# Map Anything Challenge Discussion

D'Azevedo, Gloria
gloria.dazevedo@gmail.com

May 2018

## 1 Question 1: Create an adjacency graph using only a list of latitudes and longitude pairs

### 1.1 Approach 1: List Solution

The idea of using this solution was such to preserve the order of the locations (only denoted by a latitude and longitude without a unique identifier) and then create an adjacency matrix for distances between points i and j. This solution when implemented correctly would save about half of the storage space to be on the magnitude of $\frac{n^2-n}{2}$ instead of $n^2$.Assuming the distance between two points are the same, no matter the direction, we only need to construct the upper right half of the matrix denoted $x_{ij}, j > i$. Also note that the diagonal would always be 0 since a point is always 0 distance from itself and we would not need to store it. I have implemented a "list of lists" solution where the wrapping list denotes the row number while the second index is a function of both the row number and the location number.

There are limitations to this approach. To access the distances between two points, the user would first have to know the order of the locations in the input file which would not necessarily be the easiest to go through. The current assumed implementation does not have a key or location identifier which could potentially make that lookup easier as well. In addition, due to the construction of the list of lists, the length of each sublist decreases as the rows increase. Thus, to access $A[i, j]$ as expected in mathematical notation, the actual lookup would look more like $A[i][(j-1)-i]$ assuming that $i < j$

### 1.2 Approach 2: Dictionary Solution V1

This idea improves from the first approach of requiring the knowledge of the order of the locations and their indices in the original list. The major differences between this approach and Approach 1 is that instead of storing the adjacency matrix as a list of lists, we create one tuple for each possible combination of location pairs.

When accessing the dictionary (done in $O(1)$) the code is easier to interpret since the actual latitudes and longitudes are used to access the values in the dictionary. However, since this version of the implementation uses a tuple to denote a pair of locations, only one (ordered) copy of the pair is stored as key. As an example, we would store the distance between $A$ and $B$ as $dict[(A, B)]$. However, if we called $dict[(B, A)]$ then no such key would exist. Thus we would need to know the specific order of the locations in the tuple to access that distance.

### 1.3 Approach 3: Dictionary Solution V2

This approach is essentially the same as the Dictionary Solution V1 except instead of adding just one tuple to the dictionary as a key, we add both "versions" of the latitude and longitude order as tuples in the dictionary. To illustrate, we would add both $(A, B)$ and $(B, A)$ to the dictionary keys. In this method, to access the distance is still $O(1)$ time but the amount of storage would essentially double from Approach 2.

If there was some previously defined location pair convention method such as the location with the lowest value of latitude goes first in the tuple for accessing and storing the dictionary key, then this solution would satisfy both the storage problem and the location pair access problem!

## 2 Question 2: Optimize the route

Goal: Need to minimize the number of drivers such that all clients are mostly satisfied, we can make all deliveries within the week and other constraints such as client service time and travel time are reasonable (full list in problem specifications).