Assignment 3 –

//IMP NOTE: This TSP code is not version of Branch&Bound.

// This implementation is of TSP using Reccurrsion.

package AP;

import java.util.\*;

public class TSP {

static int[][] graph; // The graph representation (adjacency matrix)

static int numNodes; // Number of nodes in the graph

static boolean[] visited; // Keep track of visited nodes

static int minCost = Integer.MAX\_VALUE; // Initialize minimum cost

static void tsp(int source, int currPos, int count, int cost) {

if (count == numNodes && graph[currPos][source] > 0) {

minCost = Math.min(minCost, cost + graph[currPos][source]);

return;

}

/\* count == numNodes: Checks if the number of nodes visited so far (count) equals the total number of nodes in the graph (numNodes).

This condition ensures that all nodes have been visited exactly once before evaluating a potential complete tour.

graph[currPos][source] > 0: Verifies if there's a valid edge from the current position (currPos) back to the starting node (source).

This condition ensures that the salesman can return to the starting node if all other nodes have been visited.

If both conditions are met, it means a full tour has been completed, and the algorithm has reached a valid solution.

It calculates the total cost of this tour (including the return to the starting node), compares it to the current minimum cost (minCost),

and updates minCost if this tour's cost is smaller. \*/

for (int i = 0; i < numNodes; i++) {

if (!visited[i] && graph[currPos][i] > 0) {

visited[i] = true;

tsp(source, i, count + 1, cost + graph[currPos][i]);

visited[i] = false;

}

}

/\* The for loop iterates through all the nodes in the graph.

For each node i, it checks if the node hasn't been visited (!visited[i]) and if there exists

a valid edge from the current position (currPos) to node i (graph[currPos][i] > 0).

If both conditions are met, it marks node i as visited (visited[i] = true), updates the count of visited nodes (count + 1),

and the cost incurred by moving from the current position to node i (cost + graph[currPos][i]).

Then it recursively calls the tsp function to explore further paths starting from node i.

After exploring all paths from node i, it marks node i as unvisited (visited[i] = false) to backtrack and

explore other paths from the current node. \*/

}

public static int solveTSP(int[][] adjacencyMatrix) {

numNodes = adjacencyMatrix.length;

graph = adjacencyMatrix;

visited = new boolean[numNodes];

visited[0] = true;

tsp(0, 0, 1, 0);

return minCost;

}

public static void main(String[] args) {

int[][] adjacencyMatrix = {

{0, 10, 15, 20},

{10, 0, 35, 25},

{15, 35, 0, 30},

{20, 25, 30, 0}

};

int minCost = solveTSP(adjacencyMatrix);

System.out.println("Minimum cost for TSP: " + minCost);

}

}