City Connector in China (Report)

Aim of the project:

The *main aim* of our project is to connect 300 of most populated cities in China by laying the shortest route through all these cities. Moreover, these roads should be constructed the way that each city can be reachable from any other city(among these 300) in China. The best way to do it is to write an algorithm which will find the minimal spanning tree. For the better understanding, the target spanning tree should be visualised. To conclude our project we will compare what we are going to obtain with real data.

Methods:

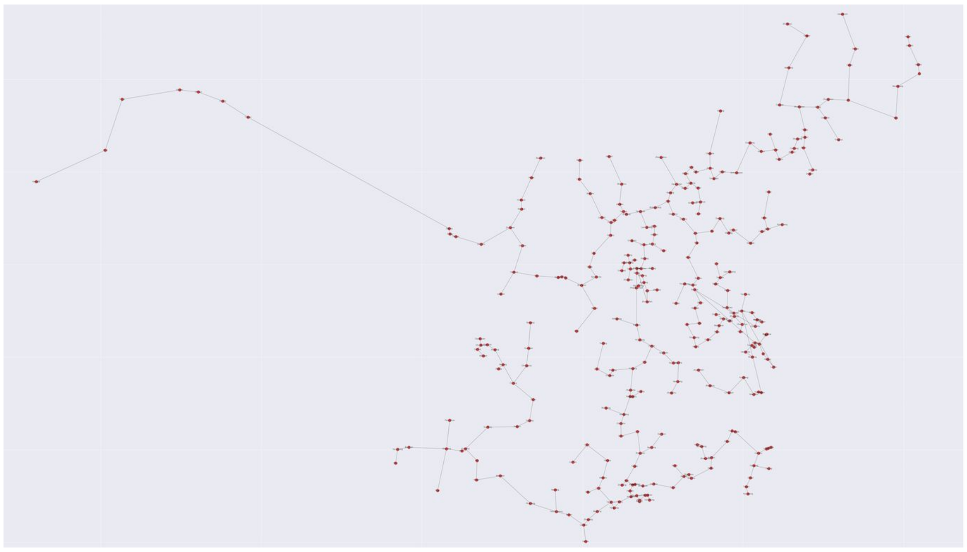
Our group decided to write algorithms and code in C++, however for the visualization we decided to use Python.

All cities in descending order by population we found on the web-site (*geo.koltyrin.ru)* and the coordinates of the cities we found via Yandex maps.To check the authenticity of our results we used Google maps.

The process of work (step by step):

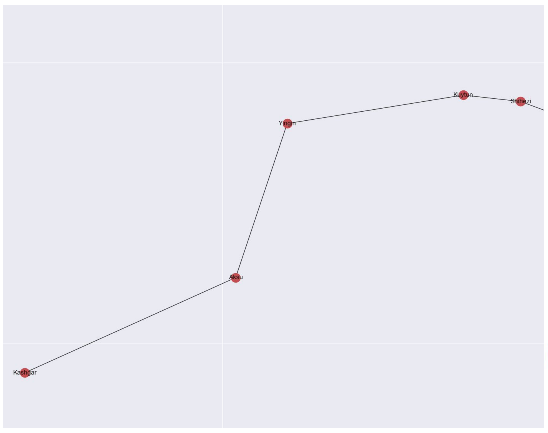
1) Firstly, we wrote down the table of all cities with coordinates in a csv file and then, in order to use it in our code, we transformed it to txt file.
2) We built a distance matrix via C++. Where we counted distance as a Euclidean distance($d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$).
   https://github.com/innpei/city-connection/blob/master/new_dist_mat.h
3) Then, we created a graph with the use of all information about cities.
   https://github.com/innpei/city-connection/blob/master/graph.h
4) We separately implemented three algorithms to find minimal spanning tree for this graph.
   a) ***Boruvka algorithm***. The main idea of algorithm is: Input is a connected, weighted and directed graph. Initialize all vertices as individual components (or sets). Initialize MST as empty. While there are more than one components, do following for each component. Find the closest weight edge that connects this component to any other component. Add this closest edge to MST if not already added. Return MST. We will explain it in more details in presentation. The implementation of the algorithm.
   https://github.com/innpei/city-connection/blob/master/boruvka.h
   b) ***Kruskal algorithm.*** The main idea: Sort all the edges in non-decreasing order of their weight. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it. Repeat until there are (V-1) edges in the spanning tree. We will explain it in more details in presentation. The implementation of the algorithm.
   https://github.com/innpei/city-connection/blob/master/kruskal.h
   c) ***Prim algorithm.*** The main idea: a spanning tree means all vertices must be connected. So the two disjoint subsets of vertices must be connected to make a Spanning Tree. And they must be connected with the minimum weight edge to make it a Minimum Spanning Tree. The implementation of the algorithm.
   https://github.com/innpei/city-connection/blob/master/prim.

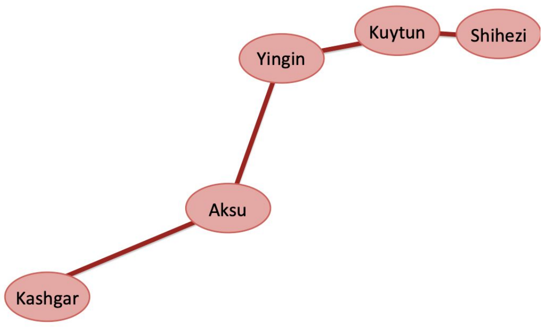
5) After using these algorithms we obtained an adjacency matrix (https://github.com/innpei/city-connection/blob/master/adj_mat.txt), that shows which cities are connected to each other in the minimal spanning tree. Therefore, with the help of this matrix we can understand what roads must be constructed to make each city reachable from any other city with the less amount of kilometers.
6) Then, we did visualization of this adjacency matrix by using Networkx Python. https://github.com/innpei/city-connection/blob/master/graph_visualization.ipynb Here is what we obtained; graph on 300 vertices (each vertex is a city in China and each edge is the most efficient path). Compare it with China on a map.



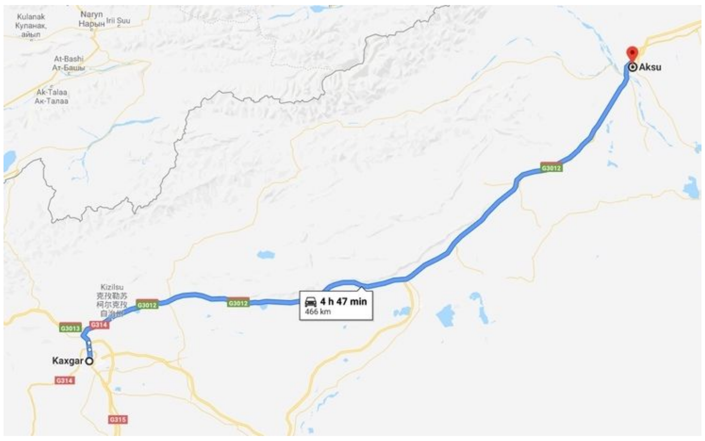
7) On the last step we should compare our result with actual data. The proper way to do it is to take a small subgraph from out MST graph, for instance, we can take 5 cities that are close to each other and find out how many kilometers will it take to reach any other city from this subgraph and compare our results with results on Google maps. So, we took 5 cities: Kashgar, Aksu, Yingin, Kuytun and Shihezi and by using our graph checked km between them. Here is the data that we obtained from MST. It is given in degrees, therefore we should convert to kilometers using the formula: **L** = **π** * **R** * **a** / **180** (where **L** - arc length, **R** - radius of circle, **a** - angle)

Kashgar-Aksu: 4.61376° ---> 514 km
Aksu-Yingin : 2.94124° ---> 327 km
Yingin-Kuytun: 3.61967° ---> 403 km
Kuytun-Shihezi: 1.17393° ---> 130 km



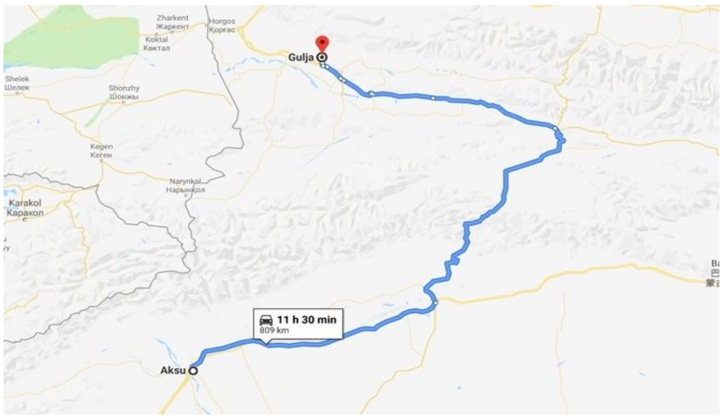


a) *Distance between* *__Kashgar__* *and* *__Aksu__* *≈ 514 km by our graph. Google maps:*



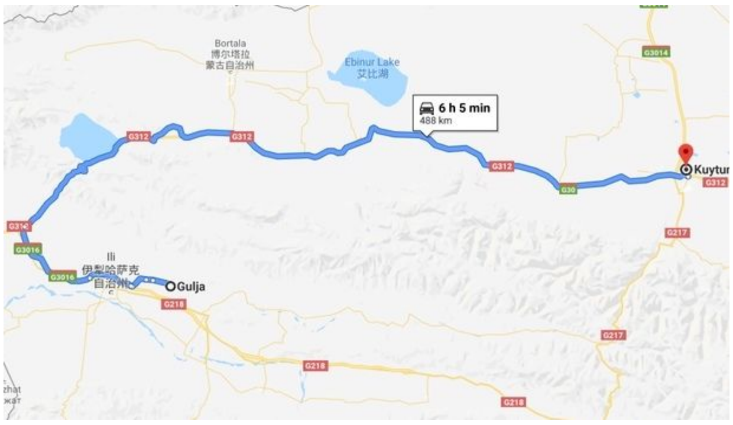
466 km.

b) *Distance between* *__Aksu__* *and* *__Yingin__* *≈ 327 km. Google maps:*



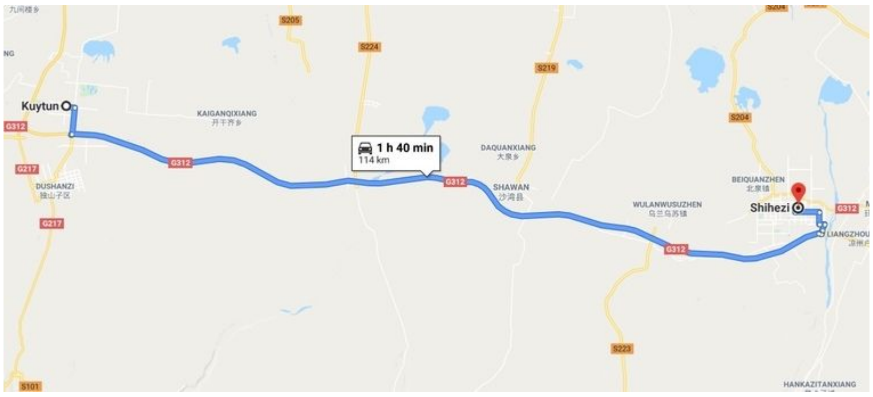
809 km.

c) *Distance between* *__Yingin__* *and* *__Kuytun__* *≈ 403 km. Google maps:*



488 km.

d) *Distance between <u>Kuytun</u> and <u>Shihezi</u>: ≈ 130 km. Google maps:*



114 km.

By analyzing these maps it can be easily seen that approximate number of kilometers from our graph coincide with data from Google maps. For this reason our program can be used to calculate the most efficient choice of routes between cities.

For example, in the sphere of tourism this program can be very useful to help clients to choose the best program for their traveling. What is more, another application of our program could be transportation of products in China; for any chinese company that spread its products throughout China our program can choose the least expensive option of transportation as it will choose the shortest path.