Dokumentacja Wyznaczania najkrótszej trasy lotu z wykorzystaniem języka Python

1. **Importowanie Bibliotek:**
   - W pierwszej sekcji importowane są niezbędne biblioteki do obsługi matematycznych obliczeń, rysowania wykresów i animacji.

```
1.. import numpy as np
2.  import matplotlib.pyplot as plt
3.. from mpl_toolkits.basemap import Basemap
4.. from itertools import permutations
5.  from matplotlib.animation import FuncAnimation
6.
```

2. **Współrzędne Lotnisk:**
   - Lotniska są zdefiniowane jako słownik, gdzie kluczem jest nazwa lotniska, a wartością są współrzędne geograficzne.

```
1.  airports = {
2.          "Katowice": (50.4745, 19.0808),
3.           "London": (51.4694, -0.4474),
4.          "Paris": (49.0097, 2.5479),
5.          "Reykjavik": (63.9850, -22.6051),
6.           "Oslo": (60.1976, 11.1004),
7.           "Ankara": (40.1289, 32.9959)
8. }
```

3. **Generowanie Połączeń:**
   - Funkcja `cities_connection` generuje listę możliwych połączeń między lotniskami.

```
1.  def cities_connection(cities):
2.      list_of_connections = []
3.      for city in cities.keys():
4.          for city2 in cities.keys():
5.              if city < city2:
6.                  list_of_connections.append((city, cities[city], city2, cities[city2]))
7.      return list_of_connections
8.
9.
```

Funkcja `cities_connection` tworzy listę możliwych połączeń między lotniskami na podstawie słownika zawierającego nazwy lotnisk i ich współrzędne (x, y). Funkcja tworzy pustą listę `list_of_connections` i następnie używa dwóch zagnieżdżonych pętli do iteracji przez nazwy lotnisk. Warunek sprawdza, czy nazwa pierwszego lotniska jest alfabetycznie wcześniejsza niż nazwa drugiego. Jeśli warunek jest spełniony, dodaje połączenie do listy `list_of_connections` jako krotkę zawierającą informacje o obu lotniskach. Po zakończeniu obu pętli, funkcja zwraca listę `list_of_connections` z unikalnymi połączeniami, eliminując powtórzenia (np. A-B i B-A).

4. **Obliczenia Odległości:**
    - Funkcja `haversine` wykorzystuje wzór haversine do obliczenia odległości między lotniskami, tworząc słownik odległości dla każdego połączenia.

```
1. def haversine(list_of_connections):
2.     dic = {}
3.     for i in range(len(list_of_connections)):
4.         lat1 = list_of_connections[i][1][0]
5.         lon1 = list_of_connections[i][1][1]
6.         lat2 = list_of_connections[i][3][0]
7.         lon2 = list_of_connections[i][3][1]
8.
9.         con = list_of_connections[i][0] + " - " + list_of_connections[i][2]
10.
11.         dLat = (lat2 - lat1) * np.pi / 180.0
12.         dLon = (lon2 - lon1) * np.pi / 180.0
13.         lat1 = (lat1) * np.pi / 180.0
14.         lat2 = (lat2) * np.pi / 180.0
15.
16.         a = (np.sin(dLat / 2) ** 2 +
17.              np.sin(dLon / 2) ** 2 *
18.              np.cos(lat1) * np.cos(lat2))
19.         rad = 6371
20.         c = 2 * np.arcsin(np.sqrt(a))
21.         dic[con] = round(rad * c)
22.     return dic
23.
```

Funkcja `haversine` oblicza odległości między lotniskami na podstawie ich współrzędnych, korzystając z wzoru haversine. Tworzy pusty słownik `dic` do przechowywania informacji o odległościach między poszczególnymi lotniskami. Następnie, w pętli `for`, iteruje przez elementy listy połączeń, pobierając współrzędne geograficzne dla dwóch lotnisk. Oblicza różnice szerokości i długości geograficznej (`dLat` i `dLon`), konwertuje je na radiany. Wykorzystuje wzór haversine do obliczenia odległości między dwoma punktami na sferze ziemi. Wynik jest zaokrąglany i dodawany do słownika `dic` pod kluczem, który składa się z nazw obu lotnisk połączonych myślnikiem. Słownik z odległościami jest zwracany na koniec funkcji.

5. **Obliczenie całkowitej długości trasy:**
    - Funkcja `calculate_total_distance` oblicza łączną odległość dla danej trasy na podstawie słownika odległości.

```
1. def calculate_total_distance(route, distances):
2.     total_distance = 0
3.     for i in range(len(route) - 1):
4.         key_forward = f'{route[i]} - {route[i + 1]}'
5.         key_backward = f'{route[i + 1]} - {route[i]}'
6.         if key_forward in distances:
7.             total_distance += distances[key_forward]
```

```
        elif key_backward in distances:
            total_distance += distances[key_backward]

    key_return = f'{route[-1]} - {route[0]}'
    if key_return in distances:
        total_distance += distances[key_return]
    return total_distance

```

Funkcja `calculate_total_distance` oblicza łączną odległość trasy lotniczej na podstawie słownika odległości między poszczególnymi lotniskami. Tworzona jest zmienna `total_distance` o początkowej wartości 0. W pętli `for` iterującej przez indeksy trasy, dla każdego odcinka trasy obliczany jest klucz do sprawdzenia w słowniku odległości. Warunek sprawdza, czy odległość między lotniskami (w obie strony) istnieje w słowniku. Jeśli tak, dodaje tę odległość do łącznej odległości. Ponieważ trasa jest zamknięta, sprawdzane jest także połączenie pomiędzy ostatnim a pierwszym lotniskiem. Na koniec funkcja zwraca obliczoną łączną odległość trasy.

6. **Rozwiązanie Problemu Komiwojażera:**
    - Funkcja `tsp_bruteforce` implementuje algorytm bruteforce w celu znalezienia najoptymalniejszej trasy.

```
def tsp_bruteforce(distances):
    cities = set()
    for key in distances:
        cities.update(key.split(' - '))

    shortest_route = None
    shortest_distance = float('inf')

    for route_permutation in permutations(cities):
        route_permutation += (route_permutation[0],)
        total_distance = calculate_total_distance(route_permutation, distances)

        if total_distance < shortest_distance:
            shortest_distance = total_distance
            shortest_route = route_permutation
    return shortest_route, shortest_distance

```

Funkcja `tsp_bruteforce` jest odpowiedzialna za rozwiązanie problemu komiwojażera za pomocą metody brute-force. Poniżej jest jej opis bez punktów: Funkcja rozpoczyna od identyfikacji unikalnych miast na podstawie kluczy w słowniku odległości między lotniskami. Inicjalizuje zmienną `shortest_route` na `None` oraz `shortest_distance` na nieskończoność, aby przechowywać informacje o najkrótszej trasie i jej długości. Następnie, za pomocą pętli `for`, generuje wszystkie możliwe permutacje miast. Każda permutacja jest rozszerzana o pierwsze miasto, aby zamknąć trasę. Dla każdej permutacji obliczana jest łączna odległość za pomocą funkcji `calculate_total_distance`. Jeśli uzyskana odległość jest krótsza niż dotychczasowa najkrótsza, aktualizowane są zmienne

`shortest_distance` i `shortest_route`. Po przejściu przez wszystkie permutacje, funkcja zwraca najkrótszą trasę i jej długość.

7. **Rysowanie Trasy na Mapie:**
    - Funkcja `draw_shortest_route` używa biblioteki Basemap do utworzenia mapy z zaznaczoną najkrótszą trasą, lotniskami i dodatkowymi informacjami.

```
1. def draw_shortest_route(frame):
2.     plt.clf()
3.     m = Basemap(projection='mill', llcrnrlat=30, urcrnrlat=75,
4.                 llcrnrlon=-30, urcrnrlon=60, resolution='c')
5.     m.drawcoastlines()
6.     m.drawcountries()
7.     route_coordinates = [airports[city] for city in shortest_route[:frame+1]]
8.     route_coordinates = np.array(route_coordinates)
9.     x, y = m(route_coordinates[:, 1], route_coordinates[:, 0])
10.    m.plot(x, y, 'mo--', markersize=10, linewidth=2)
11.    for city, coordinates in airports.items():
12.        x, y = m(coordinates[1], coordinates[0])
13.        m.plot(x, y, 'ro', markersize=8)
14.        plt.text(x, y, city, fontsize=12, ha='left', color='blue')
15.
16.    plt.title(f'Optimal route for DreamFlight airline')
17.    plt.axis('off')
18.    plt.subplots_adjust(bottom=0.1)
19.    shortest_route_str = ""
20.    for airport in shortest_route:
21.        shortest_route_str += airport + " - "
22.    shortest_route_str = shortest_route_str[:-2]
23.    plt.text(0.5, -0.05, f'Shortest route: {shortest_route_str}', fontsize=12, ha='center',
transform=ax.transAxes)
24.    plt.text(0.5, -0.09, f'Shortest distance: {shortest_distance} km', fontsize=12, ha='center',
transform=ax.transAxes)
25.
```

Funkcja `draw_shortest_route` odpowiada za rysowanie na mapie najkrótszej trasy lotniczej. Poniżej opis działania tej funkcji bez punktów: Funkcja inicjuje nową mapę Basemap z określonymi parametrami projekcji i obszaru. Następnie dodaje elementy mapy, takie jak linie brzegowe i granice państw. Współrzędne lotnisk dla aktualnej części trasy są pobierane z globalnej zmiennej `airports`. Te współrzędne są przekształcane na macierz NumPy. Następnie, przy użyciu funkcji `m.plot`, rysowana jest część trasy na mapie w postaci niebieskich kropek połączonych liniami przerywanymi. Dodatkowo, każde lotnisko jest oznaczone czerwoną kropką, a jego nazwa jest wyświetlana obok niego na mapie. Tytuł mapy oraz dodatkowe informacje o najkrótszej trasie i jej długości są dodawane do wykresu. Na koniec funkcja przystosowuje wykres, usuwając osie i dostosowując położenie tekstu. Funkcja ta jest wywoływana w animacji, gdzie kolejne klatki prezentują stopniowe uzupełnianie się trasy.

8. **Wizualizacja Wyników:**
    - Na zakończenie działania, wizualizacja prezentuje najkrótszą trasę, jej długość oraz dodatkowe informacje na temat lotnisk.

```
1. print("Najkrótsza trasa:", shortest_route)
2. print("Najkrótsza odległość:", shortest_distance)
3.
```

9. **Animacja Trasy:**
    - Wykorzystano `matplotlib.animation` do stworzenia animacji prezentującej kolejne etapy najkrótszej trasy.

```
1.  fig, ax = plt.subplots(figsize=(12, 10))
ani = FuncAnimation(fig, draw_shortest_route, frames=len(shortest_route), interval=200,
repeat=True)

2. plt.show()
```

Inicjalizacja obiektu fig, ax za pomocą funkcji `plt.subplots`, ustawiając rozmiar wykresu na (12, 10). Następnie tworzony jest obiekt animacji ani przy użyciu `FuncAnimation`. Ten obiekt wykorzystuje wcześniej zdefiniowaną funkcję draw_shortest_route do rysowania kolejnych etapów trasy. Parametr frames określa liczbę klatek animacji, którą dostarcza długość najkrótszej trasy. Parametr interval określa czas trwania jednej klatki w milisekundach. repeat=True oznacza, że animacja będzie się powtarzać. Wywołanie `plt.show()` prezentuje wygenerowany wykres w interaktywnym trybie.