# Graphs Practice Problems

P1. Traverse the graph below in **Breadth-First** order**, starting from vertex 2**. Whenever you examine neighbors of a vertex, process them in **increasing order** (see an example of neighbors ordering in the table). Half the points will be lost if you visit the neighbors in a different order.

Fill in the table below (the vertices, edges and the distance).
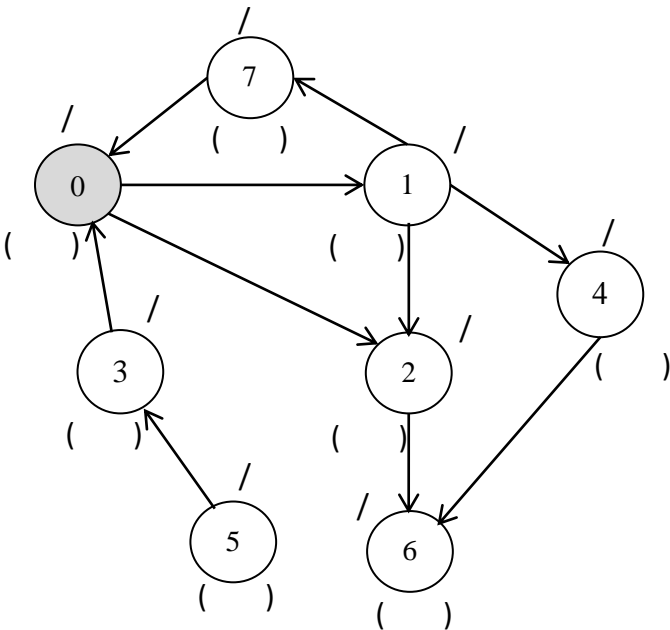In addition to that, write the predecessor in **( )** and the distance (as the number of edges) from the source in **[ ]** .
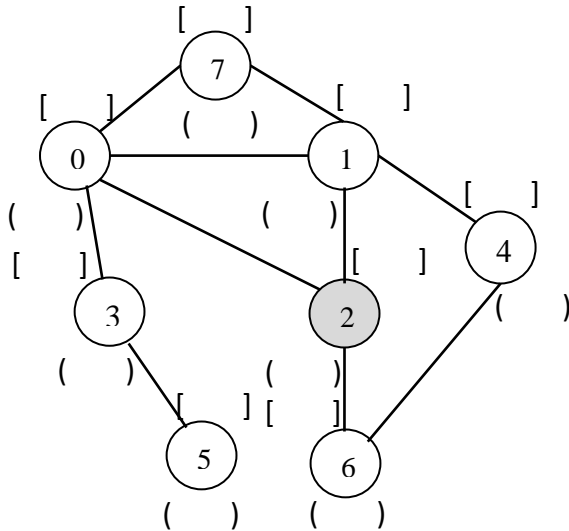
Order from first to last edges added to the BFS tree	Vertex, v, (in order visited by BFS)	Edge: u, v (edge that adds v to the BFS tree)	Distance (as number of edges) from the source
1st			
2nd			
...			

Helper table (to help process the neighbors

in the correct order)

Vertex	Neighbors in increasing order
0	1,2,3,7
1	
2	
3	
4	
5	
6	
7	

P2. Traverse this graph with Depth First Search (DFS) (start at vertex 0) and:

a) Write **next to each node** the **start** and **finish** times, as start/end, and also the **predecessor node** in ( ). If a vertex has more than one neighbors **you must visit them in increasing order,** for example from node 1 you would visit the neighbors in order: 2,4,7. Half the points will be lost if you visit the neighbors in a different order. (You can write the predecessor under the vertex or anywhere else around it.)

b) Write the predecessor below each node.

c) Label the edges with letters: **T** (tree), **B** (backward), **C** (cross), **F** (forward).

Order from first to last	Visited vertex	Start	Finish	Pred
1st	0			
2nd				
...				

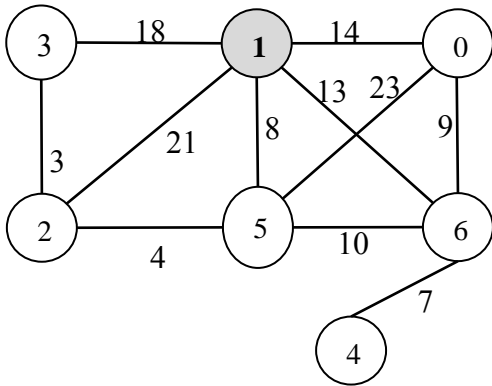| Vertex	Neighbors in increasing order
0	
1	2,4,7
2	
3	
4	
5	
6	
7	

P3. Run **Prim's algorithm** on this graph, to produce the minimum spanning tree (MST) **starting from vertex 5**. **Fill out the table below with edges** in the order in which they are selected by the algorithm and added to the MST. For each edge list the vertices and its weight, e.g. (2,5,14)

Order from first to last	Edge: u, v, weight
1st	
2nd	
...	
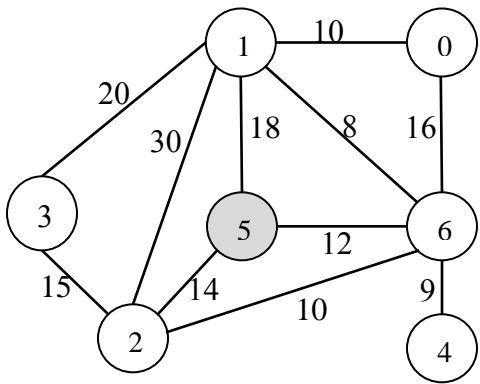
P4. Run **Kruskall's algorithm** on this graph, to produce the minimum spanning tree (MST). **Fill out in the table below edges** in the order in which they are selected by the algorithm and added to the MST. For each edge list the vertices and its weight, e.g. (2,5,40).

| Order from first to last	Edge: u, v, weight
1st	
2nd	
...	
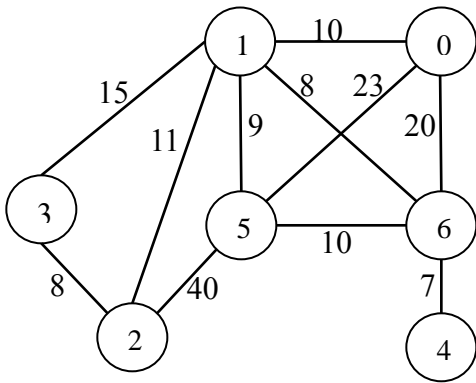
P5. (10 points) Run Dijkstra's algorithm for Single Source Shortest Path (SPST) **starting from vertex 1** in the graph below. If it matters, whenever you examine/process neighbors of a vertex, process them in **increasing order** of the vertex value. (If the order matters, the correct answer will be the one according to this processing) Print the vertices and their distance from vertex 1, in the order in which they are added to the tree. (Remember that it is looking for the shortest path from vertex 1 to any other vertex.)

Graph labels — vertices: 3, 1, 0, 2, 5, 6, 4. Edge weights: 18, 14, 13, 23, 21, 8, 9, 3, 3, 4, 10, 7.

Order from first to last vertex added to the SPST tree	Vertex, v, (in order added to the SPST)	Distance from vertex 1 to vertex v.	Pred
1st	**1**		
2nd			
…			

Vertex	Neighbors in increasing order
0	
1	
2	
3	
4	
5	
6	

P6. (9 points) (7 points) Label the edges for the graph below (T-tree, B-backward, C-cross, F-forward).

(2 points) Order in which vertices are first discovered: _____
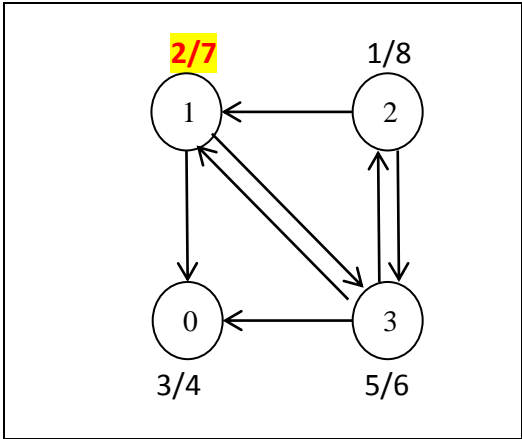
Edge	Label T/B/C/F
(1,0)	
(1,3)	
(2,1)	
(2,3)	
(3,0)	
(3,1)	
(3,2)	

P7. (6 points) If you are given only the finish times for the vertices of a graph traversed with Depth-First Search (DFS), **CAN you** list the graph vertices in **topological order**?
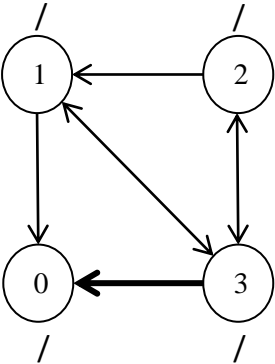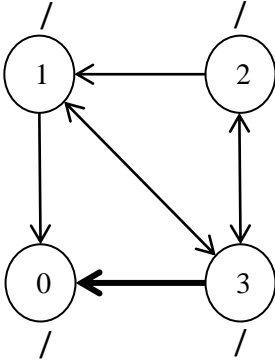
If Yes, **list the vertices** in topological order.

If No, justify **briefly**.

Vertex	0	1	2	3	4	5	6
Finish	10	5	14	13	7	4	9

P8. (16 points) An edge can be a **cross** edge or a **forward** edge depending on the order in which the nodes are visited. Give two DFS-Visit(G,s) traversals (including the start/end times) on the graph below s.t. the edge (3,0) is a **cross** edge one time and a **forward** edge the other time. In order to obtain this, you can process the neighbors of each vertex in whatever order you want and you can start from whatever vertex you want.

   a) (4 points) Both traversals start from the same vertex (s is the same).
   b) (4 points) Correct start/end times.
   c) (4 points) Traversal for **cross** edge.
   d) (4 points) Traversal for **forward** edge.

Traversal that makes (3,0) a **cross** edge	Traversal that makes (3,0) a **forward** edge
Vertices visited in this order: _____	Vertices visited in this order: _____