МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ”ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра «Систем Автоматизованого Проектування»

Звіт
Лабораторна робота № 4
з курсу *“*Дискретні моделі*”*
на тему: « ПОТОКОВІ АЛГОРИТМИ»

Виконав:
Ст.гр. КН-409
Єлечко Олег
Прийняв:
Кривий Р.З.

Львів 2023

Мета роботи: Метою даної лабораторної роботи є вивчення потокових алгоритмів.

Теоретичні відомості:

Потік-визначає спосіб пересилання деяких об'єктів з одного пункту в інший. Розв'язання задачі потоку зводиться до таких основних підзадач:

- Максимізація сумарного обсягу перевезень
- Мінімізація вартості пересилань предметів з одного пункту в інший
- Мінімізація часу перевезень в заданій системі Сітка - це граф, в якому кожній дузі приписана деяка пропускна здатність. Введемо позначення: c(x,y) - пропускна здатність дуги (x,y), a(x,y) - вартість переміщення одиниці потоку по дузі (x,y), T(x,y) - час проходження потоку, k(x,y) - коефіцієнт підсилення потоку в дузі (x,y). Припустимо, що є граф, в якому деяка кількість одиниць потоку проходить від джерела до стоку і для кожної одиниці потоку відомий маршрут руху. Назвемо кількість одиниць, що проходять по дузі (x,y), потоком в даній дузі. Будемо потік в дузі(x,y) позначати через f(x,y) вочевидь 0 f(x,y) c(x,y).

Дуги графа можна віднести до трьох різних категорій:

1. дуги, в яких потік не може ні збільшуватись, ні зменшуватись (множина таких дуг позначається через - N);
2. дуги, в яких потік може збільшуватись (множина таких дуг позначається через - I);
3. дуги, в яких потік може зменшуватись (множина таких дуг позначається через - R);

Наприклад, дуги, що мають нульову пропускну здатність або значну вартість проходження потоку, повинні належати множині N. Дуги, в яких потік менше пропускної здатності, повинні належати множині I. Дуги, по яких вже проходить деякий потік, повинні належати множині R. Дуги з множини I називають збільшуючими, а дуги з множини R - зменшуючими. Будь-яка дуга графа належить хоча б одній з трьох введених множин - I, R або N.

Можливо, що якась дуга належить як множині I, так і множині R. Це має місце в тому випадку, коли по дузі вже протікає деякий потік, який можна збільшувати чи зменшувати. Відповідні дуги називаються проміжними. Позначимо через i(x,y) максимальну величину, на яку може бути збільшений потік в дузі (x,y). Відповідно позначимо через r(x,y) максимальну величину, на яку може бути зменшений потік в дузі (x,y). Очевидно, i(x,y) = c(x,y) - f(x,y) , а r(x,y) = f(x,y)

ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Отримати у викладача індивідуальне завдання.
2. Підготувати програму для вирішення виданого завдання.
3. Запустити на виконання програму відповідного методу.
4. Проглянути результат роботи програм. Результат роботи може бути позитивним (шлях знайдено) або негативним (шлях відсутній).
5. Порівняти результати, отримані за допомогою різних алгоритмів і зробити висновок.
6. Зафіксувати результати роботи у викладача.
7. Оформити і захистити звіт

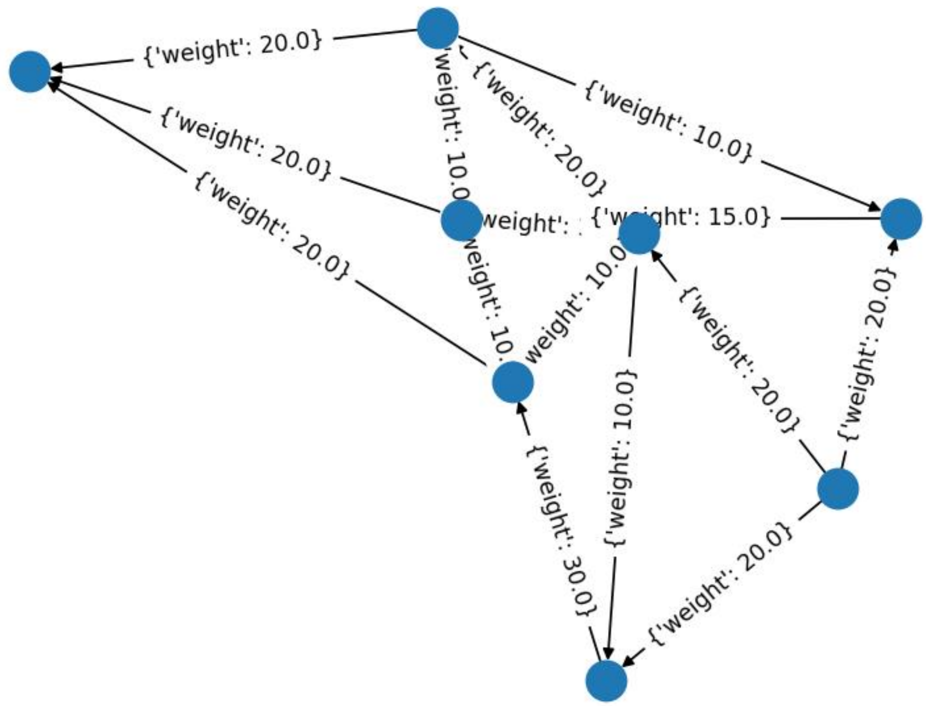
Github-link: https://github.com/OlehYelechko/Labs_DM/tree/main/Lab4

Код алгортиму Форда-Фалкерсона.

```
# Імпортуємо необхідні бібліотеки
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np



path1 = "l4_1.txt"
path2 = "l4_2.txt"



def ford_fulkerson(adj_matrix):
    n = len(adj_matrix)
    flow_matrix = np.zeros((n, n))

    # Create residual graph
    residual_graph = np.copy(adj_matrix)

    # Initialize variables
    max_flow = 0
    path = find_augmenting_path(residual_graph, 0, n-1)

    # Iterate until no augmenting path can be found
    while path:
        # Compute bottleneck capacity
        bottleneck_capacity = min(residual_graph[u][v] for u, v in path)

        # Update flow matrix and residual graph
        for u, v in path:
            flow_matrix[u][v] += bottleneck_capacity
            residual_graph[u][v] -= bottleneck_capacity
            residual_graph[v][u] += bottleneck_capacity

        # Update max flow and find next augmenting path
        max_flow += bottleneck_capacity
        path = find_augmenting_path(residual_graph, 0, n-1)

    return max_flow, flow_matrix

def find_augmenting_path(residual_graph, source, sink):
    n = len(residual_graph)

    # Initialize variables
    visited = [False] * n
    queue = [(source, [])]
```

```
    # BFS search
    while queue:
        current, path = queue.pop(0)
        visited[current] = True

        # Check if sink node is reached
        if current == sink:
            return path

        # Add unvisited neighbors to queue
        for neighbor in range(n):
            if residual_graph[current][neighbor] > 0 and not visited[neighbor]:
                queue.append((neighbor, path + [(current, neighbor)]))

    # No augmenting path found
    return None


with open(path1) as f:
    lines = (line for line in f if not line.startswith('#'))
    graph = np.loadtxt(lines, skiprows=1)

G = nx.from_numpy_matrix(np.matrix(graph), create_using=nx.DiGraph)
layout = nx.spring_layout(G)
nx.draw(G, layout)
nx.draw_networkx_edge_labels(G, pos=layout)
plt.show()

print(graph, end="\n\n")
# Визначаємо кількість вершин у графі
V = len(graph)

max_flow, flow_matrix = ford_fulkerson(graph)
# викликаємо функцію tsp()
print("Найбільший потік:" + str(max_flow))
print(flow_matrix)
```

Результат виконання:




```
[[ 0. 20. 20. 20.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. 30.  0.  0.  0.]
 [ 0. 10.  0.  0.  0. 10. 20.  0.]
 [ 0.  0.  0.  0.  0. 15.  0.  0.]
 [ 0.  0. 10.  0.  0. 10.  0. 20.]
 [ 0.  0.  0.  0.  0.  0. 10. 20.]
 [ 0.  0.  0. 10.  0.  0.  0. 20.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]]

Найбільший потік:55.0
[[ 0. 20. 20. 15.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. 20.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. 10. 10.  0.]
 [ 0.  0.  0.  0.  0. 15.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0. 20.]
 [ 0.  0.  0.  0.  0.  0.  5. 20.]
 [ 0.  0.  0.  0.  0.  0.  0. 15.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]]
```

Висновок: на цій лабораторній роботі було реалізовано метод Форда-Фалкерсона для вирішення задачі Комівояжера на мові Python. Максимальний потік = 55