LAB6: Algorytmy grafowe – algorytmy zachłanne dla zagadnienia komiwojażera

Zadanie 1

Implementacje algorytmów poszukiwania rozwiązania problemu TSP:

- FARIN

In [1]:
```
from typing import List, Dict, Tuple, Set
import matplotlib.pyplot as plt
import networkx as nx
import copy
edge = Tuple[int, int]
graph = Dict[int, List]
inf = float('inf')



def FARIN(G: graph, a: List[List[int]], s: int):
    """
    Algorytm Farthest Insertion Heuristik FARIN, rozwiązujący problem komiwojażera(
    przyjmujący graf "G", listę list wag krawędzi grafu "a" oraz wierzchołek starto
    Algorytm zwraca krotkę zawierającą cykl Hamiltona i jego sumaryczną wagę
    """
    cost = 0     #inicjalizacja kosztu drogi
    path = [s]      #inicjalizacja listy wierzchołków cyklu Hamiltona, zaczynając o
    unvisited = set(G.keys()) #zbiór wierzchołków nieodwiedzonych
    unvisited.remove(s) #ze bioru tego usuwamy wierzchołek startowy
    u_prev = s #ustawiamy wierzchołek startowy jako poprzedni

    while unvisited: #wykonujemy pętlę główną tak długo jak zostają jakieś wierzcho

        #wybór następnego wierzchołka najbardziej odległy lub kilka odległych
        a_row = [inf for _ in range(len(a))] #inicjalizacja listy
        for unv in unvisited:
            a_row[unv-1] = a[u_prev-1][unv-1] #przypisanie wag krawędzi z u_prev do
        sorted_a = sorted(a_row, reverse=True) #posortowanie
        to_visit = []
        curr_max = -inf
        #wybór najdalszych
        for v in sorted_a:
            #jeżeli wierzchołek nie jest odwiedzony, jego waga != inf oraz jest wię
            if a_row.index(v)+1  in unvisited and v != inf and v >= curr_max:
                curr_max = v
                idx = a_row.index(v)
                a_row[idx] = inf
                to_visit.append(idx+1) #dodanie najdaljszych do listy do odwiedzeni
```

```python
        #przejscie przez wierzchołek lub wierzchołkiktóre są najdalej od u_prev
        for next in to_visit:
            #ilość opcji wstawienia nowego wierzchołka
            places = len(path)
            #ustawienie zmiennych do aktualizowania ścieżki o najlepszym koszcie
            min_cost = inf
            best_cost_path = path
            #iteracja przez możliwości wstawienia wierzchołka
            for i in range(places):
                #skopiowanie dotychczasowej ścieżki
                optional_path = path[:]
                #dodanie nowego wierzcholka na nową pozycję
                optional_path.insert(i+1, next)
                optional_path.append(s)

                #obliczenie kosztu nowej opcjonalej ścieżki
                new_cost = 0
                for idx, elem in enumerate(optional_path[:-1]):
                    new_cost += a[elem-1][optional_path[idx+1]-1]
                #jeżeli nowy koszt jest mniejszy bądz równy to aktualizujemy ścieżk
                if min_cost >= new_cost:
                    min_cost = new_cost
                    best_cost_path = optional_path[:-1]

            #aktualizacja ścieżki do tej o najmniejszym koszcie
            path = best_cost_path
            #usunięcie wierzchołka obecnego z nieodwiedzonych
            unvisited.remove(next)

        #ustawienie wierzchołka ostatnio dodanego na poprzedni
        u_prev = next

    cost = min_cost #dodanie minimalnego kosztu
    path.append(s) #dodanie wierzchołka początkwoego do domknięcia cyklu Hamiltona

    return cost, path
```

Test alorytmu dla przykładu z wykładu

In [2]:

```python
graph = {
    1: [2,3,4,5],
    2: [1,3,4,5],
    3: [1,2,4,5],
    4: [1,2,3,5],
    5: [1,2,3,4],
}

a = [
    [inf, 5, 4, 6, 6],
    [8, inf, 5, 3, 4],
    [4, 3, inf, 3, 1],
    [8, 2, 5, inf, 6],
    [2, 2, 7, 0, inf]
]
```

```
print(FARIN(graph, a, 1))
```

```
(16, [1, 4, 2, 3, 5, 1])
```

Zadanie 2

- Z punktu widzenia działania algorytmu należy zaznaczyć, że na wynik działania nie ma wpływu skierowanie grafu, gdyż może badany być graf skierowany i nieskierowany, także o wagach ujemnych. Kluczowym faktem jest to, że alogrytmy te są zachłanne przez co nie zawsze możemy otrzymać optymalne rozwiązanie. Rozwiązania będą też różnić się w zależności od wyboru wierzchołka startowego, gydż wtedy analizować będziemy inne przejścia pomiędzy danymi wierzchołkami.

Kluczową własnościa jest spójność grafu, gdyż dla gradu niespójnego algorytm nie wykona się (nie znajdzie scieżki pomiędzy 2 niespójnymi częściami).

Zadanie 3