Matthew Soicher
z5160737

Question 4

First, we represent our graph as an adjacency matrix; each connection from vertex i to vertex j with weight a is denoted as $AdjMatrix[i][j] = a$

If no such edge from vertex x to vertex y exists, then $AdjMatrix[x][y] = -\infty$.

The idea behind the solution uses Dynamic Programming in which we build a 3D table: dimension 1 is our source vertex, dimension 2 is our destination vertex, dimension 3 is the number of edges from source to destination and the value in each position is the total weight.

Algorithm

int maxWeight(graph G, int src, int dest, int K)

// form adjacency matrix = graph
Int currentMaxPath = $-\infty$

for (int e = 0 to K)
    for (int i = 0 to v)
        for (int j = 0 to v)

// initialise current position
map[i][j][e] = $-\infty$

if (number of edges e == 0 and src i = destination j)
map[i][j][e] = 0

if (number of edges == 1 and our AdjMatrix[i][j] $\neq -\infty$***)***
an edge exists and map[i][j][e] = AdjMatrix[i][j]

----------- Now, we check the case when more than 1 edge is at available between the current
----------- src and dest vertices

If (e > 1)

----------- enter a loop to find an edge from i to m. Since our path can be self-intersecting, i.
----------- can equal m, given that i is not equal to j

end if

for (int m = 0 to v)

if (AdjMatrix[i][m] $\neq -\infty$ ***and i*** $\neq$ ***m and map[m][K][e-1]*** $\neq -\infty$***)***
map[i][j][e] = max(map[i][j][e], AdjMatrix[i][m] + map[m][j][e-1]
end if

Matthew Soicher
z5160737

end for

Here, we update the current maximum path

if (map[i][j][e] > currentMaxPath and e = K)

currMaxPath = map[i][j][e]
src = i
dest = K

end if

end for

end for

end for

Finally, we return *currentMaxPath* as our final result.

The time complexity of the algorithm is $O(v^3K)$