NAME: UVIN ABEYSINGHE STUDENT ID: 789931

‘n’ variable which is passed will have the number of vertices in the graph. The Adjacent Matrix would be the size of n by n (n x n).
Assuming that the vertical column contains the starting vertex of the edge. (since undirected)

PART 6
DFS-The only thing that is going to change is the way finding the next edge.
We go to the row of the starting vertex and we parse through the row from left-to-right until a cell with ‘1’ is found. When it is found, the column number is used to identify the pointing vertex. Current vertex is pushed into a stack and the next vertex edges are parsed and pushed into a stack the same way as part 1. When a row with all the edges are visited or no edges are parsed, the stack is popped and again the rows are checked for edges which are not visited. This is repeated until all the vertices are visited.

BFS-We use a queue in the same way that is used in part 2. We go to the row of the starting vertex, and parse through the row from the left to the right until the end of the row (n cells). Every time ‘1’ is found in a cell, the column number is enqueued. (The vertex number the edge is pointing at.) When we reach the end of the row, we dequeue the queue and use the next vertex number to select the row we are going to parse. If the vertices the edges are pointing to are visited, they can be ignored. Whenever the queue is enqueued, the vertex being enqueued is printed out. This is repeated until the queue is empty.

PART 6
3-There are two lists of integers that run side by side containing the current path and the distance from each vertex to vertex in the current path. Depth-First traversal (explained in part 6) is run until an edge that points to the destination vertex is found. When an edge points to a vertex, if it’s not visited, the vertex id will be added to the current path list (at the end). Whenever a vertex is added to the list (at the end), the weight of the edge is added to the list (at the end). When all the edges of the current vertex are visited and destination not found, 1 distance and 1 vertex is removed from the back of the lists. When the destination is found, the lists are removed from the start, distances are added to a variable to give the cumulative distances (also printed) while the vertices are printed until the lists are empty.

4-Recursion was used with a helper function. All the vertices each edge is pointing is passed as the starting vertex while the vertices in the current path is noted as visited. Every time the function is called, we check if the starting vertex is the same as the destination. When that happens, we print out the list containing the current path. When returning we remove the end element in the list. This is continued until all the paths are found.
5-It used the same approach as **all paths.** Additionally, it contains another list with the shortest path till that point in the program, the shortest distance till now and the current distance (cumulative distance of the current path). Every time the destination is met, its checked if the current distance is smaller than the previously shortest one, if so the shortest path and distance is changed, if not a new path is checked. After all the paths are checked, the shortest path and the shortest distance is printed out.

Bonus Question

1) Done in part 1
2) n- number of vertices
   m- number of edges
   Part3- O(n+m)
   Part4- O((n^2)+(m^2))
   Part4- O((n^2)+(m^2))