## Implementacja algorytmu Fleurego w Języku Python

Autor:

Dawid Kulig

dawid.kulig[at]uj.edu.pl

Wersja dokumentu:

**0.1**

## Spis treści

Wprowadzenie 2

Opis interfejsu 3

Implementacja 4

Podsumowanie 5

Literatura 6

## 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.

## Opis interfejsu

## 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 uruchamiajaca 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 sprawdzajaca 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, ktora liczbe nieparzystych krawedzi w grafie  
 Returns: lista nieparzystych krawedzi 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\_odes, graph\_len):  
 *"""  
 Sprawdzenie czy podany graf nieskierowany jest grafem Eulerowskim  
 Returns: true / false  
 """* **return** graph\_len - len(even\_degree\_odes) == 0  
  
  
 **def convert\_graph**(self, G):  
 *"""  
 Funkcja, ktora zmienia strukture grafu.  
 Przkladowe dane wejsciowe {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 znajdujaca cykl eulerowski w podanym grafie  
 Returns: lista krawedzi (cykl eulerowski)  
 """* edn = self.even\_degree\_nodes(G)  
 # sprawdzenie, czy graf jest grafem eulerowskim  
 **if not** self.is\_eulerian(edn, len(G)):  
 **raise** FleuryException('Podany graf nie jest grafem Eulerowskim!')  
 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]: # NIEDOBRE, BO ZMIENIA SIE W PETLI  
 **for** u **in** list(g[current\_vertex]): # OSOBNA 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 (krawedz - 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 testow  
 """* 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 zwracajacej 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 sprawdzajacej 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()

## Podsumowanie

## Literatura

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