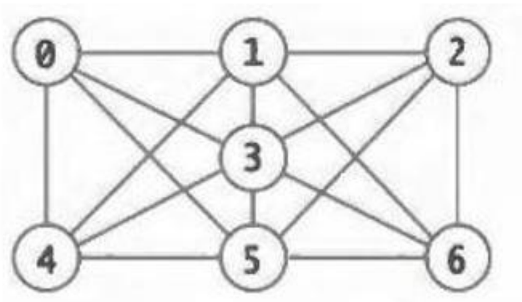
GIT Department of Computer Engineering

CSE 222/505 - Spring 2020

Homework #8 Part 1 Report

Murat YILDIZ

1801042004



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

İndexes / vertices	Edge Relations(Linked list)
0	→ 1 → 3 → 4 → 5
1	→ 0 → 2 → 3 → 4 → 6
2	→ 1 → 3 → 5 → 6
3	→ 0 → 1 → 2 → 4 → 5 → 6
4	→ 0 → 1 → 3 → 5
5	→ 0 → 2 → 3 → 4 → 6
6	→ 1 → 2 → 3 → 5

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

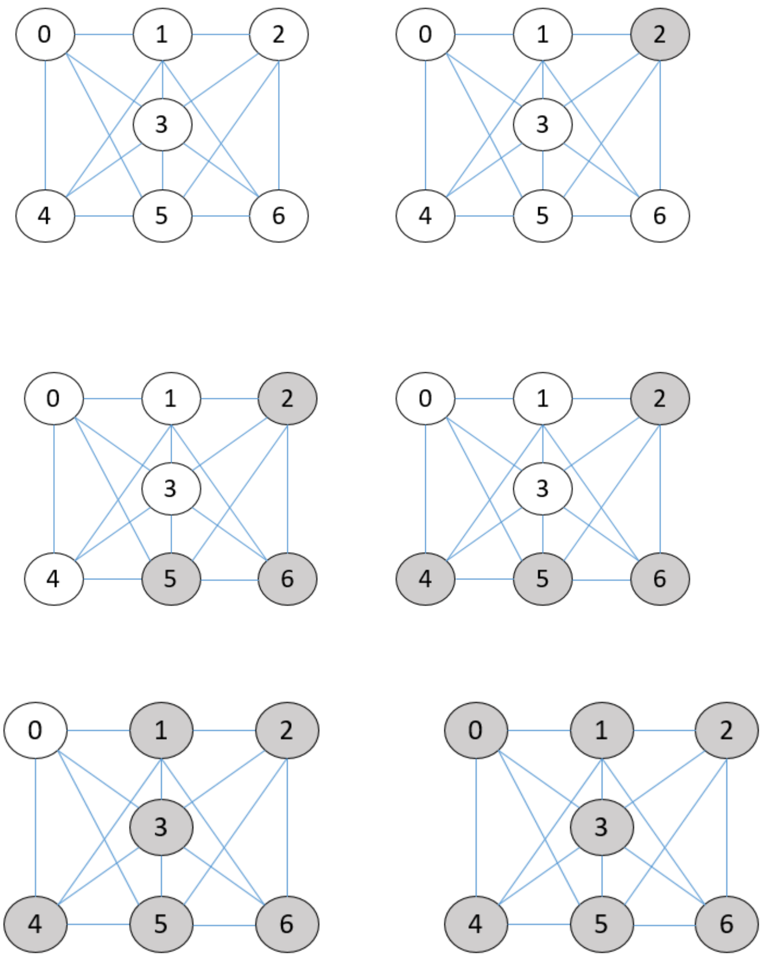
İndexes / vertices	0	1	2	3	4	5	6
0		1.0		1.0	1.0	1.0	
1	1.0		1.0	1.0	1.0		1.0
2		1.0		1.0		1.0	1.0
3	1.0	1.0	1.0		1.0	1.0	1.0
4	1.0	1.0		1.0		1.0	
5	1.0		1.0	1.0	1.0		1.0
6		1.0	1.0	1.0		1.0	

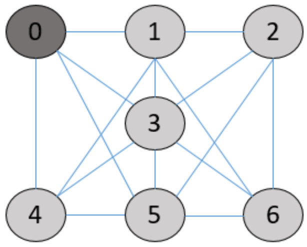
- For each graph above, what are the IVI=n, the IEI=m, and the density? Which representation is better for each graph? Explain your answers.

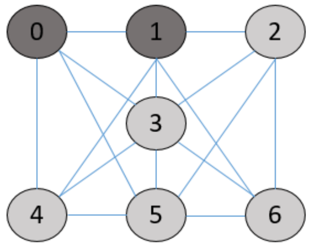
|V| = 7 vertices |E| = 32 edges Density = $|E|/|V|^2$ = 32 / 49 = 0.65

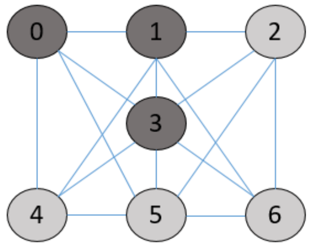
It is better to use matrix representation for graph because usage of storgae(memory space) is almost same but matrix representation is much faster than list graph.

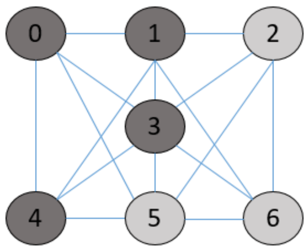
- Draw DFS tree starting from vertex 2 and traversing the vertices adjacent to a vertex in descending order (largest to smallest).

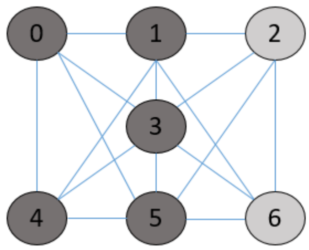


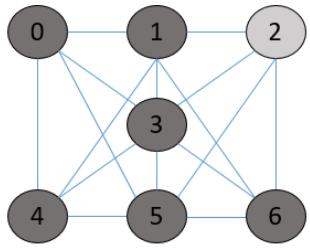


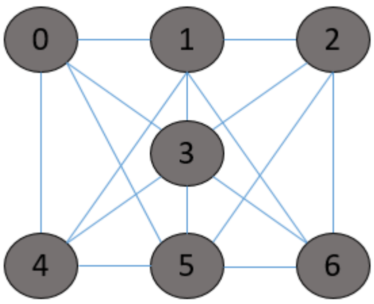






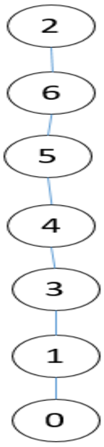




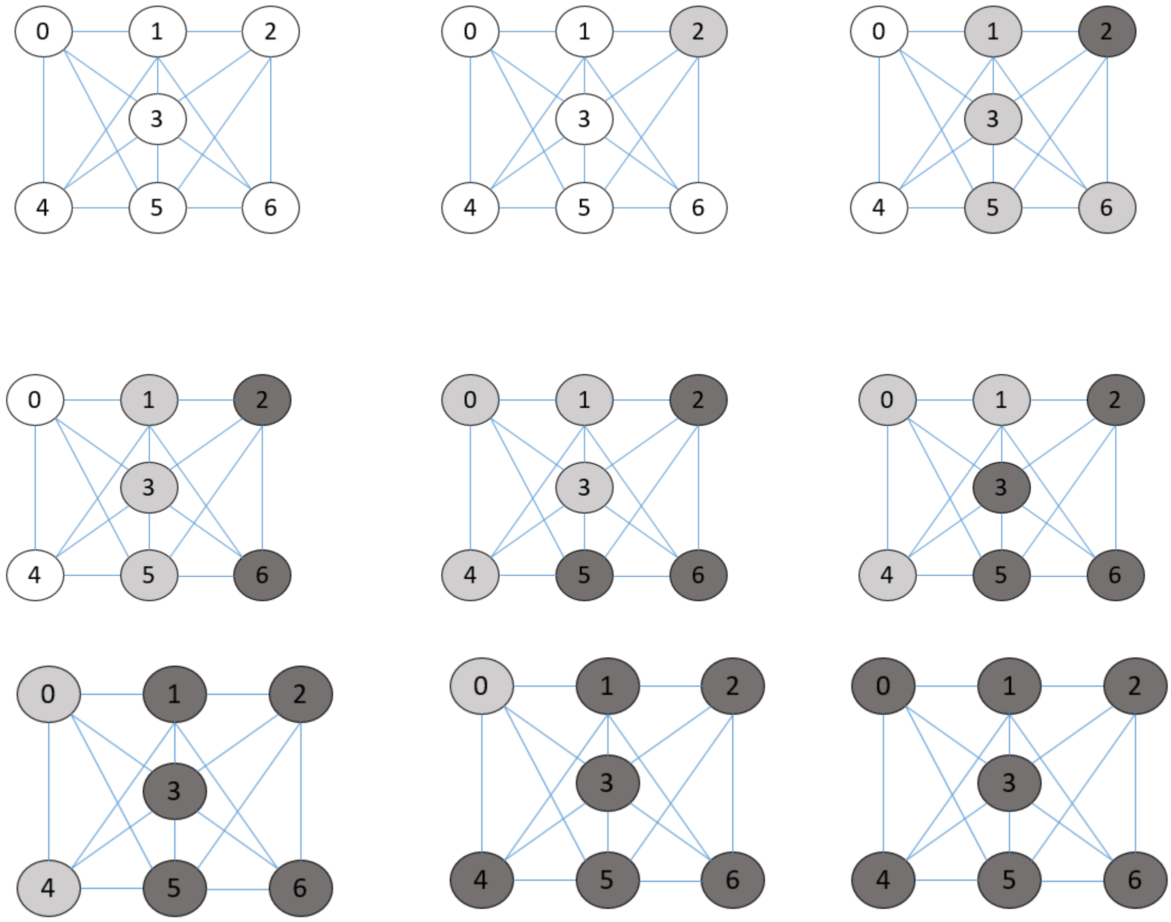


Operation	Adjacent Vertices	Discovery (Visit) Order	Finish Order
Visit 2	1, 3, 5, 6	2	
Visit 6	1, 2, 3, 5	2, 6	
Visit 5	0, 3, 4	2, 6, 5	
Visit 4	0, 1, 3, 5	2, 6, 5, 4	
Visit 3	0, 1, 2, 4, 5, 6	2, 6, 5, 4, 3	
Visit 1	0, 2, 3, 4, 6	2, 6, 5, 4, 3, 1	
Visit 0	1, 3, 4, 5	2, 6, 5, 4, 3, 1, 0	
Finish 0			0
Finish 1			0, 1
Finish 3			0, 1, 3
Finish 4			0, 1, 3, 4
Finish 5			0, 1, 3, 4, 5
Finish 6			0, 1, 3, 4, 5, 6
Finish 2			0, 1, 3, 4, 5, 6, 2

Depth-First Search Tree

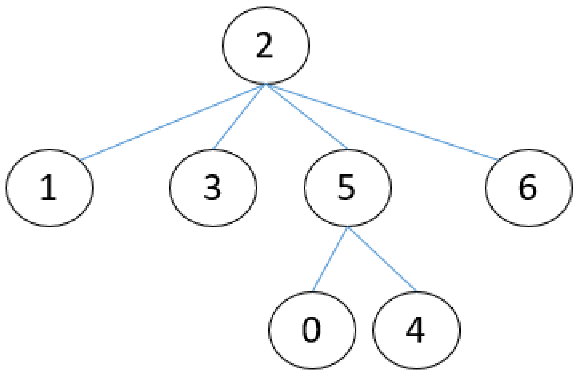


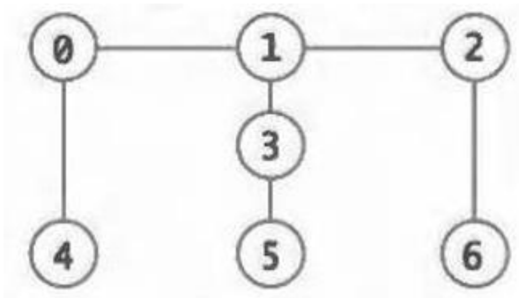
- Draw BFS tree starting from vertex 2 and traversing the vertices adjacent to a vertex in descending order (largest to smallest).



Vertex Being Visited	Queue Contents after Visit	Visit Sequence
2	6, 5, 3, 1	2
6	5, 3, 1	2, 6
5	3, 1, 4, 0	2, 6, 5
3	1, 4, 0	2, 6, 5, 3
1	4, 0	2, 6, 5, 3, 1
4	0	2, 6, 5, 3, 1, 4
0	Empty	2, 6, 5, 3, 1, 4, 0

Breadth-First Search Tree





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

İndexes / vertices	Edge Relations(Linked list)
0	→ 1 → 4
1	→ 0 → 2 → 3
2	→ 1 → 6
3	→ 1 → 5
4	→ 0
5	→ 3
6	→ 2

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

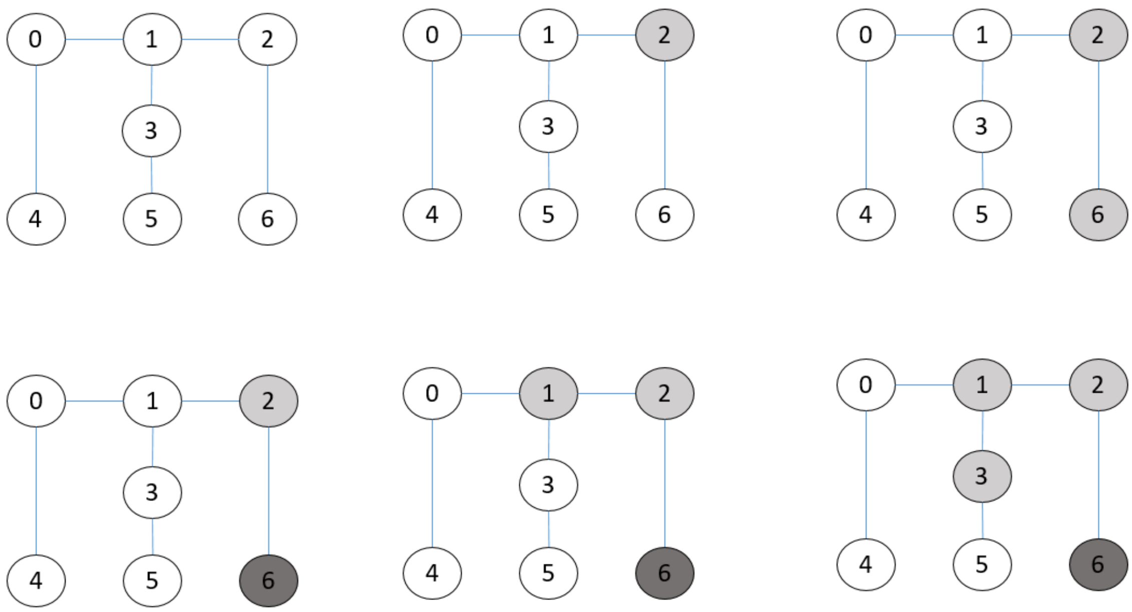
İndexes / vertices	**0**	**1**	**2**	**3**	**4**	**5**	**6**
0		1.0			1.0		
1	1.0		1.0	1.0			
2		1.0					1.0
3		1.0				1.0	
4	1.0						
5				1.0			
6			1.0				

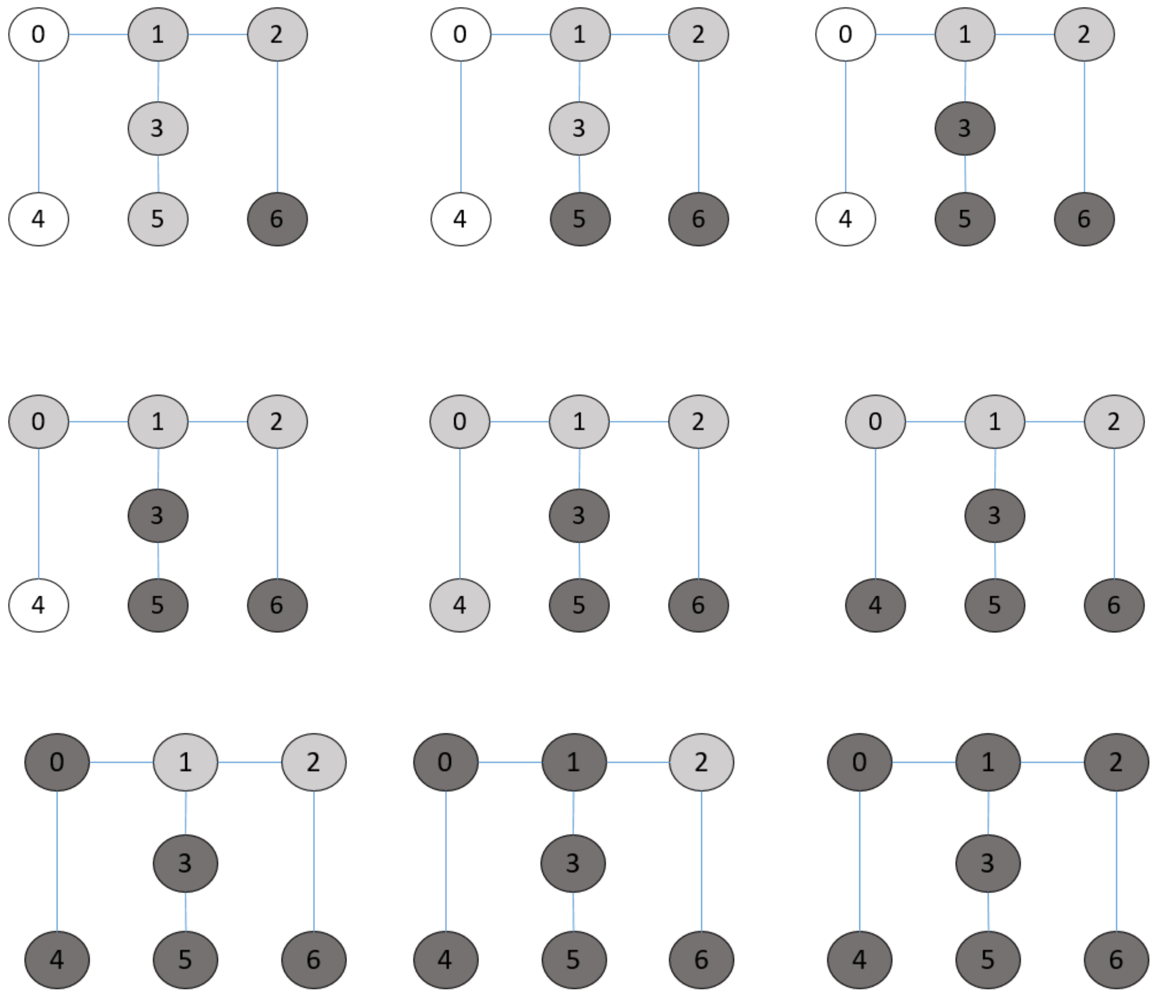
- For each graph above, what are the IVI=n, the IEI=m, and the density? Which representation is better for each graph? Explain your answers.

|V| = 7 vertices |E| = 12 edges Density = $|E|/|V|^2$ = 12 / 49 = 0.24

It is better to use list representation for graph because usage of storgae(memory space) is wasted in matrix representation altough it is much faster.

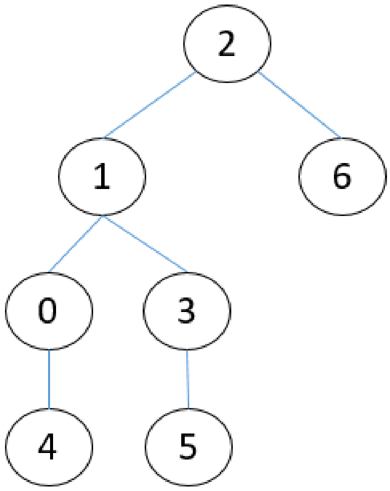
- Draw DFS tree starting from vertex 2 and traversing the vertices adjacent to a vertex in descending order (largest to smallest).



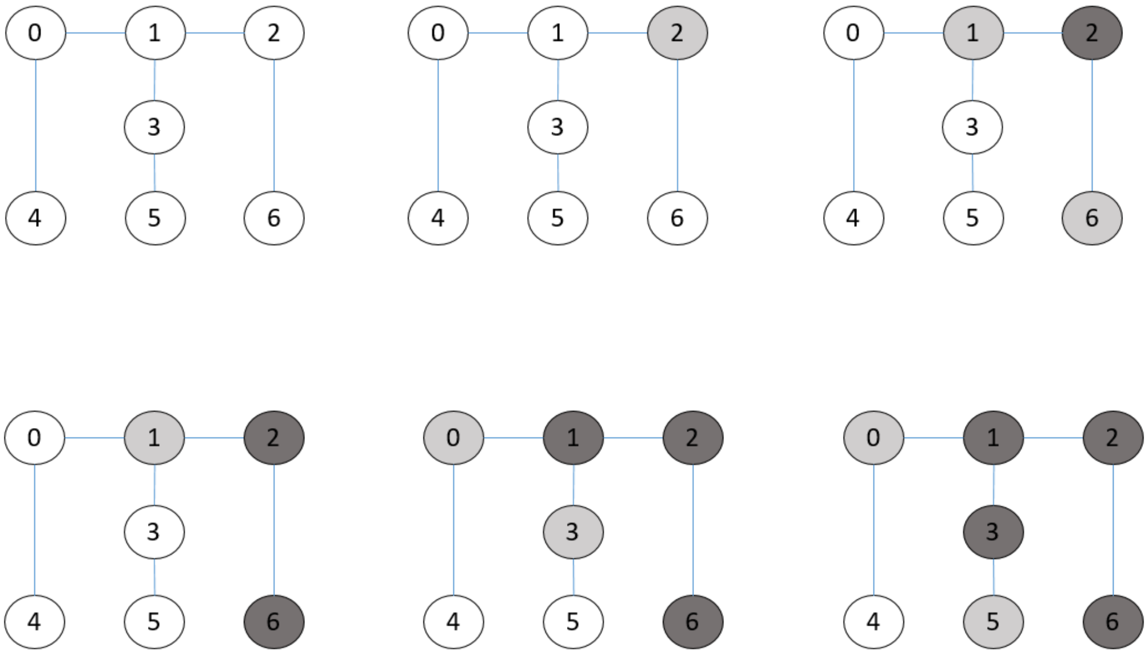


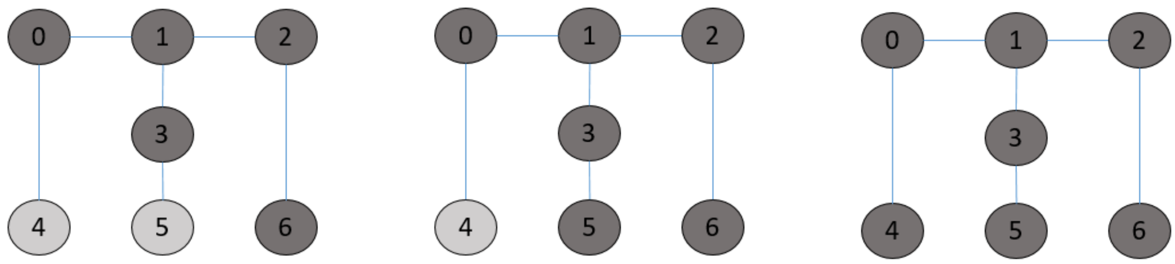
Operation	Adjacent Vertices	Discovery (Visit) Order	Finish Order
Visit 2	1, 6	2	
Visit 6	2	2, 6	
Finish 6			6
Visit 1	0, 3	2, 6, 5	
Visit 3	1, 5	2, 6, 5, 4	
Visit 5	3	2, 6, 5, 4, 3	
Finish 5			6, 5
Finish 3			6, 5, 3
Visit 0	1, 4	2, 6, 5, 4, 3, 1	
Visit 4	0	2, 6, 5., 4, 3, 1, 0	
Finish 4			6, 5, 3, 4
Finish 0			6, 5, 3, 4, 0
Finish 1			6, 5, 3, 4, 0, 1
Finish 2			6, 5, 3, 4, 0, 1, 2

Depth-First Search Tree



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





Vertex Being Visited	Queue Contents after Visit	Visit Sequence
2	**6, 1**	**2**
6	**1**	**2, 6**
1	**0, 3**	**2, 6, 1**
3	**0, 5**	**2, 6, 1, 3**
5	**0**	**2, 6, 1, 3, 5**
0	**4**	**2, 6, 1, 3, 5, 0**
4	**Empty**	**2, 6, 1, 3, 5, 0, 4**

Breadth-First Search Tree