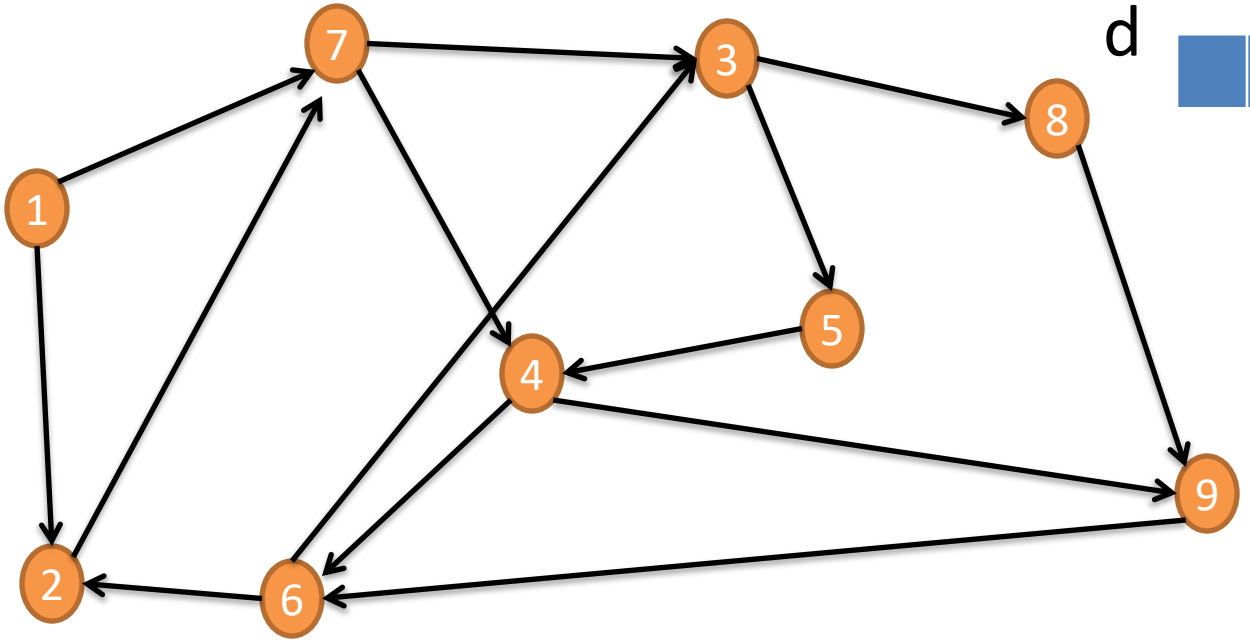
end

Breadth First Search (BFS)

Q

π

d



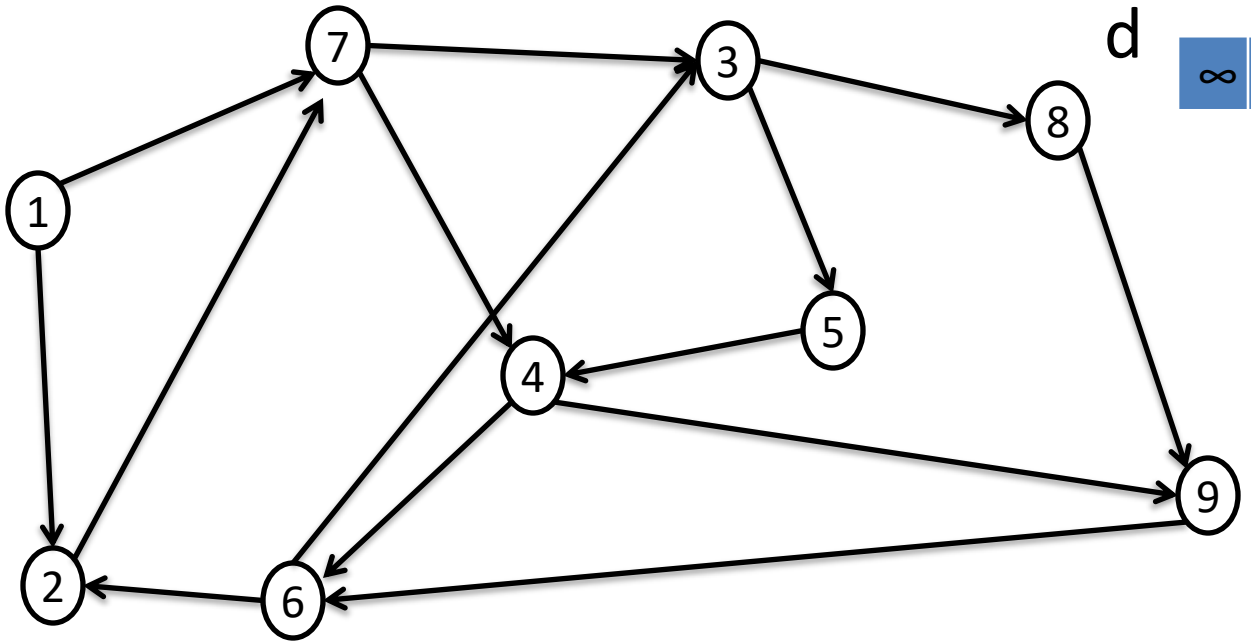
Breadth First Search (BFS)

∞

Q

π

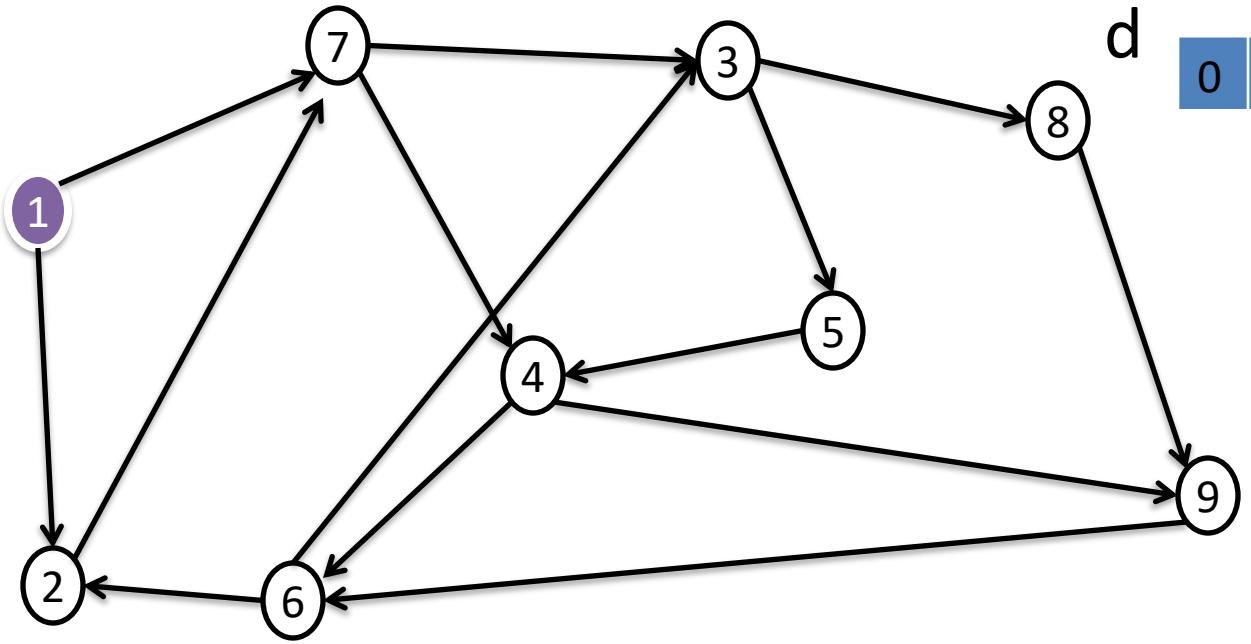
d



Breadth First Search (BFS)

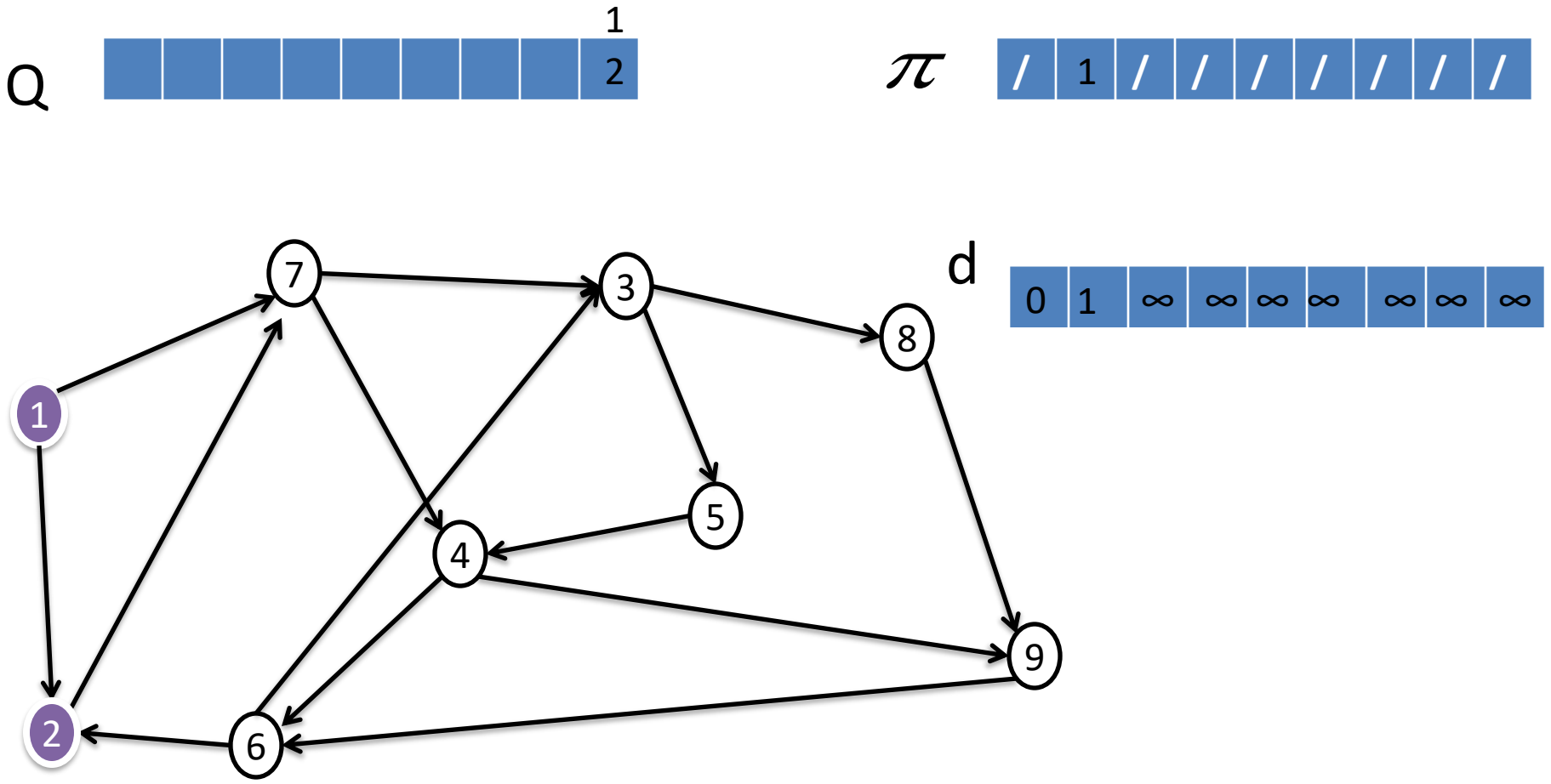
Q | | | | | | | | 1

π | / | / | / | / | / | / | / | / | /



d | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞

Breadth First Search (BFS)



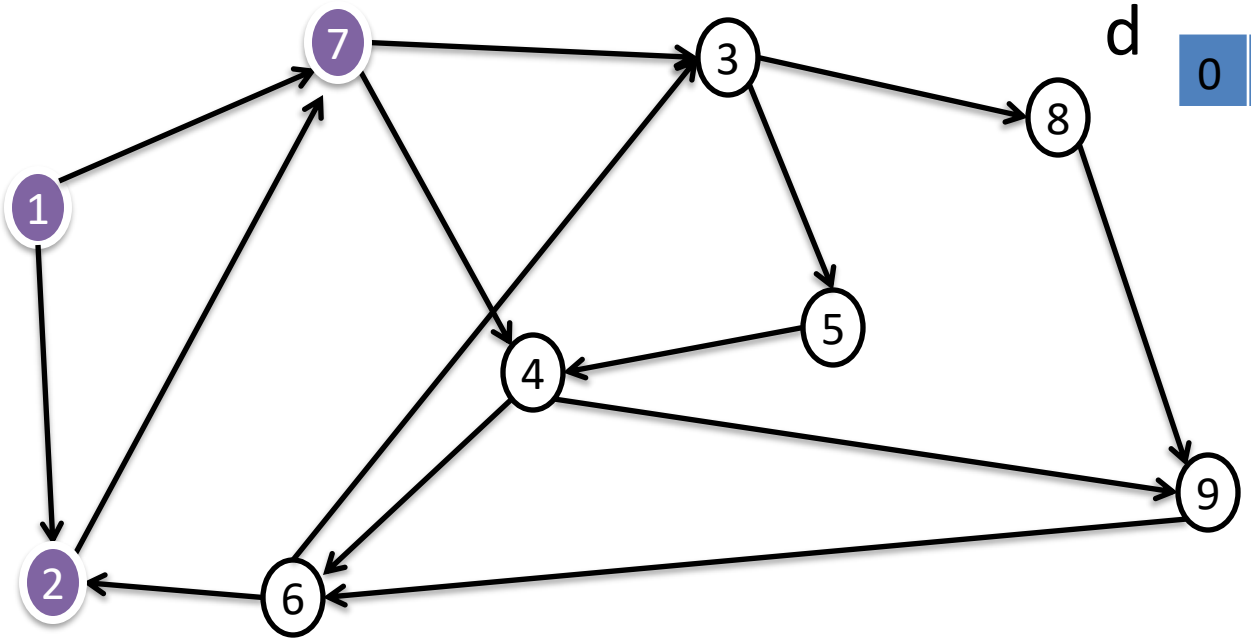
Breadth First Search (BFS)

Q

							7	2
								1

π

/	1	/	/	/	/	1	/	/



d

0	1	∞	∞	∞	∞	1	∞	∞

Breadth First Search (BFS)

Q

								7

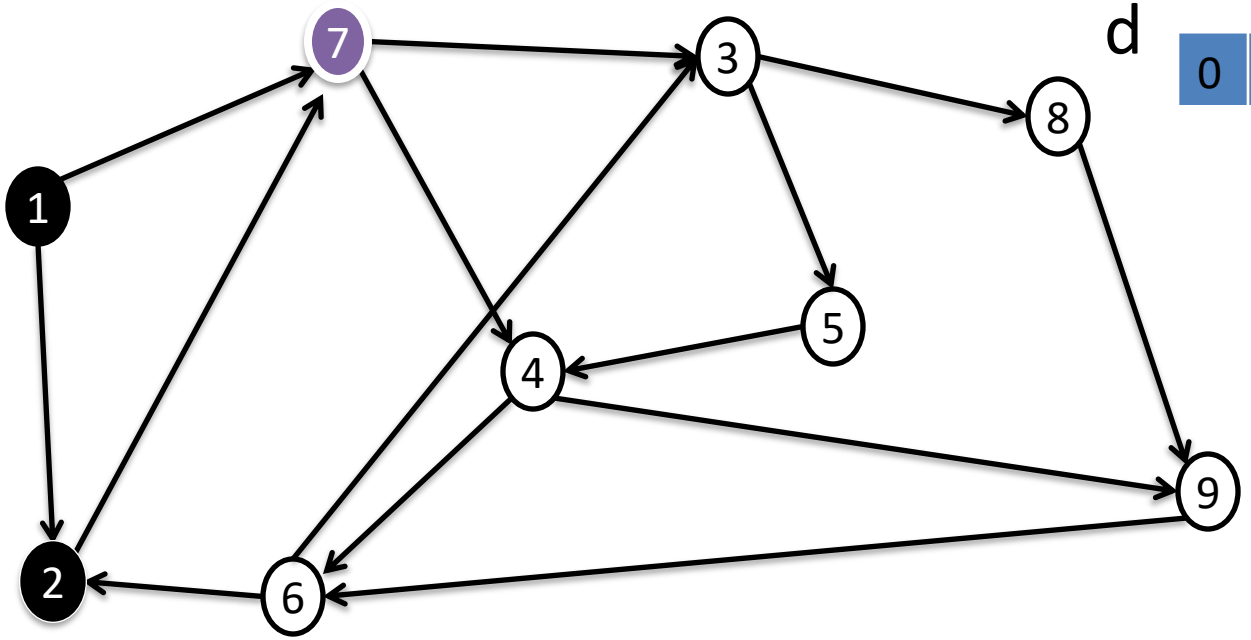
2

π

/	1	/	/	/	/	1	/	/

d

0	1	∞	∞	∞	∞	1	∞	∞



Breadth First Search (BFS)

Q

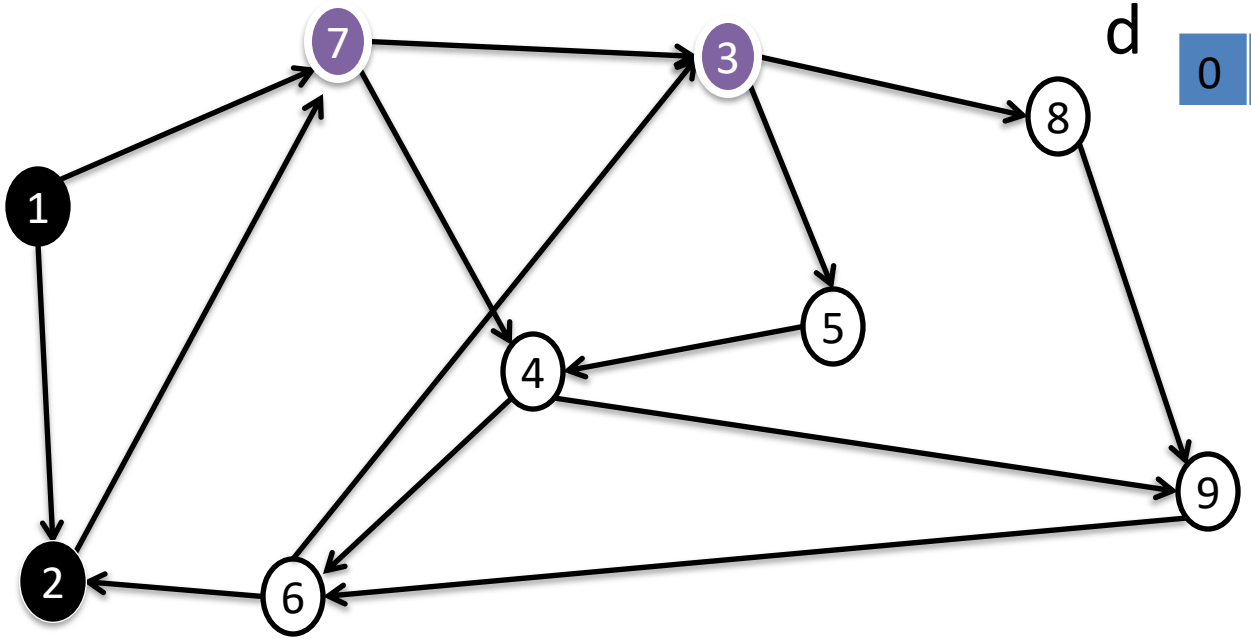
								7
								3

π

/	1	7	/	/	/	1	/	/

d

0	1	2	∞	∞	∞	1	∞	∞



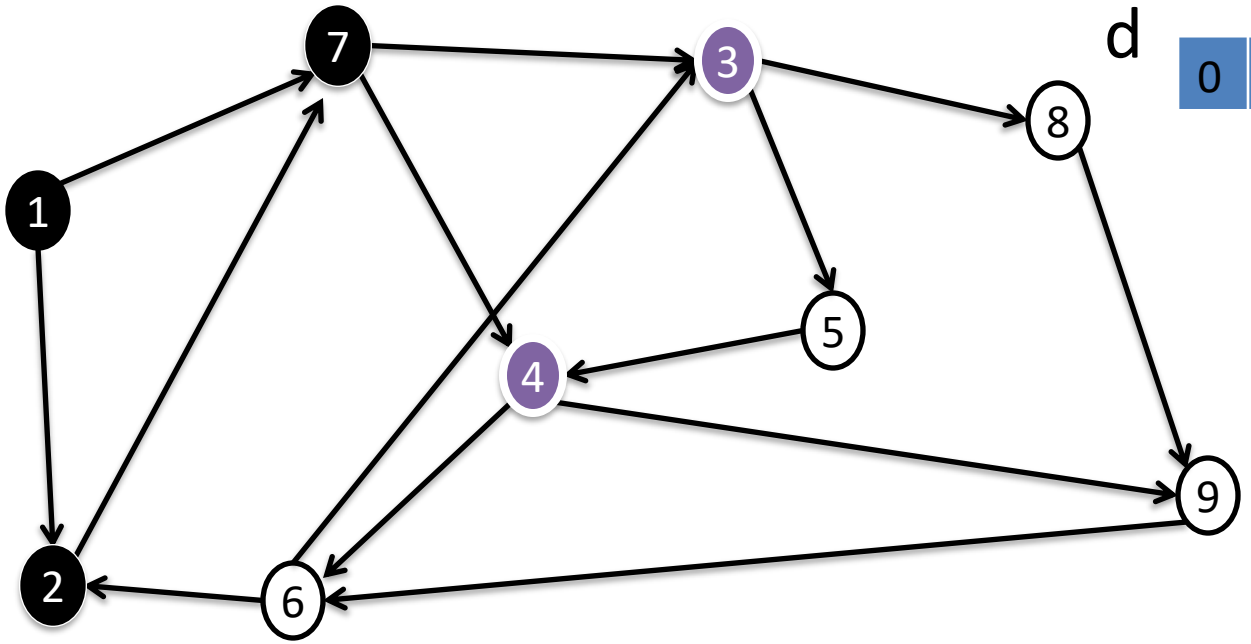
Breadth First Search (BFS)

Q

							4	3
								7

π

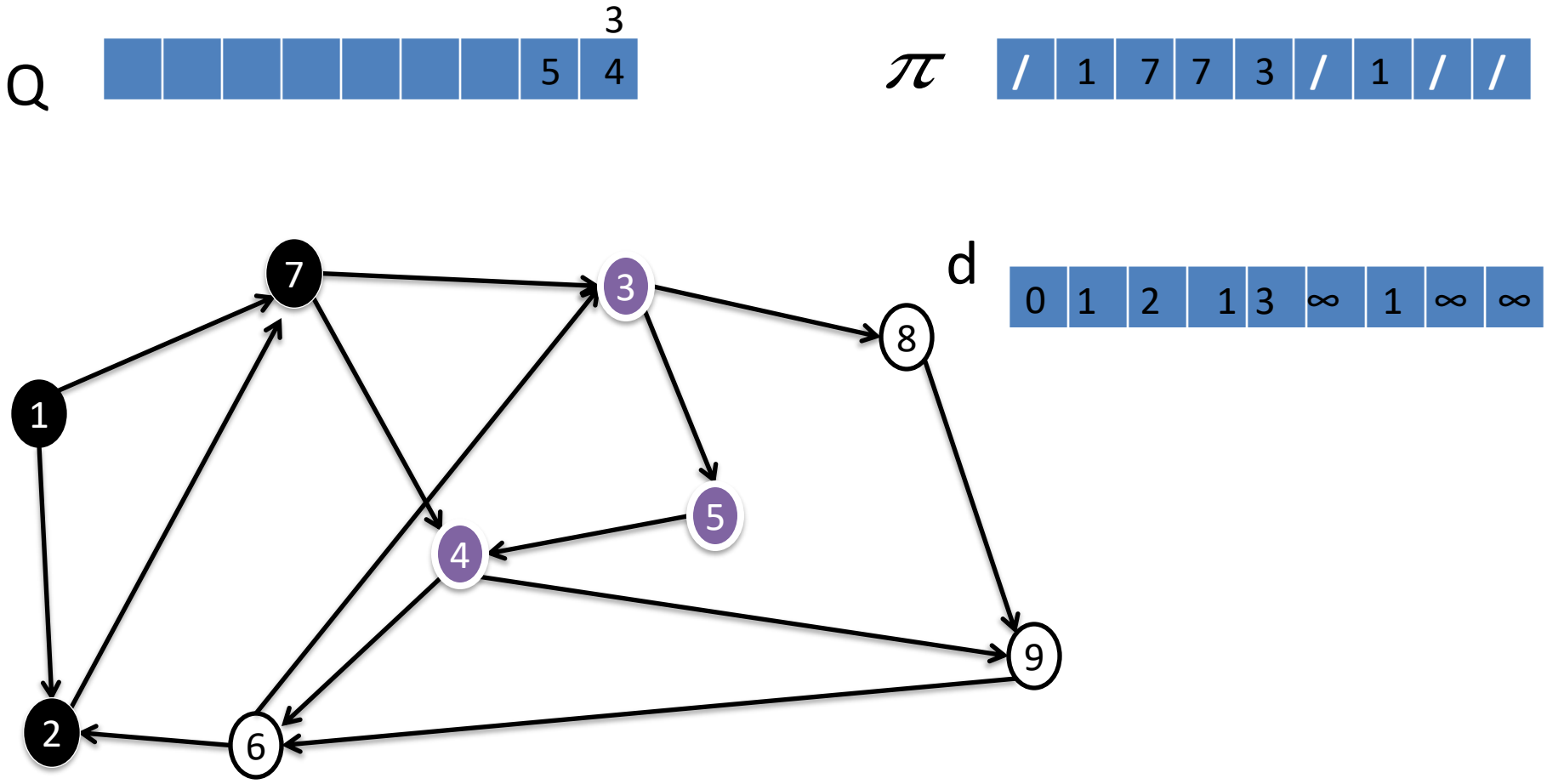
/	1	7	7	/	/	1	/	/



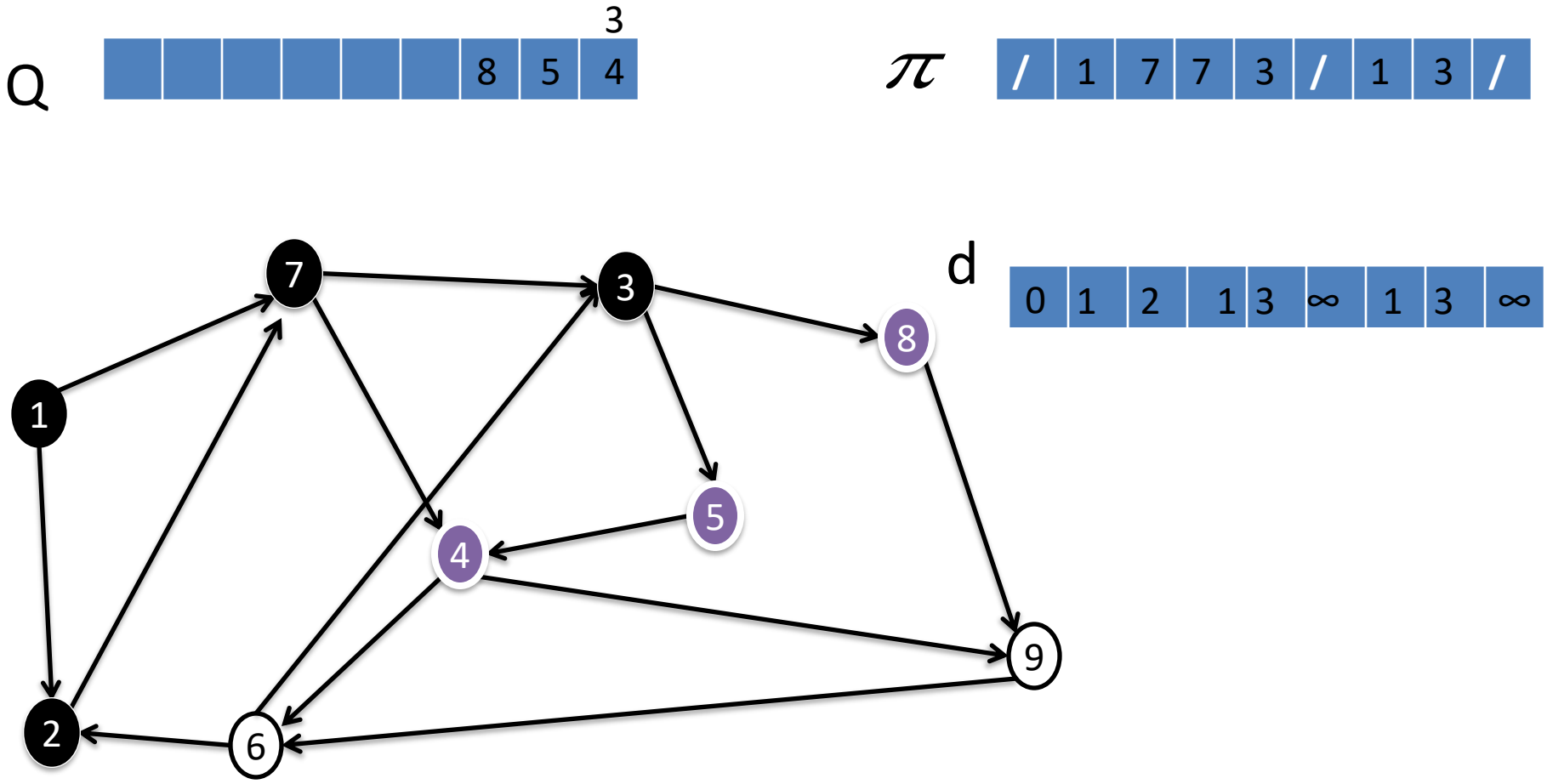
d

0	1	2	1	∞	∞	1	∞	∞

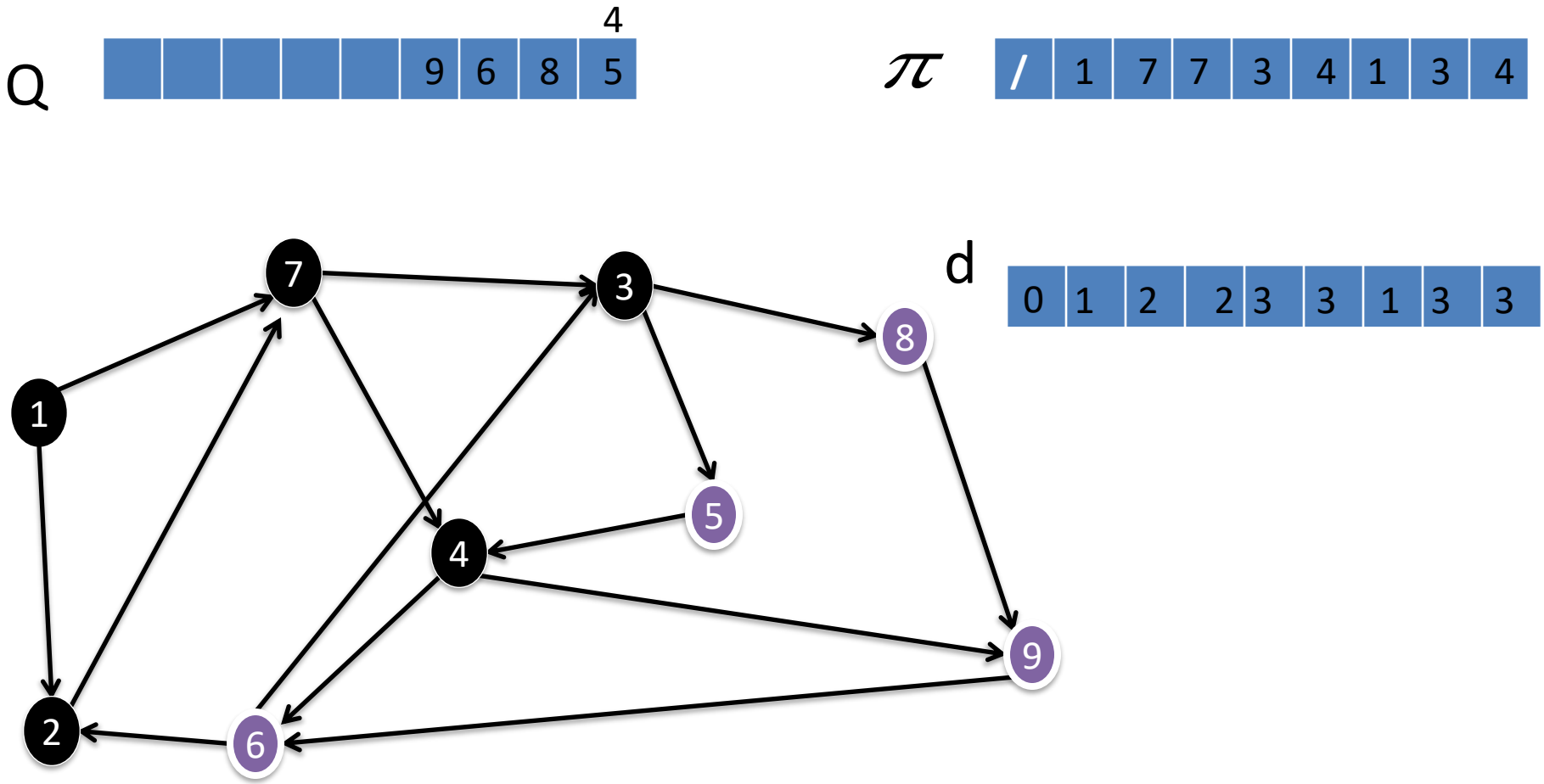
Breadth First Search (BFS)



Breadth First Search (BFS)



Breadth First Search (BFS)



Breadth First Search (BFS)

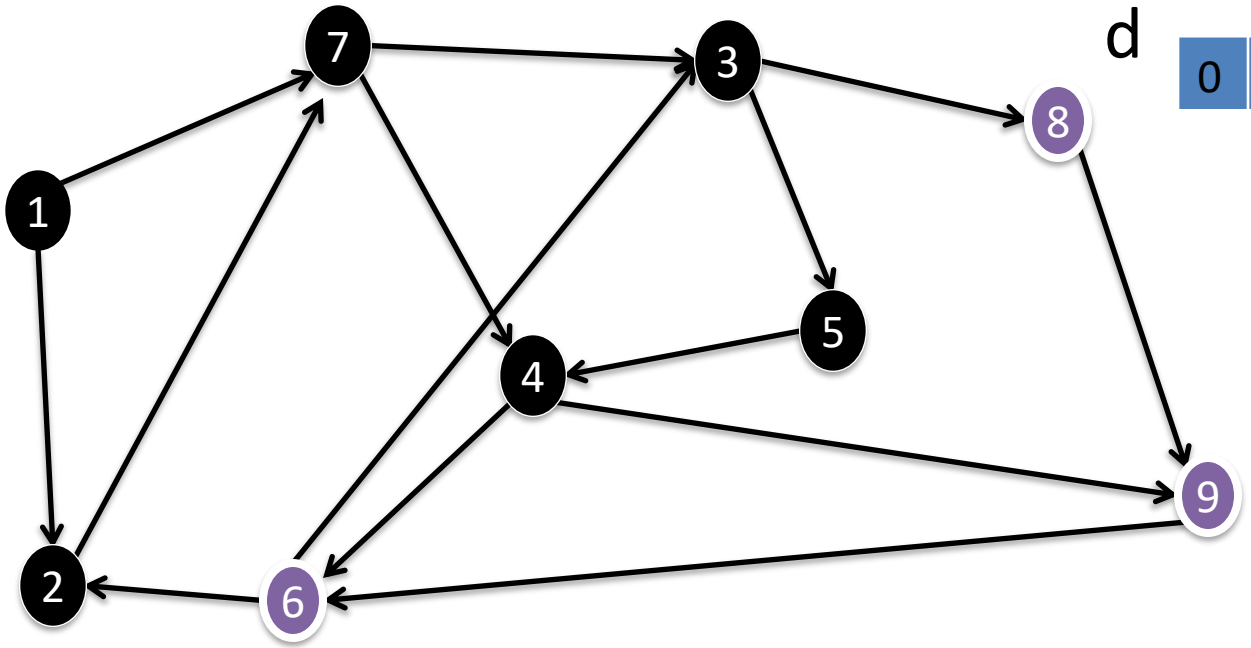
Q

						9	6	8

5

π

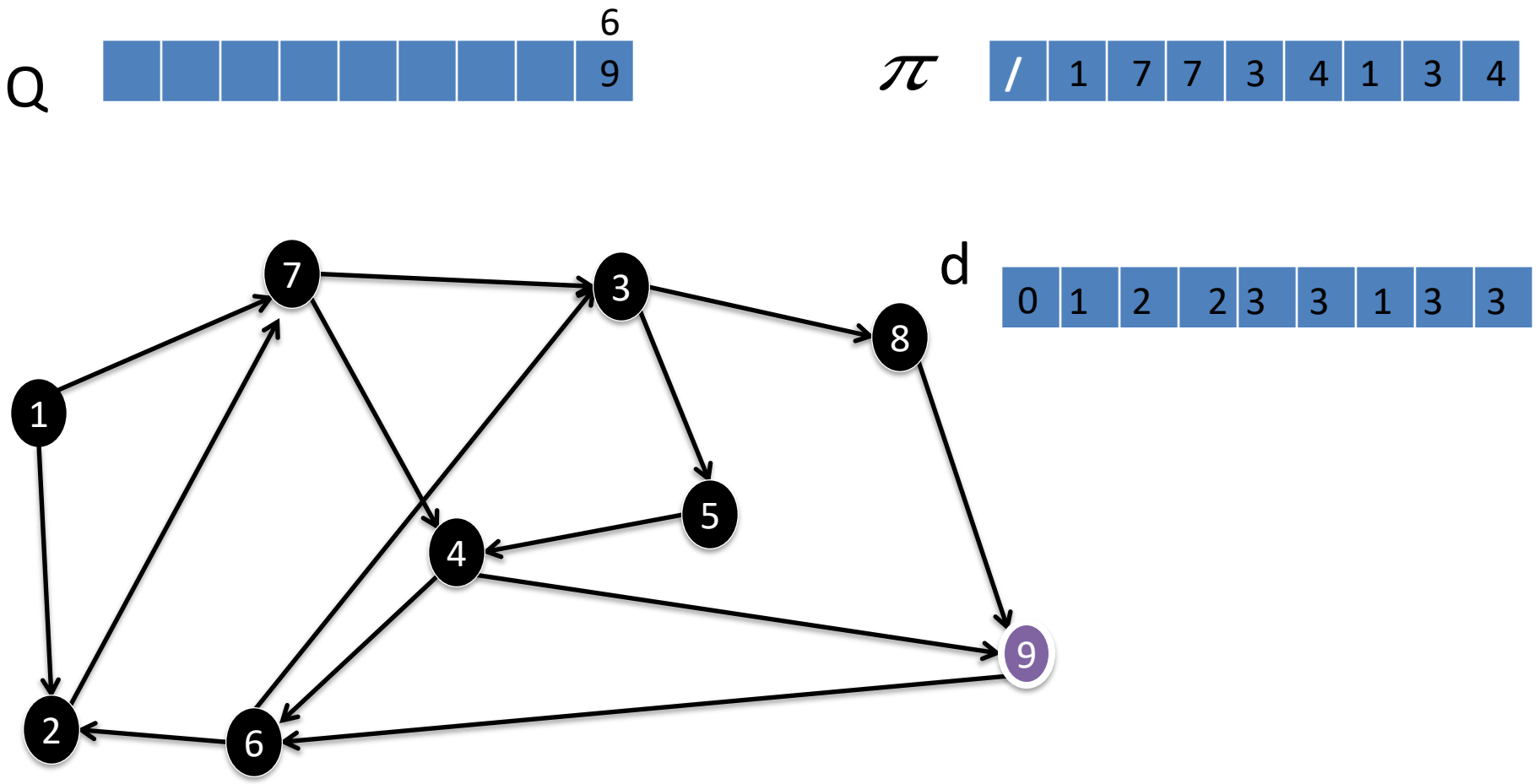
/	1	7	7	3	4	1	3	4



d

0	1	2	2	3	3	1	3	3

Breadth First Search (BFS)

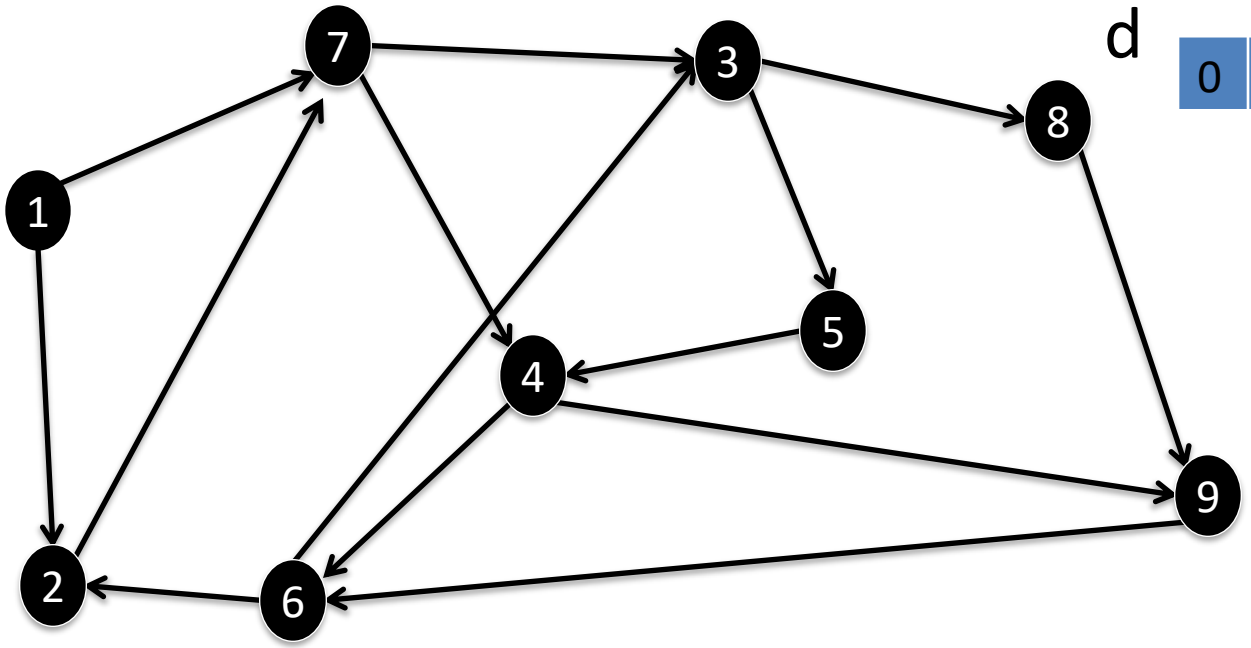


Breadth First Search (BFS)

Q

π

/	1	7	7	3	4	1	3	4



d

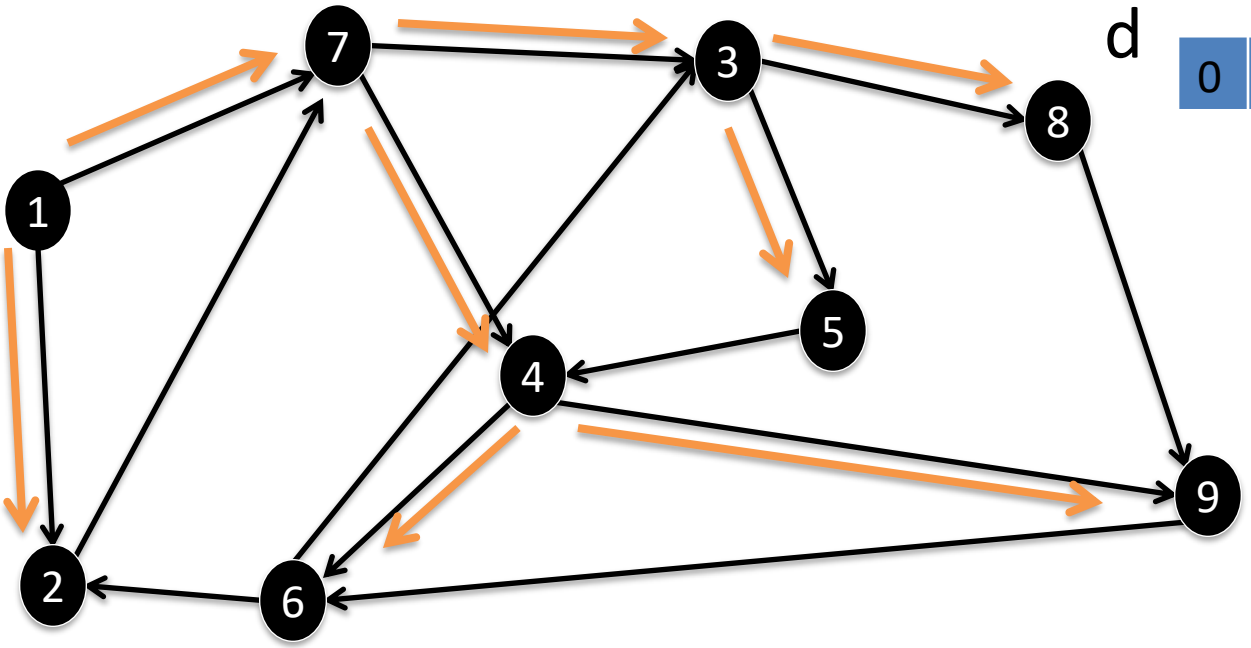
0	1	2	2	3	3	1	3	3

Breadth First Search (BFS)

Q

π

/	1	7	7	3	4	1	3	4



d

0	1	2	2	3	3	1	3	3

Depth First Search (DFS)

In the DFS the management of the waiting list of black coloring Is achieved using stack (LIFO:Last In First Out), in other words we consider each time the last gray vertex pushed in the stack, and we push above it all its white successors.

Depth First Search (DFS)

Algorithm
Initialize_stack(P)
For each vertex $v_i \in V$ do

$$\pi(v_i) \leftarrow nil$$
$$color[v_i] \leftarrow white$$

Endfor

$$t \leftarrow 0$$
$$disc[v_0] \leftarrow t$$
$$Push(P, v0)$$
$$color[v_0] \leftarrow gray$$

Depth First Search (DFS)

While (is-empty(P)=false do

$t \leftarrow t+1$

$vi \leftarrow Top(p)$

If $\exists vj \in succ(vi)$ where $color[v_j] = white$ then

Push(P,vj)

$color[v_j] \leftarrow gray$

$\pi[vj] = vi$

$disc[vj] = t$

Else

Pop(P,vi)

$color[v_i] = Black$

$endh[vi] \leftarrow t$

Endif

Endwhile

end

Shortest Path Problem 50

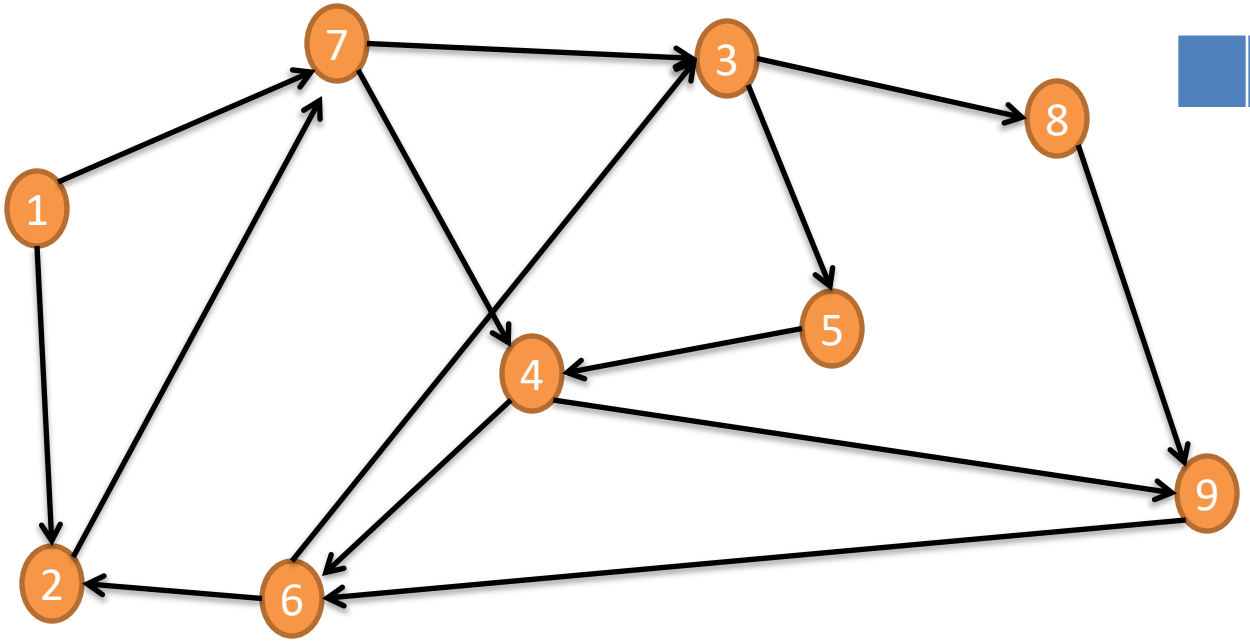
Depth First Search (DFS)

π

endh

t=

disc



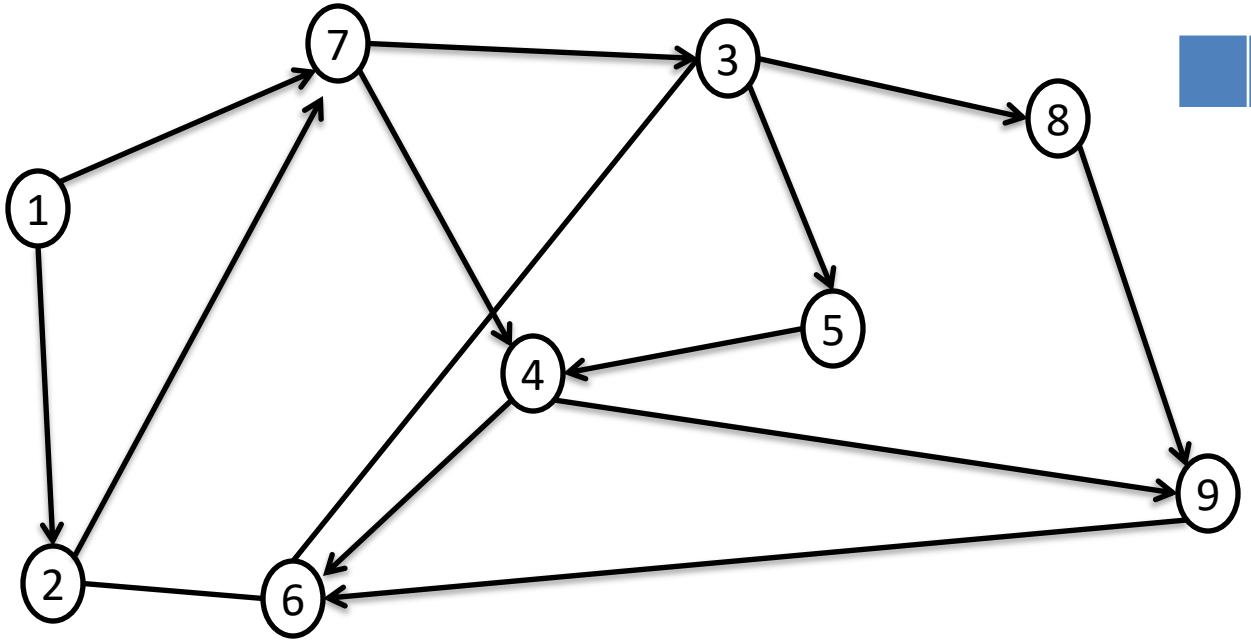
Depth First Search (DFS)

π

endh

t=

disc



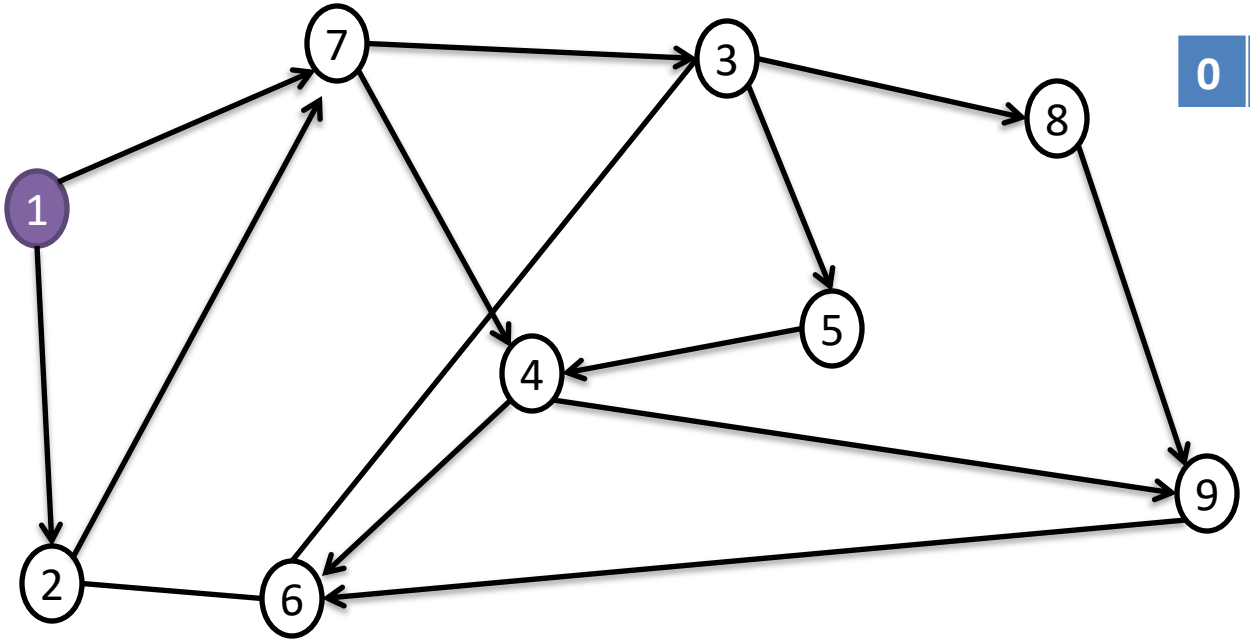
Depth First Search (DFS)

π

endh

t=0

disc



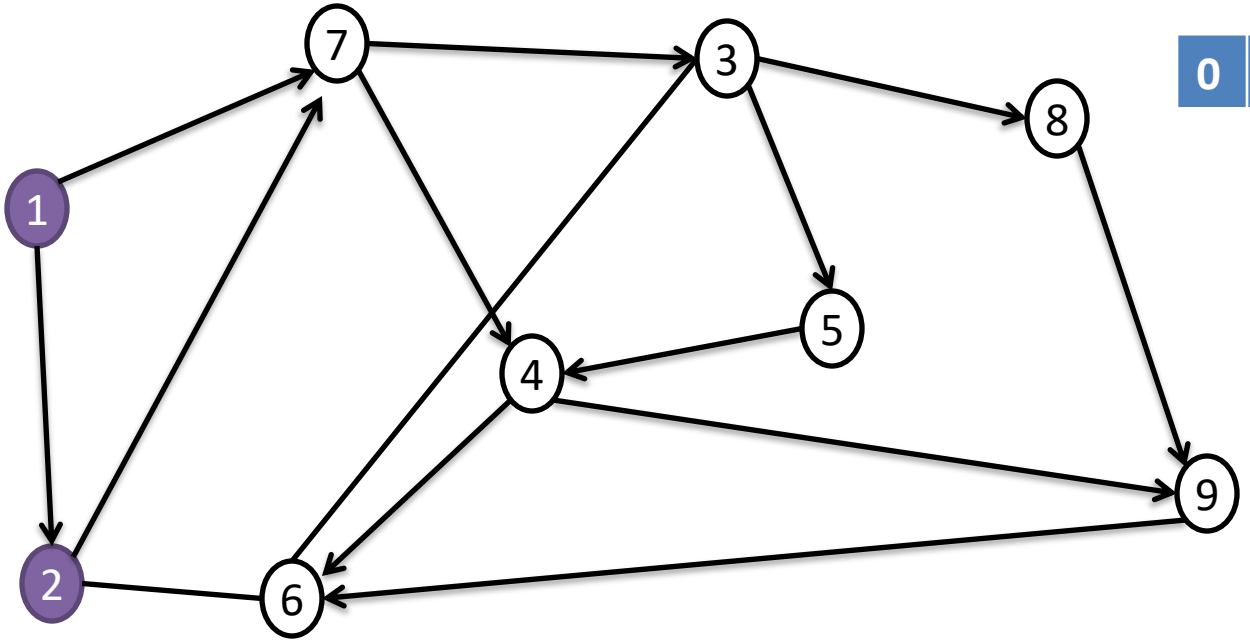
Depth First Search (DFS)

π

endh

T=1

disc

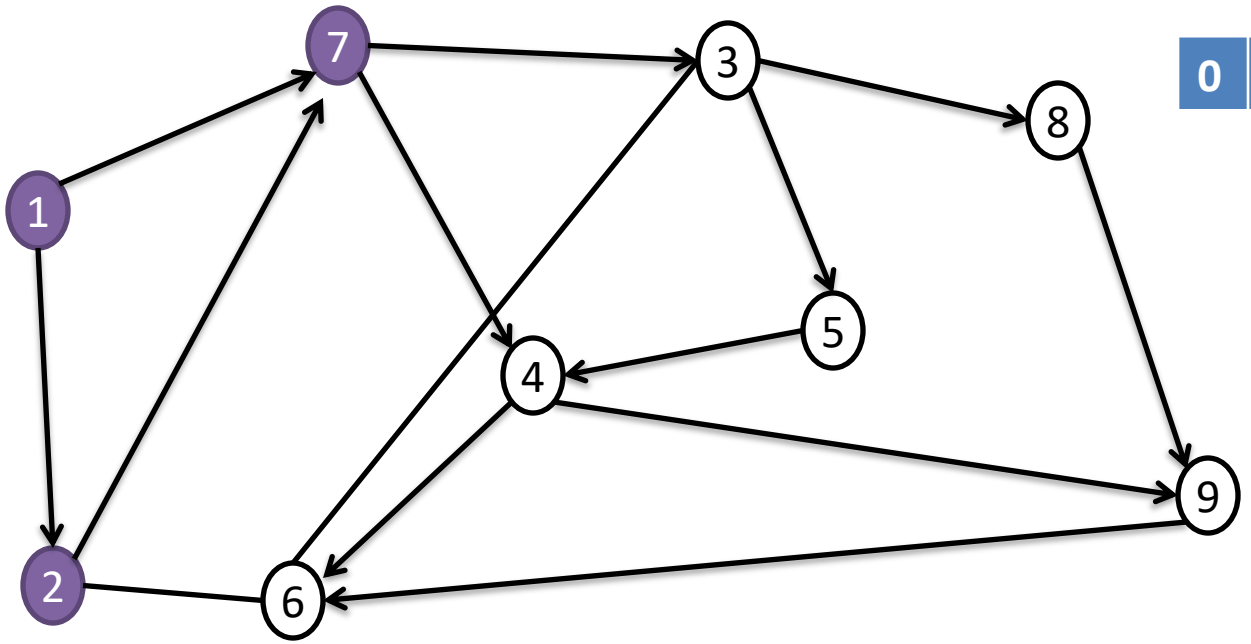


Depth First Search (DFS)

π

endh

T=2



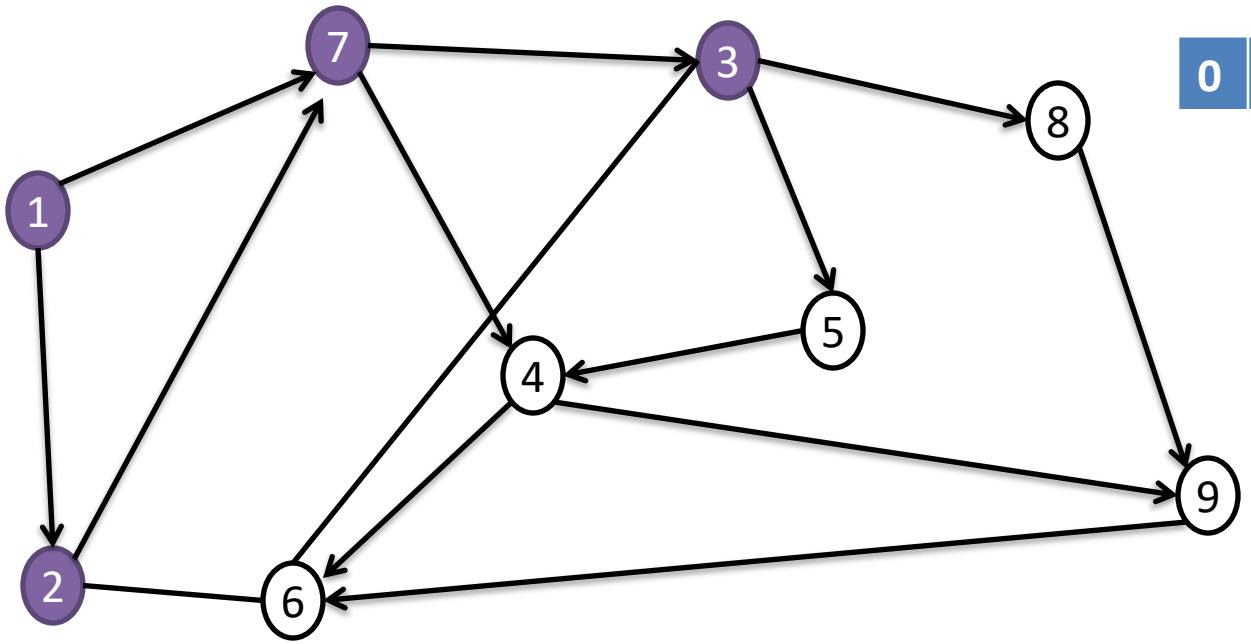
disc

Depth First Search (DFS)

π

endh

T=3



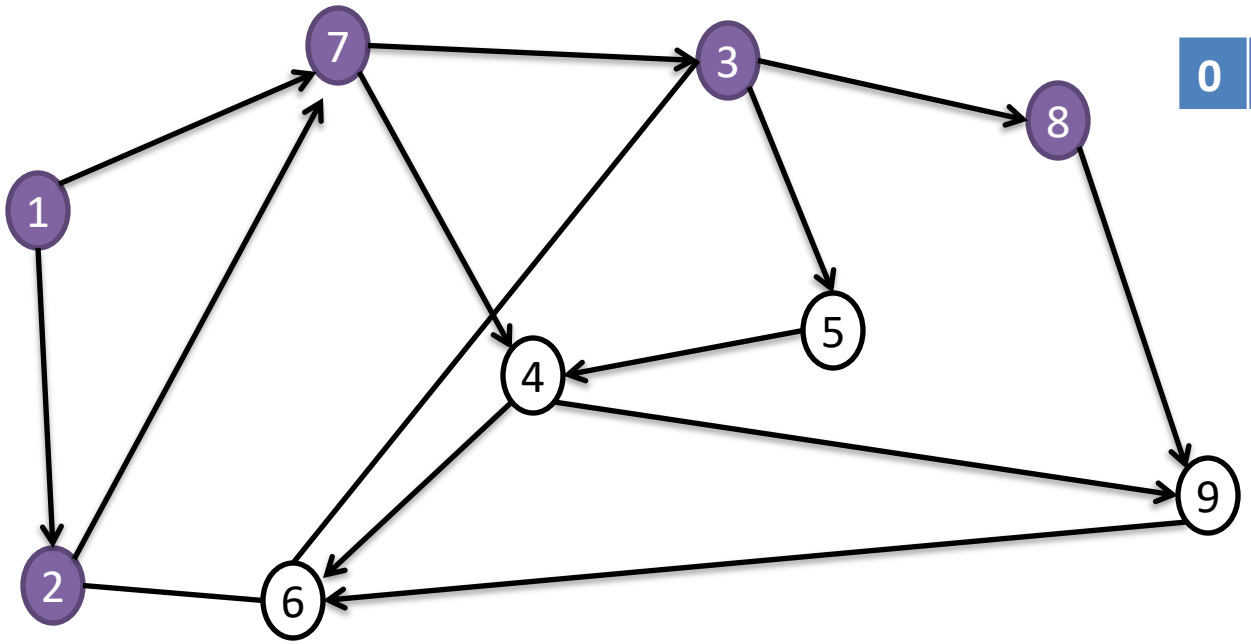
disc

Depth First Search (DFS)

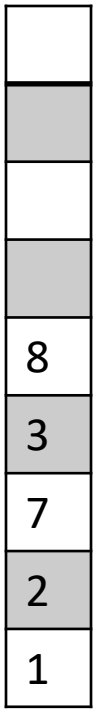
π

endh

T=4



disc

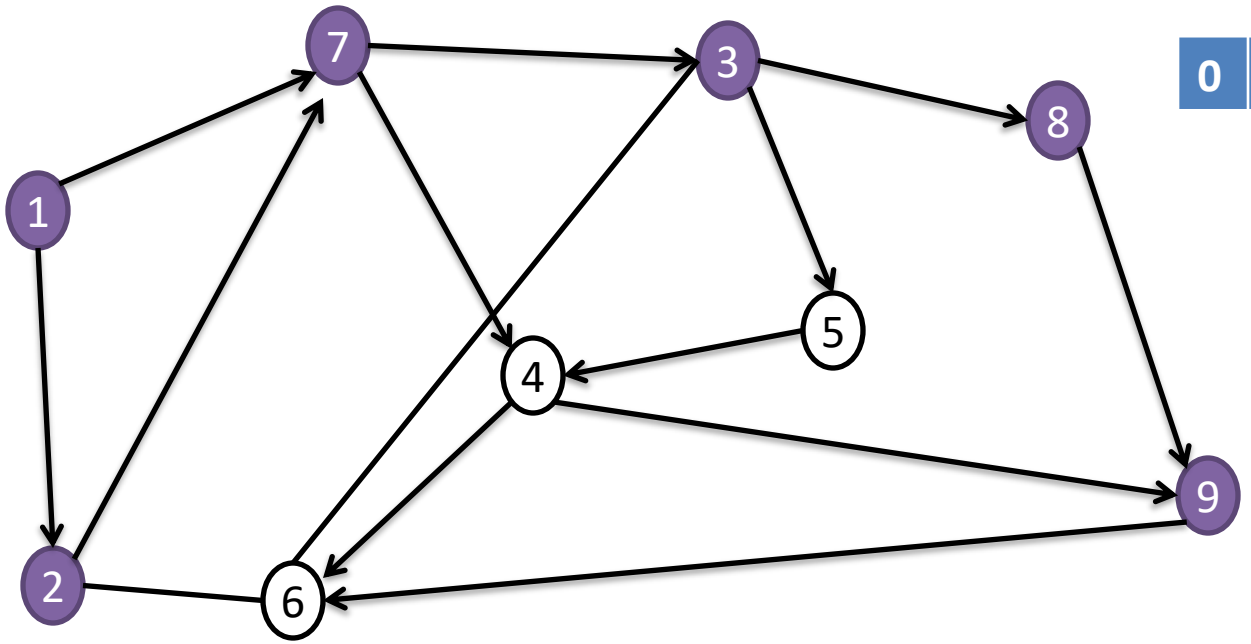


Depth First Search (DFS)

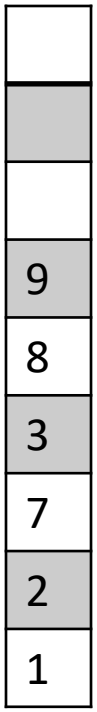
π

endh

T=5



disc

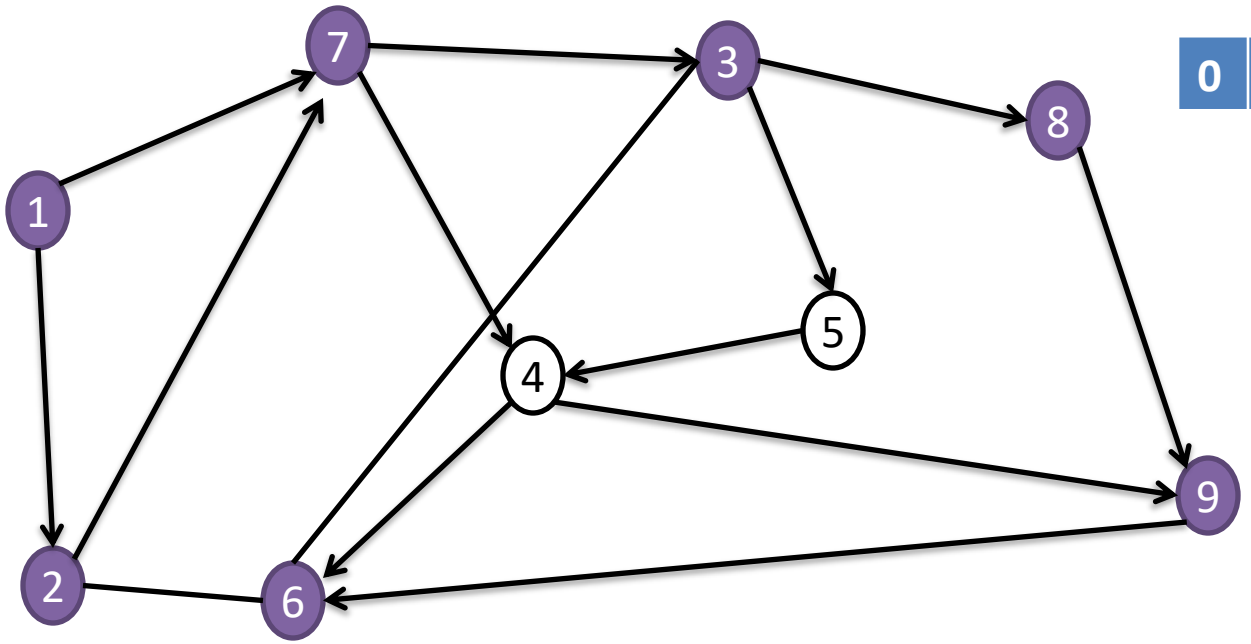


Depth First Search (DFS)

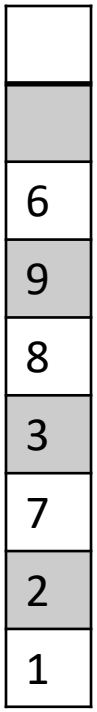
π

endh

T=6



disc



Depth First Search (DFS)

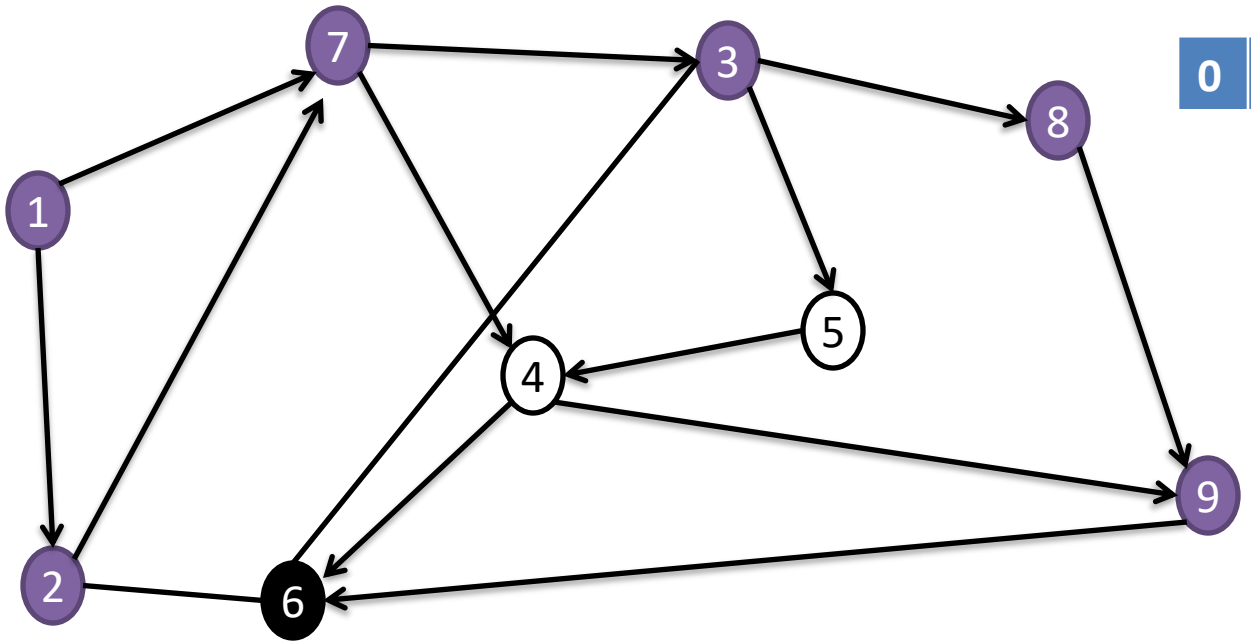
					7			

π

/	1	7	/	/	9	2	3	8

endh

T=7



disc

0	1	3			6	2	4	5

9
8
3
7
2
1

Depth First Search (DFS)

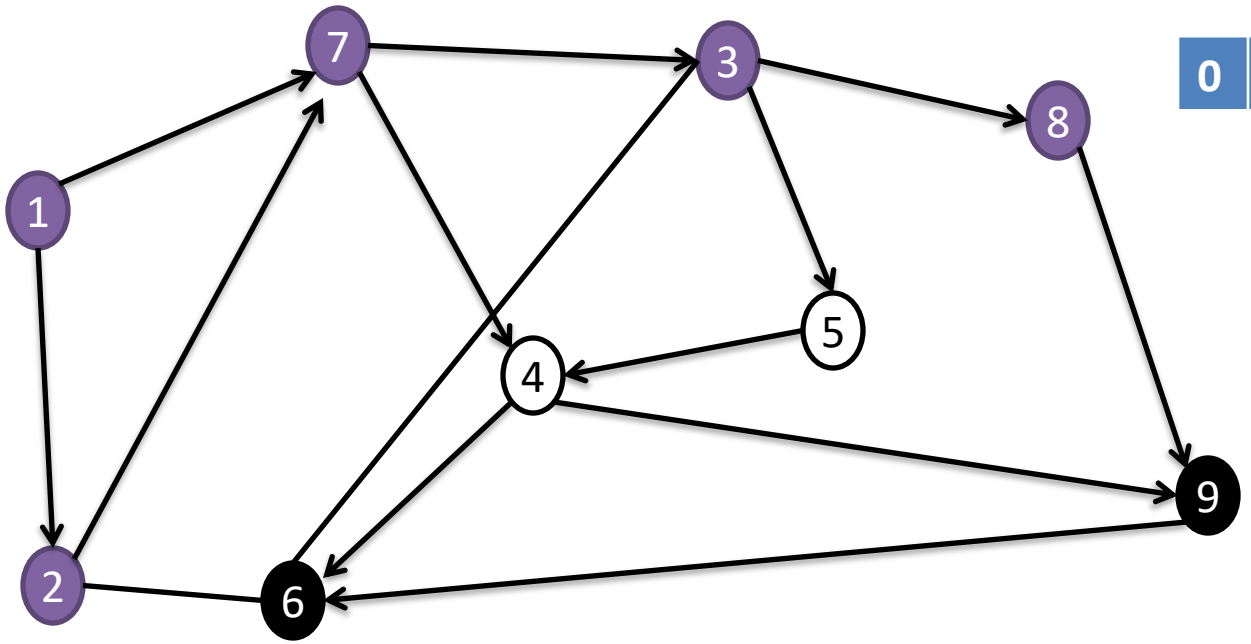
					7			8

π

/	1	7	/	/	9	2	3	8

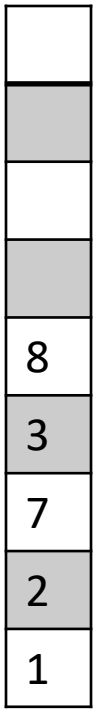
endh

T=8



disc

0	1	3			6	2	4	5



Depth First Search (DFS)

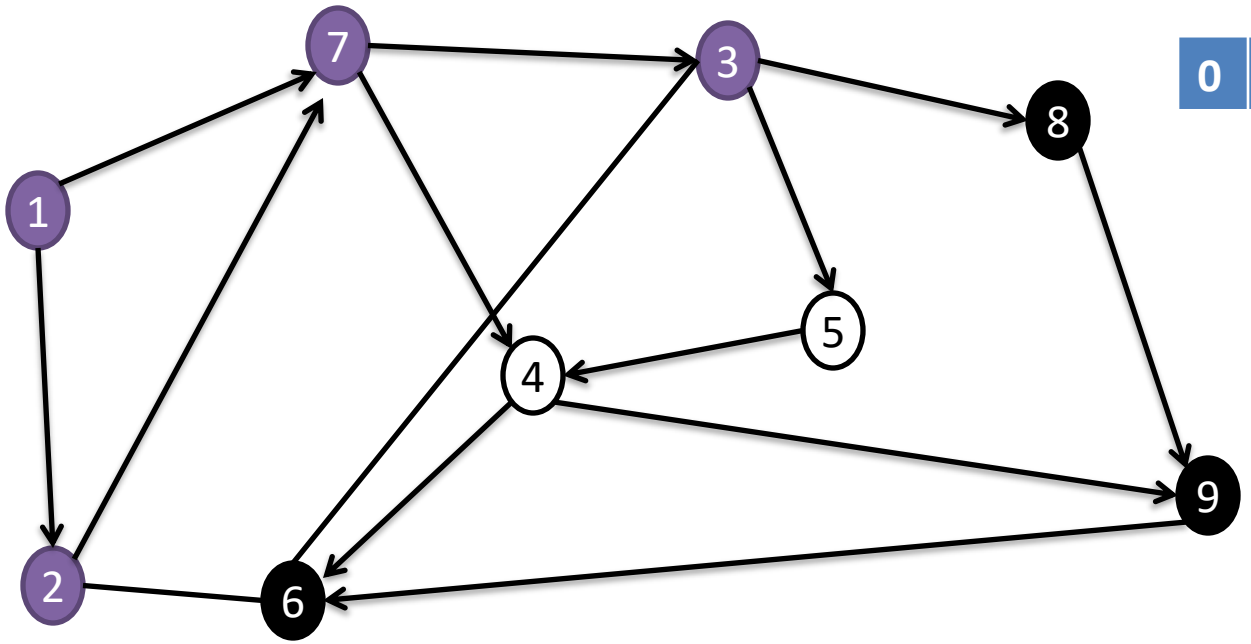
					7		9	8

π

/	1	7	/	/	9	2	3	8

endh

T=9



disc

0	1	3			6	2	4	5

Depth First Search (DFS)

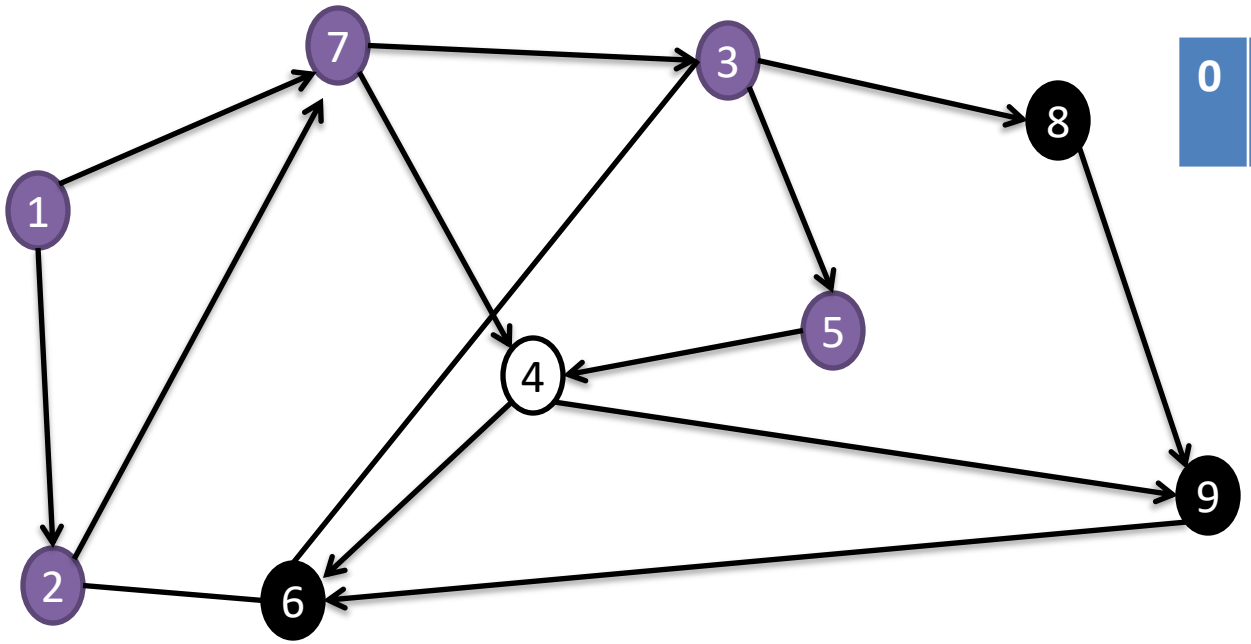
					7		9	8

π

/	1	7	/	3	9	2	3	8

endh

T=10



disc

0	1	3		10	6	2	4	5

5
3
7
2
1

Depth First Search (DFS)

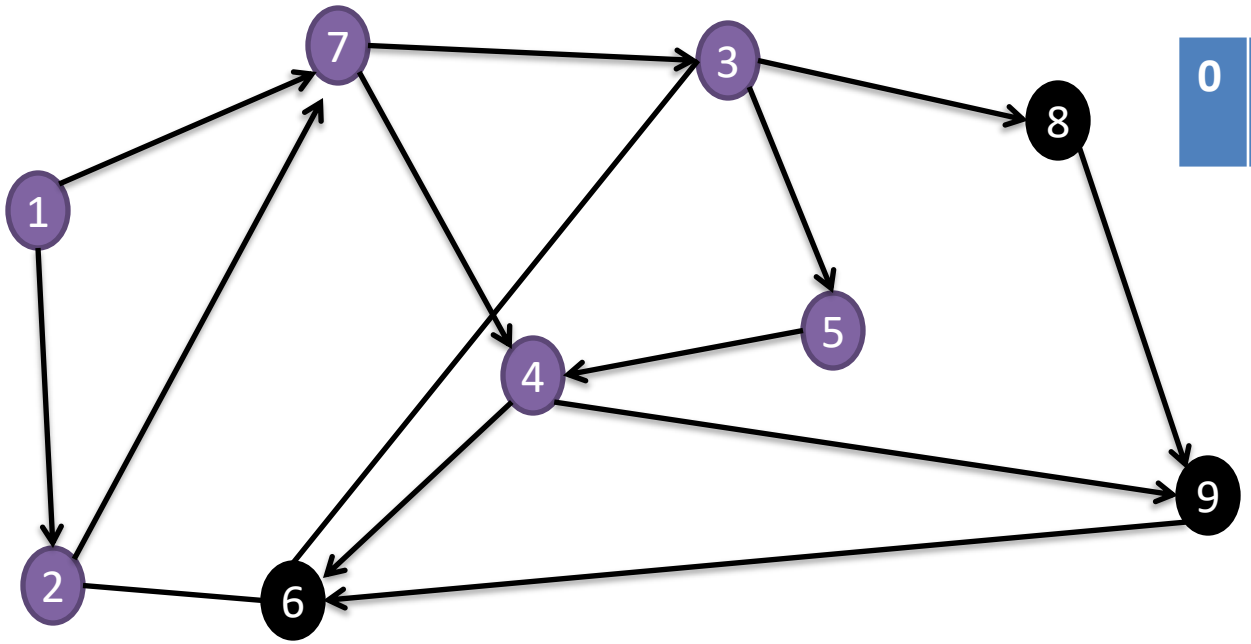
					7		9	8

π

/	1	7	5	3	9	2	3	8

endh

T=11



disc

0	1	3	11	10	6	2	4	5

4
5
3
7
2
1

Depth First Search (DFS)

			12		7		9	8

endh

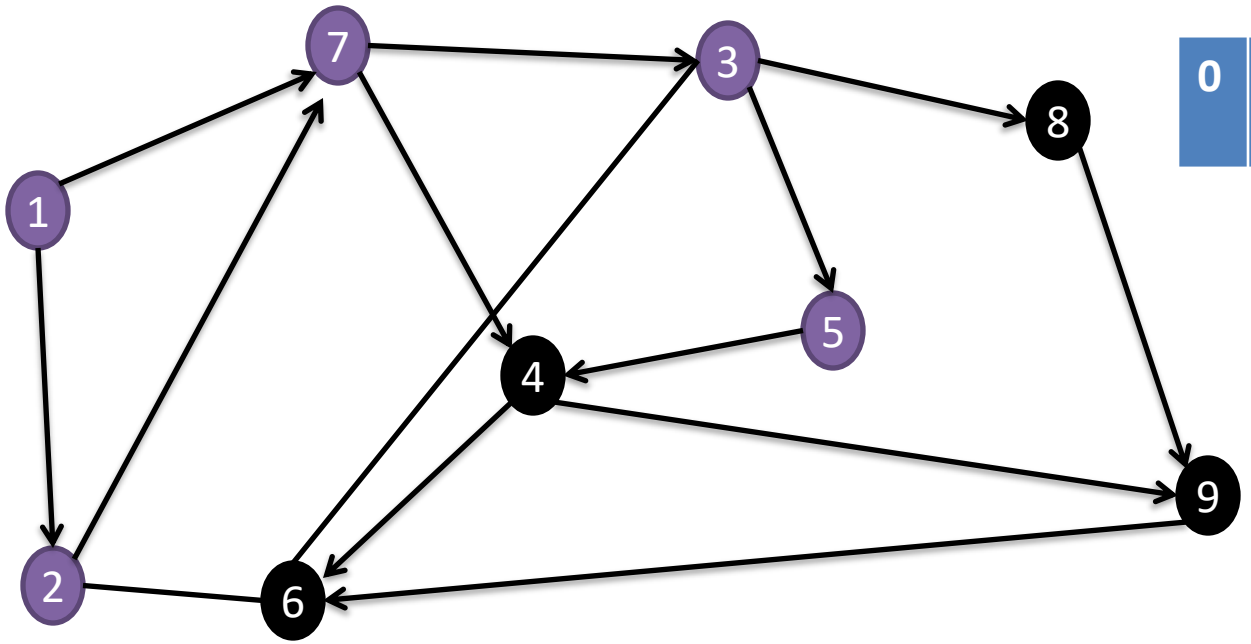
T=12

π

/	1	7	5	3	9	2	3	8

disc

0	1	3	11	10	6	2	4	5



Depth First Search (DFS)

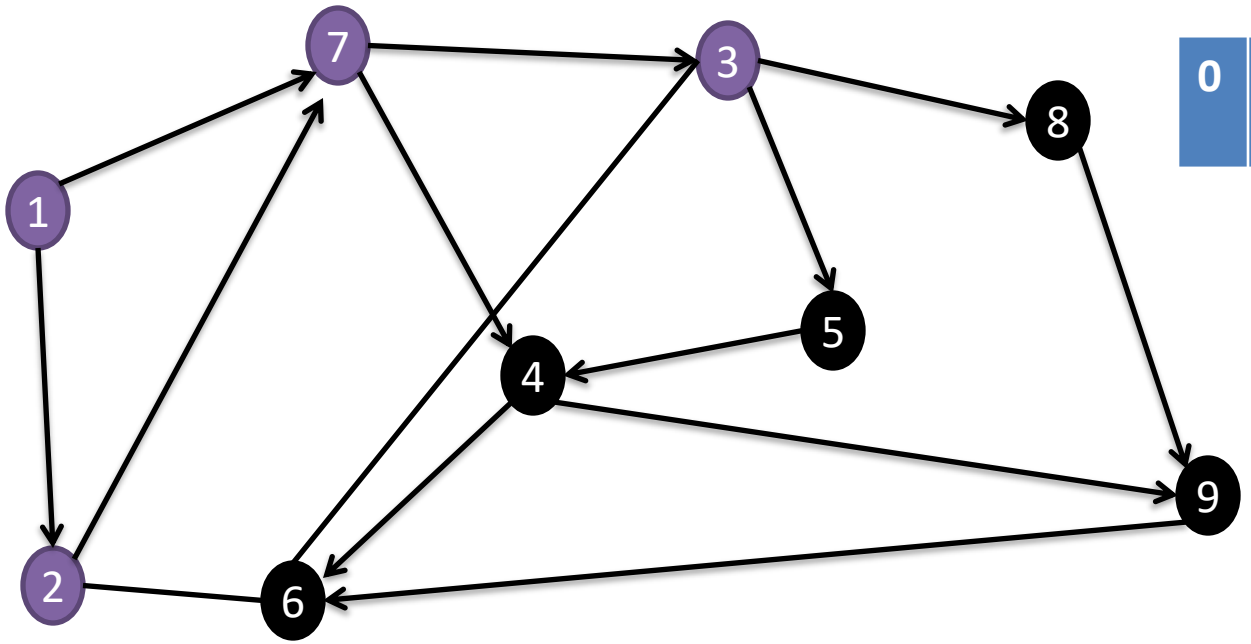
			12	13	7		9	8

endh

T=13

π

/	1	7	5	3	9	2	3	8



disc

0	1	3	11	10	6	2	4	5

Depth First Search (DFS)

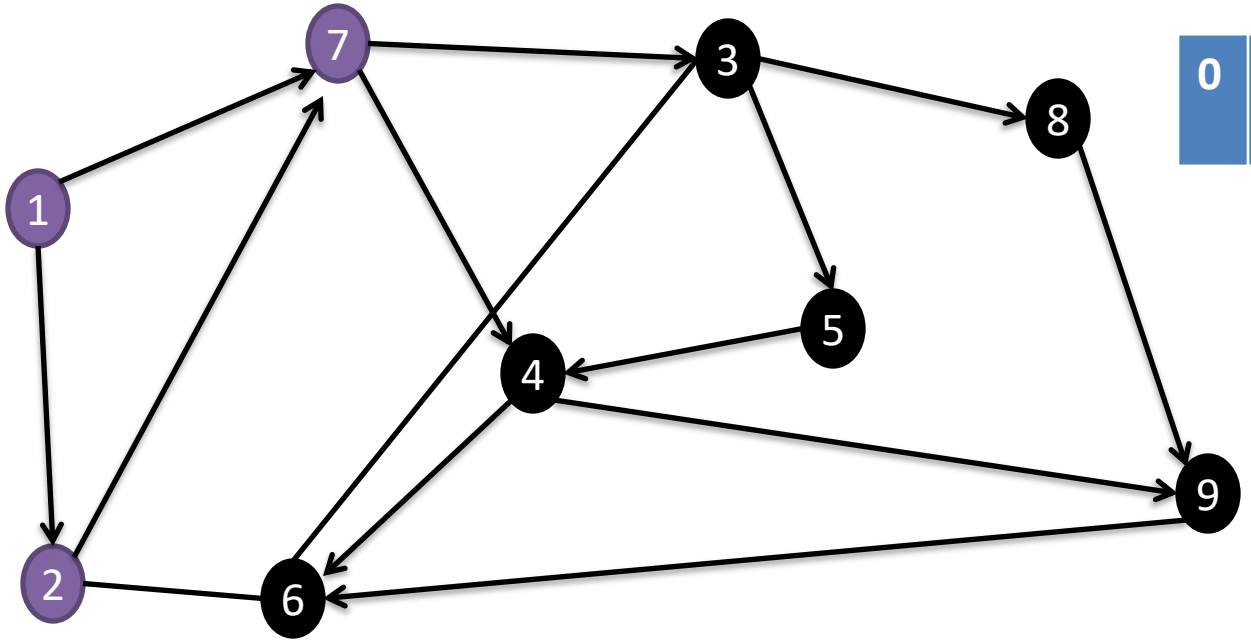
		14	12	13	7		9	8

endh

T=14

π

/	1	7	5	3	9	2	3	8



disc

0	1	3	11	10	6	2	4	5

Depth First Search (DFS)

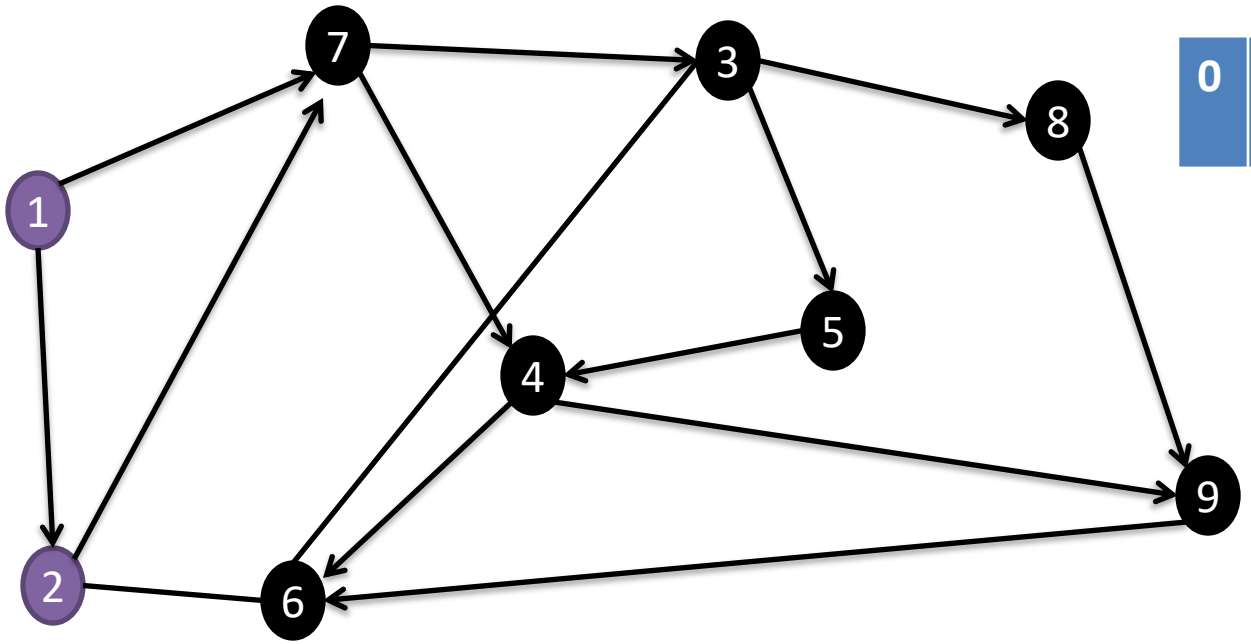
		14	12	13	7	15	9	8

endh

π

/	1	7	5	3	9	2	3	8

T=15



disc

0	1	3	11	10	6	2	4	5

Depth First Search (DFS)

17	16	14	12	13	7	15	9	8

endh

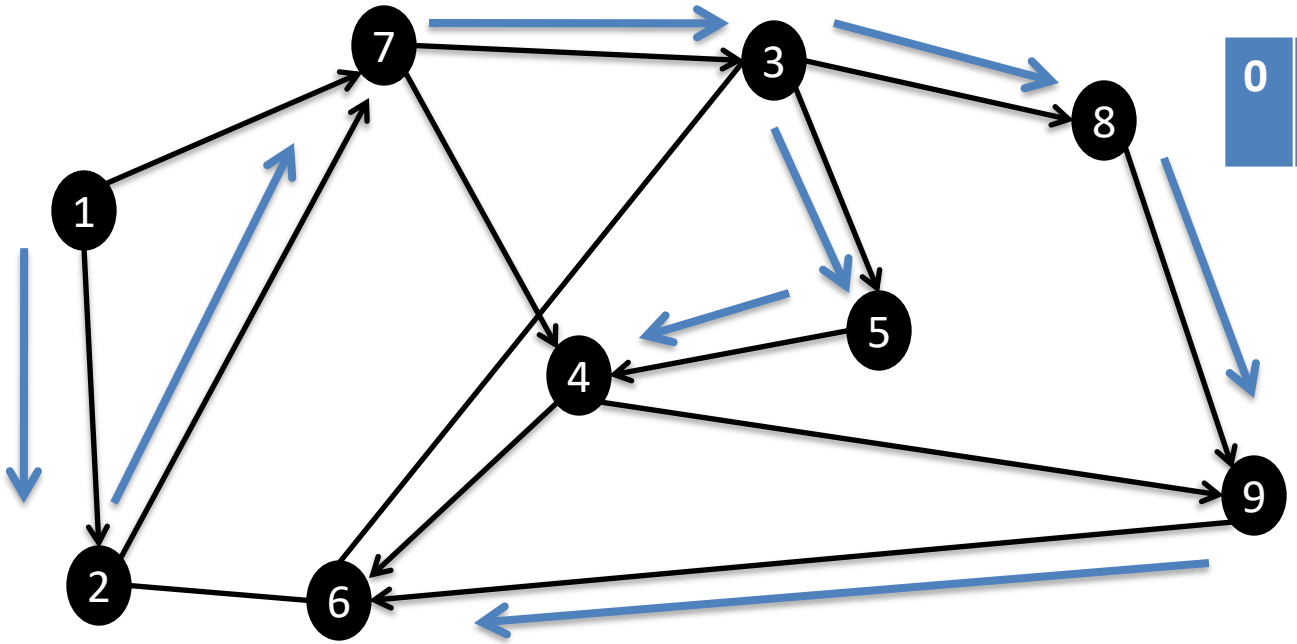
T=17

π

/	1	7	5	3	9	2	3	8

disc

0	1	3	11	10	6	2	4	5



Depth First Search (DFS)

Stack management:

- LIFO(Last In First Out)

Application :

- Search of circuits.
- Determine the connected components
- Used in strongly connected components search
- May be used to search the shortest path in function of the number of edges.

Complexity

- $O(n^2)$: With adjacency matrix
- $O(n+m)$: With adjacency list.