CIS 422 Software Methodology 1

Project <1> Report Collection

Submitted to:

Prof. Michal Young

Author:

John Zhou

Yiran Liu

Report

Introduction

This project is to convert coordinates from user's inputs into navigating information including road name, travel distance, and turning direction. The input should include manual input and .gpx file input, and the user interface should be a webpage that allows users to enter or drag files to upload. But because of something that happened during the project, our program only performs a command-line UI and only allows .gpx file as standard input.

Background

The first thing is how to convert coordinates including latitude and longitude to places names such as 123 main rd, Eugene, OR, 97401. Thus, we have to use Geocoding services which is a server that we can use to request coordinates from giving place names to the server. Vice versa, we can use reverse geocoding to get place names by giving coordinates.

Second, users' inputs are limited to only coordinates, we also have to convert .gpx files to pairs of coordinates so that we can send it to the server. There is a package in python called *gpxpy* that allows us to read .gox files and returns pairs of coordinates. Therefore, we choose to write this project in python because python is good at handling web requests, and because of this package that we can easily extract data.

Since we can get coordinates from inputs, we have to convert information into road names, travel distances, and turning directions. Thus, we have to group coordinates that on the same street to generate these data. There are two algorithms that we can use. The first is doing binary research on the coordinates to get the start coordinate and end coordinates of a road, and the second is use the webserver API to generate the routing from the first coordinate and the last coordinate we got from the input, and differentiate the routing with the rest of coordinates. It is best if the routing includes the rest of the coordinates so we can use it directly, otherwise, we have to subdivide the coordinates list and do the same thing again. Considering the API key limits, we chose to use binary research on the coordinates.

Implementation

We considered building the project from the small parts to the big picture. We have separated the project into three parts, which are 1) extracting data from .gpx files, return a list of pairs of coordinates, 2) doing BR on the list of coordinates, return a list of pairs of start coordinate and end coordinate of each road, and 3) use the returned list to generate travel distance and turning directions.

- Extracting data from .gpx files. by John Zhou

  There is a python library called *gpxpy* that allows us to extract data directly from .gpx files. We can simply use the build-in function *gpxpy.parse(<gpx file>)* to get latitude and longitude from ./gpx files. Append the pair of *(lon, lat)* into the list, and return the list in the end. Also, use the API of /geoservices.tamu.edu/ to request the places names.

  List = [(lat, lon), (lat, lon), …]

- Group coordinates on the same street. By Yiran Liu

  In this part, we have to traversal the whole list of coordinates to find the start coordinate and end coordinate of each road. But the traversal is not efficient since there are a lot of pairs in the list. We considered using

binary research on the coordinate list. The return list in this part should be a 2D list that each index contains at least two coordinates.

Recursion is considered here. There is a base case that if the input list is empty, which means we have looked at all inputted coordinates, simply return the grouped list. At each time we are comparing the streets' names, we are looking at three points, which is head, mid, and tail. Before it hits the base case, there are three conditions which are:

- the head coordinate's street's name is not equal to the street we are looking for, simply append this coordinate to the *groupedList[index + 1]*, and set this street's name as the street name we are looking for.

```
if ( head_street_name != street_name_searching ):
        groupList.append( this_coordinate )
        delete this coordinate from list
        do BR on the list
//
```

- if the last condition is not true, the tail coordinate's street's name is equal to the street we are looking for, append this coordinate to the *groupList[current index]* and this searching is done.

```
else if ( tail_street_name == street_name_searching ):
        groupList[-1].append( this_coordinate )
        delete the whole inputted list
//
```

- if the last two conditions are not true, if the mid coordinate's street's name is equal to the street we are looking for, append this coordinate to the *groupList[current index]*, and do BR on the second half of the inputted list.

```
else if ( mid_street_name == street_name_searching ):
        groupList[-1].append( this coordinate )
        delete list[:mid + 1]
        do BR on the list
//
```

Then at the end of this function, a grouped list that contains at least the start and the end coordinates of each street will be returned.

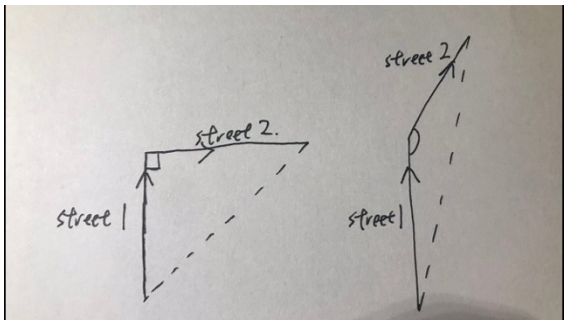
```
groupList = [ (start_coor, end_coor), (start_coor, coor_2, coor_3..., end_coor), ...]
coor is a struct that contains lat, lon, address
```

- Direction and travel distance. By John Zhou

  The slope and the angle are used to determine the direction. In order to find the direction, we assume all the roads are straight and try to build a triangle by one interaction and two streets – the slope of the dotted line determines the direction and angle between streets decide either turning or slighting.

  a). positive slope of a line indicates that this line is from left to right; contrary, negative slope indicates right to left.

  *b). tan(angle) = abs((slope2 – slope1)/(1 + slope2 * slope1))*



  Since roads are not always straight, using the last two coordinates of one street and the first two coordinates of the next street from the returning of the binary searching, is more accurate than using the first and last coordinates of the streets.

  To find the travel distance for each street, adding the distances between adjoining coordinates on the same street, and each sum calculated is the travel distance for each street. In order to do that, the python library geopy is needed.

Performance Results and Discussion

We were considering building a webserver user interface so that the user can both manually input or drag files to upload. However, because one team member was disappeared in the middle, he was supposed to take care to implement the webserver using python flask, we do not have a good interface at this time. Our project only takes input and output to the console. Each member was supposed to take care of one part of this project, put them together, and then we were thinking to optimize our code to enhance the performance. However, in this condition, we two have to take care of the part that we were not supposed to do, we put a lot of time to look into the details, and to write the code, we do not have enough time to optimize our codes. Our project, in the end, is simply based on the basic idea we came up with at the beginning, but it works just fine and performs like expected, but may not that efficient.

Conclusion

We learned how to use web service API as a part of our program, which is convenient for inputting data. Reviewed the binary research and learned how to work with others. We thought work with others was difficult because each part requires inputs from another part, they are not working alone. However, we found out that we can regulate the inputs and outputs of each part before writing codes and generate “fake data” to feed our part to test, which is a huge improvement for ourselves. We believe we can do better in the next project, or even in the future works.