Traveller

ID:320210042

Name:Eman Saad Mohamed

Description of algorithm:

The programme is written in Python and has two input parameters: num\_cities, which denotes the number of cities the user wants to visit, and budget, which indicates the total amount of money available for travel. As this problem is extremely similar to the Knapsack problem, where the items to be packed are cities to visit and the weight of each item is the cost of travel to that city, the function employs dynamic programming to solve it using bottom-up approach .

The function begins by reading flights and hotel information from CSV files for various cities. It computes the minimum cost of flights and hotels for each city and stores this information in a list of tuples city\_costs, where each tuple contains the name of a city and its minimum cost of travel. The cities are sorted by cost in ascending order using the sort() method and a lambda function.

The function then initializes a 2D array max\_values with dimensions (num\_cities+1, budget+1), where max\_values[i][j] represents the maximum number of cities that can be visited with a budget of j and considering only the first i cities. The function uses dynamic programming to fill in the max\_values array by considering the maximum number of cities that can be visited by either excluding a city or including it and subtracting its cost from the budget.

The function then traverses the max\_values array to find the cities that can be visited within the given budget. Starting at max\_values[num\_cities][budget], the function follows the path of maximum values to determine which cities were included in the optimal solution. The function appends the names of these cities to the recommended\_cities list.

Finally, the function reverses the order of the recommended\_cities list (since the cities were added in reverse order) and returns it.

The function then traverses the max\_values array to find the cities that can be visited within the given budget. Starting at max\_values[num\_cities][budget], the function follows the path of maximum values to determine which cities were included in the optimal solution. The function appends the names of these cities to the recommended\_cities list.

Finally, the function reverses the order of the recommended\_cities list (since the cities were added in reverse order) and returns it.

Source code:

import numpy as np

import pandas as pd

def recommend\_cities( budget, num\_cities):

    # Load the flights and hotels data

    flights = pd.read\_csv(r"C:\Users\Eman\Documents\projects\Travel Recomendation project\flights.csv")

    hotels = pd.read\_csv(r"C:\Users\Eman\Documents\projects\Travel Recomendation project\hotels.csv")

    # Get the unique list of destination cities from the flights data

    # Define a list of cities and their corresponding costs

    cities = flights['to'].unique()

    costs = np.array([1000, 1080, 9000, 2000, 2500, 800, 700,600, 550])

    # Compute the minimum cost of flights and hotels for each city

    city\_costs = []

    for city in cities:

        city\_flights = flights[flights['to'] == city]

        city\_hotels = hotels[hotels['place'] == city]

        if len(city\_flights) > 0 and len(city\_hotels) > 0:

            min\_flight\_cost = city\_flights['price'].min()

            min\_hotel\_cost = city\_hotels['price'].min()

            city\_costs.append((city, min\_flight\_cost + min\_hotel\_cost))

    # Sort the cities by cost in ascending order

    city\_costs.sort(key=lambda x: x[1])

    print(type(costs))

    # Initialize a 2D array to store the maximum value for each subproblem

    max\_values = np.zeros((num\_cities+1, budget+1))

    print(max\_values)

    # Use dynamic programming to fill in the max\_values array

    for i in range(1, num\_cities+1):

        for j in range(1, budget+1):

            max\_val\_without\_current\_city = max\_values[i-1][j]

            if j >= costs[i-1]:

                max\_val\_with\_current\_city = max\_values[i-1][j-int(costs[i-1])] + 1

                max\_values[i][j] = max(max\_val\_without\_current\_city, max\_val\_with\_current\_city)

            else:

                max\_values[i][j] = max\_val\_without\_current\_city

    print(max\_values)

    # Traverse the max\_values array to find the recommended cities

    recommended\_cities = []

    i = num\_cities

    j = budget

    while i > 0 and j > 0:

        if max\_values[i][j] != max\_values[i-1][j]:

            recommended\_cities.append(cities[i-1])

            j -= costs[i-1]

        i -= 1

    # Reverse the order of the recommended cities list (since we added them in reverse order)

    recommended\_cities.reverse()

    # Return the recommended cities list

    return recommended\_cities

budget = 10000

num\_cities =7

recommended\_cities = recommend\_cities(budget, num\_cities)

print(f"Based on  budget of ${budget} and desire to visit {num\_cities} cities, we recommend the following cities:")

for city in recommended\_cities:

print(city)

Test Cases :

1)budget = 0

num\_cities =7

= will generate empty list.

2)budget = 1000

num\_cities =7

=['Campo Grande (MS)', 'Rio de Janeiro (RJ)', 'Sao Paulo (SP)', 'Aracaju (SE)', 'Natal (RN)', 'Brasilia (DF)', 'Recife (PE)']

4)budget = 5000

num\_cities =5

=Florianopolis (SC)

Recife (PE)

Salvador (BH)