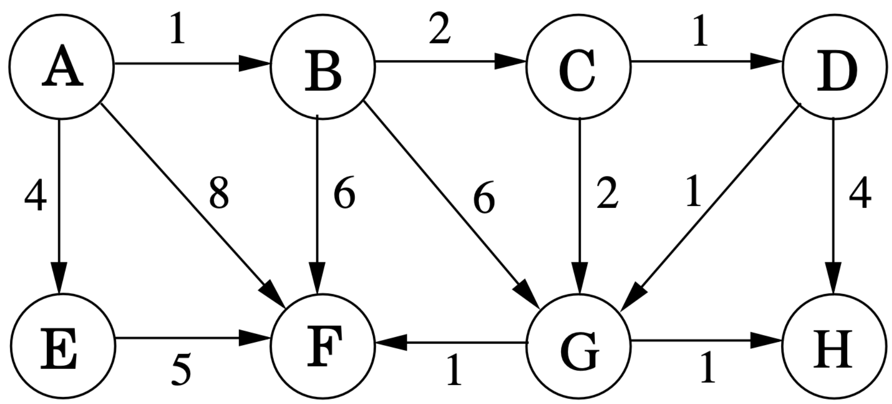
Worksheet 7

Wed, February 28, 2024

1. **Dijkstra's.** Suppose Dijkstra's Algorithm is run on the following graph, starting at node A.



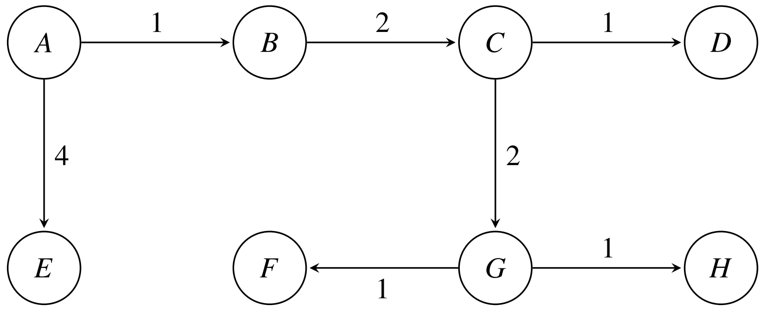
(i) Draw a table showing the intermediate distance values of all the nodes at each iteration of the algorithm.

(ii) Show the final shortest-path tree.

Solution:

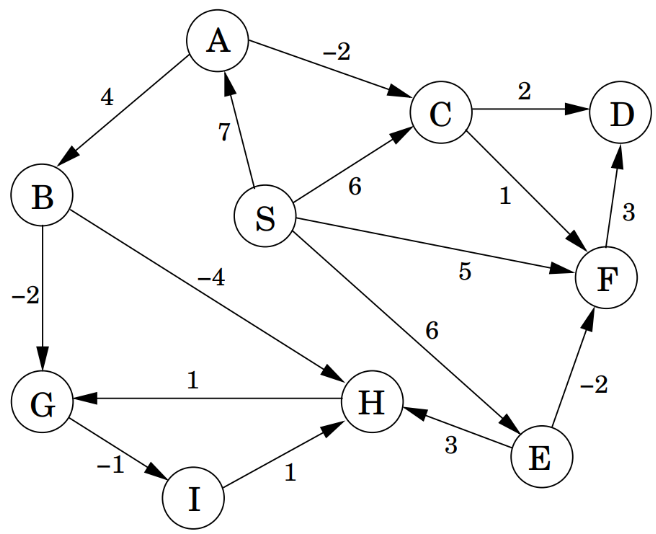
(i)

	Iteration							
Node	0	1	2	3	4	5	6	7
A	0	0	0	0	0	0	0	0
B	∞	1	1	1	1	1	1	1
C	∞	∞	3	3	3	3	3	3
D	∞	∞	∞	4	4	4	4	4
E	∞	4	4	4	4	4	4	4
F	∞	8	7	7	7	7	6	6
G	∞	∞	7	5	5	5	5	5
H	∞	∞	∞	∞	8	8	6	6



(ii)

2. Bellman-Ford. Suppose Bellman-Ford is used to find all the shortest paths from node S.



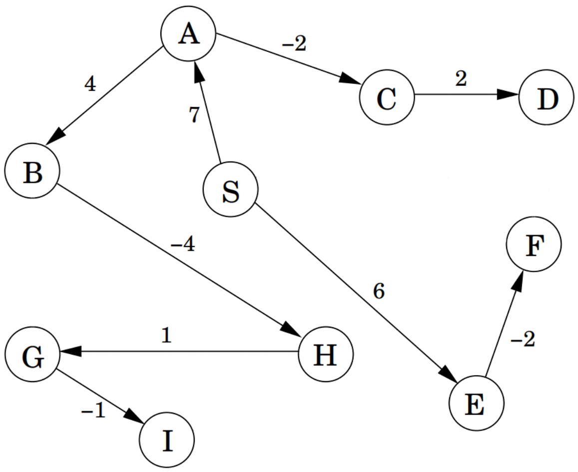
(i) Draw a table showing the intermediate distance values of all the nodes at each iteration of the algorithm.

(ii) Show the final shortest-path tree.

Solution:

(i)

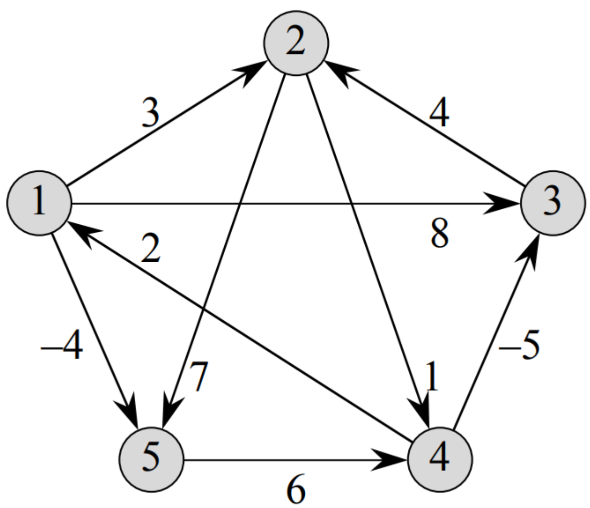
	Iteration						
Node	0	1	2	3	4	5	6
S	0	0	0	0	0	0	0
A	∞	7	7	7	7	7	7
B	∞	∞	11	11	11	11	11
C	∞	6	5	5	5	5	5
D	∞	∞	8	7	7	7	7
E	∞	6	6	6	6	6	6
F	∞	5	4	4	4	4	4
G	∞	∞	∞	9	8	8	8
H	∞	∞	9	7	7	7	7
I	∞	∞	∞	∞	8	7	7



(ii)

Note that edge FD could be included instead of edge CD. Which one is chosen is dependent on the order the edges are updated.

3. **Floyd-Warshall.** Run Floyd-Warshall to find all pairs of shortest paths in the following graph. Show the distance matrix for each step of the algorithm, including the initial and final matrices.



Solution:

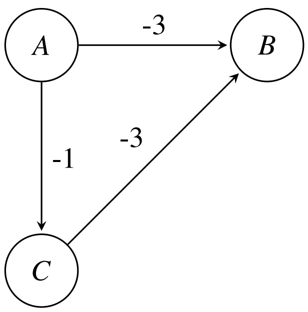
Let D_k be the distance matrix after step k, where the entry in row i and column j is the distance from node i to node j.

$$D_0 = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} D_1 = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} D_2 = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} D_4 = \begin{bmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix} D_5 = \begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

4. **Dijkstra's with Negative Edges.** Professor F. Lake suggests the following algorithm for finding the shortest path from node s to node t in a directed graph with some negative edges: add a large constant to each edge weight so that all the weights become positive, then run Dijkstra's algorithm starting at node s, and return the shortest path found to node t.
Is this a valid method? Either prove that it works correctly or give a counterexample.

Solution: Counter Example:



As per the algorithm, we will add +4 to each edge. The shortest path from A to B in this new graph will be A→B with edge weight 1. However, the shortest path from A to B in the original graph is A→C→B. In general, this algorithm overly favors short paths. A path of length one has its total increased by the large constant, but a path of length ten has its total weight increased by ten times the constant.

5. **Shortest Path via a Node.** You are given a strongly connected directed graph $G = (V,E)$ with positive edge weights along with a particular node $v_0 \in V$. Give an efficient algorithm for finding the shortest paths between all pairs of nodes, with the one restriction that these paths must all pass through v_0.

Solution:

Brute force Approach: Run Dijkstra's from each node in the graph. However this is not efficient.

Let P be shortest path from vertex u to v passing through v_0. Note that, between v_0 and v, P must necessarily follow the shortest path from v_0 to v. By the same reasoning, between u and v_0, P must follow the shortest path from v_0 and u in the reverse graph. Both these paths are guaranteed to exist as the graph is strongly connected. Hence, the shortest path from u to v through v_0 can be computed for all pairs u,v by performing two runs on Dijkstra's algorithm from v_0, one on the input graph G and the other on the reverse of G. The running time is dominated by looking up all the $O(|V^2|)$ pairs of distances.

6. **Good Nodes in a Binary Tree.** Given a binary tree, a node X in the tree is named good if in the path from the root to X there are no nodes with a value greater than X. Give an algorithm to find the

number of good nodes in the binary tree.

Solution:

We can use BFS as follows.
Algorithm:

(i) Initialize a queue to use for BFS, which will store pairs of values (i, L_i), representing a node i and that node's largest ancestor. The queue should initially contain a pair consisting of the root and a very small value (like INT_MIN). Additionally, initialize a counter to zero to store the number of good nodes.

(ii) Execute BFS: while the queue is not empty, pop from the front of the queue. At each node, first check if the value of the node is greater than or equal to the value of its largest ancestor. If it is, then increment the number of good nodes. Next, push its children onto the queue, setting the value of each child's largest ancestor to the maximum between the parent's value and the parent's largest ancestor.

(iii) At the end of BFS, we will have the total number of good nodes.

The running time of this algorithm is $O(|V|+|E|) = O(|V|)$ since the graph is a tree. DFS can also be used in the same manner since the graph is a tree.