



You:

Design and create algorithm for finding the optimal routes for delivery agents to deliver parcels from a store to consumers. Requirements and constraints: • There are 100 orders which are to be delivered by 10 delivery agents • There is one store location from where all orders will be delivered and after finishing the deliveries, the agents will come back to the store • You are given 100 random addresses where parcels are to be delivered • Map an optimal route for each delivery person for doing the deliveries and in what order they will do the deliveries • It is not required that all delivery agents will deliver the same number of orders • Each delivery agent can pick multiple orders in one trip. Each agent will deliver at least one order. Assumptions: • All 100 orders are ready at the same time • All delivery agents go at the same speed • Consider straight line distance between any two coordinates Problem: Design optimal routes for all 10 delivery agents so that all packages are delivered and delivery agents return to the store in shortest amount of time - Input Store location Latitude - 28.9428, Longitude - 77.2276 Addresses to deliver <https://zorang-recruitment.s3.ap-south-1.amazonaws.com/addresses.json> - Output Return array of location id each agent will deliver Consider agents at index 0-9 Each subarray will contain addresses id that agent will deliver and in order they deliver. [[1,2,3,9...], [33, 45, 56], ..., [99], [100, 23]] first of explain me this in lay man language then give a example , input and output then tell me how can I make it provide if any resource available on internet



Explanation in Layman's Language

Imagine you're running a delivery service. You have one store where all your products are stored, and you need to deliver 100 orders to different addresses. You have 10 delivery agents to do this job.

Your goal is to figure out the best way to divide these 100 deliveries among your 10 agents so that:

1. All orders are delivered.
2. The total time taken for all deliveries is minimized.
3. Each agent starts from the store, makes their deliveries, and returns to the store.

To do this, you have to consider the location of each delivery address and the store. The challenge is to map out the most efficient routes for each agent so they spend the least amount of time traveling.

Example

Let's simplify with a smaller example:

- **Store Location:** Central Park (Suppose its coordinates are X)
- **Delivery Addresses:** 5 addresses (A, B, C, D, E) around the city
- **Delivery Agents:** 2 agents

We need to find the best way to distribute these 5 addresses between the 2 agents to minimize overall travel time.

Input

- Store Coordinates: X
- Addresses: A, B, C, D, E

Output

This could be something like:

- Agent 1: [A, B, C] - Meaning Agent 1 will deliver to addresses A, B, and C.
- Agent 2: [D, E] - Agent 2 will deliver to addresses D and E.

How to Make It

This problem is a variation of the "Vehicle Routing Problem," a classic problem in logistics and operations research. You can solve it using algorithms that consider various factors like distance, number of deliveries, etc.

Here's a step-by-step approach:

1. **Collect Data:** Get the coordinates of the store and all delivery addresses.
2. **Distance Calculation:** Calculate the distance between each address and between each address and the store.
3. **Optimization Algorithm:** Use an algorithm to assign addresses to each agent in a way that minimizes total travel time. Common algorithms include:
 - Greedy Algorithm: Start with the nearest address and keep adding the next nearest until it's no longer efficient.

- Genetic Algorithm: A more complex approach that simulates natural selection.
- Other algorithms like K-means for clustering addresses and then solving a smaller Traveling Salesman Problem for each cluster.

4. Route Mapping: After the algorithm assigns addresses to each agent, map out the route for each agent.

Resources

- Tutorials and Guides:** Look for online tutorials on Vehicle Routing Problems or Traveling Salesman Problems. These will give you a foundation in the algorithms used for such tasks.
- Programming Libraries:** If you're coding this, look for libraries in programming languages like Python. Libraries like `scipy`, `numpy`, and `networkx` can be helpful.
- Online Courses:** Platforms like Coursera or Udemy offer courses in logistics optimization and algorithms, which can provide more structured learning.

Implementation

This problem requires a combination of geographic information processing and optimization algorithms. You might use programming languages like Python, along with libraries for geospatial data (like `geopandas`) and optimization (like `pulp` or `scipy.optimize`).

By following this approach, you can design a system that maps out efficient routes for your delivery agents, ensuring quick and efficient delivery of all orders.



You:

can you write a code for it ?



ChatGPT:

Certainly! Writing a complete code for this problem involves a few complex steps, but I'll outline a basic version. We'll use Python as it has excellent libraries for this kind of task. The steps will include:

1. Reading the addresses and calculating distances.
2. Using an algorithm to divide these addresses among the delivery agents.
3. Optimizing the delivery route for each agent.

For simplicity, we'll use a basic approach to allocate addresses to agents and optimize routes. More sophisticated methods, like genetic algorithms or machine learning approaches, can provide better results but are much more complex to implement.

First, you'll need to install some libraries, if you haven't already:

```
bash
pip install numpy requests haversine
```

Here's a sample code snippet:

```
python
import numpy as np
import requests
from haversine import haversine

# Function to calculate distance
def calculate_distance(coord1, coord2):
    return haversine(coord1, coord2)

# Load addresses
response = requests.get('https://zorang-recruitment.s3.ap-south-1.amazonaws.com/addresses.json')
addresses = response.json()

# Store coordinates
store_coordinates = (28.9428, 77.2276)

# Calculate distances from the store to each address
distances = []
for address in addresses:
```

```

addr_coordinates = (address['latitude'], address['longitude'])
distance = calculate_distance(store_coordinates, addr_coordinates)
distances.append((address['id'], distance))

# Sort addresses by distance
distances.sort(key=lambda x: x[1])

# Assign addresses to agents
num_agents = 10
agent_routes = [[] for _ in range(num_agents)]

for i, (addr_id, _) in enumerate(distances):
    agent_routes[i % num_agents].append(addr_id)

# Print routes for each agent
for i, route in enumerate(agent_routes):
    print(f"Agent {i+1}: {route}")

```

This code does the following:

- Fetches the addresses from the provided URL.
- Calculates the distance of each address from the store.
- Sorts these addresses by distance.
- Distributes the addresses to each agent in a round-robin fashion.

Please note:

- This approach is quite basic and may not be the most efficient. More complex algorithms like Genetic Algorithms or the Clark-Wright Savings algorithm can be used for better efficiency.
- The code assumes all required libraries are installed and that the address data is in the correct format and accessible.

For more advanced routing optimization, consider exploring dedicated libraries or platforms that specialize in route planning and optimization.



You:

can I plot these coordinates somewhere to see , if yes where give me link of that website



ChatGPT:

Yes, you can plot these coordinates on a map to visually see the locations and routes. There are several online tools and websites where you can do this. Here are a few options:

1. **Google Maps:** You can use Google Maps to manually plot points. This is straightforward but might be time-consuming for a large number of points. You can't directly import a list of coordinates, but you can enter them one by one.

Link: [Google Maps](#)

2. **MapCustomizer:** This website allows you to plot multiple points on a map by pasting in a list of coordinates. It's quite user-friendly for a large number of points.

Link: [MapCustomizer](#)

3. **BatchGeo:** It's another excellent tool for creating maps from a list of addresses or coordinates. You can copy and paste data directly from a spreadsheet.

Link: [BatchGeo](#)

4. **GPS Visualizer:** This website provides a range of tools for plotting GPS data on maps. It can handle data in various formats, including lists of coordinates.

Link: [GPS Visualizer](#)

5. **Python Libraries:** If you're comfortable with programming, Python libraries like folium or matplotlib with basemap can be used to plot and visualize these coordinates programmatically.

To use these tools, you generally need to have your coordinates in a format like latitude, longitude (e.g., 28.9428, 77.2276). You can input these manually or, in some cases, upload a file containing all the coordinates. This will allow you to see all the locations on a map and can

help in visualizing the distribution and planning the routes more effectively.



You:

can you write a python code to plot these lines ?



ChatGPT:

Certainly! To plot the coordinates and the routes, we can use Python's folium library, which is excellent for creating interactive maps. This example will plot the store location, all delivery addresses, and lines representing the routes for each delivery agent.

First, you'll need to install folium if you haven't already:

```
bash
pip install folium
```

Then, you can use the following Python script:

```
python
import folium
import requests

# Load addresses
response = requests.get('https://zorang-recruitment.s3.ap-south-1.amazonaws.com/addresses.json')
addresses = response.json()

# Store coordinates
store_coordinates = (28.9428, 77.2276)

# Create a map centered around the store location
m = folium.Map(location=store_coordinates, zoom_start=12)

# Mark the store location
folium.Marker(store_coordinates, popup='Store', icon=folium.Icon(color='red', icon='info-sign')).add_to(m)

# Assign addresses to agents (assuming you have already done this and have a list `agent_routes`)
# Example: agent_routes = [[1, 2, 3], [4, 5, 6], ...]
num_agents = 10
agent_routes = [[] for _ in range(num_agents)]
for i, address in enumerate(addresses):
    agent_routes[i % num_agents].append(address['id'])

# Function to find address by id
def find_address_by_id(addr_id):
    return next(addr for addr in addresses if addr['id'] == addr_id)

# Plotting routes for each agent
for i, route in enumerate(agent_routes):
    last_coords = store_coordinates
    for addr_id in route:
        addr = find_address_by_id(addr_id)
        addr_coords = (addr['latitude'], addr['longitude'])
        folium.Marker(addr_coords, popup=f'Address {addr_id}', icon=folium.Icon(color='blue')).add_to(m)
        folium.PolyLine([last_coords, addr_coords], color='blue').add_to(m)
        last_coords = addr_coords
    # Connecting back to store
    folium.PolyLine([last_coords, store_coordinates], color='blue').add_to(m)

# Save the map to an HTML file
m.save('delivery_routes_map.html')
```

This script does the following:

- Fetches the addresses from the given URL.
- Initializes a map centered around the store.
- Adds markers for the store and each delivery address.
- Assigns addresses to agents (this part assumes you have already divided the addresses among the agents, as discussed in the previous message).
- Draws lines (routes) for each agent from the store to each assigned address and back to the store.

After running this script, it will generate an HTML file named `delivery_routes_map.html`. You can open this file in a web browser to view the interactive map with your plotted routes.

Note: This script is a basic example and plots direct lines between points. In real-world scenarios, you would use routing APIs to get actual travel routes.



You:

File "f:\Projects\Zorang.py", line 22, in <module> agent_routes[i % num_agents].append(address['id']) KeyError: 'id' PS F:\Projects>



ChatGPT:

The error you're encountering, `KeyError: 'id'`, suggests that the dictionary address does not have a key named `'id'`. This could happen if the structure of the data in the JSON file you are fetching is different from what the code expects.

To resolve this issue, let's first check the structure of the data returned by the JSON file. You can print out a portion of the addresses list to see what keys are available in each address dictionary. Here's how you can modify the script to do this:

```
python
import requests

# Load addresses
response = requests.get('https://zorang-recruitment.s3.ap-south-1.amazonaws.com/addresses.json')
addresses = response.json()

# Print the first few addresses to check their structure
for address in addresses[:5]:
    print(address)
```

Run this script and check the output. It will show you the structure of the address dictionaries. You're looking for the key that uniquely identifies each address (it might be `'id'`, `'address_id'`, or something similar). Once you identify the correct key, you can replace `'id'` in your original script with this key.

If you share the output here, I can assist further in adjusting the script accordingly.



You:

I opened the link <https://zorang-recruitment.s3.ap-south-1.amazonaws.com/addresses.json> it shows `{ "_id": 1, "latitude": 77.26398177444935, "longitude": 29.08908139212639 }, { "_id": 2, "latitude": 77.2652468523133, "longitude": 29.092540182173252 }, { "_id": 3, "latitude": 77.26404986407397, "longitude": 29.09916289150715 }, { "_id": 4, "latitude": 77.26563535630703, "longitude": 29.095355203949506 }, { "_id": 5, "latitude": 77.26583819836377, "longitude": 29.095966333768118 }, { "_id": 6, "latitude": 77.26591865459652, "longitude": 29.096030741930008 }`,



ChatGPT:

Thanks for providing the structure of the JSON data. It appears the unique identifier for each address is `_id` rather than `id`. To fix the script, you should replace `address['id']` with `address['_id']`.

Here's the revised version of the script:

```

python
import folium
import requests

# Load addresses
response = requests.get('https://zorang-recruitment.s3.ap-south-1.amazonaws.com/addresses.json')
addresses = response.json()

# Store coordinates
store_coordinates = (28.9428, 77.2276)

# Create a map centered around the store location
m = folium.Map(location=store_coordinates, zoom_start=12)

# Mark the store location
folium.Marker(store_coordinates, popup='Store', icon=folium.Icon(color='red', icon='info-sign')).add_to(m)

# Assign addresses to agents (assuming you have already done this and have a list `agent_routes`)
# Example: agent_routes = [[1, 2, 3], [4, 5, 6], ...]
num_agents = 10
agent_routes = [[] for _ in range(num_agents)]
for i, address in enumerate(addresses):
    agent_routes[i % num_agents].append(address['_id'])

# Function to find address by _id
def find_address_by_id(addr_id):
    return next(addr for addr in addresses if addr['_id'] == addr_id)

# Plotting routes for each agent
for i, route in enumerate(agent_routes):
    last_coords = store_coordinates
    for addr_id in route:
        addr = find_address_by_id(addr_id)
        addr_coords = (addr['latitude'], addr['longitude'])
        folium.Marker(addr_coords, popup=f'Address {_id}', icon=folium.Icon(color='blue')).add_to(m)
        folium.PolyLine([last_coords, addr_coords], color='blue').add_to(m)
        last_coords = addr_coords
    # Connecting back to store
    folium.PolyLine([last_coords, store_coordinates], color='blue').add_to(m)

# Save the map to an HTML file
m.save('delivery_routes_map.html')

```

This revised script should work correctly with the JSON data you have. It will create a map with markers for each delivery address and lines representing the routes for each delivery agent. Remember, this code plots straight lines between points and does not represent actual driving routes. For more accurate route plotting, you would need to integrate with a mapping service that provides routing information.



You:

File "f:\Projects\Zorang.py", line 34, in <module> folium.Marker(addr_coords, popup=f'Address {_id}', icon=folium.Icon(color='blue')).add_to(m)
NameError: name '_id' is not defined. Did you mean: 'id'?



It looks like there's a small mistake in the script. The variable used in the popup attribute of the folium.Marker should be `addr_id` instead of `_id`. Let me correct that for you:

```

python
import folium
import requests

# Load addresses
response = requests.get('https://zorang-recruitment.s3.ap-south-1.amazonaws.com/addresses.json')

```

```

addresses = response.json()

# Store coordinates
store_coordinates = (28.9428, 77.2276)

# Create a map centered around the store location
m = folium.Map(location=store_coordinates, zoom_start=12)

# Mark the store location
folium.Marker(store_coordinates, popup='Store', icon=folium.Icon(color='red', icon='info-sign')).add_to(m)

# Assign addresses to agents
num_agents = 10
agent_routes = [[] for _ in range(num_agents)]
for i, address in enumerate(addresses):
    agent_routes[i % num_agents].append(address['_id'])

# Function to find address by _id
def find_address_by_id(addr_id):
    return next(addr for addr in addresses if addr['_id'] == addr_id)

# Plotting routes for each agent
for i, route in enumerate(agent_routes):
    last_coords = store_coordinates
    for addr_id in route:
        addr = find_address_by_id(addr_id)
        addr_coords = (addr['latitude'], addr['longitude'])
        # Corrected line here
        folium.Marker(addr_coords, popup=f'Address {addr_id}', icon=folium.Icon(color='blue')).add_to(m)
        folium.PolyLine([last_coords, addr_coords], color='blue').add_to(m)
        last_coords = addr_coords
    # Connecting back to store
    folium.PolyLine([last_coords, store_coordinates], color='blue').add_to(m)

# Save the map to an HTML file
m.save('delivery_routes_map.html')

```

This corrected script should work without the `NameError`. It creates a map with the store location, the delivery addresses, and lines indicating the assigned routes for each delivery agent. Once you run this script, you can open the generated `delivery_routes_map.html` file in a web browser to view the map.



You:

I am running this code in vscode with file name Zorang.py but it successfully executes but doesn't do anything



ChatGPT:

If the script executes successfully but you're not seeing any output, it's likely because the script is designed to create an HTML file (`delivery_routes_map.html`) rather than display output directly in the console. This HTML file contains the map with your plotted routes and markers.

To view the map, you should:

1. Locate the `delivery_routes_map.html` file in your project directory (the same directory where your `Zorang.py` script is).
2. Open this HTML file with any web browser (like Chrome, Firefox, Edge, etc.).

When you open the HTML file in a browser, you should see an interactive map with the store location, delivery addresses, and the routes for each delivery agent marked.

If you can't find the `delivery_routes_map.html` file, make sure the script is running to completion without any errors. If there are any errors during execution, the file might not be created. You can add some print statements in the script to confirm that it's running each section of the code. For example, add `print("Map created")` just before the `m.save('delivery_routes_map.html')` line. This way, you'll know the script ran all the way to the end.



You:

I saw the map does it makes map using google map Or on its own ?



ChatGPT:

The map you see is generated using the folium library in Python, which is not directly based on Google Maps. Instead, folium typically uses Leaflet.js, an open-source JavaScript library for mobile-friendly interactive maps.

Here's how folium works in relation to Google Maps and other mapping services:

1. **Mapping Library:** Folium is a Python wrapper around the Leaflet.js library. It allows for the creation of interactive maps within a Python environment that can be rendered in a web browser.
2. **Base Maps:** By default, folium maps do not use Google Maps as their base layer. Instead, they typically use open-source map layers provided by services like OpenStreetMap, Mapbox, or others. However, you can customize folium to use different base map layers, including some that are visually similar to Google Maps.
3. **Data Overlay:** folium allows you to overlay your own data (like markers, lines, or polygons) on top of these base maps. This is how your delivery routes and store locations are displayed.
4. **Rendering:** The map is rendered as an HTML file which, when opened in a web browser, displays the interactive map with your data overlaid on the chosen base map.
5. **Google Maps API:** If you prefer to use Google Maps, including its routing and geocoding services, you would need to use the Google Maps API directly, which is separate from folium. Google Maps API provides more detailed and accurate map data, especially for route planning, but it requires an API key and may incur costs depending on usage.

In summary, the map you're seeing is a Leaflet.js map created and customized by folium, not a direct Google Maps interface. This approach is quite popular in data science and geographic information systems (GIS) for visualizing geospatial data due to its ease of use and flexibility.



You:

so is google map is more accurate than folium ?



ChatGPT:

Yes, Google Maps is generally more accurate and detailed than the maps typically used in folium for several reasons:

1. **Data Quality and Coverage:** Google Maps has access to extensive, high-quality data, including detailed road networks, business listings, traffic conditions, and satellite imagery. This data is constantly updated and refined, making Google Maps one of the most accurate and comprehensive mapping services available.
2. **Routing Algorithms:** For route planning, Google Maps uses sophisticated algorithms that account for current traffic conditions, road closures, and other real-time data. This makes its routing suggestions more accurate and reliable for navigation purposes compared to the basic route plotting in folium.
3. **Additional Features:** Google Maps offers a range of features like Street View, turn-by-turn navigation, location reviews, and public transit information, which are not available in the basic folium maps.

On the other hand, folium is primarily used for:

- **Data Visualization:** It excels at creating interactive maps that can visualize data. It's often used in data analysis and geographic information systems (GIS) to plot data points, draw lines, and create choropleth maps.
- **Customization and Integration:** folium allows for a high degree of customization and can be integrated directly into Python scripts and Jupyter notebooks. It's ideal for projects where you need to programmatically generate maps based on data processing or analysis.

- **Open-Source Base Maps:** folium typically uses open-source map layers like OpenStreetMap, which are free to use. While these maps are less detailed than Google Maps, they are sufficient for many applications, especially those focused on data visualization rather than navigation.

In summary, if you need highly accurate, detailed maps with the latest routing information, Google Maps is more suitable. If your focus is on data visualization and you want to integrate maps into a data analysis workflow, folium is a great tool.



You:

```
I have a code import numpy as np import requests from haversine import haversine # Function to calculate distance def calculate_distance(coord1, coord2): return haversine(coord1, coord2) # Load addresses response = requests.get('https://zorang-recruitment.s3.ap-south-1.amazonaws.com/addresses.json') addresses = response.json() # Store coordinates store_coordinates = (28.9428, 77.2276) # Calculate distances from the store to each address distances = [] for address in addresses: addr_coordinates = (address['latitude'], address['longitude']) distance = calculate_distance(store_coordinates, addr_coordinates) distances.append((address['id'], distance)) # Sort addresses by distance distances.sort(key=lambda x: x[1]) # Assign addresses to agents num_agents = 10 agent_routes = [[] for _ in range(num_agents)] for i, (addr_id, _) in enumerate(distances): agent_routes[i % num_agents].append(addr_id) # Print routes for each agent for i, route in enumerate(agent_routes): print(f"Agent {i+1}: {route}") This code distributes the addresses to each agent in a round-robin fashion. can you make this code visually easy like I can see on map what is the route of a agent name the agents like agent1 , agent2..etc till agent10. if i click on agent1 it should show the all paths assigned to agent1 only then for same for the agent2 it should show the all paths assigned to agent2 only like that for agent3 .....till agent10. you gave me this code to show address on map , may be helpful to you import folium import requests # Load addresses response = requests.get('https://zorang-recruitment.s3.ap-south-1.amazonaws.com/addresses.json') addresses = response.json() # Store coordinates store_coordinates = (28.9428, 77.2276) # Create a map centered around the store location m = folium.Map(location=store_coordinates, zoom_start=12) # Mark the store location folium.Marker(store_coordinates, popup='Store', icon=folium.Icon(color='red', icon='info-sign')).add_to(m) # Assign addresses to agents (assuming you have already done this and have a list `agent_routes`) # Example: agent_routes = [[1, 2, 3], [4, 5, 6], ...] num_agents = 10 agent_routes = [[] for _ in range(num_agents)] for i, address in enumerate(addresses): agent_routes[i % num_agents].append(address['id']) # Function to find address by _id def find_address_by_id(addr_id): return next(addr for addr in addresses if addr['id'] == addr_id) # Plotting routes for each agent for i, route in enumerate(agent_routes): last_coords = store_coordinates for addr_id in route: addr = find_address_by_id(addr_id) addr_coords = (addr['latitude'], addr['longitude']) folium.Marker(addr_coords, popup=f'Address {addr_id}', icon=folium.Icon(color='blue')).add_to(m) folium.PolyLine([last_coords, addr_coords], color='blue').add_to(m) last_coords = addr_coords # Connecting back to store folium.PolyLine([last_coords, store_coordinates], color='blue').add_to(m) # Save the map to an HTML file m.save('delivery_routes_map.html')
```



ChatGPT:

Creating a map where you can interactively see the routes for each agent is a great idea. This can be done by modifying the existing folium map code to create separate layers for each agent. Each layer will represent the route for one agent, and you'll be able to toggle these layers on and off to see individual routes.

Here's how you can modify the script to achieve this:

```
python
import folium
import requests

# Load addresses
response = requests.get('https://zorang-recruitment.s3.ap-south-1.amazonaws.com/addresses.json')
addresses = response.json()

# Store coordinates
store_coordinates = (28.9428, 77.2276)

# Create a map centered around the store location
m = folium.Map(location=store_coordinates, zoom_start=12)

# Mark the store location
folium.Marker(store_coordinates, popup='Store', icon=folium.Icon(color='red', icon='info-sign')).add_to(m)

# Function to find address by _id
def find_address_by_id(addr_id):
    return next(addr for addr in addresses if addr['id'] == addr_id)

# Distribute addresses to agents as per your existing code
num_agents = 10
agent_routes = [[] for _ in range(num_agents)]
for i, address in enumerate(addresses):
```

```

agent_routes[i % num_agents].append(address['_id'])

# Creating a feature group for each agent
for i, route in enumerate(agent_routes):
    route_group = folium.FeatureGroup(name=f'Agent {i+1}')
    last_coords = store_coordinates
    for addr_id in route:
        addr = find_address_by_id(addr_id)
        addr_coords = (addr['latitude'], addr['longitude'])
        folium.Marker(addr_coords, popup=f'Address {addr_id}', icon=folium.Icon(color='blue')).add_to(route_group)
        folium.PolyLine([last_coords, addr_coords], color='blue').add_to(route_group)
        last_coords = addr_coords
    # Connecting back to store
    folium.PolyLine([last_coords, store_coordinates], color='blue').add_to(route_group)
    route_group.add_to(m)

# Add LayerControl to toggle each agent's route
folium.LayerControl().add_to(m)

# Save the map to an HTML file
m.save('delivery_routes_map.html')

```

In this script, each agent's route is added to a separate FeatureGroup. The LayerControl is then added to the map, allowing you to toggle each agent's route on and off. When you open the resulting `delivery_routes_map.html` file in a web browser, you'll see checkboxes for each agent. Clicking on these checkboxes will show or hide the respective agent's route.

This approach makes it easy to visualize and compare the routes assigned to each agent.



You:

when I click on i button it shows me address 91 , 41, 31 like that what it is ?



ChatGPT:

When you click on the "i" (information) button on the markers in the `folium` map, it displays a popup with the information provided in the `popup` argument of the `folium.Marker` function. In the script I provided, this popup is set to show the address ID (such as 'Address 91', 'Address 41', etc.), which corresponds to the unique identifier of each delivery address from the JSON data.

Here's the relevant line in the script:

```

python
folium.Marker(addr_coords, popup=f'Address {addr_id}', icon=folium.Icon(color='blue')).add_to(route_group)

```

In this line, `addr_id` represents the unique identifier of an address (`_id` in your JSON data). So, when you click on a marker, it shows you the ID of that particular address. This ID is used to distinguish each address and is useful for tracking which addresses are assigned to which delivery agent.

If you want to display more detailed information in the popup, you would need to modify the `popup` argument to include the details you want to show. For instance, if you want to display the latitude and longitude, or any other information available in your addresses data, you can format the `popup` string accordingly.

For example, if you want to show latitude and longitude as well, you could modify the line like this:

```

python
popup_text = f'Address {addr_id}\nLatitude: {addr["latitude"]}\nLongitude: {addr["longitude"]}'
folium.Marker(addr_coords, popup=popup_text, icon=folium.Icon(color='blue')).add_to(route_group)

```

This will display the address ID along with its latitude and longitude in the popup when you click on a marker.



You:

can you modify the code and show me full code



ChatGPT:

Certainly! I'll modify the script to include more detailed information in the popups, such as the address ID, latitude, and longitude. Here's the full script with these changes:

```

python
import folium
import requests

# Load addresses
response = requests.get('https://zorang-recruitment.s3.ap-south-1.amazonaws.com/addresses.json')
addresses = response.json()

# Store coordinates
store_coordinates = (28.9428, 77.2276)

# Create a map centered around the store location
m = folium.Map(location=store_coordinates, zoom_start=12)

# Mark the store location
folium.Marker(store_coordinates, popup='Store', icon=folium.Icon(color='red', icon='info-sign')).add_to(m)

# Function to find address by _id
def find_address_by_id(addr_id):
    return next(addr for addr in addresses if addr['_id'] == addr_id)

# Distribute addresses to agents as per your existing code
num_agents = 10
agent_routes = [[] for _ in range(num_agents)]
for i, address in enumerate(addresses):
    agent_routes[i % num_agents].append(address['_id'])

# Creating a feature group for each agent
for i, route in enumerate(agent_routes):
    route_group = folium.FeatureGroup(name=f'Agent {i+1}')
    last_coords = store_coordinates
    for addr_id in route:
        addr = find_address_by_id(addr_id)
        addr_coords = (addr['latitude'], addr['longitude'])
        popup_text = f'Address ID: {addr_id}\nLatitude: {addr_coords[0]}\nLongitude: {addr_coords[1]}'
        folium.Marker(addr_coords, popup=popup_text, icon=folium.Icon(color='blue')).add_to(route_group)
        folium.PolyLine([last_coords, addr_coords], color='blue').add_to(route_group)
        last_coords = addr_coords
    # Connecting back to store
    folium.PolyLine([last_coords, store_coordinates], color='blue').add_to(route_group)
    route_group.add_to(m)

# Add LayerControl to toggle each agent's route
folium.LayerControl().add_to(m)

# Save the map to an HTML file
m.save('delivery_routes_map.html')

```

This script now includes additional information in the popups for each address marker. When you click on a marker on the map, it will show the address ID, latitude, and longitude. This makes it easier to identify and differentiate each address point. After running this script, you can open the `delivery_routes_map.html` file in a web browser to view the updated interactive map.



You:
can we use google map instead of openstreet map



Using Google Maps instead of OpenStreetMap in folium is a bit tricky because folium primarily integrates with Leaflet.js and uses OpenStreetMap as its default tile source. However, you can add Google Maps tiles to a folium map using custom tile layers. This approach requires using the Google Maps API, and you should be aware of Google's API usage policies and potential costs.

Here's how you can modify the script to use Google Maps tiles:

First, you will need a Google Maps API key. You can get this from the Google Cloud Console. Ensure that the Maps JavaScript API is enabled for your project.

Once you have the API key, you can use it to add Google Maps as a tile layer in your folium map:

```
python
import folium
import requests

# Your Google Maps API key
google_api_key = 'YOUR_GOOGLE_MAPS_API_KEY'

# Function to create a Google Maps tile layer URL
def google_maps_tile_layer(api_key, map_type='roadmap', language='en'):
    return f'https://mt1.google.com/vt/lyrs={map_type}&x={x}&y={y}&z={z}&hl={language}&key={api_key}'

# Load addresses
response = requests.get('https://zorang-recruitment.s3.ap-south-1.amazonaws.com/addresses.json')
addresses = response.json()

# Store coordinates
store_coordinates = (28.9428, 77.2276)

# Create a map centered around the store location
# Set tiles to None initially
m = folium.Map(location=store_coordinates, zoom_start=12, tiles=None)

# Add Google Maps tile layer
google_maps_tiles = google_maps_tile_layer(google_api_key)
folium.TileLayer(tiles=google_maps_tiles, attr='Google', name='Google Maps').add_to(m)

# Rest of your code for markers and routes...

# Remember to add LayerControl
folium.LayerControl().add_to(m)

# Save the map to an HTML file
m.save('delivery_routes_map_google.html')
```

In this script, the `google_maps_tile_layer` function generates a URL for the Google Maps tiles. You need to replace '`YOUR_GOOGLE_MAPS_API_KEY`' with your actual API key. The `map_type` parameter can be changed to other Google Maps types like 'satellite', 'terrain', etc., if desired.

Please note:

- This approach might not fully comply with Google's terms of service for the Maps API, especially if the map is used in a commercial product or website.
- Ensure that you are aware of any costs associated with using the Google Maps API, as high usage can incur charges.



can you write full code please



To use Google Maps tiles in a folium map, you'll need to integrate a custom tile layer using your Google Maps API key. Below is the full code, including this integration. Please remember to replace 'YOUR_GOOGLE_MAPS_API_KEY' with your actual Google Maps API key.

```

python
import folium
import requests

# Your Google Maps API key
google_api_key = 'YOUR_GOOGLE_MAPS_API_KEY'

# Function to find address by _id
def find_address_by_id(addr_id, addresses):
    return next(addr for addr in addresses if addr['_id'] == addr_id)

# Load addresses
response = requests.get('https://zorang-recruitment.s3.ap-south-1.amazonaws.com/addresses.json')
addresses = response.json()

# Store coordinates
store_coordinates = (28.9428, 77.2276)

# Create a map centered around the store location
# Initialize with no tiles (tiles=None)
m = folium.Map(location=store_coordinates, zoom_start=12, tiles=None)

# Add Google Maps tile layer
google_maps_tile = f'https://mt1.google.com/vt/lyrs=m&x={x}&y={y}&z={z}&hl=en&key={google_api_key}'
folium.TileLayer(tiles=google_maps_tile, attr='Google', name='Google Street Map', overlay=False).add_to(m)

# Distribute addresses to agents as per your existing code
num_agents = 10
agent_routes = [[] for _ in range(num_agents)]
for i, address in enumerate(addresses):
    agent_routes[i % num_agents].append(address['_id'])

# Creating a feature group for each agent
for i, route in enumerate(agent_routes):
    route_group = folium.FeatureGroup(name=f'Agent {i+1}')
    last_coords = store_coordinates
    for addr_id in route:
        addr = find_address_by_id(addr_id, addresses)
        addr_coords = (addr['latitude'], addr['longitude'])
        popup_text = f'Address ID: {addr_id}\nLatitude: {addr_coords[0]}\nLongitude: {addr_coords[1]}'
        folium.Marker(addr_coords, popup=popup_text, icon=folium.Icon(color='blue')).add_to(route_group)
        folium.PolyLine([last_coords, addr_coords], color='blue').add_to(route_group)
        last_coords = addr_coords
    # Connecting back to store
    folium.PolyLine([last_coords, store_coordinates], color='blue').add_to(route_group)
    route_group.add_to(m)

# Add LayerControl to toggle each agent's route
folium.LayerControl().add_to(m)

# Save the map to an HTML file
m.save('delivery_routes_map_google.html')

```

This script creates a folium map using Google Maps as the base layer. Each delivery agent's route is added as a separate layer that can be toggled on and off.

Please note:

- Replace 'YOUR_GOOGLE_MAPS_API_KEY' with your actual Google Maps API key.
- The use of Google Maps tiles in this manner should comply with Google's terms of service, particularly if used for commercial purposes.

- The Google Maps API may incur costs depending on your usage level.

After running this script, the output will be saved in a file named `delivery_routes_map_google.html`, which you can open in a web browser to view the map.



You:

File "f:\Projects\Zorang.py", line 117, in <module> google_maps_tile = f'https://mt1.google.com/vt/lyrs=m&x={x}&y={y}&z={z}&hl=en&key={google_api_key}' NameError: name 'x' is not defined



ChatGPT:

Apologies for the confusion. The error occurs because the x, y, and z variables in the tile URL are placeholders meant to be replaced by folium internally when requesting tiles. They should not be included directly in the string formatting. Here's the corrected way to set up the Google Maps tile layer in folium:

```
python
import folium
import requests

# Your Google Maps API key
google_api_key = 'YOUR_GOOGLE_MAPS_API_KEY'

# Function to find address by _id
def find_address_by_id(addr_id, addresses):
    return next(addr for addr in addresses if addr['_id'] == addr_id)

# Load addresses
response = requests.get('https://zorang-recruitment.s3.ap-south-1.amazonaws.com/addresses.json')
addresses = response.json()

# Store coordinates
store_coordinates = (28.9428, 77.2276)

# Create a map centered around the store location
# Initialize with no tiles (tiles=None)
m = folium.Map(location=store_coordinates, zoom_start=12, tiles=None)

# Add Google Maps tile layer
google_maps_tile = f'https://mt1.google.com/vt/lyrs=m&x={{x}}&y={{y}}&z={{z}}&hl=en&key={google_api_key}'
folium.TileLayer(tiles=google_maps_tile, attr='Google', name='Google Street Map', overlay=False).add_to(m)

# Distribute addresses to agents as per your existing code
num_agents = 10
agent_routes = [[] for _ in range(num_agents)]
for i, address in enumerate(addresses):
    agent_routes[i % num_agents].append(address['_id'])

# Creating a feature group for each agent
for i, route in enumerate(agent_routes):
    route_group = folium.FeatureGroup(name=f'Agent {i+1}')
    last_coords = store_coordinates
    for addr_id in route:
        addr = find_address_by_id(addr_id, addresses)
        addr_coords = (addr['latitude'], addr['longitude'])
        popup_text = f'Address ID: {addr_id}\nLatitude: {addr_coords[0]}\nLongitude: {addr_coords[1]}'
        folium.Marker(addr_coords, popup=popup_text, icon=folium.Icon(color='blue')).add_to(route_group)
        folium.PolyLine([last_coords, addr_coords], color='blue').add_to(route_group)
        last_coords = addr_coords
    # Connecting back to store
    folium.PolyLine([last_coords, store_coordinates], color='blue').add_to(route_group)
    route_group.add_to(m)

# Add LayerControl to toggle each agent's route
```

```
folium.LayerControl().add_to(m)

# Save the map to an HTML file
m.save('delivery_routes_map_google.html')
```

This code should correctly integrate Google Maps tiles into your folium map. The {x}, {y}, and {z} placeholders are left as is, allowing folium to replace them with the appropriate values when loading the tiles. Remember to replace 'YOUR_GOOGLE_MAPS_API_KEY' with your actual Google Maps API key.



You:

m = folium.Map(location=store_coordinates, zoom_start=12, tiles=None) what does it mean by title none ?



ChatGPT:

In the folium.Map function call, the tiles=None argument specifies that the map should be initialized without any default tileset.

In folium, the tiles parameter is used to determine the style of the map's base layer, which is essentially the background over which other elements (like markers or lines) are placed. By default, folium uses OpenStreetMap's tiles. However, you can specify other tile sets, like Stamen Terrain, Stamen Toner, Mapbox Bright, and others.

Setting tiles=None is useful when you want to use a completely custom tileset, such as Google Maps tiles in your case. This means folium won't automatically load its default OpenStreetMap tiles. Instead, you manually add your desired tile layer (Google Maps in this instance) to the map after its initialization.

This approach provides more control over the map's appearance and functionality, allowing the integration of various map services that folium does not natively support, like Google Maps.



You:

```
# Distribute addresses to agents as per your existing code num_agents = 10 agent_routes = [] for _ in range(num_agents)] for i, address in enumerate(addresses): agent_routes[i % num_agents].append(address['_id']) So here , by which algorithm 100 address is distributed into 10 agents
```



ChatGPT:

The code snippet you've provided distributes the addresses to the 10 agents using a straightforward round-robin algorithm. Here's a breakdown of how it works:

1. **Number of Agents:** You have num_agents set to 10. This means there are 10 delivery agents available.
2. **Agent Routes Initialization:** agent_routes is initialized as a list of 10 empty lists, each representing the route (or set of addresses) for one agent.
3. **Round-Robin Distribution:** The for loop iterates over each address in the addresses list. For each address, it appends the address's ID (_id) to one of the agent's routes. The operator $i \% num_agents$ is used to determine which agent's route to append the address to.
 - o The % is the modulo operator, which gives the remainder of the division i / num_agents .
 - o As i increases from 0 to 99 (for 100 addresses), $i \% num_agents$ cycles through values from 0 to 9, then repeats.
4. **Result:** Each iteration of the loop assigns an address to the next agent in the sequence. Once the last agent (agent 10 in this case) receives an address, the next address goes to the first agent again, and the cycle repeats. This ensures that the addresses are distributed as evenly as possible among the agents.

Example:

- Address 1 goes to Agent 1
- Address 2 goes to Agent 2
- ...

- Address 10 goes to Agent 10
- Address 11 goes back to Agent 1
- and so on...

This algorithm is simple and effective for evenly distributing tasks (in this case, addresses) among a fixed number of agents (delivery personnel). It does not, however, take into account other factors like the actual distances or optimal routing paths, which would be considered in more complex algorithms used for route optimization in delivery services.
