POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

Instytut Informatyki

Wojciech Gawiński

Miłosz Dziurzyński

Rafał Sobański

Sprawozdanie końcowe z projektu zespołowego SI

Kolorowanie grafu typu vertex-magic total labeling (ang. VMTL problem)

10 czerwca 2020

I. Opis zadania

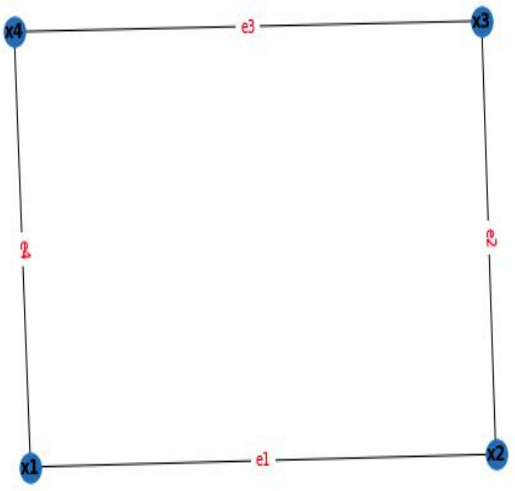
Link: [Projekt Github]

Napisać program sprowadzający problem *Vertex-Magic Total Labeling (ang. VMTL problem)* w równoważny problem SAT. Ten problem kolorowania grafu polega na znalezieniu stałej magicznej K przy poniższych założeniach:

1. Każdy wierzchołek posiada tylko jedną etykietę
2. Każda etykieta jest wykorzystana tylko raz
3. Dla każdego wierzchołka w grafie suma etykiet danego wierzchołka i krawędzi połączonych z tym wierzchołkiem równa się stałej K.


```
{
  "x1": [
    {"v": "x4", "l": "e4"},
    {"v": "x2", "l": "e1"}
  ],
  "x2": [
    {"v": "x3", "l": "e2"},
    {"v": "x1", "l": "e1"}
  ],
  "x3": [
    {"v": "x2", "l": "e2"},
    {"v": "x4", "l": "e3"}
  ],
  "x4": [
    {"v": "x3", "l": "e3"},
    {"v": "x1", "l": "e4"}
  ]
}
```


Rys. 1. Graf zapisany w standardzie Json.



Rys. 2. Reprezentacja grafu z rys. 1.

Dane wejściowe

Danymi wejściowymi jest plik json reprezentujący graf, np.:


```
{
  "x1": [
    {"v": "x5", "l": "e5"},
    {"v": "x2", "l": "e1"},
    {"v": "x4", "l": "e6"}
  ],
  "x2": [
    {"v": "x3", "l": "e2"},
    {"v": "x1", "l": "e1"},
    {"v": "x5", "l": "e7"}
  ],
  "x3": [
    {"v": "x2", "l": "e2"},
    {"v": "x4", "l": "e3"}
  ],
  "x4": [
    {"v": "x3", "l": "e3"},
    {"v": "x5", "l": "e4"},
    {"v": "x1", "l": "e6"}
  ],
   "x5": [
```

```
    {"v": "x1", "l": "e5"},
    {"v": "x4", "l": "e4"},
    {"v": "x2", "l": "e7"}
 ]
}
```

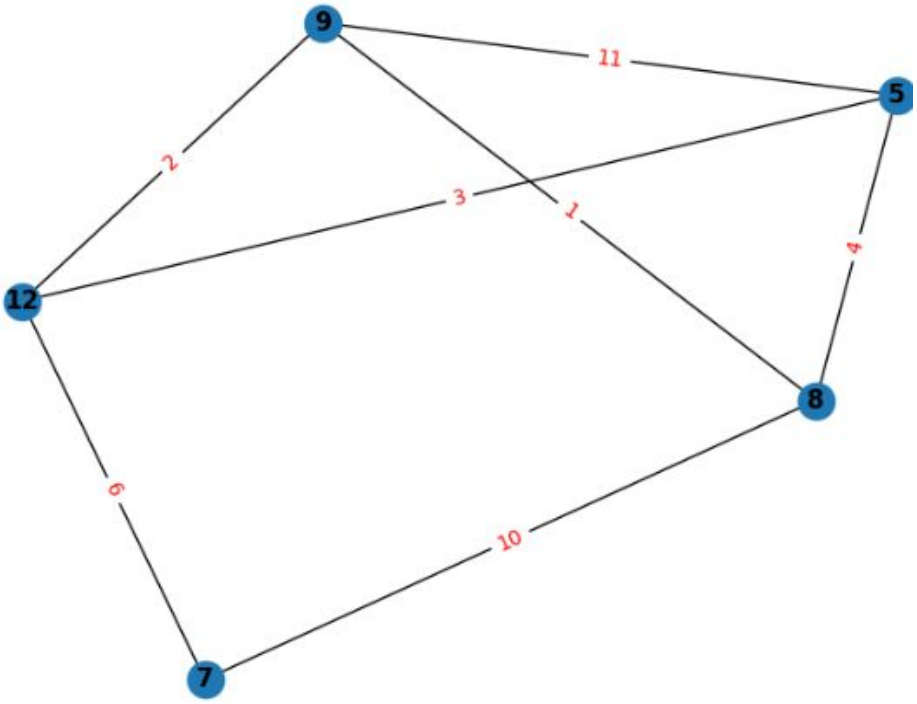
Gdzie *x1-x5* to wierzchołki, a *e1-e7* krawędzie.

Dane wyjściowe

Reprezentacja grafu w postaci pliku json wraz z wypisanymi etykietami:


```
{
  "5": [
    {"v": "9", "l": "11"},
    {"v": "8", "l": "4"},
    {"v": "12", "l": "3"}
 ],
 "8": [
    {"v": "7", "l": "10"},
    {"v": "5", "l": "4"},
    {"v": "9", "l": "1"}
 ],
 "7": [
    {"v": "8", "l": "10"},
    {"v": "12", "l": "6"}
 ],
 "12": [
    {"v": "7", "l": "6"},
    {"v": "9", "l": "2"},
    {"v": "5", "l": "3"}
 ],
  "9": [
    {"v": "5", "l": "11"},
    {"v": "12", "l": "2"},
    {"v": "8", "l": "1"}
 ]
}
```


Dodatkowo wyświetlany jest graf wynikowy:

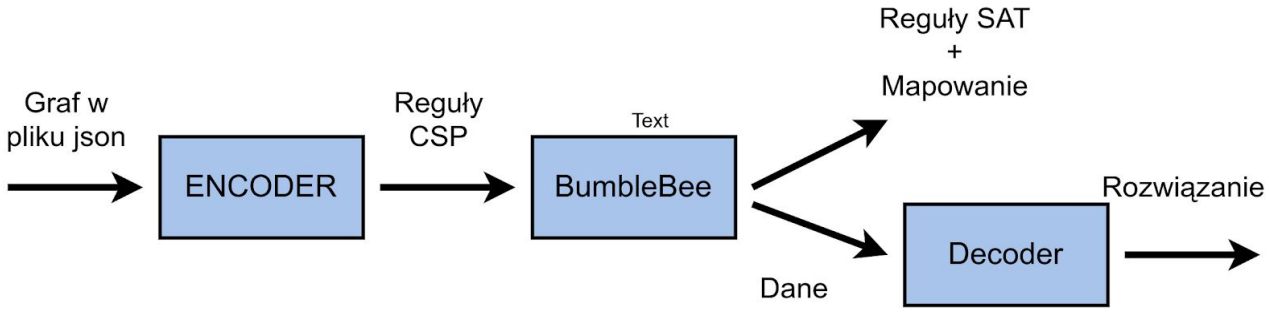


Rys. 3. Przykładowy wynikowy graf.

II. Założenia realizacyjne

1. W projekcie został wykorzystany program BumbleBee, który przekształca problem CSP w problem SAT i następnie przy pomocy wbudowanego SAT-Solvera rozwiązuje go.

2. Ogólny schemat działania programu:



Rys. 4. Schemat działania programu.

Algorytm kodowania problemu:

Dane: Odpowiedni plik json reprezentujący graf
Wyjście: Tekst w formacie json z rozwiązanym problemem VMTL
Metoda:

a) Wczytanie grafu i odpowiednie przekształcenie danych
b) Zakodowanie danych do jako reguł CSP w standardzie BumbleBee
c) Wczytanie wyniku bumbleeBee do programu
d) Wizualizacja uzyskanego wyniku

3. Do wykonania programu wykorzystaliśmy język Python w środowisku Colab oraz Pycharm (system Linux). Dodatkowo zostały wykorzystane takie narzędzia jak:
   - BumbleBee
   - NetworkX

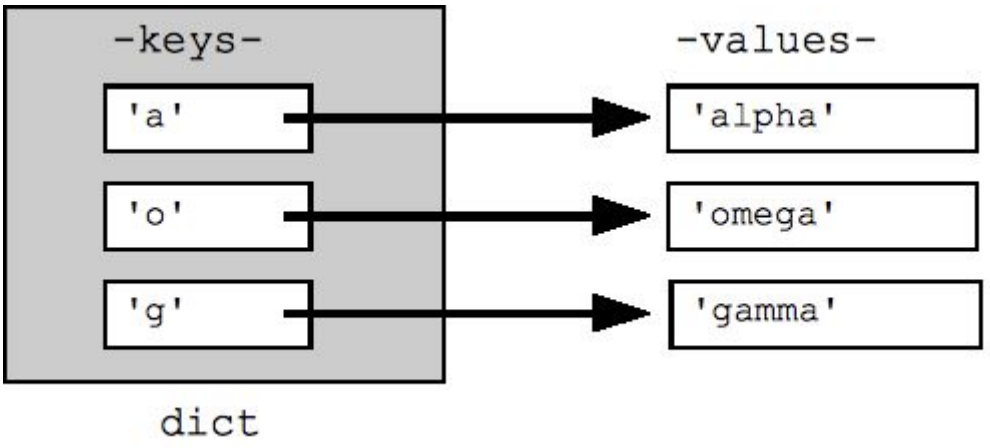
III. Podział prac

Autor	**Zadania**
Wojciech Gawiński	- Tworzenie algorytmów - Implementacja funkcji i metod - Dokumentacja
Miłosz Dziurzyński	- Wczytywanie grafu - Wyświetlanie grafu przy pomocy biblioteki NetworkX - Dokumentacja
Rafał Sobański	- Dokumentacja - Przygotowanie danych - Analiza i wczytanie otrzymanych danych

IV. Opis implementacji

1. Struktury danych

String - struktura danych reprezentująca ciąg znaków
Dictionary - struktura danych reprezentująca słownik. Można sobie wyobrazić tę strukturę jako tablicę par, gdzie pierwszą zmienną jest klucz, a drugą wartość.



2. Funkcje, procedury lub predykaty zdefiniowane w programie

   Klasa graph_helper:
   - get_all_labels(self) - metoda służąca do uzyskania wszystkich oznaczeń w grafie
   - vertex_and_edges(self) - metoda służąca do oddzielnego pozyskania wierzchołków i krawędzi
   - generate_csp_constraints(self) - metoda generująca stałe CSP
   - generate_graph(self, json_graph) - metoda generująca graf z pliku json
   - show_graph(self) - metoda pokazująca utworzony prędzej graf
   - get_result_string() - funkcja wczytująca plik wynikowy do programu
   - result_string_to_dict() - funcja zmieniajaca wczytany plik wynikowy w słownik

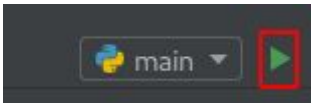
V. Użytkowanie i testowanie systemu

Aplikacja działa w taki sposób, że po jej uruchomieniu zostaje wykonana cała procedura działania, zwracany jest nam wynik i wizualizacja danych, a po zamknięciu okna cały program kończy działanie.

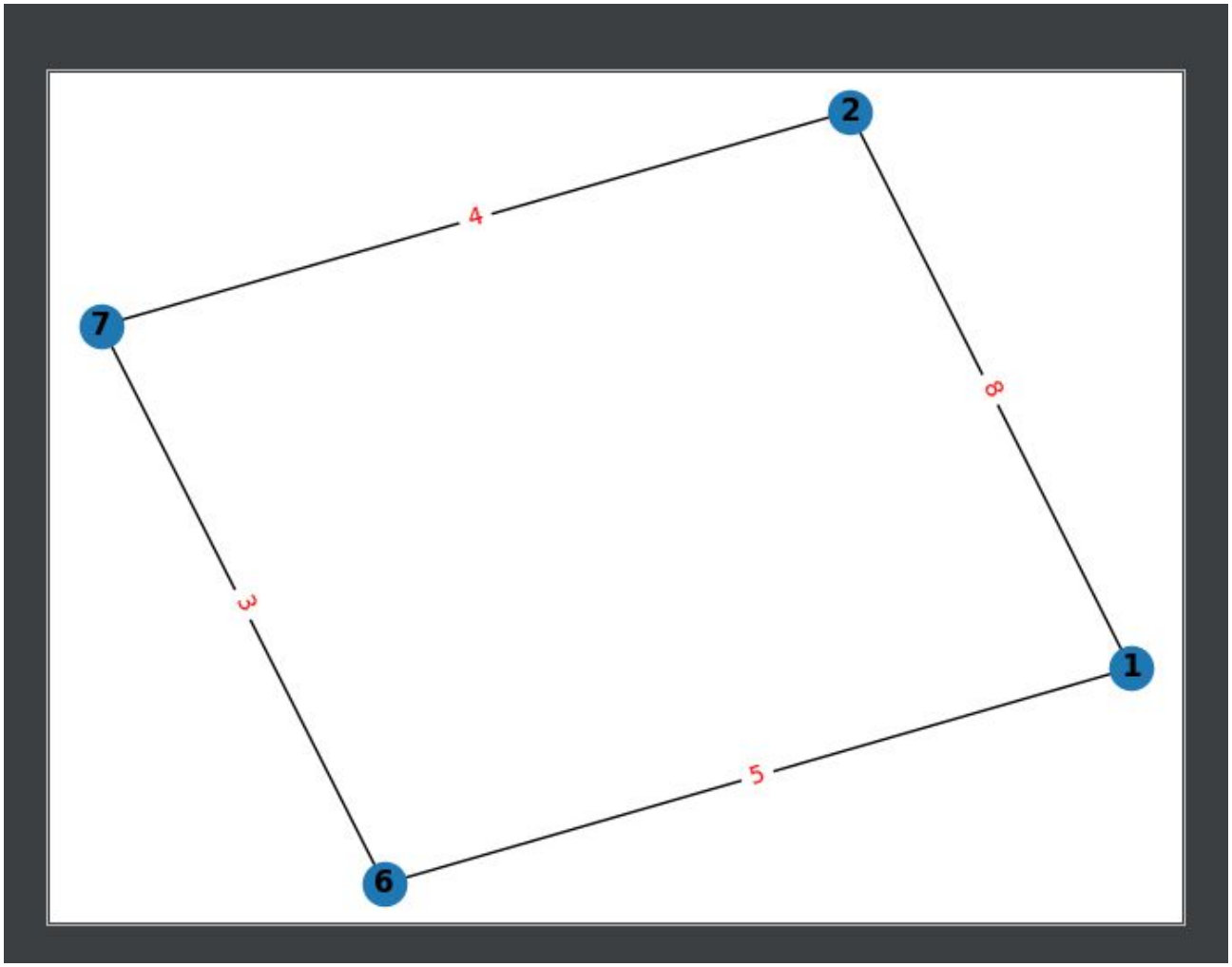
Program może zostać uruchomiony z konsoli poprzez wykonanie polecenia w folderze projektu bez argumentu lub z argumentem, który jest ścieżką do pliku z grafem

```
$ python main.py
```

```
$ python main.py "/home/sk0gen/Documents/Studia/Semestr 6/SI/SI_Proj/Projekt/Data/graph2.json"
```

lub poprzez IDE Pycharm 

Po uruchomieniu programu program wykonuje działania i zostaje nam wyświetlony wynikowy graf:



Dodatkowo utworzone zostają pliki:

- dimacs.cnf
- dimacs.map
- data/graph_result.json


```
{
 "1": [
   {"v": "2", "l": "8"},
   {"v": "6", "l": "5"}
 ],
 "6": [
   {"v": "7", "l": "3"},
   {"v": "1", "l": "5"}
 ],
 "7": [
   {"v": "6", "l": "3"},
   {"v": "2", "l": "4"}
 ],
 "2": [
   {"v": "7", "l": "4"},
   {"v": "1", "l": "8"}
 ]
}
```

- result.txt

```
% \"'/ //     BumbleBEE      / \_/ \_/ \
% -(|||)(')    (15/06/2017)    \_/ \_/ \_/
%  ^^^      by Amit Metodi    / \_/ \_/ \
%
% reading BEE file ... done
% load pl-satSolver ... % encoding BEE model ... done
% solving CNF (satisfy) ...
x1 = 1
e3 = 4
x2 = 6
e4 = 8
x3 = 7
x4 = 2
e1 = 5
e2 = 3
k = 14
```

VI. Tekst programu

main.py

```python
import os
from graph import graph_helper
import consts
import sys

constraints_filename = "Data/constraints.txt"
result_filename = "Data/result.txt"
graph_json_filename = ""
if len(sys.argv) > 1:
   graph_json_filename = sys.argv[1]
else:
   graph_json_filename = "Data/graph.json"
graph_json_result_filename = "Data/graph_result.json"

def get_result_string():
   to_return = None
   with open(result_filename, 'r') as result_file:
       to_return = result_file.read()
   to_return = to_return.replace(consts.to_delete, "")
   to_return = to_return.replace(consts.to_delete2, "")
   return to_return
```

```
def result_string_to_dict(result_string):
    temp = result_string.split('\n')
    temp = temp[:-1]
    dictionary = dict()
    for x in temp:
        key_value = x.split(' = ')
        dictionary[key_value[0]] = key_value[1]
    return dictionary


if __name__ == '__main__':
    graph_help = None
    with open(graph_json_filename) as graph:
        graph_help = graph_helper(graph)

    with open(constraints_filename, 'w') as result_file:
        result_file.write(graph_help.generated_csp)
    os.system(f'./BumbleBEE {constraints_filename} > {result_filename} ')
    os.system(f'./BumbleBEE {constraints_filename} -dimacs dimacs.cnf
dimacs.map')

    result = get_result_string()
    result_dictionary = result_string_to_dict(result)

    result_json = None
    with open(graph_json_filename) as graph:
        result_json = graph.read()
        for x in result_dictionary:
            result_json = result_json.replace(x, result_dictionary[x])

    with open (graph_json_result_filename,'w') as result_json_file:
        result_json_file.write(str(result_json))

    graph_help.generate_graph(result_json)
    graph_help.show_graph()
```

graph_helper.py

```
import networkx as nx
```

```
import json
from pprint import pprint
import matplotlib.pyplot as plt
from collections import defaultdict


class graph_helper:

    def __init__(self, graph_json):
        self.graph_json = json.load(graph_json)
        self.graph_labels, self.graph_labels_len = self.get_all_labels()
        self.vertex_edges = self.vertex_and_edges()
        self.generated_csp = self.generate_csp_constraints()
        self.graph, self.edge_labels = None, None

    def get_all_labels(self):
        graph_labels = set()
        for g_node in self.graph_json:
            g_vertex = self.graph_json[g_node]
            graph_labels.add(g_node)
            for g_sub in g_vertex:
                graph_labels.add(g_sub["l"])
        return graph_labels, len(graph_labels)

    def vertex_and_edges(self):
        vertex_edges = defaultdict(list)
        for g_node in self.graph_json:
            g_vertex = self.graph_json[g_node]
            for g_sub in g_vertex:
                vertex_edges[g_node].append(g_sub["l"])
        return vertex_edges

    def generate_csp_constraints(self):

        labels_len = len(self.graph_labels)
        generated_string = ''

        # new int labels
        temp = ''
        for label in self.graph_labels:
            generated_string = generated_string +
```

```
f"new_int({label},1,{labels_len})\n"
            temp = temp + f'{label},'
        # create magic label => k
        generated_string = generated_string +
f'int_array_allDiff([{temp[:-1]}])\n'

        generated_string = generated_string + f'new_int(k,1,{labels_len *
10})\n'

        for key in self.vertex_edges.keys():
            temp = f'{key},'
            for edge in self.vertex_edges[key]:
                temp = temp + f'{edge},'
            temp = temp[:-1]
            generated_string = generated_string +
f"int_array_plus([{temp}],k)\n"

        generated_string = generated_string + 'solve satisfy'
        return generated_string

    def generate_graph(self, json_graph):
        graph_json = json.loads(json_graph)
        graph = nx.Graph()
        edges_labels = dict()

        for g_node in graph_json:
            g_vertex = graph_json[g_node]
            for g_sub in g_vertex:
                graph.add_edge(g_node, g_sub["v"])
                g_vertex = (g_node, g_sub["v"])
                edges_labels[g_vertex] = g_sub["l"]
        self.graph = graph
        self.edge_labels = edges_labels

    def show_graph(self):
        pos = nx.spring_layout(self.graph)
        plt.figure()
        nx.draw(self.graph, pos, with_labels=True, font_weight='bold',
                labels={node: node for node in self.graph.nodes()})
        nx.draw_networkx_edge_labels(self.graph, pos,
edge_labels=self.edge_labels, font_color='red')
```

```
        plt.axis('off')
        plt.show()
```