|  |
| --- |
| Detailed Design-: Travelling Salesman Problem using Held – Karp Algorithm |
|  |
|  |
|  |

|  |  |
| --- | --- |
| Document Status: | Draft |
| Document Version No: | 0.1 |
| Version Date: | 19/11/2018 |
| Project ID |  |

# Introduction

Travelling salesman problem is an algorithmic problem focussed on optimization, wherein Given a list of cities and the distances between each pair of cities, we need to compute the shortest possible route that visits each city and returns to the origin city.

* 1. Document Summary

The purpose of the document is to outline the technical design of TSP and provide an overview for the implementation adopted.

This document contains the design details and decisions for implementing the solution for travelling salesman problem. The document includes the below sections:

* Algorithm Details
* Solution Architecture
* Unit test cases
  1. Scope and Context

The solution implemented here is based on the requirements shared. However, we’ve made few assumptions listed below.

Assumptions:

1. The distances between cities will be given in the form of square symmetric matrix.
2. The first row of the matrix will represent the source city always.
3. Each city is interconnected to every other city.
4. The distance between city A to city B is same as travelled from city B to city A.

# Algorithm Details

The simplest approach for the TSP would be brute force, where in we calculate all the possible paths and then out of all of them select the one with the least cost. But this approach would consume lot of time and resources.

For instance if the number of cities provided is n, then the no of possible routes would be n!. Now if we add our restriction to it of having same starting and ending city, then the no of possible routes would become (n-2)! . Therefore, the time required for calculation of shortest possible route would be O((n-2)!).

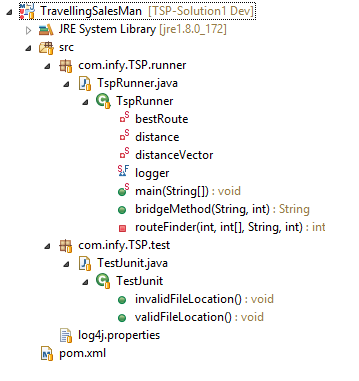
Therefore, we needed a better approach, hence we opted for Held-Karp algorithm.

## Held – Karp Algorithm

The Held-Karp algorithm is a dynamic programming algorithm. The algorithm focuses on computing the solutions of all subproblems starting with the smallest. Whenever computing a solution requires solutions for smaller problems using the recursive equations, look up these solutions which are already computed. To compute a minimum distance tour, use the final equation to generate the 1st node, and repeat for the other nodes. For this problem, we cannot know which subproblems we need to solve, so we solve them all.

# Solution Architecture

Our solution implementation consists of below structure:



TspRunner class is the one which performs the calculations and returns the best route.

TspRunner class consists of methods:

* main() : void
* bridgeMethod(String, int) : String
* routeFinder(int, int[], String, int) : int

TestJunit class performs test cases, it consists of below methods:

* invalidFileLocation() - tests the output for invalid file location
* validFileLocation() - tests the output for invalid file location

Technology Used:

* Language : JAVA
* Testing : Junit
* Build : maven
* Environment : Docker container

## Input Matrix:

The attached file shows a sample input file whose path is to be provided to our implemented module.



The input matrix file needs to be updated and should be placed at a location accessible from the server. And the path needs to be provided along with no of cities as part of command line arguments while running the jar file.

For example : java -jar <jar name> “D://distance.txt” 5

Where “D://distance.txt” represents the file path in machine and 5 represents no of cities.

The matrix is a symmetric matrix, below sample matrix shows the order in which matrix elements needs to be populated for 4 cities i.e. A, B, C, D.

No of cities: 4

AA AB AC AD

Distance matrix = BA BB BC BD

CA CB CC CD

DA DB DC DD

Where AA = distance of city A from city A = 0

AB = distance of city A from city B = x

## Docker Container:

The implemented solution is containerised using docker and the image is hosted on dockerHub.

The image file needs to be pulled and docker file needs to be updated for the file path and no of cities which are passed to the jar file during execution and the docker file needs to be run.

# dEPLOYMENT & tESTING

TBD