

# Implementacja algorytmu Fleurego w Języku Python

## Spis treści

Wprowadzenie	2
Opis interfejsu	3
Implementacja	5
Podsumowanie	10
Literatura	10

Autor:  
Dawid Kulig  
dawid.kulig[at]uj.edu.pl

Wersja dokumentu:  
**0.1**

## 1. Wprowadzenie

**Algorytm Fleurego** - algorytm pozwalający na odszukanie cyklu Eulera w grafie eulerowskim. W czasie pracy korzysta on ze znajdowania mostów.

Przydatne terminy:

- **Cykl Eulera** - to taki cykl w grafie, który przechodzi przez każdą jego krawędź dokładnie raz. Jeżeli w danym grafie możliwe jest utworzenie takiego cyklu, to jest on nazywany grafem Eulerowskim. Cyklem nazywamy ścieżkę rozpoczynającą się i kończącą w tym samym wierzchołku grafu.
- **Stopień wierzchołka** - (ang. vertex degree) jest równy liczbie krawędzi sąsiadujących z wierzchołkiem. Jest on równy sumie liczb wszystkich łuków wchodzących, wychodzących, krawędzi i pętli; każdą pętlę liczy się jednak jak dwie krawędzie. Jest oczywiste, iż skoro cykl „wszedł” do wierzchołka jedną krawędzią, to musi on opuścić go drugą. Zatem krawędzie cyklu zawsze tworzą w wierzchołku grafu parę. Ponieważ wierzchołek może być odwiedzany kilkakrotnie po różnych krawędziach, to liczba krawędzi incydentalnych z tym wierzchołkiem zawsze musi być parzysta.
- **Stopień wejściowy** - (ang. in-degree) określa liczbę krawędzi wchodzących do wierzchołka, a stopień wyjściowy (ang. out-degree) określa liczbę krawędzi wychodzących. Równość jest wymagana z tego samego powodu, co powyżej – cykl przechodzi przez wierzchołek, zatem skoro jedną krawędzią wszedł, to drugą musi wyjść.
- **Most** - Mostem (ang. bridge) nazywamy krawędź grafu, której usunięcie zwiększa liczbę spójnych składowych

### **Schemat działania algorytmu:**

Wybieramy dowolny wierzchołek w grafie o niezerowym stopniu. Będzie to wierzchołek startowy cyklu Eulera. Następnie wybieramy krawędź, która nie jest mostem (przejście przez most oznacza brak

możliwości powrotu do tego wierzchołka, zatem jeśli zostały w nim nieodwiedzone krawędzie, to tych krawędzi już byśmy nie odwiedzili i cykl Eulera nie zostałby znaleziony), chyba że nie mamy innego wyboru, tzn. pozostała nam jedynie krawędź-most. Zapamiętujemy tę krawędź na liście lub na stosie. Przechodzimy wybraną krawędzią do kolejnego wierzchołka grafu. Przebytą krawędź usuwamy z grafu. W nowym wierzchołku całą procedurę powtarzamy, aż zostaną przebyte wszystkie krawędzie.

## 2. Opis interfejsu

Cały projekt implementacji algorytmu Fleurego (**Python**) został podzielony na osobne moduły:

- **moduł główny (main.py)** – jest odpowiedzialny za uruchomienie testów jednostkowych oraz przygotowanie środowiska do uruchomienia algorytmu Fleurego. W tymże module, importujemy zawartość moduły **Fleury** (algorytm) oraz modułu **tests**.

Testy uruchamiane są za pomocą wywołania funkcji **runTests()** pochodzącej z modułu **tests**.

Aby uruchomić algorytm Fleurego dla zadanego grafu, należy:

- 1) Stworzyć graf:

```
G = {0: [4, 5], 1: [2, 3, 4, 5], 2: [1, 3, 4, 5], 3: [1, 2], 4: [0, 1, 2, 5], 5: [0, 1, 2, 4]}
```

- 2) Utworzyć obiekt klasy **Fleury** oraz wywołać funkcję **Fleury.run()**:

```
test = Fleury(G)
test.run()
```

- **moduł algorytmu (Fleury.py)** – jest to moduł implementujący działanie algorytmu Fleurego. Zastosowane podejście

obiektywne (tworzenie klasy). Konstruktor klasy Fleury przyjmuje jako argument graf dla którego będzie uruchomiony algorytm. Moduł ten posiada drugą klasę (**FleuryException**) jest to klasa dziedzicząca po klasie Wyjątku. Wyjątek FleuryException jest rzucany wtedy, gdy podany graf nie jest Grafem Eulerowskim (nie spełnia podstawowego założenia działania algorytmu). Klasa Fleury implementuje następujące funkcje:

- **run()** – funkcja uruchamiająca działanie algorytmu.
  - **is\_connected()** – funkcja sprawdzająca czy podany graf jest połączony za pomocą algorytmu DFS ze stosem.
  - **even\_degree\_nodes()** – funkcja zwracająca listę parzystych krawędzi w grafie
  - **is\_eulerian()** – funkcja sprawdzająca czy dany graf jest grafem Eulerowskim
  - **convert\_graph()** – funkcja która spłaszcza strukturę grafu
  - **fleury()** – uruchomienie algorytmu (znajdowanie cyklu Eulera w podanym grafie)
- **moduł testów (tests.py)** - moduł implementujący testy jednostkowe dla klasy Fleury. Testowane są następujące funkcje:

even\_degree\_nodes(),      is\_eulerian(),      is\_connected(),  
convert\_graph()

### 3. Implementacja

#### main.py – moduł uruchamiający skrypt

```
#!/usr/bin/python
# -*- coding: iso-8859-2 -*-
#
# Fleury's Algorithm implementation
# Dawid Kulig
# dawid.kulig[at]uj.edu.pl

from Fleury import *
from tests import *

# Uruchomienie unit-testow
# runTests()

#G = {0: [2, 2, 3], 1: [2, 2, 3], 2: [0, 0, 1, 1, 3], 3: [0, 1, 2]}

#G = {0: [1, 4, 6, 8], 1: [0, 2, 3, 8], 2: [1, 3], 3: [1, 2, 4, 5],
4: [0, 3], 5: [3, 6], 6: [0, 5, 7, 8], 7: [6, 8], 8: [0, 1, 6, 7]}

#G = {1: [2, 3, 4, 4], 2: [1, 3, 3, 4], 3: [1, 2, 2, 4], 4: [1, 1,
2, 3]}

#G = {1: [2, 3], 2: [1, 3, 4], 3: [1, 2, 4], 4: [2, 3]}

G = {0: [4, 5], 1: [2, 3, 4, 5], 2: [1, 3, 4, 5], 3: [1, 2], 4: [0,
1, 2, 5], 5: [0, 1, 2, 4]}

test = Fleury(G)
test.run()
```

#### Fleury.py – moduł implementujący algorytm Fleurego

```
# -*- coding: iso-8859-2 -*-
#
# Fleury's Algorithm implementation
# Dawid Kulig
# dawid.kulig[at]uj.edu.pl

import copy

class FleuryException(Exception):
    def __init__(self, message):
        super(FleuryException, self).__init__(message)
        self.message = message

class Fleury:

    COLOR_WHITE = 'white'
```

```

COLOR_GRAY = 'gray'
COLOR_BLACK = 'black'

def __init__(self, graph):
    """
    Funckaj przypisująca graf
    :param graph:
    :return:
    """

    self.graph = graph

def run(self):
    """
    Funkcja uruchamiająca działanie algorytmu
    :return:
    """

    print '** Running Fleury algorithm for graph : ** \n'
    for v in self.graph:
        print v, ' => ', self.graph[v]
    print '\n'
    output = None
    try:
        output = self.fleury(self.graph)
    except FleuryException as (message):
        print message

    if output:
        print '** Found Eulerian Cycle : **\n'
        for v in output:
            print v
    print '\n** DONE **'

def is_connected(self, G):
    """
    Funkcja sprawdzająca czy podany graf jest polaczony
    za pomoca algorytmu DFS ze stosem
    :param G: GRAF
    :return: True / False
    """

    start_node = list(G)[0]
    color = {}
    iterator = 0;
    for v in G:
        color[v] = Fleury.COLOR_WHITE
    color[start_node] = Fleury.COLOR_GRAY
    S = [start_node]
    while len(S) != 0:
        u = S.pop()
        for v in G[u]:
            if color[v] == Fleury.COLOR_WHITE:
                color[v] = Fleury.COLOR_GRAY
                S.append(v)
            color[u] = Fleury.COLOR_BLACK
    return list(color.values()).count(Fleury.COLOR_BLACK) ==
len(G)

def even_degree_nodes(self, G):
    """

```

```

Funkcja, która liczy nieparzyste krawędzie w grafie
Returns: lista nieparzystych krawędzi w grafie
"""

even_degree_nodes = []
for u in G:
    if len(G[u]) % 2 == 0:
        even_degree_nodes.append(u)
return even_degree_nodes

def is_eulerian(self, even_degree_nodes, graph_len):
    """
    Sprawdzenie czy podany graf nieskierowany jest grafem
    Eulerskim
    Returns: true / false
    """

    return graph_len - len(even_degree_nodes) == 0

def convert_graph(self, G):
    """
    Funkcja, która zmienia strukturę grafu.
    Przykładowe dane wejściowe {0: [4, 5], 1: [2, 3, 4, 5]}
    Returns: [(0, 4), (0, 5), (1, 2), (1, 3), (1, 4), (1, 5)]
    """

    links = []
    for u in G:
        for v in G[u]:
            links.append((u, v))
    return links

def fleury(self, G):
    """
    Funkcja znajdująca cykl eulerski w podanym grafie
    Returns: lista krawędzi (cykl eulerski)
    """

    edn = self.even_degree_nodes(G)
    # sprawdzenie, czy graf jest grafem eulerskim
    if not self.is_eulerian(edn, len(G)):
        raise FleuryException('Podany graf nie jest grafem
Eulerskim!')
    g = copy.copy(G)
    cycle = []
    # wybieramy dowolny wierzchołek w grafie o niezerowym stopniu
    u = edn[0]
    while len(self.convert_graph(g)) > 0:
        current_vertex = u
        #for u in g[current_vertex]: # NIE DOBRE, BO ZMIENIA SIE W
PETLI
        for u in list(g[current_vertex]): # OSOBNĄ KOPIA
            g[current_vertex].remove(u)
            g[u].remove(current_vertex)
            # wybieramy krawędź, która nie jest mostem
            # (przejście przez most oznacza brak możliwości
powrotu
            # do tego wierzchołka

```

```

# zatem jeśli zostały w nim nieodwiedzone krawędzie,
# to tych krawędzi już byśmy nie odwiedzili
# i cykl Eulera nie zostałby znaleziony)
bridge = not self.is_connected(g)
if bridge:
    # nie ma innego wyboru (krawędź - most)
    # zapamiętujemy tę krawędź na liście lub na
stosie
    g[current_vertex].append(u)
    g[u].append(current_vertex)
else:
    break
if bridge:
    # przechodzimy wybraną krawędzią do kolejnego
wierzchołka grafu
    # przebytą krawędź usuwamy z grafu
    g[current_vertex].remove(u)
    g[u].remove(current_vertex)
    g.pop(current_vertex)
    cycle.append((current_vertex, u))
return cycle

```

## tests.py – mduł implementujący testy jednostkowe klasy Fleury

```

# -*- coding: iso-8859-2 -*-
#
# Fleury's Algorithm implementation
# Dawid Kulig
# dawid.kulig[at]uj.edu.pl

import unittest
from Fleury import *

class TestFleury(unittest.TestCase):

    def setUp(self):
        """
        Przygotowanie testów
        """

        G = {0: [4, 5], 1: [2, 3, 4, 5], 2: [1, 3, 4, 5], 3: [1, 2],
4: [0, 1, 2, 5], 5: [0, 1, 2, 4]}
        self.graph_a = G
        G = {0: [2, 2, 3], 1: [2, 2, 3], 2: [0, 0, 1, 3], 3: [0, 1,
2]}
        self.graph_b = G

    def testEven_degree_nodes(self):
        """
        Testowanie funkcji zwracającej liste krawedzi parzystych
        """

        fl_a = Fleury(self.graph_a)

```



```

        fl_b = Fleury(self.graph_b)

        list_a_expected = [0, 1, 2, 3, 4, 5]
        list_a_result = fl_a.even_degree_nodes(self.graph_a)

        list_b_expected = [2]
        list_b_result = fl_b.even_degree_nodes(self.graph_b)

        self.assertTrue(list_b_expected == list_b_result)
        self.assertTrue(list_a_expected == list_a_result)

    def testIs_eulerian(self):
        """
        Testowanie funkcji sprawdzającej czy graf jest EULEROWSKI
        """
        fl_a = Fleury(self.graph_a)
        fl_b = Fleury(self.graph_b)

self.assertTrue(fl_a.is_eulerian(fl_a.even_degree_nodes(self.graph_a)
, len(self.graph_a)))

self.assertFalse(fl_b.is_eulerian(fl_b.even_degree_nodes(self.graph_b)
), len(self.graph_b)))

    def testIs_connected(self):
        """
        Testowanie algorytmu DFS
        """

        fl_a = Fleury(self.graph_a)
        self.assertTrue(fl_a.is_connected(self.graph_b))

    def testConvert_graph(self):
        """
        Testowanie konwersji grafu na liste
        """

        fl_a = Fleury(self.graph_a)
        fl_b = Fleury(self.graph_b)

        converted_a_expected = [(0, 4), (0, 5), (1, 2), (1, 3), (1,
4), (1, 5), (2, 1), (2, 3), (2, 4), (2, 5), (3, 1), (3, 2), (4, 0),
(4, 1), (4, 2), (4, 5), (5, 0), (5, 1), (5, 2), (5, 4)]
        converted_a_result = fl_a.convert_graph(self.graph_a)

        converted_b_expected = [(0, 2), (0, 2), (0, 3), (1, 2), (1,
2), (1, 3), (2, 0), (2, 0), (2, 1), (2, 3), (3, 0), (3, 1), (3, 2)]
        converted_b_result = fl_b.convert_graph(self.graph_b)

        self.assertTrue(converted_a_expected == converted_a_result)
        self.assertTrue(converted_b_expected == converted_b_result)

    def runTests():
        """
        Uruchomienie testow
        """
        unittest.main()

```

## 4. Podsumowanie

Projekt został napisany w środowisku PyCharm (JetBrains).

<https://www.jetbrains.com/pycharm/>

Całość projektu - (pliki źródłowe oraz dokumentacja) są przechowywane w systemie kontroli wersji – **GitHub** pod adresem:

<https://github.com/dejvidk/fleury-algorithm>

## 5. Literatura

- [http://edu.i-lo.tarnow.pl/inf/alg/001\\_search/0135.php#P1](http://edu.i-lo.tarnow.pl/inf/alg/001_search/0135.php#P1)
- [http://pl.wikipedia.org/wiki/Algorytm\\_Fleury'ego](http://pl.wikipedia.org/wiki/Algorytm_Fleury'ego)
- <https://algizlo.wordpress.com/2013/06/11/algorym-fleuryego/>