**About this Project**

In this project, you will **build** a route-planning algorithm like the one used in Google Maps to calculate the shortest path between two points on a map.

# Project Instructions

The next section is a Jupyter Notebook which contains all the instructions for the project. We recommend that you have two files open in separate tabs while working on this project: project\_notebook.ipynb. Further instructions and tips for the functions to implement can be found in the notebook!

A screenshot of a cell phone

Description automatically generated

Further instructions and tips for the functions to implement can be found in the notebook!

## Evaluation

Once you have completed your project, use the [Project Rubric](https://review.udacity.com/#!/rubrics/1210/view) to review the project. If you have covered all of the points in the rubric, then you are ready to submit! If you see room for improvement in any category in which you do not meet specifications, keep working!

Your project will be evaluated by a Udacity reviewer according to the same Project Rubric. Your project must "meet specifications" in each category in order for your submission to pass.

When you're done with the project please double check any **outside sources have been correctly cited** before submitting. You should submit **from the Jupyter notebook** by clicking the big Submit button in the bottom right.

## Requires Changes

### **1 specification requires changes**

Great job. However, you need to correct all the observations I made.  
Don't be discouraged by these little inconveniences. We are here to help you to become a better computer scientist.  
Thanks for your time and efforts. Keep working hard and stay motivated. Good luck for your next submission.

Don't hesitate to ask questions in Knowledge:  
[https://knowledge.udacity.com](https://knowledge.udacity.com/)  
Here you can interact with many mentors who will help you to solve your problems with this project.

## Correctness

**Running test.py shows "all tests pass".**

All tests passed. Congrats.

**The student implements all required methods.**

All required methods were implemented.

**The heuristic function used to estimate the distance between two intersections is guaranteed to return a distance which is less than or equal to the true path length between the intersections.**

You used the Euclidean distance, which is good enough for this project. Because the Euclidean distance is an admissible heuristic for this problem.

**Student answered all question correctly.**

Your answers are correct. Thank you.

## Choice and Usage of Data Structures

**Code avoids obvious inappropriate use of lists and takes advantage of the performance improvement afforded by sets / dictionaries where appropriate. For example, a data structure like the "open\_set" on which membership checks are frequently performed (e.g. if node in open\_set) should not be a list.**

Very good. Sets, lists, and dictionaries are good enough for this project.  
For further improvements, you can use priority queues in Python: The module heapq.

Priority queues or heap queues <https://docs.python.org/3.0/library/heapq.html> allow you to insert elements into a queue in a sorted way. You can configure the way elements are sorted. In this project, you can sort elements by their value f = g + h. So, when you consult the next node to explore, you pick the most promising node in O(1) time.

You can consult these links in order to know better the time complexities of algorithms and data structures:  
<https://www.bigocheatsheet.com/>  
<https://wiki.python.org/moin/TimeComplexity>

**This item is a judgement call. Student code doesn't need to be perfect but it should avoid big performance degrading issues like...**

**...unnecessary duplication of lists  
...looping through a large set or dictionary when a single constant-time lookup is possible**

I'm sorry. Your function has a time greater than linear O(n).  
Why? Because loops and the function min() have a time of O(n) and access to dictionaries has a time of O(log n).

def get\_current\_node(self):

""" Returns the node in the open set with the lowest value of f(node)."""

# Return the node in the open set with the lowest value of f(node).

dict\_of\_f\_scores = {}

for node in self.openSet:

dict\_of\_f\_scores[node] = self.calculate\_fscore(node)

return min(dict\_of\_f\_scores, key = dict\_of\_f\_scores.get)

This function is more concise and has a time of just O(n):

def get\_current\_node(self):

return min(self.openSet, key = self.calculate\_fscore)

Please correct it.

You can consult these links in order to know better the time complexities of algorithms and data structures:  
<https://www.bigocheatsheet.com/>  
<https://wiki.python.org/moin/TimeComplexity>

A screenshot of a social media post

Description automatically generated

## Meets Specifications

Dear student,

well done, I see all the effort you have put into the project!

You implemented the A-Star search algorithm and all methods successfully and use a correct heuristic.

You also pass all tests.

Your code is very lean and efficient.

Good job providing correct answers to each question!

You passed with flying colors.

Keep up the good work!

## Correctness

**Running test.py shows "all tests pass".**

I ran your code locally and it passed all tests, great!

All tests pass! Congratulations!

**The student implements all required methods.**

You implemented all methods correctly and your submission is always finding the correct path.

**The heuristic function used to estimate the distance between two intersections is guaranteed to return a distance which is less than or equal to the true path length between the intersections.**

You used the Euclidian distance for the heuristic here, which guarantees to return the minimum distance between two intersections, great!

**Student answered all question correctly.**

Well done choosing the correct answers in the multiple choice questions.

Good job providing correct answers to each question!

## Choice and Usage of Data Structures

**Code avoids obvious inappropriate use of lists and takes advantage of the performance improvement afforded by sets / dictionaries where appropriate. For example, a data structure like the "open\_set" on which membership checks are frequently performed (e.g. if node in open\_set) should not be a list.**

Good use of sets and dictionaries here to take advantage of their membership test speed over lists!

Hint: Why not use a defaultdict for gScores and fScores? Check it out this example:

from collections import defaultdict

fScore = defaultdict(lambda: float("inf"))

Don't forget to add the start node!

Otherwise a slightly shorter way to write your solution to initialize a dict with infinite values would be for example:

fScore = {road : float("inf") for road in list(self.map.intersections.keys())}

Do not forget to add the start node value afterwards!

**This item is a judgement call. Student code doesn't need to be perfect but it should avoid big performance degrading issues like...**

**...unnecessary duplication of lists  
...looping through a large set or dictionary when a single constant-time lookup is possible**

I could not find any performance issues.

Your code is lean and efficient, well done.

All tests paths were found near instantly.

Well done and good update to use the self.calculate\_fscore directly as a key in your mean function to achieve O(n) as complexity!