Assignment 1: Problem 2
ROAD MAP

CITY_MAP :
It is my global variable (dictionary) in the program which stores all the information about the cities like ‘connected cities’, ‘isVisited’, ‘distance’, etc. Refer code for the detailed explanation of all the entries in the dictionary.

State Space: The state space consists of all the cities and intersections of highways in the city-gps.txt and road-segments.txt.

Successor function: The successor function returns the connected cities as the successors of the current city.

Edge weights: The edge weights vary for the cost functions as follows.

	bfs	**dfs**	**ids**	**unform**	**astar**
segments	segments	segments	segments	segments	segments
distance	segments	segments	segments	distance	distance
time	segments	segments	segments	time	time

Here the segments is the edge connecting two cities. The two neighbouring cities are connected by the single segment.
Hence BFS algorithm will always find the optimal solution when the cost function is ‘segments’

Goal State: The goal state is the destination city.

Initial State: The initial state is the source city.

Search techniques:

BFS: The BFS algorithm is straightforward and travels the successors breadth-wise. If the cost function is *segments* then the BFS algorithm will always be optimal. The BFS algorithm does not take into account the other cost functions like distance and time.

DFS: The DFS algorithm travels the successors depth wise. Here also the algorithm only takes into account the segments as the cost function. The DFS algorithm is not optimal.

IDS: IDS iteratively calls the DFS algorithm with depths ranging from 1 to the total number of nodes in the city_map.

Uniform Search: The uniform search algorithm is the priority queue implementation. It takes expands the least cost node (here city) and adds its successors to the fringe.
The cost of the city which is already in the fringe is updated if the algorithm finds the better route to that particular city.

Here the cost function, f(s) = g(s); where g(s) is the actual cost from the source to the current city.

The search works for all the cost functions and gives the optimal solution for all of them

AStar Search: The astar search technique always gives the optimal solution given the heuristic is consistent and admissible.

But in our case, the data is noisy and hence the heuristic cannot be consistent. There are some routes where the latitude-longitude distance is larger than the uniform cost between them. (which is the optimal cost).
Hence the heuristic is not consistent owing to which the astar search does not always guarantee the optimality of the solution.

Astar prioritizes on the cost function, f(s) = g(s) + h(s)

As the h(s) is not consistent, the astar results may vary slightly from the optimal solution.

Heuristic Function: The heuristic function takes into consideration the noisy data to some extent.
The heuristic cost for any given city is calculated as follows:
cityA - for which the heuristic is to be estimated till the goal city
Consider different cases:

CityA has lat-long and destination has lat-long:

- It directly gives the euclidean distance between the two cities .

  ```
  eucledian_distance =
  np.sqrt((np.square(69*(float(city_map[source]['lat']) -
  float(city_map[destination]['lat']))) +
  (np.square(55*(float(city_map[source]['long']) - float(city_map[destination]['long'])))))
  ```

- One of the cities doesn't have the lat-long pair.
  In this case, the algorithm will go deep towards the destination to find the city with the lat-long. It will return the city which has the minimum total cost including the uniform distance from cityA to this city plus the euclidian distance from this city to the destination.

- Both of the cities doesn’t have lat-long: In this case the neighbouring city of the cityA is fed to the algorithm and the direct distance between these cities is subtracted from the Euclidean distance.

Note: To overcome for the noisy data I have multiplied the euclidean distance by the factor of 0.5 so that the heuristic function stays admissible because it will always underestimate.