Class - **Shortest\_path\_finder**

Variables :

* vector<vector<int>> all\_pair\_shortest\_distance; //this 2d array stores floyd warshall mini distance btw two vertices
* vector<vector<int>> all\_pair\_shortest\_distance\_intermediate\_vertex; //stores the final intermediate vertices btw every pair

Functions :

* Shortest\_path\_finder(map<int,vector<pair<int,int>>> &graph) //constructor that takes adjacency list of graph to perform Floyd Warshall
* Shortest\_path\_finder() //empty Constructor for object overloading
* vector<vector<int>> adj\_list\_to\_matrix(map<int,vector<pair<int,int>>> &graph) //this function just convertes adj\_list to adj matrix to apply floyd warshall
* vector<vector<int>> adj\_list\_to\_matrix(map<int,vector<pair<int,int>>> &graph) //this function just convertes adj\_list to adj matrix to apply floyd warshall
* void Floyd\_Warshall(map<int,vector<pair<int,int>>> &graph) //this function applies floyd warshall on the converted matrix
* vector<int> calc\_path(int v1,int v2) //returns the shorted path between two vertices as a vector
* vector<int> get\_path(int v1,int v2) //this function returns optimal path between two vertices
* vector<vector<int>> get\_traversal\_path(vector<vector<int>> &traversal\_order,int src) //this function takes the cluster and converts in to a path in that order using floyd warshall

Class - **Brute\_all\_paths**

Variables –

* vector<vector<int>> mini\_time\_traversal; //this 2d vector stores the cluster obtained to get mini time
* vector<vector<int>> mini\_distance\_traversal; //this 2d vector stores the cluster obtained to get mini distance
* vector<vector<int>> mini\_cost\_traversal; //this 2d vector stores the cluster obtained to get mini cost
* set<pair<set<vector<int>>,vector<int>>> Global\_Memoize; //this set of pair of (vector,vector) is used to memoize the brute to obtain all paths
* int no\_of\_Drivers; // stores no of drivers
* int mini\_cost=INT\_MAX; // stores minimum cost
* int mini\_time=INT\_MAX; //stores minimum time
* int mini\_distance=INT\_MAX; //stores minimum distance
* vector<vector<int>> path; //stores the currently obtained path in brute
* int penalty=100; //stores the currently set penalty
* vector<int> delivery\_deadline; //stores the delivery deadlines as a hash (index-> vertex no and value is the deadline for that index )
* vector<vector<int>> floyd\_Warshall\_cost; //2d vector obtained from shortest path finder class
* struct path\_desc //structure that contains the description of the currently bruted path

Functions :

* void Print\_cluster(vector<vector<int>> &cluster) //just a cluster printer function
* path\_desc calc\_path\_desc(int src) //calculates mini distance , time and cost of the currently bruted path
* void cal\_all\_paths(vector<int> &delivery\_locations,int index,int no\_of\_drivers,int src) //recursive function that brutes all possible paths
* void print\_path\_traversed\_by\_drivers(vector<vector<int>> &traversal\_order,int src,Shortest\_path\_finder &g1) //takes the cluster and prints the path followed by each driver
* void display(Shortest\_path\_finder &g1,int src) //display details of the best possible path
* Brute\_all\_paths(vector<vector<int>> &floyd\_Warshall\_costs,int src,vector<int> &delivery\_locations,int no\_of\_drivers,vector<int> &delivery\_deadlines) //constructor that calls all permutations of delivery location to be bruted

Class - **Greedy Cluster**

Variables –

* Shortest\_path\_finder g1; // to store a copy of already computed object of Shortest\_path\_finder class
* int src; //stores the source vertex
* vector<int> delivery\_locations; //delivery locations
* int no\_of\_drivers; //stores no of drivers
* vector<int> delivery\_deadlines; //stores the delivery deadlines as a hash (index-> vertex no and value is the deadline for that index )
* vector<int> cluster\_initializers; //stores the initial cluster initilizers
* vector<vector<int>> clusters; //stores the final cluster obtained through greedy
* vector<int> rem\_delivery\_locations; //store the delivery locations remaining after removing the initial cluster initialisers
* int penalty = 100; //stores the penalty for delay
* struct path\_desc //structure that contains the description of the current path

Functions –

* void calc\_path\_desc() //calculates mini distance , time and cost
* Greedy\_Cluster(Shortest\_path\_finder &g,int source,vector<int> &Delivery\_locations,int No\_of\_drivers,vector<int> &Delivery\_deadlines) //constructor that initializes all the variables
* void calc\_cluster\_initilizers() //this function finds the initial two cluster initializers
* void find\_rem\_cluster\_sources() //this function finds remaining cluster initializers
* void populate\_clusters() //this function assigns a cluster to all remaining delivery locations
* void display\_clusters() //just a cluster printer function
* void job\_scheduling\_on\_clusters() //sorts the clusters on the basis of their deadline
* void Display\_cluster\_path\_traversed() //prints the path from the cluster

**INPUT FORMAT :**

Input Format –

no of vertex , no\_of\_edge , src , no\_of\_drivers

v11 v12 w1

v21 v22 w2

.

.

vno\_of\_edge1 vno\_of\_edge2 wno\_of\_edge

no\_of\_delivery\_locations

d1 t1 // delivery vertex , deadline

d2 t2

d3 t3

.

.

d (no\_of\_delivery\_locations) , t(no\_of\_delivery\_locations)