1/5

CSC 226 fall 2015 Homework 2 – Theory Part

Instructions: Upload a single pdf file to conneX. Problems 3 and 4 contain tables for you to complete. Print out these tables, complete them, and scan them together with your answers to the other parts to create a single pdf file.

Do not copy answers from other sources. Your write-up must be in your own words. If you consult with others, you must provide their names.

Study the posted slides for lectures 4-6.

1 Single Source Shortest Paths.
Consider an undirected graph G = (V,E), with vertex set V = {S, A, B, C} and edge set E = {(S,A), (S,B), (A,C), (B,C)}. The weights on the edges are as follows:

wt(S,A) = 4
wt(S,B) = 2
wt(A,C) = 1
wt(B,C) = 3

a) The weights are all distinct, so we know the minimum spanning tree MST is unique. What is the (unique) MST for G, and what is its weight?

b) List all the Single Source Shortest Path Trees for G that have S as the source (start) vertex. We haven't proved such trees must be unique, so there may be more than one.

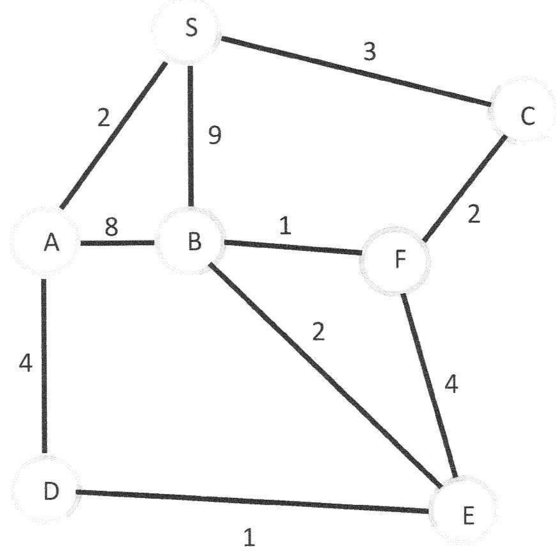
c) Give the weight of each Single Source Shortest Path Tree that you found in b).

d) Compare your answer to c) with your answer to b). Are the weights of the MST and the Single Source Shortest Path Tree(s) with source (start) vertex S equal? Must all Single Source Shortest Path Trees with start vertex S have the same weight? Explain.

e) Let's record the lowest cost D[V] for travelling from S to each vertex V in G. Then let's define the *total cost* of a Single Source Shortest Path Tree SPT with source S to be the sum of all these D[V]s. Is the total cost of an SPT the same for all such trees with source S? Explain.

2/5

2. Dijkstra's algorithm example.



Graph for Dijkstra's Algorithm
S is the start vertex. The edges have costs (weights).

There are tables numbered 0 to 7 on the next page. In each table, the first column lists the vertices in the graph. The 2nd column shows whether the vertex has been added to the Shortest Path Tree (SPT) T yet. The 3rd column gives the current estimated cost D[] from the start vertex S to the vertex. The 4th column gives the parent vertex P[] on the current best known path from S to the vertex; if no path from S to the vertex is known yet, then the entry for the vertex in the 4th column shows a "-".

For the start vertex S, the cost D[S] from S to itself is always 0, so the entry in the 3rd column for vertex S is 0 for all the tables. Also, vertex S doesn't have a parent vertex in going from S to itself, so the entry in the 4th column for vertex S is N/A (meaning "non-applicable") for all the tables.

Table 0 indicates that no vertices have been added to the tree so far (shown by X's in 2nd column). Initially, all vertices other than S have an infinity symbol in the 3rd column. The "infinity" for D[] is updated to a numerical value when some vertex W that has just been added to the T is being "relaxed", and that vertex W has an edge (W,V) that improves (decreases) D[V] to D[W] + wt(W,V) .

Which vertex is added to T next? That vertex V not in T for which the current estimated cost D[V] from S is smallest. Ties can be broken arbitrarily, but as a convention, when ties occur, *choose the vertex that is lexicographically first.* Note that V = S is the first vertex added to T, as initially, S is the only vertex with a non-infinity value for D[V].

When you add a vertex V to T, how do you "relax" V? Consider each of its neighbors W not in T, and update D[W] to D[V] + wt (V,W) if this lowers the cost to reach W from S.

Table 1 shows the results of relaxing S. Tables 2-7 are only partly complete. Your job is to complete them. Table 7 should give the lowest cost from S to each other vertex, and its 4th column should show how to backtrack from any vertex back to S along some lowest cost path. Note that the last vertex added to T has no edges to relax.

Table 0
(Initial table)

V	V in T?	est. cost D[V]	parent
S	X	0	N/A
A	X	∞	-
B	X	∞	-
C	X	∞	-
D	X	∞	-
E	X	∞	-
F	X	∞	-

Table 1
new vertex to relax: _S_

V	V in T?	est. cost D[V]	parent
S	✓	0	N/A
A	X	2	S
B	X	9	S
C	X	3	S
D	X	∞	-
E	X	∞	-
F	X	∞	-

Table 2
new vertex to relax _A_

V	V in T?	est. cost D[V]	parent
S	✓	0	N/A
A	✓	2	S
B	X		
C	X		
D	X		
E	X		
F	X		

Table 3
new vertex to relax: __

V	V in T?	est. cost D[V]	parent
S	✓	0	N/A
A	✓	2	S
B			
C			
D			
E			
F			

Table 4
new vertex to relax: __

V	V in T?	est. cost D[V]	parent
S	✓	0	N/A
A	✓	2	S
B			
C			
D			
E			
F			

Table 5
new vertex to relax: __

V	V in T?	est. cost D[V]	parent
S	✓	0	N/A
A	✓	2	S
B			
C			
D			
E			
F			

Table 6
new vertex to relax: __

V	V in T?	est. cost D[V]	parent
S	✓	0	N/A
A	✓	2	S
B			
C			
D			
E			
F			

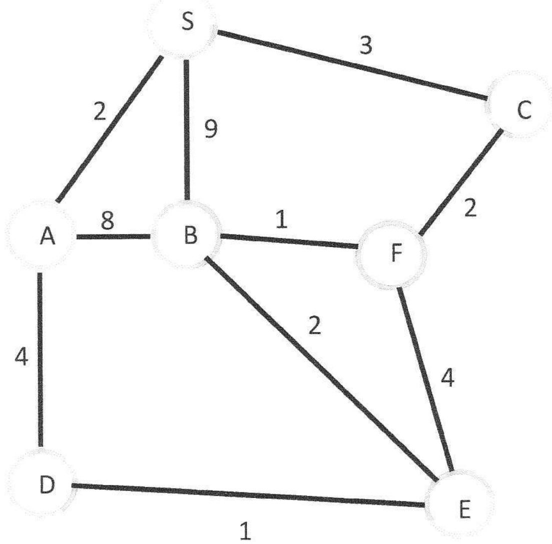
Table 7
last vertex added to T: __

V	V in T?	est. cost D[V]	parent
S	✓	0	N/A
A	✓	2	S
B	✓		
C	✓		
D	✓		
E	✓		
F	✓		

Draw the shortest path tree T:

4/5

3. Bellman-Ford algorithm example.



Graph for Bellman-Ford Algorithm (same as for Dijkstra's)

S is the start vertex. The edges have costs (weights) and are 2-way. There are 7 vertices and 10 edges.

The edge list **E** is as follows:

(A,B) (A,D) (A,S) (B,E) (B,F) (B,S) (C,F) (C,S) (D,E) (E,F)

The Bellman-Ford algorithm makes |**V**| = 7-1 = 6 passes through the edge list **E**. Each pass relaxes the edges in the order they appear on the list. As with Dijkstra's algorithm, we record the current best known cost D[V] to reach each vertex V from the start vertex S, and we also record the parent P[V] of each vertex on the current lowest cost path from S. Initially, D[S] = 0 and P[S] = N/A; also, initially D[V] = "infinity" and P[V] = "-" for all the other vertices. There are 6 tables shown, one for each pass through **E**. The first table has been completed. Your first job to complete the tables for passes 2-6. As some table entries may change during a pass, start by copying each entry from the previous table, then show the changes within each table entry. Table 2 is partly done, to illustrate the sequence of changes.

5/5

E = (A,B) (A,D) (A,S) (B,E) (B,F) (B,S) (C,F) (C,S) (D,E) (E,F)

Pass 1		
V	est. cost D[V]	parent
S	0	N/A
A	2	S
B	9	S
C	3	S
D	∞	-
E	∞	-
F	∞	-

Pass 2		
V	est. cost D[V]	parent
S	0	N/A
A	2	S
B	9	S
C	3	S
D	∞, 6	-, A
E	∞, 11, 7	-, B, D
F	∞, ?	-, ?

Pass 3		
V	est. cost D[V]	parent
S	0	N/A
A		
B		
C		
D		
E		
F		

Pass 4		
V	est. cost D[V]	parent
S	0	N/A
A		
B		
C		
D		
E		
F		

Pass 5		
V	est. cost D[V]	parent
S	0	N/A
A		
B		
C		
D		
E		
F		

Pass 6		
V	est. cost D[V]	parent
S	0	N/A
A		
B		
C		
D		
E		
F		

a) Complete the tables above.

b) Compare the final values for D[V] with those from Dijkstra's algorithm. In general (not just for this example), must they be the same? Explain.

c) In general (not just for this example), in the final tables for Dijkstra and Bellman-Ford, must the parent entries be the same? Explain.

d) When executing the Bellman-Ford algorithm, suppose that some pass through the edge list E does not cause any changes to the estimated costs D[V]. Can the algorithm terminate without doing the remaining passes? Explain.