ECE-250 – Algorithms and Data Structures (Winter 2012)

Tutorial 8 (2012-03-22) – Suggested solutions

1 – Given the following directed acyclic graph $G = (V, E)$, with $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$ and $E = \{(1, 2),\ (1, 3),\ (2, 3),\ (3, 8),\ (4, 3),\ (4, 5),\ (5, 3),\ (5, 6),\ (7, 3),\ (7, 8)\}$, execute a topological sort of the vertices. You'll want to show:

- The $\Theta(|V| + |E|)$ procedure to obtain the array of in-degrees.
- The contents of the queue and the in-degrees array at each step.

Solution:

For the array of in-degrees, we traverse the list of edges, increasing the in-degree (initialized to 0) for the target vertex.

The first edge increases the in-degree of vertex 2 to 1; second and third edges increase twice the in-degree of vertex 3, which would be 2 by then. Then, in-degree of vertex 8 is increased, then in-degree of vertex 3 is increased to 3, etc.

The array of in-degrees ends with values: 0, 1, 5, 0, 1, 1, 0, 2.

We can start with either vertex 1, 4, or 7 (the ones that have in-degree 0). Say that we start with vertex 1; we enqueue them in any arbitrary order; say, 1,4,7.

Then, dequeue 1, which decreases the in-degree of 2 to 0, meaning that we enqueue 2, and the in-degree of 3 to 4. At this point, the queue contains 4, 7, 2. Then, we deque 4, decreasing the in-degree of 3 to 3, and the in-degree of 5 to 0, meaning that we enqueue 5. At this point, the queue contains 7, 2, 5. We dequeue 7, decreasing the in-degree of 3 to 2, and the in-degree of 8 to 1. Dequeue 2 and decrease the in-degree of 3 to 1. Dequeueing 5 decreases the in-degree of 3 to 0, and that of 6 to 0; we enqueue 3 and 6, which are at this point the two elements in the queue.

Etc.

2 – For the same graph given in question 1, assume we have the following weights for the edges (values given in the order corresponding to the order in which the edges were given): 2, 5, 3, 6, 1, 1, 5, 4, 5, 2.

Run Dijkstra's shortest path algorithm to show that there is no path from vertex 1 to vertex 6. What is the condition that determines that no such path exists?

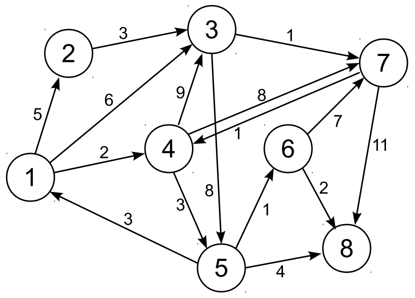
Solution:

At the first iteration (after initializing all distances to ∞ except for vertex 1, with distance 0), we set vertex 1 as visited, and mark vertices 2 and 3 with distances 2 and 5, respectively. Then, choose vertex 2 (smallest distance among unvisited vertices), check vertex 3 (with distance 5) against the distance of 2 plus the weight of the edge from 2 to 3 — $2 + 3 = 5$, so we don't update vertex 3 (this distance is not smaller than what the vertex has).

Then, select vertex 3, updating vertex 8 distance to 6. Then, set vertex 8 to visited.

At this point, there are no unvisited vertices with non-∞ distance, and since we have not reached the target vertex, this tells us that there is no path from vertex 1 to vertex 6.

3 – Given the following directed graph:



Use Dijkstra's algorithm to find the shortest path from 1 to 8.

Solution:

The iterations go as follows:

1 visited — dist(2) = 5 / p(2) = 1, dist(3) = 6 / p(3) = 1, dist(4) = 2 / p(4) = 1
4 visited — dist(3) unaffected (2+9=11 > 6), dist(7) = 2+8=10 / p(7) = 3, dist(5) = 2+3=5 / p(5) = 4
2 visited — dist(3) unaffected (5+3=8 > 6)
5 visited — dist(6) = 5+1=6 / p(6) = 5, dist(8) = 5+4=9 / p(8) = 5
3 visited — dist(5) unaffected (6+8=14 > 5), dist(7) = 6+1=7 / p(7) = 3
6 visited — dist(7) unaffected (6+7=13 > 7), dist(8) = 8 (6+2=8 < 9) / p(8) = 6
7 visited — dist(8) unaffected (7+11=18 > 8)
8 visited — Algorithm stops

Tracing back through the pointers gives us the shortest path — p(8) = 6, p(6) = 5, p(5) = 4, p(4) = 1

4 – Assuming the graph from question 3 is undirected, and considering only the edge from 4 to 7 with weight 8 (thus, eliminating the edge from 7 to 4 of weight 1), find the minimum spanning tree, using Prim's algorithm starting from vertex 1.

Solution:

Visit 1, setting dist(2)=5, dist(3)=6, dist(4)=2, dist(5)=3.

Then, visit 4 (smallest distance), setting dist(7)=8, dist(5) unaffected (it is already 3).

Visit 5, setting dist(6)=1, dist(8)=4

Visit 6, setting dist(8)=2, dist(7)=7

Visit 8, dist(7) unaffected

Visit 2, setting dist(3)=3

Visit 3, setting dist(7)=1

Finally, visit 7.

The obtained minimum spanning tree has weight 17, covering (in linear sequence) vertices 8 - 6 - 5 - 4 - 1 - 2 - 3 - 7.

5 – If enough time, use Kruskal's algorithm to find the minimum spanning tree of the graph from question 4.

Solution:

We start by including edges 5-6 and 3-7 (with lowest weight of 1); then, 1-4 and 6-8 (weight 2); then 1-5, 2-3, and we discard 4-5 since it would produce the cycle $(1, 4, 5, 1)$. Next we consider edge 5-8 with weight 4, but discard it since it produces a cycle (with 5 and 6).

Next, we include 1-2 (weight 5), and we're done.

The tree that we obtain is essentially the same; just edge 1-5 instead of 4-5 — with either one of the algorithms, the choice between these two was arbitrary; so either one could have ended up producing either one of the two possible minimum spanning trees.