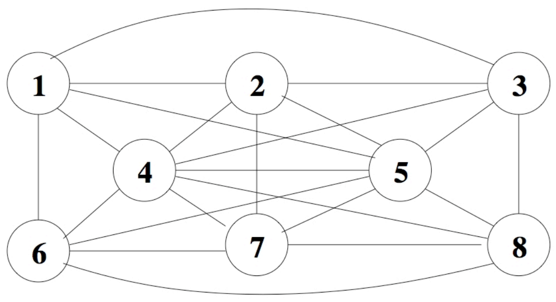
Data Structures and Algorithms – 2018

Assignment 5 – 0% of Continuous Assessment Mark

Not marked – do not submit answers to Canvas

Question 1 (16 marks)

Represent the following undirected graph as an adjacency matrix:



	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1	0	1	1	1	1	1	0	0
2	1	0	1	1	1	0	1	0
3	1	1	0	1	1	0	0	1
4	1	1	1	0	1	1	1	1
5	1	1	1	1	0	1	1	1
6	1	0	0	1	1	0	1	1
7	0	1	0	1	1	1	0	1
8	0	0	1	1	1	1	1	0

State in detail how depth-first traversal of the graph, starting from vertex 1, can be performed using a stack or queue. Write down the stack or queue and visited vertices at each stage.

Start with the initial vertex 1 in a stack. Then repeatedly remove the first vertex in the stack and add it to the list of vertices visited if it is not already in it, replacing it in the stack by the set of vertices connected to it. There is no need to add vertices to the stack that have already been visited. This gives the series of stages:

Stack	Visited vertices
1	
2, 3, 4, 5, 6	1
3, 4, 5, 7, 3, 4, 5, 6	1, 2
4, 5, 8, 4, 5, 7, 3, 4, 5, 6	1, 2, 3
5, 6, 7, 8, 5, 8, 4, 5, 7, 3, 4, 5, 6	1, 2, 3, 4
6, 7, 8, 6, 7, 8, 5, 8, 4, 5, 7, 3, 4, 5, 6	1, 2, 3, 4, 5
7, 8, 7, 8, 6, 7, 8, 5, 8, 4, 5, 7, 3, 4, 5, 6	1, 2, 3, 4, 5, 6
8, 8, 7, 8, 6, 7, 8, 5, 8, 4, 5, 7, 3, 4, 5, 6	1, 2, 3, 4, 5, 6, 7
8, 7, 8, 6, 7, 8, 5, 8, 4, 5, 7, 3, 4, 5, 6	1, 2, 3, 4, 5, 6, 7, 8

The remainder of the stack will then be emptied in order without adding any more vertices to the stack or list of vertices visited. (There are other possible orderings since the connected vertices are unordered at each stage.)

State in detail how breadth-first traversal of the graph, starting from vertex 1, can be performed using a stack or queue. Write down the stack or queue and visited vertices at each stage.

Start with the initial vertex 1 in a queue. Then repeatedly remove the first vertex in the queue and add it to the list of vertices visited if it is not already in it, and add the set of vertices connected to it to the end of the queue. There is no need to add vertices to the queue that have already been visited or are already in the queue. This gives the series of stages:

Queue	Visited vertices
1	
2, 3, 4, 5, 6	1
3, 4, 5, 6, 7	1, 2
4, 5, 6, 7, 8	1, 2, 3
5, 6, 7, 8	1, 2, 3, 4
6, 7, 8	1, 2, 3, 4, 5
7, 8	1, 2, 3, 4, 5, 6
8	1, 2, 3, 4, 5, 6, 7
	1, 2, 3, 4, 5, 6, 7, 8

(There are other possible orderings since the connected vertices are unordered at each stage.)

Question 2 (12 marks)

An undirected graph is said to be *connected* if and only if for every pair of non-identical vertices there exists a path from one vertex to the other. Explain in words how you could use graph traversal to determine whether a given graph is connected.

Perform either breadth-first or depth-first graph traversal from any vertex, and count the number of vertices visited. If that number equals the number of vertices in the graph, the graph is connected. Otherwise, it is not connected.

What is the computational complexity of breadth-first and depth-first traversal in terms of the number of vertices v and the number of edges e when the graph is represented as an adjacency matrix?

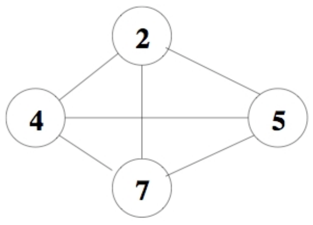
The complexity of both forms of traversal is $O(v^2)$ because every entry in the $v \times v$ adjacency matrix must be checked, regardless of whether a link exists.

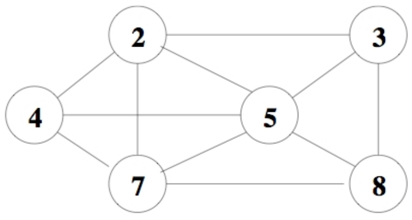
How do those computational complexities change if the graph is instead represented as an adjacency linked list?

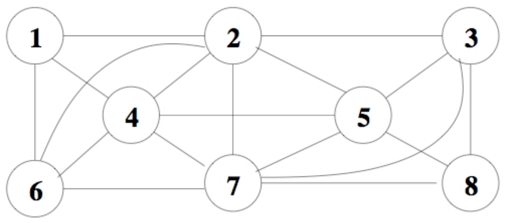
In this case, the complexity of both forms of traversal will be $O(v + e)$ because there are v linked lists and e links to follow.

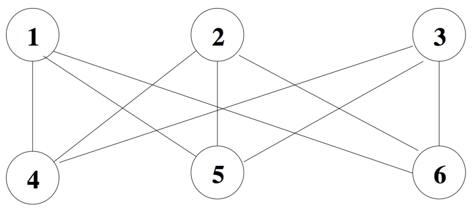
Question 3 (14 marks)

What property must a graph satisfy to be called *planar?* Determine, without resorting to any theorems concerning K_5 and $K_{3,3}$, which of the following graphs are planar, and which are not? In each case, explain in detail how you arrived at your answer.

(a)


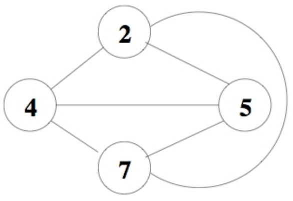
(b)


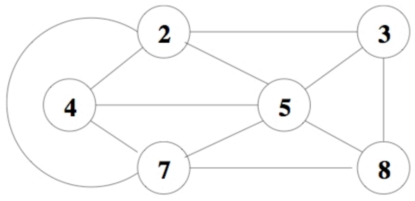
(c)


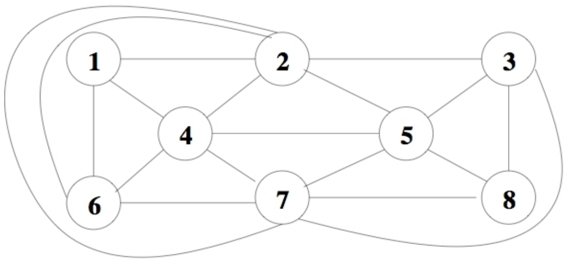
(d)


A graph is said to be *planar* if all its edges can be drawn in a two-dimensional plane with no edges crossing each other.

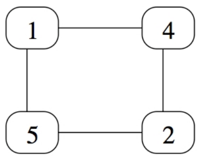
Graphs (a), (b) and (c) are planar because they can be redrawn without crossing edges:

(a)


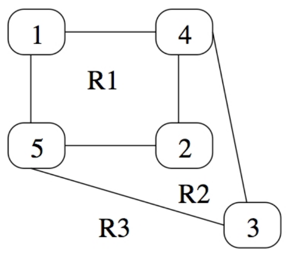
(b)


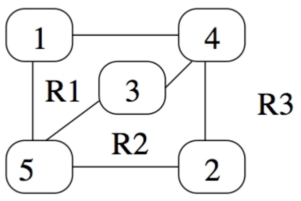
(c)


Graph (d) is not so straightforward, so one needs to explore systematically all possible ways of redrawing the graph, and conclude whether they will all involve at least one pair of edges crossing. There are many ways this can be done. For example, start with vertices 1, 4, 2 and 5 that form a circular sub-graph:



Then, if we next add vertex 3, it can either be placed inside or outside that circle:

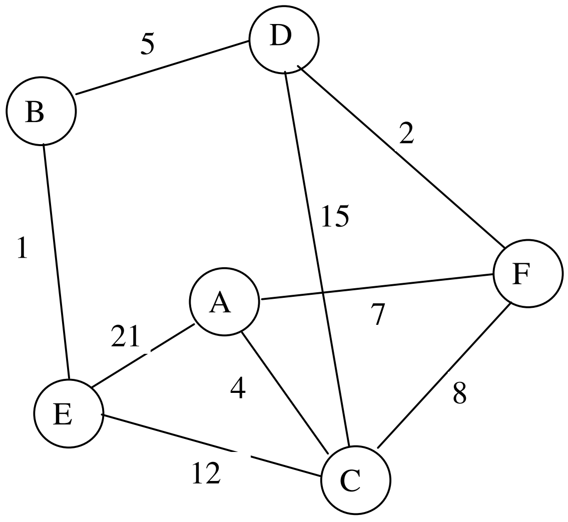




Either way leaves only three distinct regions R1, R2 and R3. If we then add the remaining vertex 6, whichever region we put it in leaves it unable to connect to at least one of vertices 1, 2 and 3 without crossing one of the existing edges. Therefore the graph is not planar.

Question 4 (24 marks)

Represent the following directed graph as a weight matrix:



	A	*B*	*C*	*D*	*E*	*F*
A	0	∞	4	∞	21	7
B	∞	0	∞	5	1	∞
C	4	∞	0	15	12	8
D	∞	5	15	0	∞	2
E	21	1	12	∞	0	∞
F	7	∞	8	2	∞	0

Represent the same graph as an array of adjacency lists.

[A] → [C | 4] → [E | 21] → [F | 7]
[B] → [D | 5] → [E | 1]
[C] → [A | 4] → [D | 15] → [E | 12] → [F | 8]
[D] → [B | 5] → [C | 15] → [F | 2]
[E] → [A | 21] → [B | 1] → [C | 12]
[F] → [A | 7] → [C | 8] → [D | 2]

Use an array-based version of Dijkstra's algorithm to determine a shortest path from vertex A to vertex E in this graph. At each stage show the estimated distances, which nodes are tight, the previous vertices, and which non-tight node has minimal estimate. At the end, explain how the shortest path is extracted from your computations, and state its length.

Computing shortest paths from A:

	A	B	C	D	E	F
D	0	∞	∞	∞	∞	∞
tight	no	no	no	no	no	no
pred	none	none	none	none	none	none

Vertex A has minimal estimate, and so is tight
Neighbours E, C and F have estimates reduced by shortcuts via A:

	A	B	C	D	E	F
D	0	∞	4	∞	21	7
tight	yes	no	no	no	no	no
pred	none	none	A	none	A	A

Vertex C has minimal estimate, and so is tight
Neighbours E and D have estimates reduced by shortcuts via C:

	A	B	C	D	E	F
D	0	∞	4	19	16	7
tight	yes	no	yes	no	no	no
pred	none	none	A	C	C	A

Vertex F has minimal estimate, and so is tight
Neighbour D has estimate reduced by shortcuts via F:

	A	B	C	D	E	F
D	0	∞	4	9	16	7
tight	yes	no	yes	no	no	yes
pred	none	none	A	F	C	A

Vertex D has minimal estimate, and so is tight
Neighbour B has estimate reduced by shortcuts via D:

	A	B	C	D	E	F
D	0	14	4	9	16	7
tight	yes	no	yes	yes	no	yes
pred	none	D	A	F	C	A

Vertex B has minimal estimate, and so is tight
Neighbour E has estimate reduced by shortcuts via B:

	A	B	C	D	E	F
D	0	14	4	9	15	7
tight	yes	yes	yes	yes	no	yes
pred	none	D	A	F	B	A

Vertex E has minimal estimate, and so is tight
No neighbour has estimate reduced by shortcuts via E:

	A	B	C	D	E	F
D	0	14	4	9	15	7
tight	yes	yes	yes	yes	yes	yes
pred	none	D	A	F	B	A

An acceptable shorthand representation for the whole seven stage process, using a “*” to represent tight, would be:

Stage	A	B	C	D	E	F
1	0	∞	∞	∞	∞	∞
2	0*	∞	4 A	∞	21 A	7 A
3	0*	∞	4* A	19 C	16 C	7 A
4	0*	∞	4* A	9 F	16 C	7* A
5	0*	14 D	4* A	9* F	16 C	7* A
6	0*	14* D	4* A	9* F	15 B	7* A
7	0*	14* D	4* A	9* F	15* B	7* A

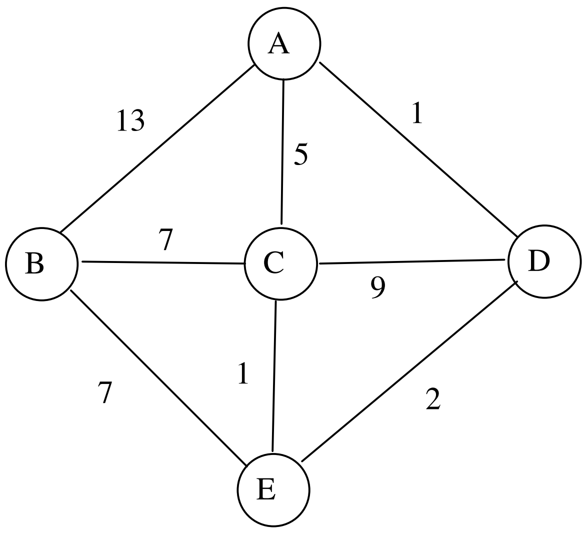
Either way, the minimal cost for E is 15 and the reverse path is E, B, D, F, A, so the required path is A, F, D, B, E.

Explain whether Dijkstra’s algorithm would work correctly on graphs that may have negative weights.

Dijkstra’s algorithm will not work correctly in such cases. It relies on knowing when further short-cuts are no longer available, and negative weights would make that impossible.

Question 5 (14 marks)

Use Floyds’s algorithm to determine the lengths of the shortest paths between all vertices in the following undirected weighted graph:



Explain what you have computed at each stage.

An array of distances is initialized with the weights from the graph (table 0), and then each vertex in turn (surrounded by double lines) is considered as a possible shortcut, with any shorter distances updated in the following table (in bold).

0	A	B	C	D	E
A	0	13	5	1	∞
B	13	0	7	∞	7
C	5	7	0	9	1
D	1	∞	9	0	2
E	∞	7	1	2	0

1	A	B	C	D	E
A	0	13	5	1	∞
B	13	0	7	**14**	7
C	5	7	0	**6**	1
D	1	**14**	**6**	0	2
E	∞	7	1	2	0

2	A	B	C	D	E
A	0	13	5	1	**20**
B	13	0	7	14	7
C	5	7	0	6	1
D	1	14	6	0	2
E	**20**	7	1	2	0

3	A	B	C	D	E
A	0	**12**	5	1	**6**
B	**12**	0	7	14	7
C	5	7	0	6	1
D	1	14	6	0	2
E	**6**	7	1	2	0

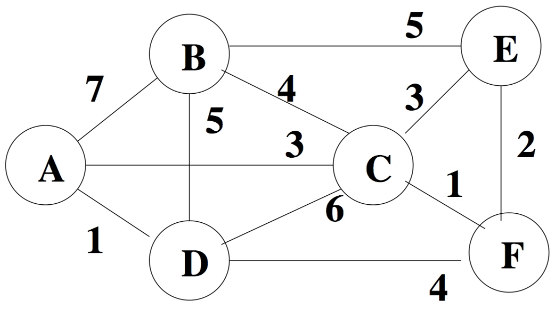
4	A	B	C	D	E
A	0	12	5	1	**3**
B	12	0	7	14	7
C	5	7	0	6	1
D	1	14	6	0	2
E	**3**	7	1	2	0

5	A	B	C	D	E
A	0	**10**	**4**	1	3
B	**10**	0	7	**9**	7
C	**4**	7	0	**3**	1
D	1	**9**	**3**	0	2
E	3	7	1	2	0

The final table (table 5) is the array of shortest paths.

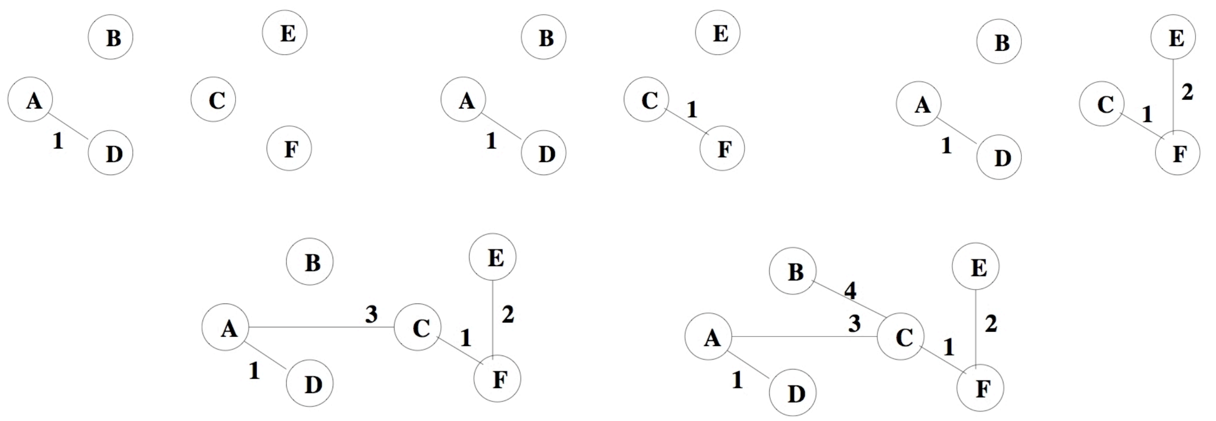
Question 6 (20 marks)

Consider the following weighted graph:



Use Kruskal's algorithm to determine a minimal spanning tree. For each stage, show the set of edges that have been added so far.

A minimal spanning tree is constructed by adding minimal weight edges without introducing circles as follows:

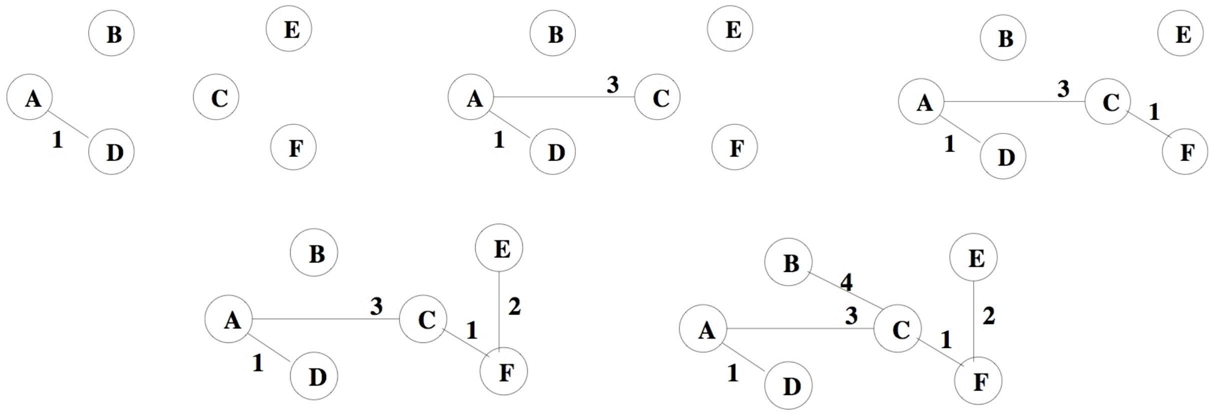


<span style="color:red">Explain whether Kruskal’s algorithm would work correctly on graphs that may have negative weights.</span>

Kruskal’s algorithm will work correctly. At each step it simply adds the minimal weight edge that does not create a circle, and the sign of the weight makes no difference.

For the same graph, use Prim’s algorithm to determine a minimal spanning tree starting from vertex A. For each stage, show the tree that have been created so far.

A minimal spanning tree is built up by adding unconnected vertices using the connecting edge with minimal weight as follows:



Explain whether Prim’s algorithm would work correctly on graphs that may have negative weights.

Prim’s algorithm will work correctly. Starting with any vertex, it always chooses to add an edge with lowest weight into the non-connected set of vertices until a spanning tree is generated. The sign of the weights makes no difference.