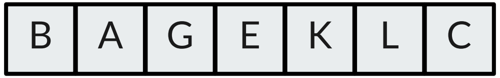
CIS 263 Introduction to Data Structures and Algorithms

Graph Algorithms

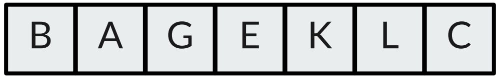
Linear Data Structures

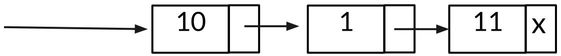
- Array



Linear Data Structures

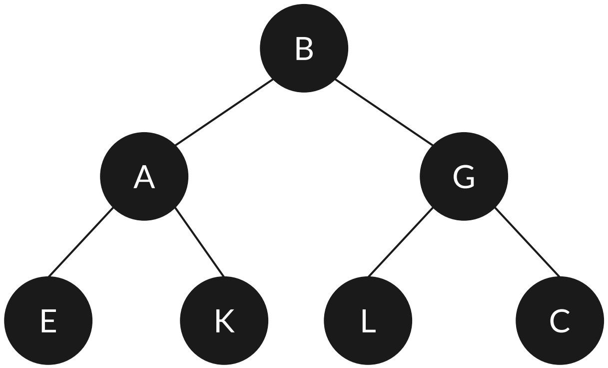
- Array
- Linked List





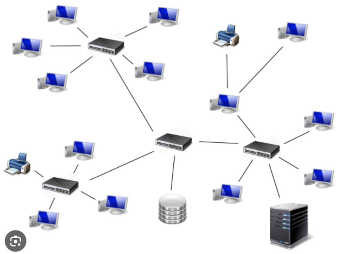
Hierarchical relationship

- Tree
  - Parent child relationship



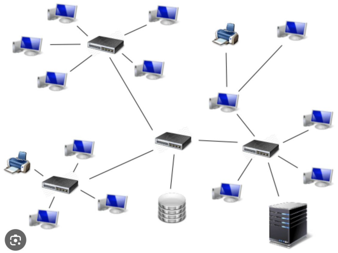
Graphs

- **Pairwise** relationship
- Examples
  - Computer networks



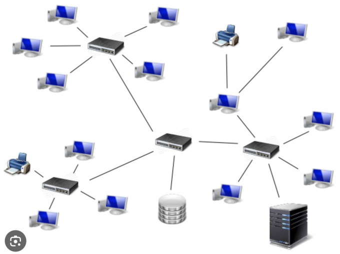
Graphs

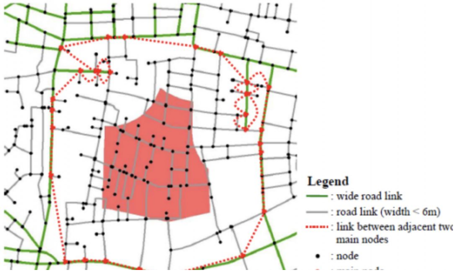
- **Pairwise** relationship
- Examples
  - Computer networks
  - Social networks



Graphs

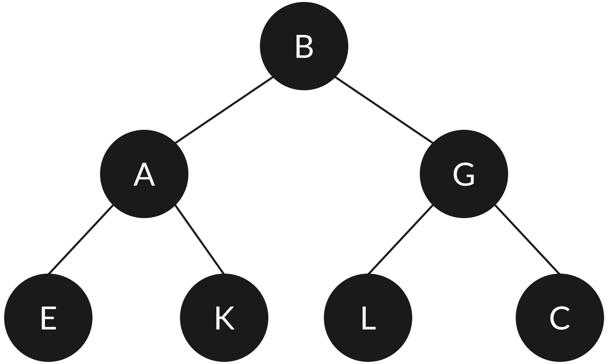
- Pairwise relationship
- Examples
  - Computer networks
  - Social networks
  - Road networks
  -





A Tree is a special instance of Graph

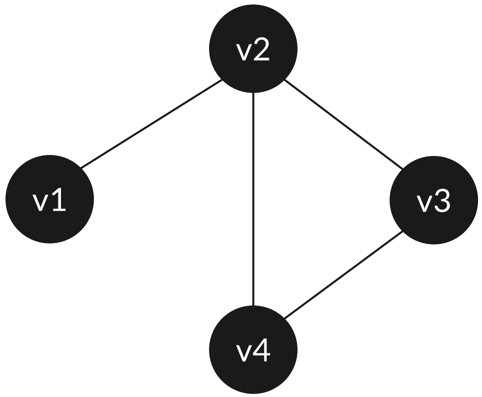
- Tree
    - Parent child relationship



Graph (Formal Definition)

- **Pairwise** relationship
- A Graph, ***G***, is an ordered pair of vertices, ***V***, and edges, ***E***.
  - **G = (V, E)**

    V = {v1, v2, v3, v4}
    E = {{v1,v2}, {v2,v3}, {v3,v4}, {v2,v4}}

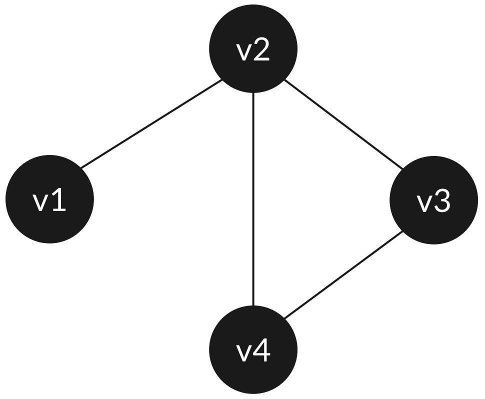


Graph (Formal Definition)

- **Pairwise** relationship
- A Graph, ***G***, is an ordered pair of vertices, ***V***, and edges, ***E***.
    - **G = (V, E)**

      V = {v1, v2, v3, v4}
      E = {{v1,v2}, {v2,v3}, {v3,v4}, {v2,v4}}

Undirected Graph

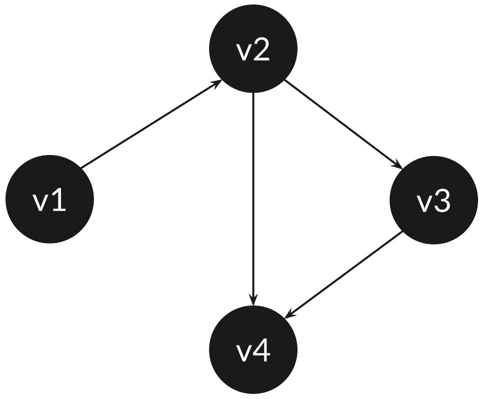


Directed Graph

- Pairwise relationship

- A Graph, ***G***, is an ordered pair of vertices, ***V***, and edges, ***E***.
    - **G = (V, E)**

      V = {v1, v2, v3, v4}
      E = {(v1,v2),({v2,v3), (v3,v4), (v2,v4)}



Graph (Data Structure) - Undirected

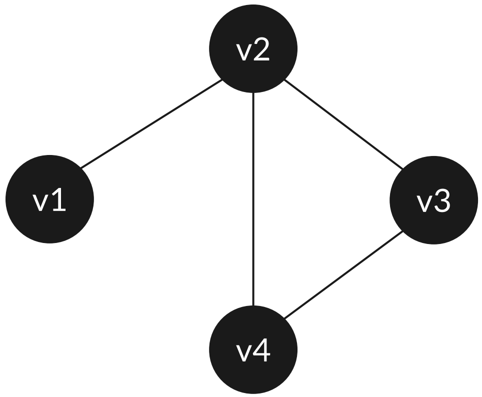
G = (V, E)

V = {v1, v2, v3, v4}

E = {{v1,v2}, {v2,v3}, {v3,v4}, {v2,v4}}

Using pointer

- non-static node definitions



Graph (Data Structure) - Undirected

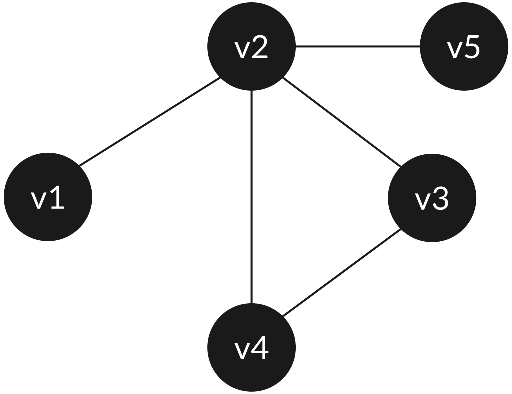
G = (V, E)

V = {v1, v2, v3, v4}

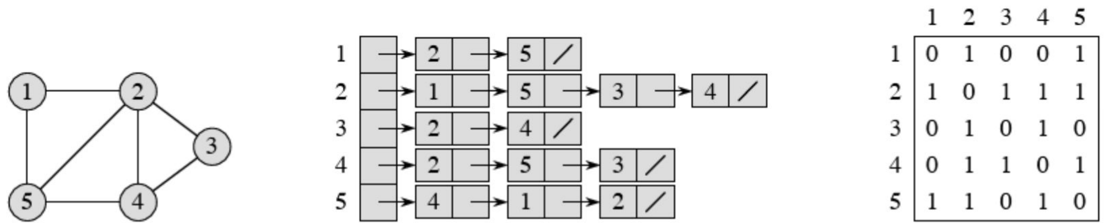
E = {{v1,v2}, {v2,v3}, {v3,v4}, {v2,v4}}

Using pointer

- non-static node definitions



Representation of an undirected graph

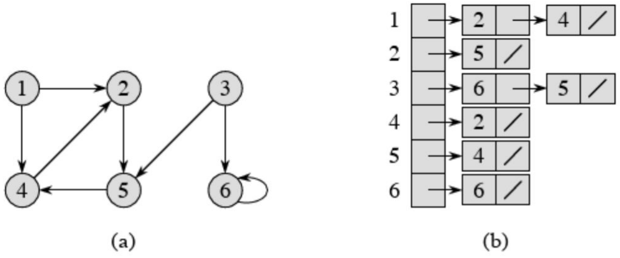


(a) (b) (c)

Figure 22.1 Two representations of an undirected graph. **(a)** An undirected graph G with 5 vertices and 7 edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Ref: Introduction to Algorithms, by Thomas Cormen

Representation of an undirected graph



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

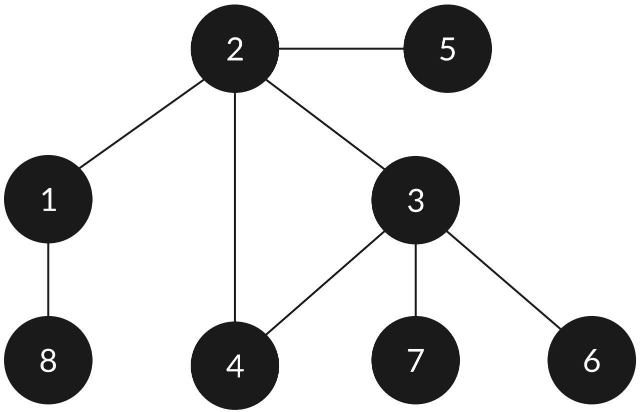
(c)

Figure 22.2 Two representations of a directed graph. **(a)** A directed graph G with 6 vertices and 8 edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Ref: Introduction to Algorithms, by Thomas Cormen

Graph Traversals

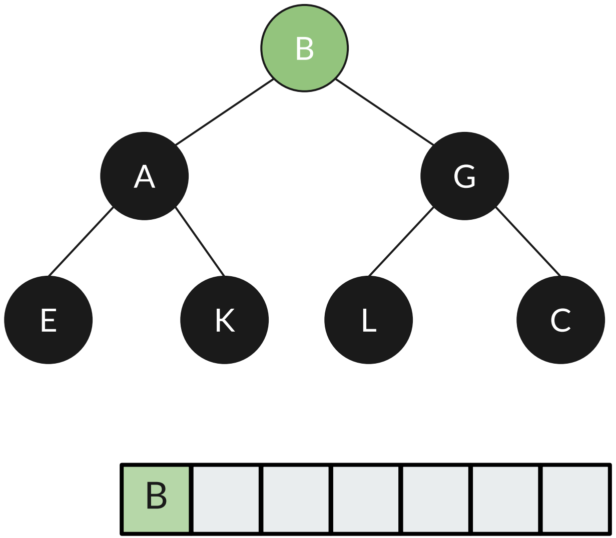
- Breadth First (level order)
- Depth First (pre order in case of BT)



Graph Traversals

- Breadth First (level order)

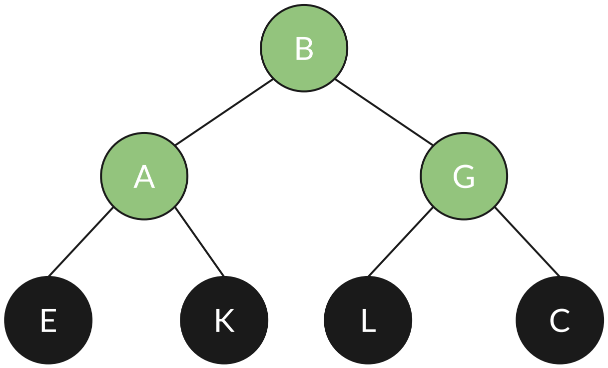
1. Start with any node, say (B)
2. Traverse one level at a time



Graph Traversals

- Breadth First (level order)

1. Start with any node, say (B)
2. Traverse one level at a time

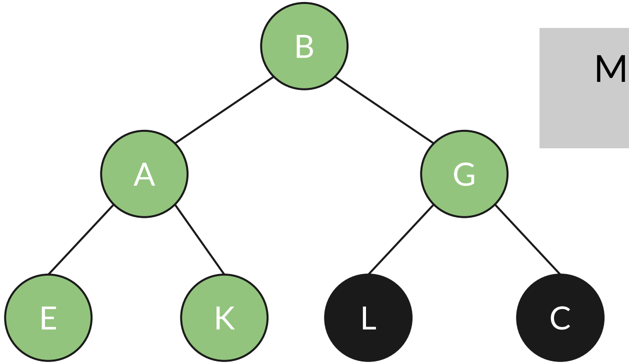


Both valid; we will track the first instance

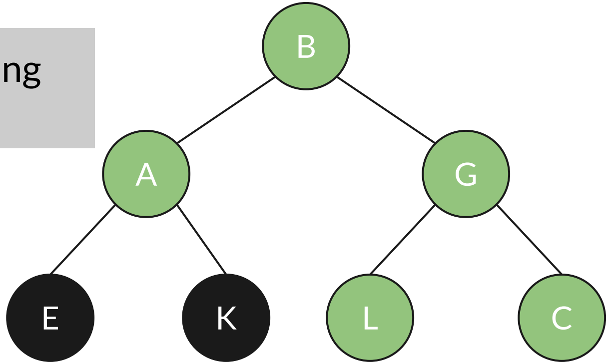
B	A	G				

B	G	A				

Graph Traversals



Multi-level ordering
(all valid)



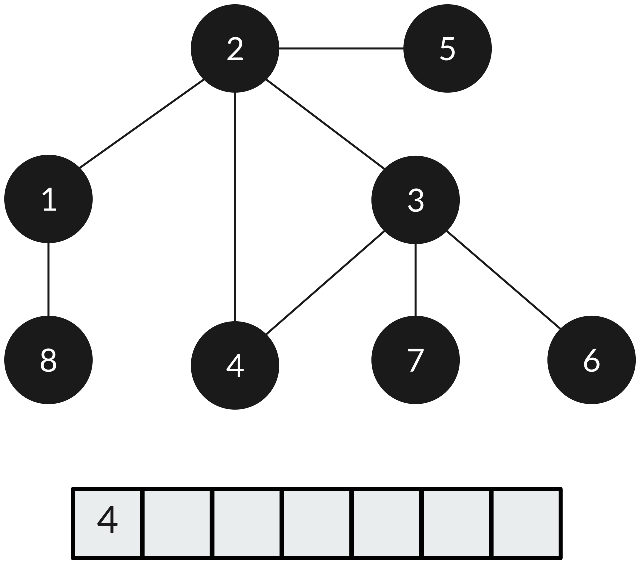
B	A	G	E	K		
B	A	G	K	E		

B	A	G	L	C		
B	A	G	C	L		

Graph Traversals

- Breadth First (level order)

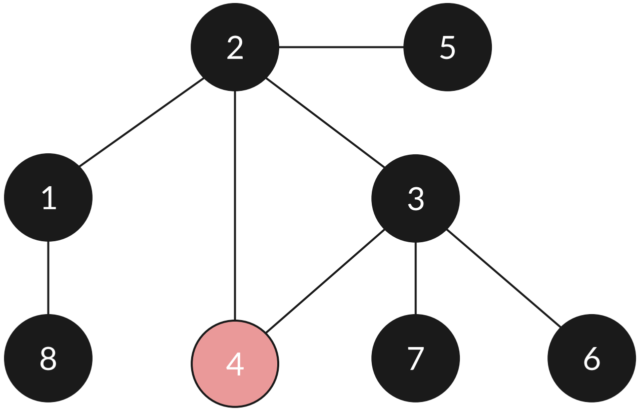
1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue**



Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue, Q**
    a. **Append data/key in Q**



Q

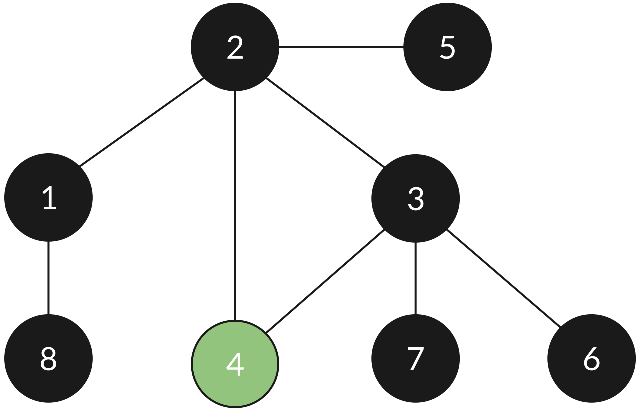
4						

Results

Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue, Q**
    a. Append data/key in Q
    b. **Process data/key**



Q							
	4						

Results								
	4							

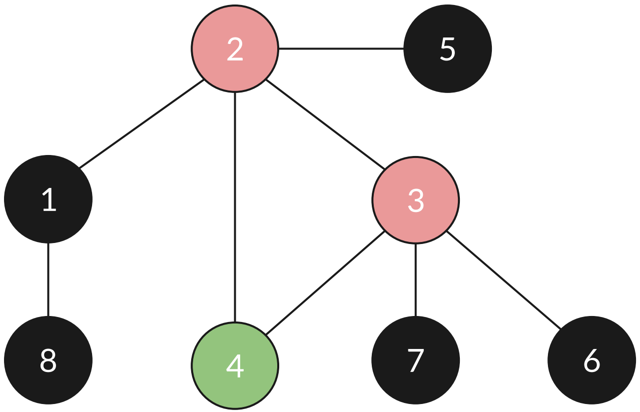
Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue, Q**
    a. **Update Queue (a) with neighbors (non-visited)**
    b. Process data/key

Q	3	2					

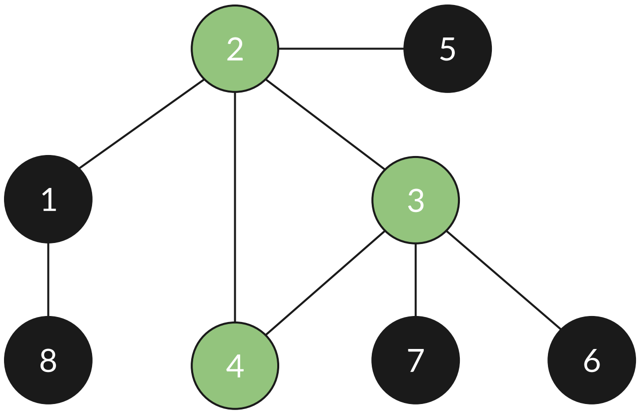
Results	4							



Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue, Q**
    a. Update Queue (a) with neighbors **(non-visited)**
    b. **Process data/key**



Q

3	2					

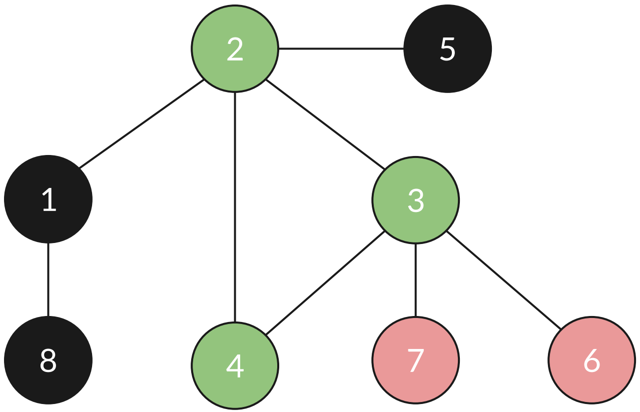
Results

4	3	2					

Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue, Q**
    a. **Update Queue (a) with neighbors (non-visited)**
    b. Process data/key



Q							
	2	7	6				

Results								
	4	3	2					

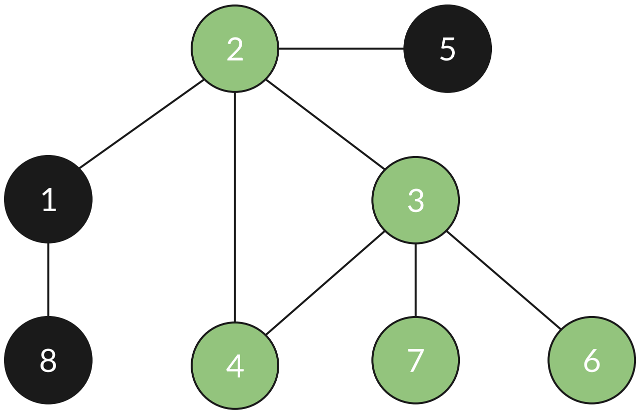
Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue, Q**
    a. Update Queue (a) with neighbors (non-visited)
    b. **Process data/key**

Q	2	7	6				

Results	4	3	2	7	6			



Graph Traversals

- Breadth First

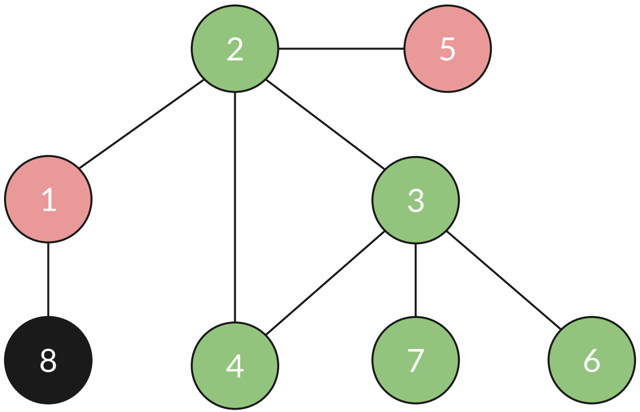
1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue, Q**
    a. **Update Queue (a) with neighbors (non-visited)**
    b. Process data/key

Q

7	6	1	5			

Results

4	3	2	7	6			



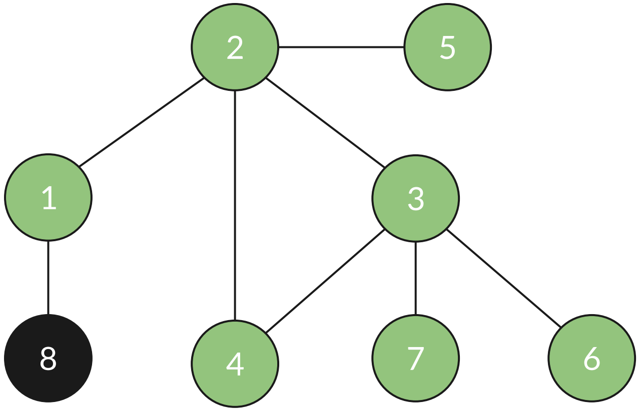
Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue, Q**
    a. Update Queue (a) with neighbors (non-visited)
    b. **Process data/key**

Q	7	6	1	5			

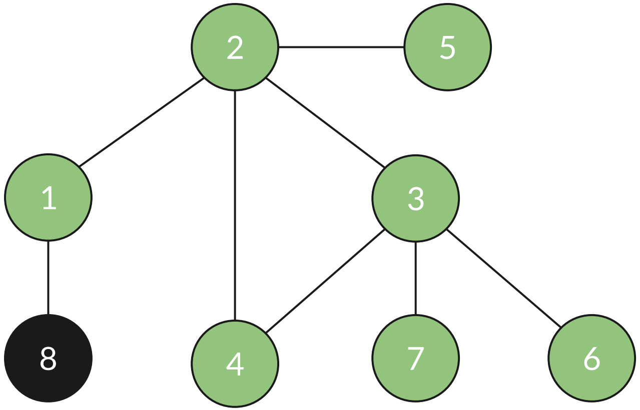
Results	4	3	2	7	6	1	5	



Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue, Q**
   a. **Update Queue (a) with neighbors (non-visited)**
   b. Process data/key



Q	7	6	1	5			

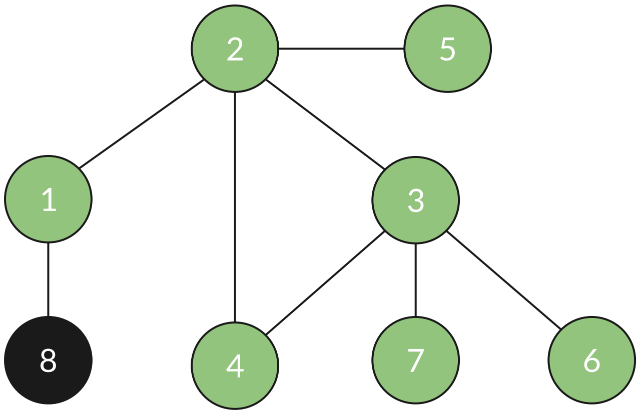
Results	4	3	2	7	6	1	5	

7’s neighbor 3 is already visited …

Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue, Q**
    a. **Update Queue (a) with neighbors (non-visited)**
    b. Process data/key



Q							
	6	1	5				

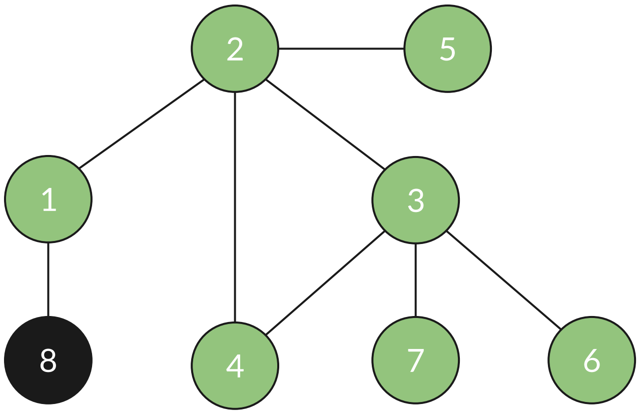
Results								
	4	3	2	7	6	1	5	

6’s neighbor 3 is already visited …

Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue, Q**
   a. **Update Queue (a) with neighbors (non-visited)**
   b. Process data/key



Q							
	1	5					

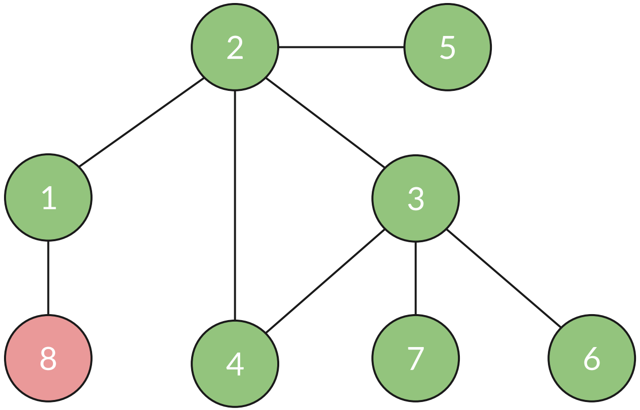
Results								
	4	3	2	7	6	1	5	

1’s neighbor 2 is already visited; 8 not

Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue, Q**
    a. **Update Queue (a) with neighbors (non-visited)**
    b. Process data/key



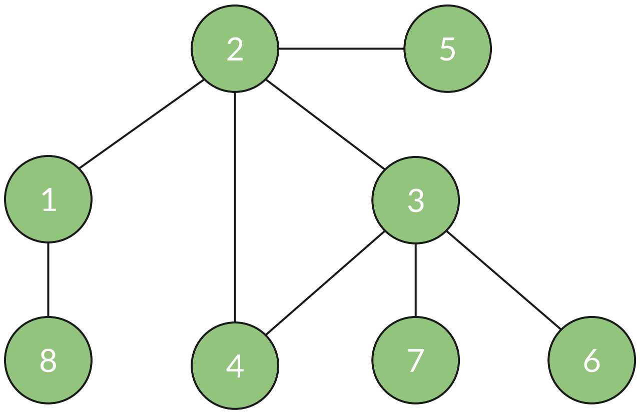
Q							
	5	8					

Results								
	4	3	2	7	6	1	5	

Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue, Q**
    a. Update Queue (a) with neighbors (non-visited)
    b. **Process data/key**



Q

5	8					

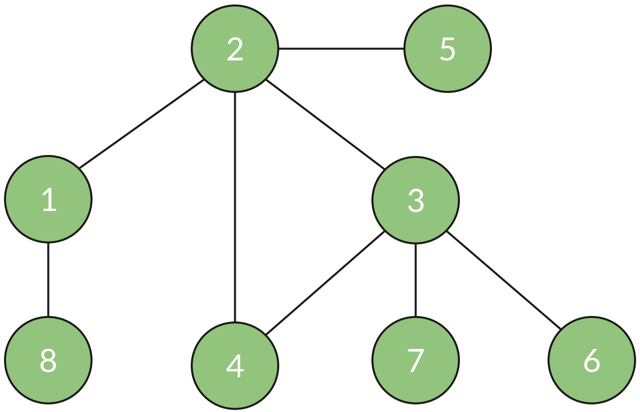
Results

4	3	2	7	6	1	5	8

Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue, Q**
    a. **Update Queue (a) with neighbors (non-visited)**
    b. Process data/key



Q							
	5	8					

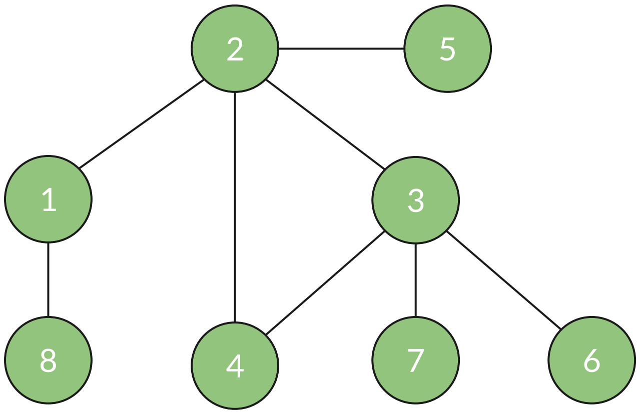
Results								
	4	3	2	7	6	1	5	8

5’s neighbor 2 is already visited …

Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue, Q**
    a. **Update Queue (a) with neighbors**
    b. Process data/key



Q

8						

Results

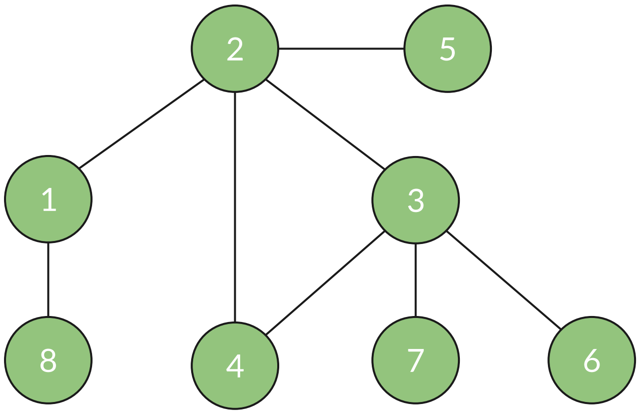
4	3	2	7	6	1	5	8

8’s neighbor 1 is already visited …

Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. Traverse one level at a time
3. To generalize, we will use an additional **Queue, Q**
    a. Update Queue (a) with neighbors
    b. **Process data/key**



Q							

Results	4	3	2	7	6	1	5	8

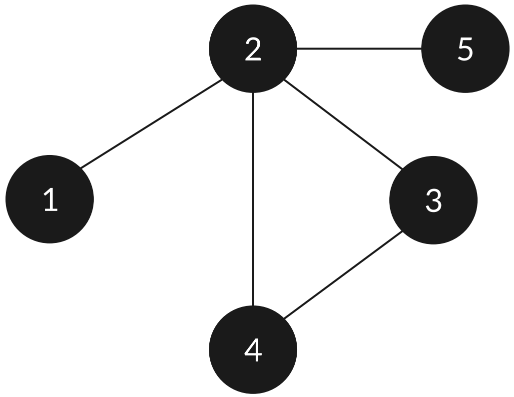
Q is empty; we finish

Graph (Data Structure) - Undirected

G = (V, E)

V = {v1, v2, v3, v4}
E = {{v1,v2}, {v2,v3}, {v3,v4}, {v2,v4}}

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

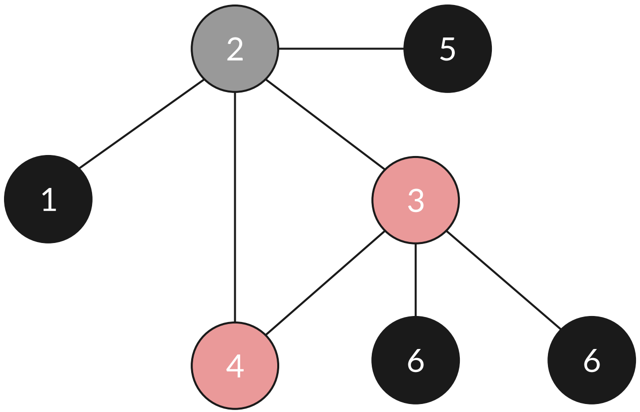


Using 2D Array
- n x n Matrix
- Sparse Matrix

Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. (a) Perform node operation, say (print (key)), if not already done
   (b) Perform Exploration (1-st level neighbors), any order
   (c) Iterate until all node visited



Results

4	3					

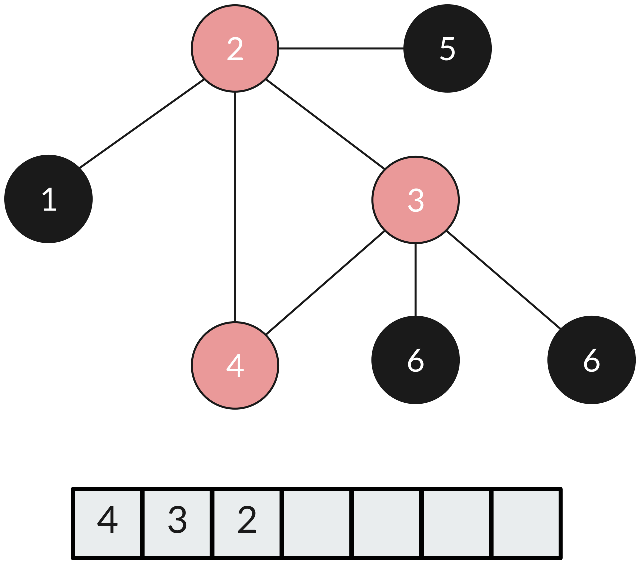
Q

4	3					

Graph Traversals

- Breadth First

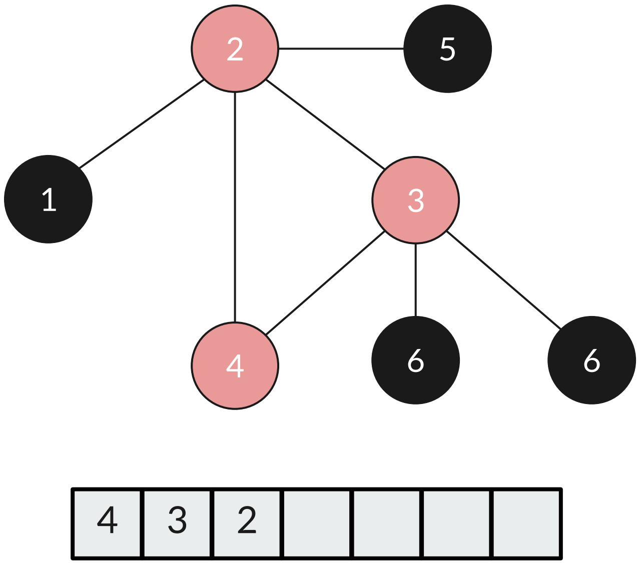
1. Start with any node, say (4)
2. (a) Perform node operation, say (print (key)), if not already done
   (b) Perform Exploration (1-st level neighbors), any order
   (c) Iterate until all node visited



Graph Traversals

- Breadth First

1. Start with any node, say (4)
2. (a) Perform node operation, say (print (key)), if not already done
   (b) Perform Exploration (1-st level neighbors), any order
   (c) Iterate until all node visited



QA