QUESTION NO. 1 (a)

```
route=0; // Global var
No of Routes (G, source, m, dest){
    route=0;
    DFS_Visit(G, source, dest, m, 0);
    return route;
}
```

```
DFS_Visit(G, src, dest, m, count){
    if(count==n){
        if(src.val=dest.val){
            route++;
        }
    }else{
        count++;
        for each u ∈ G.Adj[src]{
            DFS_Visit(G, u, dest, m, count);
        }
    }
}
```

Explanation:-

As we need to explore all the paths but only till first 4 edges. After exploration of all the paths, if we reached our dest, then route ++; else not.

To explore all paths, we apply DFS_visit (starting from source). The source will call DFS_Visit for all its adjacent nodes & ~~node~~those adjacent nodes will call DFS_Visit for all their adjacent noodes and so on untill 4 edges are complete in each path. Since every node can be connected to very other node say v no. of nodes, so, For source, there will be V DFS_visit calls. For each of these calls, there will be v more calls & so on till 4 edges have been completed. So, time complexity will be V^4, V= no of vertices

QUESTION NO. 2 (a)

```
UpdateMST((a,b)){
    flag=0;
for each edge(u,v) ∈ T.E
{
    if ((u,v) ==(a,b)) {
        flag=1;
        break;
    }
}
if(!flag) { return; }
Comp1Vertices={};
Comp2Vertices={};
for each u∈ T.V
{   u.visited= 0; }
comp1vertices.add(a)
a.visited= true;

Q1.add(a)
while ( Q1 != {}){
    v= Q1.remove();
  for each u ∈ T.Adj[v]
{   if (u.visited != true){
        u.visited = 1;
        comp1vertices.add(u);
        Q1.add(u)
    }
  }
}
comp2Vertices.add(b);
b.visited = 1;

make Q2
Q2.add(b);
while (Q2 != { }){
    v= Q2.remove;
for each u ∈ T.Adj[v]
{   if (u.visited != true){
        u.visited = 1;
        comp2Vertices.add(u);
        Q2.add(u)
}}}
shortestEdge = infinity;
AlternateEdgeset= G-E-T.E'
for each edge (u,v) ∈
    AlternateEdgeset {
if((u ∈ comp1Vertices && v ∈ comp2Vertices)||
(v ∈ comp1Vertices && v∈comp2Vertices)&&
    (u,v) < shortestEdge) {
```

Explanation:-

If deleted Edge isn't in MST, then no need to update MST. else MST has been divided into 2 connected comps. If edge (a,b) is removed then vertex 'a' belongs to one connected comp && vertex 'b' ∈ other connected comp, we are to connect to these comps back together && convert them to a single connected comp. Find all vertices in component to which vertex 'a' && 'b' belongs. now take difference of set E & E'. (it gives edges that belong to orignal graph but not MST). now we have to choose min edge from (E − E') such that 2 vertices connected by Edge ∈ differ comps. Add this min edge to set E' to complete MST.

```
shortestEdge = (u, v)
}
T.E'.add(shortestEdge);
}
```

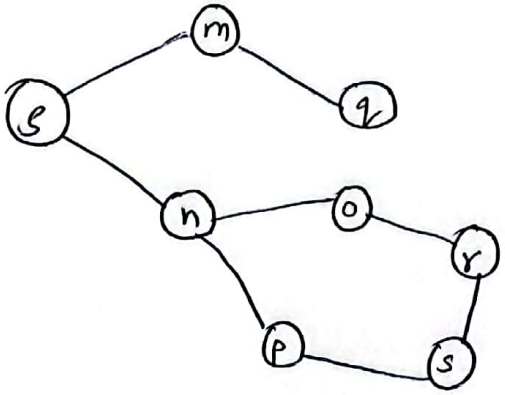
QUESTION NO. 3 (4)

```
DFS_CountPath(G,s,t){
    for each vertex v ∈ G.V{
        v.color = white;
        V.π = NULL;
    }
    for each v ∈ G.V {
        if(v.color == white){
            DFS_Visit(G, v, s,t);}
    }
}
```

```
DFS_Visit(G, src, s, t){
    time=0;
    time++;
    src.color= grey;
    src.d = time;
    src.count =0;
    for each v to G.adj[src]{
        if (v.color==white){
            v.π = src;
            DFS_Visit(G,v,s,t)
            src.count += v.count;
        }
        src.color= black;
        if(src == t){ src.count=1;}
    }
```

Explanation :-

As we cant ensure that order in which the vertices are visited, DFS will fail in finding no. of Distinct Paths in graph as below. If (m) is visited by (n) before, it will visit both o,p then path. s→m→n→p→o→r→s will be missed considering we only visit each node once.



QUESTION NO. 4

```
DFS_1(){
for each v ∈ G.V
    v.key = ∞
    v.PI = NULL
for each v ∈ G.V {
  if (v.color = white)
  {
    DFS_visit_1(G,v)
  }}
}
//Create stack s & insert the
//vertices when they are completly
//visited
  time=0 // Global
DFS_Visit_1(G, src){
      time++;
      src.d = time;
      src.color = gray;
    for each v ∈ G.Adj[src]{
      if (v.color==white){ v.PI=src;
        DFS_Visit(G,v);}
    {  time++;
       v.color = black;
       s.push(v);
}
```

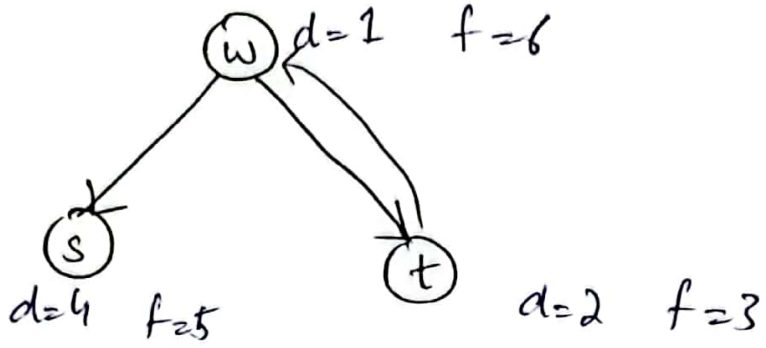
```
DFS-2(G, src){
  for each v ∈ G.V {
      v.key = ∞;
      v.π = NULL;
      count=0;
  while(s!={}){
      v = s.pop();
    if (v.color=white){
      count++;
      DFS_visit-2(G,v,count);
    }else if(v.color==black){
      joinvertice(Gt,v),
    }
}
Join Vertice(graph Gt,vertice v)
{ // Assuny that Graph is
    transposed weighted graph
    GT.
  minreachver=NULL;
  min reachvertex.weight=max
                int;
  for(all vertices of GT except
    V itself) {
  if( adjacentvertice transpose !=
    adjacent vertice){
```

```
DFS-Visit-2(G, src, n){
    src.color = gray;
    src.compNo = n;
//vertices sharing comp(same)_no
belong to same comp.
for each v ∈ (G.Adj[src]
{
    if (v.color == white){
        v.π = src
        DFS-Visit-2(G, v, n);
    }
    src.color = black;
}
```

```
if(adjacentvertice transpose.weight < minreach
                                    vertex
                                    weight){

    minreachvertex = v;
minreachver.weight = adjacentvertice
                        transpose.
                        weight;
}
}
G.adj[v] += adjacentverticetranspose;
//add adjacent transpose vertice
```

QUESTION NO. 6 (a)



Starting DFS from w.

DFS(w)

→ DFS(t) // First visit t

→ DFS(s)

v.d > u.f

4 > 3

Design analysis and Algorithm

Name: Ahmed Mukhtar (22L-8225)
Amar Ahmad (22L-6837) (BCS-4G)
Faizan Shabir (22L-6552)

Section: BCS-4H

Q.5) Bellman Ford (G, W, src)

```
{ Initialize(G, src);
  for (int i=1 to 8)
  {
     for each edge(u,v) ∈ G.E
     {  Relax (u,v,w) }
  }
}

Relax (u,v,w)
{
   if (v.cost > u.cost + w(u,v))
   {
      v.cost = u.cost + w(u,v)
      v.π = u
   }
}

Initialize (G, src)
{  for each "v" belongs to G.V || which are 8
   {  v.cost = infinity
      v.π = NULL
   }
   src.cost = 0
}
```

Belman For overall time complexity is $O(V*E)$ however in this as we only have 8 vertices so $8 \times E$ and $E = V-1$ so overall time complexity of this code is $O(8V) = O(V)$