UDACITY

< Return to Classroom

Build an OpenStreetMap Route Planner

REVIEW

CODE REVIEW 3

HISTORY

Meets Specifications

You did a great job implementing the A* search algorithm in C++! Your code is structured well, builds without any problems and passes all of the required unit tests! I have left some remarks in the code review on how you could further improve your code in some areas. On to the next project!

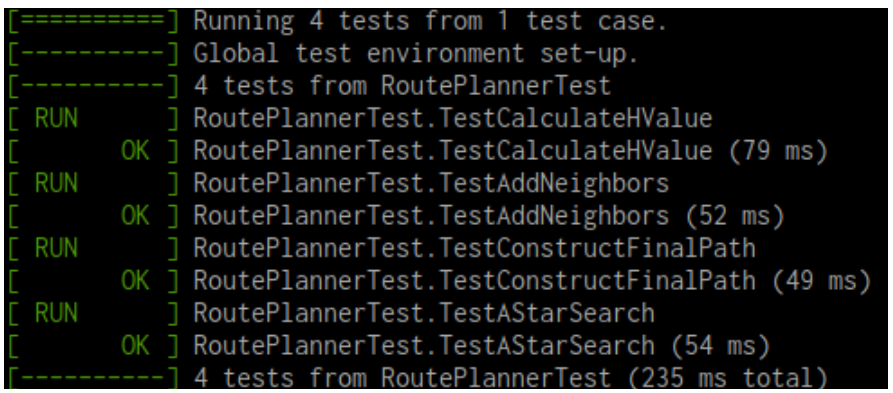
Compiling and Testing

The project code must compile without errors using `cmake` and `make`.

Your project builds without problems!

Code must pass tests that are built with the `./test` executable from the `build` directory of the project. See the project submission instructions for more details on how to run the tests.

You implementation passes all of the unit tests!



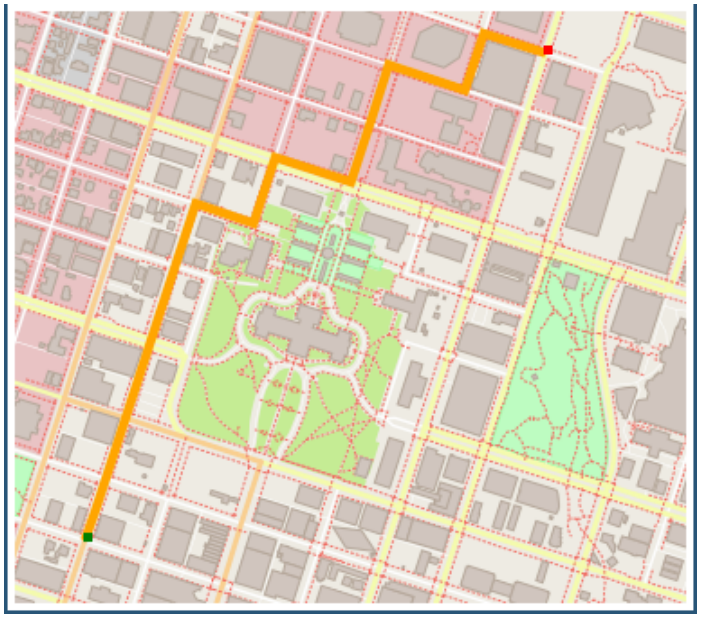
```
[----------] Global test environment tear-down
[==========] 4 tests from 1 test case ran. (235 ms total)
[  PASSED  ] 4 tests.
```

User Input

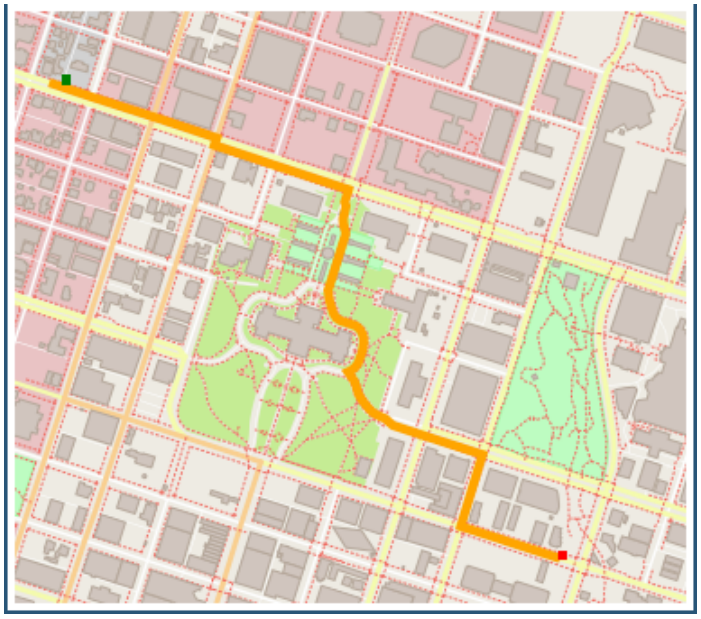
A user running the project should be able to input values between 0 and 100 for the start x, start y, end x, and end y coordinates of the search, and the project should find a path between the points.

Your implementation of the A* search algorithm successfully finds a path between two user-specified points in the map!

(10,10) to (90,90):



(10,90) to (90,10):



The coordinate (0, 0) should roughly correspond with the lower left corner of the map, and (100, 100) with the upper right.

Note that for some inputs, the nodes might be slightly off the edges of the map, and this is fine.

The user input is correctly converted into map coordinates! Well done!

Code Efficiency

Your code does not need to sacrifice comprehension, stability, or robustness for speed. However, you should maintain good and efficient coding practices when writing your functions.

Here are some things to avoid. This is not a complete list, but there are a few examples of inefficiencies.

- Running the exact same calculation repeatedly when you can run it once, store the value and then reuse the value later.
- Loops that run too many times.
- Creating unnecessarily complex data structures when simpler structures work equivalently.
- Unnecessary control flow checks.

Your code is very readable and you avoid unnecessary calculations.

DOWNLOAD PROJECT

3 CODE REVIEW COMMENTS ›