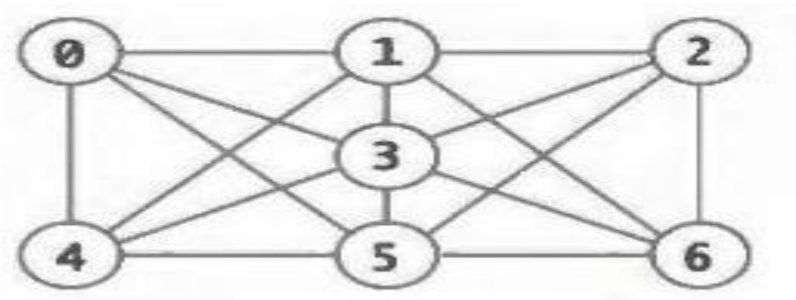


1-)Represent the graphs above using adjacency lists. Draw the corresponding data structure

NodeList index	Node1	Node2	Node3	Node4	Node5	
0	Value=1	Value=3	Value=5	Value=4		
1	Value=0	Value=4	Value=3	Value=6	Value=2	
2	Value=1	Value=3	Value=4	Value=6		
3	Value=0	Value=1	Value=2	Value=4	Value=5	Value=6
4	Value=0	Value=1	Value=3	Value=5		
5	Value=4	Value=0	Value=3	Value=2	Value=6	
6	Value=5	Value=3	Value=1	Value=2		

2-) Represent the graphs above using an adjacency matrix. Draw the corresponding data

Structure

Nodes/Nodes	Node0	Node1	Node2	Node3	Node4	Node5	Node6
Node0	INFINITY	1.0	INFINITY	1.0	1.0	1.0	INFINITY
Node1	1.0	INFINITY	1.0	1.0	1.0	INFINITY	1.0
Node2	INFINITY	1.0	INFINITY	1.0	INFINITY	1.0	1.0
Node3	1.0	1.0	1.0	INFINITY	1.0	1.0	1.0
Node4	1.0	1.0	INFINITY	1.0	INFINITY	1.0	INFINITY
Node5	1.0	INFINITY	1.0	1.0	1.0	INFINITY	1.0
Node6	INFINITY	1.0	1.0	1.0	INFINITY	1.0	INFINITY

3-) What are the IVI=n, the IEI=m, and the density? Which representation is better for each graph? Explain your answers.

Solution:

|V| means number of vertex's which equals to number of nodes in our implementation and |E| means number of edges. For undirected graphs which shown above, graph density would be defined as the ratio of the number of edges and the number of possible edges. In this graph, most of the possible choices that nodes can make is done so we can say that this graph is dense graph and fort his graph, all possible choices that graph can make can be calculated as $m_{max} = (n * (n-1)) / 2$ which is 7 * 6 / 2 = 21. If we calculate density, we get 16 / 21. This value approximately 0.76. For dense graphs, we use adjecency matrices because most of its cells are full and we will have quick insertions and deletions of edges.

4-) Draw DFS tree starting from vertex 2 and traversing the vertices adjacent to a vertex in descending order (largest to smallest).

Node that will be visit	Tree	Visited	Stack	Explanation
2	2	2		I put 2 all of them because 2 is first value
1	2 / 1	2 1	2	1 is less than every other nodes so we pass 2 to stack
0	2 / 1 / 0	2 1 0	2 1	0 is less than every other nodes and some of them visited so these nodes are not valid so we pass 1 to stack
3	2 / 1 / 0 / 3	2 1 0 3	2 1 0	3 is less than every other nodes and some of them visited so these nodes are not valid so we pass 0 to stack
4	2 / 1 / 0	2 1 0 3 4	2 1 0 3	4 is less than every other nodes and some of them visited so these nodes are not valid so we pass 3 to stack

	/ 3 / 4			
5	2 / 1 / 0 / 3 / 4 / 5	2 1 0 3 4 5	2 1 0 3 4	5 is less than every other nodes and some of them visited so these nodes are not valid so we pass 4 to stack

6	2 / 1 / 0 / 3 / 4 / 5 / 6	2 1 0 3 4 5 6	2 1 0 3 4 5	6 is less than every other nodes and some of them visited so these nodes are not valid so we pass 5 to stack
5	2 / 1 / 0 / 3 / 4 / 5	2 1 0 3 4 5 6	2 1 0 3 4	There is no childeren node of 6 so 5 will be popped from stack and look node that has value 5

	/ 6			
4	2 / 1 / 0 / 3 / 4 / 5 / 6	2 1 0 3 4 5 6	2 1 0 3	There is no childeren node of 5 that is unvisited so 4 will be popped from stack and look node that has value 4
3	2 / 1 / 0 / 3 / 4 /	2 1 0 3 4 5 6	2 1 0	There is no childeren node of 4 that is unvisited so 3 will be popped from stack and look node that has value 3

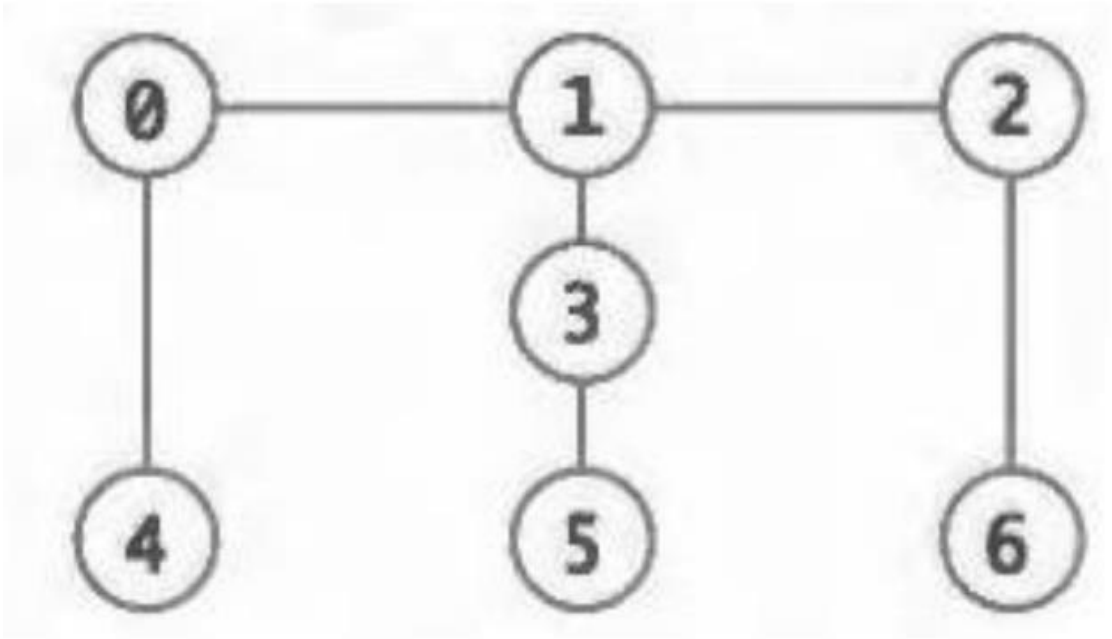
	5 / 6			
0	2 / 1 / 0 / 3 / 4 / 5 / 6	2 1 0 3 4 5 6	2 1	There is no childeren node of 3 that is unvisited so 0 will be popped from stack and look node that has value 0
1	2 / 1 / 0 / 3 / 4	2 1 0 3 4 5 6	2	There is no childeren node of 0 that is unvisited so 1 will be popped from stack and look node that has value 1

	/ 5 / 6			
2	2 / 1 / 0 / 3 / 4 / 5 / 6	2 1 0 3 4 5 6		There is no childeren node of 1 that is unvisited so 2 will be popped from stack and look node that has value 2. There is no value at stack left so that tree is our final tree

5-) Draw BFS tree starting from vertex 2 and traversing the vertices adjacent to a vertex in descending order (largest to smallest)

Node that will be visit	Tree	Visited	Queue	Explanation
2	2	2		Starting node added to tree as root node
1, 3, 5, 6	2 / / \ \ 1 3 5 6	2 1 3 5 6	1 3 5 6	Second degree nodes are added to tree
0, 4	2 / / \ \ 1 3 5 6 / \ 0 4	2 1 3 5 6 0 4	3 5 6 0 4	Third degree nodes are added to tree as child nodes of node with value 1
	2 / / \ \ 1 3 5 6 / \ 0 4	2 1 3 5 6 0 4	5 6 0 4	There is no node that is unvisited to node 3 so we dequeue 3
	2 / / \ \ 1 3 5 6 / \ 0 4	2 1 3 5 6 0 4	6 0 4	There is no node that is unvisited to node 5 so we dequeue 5

	2 / / \ \ 1 3 5 6 / \ 0 4	2 1 3 5 6 0 4	0 4	There is no node that is unvisited to node 6 so we dequeue 6
	2 / / \ \ 1 3 5 6 / \ 0 4	2 1 3 5 6 0 4	4	There is no node that is unvisited to node 0 so we dequeue 0
	2 / / \ \ 1 3 5 6 / \ 0 4	2 1 3 5 6 0 4		There is no node that is unvisited to node 4 so we dequeue 4. There is no node at queue so we end with that tree.



1-)Represent the graphs above using adjacency lists. Draw the corresponding data structure

NodeList index	Node1	Node2	Node3
0	Value=1	Value=4	
1	Value=0	Value=2	Value=3
2	Value=1	Value=6	
3	Value=1	Value=5	
4	Value=0		
5	Value=3		
6	Value=2		

2-) Represent the graphs above using an adjacency matrix. Draw the corresponding data

Structure

Nodes/Nodes	Node0	Node1	Node2	Node3	Node4	Node5	Node6
Node0	INFINITY	1.0	INFINITY	INFINITY	1.0	INFINITY	INFINITY
Node1	1.0	INFINITY	1.0	1.0	INFINITY	INFINITY	INFINITY
Node2	INFINITY	1.0	INFINITY	INFINITY	INFINITY	INFINITY	1.0
Node3	INFINITY	1.0	INFINITY	INFINITY	INFINITY	1.0	INFINITY
Node4	1.0	INFINITY	INFINITY	INFINITY	INFINITY	INFINITY	INFINITY
Node5	INFINITY	INFINITY	INFINITY	1.0	INFINITY	INFINITY	INFINITY
Node6	INFINITY	INFINITY	1.0	INFINITY	INFINITY	INFINITY	INFINITY

3-) What are the IVI=n, the IEI=m, and the density? Which representation is better for each graph? Explain your answers.

Solution:

|V| = n means number of vertex's which equals to number of nodes in our implementation and |E| = m means number of edges. For undirected graphs which shown above, graph density would be defined as the ratio of the number of edges and the number of possible edges. In this graph, not most of the connections between vertices are done so we can say that this graph is sparse graph and for this graph, all possible choices that graph can make can be calculated as m_{max} = (n * (n-1)) / 2 which is 7 * 6 / 2 = 21. If we calculate density, we get 6 / 21. This value approximately 0.28. For sparse graphs, we use adjecency lists because if we use adjecency matrix, most of our cells in our matrix will be empty so we use unnecessary space for empty cells. Because of this, we use adjecency list for keep sparse graphs.

4-) Draw DFS tree starting from vertex 2 and traversing the vertices adjacent to a vertex in descending order (largest to smallest).

Node that will be visit	Tree	Visited	Stack	Explanation
2	2	2		I put 2 all of them because 2 is first value
1	2 / 1	2 1	2	1 is less than every other nodes so we pass 2 to stack
0	2 / 1 / 0	2 1 0	2 1	0 is less than every other nodes so we pass 1 to stack

4	2 / \ 1 4 / 0	2 1 0 4	2 1 0	4 is only node that is connected to node that is not visited so 0 will be added to stack.
	2 / \ 1 4 / 0	2 1 0 4	2 1	There is no node left that is not visited so we will pop element from stack
3	2 / \ 1 4 / / 0 3	2 1 0 4 3	2 1 3	There is a node with value 3 that is not visited so we will add it to visited, stack and tree
5	2 / \ 1 4 / / \ 0 3 5	2 1 0 4 3 5	2 1 3 5	5 is only node that is connected to node 3 that is not visited so 5 will be added to stack, tree and visited.
5	2 / \ 1 4 / / \ 0 3 5	2 1 0 4 3 5	2 1 3	There is no node connected to node 5 so we just pop from the stack

3	2 / \ 1 4 / / \ 0 3 5	2 1 0 4 3 5	2 1	There is no node connected to node 3 that is unvisited so we just pop from the stack
1	2 / \ 1 4 / / \ 0 3 5	2 1 0 4 3 5	2	There is no node connected to node 1 that is unvisited so we just pop from the stack
6	2 / \ \ 1 4 6 / / \ 0 3 5	2 1 0 4 3 5 6	2 6	There is one node unvisited connected to 2 so we push node 6 to stack and add tree and as visited.
	2 / \ \ 1 4 6 / / \ 0 3 5	2 1 0 4 3 5 6	2	There is no node connected to node 6 that is unvisited so we just pop from the stack
	2 / \ \ 1 4 6 / / \ 0 3 5	2 1 0 4 3 5 6		There is no node connected to node 2 that is unvisited so we just pop from the stack

5-) Draw BFS tree starting from vertex 2 and traversing the vertices adjacent to a vertex in descending order (largest to smallest)

Node that will be visit	Tree	Visited	Queue	Explanation
2	2	2	2	I put 2 all of them because 2 is first value
1, 6	2 / \ 1 6	2 1 6	2 1 6	We see two nodes that is connected to node 2 so added them into queue and deque node 2
0, 3	2 / \ 1 6 / \ 0 3	2 1 6 0 3	1 6 0 3	We see two nodes that is connected to node 1 so added them into queue and deque node 1
	2 / \ 1 6 / \ 0 3	2 1 6 0 3	6 0 3	There is no node connected to 6 that is unvisited so deque

4	2 / \ 1 6 / \ 0 3 \ 4	2 1 6 0 3 4	0 3 4	There is an edge unvisited that is node 4 . This node added as children of node 0, stack and visited. There is no other node left so 0 will be deleted
5	2 / \ 1 6 / \ 0 3 \ \ 4 5	2 1 6 0 3 4 5	3 4 5	There is an edge unvisited that is node 5 . This node added as children of node 3, stack and visited. There is no other node left so 3 will be deleted
	2 / \ 1 6 / \ 0 3 \ \ 4 5	2 1 6 0 3 4 5	4 5	There is no node connected to 4 that is unvisited so deque

	2 / \ 1 6 / \ 0 3 \ \ 4 5	2 1 6 0 3 4 5	5	There is no node connected to 5 that is unvisited so deque
	2 / \ 1 6 / \ 0 3 \ \ 4 5	2 1 6 0 3 4 5		There is no node at stack so BFS finished