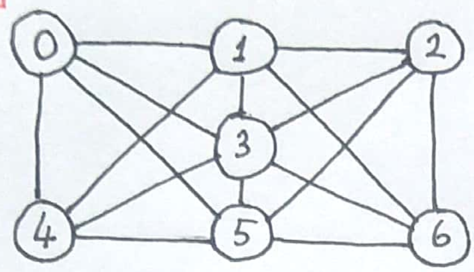
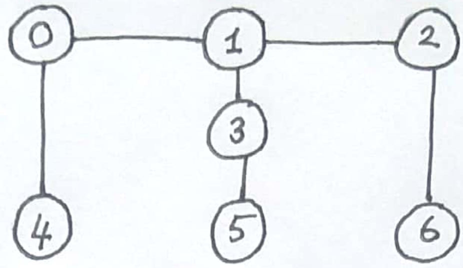
Esra Eryılmaz
171044046

Q1

First graph

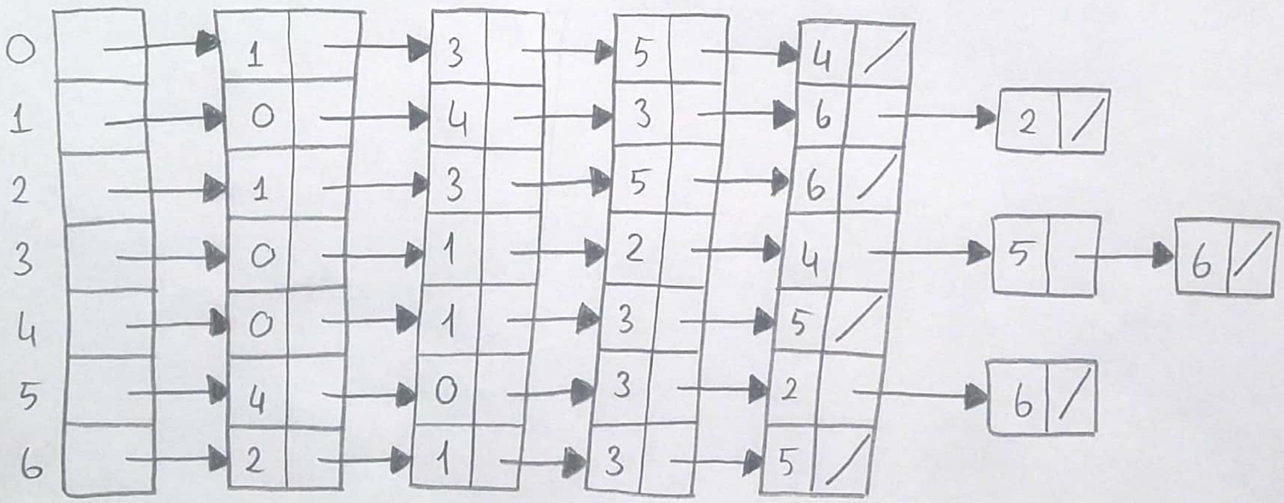


second graph

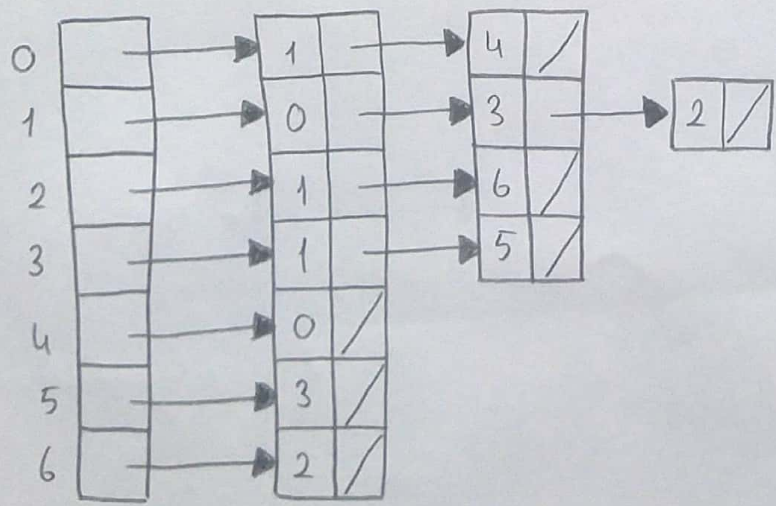


-Represent the graphs above using adjacency lists. Draw the corresponding data structure.

For the first graph



For the second graph



*Edges are represented by an array of list called adjacency lists where each list stores the vertices adjacent to a particular vertex.

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

For the first graph

column

row	[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	

* The adjacency matrix uses a two-dimentional array to represent the graph. Number of rows / columns will be equal to number of vertices.

Edge is indicated by value 1.0 and lack of an edge is indicated by blank space.

For the second graph

column

row	[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 $|V| = n$, the $|E| = m$, and the density? which representation is better for each graph?. Explain your answers.

For the first graph

∝ Number of vertex $= |V| = n = 7$

∝ Number of edges $= |E| = m = 16$

∝ The density of a graph is $\frac{|E|}{|V|^2} = \frac{16}{49} = 0.327$

Dense graph → too many edges → Density is close to 1, but less than 1.

∝ It will be dense graph, so adjacency matrix representation is better.

For the second graph

∝ Number of vertex $= |V| = n = 7$

∝ Number of edges $= |E| = m = 6$

∝ The density of a graph is $\frac{|E|}{|V|^2} = \frac{6}{49} = 0.122$

Sparse graph → too few edges → Density is much less than 1.

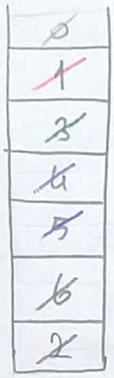
∝ It will be sparse graph, so adjacency list representation is better

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

For the first graph

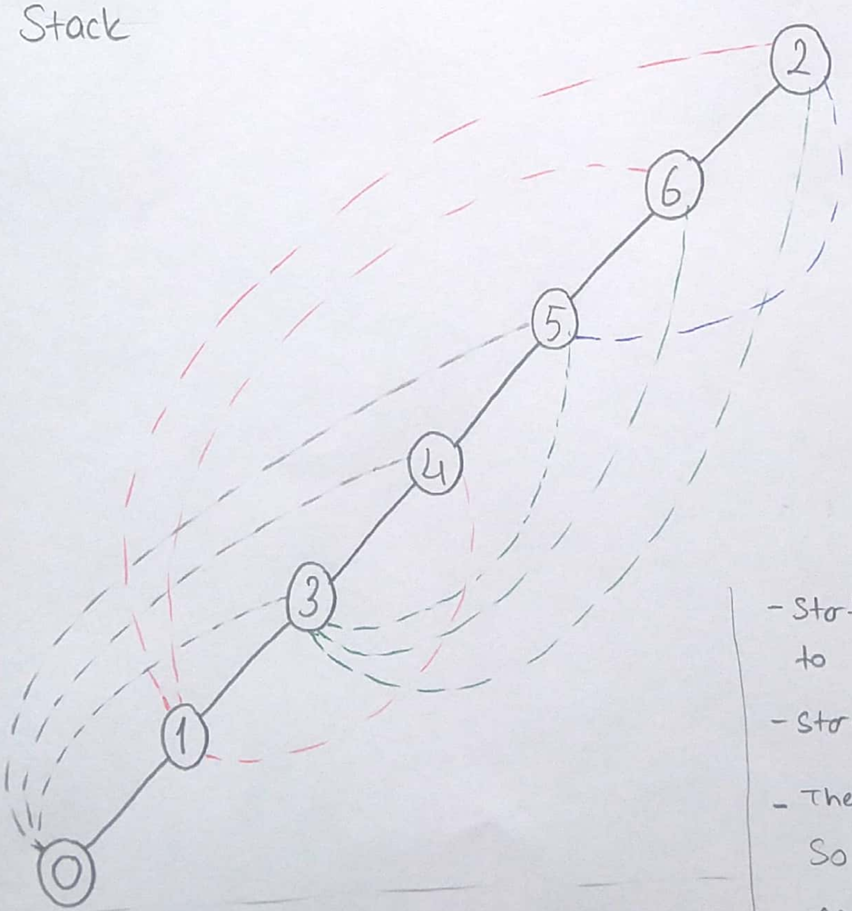
DFS : 2, 6, 5, 4, 3, 1, 0 → pre order traversal

unused vertices



Stack

- Start from vertex 2. Start explore 2. 1, 3, 5 and 6 adjacent to 2.
- Start exploring 6. 1, 3 and 5 adjacent to 6.
- Start exploring 5. 4, 0 and 3 adjacent to 5.
- Start exploring 4. 0, 1 and 3 adjacent to 4.
- Start exploring 3. 0 and 1 adjacent to 3.



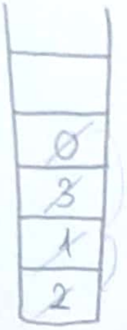
- Start exploring 1. 0 adjacent to 1.
- Start exploring 0.
- There is no vertex to explore So go back to stack and combine the remaining edges.

Finished

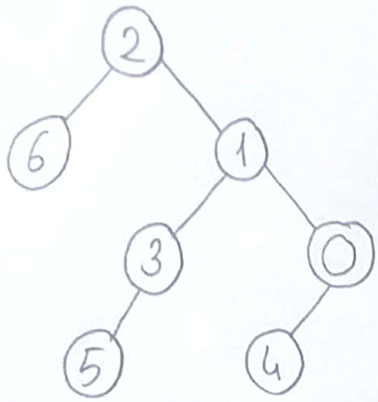
★ The rule in depth first search is, ones you have visited one vertex and still one more remaining; leave that. We will see it afterwords. We use stack for that reason.

For the second graph

DFS : 2, 6, 1, 3, 5, 0, 4 → pre order travelsal



Stack



- Start from vertex 2. Start explore 2.
  6 and 1 adjacent to 2.
- Start explore 6. Nothing explore to 6. Go back to 2.
- Start explore 1. 0 and 3 adjacent to 1.
- Start explore 3. 5 adjacent to 3.
- Stort explore 5. Nothing explore to 5. Go back to 3.
  Nothing explore to 3. Go back to 1.
- Start explore 0. 4 adjacent to 0.
- Start explore 4. Nothing explore to 4. Go back to 0.
  Go back 1 and 2. Finished

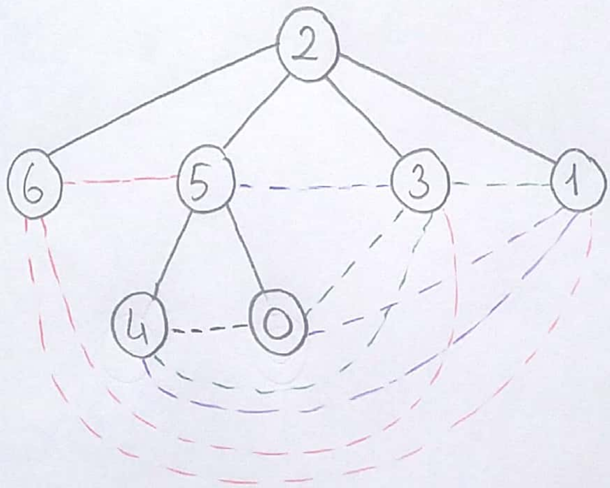
- Draw BFS tree starting from vertex 2 and traversing the vertices adjacent to a vertex in descending order (largest to smallest).

For the first graph

BFS: 2, 6, 5, 3, 1, 4, 0 → level order traversal

Queue

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



- Start from vertex 2. And write all adjacent vertices largest to smallest (6 - 5 - 3 - 1). 2 is completely explored.
- Explore 6.
- Explore 5. Write all adjacent vertices. (4 - 0).
- Explore 3.
- Explore 1.
- Explore 4.
- Explore 0.

Finished

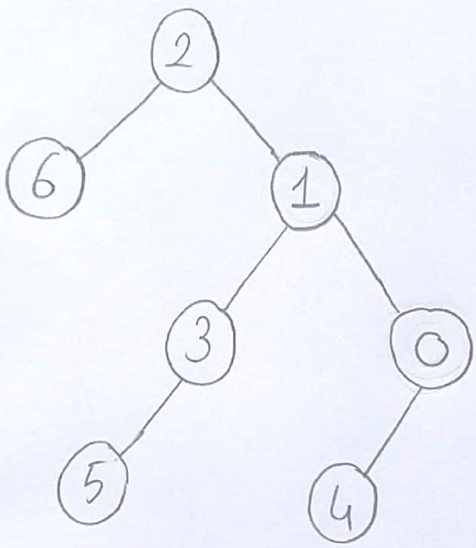
For the second graph

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

Queue

2	6	1	3	0	5	4

→ level order traversal



- Start from vertex 2. And write all adjacent vertices largest to smallest (6 - 1). 2 is completely explored.
- Explore 6. Nothing to explore.
- Explore 1. Write all adjacent vertices (3 - 0).
- Explore 3. Write all adjacent vertices (5).
- Explore 0. Write all adjacent vertices. (4).
- Explore 5. Nothing to explore.
- Explore 4. Nothing to explore.

(unused vertices)

Finished