Raport z laboratorium nr: 8

Imię i nazwisko autora: Grzegorz Piśkorski

Nr indeksu: 405879

1. **Najważniejszy fragment kodu z laboratoriów:**

```
V = len(G)
Vertices = [None for i in range(V)]
Edges = PriorityQueue()  # w postacie (waga, z, do)
MST = []
for i in range(V):
    Vertices[i] = Node(i)

for v in range(V):
    for u in range(v, V):
        if G[u][v] > -1:
        Edges.put((G[v][u], v, u))

while not Edges.empty():
    weight, u, v = Edges.get()
    x = find(Vertices[u])
    y = find(Vertices[v])

    if x != y:
        MST.append((u, v))
        union(x, y)
```

Uzasadnienie wyboru:

Wybrałem część kodu związaną z algorytmem Kruskala (bez funkcji find(x) oraz union(x, y) i deklaracji klasy Node, ponieważ nie chciałem aż tak bardzo przekraczać limity wybranych linijek). Moim zdaniem jest to najważniejsza część całego niżej opisanego algorytmu, ponieważ tworzy nam on minimalne drzewo rozpinające, które jest kluczem dla mojego algorytmu. To właśnie na podstawie tego drzewa obliczana jest później optymalniejsza trasa.

Podsumowanie

W Zadaniu 2 skorzystałem z podanego poniżej algorytmu w liście kroków:

1) Tworzymy graf warzony reprezentujący nasze wszystkie miasta, gdzie każda krawędź $\{u, v\} \epsilon V$ oznacza odległość między miastem u i v (macierz sąsiedztwa).
2) Szukamy minimalnego drzewa rozpinającego dla tego grafu algorytmem Kruskala.
3) Wcześniej stworzony graf przekształcamy w graf reprezentujący nasze minimalne drzewo rozpinające.
4) Przechodzimy algorytmem DFS po naszym drzewie w tym samym czasie tworząc listę zawierającą kolejność odwiedzonych przez nas wierzchołków.

5) Po otrzymaniu naszej listy wierzchołków obliczamy naszą optymalniejszą ścieżkę w kolejności takiej jaką zwrócił nam DFS.

Natomiast w Zadaniu 1 moje rozwiązanie polega na wylosowaniu jakiejś permutacji na samym początku i obliczeniu wylosowanej trasy.

10 wyników z Zadania 1:

```
Sciezka wybrana losowo 1: 5233.83909702383
Sciezka wybrana losowo 2: 5016.861351786135
Sciezka wybrana losowo 3: 5142.936210708904
Sciezka wybrana losowo 4: 5137.834101514168
Sciezka wybrana losowo 5: 5115.548085130687
Sciezka wybrana losowo 6: 5046.534535763458
Sciezka wybrana losowo 7: 4986.285639349774
Sciezka wybrana losowo 8: 4800.1128592557125
Sciezka wybrana losowo 9: 4883.410522352029
Sciezka wybrana losowo 10: 4851.701394805346
```

Jak widzimy trasy nie wydają się być zbliżone do optymalnej, która tak jak Pan powiedział na początku laboratoriów powiedział, że może być nawet w okolicach 600.

Zadanie 2 w moim przypadku za każdym razem daje takie samo rozwiązanie:

```
Sciezka typu 1 -> 2 -> 3 -> ... -> 100: 5084.461369051879
Moja optymalniejsza sciezka:  1058.0727137105723
Czas wykonania sie algorytmu: 0.027740478515625
Stosunek pierwszej sciezki 1 - 100 do znalezionej sciezki: 4.805398819161586
```

Jak widzimy powyższy algorytm znalazł ścieżkę prawie 5 razy krótszą od leksykograficznej kolejności odwiedzania miast. Wykonał się też relatywnie szybko, ale trzeba zwrócić uwagę na to, że dane nie są zbyt duże.

Złożoność obliczeniowa wykorzystanego algorytmu (V – liczba wierzchołków, E – liczba krawędzie):

- Stworzenie grafu – O(V^2)

- Algorytm Kruskala – O(E*logV)

- DFS dla grafu danego macierzą – O(V^2)

- Obliczenie wartości ścieżki – O(V)

Podsumowując, złożoność tego algorytmu to O(V^2)