**DELIEVERY ROUTE OPTIMIZATION**

PROJECT REPORT

for

21CSC204J – DESIGN AND ANALYSIS OF ALGORITHMS

*Submitted by*

**RISHIRAJ DUTTA [RA2311026010142]**

**SARNAV BHARDWAJ [RA2311026010164]**

*Under the Guidance of*

Dr. U Sakthi

(Associate Professor, Department of computing technologies)

*In partial fulfillment of the requirements for the degree of*

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE ENGINEERING



DEPARTMENT OF COMPUTING TECHNOLOGIES SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR- 603203

March 2025

**Problem Selection and Description**

#### **Problem Statement: Delivery Route Optimization Using TSP**

Logistics and delivery companies face challenges in optimizing delivery routes to minimize time and costs. The **Travelling Salesman Problem (TSP)** provides an effective mathematical model to determine the most efficient route.

This project aims to develop a **TSP-based Delivery Route Optimizer** using **Python**. The system will take a list of delivery locations, compute the shortest route using optimization algorithms, and output the optimal path along with performance metrics.

#### **Objectives:**

· Accept input via **command-line arguments** or **file-based inputs**.

· Display the computed **optimal route and total distance**.

· Optimize algorithm performance and compare execution times.

· Support different numbers of delivery locations (scalability).

#### **Expected Outcomes:**

* A fully functional **Python-based delivery route optimizer** without a GUI.
* Efficient computation of the **shortest delivery route** using various TSP algorithms.
* Output of optimal route and **performance metrics (distance, time, recursive calls)**.
* Optionally, the results can be **exported to a file or visualized using Matplotlib**.

### **Algorithmic Technique Used: Dynamic Programming (Held-Karp Algorithm)**

#### **Overview of the Dynamic Programming Approach**

The **Delivery Route Optimization** problem, modeled as the **Travelling Salesman Problem (TSP)**, is solved using the **Held-Karp Algorithm**, a **Dynamic Programming** approach that efficiently finds the shortest delivery route.

This algorithm uses a **bitmasking** technique to store previously computed results, reducing redundant calculations and improving efficiency. It follows a **bottom-up approach**, breaking the problem into smaller subproblems and solving them recursively while storing intermediate results for reuse.

The **Held-Karp Algorithm** ensures an **optimal solution** for small to medium-sized delivery problems, making it ideal for optimizing logistics and route planning where efficiency and accuracy are crucial.

### **Steps to Solve the Delivery Route Optimization Problem Using TSP**

#### **Step 1: Read Input & Initialize Locations**

* The user provides **delivery locations** as a list of coordinates (latitude, longitude) or as a **distance matrix**.
* The program initializes a **graph representation**, where each location is a node and the edges represent travel distances between them.

#### **Step 2: Generate All Possible Routes**

* The algorithm considers all possible sequences in which the delivery points can be visited.
* The starting point (warehouse or hub) is fixed, and the remaining locations are permuted to form different possible routes.

#### **Step 3: Compute the Shortest Route Using Dynamic Programming**

* The **Held-Karp Algorithm (Dynamic Programming + Bitmasking)** is used to compute the minimum cost route efficiently.
* A **memoization table** stores previously computed distances to avoid redundant calculations.
* The function checks for the shortest path recursively while keeping track of visited locations.

#### **Step 4: Optimize Route Using Pruning (Optional for Large Inputs)**

* If the number of locations is large (**N > 15**), heuristics like **Nearest Neighbor** or **2-Opt** can be applied to improve performance.
* The algorithm avoids unnecessary computations by **pruning non-optimal paths** early.

#### **Step 5: Solution Found & Display Results**

* Once the shortest route is identified, the program **prints the optimal delivery sequence** and the **total travel distance**.
* Optionally, **performance metrics** like time complexity, number of recursive calls, and execution time can be displayed.
* The final route can be **exported as a text file** or visualized using Matplotlib (optional).

### **Pseudocode for Delivery Route Optimization Using TSP (Dynamic Programming - Held-Karp Algorithm)**

FUNCTION tsp(current\_location, visited\_mask):

IF all locations are visited:

RETURN distance[current\_location][start\_location] // Return to the starting point

IF dp[current\_location][visited\_mask] is already computed:

RETURN dp[current\_location][visited\_mask] // Use stored value to avoid recomputation

min\_cost = INFINITY

FOR each next\_location in locations:

IF next\_location is not visited:

new\_visited\_mask = visited\_mask | (1 << next\_location) // Mark next\_location as visited

cost = distance[current\_location][next\_location] + tsp(next\_location, new\_visited\_mask)

min\_cost = MIN(min\_cost, cost) // Update minimum cost

dp[current\_location][visited\_mask] = min\_cost // Store computed result for future use

RETURN min\_cost

// Main function to initialize the algorithm

FUNCTION main():

READ number\_of\_locations and distance\_matrix

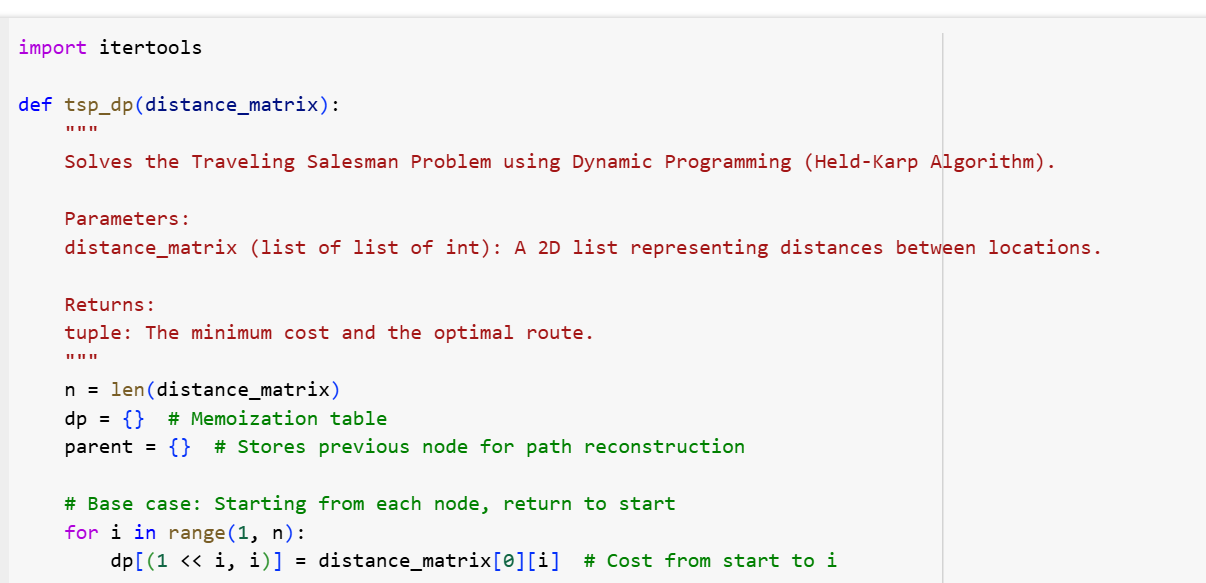
dp = initialize\_2D\_array(number\_of\_locations, 2^number\_of\_locations, -1) // Memoization table

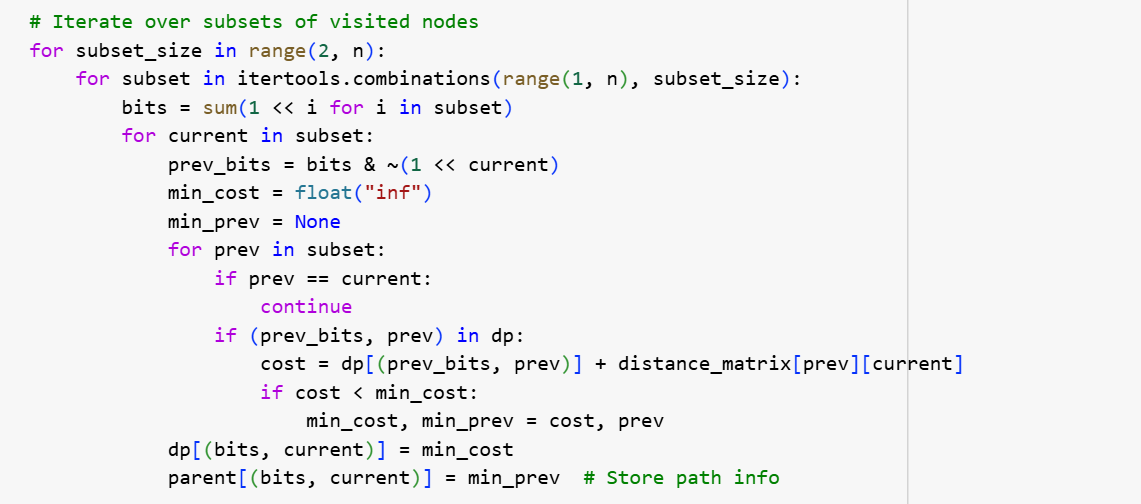
visited\_mask = 1 << start\_location // Start from the first location

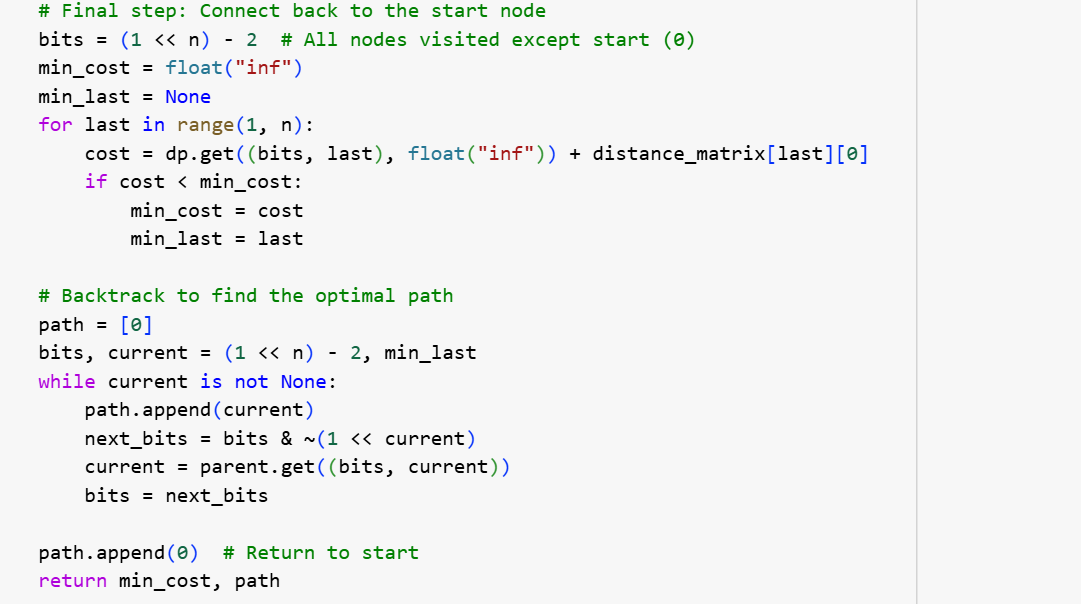
shortest\_route\_cost = tsp(start\_location, visited\_mask)

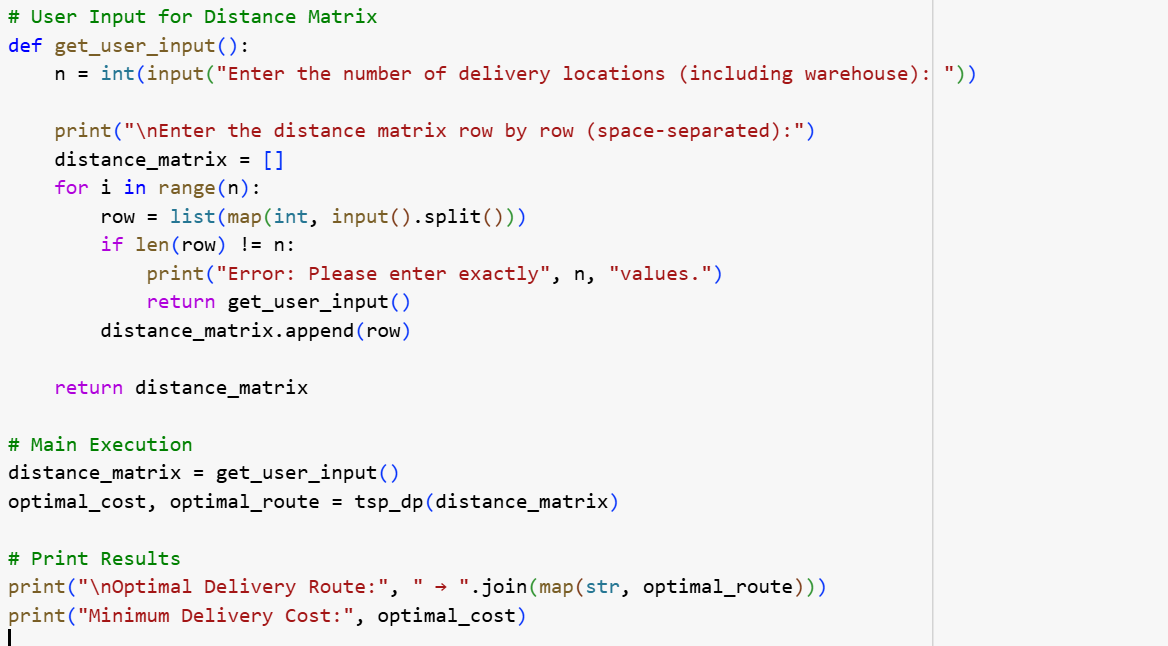
PRINT "Optimal delivery route cost:", shortest\_route\_cost

**Code**:

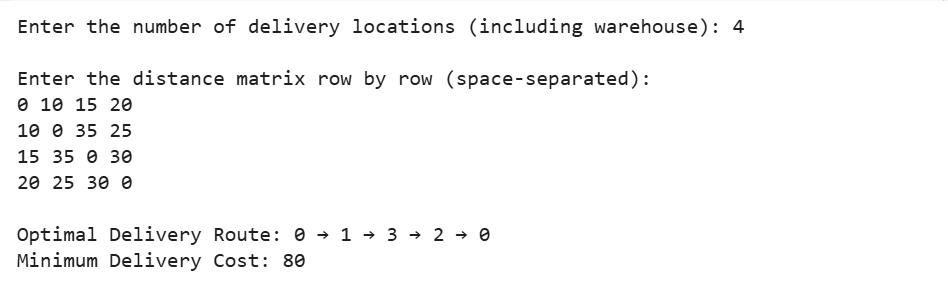








Sample Input and Output:



**Time Complexity Analysis**

### **1. Best Case (Ω(n² × 2ⁿ))**

**When does it occur?**

* If the **distance matrix** has many **zero distances** (e.g., identical locations).
* If an **optimal route is found early**, reducing recursive calls.

**2. Average Case (Θ(n² × 2ⁿ))**

**When does it occur?**

* When the algorithm explores a **balanced number of states** across recursive calls.
* The number of subsets remains **manageable**, and memoization helps optimize redundant calculations.

**3. Worst Case (O(n² × 2ⁿ))**

**When does it occur?**

* When **every subset of nodes** needs to be explored fully.
* **No early terminations** or optimizations can reduce the search space.
* Typically happens when the distance matrix has **random values**, forcing the algorithm to compute **all possible routes**.

The worst-case complexity occurs because:

* There are **2ⁿ subsets of nodes**.
* For each subset, we evaluate **n possible transitions** (trying to visit every city from each state).