ABSTRACT:

The shortest path is a very important thing in a business sector, especially in sending goods by land. One of the obstacles faced in the shipment of goods is the determination of the trajectory. Determining the best trajectory can minimize shipping costs and time is more efficient A designated algorithm or system is required to search for the shortest and the most optimal traffic path from various numbers of existing alternative pathways.

We tried to impement this idea by using DIJKSTRA Algorithm i.e. to find the minimum distance between any two city.

INTRODUCTION:

This project focuses on finding the shortest paths among city/place in any state by repeatedly combining the start node's nearest neighbor to implement Dijkstra algorithm. Node combination is used to find the shortest path among cities in state by deleting the node nearest to the start node.To implement our idea we created a dummy version for this which can be modified easily by collecting database containing the real distance among cities.This method is used by many navigating sites.We also created user friendly interface by using Python language which includes a best library called Tkinter.Using Tkinter we created this GUI.

In performing daily activities, transportation problem affects people's lives. To reach potential and remote areas, transportation network is needed in order to help develop the area.

We choose DJIKSTRA algorithm becausethe experiment results show that the three issues have been effectively resolved to find the shortest path. One-way separators will be used in the graph to cross it only once. These separators give divide-and-conquer solutions to find the shortest paths between two vertices/cities.

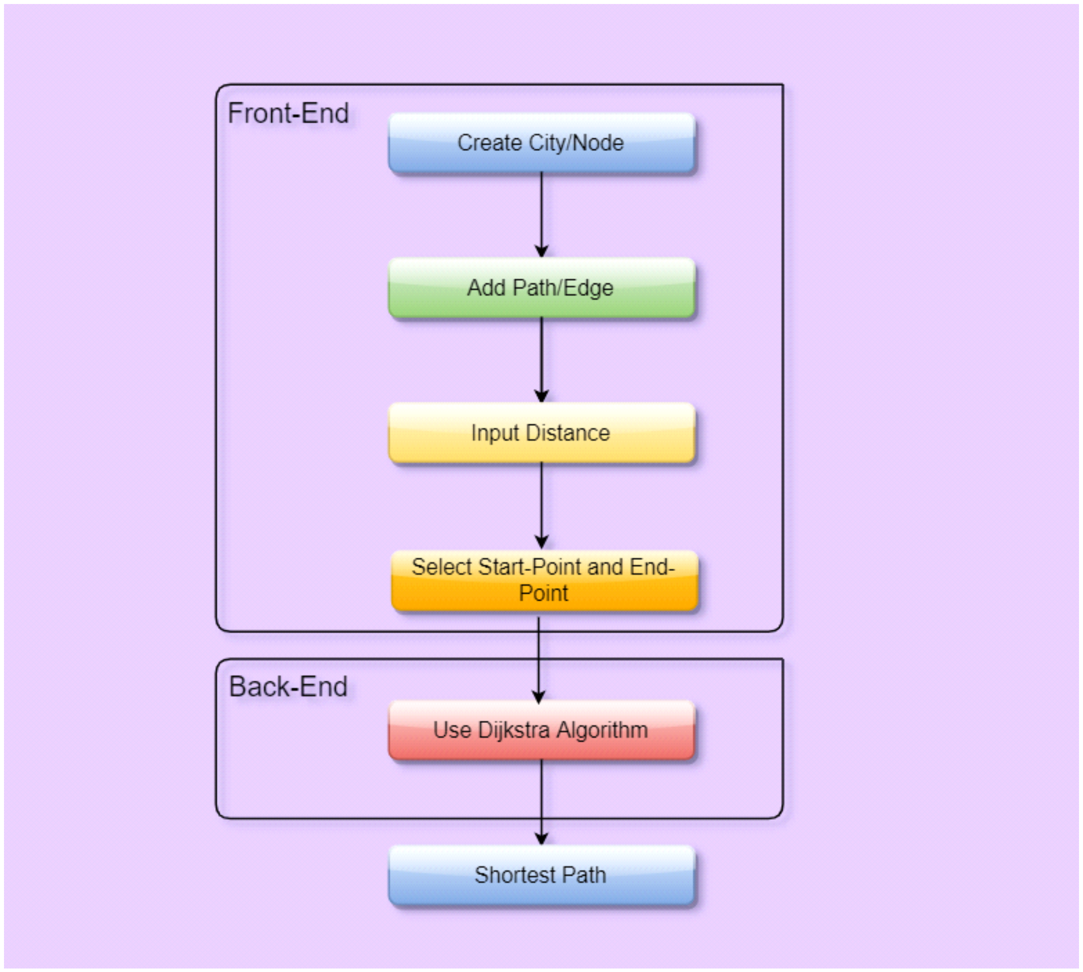
The successful implementation of algorithm in finding the shortest path on the real problem is a good point; therefore, the algorithm can be developed to solve many probem along with transportation network problem.

WORK DONE/METHODOLOGY:

The problem statement is **" The shortest path between two cities GUI** ".So to create this GUI we all are worked together and made an Graphical User Interface.The GUI mainly contains a square shaped canvas in which we can add cities/Nodes anywhere by moving the mouse and clicking anywhere on that canvas.

And after adding the Node/city we can just drag our mouse to make path/edge between two cities/Nodes.Also this GUI includes User controls label in which we added 3 buttons namely CLERA,INFO and SHORTEST PATH.Use of these will be described in the Result section.

BLOCK DIAGRAM



As we can see the block diagram which includes front-end part and also back-end part.The front-end part is basically user friendly part in which user just need to input the data.In our GUI the user has to make a node/city and also has to add the path/edge and he/she has to mention the distance between two cities.After that the last but not least you have to select whcih is starting-point and ending-point.These all things related to Front-end part.

The Front-End part is mainly devided among us and we are worked and created this GUI.This GUI is basically we done using python language.In python there is one famous library called tkinter which is basically used to create the GUI application.Using this(tkinter) library we created the front-end part.

Also there is another part which includes back-end also.Now this for this back-end part we have given some time to learn about Dijkstra algorithm.Because using Dijkstra algorithm only we calculated shortest path.Successfully we learnt and added this algorithm to our back-end part to get the shortest path.

At the last the output will show you the shortest path between the selected starting-point and ending-point.

ANALYSIS OF DIJKSTRA ALGORITHM:

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph.

Algorithm:

- Step 1: Set the distance to the source to 0 and the distance to the remaining vertices to infinity.
- Step 2: Set the current vertex to the source.
- Step 3: Flag the current vertex as visited.
- Step 4: For all vertices adjacent to the current vertex, set the distance from the source to the adjacent vertex equal to the minimum of its present distance and the sum of the weight of the edge from the current vertex to the adjacent vertex and the distance from the source to the current vertex.
- Step 5: From the set of unvisited vertices, arbitrarily set one as the new current vertex, provided that there exists an edge to it such that it is the minimum of all edges from a vertex in the set of visited vertices to a vertex in the set of unvisited vertices. To reiterate: The new current vertex must be unvisited and have a minimum weight edges from a visited vertex to it. This can be done trivially by looping through all visited vertices and all adjacent unvisited vertices to those visited vertices, keeping the vertex with the minimum weight edge connecting it.
- Step 6: Repeat steps 3-5 until all vertices are flagged as visited.

Pseudo Code:

```
dijkstra(v) :
    d[i] = inf for each vertex i
    d[v] = 0
    s = new empty set
    while s.size() < n
        x = inf
        u = -1
        for each i in V-s
            if x >= d[i]
                then x = d[i], u = i
        insert u into s
        for each i in adj[u]
            d[i] = min(d[i], d[u] + w(u,i))
```

Time Complexity Analysis:

Case-01:

The given graph G is represented as an adjacency matrix.

Priority queue Q is represented as an unordered list.

A[i,j] stores the information about edge (i,j).

Time taken for selecting i with the smallest dist is O(V).

For each neighbor of i, time taken for updating dist[j] is O(1) and there will be maximum V neighbors.

Time taken for each iteration of the loop is O(V) and one vertex is deleted from Q.

Thus, total time complexity becomes O(V^2).

Case-02:

The given graph G is represented as an adjacency list.

Priority queue Q is represented as a binary heap.

With adjacency list representation, all vertices of the graph can be traversed using BFS in O(V+E) time.

In min heap, operations like extract-min and decrease-key value takes O(logV) time.

So, overall time complexity becomes O(E+V) x O(logV) which is O((E + V) x logV) = O(ElogV)

This time complexity can be reduced to O(E+VlogV) using Fibonacci heap.

Worst case time complexity: O(E+V log V)

Average case time complexity: O(E+V log V)

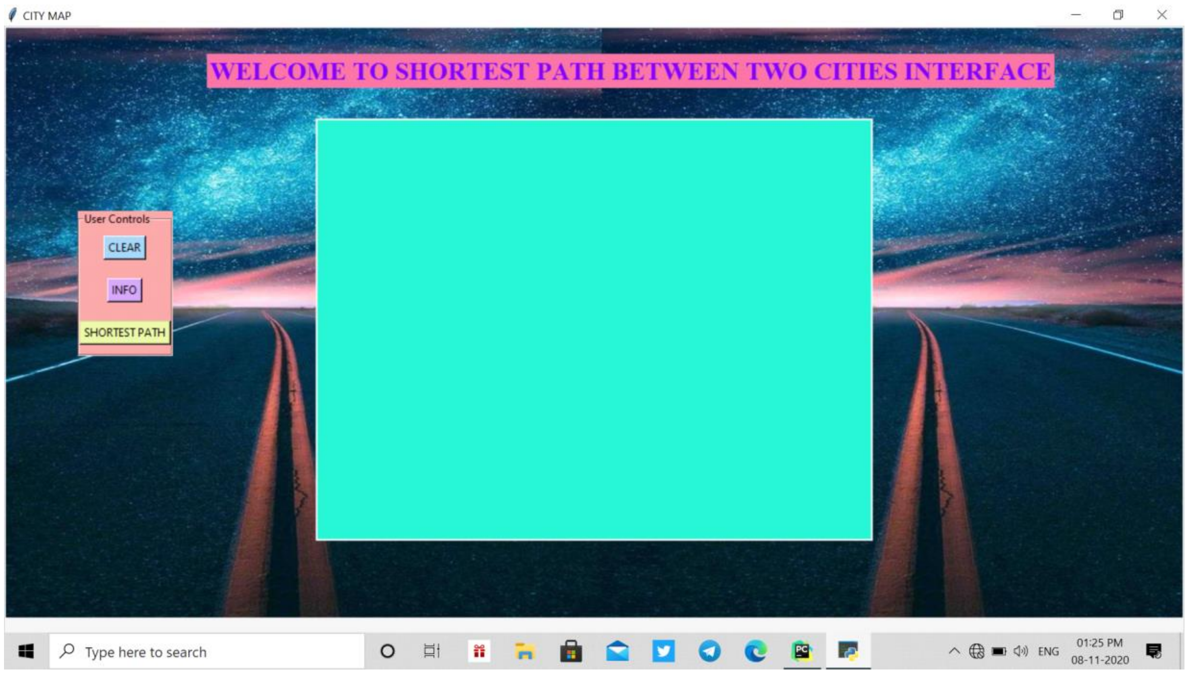
Best case time complexity: O(E+V log V)

Space complexity: O(V)

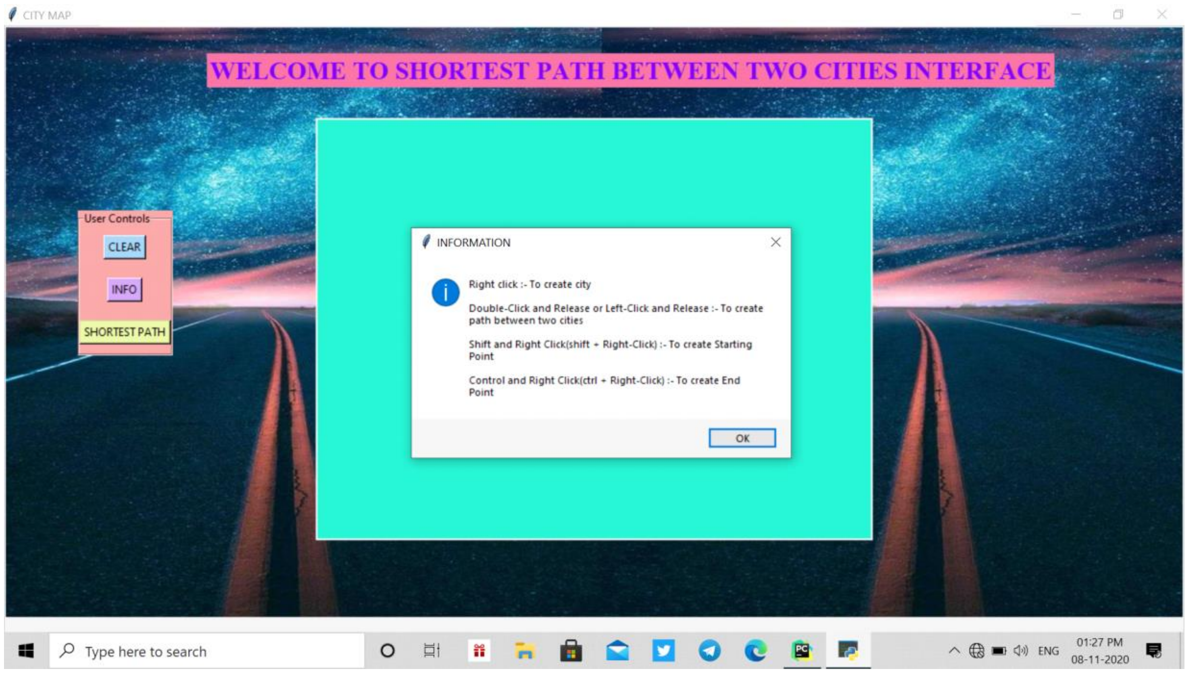
RESULT:

This section will includes all the snapshots related to our project output.

The main interface of **" The shortest path between two cities GUI "** is shown below.



The main interface includes a labelFrame in which a text content is written showing "**WELCOME TO SHORTEST PATH BETWEEN TWO CITIES INTERFACE".** Also it includes a square shaped box which is usually known as canvas on which user can add city ,edge etc.. There is one user Control LabelFrame which includes three buttons one is clear,second one info and third one is shortest path.



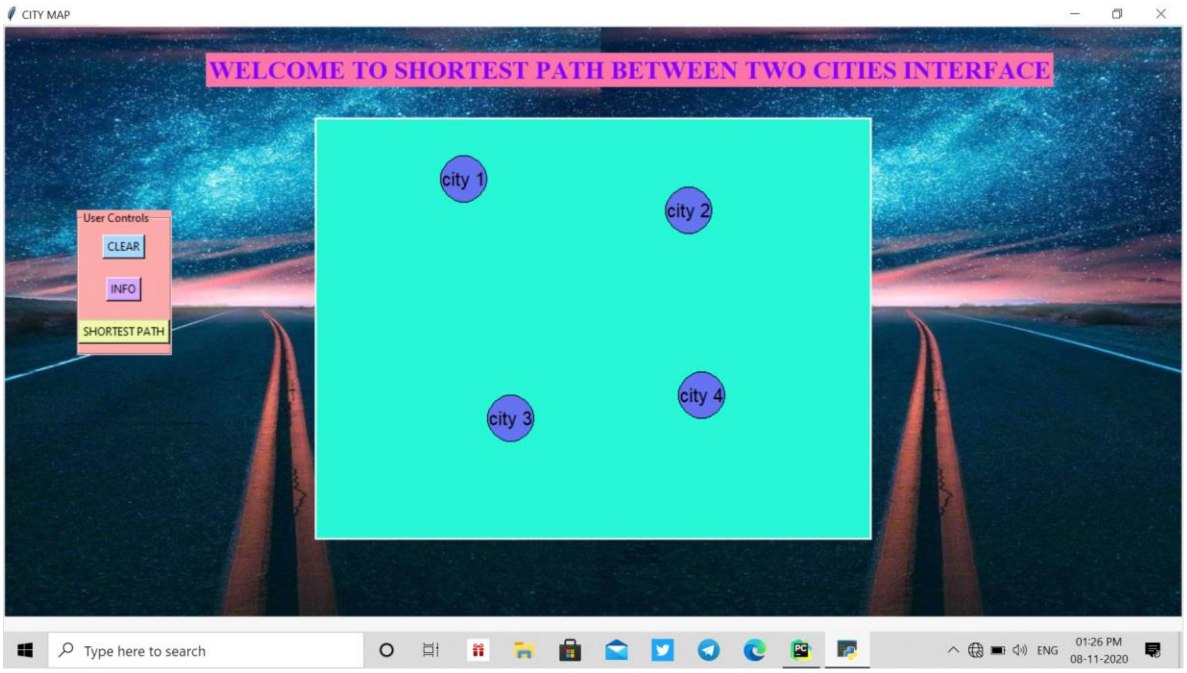
Right click :- To create city

Double-Click and Release or Left-Click and Release :- To create path between two cities

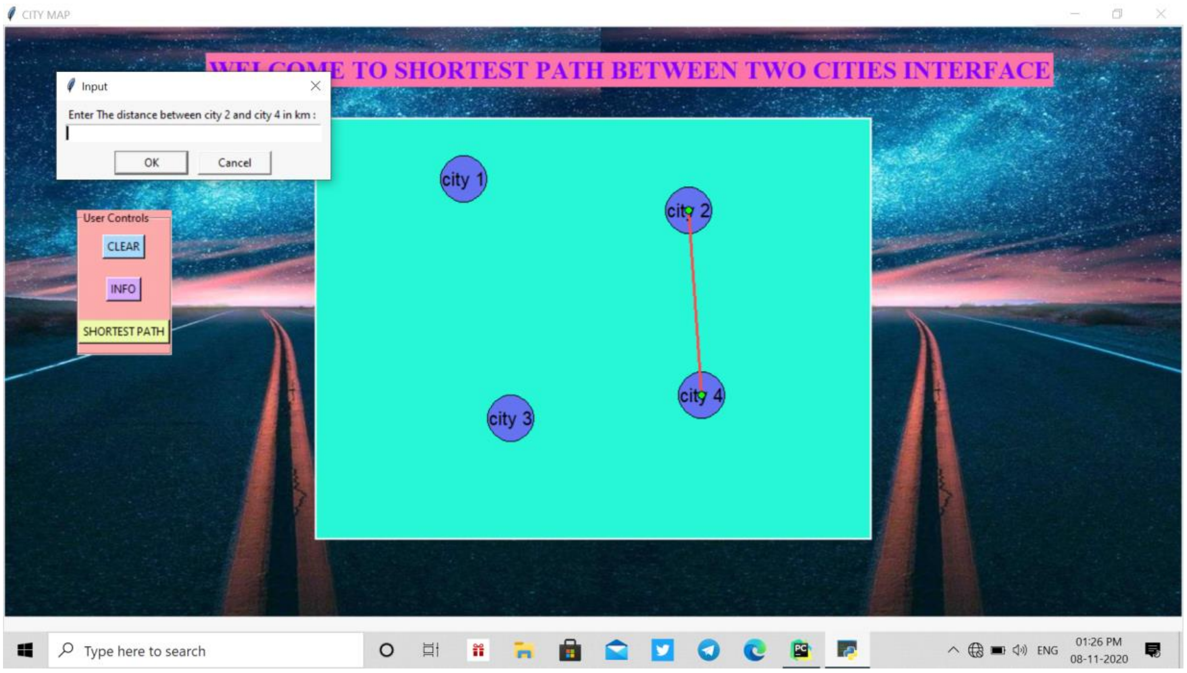
Shift and Right Click(shift + Right-Click) :- To create Starting Point

Control and Right Click(ctrl + Right-Click) :- To create End Point

By clicking on the INFO button which is in the User Control LabelFrame the information will be displayed which shows you how to create city ,how to add path,how to select start point and how to create end point.

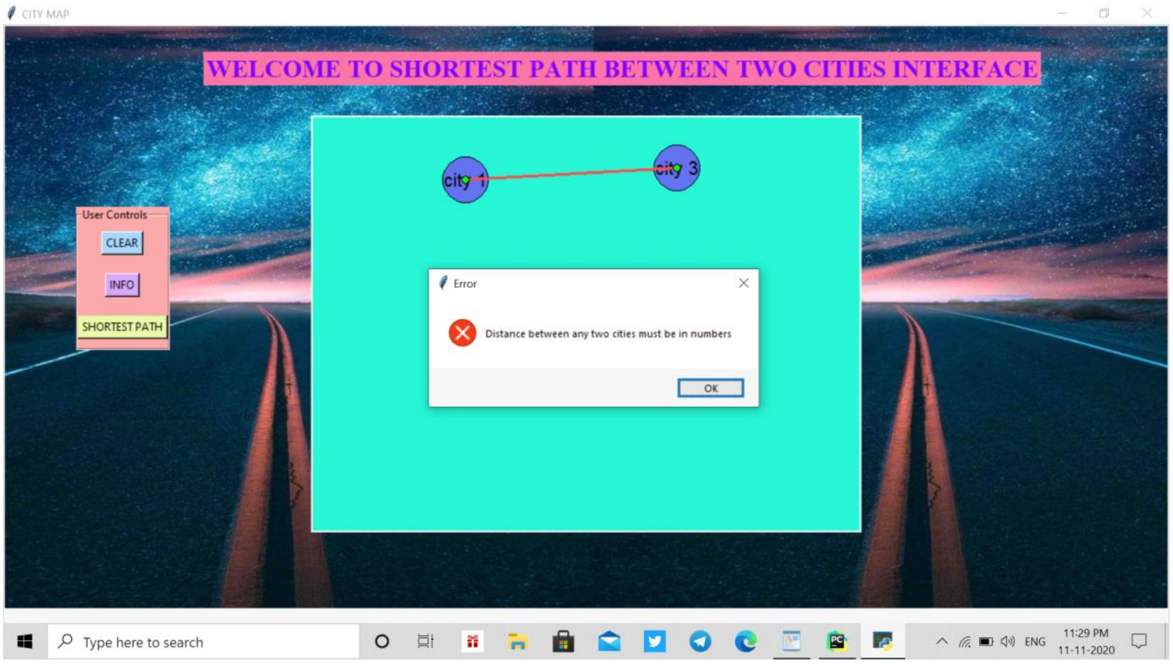


Above figure shows how the canvas looks after adding cities.By Right clicking on the canvas widget we can add city.Which will be in rounded shape like Node on that city number is written.



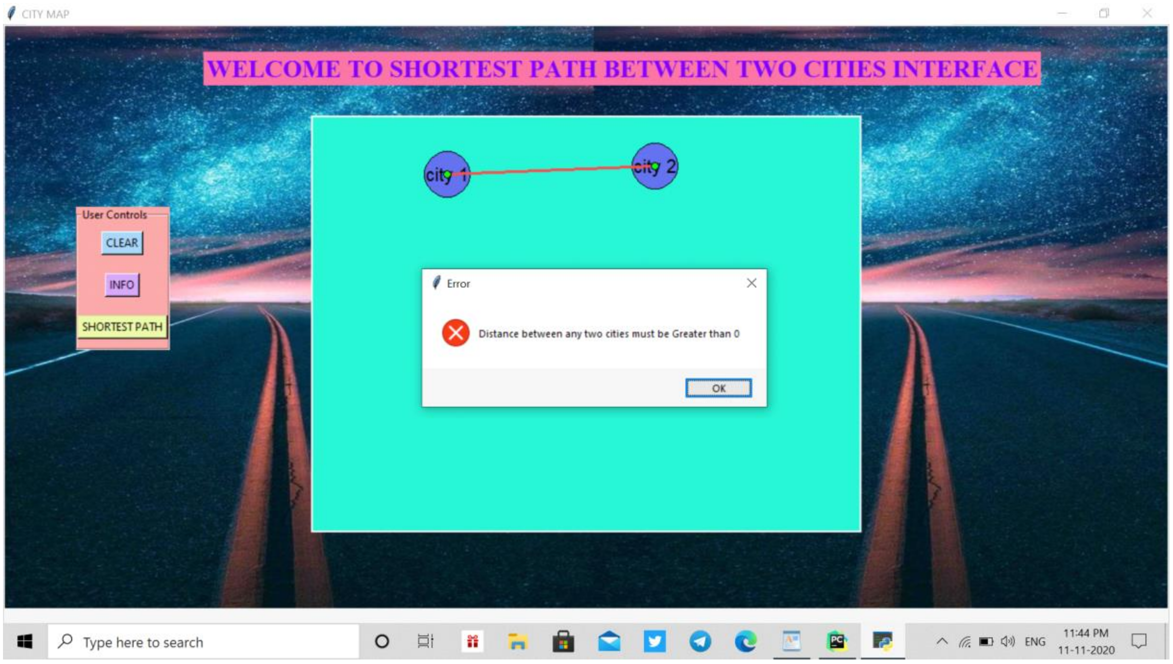
Enter The distance between city 2 and city 4 in km :

Now,we have to create a path between two cities to do that just right click on one city and drag upto another city after lifting up a window named as input will be displayed on the existing window.There the user has to enter the distance between the two cities which will be in kilometer.



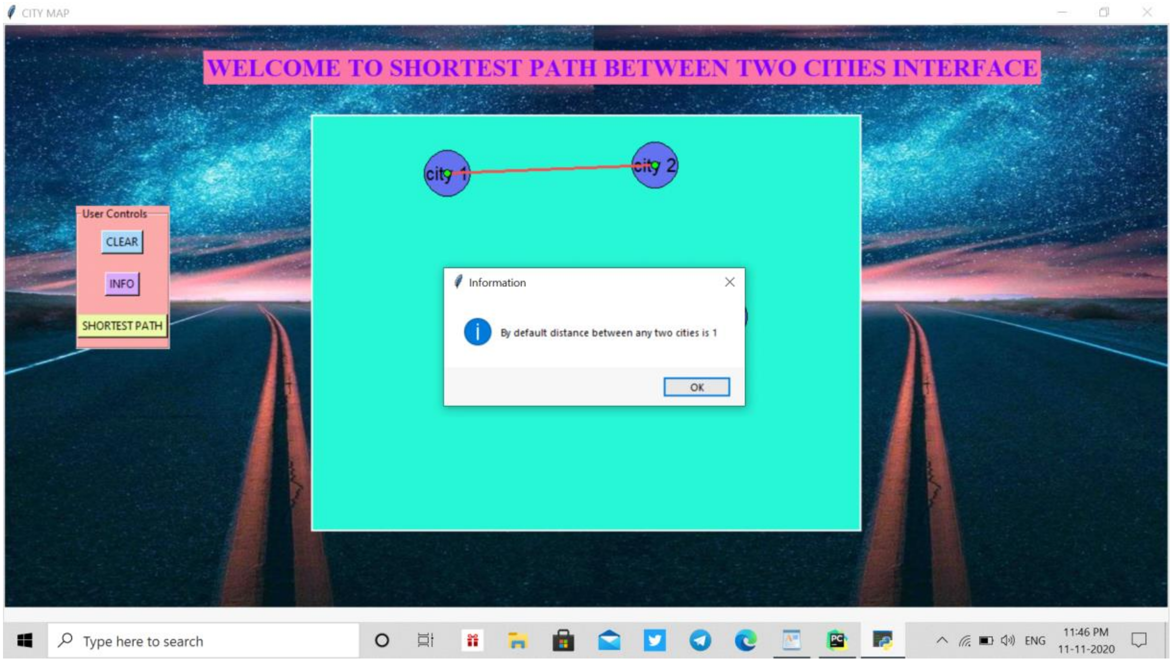
Distance between any two cities must be in numbers

If the entered distance between two cities is not a number then this pop up message will be displayed.



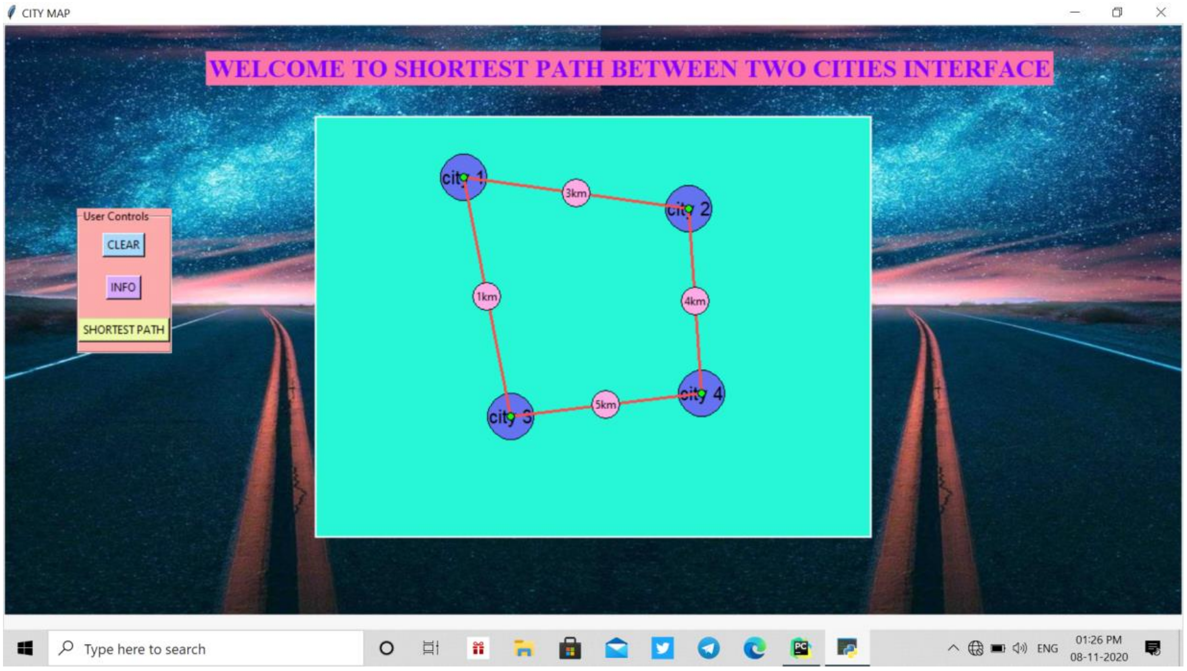
Distance between any two cities must be Greater than 0

If the entered distance between two cities is a number but less than or equal to 0 then this pop up message will be displayed.

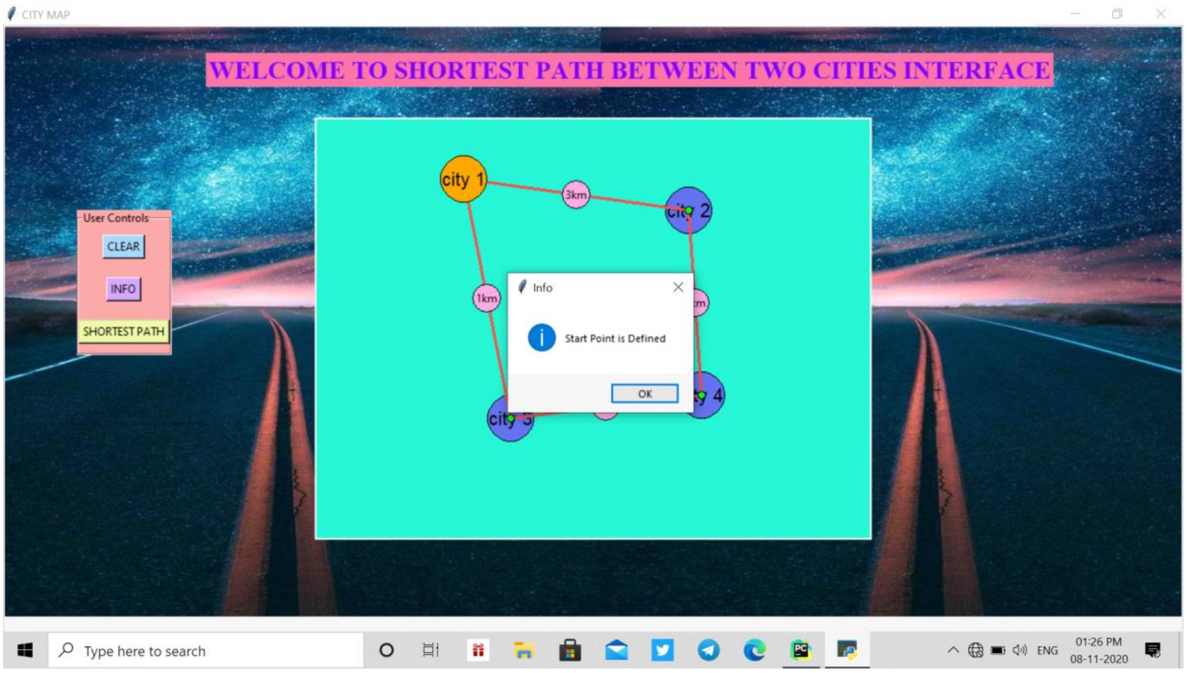


By default distance between any two cities is 1

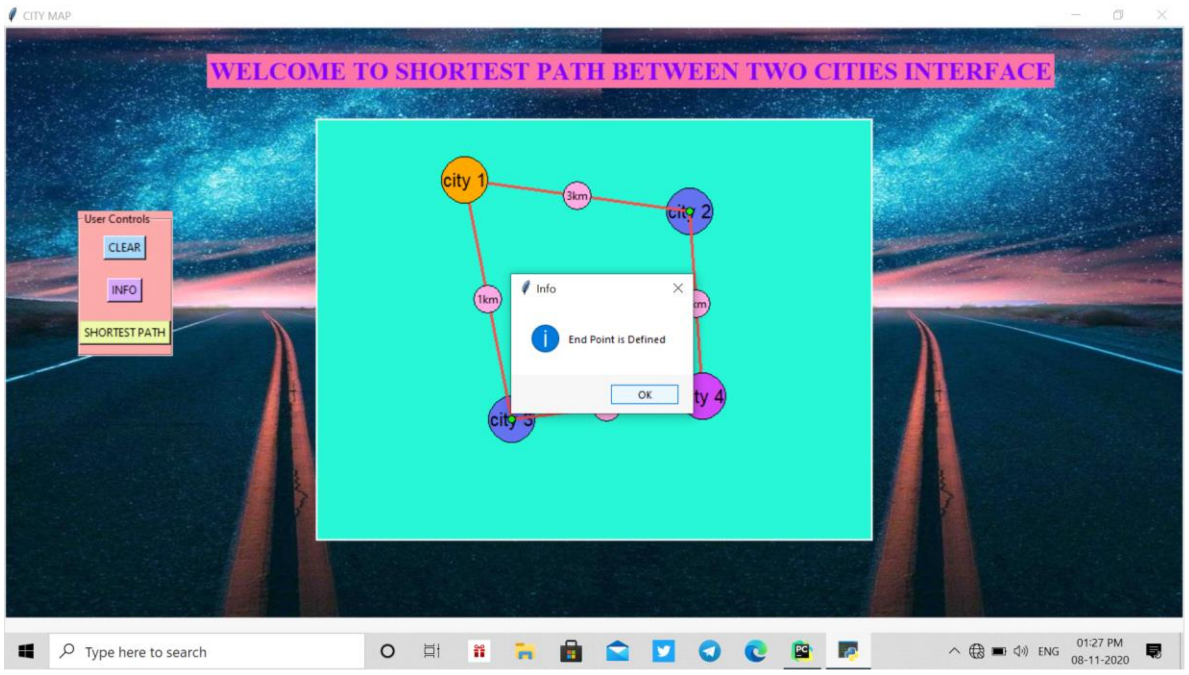
If bymistakely or intentially user not able to provide the distance between two cities then bydefault system will assign it to 1km.



After adding all the path and distances the interface looks like this.In which one can see the city number and path and distance between the two cities which is written exactly between the path in a round circle and that is in kilometer(km).

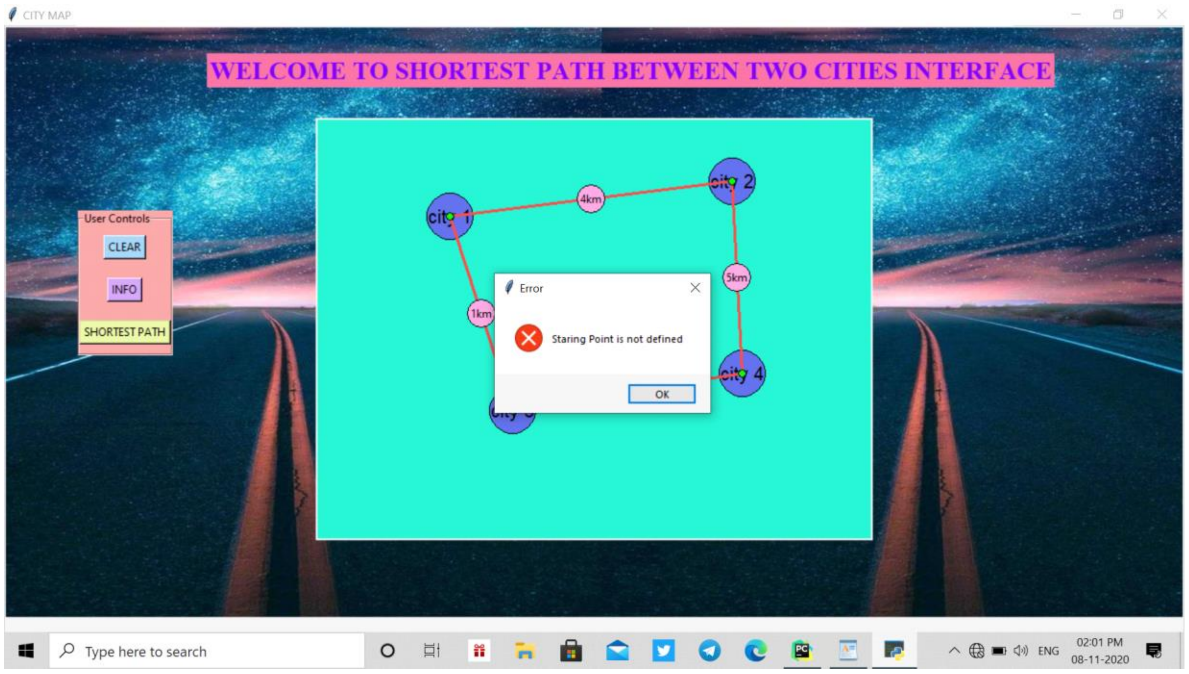


Now,the user must select a starting point.After selecting the starting point the program will display a popup message showing Start Point is defined.After selecting the starting point the city color will be changed from blue to orange.



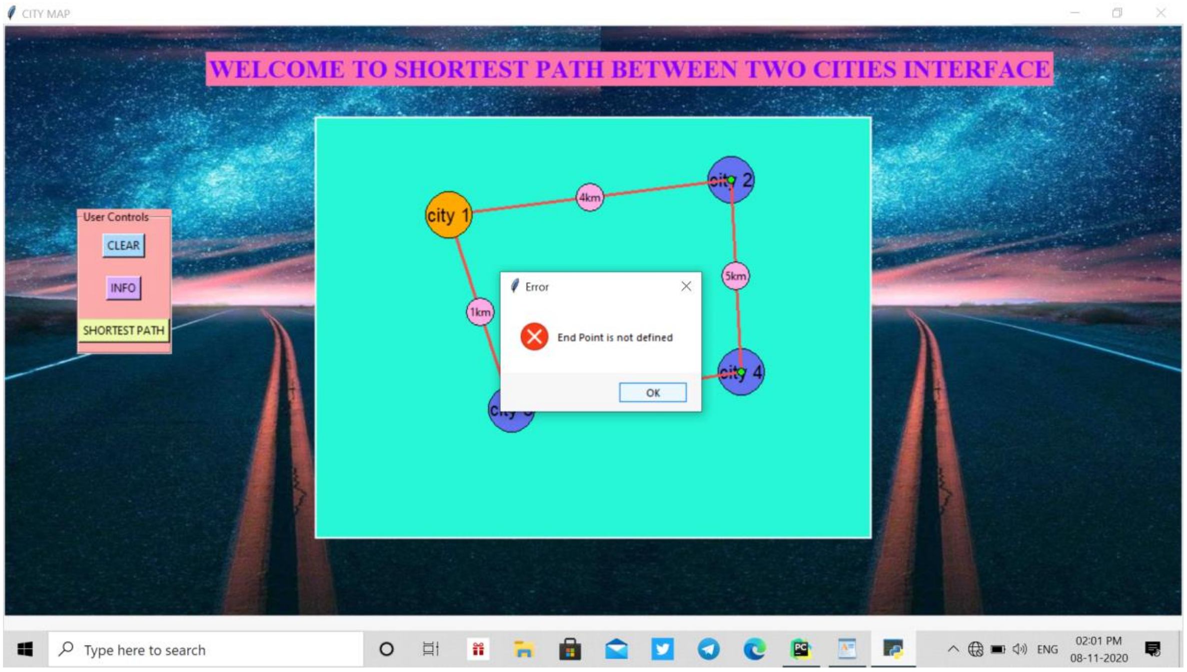
End Point is Defined

Next,the user must select a ending point.After selecting the ending point the program will display a popup message showing End Point is defined.After selecting the ending point the city color will be changed from blue to pink.



Staring Point is not defined

If user failed to select starting point then the error message will be displayed showing that starting point is not defined.

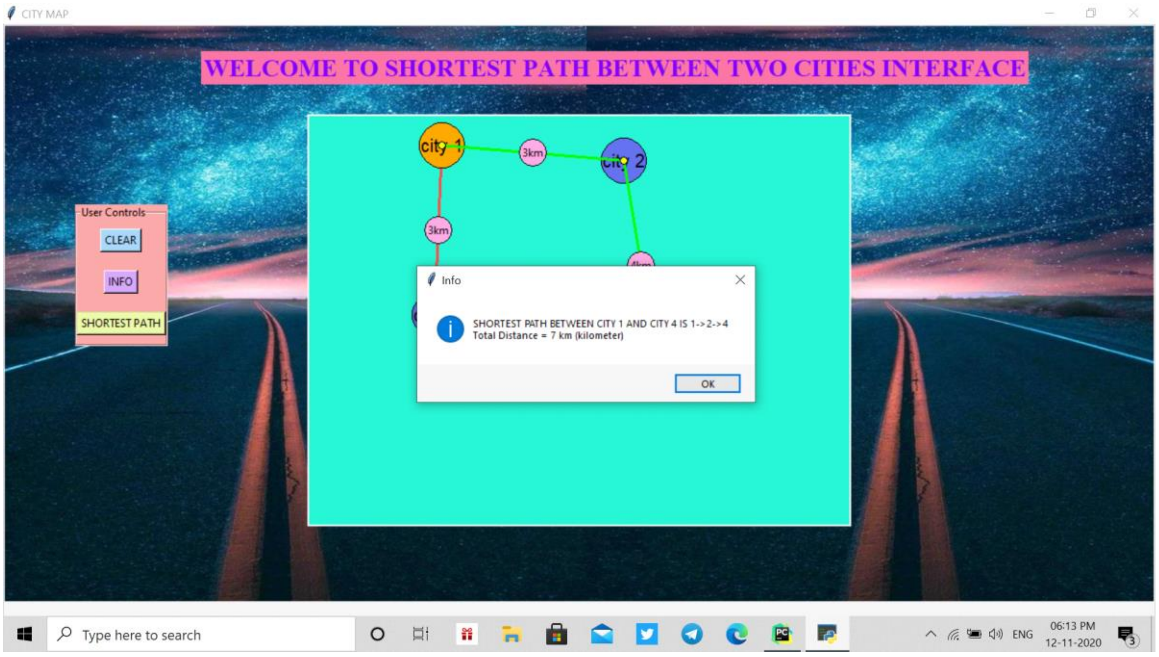


Error

End Point is not defined

OK

If startin point is defined but end point is not defined then the error messge will be displayed showing that End Point is not Defined.



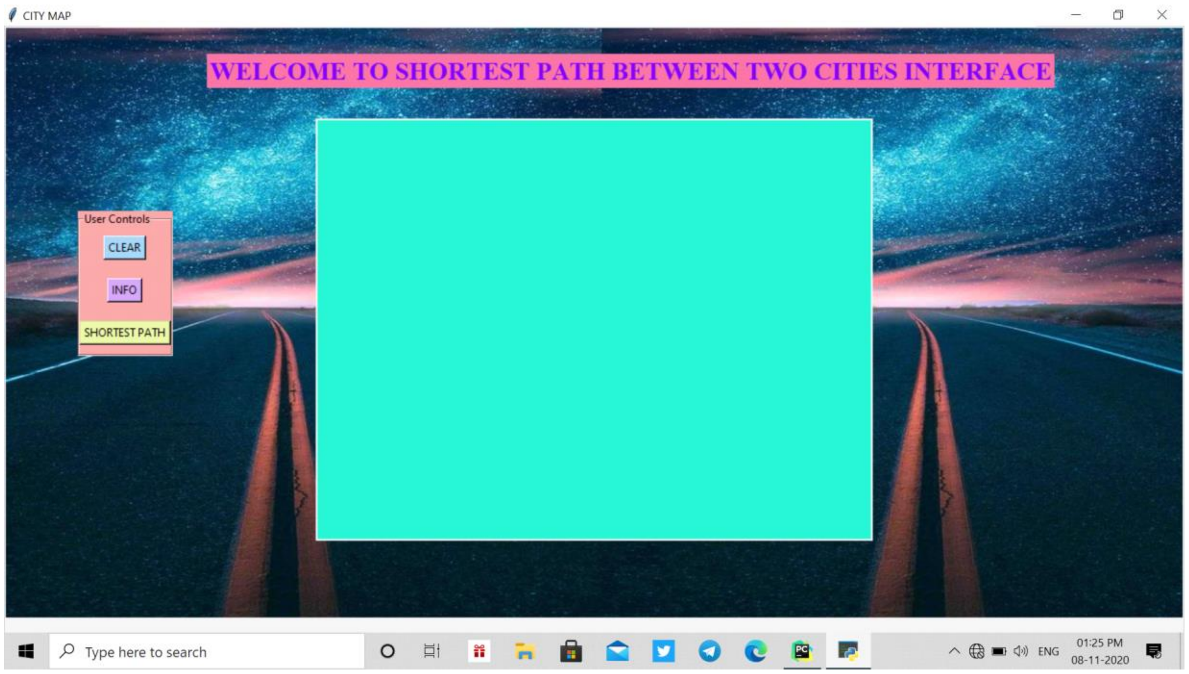
Info

SHORTEST PATH BETWEEN CITY 1 AND CITY 4 IS 1->2->4
Total Distance = 7 km (kilometer)

OK

Now,the user want the shortest path so the user need to click on the SHORTEST PATH button which is present in the User Control LabelFrame.After clicking on the button the shortest path will be shown in green color between starting and ending point.And also the popup message will be displayed which includes the path and the total distance requiresd to reach the destination.

After this there is another feature i.e The user can listen the path and the total distance need to be travel.Which will be done using one of the famous library in python named as pyttsx3.



To remove all the content from the canvas click on the CLEAR button which is present in the user control LabelFrame.

CONCLUSION:

While creating GUI and implementing this method we learned various new library,operations and their working.Also we learnt about the famous DJIKSTRA algorithm i.e about its working and implementation.As per the expectations this idea fulfilled our objectives but this is only dummy version. In future we will try to plot this on big stage i.e. by collecting data of actual distances.

We will actually start setting this on a big stage by implementing it on remote or rural areas because of unavailability of such ideas in this areas.

Also in future research , expected to be developed by adding the travel time and traffic conditions at the actual time, and can compare with other shorted path methods.