## To Run

Place python script file into the same directory as CareDist\_Full.csv and CareDurations.txt, install the necessary packages used, e.g. NumPy and then run the python script in your terminal to receive an output using the random algorithm and the minimum distance algorithm.

## Algorithm Details

The algorithm begins by reading in the CareDist\_Full file as a string due to the User ID column being present in the first row, it then replaces this with a dummy variable of 0 which allows for the matrix to be read into NumPy as a distance matrix. The 0 distances are replaced with an arbitrarily large number to ensure that the algorithm will not select these connections due to their “low” weighting.

Next the Care\_Durations are read in, with the durations being converted from minutes into seconds to match the time format of the Distance matrix. Durations of 0 for the 6 centres are added to so that the same function can be used to parse both the Distance Matrix and the Duration table simultaneously later on.

The algorithm used is a greedy algorithm, specifically the nearest neighbour algorithm. It is greedy because it makes all its choices based on the local optima (selecting the shortest distance one node at a time) rather than working out the shortest path overall.

The algorithm first marks all the nodes in the distance matrix as unserved. It then selects one of the centres, by iterating through them until it finds the one with the shortest connection to another node. This was compared to an algorithm that selected the start node randomly with the random selection occasionally leading to better routes, but the non-random method would consistently create roots of a higher quality.

After selecting the start node, the algorithm then selects the next nearest node, and adds the travel time between the two nodes, and the duration of care to be carried out at the destination node to a running tally of the time the route takes. It continues this process for each node until the cumulative time for the route hits 25200 (7 hours) or higher as this is the length of a volunteer’s shift. After it hits this limit, it adds the current path so far to an array of routes, before resetting the path and amount of time passed. A counter keeping track of the number of staff required is then incremented by 1. The algorithm then begins again by selecting another starting centre, either randomly or with the shortest connection, until it has iterated through all the nodes and marked them as served. The list containing all the routes for each staff member is then outputted for the next step.

Up to this point the algorithm has been referring to each node according to its index in the Distance matrix and so a dictionary of the indexes and the actual node names is created to allow the routes to be converted from the number 0 to 236 back to the node IDs, so they can be more easily parsed, either by a human or by the 2-opt algorithm seen in part D.

## Output Details

The routes are then converted back using the dictionary of nodes and then each route is written to a text file, along with the total staff required, in the following format:

Route 1: a, b, c, …

Route 2: d, f, g, …

…

Route n: x, y, z, …

Number of staff = n

A machine-readable version of the minimum distance results is also included, which just lists each route with square brackets and separating commas, making it easier to be fed into the 2-opt algorithm in part d.

## Details of Algorithmic Differences

The random algorithm is different from the minimum distance algorithm in one key regard. After connecting the start point to the nearest node, the random algorithm selects its next start point arbitrarily whereas the minimum distance algorithm finds the closest depot to the last point visited up to that point and then begins there when finding the first node for the new route. However, when there are no longer any unserved nodes directly connected to one of the starting points, the minimum distance algorithm then defaults back to randomly selecting a start point and working outwards from there once more.

## Potential Improvements

There are a few ways in which the algorithm could be improved. For instance, the work duration and travel time could be combined in a data frame to allow for both to be used as weightings instead of needing a more complicated function to reference them. I would also like to improve upon the applicability of my minimum distance algorithm by allowing it to see more than one node ahead from each of the starting points, allowing for a better overall route to be found.

As we made use of a locally greedy algorithm we often did not end up with anywhere near close to the optimum route in some runs on the data. Instead, we could use an algorithm better suited for finding an optimum route, such as Christofides algorithm which is a more complex algorithm that is able to find a guaranteed result within a factor of 3/2 of the optimal solution length. Alternative methods such as dynamic programming or contraction hierarchies could be used to speed up more computationally complex algorithms like this, allowing them to find more optimum solutions in a comparable timeframe to the nearest neighbours’ algorithm.