Universidade Federal de Roraima

Inteligência Artificial

APLICAÇÃO DO ALGORITMO A* PARA ENCONTRAR O MENOR CAMINHO DE UM PONTO A ATÉ O PONTO B

Nome: Venícius Jacob Pereira de Oliveira

N° de matrícula: 2020014633

1. Introdução

Este relatório descreve o desenvolvimento de um sistema de roteamento de mapas para cidades utilizando as bibliotecas OSMnx (OpenStreetMap Networkx), NetworkX, Google Maps e Matplotlib. O objetivo do sistema é calcular e visualizar o caminho mais curto entre dois pontos em uma cidade específica, utilizando dados de redes de ruas.

2. Tecnologias Utilizadas

- OSMnx: Uma biblioteca Python para download, modelagem, análise e visualização de redes viárias de OpenStreetMap.
- Google Maps API: Utilizada para geocodificação de endereços e para visualização dos mapas, converte endereços em coordenadas de latitude/longitude e vice-versa.
- NetworkX: Uma biblioteca Python para criação, manipulação e estudo de estruturas complexas de redes.
- Matplotlib: Uma biblioteca de plotagem em Python para criar visualizações estáticas, interativas e animadas em Python.

3. Funcionalidades do Sistema

- Geocodificação de Endereços: Converte endereços em coordenadas de latitude e longitude utilizando a API Geocoding.
- Construção do Grafo de Rede de Ruas: Utiliza OSMnx para construir um grafo representando a rede viária da cidade especificada.
- Encontrar Nós Mais Próximos: Utiliza OSMnx para encontrar os nós mais próximos às coordenadas dos endereços de origem e destino.
- Calcular Caminho Mais Curto: Utiliza o algoritmo A* implementado no NetworkX para calcular o caminho mais curto entre os nós de origem e destino no grafo de rede viária.
- Visualização do Mapa e do Caminho Mais Curto: Utiliza Matplotlib e OSMnx para visualizar o mapa da cidade com o caminho mais curto destacado.

4. Procedimento de Uso

- O usuário fornece os endereços de origem e destino desejados de forma completa, incluindo bairro e CEP;
- O sistema geocodifica os endereços para obter as coordenadas de latitude e longitude;
- O sistema constrói o grafo de rede de ruas da cidade;
- O sistema encontra os nós mais próximos às coordenadas dos endereços de origem e destino;
- O sistema calcula o caminho mais curto entre os nós de origem e destino;
- O sistema visualiza o mapa da cidade com o caminho mais curto destacado.

5. Implementação das funções do código

5.1. Inicialização do cliente para a API Geocoding do Google Maps

Esta função inicializa um cliente para a API Geocoding do Google Maps. Para usar esta função, substitua 'API_KEY' por uma chave válida da API. Essa chave é necessária para autenticar a aplicação e permitir o acesso à API Geocoding. Retorna um cliente inicializado para acessar a API Geocoding armazenada na variável “gmaps”.

```
def inicializar_cliente_google_maps():
    gmaps = googlemaps.Client(key='API_KEY',)
    return gmaps
```

5.2. Conversão do endereço em coordenadas

Esta função recebe dois parâmetros: “gmaps” (cliente do Google Maps previamente inicializado) e um endereço (uma string representando o endereço a ser convertido para coordenadas).

Usando o cliente “gmaps”, a função realiza uma consulta de geocodificação para o endereço fornecido, armazenando o resultado na variável “geocode_result”.

As coordenadas geográficas (latitude e longitude) do endereço são então extraídas de “geocode_result” e retornadas como um dicionário armazenado na variável “coordenadas”.

```
def converter_endereco(gmaps, endereco):
    geocode_result = gmaps.geocode(endereco)
    coordenadas = geocode_result[0]['geometry']['location']
    return coordenadas
```

5.3. Construção do grafo das ruas para a cidade fornecida

Esta função recebe um parâmetro “cidade” (String que representa a cidade para a qual se deseja construir um grafo de ruas).

Utilizando a biblioteca OSMnx, a função “ox.grafo_from_place” cria um grafo de ruas para a cidade especificada, considerando apenas as vias adequadas para veículos motorizados, especificado pelo parâmetro “drive”, tal parâmetro também pode ser “walk”, “bike”, “all”. O grafo de ruas é então retornado pela função e armazenado na variável “G”.

```
def construir_grafo_de_ruas(cidade):
    G = ox.grafo_from_place(cidade, network_type='drive')
    return G
```

5.4. Encontrar nó mais próximo ao ponto especificado

Esta função recebe três parâmetros: “grafo” (o grafo de ruas), “latitude” e “longitude". Utilizando a função “nearest_nodes()” da biblioteca OSMnx, esta função encontra o nó mais próximo no grafo de ruas das coordenadas geográficas específicas. O nó mais próximo é encontrado e armazenado na variável “no_mais_proximo”, então é retornado pela função.

```
def encontrar_nos_mais_proximos(grafo, latitude, longitude):
    no_mais_proximo = ox.distance.nearest_nodes(grafo, longitude, latitude)
    return no_mais_proximo
```

5.5. Encontrar o menor caminho da origem para o destino

Esta função, recebe três parâmetros: “grafo” (o grafo de ruas), “origem” (o nó de origem) e “destino” (o nó de destino).

Utilizando o algoritmo A* implementado na biblioteca NetworkX, esta função encontra o caminho mais curto no grafo de ruas, considerando o comprimento das arestas como pesos. O caminho mais curto encontrado é então armazenado na variável “caminho_mais_curto” e é retornada pela função.

```
def encontrar_caminho_mais_curto(grafo, origem, destino):
    caminho_mais_curto = nx.astar_path(grafo, source=origem, target=destino, weight='length')
    return caminho_mais_curto
```

5.6. Plotagem do grafo

Esta função, recebe cinco parâmetros: “G” (o grafo de ruas), “rota” (a rota a ser plotada), essa rota é uma lista de nós, que será usada para destacar o melhor caminho, “coord_origem” (as coordenadas da origem) e “coord_destino” (as coordenadas do destino).

Utilizando a função “plot_grafo_route()” da biblioteca OSMnx, esta função plota o grafo de ruas com a rota especificada, destacando-a em azul. Além disso, essa função usa a função “plt.plot()” da biblioteca Matplotlib para adicionar pontos de origem e destino ao gráfico, a função “plt.plot()” desenha os pontos de origem (ponto A) e destino (ponto B) no gráfico, representados por marcadores vermelho e amarelo, respectivamente. O gráfico resultante é então exibido.

```
def plotar_grafo_com_rota(G, rota, coord_origem, coord_destino):
    fig, ax = ox.plot_graph_route(G, rota, route_color='b', show=False, close=False)
    plt.plot(coord_origem['lng'], coord_origem['lat'], 'ro')
    plt.plot(coord_destino['lng'], coord_destino['lat'], 'yo')
    plt.show()
```

6. Resultados

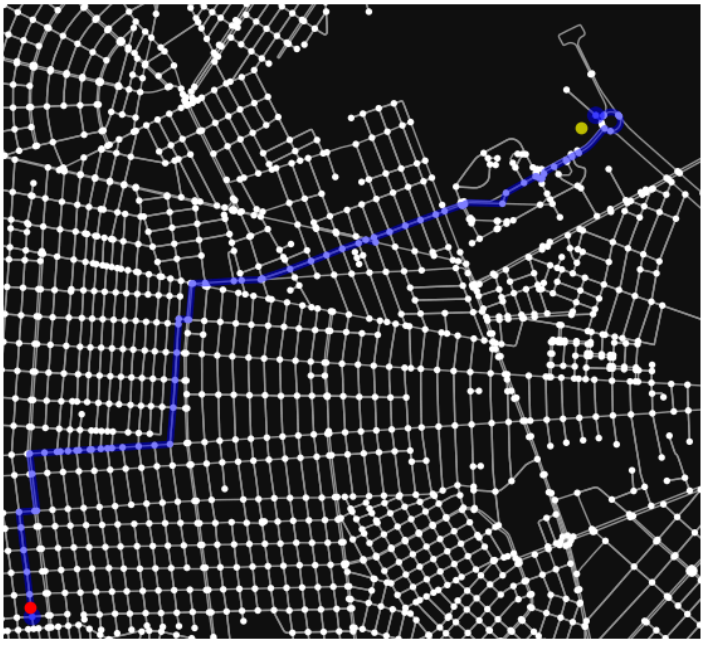
Aqui estão os resultados da execução do sistema para os endereços de origem e destino. A cidade usada para o teste é a cidade de Boa Vista, Roraima. Os endereços de origem são aleatórios, o endereço de destino é a localização do bloco V da Universidade Federal de Roraima.

6.1 Output teste 1

Endereço de origem: R. Francisco Custodio de Andrade, 127
Endereço de destino: Av. Cap. Ene Garcês, 2413 - Bloco V - Aeroporto
Coordenada Origem: {'lat': 2.8097039, 'lng': -60.7215292}
Coordenada Destino: {'lat': 2.8358106, 'lng': -60.6915276}
Nó mais próximo da origem: 1712747490

Nó mais próximo do destino: 1706580384

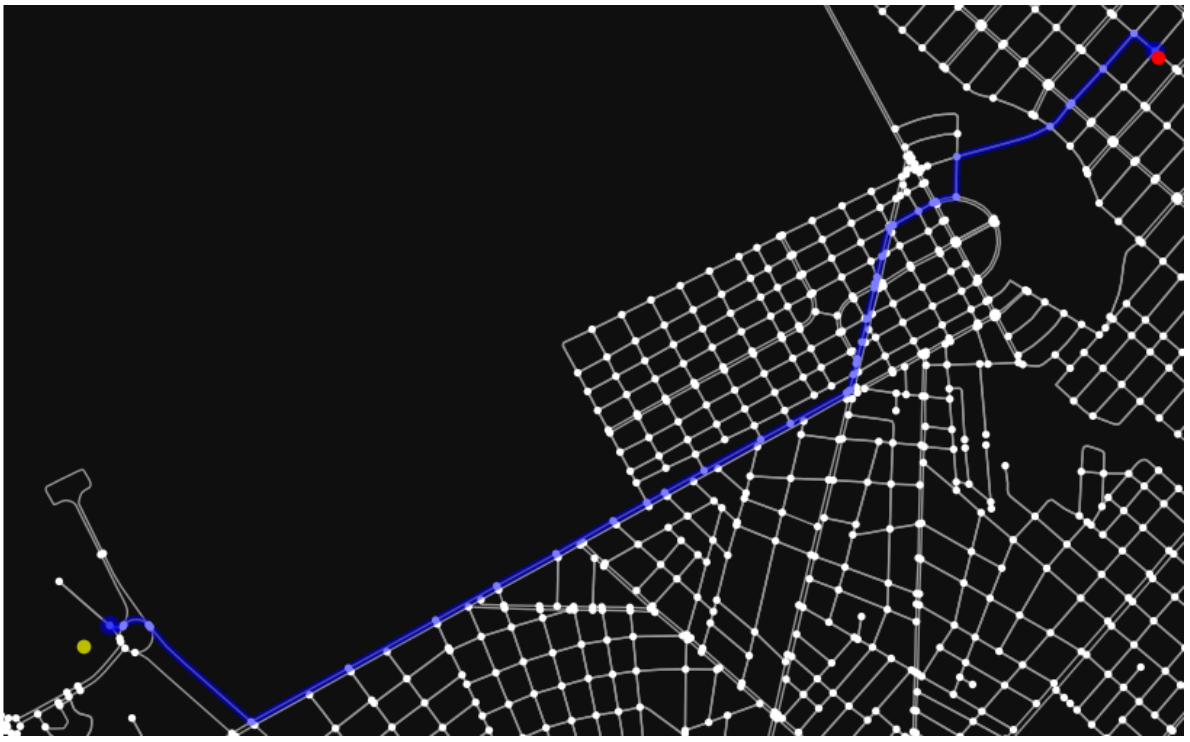
Caminho mais curto encontrado: [1712747490, 1713304353, 3734466531, 3734466528, 1713304290, 1713304272, 1700532505, 1700532489, 1700532508, 1686127937, 1712774139, 1719383860, 1712774219, 1719383808, 1712774222, 1719383880, 1719383785, 1712774199, 1719383850, 1712774142, 1719383872, 1712774168, 1712774141, 1712774229, 1712774169, 1719383874, 1719383868, 1719383864, 1719383835, 1712668321, 1712668297, 1712668273, 1426836649, 5109792580, 1687960928, 1700532485, 1687960933, 1687960917, 1687960963, 1687960962, 1687960916, 1687960929, 1687960981, 1687953330, 1687960967, 318385426, 1685344778, 1685344826, 1685344708, 1685344825, 1685344706, 1684682950, 1683637130, 1683637099, 1683637113, 1700892768, 1700892760, 1683637067, 1683637149, 1683637092, 1683636985, 1683637090, 1683637084, 1683637105, 2318492319, 2318492331, 1683637147, 1067748823, 1706580378, 1067748533, 1067749043, 6952403776, 1706580384]



Output do grafo - teste 1

6.2 Output teste 2

Endereço de origem: R. Deusdete Coelho, 309 - Paraviana, 69307-273
Endereço de destino: Av. Cap. Ene Garcês, 2413 - Bloco V - Aeroporto
Coordenada Origem: {'lat': 2.8543569, 'lng': -60.657672}
Coordenada Destino: {'lat': 2.8358106, 'lng': -60.6915276}
Nó mais próximo da origem: 1710134584
Nó mais próximo do destino: 1706580384
Caminho mais curto encontrado: [1710134584, 1710572129, 1710134573, 1710115588, 1710115602, 1710072284, 1709979655, 3720761334, 1709276183, 1709979460, 1709979587, 1709276377, 1709979479, 1067748751, 1709979418, 1709276380, 1709276263, 2171726921, 1709276201, 1709276176, 1709276274, 1709276435, 1673621519, 1673621518, 1709276276, 1673621506, 1709276228, 6929265739, 4816528961, 6929265740, 1709206494, 1707699316, 3727779431, 3727779429, 3589994404, 1706580378, 1067748533, 1067749043, 6952403776, 1706580384]



Output do grafo - teste 2

5. Código Completo

5.1. Main

```python
from functions import *

#Endereços de origem e destino, priorizar endereços completos com CEP
endereco_A = 'R. Deusdete Coelho, 309 - Paraviana, 69307-273'
endereco_B = 'Av. Cap. Ene Garcês, 2413 - Bloco V - Aeroporto'
cidade = "Boa vista"

print(f"Endereço de origem: {endereco_A}")
print(f"Endereço de destino: {endereco_B}")

# Inicialização do cliente da API do Google Maps
gmaps = inicializar_cliente_google_maps()

# Geocodificação dos endereços para obter as coordenadas
coord_A = converter_endereco(gmaps, endereco_A)
coord_B = converter_endereco(gmaps, endereco_B)

#Checar coordenas dos endereços(Descomentar)
print(f"Coordenada Origem: {coord_A}")
print(f"Coordenada Destino: {coord_B}")

# Construção do Grafo de Rede de Ruas da Cidade
G = construir_grafo_de_ruas(cidade)

#encontra o nó mais próximo da origem e o nó mais próximodo destino
origem = encontrar_nos_mais_proximos(G, coord_A['lat'], coord_A['lng'])
destino = encontrar_nos_mais_proximos(G, coord_B['lat'], coord_B['lng'])

#Checar nó mais proximo da origem e do nó de destino(descomentar)
print("Nó mais próximo da origem:", origem)
print("Nó mais próximo do destino:", destino)

#checar todos nós no grafo
#print("Nós no grafo:", G.nodes())

# Calculando o caminho mais curto usando o algoritmo A*
caminho_mais_curto = encontrar_caminho_mais_curto(G, origem, destino)

#Checar nós do caminho mais curto encontado(descomentar)
print("Caminho mais curto encontrado:", caminho_mais_curto)

# Plotar o Grafo com o menor caminho destacado
plotar_grafo_com_rota(G, caminho_mais_curto, coord_A, coord_B)
```

5.1.2. Funções

```python
import googlemaps
import osmnx as ox
import networkx as nx
import matplotlib.pyplot as plt

#incialização da API
def inicializar_cliente_google_maps():
    gmaps = googlemaps.Client(key='AIzaSyDzqv4Djq5ytLc0AF69LDGUyEr__UK71Os',)
    return gmaps

# essa função recebe um endereço e um cliente do Google Maps,
# realiza a geocodificação desse endereço usando o cliente fornecido e retorna
# as coordenadas geográficas correspondentes.
def converter_endereco(gmaps, endereco):
    geocode_result = gmaps.geocode(endereco)
    coordenadas = geocode_result[0]['geometry']['location']
    return coordenadas

#esta função constrói um grafo de ruas para a cidade fornecida, utilizando a biblioteca OSMnx.
def construir_grafo_de_ruas(cidade):
    G = ox.graph_from_place(cidade, network_type='drive')
    return G

#essa função encontra o nós mais próximo das coordenadas informadas
def encontrar_nos_mais_proximos(grafo, latitude, longitude):
    no_mais_proximo = ox.distance.nearest_nodes(grafo, longitude, latitude)
    return no_mais_proximo

#esta função encontra e retorna o caminho mais curto entre
#os nós de origem e destino no grafo de ruas, utilizando o algoritmo A*.
def encontrar_caminho_mais_curto(grafo, origem, destino):
    caminho_mais_curto = nx.astar_path(grafo, source=origem, target=destino, weight='length')
    return caminho_mais_curto

def plotar_grafo_com_rota(G, rota, coord_origem, coord_destino):
    fig, ax = ox.plot_graph_route(G, rota, route_color='b', show=False, close=False)

    # Desenhar os pontos de origem e destino
    plt.plot(coord_origem['lng'], coord_origem['lat'], 'ro')  # ponto A (origem) em vermelho
    plt.plot(coord_destino['lng'], coord_destino['lat'], 'yo')  # ponto B (destino) em amarelo

    plt.show()
```

6. Conclusão

O desenvolvimento deste sistema de roteamento de mapas utilizando OSMnx, Google Maps e NetworkX proporciona uma ferramenta útil e eficaz para calcular e visualizar rotas de viagem em cidades. A integração dessas tecnologias permite uma análise detalhada da rede viária urbana, facilitando o planejamento de deslocamentos pelo menor caminho, não garantindo que seja o mais rápido.

7. Referências

Open Source Transportation Analysis Package. OSMnx User Reference — OSMnx documentation. Disponível em: https://osmnx.readthedocs.io/en/stable/user-reference.html. Acesso em: 03/04/2024.

NetworkX Developers. Tutorial — NetworkX documentation. Disponível em: https://networkx.org/documentation/stable/tutorial. Acesso em: 03/04/2024.

Google Developers. Documentação de Geocodificação — Google Maps. Disponível em: https://developers.google.com/maps/documentation/geocoding. Acesso em: 03/04/2024.

Matplotlib. Documentação Matplotlib. Disponível em: https://matplotlib.org/stable/index. Acesso em: 03/04/2024.