DSA ASSIGNMENT (TASK 2)

Name : **Avinash R Changrani**

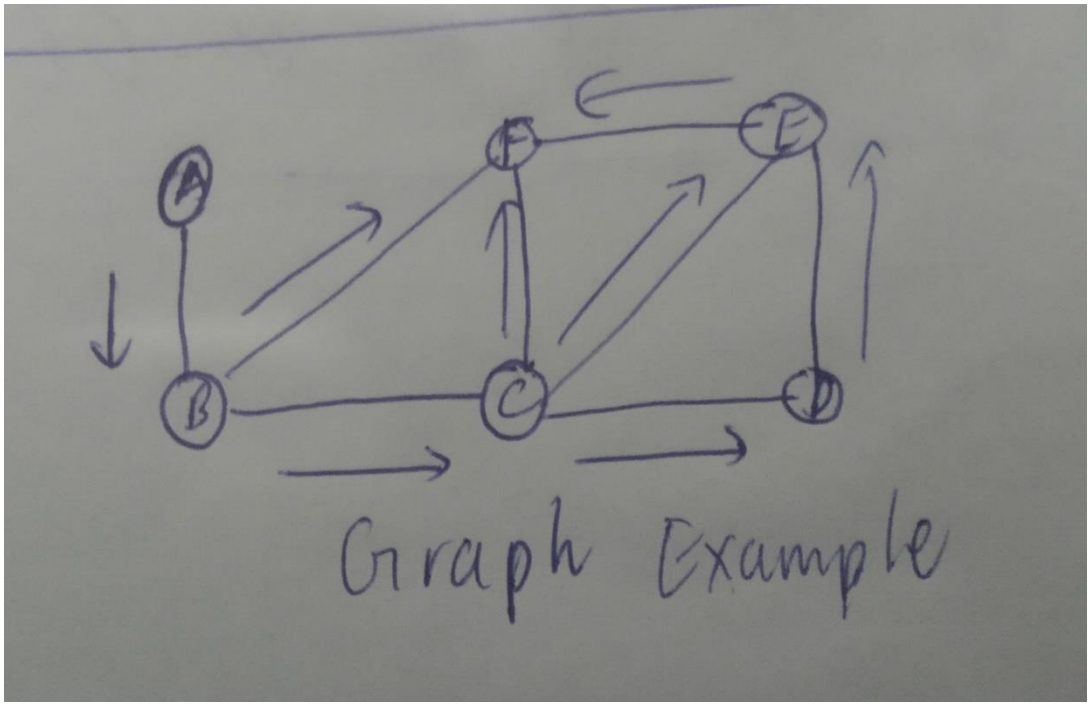
Roll_no. : **CS20B1044**

Date of Submission : **20-06-21**

Problem Title : **To find the shortest/fastest path from one place to another.**

My problem is finding the shortest path (As shorter path means reaching the location at a faster time it may also be called the fastest path) from one location (say A) to another location (say B) (where a location is connected to other locations). For simplicity of the problem I've decided to take (line connecting the locations connected to another location) edges are equal(weights are same (1) => Unweighted and Undirected graphs). I used graph data structure to store the location points and the edges as graph is a non-linear data structure, which consists of vertices (or nodes) connected by edges. My problem is to find the shortest paths between 2 nodes(locations).

The graph example below is used to explain my problem :



Suppose we want to travel from A to F, we have many paths possible :

(A-B-F, A-B-C-F,A-B-C-E-F,A-B-C-D-E-F), But we want the shortest path that saves time(fuel (decreases carbon emission), lives (if ambulance wanted a shortest route to reach a hospital) etc.).

So, the shortest route from the above graph is A-B-F.
Graph Related Data structures :
We use Adjacency Matrix/ List for knowing whether node is connected with any adjacent node. I'll use adjacency list as It takes less time.
Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. (TIme complexity O(v^2)).
For Adjacency lists, An array of linked lists is used. The size of the array is equal to the number of vertices. Let the array be an array[]. An entry array[i] represents the list of vertices adjacent to the i^{th} vertex. (Time complexity is O(e + v). (where e is number of edges and v is number of vertices).
For the Algorithm related Data structures :
We use a modified version of Breadth-first search in which we keep storing the predecessor of a given vertex while doing the breadth-first search(an algorithm used to traverse graphs). For the breadth-first search algorithm, we use queues(we use the FIFO principle to enqueue and dequeue, It contains vertices whose distance from the source node has been computed and their adjacent vertices are to be examined) and arrays(which store whether we visited or not, distance and previous(like the previous vertex which gave distance)).