1) The way to update it is:

Add the new weighted edges from the new vertex v to other vertexes to the the adjacency list of the current MST. The current adjacency list will look like

v1: v2
v2: v1,v3,v
v3: v2,v4,v
v4: v3, v5,v
v5: v6

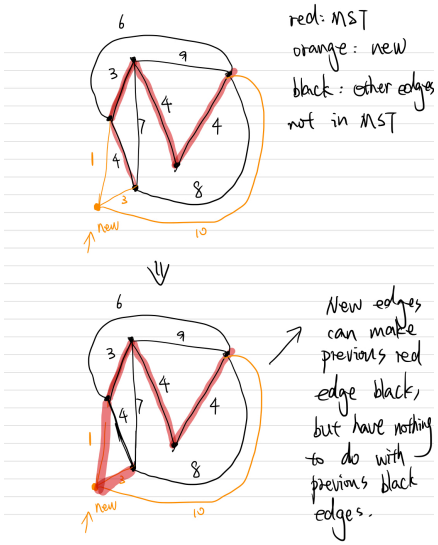
The actual adjacency of the graph: (Blue vertex means the edge between these two are not in MST before)

v1: v2, v4,v6
v2: v1,v3,v4,v5,v
v3: v2,v4,v5,v
v4: v3, v5,v1,v2,v
v5: v4,v6,v2,v3,v4
v6: v5,v1

Base on this, run Prims algorithm again.Start with any vertex and set its weight to 0 while any else vertex as a infinity. Put all them in priority queue. Update its neighbors edge weight in the priority queue according to the searching in current adjacency list. Pop the smallest edge and add it into the new MST(connect two vertex). Update the new neighbors weight around the added vertices again and repeat the process until all the vertexes in G are in T.

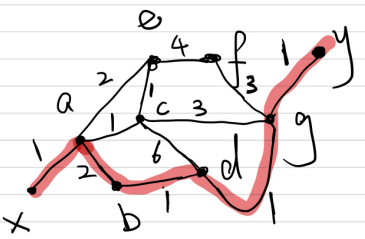
Let # vertex at last be V, the run time will be O(Vlog(V))

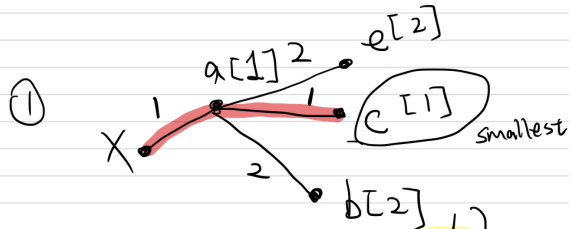
Because we have a minimum spanning tree before, so there are a bunch of edges such as v1-v4, v3-v5 that are the connections with high cost with old vertexes. As a result, when a new node come in with new edges, it is not necessary to compare the new edge weights with those edges are not in the MST before. The only possibility is that the new vertex with new edges can replace part of the edge in the old MST.

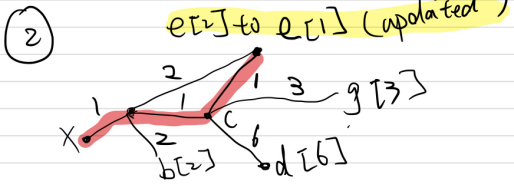


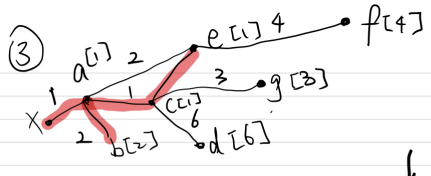
As a result, the new vertex at most can connect to every single vertex in the graph(V-1 edges), plus the edges in the old MST (V-2), the edges in the priority queue for new MST will be at most 2V-3. Search and update the priority queue then takes O(log(V) and this process is run for V times for V vertexes. So O(Vlog(V))

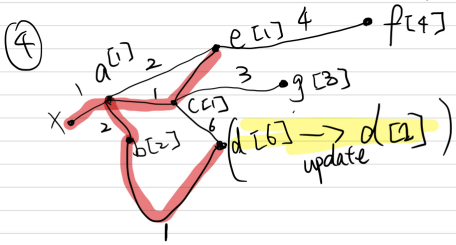
2) Solution will be run dijkstra on x until we reach the node to y from some path. Instead of adding the edge to get the vertex score, we multiply it. By containing update the smallest product score for the vertex in between the shortest path from x to y, the process is shown as:

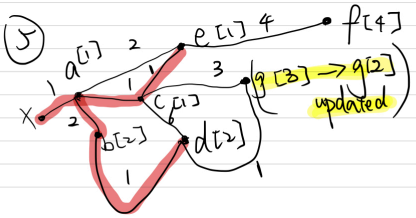


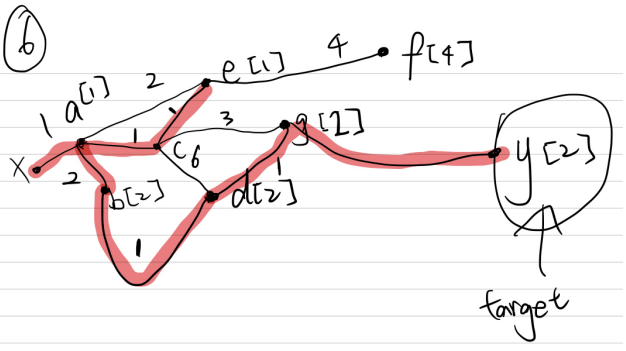


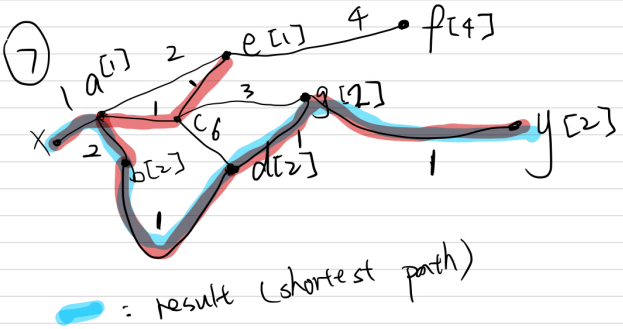












3.

The solution will be run Dijkstra on the same time from vertex x and vertex y. The weight of each edge is the the time. If x reach the some random vertex v first (the path between x and v is shorter than y to v), x will mark v as x-visited and relax edges from v. (If y come first , y will make it as y-visited and relax edges from v). If y come to v later, it will see v is x marked and therefore will not relax edges from v (vice versa). x and y will stop until they have no edges to relax anymore. After that, we are going to iterate through the vertexes to see how many of them are x-visited and how many of them are y- visited.

The time complexity by using a adjacency list represent graph will be 2* O(Elog(V)) + v, basically O(Elog(V))
A matrix table will be 2* O(V^2) + v, basically O(V^2)