MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
Department of Aeronautics and Astronautics

6.263/16.37

Problem Set No.9 Solutions

Posted: Nov 13, 2008

Problem 1: text problem 5.1
The Prim-Dijkstra Algorithm: Arbitrarily select node e as the initial fragment. Arcs are added in the following order: (d, e), (b, d), (b, c) {tie with (a, b) is broken arbitrarily}, (a, b), (a, f).

Kruskal's Algorithm: Start with each node as a fragment. Arcs are added in the following order: (a, f), (b, d), (a, b) {tie with (b, c) is broken arbitrarily}, (b, c), (d, e).

The weight of the MST in both cases is 15.

Problem 2: text problem 5.2
The Bellman-Ford Algorithm: By convention, $D_1^{(h)} = 0$, for all h. Initially, $D_1^{(i)} = d_{1i}$, for all $i \neq 1$. For each successive $h \geq 1$, we compute $D_i^{(h+1)} = \min_j[D_j^{(h)} + d_{ji}]$, for all $i \neq 1$. The results are summarized in the following table.

i	D_i^1	D_i^2	D_i^3	D_i^4	D_i^5	Shortest path arcs
1	0	0	0	0	0	
2	4	4	4	4	4	(1,2)
3	5	5	5	5	5	(1,3)
4	∞	7	7	7	7	(2,4)
5	∞	14	13	12	12	(6,5)
6	∞	14	10	10	10	(4,6)
7	∞	∞	16	12	12	(6,7)

The arcs on the shortest path tree are computed after running the Bellman-Ford algorithm. For each $i \neq 1$, we include in the shortest path tree one arc (j, i) that minimizes Bellman's equation.

Dijkstra's Algorithm: Refer to the algorithm description in the text. Initially: $D_1 = 0$; $D_i = d_{1i}$ for $i \neq 1$; $P = \{1\}$. The state after each iteration is shown in the table below. P is not shown but can be inferred from i. Only the D_j's which are updated at each step are shown.

Iteration	i	D_1	D_2	D_3	D_4	D_5	D_6	D_7	Arc added
initial		0	4	5	∞	∞	∞	∞	
1	2			5	7	14	∞	∞	(1,2)
2	3				7	14	14	∞	(1,3)
3	4					13	10	∞	(2,4)
4	6					12		12	(4,6)
5	5							12	(6,5)
6	7								(6,7)

Problem 3: text problem 5.3

Let p_{ij} be the probability that link (i, j) fails during the lifetime of a virtual circuit. Let P_k be the probability that a path $k = (A, i, ..., j, B)$ remains intact. Since links fail independently we have:

$$P_k = (1 - p_{Ai}) \cdots (1 - p_{jB}).$$

We want to find the path k for which P_k is maximized. Equivalently, we can find the path k for which $-\ln P_k$ is minimized.

$$-\ln P_k = -\ln(1 - p_{Ai}) - \cdots - \ln(1 - p_{jB}).$$

Since the arc weights p_{ij} are small, $1 - p_{ij}$ is close to 1 and we may use the approximation $\ln z \approx z - 1$. This gives:

$$-\ln P_k \approx p_{Ai} + \cdots + p_{jB}$$

Therefore, the most reliable path from A to B is the shortest path using the weights given in the figure. Applying Dijkstra's algorithm gives the shortest path tree. We proceed as in problem 5.2. The most reliable path from A to B is (A, C, E, D, G, B). The probability

Iteration	i	D_A	D_B	D_C	D_D	D_E	D_F	D_G	Arc added
initial		0	∞	0.01	∞	0.03	∞	∞	
1	C		∞		0.06	0.02	∞	∞	(A,C)
2	E		∞		0.04		0.06	∞	(C,E)
3	D		0.1				0.05	0.06	(E,D)
4	F		0.1					0.06	(D,F)
5	G		0.09						(D,G)
6	B								(G,B)

that this path remains intact is

$$P_{ACEDGB} = (0.99)(0.99)(0.98)(0.98)(0.97) = 0.913.$$

Problem 4: text problem 5.4
Let the weights for arcs AB, BC, and CA be 1, 2, and 2, respectively. Then an MST is $\{AB, BC\}$ whereas the shortest path tree rooted at C is $\{CA, CB\}$.

Problem 5: text problem 5.17
a) In the algorithm that follows, each node has two possible states, "connected" or "not connected". In addition, each node marks each of its neighbors with one of the following: "unknown", "not in tree", "incoming", or "outgoing". There are two kinds of messages used: "attach" and "ack". The following are the procedures executed at each node i.

```
Initially at each node $i$
state = "not connected"
mark($j$)="unknown" for each neighbor $j$

Node 1 only
state = "connected"
send "attach" to each neighbor

Receive "attach" from $j$
if state = "not connected"
    then state = "connected"
        mark($j$) = "outgoing"
        if node $i$ has neighbors other than $j$
            then send "attach" to each neighbor except $j$
        else send "ack" to $j$
        end
else mark($j$) = "not in tree"
    if mark($k$) $\neq$ "unknown" for each neighbor $k$
        then send "ack" to the neighbor $k$ such that mark($k$) = "outgoing"*
end

Receive "ack" from $j$
mark($j$)="incoming"
if mark($k$) $\neq$ "unknown" for each neighbor $k$
    then send "ack" to the neighbor $k$ such that mark($k$) = "outgoing"*
end
```

*Node 1 just terminates the algorithm; it has no "outgoing" neighbor.

The above algorithm sends one "attach" and one "ack" message on each spanning tree link, and sends two "attach" messages (one in each direction) on each link not in the tree. Therefore, it sends a total of $2A$ messages.

b) We use the spanning tree constructed in part a) to simplify counting the nodes. Each node marks its "incoming" neighbors with either "heard from" or "not heard from". There are two messages used: "count nodes", and a message containing a single number $j : 0 < j < N$. The following is the procedure for each node i.

```
Initialization
mark($j$) = "not heard from" for all "incoming" neighbors
children = 0

Start (node 1 only)
send "count nodes" to all "incoming" neighbors

Receive "count nodes" from "outgoing" neighbor $j$
if there are any "incoming" neighbors
    then send "count nodes" on all incoming links
else send "1" to $j$
end

Receive $n$ from "incoming" neighbor $j$
children = children + $n$
mark($j$) = "heard from"
if mark($k$) = "heard from" for all "incoming" neighbors $k$
    then send (children + 1) to the "outgoing" neighbor*
end
```

*Node 1 has no outgoing neighbor. When it reaches this step, N = children + 1.

c) The worst case for both algorithms is a linear network. Messages must propagate from node 1 to the end of the tree and then back again. This gives an upper bound of $2(N-1)T$ for both algorithms.