INFO 6205 Spring 2023 Project
Traveling Salesman

Mani Charan Reddy Loka (NUID 002727403)
Nishaanth Reddy Tummalapalli (NUID 002707338)
Prashanth Vobilishetty (NUID 002707220)

- Introduction
    - Aim
    - Approach
- Program
    - Data Structures & classes
    - Algorithm
    - Invariants
- Flow Charts (inc. UI Flow)
- Observations & Graphical Analysis
- Results & Mathematical Analysis
- Unit tests
- Conclusion
- References

INFO 6205 Spring 2023 Project
Traveling Salesman

INTRODUCTION:

***AIM**:*

The Traveling Salesman concept is about finding the shortest possible route to travel **n** cities exactly once and return to the original city given the distances between them. The aim of our project is to find the optimal route between n cities given a list of cities identified by their id, longitude, and latitude.

APPROACH:

The Traveling Salesman Problem is an NP-Hard problem which means for a very large number of input cities, it is almost impossible to find the shortest route. We approached the solution by using Christofide's algorithm, which guarantees that the algorithm, in the worst case, yields an output that is 1.5 times longer than the optimal solution.
The algorithm can be described as follows:
Given a graph G:

1) Create a Minimum Spanning Tree MST from Graph G
2) Find Odd Degree Verticess O from MST
3) Find the minimum weight perfect matching graph MG the Odd Degree Vertices O
4) Combine the edges of M and T to form Multigraph H
5) Form an Eulerian circuit in H
6) Extract the Hamiltonian circuit by skipping repeated vertices

After getting the initial tour using Christofide's algorithm, we optimized the tour further using tactical optimizations as 2-Opt and 3-Opt and strategic optimization as Simulated Annealing, and Ant Colony Optimization

INFO 6205 Spring 2023 Project
Traveling Salesman

PROGRAM:

DATA STRUCTURES AND CLASSES:

We have used various built-in Data Structures, such as ArrayList, HashSet, HashMap, etc. We have also defined a few data structures, such as UnionFind, while finding the Minimum Spanning Tree using Krushkal's Algorithm, Graph, etc.

We have the following classes in our solution

Vertex class:
This class has fields id, longitude, and latitude, which are read from the input .csv file

Edge class:
This class has three fields Vertex u, Vertex v, and the weight, which can be compared by weight

Graph class:
This class has List<Vertex> vertices, List<Edge> edges, HashMap<Integer,List<Integer>> adjList, and double weight. It also has methods for adding vertices, and edges, a static method for computing distance between two vertices, getIndex of a particular vertex, and also getVertex given an index.

ReadData class:
This class is a utility class to read input data from the file, and it also initializes the graph object by adding vertexes and edges based on the input file. The JavaUI class also uses this class to initialize points on the canvas.

UnionFind class:
This class has methods to check if two vertices are connected and union two vertices and mark them as connected. This is helpful when computing the minimum spanning tree.

MST class:
This class has static methods for computing the Minimum Weight Spanning Tree graph using Krushkal's Algorithm from the original graph, finding odd degree vertices from the Minimum Spanning Tree, and finally generating the minimum weight perfect matching from odd degree vertices

Eulerian class:
This class has static methods for combining the matching graph and minimum spanning tree to form the Eulerian graph and also finding the Eulerian cycle from the Eulerian graph using a simple Depth First Search Traversal

TSP class:
This is where the program execution begins, the first step is reading data from the input file and initializing the graph, then finding the Minimum Spanning Tree from the graph, then finding odd-degree vertices, forming a matching graph, combining matching graph with MST to find Eulerian cycle and extracting Eulerian tour by removing repeated vertices to form TSP

INFO 6205 Spring 2023 Project
Traveling Salesman

It then calls two-opt and three-opt methods from Tactical optimizations class and Simulated Annealing and Ant Colony Optimization methods. It also has methods to find the path's distance and print the path to the console.

Tactical Optimizations:
It has static methods to optimize the path using two-opt and three-opt techniques.

Simulated Annealing:
It has static methods to optimize the path using Simulated Annealing

Ant Colony Optimization:
It is used to optimize the path using Ant Colony optimization technique.

JavaUI:
This class is used to create the UI for Minimum Spanning Tree and Travelling Salesman Path before and after optimizations.

ALGORITHM:

The primary algorithm we used to find the path of the traveling salesman problem is Christofide's, and for finding the Minumum Spanning Tree, we used Kushkal's algorithm. The total steps involved in our solution are as follows
The steps involved in our implementation are as follows:

1. Initialize the vertices by reading the input data, create edges between all the vertices, and add the vertices and edges to the initial graph
2. Find the Minimum Spanning Tree(MST) using Krushkal's algorithm
3. Find the list of vertices from the MST that have an even degree of edges
4. Construct a minimum weight perfect matching graph(M) by using these odd-degree vertices
5. Combine the MST and matching graph M to form an Eulerian graph(EG)
6. Form an Euler tour of the Eulerian graph using Depth First Search Traversal shortcutting the path by removing repeated vertices which is indeed a TSP tour
7. Perform optimizations as 2-opt, 3-opt, simulated annealing, and ant colony optimization

INVARIANTS:

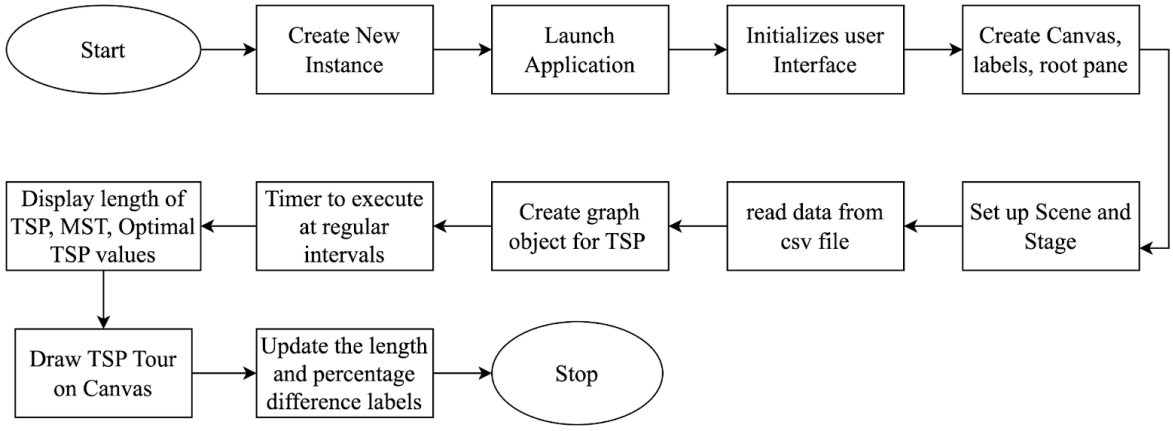
The TSP tour has the following invariants:

1. Each tour must start and end at the same city visiting every other city exactly once
2. The number of odd-degree vertices in an MST is always even
3. The tour of the TSP is at least as much as the length of MST
4. The optimization techniques yield a solution that is at least as optimal as that of Christofide's.

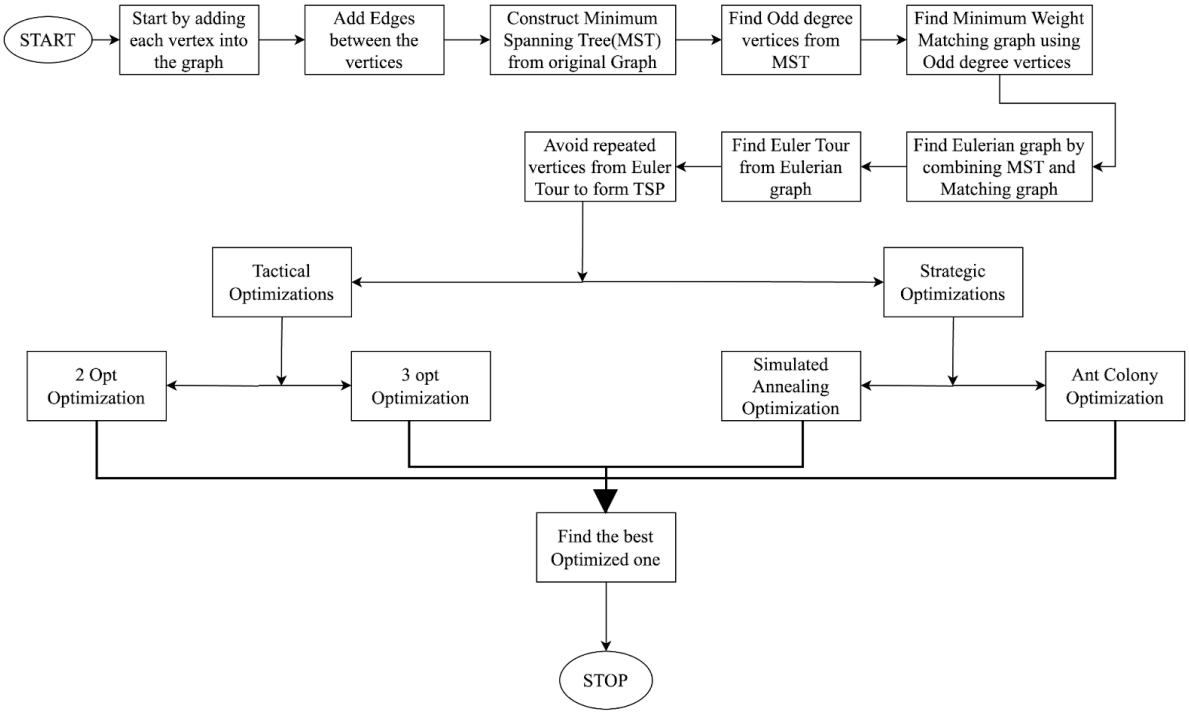
INFO 6205 Spring 2023 Project
Traveling Salesman

FLOW CHARTS:

Below is the Graphical User Interface Flow using JavaFX :



Program flow:



INFO 6205 Spring 2023 Project

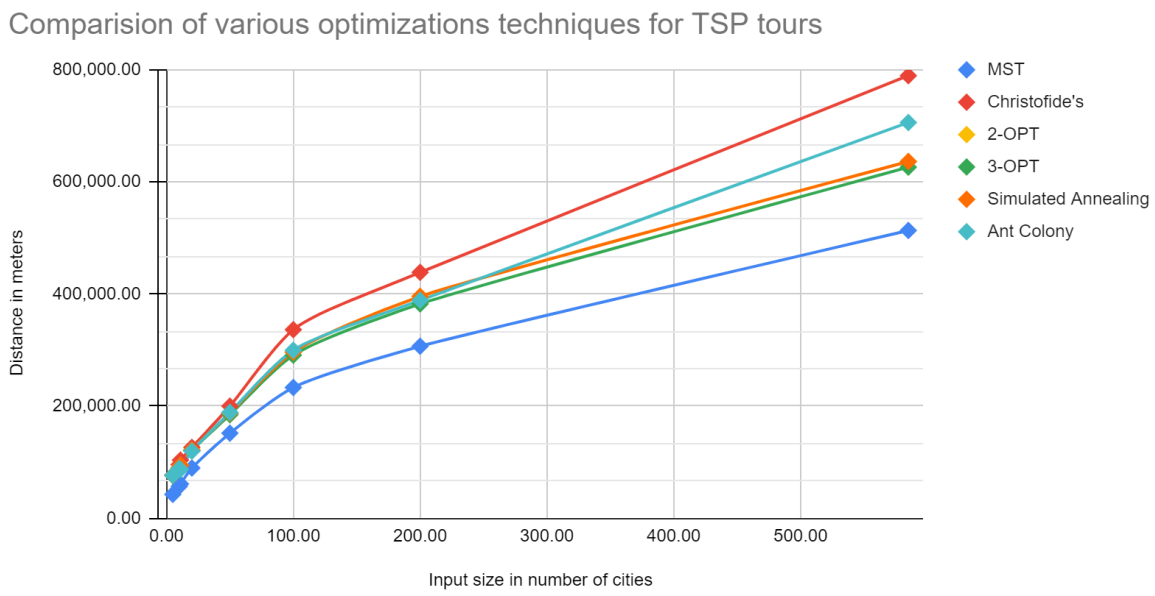
Traveling Salesman

OBSERVATIONS AND GRAPHICAL ANALYSIS:

Below are the observations of tour at different number of cities as compared with different optimization techniques

TSP Tour length							
Input length\Techniques	MST	Brute-Force	Christofide's	2-OPT	3-OPT	Simulated Annealing	Ant Colony
5.00	42,592.25	72,257.82	77,080.60	76,527.99	76,527.99	76,527.99	76,527.99
10.00	56,797.77	86,283.76	95,708.00	89,946.14	86,283.76	89,946.14	88,571.56
11.00	61,037.79	86,675.34	103,722.73	94,361.56	93,675.23	94,361.56	87,031.90
20.00	89501.96	-	126,325.68	121,723.78	120,723.78	121,723.78	119,598.78
50.00	151629.5	-	199,965.56	187,641.06	184,641.06	187,641.06	188,865.89
100.00	233204.26	-	336,417.87	296,233.56	290,830.90	296,233.56	299,675.89
200.00	306776.13	-	438,695.74	395,721.96	382,359.00	395,721.96	388,876.45
585.00	513,326.09	-	789,634.39	636,408.38	626,299.25	625,276.99	705,983.32

Below is the graph comparing different optimizations against the input size measured as the number of cities



RESULTS AND MATHEMATICAL ANALYSIS:

For the whole dataset of 585 points,
the weight of MST is 513326.09 meters.
The initial TSP tour using Christofide's algorithm is 789634.39 meters which is 53.82% more than that of MST
After optimizing the tour using 2-opt technique, the distance comes down to 636408.38 meteres which is 23.97% more than that of MST
After 3-opt the distance is 626299.25, which is 22.08% more than that of MST

INFO 6205 Spring 2023 Project

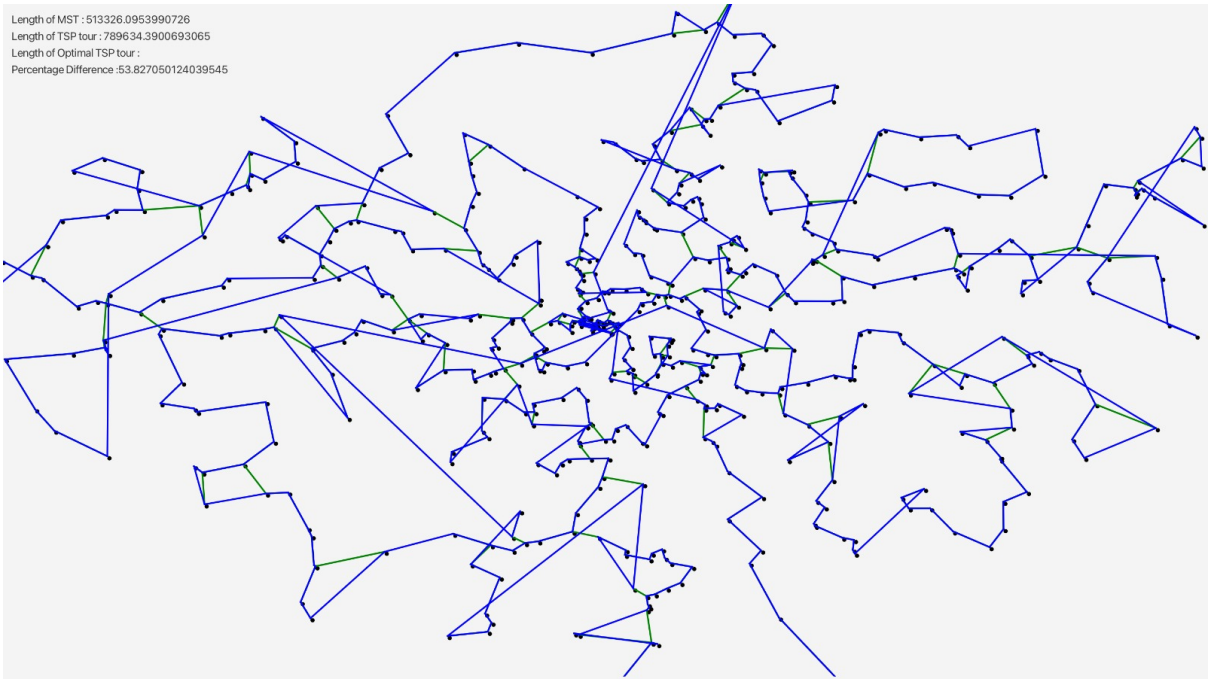
Traveling Salesman

After simulated annealing, the distance is again 636408.38, which is 23.97% more than that of MST
After Ant colony optimization, the distance is 705983.32, which is 37.53% more than that of MST.

So, the best solution obtained after various optimizations is 626299.25 after running 3-opt optimization, which is 22.08% more than MST

Hence, we can conclude that after applying various optimizations on the TSP tour, we can bring down the distance to a value that is somewhere around 10-25% of that of MST.

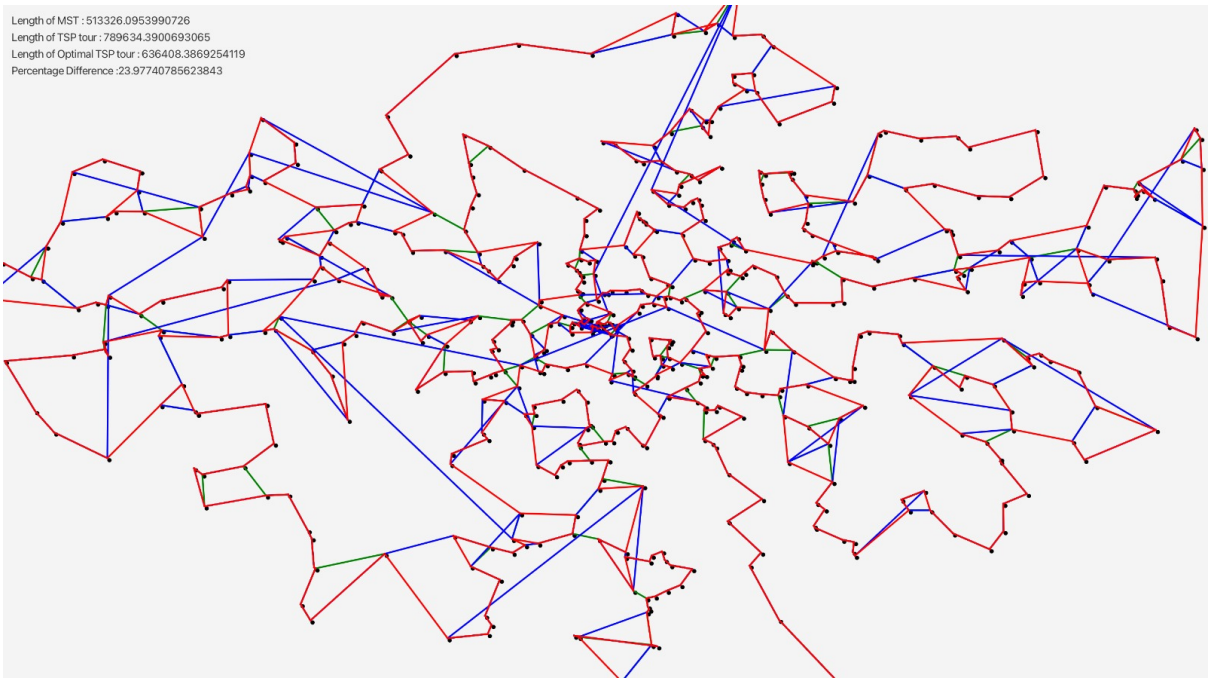
Below are the screenshots of real-time UI generated for different paths using various optimization techniques
TSP using Christofide's algorithm



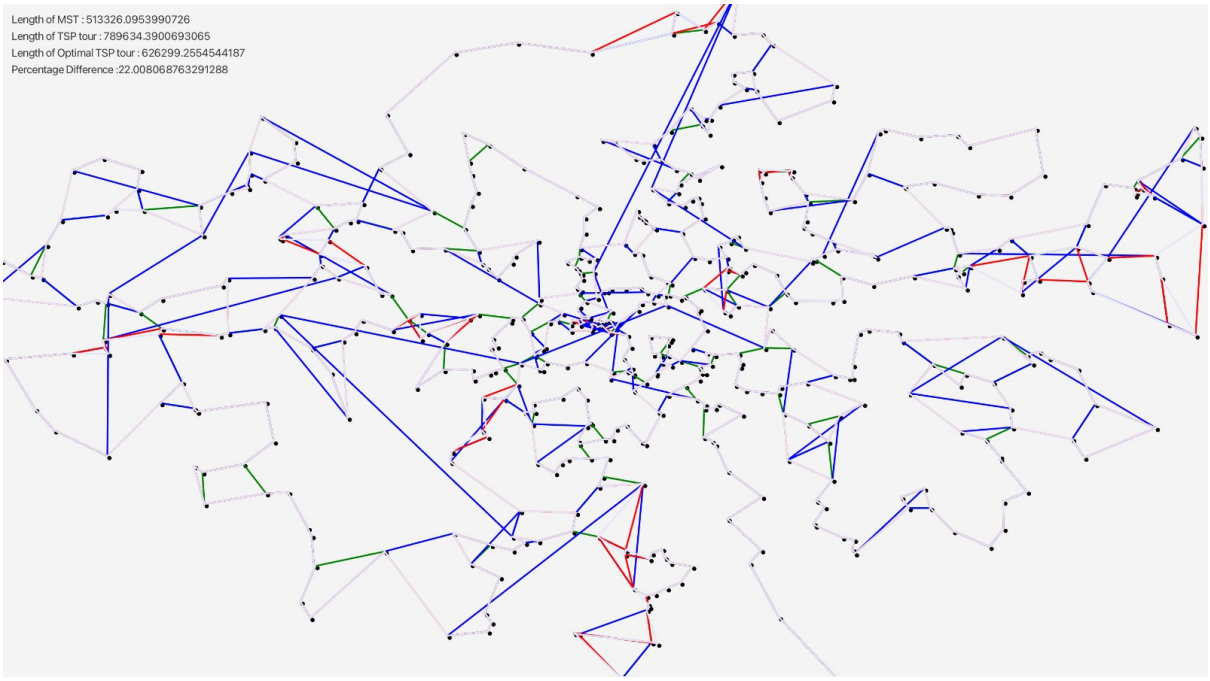
INFO 6205 Spring 2023 Project

Traveling Salesman

TSP after 2-opt optimization



TSP after 3-opt optimization



INFO 6205 Spring 2023 Project

Traveling Salesman

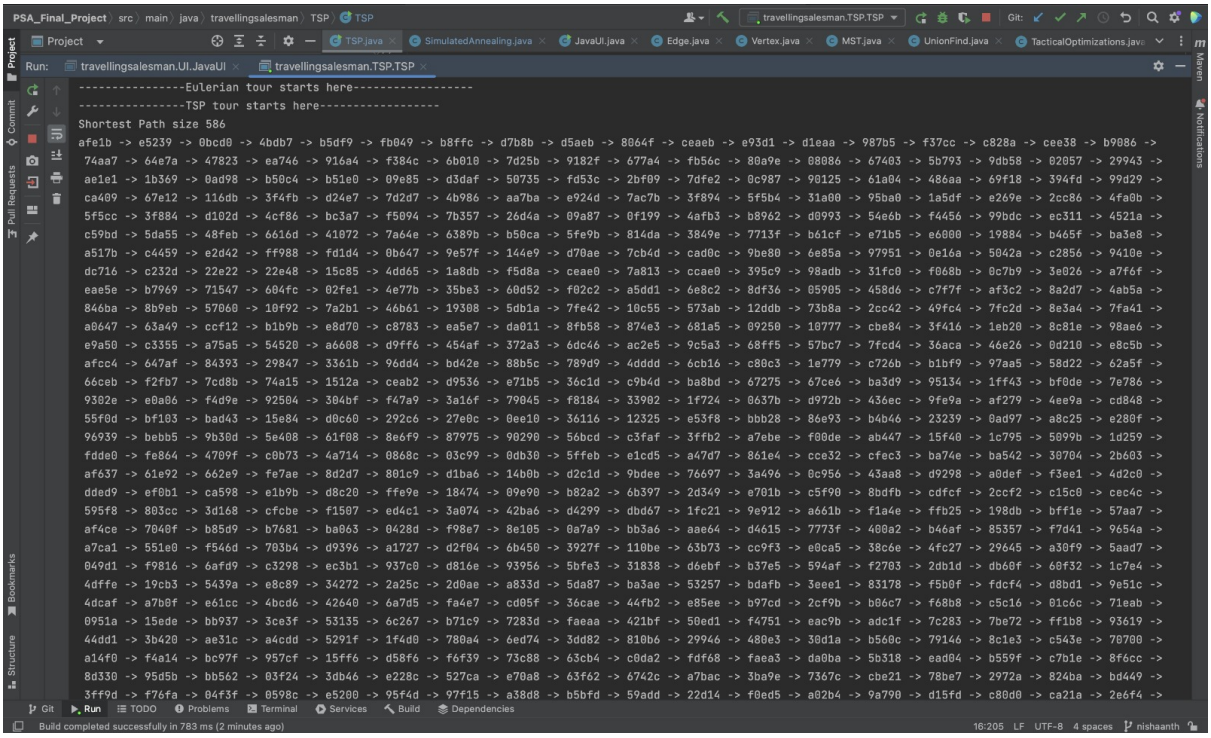
Below are the screenshots of path of TSP for various optimizations

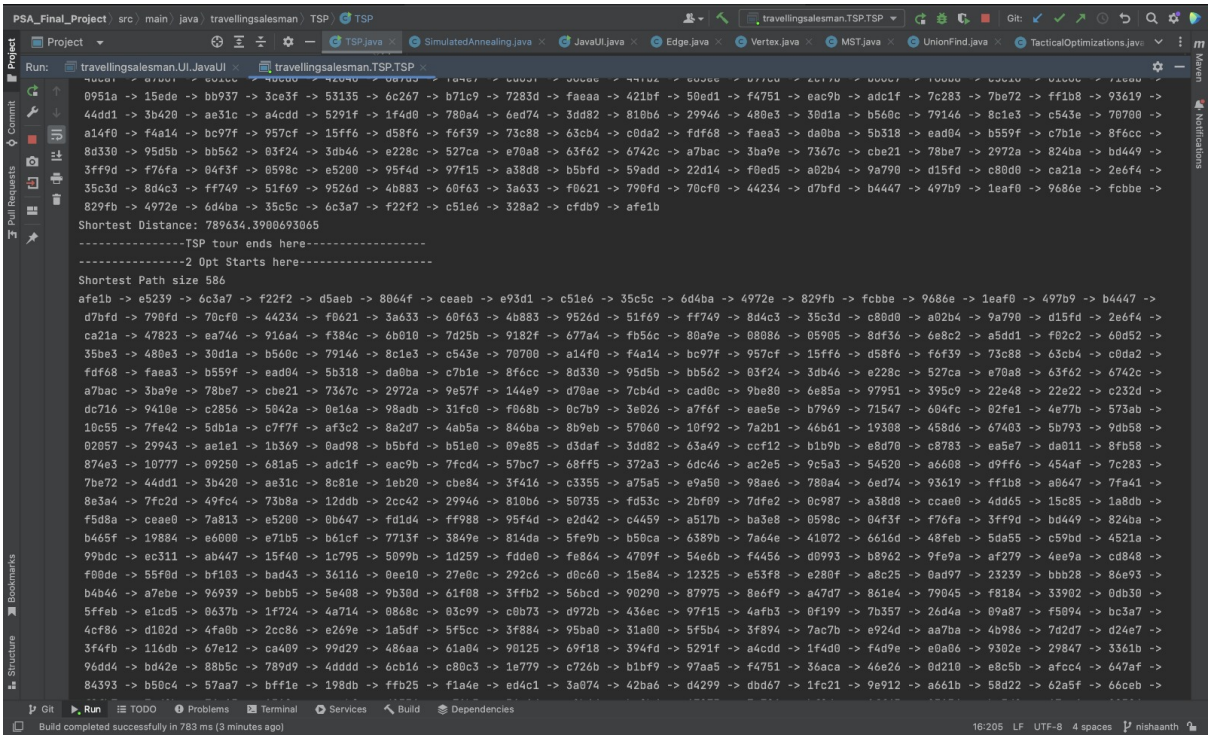
```
/Library/Java/JavaVirtualMachines/jdk-17.0.5.jdk/Contents/Home/bin/java ...
----------------Original Graph starts here------------------
Original graph with 170820 edges and 585 vertices
----------------Original Graph ends here------------------
----------------MST Graph starts here----------------------
Weight of MST 513326.0953990726 vertices 585 edges 584
----------------MST Graph ends here------------------------
Odd-degree vertices size: 232 vertices [1, 2, 3, 4, 5, 519, 521, 522, 12, 525, 14, 528, 529, 18, 19, 20, 533, 22, 23, 24, 25, 538, 539, 28, 540, 29, 544, 33, 545,
 34, 546, 547, 550, 39, 553, 555, 557, 559, 561, 50, 51, 563, 52, 564, 565, 54, 566, 568, 58, 570, 59, 571, 60, 572, 61, 573, 574, 63, 64, 576, 577, 579, 68, 71,
 584, 77, 84, 85, 86, 87, 88, 91, 92, 96, 102, 103, 104, 108, 110, 117, 119, 120, 129, 131, 135, 136, 138, 143, 144, 150, 153, 154, 156, 161, 162, 163, 167, 168,
 169, 176, 177, 179, 180, 182, 184, 187, 188, 192, 195, 196, 198, 199, 202, 208, 209, 214, 216, 218, 220, 227, 232, 235, 243, 246, 248, 250, 253, 256, 259, 263,
 264, 268, 269, 271, 272, 273, 277, 278, 279, 280, 281, 282, 283, 287, 290, 292, 294, 296, 298, 301, 302, 309, 311, 313, 319, 323, 325, 327, 331, 334, 340, 345,
 347, 348, 349, 353, 354, 356, 357, 359, 360, 361, 362, 365, 367, 371, 372, 376, 377, 386, 387, 388, 389, 397, 398, 400, 402, 404, 405, 406, 407, 411, 416, 417,
 421, 422, 423, 424, 425, 428, 431, 433, 438, 439, 441, 448, 456, 457, 459, 460, 461, 462, 466, 467, 468, 469, 476, 478, 481, 483, 484, 489, 490, 492, 494, 497,
 498, 502, 503, 504, 506, 508]
----------------Matching Graph starts here------------------
Matching Graph 232 116
----------------Matching Graph ends here------------------
----------------Eulerian Graph starts here------------------
Eulerian Graph 585 700
----------------Eulerian Graph ends here--------------------
----------------Eulerian tour starts here------------------
[202, 396, 476, 314, 165, 475, 63, 493, 86, 325, 547, 5, 418, 62, 112, 140, 7, 305, 106, 368, 0, 141, 313, 554, 175, 338, 527, 73, 539, 162, 358, 150, 269, 31, 55,
 345, 172, 119, 521, 156, 307, 262, 169, 104, 247, 168, 90, 460, 559, 82, 234, 41, 474, 296, 25, 468, 205, 206, 291, 441, 357, 337, 148, 27, 442, 154, 20, 390,
 216, 388, 378, 469, 433, 93, 37, 316, 333, 412, 277, 16, 198, 259, 320, 370, 328, 489, 332, 443, 123, 495, 243, 576, 520, 444, 461, 462, 283, 538, 128, 551, 381,
 57, 479, 448, 394, 439, 323, 87, 411, 26, 18, 212, 220, 530, 222, 484, 164, 371, 339, 496, 335, 572, 4, 239, 67, 464, 113, 76, 180, 170, 563, 225, 369, 10, 66,
 110, 420, 555, 51, 177, 430, 524, 215, 178, 133, 322, 560, 273, 508, 210, 226, 573, 410, 298, 138, 402, 157, 517, 413, 579, 135, 301, 456, 246, 353, 33, 317, 38,
 422, 159, 97, 519, 34, 373, 532, 533, 103, 545, 289, 485, 98, 235, 503, 432, 342, 143, 261, 160, 360, 447, 399, 543, 191, 250, 161, 174, 290, 504, 183, 182, 466,
 582, 111, 120, 240, 292, 523, 255, 467, 30, 435, 421, 190, 230, 458, 511, 237, 124, 288, 364, 185, 181, 114, 19, 514, 49, 45, 282, 272, 153, 449, 167, 115, 453,
 326, 286, 85, 59, 264, 417, 434, 147, 118, 408, 574, 271, 6, 534, 482, 74, 260, 565, 568, 56, 300, 549, 48, 275, 233, 96, 562, 319, 276, 564, 253, 50, 209, 384,
 242, 367, 480, 304, 197, 350, 64, 184, 145, 256, 416, 132, 231, 224, 17, 12, 203, 274, 251, 389, 268, 245, 570, 398, 404, 94, 193, 270, 23, 58, 340, 208, 163,
 331, 101, 303, 581, 505, 3, 279, 127, 312, 471, 69, 284, 415, 166, 457, 144, 266, 249, 194, 29, 347, 395, 146, 102, 386, 267, 362, 454, 450, 295, 200, 75, 437,
 424, 556, 257, 315, 263, 11, 409, 561, 287, 490, 24, 392, 89, 571, 149, 473, 510, 361, 488, 584, 329, 546, 134, 299, 423, 15, 403, 293, 171, 318, 302, 9, 336,
 513, 321, 379, 122, 498, 387, 109, 72, 537, 21, 451, 186, 324, 553, 455, 445, 187, 158, 515, 343, 61, 427, 414, 382, 22, 569, 207, 2, 425, 341, 506, 32, 501, 95,
```

```
----------------Eulerian Graph ends here--------------------
----------------Eulerian tour starts here------------------
[202, 396, 476, 314, 165, 475, 63, 493, 86, 325, 547, 5, 418, 62, 112, 140, 7, 305, 106, 368, 0, 141, 313, 554, 175, 338, 527, 73, 539, 162, 358, 150, 269, 31, 55,
 345, 172, 119, 521, 156, 307, 262, 169, 104, 247, 168, 90, 460, 559, 82, 234, 41, 474, 296, 25, 468, 205, 206, 291, 441, 357, 337, 148, 27, 442, 154, 20, 390,
 216, 388, 378, 469, 433, 93, 37, 316, 333, 412, 277, 16, 198, 259, 320, 370, 328, 489, 332, 443, 123, 495, 243, 576, 520, 444, 461, 462, 283, 538, 128, 551, 381,
 57, 479, 448, 394, 439, 323, 87, 411, 26, 18, 212, 220, 530, 222, 484, 164, 371, 339, 496, 335, 572, 4, 239, 67, 464, 113, 76, 180, 170, 563, 225, 369, 10, 66,
 110, 420, 555, 51, 177, 430, 524, 215, 178, 133, 322, 560, 273, 508, 210, 226, 573, 410, 298, 138, 402, 157, 517, 413, 579, 135, 301, 456, 246, 353, 33, 317, 38,
 422, 159, 97, 519, 34, 373, 532, 533, 103, 545, 289, 485, 98, 235, 503, 432, 342, 143, 261, 160, 360, 447, 399, 543, 191, 250, 161, 174, 290, 504, 183, 182, 466,
 582, 111, 120, 240, 292, 523, 255, 467, 30, 435, 421, 190, 230, 458, 511, 237, 124, 288, 364, 185, 181, 114, 19, 514, 49, 45, 282, 272, 153, 449, 167, 115, 453,
 326, 286, 85, 59, 264, 417, 434, 147, 118, 408, 574, 271, 6, 534, 482, 74, 260, 565, 568, 56, 300, 549, 48, 275, 233, 96, 562, 319, 276, 564, 253, 50, 209, 384,
 242, 367, 480, 304, 197, 350, 64, 184, 145, 256, 416, 132, 231, 224, 17, 12, 203, 274, 251, 389, 268, 245, 570, 398, 404, 94, 193, 270, 23, 58, 340, 208, 163,
 331, 101, 303, 581, 505, 3, 279, 127, 312, 471, 69, 284, 415, 166, 457, 144, 266, 249, 194, 29, 347, 395, 146, 102, 386, 267, 362, 454, 450, 295, 200, 75, 437,
 424, 556, 257, 315, 263, 11, 409, 561, 287, 490, 24, 392, 89, 571, 149, 473, 510, 361, 488, 584, 329, 546, 134, 299, 423, 15, 403, 293, 171, 318, 302, 9, 336,
 513, 321, 379, 122, 498, 387, 109, 72, 537, 21, 451, 186, 324, 553, 455, 445, 187, 158, 515, 343, 61, 427, 414, 382, 22, 569, 207, 2, 425, 341, 506, 32, 501, 95,
 494, 81, 567, 91, 502, 558, 310, 377, 77, 544, 452, 580, 52, 346, 405, 179, 36, 280, 348, 536, 459, 478, 44, 223, 429, 211, 40, 512, 42, 483, 311, 70, 60, 541,
 470, 79, 385, 88, 285, 83, 71, 136, 406, 566, 578, 142, 176, 351, 199, 236, 487, 43, 8, 155, 525, 497, 548, 53, 244, 65, 238, 376, 426, 241, 334, 528, 214, 472,
 531, 465, 227, 446, 400, 217, 577, 54, 356, 68, 84, 550, 47, 108, 281, 131, 46, 116, 431, 39, 481, 516, 492, 139, 232, 1, 419, 407, 107, 509, 397, 344, 352, 428,
 265, 78, 477, 252, 391, 436, 440, 374, 486, 294, 359, 213, 258, 355, 195, 218, 499, 229, 375, 575, 557, 393, 126, 80, 552, 535, 330, 383, 380, 228, 463, 105, 130,
 529, 248, 129, 491, 254, 526, 219, 189, 354, 201, 14, 297, 192, 278, 309, 117, 518, 327, 188, 438, 372, 507, 221, 363, 121, 401, 542, 173, 308, 35, 196, 365,
 151, 152, 306, 204, 100, 583, 125, 500, 99, 137, 522, 349, 13, 366, 540, 28, 92, 202]
----------------Eulerian tour starts here------------------
----------------TSP tour starts here------------------
Shortest Path size 586
afe1b -> e5239 -> 0bcd0 -> 4bdb7 -> b5df9 -> fb049 -> b8ffc -> d7b8b -> d5aeb -> 8064f -> ceaeb -> e93d1 -> d1eaa -> 987b5 -> f37cc -> c828a -> cee38 -> b9086 ->
 74aa7 -> 64e7a -> 47823 -> ea746 -> 916a4 -> f384c -> 6b010 -> 7d25b -> 9182f -> 677a4 -> fb56c -> 80a9e -> 08086 -> 67403 -> 5b793 -> 9db58 -> 02057 -> 29943 ->
 ae1e1 -> 1b369 -> 0ad98 -> b50c4 -> b51e0 -> 09e85 -> d3daf -> 50735 -> fd53c -> 2bf09 -> 7dfe2 -> 0c987 -> 90125 -> 61a04 -> 486aa -> 69f18 -> 394fd -> 99d29 ->
 ca409 -> 67e12 -> 116db -> 3f4fb -> d24e7 -> 7d2d7 -> 4b986 -> aa7ba -> e924d -> 7ac7b -> 3f894 -> 5f5b4 -> 31a00 -> 95ba0 -> 1a5df -> e269e -> 2cc86 -> 4fa0b ->
 5f5cc -> 3f884 -> d102d -> 4cf86 -> bc3a7 -> f5094 -> 7b357 -> 26d4a -> 09a87 -> 0f199 -> 4afb3 -> b8962 -> d0993 -> 54e6b -> f4456 -> 99bdc -> ec311 -> 4521a ->
 c59bd -> 5da55 -> 48feb -> 6616d -> 41072 -> 7a64e -> 6389b -> b50ca -> 5fe9b -> 814da -> 3849e -> 7713f -> b61cf -> e71b5 -> e6000 -> 19884 -> b465f -> ba3e8 ->
 a517b -> c4459 -> e2d42 -> ff988 -> fd1d4 -> 0b647 -> 9e57f -> 144e9 -> d70ae -> 7cb4d -> cad0c -> 9be80 -> 6e85a -> 97951 -> 0e16a -> 5042a -> c2856 -> 9410e ->
 dc716 -> c232d -> 22e22 -> 22e48 -> 15c85 -> 4dd65 -> 1a8db -> f5d8a -> ceae0 -> 7a813 -> ccae0 -> 395c9 -> 98adb -> 31fc0 -> f068b -> 0c7b9 -> 3e026 -> a7f6f ->
 eae5e -> b7969 -> 71547 -> 604fc -> 02fe1 -> 4e77b -> 35be3 -> 60d52 -> f82c2 -> a5dd1 -> 6e8c2 -> 8df36 -> 05905 -> 458d6 -> c7f7f -> af3c2 -> 8a2d7 -> 4ab5a ->
 846ba -> 8b9eb -> 57060 -> 10f92 -> 7a2b1 -> 46b61 -> 19308 -> 5db1a -> 7fe42 -> 10c55 -> 573ab -> 12ddb -> 73b8a -> 2cc42 -> 49fc4 -> 7fc2d -> 8e3a4 -> 7fa41 ->
 a0647 -> 63a49 -> ccf12 -> b1b9b -> e8d70 -> c8783 -> ea5e7 -> da011 -> 8fb58 -> 874e3 -> 681a5 -> 09250 -> 10777 -> cbe84 -> 3f416 -> 1eb20 -> 8c81e -> 98ae6 ->
```

INFO 6205 Spring 2023 Project

Traveling Salesman





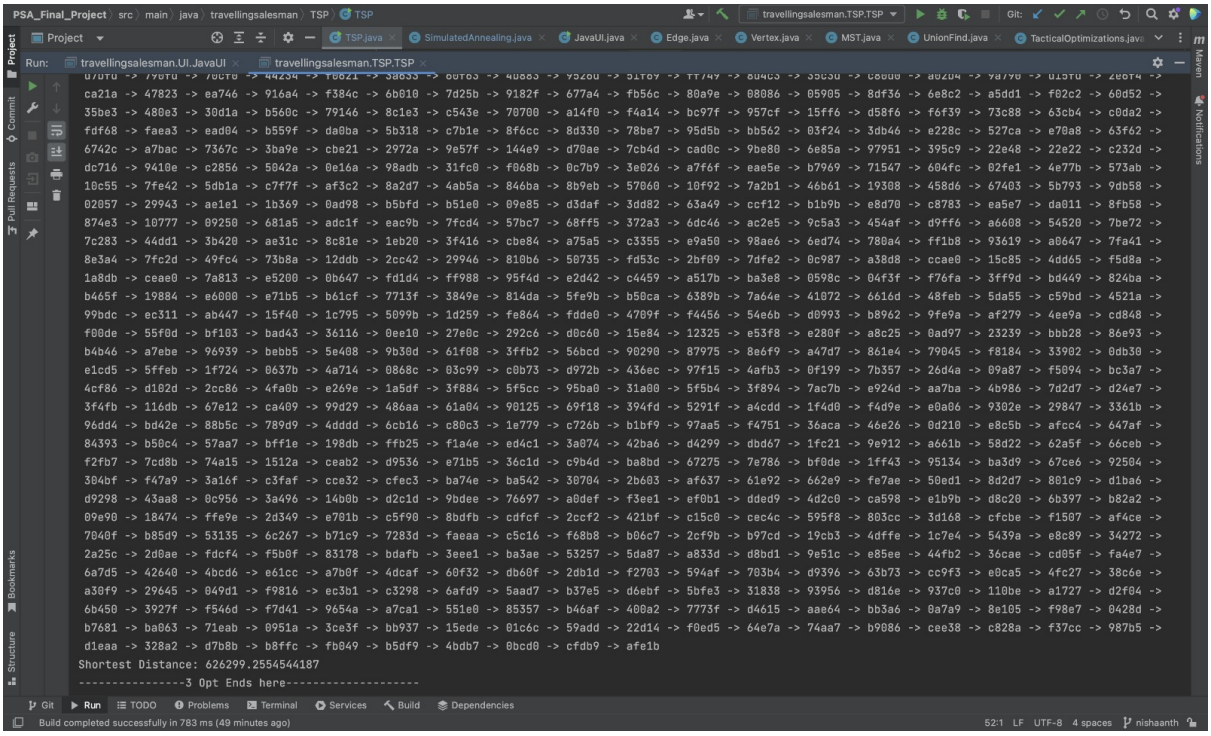
INFO 6205 Spring 2023 Project

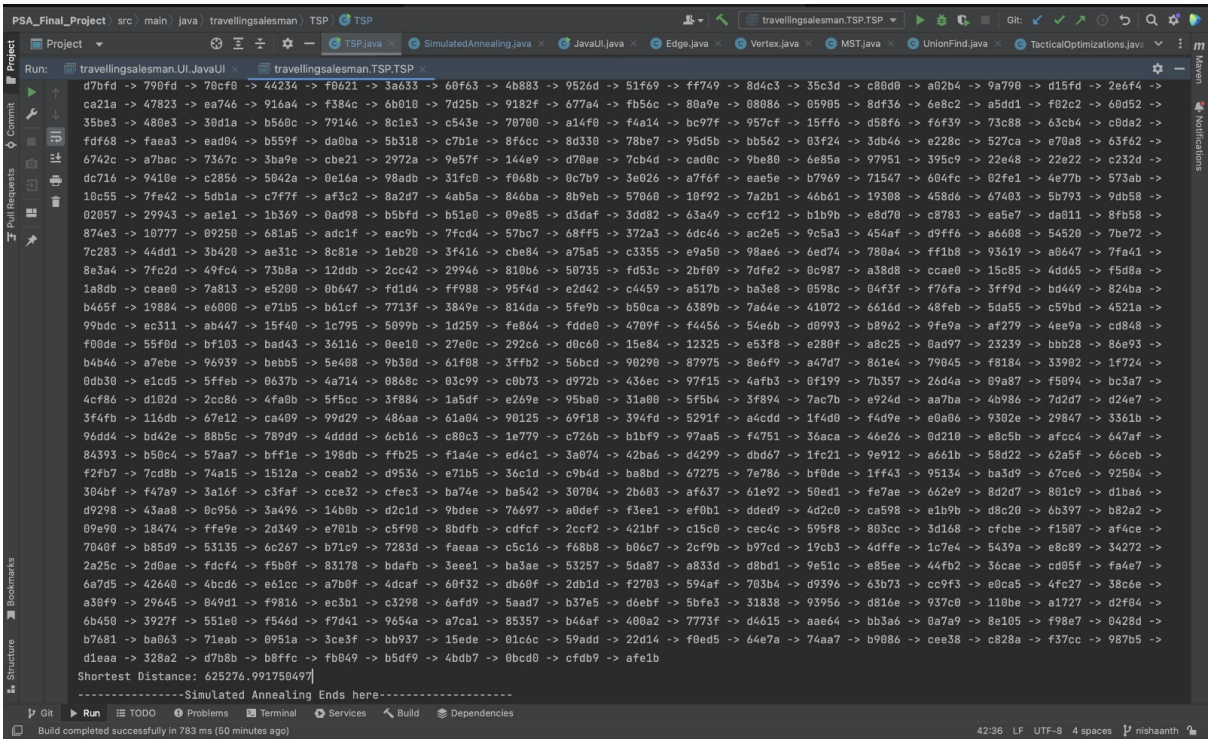
Traveling Salesman

```
f5d8a -> ceae0 -> 7a813 -> e5200 -> 0b647 -> fd1d4 -> ff988 -> 95f4d -> e2d42 -> c4459 -> a517b -> ba3e8 -> 0598c -> 04f3f -> f76fa -> 3ff9d -> bd449 -> 824ba ->
b465f -> 19884 -> e6000 -> e71b5 -> b61cf -> 7713f -> 3849e -> 814da -> 5fe9b -> b50ca -> 6389b -> 7a64e -> 41072 -> 6616d -> 48feb -> 5da55 -> c59bd -> 4521a ->
99bdc -> ec311 -> ab447 -> 15f40 -> 1c795 -> 5099b -> 1d259 -> fdde0 -> fe864 -> 4709f -> 54e6b -> f4456 -> d0993 -> b8962 -> 9fe9a -> af279 -> 4ee9a -> cd848 ->
f00de -> 55f0d -> bf103 -> bad43 -> 36116 -> 0ee10 -> 27e0c -> 292c6 -> d0c60 -> 15e84 -> 12325 -> e53f8 -> e280f -> a8c25 -> 0ad97 -> 23239 -> bbb28 -> 86e93 ->
b4b46 -> a7ebe -> 96939 -> bebb5 -> 5e408 -> 9b30d -> 61f08 -> 3ffb2 -> 56bcd -> 90290 -> 87975 -> 8e6f9 -> a47d7 -> 861e4 -> 79045 -> f8184 -> 33902 -> 0db30 ->
5ffeb -> e1cd5 -> 0637b -> 1f724 -> 4a714 -> 0868c -> 03c99 -> c0b73 -> d972b -> 436ec -> 97f15 -> 4afb3 -> 0f199 -> 7b357 -> 26d4a -> 09a87 -> f5094 -> bc3a7 ->
4cf86 -> d102d -> 4fa0b -> 2cc86 -> e269e -> 1a5df -> 5f5cc -> 3f884 -> 95ba0 -> 31a00 -> 5f5b4 -> 3f894 -> 7ac7b -> e924d -> aa7ba -> 4b986 -> 7d2d7 -> d24e7 ->
3f4fb -> 116db -> 67e12 -> ca409 -> 99d29 -> 486aa -> 61a04 -> 90125 -> 69f18 -> 394fd -> 5291f -> a4cdd -> 1f4d0 -> f4d9e -> e0a06 -> 9302e -> 29847 -> 3361b ->
96dd4 -> bd42e -> 88b5c -> 789d9 -> 4dddd -> 6cb16 -> c80c3 -> 1e779 -> c726b -> b1bf9 -> 97aa5 -> f4751 -> 36aca -> 46e26 -> 0d210 -> e8c5b -> afcc4 -> 647af ->
84393 -> b50c4 -> 57aa7 -> bff1e -> 198db -> ffb25 -> f1a4e -> ed4c1 -> 3a074 -> 42ba6 -> d4299 -> dbd67 -> 1fc21 -> 9e912 -> a661b -> 58d22 -> 62a5f -> 66ceb ->
f2fb7 -> 7cd8b -> 74a15 -> 1512a -> ceab2 -> d9536 -> e71b5 -> 36c1d -> c9b4d -> ba8bd -> 67275 -> 7e786 -> bf0de -> 1ff43 -> 95134 -> ba3d9 -> 67ce6 -> 92504 ->
304bf -> f47a9 -> 3a16f -> c3faf -> cce32 -> cfec3 -> ba74e -> ba542 -> 30704 -> 2b603 -> 61e92 -> af637 -> fe7ae -> 662e9 -> 50ed1 -> 8d2d7 -> 801c9 -> d1ba6 ->
d9298 -> 43aa8 -> 0c956 -> 3a496 -> 14b0b -> d2c1d -> 9bdee -> 76697 -> a0def -> f3ee1 -> ef0b1 -> dded9 -> 4d2c0 -> ca598 -> e1b9b -> d8c20 -> 6b397 -> b82a2 ->
09e90 -> 18474 -> ffe9e -> 2d349 -> e701b -> c5f90 -> 8bdfb -> cdfcf -> 2ccf2 -> 421bf -> c15c0 -> cec4c -> 595f8 -> 803cc -> 3d168 -> cfcbe -> f1507 -> af4ce ->
7040f -> b85d9 -> 53135 -> 6c267 -> b71c9 -> 7283d -> faeaa -> c5c16 -> f68b8 -> b06c7 -> 2cf9b -> b97cd -> 19cb3 -> 4dffe -> 1c7e4 -> 5439a -> e8c89 -> 34272 ->
2a25c -> 2d0ae -> fdcf4 -> f5b0f -> 83178 -> 3eee1 -> bdafb -> 53257 -> ba3ae -> 5da87 -> a833d -> d8bd1 -> 9e51c -> e85ee -> 44fb2 -> 36cae -> cd05f -> fa4e7 ->
6a7d5 -> 42640 -> 4bcd6 -> e61cc -> a7b0f -> 4dcaf -> 60f32 -> db60f -> 2db1d -> f2703 -> 594af -> 703b4 -> d9396 -> 63b73 -> cc9f3 -> e0ca5 -> 38c6e -> 4fc27 ->
29645 -> a30f9 -> 049d1 -> f9816 -> ec3b1 -> c3298 -> 6afd9 -> 5aad7 -> 5bfe3 -> 31838 -> d6ebf -> b37e5 -> 937c0 -> 110be -> 93956 -> d816e -> d2f04 -> 6b450 ->
3927f -> a1727 -> f546d -> f7d41 -> 9654a -> a7ca1 -> 551e0 -> 85357 -> b46af -> 400a2 -> 7773f -> d4615 -> aae64 -> bb3a6 -> 0a7a9 -> 8e105 -> f98e7 -> 0428d ->
ba063 -> b7681 -> 0951a -> 71eab -> 3ce3f -> bb937 -> 15ede -> 01c6c -> 59add -> 22d14 -> f0ed5 -> 64e7a -> 74aa7 -> b9086 -> cee38 -> c828a -> f37cc -> 987b5 ->
d1eaa -> 328a2 -> d7b8b -> b8ffc -> fb049 -> b5df9 -> 4bdb7 -> 0bcd0 -> cfdb9 -> afe1b
Shortest Distance: 636408.3869254119
------------------2 Opt Ends here--------------------
Run   TODO   Problems   Terminal   Services   Build   Dependencies
mpleted successfully in 783 ms (22 minutes ago)                    29:2  LF  UTF-8  4 spaces  nishaanth
```

INFO 6205 Spring 2023 Project

Traveling Salesman





INFO 6205 Spring 2023 Project

Traveling Salesman

```
----------------Ant Colony Optimization Starts here--------------------

Shortest Path size 586
93619 -> 6ed74 -> cfcbe -> 3d168 -> 803cc -> 595f8 -> cec4c -> c15c0 -> 421bf -> 2ccf2 -> cdfcf -> 8bdfb -> c5f90 -> e701b -> 2d349 -> ffe9e -> 18474 -> 09e90 -> b82a2
-> 6b397 -> d8c20 -> e1b9b -> ca598 -> 4d2c0 -> dded9 -> ef0b1 -> f3ee1 -> a0def -> 76697 -> 9bdee -> d2c1d -> 14b0b -> 3a496 -> 0c956 -> 43aa8 -> d9298 -> d1ba6 ->
801c9 -> 8d2d7 -> 50ed1 -> 662e9 -> fe7ae -> af637 -> 61e92 -> 2b603 -> 30704 -> ba542 -> ba74e -> cfec3 -> cce32 -> c3faf -> 3a16f -> f47a9 -> 304bf -> 92504 -> 67ce6
-> ba3d9 -> 95134 -> 1ff43 -> bf0de -> 7e786 -> 67275 -> ba8bd -> c9b4d -> 36c1d -> e71b5 -> d9536 -> ceab2 -> 1512a -> 74a15 -> 7cd8b -> f2fb7 -> 66ceb -> 62a5f ->
58d22 -> a661b -> 9e912 -> 1fc21 -> dbd67 -> d4299 -> 42ba6 -> 3a074 -> ed4c1 -> f1a4e -> ffb25 -> 198db -> bff1e -> 57aa7 -> b50c4 -> 84393 -> 647af -> afcc4 -> e8c5b
-> 0d210 -> 46e26 -> 36aca -> f4751 -> 97aa5 -> b1bf9 -> c726b -> 1e779 -> c80c3 -> 8c81e -> 1eb20 -> cbe84 -> 3f416 -> c3355 -> 98ae6 -> 780a4 -> 3b420 -> 44dd1 ->
7be72 -> 7c283 -> 454af -> d9ff6 -> a6608 -> 54520 -> 9c5a3 -> ac2e5 -> 6dc46 -> 372a3 -> 68ff5 -> 57bc7 -> 7fcd4 -> eac9b -> adc1f -> 681a5 -> 09250 -> 10777 -> 874e3
-> 8fb58 -> da011 -> ea5e7 -> c8783 -> e8d70 -> b1b9b -> ccf12 -> 63a49 -> 3dd82 -> d3daf -> 09e85 -> b51e0 -> b5bfd -> 0ad98 -> 1b369 -> ae1e1 -> 29943 -> 02057 ->
9db58 -> 5b793 -> 67403 -> 458d6 -> 19308 -> 46b61 -> 7a2b1 -> 10f92 -> 57060 -> 8b9eb -> 846ba -> 4ab5a -> 8a2d7 -> af3c2 -> c7f7f -> 5db1a -> 7fe42 -> 10c55 -> 573ab
-> 4e77b -> 02fe1 -> 604fc -> 71547 -> b7969 -> eae5e -> a7f6f -> 3e026 -> 0c7b9 -> f068b -> 31fc0 -> 15ede -> bb937 -> 3ce3f -> 71eab -> 0951a -> b7681 -> ba063 ->
0428d -> f98e7 -> 8e105 -> 0a7a9 -> bb3a6 -> aae64 -> d4615 -> 7773f -> 400a2 -> b46af -> 85357 -> 551e0 -> a7ca1 -> 9654a -> f7d41 -> f546d -> a1727 -> 3927f -> 6b450
-> d2f04 -> d816e -> 93956 -> 110be -> 937c0 -> b37e5 -> d6ebf -> 31838 -> 5bfe3 -> 5aad7 -> 6afd9 -> c3298 -> ec3b1 -> f9816 -> 049d1 -> a30f9 -> 29645 -> 4fc27 ->
38c6e -> e0ca5 -> cc9f3 -> 63b73 -> d9396 -> 703b4 -> 594af -> f2703 -> 2db1d -> db60f -> 22d14 -> f0ed5 -> 64e7a -> 74aa7 -> b9086 -> cee38 -> c828a -> f37cc -> 987b5
-> d1eaa -> 328a2 -> d7b8b -> b8ffc -> fb049 -> b5df9 -> 4bdb7 -> 0bcd0 -> cfdb9 -> b4447 -> d7bfd -> 790fd -> 70cf0 -> 44234 -> f0621 -> 3a633 -> 60f63 -> 4b883 ->
9526d -> 51f69 -> ff749 -> 8d4c3 -> 35c3d -> c80d0 -> a02b4 -> 9a790 -> d15fd -> 2e6f4 -> ca21a -> 47823 -> ea746 -> 916a4 -> f384c -> 6b010 -> 7d25b -> 9182f -> 677a4
-> fb56c -> 80a9e -> 08086 -> 05905 -> 8df36 -> 6e8c2 -> a5dd1 -> f02c2 -> 60d52 -> 35be3 -> 480e3 -> 30d1a -> b560c -> 79146 -> 8c1e3 -> c543e -> 70700 -> a14f0 ->
f4a14 -> bc97f -> 957cf -> 15ff6 -> d58f6 -> f6f39 -> 73c88 -> 63cb4 -> c0da2 -> fdf68 -> faea3 -> b559f -> ead04 -> 5b318 -> da0ba -> c7b1e -> 8f6cc -> 8d330 -> 95d5b
-> bb562 -> 03f24 -> 3db46 -> e228c -> 527ca -> e70a8 -> 63f62 -> 6742c -> a7bac -> 3ba9e -> 78be7 -> cbe21 -> 7367c -> 2972a -> 9e57f -> 144e9 -> d70ae -> 7cb4d ->
cad0c -> 9be80 -> 6e85a -> 97951 -> 395c9 -> 22e48 -> 22e22 -> c232d -> dc716 -> 9410e -> c2856 -> 5042a -> 0e16a -> 98adb -> 6616d -> 41072 -> 7a64e -> 6389b -> b50ca
-> 5fe9b -> 814da -> 3849e -> 7713f -> b61cf -> e71b5 -> e6000 -> 19884 -> b465f -> 824ba -> bd449 -> 3ff9d -> f76fa -> 04f3f -> 0598c -> ba3e8 -> a517b -> c4459 ->
e2d42 -> 95f4d -> ff988 -> fd1d4 -> 0b647 -> e5200 -> 7a813 -> ceae0 -> f5d8a -> 1a8db -> 15c85 -> 4dd65 -> ccae0 -> a38d8 -> 0c987 -> 7dfe2 -> 2bf09 -> fd53c -> 50735
-> 810b6 -> 29946 -> 2cc42 -> 12ddb -> 73b8a -> 49fc4 -> 7fa41 -> a0647 -> ff1b8 -> afe1b -> e5239 -> 6c3a7 -> f22f2 -> d5aeb -> 8064f -> ceaeb -> e93d1 -> c51e6 ->
35c5c -> 6d4ba -> 4972e -> 829fb -> fcbbe -> 9686e -> 1eaf0 -> 497b9 -> b8962 -> d0993 -> f4456 -> 54e6b -> 4709f -> fe864 -> fdde0 -> 1d259 -> 5099b -> 1c795 -> 15f40
-> ab447 -> ec311 -> 99bdc -> 4521a -> c59bd -> 3f4fb -> d24e7 -> 7d2d7 -> 4b986 -> aa7ba -> e924d -> 7ac7b -> 3f894 -> 5f5b4 -> 31a00 -> 95ba0 -> 3f884 -> 5f5cc ->
1a5df -> e269e -> 2cc86 -> 4fa0b -> d102d -> 4cf86 -> bc3a7 -> f5094 -> 09a87 -> 26d4a -> 7b357 -> 0f199 -> 4afb3 -> 97f15 -> 436ec -> d972b -> c0b73 -> 03c99 -> 0868c
-> 4a714 -> 1f724 -> 0637b -> e1cd5 -> 5ffeb -> 0db30 -> 33902 -> f8184 -> 79045 -> 861e4 -> a47d7 -> 8e6f9 -> 87975 -> 90290 -> 56bcd -> 3ffb2 -> 61f08 -> 9b30d ->
5e408 -> bebb5 -> 96939 -> a7ebe -> b4b46 -> 86e93 -> bbb28 -> 23239 -> 0ad97 -> a8c25 -> e280f -> e53f8 -> 12325 -> 15e84 -> d0c60 -> 292c6 -> 27e0c -> 0ee10 -> 36116
-> bad43 -> bf103 -> 55f0d -> f00de -> cd848 -> 4ee9a -> af279 -> 9fe9a -> 4dddd -> 6cb16 -> 486aa -> 61a04 -> 90125 -> 69f18 -> 394fd -> 5291f -> a4cdd -> 1f4d0 ->
f4d9e -> e0a06 -> 67e12 -> 116db -> 9302e -> 29847 -> 3361b -> 96dd4 -> bd42e -> 88b5c -> 789d9 -> b85d9 -> 53135 -> 6c267 -> b71c9 -> 7283d -> faeaa -> c5c16 -> 5da55
-> 48feb -> e9a50 -> a75a5 -> 99d29 -> ca409 -> 2d0ae -> fdcf4 -> f5b0f -> 83178 -> 3eee1 -> bdafb -> 53257 -> ba3ae -> 5da87 -> a833d -> d8bd1 -> 9e51c -> e85ee ->
44fb2 -> 36cae -> cd05f -> fa4e7 -> 6a7d5 -> 42640 -> 4bcd6 -> e61cc -> a7b0f -> 4dcaf -> 60f32 -> b97cd -> 2cf9b -> b06c7 -> f68b8 -> f1507 -> af4ce -> 7040f -> 59add
-> 01c6c -> 34272 -> 2a25c -> 19cb3 -> 4dffe -> 1c7e4 -> 5439a -> e8c89 -> 7fc2d -> 8e3a4 -> ae31c -> 93619
Shortest Distance: 705983.3271667307
```

UNIT TESTS:

We wrote various test cases to test various parts of the code individually

Test to check the static method of computeDistance from graph class for calculating distance between two vertices

MST.java | Vertex.java | TSP.java | sample.csv | JavaUI.java | SimulatedAn... | Graph.java | Tacti

```java
public class TSPTest {
    Vertex v0, v1, v2, v3, v4, v5, v6, v7, v8, v9;

    public TSPTest() {
        v0 = new Vertex("0", -0.009691, 51.483548);
        v1 = new Vertex("1", -0.118888, 51.513075);
        v2 = new Vertex("2", 0.076327, 51.540042);
        v3 = new Vertex("3", -0.418139, 51.500839);
        v4 = new Vertex("4", -0.134987, 51.46327);
        v5 = new Vertex("5", 0.063946, 51.492689);
        v6 = new Vertex("6", -0.198751, 51.542493);
        v7 = new Vertex("7", 0.01742, 51.49214);
        v8 = new Vertex("8", 0.108427, 51.575913);
        v9 = new Vertex("9", -0.097732, 51.559728);
    }

    @Test
    public void testComputeDistance() {
        assertEquals(8241, Graph.computeDistance(v0, v1), 1);
        assertEquals(8241, Graph.computeDistance(v1, v0), 1);
        assertEquals(2106, Graph.computeDistance(v0, v7), 1);
        assertEquals(3221, Graph.computeDistance(v7, v5), 1);
        assertEquals(5334, Graph.computeDistance(v5, v2), 1);
        assertEquals(4564, Graph.computeDistance(v2, v8), 1);
        assertEquals(15869, Graph.computeDistance(v3, v6), 1);
        assertEquals(6420, Graph.computeDistance(v6, v1), 1);
        assertEquals(5389, Graph.computeDistance(v1, v9), 1);
        assertEquals(5649, Graph.computeDistance(v1, v4), 1);
        assertEquals(0, Graph.computeDistance(v1, v1), 0);
    }
```

INFO 6205 Spring 2023 Project

Traveling Salesman

Then, a couple of test cases to check the MST and its properties

```
@Test
public void testMST03() {
    Graph graph = new Graph();
    graph.addVertex(v0);
    graph.addVertex(v1);
    graph.addVertex(v2);
    graph.addVertex(v3);
    graph.addVertex(v4);
    graph.addEdge(v0, v1, Graph.computeDistance(v0, v1));
    graph.addEdge(v0, v2, Graph.computeDistance(v0, v2));
    graph.addEdge(v0, v3, Graph.computeDistance(v0, v3));
    graph.addEdge(v0, v4, Graph.computeDistance(v0, v4));
    graph.addEdge(v1, v2, Graph.computeDistance(v1, v2));
    graph.addEdge(v1, v3, Graph.computeDistance(v1, v3));
    graph.addEdge(v1, v4, Graph.computeDistance(v1, v4));
    graph.addEdge(v2, v3, Graph.computeDistance(v2, v3));
    graph.addEdge(v2, v4, Graph.computeDistance(v2, v4));
    graph.addEdge(v3, v4, Graph.computeDistance(v3, v4));
    Graph mst = MST.computeMinimumWeightSpanningTree(graph, null);
    assertEquals(5, graph.getNumVertices(), 0);
    assertEquals(10, graph.getNumEdges(), 0);
    assertEquals(3, graph.getIndex("3"), 0);
    assertEquals(4, graph.getDegree("0"), 0);
    assertEquals(42592, mst.getWeight(), 1);
    assertEquals(0, MST.findOddDegreeVertices(mst, graph).size() % 2, 0);
}
```

INFO 6205 Spring 2023 Project

Traveling Salesman

A couple of them to test the functionality of UnionFind

```java
@Test
public void TestUF01() {
    UnionFind uf = new UnionFind(7);
    uf.union(1, 2);
    uf.union(3, 4);
    uf.union(5, 6);
    uf.union(1, 4);
    uf.union(5, 2);
    uf.union(6, 1);
    assertEquals(uf.find(1), uf.find(2));
    assertEquals(uf.find(2), uf.find(4));
    assertEquals(uf.find(3), uf.find(4));
    assertEquals(uf.find(5), uf.find(6));
    assertEquals(uf.find(1), uf.find(4));
    assertEquals(uf.find(2), uf.find(5));
    assertEquals(uf.find(1), uf.find(6));
    assertEquals(uf.find(3), uf.find(5));
    assertEquals(uf.find(3), uf.find(6));
}

@Test
public void TestUF02() {
    UnionFind uf = new UnionFind(7);
    uf.union(1, 2);
    uf.union(2, 3);
    uf.union(4, 5);
    uf.union(5, 6);
    uf.union(2, 5);
    assertTrue(uf.find(1) == uf.find(4));
    assertTrue(uf.find(0) != uf.find(6));
}
```

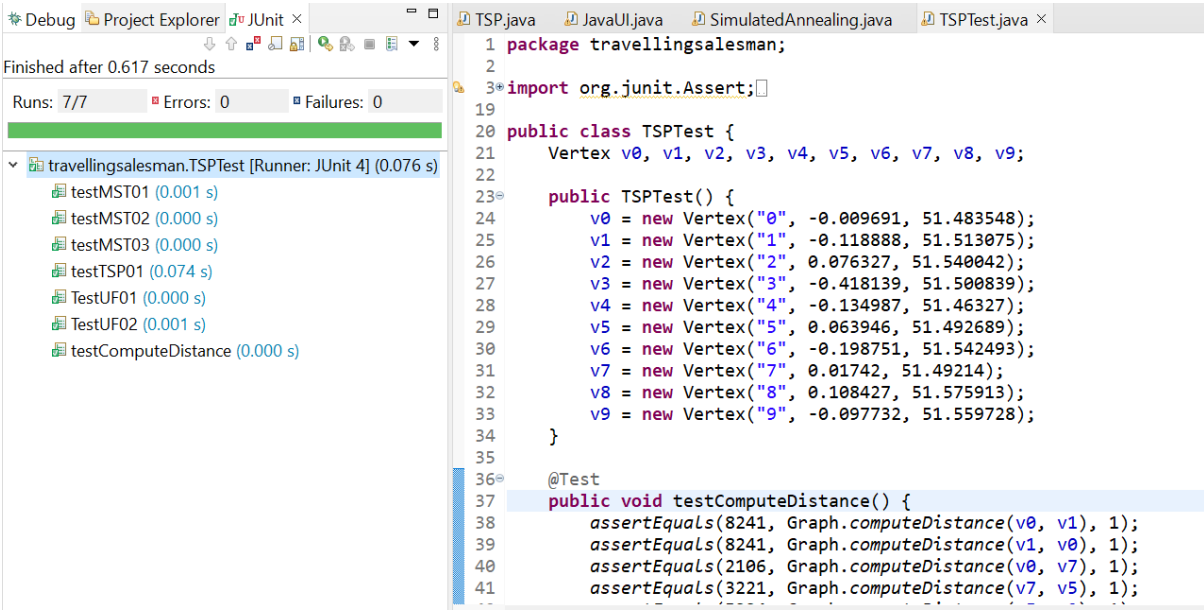
And then, finally, to test the TSP before and after optimizations

```java
public void testTSP01() {
    Graph graph = new Graph();
    graph.addVertex(v0);
    graph.addVertex(v1);
    graph.addVertex(v2);
    graph.addVertex(v3);
    graph.addVertex(v4);
    graph.addEdge(v0, v1, Graph.computeDistance(v0, v1));
    graph.addEdge(v0, v2, Graph.computeDistance(v0, v2));
    graph.addEdge(v0, v3, Graph.computeDistance(v0, v3));
    graph.addEdge(v0, v4, Graph.computeDistance(v0, v4));
    graph.addEdge(v1, v2, Graph.computeDistance(v1, v2));
    graph.addEdge(v1, v3, Graph.computeDistance(v1, v3));
    graph.addEdge(v1, v4, Graph.computeDistance(v1, v4));
    graph.addEdge(v2, v3, Graph.computeDistance(v2, v3));
    graph.addEdge(v2, v4, Graph.computeDistance(v2, v4));
    graph.addEdge(v3, v4, Graph.computeDistance(v3, v4));
    Graph mst = MST.computeMinimumWeightSpanningTree(graph, null);
    Graph matching = MST.findMinimumWeightPerfectMatching(MST.findOddDegreeVertices(mst, graph), graph);
    Graph eulerian = Eulerian.combineGraphs(mst, matching, graph);
    List<Integer> eulerianPath = Eulerian.findEulerianCycle(eulerian);
    double distance = TSP.calculateDistance(eulerianPath, eulerian);
    assertEquals(5, graph.getNumVertices(), 0);
    assertEquals(10, graph.getNumEdges(), 0);
    assertEquals(42592, mst.getWeight(), 1);
    assertEquals(2, MST.findOddDegreeVertices(mst, graph).size(), 0);
    assertEquals(5, eulerian.getNumEdges(), 0);
    assertEquals(5, eulerian.getNumVertices(), 0);
    assertEquals(6, eulerianPath.size(), 0);
    //Testing TSP tour distance
    assertEquals(1,72527/distance, 0.25);
    assertEquals(1,72527/TSP.calculateDistance(TacticalOptimizations.twoOpt(eulerianPath, distance, graph),eul
}
```

INFO 6205 Spring 2023 Project

Traveling Salesman

Screenshot demonstrating the passing of all the test cases



CONCLUSION:

After performing all the optimizations and carefully analyzing the Travelling Salesman Problem, it can be concluded that for a very large data set, no particular algorithm guarantees to find the shortest solution in less than polynomial time. But Christofide's algorithm guarantees that it gives a solution that is 1.5 times larger than the actual solution in the worst case. And then, we can further improve the solution by using various optimization techniques such as two-opt, three-opt, simulated annealing, and ant colony optimization to get to a solution that is most often 10-25% larger than the MST.

REFERENCES:

- https://en.wikipedia.org/wiki/Christofides_algorithm
- https://algs4.cs.princeton.edu/home/
- https://www.youtube.com/watch?v=GiDsjIBOVoA&t=194s
- https://docs.oracle.com/en/java/javase/17/
- https://stackoverflow.com/questions/27928/calculate-distance-between-two-latitude-longitude-points-haversine-formula