The Travelling Salesman Problem

The **travelling salesman problem** asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

The objective of this project is to find the shortest possible path among a set of given points (along with their coordinates) that traverses every single point exactly once, and then returns back to the starting point, thus attempting to solve the travelling salesman problem.

This is done with the help of an algorithm, used to find the optimal path. The algorithm is explained hand-in-hand with a live example as that will be much easier to understand.

Working of the algorithm :

- We assume that n distinct points are fed by the user to the algorithm, along with their coordinates.
- Then, we create an nXn matrix which contains the distances from each point to every other point as the elements. (As in, the element M (a, b), where M is the matrix, contains the distance of b from a.) Note that distance from one point to itself is considered as infinity, hence an entry of M (i, i) is considered infinity for all i in n. A set of values in the example case has been taken and formed into a matrix as shown in fig 1.
- Now, we “reduce” this matrix. This is done by finding the minimum element from across each row and column, and subtracting them from each of the elements. The sum of all the removed elements is called the **reduced cost**. The resultant matrix achieved is the **first reduced matrix**. It

shall be referred to as C. The reduced matrix has been shown in fig 2. The reduced cost as calculated is 25.

- Now, it is assumed that the path starts from one point. In this case, we consider point 1 to be the starting point (With regards to length of path, starting point won't matter). As visible in the flowchart in fig 3, we have four possible options to go from point 1.
- First, consider the path 1 ⟹ 2. Now, to find the cost of node (or point) 2, we need to reduce the first reduced matrix in a particular way:

    - First, make all the elements in the row of the starting point and the column of the ending point as infinity. So, all the elements in the row 1 and column 2 are made infinity.
    - Then, as once we come from a point to another, it can't go back, the element of the order (from_ point, to_point) must also be set to infinity. In this case, C (2,1) is set to infinity.
    - The reduced matrix for path 1 ⟹ 2 is given in fig. 3.
    - Finally, the reduced cost for this node shall be calculated according to the formula:
      C(from_node, to_node) +r + r^, where r is the reduced cost of the current matrix, and r^ is the reduced cost of the first reduced matrix.
      So, reduced cost for node 2 in this example is:
      10 + 25 + 0 = 35.

- Similarly, the reduced matrix and the corresponding cost of reduction is found for all the points from the first node. In our example, the reduced costs for node 3,4 and 5 are 53, 25 and 31 respectively.
- Then, the node with least cost of reduction is found, and that is assumed to be the next node in the optimal

path. Hence, in our example, the point 4 will be next node.

- Now, from the next node, the flowchart is further extended to possible points. Then, just like in the previous case, reduced matrixes are taken cost of reductions are found.
    - From node 4, the path can go to nodes 2,3 and 4. The reduced matrix obtained for node 4 is taken as C^ (fig. 4). This matrix is then reduced, in similar manner as the previous case. The cost of reduction here is C (from_node, to_node) + r + r^, where r is the cost of reduction of to_node and r^ is cost of reduction of from_node.

    - For the path 4 ⟶ 2, The elements of the 4th row and the 2nd column are turned to infinity. The elements C^ (2,1) and C^ (2,4) are also turned to infinity. Then, the cost of reduction is found.
      The cost of reduction of node 4 to node 2,3 and 5 happens to be 28, 50 and 36 respectively.
- Again, the node with the least cost of reduction is chosen as the next point in the path. Hence, the path now becomes: 1 ⟶ 4 ⟶ 2.
- Next, the similar method is followed to find the next node (from 3 and 5), by reducing the matrix and hence finding the cost of reduction for the remaining points.
- So, the cost of reduction for node 3 and 5 is 52 and 28 respectively.
- Hence, the next node would then be 5 and the node after that automatically becomes 3.
- The path eventually returns to where it started, so it returns to 1.

- Hence, the most optimal path according to our algorithm is:
  1 ⟶ 4 ⟶ 2 ⟶ 5 ⟶ 3 ⟶ 1.

  In this way, the algorithm is able to provide the optimal path for n number of inputted points.