УКРАЇНСЬКИЙ КАТОЛИЦЬКИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНИХ НАУК

Комп’ютерні науки

Graph Coloring Problem

Автори: Валявка Маркіян

Бек Андрій

31 травня 2020

1. Вступ.

Завдання програми - оприділити чи можна граф, заданий матрицею суміжності, розмалювати у чотири кольори. В нашій програмі, матриця задається у файлі matrix.txt . У модулі цей файл зчитується і перевіряється на симетричність. Якщо матриця симетрична - певна функція перевіряє чи можна задати кожній вершині унікальний колір, що відрізняється від кольору суміжних вершин, використовуючи **backtracking.** Якщо це можливо - на екран виводиться зображення розмальованого графа; В іншому випадку - користувачу виводиться повідомлення про те, що граф розмалювати не можливо.

Для реалізації цього алгоритму нам знадобилися знання про прості графи і графи в цілому, інформація про зв'язність графа, симетрію, і як оприділити ці властивості по матриці суміжності. Також для реаліції **backtrackin'**гу знадобилися знання про **DFS** - пошук вглиб**.**

2. Псевдокод.

1) Задається матриця суміжності у файлі matrix.txt

```
0,1,0,0,1,1,0,0,0,0
1,0,1,0,0,0,1,0,0,0
0,1,0,1,0,0,0,1,0,0
0,0,1,0,1,0,0,0,1,0
1,0,0,1,0,0,0,0,0,1
1,0,0,0,0,0,0,1,1,0
0,1,0,0,0,0,0,0,1,1
0,0,1,0,0,1,0,0,0,1
0,0,0,1,0,1,1,0,0,0
0,0,0,0,1,0,1,1,0,0
```

Приклад матриці здевятьма вершинами

2) Фунція read_from_file читає файл з матрицею і формує словник, який буде використовуватись в подальшому

```
mf = open("matrix.txt", "r")
for line in mf:
    line = line.strip()
    line_list = line.split(",")
    line_list = [int(elm) for elm in line_list]
    matrix.append(line_list)
    for vertice in range(1, len(line_list) + 1):
        if line_list[vertice - 1] == 1:
            if vertex in adj_vertices_dict.keys():
                adj_vertices_dict[vertex][0].append(vertice)
            else:
                adj_vertices_dict[vertex] = [[vertice]]
    vertex += 1
mf.close()
```

```python
if matrix_symetry_check(matrix) == True:
    return adj_vertices_dict
else:
    return None
```

Перед ретурном викликається функція matrix_symetry_check для перевірки на симетричність

3) Сформований словник передається в основну фунцію модуля - graph_color , яка “розмальовує” вершини, використовуючи backtracking.

```python
def graph_color(v: int, vDict: dict, depth=1, v_real=set(), color_c=False) -> dict:
    """
    (int,dict) -> vDict
    Main function that (using DFC path) tries to color each v in vDict
    using backtracking
    """
    while 1:
        if not color_c:
            sync(v, vDict)
        v_real.add(v)
        for c in range(vDict[v][1] + 1, 5):
            if safeCheck(c, v, vDict):
                vDict[v][1] = c
                break
        if vDict[v][1] == 5:
            vDict[v][1] = 0
            v_real.remove(v)
            return vDict, True
        if ((sum(vDict[v][-1]) == len(vDict[v][-1])) and v in v_real):
            return vDict, False
        for i, h in enumerate(vDict[v][-1]):
            if not h:
                vDict, color_c = graph_color(
                    vDict[v][0][i], vDict, depth=depth + 1)
                break
```

В свою чергу функція graph_color використовує safeCheck та sync

```
def safeCheck(c, v, vDict):
    """
    (int,int,dict) -> bool
    This function checks if it is safe to put color c in vertex v, comparing
    colors of adjacent vertices
    """
    for k, value in vDict.items():
        if k in vDict[v][0] and vDict[k][1] == c:
            return False
    return True
```

safeCheck перевіряє чи безпечно розмалювати вершину в певний колір не викликаючи *конфліктів і не порушивши умови задачі.*

```
def sync(s_vertex, vDict):
    """
    (int, dict) -> None
    Saves information in vDict whether s_vertex was
    visited before
    """
    for k, v in vDict.items():
        for indx, vertex in enumerate(v[0]):
            if vertex == s_vertex:
                v[-1][indx] = 1
```

Sync - функція необхідна *для пошуку з поверненням - зберігає інформацію про те, які вершини були відвідані*

4) Формується розмальований граф і виводиться на екран за допомогою бібліотек matplotlib та networkx

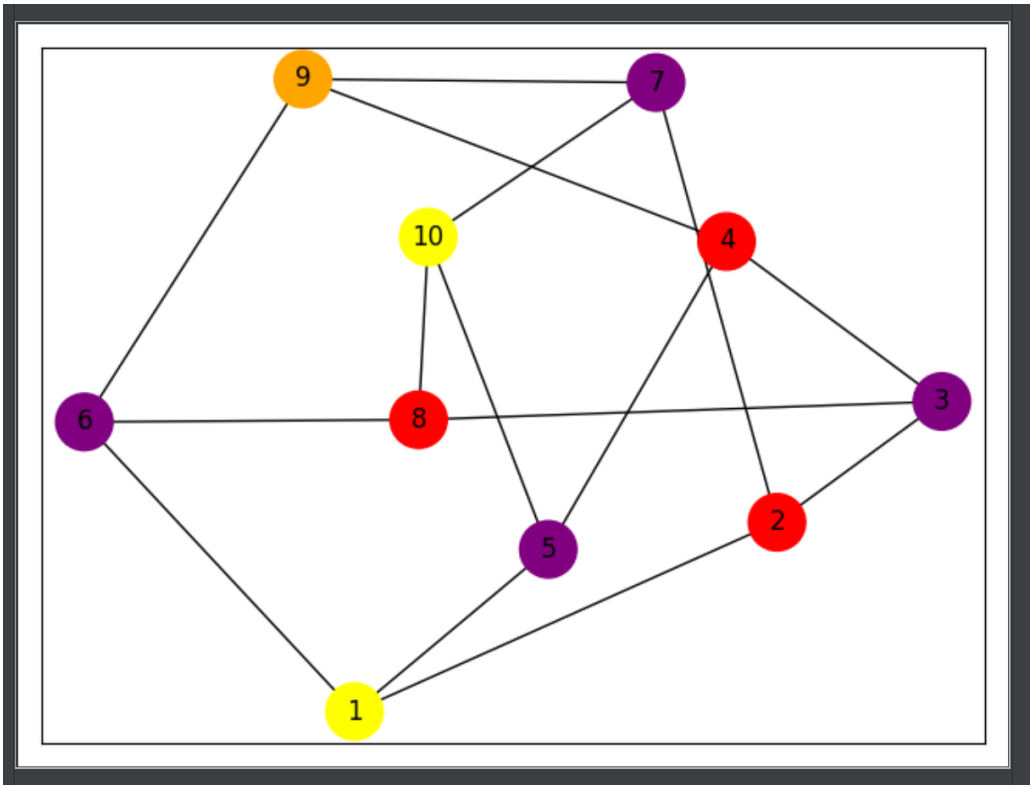
```
def color_graph(vertex_color_dict, adj_vertices_dict):
    """
    (dict,dict) -> None

    Function gets two dictionaries: 1) dictinary of vertices with colors they should be colored in
                                    2) dictionary of vertices adjacent to each other
    .. and draws a colored graph using matplotlib and networkx libraries.
    """

    colors_dict = {1:'r',2:'g',3:'b'}
    G = nx.Graph()

    for vertice,adjacent_vertices in adj_vertices_dict.items():
        G.add_node(vertice)
        G.nodes[vertice]['color'] = colors_dict[vertex_color_dict[vertice]]
        for i in adjacent_vertices:
            G.add_edge(vertice, i)

    color = [node[1]['color'] for node in G.nodes(data=True)]
    nx.draw_networkx(G, with_labels = True, node_color=color, node_size=700)
    plt.show()
```



Приклад розмальованого графа

3. Висновки**.**

Робимо висновок, що дуже важко оприділити чи можна розмалювати гр
у чотири кольори просто подивившись на матрицю суміжності або на сам граф.
Потрібно пройтися по кожній вершині і пробувати різні кольори. Для цього
найкраще використовувати пошук з поверненням (backtracking).