CS 3110 Final Project Milestone 3

System description

Core Vision： Ithaca Map is a location based map services that resembles Google Map, including zooming in and out the map, moving the map, auto-completion enabled search, finding shortest path, and category search.

Key features: GUI, Ithaca based map, Server, Networks

Description: The project is a Ithaca-based map service inspired by Google Map. The user can click on a point on the map and the image corresponding to the location will pop up. The user can further zoom in or zoom out to see a more detailed image or a more general image with that location as the center. We download the tile images and map feature data from OpenStreetMap, whose real-world mapping data are updated weekly and free for access.

The location pinning will be implemented through calling remote APIs on the client side. The map images will be provided by the client application through connecting and requesting information from the server. The server side is responsible for storing images, parsing and storing route information fetched from OpenStreetMap, a non-profit organization releasing map data under open license. We will use a graph tree to store the images. Algorithmic efficiency on searching, routing, map displaying is also an important part to make data fetching faster, providing a smooth user experience. We will use AWS to host the server.

Another feature is that clients can search path between two locations in our Ithaca Map. We will work with real-world mapping data, and we will return the shortest path that starts from the closest connected node to the start point and ends at the closest connected node to the endpoint.

The auto-completion feature has enabled user to type in first several letters and all the names beginning with those letters will pop up.

The application will also provide a key location search system that enables user to type and search key locations, such as libraries, eateries, gas stations, etc. The locations will be represented as related icons, and when users click on the icons, the name of the place will be displayed.

System design

Module 1: Server

Purpose: The server is located on a VM on the cloud that actively receives client connection and send them relevant information upon request.

Sub-Module 1: Map Tree

Purpose: The Map Tree component of our system is used to efficiently locate the map image the client requests in order to minimize response time. The tree is created such that maps with lowest resolution is placed near root and each (non leaf) node has 4 children with higher resolution representing the map of the same area, using the a spatial data structure, **QuadTree**

Sub-Module 2: Graph

Purpose: A graph representation of the map information parsed from the XML file with nodes denoting locations and edges denoting paths. Graph module will also include the algorithm of finding the shortest path between two nodes.

A* algorithm will be used to calculate the shortest path between two locations.

KD tree algorithm will be used to find the closest node of a given point.

Module 2: Client

Purpose: This part of this module is stored on the client's computer that interacts directly with the client and request information from the server.

Sub-Module 1: GUI

Purpose: GUI module is the user interface for the interaction between the user and the server. It has the following functions: 1. Display the map of suitable size; 3. Including Zoom In and Zoom Out buttons; 4. Display a search bar where user can type in the destination; 5. Shortest path on the map; 6. Autocompletion, etc.

Sub-Module 2: Trie

Purpose: Trie module is the ordered tree data structure to store a dynamic set of strings to enable the "auto-complete" search function. For example, we have a number of place names and when the user types "Co", it can automatically displays "Colgate", "Cornell", etc, for the user to choose from.

Changes:

Change 1: Before, when the user wants to find the shortest path between two locations, he can only click on the map to indicate the starting or ending point. Now, the user can indicate a starting place / ending place by either typing its name into the search box or directly click on the map.

Reason: This gives users more freedom to find places and also adds more functionality to our system.

Change 2: When user type in the name of a place, all the places with that name will be tagged on the map. Then the user can click the Starbucks he wants and all the other tags of Starbucks will disappear.

Reason: This change make more sense for users since if the user wants to find "Starbucks", he will want to see all the Starbucks to choose which he wants.

Change 3: Shortest route finding supports two kind of transportation: on foot or by car.

Reason: Since most people will only search for one kind of transportation, this change enables users to make more use of our system. We also store a tag with each road, indicating whether it's a walkable road, a car road, or both.

Change 4: Server uses cohttp library instead of unix socket.

Reason: Since we are using js_of_ocaml for our gui, and since javascipt can't connect with unix socket, we instead choose to implement server using cohttp.

Testing:

- Trie data structure unit testing
- Map image rastering unit testing
- Route finding between two places interactive testing
- Graph function unit testing
- KD tree unit testing
- Server interactive testing
- GUI interactive testing

Division of labor:

Yuntian Lan:

Implemented the graph module for coordinate / name / location translation and route finding. Later worked together with teammates to implement the HTTP server module and part of GUI. Spend around ~85 hours for this project.

Xinzhe Yang:

Worked on raw tile image preprocessing, Image module and Trie module with testing; worked on the HTTP request and client state part of GUI and helped connect between GUI and server development. Spent ~85 hours on this project.

Xinqi Lyu:

Mainly implemented the GUI of the project, including the autocompletion of the textbox, rendering of the canvas, dragging and zooming, and the category search. Later worked with teammates to connect server and gui together. Spent ~75 hrs on the project.

Hanqing Jiang:

Worked on parsing xml file; worked on building Graph module with testing; implemented KD tree; implemented some GUI functions; worked on server handling request.
Spent ~65 hours on this project.

Acknowledgements:
The CSS styles for GUI are borrowed from the skeleton code of a class project but our GUI is entirely coded in OCaml. The tile images are made possible by OpenStreetMap under Creative Commons Attribution-ShareAlike 2.0 licence.

For TA:

Group NetIDs: yl652, hj284, xl358, xy269

Grader NetID: jb965

Overall comments:

A high-quality, ambitious project. Excellent work!

The project integrates server, gui, algorithms and a lot of data structures, which makes the project of great complexity and difficulty.

Features:

The group implements an interactive map that is based on Ithaca, which resembles Google Map. The features of the map include zooming in and out, dragging, auto-completion enabled search and route finding.

For zooming, user can click on the zooming buttons to make the map more detailed or more general. For dragging, user can click and move the mouse to drag the map. The tile images and map feature data are downloaded from OpenStreetMap. They use quadtree to store the tile images and parse the map data into json to extract information. The map images will be provided by the client application through connecting and requesting information from the server. The server is in charge of storing images, parsing and storing route information. Both Google Cloud and local host can be used to host the server.

Another feature is to search route between two locations on the map. The group uses real-world mapping data and returns the shortest path starting and ending at the closest nodes connected to the start and ending point. They use A* algorithm to return the shortest path. They use KD tree to find the closest connected node of a given point. User can either enter names into the search boxes as starting or ending locations or can double click the map to determine the starting and ending locations. There are two kinds of route searching: by automobile or by foot.

They have also enabled auto-complete search in the search box. They use Trie data structure to store all the names. User can enter the first several alphabets into the search box and all the names beginning with those alphabest will pop up for the user to choose.

The last feature is the key type location search. The user can click on the buttons to search for libraries, eating places, shopping malls, gas station, etc. Then the locations will be displayed on the map. If the user click the icons, the name of the location will display next to the place.

Testing:

The group implements a great mix of different testing technique. For their trie data structure, map image rastering, graph functions, KD tree, they mainly use unit testing. For the server, the gui and route finding, the group mainly uses interactive testing to make sure that things work as expected.

The unit tests are exhaustive and they have considered corner cases. The interactive tests are adequate. The group has spent a great time making sure that the system is robust. They successfully accomplished what they planned to do with no bugs.

Code quality:

Overall the code quality is excellent. The codes are readable and the comments are detailed and easy to understand. The indenting, naming and pattern matching are all correct and concise. Helper functions are created to avoid repetitiveness between codes. The logic and flow of cods are great. They met all the requirements in the Ocaml style guide.

Demo:

The system is robust and easy to use. It doesn't crash under any circumstances. The features are all implemented great. The group have greatly improved the time needed to complete each action, such as dragging, showing route, etc. Now data fetching is very fast, providing a smooth user experience. All the features work as expected.

The GUI is pretty; the buttons, textboxes and route showing are all easy to navigate. There is also a question mark button which gives adequate information on how to use the system.

During the demo meeting, the group clearly explained the system they built and every person knows the project very well.