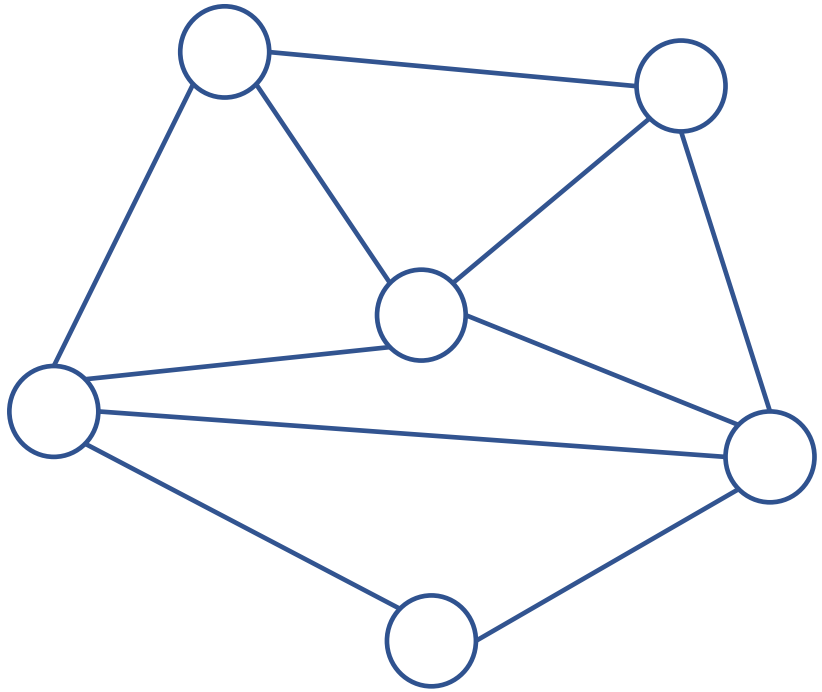
CS2x1:Data Structures and Algorithms

Koteswararao Kondepu

k.kondepu@iitdh.ac.in

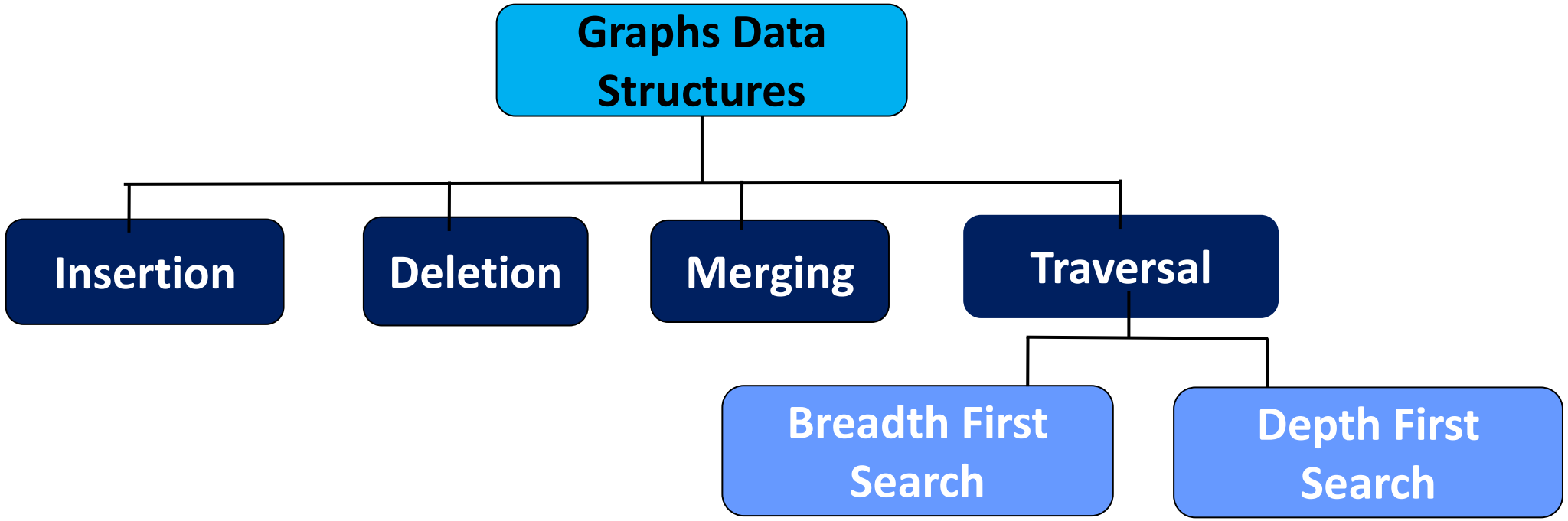
Recap: Graphs

- Definition: Graphs
- Terminology: Graphs
  - *Undirected Graph*
  - *Directed Graph*
  - *Weighted Graph*
  - *Simple Graph*
  - *Complete Graph*
  - *Acyclic Graph*
  - *Connected Graph*
  - *Bipartite Graph*



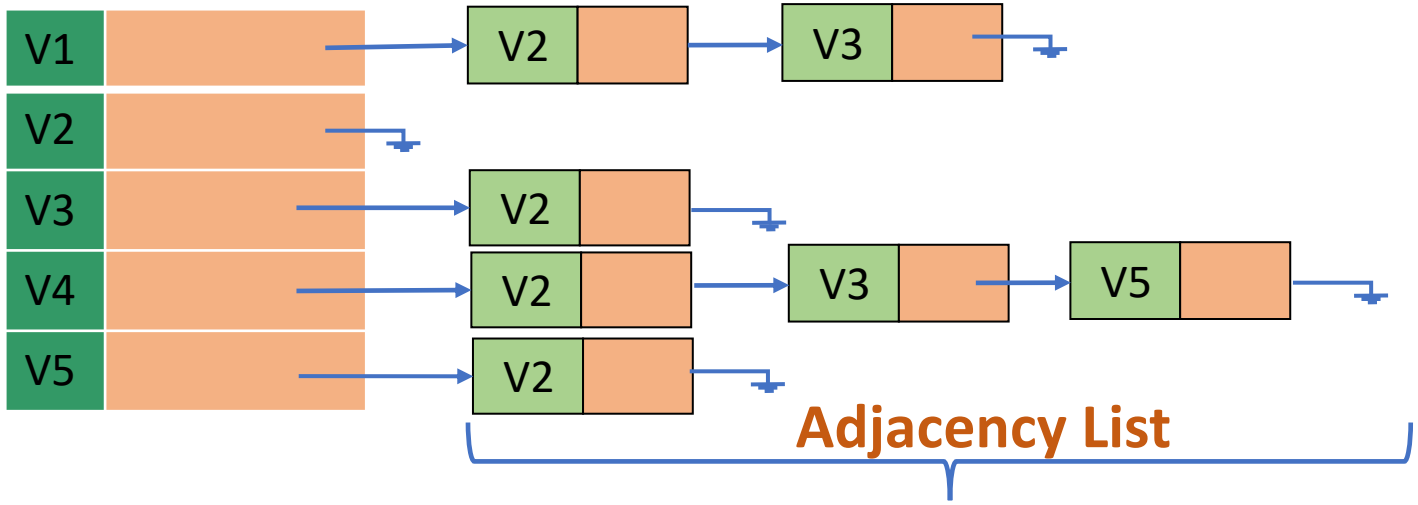
- Representation of Graphs
  - *Set representation (Adjacency Set)*
  - *Linked representation (Adjacency List)*
  - *Sequential (Matrix) representation (Adjacency Matrix)*

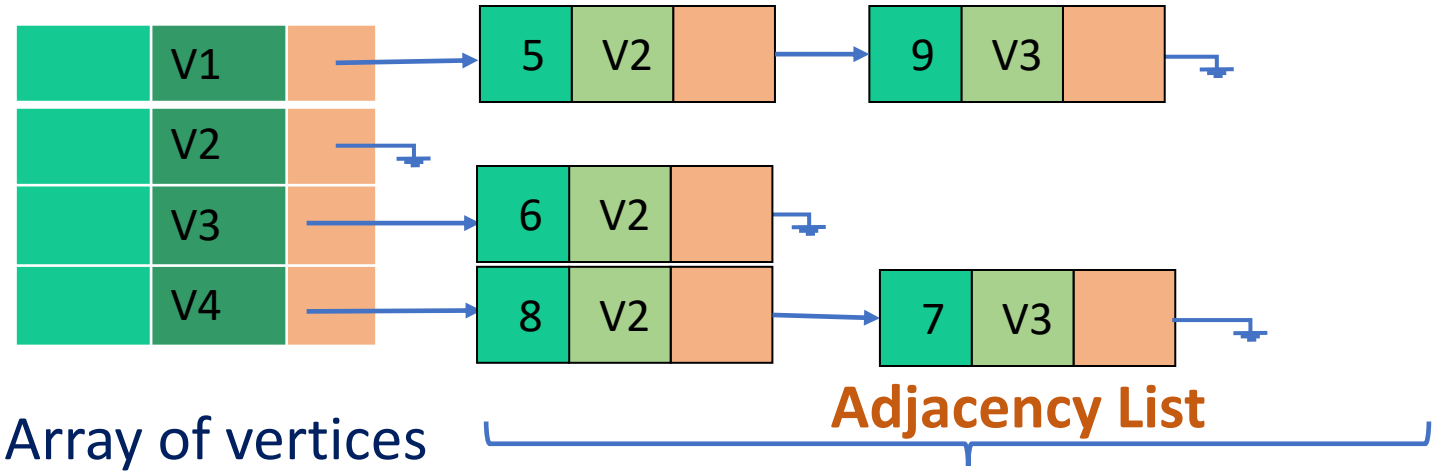
Graphs: Operations

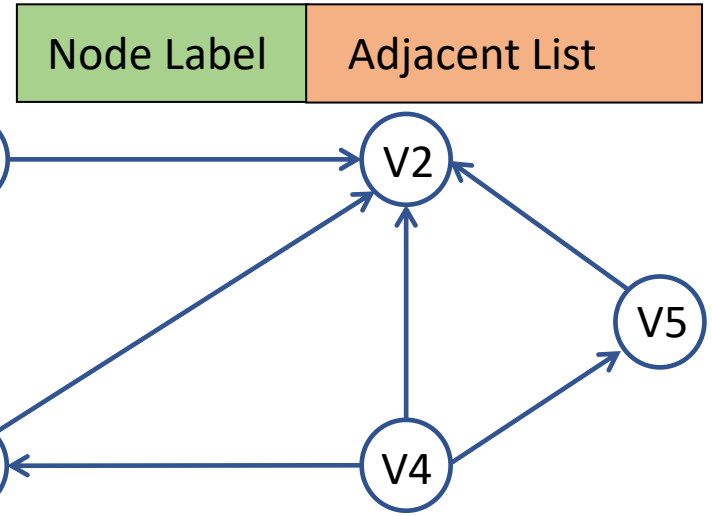


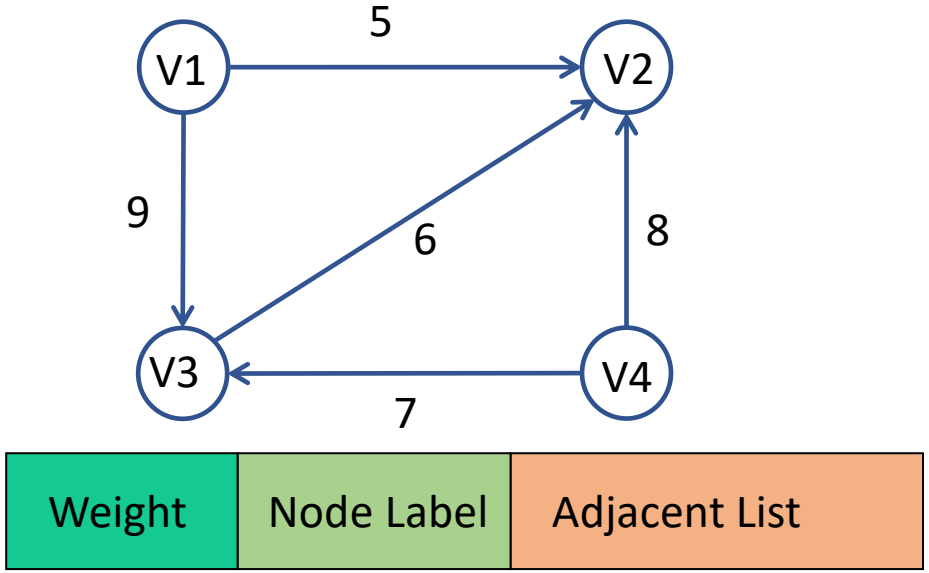
Graphs: Insertion (1)

- Linked representation is another space-saving way of graph representation

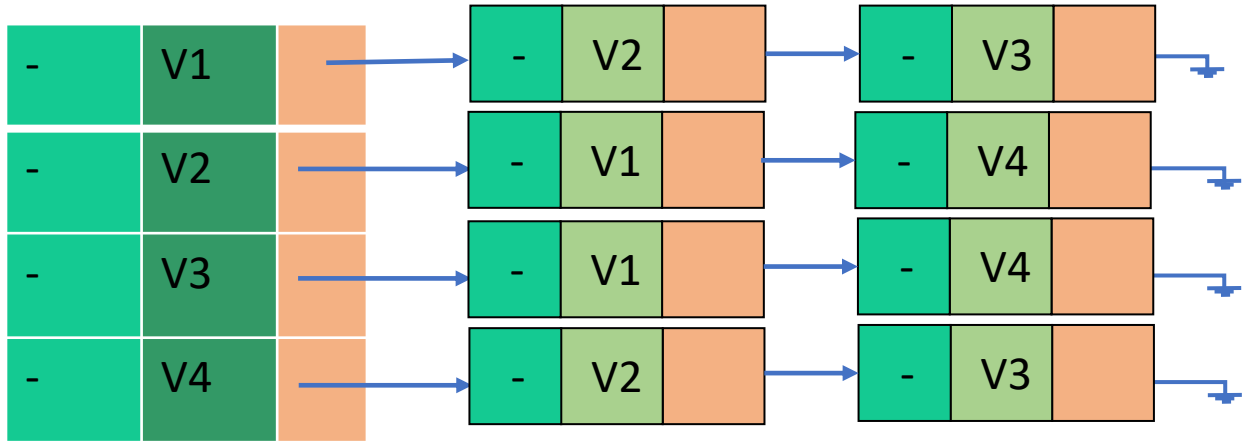


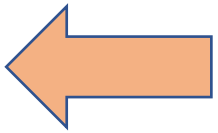


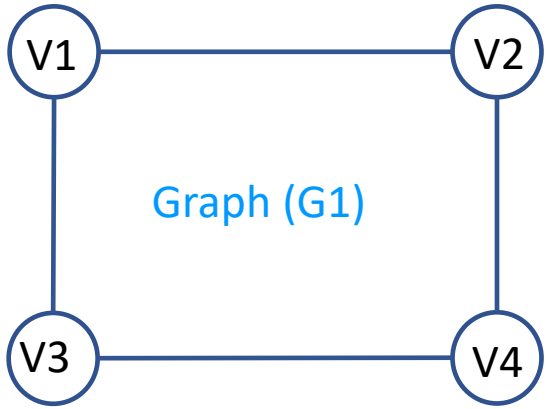


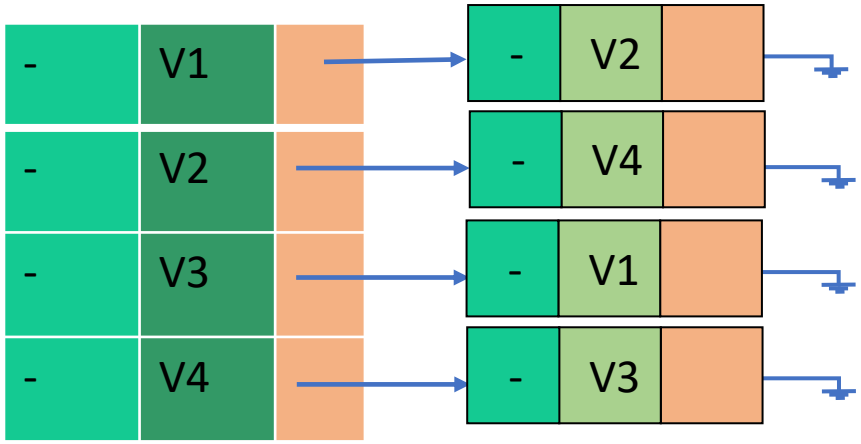


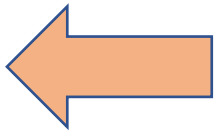
Graphs: Insertion (2)

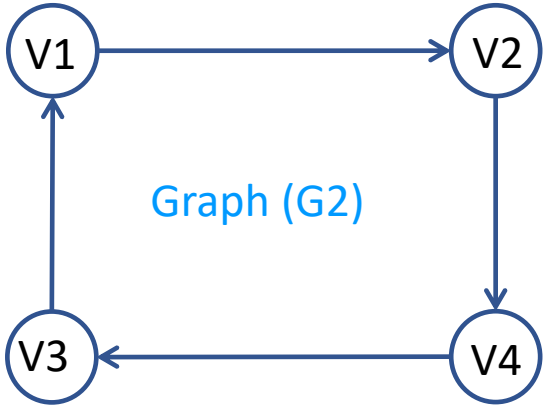












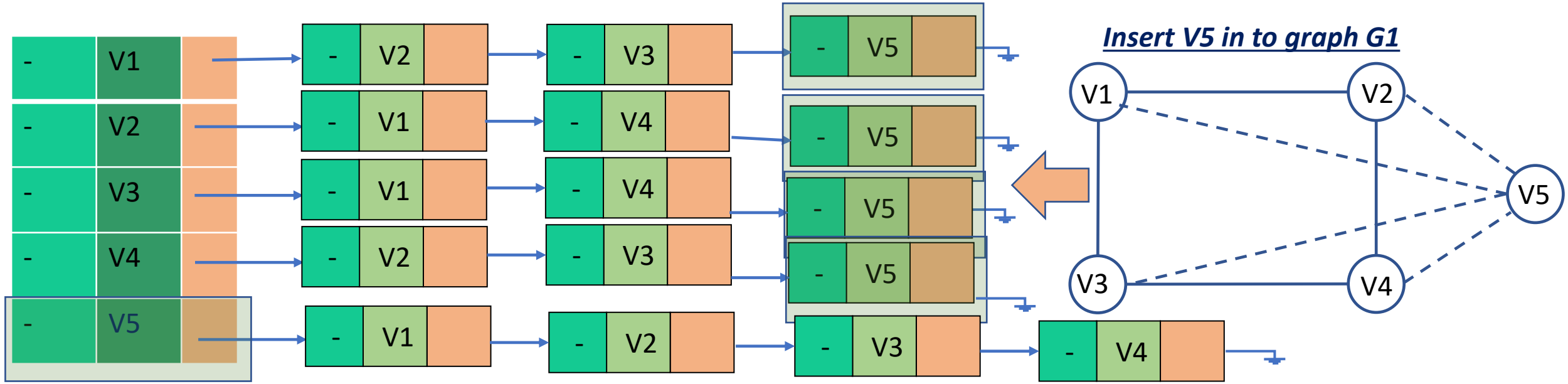
Graphs: Insertion (3)

***Policy:* (i) Insertion into undirected graph:**

if V_x is to be inserted and V_i is its adjacent vertex

→ V_i has to be incorporated in the adjacency list of V_x

→ V_x has to be incorporated in the adjacency list of V_i

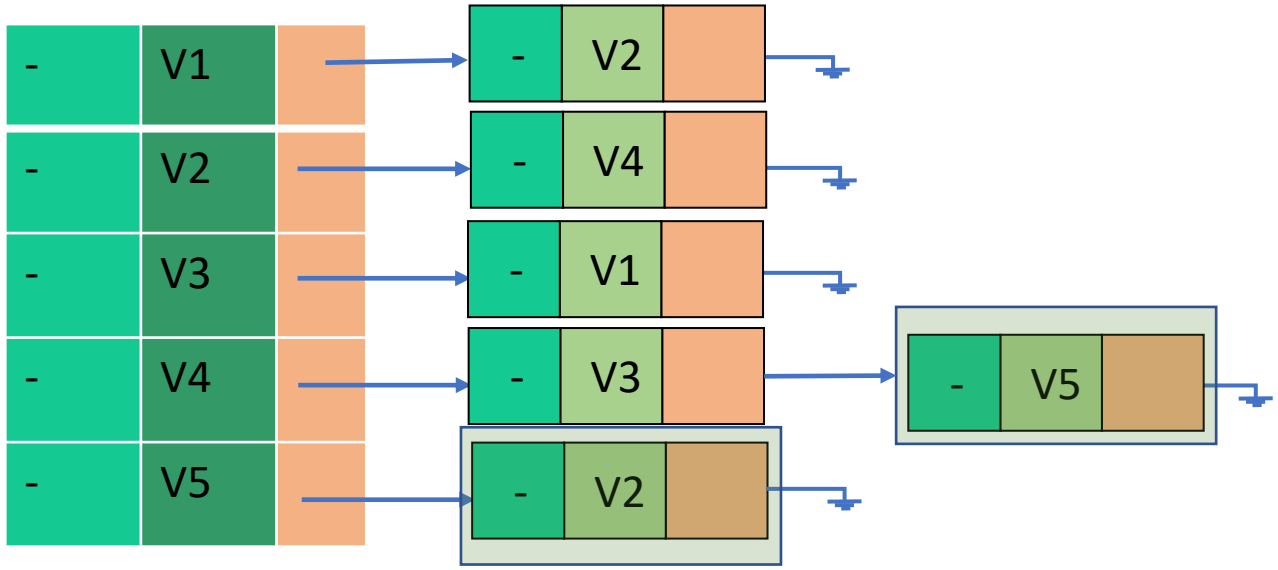


Graphs: Insertion (4)

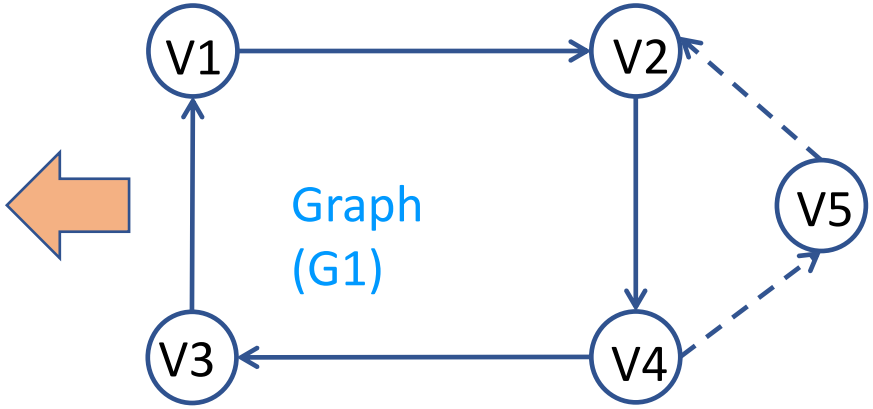
***Policy:* (i) Insertion into directed graph:**

if there is a path from V_x to V_i → V_i has to be incorporated in the adjacency list of V_x

if there is a edge from V_i to V_x → V_x has to be incorporated in the adjacency list of V_i



Insert V5 in to graph G2



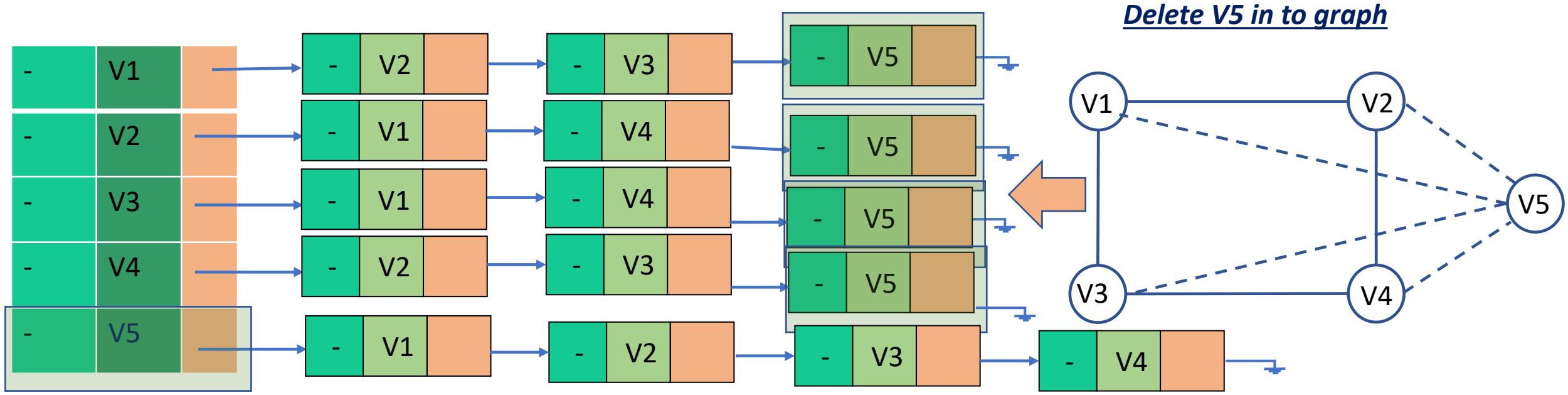
Graphs: Deletion (1)

Policy: ***(i)*** **Delete the vertex from the undirected graph:**

if V_x is to be deleted from the graph →

(i) Look for the adjacency list of V_x

(ii) From all the vertices which are presented in the adjacency list of V_x the node labelled V_x has to be deleted from the adjacent vertices V_i



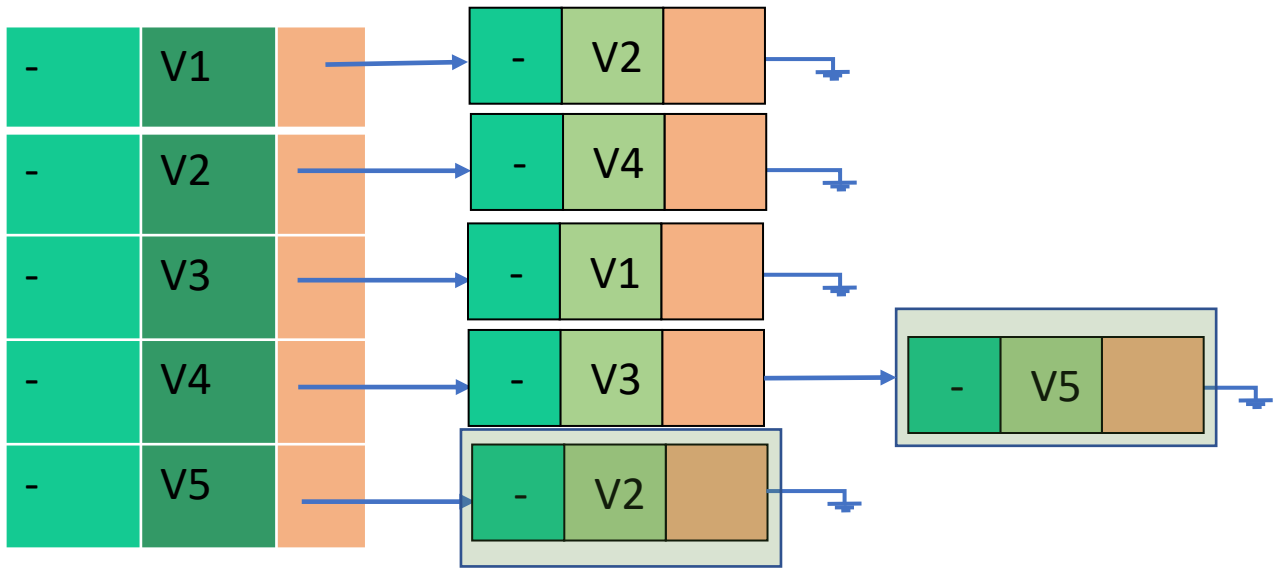
Graphs: Deletion (2)

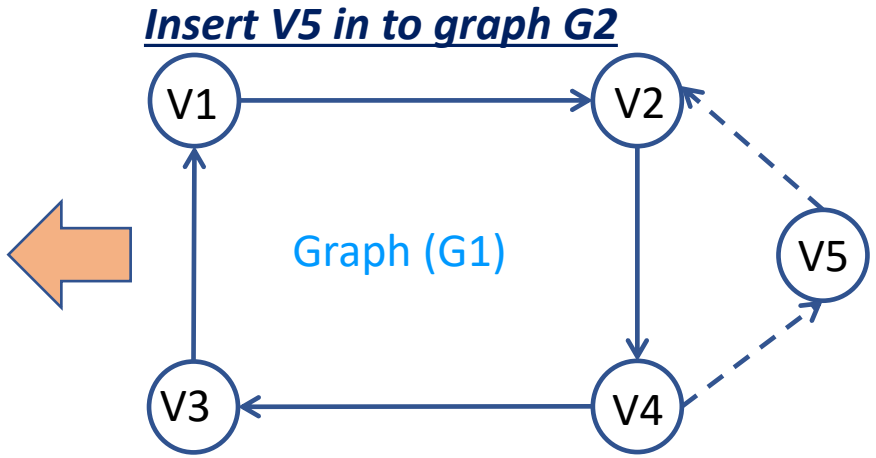
Policy: ***(i)*** **Delete the vertex from the directed graph:**

if V_x is to be deleted from the graph →

(i) delete adjacency list of V_x - this removes all the outgoing edges from V_x

(ii) Search adjacency list for all vertices, if there is any vertex V_i which has and edge from V_i to V_x





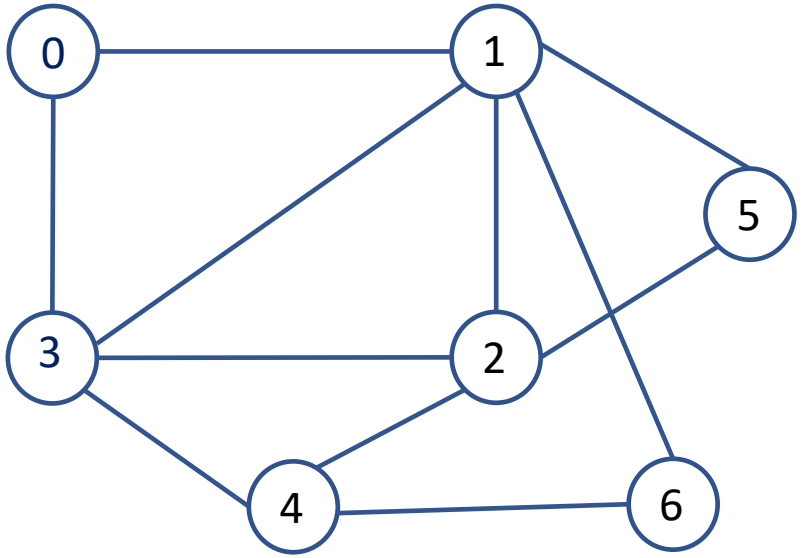
Graphs: Traversal

❖ ***Traversal Graph*** → **Visiting all the vertices in the graph exactly once**

❖ ***Traversal Graph Methods*** → **(i) Breadth First Search (BFS) ; (ii) Depth First Search (DFS)**

❖ Starting from the given node (vertex), we visit all the nodes which are reachable from the starting node

❖ Graph traversal algorithms → Graph search algorithms



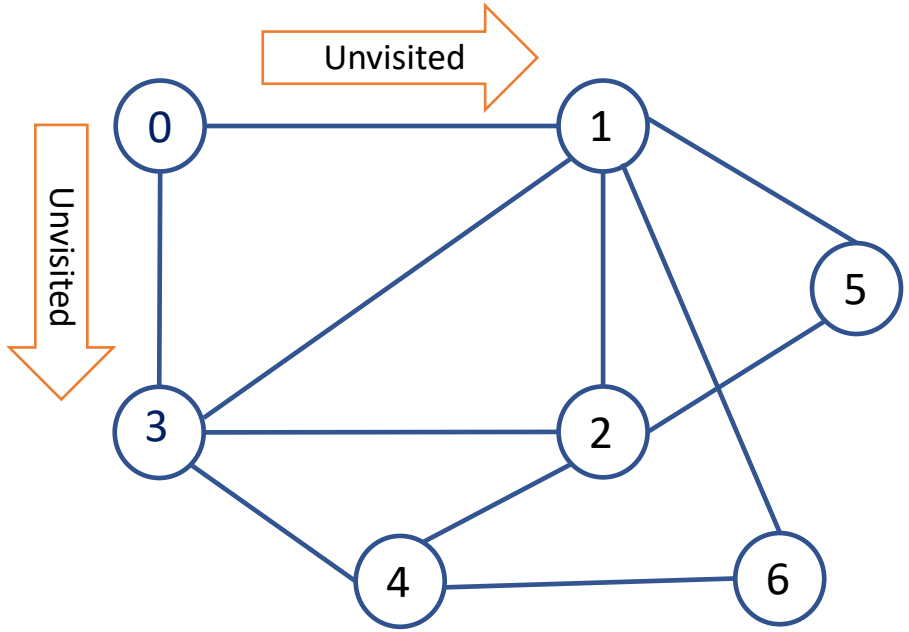
Graph Traversal: BFS (1)

- BFS is useful for finding shortest path distance in the graph.
- In BFS nodes can be visited level-by-level, so it is called level-order traversal
- The implementation of BFS → Queue data structure.

- *Procedure*:
  - ✓ Initially, BFS starts at a given vertex, which is at level#0. In the first stage it visits all vertices at level#1 (i.e., adjacent vertices of the starting vertex of the graph).
  - ✓ After that, it visits all vertices at the level#2. These new vertices are the ones which are adjacent to level#1 vertices.
  - ✓ Repeat this process until all the levels of the graph is completed.

Graph Traversal: BFS (2)

- *Policy:*
  - ✓ Enqueue the starting vertex;
  - ✓ Dequeue the starting vertex and Enqueue the adjacent unvisited verities
  - ✓ Mark it as visited Dequeued vertex
  - ✓ If no adjacent vertex is found, remove the first vertex from the Queue.
  - ✓ Repeat the above steps until the Queue is empty.

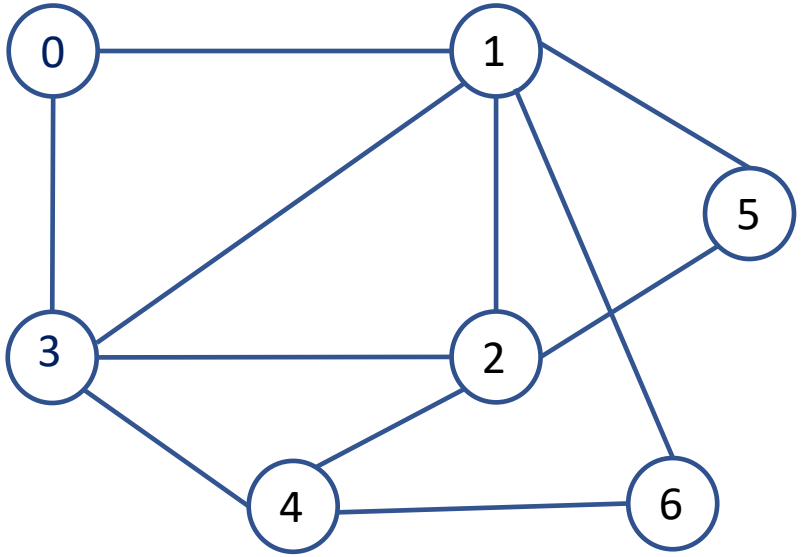


0

BFS:

Graph Traversal: BFS (3)

- *Policy:*
  - ✓ Dequeue the vertex and Enqueue the adjacent unvisited verities
  - ✓ Mark it as visited Dequeued vertex
  - ✓ If no adjacent vertex is found, remove the first vertex from the Queue.
  - ✓ Repeat the above steps until the Queue is empty.

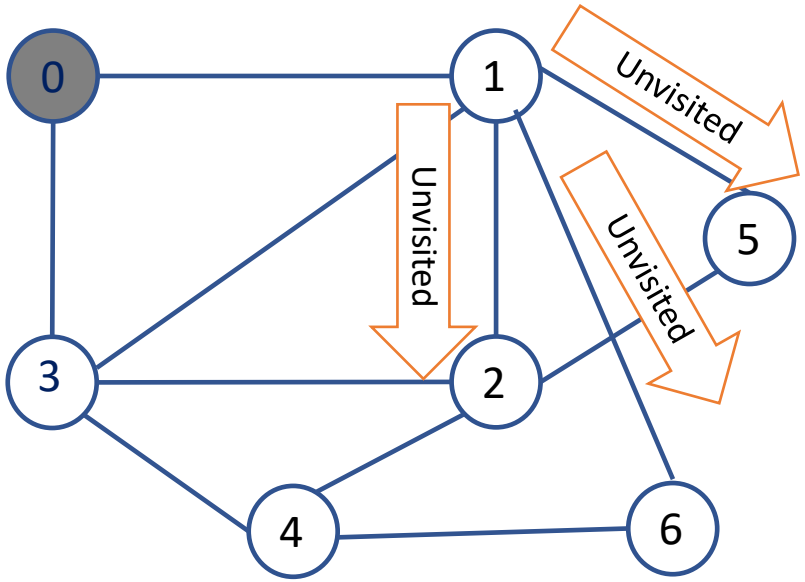


1 3

BFS: 0 1

Graph Traversal: BFS (4)

- *Policy:*
  - ✓ Dequeue the vertex and Enqueue the adjacent unvisited verities
  - ✓ Mark it as visited Dequeued vertex
  - ✓ If no adjacent vertex is found, remove the first vertex from the Queue.
  - ✓ Repeat the above steps until the Queue is empty.

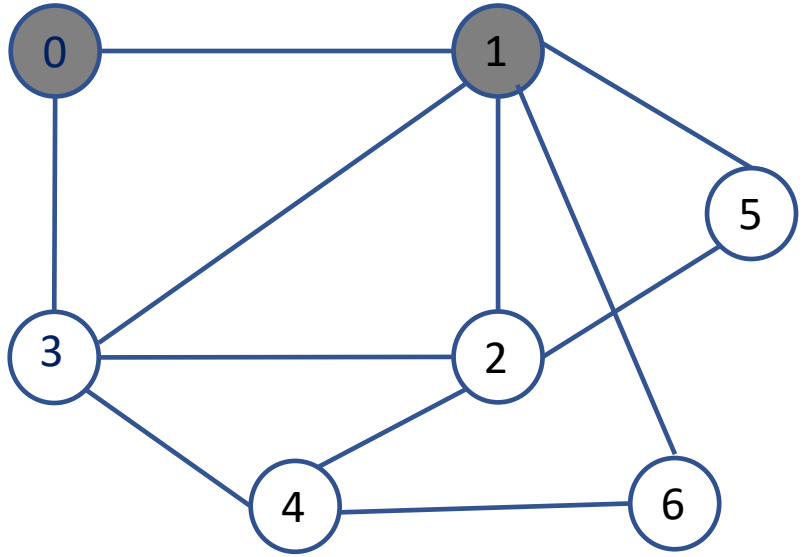


3 2 5 6

BFS: 0 1

Graph Traversal: BFS (5)

- *Policy:*
  - ✓ Dequeue the vertex and Enqueue the adjacent unvisited verities
  - ✓ Mark it as visited Dequeued vertex
  - ✓ If no adjacent vertex is found, remove the first vertex from the Queue.
  - ✓ Repeat the above steps until the Queue is empty.

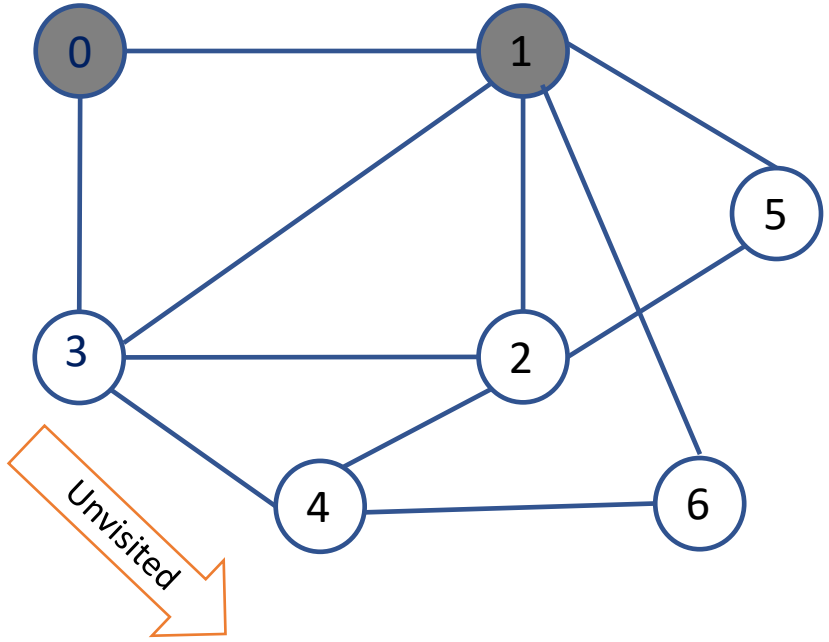


3 2 5 6

BFS: 0 1 3

Graph Traversal: BFS (6)

- *Policy:*
  - ✓ Dequeue the vertex and Enqueue the adjacent unvisited verities
  - ✓ Mark it as visited Dequeued vertex
  - ✓ If no adjacent vertex is found, remove the first vertex from the Queue.
  - ✓ Repeat the above steps until the Queue is empty.

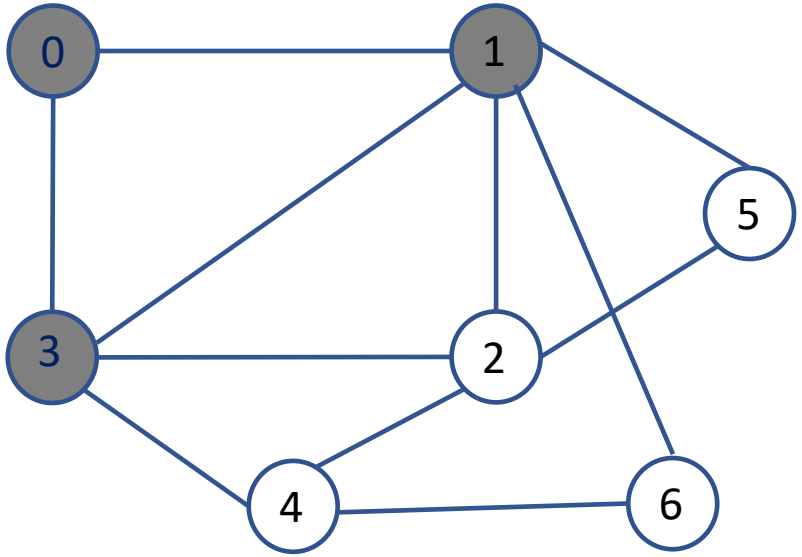


2 5 6 4

BFS: 0 1 3

Graph Traversal: BFS (7)

- *Policy:*
  - ✓ Dequeue the vertex and Enqueue the adjacent unvisited verities
  - ✓ Mark it as visited Dequeued vertex
  - ✓ If no adjacent vertex is found, remove the first vertex from the Queue.
  - ✓ Repeat the above steps until the Queue is empty.

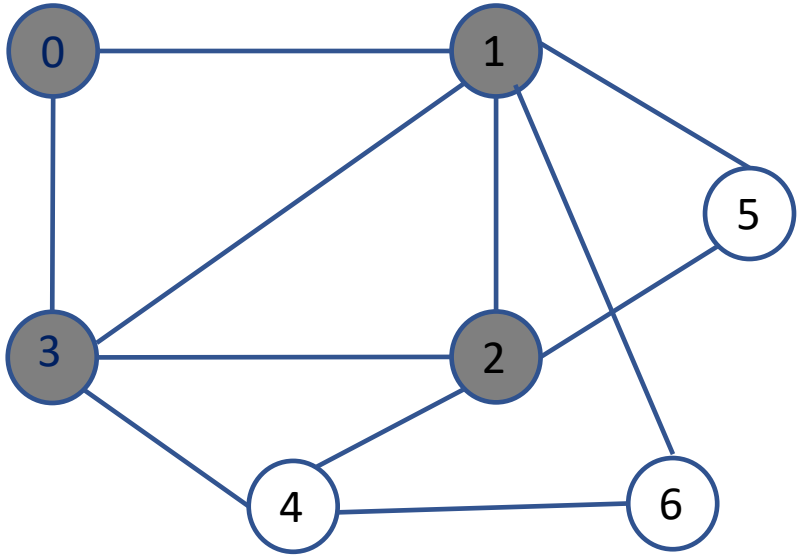


2 5 6 4

BFS: 0 1 3

Graph Traversal: BFS (8)

- *Policy:*
  - ✓ Dequeue the vertex and Enqueue the adjacent unvisited verities
  - ✓ Mark it as visited Dequeued vertex
  - ✓ If no adjacent vertex is found, remove the first vertex from the Queue.
  - ✓ Repeat the above steps until the Queue is empty.

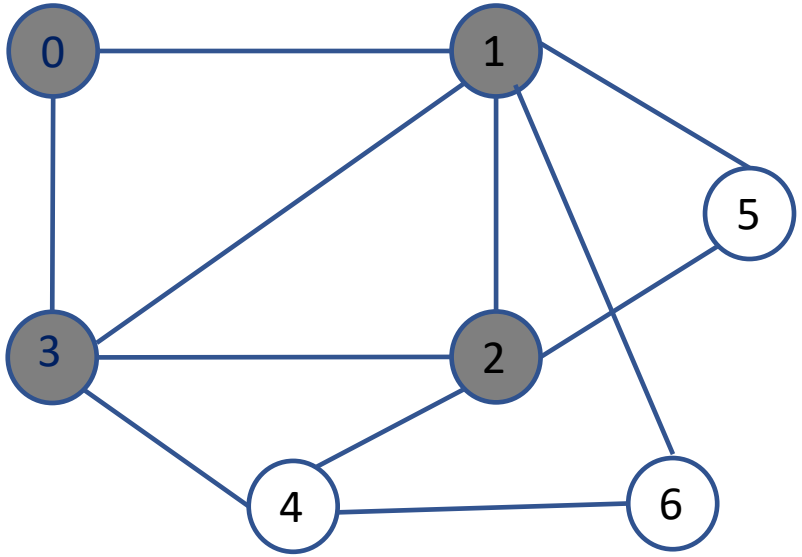


5 6 4

BFS: 0 1 3 2

Graph Traversal: BFS (9)

- *Policy:*
  - ✓ Dequeue the vertex and Enqueue the adjacent unvisited verities
  - ✓ Mark it as visited Dequeued vertex
  - ✓ If no adjacent vertex is found, remove the first vertex from the Queue.
  - ✓ Repeat the above steps until the Queue is empty.

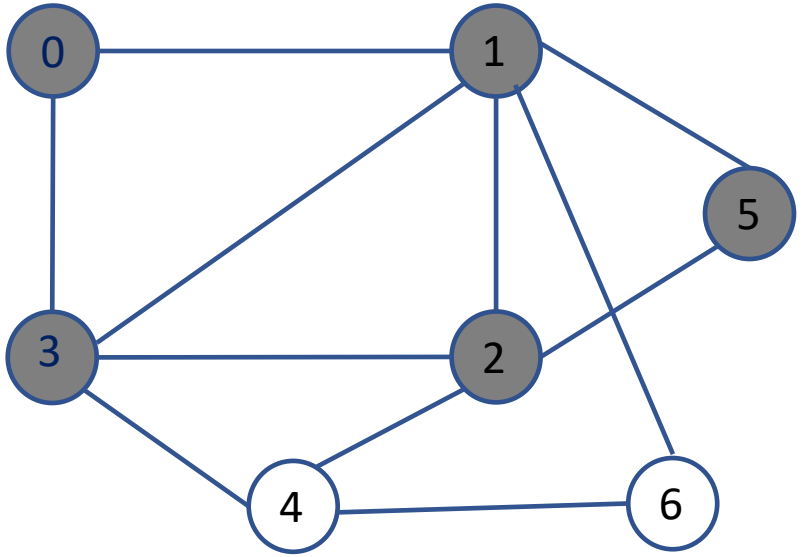


5 6 4

BFS: 0 1 3 2

Graph Traversal: BFS (10)

- *Policy:*
  - ✓ Dequeue the vertex and Enqueue the adjacent unvisited verities
  - ✓ Mark it as visited Dequeued vertex
  - ✓ If no adjacent vertex is found, remove the first vertex from the Queue.
  - ✓ Repeat the above steps until the Queue is empty.

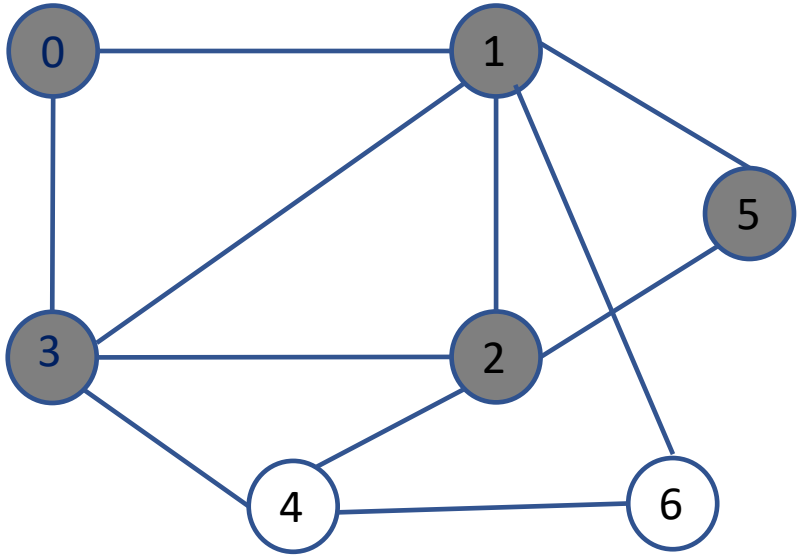


6 4

BFS: 0 1 3 2 5

Graph Traversal: BFS (11)

- *Policy:*
  - ✓ Dequeue the vertex and Enqueue the adjacent unvisited verities
  - ✓ Mark it as visited Dequeued vertex
  - ✓ If no adjacent vertex is found, remove the first vertex from the Queue.
  - ✓ Repeat the above steps until the Queue is empty.

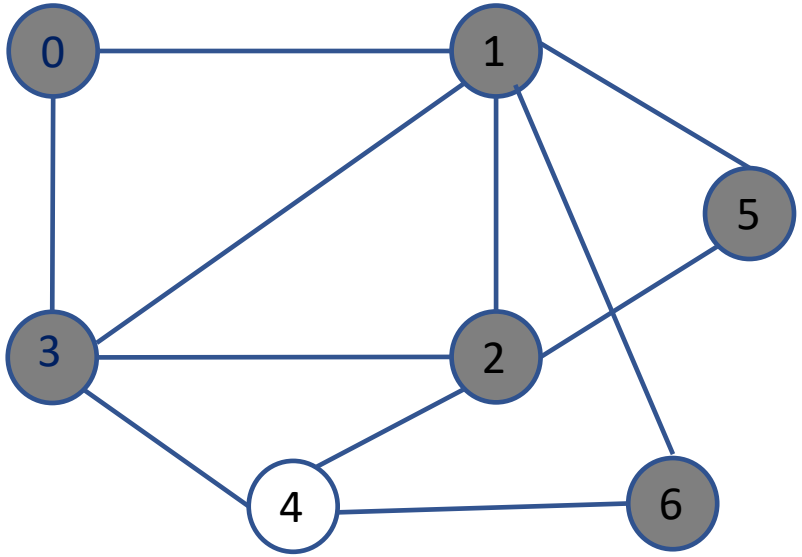


6 4

BFS: 0 1 3 2 5

Graph Traversal: BFS (12)

- *Policy:*
  - ✓ Dequeue the vertex and Enqueue the adjacent unvisited verities
  - ✓ Mark it as visited Dequeued vertex
  - ✓ If no adjacent vertex is found, remove the first vertex from the Queue.
  - ✓ Repeat the above steps until the Queue is empty.

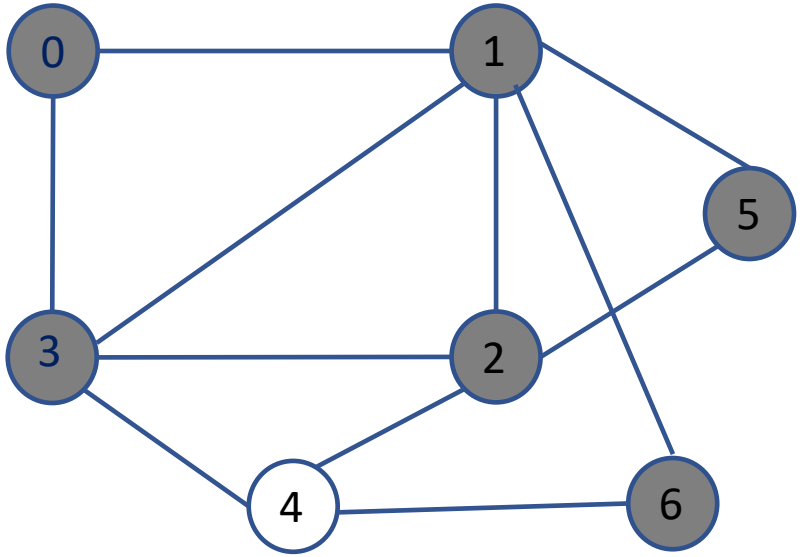


4

BFS: **0 1 3 2 5 6**

Graph Traversal: BFS (13)

- *Policy:*
  - ✓ Dequeue the vertex and Enqueue the adjacent unvisited verities
  - ✓ Mark it as visited Dequeued vertex
  - ✓ If no adjacent vertex is found, remove the first vertex from the Queue.
  - ✓ Repeat the above steps until the Queue is empty.

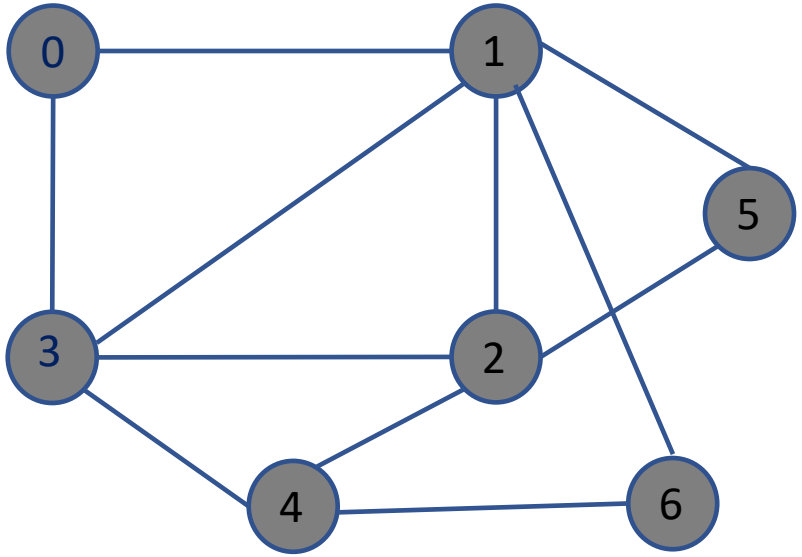


4

BFS: **0 1 3 2 5 6**

Graph Traversal: BFS (14)

- *Policy:*
  - ✓ Dequeue the vertex and Enqueue the adjacent unvisited verities
  - ✓ Mark it as visited Dequeued vertex
  - ✓ If no adjacent vertex is found, remove the first vertex from the Queue.
  - ✓ Repeat the above steps until the Queue is empty.

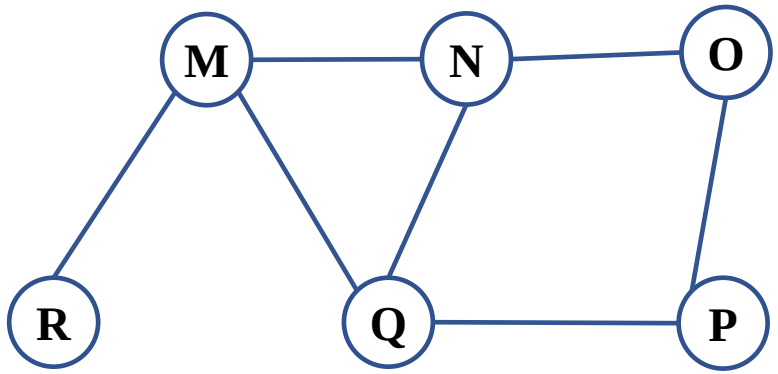


BFS: **0 1 3 2 5 6 4**

Exercise: Graphs Traversal (1)

The Breadth First Search algorithm has been implemented using the queue data structure. One possible order of visiting the nodes of the following graph is

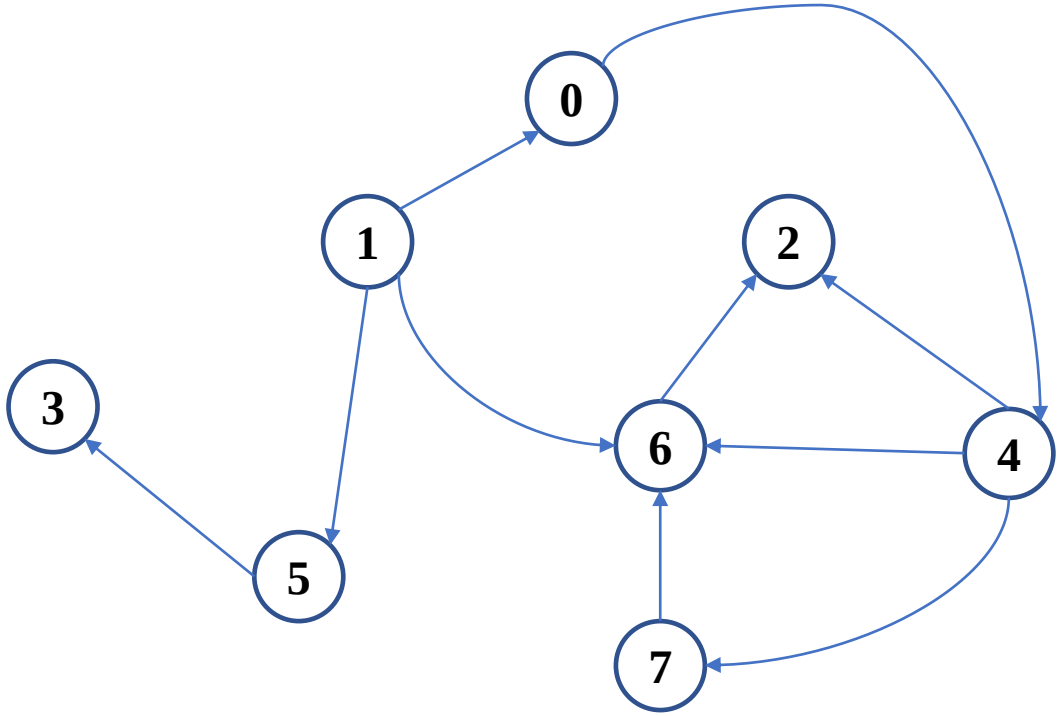
A. MNOPQR
B. NQMPOR
C. QMNPRO
D. QMNPOR



Exercise: Graphs Traversal (2)

The Breadth First Search algorithm has been implemented using the queue data structure. One possible order of visiting the nodes of the following graph is: Consider that starting vertex → 1

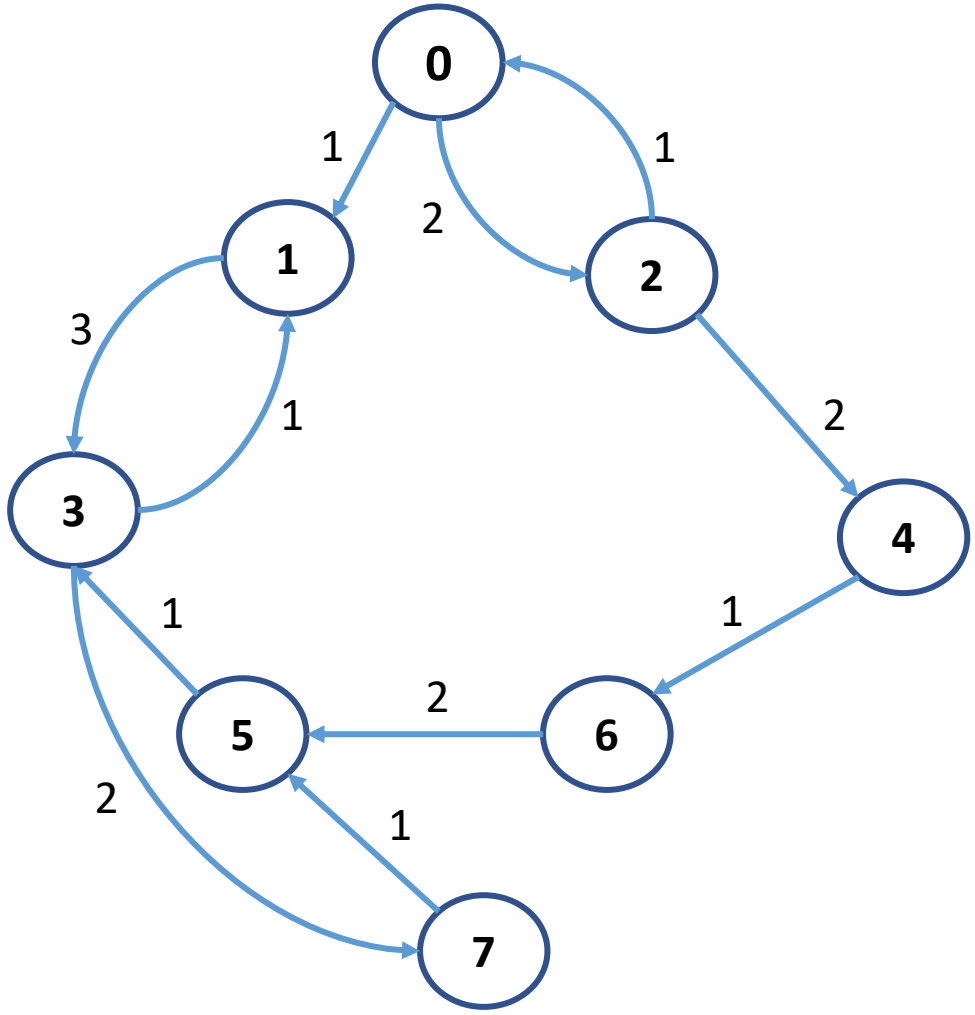
A. 10764325
B. 10564357
C. 10564327
D. None



Exercise: Graphs Traversal (3)

1. Which of the following ordering(s) is/are possible using BFS with starting **vertex 0**
   a. 0, 1, 2, 3, 4, 7, 6, 5
   b. 0, 2, 1, 4, 3, 6, 7, 5
   c. Both a and b
   d. None

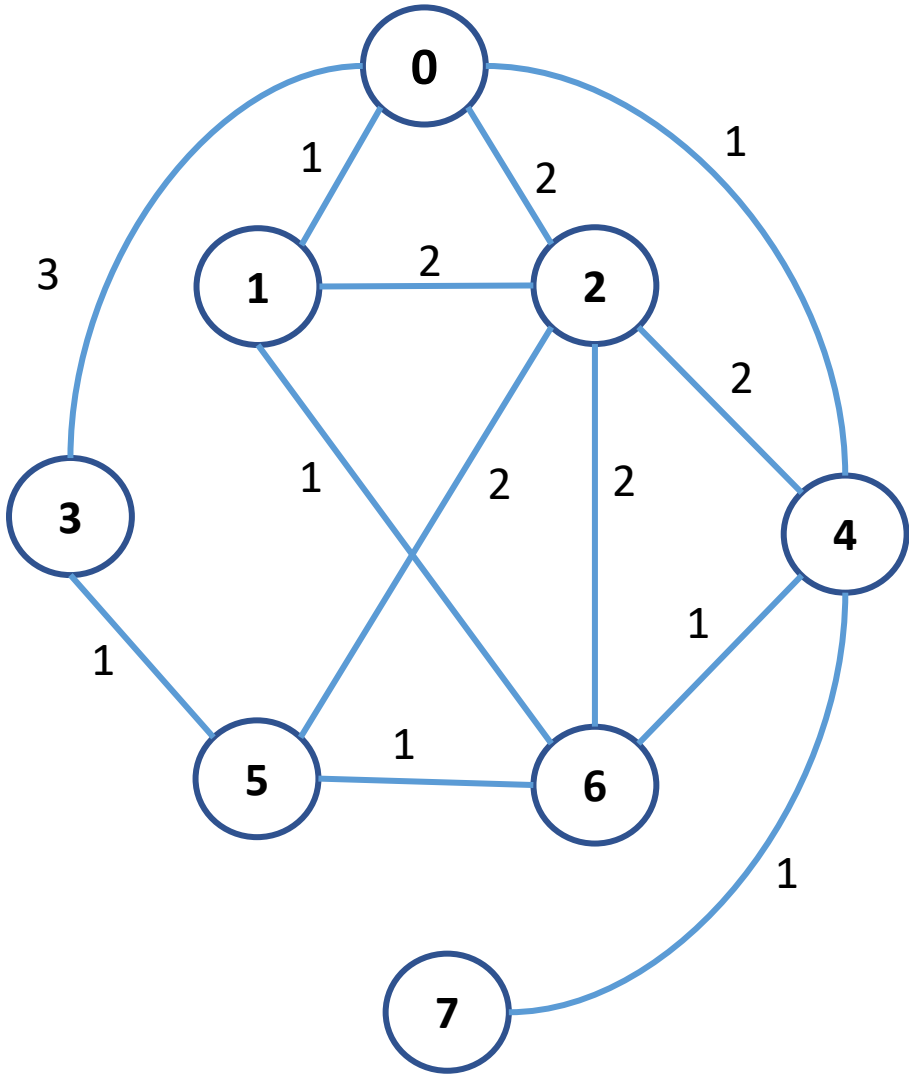
2. What is the possible shortest path for the given graph using BFS with starting **vertex 0**



Exercise: Graphs Traversal (4)

1. Which of the following ordering are possible using BFS with starting **vertex 0**
   a. 0, 3, 1, 2, 4, 5, 6, 7
   b. 0, 1, 2, 3, 4, 5, 6, 7
   c. 0, 4, 3, 2, 1, 6, 7, 5
   d. All of the above

2. What is the possible shortest path for the given graph using BFS with starting **vertex 0**



thank you!

email:
k.kondepu@iitdh.ac.in