QUESTION1

a) Shortest path between node 1 and node 4 is [1 -> 5 -> 4] with weight value is 8. We basically compute all intermediate paths between 1 to 4 and store in a data structure, each iteration we compare new shortest path value and old shortest path value and update new shortest path value with less one. To calculate all paths firstly we can start from [1 -> 3 -> 2 -> 4] with value 22 and it's lower than infinity so we select as candidate answer, then we can continue with another path which is [1 -> 2 -> 4] with value 10 and select this path because it has lower weight rather than first path. Then we find third path which is answer [1 -> 5 -> 4] with weight value is 8 and select as final answer.

b) There are 2 different type shortest path algorithms. One of them is single-source shortest path algorithms. With given source node, it aims to find shortest path to all other nodes another shortest path algorithms is aim to find the shortest path between all pair nodes. Dijkstra and Bellman Ford's algorithm are single-source shortest path algorithms. Floyd-Warshall is an example of all-pairs shortest path algorithm.

We can use dijkstra algorithm which uses greedy approach it works only positive weights in graphs. Dijkstra algorithm have O($ElogV$) worst-time complexity which V is node number and E is edge number in a given graph.

We can use bellman ford's algorithm which uses dynamic programming approach, it also works in negative weights. Bellman Ford's algorithm have O(VE) time complexity which worst than dijkstra algorithm.

We can use floyd-warshall algorithm which uses dynamic programming approach, it works positive and negative weights. It have O(V^3) time complexity which is worst than other algorithms.

If we want to look advantages or disadvantages of given algorithms, we can use dijkstra algorithm if we want to only single-source shortest path and our graph have positive edge values, if we have negative weights in graph we can use bellman ford's algorithm. Floyd-Warshall have advantage to work on both positive and negative values also it finds all-pair shortest paths in a given graph. Dijkstra better than Bellman Ford's algorithm in space complexity since it don't use extra data structure.

c) You can find implementation of Floyd-Warshall algorithm in zip file as que1.py file. Code contains 3 different functions. printSolutionSteps take distance matrix as input and print current shortestPath weight between any 2 node in graph as matrix representation. printTotalWeights take distance matrix as input and print shortest path weight between any 2 node in graph. We use these functions to print values in FindsShortestPath function. FindShortestPath function takes adjacency matrix as input. If there are an edge between nodes we put weight of edge in matrix with respect to indexes but if there are not an edge between two node we put infinite value. We put specific infinite value because it's help to find possible paths and not-possible paths. First we create a distance matrix from given adjacency matrix representation. It contains shortest distances between every pair of nodes. Firstly we construct distance matrix from using adjacency matrix values. Then we use three loops, with first loop we use k variable to find intermediate node and use i and j variables use as source and destination nodes. if node k is on the shortest path from i to j then we update the value of distance matrix using intermediate k node. We basically update distance matrix according to all nodes as an intermediate node. In each iteration we pick a k node as intermediate nodes and compare with (i, j) pair which i is source and j is destination. If k is not an intermediate node in shortest path from i to j, we don't change value of distance matrix in i row and j column but if k is an intermediate

node in shortest path from i to j then we update distance matrix in i row and j column according to dist[i][j] = dist[i][k] + dist[k][j] if dist[i][j] greater than dist[i][k] + dist[k][j], according to this calculation we find shortest-path weights between direct edge values or multiple hop traversal.

d) According to our code, in FindShortestPath function we have three inner loops to calculate each shortest path between nodes, except these we have two different print function with two inner loops and two inner lambda function to initiate distance matrix. So worst-case time complexity will be O($(V^3) + (V^2) + (V^2) + (V^2)$) -> O($(V^3)$) where V is count of nodes in given graph. In 3 inner loops we use for loop from 0 to node number which is V and in our case V=5. Also we use V in print function since we have matrix to store. So in addition we have O((V^2)) worst-case space complexity since we use distance matrix to store shortest path total weights in data structure. Since between each node we have a weight we use VxV matrix. In this problem we used dynamic programming approach to solve this problem. We use a matrix data structure to hold our shortest path weights and iterate over nodes to update shortest path weights. In this data structure we basically find shortest path weights between neighbour nodes and keep these calculations to find further shortest path calculations.

QUESTION2

a) If we consider as graph with 2-colorable graph, we can color every node either red our blue and no edge have both endpoints colored same color. We can think that A denote the subset of nodes colored red and B denote the subset of nodes colored blue like two disjoint parts. $B1 \cup A2 = N$ Since all nodes of A are red there must be no edges within A and also within B. This implies that every edge has one endpoint in A and B which shows G is a bipartite.

We assume that given graph is bipartite and start with BFS algorithm, we can divide hops in two different groups with odd and even hops. In each hop we alternate color in given node so that each hop group have to same color. If two neighbour in same colour it could not a bipartite graph since they are in same hop group, if all neighbour nodes have different colors it must be a bipartite graph.

b) You can find implementation of isBipartite algorithm in zip file as que2.py. Algorithm uses BFS to check given graph is bipartite or not. It's randomly select a node and color as red. Then color all neighbours with blue color then color all neighbours of blue coloured nodes with red color and loop continues. With these loop we assign a color all nodes to check if it satisfies prove in given a section. If it satisfies we return True if it is not we return False. When we assigning colors if we find a neighbour which is colored with same color then graph cannot colored with 2 nodes so it's not bipartite graph. In code firstly we create a color array to store colors and assigned all nodes as -1 since they are not visited. With 1 we indicate that node coloured as red with 0 we indicate that node coloured as blue. Then we create a queue data structure to use in BFS. Then we select a random node and color as red and visit neighbour nodes and color as blue and loop continues until graph traverse completed. If any neighbour node is coloured with same color in source node we return it's not bipartite graph, if graph traverse is completed we return it's a bipartite graph..

Question3

Let's think given graph with 4 nodes such as x,y,z,d. We can think that edges of the graph can be (x,y),(x,z),(z,d) with weights 3, 1, and 2. We can suppose that A is the set of (x,z) so let S equals A, then we can think that with S respects to A, since given G is a tree, it must be minimum spanning tree itself so A must be a subset of a minimum spanning tree and every edge is safe, Also (x,y) must be safe but it's not a light edge for this cut since it's a incorrect theorem with given this counter example.