# Exercise 1

The goal of this exercise is to implement and solve the travelling salesman problem.

Firstly, with the use of function **choose\_random\_cities**() a certain number of cities as a points are chosen randomly on plane defined by a user-defined boundary ([-100, 100] X [-100, 100] in this exercise).   
Creating connections is done by a function **create\_distances\_matrix()** that calculates the distances between the cities as a Euclidean distance if there exists a road. Distance between the cities not connected directly is set to math.inf that indicates there is no road connecting these two cities.  
Functions **create\_connections()** and **plot\_cities\_graph()** are used to represent the graph graphically and need the matplotlib library to work correctly.

Actual implementations of the travelling salesman problem can be done with a various of ways.  
Starting with the full searches of the tree, the Breadth First Search or BFS for short is implemented with the use of a function **bfs()**. This way of searching is done by traversing the tree level by level. My implementation is a recursive function that stores the current paths taken and the corresponding sums of distances. This function returns a list of distances and paths in a form of [ (cost, (path)), …].

Another implementation is the Depth First Search or DFS for short implemented with a function **dfs()**. In this way of searching, the tree is traversed from the first city down to the furthermost city and then backtracks. My implementation as previously is a recursive function and returns a list of distances with the same form.

Then the approximation of the lowest-cost road is done with a greedy search. This way of approximation is done by choosing the lowest-cost paths as the tree is traversed. It does not guarantee the best possible result, but is much faster and memory-efficient than the full search. My implementation of the greedy search is contained in a function **greedy\_search()**, it is based on a DFS method. For each step we choose the best possible road and backtrack if we hit the dead-end.

# The results:

For the number of cities equal to 5 and all possible connections:

A picture containing text, boat

Description automatically generated

The results found are:

Text

Description automatically generated

Another run for the same number of cities but with only 80% of roads available:

Chart, line chart

Description automatically generated

And results:  
Text

Description automatically generated