UNIFOR
ENSINANDO E APRENDENDO

Mestrado em Informática Aplicada
Igor Feijó dos Santos

Análise de Dados em Grafos

Informações preliminares:

Para responder a atividade proposta foi realizado consulta no livro Fundamento da Teoria dos Grafos para a Computação, 3º edição. Em consulta ao livro ficou claro que os problemas passados poderiam ser reduzidos a problema de coloração de vértices de um grafo.

O material supracitado trás, na página 212 as seguintes definições:
Seja G um grafo. Uma coloração de vértices de G atribui cores, geralmente notadas por 1, 2, 3, ..., aos vértices de G. Uma k-coloração de G é uma coloração que consiste em k diferentes cores: Neste caso G é chamado de k-colorível.

Seja G um grafo. O número mínimo n para o qual existe uma n-coloração do grafo G é chamado de índice cromático (ou número cromático) de G e é denotado por $\chi(G)$. Se $\chi(G) = k$, diz-se que G é k-cromático.

O problema da coloração de grafos:

Considere um grafo formado por **n** vértices e deseja-se pintar os vértices com uma cor de modo que vértices adjacentes (conectados por arestas) não tenham a mesma cor.

Não existe um bom algoritmo para resolver este problema, entretanto podemos resolver este problema seguindo os seguintes passos: (usaremos v_i e c_i para fazer referência a vértices e cores)

1. Não importando a ordenação dos vértices, atribua ao vértice v_1 a cor c_1
2. Atribua ao v_2 a cor c_1, caso ele não seja adjacente ao $v1$; caso seja adjacente atribua a cor $c2$
3. O vértice $v3$ receberá a cor $c1$ caso não seja adjacente ao $v1$, se for adjacente a $v1$ receberá a cor c_2 se não for adjacente a $v2$, se for adjacente a v_2 receberá a cor $c3$
4. Repita o processo sempre tentando atribuir a primeira cor ao novo vértice, caso não seja possível passe para a próxima cor e vá seguindo os passos até encontrar uma cor disponível.

Para implementar este algoritmo iremos precisar do conceito de matriz de adjacência.

A matriz de adjacência é um modo de representar grafos por meio de uma matriz.

	V1	V2	V3	...	Vn
V1					
V2					
V3					
⋮					
Vn					

Os elementos V_i representam os vértices do grafo e as linhas em branco indicam quantas arestas ligam os vértices, por exemplo, para o grafo.

	a	b	c	d
a	0	1	0	1
b	1	0	1	0
c	0	1	0	1
d	1	0	1	0

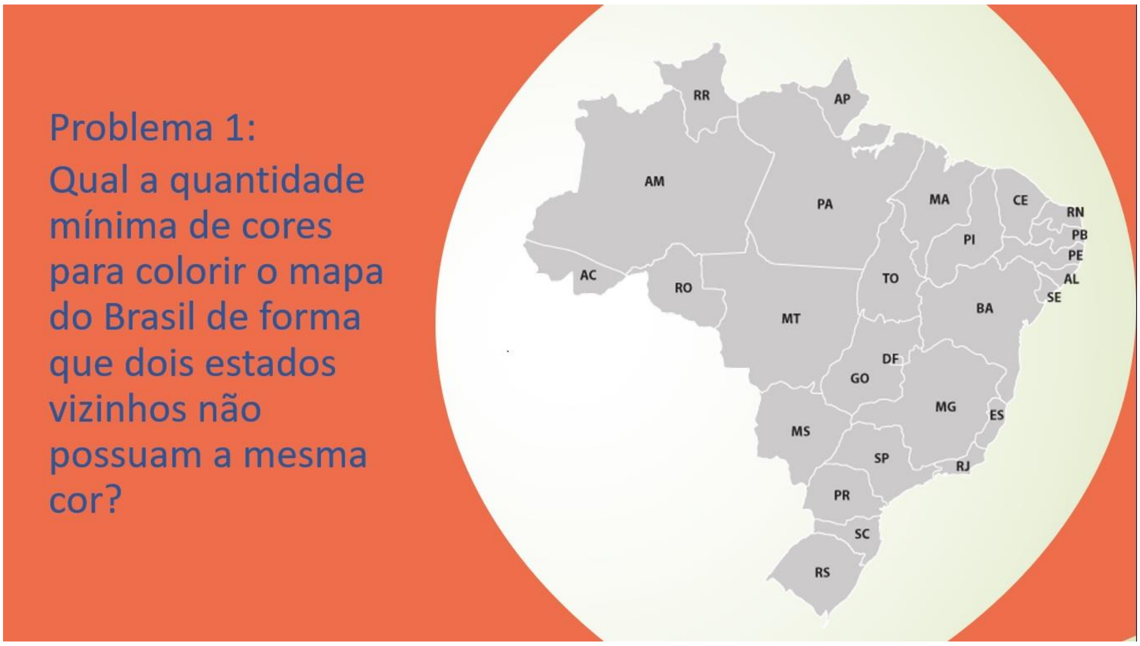
Sabemos que o vértice a está conectado com os vértices b e d.

Com a noção de matriz de adjacência para representar um grafo fica mais fácil implementar nosso algoritmo para coloração mínima de um grafo.

```
class Graph():
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)] \
                      for row in range(vertices)]

    # Verifica as cores
    def isSafe(self, v, colour, c):
        for i in range(self.V):
            if self.graph[v][i] == 1 and colour[i] == c:
                return False
        return True

    #colori
    def graphColourUtil(self, m, colour, v):
        if v == self.V:
            return True

        for c in range(1, m + 1):
            if self.isSafe(v, colour, c) == True:
                colour[v] = c
```

```
                if self.graphColourUtil(m, colour, v + 1) == True:
                    return True
                colour[v] = 0

    def graphColouring(self, m):
        colour = [0] * self.V
        if self.graphColourUtil(m, colour, 0) == None:
            return False

        # Print the solution
        print("A sequencia de cores é:")
        for c in colour:
            print(c)
        return True

def adicionar(u, v):
    g.graph[u - 1][v - 1] = 1
    g.graph[v - 1][u - 1] = 1

tamanho = int(input('Qual o número de vértices?'))
g = Graph(tamanho)
arestas = int(input('Quantas arestas tem no grafo?'))
for i in range(arestas):
    u = int(input('aresta com início em:'))
    v = int(input('aresta com fim em:'))
    adicionar(u,v)
m = 5 #define o número de cores (utilizei 5 me  baseando no teorema
que afirma que qualquer mapa pode ser pintado com 4 cores. Exagerei
pra 5)
g.graphColouring(m)
```

Questão 1:



Para resolver este problema iremos associar cada estado como sendo um vértice de um grafo e usaremos arestas para conectar vértices que que representam estados que fazem fronteira com algum estado.

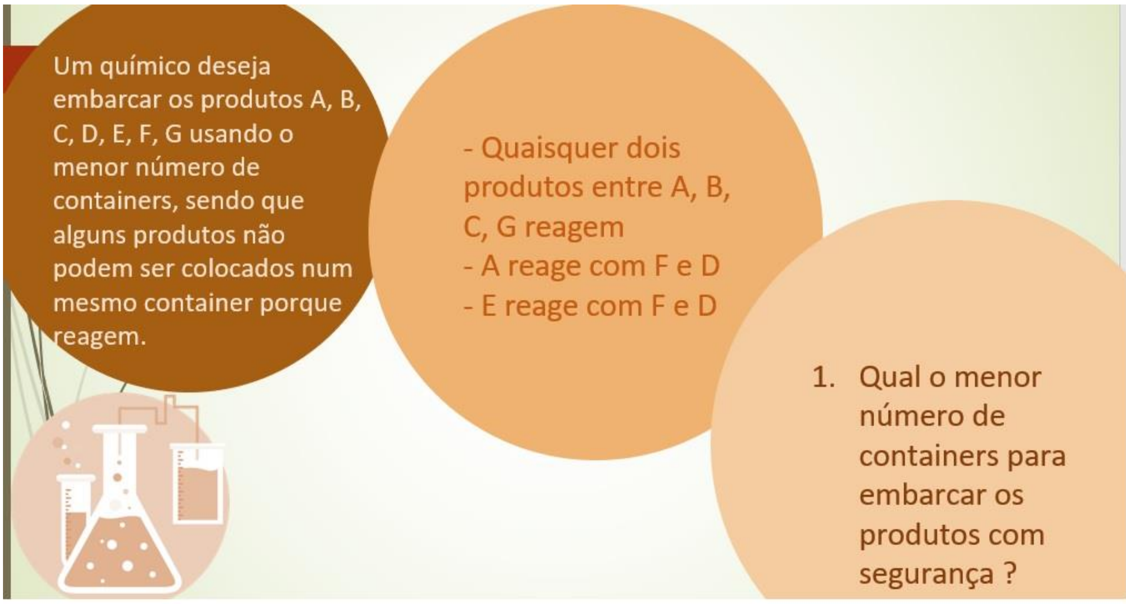
Feito isso, basta gerar o grafo do problema, em seguida gerar a matriz de adjacência do grafo e inserir os dados no algoritmo.

Nota: Não farei a matriz de adjacência para este problema, pois é muito demorado, entretanto o algoritmo fornecerá o número mínimo de cores e informará uma sequência de cores para ser utilizada.

O algoritmo retornar 4 cores e mostrará a sequência a ser pintada

Nota: Todos os demais problemas são resolvidos de igual modo, farei o próximo apenas para ilustrar a solução e os demais seguem igual.

Questão:



Solução:

Como nosso algoritmo foi implementado para tratar vértices como números, teremos que converter as letras para números. Deste modo, teremos:

A = 1

B = 2

C = 3

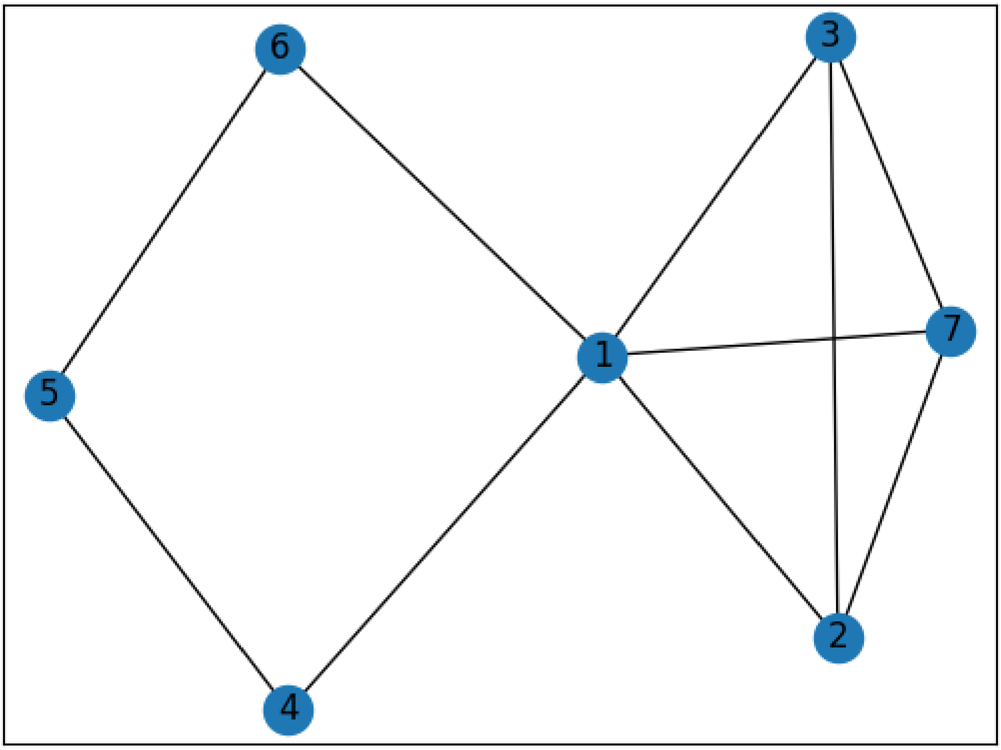
D = 4

E = 5

F = 6

G = 7

Usaremos arestas para indicar que um produto não pode ficar perto de outro. (Estar no mesmo container)



A matriz de adjacência para este grafo é:

	1	2	3	4	5	6	7
1	0	1	1	1	0	1	1
2	1	0	1	0	0	0	1
3	1	1	0	0	0	0	1
4	1	0	0	0	1	0	0
5	0	0	0	1	0	1	0
6	1	0	0	0	1	0	0
7	1	1	1	0	0	0	0

Os números em vermelho indicam os vértices.

Basta rodar o algoritmo e informar que o grafo tem 7 vértices e 10 arestas e em seguida informar de onde parte cada aresta e onde ela chega.

O programa retornará a sequencia

1	2	3	2	1	2	4

Que indica que serão necessárias quatro cores para pintar o grafo, sendo que o primeiro vértice receberá a cor 1, o segundo vértice a cor 2, e assim por diante.

A solução indica que serão necessários 4 containers. Para ver a divisão nos contêineres basta pintar o grafo e ver. (Sou relativamente iniciante no Python e não conheço o suficiente de bibliotecas de grafos para otimizar o trabalho)