Name: Pratik Suryawanshi

ASU ID: 1213231238

1. To find the outdegree of every vertex V, we need to iterate through entire neighbour list of every vertex.

∴ To find sum of edges for a vertex it ~~will~~ would take O(|E|) time where E is number of edges. and we do it for all vertices, hence it will take O(|E|+|V|) time.

To compute indegree, we need to scan all adjecency list neighbours to find how many times vertex V has appeared.. Hence it will take O(|E|+|V|) time.

Problem 2.

For Adjecency list representation, we can create new Adjecency list to store the transpose of graph. Here we can check through every list of every vertex v to get vertex u and put value of v in adjecency list of u in new transpose graph presentation.

This will take $O(|V|+|E|)$ time

```
Graph getTransponse ( Graph G ) {
    Graph Gt ;
    for ( vertex v in G.vertices) {
        for ( vertex u in v.adjList) {
            Gt [u]. adjList. append (v)
        }
    }
    return Gt ;
}.
```

Scanned by CamScanner

For Adjecency matrix, we can swap the values to get transpose matrix. Hence to get transpose graph in matrix representation, we can swap values above diagonal with values below diagonal.

```
function getTranspose ( Graph G[][]) {
    For (i=0 ; i< V/2 ; i++) {
        for (j=0 ; j< V/2 ; j++) {
            swap ( G[i][j] , G[j][i])
        }
    }
}
```

Above method will take $O(|V^2|)$ time.

Problem 3.

Here q is first visited node ∴ visited[q] = 1.

Adj List for q = [s, w, t].

Now given is that nodes in Adj list are alphabetically sorted ∴ Adjlist q = [s, t, w].

According to dfs, we now explore adjecency list of s ∴ visited [s] = 2.

AdjList for s = [v] ∴ visited [v] = 3

AdjList for v = [w] ∴ visited [w] = 4

AdjList for w = [ ] ∴ finished [w] = 5

Now that w is finished and marked visited, we return back to v. ∴ finished [v] = 6.

Return back to s ∴ finished [s] = 7.

Now get next node from Adj List of q which is not visited. That node is t.

∴ visited [t] = 8

AdjList t = [x, y]

∴ visited [x] = 9.

AdjList of x = [z] ∴ visited [z] = 10.

AdjList of z = [ ] ∴ finished [z] = 11

Return back to x ∴ finished [x] = 12

Get next unvisited node from t ∴ visited [y] = 13

AdjList for y = [q] but q is already visited

∴ finished [y] = 14.

Now all nodes of t are visited ∴ finished [t] = 15.

Last node of q is already visited ∴ finished [q] = 16

Now get adj List for r = [u, y] ∴ visited [r] = 17.

get AdjList for u = [y] ∴ visited [u] = 18

but y is already visited ∴ finished [u] = 19.

Nodes in r are over ∴ finished [r] = 20

∴ Following table shows discovery & finish time for all vertices.

vertex	visited	finished.
q	1	16
r	17	20
s	2	7
t	8	15
u	18	19
v	3	6
w	4	5
x	9	12
y	13	14
z	10	11

Tree edge : encounter new edge

∴ Tree edges = (q,s) (s,v) (v,w) (q,t) (t,x) (x,z) (t,y) (r,u)

Back edge : from decendent to ancestor

Back edges = (w,s) (z,x) (y,q)

Forward edge = (q,w)

cross edges = (u,y) (r,y).

Problem 4.

If graph is represented using adjecency matrix instead of adjecency list then while searching for adjecent vertices in graph, we have to check entire row in matrix. If there is no edge between 2 vertices then weight stored in matrix is 0. Consider below algorith where M is adjeceney matrix calculated and passed to function:

```
MST-Prim (G, w, r, M)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);

        for each v ∈ V[G] do
            if (M[u][v] != 0 and v ∈ Q and
                    M[u][v] < key[v])
                p[v] = u;
                key[v] = M[u][v];
```

In this algorithm, we have changed the if condition required to update parent and key where instead of comparing weight in adj list, we are comparing M[U][V]. with 0 to determine if there is edge between U and V.

Again M[U][V] is compared with key[v] to set minimum value

Outer while loop executes for all vertices in Q and inner for loop also executes for all V[G]

Hence outer while loop for O(|V|) and inner for loop for O(|V|) which results in running time of O(|V^2|). for this algorithm.

Problem 5

In first step of algorithm we set parent of all nodes as NILL and distance as infinite except source ∴ we have.

∴	z	t	x	s	y.
	0	∞	∞	∞	∞.

In ~~second~~ 1st iteration, we explore vertices x and s

∴	z	t	x	s	y
d	0	∞	7	2	∞
π	N	N	z	z	N

$d(x) = \min(\infty, 0+7) = 7$

$d(s) = \min(\infty, 0+2) = 2$

In ~~third~~ 2nd iteration, we explore vertex t from x and y from s.

	z	t	x	s	y
d	0	5	7	2	9
π	N	x	z	z	s

$d(t) = \min(\infty, 7-2) = 5$

$d(y) = \min(\infty, 7+2) = 9$

In ~~fourth~~ 3rd iteration because of edge (y, x) we found distance from y to x decrease existing value to reach x from source z.

∴

	z	t	x	s	y.
d	0	5	6	2	9
π	N	x	y	z	s.

$d(x) = \min(7, 9-3)$
$= 6$

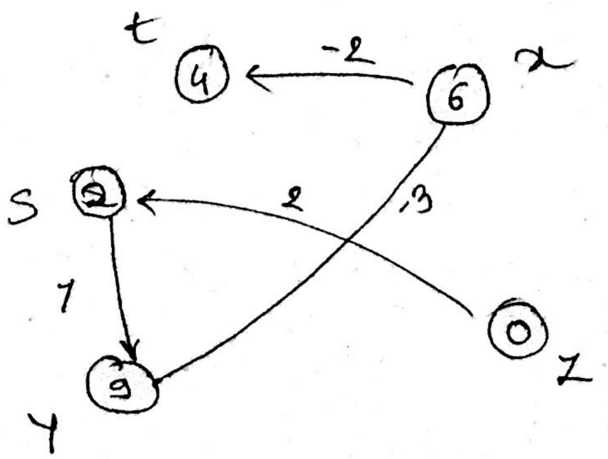
In last iteration due to edge (x, t) we can ~~up~~ update value to reach ~~nod~~ vertex t since x dista. is also changed.

∴

	z	t	x	s	y.
d	0	4	6	2	9
π	N	x	y	z	s

$d(t) = \min(5, 6-2)$
$= 4.$

∴ Final graph ~~sou~~, showing single path from vertex z as source would be:



Now edge (z,x) is changed from 7 to 4.

Running Bellman ford's Algorithm with source s,

Initially set distance as ∞ and parent as NILL. except source.

	s	t	x	z	y.
d	0	∞	∞	∞	∞
π	N	N	N	N	N

Now in first iteration, vertex t and y are discovered

	s	t	x	z	y
d	0	6	∞	∞	7
π	N	s	N	N	s

$dis(t) = \min(\infty, 0+6)$
$= 6$

$dis(y) = \min(\infty, 0+7)$
$= 7.$

In second iteration, vertex x and z are discovered.

	s	t	x	z	y.
d	0	6	4	2	7
π	N	s	y	t	s

$dis(x) = \min(\infty, 6+5) = 11$

$dis(x) = \min(11, 7-3) = 4.$

$dis(z) = \min(\infty, 6-4) = 2$

In third iteration, dis(t) gets updated since there is edge (x, t) and x was updated in last iteration.

	s	t	x	z	y
d	0	2	4	2	7
π	N.	x	y	t	s

dis(t) = min(6, 4-2) = 2

In fourth iteration, dis(z) updated due to w(t, z) and dis(x) due to w(z, x)

	s	t	x	z	y
d	0	2	2	-2	7
π	N	x	z	t	s

dis(z) = min(2, 2-4) = -2

dis (x) = min(4, 4-2) = 2.

~~Algo ends here.~~ Since there is -ve cycle present in graph, while checking d[v] > d[u] + w(u,v) for edge (x, t) if condition gets satisfied and 'no solution' is returned.

Scanned by CamScanner

Problem 6,

Initially we set distance as ∞ and parent NIL for all vertices ~~from~~ except source.

$\therefore$

	s	t	x	y	z
d	0	∞	∞	∞	∞
π	N	N	N	N	N.

$S = \{\ \}$

$Q = \{s, t, x, y, z\}$

In first iteration, we get min. from Q.

$\therefore$ source s is put into S. and relax is called.

	s	t	x	y	z
d	0	3	∞	5	∞
π	N	s	N	s	N

$S = \{s\}$

$Q = \{t, x, y, z\}$.

In second iteration, min. from Q is $t = 3$ which is put in S and relax is called.

	s	t	x	y	z
d	0	3	9	5	∞
π	N	s	t	s	N.

$S = \{s, t\}$

$Q = \{x, y, z\}$.

In third iteration, min from Q is y.

∴ Putting y in S and relax gives following:

	s	t	x	y	z
d	0	3	9	5	11
π	N	s	t	s	y

$S = \{s, t, y\}$

$Q = \{x, z\}$

In fourth iteration x is min in Q hence it is putted in S and relax is called.

	s	t	x	y	z
d	0	3	9	5	11
π	N	s	t	s	y.

$S = \{s, t, y, x\}$

$Q = \{z\}$.

In fifth iteration z is putted into S

	s	t	x	y	z
d	0	3	9	5	11
π	N	s	t	s	y

$S = \{s, t, y, x, z\}$

$Q = \{\}$

Algo ends at this step. Final shortest path: