SOLO

Assignment-5
CS-202
Theory

Anukool Dwivedi
B19071
Grp-13

① In MST, we take (V-1) edges from smallest to largest.
So, while finding MST, we always ~~the~~ add the smallest edge between two vertices.
So, we have the smallest edge between any two vertices.
Hence, in coding Question 2; we get the minimum of all the maximum edges, between the paths between two given vertices.

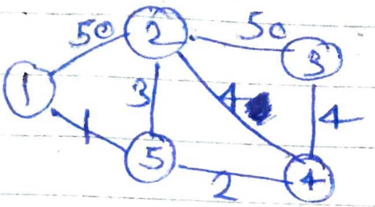
② Minimum Spanning Tree (MST) :-
MST of a given graph is a spanning tree whose length is minimum among all the spanning trees of a graph.

Minimum Bottleneck Spanning Tree (MBST) :-
MBST of a given graph is a spanning tree whose maximum edge weight is minimum among all the spanning trees of a graph.

While creating MST, we chose the edges from smallest to the largest.
So, we chose the minimum edges that

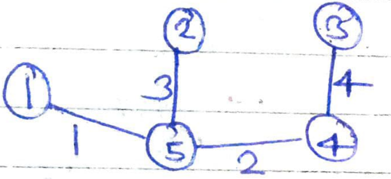
forms the tree. So; the maximum edge is minimized.

But while creating MBST; we do not guarantee MST.
For example; let us consider a graph:-



Graph

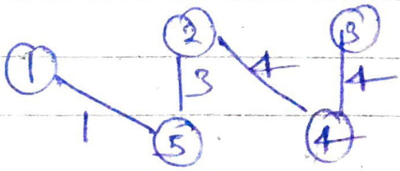
MST →



Tree 1

Sum of the edges :- 10

MBST →



Tree 2

Sum of the edges :- 12

As both Tree1 and Tree2 are spanning trees but Sum of edges in MST is 10 which is less than Sum of edges in MBST (which is 12).

So; MST implies MBST, but MBST does not imply MST.

③ Here, we consider clique as strongly connected graph of size 3 or more. So, we will find the strongely connected components of the given graph.

A strongely connected component is one in which there is a path between all pairs of vertices.

Now, we find the strongely connected components.

Algorithm :-

① Do a DFS on the original graph, keeping track of the finish times of each node. This can be done using stack, when a DFS finishes put the source vertex on the stack. This way node with highest finishing time will be on top of the stack.

② Reverse the graph using an adjacency list.

③ Do DFS on the reversed graph, with the source vertex as the vertex on top of the stack. When DFS finishes, all nodes visited will form one strongely connected component.

If any more nodes remain unvisited, this means there are more strongly connected components. So pop vertices from top of the stack until a valid unvisited node is found. This will have the highest finishing time of all the currently unvisited nodes. This step is repeated untill all nodes are visited.

(4) Now; we find max size of strongly connected components.
If max >= 3, then there is a clique
else; there is no clique.

4 Given an adjacency - list representation Adj of a directed graph, the out-degree of a vertex u is equal to the length of Adj [u], and the sum of the lengths of all the adjacency lists in Adj is $|E|$. Thus, the time complexity of out-degree of one vertex is $\Theta(|Adj(v)|)$ and for all vertices is $\Theta(V+E)$.

The in-degree of a vertex u is equal to the number of times it appears in ~~the list~~ all the lists in Adj. If we ~~searched~~ search all the lists for each vertex, the time complexity of in-degree of all the vertices is $\Theta(VE)$.

(5) Let A denote the adjacency-matrix representation of G. The adjacancy -matrix representation of G^2 is the square of A.

The square of A has the property that s_{ij} represents the number of walks of length twice (atmost) from vertex i to vertex j.

So; $G' = G^2$.

or $G' = A(G)^2$

We can find the $\cancel{A^2}$ square of A in $O(V^3)$ time.

But in $O(V^{\lg 7})$ using Strassens multiplication theorem.

It is a divide and conquer algorithm.

The recurrence relation:-

$$T(N) = 7T(N/2) + O(N^2)$$

From Master's Theorem; the time complexity of the above method is $O(N^{\lg 7})$ which is approximately $O(N^{2.807})$.