گزارش پروژه اول قسمت اول

پروژه یافتن بهترین پرواز

درس: مبانی و کاربردهای هوش مصنوعی

استاد راهنما: دکتر حسین کارشناس نجف آبادی

اعضای گروه:

علی‌اکبر احراری- ۴۰۰۳۶۱۳۰۰۱

مهرآذین مزروق- ۴۰۰۳۶۱۳۰۵۵

پاییز ۱۴۰۲

فهرست

گزارش کار الگوریتم ۳

توابع مشترک ۳

create_graph ۳

generated_cost ۴

desired_result_string ۵

main function ۶

الگوریتم Dijkstra ۸

dijkstra_algorithm ۸

الگوریتم A* ۹

Calculate_distance ۹

a_star_heuristic ۱۰

a_star_algorithm ۱۱

نمونه‌ای از خروجی ۱۲

A* ۱۲

Dijkstra ۱۳

کتابخانه‌های استفاده شده ۱۴

منابع ۱۵

گزارش کار الگوریتم

توابع مشترک

create_graph

```python
def create_graph():
    for i in range(6836):
        cost = generated_cost(df.iloc[i])
        G.add_edge(df.iloc[i, 1], df.iloc[i, 2], weight=cost,
                   Distance=df.iloc[i, 13], FlyTime=df.iloc[i, 14], Price=df.iloc[i, 15], Airline=df.iloc[i, 0])
```

در این الگوریتم، تمامی بلیت‌ها به عنوان یال گراف، به گراف اضافه می‌شوند.

generated_cost

```python
def generated_cost(param):
    fly_time = param['FlyTime']
    distance = param['Distance']
    price = param['Price']

    w1 = 100  # Weight for time
    w2 = 3  # Weight for distance
    w3 = 20  # Weight for price

    cost = w1 * fly_time + w2 * distance + w3 * price
    return cost
```

این تابع، هزینه هر بلیت را محاسبه می‌کند. هزینه هر بلیت، تابعی خطی از قیمت بلیت، فاصله مبدا و مقصد و زمان پرواز می‌باشد

به علت اینکه FlyTime به‌طور میانگین، عددی تک رقمی است و اهمیت کمتری نسبت به دو پارامتر دیگر دارد، ضریب ۱۰۰ را دریافت می‌کند.

Distance به‌طور میانگین، سه‌رقمی است واولویت اول را در مسیریابی دارد؛ درنتیجه ضریب ۳ را دریافت می‌کند.

Price به‌طور میانگین، ۲ رقمی‌است و اولویت دوم را دارد؛ بنابراین ضریب 20 را دریافت می‌کند.

desired_result_string

```python
def desired_result_string(path):
    flight_number = 1
    total_time = 0
    total_price = 0
    total_distance = 0
    result_string = ""  # Initialize an empty string to store the output
    for u, v in zip(path, path[1:]):
        edge_data = G[u][v]
        distance = round(edge_data['Distance'])
        price = round(edge_data['Price'])
        fly_time = round(edge_data['FlyTime'])
        total_time += fly_time
        total_price += price
        total_distance += distance
        if path is not None:
            result_string += f'''
Flight #{flight_number} ({edge_data['Airline']}):
From: {u}
To: {v}
Duration: {distance}km
Time: {fly_time}h
Price: {price}$
----------------------------'''
            flight_number += 1
        else:
            result_string += "No path found."
    result_string += f'''
Total Price: {total_price}$
Total Duration: {total_distance} km
Total Time: {total_time}h
                '''
    return result_string  # Return the result string
```

در این تابع، رشته‌ای که در نهایت در فایل‌ها ذخیره می‌شود، طبق صورت سوال، تولید می‌شود.

main function

```python
filename = 'Flight_Data.csv'
df = pd.read_csv(filename)
G = nx.DiGraph()
create_graph()
print("Enter The Source Airport And The Destination Airport")
user_input = input()
source_airport, destination_airport = user_input.split(" - ")
end_line = "\n.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-."

file = open('[2]-UIAI4021-PR1-Q1([A STAR]).txt', 'w')
start_time = time.time()
a_star_path = a_star_algorithm(source_airport, destination_airport)
end_time = time.time()
minute, second = divmod(end_time - start_time, 60)
a_star_time = f'{round(minute)}m{round(second)}s'
a_star_beginner = "A* Algorithm\nExecution Time: "
line = a_star_beginner + str(a_star_time) + end_line
file.write(line)
file.write(desired_result_string(a_star_path))
file.close()
file = open('[2]-UIAI4021-PR1-Q1([DIJKSTRA]).txt', 'w', encoding='utf-8')
start_time = time.time()
dijkstra_path = dijkstra_algorithm(source_airport, destination_airport)
end_time = time.time()
minute, second = divmod(end_time - start_time, 60)
dijkstra_time = f'{round(minute)}m{round(second)}s'
dijkstra_beginner = "Dijkstra Algorithm\nExecution Time: "
line = dijkstra_beginner + str(dijkstra_time) + end_line
file.write(line)
file.write(desired_result_string(dijkstra_path))
file.close()

print("Files Generated")
```

در این بخش، ابتدا گراف ساخته می‌شود، سپس از کاربر، فرودگاه مبدا و مقصد طبق صورت سوال گرفته می‌شود.

فایل مربوط به الگوریتم *A ساخته و خود الگوریتم اجرا می‌شود. نتیجه‌ی اجرای الگوریتم در فایل ذخیره می‌شود و سپس دقیقا همین روند برای الگوریتم Dijkstra اجرا می‌شود.

پس از پایان اجرای هر دو الگوریتم و ساخت هر دو فایل output، برنامه پیامی مبنی بر پایان ساخت فایل نشان می‌دهد.

الگوریتم Dijkstra

dijkstra_algorithm

```python
def dijkstra_algorithm(source, target):
    shortest_paths = {source: (None, 0)}
    queue = [(0, source)]
    while queue:
        (dist, current) = heapq.heappop(queue)
        for neighbor, data in G[current].items():
            old_cost = shortest_paths.get(neighbor, (None, float('inf')))[1]
            new_cost = dist + data['weight']
            if new_cost < old_cost:
                heapq.heappush( *args: queue, (new_cost, neighbor))
                shortest_paths[neighbor] = (current, new_cost)
    path = []
    while target is not None:
        path.append(target)
        next_node = shortest_paths[target][0]
        target = next_node
    path = path[::-1]
    return path
```

این تابع دو ورودی مبدا و مقصد را گرفته و یک لیست به عنوان خروجی باز می‌گرداند. کار این تابع بدین شکل است که ابتدا یک دیکشنری به عنوان shortest_paths ایجاد می‌کند که در آن کوتاه ترین مسیر تا هر نقطه را ذخیره می‌کند.(واضح است که فاصله اولین نقطه تا خودش ۰ است). سپس یک صف با نام queue ایجاد می‌شود که در آن نقاط با فاصله کمتر دارای اولویت بیشتری می‌باشند.

با وارد شدن به while، حلقه تا زمانی که صف خالی نشود ادامه می‌دهد. در خط بعدی با استفاده از heapq، نقطه با کمترین فاصله را حذف می‌کنیم. با وارد شدن به حلقه for، برای هر همسایه از نقطه فعلی: ۱- فاصله قدیمی را بدست می‌آوریم. ۲- فاصله جدید را نیز بدست می‌آوریم. ۳- اگر فاصله جدید از قدیم کوتاه تر بود آنگاه مسیر را ذخیره می‌کنیم.

سپس یک لیست خالی از مسیر می‌سازیم که قرار است با پیمایش مسیر از مقصد به مبداء رسیده و مسیر مورد نظر را به عنوان خروجی تابع برگردانیم.

الگوریتم A*

Calculate_distance

```python
def calculate_distance(lat1, lon1, lat2, lon2):
    # Convert latitude and longitude from degrees to radians
    lat1 = math.radians(lat1)
    lon1 = math.radians(lon1)
    lat2 = math.radians(lat2)
    lon2 = math.radians(lon2)

    # Haversine formula
    d_longitude = lon2 - lon1
    d_latitude = lat2 - lat1
    a = math.sin(d_latitude / 2) ** 2 + math.cos(lat1) * math.cos(lat2) * math.sin(d_longitude / 2) ** 2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))

    # Radius of earth in kilometers. Use 3956 for miles
    r = 6371.0

    # Calculate the distance
    distance = r * c
    return distance
```

این تابع با دریافت دو پارامتر Latitude و Longitude فرودگاه مبدا و مقصد، فاصله‌ی فیزیکی دو فرودگاه را محاسبه می‌کند. دلیل محاسبات بالا، کروی بودن شکل زمین و تفاوت فاصله در شکل کروی نسبت به شکل مسطح می‌باشد.

a_star_heuristic

```python
def a_star_heuristic(DestinationAirport):
    desLatitude = 0
    desLongitude = 0
    for i in range(6836):
        if df.iloc[i, 2] == DestinationAirport:
            desLatitude = df.iloc[i, 10]
            desLongitude = df.iloc[i, 11]
            break

    for node in enumerate(G.nodes):
        for i in range(6836):
            if df.iloc[i, 1] == node[1]:
                soLatitude = df.iloc[i, 5]
                soLongitude = df.iloc[i, 6]
                distance = calculate_distance(soLatitude, soLongitude, desLatitude, desLongitude)
                G.nodes[df.iloc[i, 1]]['heuristic'] = 67043 * distance
                break
            elif df.iloc[i, 2] == node[1]:
                soLatitude = df.iloc[i, 10]
                soLongitude = df.iloc[i, 11]
                distance = calculate_distance(soLatitude, soLongitude, desLatitude, desLongitude)
                G.nodes[df.iloc[i, 2]]['heuristic'] = 67043 * distance
                break
```

این تابع با دریافت نام فرودگاه مقصد، در حلقه‌ی for اول، ابتدا Latitude و Longitude فرودگاه را به‌دست می‌آورد. سپس در حلقه‌ی بعد، از کل جدول، پارامترهای Latitude و Longitude بقیه فرودگاه‌ها را به‌دست می‌آورد. این دو پارامتر به محاسبه‌ی فاصله‌ی فیزیکی دو فرودگاه، کمک می‌کند. دلیل اینکه ضریب فاصله 67043 است، این است که در دیتاست، قیمت و فاصله‌ی زمانی، رابطه‌ی مستقیم و خطی با فاصله‌ی فیزیکی دارند. بنابراین می‌توان از جمع ضرایبی که از تقسیم فاصله‌فیزیکی، به فاصله‌ی زمانی یا قیمت به‌دست می‌آیند، به عنوان ضریب مناسبی برای تخمین تابع heuristic استفاه کرد.

a_star_algorithm

```python
def a_star_algorithm(SourceAirport, DestinationAirport):
    a_star_heuristic(DestinationAirport)
    queue = [(0, SourceAirport)]
    visited = set()
    cost_so_far = {SourceAirport: 0}
    came_from = {SourceAirport: None}
    current = None

    while queue:
        cost, current = heapq.heappop(queue)
        if current == DestinationAirport:
            break
        visited.add(current)
        for node in list(G.successors(current)):
            new_cost = cost_so_far[current] + G.get_edge_data(current, node)['weight']
            if node not in cost_so_far or new_cost < cost_so_far[node]:
                cost_so_far[node] = new_cost
                priority = new_cost + G.nodes[node]['heuristic']
                heapq.heappush( *args: queue, (priority, node))
                came_from[node] = current

    path = []
    while current is not None:
        path.append(current)
        current = came_from[current]
    path.reverse()  # Reverse the path
    return path
```

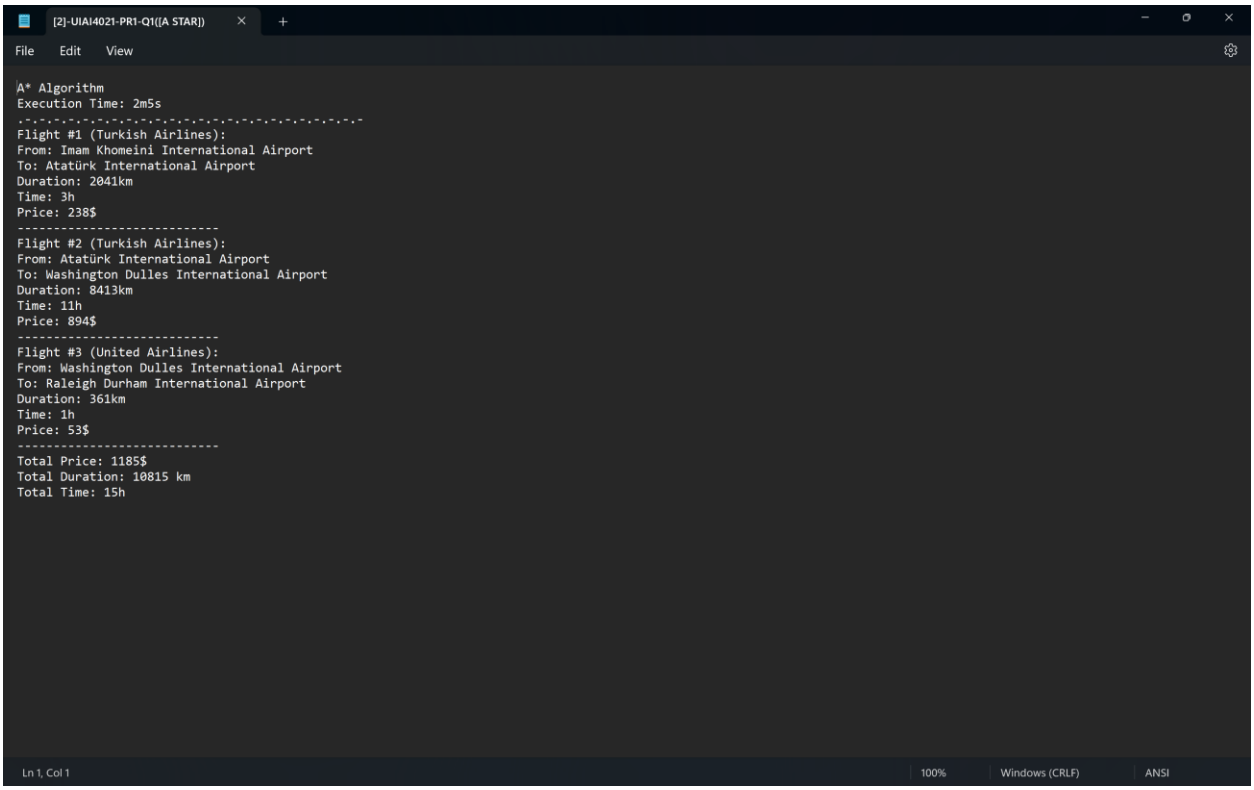
این تابع در ابتدا با دریافت نام فرودگاه مبدا و مقصد، با توجه به فرودگاه مقصد، مقدار heuristic هر فرودگاه را ذخیره می‌کند.سپس صف اولویتی با فرودگاه مبدا می‌سازد؛ و visited را ستی از فرودگاه‌های بازدیدشده تشکیل می‌دهد. cost_so_far هزینه هر گره تا الان می‌باشد. came_from گره‌ای که از آن آمده‌ایم. current به معنای گره فعلی می‌باشد.

تا زمانی که صف خالی نشده‌است، گره‌ای که کمترین هزینه را دارد از صف برمیداریم و اگر گره فعلی، مقصد باشد، حلقه متوقف می‌شود. درغیراین‌صورت، گره فعلی به مجموعه گره‌های بازدید شده اضافه می‌شود. سپس برای هر گره همسایه (فرودگاهی که از این فرودگاه برایش بلیت موجود است) هزینه جدید را محاسبه می‌کنیم و اگر گره جدید کمترین هزینه را داشته باشد، هزینه اصلی بروزرسانی می‌شود، اولویت این گره مشخص می‌شود، گره را به صف اولویت اضافه می‌کنیم.

در نهایت، از روی came_from مسیر نهایی را مشخص می‌کنیم.

نمونه‌ای از خروجی

A*



```
A* Algorithm
Execution Time: 2m5s
.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-
Flight #1 (Turkish Airlines):
From: Imam Khomeini International Airport
To: Atatürk International Airport
Duration: 2041km
Time: 3h
Price: 238$
---------------------------
Flight #2 (Turkish Airlines):
From: Atatürk International Airport
To: Washington Dulles International Airport
Duration: 8413km
Time: 11h
Price: 894$
---------------------------
Flight #3 (United Airlines):
From: Washington Dulles International Airport
To: Raleigh Durham International Airport
Duration: 361km
Time: 1h
Price: 53$
---------------------------
Total Price: 1185$
Total Duration: 10815 km
Total Time: 15h
```

Dijkstra

```
Dijkstra Algorithm
Execution Time: 0m0s
.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-
Flight #1 (Turkish Airlines):
From: Imam Khomeini International Airport
To: Atatürk International Airport
Duration: 2041km
Time: 3h
Price: 238$
----------------------------
Flight #2 (Turkish Airlines):
From: Atatürk International Airport
To: John F Kennedy International Airport
Duration: 8052km
Time: 10h
Price: 837$
----------------------------
Flight #3 (United Airlines):
From: John F Kennedy International Airport
To: Washington Dulles International Airport
Duration: 366km
Time: 1h
Price: 52$
----------------------------
Flight #4 (United Airlines):
From: Washington Dulles International Airport
To: Raleigh Durham International Airport
Duration: 361km
Time: 1h
Price: 53$
----------------------------
Total Price: 1180$
Total Duration: 10820 km
Total Time: 15h
```

کتابخانه‌های استفاده شده

networkx : برای ساخت گراف، استفاده از گره‌ها و یال‌های گراف

pandas : استفاده از دیتاهای دیتاست

math : به‌دست آوردن فاصله‌ی دو فرودگاه با استفاده از طول و عرض جغرافیایی فرودگاه‌ها

heapq : صف هر دو الگوریتم

time : به‌دست آوردن زمان اجرای هر الگوریتم

منابع

https://www.w3schools.com/python/pandas/default.asp

https://www.udacity.com/blog/2021/10/implementing-dijkstras-algorithm-in-python.html

https://pypi.org/project/networkx/

https://blog.enterprisedna.co/python-write-to-file/#:~:text=The%20write()%20method%20is,it%20to%20the%20specified%20file

Bing Chat with GPT-4