Graph Project

Architecture

The project architecture is based on **Jupyter Notebooks** which makes visualization easier when coding python.
We have used the python package **Osmnx** which gives us the tools for handling geospatial data from **OpenStreetMap**.
By utilizing these tools we can visualize the shortest path between two nodes by plotting the geo-data using the package **matplotlib**.

Data Structures

The geo-data is stored using the package **geopandas**, which is essentially pandas just with geodata. This means we store the data in geodataframes and series.

Algorithms & Complexity

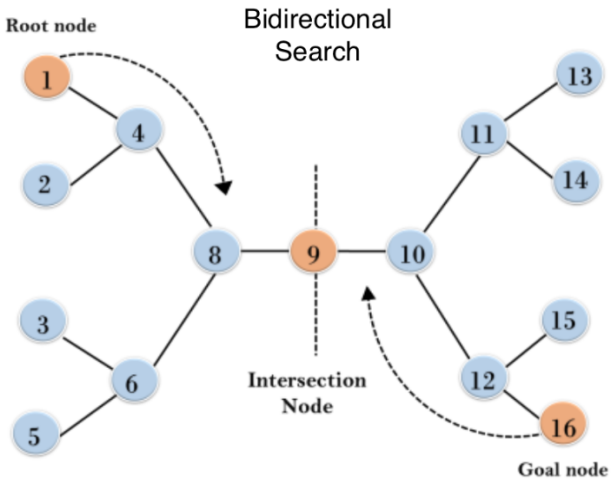
Dijkstra
This algorithms makes use of three main structures:

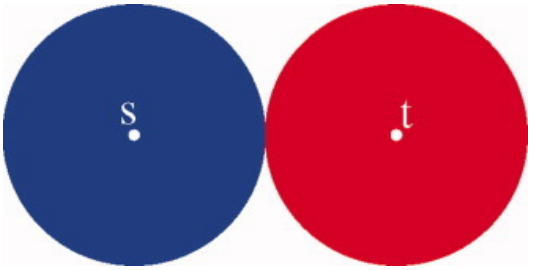
- *minDist* - An array that in the end will store the minimum distance values from the source node to each node in the graph.
- *visited* - A set which stores all the nodes that have been visited by the algorithm.
- queue - A queue that holds all nodes in the graph, and prioritizes the node with the least cost/distance from the source. (Queue will be empty when shortest path has been found)

The algorithm will follow these steps:

- Explore the node with the smallest distance from the source by popping it from the queue. In the first iteration this will be the source node itself since the distance from the source to itself is 0.
- Add the now explored node to the "*visited*"-set.
- Update the minimum distance values for each adjacent node *a* of the node currently being explored.
  If the minimum distance to the node currently being explored + the weight of the edge to the adjacent node *a* is less than the minimum distance to the adjacent node (which will be infinity if adjacent node hasn't been explored), then update the minimum distance value of the adjacent node.
- When the "*visited*"-set is full and the queue then is empty, the algorithm stops.

Bidirectional Dijkstra





This algorithm is in most cases an optimized version of the normal Dijkstra shortest path. The main difference is that instead of only performing a forward search from the source node to the target node, we now perform a search in parallel (or alternating) from both directions. By doing so we minimize the nodes we need to visit before being able to determine the shortest path. The shortest path has been found when the two searches intersect. (In some cases this is not true which is why the algorithm must finish expanding all the nodes that have the same “layer” or “depth” in both searches before determining the final shortest path).

Bidirectional Dijkstra is approximately 2-4x faster than normal Dijkstra. We can show this mathematically by using the two circles:

The area of the single blue circle *s* (normal dijkstra expansion):

$$area = \pi * r^2$$

When we then consider the area of the blue circle *s* and green circle *t* combined, we have that the radius of each of the two circles make up half of the entire radius of the area. So since we now have two circles the equation is as follows:

$$area = 2\pi * (r/2)^2$$

$$area = \pi * r^2/2$$

So by doing this we get the same result as with one circle, but divided by 2. Therefore this would be two times faster.

(Euclidean) K-Nearest Neighbours

This algorithm classifies a datapoint based on the classification of its neighbours. In our case it classifies the addresses/points to the nearest nodes on the graph from which we are then able to calculate the shortest path between.