Design and Analysis of Algorithms

Q1) You are given a weighted undirected graph G(V, E) and its MST T(V, E'). Now suppose an edge (a, b) ϵ E' has been deleted from the graph. You need to devise an algorithm to update the MST after deletion of (a, b).

Describe an algorithm for updating the MST of a graph when an edge (a, b) is deleted from the MST (and the underlying graph). It's time complexity must be better than running an MST algorithm from scratch. State and explain the time complexity of your algorithm. Analyze time complexity of your algorithm.

You can assume that all edge weights are distinct, that the graph has E edges and V vertices after the deletion, that the graph is still connected after the deletion of the edge, and that your graph and your MST are represented using adjacency lists.

Solution:

The following algorithm has time complexity O(E). Run DFS twice on what remains of the MST, once starting on a and once starting on b to determine the two connected components A and B, which you store in two HashSets. This takes O(E). Then iterate through all edges going out from the elements of A. If an edge leads to an element of B (can be check in O(1) using HashSet), check whether it is the cheapest edge so far. If it is cheaper than any other edge so far, store it. So, finding the cheapest edge can also be done in O(E). In the end, add the cheapest edge to the MST.

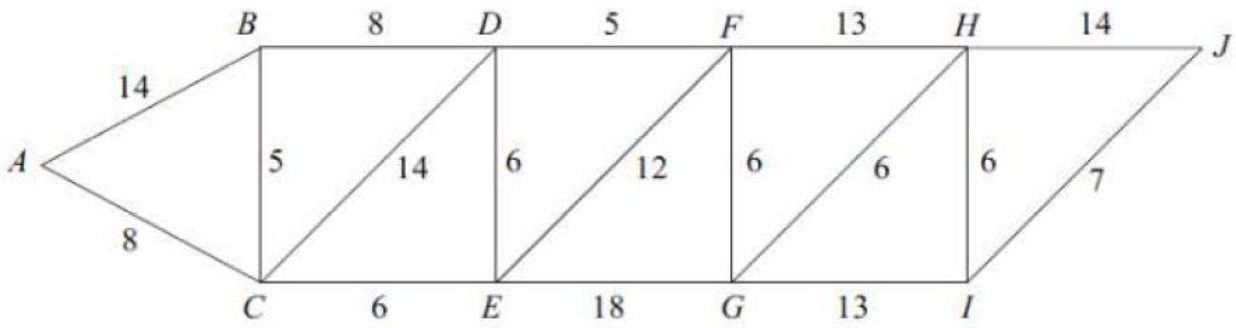
Q2) Can you use the DFS algorithm to compute the number of distinct paths between two given nodes, s and t? Two paths are considered distinct if they differ by 1 or more edges. If you think the answer is yes, then provide the pseudo-code for a DFS based algo that computes this number in O(|V|+|E|). If you think the answer is no, draw a graphwith eight vertices in total,with two of its vertices labeleds and t, in which DFS will fail to compute the number of distinct pathsbetween s and t. [5 Marks]

Solution

Yes it is possible. Call DFS on s. Keep a count at each node for how many times it is visited.

The total distinct paths from s to t will be product of count (how many times they are visited) of all nodes in the path.

Q3) The following network show times, in minutes, to travel between 10 towns. [3+2 Marks]



a) Use Dijekstra's algorithm on above figure to find minimum travel time from A to J.

b) State the corresponding route.

Q4) CM of Punjab has decided to build a new road in Lahore. He has a choice of k roads from which he can only build one. However, secretly, CM is only interested in two places in the city: s and t (his office and home). He wishes to build the road which, when added into the city, reduces the cost of the shortest path from s to t the most. He has hired you to write a computer program to tell him which of the k roads to build for this purpose. The input to your program is going to be a directed weighted graph and list of k weighted edges not already in the graph.

(a)Write an algorithm that does the job in O(k(|E|)lg|V|).Note that since k itself is $O(|V|^2)$this is a pretty slow algorithm. [4 Marks]

Solution:

Run Dijekstra to get shortest path from s to t. Then add each of k edges one by one and run dijekstra for each of the k edges to find shortest path from s to t. Select the edge which minimizes the shortest path from s to t. We will run Dijekstra k+1 times do it will take O(k(|E|)lg|V|) time.

(b) CM is in a hurry. He wants you to write an algorithm that works in O (k+|E|lg|V|), only. Can you do it?

Solution:
We use Dijkstra from s to every node x and again Dijkstra from t to every node y. This will give us the shortest distances from s ($d_s(x)$) and t ($d_t(y)$). Then we iterate over all $e' = (x, y) \in k$ and find the edge that minimizes $d_s(x) + l_{e'} + d_t(y)$. The total running time is Dijkstra plus O(K), which is O (k+|E|lg|V|)