# НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

**Самостійна робота модуль 1**

## Виконав

студент групи ІПЗ-23 1ст

Свириденко Т. А.

[TimaSv-Uk/nubip\_graphs (github.com)](https://github.com/TimaSv-Uk/nubip_graphs)

У цьому коді використовуються три основні бібліотеки Python: matplotlib, networkx та matplotlib.colors.

1. **NetworkX**: Це бібліотека для створення, маніпулювання та аналізу структури, динаміки та функцій складних мереж. У цьому коді вона використовується для створення багатократного графа (MultiGraph), додавання країв та ваги до графа.
2. **Matplotlib**: Це бібліотека для створення статичних, анімованих та інтерактивних візуалізацій в Python. У цьому коді вона використовується для візуалізації графа.
3. **Matplotlib.colors**: Це модуль у matplotlib, який використовується для перетворення кольорів між RGB та іншими системами кольорів. У цьому коді він використовується для створення словника кольорів для вузлів графа.
4. Алгоритм для знаходження гамільтонових ланцюгів та циклів адаптовано з сайту [Print all Hamiltonian paths present in a graph | Techie Delight](https://www.techiedelight.com/print-all-hamiltonian-path-present-in-a-graph/)

Особливості роботи цих бібліотек:

* **NetworkX** дозволяє легко створювати графи за допомогою простих команд. Ви можете додавати вузли, краї та ваги до графа. Ви також можете виконувати різні операції з графами, такі як пошук найкоротшого шляху, обчислення центральності вузла тощо.
* **Matplotlib** дозволяє візуалізувати дані на графіках. Ви можете налаштувати різні параметри графіка, такі як кольори, розміри, мітки осей тощо.
* **Matplotlib.colors** дозволяє працювати з кольорами у ваших графіках. Ви можете використовувати його для перетворення кольорів між різними системами кольорів або для створення власних кольорових карт.
* **Якщо граф не е ейлеревим ми робимо його таким бібліотекою nx.eulerize(G) NetworkX**

Структура графу береться с txt файлу де на кожній лінії 3 цифри вершини які з’їданні ребром та вага ребра, в звіті використан граф з роботи 3

Програма проводить розрахунки:

1. Вивести визначення та значення основних характеристик графа (матриця суміжності, інциндентності, хроматичне число та реберне хроматичне число та інші)
2. Написати програму для знаходження всіх ейлерових ланцюгів та циклів
3. Написати програму для знаходження всіх гамільтонових ланцюгів та циклів
4. Реалізувати програмно обхід графа пошуком углиб
5. Реалізувати програмно обхід графа пошуком вшир
6. На мові програмування реалізувани пошук найкородших відстаней на графі від заданої вершини (алгоритм Дейкстри). Вивести схему маршрутів і довжину відстаней.
7. На мові програмування реалізувани пошук найкородших відстаней між будь-якими двома вершинами графу (алгоритм Флойда). Вивести схему маршрутів і довжину відстаней.
8. Знайти між якою парою вершин найкоротша відстань є найдовшою серед всіх найкоротших відстаней.
9. Знайти між якою парою вершин найкоротша відстань є найкоротшою серед всіх найкоротших відстаней.

гамільтоновиі ланцюги

**Програма виводить в консоль:**

Матриця суміжності:

(0, 1) 6

(0, 2) 8

(0, 3) 3

(0, 4) 3

(1, 0) 6

(1, 2) 8

(1, 3) 4

(2, 0) 8

(2, 1) 8

(2, 4) 6

(3, 0) 3

(3, 1) 4

(3, 4) 4

(3, 5) 3

(4, 0) 3

(4, 2) 6

(4, 3) 4

(5, 3) 3

Матриця інциндентносi:

(0, 0) 1.0

(1, 0) 1.0

(0, 1) 1.0

(2, 1) 1.0

(0, 2) 1.0

(3, 2) 1.0

(0, 3) 1.0

(4, 3) 1.0

(1, 4) 1.0

(2, 4) 1.0

(1, 5) 1.0

(3, 5) 1.0

(2, 6) 1.0

(4, 6) 1.0

(3, 7) 1.0

(5, 7) 1.0

(3, 8) 1.0

(4, 8) 1.0

Хроматичне число: 3

Реберне хроматичне число: 5

Граф не є ейлеровим

Ребра ейлерового циклу: [(1, 6), (6, 3), (3, 6), (6, 4), (4, 5), (5, 4), (4, 2), (2, 4), (4, 1), (1, 3), (3, 2), (2, 1)]

Граф не має ейлерового шляху.

Оригінальний граф:

[(1, 2), (1, 3), (1, 4), (1, 6), (2, 3), (2, 4), (3, 6), (4, 5), (4, 6)]

Новий ейлерів граф:

[(1, 2), (1, 3), (1, 4), (1, 6), (2, 3), (2, 4), (2, 4), (3, 6), (3, 6), (4, 5), (4, 5), (4, 6)]

Ребра ейлерового шляху: [(1, 2), (2, 3), (3, 1), (1, 4), (4, 2), (2, 4), (4, 5), (5, 4), (4, 6), (6, 3), (3, 6), (6, 1)]

Гамільтоновиі ланцюги:

[1, 2, 3, 6, 4, 5]

[1, 6, 3, 2, 4, 5]

[2, 1, 3, 6, 4, 5]

[2, 3, 1, 6, 4, 5]

[2, 3, 6, 1, 4, 5]

[3, 2, 1, 6, 4, 5]

[3, 6, 1, 2, 4, 5]

[6, 1, 3, 2, 4, 5]

[6, 3, 1, 2, 4, 5]

[6, 3, 2, 1, 4, 5]

[5, 4, 1, 2, 3, 6]

[5, 4, 1, 6, 3, 2]

[5, 4, 2, 1, 3, 6]

[5, 4, 2, 1, 6, 3]

[5, 4, 2, 3, 1, 6]

[5, 4, 2, 3, 6, 1]

[5, 4, 6, 1, 2, 3]

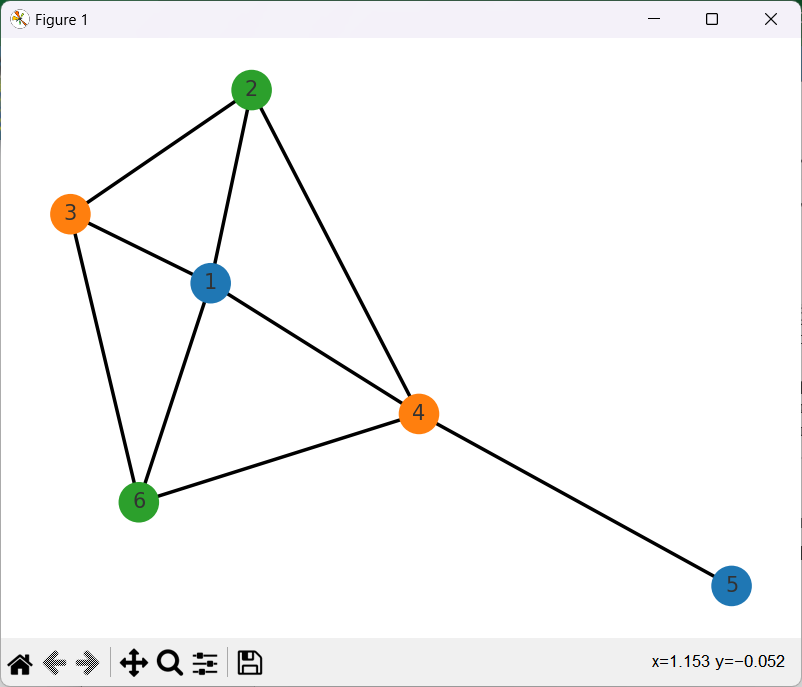
[5, 4, 6, 1, 3, 2]

, 6: [1, 6], 5: [1, 4, 5]}, 2: {2: [2], 1: [2, 1], 3: [2, 3], 4: [2, 4], 5: [2, 4, 5], 6: [2, 4, 6]}, 3: {3: [3], 1: [3, 1], 2: [3, 2], 6: [3, 6], 4: [3, 6, 4], 5: [3, 6, 4, 5]}, 4: {4: [4], 1: [4, 1], 2: [4, 2], 5: [4, 5], 6: [4, 6], 3: [4, 6, 3]}, 6: {6: [6], 1: [6, 1], 3: [6, 3], 4: [6, 4], 2: [6, 4, 2], 5: [6, 4, 5]}, 5: {5: [5], 4: [5, 4], 1: [5, 4, 1], 2: [5, 4, 2], 6: [5, 4, 6], 3: [5, 4, 6, 3]}, 'weight': 'weight'}

Алгоритм Флойда: {1: {1: 0, 2: 6, 3: 8, 4: 3, 6: 3, 5: 6}, 2: {2: 0, 1: 6, 3: 8, 4: 4, 6: 8, 5: 7}, 3: {3: 0, 1: 8, 2: 8, 6: 6, 4: 10, 5: 13}, 4: {4: 0, 1: 3, 2: 4, 5: 3, 6: 4, 3: 10}, 6: {6: 0, 1: 3, 3: 6, 4: 4, 2: 8, 5: 7}, 5: {5: 0, 4: 3, 1: 6, 2: 7, 3: 13, 6: 7}}

Найкоротша відстань є найдовшою серед всіх найкоротших відстаней: (3, 5), 13.

Найкоротша відстань є найкоротшою серед всіх найкоротших відстаней: (1, 4), 3.



Теоретичні відомості: elearn, [Print all Hamiltonian paths present in a graph | Techie Delight](https://www.techiedelight.com/print-all-hamiltonian-path-present-in-a-graph/),wiki, [Software for Complex Networks — NetworkX 3.3 documentation](https://networkx.org/documentation/stable/index.html)

**Код програм**

import matplotlib.colors as mpl

import matplotlib.pyplot as plt

import networkx as nx

from numpy import inf

G = nx.MultiGraph()

f = open("input.txt", "r")

for line in f:

node1 = int(line.split(" ")[0])

node2 = int(line.split(" ")[1])

weight = int(line.split(" ")[2])

G.add\_edge(node1, node2, weight=weight)

f.close()

# 3. Вивести визначення та значення основних характеристик графа (матриця суміжності, інциндентності, хроматичне число та реберне хроматичне число та інші)

print("\nMатриця суміжності:\n", nx.adjacency\_matrix(G))

print("\nMатриця інциндентносi:\n", nx.incidence\_matrix(G))

graph\_coloring = nx.coloring.greedy\_color(G)

unique\_colors = set(graph\_coloring.values())

max\_degree = max(G.degree, key=lambda x: x[1])[1]

edge\_chromatic\_number = max\_degree + 1

print(f"\nХроматичне число: {len(unique\_colors)}")

print(f"Реберне хроматичне число: {edge\_chromatic\_number}")

# 4. Написати програму для знаходження всіх ейлерових ланцюгів та циклів

if nx.is\_eulerian(G):

eulerian\_circuit\_edges = list(nx.eulerian\_circuit(G))

print("Ребра ейлерового циклу:", eulerian\_circuit\_edges)

else:

print("\nГраф не є ейлеровим")

H = nx.eulerize(G)

eulerian\_circuit\_edges = list(nx.eulerian\_circuit(H))

print("Ребра ейлерового циклу:", eulerian\_circuit\_edges)

if nx.has\_eulerian\_path(G):

eulerian\_path\_edges = list(nx.eulerian\_path(H))

print("Ребра ейлерового шляху:", eulerian\_path\_edges, "\n")

else:

print("Граф не має ейлерового шляху.")

H = nx.eulerize(G)

print("Оригінальний граф:")

print(nx.edges(G))

print("\nНовий ейлерів граф:")

print(nx.edges(H))

eulerian\_path\_edges = list(nx.eulerian\_path(H))

print("Ребра ейлерового шляху:", eulerian\_path\_edges, "\n")

# 5. Написати програму для знаходження всіх гамільтонових ланцюгів та циклів

def hamiltonianPaths(G, v, visited, path):

# if all the vertices are visited, then the Hamiltonian path exists

if len(path) == len(G.nodes):

if path[0] in G.neighbors(path[-1]):

print("Гамільтоновий цикл:", path + [path[0]])

print("Граф є гамільтоновим")

print(path)

return

# Check if every edge starting from vertex `v` leads to a solution or not

for w in G.neighbors(v):

if not visited[w]:

visited[w] = True

path.append(w)

hamiltonianPaths(G, w, visited, path)

visited[w] = False

path.pop()

def findHamiltonianPaths(G):

print("\nГамільтоновиі ланцюги:")

for start in G.nodes:

path = [start]

visited = {node: False for node in G.nodes}

visited[start] = True

hamiltonianPaths(G, start, visited, path)

n = G.number\_of\_nodes()

findHamiltonianPaths(G)

# 6. Реалізувати програмно обхід графа пошуком углиб

dfs\_edges = list(nx.dfs\_edges(G, source=list(G.nodes)[0]))

print("\nОбхід графа пошуком углиб:", dfs\_edges)

# 7. Реалізувати програмно обхід графа пошуком вшир

dfs\_edges = list(nx.bfs\_edges(G, source=list(G.nodes)[0]))

print("\nОбхід графа пошуком вшир:", dfs\_edges)

# 8. На мові програмування реалізувани пошук найкородших відстаней на графі від заданої вершини (алгоритм Дейкстри). Вивести схему маршрутів і довжину відстаней.

path = dict(nx.all\_pairs\_dijkstra\_path(G), weight="weight")

print("\nАлгоритм Дейкстри:", path)

# 9. На мові програмування реалізувани пошук найкородших відстаней між будь-якими двома вершинами графу (алгоритм Флойда). Вивести схему маршрутів і довжину відстаней.

fw = nx.floyd\_warshall(G, weight="weight")

results = {a: dict(b) for a, b in fw.items()}

print("\nАлгоритм Флойда:", results)

# 10. Знайти між якою парою вершин найкоротша відстань є найдовшою серед всіх найкоротших відстаней.

# results

max\_distance = 0

vertices = ()

for start\_node, connections in results.items():

for end\_node, distance in connections.items():

if distance > max\_distance and distance != inf:

max\_distance = distance

vertices = (start\_node, end\_node)

print(

f"\nНайкоротша відстань є найдовшою серед всіх найкоротших відстаней: {vertices}, {max\_distance}."

)

# 11. Знайти між якою парою вершин найкоротша відстань є найкоротшою серед всіх найкоротших відстаней.

min\_distance = max\_distance

for start\_node, connections in results.items():

for end\_node, distance in connections.items():

if distance < min\_distance and start\_node != end\_node:

min\_distance = distance

vertices = (start\_node, end\_node)

print(

f"\nНайкоротша відстань є найкоротшою серед всіх найкоротших відстаней: {vertices}, {min\_distance}."

)

graph\_color\_to\_mpl\_color = dict(zip(unique\_colors, mpl.TABLEAU\_COLORS))

node\_colors = [graph\_color\_to\_mpl\_color[graph\_coloring[n]] for n in G.nodes()]

pos = nx.spring\_layout(G, seed=14)

nx.draw(

G,

pos,

with\_labels=True,

node\_size=500,

node\_color=node\_colors,

font\_size=12,

font\_color="#333333",

width=2,

)

plt.show()