Experiment 10

Routing Algorithms

A. Write a C/C++ program for Dijkstra's algorithm to find the shortest path

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 99
#define MAX 10
voiddijkstra(int G[MAX][MAX], int n, intstartnode);
void main()
{
        int G[MAX][MAX], i, j, n, u;
        printf("\nEnter the no. of vertices:: ");
        scanf("%d", &n);
        printf("\nEnter the adjacency matrix::");
        for(i=0; i<n; i++)
        for(j=0; j<n; j++)
                scanf("%d", &G[i][j]);
        printf("\nEnter the starting node:: ");
        scanf("%d", &u);
        dijkstra(G,n,u);
        getch();
}

voiddijkstra(int G[MAX][MAX], int n, intstartnode)
{
        int cost[MAX][MAX], distance[MAX], pred[MAX];
        int visited[MAX], count, mindistance, nextnode, i,j;
        for(i=0;i <n;i++)
                for(j=0;j <n;j++)
                        if(G[i][j]==0)
                                cost[i][j]=INFINITY;
                        else
                                cost[i][j]=G[i][j];
```

```
for(i=0;i<n;i++)
{
          distance[i]=cost[startnode][i];
          pred[i]=startnode;
          visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
count=1;
while(count < n-1)
{
          mindistance=INFINITY;
          for(i=0;i <n;i++)
                    if(distance[i] <mindistance&&!visited[i])
                    {
                              mindistance=distance[i];
                              nextnode=i;
                    }
          visited[nextnode]=1;
          for(i=0;i <n;i++)
                    if(!visited[i])
                              if(mindistance+cost[nextnode][i] < distance[i])
                              {
                                        distance[i]=mindistance+cost[nextnode][i];
                                        pred[i]=nextnode;
                              }
                    count++;
}
for(i=0;i< n;i++)
          if(I !=startnode)
          {
                    printf("\nDistance of %d = %d", i, distance[i]);
                    printf("\nPath = %d", i);
                    j=i;
                    do
                    {
                              J=pred[j];
                              printf(" <-%d", j);
                    }
                    while(j!=startnode);
          }
}
```

B. Write a C/C++ program for Bellman-Ford algorithm to find the shortest path

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <iostream>

struct Edge
{intsrc, dest, weight;};

struct Graph
{
          int V, E;          // V-> Number of vertices, E-> Number of edges
          struct Edge* edge;      // graph is represented as an array of edges.
};

// Creates a graph with V vertices and E edges
struct Graph* createGraph(int V, int E)
{
          struct Graph* graph = (struct Graph*) malloc( size of (struct Graph) );
          graph->V = V;
          graph->E = E;
          graph->edge =(struct Edge*) malloc( graph->E * size of (struct Edge ) );
          return graph;
}
// A utility function used to print the solution
```

```
voidprintArr(intdist[], int n)
{
        printf("Vertex   Distance from Source\n");
        for (inti = 0; i< n; ++i)
                printf("%d \t\t %d\n", i, dist[i]);
}
voidBellmanFord(struct Graph* graph, intsrc)
{
        int V = graph->V;
        int E = graph->E;
        intdist[V];

   // Step 1: Initialize distances from src to all other vertices as INFINITE
        for (inti = 0; i< V; i++)
                dist[i] = INT_MAX;
        dist[src] = 0;
        for (inti = 1; i<= V-1; i++)
        {
                for (int j = 0; j < E; j++)
                {
                int u = graph->edge[j].src;
                int v = graph->edge[j].dest;
                int weight = graph->edge[j].weight;
                if (dist[u] != INT_MAX &&dist[u] + weight <dist[v])
                dist[v] = dist[u] + weight;
                }
        }
        for (inti = 0; i< E; i++)
        {
                int u = graph->edge[i].src;
                int v = graph->edge[i].dest;
                int weight = graph->edge[i].weight;
                if (dist[u] != INT_MAX &&dist[u] + weight <dist[v])
        printf("Graph contains negative weight cycle");
        }
        printArr(dist, V);
        return;
}
int main()
{
        int V,E;
        printf("\nEnter the no. of vertics: ");
        scanf("%d", &V);
        printf("\nEnter the no. of Edges: ");
        scanf("%d", &E);
        struct Graph* graph = createGraph(V, E);
        int p,q,r;
        char a='y';
        inti=0;
        while(i<E)
        {
                printf("for  %d edge Enter the  source:", i);
                scanf("%d",&p);
                graph->edge[i].src = p;
                printf("Enter the destination:");
                scanf("%d", &q);
                graph->edge[i].dest =q;
                printf("Enter the weight:");
                scanf("%d",&r);
                graph->edge[i].weight = r;
                i++;
        }
        BellmanFord(graph, 0);
        return 0;
}
```

Exercise: Write a C/C++ program to implement the Distance Vector routing algorithm.