**Priority queue evaluation based on road type**

**In one paragraph, describe what you did with your extension to someone who will just be using your application and doesn't know anything about programming. (Keep this answer, copy-paste to another document- if you want to submit it for the required quiz this week.)**

I have decided to slightly implement the change in the distance to source computation metric. Every map data source contains additional information about a name of the street (edge) and its type. So far I have counted eight different road types (motorway, motorway\_link, primary, residential, secondary, tertiary, trunk, unclassified).

I have implemented queue comparison based on the type of the road which leads to the neighbor. Highways and motorways are privileged over country and dirt roads.

The solution takes the road which leads to the node neighbor and in time of addition into the queue, it evaluates its value.

In a mash of similar routes, it does not have much advantage in search than BFS algorithm. On the other hand, I believe that in the mash of different routes, where ending point ends close to the highway offers significant improvement.

**In one-two paragraphs, describe what you did with your extension to someone else on the same project team. What problems did you encounter and how did you fix them? (Keep this answer, copy-paste to another document- if you want to submit it for the required quiz this week.)**

First I am not sure if there are any other road types than I have found out. More specification would be nice to be sure.

Second, this solution does not try to come with any faster or slower way to find a path. It only changes the way to evaluate the priority queue.

To change evaluation metric it is necessary to manually comment/uncomment MapNode compareTo method code.

Few more additional fixes and refactoring was made during this week.

Deeper implementation with an option for UI would require much more time and drill-down more code.

As I was thinking, for finding an ideal path between starting and ending point should be used a metric which would be a combination of several factors such as road type, distance from the source, road blocks (traffic jams, constructions), drive cost (toll), etc. Therefore simple metric such as air-distance from source is not in praxis enough.

**Other people – User 1**

Q1

Before returning the shortest path obtained from Dijkstra algorithm or A star search algorithm, I not only track each node's parent and add it to a list but also find its edge. Then I use speed limit to divide the distance obtained from each edge and add these values to get the traffic time of the shortest path.

Q2

In order to get much accurate traffic time, we not only should consider each road's speed limit but also should consider other conditions such as car speed and weather. Different traffic conditions can affect the length of traffic time.

**Other people – User 2**

Q1

Give options to users to select fastest route or shortest route like normal GPS.

User can even avoid using highway (motorway).

Intersections will add more minutes to overall time.

Q2

1. Add methods to classes of MapEdge, MapNode for processing of time parameters (current used time from start, estimate time to goal, time on current edge).

2. Update distance methods in djk and astar functions in MapGraph with time methods.

3. Road types determines road speed. But types are unknown. I read GraphLoader.java and add a method to get all roadtypes for different maps. Then in class Edge, I add setSpeed method to set speed based on roadtypes and getTime method to calculate time.

4. When user select to not use motorway, edges with roadtype motorway are not added to map.

**Other people – User 3**

Q1

I added an extension so that popular routes are cached for retrieval for the map graph application.

Q2

Debugging is the hardest part.

**Other people – User 4**

Q1

I added functionality to store a path, which previously has been found to speed up the search. This feature was added to Dijkstra and AStar.

Q2

1. LocationPair class was added.

The class has:

- two location points, getters, and setters,

- isEqual() method (returns true/false)

- overridden equals() and hashCode() to be able to compare different instances with the same content.

2. LastPaths class was added with HashMap, which has a LocationPair as a key and List of locations as a path.

The class has two methods getPath() and setPath().

How to use:

Create an instance of LastPaths. Before calling Dijkstra or AStar, call getPath() with the same parameters. If getPath() returns null, call Dijkstra or AStar and then call setPath().

Future development: this functionality could be added inside the path searching algorithm to speed up the intermediate process.

**Other people – User 5**

Q1

There are two buttons called calculateDist and calculateTime. By selecting some vertex and then click one of this button, the application knows the distance between any pair of node you choose, then by clicking generateGreedyTSP button, a TSP solution is calculated and displayed on the application. The difference between button calculateDist and calculateTime is that the first button calculate length distance the second calculate expected travel time.

Q2

In this extension, I calculate the travel speed based on road type and then calculate expected travel time on each road. All those calculation was done in a support class I created before called Street. Then I create a new class DistanceGraphAdjList extends fomr GraphAdjLists class. Compare to the parent class GraphAdjLists, I add a new matrix to store the shortest "cost" between each pair of vertex: you can store either shortest distance or shortest travel time based on your preference. The matrix is empty when initiated, and the value is updated whenever a Dijkstra algorithm is run. Functions like generateDistanceMatrix() and generateTravelTimeMatrix() are designed for generating the whole distance matrix between vertexes by running Dijkstra starting from each vertex once. After getting a distance matrix, a greedy TSP algorithm is implemented in a function called greedyTSP() in MapGraph class. This function returns a list of GeographicPoint, which denotes the TSP cycle. I do this extension on my own. I think the most challenging problems are all about class design and refactoring. For example, I know I should have a matrix to store distances between each pair of vertexes but it took some time to finally decide to put it in the DistanceGraphAdjList object called data, which is a private attribute in class MapGraph. Also, I decide to add a boolean variable in Dijkstra algorithm function to decide whether to update my distance matrix by distance or travel time.