Teoria Grafów - projekt zaliczeniowy

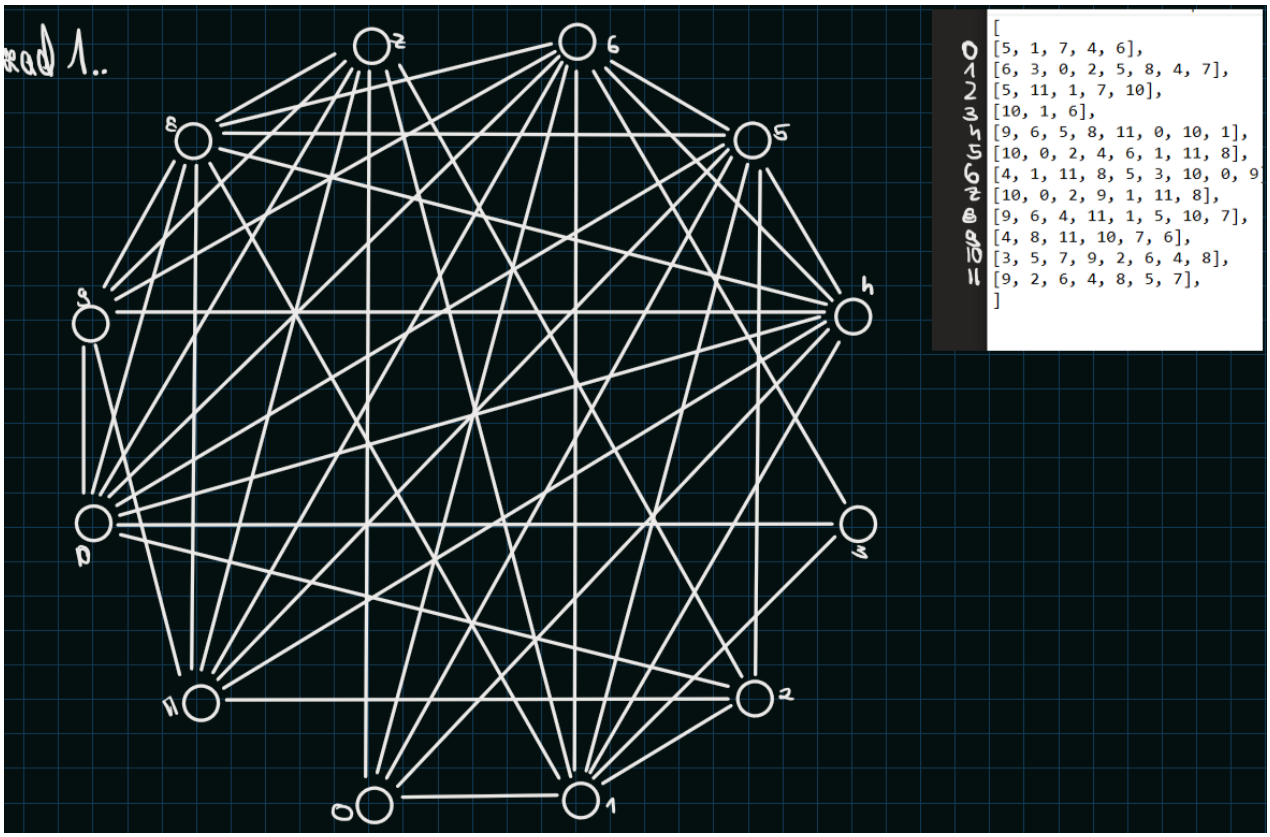
Zadania dla: Michał Pałucki

Część analityczna

W załączniku, w pliku Michał_Pałucki.json znajduje się lista sąsiedztwa dla grafu do przeanalizowania. Zadania w części analitycznej (1-8) mają zostać wykonane w oparciu o ten właśnie graf. Zadania mogą być rozwiązane na kartce i zeskanowane lub wykonane w dowolnym programie, np. OneNote. Proszę o wyniki w formie pliku pdf.

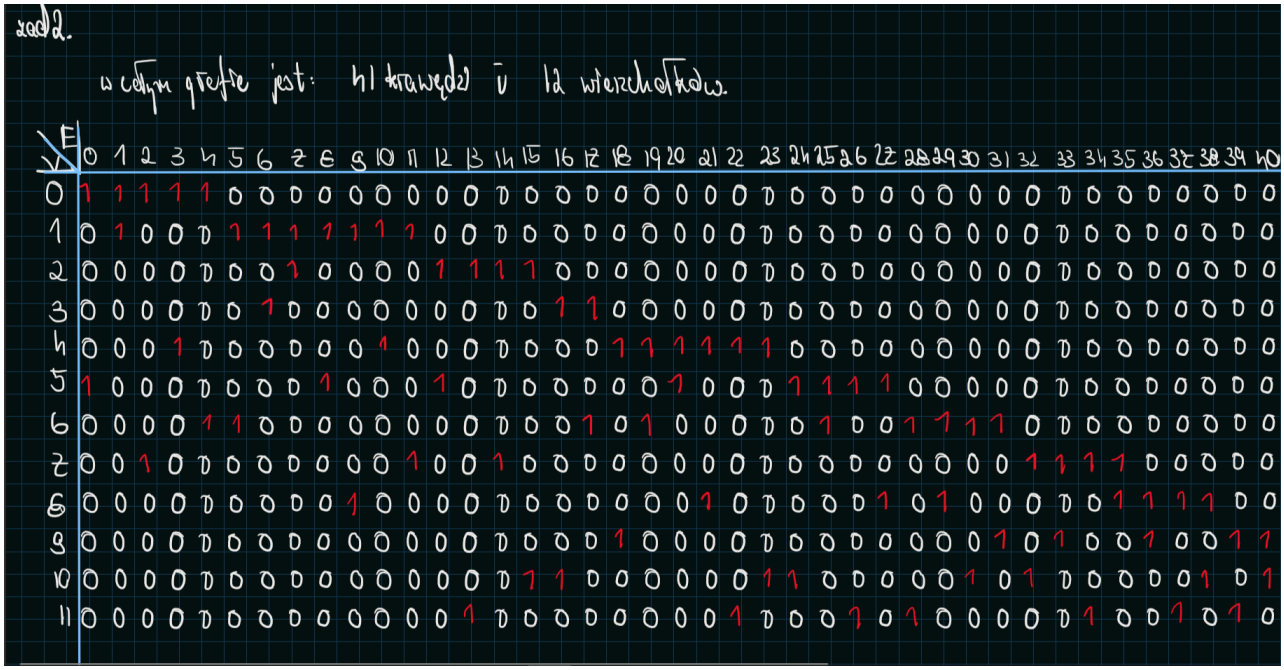
Zadanie 1 (1pkt)

Wykonaj szkic grafu.



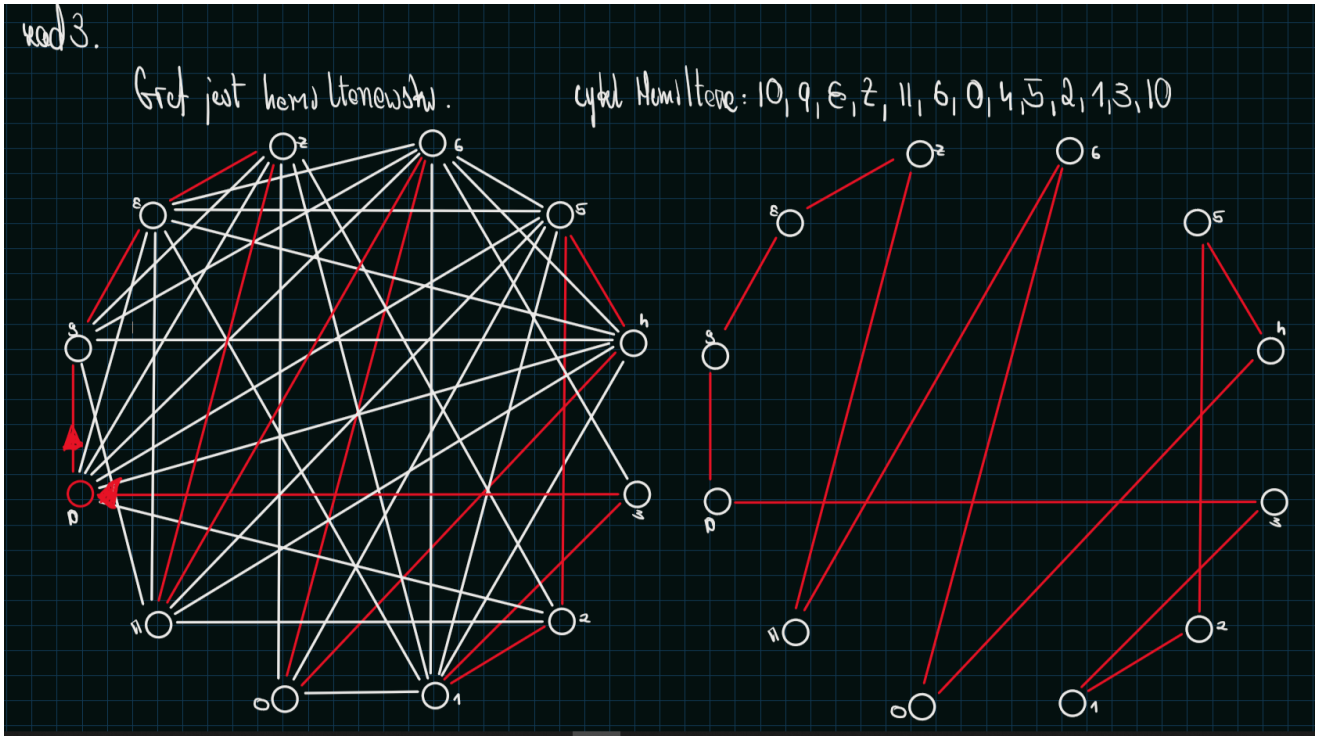
Zadanie 2 (1pkt)

Opisz graf w formie macierzy incydencji.



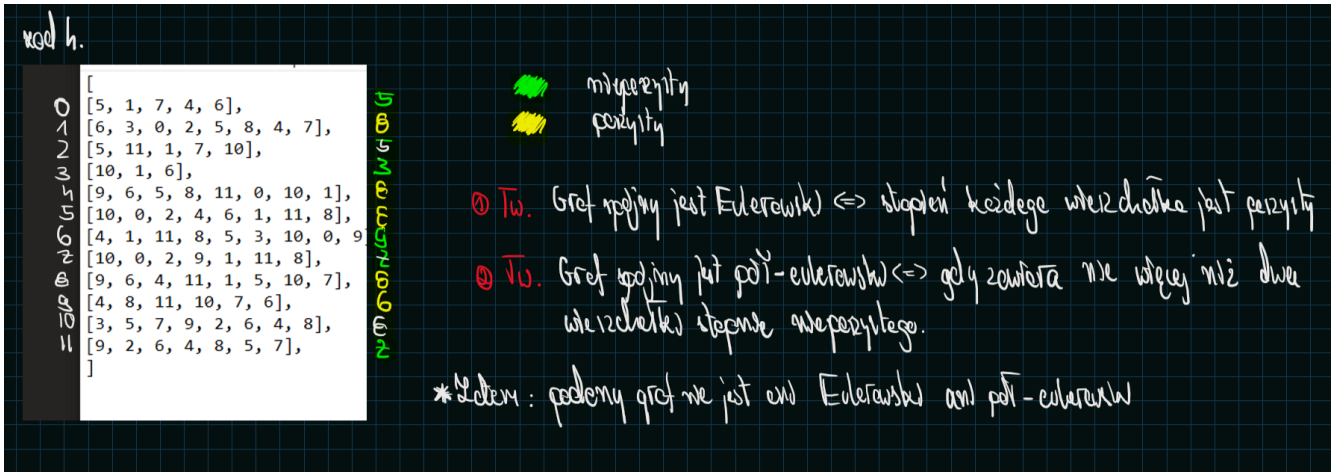
Zadanie 3 (3pkt)

Czy ten graf jest hamiltonowski/pół-hamiltonowski? Jeśli tak to podaj ścieżkę/cykl Hamiltona.



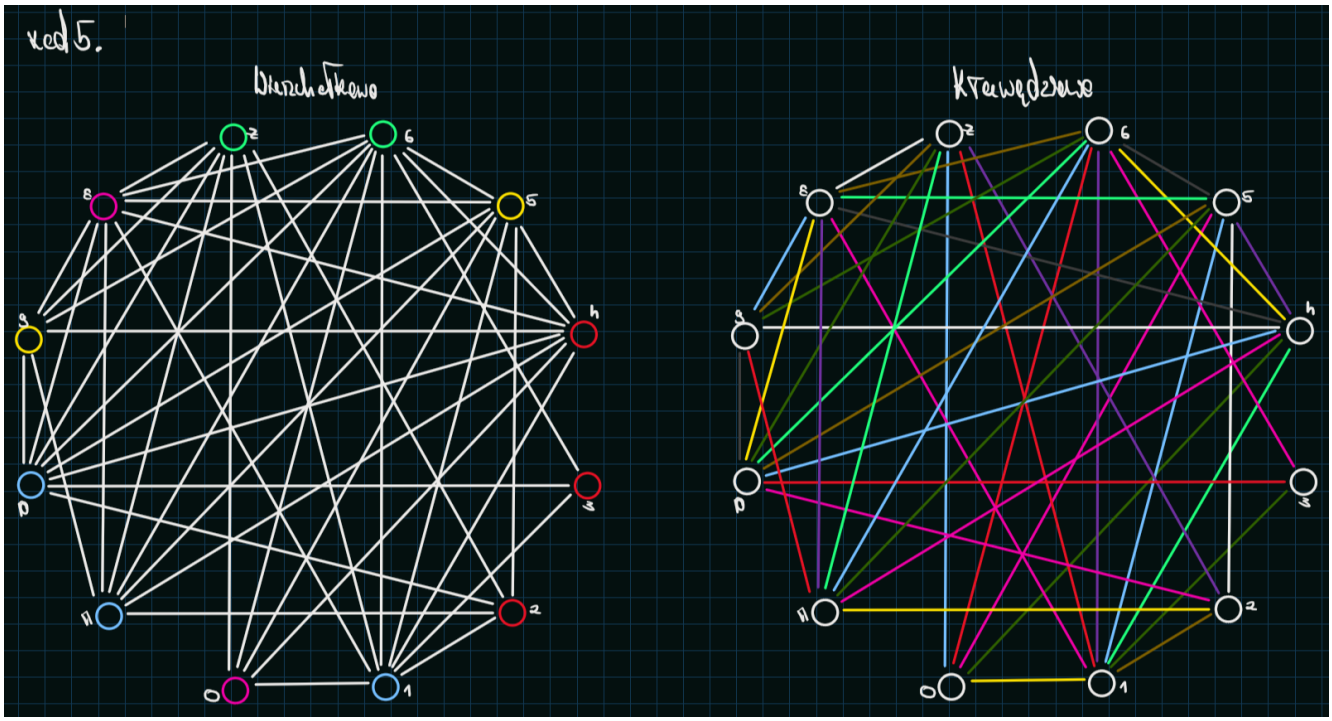
Zadanie 4 (3pkt)

Czy ten graf jest eulerowski/pół-eulerowski? Jeśli tak to podaj ścieżkę/cykl Eulera.



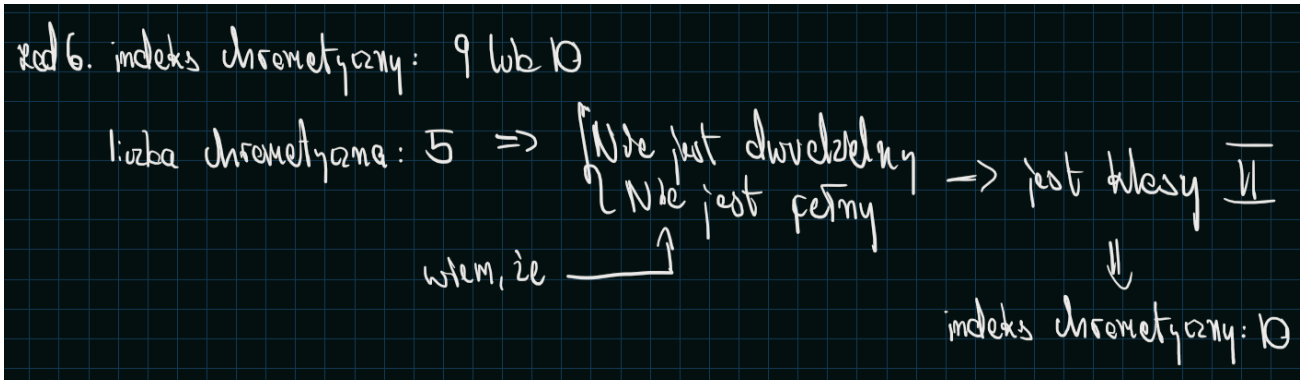
Zadanie 5 (2pkt)

Pokoloruj graf wierzchołkowo oraz krawędziowo.



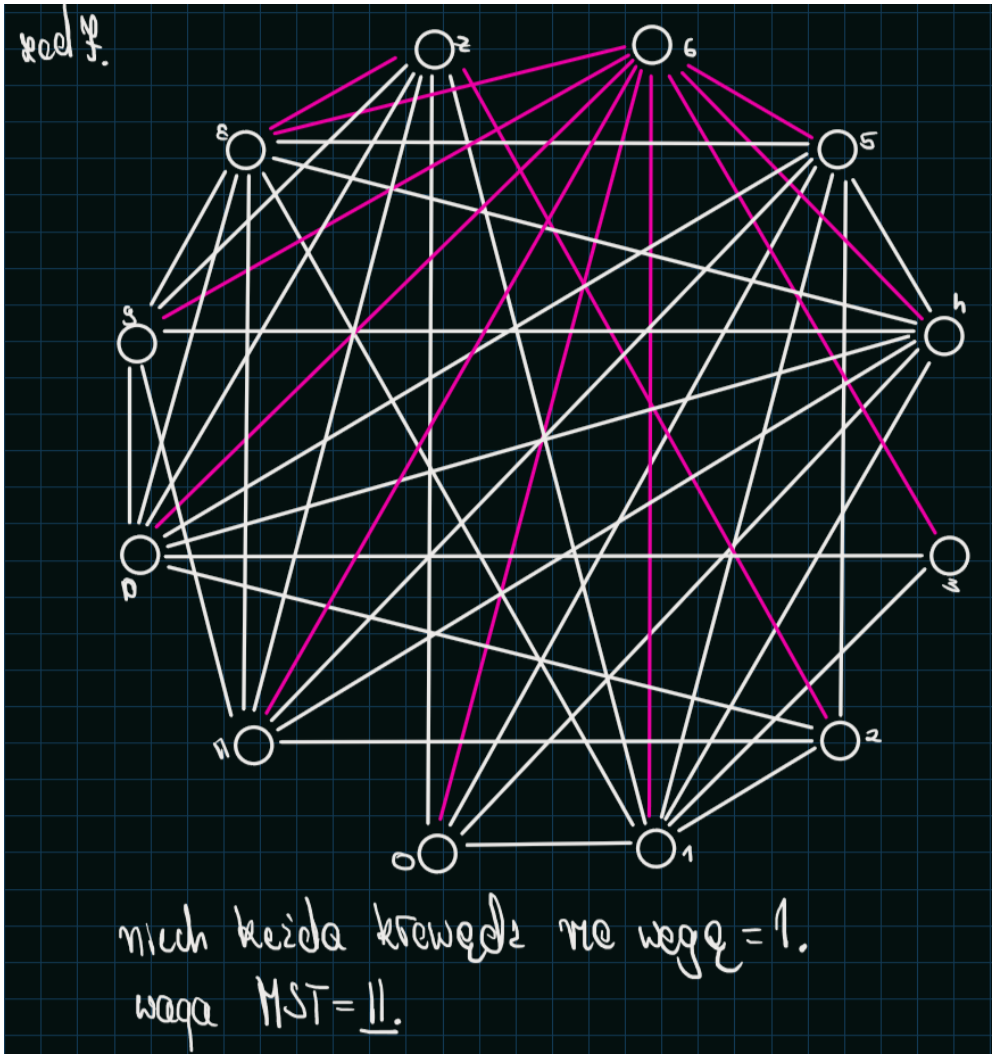
Zadanie 6 (1pkt)

Podaj liczbę chromatyczną oraz indeks chromatyczny dla grafu.



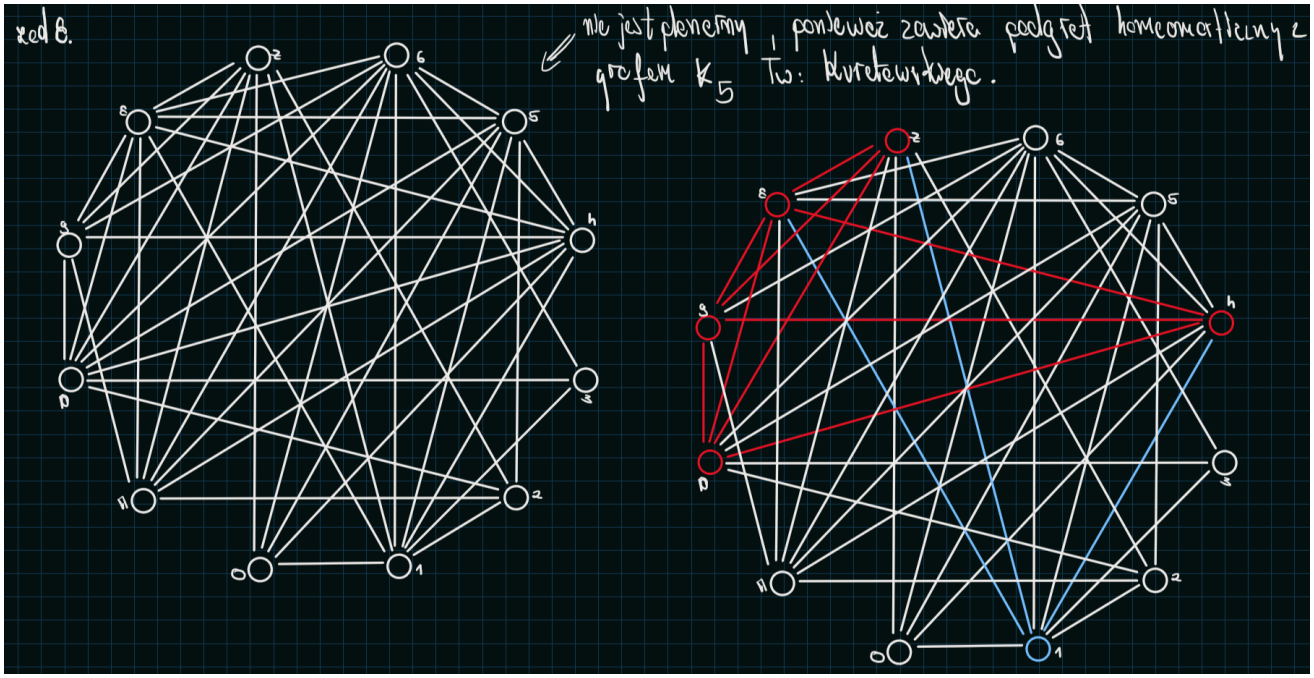
Zadanie 7 (1pkt)

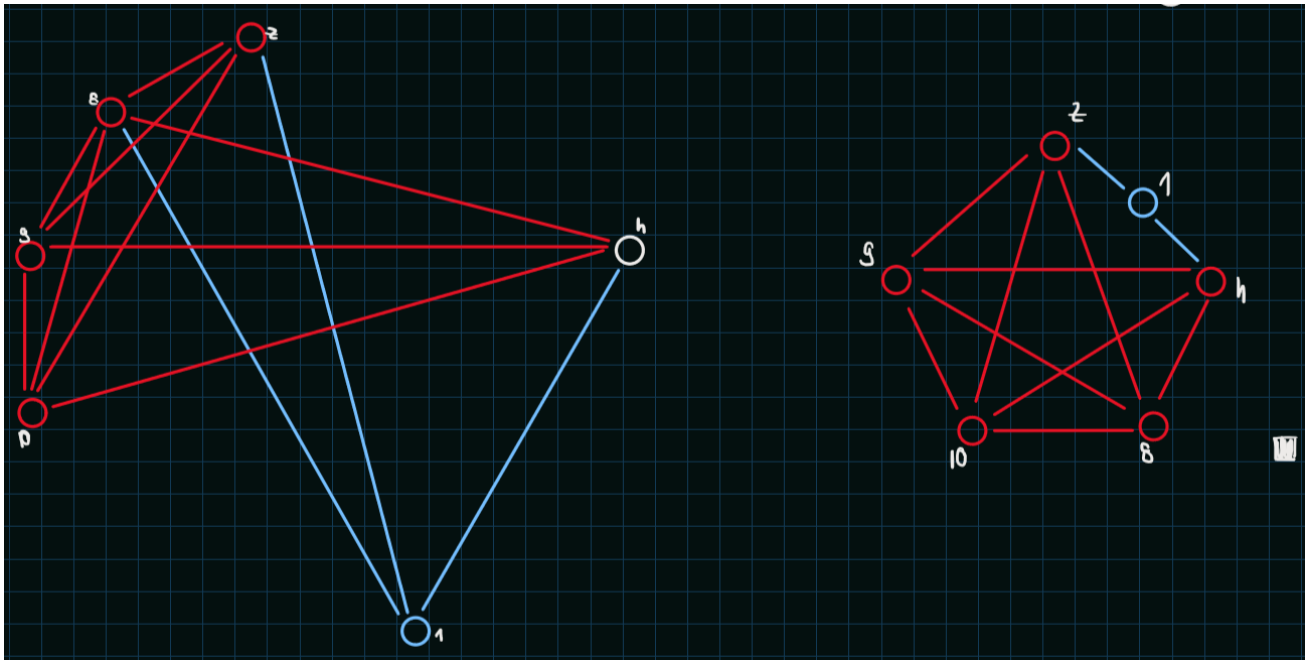
Wyznacz minimalne drzewo rozpinające dla analizowanego grafu.



Zadanie 8 (2pkt)

Czy rysunek tego grafu jest planarny? Jeśli nie, to czy da się go przedstawić jako planarny? Jeśli tak, to ile ścian można w nim wyznaczyć? Proszę to wykazać na rysunku





Część programistyczna

Zaimplementuj poniższy algorytm w wybranym języku.

Algorytm może zostać zaimplementowany w wybranym języku - Java, Kotlin, C, C++, Python, JS, TS, C#. Implementację proszę dostarczyć w formie linku do repozytorium (GitHub, GitLab - preferowane) lub archiwum zip. Program ma wczytywać graf z pliku (lista sąsiedztwa bądź macierz incydencji), a następnie uruchomić zaimplementowany algorytm na tym grafie. W repozytorium musi znajdować się instrukcja uruchomienia projektu.

Zaimplementuj algorytm Bellmana-Forda (10pkt)-Instrukcja

- W "src/Bellman_Ford_algorithm/" znajdują się:
    1. Plik: main.py (do otwarcia i uruchomienia za pomocą np.: Pycharm)
    2. Przykładowe pliki testowe: example_data_1.json
example_data_2.json
example_data_3.json
example_data_4.json
    3. Przykładowy output programu:
output_graph_example_data_1.png
output_graph_path_example_data_1.png
output_info_example_data_1.json
output_info_example_data_1.txt
- Przed uruchomieniem należy zaimportować:

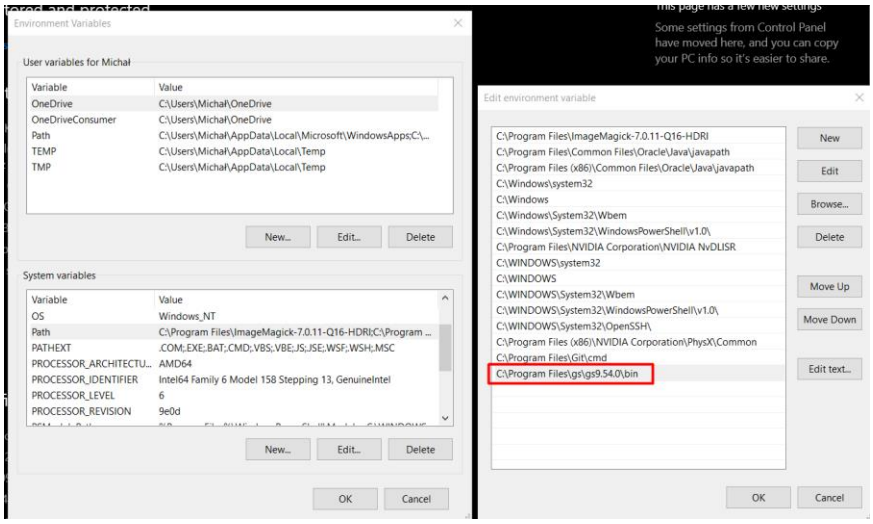
```
import numpy as np
import json
import math
import turtle
import os
import glob
from copy import deepcopy
from PIL import Image
```

- Po uruchomieniu - w wypadku następującego błędu, należy pobrać "ghostscripta" z poniższego linku:

```
Traceback (most recent call last):
  File "C:/Users/sawic/PycharmProjects/pythonProject/main.py", line 357, in <module>
    g.try_to_show_turtle_graph(file_name_=file_name)
  File "C:/Users/sawic/PycharmProjects/pythonProject/main.py", line 202, in try_to_show_turtle_graph
    pic.load(scale=10)
  File "C:\Users\sawic\PycharmProjects\pythonProject\venv\lib\site-packages\PIL\EpsImagePlugin.py", line 332, in load
    self.im = Ghostscript(self.tile, self.size, self.fp, scale)
  File "C:\Users\sawic\PycharmProjects\pythonProject\venv\lib\site-packages\PIL\EpsImagePlugin.py", line 134, in Ghostscript
    raise OSError("Unable to locate Ghostscript on paths")
OSError: Unable to locate Ghostscript on paths
```

https://www.ghostscript.com/download/gsdnld.html

- Pobranego i zainstalowanego “ghostscripta” należy dodać do „Path”.



- Prawidłowo uruchomiony program:
  1. Usunie wcześniejsze outputy.
  2. Zapyta o plik .json, zawierający listę sąsiedztwa z wagami np.:

[[[1,5],[8,2]],	V0-5->V1, V0-2->V8
[[2,7],[6,3]],	V1-7->V2, V1-3->V6
[[3,8],[5,5],[7,7]],	V2-8->V3, V2-5->V5, V2-7->V7
[[4,4]],	V3-4->V4
[],	
[[7,2]],	V5-2->V7
[[2,2],[5,6],[8,-2]],	V6-2->V2, V6-6->V5, V6-(-2)->V8
[[3,3]],	V7-3->V3
[[1,2],[5,9]]]	V8-2->V1, V8-5->V5

  3. Wyświetli animację rysowania grafu i w wypadku braku ujemnych cykli animację rysowania ścieżek.
  4. Wypisze:
     a) do pliku .txt podstawowe informacje o grafie
     b) do pliku .json podstawowe informacje o grafie
     c) stworzy rysunek .png grafu
     d) w wypadku braku ujemnych cykli stworzy rysunek .png ścieżek

Przeanalizuj powyższy algorytm: jakie problemy rozwiązuje, konkretne przykłady wykorzystania, z jakich metod korzysta się obecnie do rozwiązywania tych problemów (4pkt)

Algorytm Bellmana Forda służy do odnalezienia w spójnym grafie ważonym (wagi mogą być ujemne) najkrótszej ścieżki od wierzchołka startowego do wszystkich pozostałych wierzchołków.

Dla grafu liczącego *n* wierzchołków i *e* krawędzi złożoność pesymistyczna jest równa $O(en)$. Biorąc pod uwagę, że w przypadku braku krawędzi wielokrotnych liczba krawędzi jest zawsze mniejsza od n^2, można powiedzieć, że złożoność czasowa algorytmu to $O(n^3)$.

Algorytm Bellmana Forda wykorzystywany jest np. w routingu:

1. Każdy węzeł (router) oblicza odległości między sobą a wszystkimi innymi węzłami w systemie autonomicznym i przechowuje te informacje w tabeli.
2. Każdy węzeł wysyła swoją tabelę do wszystkich sąsiednich węzłów.
3. Gdy węzeł otrzymuje tabele odległości od swoich sąsiadów, oblicza najkrótsze trasy do wszystkich innych węzłów i aktualizuje własną tabelę, aby odzwierciedlić wszelkie zmiany.

Innymi algorytmami poszukującymi najkrótszych ścieżek są:

1. Algorytm Dijkstry
2. Algorytm Floyda-Warshalla
3. Algorytm Johnsona