CS917 Foundations of Computing - Algorithms and Complexity

This assessed piece of work must be submitted by **Midday, Monday 12th December, 2016**. Submission is done electronically through Tabula, and two files are expected for submission:

1. A PDF of your solutions;
2. A file named Graph.java for the solution to Question 8.

The solutions submitted must be your **own** work. You are expected to work individually, not in groups. Please show full working where appropriate, detailing how you arrived at your answer.

If you have questions or issues, please feel free to email me at jad@dcs.warwick.ac.uk

Question 1 (5 points)

Suppose you have a machine that can perform 987 operations per second. Given the following functions (assume this is the exact number of operations as a function of n), what is the maximum size of n for which you can complete computation within an hour?

(a) $f(n) = n$

(b) $f(n) = n^3$

(c) $f(n) = 5.log_2(n) + 6$

(d) $f(n) = 2n.log_2(n)$

(e) $f(n) = 2^n$

Question 2 (5 points)

Determine whether the following complexity statements are true or false. Explain your reasoning in the context of the formal definition of O, Ω and Θ.

(a) $12n^3 - 7n + 6$ is $O(n^3)$

(b) $8n^2 + 4$ is $O(n^2)$

(c) $3n^4 - 2n^2 + 3$ is $\Omega(n^2)$

(d) $3n^4 - 2n^2 + 3$ is $O(n^2)$

(e) $18n + 19$ is $\Theta(n)$

Question 3 (10 points)

Where possible, apply the master theorem for the following. If not possible, state the reason why.

(a) $4T(\frac{n}{2}) + 16n^2$

(b) $8T(\frac{n}{2}) + 160n$

(c) $2^nT(\frac{n}{2}) + n$

(d) $9T(\frac{n}{3}) + 97n^3$

(e) $T(\frac{2n}{3}) + 1$

Question 4 (10 points)

For each of the following input lists, state what you believe to be the most effective sorting algorithm. Justify your reasoning, taking into account complexity, number of operations, memory usage and any other properties of sorted lists. If there are features or attributes of the algorithm that are implementation dependent, state these in your answer.

(a) $1, 2, 3, 4, 6, 5$

(b) $6, 2, 3, 4, 5, 1$

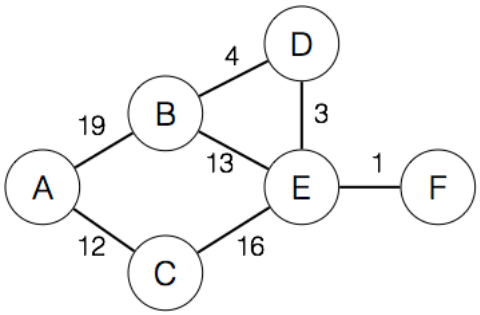
(c) $6, 5, 4, 3, 2, 1$

(d) $1, 5, 4, 6, 3, 2$

(e) ((9,3), (2,5), (2,4), (2,7), (1,4), (5,7)) where each tuple(key, value) is sorted by the key. The ordering of the values must be retained where the keys are equivalent.

Question 5 (5 points)

Apply Dijkstra's Algorithm to the following graph, computing the shortest path for all vertices from vertex A. Present the results after each vertex has been processed.



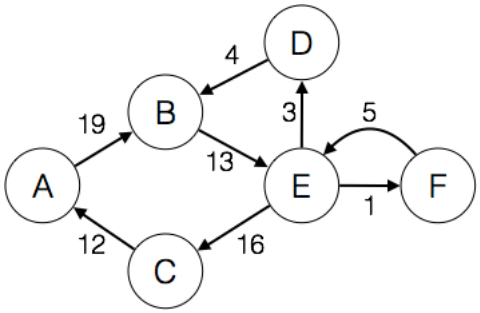
You may wish to present the results in the format of the following table:

Stage	Current Vertex	Labels and Distances											
		A		B		C		D		E		F	
0	-	A	0	-	∞	-	∞	-	∞	-	∞	-	∞
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
n	F	A	0	D	231	C	213	E	4	F	21	A	90

Each row states (a) the current stage, (b) the vertex just added to the search graph, and (c) the current updated predecessor node label and distance from A for each vertex in the graph. Any values given here in the table are merely for example, and do not correspond to the solution for this problem. The vertices should be processed in the order dictated by Dijkstra's Algorithm.

Question 6 (5 points)

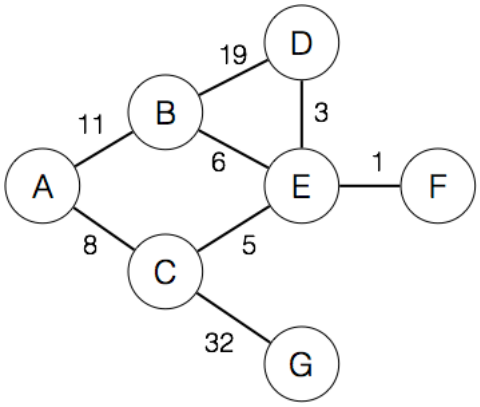
Apply Dijkstra's Algorithm as in Question 5, once again providing the results at each stage, to the following directed graph:



As before, compute the shortest path for all vertices from vertex A.

Question 7 (10 points)

For the following graph:



(a) Apply Prim's Algorithm, beginning at vertex A. Show the results of each stage of the algorithm. In this case, we define a stage as the addition of a new vertex and a new edge to the MST S ={V,E}.

You may wish to present the results in the format of the following table (values provided are only examples and do not correspond to the solution):

Stage	V	E
0	(A)	()
1	(A,B)	((A,B))
⋮	⋮	⋮
n	(A,B,C,D,E,F)	((A,B), (B,C), (C,D), (D,E), (E,F))

(b) Apply Kruskal's Algorithm, beginning at vertex A. Show the results of each stage of the algorithm. In this case, we define a stage as the processing of an edge from the graph, whether or not it is added to the MST S={V,E}.

You may wish to present the results in the format of the following table (values provided are only examples and do not correspond to the solution):

Stage	Edges	Components	E
0	((A,B), (B,C), (C,D), (D,E), (E,F))	((A),(B),(C),(D),(E),(F))	()
1	((B,C), (C,D) (D,E), (E,F))	((A,B),(C),(D),(E),(F))	((A,B))
⋮	⋮	⋮	
n		((A,B,C,D,E,F))	((A,B), (B,C), (C,D), (D,E), (E,F))

Note the division of the MST Vertices Set into Connected Components.

Question 8 (50 points)

Implement a Java class called Graph and store it in a file called Graph.java. In this class you should implement the basics required to store the data for an undirected, unweighted graph. In particular, you should include the following methods:

(a) public Graph():
A constructor method that takes no arguments - This will be responsible for setting up any internal data structures, but by default the Graph is empty and does not take any initial data.

(b) public void addVertex(String label):
A method that adds a vertex with a label as stored in the contents of the input argument *label*. If the vertex already exists, nothing is changed.

(c) public void removeVertex(String label):
A method that removes the vertex with the label matching the contents of the *label* variable. This should also remove any edges that connect to this vertex. If the vertex does not exist, nothing is changed.

(d) public void addEdge(String vertexA, String vertexB):
A method that adds an edge between the vertices with labels matching the contents of $vertexA$ and $vertexB$. If either vertex does not exist, no edge is added. Likewise, if the edge already exists, nothing is changed.

(e) public void removeEdge(String vertexA, String vertexB):
A method that removes the edge between the vertices with labels matching the contents of $vertexA$ and $vertexB$. If no such edge exists, then nothing is changed.

(f) public boolean depthFirstSearch(String startVertex, String searchVertex):
A method that performs a depth-first search across the graph. It returns true if it finds a vertex with the label matching the contents of $searchVertex$ when starting from vertex $startVertex$, else it returns false.

(g) public boolean breadthFirstSearch(String startVertex, String searchVertex):
A method that performs a breadth-first search across the graph. It returns true if it finds a vertex with the label matching the contents of $searchVertex$ when starting from vertex $startVertex$, else it returns false.

(h) public void printCutSet(String[] subVerticesA, String[] subVerticesB):
A method that takes two inputs, $componentA$ and $componentB$. $componentA$ is an array of labels for a first set of vertices, and $componentB$ is an array of labels for a second set of vertices - i.e. a sub-graph componentA and a sub-graph componentB of the current graph (assuming using edges of current graph). For this exercise you may assume the union of the two sets of values gives the full vertex set for the graph.

This method should print out the collection of edges that constitute the cut set for these two sub-graphs of the current graph. This is the set of edges which have a vertex u in componentA, and a vertex v in componentB, such that their removal splits a connected component of the graph into two. E.g. If there exist two vertices, 'A' and 'B', and an edge (A,B), then the cut set for componentA = 'A', componentB = 'B' is the edge (A,B), since its removal separates the two sub-graphs.

If an invalid input is provided, such as a vertex that does not exist in the graph, or a vertex is present in both componentA and componentB, then you may simply print out 'Error'. If a valid input is provided, then you should print out the list of edges in the format of tuples. E.g. ((A,B),(A,C),(B,C)).

This exercise will be assessed not only on functionality but also on efficiency. As part of this, you are free to use any of the data structures available to you from the Java library, such as java.util.collections (however, you are not permitted to use a Graph class). You are also free to implement your own additional data representations such as a node with further classes if you so choose, but you must include these additional classes within the Graph.java file for submission.

You should also provide comments that justifies the decisions you made when constructing this class.