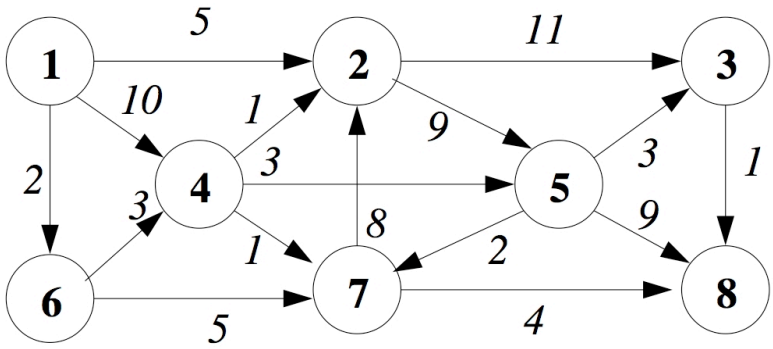
Foundations of Computer Science (Semester 2) – 2015

Assessed Exercise Sheet 10 – 10% of Continuous Assessment Mark

Deadline : 11pm Sunday 29th March, via Canvas

Question 1 (22 marks)

Use Dijkstra's algorithm to determine a shortest path from vertex 1 to vertex 8 in the following directed graph:



At each stage show which nodes are tight, the estimated distances, 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.

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Dist	0 min	∞	∞	∞	∞	∞	∞	∞
Tight	no	no	no	no	no	no	no	no
Prev	–	–	–	–	–	–	–	–
Dist	0	5	∞	10	∞	2 min	∞	∞
Tight	yes	no	no	no	no	no	no	no
Prev	–	1	–	1	–	1	–	–
Dist	0	5 min	∞	5	∞	2	7	∞
Tight	yes	no	no	no	no	yes	no	no
Prev	–	1	–	6	–	1	6	–
Dist	0	5	16	5 min	14	2	7	∞
Tight	yes	yes	no	no	no	yes	no	no
Prev	–	1	2	6	2	1	6	–
Dist	0	5	16	5	8	2	6 min	∞
Tight	yes	yes	no	yes	no	yes	no	no
Prev	–	1	2	6	4	1	4	–
Dist	0	5	16	5	8 min	2	6	10
Tight	yes	yes	no	yes	no	yes	yes	no
Prev	–	1	2	6	4	1	4	7

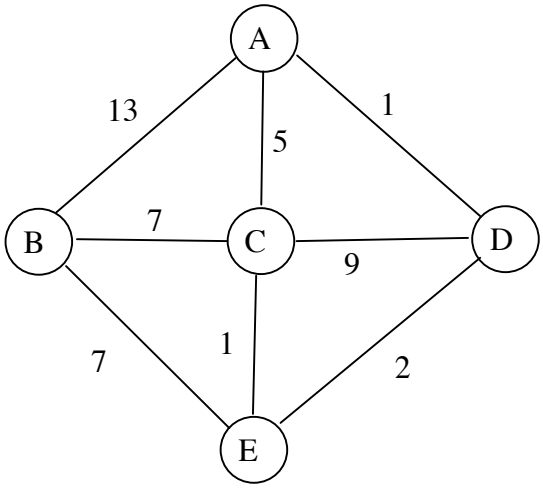
Dist	0	5	11	5	8	2	6	10 min
Tight	yes	yes	no	yes	yes	yes	yes	no
Prev	–	1	5	6	4	1	4	7
Dist	0	5	11 min	5	8	2	6	10
Tight	yes	yes	no	yes	yes	yes	yes	yes
Prev	–	1	5	6	4	1	4	7
Dist	0	5	11	5	8	2	6	10
Tight	yes	yes	yes	yes	yes	yes	yes	yes
Prev	–	1	5	6	4	1	4	7

Starting from the end vertex (8) read off the previous vertices back to the start vertex (1) to give 8, 7, 4, 6, 1, and reverse to give the shortest path: 1, 6, 4, 7, 8. The length of that shortest path is 10.

Simplified versions of the table are perfectly acceptable, as long as they contain the same basic information.

Question 2 (22 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 started with the weights from the graph (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). The final table (5) is the array of shortest paths.

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

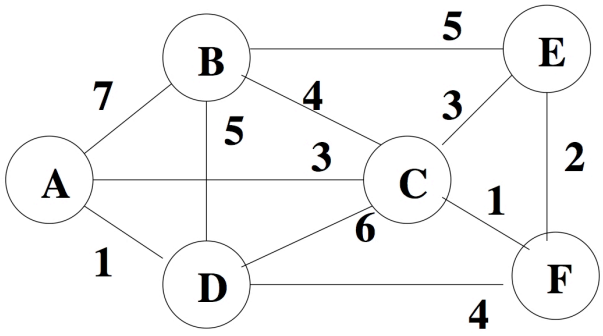
3	A	B	C	D	E
A	0	**12**	5	1	**6**
B	**12**	0	7	14	7
C	5	7	0	6	1
D	1	14	6	0	2
E	**6**	7	1	2	0

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

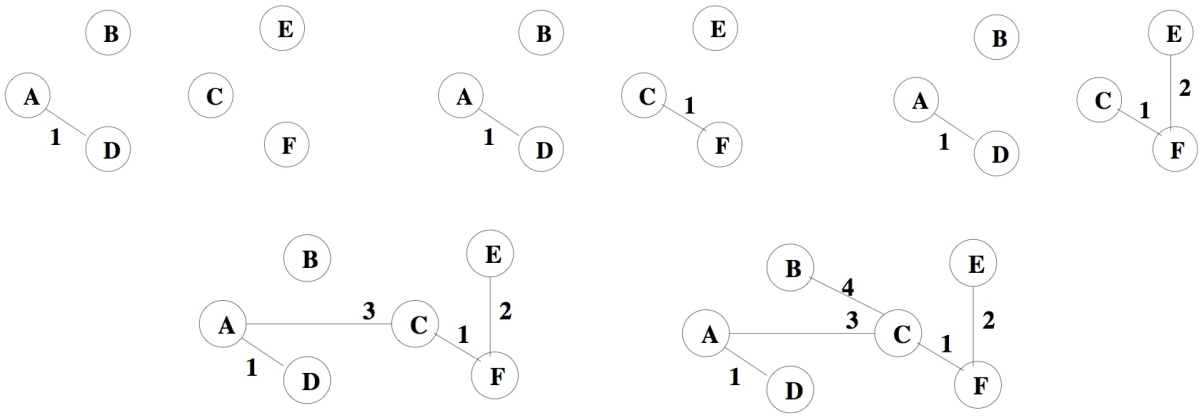
Question 3 (14 marks)

Use Kruskal's algorithm to determine a minimal spanning tree for the following graph:



Show the edges that have been added to your tree at each stage.

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

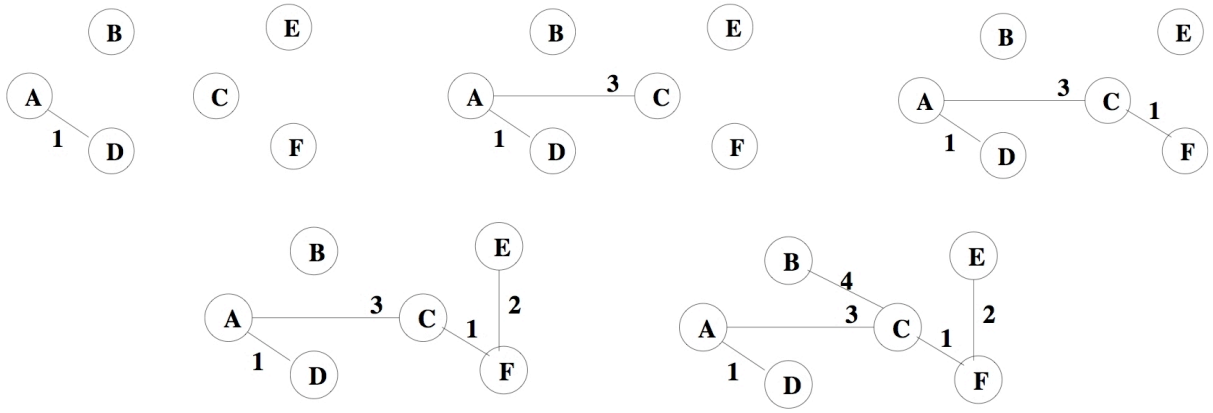


Question 4 (14 marks)

Use Prim's algorithm to determine a minimal spanning tree for the graph in question 3, starting from vertex A.

Show the state of your tree at each stage.

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



Question 5 (12 marks)

Explain whether either of Dijkstra's or Prim's algorithms will work correctly on graphs that may have negative weights.

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

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.

Question 6 (16 marks)

Suppose you have an undirected weighted graph, and starting at a given node you wish to visit each vertex exactly once, returning to the starting point, with minimal overall cost. This is the classic Travelling Salesperson Problem. Outline a simple, though not necessarily efficient, algorithm that would solve that problem. What is it's computational complexity?

Three steps are required:

1. Generate all possible orders of the vertices.

2. Compute the cost of traversing the vertices for each of those orders.

3. Find the order with the minimum cost.

If there are n vertices, there will be $(n\text{-}1)!$ different orders to compute and compare, each with $O(n)$ complexity, so the overall computational complexity will be $O(n!)$.