Srinivasan Sundar
1217212009

SER 564
Assignment 1

Solution Explanation:

To solve travelling salesman problem I used greedy algorithm, nearest neighbor. I divided my solution into two modules. First Module is DataExtractor Module and second module is TSPSolution Module. Solution is designed in such a way that each module is independent of one another. Each module is separated based on its responsibility and the requirement of the solution. DataExtractor module is responsible for extracting data from the file input and provide the data in a single dimensional array so that it can be used by the TSPSolution module. DataExtractor can process both Symmetric and Asymmetric data and provide the necessary data format. TSPSolution Module interacts with user to get the input file that needs to be processed. It computes a dummy route connecting all the cities and provides the total distance. It also computes the shortest distance to cover all the cities and displays the shortest route and shortest distance. Greedy approach is used in this solution to solve the travelling salesman problem. Since, travelling salesman problem is NP-Hard problem there is no optimal solution in polynomial time. Greedy approach uses nearest neighbor algorithm which can be used to approximately solve this problem in polynomial time. Since greedy algorithm is easy to implement in 1D array this algorithm is been selected.

Information Hiding:

Solution contains two modules. Modules are divided based on the responsibility it should handle. DataExtractor module is responsible for handling the data extraction for ‘tsp’ and ‘atsp’ type of data files. This module reads the contents of file and process it contents based on the file type. This responsibility of the DataExtractor is independent of the TSPSolution module. Any changes related to the way data has to be extracted won’t affect TSPSolution module. Both Asymmetric and Symmetric function provides distances in a single dimensional array so that TSPSolution is not required to know about for what data it is computing the shortest distance. Responsibility of TSPSolution module is to compute the dummy route to connect all the cities and to compute shortest route to connect all the cities. This module is also responsible to interact with the user to get the filename to process and displays the output. Any changes that is related to algorithm of connecting the cities are taken care by the TSPSolution module. TSPSolution module interacts with DataExtractor module by making a static method call of parseData. ParseData function based on the input file call the necessary functions and provides the distance array that is required by TSPSolution module. Since two modules are independent of each other and the information handled by one module is not reflected on another module. By this way, Information hiding is achieved in this solution.

Functional Decomposition:

DataExtractor module is decomposed into smaller functions based on the functionality each function should perform. DataExtractor has 7 functions in total. ParseData function is the calling function of TSPSolution module. This function handles which function needs to be called for which input files. For reading file content we have a function ‘readFile’, which opens a file and returns the content of the file. For parsing Asymmetric and Symmetric data we have different functions. For Asymmetric data, ‘parseAsymmetric’ function parses the distance between the cities and stores in single dimensional Array. For Symmetric data, ‘parseSymmetric’ function parses the points of each city and finds the distance of each city to every other city. This function also provides the distance in single dimensional array so that TSPSolution module is independent of the data that is processed in DataExtractor. EuclideanDistance function is used to calculate the distance between two points, which is used by parseSymmetric function to calculate distance between two function.

TSPSolution module is decomposed into 5 functions. The main function interacts with the user to get the file that needs to be processed. DummyRoute function computes the total distance to connect all the cities, it is not required that this is a shortest distance. NearestNeighbor function computes the shortest distance to connect all the cities and provides the route. It also has the scheduleTask function to maintains a timer so that if the function is running for long time it’ll terminate the function. DisplayOutput function is used by both dummyRoute and nearestNeighbor functions to display their results. Each function interacts with each other by function call and passing the parameters into the function. Each function is decomposed to smaller units based on the functionality.

Quality of Solution:

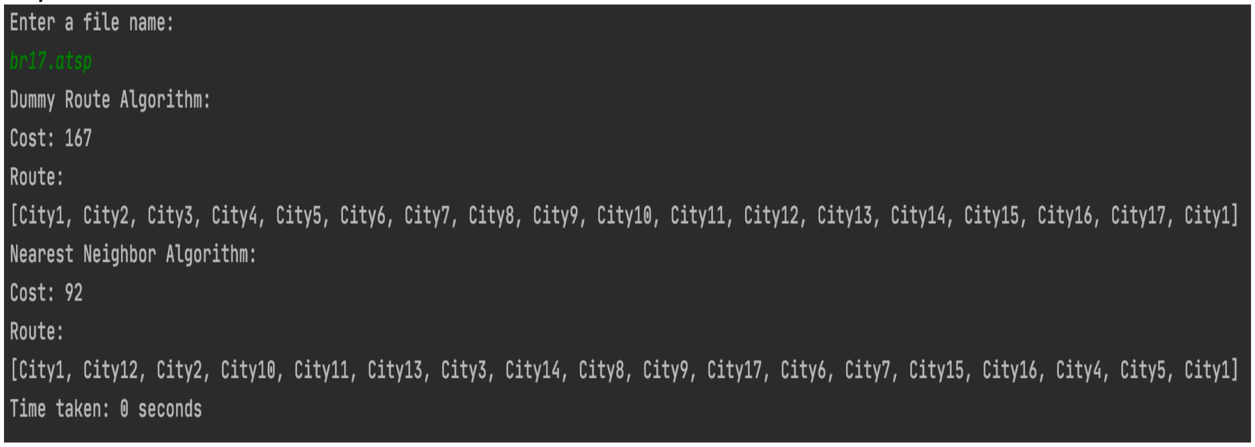
My solution is designed based on modularization and stepwise refinement. In this manner each component in the solution is independent of each other and it’s easy to make changes in the solution. My solution is able to run all the Asymmetric data and provide results in 0 seconds. For Symmetric data, my solution is able to run till 16862 cities and provide results in 4 seconds. For data above 16862 cities I’m getting OutOfMemory Error because the java heap size is enough to allocate the required memory to compute the shortest distance. This error is dependent on the machine it is being executed. So, the largest file that can be processed in my local machine in by this solution is 16862 cities.

Improvements:

Largest file that can be processed by my solution was 16862 cities. For file with 22775 cities I was getting OutOfMemory error. The reason for this is, I was not able to allocate the memory required to store the distance between each city. It is based in how much memory is being allocated to java heap size. My solution is a sequential solution, meaning the solution in run in a single thread. To improve my solution, I need to decompose the solution and utilize threads in java to run the parts of my solution in parallel. By using java threads and dividing the problem into smaller parts, the java heap size required for each thread will be reduced significantly. Another way of improving the solution is by solving Symmetric using different algorithm where distance array is not computed upfront, instead it is computed each time we need the distance between two cities. By this way we’ll handle OutOfMemory error, but it affects the performance of the solution.

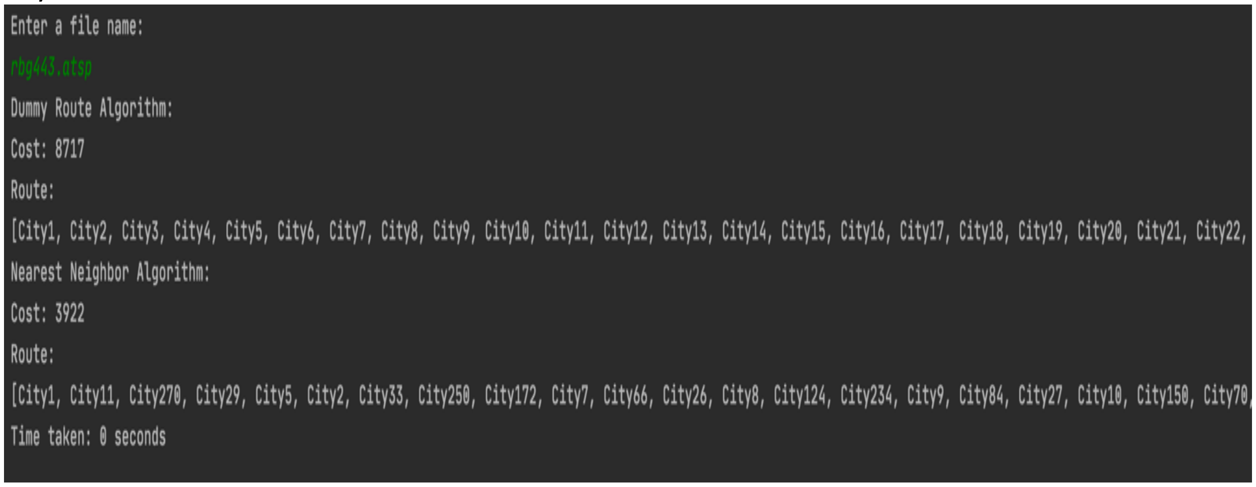
Output:

Asymmetric – 17 Cities



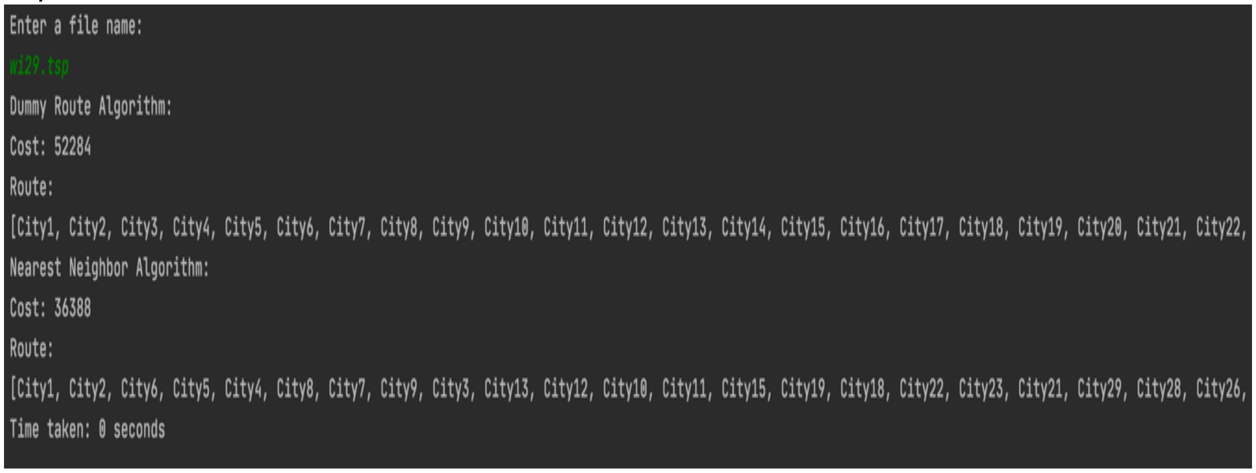
```
Enter a file name:
br17.atsp
Dummy Route Algorithm:
Cost: 167
Route:
[City1, City2, City3, City4, City5, City6, City7, City8, City9, City10, City11, City12, City13, City14, City15, City16, City17, City1]
Nearest Neighbor Algorithm:
Cost: 92
Route:
[City1, City12, City2, City10, City11, City13, City3, City14, City8, City9, City17, City6, City7, City15, City16, City4, City5, City1]
Time taken: 0 seconds
```

Asymmetric – 443 Cities



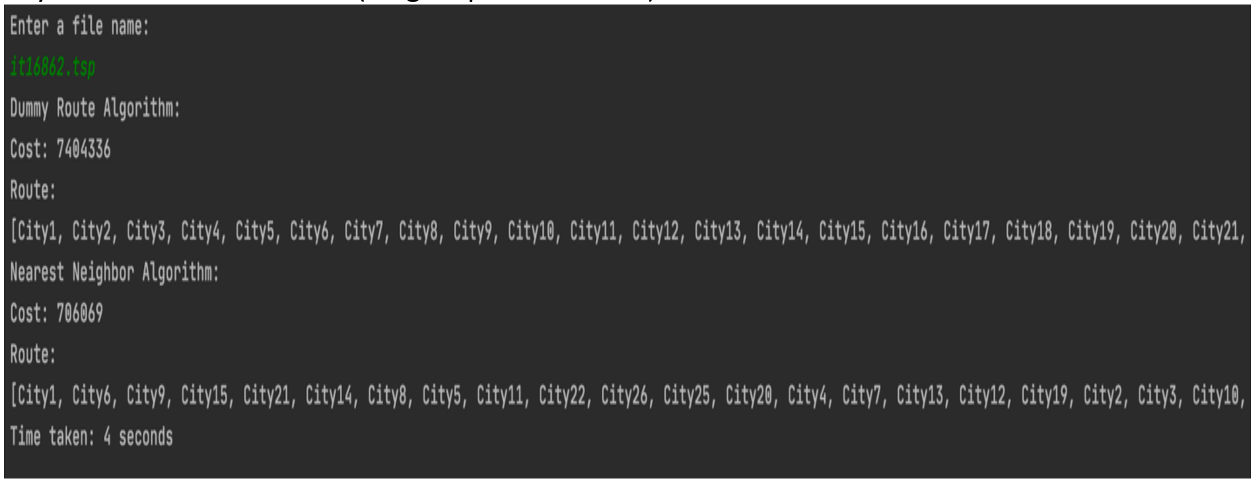
```
Enter a file name:
rbg443.atsp
Dummy Route Algorithm:
Cost: 8717
Route:
[City1, City2, City3, City4, City5, City6, City7, City8, City9, City10, City11, City12, City13, City14, City15, City16, City17, City18, City19, City20, City21, City22,
Nearest Neighbor Algorithm:
Cost: 3922
Route:
[City1, City11, City270, City29, City5, City2, City33, City250, City172, City7, City66, City26, City8, City124, City234, City9, City84, City27, City10, City150, City70,
Time taken: 0 seconds
```

Asymmetric – 29 Cities



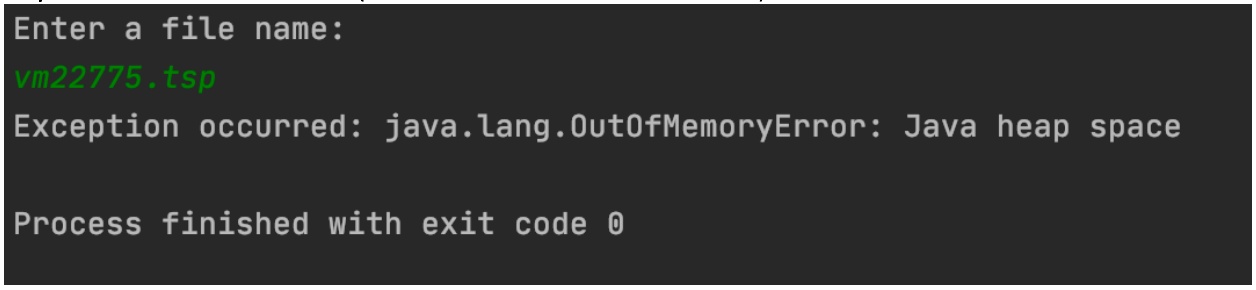
```
Enter a file name:
wi29.tsp
Dummy Route Algorithm:
Cost: 52284
Route:
[City1, City2, City3, City4, City5, City6, City7, City8, City9, City10, City11, City12, City13, City14, City15, City16, City17, City18, City19, City20, City21, City22,
Nearest Neighbor Algorithm:
Cost: 36388
Route:
[City1, City2, City6, City5, City4, City8, City7, City9, City3, City13, City12, City10, City11, City15, City19, City18, City22, City23, City21, City29, City28, City26,
Time taken: 0 seconds
```

Asymmetric – 16862 Cities (Largest processed file)



```
Enter a file name:
it16862.tsp
Dummy Route Algorithm:
Cost: 7404336
Route:
[City1, City2, City3, City4, City5, City6, City7, City8, City9, City10, City11, City12, City13, City14, City15, City16, City17, City18, City19, City20, City21,
Nearest Neighbor Algorithm:
Cost: 706069
Route:
[City1, City6, City9, City15, City21, City14, City8, City5, City11, City22, City26, City25, City20, City4, City7, City13, City12, City19, City2, City3, City10,
Time taken: 4 seconds
```

Asymmetric – 22775 Cities (File where the solution breaks)



```
Enter a file name:
vm22775.tsp
Exception occurred: java.lang.OutOfMemoryError: Java heap space

Process finished with exit code 0
```