Math 251/Buad427 **ACTIVITY 5:** Shortest Path networks and Dijkstra's algorithm

Why

Networks are often used to model transportation networks — highway networks, railroads, pipeline networks – and one of the frequent questions involves finding the shortest route through the network between two given points. Since "distance" can also be used to represent "cost" or "time", the shortest path problem extends to many other optimizations situations. Dijkstra's algorithm provides an efficient search method and provides an example of "fanning-out" search methods that is used in constructing other models and solution techniques.

LEARNING OBJECTIVES

1. Work as a team, using the team roles
2. Learn how to set up and interpret shortest-path models.
3. Learn a fanning-out search method (Dijkstra's algorithm) for searching a network.
4. Be able to find a shortest path through a network by either Dijkstra's algorithm or the spreadsheet model.
5. Be able to implement a search algorithm from its description.

CITERIA

1. Success in working as a team and in fulfilling the team roles.
2. Understanding of the material by all team members
3. Success in completing the exercises.

RESOURCES

1. Your class notes on transportation models
2. Your text – section 4.5 and the CD Supplement on shortest path (relevant pages attached)
3. Microsoft Excel, on the college network
4. 50 minutes

PLAN

1. Select roles, if you have not already done so, and decide how you will carry out steps 2 and 3
2. Work through the exercises given below you will submit one (team) copy of the work, with the usual reports [see the syllabus]
3. Assess the team's work and roles performances and prepare the Reflector's and Recorder's reports including team grade.
4. Be prepared to discuss your results

DISCUSSION In a network in which the nodes represent locations and the arcs represent possible travel (there is an arc from u to v if and only if we can travel directly – without going through any other nodes – from u to v) it is often desirable to find the shortest route through the network from a given node a (the "origin") to a given node v (the "destination"). Sometimes all arcs are considered to have the same length (which is generally considered to be "1") and sometimes there are values on the arcs representing the length (or the cost to travel, or the time required to travel through the arc). [Mapquest generates a lot of internet traffic – and a lot of advertising revenue – giving solutions to this problem for the highway grid of the US – their approach is a bit more complex but rooted in the problem here]
Dijkstra's algorithm solves the problem by building up a *collection* of shortest paths, reaching further and further from the origin (giving the shortest path and the distance to "completed" in order of distance]
The text [the CD supplement] gives a description of the algorithm. Here's another (very similar): Each node will be assigned a two-part label $d(n)$ - d will be the distance to our node from the origin along some path (sequence of arcs) and n is the node *from* which we arrive.

At the start, each node is assigned a "temporary" label - showing " the best route we know so far" - we try to improve these (get shorter distances) and progressively make labels "permanent" – showing an absolute best distance and arrival node– as we work out from the origin. The process:

1. Initial step [not repeated]: The origin is labeled $0(-)$ (distance 0, arrive from nowhere – it's the origin) and this label is permanent. All others are labeled $\infty(-)$ (but we don't usually write these down) and these are all temporary.
2. When a label becomes permanent [only one at a time] — suppose the label $d(v)$ on node u has been made permanent— we "scan" u. That is, we look down all the arcs *from* that u and consider the label on each node at the end of such an arc:
   (a) If that node is labeled $\infty(-)$ (meaning we didn't have any path to it) we put on a temporary label ($d +$ length of the arc(u)) (showing the total distance to the node – distance to u plus length of the arc from u– and "path arrives from u")
   (b) If the node has a temporary label $d_1(v)$ and $d +$ length of the arc $< d_1$ (the distance by way of u is less than the distance shown by the label) we change the label to $d +$ length of the arc(u)
   (c) If the node has a permanent label, or a label $d_1(v)$ with $d_1 \leq d +$ length of the arc, we leave the label alone (coming by way of u, rather than the previous way, wouldn't give a shorter path
3. After the node has been scanned, we find the temporary label showing the smallest distance (smallest d) and make that label permanent.
   (a) If the destination does not yet have a permanent label, go back to step 2, using the node whose label just became permanent.
   (b) If we have a permanent label on the destination node, the d tells us the distance from the origin and the (v) tells us the node the shortest path arrives *from* - by tracing these predecessors back we find the actual shortest path (in reverse order).

This can be used to find *all* the shortest paths (to all nodes) from a particular origin - just don't stop until all nodes have permanent labels.
If the shortest path from a to b goes through x, then we have also found the shortest path from a to x and the shortest path from x to b. (the pieces of the shortest are also shortest - this is an example of what is called "the principle of optimality") To carry this out on paper, it is often easiest to make a list of the nodes and write the labels next to them - changing temporary labels as appropriate and circling labels when they become permanent.

See the example in the CD text.
Here's how the "node list" idea would look for this example (except that 1. you probably would not include the last two columns –they are there just so you can trace the steps and 2. you would probably circle the permanent label, rather than copying it into a new column):

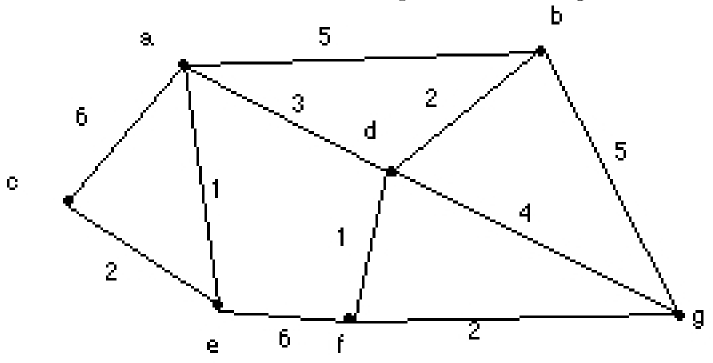
node	temporary label(s)	permanent label	first labeled	permanent
1	none	0(-)	0	0
2	18(1), 17(3), 16(4)	16(4)	1	3
3	10(1)	10(1)	1	1
4	14(3)	14(3)	2	2
5	29(4), 24(2)	24(2)	3	4
6	28(3)	28(3)	2	5
7	30(5)	30(5)	5	6

Shortest path 1 to 7 has length 30, gets to 7 from 5, comes to 5 from 2, comes to 2 from 4, comes to 4 from 3, comes to 3 from 1 so the path is $1-3-4-2-5-7$.

There is a template in the NETWORKS.xls workbook for shortest path. It works like the Transshipment template [in fact you could use the Transshipment template with one source - capacity 1 - and one destination - demand 1 - and all capacities set at 1). You enter the node names and the length for each arc. The first node listed *must* be the origin, the last node listed *must* be the destination. Unlike Dijkstra's algorithm (which finds *all* the shortest paths from the origin) the Excel solution can only find the shortest route for one destination at a time.

NOTE that if arcs are not directed ("between" rather than "from ... to ...") then they can be used either way in Dijkstra's algorithm, and must be entered *twice* in the spreadsheet. For this reason, finding the shortest path by hand is (with practice) often faster than entering the data in the spreadhseet. For large applications [computerized search], the graph would be represented by a matrix and a program would implement Dijkstra's algorithm.

EXERCISES Find the shortest path from c to g in this network



1. Using Dijkstra's algorithm
2. Using the Shortest Path spreadsheet template

READING ASSIGNMENT (in preparation for next class meeting)

Section 4.6 Maximal flow networks

SKILL EXERCISES:(hand in - individually - at next class meeting)

P. 239 #6, 18, 22