Problem 1

Steps to run -

1) Ensure that the input files - road_segments.txt and city_gps.txt are in the same folder as the problem1_solution.py program file.

2) Use the command line format - python problem1_solution.py [Bloomington,_Indiana] [Chenoa,_Illinois] [distance] [idfs]

Q1.Which search algorithm seems to work best for each routing options?
Ans. A* seems to work best for each routing option according to our experiments because of the following reasons -

1) It gives an optimized goal path
2) Least run-time for different routing options
3) Least fringe memory (fringe is maintained as a priority queue)

Q2.Which algorithm is fastest in terms of the amount of computation time required by your program, and by how much, according to your experiments?
Ans. A* search seems to be the fastest algorithm as it keeps track of the lowest cost of every expanded path and switches paths based on the same measure.

Following are some run-times on the input:

python problem1_solution_github_synced.py [Chicago,_Illinois] [Schaumburg,_Illinois] [routing-option] [routing- algorithm]
where **routing-option** is one of **distance**, **segments**, **time** or **scenic** and **routing-algorithm** is one of **bfs**, **dfs**, **ids** or **astar**

	BFS	DFS	IDS	A-Star
Distance	0.200127840042	26.8855080605	0.47216796875	0.141630887985
Time	0.204982042313	25.8666880131	0.475928068161	0.17330789566
Segments	0.191943883896	25.7925200462	0.493514060974	0.173902988434
Scenic	0.194792032242	24.9982881546	0.472812891006	0.17181801796

Comparison of run-times for different algorithms with different routing options (all units in hours)

Q3.Which algorithm requires the least memory, and by how much, according to your experiments?
Ans. According to our experiments, the memory consumption (found by calculating maximum fringe size) is in the order - DFS > BFS > IDS > A-star

Q4.Which heuristic function did you use, how good is it, and how might you make it better?
Ans. We calculated the geographical distance between the current city and the destination city from the given latitude and longitude from city_gps.txt and in case when this information was missing from city_gps file for highway intersections we calculated the nearest towns to the highway intersection which is closest to the goal and substituted its latitude and longitude calculated (geographical) distance plus the distance from highway intersection to that nearest town as the current cost.

This heuristic works good for distance, scenic and segment routing options. For the time routing option we divided the lat-long calculated distance by the maximum speed limit to calculate the heuristic.

This heuristic function can be made better by keeping track of a metalevel-state-space and using a metalevel searching algorithm on the state space to avoid exploring unpromising paths.

Problem 2

Manhattan distance, was used as a heuristic for a given tile from its current position to its goal state position. This heuristic works well for most problem states.

Another possible heuristic function is the number of misplaced tiles on the board. Both the aforementioned heuristics are admissible, but the value given by the heuristic function of Manhattan distance, is at least as high the value given by the latter, which makes it the better heuristic function between the two.

Yet another possible heuristic function is a sort of a modified Manhattan distance, which takes into consideration the presence of the moves added to modified 15-puzzle problem. This makes the heuristic more accurate, but does not always make it stay consistent. It may improve the overall running time of the algorithm and help the program come across the solutions quicker.