George Verghese
verghese@mit.edu

6.02
11/16/2010

Rec 17 (and part of 18)

Little's Law for steady-state or equilibrium operation of a queueing-type system:

(longest run of vowels in any English word, I'm told!)

$$N = \lambda D$$

- N: avg. # of items in system at any time
- λ: avg. # items arriving or processed per unit time
- D: avg. delay or waiting or processing time

Dimensionally: $\#\text{items} = \frac{\#\text{items}}{\text{time}} \times \text{time}$

e.g. At rush hour on a freeway, 5 toll booths manage to process 4000 cars/hr, with an average queue of 20 cars at each toll booth. What is the average processing time?

$$D = \frac{N}{\lambda} = \frac{20 \text{ cars}}{\frac{4000}{5} \text{ cars/hr}} = \frac{1}{40} \text{ hr} = \underline{1.5 \text{ min}}$$

e.g. (Prob 6d of Ch 17)
Your 6.02 prof gets 200 non-spam emails/day and 50 of these on average need a reply. The avg. # of emails still needing a reply remains at 100, a steady state

(i) What is the prof's avg. time to respond to email that needs a response? →

Solution to (i): $D = \frac{N}{\lambda} = \frac{100}{50/\text{day}} = 2 \text{ days}$

(ii) 6.02-related mail turns out to be 25% of his email needing a reply, and he responds to this 6.02 email in 60 minutes, on average. How much time, on average, does it take him to respond to the rest of the mail he responds to?

Solution: $(0.25 \times 1 \text{ hour}) + (0.75 \times r \text{ hours}) = 48 \text{ hours} = 2 \text{ days}$, from (i)

(r: avg. response rate for rest of mail)

$\Rightarrow r = \frac{47.75}{0.75} = 63.67$ hours

e.g. Prob 4 of Ch 17

4. Alyssa P. Hacker has set up eight-node shared medium network running the Carrier Sense Multiple Access (CSMA) MAC protocol. The maximum data rate of the network is 10 Megabits/s. Including retries, each node sends traffic according to some unknown random process at an average rate of 1 Megabit/s per node. Alyssa measures the network's utilization and finds that it is 0.75. No packets get dropped in the network except due to collisions, and each node's average queue size is 5 packets. Each packet is 10000 bits long.

higher than ALOHA, achievable with CSMA.

(a) What fraction of packets sent by the nodes (including retries) experience a collision? Offered load = 8 Mb/s (including retries), throughput = .75 × 10 Mb/s = 7.5 Mb/s data

So fraction = $1 - \frac{7.5}{8} = \frac{1}{16}$

(b) What is the average queueing delay, in milliseconds, experienced by a packet before it is sent over the medium?

Little's law, with service rate $\lambda = 1$ Mb/s, i.e. $\frac{10^6}{10^4} = 100$ packet/sec.

Queue size $N = 5$ packets.

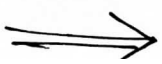
So service time $D = \frac{5}{100}$ sec $= 50$ ms.

Just for fun, an 'economical' proof of Little's Law:

Suppose we charge the items in the system (e.g. guests at a (very cheap) hotel) $1 per unit time. How much do we collect per unit time, on average, in steady state?
→ $N.

An equivalent way to collect this money is to charge each item on entry to the system. How much? $d if the stay is d days, so $D on average. So how much, on average, do we collect per unit time this way?
→ λD.

So $\boxed{N = \lambda D}$, as before.



Routing A quick recap of class-note highlights:

Distance vector routing: Each node comes up with an estimate of the minimum cost (of sending a packet) from it to every other node in the network, then shares this information with its direct neighbors (at the 'advertisement' stage). It also collects the corresponding information from each of its direct neighbors (whenever this information happens to arrive — recall that each node operates independently, they're not on a common clock). Each node is also assumed to have (or to dynamically obtain) the costs on the links that connect it to its direct neighbors. Now using this information, along with the distance vector of each of its direct neighbors, it does a Bellman-Ford calculation (Eq. 18.1 of the notes) to determine on which link it will send a packet destined for a particular node. It can also use this computation to update its own distance vector in preparation for the next 'advertisement' stage.

(i.e., its distance vector)

(This is the 'integration' stage)

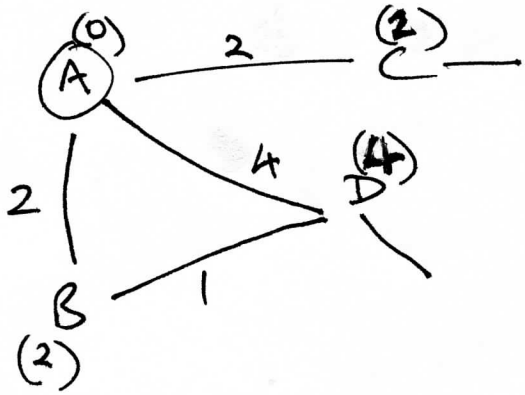
Under distance vector routing, a node does not attempt to assemble a representation of the whole network. In link-state routing, by contrast, each node tries to assemble a picture of the whole network, based on receiving link-state information from throughout the network (a result of the 'flooding' operation). It then runs its own Dijkstra algorithm to figure out the optimum path to each destination, so it can send a packet out on the best link towards that destination.

Note that these routing calculations happen on a much slower time-scale than that of packet forwarding.

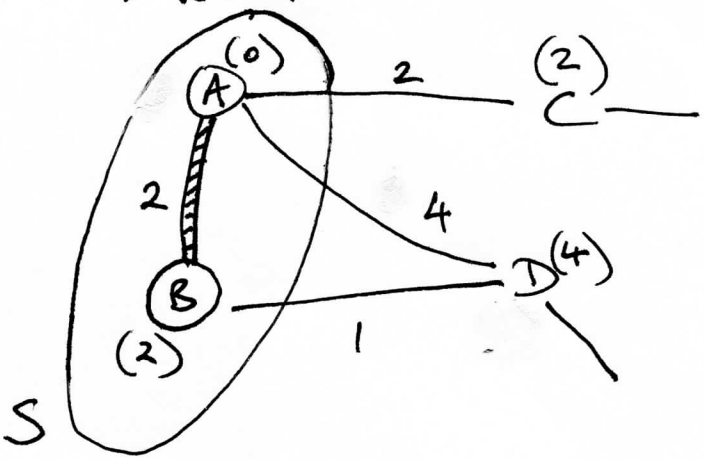
Dijkstra's Algorithm for determining a minimum-cost path from a starting node (A) to every other node, when link costs are nonnegative and given.

Step 1. Label (A) with (0), to signify that the minimum cost for going from (A) to (A) is 0.

Step 2. Label each of (A)'s direct neighbors with the cost of the link to that neighbor, written as (cost).



Step 3. Pick a direct neighbor with minimal value of that label, e.g., B in the example ← here, circle it, and shade the connecting link

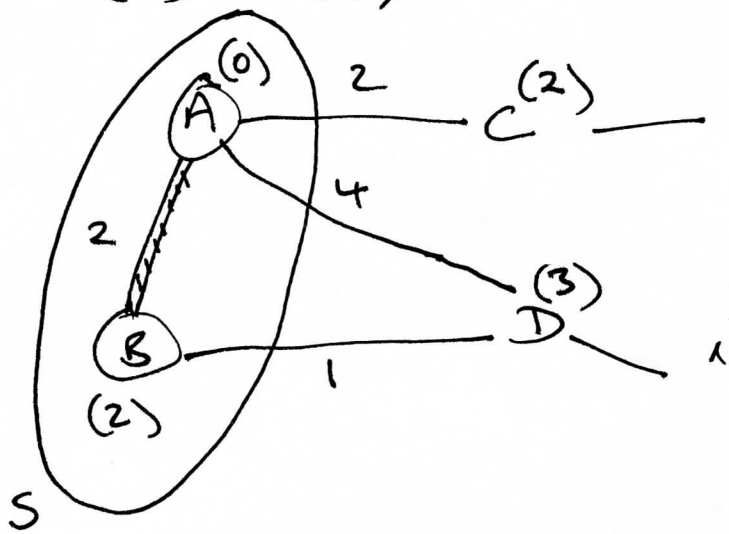


At this stage of the algorithm, the nodes (A), (B) and the edge (link) connecting them form the settled subset S.

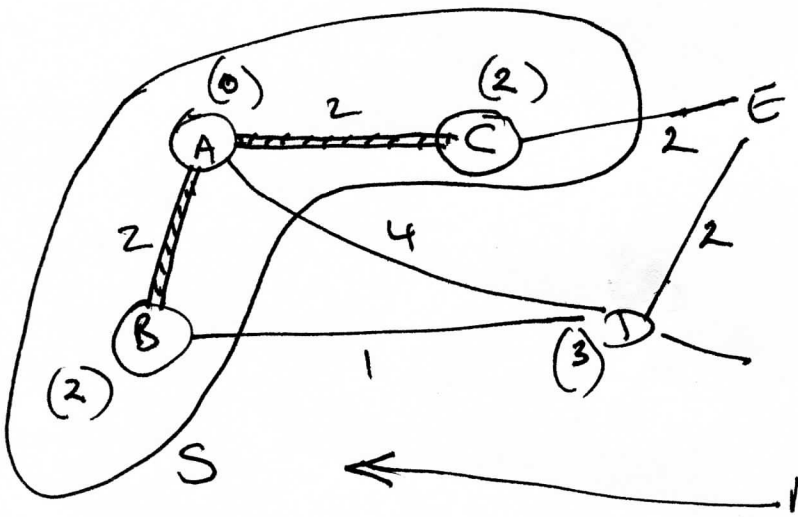
The shaded link gives a minimum-cost path from (A) to (B). Why? Because any other path would require leaving (A) on another link, thereby immediately incurring a cost at least equal to that on the shaded link.

Step 4. Update the labels on all nodes outside S that are directly connected to the newly settled node (B), to reflect the minimum cost of getting to them via (B) rather than directly from (A), if the former is lower.

In our example, the label on D is changed from (4) to (3), because $2+1 < 0+4$:



Step 5. Pick a direct neighbor of the set S (i.e., one of the neighbors outside S of a node in S) that has minimal value of the label among all neighbors, circle it, and shade the link connecting it to S, as shown — in our example, the neighbor in question is C.

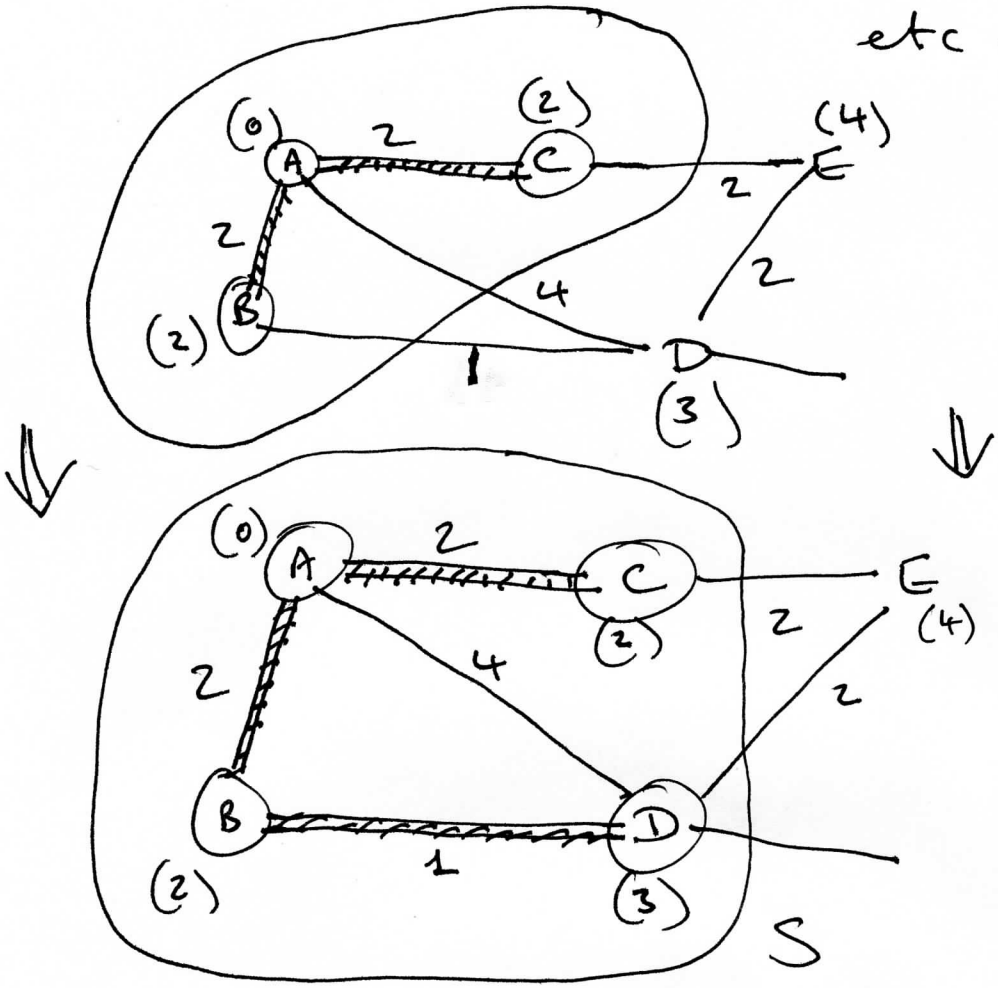


Now the settled set S is expanded to contain the nodes (A), (B), (C) and the edges connecting them. The newly shaded link is on a minimal-cost path from (A) to (C). Why? Because any other path would require leaving S on some other link, thereby immediately

incurring a cost at least equal to that on the newly shaded link.

Step 6. Same as Step 4, but using the newly settled node (C) instead of (B).

etc.



etc.

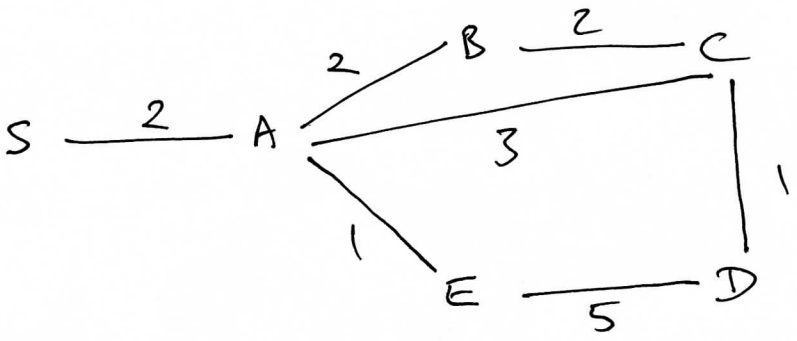
etc.

Summary:

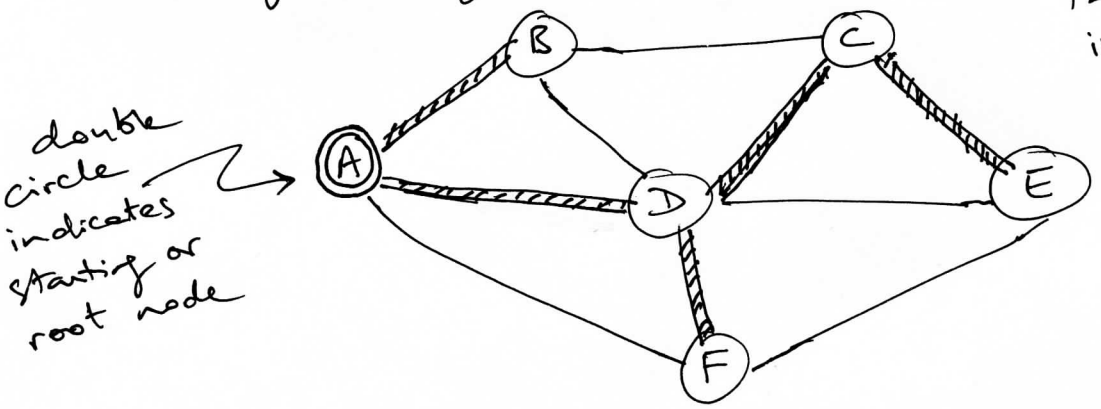
Determine what the minimum cost would be to go from (A) to each direct neighbor of the settled set (it suffices to update minimum costs for neighbors of the most recently settled node), and what associated link you would use. Choose whichever of these neighboring nodes has smallest minimum cost from (A) (if more than one candidate, pick arbitrarily), circle it and absorb it into the settled set; also highlight and absorb into S the associated link. The highlighted links will

form a tree in S, connecting all the nodes in S, but with no loops. (If S has n nodes, the tree will have n−1 links.)

For practice, Prob 1 of Chapter 18 notes.



Also Prob 5, but replace the two figures in Fig. 18-8 by the following single figure (which shows the equivalent information).



And for parts (c), (d), assume the minimum-cost tree rooted at C is unique, and redraw Fig 18-9 so the correspondence with the above network is more apparent: