#Vatsal Rathod
#student-id:801259046
#DSA-Project1
================================

I have written the project code using python3.
The program starts with the main function and creates an instance of the graph class and with the help of the input file, we create a graph based on the inputs. Further, on the basis of queries, we update, print, or calculate the path distance of the edges between 2 vertexes as per the input.

Structure & Program Design:
In the graph, we have used a dictionary to store all the vertex objects of the graph. We have defined a class that stores all the vertex information i.e. to store the name, adjacency list, and status of the vertex. For adding and deleting edges I have created addedge and deleteedge. In Dijkstra's for priority queue we have used binary heap and for BFS I have implemented queue using deque from collections.

Graph Creation:
As we read the input file, we call the addEdge() function where we check whether the mentioned source and destination node are available, if yes we append the destination node in the source adjacency list and vice versa. If the node doesn’t exist we create and append the node.

Addedge:
we use addEdge_func() function to create a edge from source to destination. As above, we check if both nodes exist or not and then append the destination node to the source node.

Deleteedge:
With the use of deleteedge() function we check whether the destination vertex is present in the adjacency list of sourcenode, we delete it if it is present in the list.

Edgedown:
With the edgedown() function we check whether such node exists, if yes we update the status of that edge to down.

Edgeup:
With the use of edgeup() function we check whether such node exists, if yes, we update the status of that edge to up

vertexup:
With the use of vertexup() function, we update the status attribute of the mentioned vertex as up

vertexdown:
Using vertexdown() function, we update the status attribute of the

mentioned vertex as down

Dijkstras:
created two dictionaries, one for distance and set all the path to infinity and one dictionary to store parent and set all values to nil. Defined a priority queue and inserted source node, later iterated until all the nodes are not visited. In the end we print the distance between the 2 edges and the path between them using printPath_() function.

Print:
printvertexmap() sorts the vertexMap dictionary and iterates over each vertex and calls the function printvertex() to print all the vertex in its adjacency list with their weights. Similarly, printvertex() sorts the list and prints it.

Reachable:
In this part, we call BFSvertex(), where we pass each vertex to BFS() to find its reachable vertex and print them. We define 2 dictionaries, one to store parent and another to store the color of the vertex. We use a queue and keep appending the vertex to the queue and keep traversing them until all the reachable vertex is reached. We keep printing each vertex as we traverse them. The complexity for the reachable algorithm is O(N*(V+E)) as N is given by the number of nodes, V is given by adjacent vertices and E is no. of edges

Shortcomings:
In my code, I am not keeping a check on the values of weight because of which there might be a chance where weight may be negative and hence we might not get an optimal solution using Dijkstra’s. We can improve the efficiency of reachable code by keeping a track of reachable nodes.