Thursday, 31 August 2017

CSD343

Travelling thief problem

Group No: 17

Group Members: Pallavi Gupta, Vijay Nandwani, Shubhang Arora, Shubhangi Kishore

Problem Statement

There is thief who will be stealing and collecting valuables in ‘N’ cities. There is one item available with price of each item defined as Pi and Volume of each item as Vi. The distance between each city is represented as an adjacency matrix. She intends to visit all the tier two cities and museum with known jewellery, prices and weights. The thief must visit each of the cities exactly once starting from the first city and return back to it in the end.

Any item may be selected as long as the total volume of collected items does not exceed the volume V of the car. She has researched about the location,value and weight of all the items. The goal is to steal maximum value that fits in her car and minimize the fuel consumed by optimizing the routes.

Assumptions

- The products in every city, their weights and prices are known by the thief.
- We are modelling it as a fractional knapsack problem.
- The time required to travel to other cities is fetched from Google API

- The fuel tank is full in the beginning and the tank can be refilled. The capacity of the fuel tank is known.

- The rate of the fuel is known (Rs 68.5/litre).

- There is a constraint on the size of knapsack ---> the car, here we consider the volume.

- The thief has to travel ‘n’ cities and return to the starting city, visiting all cities only once.

- The distances between cities is known, the distance is related to the cost of travelling to the city.

Algorithm

So we have a fully connected graph of cities. We have 28 cities and distance betweenn each city.

The first step we do is calculate the profit matrix. Where profit of each city is calculated using the formula:

profit = Price of the object in city - Cost of travelling to city

*Cost of travelling to city = fuel_per_km * price of fuel * distance (in kms)*

Since it is a fully connected graph, we use the simple greedy approach and adding the item to the car. If the item cannot be accommodated, we take the fraction of the item. Simple greedy works in our case and gives us optimal solution always.

Data Used

- We have a list of tier 2 cities

```
city_names = ["Vijayawada", "Warangal", "Vishapatnam", "Guntur", "Ahmedabad", "Rajkot",
"Jamnagar", "Vadodara", "Surat", "Amravati", "Nagpur", "Aurangabad", "Nashik", "Bhiwandi",
"Pune", "Solapur", "Kolhapur", "Moradabad", "Meerut", "Ghaziabad", "Aligarh", "Agra",
"Bareilly", "Lucknow", "Kanpur", "Allahabad", "Gorakhpur", "Varanasi"]
```

- We use Google Distance Matrix Api to obtain the distances. It looks like:

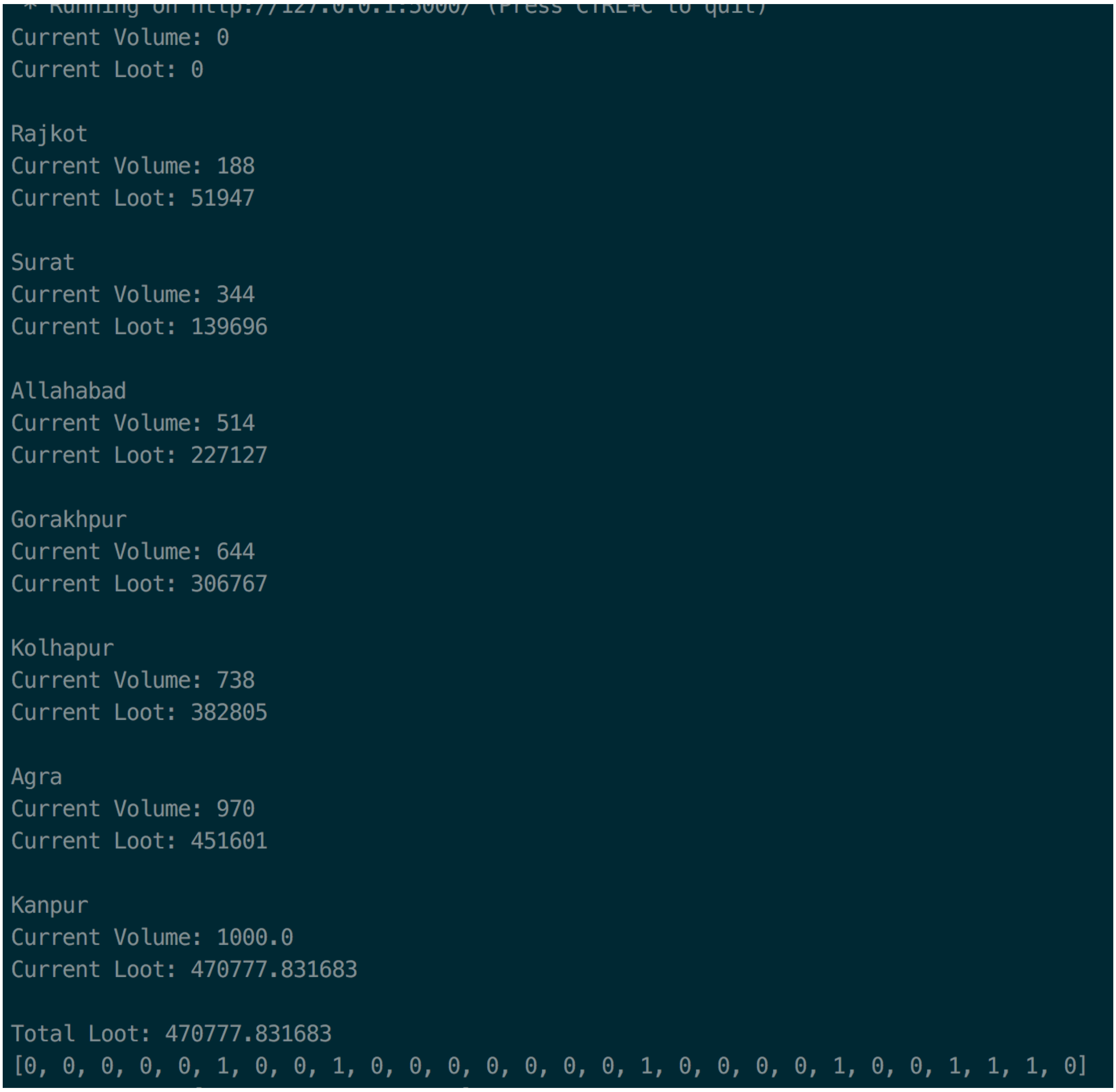
```
city_distance = [
    [0, 347410, 40040, 1437802, 1618678, 1710162, 1324441, 1196375, 779249, 771522,
827982, 949336, 1000383, 844447, 594154, 807806, 1935204, 1861336, 1824298, 1696080,
1604922, 1838182, 1695860, 1608265, 1406205, 2041334, 1512170, 259184],
    [347198, 0, 384953, 1636044, 1816920, 1908404, 1528275, 1540745, 935425, 769019,
1172352, 1293705, 1344753, 1188816, 938524, 1152175, 1857604, 1783736, 1746698,
1616509, 1527322, 1760582, 1325183, 1306906, 1120910, 1963734, 1200194, 499464],
    [38447, 384976, 0, 1469371, 1650246, 1741730, 1356009, 1227944, 810818, 803090,
859551, 980904, 1031952, 876016, 625723, 839374, 1966773, 1892904, 1855867, 1725677,
1636490, 1869751, 1727428, 1639834, 1437774, 2072903, 1543739, 290752],

;
;
;
]
```

- The cost of fuel is known
- The mileage is known
- The list of items and prices are generated randomly.
- The distance travelled is taken from Google API

Analysis and Conclusion

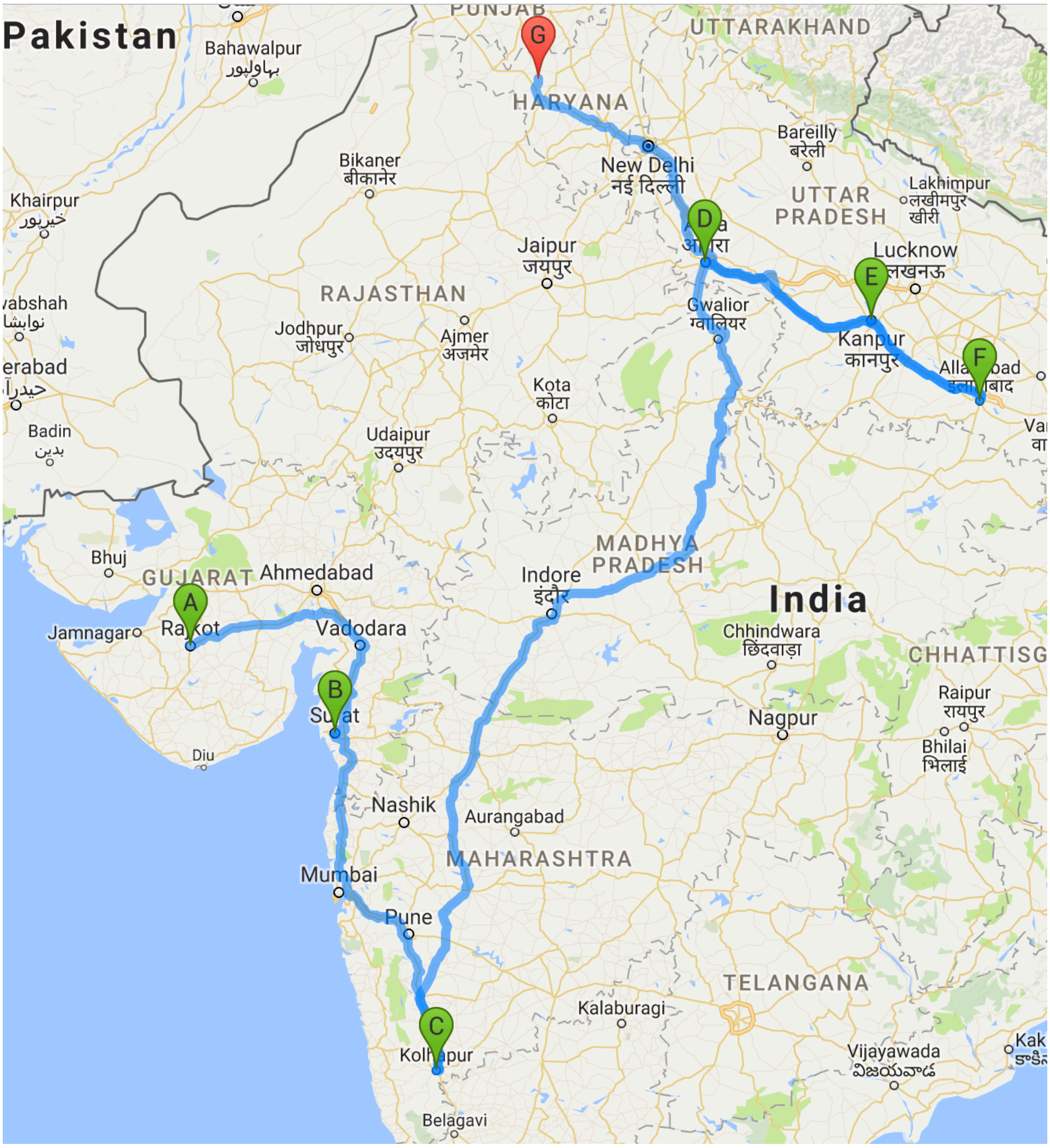
We found that Google Distance matrix API gave different distances from A to B and B to A.

Screenshots



```
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
Current Volume: 0
Current Loot: 0

Rajkot
Current Volume: 188
Current Loot: 51947

Surat
Current Volume: 344
Current Loot: 139696

Allahabad
Current Volume: 514
Current Loot: 227127

Gorakhpur
Current Volume: 644
Current Loot: 306767

Kolhapur
Current Volume: 738
Current Loot: 382805

Agra
Current Volume: 970
Current Loot: 451601

Kanpur
Current Volume: 1000.0
Current Loot: 470777.831683

Total Loot: 470777.831683
[0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0]
```

Thursday, 31 August 2017



Code

Please download the code from: https://github.com/shubhang-arora/Travelling-Thief

```
from flask import Flask, render_template, jsonify
import json
from flask.ext.cors import CORS
from random import randint
import random


city_names = ["Vijayawada", "Warangal", "Vishapatnam", "Guntur", "Ahmedabad", "Rajkot",
"Jamnagar", "Vadodara", "Surat", "Amravati", "Nagpur", "Aurangabad", "Nashik", "Bhiwandi",
"Pune", "Solapur", "Kolhapur", "Moradabad", "Meerut", "Ghaziabad", "Aligarh", "Agra", "Bareilly",
"Lucknow", "Kanpur", "Allahabad", "Gorakhpur", "Varanasi"]

def get_fuel_cost(distance):
  millage = 17.0
  fuel_per_mile = float(1/millage)
  fuel_rate = 68.5/1000

  return float(fuel_per_mile*fuel_rate*distance)

#############################################################################
###########

def initialize_val(item_price, item_vol):
  for i in range(0,len(city_names)):
    x = randint(10000, 90000)
    item_price.append(x)

  for i in range(0,len(city_names)):
    vol = randint(70, 250)
    item_vol.append(vol)

  return item_price,item_vol

#############################################################################
#########

def init_profit(profit, city_distance, num_cities, item_price):

  for row in range (0, num_cities):
    for col in range(0,num_cities):
      if row != col:
        fuel_cost = get_fuel_cost(city_distance[row][col])
        profit[row][col] = item_price[col] - fuel_cost
  return  profit

#############################################################################
###########

def find_next_city(curr_city, profit,visited):

  temp = profit[curr_city]
  max_city = 0
  max_city_index = -1
  count = 0
  #print(temp)
  for i in temp:
    if i>max_city and visited[count]!=1 and count!= curr_city:
      max_city = i
      max_city_index = count
    count+=1
  return max_city_index


#############################################################################
############




app = Flask(__name__)
cors = CORS(app, resources={r"/*": {"origins": "*"}})

@app.route('/hello')
def summary():

    city_distance = [
    [0, 347410, 40040, 1437802, 1618678, 1710162, 1324441, 1196375, 779249, 771522,
827982, 949336, 1000383, 844447, 594154, 807806, 1935204, 1861336, 1824298, 1696080,
1604922, 1838182, 1695860, 1608265, 1406205, 2041334, 1512170, 259184],
    [347198, 0, 384953, 1636044, 1816920, 1908404, 1528275, 1540745, 935425, 769019,
1172352, 1293705, 1344753, 1188816, 938524, 1152175, 1857604, 1783736, 1746698,
1616509, 1527322, 1760582, 1325183, 1306906, 1120910, 1963734, 1200194, 499464],
    [38447, 384976, 0, 1469371, 1650246, 1741730, 1356009, 1227944, 810818, 803090,
859551, 980904, 1031952, 876016, 625723, 839374, 1966773, 1892904, 1855867, 1725677,
1636490, 1869751, 1727428, 1639834, 1437774, 2072903, 1543739, 290752],
    [1438536, 1644591, 1435756, 0, 214521, 306005, 110822, 268843, 734402, 870309, 609698,
479277, 505236, 659893, 912266, 887024, 1104778, 1007150, 971521, 966737, 917955,
1203740, 1175710, 1088115, 1293631, 976514, 1412629, 1183934],
    [1619959, 1826014, 1617178, 214492, 0, 90551, 288607, 450266, 915825, 1051732, 791120,
660699, 686659, 841316, 1093689, 1068447, 1279254, 1181625, 1145996, 1141212, 1092431,
1378216, 1428307, 1300500, 1506015, 1150989, 1625013, 1365357],
    [1710800, 1916855, 1708020, 305334, 90316, 0, 379448, 541107, 1006667, 1142574,
881962, 751541, 777500, 932157, 1184530, 1159289, 1368957, 1271329, 1235700, 1230915,
1182134, 1467919, 1518010, 1429591, 1635107, 1240692, 1754105, 1456199],
    [1325760, 1536794, 1322979, 111228, 288573, 380058, 0, 156067, 626605, 762512, 496921,
366500, 392460, 547117, 799490, 774248, 1165626, 1067998, 1032369, 1027584, 978803,
1264588, 1153682, 1066087, 1271602, 1037362, 1390601, 1071158],
    [1197871, 1545946, 1195091, 267287, 448163, 539647, 153926, 0, 593130, 801861, 369033,
238620, 264579, 419236, 671609, 646367, 1317746, 1220117, 1184489, 1179704, 1130923,
1416708, 1305801, 1218206, 1279087, 1189481, 1398085, 943270],
    [779752, 943867, 776972, 734756, 915632, 1007116, 626987, 591190, 0, 152003, 324867,
509078, 626165, 561866, 517214, 760697, 1261746, 1187877, 1150840, 1020650, 931463,
1164724, 1022401, 934806, 790694, 1367876, 896659, 473131],
    [771862, 778107, 769082, 869893, 1050769, 1142253, 762124, 800410, 152303, 0, 475340,
659551, 776639, 712339, 594085, 911170, 1165076, 1091208, 1054170, 923980, 834794,
1068054, 925732, 838137, 631255, 1271206, 737220, 465242],
    [828838, 1176912, 826058, 611097, 791972, 883456, 497735, 369670, 325931, 474222, 0,
182785, 299872, 235573, 339043, 434404, 1434312, 1336683, 1301055, 1248218, 1159031,
1362354, 1276387, 1188792, 1112913, 1317453, 1218878, 574237],
    [941868, 1289942, 939088, 480878, 661753, 753237, 367516, 239456, 511277, 659567,
182623, 0, 130669, 210970, 425416, 450714, 1436585, 1338956, 1303327, 1250491, 1161304,
1364627, 1278660, 1191065, 1251532, 1319726, 1370530, 759582],
    [998306, 1346380, 995526, 506126, 687001, 778486, 392765, 264705, 628703, 776994,
300049, 132386, 0, 155928, 408301, 383059, 1556585, 1458956, 1423327, 1418543, 1291315,
1494639, 1408672, 1321077, 1381543, 1428320, 1500542, 854354],
    [843675, 1191749, 840895, 661831, 842706, 934190, 548470, 420409, 563768, 712059,
235114, 211720, 156644, 0, 253670, 233782, 1610744, 1513115, 1477486, 1424650, 1335463,
1538786, 1452819, 1365224, 1425691, 1493885, 1544689, 699723],
    [593781, 941855, 591001, 914837, 1095712, 1187196, 801475, 673415, 519954, 594815,
338449, 425228, 409650, 253714, 0, 230389, 1731835, 1634207, 1598578, 1504850, 1415663,
1648924, 1506601, 1419007, 1229499, 1614977, 1335464, 449829],
    [806871, 1154946, 804091, 886919, 1067794, 1159278, 773557, 645497, 762330, 910620,
433676, 448415, 381732, 231363, 230328, 0, 1821925, 1724296, 1688667, 1635831, 1546644,
1749967, 1664000, 1576405, 1549311, 1705066, 1755870, 692637],
    [1940435, 1863401, 1937655, 1098442, 1273029, 1368718, 1158371, 1312390, 1258749,
1161593, 1424999, 1430357, 1548783, 1603662, 1725678, 1815891, 0, 115614, 138356,
135819, 316717, 98108, 380803, 367956, 568675, 375802, 687719, 1633814],
    [1877148, 1800114, 1874367, 1005817, 1180403, 1276093, 1065746, 1219765, 1195461,
1098306, 1332374, 1337732, 1456158, 1511037, 1633052, 1723265, 119777, 0, 46144,
136873, 253429, 218739, 574510, 517431, 722947, 250623, 841945, 1570527],
    [1831118, 1754084, 1828338, 964607, 1139194, 1234883, 1024536, 1178555, 1149431,
1052276, 1291164, 1296522, 1414948, 1469827, 1591842, 1682055, 139120, 45965, 0,
122172, 207399, 238082, 528480, 471401, 676917, 240998, 795915, 1524497],
    [1702210, 1625176, 1699430, 965170, 1139757, 1235447, 1025100, 1179118, 1020524,
923368, 1251010, 1256368, 1415511, 1429673, 1504112, 1641902, 136278, 135885, 126458,
0, 91326, 167730, 438168, 381090, 586605, 355210, 705603, 1395589],
    [1612896, 1535862, 1610116, 917288, 1091875, 1187565, 977218, 1131236, 931210,
834054, 1158280, 1163638, 1292924, 1336943, 1414798, 1549172, 317441, 243572, 206535,
89296, 0, 220163, 335876, 278798, 484313, 423571, 603311, 1306275],
    [1891357, 1814323, 1888577, 1197938, 1372525, 1468214, 1257868, 1411886, 1209671,
1112515, 1376047, 1381405, 1510690, 1554710, 1693259, 1766938, 97785, 215110, 237852,
167386, 222102, 0, 282243, 269396, 470115, 475299, 589160, 1584736],
   [1703969, 1330173, 1701189, 1175606, 1426723, 1522412, 1153305, 1307323, 1022283,
925127, 1274324, 1279682, 1408968, 1452987, 1505872, 1665216, 381562, 567334, 530296,
439235, 335151, 282869, 0, 94507, 200403, 747333, 319447, 1397349],
   [1616442, 1312685, 1613662, 1088079, 1302664, 1436062, 1065777, 1219796, 934756,
837600, 1186796, 1192154, 1321440, 1365460, 1418344, 1577688, 368034, 521626, 484589,
393528, 289443, 269341, 94443, 0, 212364, 701625, 331362, 1309821],
   [1408370, 1131716, 1405590, 1266874, 1481460, 1614858, 1244573, 1279860, 797822,
634963, 1120860, 1252750, 1382036, 1426056, 1230592, 1556690, 569396, 727044, 690006,
598945, 494860, 470703, 200365, 212397, 0, 907042, 130909, 1101749],
   [2057435, 1980401, 2054655, 976785, 1151372, 1247062, 1036715, 1190733, 1375749,
1278593, 1315829, 1321187, 1427126, 1494492, 1616508, 1706721, 380954, 265855, 244841,
363260, 433717, 479916, 754797, 697719, 903234, 0, 1022233, 1750814],
   [1512496, 1199645, 1509716, 1386047, 1600633, 1734030, 1363745, 1399033, 901949,
739090, 1224987, 1371923, 1501209, 1545229, 1334718, 1757457, 688615, 846217, 809179,
718118, 614033, 589922, 319584, 331570, 130843, 1026215, 0, 1205876],
   [247949, 500263, 280022, 1185604, 1366479, 1457963, 1072242, 944177, 473505, 465777,
575784, 759995, 855574, 699637, 449345, 692693, 1629460, 1555592, 1518554, 1388364,
1299178, 1532438, 1390116, 1302521, 1100461, 1735590, 1206426, 0]
   ]


   city_names = ["Vijayawada", "Warangal", "Vishapatnam", "Guntur", "Ahmedabad", "Rajkot",
"Jamnagar", "Vadodara", "Surat", "Amravati", "Nagpur", "Aurangabad", "Nashik", "Bhiwandi",
"Pune", "Solapur", "Kolhapur", "Moradabad", "Meerut", "Ghaziabad", "Aligarh", "Agra", "Bareilly",
"Lucknow", "Kanpur", "Allahabad", "Gorakhpur", "Varanasi"]
   num_cities = len(city_names)
   lat = [16.506174, 17.968901, 17.686816, 16.306652, 23.022505, 22.303894, 22.470702,
22.307159, 21.170240, 20.937424, 21.145800, 19.876165, 19.997453, 19.281255, 18.520430,
17.659919, 16.704987, 28.838648, 28.984462, 28.669156, 27.897394, 27.176670, 28.367036,
26.846694, 26.449923, 25.435801, 29.443817, 25.317645]
   longi = [80.648015, 79.594054, 83.218482, 80.436540, 72.571362, 70.802160, 70.057730,
73.181219, 72.831061, 77.779551, 79.088155, 75.343314, 73.789802, 73.048291, 73.856744,
75.906391, 74.243253, 78.773329, 77.706414, 77.453758, 78.088013, 78.008075, 79.430438,
80.946166, 80.331874, 81.846311, 75.670265, 82.973914]


   profit = [0] *num_cities
   profit = [profit]* num_cities

   item_price = []
   item_vol = []
   item_price, item_vol = initialize_val(item_price,item_vol)


   profit = init_profit(profit,city_distance, num_cities, item_price)


   start_city = randint(0, len(city_names)-1)

   flag = 1

   capacity = 1000.0
   curr_city = start_city
```

```
curr_loot = 0
curr_vol = 0
visited  = [0] * num_cities

output_list = []
for i in range(num_cities):
    temp = [{"pos": -1, "name": city_names[i], "vol": 0, "loot": 0, "lat": lat[i], "long": longi[i]}]
    output_list.append(temp)


#def find_profit(curr_vol, start_city, capacity, curr_loot, visited, item_vol, p):

print("Current Volume: " + str(curr_vol))
print("Current Loot: " + str(curr_loot) + "\n")
pos_count = 1
while(curr_vol<capacity):
    print(city_names[curr_city])
    if (curr_vol + item_vol[curr_city]) < capacity:
    # add everything from city
        output_list[curr_city][0]["pos"] = pos_count
        pos_count = pos_count + 1
        visited[curr_city] = 1
        curr_loot += item_price[curr_city]
        curr_vol += item_vol[curr_city]
        output_list[curr_city][0]["loot"] = curr_loot
        output_list[curr_city][0]["vol"] = curr_vol
        curr_city = find_next_city(curr_city, profit, visited)
        print("Current Volume: " + str(curr_vol))
        print("Current Loot: " + str(curr_loot) + "\n")
    else:
        remain = capacity - curr_vol
        temp = float(item_price[curr_city])/ float(item_vol[curr_city]) * float(remain);
        curr_loot += temp
        curr_vol = capacity
        visited[curr_city] = 1
        output_list[curr_city][0]["pos"] = pos_count
        output_list[curr_city][0]["loot"] = curr_loot
        output_list[curr_city][0]["vol"] = curr_vol
        pos_count = pos_count + 1
        print("Current Volume: " + str(curr_vol))
        print("Current Loot: " + str(curr_loot)+ "\n")

print ("Total Loot: " + str(curr_loot))
print (visited)
# Render template
return render_template('main.html', json = json.dumps(output_list))
```

```
if __name__ == '__main__':
    app.run()
```

main.html file

```
<!DOCTYPE html>
<html>
<head>
    <meta id="my-data" data-name="{{json}}">

    <title></title>
    <style type="text/css">
        html,
body,
#map {
  height: 100%;
  width: 100%;
  margin: 0px;
  padding: 0px
}
    </style>
    <script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false"></
script>
    <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.0/
jquery.min.js"></script>

    <script type="text/javascript">
    var geocoder;
      var js=$('#my-data').data();
      //console.log(js.name);
  var map;
  var directionsDisplay;
  var directionsService = new google.maps.DirectionsService();



   obj = js.name;
    //var obj = JSON.parse(js.name);
    var locations = [];
    obj.forEach(function(item){
        if(parseInt(item[0].pos)!=parseInt(-1))
        {
            var k = [item[0].name, item[0].lat, item[0].long, item[0].pos];
            locations.push(k)
    }
  })

 function initialize() {
  directionsDisplay = new google.maps.DirectionsRenderer();
  var map = new google.maps.Map(document.getElementById('map'), {
   zoom: 10,
   center: new google.maps.LatLng(-33.92, 151.25),
   mapTypeId: google.maps.MapTypeId.ROADMAP
  });
  directionsDisplay.setMap(map);
  var infowindow = new google.maps.InfoWindow();

  var marker, i;
  var request = {
   travelMode: google.maps.TravelMode.DRIVING
  };
  for (i = 0; i < locations.length; i++) {

   marker = new google.maps.Marker({
    position: new google.maps.LatLng(locations[i][1], locations[i][2]),
    map: map,

   });

   google.maps.event.addListener(marker, 'click', (function(marker, i) {
    return function() {
     infowindow.setContent(locations[i][0]);
     infowindow.open(map, marker);
    }
   })(marker, i));
   if (i == 0) request.origin = marker.getPosition();
   else if (i == locations.length - 1) request.destination = marker.getPosition();
   else {
    if (!request.waypoints) request.waypoints = [];
    request.waypoints.push({
     location: marker.getPosition(),
     stopover: true
    });
   }

  }
  directionsService.route(request, function(result, status) {
   if (status == google.maps.DirectionsStatus.OK) {
    directionsDisplay.setDirections(result);
   }
  });
 }

  google.maps.event.addDomListener(window, "load", initialize);

  </script>
</head>
```

```
<body>
<div id="map"></div>

</body>
</html>
```