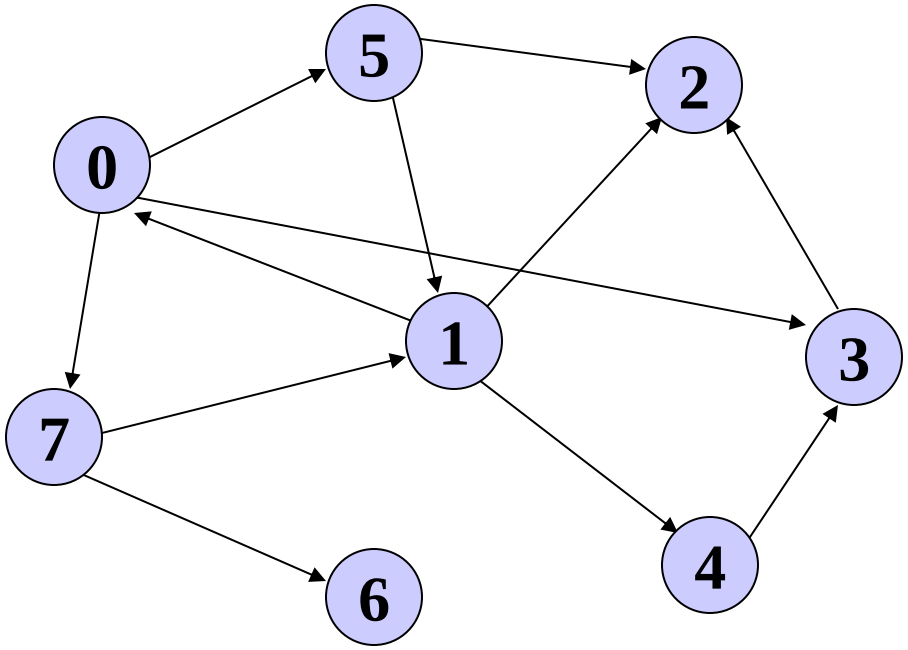
Graph Traversal

- BFS (Breadth First Search)
  - Start from a vertex, visit all the reachable vertices in a breadth first manner
  - Uses Queue for non-recursive implementation
- DFS (Depth First Search)
  - Start from a vertex, visit all the reachable vertices in a depth first manner
  - Uses Stack for non-recursive implementation

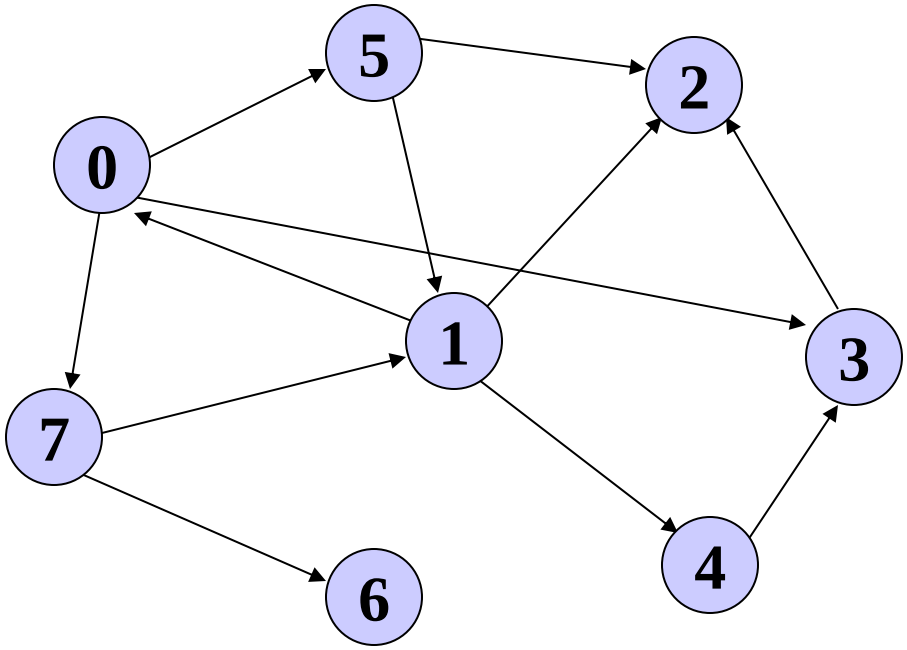
Depth-First Search

```
dfs (Node v)
1.[Push v on the stack STK]
      push(STK,v);
2.Repeat steps 3 to 5 while STK is not
  empty
3.[Pop top element from STK]
      u=pop(STK);
4.If u is not in visited list
  Add u to the list of visited nodes
5.For each w adjacent to u
   If w is not visited then    Push(STK,w);
```

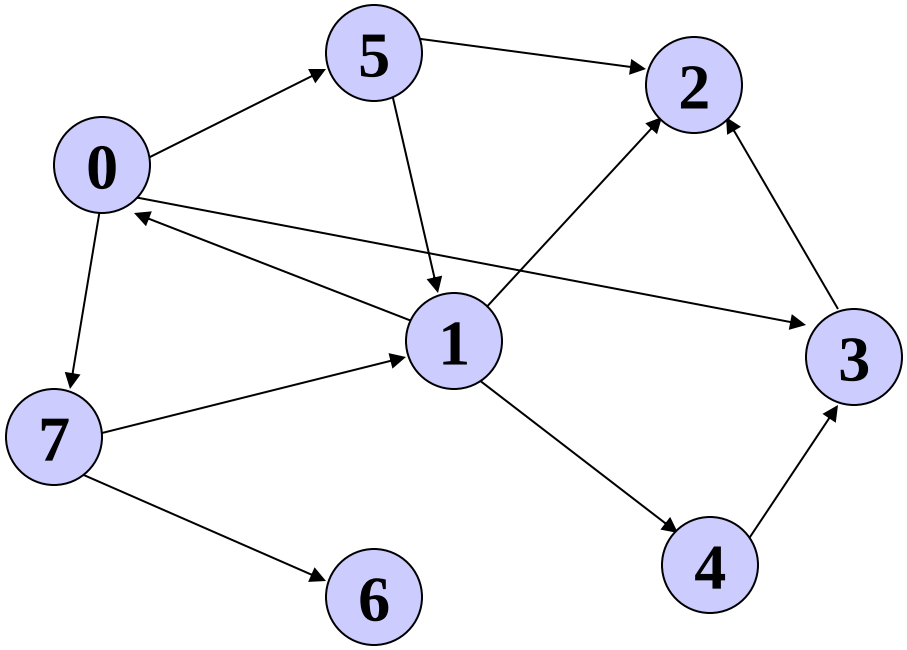
Example

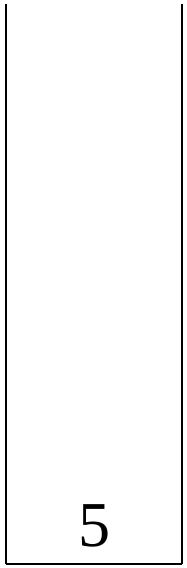


DFS: Start with Node 5



DFS: Start with Node 5

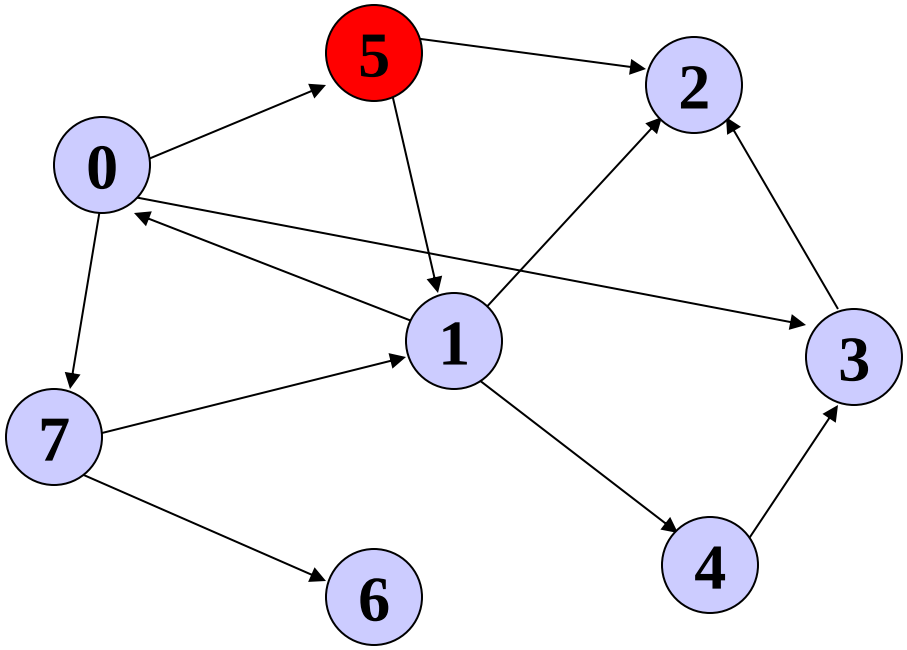


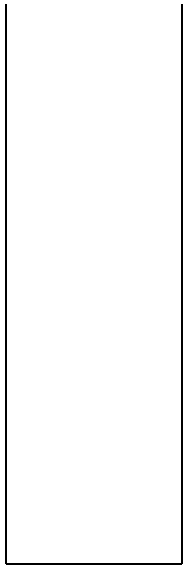


Push 5

Visited:{}

DFS: Start with Node 5

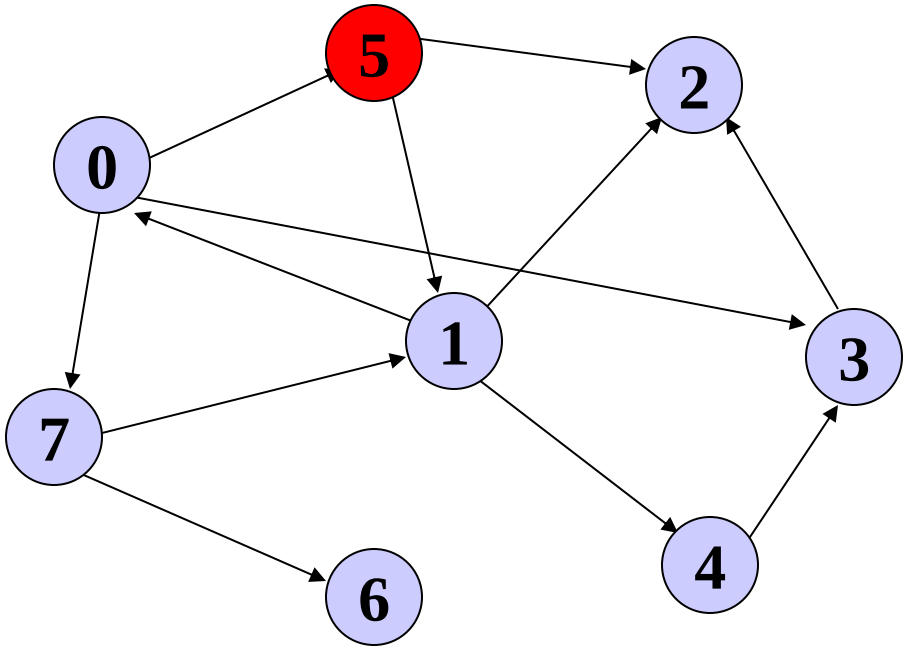


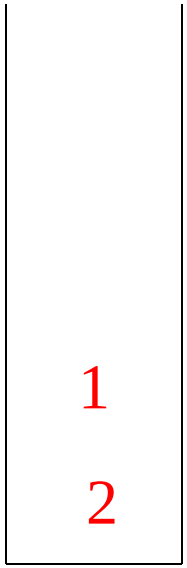


Pop/Visit 5

Visited: {5}

DFS: Start with Node 5

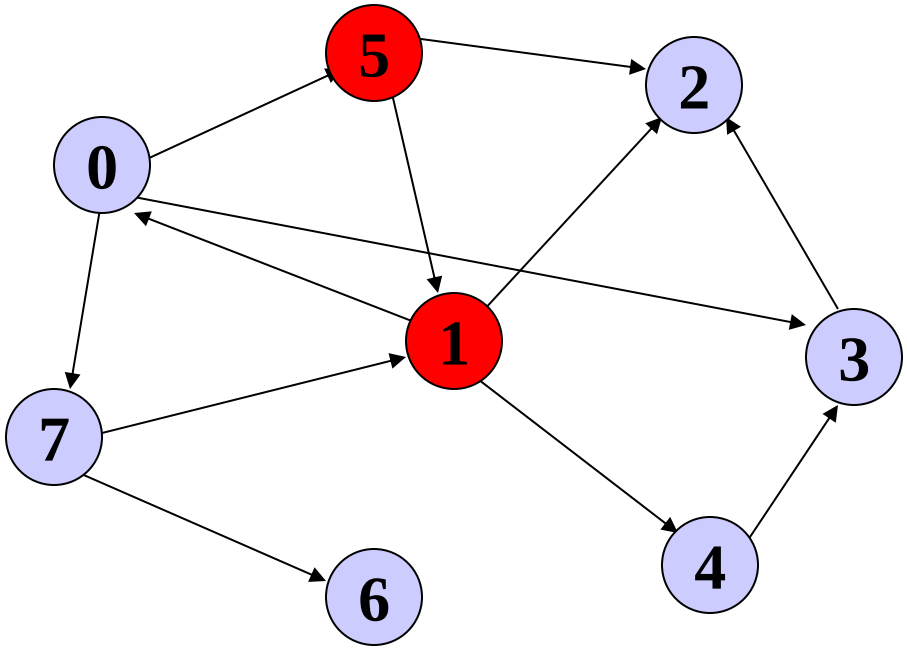


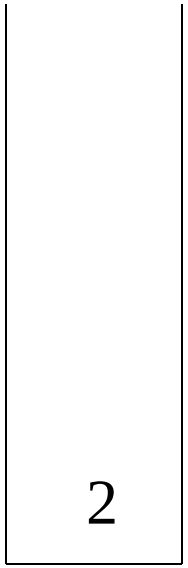


Push 2, Push 1

Visited: {5}

DFS: Start with Node 5

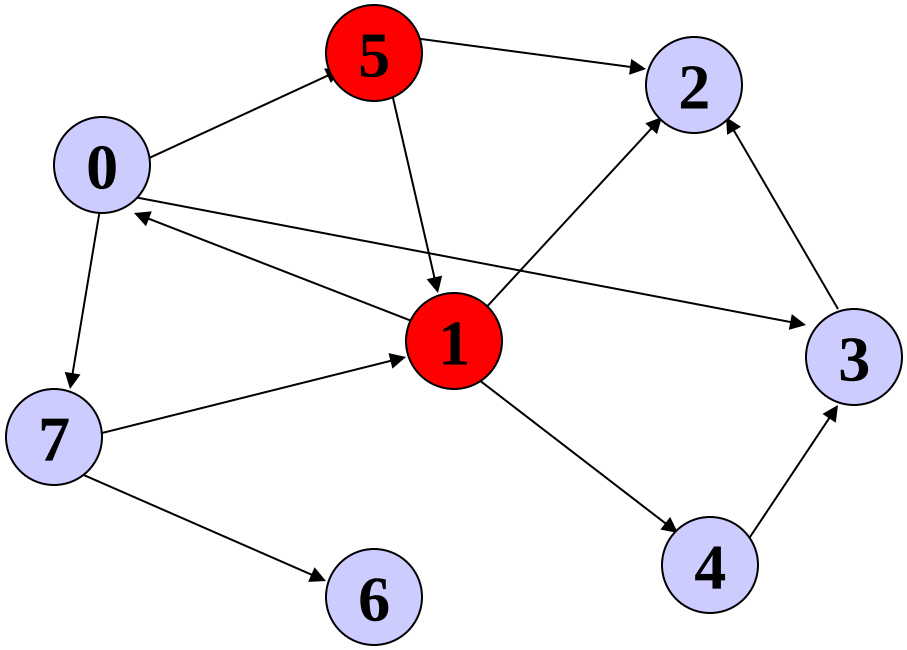




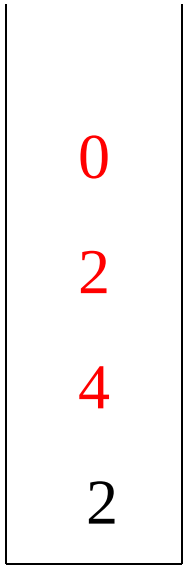
Pop/Visit 1

Visited: {5,1}

DFS: Start with Node 5

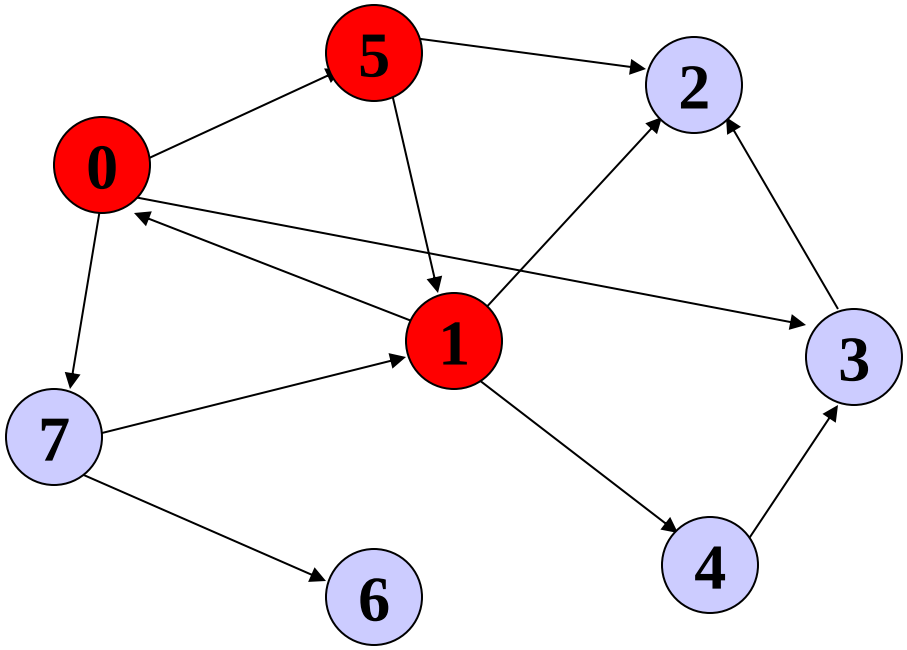


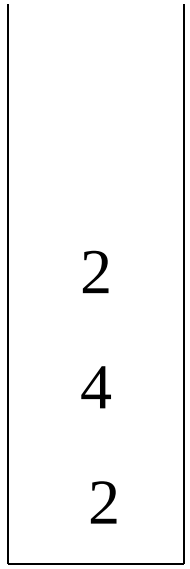
Visited: {5,1}



Push 4, Push 2,
Push 0

DFS: Start with Node 5

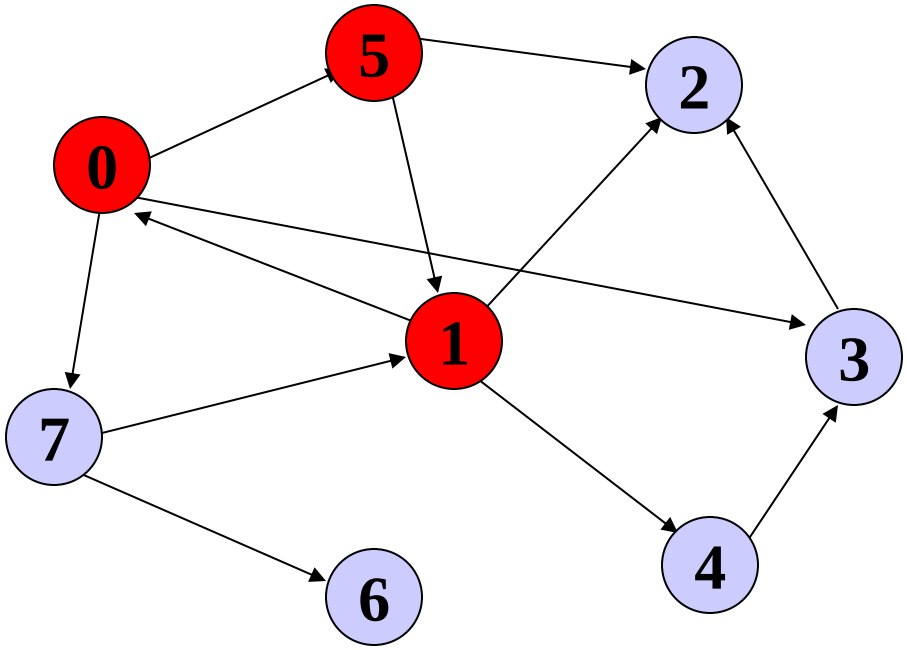


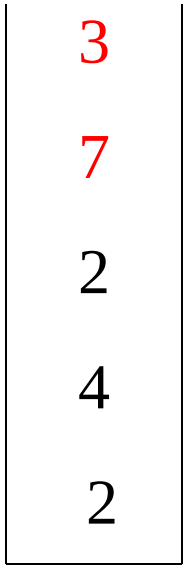


Pop/Visit 0

Visited: {5,1,0}

DFS: Start with Node 5

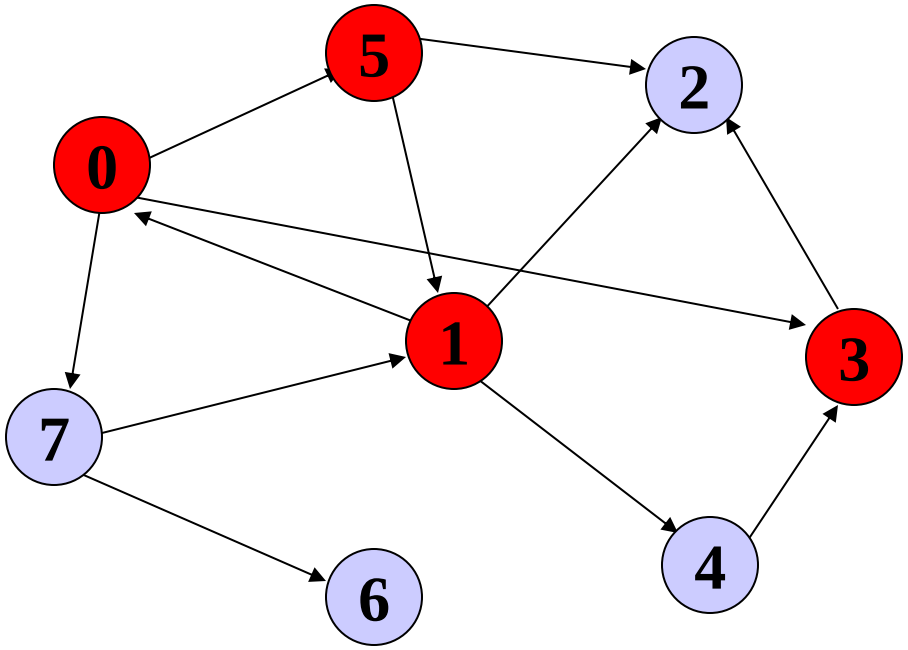


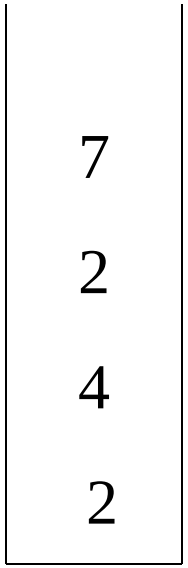


Push 7, Push 3

Visited: {5,1,0}

DFS: Start with Node 5

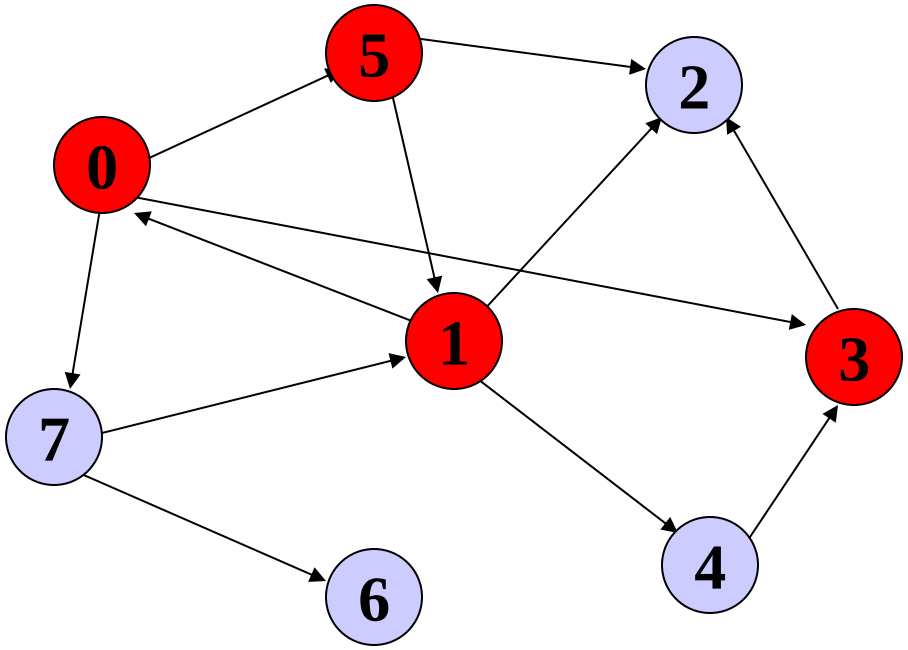


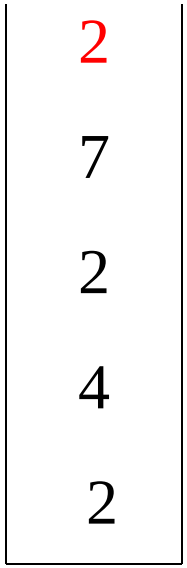


Pop/Visit 3

Visited: {5,1,0,3}

DFS: Start with Node 5

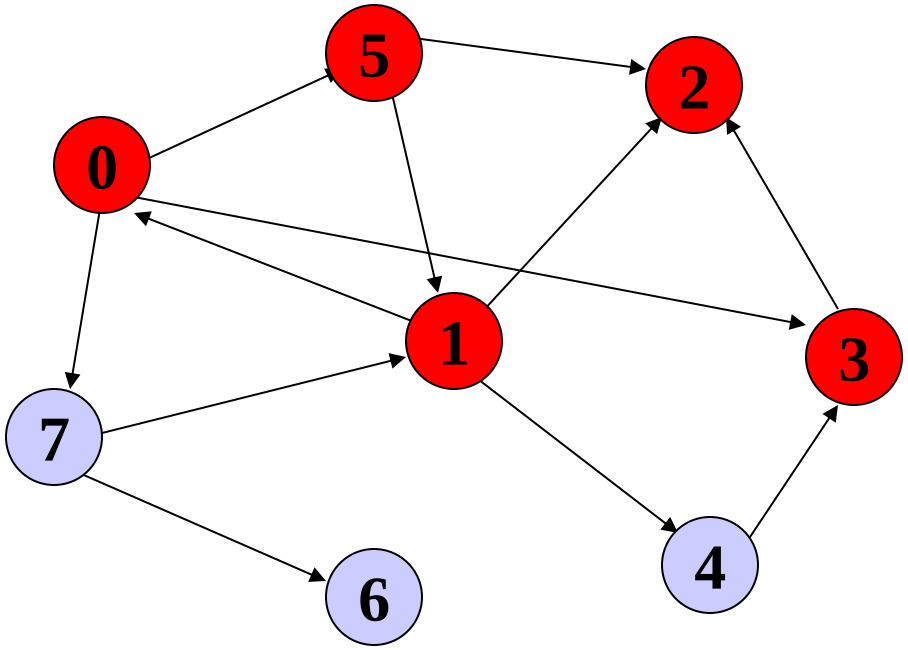


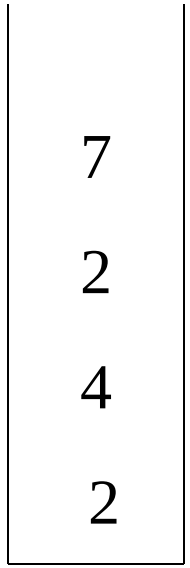


Push 2

Visited: {5,1,0,3}

DFS: Start with Node 5

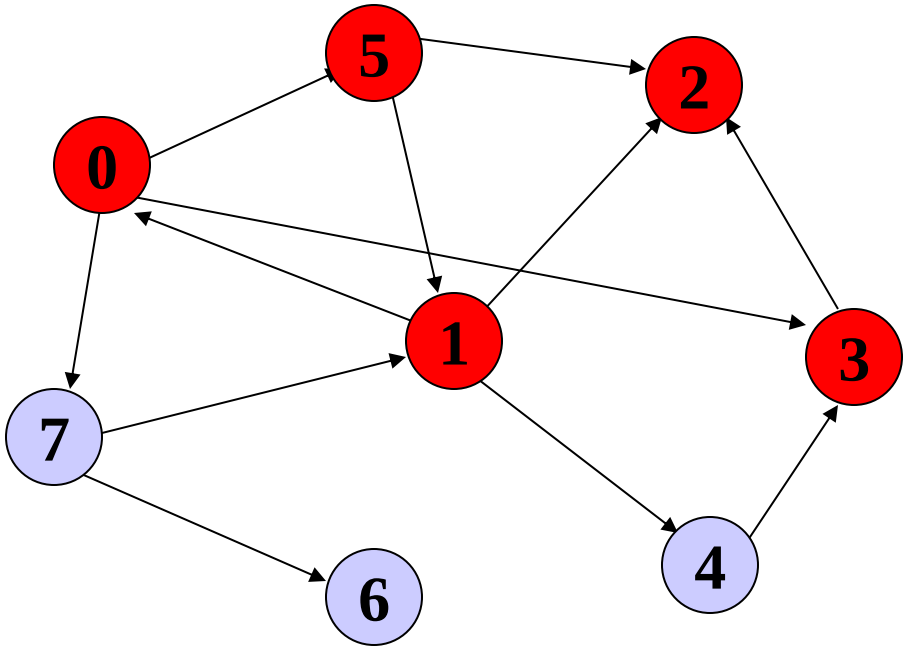


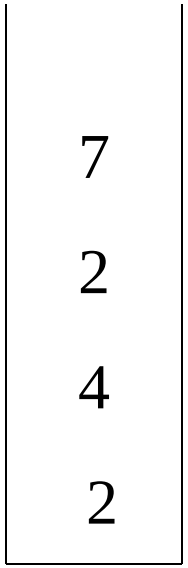


Pop/Visit 2

Visited: {5,1,0,3,2}

DFS: Start with Node 5

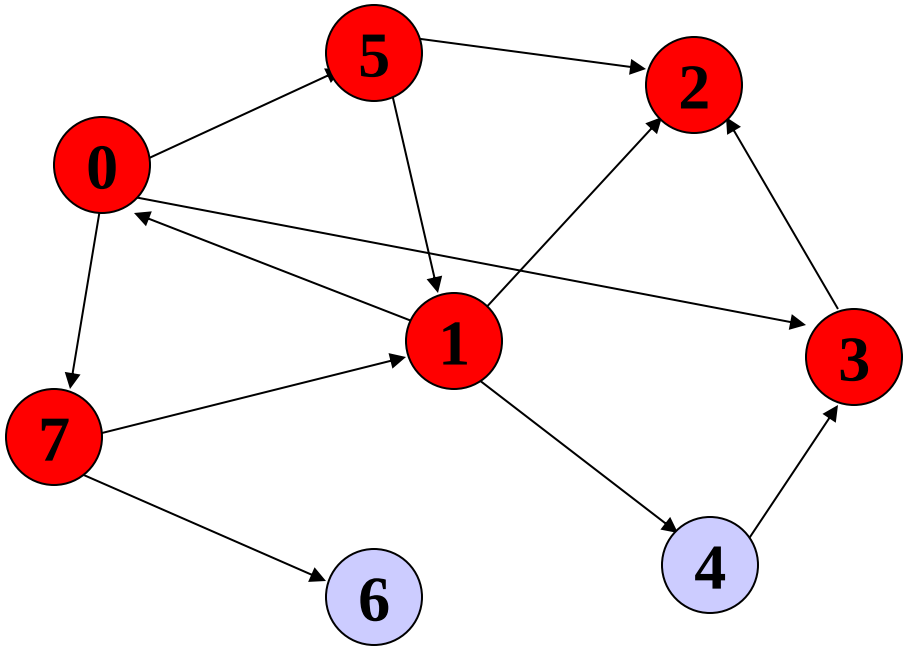


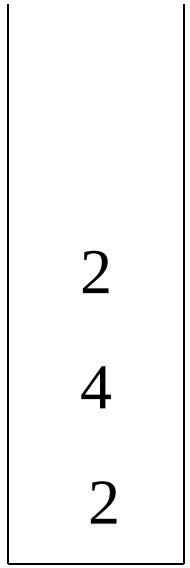


No push operation

Visited: {5,1,0,3,2}

DFS: Start with Node 5

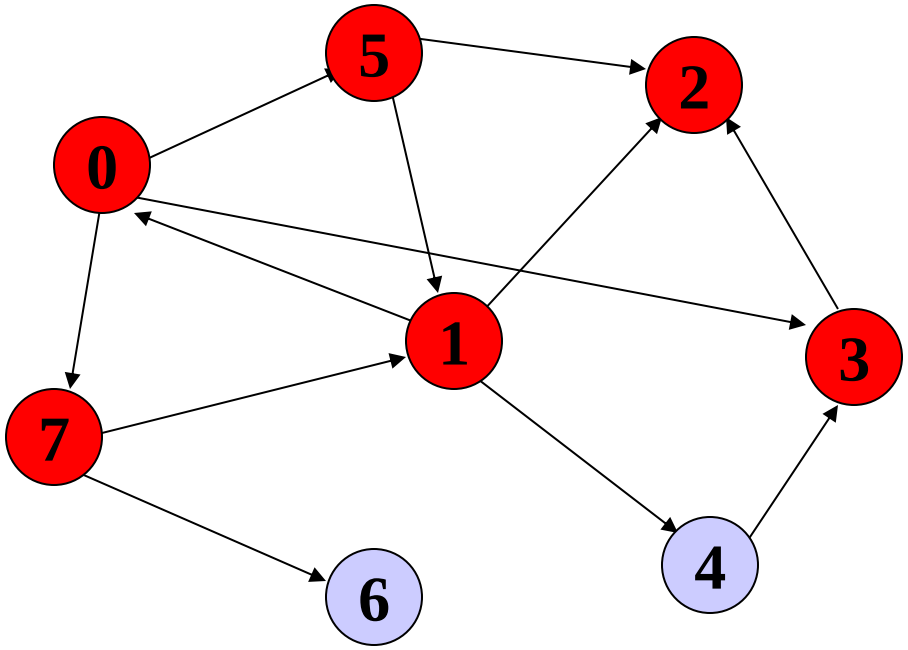


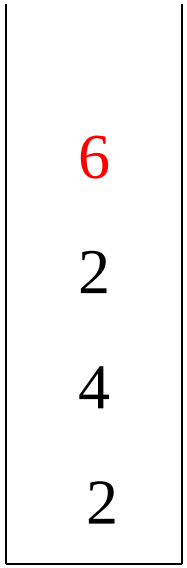


Pop/Visit 7

Visited: {5,1,0,3,2,7}

DFS: Start with Node 5

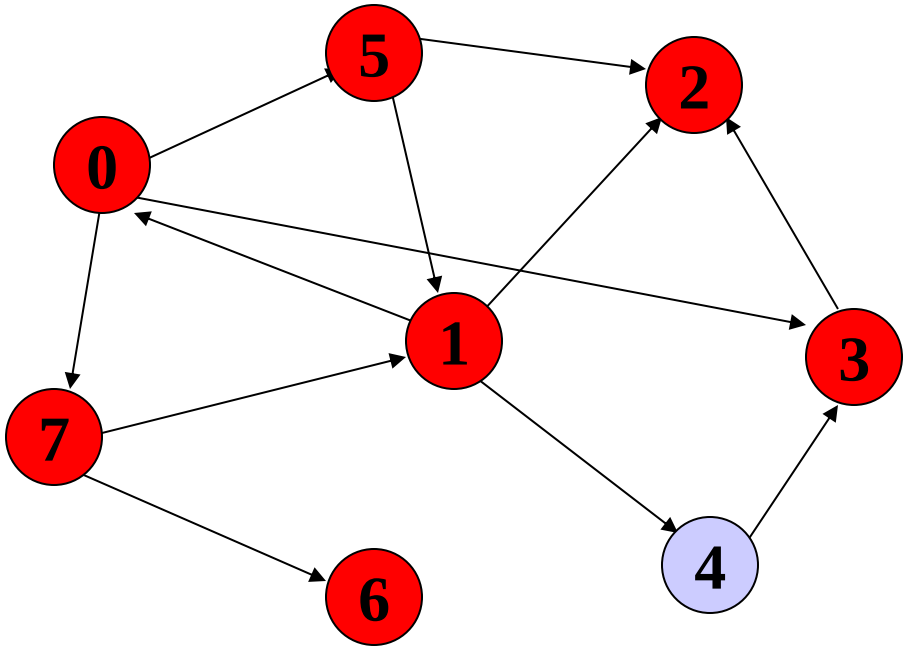


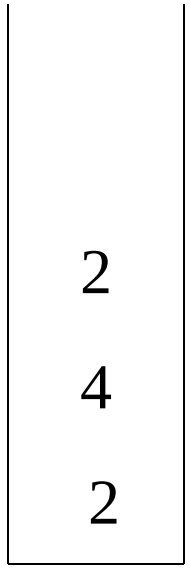


Push 6

Visited: {5,1,0,3,2,7}

DFS: Start with Node 5

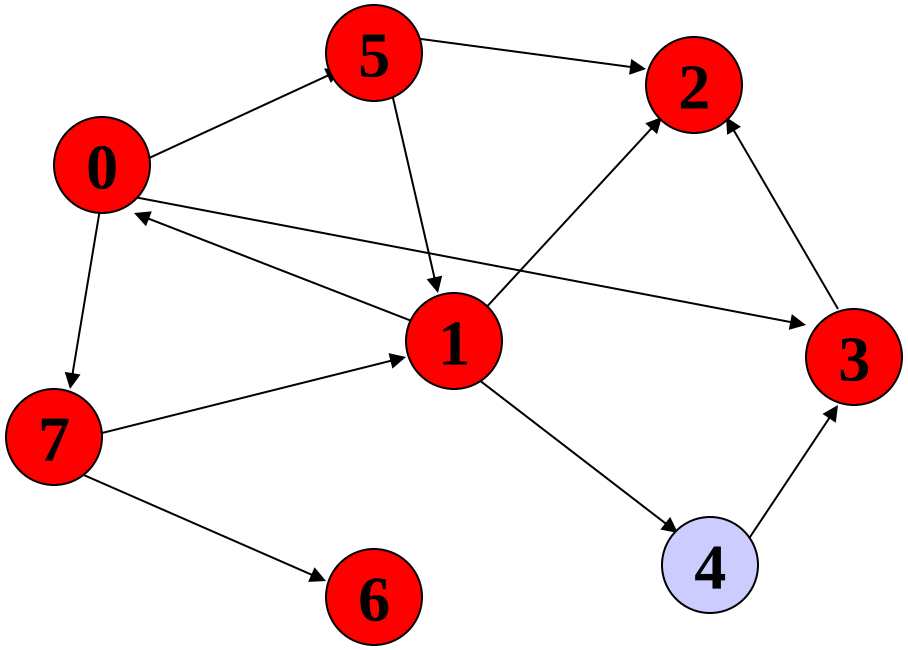


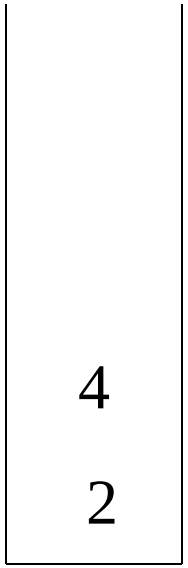


Pop/Visit 6

Visited: {5,1,0,3,2,7,6}

DFS: Start with Node 5

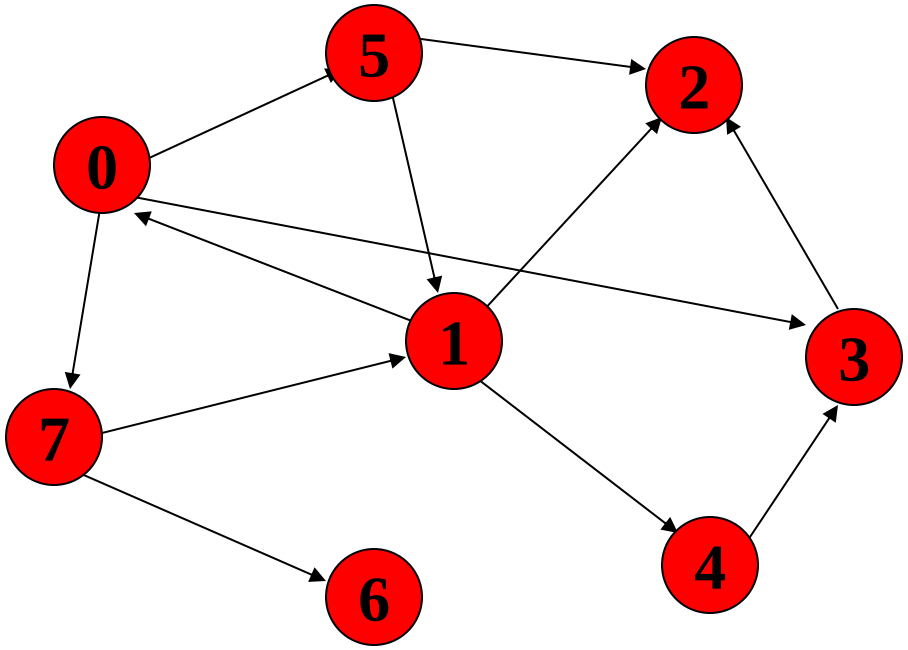


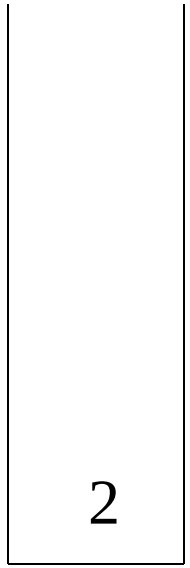


Pop (don’t visit) 2

Visited: {5,1,0,3,2,7,6}

DFS: Start with Node 5

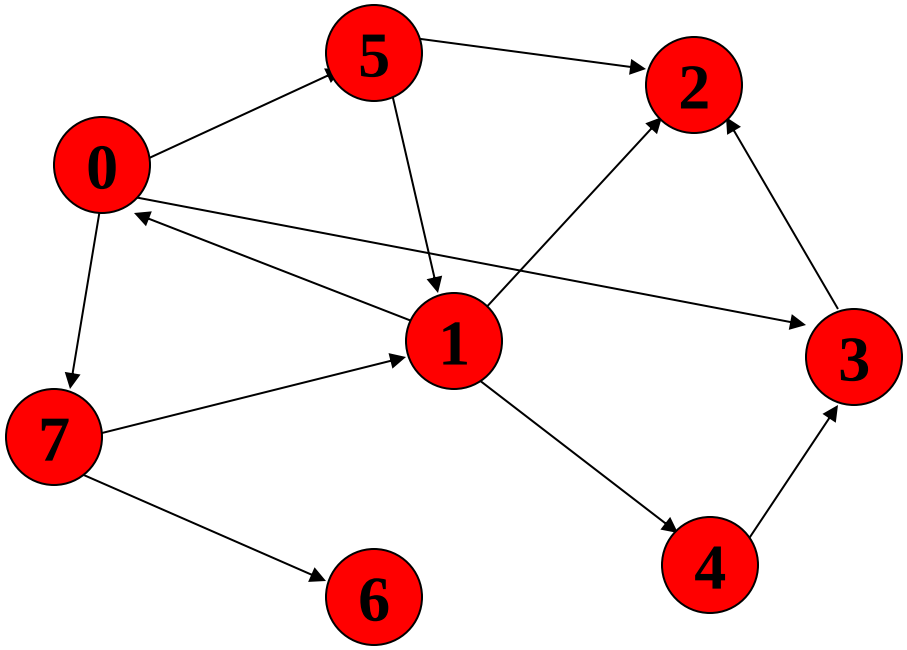


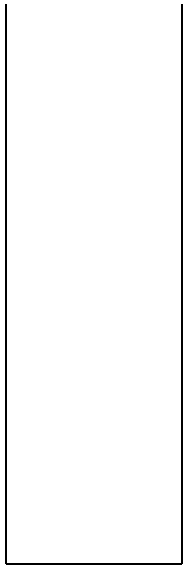


Pop/Visit 4

Visited: {5,1,0,3,2,7,6,4}

DFS: Start with Node 5

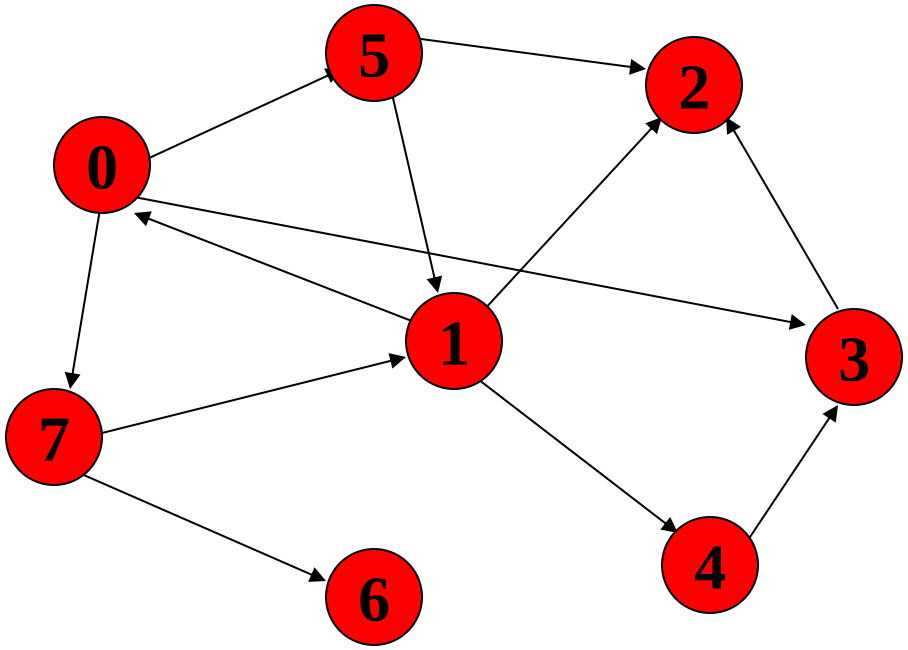


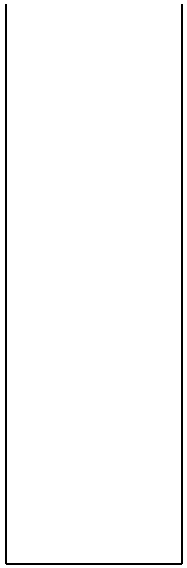


Pop (don’t visit) 2

Visited: {5,1,0,3,2,7,6,4}

DFS: Start with Node 5





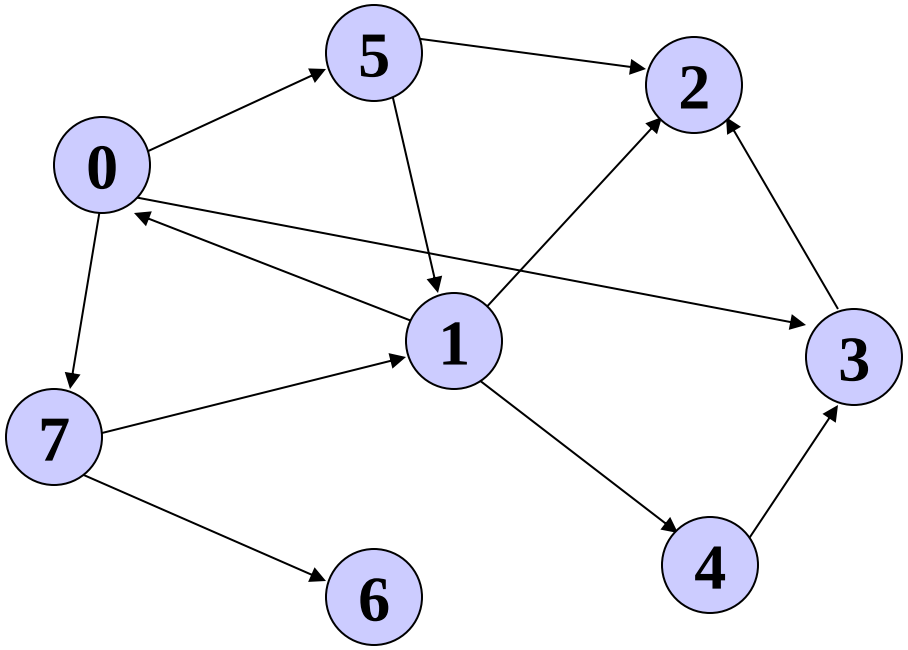
Done

Visited: {5,1,0,3,2,7,6,4}

Breadth-first Search

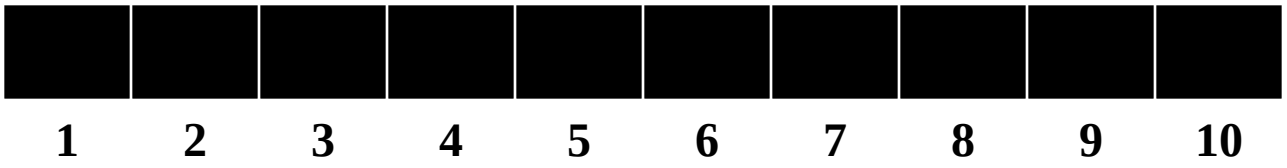
```
bfs (Node v)
1.[add v to QUEUE]
      enqueue(QUEUE,v);
2.Repeat steps 3 to 5 while QUEUE is not
  empty
3.[Remove one element from QUEUE]
      u=dequeue(QUEUE);
4.If u is not in list of visited nodes then
      Add u to list of visited nodes
5.For each w adjacent to u
   If w is not visited then
      enqueue(QUEUE,w);
```

BFS: Start with Node 5



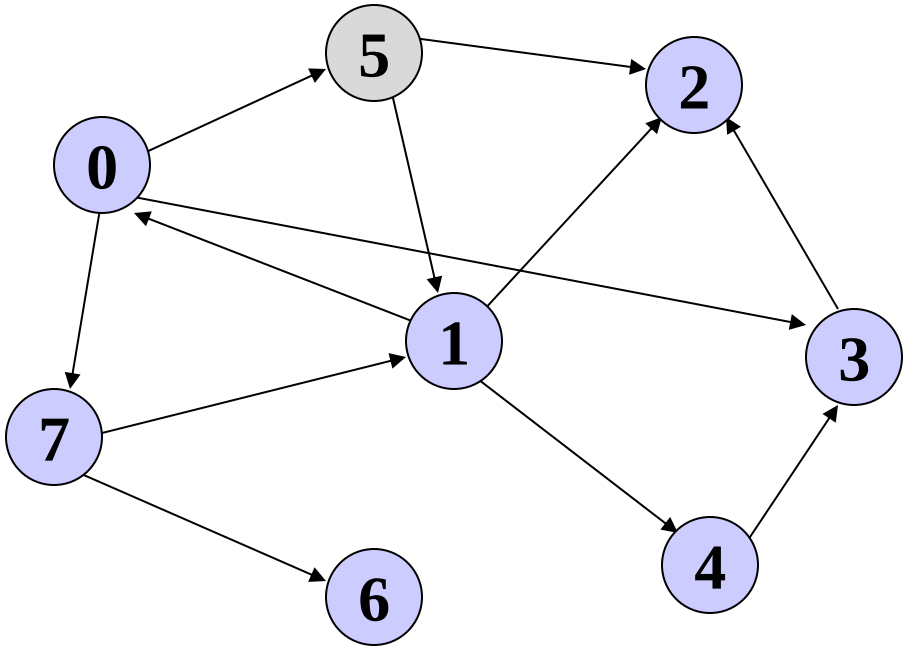
F=0

R=0



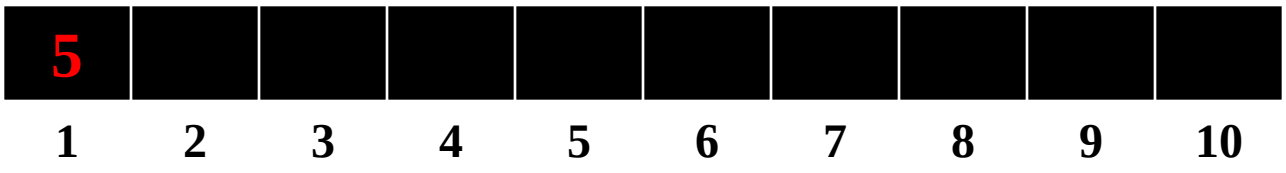
BFS: Start with Node 5

Visisted:



F=1

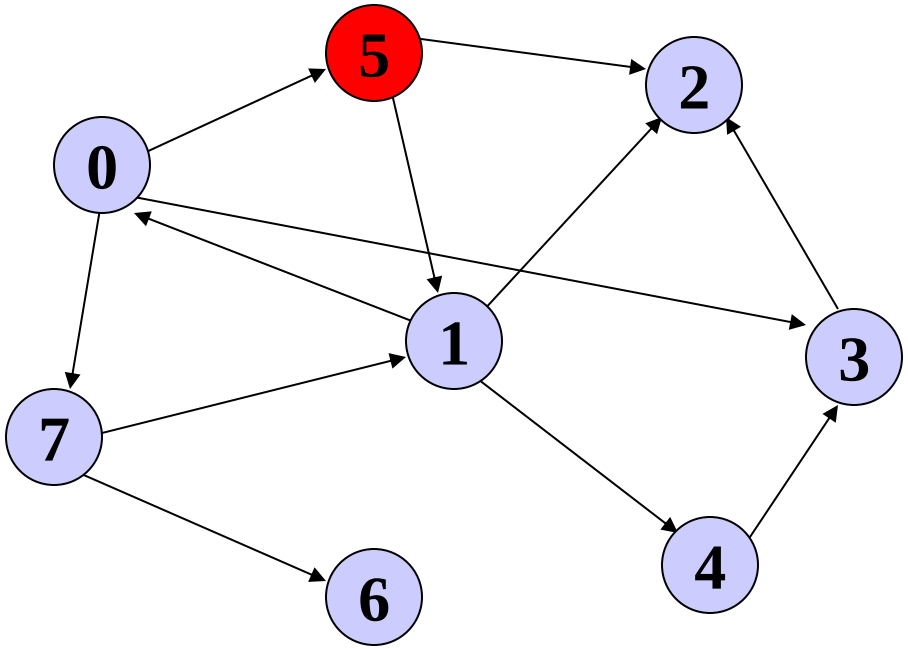
R=1



Enqueue 5

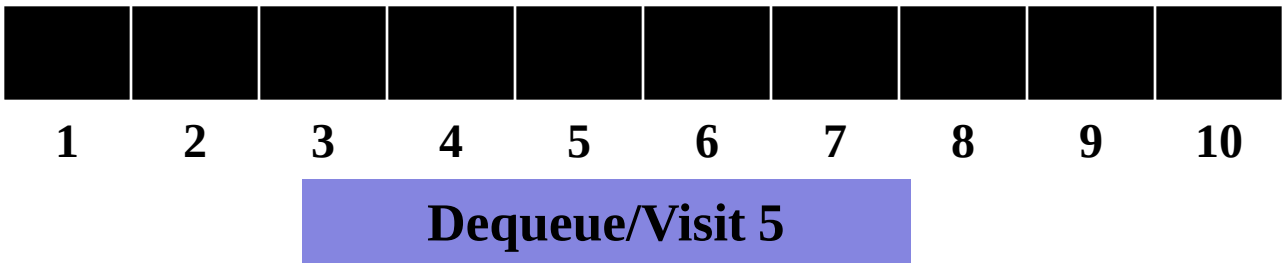
BFS: Start with Node 5

Visisted: 5



F=0

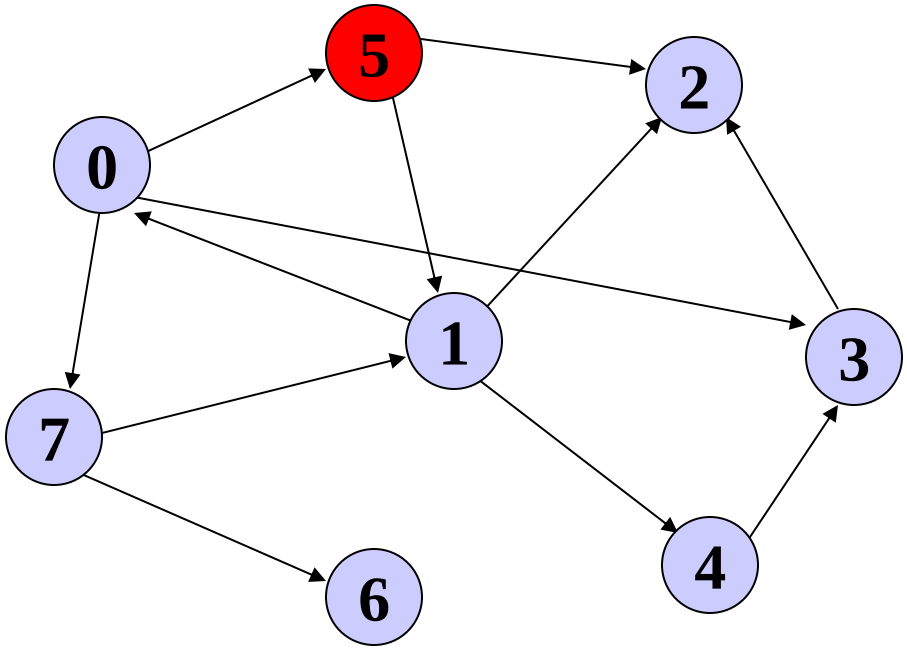
R=0



Dequeue/Visit 5

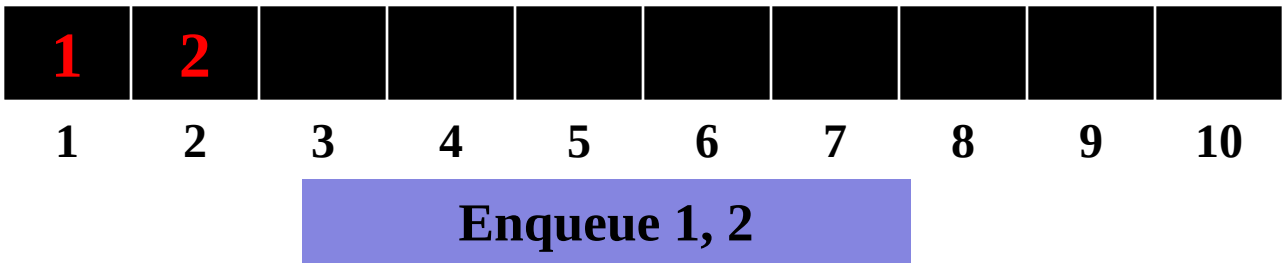
BFS: Start with Node 5

Visisted: 5



F=1

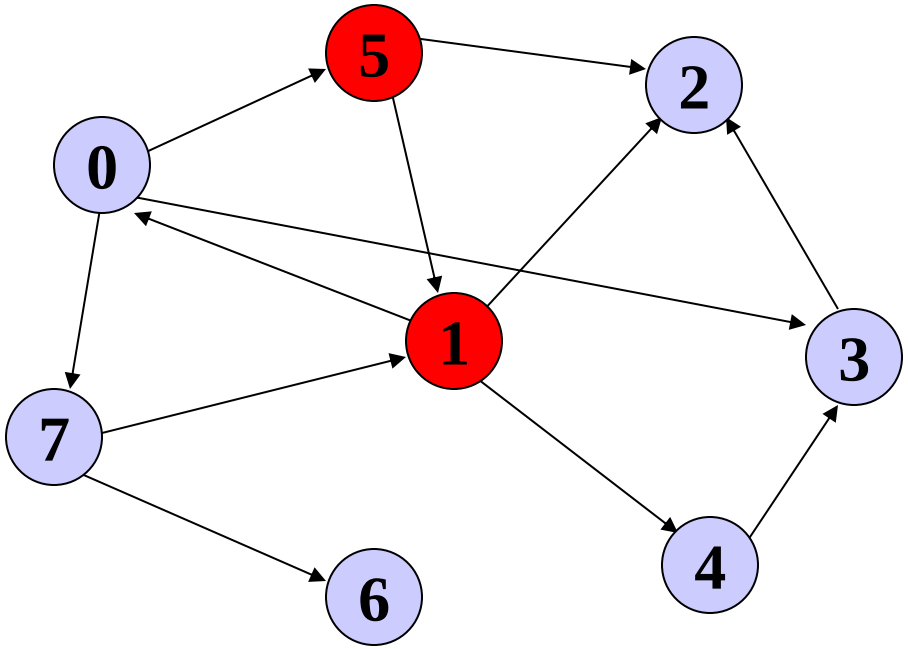
R=2



Enqueue 1, 2

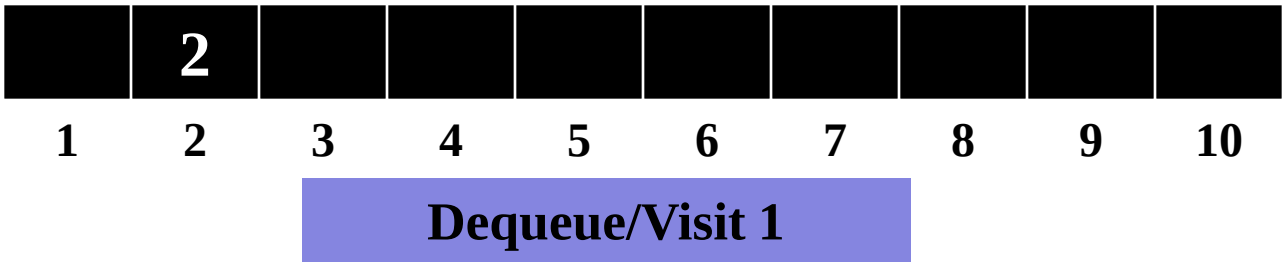
BFS: Start with Node 5

Visisted: 5 1



F=2

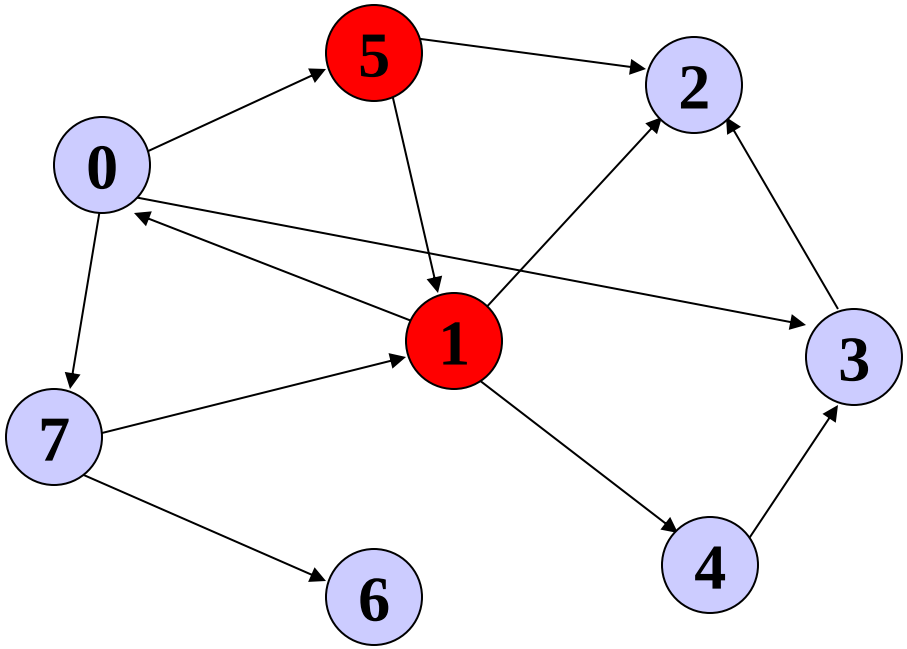
R=2



Dequeue/Visit 1

BFS: Start with Node 5

Visisted: 5 1



F=2

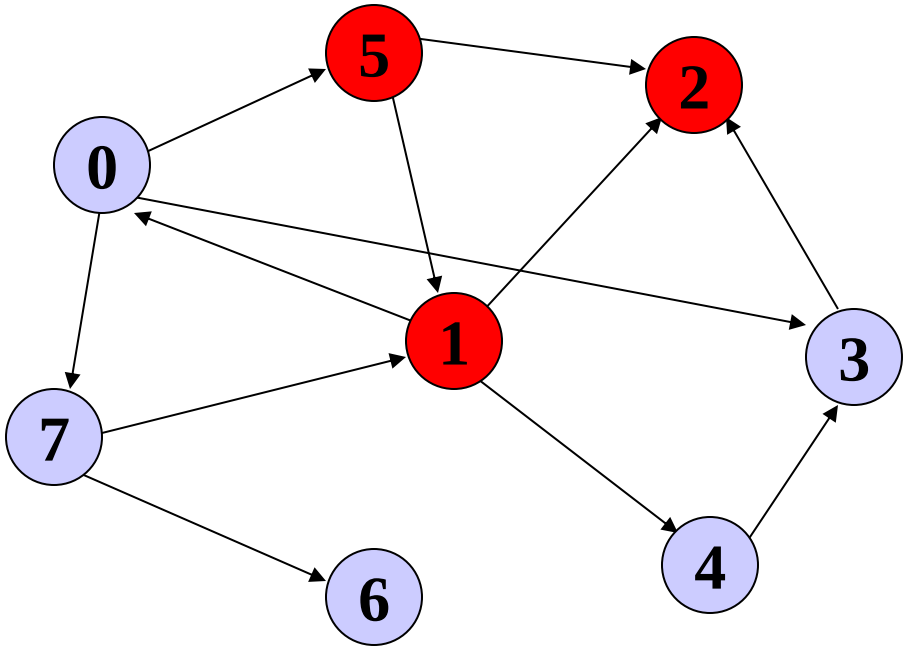
R=5

	2	0	2	4					
1	2	3	4	5	6	7	8	9	10

Enqueue 0, 2, 4

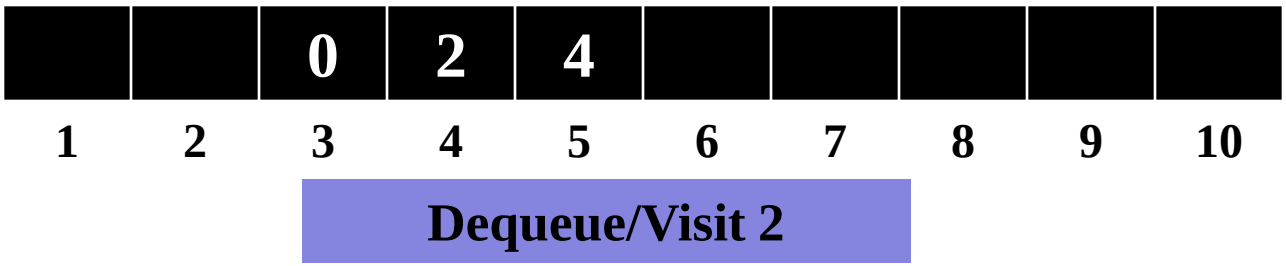
BFS: Start with Node 5

Visisted: 5 1 2



F=3

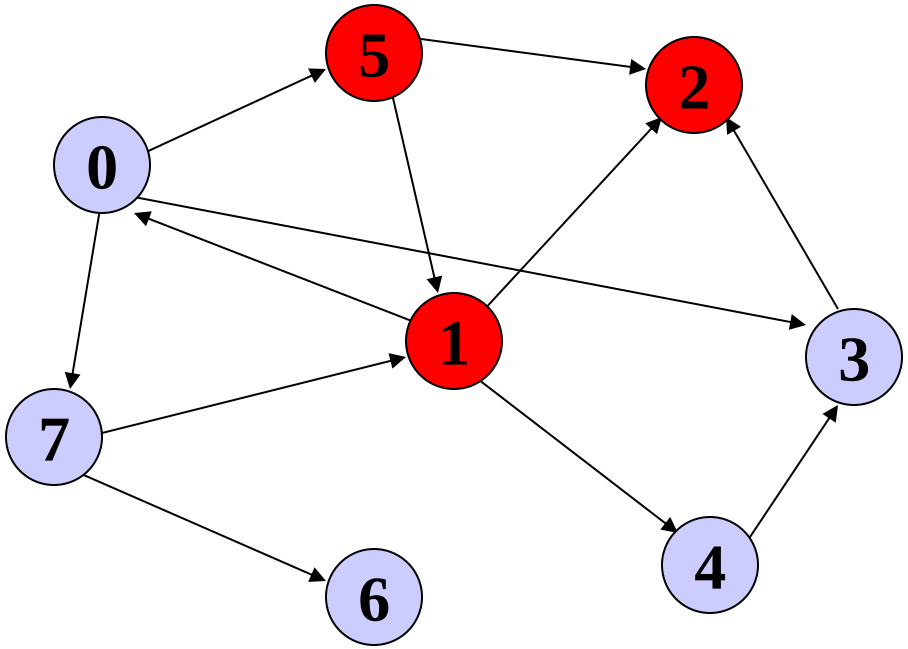
R=5



Dequeue/Visit 2

BFS: Start with Node 5

Visisted: 5 1 2



F=3

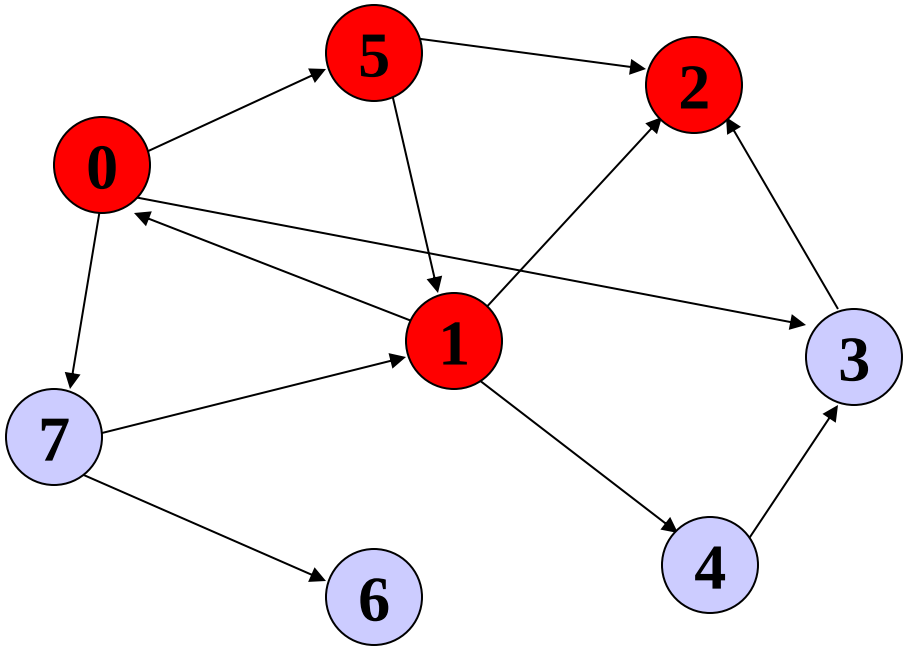
R=5

1	2	3	4	5	6	7	8	9	10
		0	2	4					

No Enqueue operation

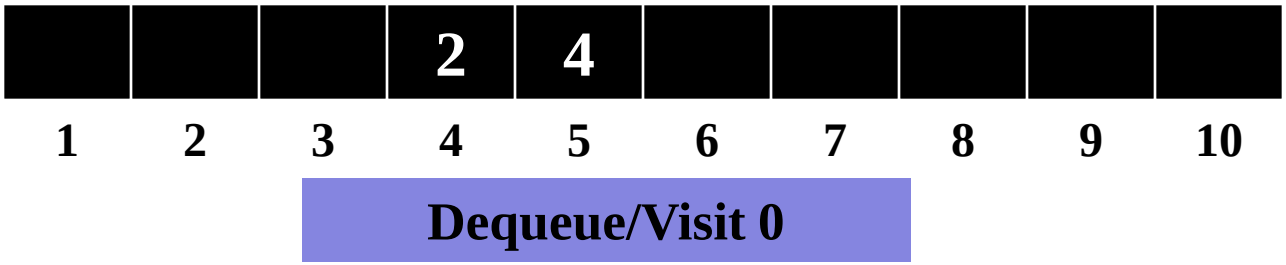
BFS: Start with Node 5

Visisted: 5 1 2 0



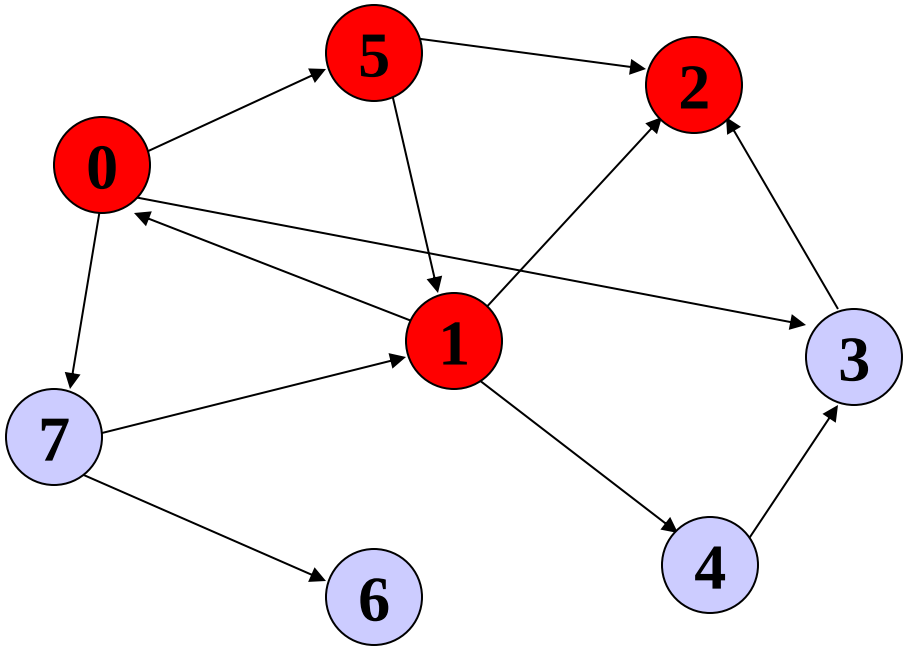
F=4

R=5



BFS: Start with Node 5

Visisted: 5 1 2 0



F=4

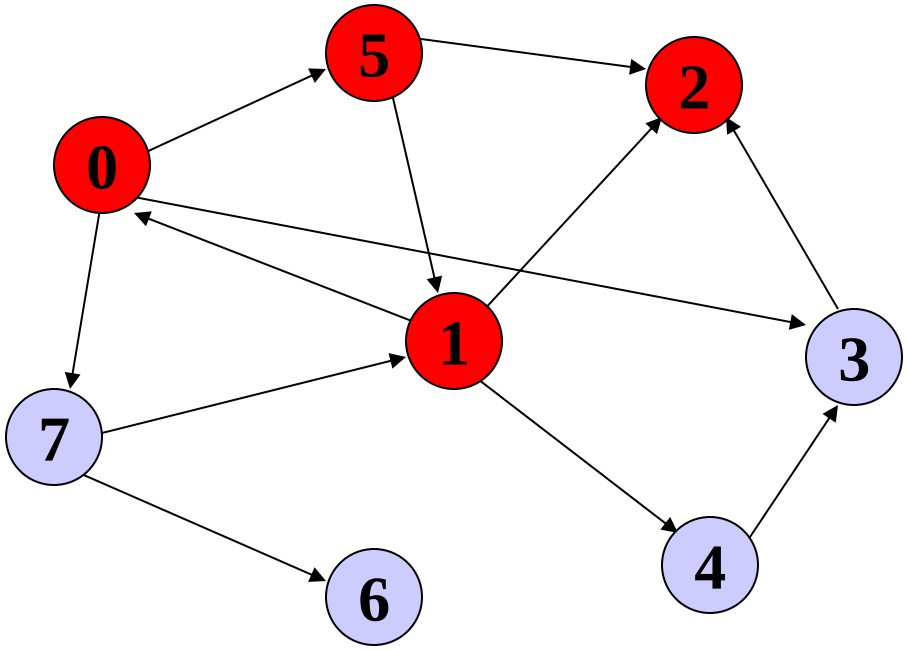
R=7

1	2	3	4	5	6	7	8	9	10
			2	4	3	7			

Enqueue 3, 7

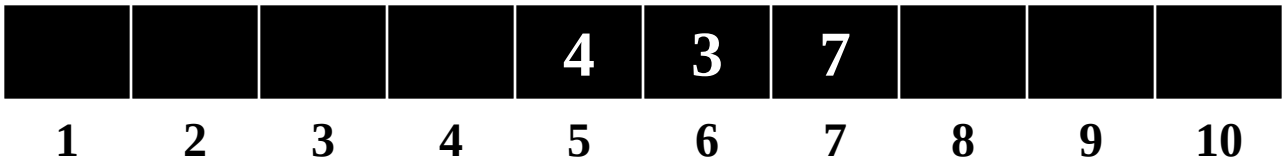
BFS: Start with Node 5

Visisted: 5 1 2 0



F=5

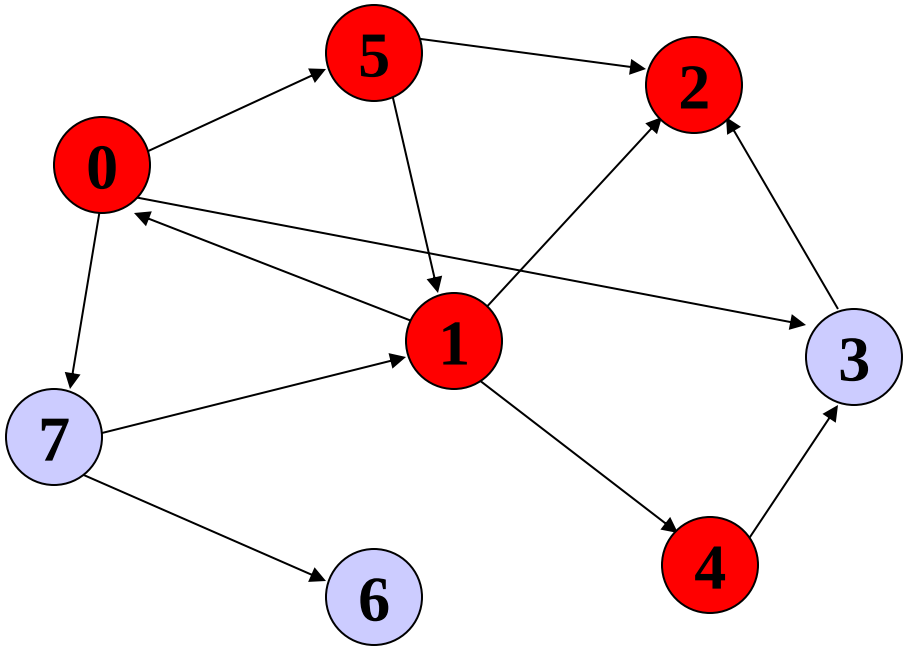
R=7



Dequeue (don’t visit) 2

BFS: Start with Node 5

Visisted: 5 1 2 0 4



F=6

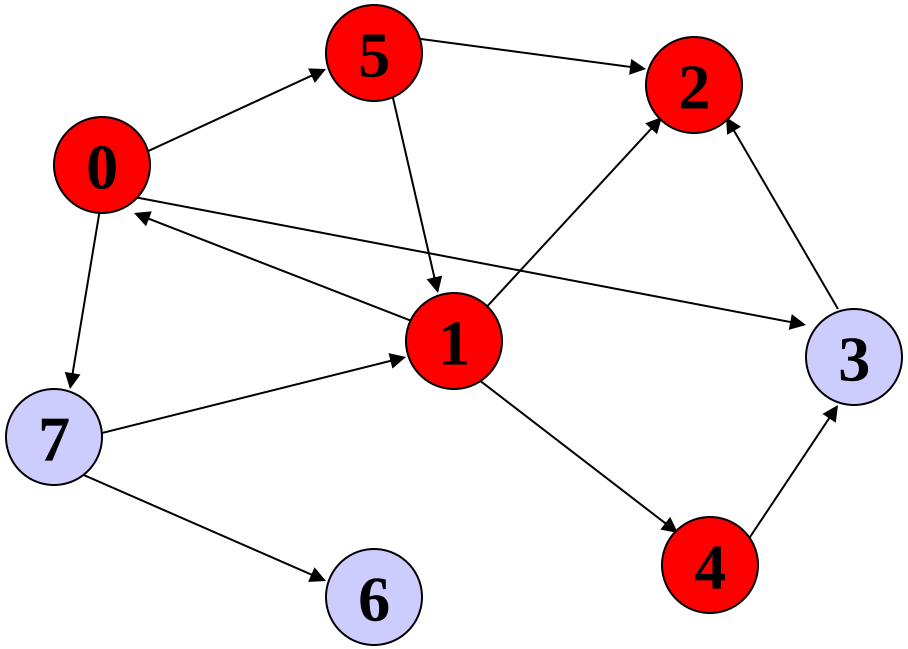
R=7

1	2	3	4	5	6	7	8	9	10
					3	7			

Dequeue/Visit 4

BFS: Start with Node 5

Visisted: 5 1 2 0 4



F=6

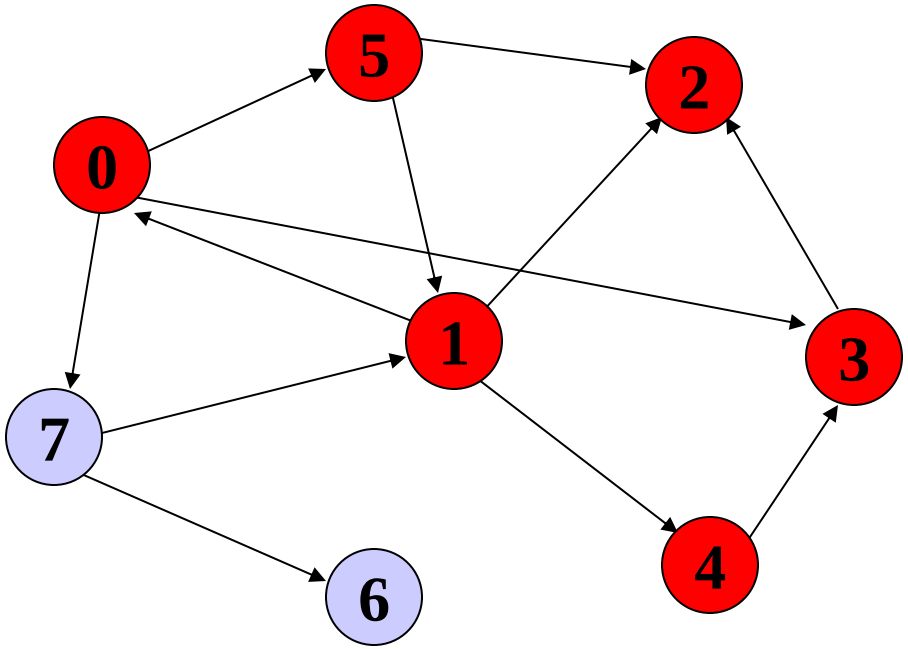
R=8

1	2	3	4	5	6	7	8	9	10
					3	7	3		

Enqueue 3

BFS: Start with Node 5

Visisted: 5 1 2 0 4 3



F=7

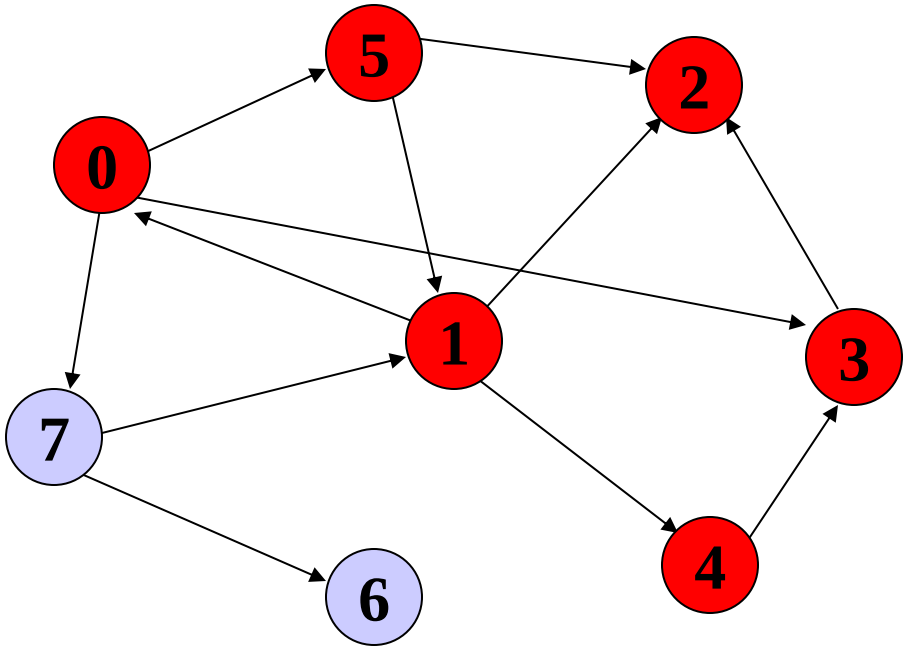
R=8

1	2	3	4	5	6	7	8	9	10
						7	3		

Dequeue/Visit 3

BFS: Start with Node 5

Visisted: 5 1 2 0 4 3



F=7

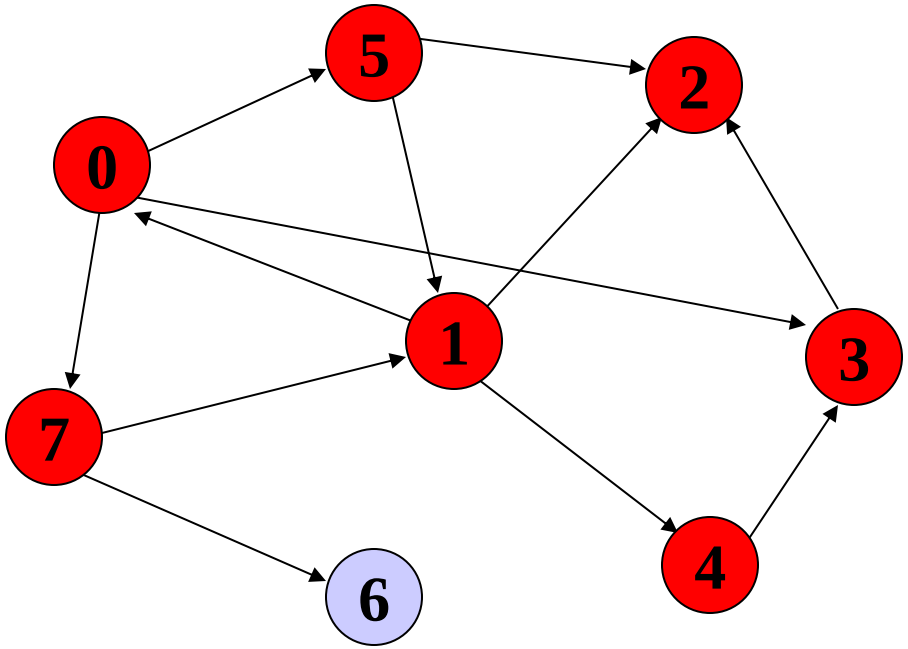
R=8

1	2	3	4	5	6	7	8	9	10
						7	3		

No Enqueue operation

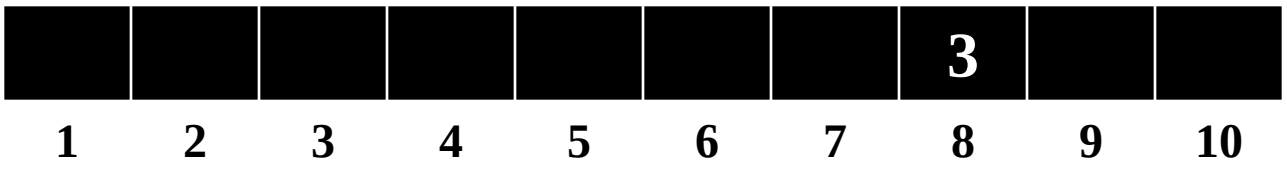
BFS: Start with Node 5

Visisted: 5 1 2 0 4 3 7



F=8

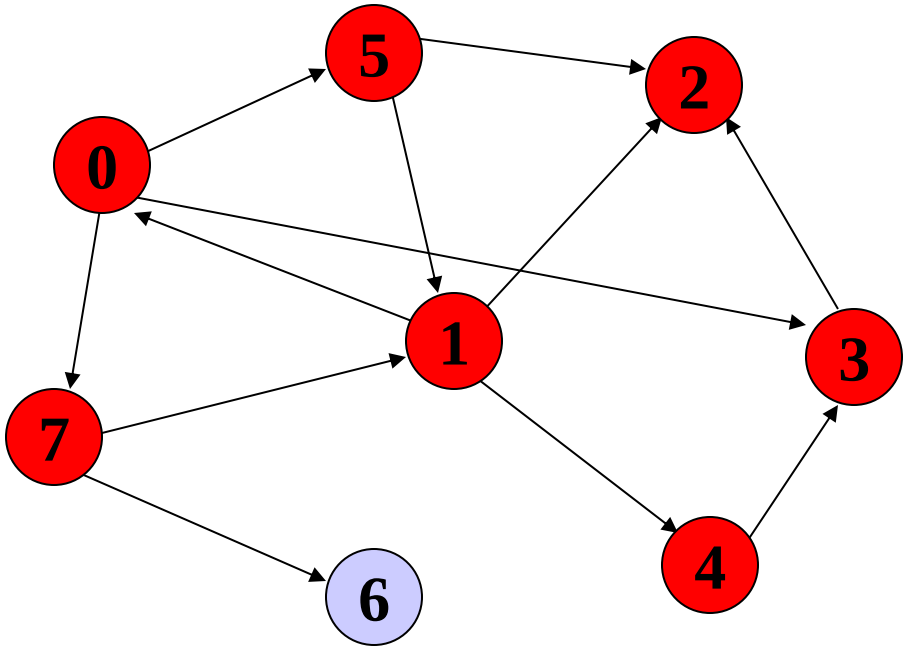
R=8



Dequeue/Visit 7

BFS: Start with Node 5

Visisted: 5 1 2 0 4 3 7



F=8

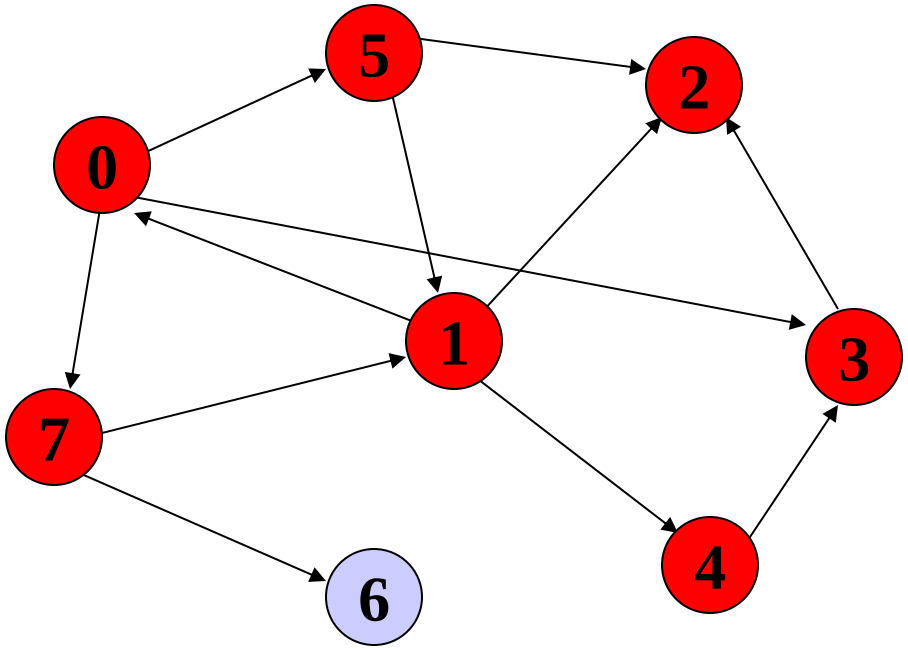
R=9

1	2	3	4	5	6	7	8	9	10
							3	6	

Enqueue 6

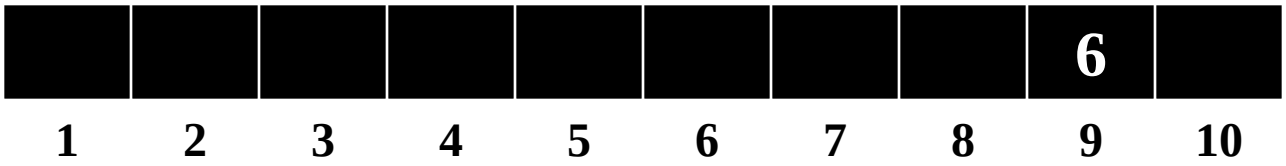
BFS: Start with Node 5

Visisted: 5 1 2 0 4 3 7



F=9

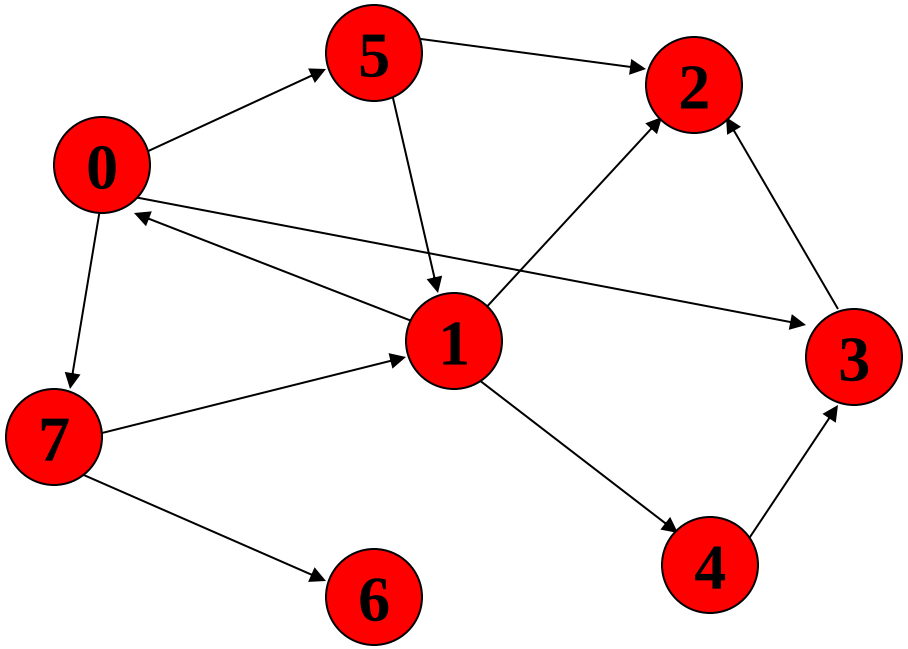
R=9



Dequeue (don’t visit) 3

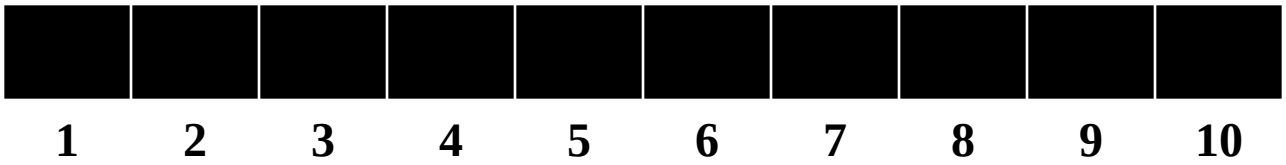
BFS: Start with Node 5

Visisted: 5 1 2 0 4 3 7 6



F=0

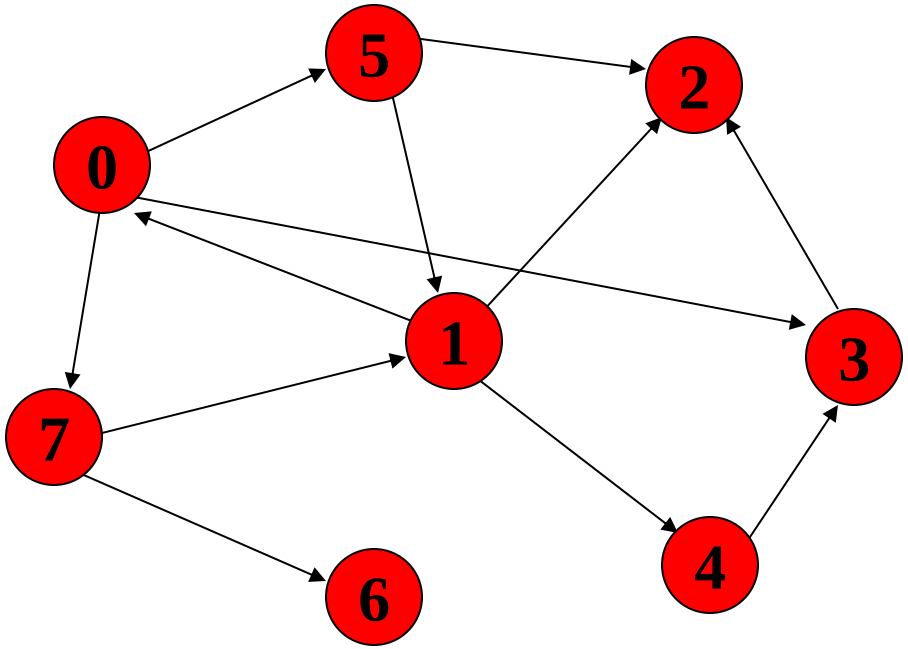
R=0



Dequeue/Visit 6

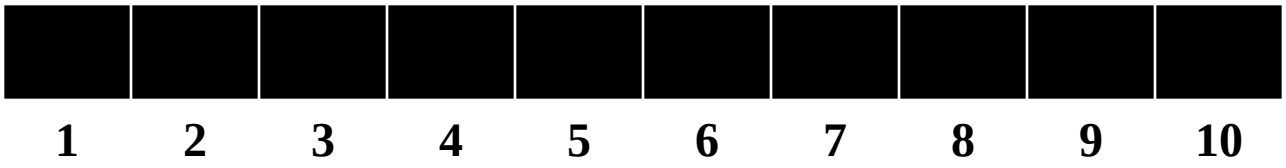
BFS: Start with Node 5

Visisted: 5 1 2 0 4 3 7 6



F=0

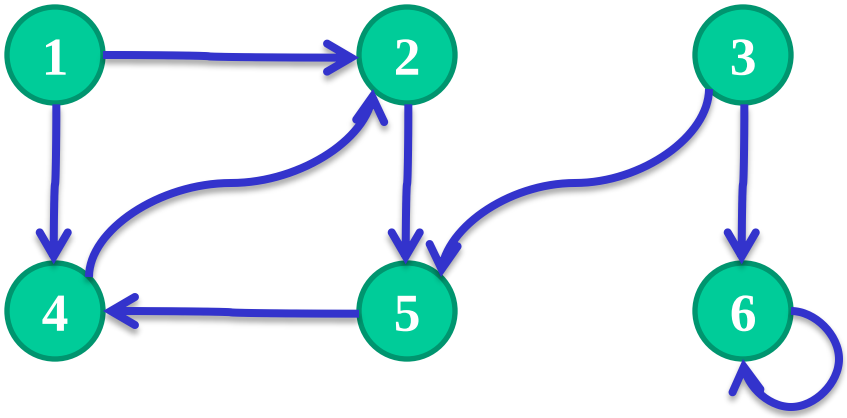
R=0



No Enqueue operation

Topological sort

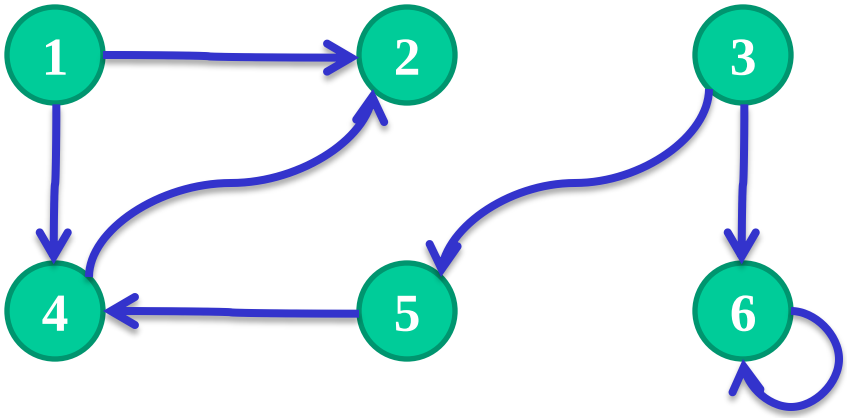
- DAG: directed acyclic graph
  - A graph without cycles



•Dag? •No

Topological sort

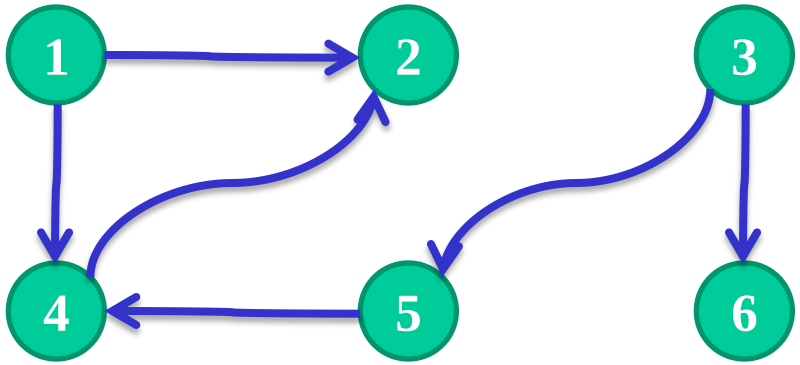
- DAG: directed acyclic graph
  - A graph without cycles



•Dag? **•No**

Topological sort

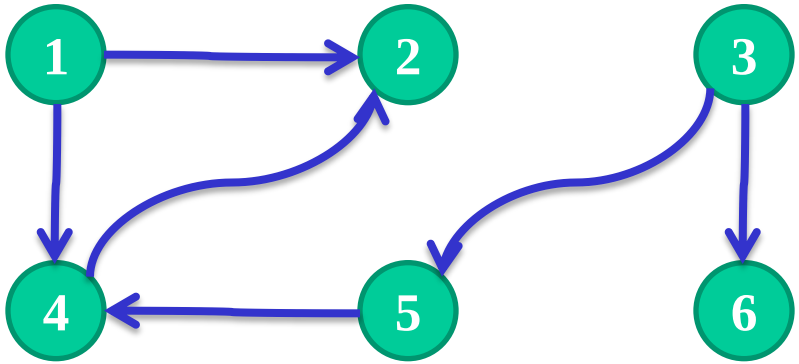
- Ordering in DAGs
  - If there is an edge <u, v>, then u appears before v in the ordering

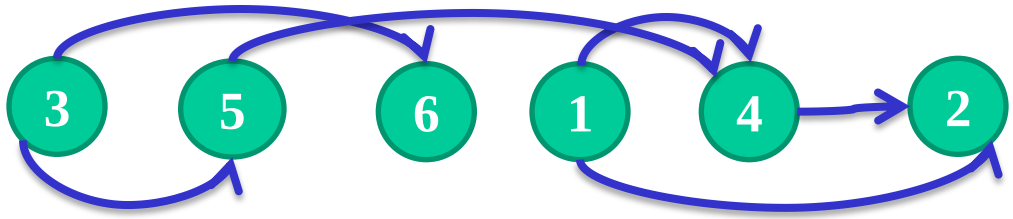


•Dag? •Yes

Topological sort

- Example



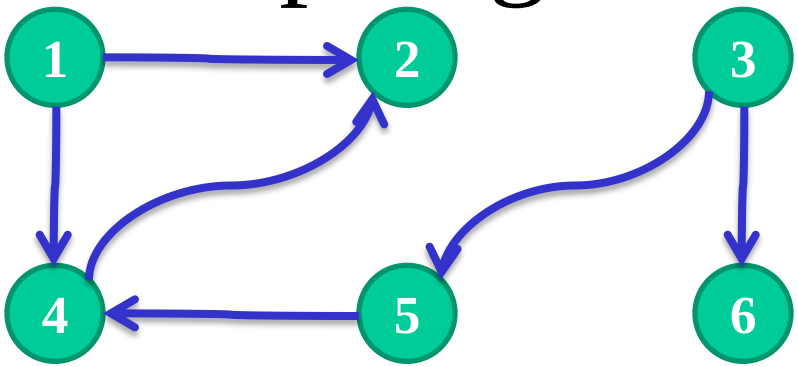


•Put all the topological sorted vertices in a line, all edges go from left to right

Topological sort

1. Store all vertices with indegree 0 in a queue Q
2. Perform steps 3 and 4 while the queue is not empty
   3. **get** a vertex U from Q and place it in the sorted sequence, L
   4. For all edges (U,V) decrease the indegree of V, and **put** V in Q if the updated indegree is 0.
5. If the graph has edges then
   - Print “Cycle exists” and EXIT
6. Else
   - Return topological sorted order, L

Topological sort



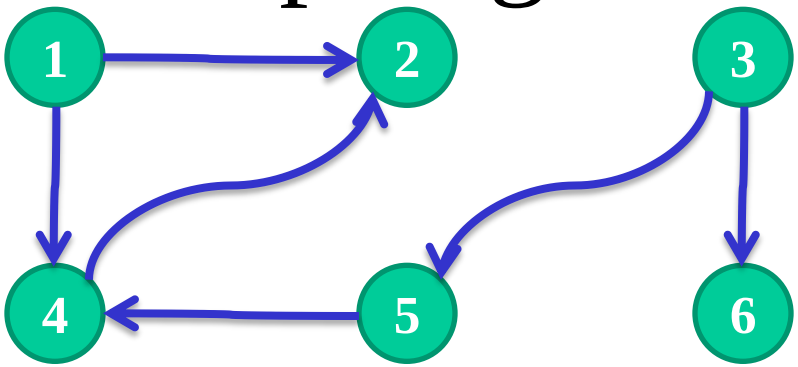
Node	Indegree
1	0
2	2
3	0
4	2
5	1
6	1

F=0

R=0

Enqueue 1, 3

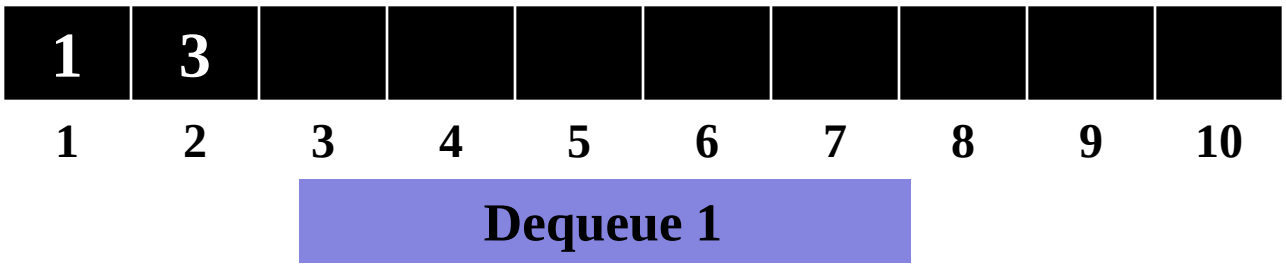
Topological sort



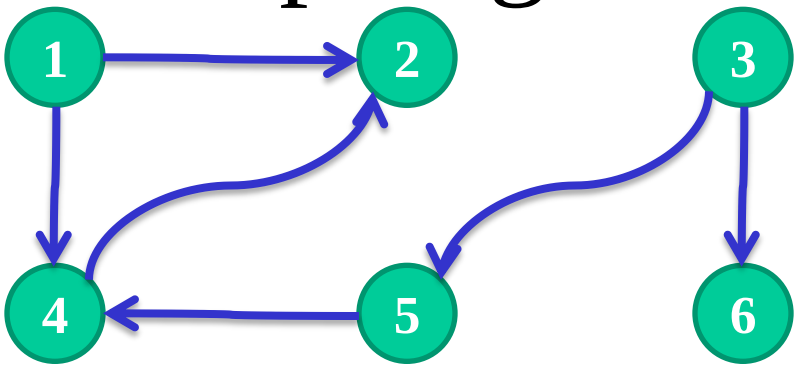
Node	Indegree
2	2
4	2
5	1
6	1

F=0

R=2



Topological sort

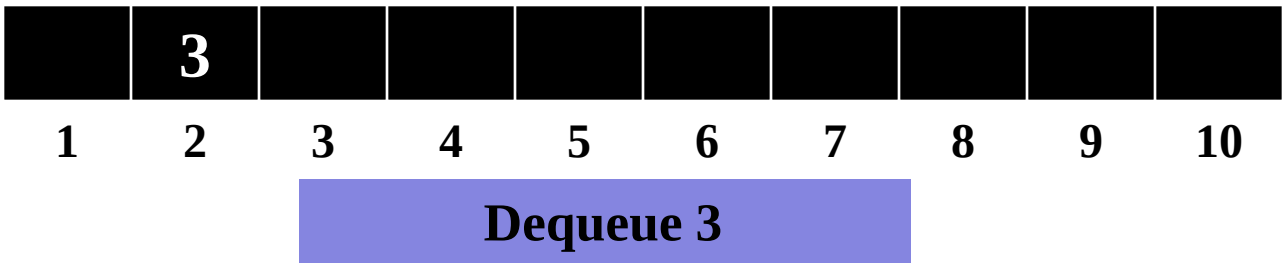


Node	Indegree
2	~~2~~ 1
4	~~2~~ 1
5	1
6	1

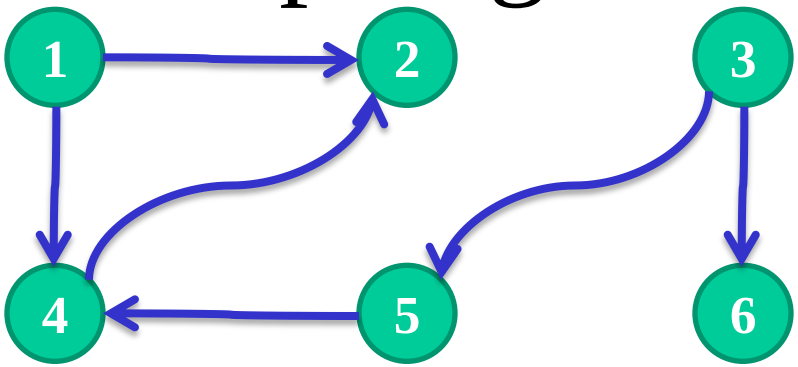
L: 1

F=1

R=2



Topological sort



Node	Indegree
2	~~2~~ 1
4	~~2~~ 1
5	~~1~~ 0
6	~~1~~ 0

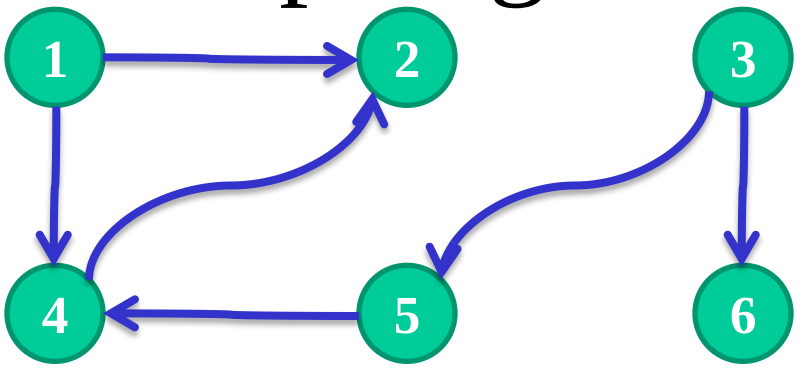
L: 1 3

F=0

R=0

Enqueue 5, 6

Topological sort

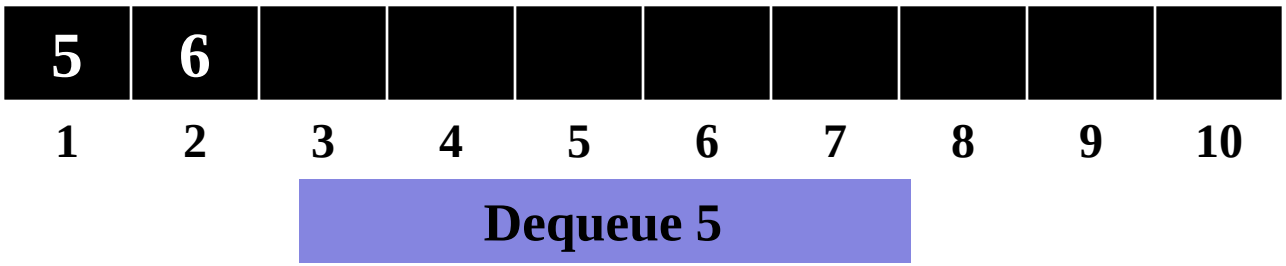


Node	Indegree
2	~~2~~ 1
4	~~2~~ 1

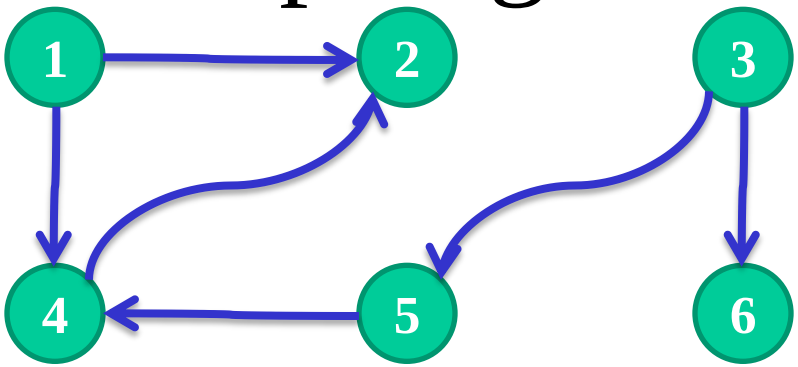
L: 1 3

F=1

R=2



Topological sort

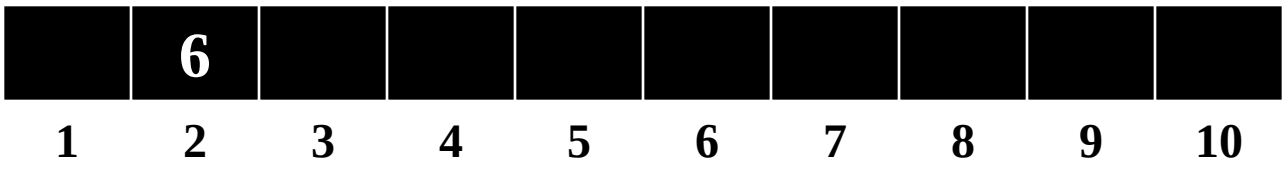


Node	Indegree
2	~~2~~ 1
4	~~2~~ ~~1~~ 0

L: 1 3 5

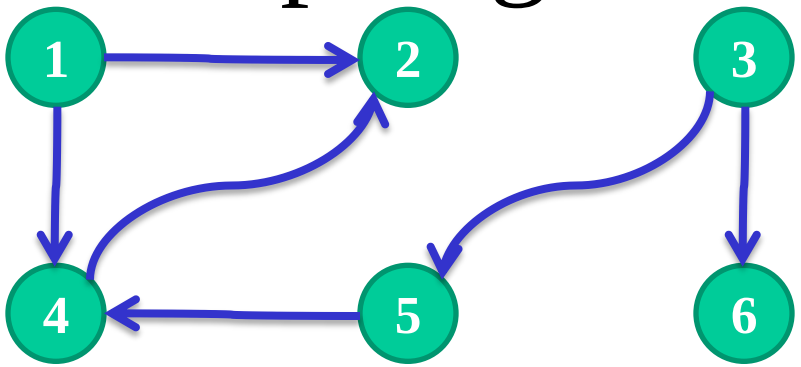
F=2

R=2



Enqueue 4

Topological sort

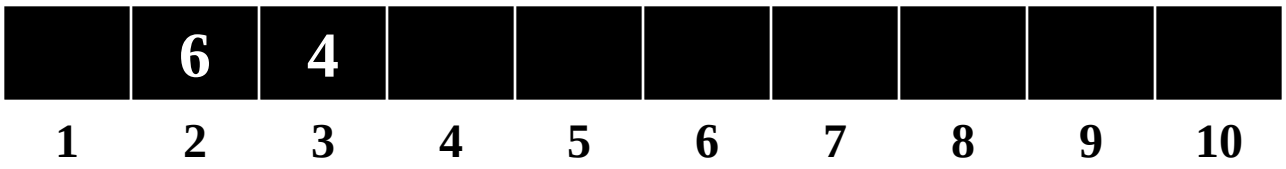


Node	Indegree
2	~~2~~ 1

L: 1 3 5

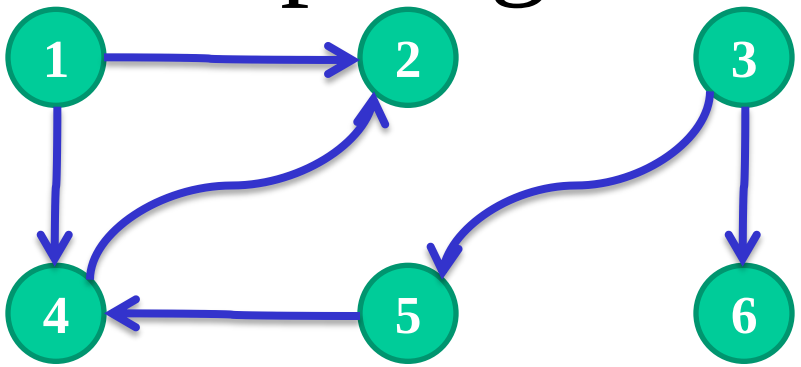
F=2

R=3



Dequeue 6

Topological sort



Node	Indegree
2	~~2~~ 1

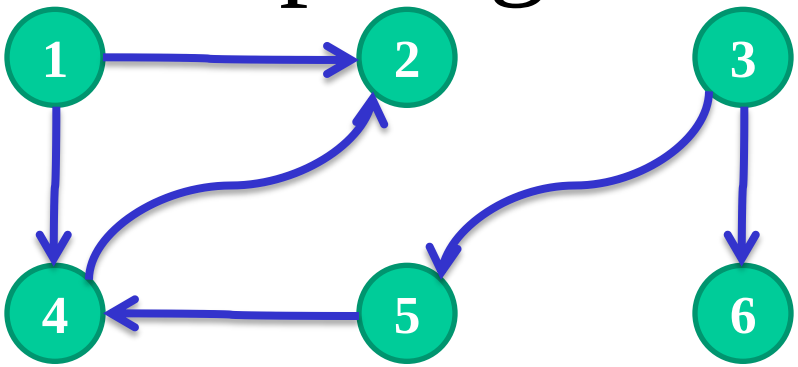
L: 1 3 5 6

F=3

R=3

Dequeue 4

Topological sort

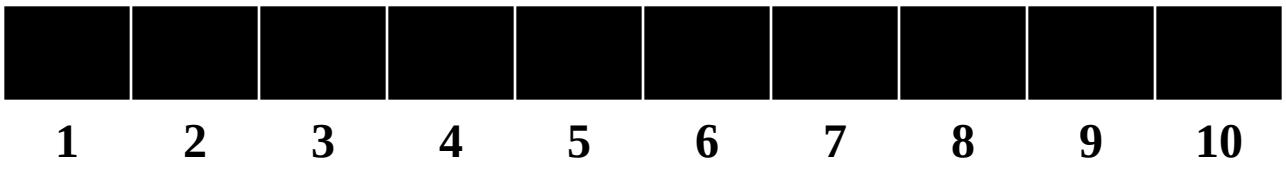


Node	Indegree
2	~~2~~ ~~1~~ 0

L: 1 3 5 6 4

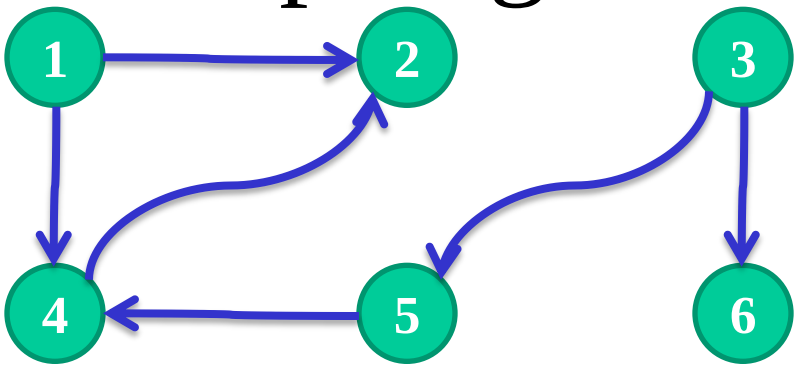
F=0

R=0



Enqueue 2

Topological sort



Node Indegree

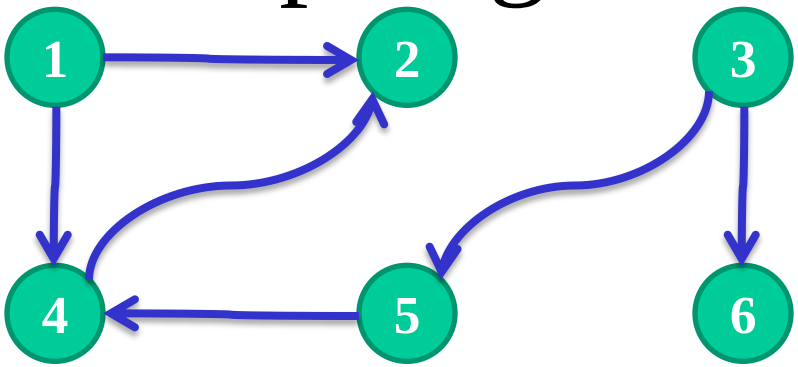
L: 1 3 5 6 4

F=1

R=1

Dequeue 2

Topological sort

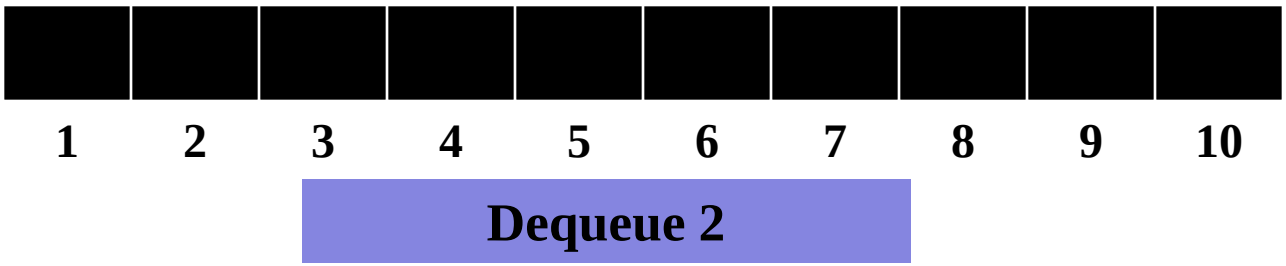


Node Indegree

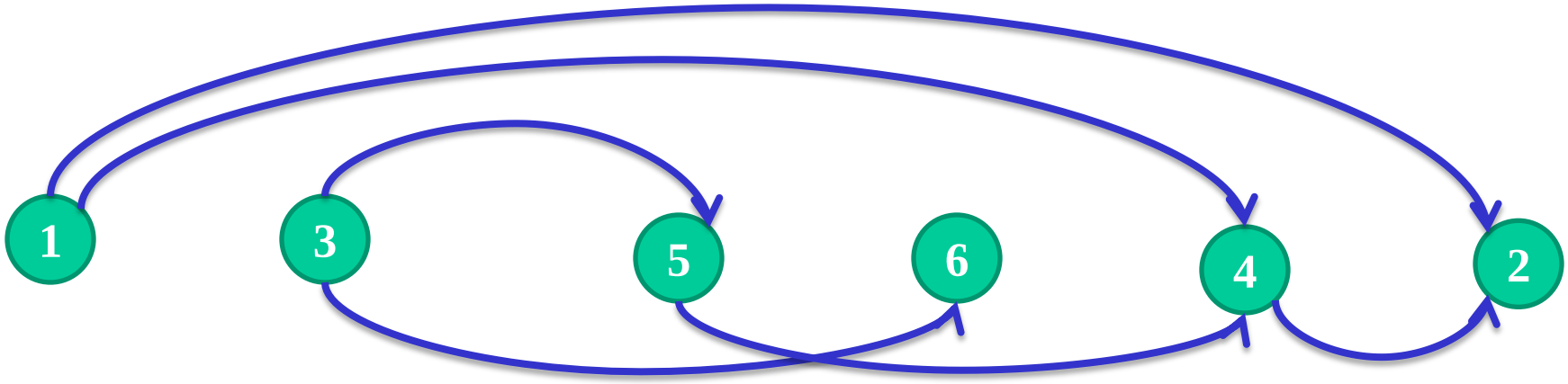
L: 1 3 5 6 4 2

F=0

R=0



Topological sort



L: 1 3 5 6 4 2

F=0

R=0

1 2 3 4 5 6 7 8 9 10