Apolonio, Karl Anthony S.

CSAL101 – M017

1. Provide the pseudocode for the Brute Force/Exhaustive Search algorithm used in solving the Traveling Salesman Problem.

1. Start
2. Initialize array cities to ["Charlotte", "Memphis", "Orlando", "Atlanta"]
3. Initialize distances matrix with given distances between cities
4. Function calculate_distance(tour):
    4.1 Initialize total_distance to 0
    4.2 Loop for each city in the tour:
        4.2.1 If it is the last city in the tour:
            4.2.1.1 Add the distance from the current city to the first city to total_distance
        4.2.2 Else:
            4.2.2.1 Add the distance from the current city to the next city to total_distance
    4.3 Return total_distance
5. Generate all possible tours using permutations of cities
6. Initialize shortest_tour to None
7. Initialize shortest_distance to positive infinity
8. Iterate through each tour in all_possible_tours:
    8.1 Calculate the total distance of the current tour using calculate_distance function
    8.2 If the current tour distance is less than the shortest_distance:
        8.2.1 Update shortest_distance with the current tour distance
        8.2.2 Update shortest_tour with the current tour
9. Output the shortest_tour and shortest_distance as the solution
10. End

2. Explain the step-by-step function of the algorithm.

1. Initialize Data

Cities and Distances: The algorithm starts by initializing a list of cities and a matrix of distances between each pair of cities. This matrix is represented as a dictionary of dictionaries in the code, where the outer dictionary's keys are the starting cities, and each associated value is another dictionary mapping destination cities to distances.

2. Define the Distance Calculation Function

calculate_distance(tour) Function: This function calculates the total distance of a given tour. A tour is a sequence of cities. The function iterates through the tour, summing up the distances between consecutive cities. When it reaches the last city in the tour, it adds the distance back to the first city, completing the loop.

3. Generate All Possible Tours

Permutations of Cities: The algorithm uses the permutations function from Python's itertools module to generate all possible arrangements (permutations) of the cities. This step is crucial as the brute force approach examines every possible tour to find the shortest one.

4. Initialize Variables for Tracking the Shortest Tour

Shortest Tour and Distance: Two variables are initialized: shortest_tour to keep track of the tour with the minimum distance found so far, and shortest_distance set to positive infinity. This setup ensures any first calculated tour distance will be shorter, updating these variables with actual values from the tours.

5. Iterate Through Each Possible Tour

For each tour generated in step 3, the algorithm:

Calculates the Tour's Total Distance: Using the calculate_distance function.

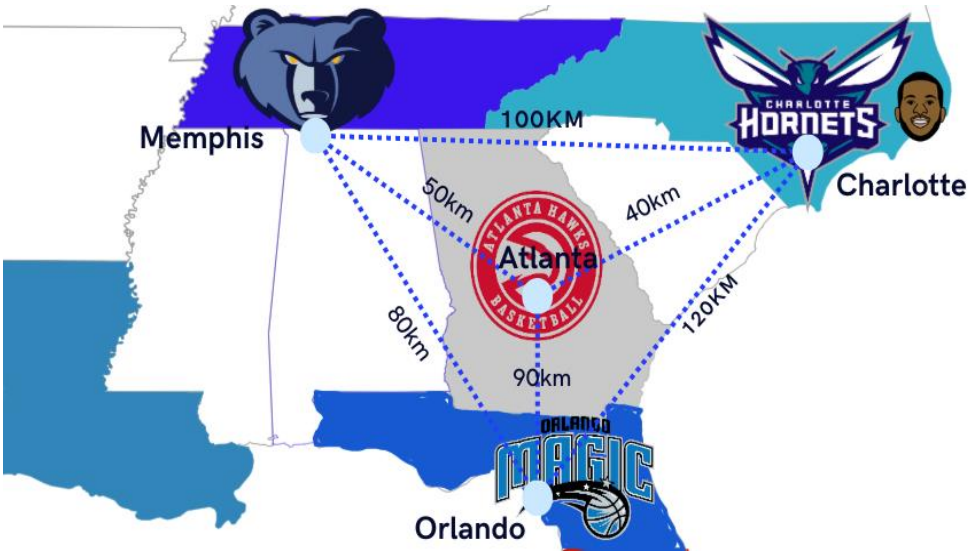
Updates Shortest Tour Information: If the total distance of the current tour is less than the shortest distance recorded so far, the algorithm updates shortest_distance and shortest_tour with the current tour's distance and sequence of cities.

6. Output the Shortest Tour

After examining all possible tours, the algorithm concludes with the shortest tour found and its distance. This result is printed out, showing the optimal path for visiting all cities once and returning to the starting point with the least total distance traveled.

3. Identify a practical application scenario for the Traveling Salesman Problem. Provide comprehensive details or a description of the chosen use case.

Practical Application Scenario: NBA Game Tour Planning with Home Return



Background

Chris Paul, a basketball enthusiast from Charlotte, is excited about watching NBA games in four cities: Charlotte, Orlando, Atlanta, and Memphis. Chris wants to plan an efficient tour that allows him to attend games in each city while also ensuring he returns to his home in Charlotte. His goal is to find the shortest path that minimizes travel costs and time spent on the road.

Scenario Description

Chris has a specific requirement to return home after completing his NBA game tour. This scenario introduces an additional constraint compared to a standard Traveling Salesman Problem (TSP) where the starting and ending points are the same. In this case, Chris needs to find the optimal route that starts and ends in Charlotte, visiting the other three cities in between.

Implementation Details

1. Data Preparation:
   Cities and Distances: Create a distance matrix representing the distances between Charlotte, Orlando, Atlanta, and Memphis. Distances could be based on travel time, driving distance, or cost of transportation.
2. Route Optimization:
   Apply a TSP-solving algorithm or heuristic that accounts for the constraint of returning to the starting point. Algorithms like the Held-Karp algorithm, branch and bound, or dynamic programming can be used to find the optimal route considering the specific constraints.

3. Cost Considerations:
   Integrate cost factors into the optimization process. This could include considerations for transportation costs (e.g., gas or ticket prices), accommodation costs, and any other relevant expenses.
4. Time Constraints:
   If Chris has specific time constraints, such as wanting to catch a game in each city on a particular day or during a specific time window, these constraints should be factored into the optimization process.

Benefits

- Efficient Return to Home: The TSP solution ensures Chris not only visits each city efficiently but also returns home in the most cost-effective manner.
- Cost Savings: Finding the shortest route helps Chris save money on transportation, reducing gas expenses or optimizing the use of public transportation.
- Time Efficiency: Optimizing the route ensures Chris spends less time on the road, allowing him to focus on enjoying the NBA games.
- Personalized Trip: The TSP solution provides a personalized itinerary, ensuring Chris can maximize his NBA game experience over the tour.

4. Transform the algorithm into executable code using a programming language of your choice. Apply the Brute Force Traveling Salesman approach to your selected use case. As an example, consider a scenario within a city where one can navigate from the current location to the destination through various streets. The program should yield the optimal route, determined by the shortest distance.

```python
from itertools import permutations

# Given cities and distances
cities = ["Charlotte", "Memphis", "Orlando", "Atlanta"]
distances = {
    "Charlotte": {"Charlotte": 0, "Memphis": 100, "Orlando": 120, "Atlanta": 40},
    "Memphis": {"Charlotte": 100, "Memphis": 0, "Orlando": 80, "Atlanta": 50},
    "Orlando": {"Charlotte": 120, "Memphis": 80, "Orlando": 0, "Atlanta": 90},
    "Atlanta": {"Charlotte": 40, "Memphis": 50, "Orlando": 90, "Atlanta": 0}
}

# Function to calculate total distance of a tour
def calculate_distance(tour):
    total_distance = 0
    for i in range(len(tour)):
        if i == len(tour) - 1: # if last city, return to first
            total_distance += distances[tour[i]][tour[0]]
        else:
            total_distance += distances[tour[i]][tour[i + 1]]
    return total_distance

# Generate all possible tours
all_possible_tours = permutations(cities)

# Initialize variables for the shortest tour and distance
shortest_tour = None
shortest_distance = float('inf')  # Set to positive infinity initially

# Iterate through all possible tours
for tour in all_possible_tours:
    # Calculate the distance of the current tour
    current_distance = calculate_distance(tour)
```

```python

    # Check if the current tour has a shorter distance than the current shortest
    if current_distance < shortest_distance:
        shortest_distance = current_distance
        shortest_tour = tour

# Print the results
print(f"Shortest tour: {shortest_tour} with a distance of {shortest_distance}km.")
```

5. Provide a detailed explanation of your code, elucidating the logic and methodology behind its implementation.

- Import Required Module

  The code starts by importing the permutations function from the itertools module. This function is crucial for generating all possible permutations of cities, which is essential for the brute force approach to solving the Traveling Salesman Problem (TSP).

- Initialize cities and distances

  The cities are defined in a list (cities), and the distances between each pair of cities are represented in a nested dictionary (distances). The outer dictionary's keys are the starting cities, and the inner dictionary maps destination cities to their respective distances.

- Define the calculation_distance function

  The calculate_distance function takes a tour (sequence of cities) as input and calculates the total distance of the tour. It iterates through the cities, adding the distances between consecutive cities. When the last city is reached, it adds the distance back to the first city to complete the loop.

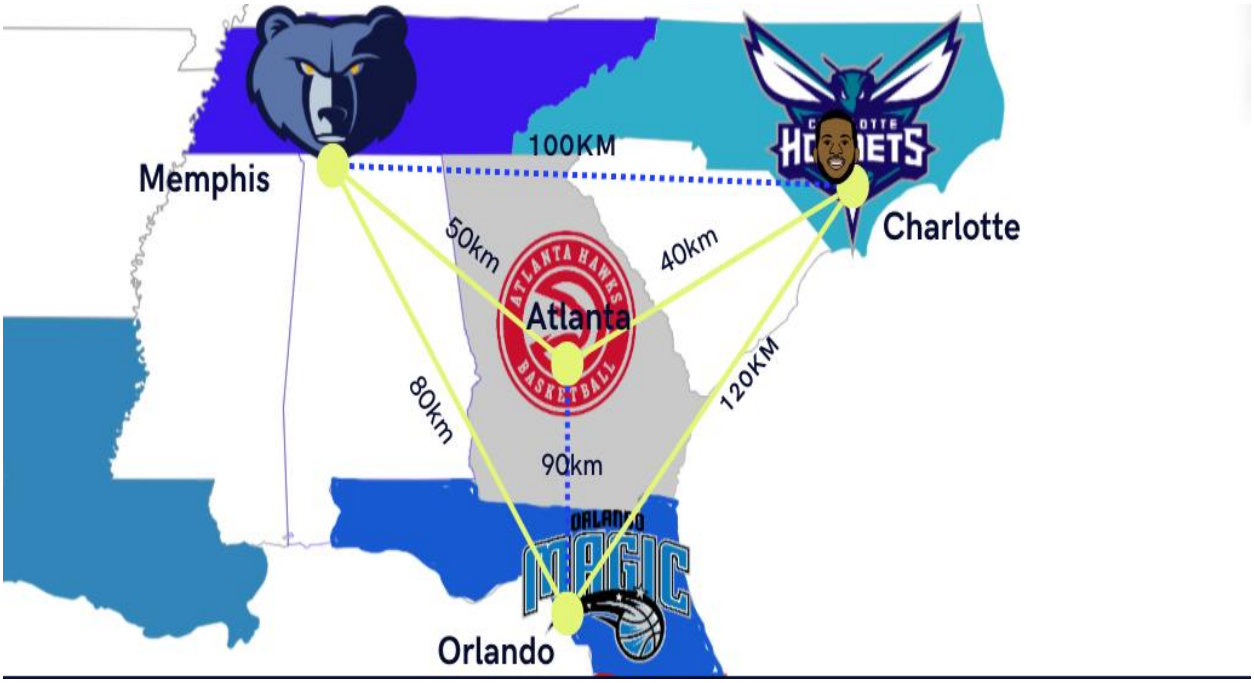
- Generate all possible tours
  The permutation’s function generates all possible arrangements (permutations) of the cities. It returns an iterator containing tuples representing different orders of the cities.

- Initialize variables for tracking the shortest tour
  Two variables (shortest_tour and shortest_distance) are initialized to keep track of the tour with the minimum distance found so far. shortest_distance is set to positive infinity initially to ensure any first calculated tour distance will replace it.

- Iterate through all possible tours
  The code iterates through each tour generated by the permutations. For each tour, it calculates the total distance using the calculate_distance function. If the current tour has a shorter distance than the current shortest tour, the variables shortest_distance and shortest_tour are updated with the values of the current tour.

- Print the results
  Finally, the code prints the result, showing the optimal tour and its total distance.

6. Output




```
[Running] python -u "c:\Users\ADMIN\Documents\Algorithm and Complexity Projects\Chris-Paul-Travel.py"
Shortest tour: ('Charlotte', 'Orlando', 'Memphis', 'Atlanta') with a distance of 290km.

[Done] exited with code=0 in 0.505 seconds
```