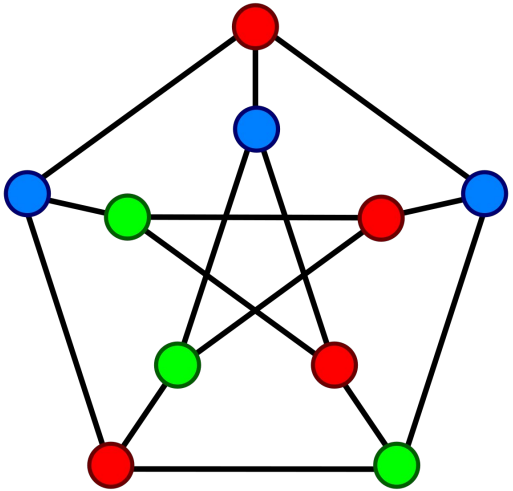
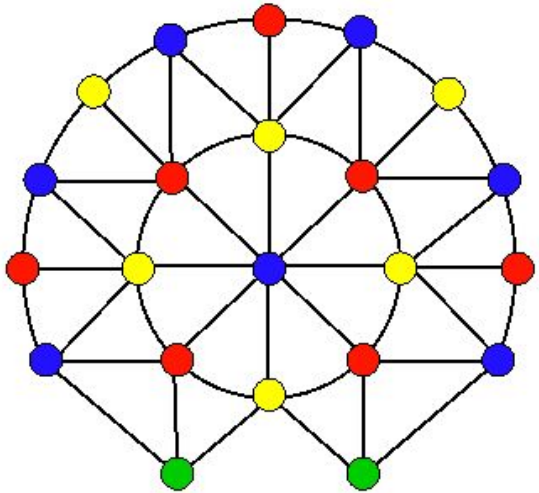


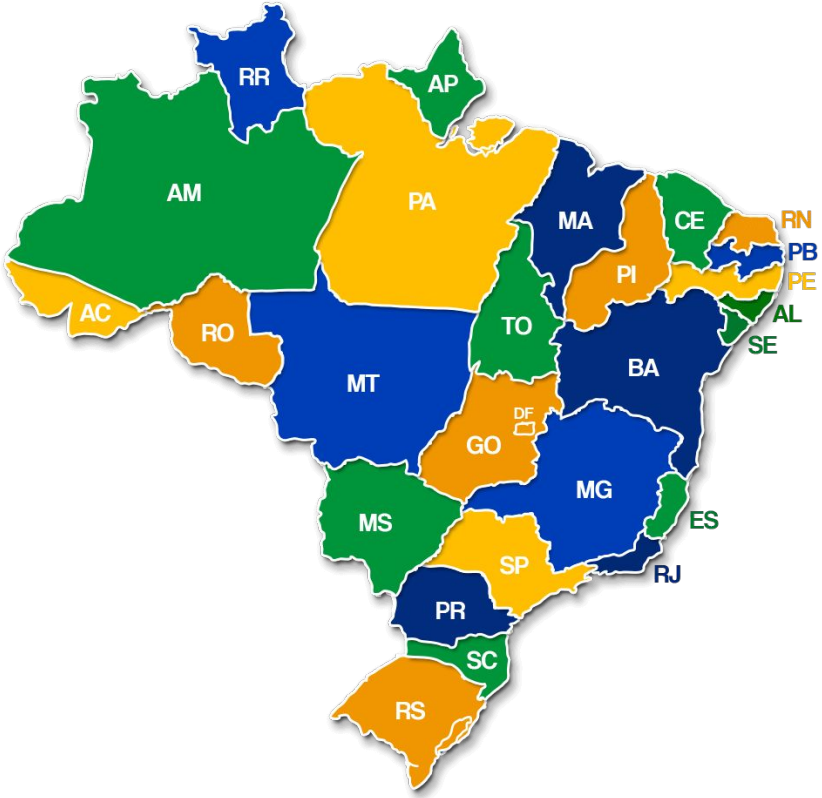
Coloração de Grafos

Renan Carlos Loewenstein



O que é o problema de coloração de grafos?

O problema de coloração de grafos é uma tarefa de otimização que consiste em colorir os vértices de um grafo de tal forma que dois vértices adjacentes não tenham a mesma cor. O objetivo é usar o menor número possível de cores.



Aplicações do problema

- Horários de aulas: cada disciplina é representada por um vértice e as restrições de horários incompatíveis são representadas pelas arestas.

- Alocações de recursos: cada tarefa é representada por um vértice e as restrições de recursos são representadas pelas arestas.

- Mapas de registros de frequência: cada torre de celular é representada por um vértice e as restrições de interferência são representadas pelas arestas.

1-Provar que o problema pertence a NP:

(a) Através de um algoritmo, descrever um verificador para o problema:

Entrada: um grafo (G) e uma coloração (C) de seus vértices

- Para cada vértice (V) em (G), verificar se ele foi colorido com uma cor em (C)
- Para cada aresta (u, v) em (G), verificar se os vértices (u) e (v) têm cores diferentes em C
- Se ambas as verificações forem verdadeiras, então (C) é uma coloração válida de (G), caso contrário, (C) não é uma coloração válida de (G)

1-Provar que o problema pertence a NP:

(b) Provar que o verificador é um algoritmo que executa em tempo polinomial

Para provar, é necessário mostrar que o tempo de execução do verificador é limitado por um polinômio em relação ao tamanho da entrada:

No caso do problema de coloração de grafos, a entrada consiste em um grafo (G) com (n) vértices e (m) arestas, juntamente com uma atribuição de cores a cada vértice. A complexidade de tempo do verificador é então determinada pelo número de verificações necessárias para determinar se a atribuição de cores é válida ou não.

1-Provar que o problema pertence a NP:

(b) Provar que o verificador é um algoritmo que executa em tempo polinomial

- O primeiro passo é verificar se cada vértice tem uma cor válida, o que requer **(n)** verificações, onde **n** é o número de vértices do grafo.
- O segundo passo é verificar se todas as arestas têm cores diferentes, o que requer no máximo **(m)** verificações, onde **m** é o número de arestas do grafo.
- Portanto, o tempo total de execução do verificador é limitado por **O(n+m)**, que é um polinômio em relação ao tamanho da entrada.

Dessa forma, pode-se concluir que o verificador para o problema de coloração de grafos é um algoritmo que executa em tempo polinomial.

1-Provar que o problema pertence a NP:

(c) Implementação do verificador

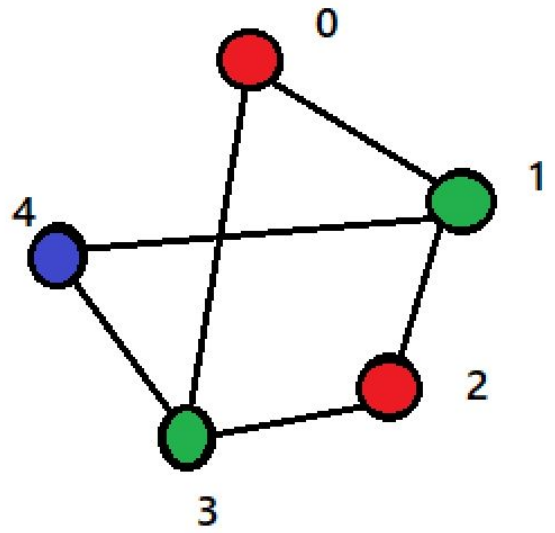
- A função validar_coloracao( ) verifica se uma atribuição de cores é uma k-coloração válida, verificando se cada vértice tem uma cor válida e se nenhum par de vértices adjacentes compartilham a mesma cor.
- Essa função recebe como entrada um grafo em forma de lista de adjacências e uma lista de cores dos seus vértices.

```python
def validar_coloracao(grafo, cores):
    n = len(grafo) #obtem o numero de vertices
    k = max(cores) + 1 #determina o numero de cores usadas+1
                       #pois as cores sao numeradas de 0 ate k-1
    if k <= 0:  #verifica se ha pelo menos uma cor sendo usada
        return False
    for i in range(n): #verifica se as cores sao um valor inteiro
        if cores[i] < 0 or cores[i] >= k:
            return False
        for j in grafo[i]: #verifica se os vertices tem cores diferentes
            if cores[i] == cores[j]:
                return False
    return True

grafo = [
    [1, 3],     #vizinhos do vertice 0
    [0, 2, 4],  #vizinhos do vertice 1
    [1, 3],     #vizinhos do vertice 2
    [0, 2, 4],  #vizinhos do vertice 3
    [1, 3]      #vizinhos do vertice 4
]
cores = [0, 1, 0, 1, 2]

if validar_coloracao(grafo, cores):
    print("Atribuicao de cores eh valida")
else:
    print("Atribuicao de cores NAO eh valida")
```

0 1 2

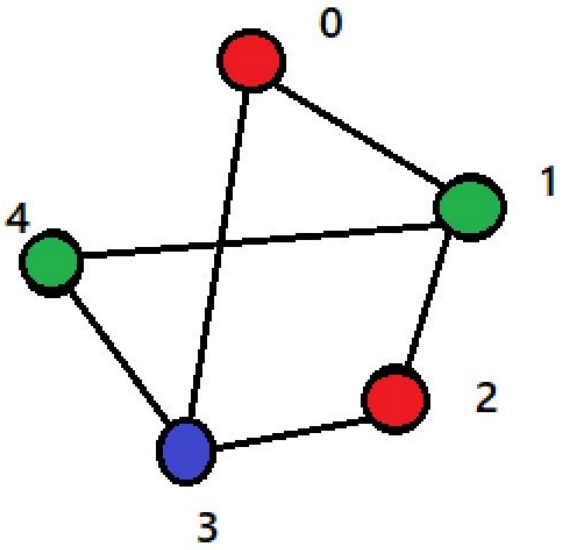


```python
def validar_coloracao(grafo, cores):
    n = len(grafo) #obtem o numero de vertices
    k = max(cores) + 1 #determina o numero de cores usadas+1
                       #pois as cores sao numeradas de 0 ate k-1
    if k <= 0:  #verifica se ha pelo menos uma cor sendo usada
        return False
    for i in range(n): #verifica se as cores sao um valor inteiro
        if cores[i] < 0 or cores[i] >= k:
            return False
        for j in grafo[i]: #verifica se os vertices tem cores diferentes
            if cores[i] == cores[j]:
                return False
    return True

grafo = [
    [1, 3],     #vizinhos do vertice 0
    [0, 2, 4],  #vizinhos do vertice 1
    [1, 3],     #vizinhos do vertice 2
    [0, 2, 4],  #vizinhos do vertice 3
    [1, 3]      #vizinhos do vertice 4
]
cores = [0, 1, 0, 2, 1]

if validar_coloracao(grafo, cores):
    print("Atribuicao de cores eh valida")
else:
    print("Atribuicao de cores NAO eh valida")
```

0 1 2



1-Provar que o problema pertence a NP:

(c) Implementação do verificador

https://colab.research.google.com/drive/1tdBXGHMszBqQlZpCQrRaygN1q3BjDTmx#scrollTo=XxwUjzxtIw7J

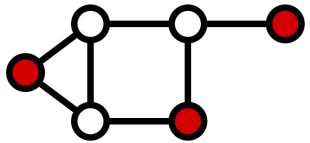
2-Provar que o problema é NP-difícil:

(a) Descrever uma redução de tempo polinomial de um problema NP-completo através de um algoritmo

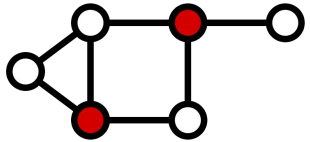
Para provar que um problema é NP-difícil, precisamos mostrar que é possível reduzi-lo a um problema NP-completo conhecido em tempo polinomial.

Nesse sentido, iremos fazer uma redução de tempo polinomial do problema do **conjunto dominante mínimo** para o problema de coloração de grafos.

(a)



(b)



2-Provar que o problema é NP-difícil:

(a) Descrever uma redução de tempo polinomial de um problema NP-completo através de um algoritmo

O problema do conjunto dominante mínimo é um problema em que se deseja encontrar um subconjunto mínimo de vértices em um grafo que domina todos os outros vértices. E o problema da coloração de grafos consiste em encontrar uma atribuição de cores para cada vértice de um grafo de forma que vértices adjacentes não tenham a mesma cor.

2-Provar que o problema é NP-difícil:

(a) Descrever uma redução de tempo polinomial de um problema NP-completo através de um algoritmo

Para realizar a redução, suponha que temos um grafo (G) e queremos verificar se ele tem uma coloração válida com, no máximo, k cores. Primeiramente, construímos um grafo (G') a partir de (G), adicionando um vértice extra (v) e ligando-o a todos os vértices de (G). Em seguida, verificamos se (G') tem um conjunto dominante mínimo de tamanho (k+1). Se isso ocorrer, podemos colorir (G') com (k+1) cores de forma que (v) tenha cor diferente dos outros vértices e, portanto, podemos remover (v) e obter uma coloração válida de (G) com (k) cores.

2-Provar que o problema é NP-difícil:

(a) Descrever uma redução de tempo polinomial de um problema NP-completo através de um algoritmo

Algoritmo:

Entrada: um grafo (G) e um inteiro (k)

- Construir o grafo G (adicionando um vértice extra v e ligando ele a todos os vértices de G.
- Executar o algoritmo de aproximação para o problema.
- Se o tamanho do conjunto encontrado for menor ou igual a k, retornará TRUE, caso contrário, retorna FALSE.

2-Provar que o problema é NP-difícil:

(b) Provar que a redução é de tempo polinomial mostrando que o algoritmo executa em tempo polinomial

O algoritmo de redução tem duas partes: a construção do grafo e a execução do algoritmo de coloração de grafos:

- Na etapa da construção do grafo, ele percorre todos os vértices do grafo original e cria um novo grafo com os mesmos vértices, adicionando arestas entre os vértices que estão conectados ao vértice sendo considerado. Como o número de vértices e arestas do novo grafo é no máximo o dobro do original, a construção pode ser feita em tempo polinomial.

2-Provar que o problema é NP-difícil:

(b) Provar que a redução é de tempo polinomial mostrando que o algoritmo executa em tempo polinomial

O algoritmo de redução tem duas partes: a construção do grafo e a execução do algoritmo de coloração de grafos:

- Na etapa da execução do algoritmo de coloração de grafos, existem vários algoritmos de coloração que executam em tempo polinomial. Portanto, podemos escolher um desses algoritmos para a execução da redução.

2-Provar que o problema é NP-difícil:

(b) Provar que a redução é de tempo polinomial mostrando que o algoritmo executa em tempo polinomial

- Assim, como ambas as partes da redução podem ser realizadas em tempo polinomial, concluímos que a redução de tempo polinomial do problema do conjunto dominante mínimo para o problema de coloração de grafos é de tempo polinomial.

Referências

https://www.ic.unicamp.br/~atilio/slidesWtisc.pdf

https://sites.icmc.usp.br/andretta/ensino/aulas/sme0216-5826-2-15/grupo5.pdf

https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/vertex-coloring.html

https://www.studocu.com/pt-br/document/universidade-regional-do-noroeste-do-estado-do-rio-grande-do-sul/complexidade-computacional/metodos-para-reducao-de-problemas/11222900

http://www.decom.ufop.br/marco/site_media/uploads/bcc204/12_aula_12.pdf

https://repositorio.bc.ufg.br/tede/bitstream/tde/2901/5/dissertacao%20rommel%20cc.pdf

https://danielsaad.com/teoria-da-computacao/assets/aulas/redutibilidade.pdf