HW #1 SOLUTION

COMS W4119 – Computer Networks
Spring 2006

PROBLEM 1

To tackle this question we first need to establish the following fact:

For a graph G that contains N nodes, an MST must contain exactly N-1 edges
Proof:
Every edge contains two different nodes. Therefore N-1 edges are needed to connect N nodes together. Any additional edge added to the spanning tree after that will increase the total cost of the tree without helping the cause of connecting nodes. Since an MST is defined such that you cannot remove edges from it without ending up with unreachable nodes, one can deduce that an MST must contain N-1 nodes exactly.

Now we proceed to the question at hand:
(a)
T is given as an MST using weight function w, which implies two things:

- T contains N-1 edges.
- Summing w for all (N-1) edges in T gives a minimal weight, i.e. $\sum_{edges}^{T} w(e)$ is minimized.

But the question is, is $\sum_{edges}^{T} v(e)$ also minimized?

To answer this, consider any rival spanning tree R that is potentially an MST using weight function v. Such a tree must also contain N-1 nodes, and must have a total weight less than that of T using function v.

In other words, it must hold that $\sum_{edges}^{R} v(e) < \sum_{edges}^{T} v(e)$

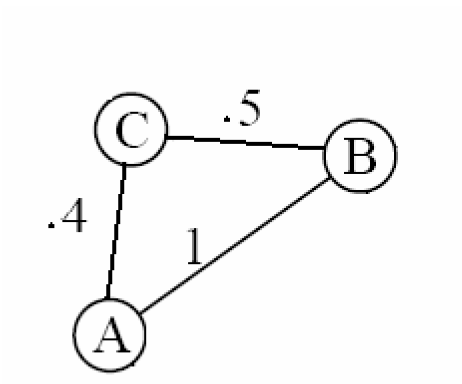
$$\Rightarrow \sum_{edges}^{R} w(e) + 1 < \sum_{edges}^{T} w(e) + 1$$

$$\Rightarrow N - 1 + \sum_{edges}^{R} w(e) < N - 1 + \sum_{edges}^{T} w(e)$$

$$\Rightarrow \sum_{edges}^{R} w(e) < \sum_{edges}^{T} w(e)$$

But this contradicts the initial assumption that T is an MST using weight function w.
Therefore that cannot exist such a tree R.
Therefore T is an MST using weight function v.

(b)
This statement can be proven wrong by considering the following counter-example. The weights shown below are w(e):



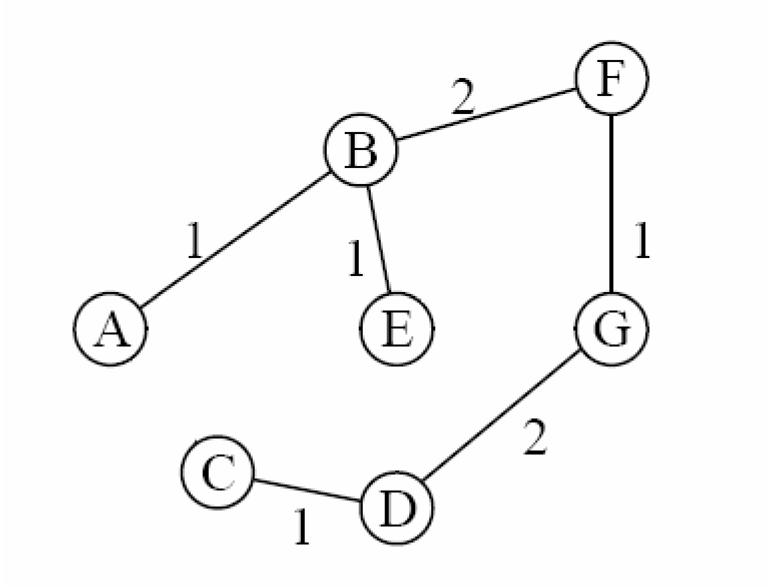
The shortest path from A to B using weight function w(e) is A-C-B at cost 0.9. However using weight function v(e)=w(e)+1 the total cost of this path becomes 2.9, which is higher than the cost of 2 for the direct path A-B.

PROBLEM 2

Using the Kruskal algorithm, we keep adding links from least cost to most cost, rejecting links that result in loops, until N-1 links have been added. We have 7 nodes so we need 6 links. We choose them as follows:

LINK	COST	DECISION
AB	1	Added: no loops
BE	1	Added: no loops
CD	1	Added: no loops
FG	1	Added: no loops
BF	2	Added: no loops
DG	2	Added: no loops

The resulting MST is as follows:



PROBLEM 3

(a)
We start with a tree containing only node A at cost 0. We repeatedly add the edge that connects a node in the tree to a node not yet in the tree, such that the total cost from A to the new node is minimized:

Starting table:

Node	Cost from A to node
A	0
B	Not in tree
C	Not in tree
D	Not in tree
E	Not in tree
F	Not in tree
G	Not in tree

Next we add edge AB, the table now reads:

Node	Cost from A to node
A	0
B	1
C	Not in tree
D	Not in tree
E	Not in tree
F	Not in tree
G	Not in tree

Next edge BE:

Node	Cost from A to node
A	0
B	1
C	Not in tree
D	Not in tree
E	2
F	Not in tree
G	Not in tree

Next edge BF:

Node	Cost from A to node
A	0
B	1
C	Not in tree
D	Not in tree
E	2
F	3
G	Not in tree

Then edge FG:

Node	Cost from A to node
A	0
B	1
C	Not in tree
D	Not in tree
E	2
F	3
G	4

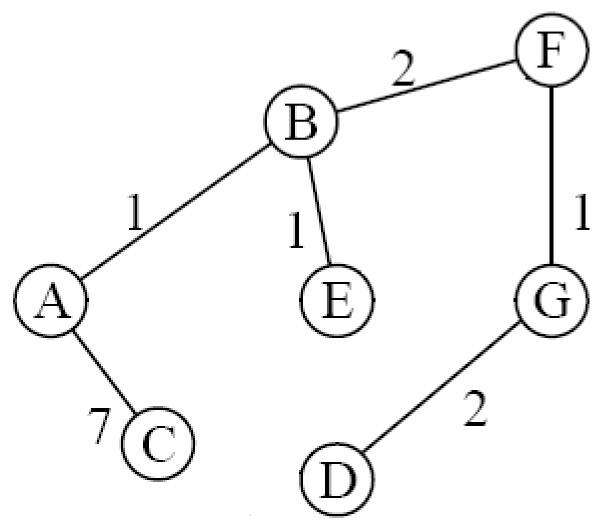
Then GD:

Node	Cost from A to node
A	0
B	1
C	Not in tree
D	6
E	2
F	3
G	4

And finally AC:

Node	Cost from A to node
A	0
B	1
C	7
D	6
E	2
F	3
G	4

The resulting shortest path tree looks like so:



(b)
We start with a table with cost 0 to node A and infinite cost to every other node:

Node	Cost from A to node	Predecessor
A	0	Null
B	∞	Null
C	∞	Null
D	∞	Null
E	∞	Null
F	∞	Null
G	∞	Null

Next for every node X, we iterate over all branching edges.
If for edge XY, node Y's cost > node X's cost + edge XY's weight, we update node Y as follows:

Y's cost become X's cost + edge XY's weight
Y's predecessor becomes X

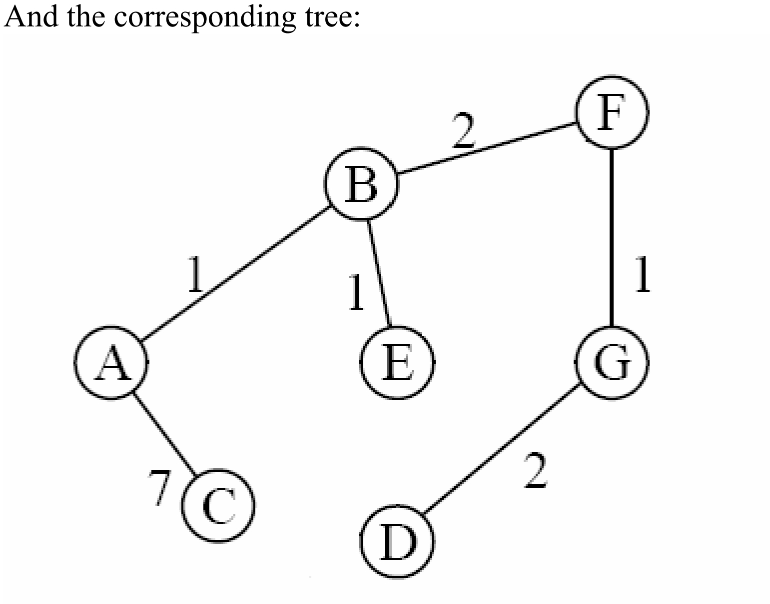
For example, after examining all edges connected to node A, the table becomes:

Node	Cost from A to node	**Predecessor**
A	0	Null
B	1	A
C	7	A
D	∞	Null
E	5	A
F	∞	Null
G	∞	Null

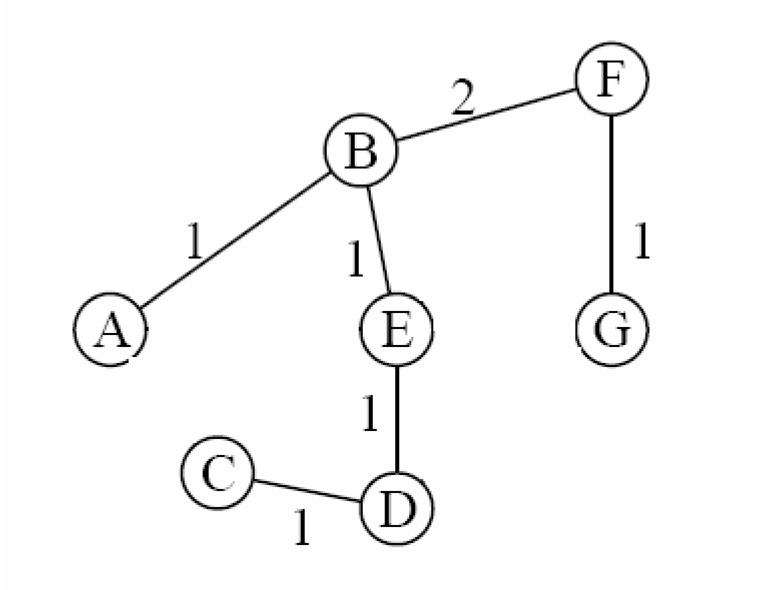
We proceed to apply the same logic to all nodes on by one. After one pass through the nodes, the table converges into the following form:

Node	Cost from A to node	**Predecessor**
A	0	Null
B	1	A
C	7	A
D	6	G
E	2	B
F	3	B
G	4	F

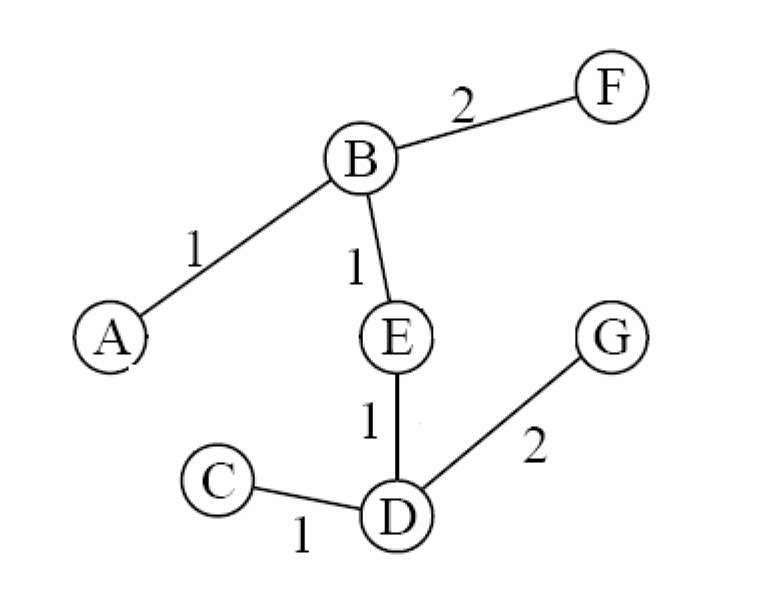
And the corresponding tree:



(c)
Given the new network reality, we continue to iterate through nodes as before, until the tables converges:



(d)
Again, we iterate through all nodes until the table converges into the new form:



PROBLEM 4

(a) Every node maintains a state table as shown below. The columns of the table are the directly neighboring nodes. The rows are all nodes in the network (except the self node). The value of the cell at column X row Y in table Z is the cost to reach Y from Z going through X. The nodes exchange vectors on round by round basis. The rounds alternate in the following order A→B→D→C→A→etc. Other orders will work just as well. The information exchanged during a round is a "vector" containing the minimum cost of every row (vectors not shown below). Upon reception of vectors, recipient nodes update their status tables accordingly. Updated cells are shown below in red. This process carries on until the tables stabilize (no more updates) as shown below:

A

	B	C
A	--	--
B	1	∞
C	∞	100
D	∞	∞

B

	A	D
A	1	∞
B	--	--
C	∞	∞
D	∞	1

C

	A	D
A	100	∞
B	∞	∞
C	--	--
D	∞	1

D

	B	C
A	∞	∞
B	1	∞
C	∞	1
D	--	--

	B	C
A	--	--
B	1	∞
C	∞	100
D	∞	∞

	A	D
A	1	∞
B	--	--
C	101	∞
D	∞	1

	A	D
A	100	∞
B	101	∞
C	--	--
D	∞	1

	B	C
A	∞	∞
B	1	∞
C	∞	1
D	--	--

	B	C
A	--	--
B	1	∞
C	102	100
D	2	∞

	A	D
A	1	∞
B	--	--
C	101	∞
D	∞	1

	A	D
A	100	∞
B	101	∞
C	--	--
D	∞	1

	B	C
A	2	∞
B	1	∞
C	102	1
D	--	--

	B	C
A	--	--
B	1	∞
C	102	100
D	2	∞

	A	D
A	1	3
B	--	--
C	101	2
D	∞	1

	A	D
A	100	3
B	101	2
C	--	--
D	∞	1

	B	C
A	2	∞
B	1	∞
C	102	1
D	--	--

	B	C
A	--	--
B	1	102
C	102	100
D	2	101

	A	D
A	1	3
B	--	--
C	101	2
D	∞	1

	A	D
A	100	3
B	101	2
C	--	--
D	∞	1

	B	C
A	2	4
B	1	3
C	102	1
D	--	--

	B	C
A	--	--
B	1	102
C	102	100
D	2	101

	A	D
A	1	3
B	--	--
C	101	2
D	3	1

	A	D
A	100	3
B	101	2
C	--	--
D	102	1

	B	C
A	2	4
B	1	3
C	102	1
D	--	--

	B	C
A	--	--
B	1	102
C	3	100
D	2	101

	A	D
A	1	3
B	--	--
C	101	2
D	3	1

	A	D
A	100	3
B	101	2
C	--	--
D	102	1

	B	C
A	2	4
B	1	3
C	3	1
D	--	--

	B	C
A	--	--
B	1	102
C	3	100
D	2	101

	A	D
A	1	3
B	--	--
C	101	2
D	3	1

	A	D
A	100	3
B	101	2
C	--	--
D	102	1

	B	C
A	2	4
B	1	3
C	3	1
D	--	--

	B	C
A	--	--
B	1	102
C	3	100
D	2	101

	A	D
A	1	3
B	--	--
C	101	2
D	3	1

	A	D
A	100	3
B	101	2
C	--	--
D	102	1

	B	C
A	2	4
B	1	3
C	3	1
D	--	--

	B	C
A	--	--
B	1	102
C	3	100
D	2	101

	A	D
A	1	3
B	--	--
C	4	2
D	3	1

	A	D
A	100	3
B	101	2
C	--	--
D	102	1

	B	C
A	2	4
B	1	3
C	3	1
D	--	--

(b)

We need only worry about the first row of the tables in the discussion below. The change of link cost is immediately reflected on the state table of neighboring node B. The change is later communicated to other nodes in the network. Note that A's contribution to the communication is irrelevant since A cannot help other nodes find the shortest path to A. The first few rounds of communication are shown below:

	A: B	A: C	B: A	B: D	C: A	C: D	D: B	D: C
A	--	--	200	3	100	3	2	4

	A: B	A: C	B: A	B: D	C: A	C: D	D: B	D: C
A	--	--	200	3	100	3	4	4

	A: B	A: C	B: A	B: D	C: A	C: D	D: B	D: C
A	--	--	200	4	100	4	4	4

	A: B	A: C	B: A	B: D	C: A	C: D	D: B	D: C
A	--	--	200	4	100	4	4	5

	A: B	A: C	B: A	B: D	C: A	C: D	D: B	D: C
A	--	--	200	5	100	4	5	5

	A: B	A: C	B: A	B: D	C: A	C: D	D: B	D: C
A	--	--	200	6	100	6	5	5

Notice that the tables have not converged yet. In fact, the way the entries are climbing upward, it will be hundreds more turns before the tables converge. This is too impractical!

(c)

We start in a state identical to part (a) above. The nodes transmit vectors in a similar way, except that reverse paths are poisoned, meaning that when node X is to transmit a vector to node Y, en entry in the vector that goes through node Y should be treated as infinitely costly. This way reverse paths are not allowed in shortest path trees, and the end result looks like so:

A

	B	C
A	--	--
B	1	102
C	3	100
D	2	101

B

	A	D
A	1	102
B	--	--
C	101	2
D	102	1

C

	A	D
A	100	3
B	101	2
C	--	--
D	102	1

D

	B	C
A	2	101
B	1	102
C	102	1
D	--	--

(d)

We proceed in the same fashion as in part (b) above, except using poison reverse:

A | **B** | **C** | **D**

	B	C
A	--	--

	A	D
A	200	102

	A	D
A	100	3

	B	C
A	2	101

	B	C
A	--	--

	A	D
A	200	102

	A	D
A	100	3

	B	C
A	201	101

	B	C
A	--	--

	A	D
A	200	102

	A	D
A	100	202

	B	C
A	201	101

Notice how the tables stabilized in two rounds only. Contrast that to the result in part (b).

(e)

A

	B	C
A	--	--
B	1	102
C	3	100
D	2	101

B

	A	C	D
A	1	6	6
B	--	--	--
C	101	3	2
D	102	4	1

C

	A	B	D
A	100	4	3
B	101	3	2
C	--	--	--
D	102	4	1

D

	B	C
A	2	5
B	1	4
C	4	1
D	--	--

(f)

A

	B	C
A	--	--

B

	A	C	D
A	200	6	6

C

	A	B	D
A	100	4	3

D

	B	C
A	2	5

	B	C
A	--	--

	A	C	D
A	200	6	6

	A	B	D
A	100	9	3

	B	C
A	7	5

	B	C
A	--	--

	A	C	D
A	200	6	6

	A	B	D
A	100	9	3

	B	C
A	7	10

	B	C
A	--	--

	A	C	D
A	200	6	13

	A	B	D
A	100	9	8

	B	C
A	7	10

	B	C
A	--	--

	A	C	D
A	200	6	13

	A	B	D
A	100	16	8

	B	C
A	7	10

The count to infinity problem is still there. The reason is that poison reverse does not eliminate all loops in shortest path trees. It only eliminates paths that contain reverse paths, so loops that consist of three or more different edges can still slip through.

(f)

While the count to infinity problem was not prevented, poison reverse did help a little bit: Since reverse paths are not allowed, the counts now only “hop” by larger amounts, proportional to the weights of loops. So we end up with a lighter “hop to infinity” problem which is not as bad.