Lab 13

Due No Due Date **Points** 1

You are suggested to use the teaching servers burrow.soic.indiana.edu or hulk.soic.indiana.edu or tank.soic.indiana.edu for practicing C programs.

Lab 13: All Pair Shortest Path

Given a weighted directed graph, an all pair shortest path algorithm finds out the shortest path between any two vertices. Unlike a single pair shortest path algorithm which finds out the shortest path from a single source, an all pair shortest path algorithm is able to find out shortest path from any vertices to any other vertices.

Floyd-Warshall Algorithm

Floyd-Warshall algorithm is a **dynamic algorithm** to find out all pair shortest paths. It works even with negative-weight edges. Only constraint is that there must not be any negative weight cycle, otherwise the shortest path distances would be minus infinity.

First we number every vertex from **1** to **n**. Let i, j be the ith vertex and jth vertex. W(i,j) is the weight of the edge between vertices i and j. Let k be the kth vertex. Define ShortestPath(i,j,k) as the shortest path between i and j possibly using node 1...k as intermediate nodes on the path. We can then define shortestPath(i, j, k) in terms of the following recursive formula:
the base case is:
ShortestPath(i, j, 0) = w(i, j) // which means no intermediate nodes is used on the path.

And the recursive case is:
ShortestPath(i,j,k) = min(ShortestPath(i,j,k-1), ShortestPath(i,k,k-1) + ShortestPath(k,j,k-1))

This formula is the heart of the Floyd–Warshall algorithm. The algorithm works by first computing shortestPath(i, j, k) for all (i, j) pairs for k = 0, then k = 1, etc. This process continues until k = n, and we have found the shortest path for all (i, j) pairs using any intermediate vertices.

The Floyd-Warshall algorithm is given below:

```
Floyd_Warshall(G)
        V = no of nodes
        E = no of edges

        initialize all D[V][V] = INF

        For each i = 1 to V
                D[i][i] = 0

        For each edge (u,v,weight)
                D[u][v] = weight

        For k from 1 to V
                For i from 1 to V
                For j from 1 to V
                        IF D[i][j] > D[i][k] + D[k][j]
                                D[i][j] = D[i][k] + D[k][j]
```

Note that the constant factor is small in Floyd-Warshall algorithm, which makes a practical algorithm for many graphs.

Tracking shortest path

The above algorithm just compute the shortest path total weight. We also want to print the actual shortest path between two vertices. We can use another 2D array intermediate[V][V] to keep track of the actual shortest path. Intermediate[u][v] = k implies that to go to v from u, we need to go to k first. We can do the tracking along with computing the shortest path weight. Shortest path with path tracking algorithm is given below:

```
Floyd_Warshall(G)
        V = no of nodes
        E = no of edges

        initialize all D[V][V] = INF
        initialize all intermediate[u][v] to null

        For each i = 1 to V
                D[i][i] = 0

        For each edge (u,v,weight)
                D[u][v] = weight
                intermediate[u][v] = v

        For k from 1 to V
                For i from 1 to V
                        For j from 1 to V
                                IF D[i][j] > D[i][k] + D[k][j]
                                        D[i][j] = D[i][k] + D[k][j]
                                        intermediate[i][j] = k

PrintPath(u, v)
{
        IF intermediate[u][v] is NULL
                Return

        IF intermediate[u][v] == v
                print v
                Return

        PrintPath(u, intermediate[u][v])
        PrintPath(intermediate[u][v], v)
}
```