ALGORITMOS DE TEORIA DOS GRAFOS E SUAS APLICAÇÕES PRÁTICAS

Yuri Tainã Neto Oliveira
Renan Silveira de Sousa Mendonça

1 – CAMINHOS EM GRAFOS UTILIZANDO O ALGORITMO DE DIJKSTRA

O algoritmo de Dijkstra é um algoritmo usado para encontrar o caminho mais curto entre dois vértices em um grafo ponderado, onde os pesos são associados às arestas.

Tal algoritmo funciona através da construção de uma árvore de caminho mínimo, começando com um vértice inicial que segue explorando todos os seus vizinhos, atualizando o custo para alcançar cada um deles. Após a construção da árvore, o algoritmo seleciona o vértice com o custo mais baixo e repete o processo. Dessa forma, ele constrói uma árvore de caminho mínimo que representa o caminho mais curto para cada vértice no grafo em relação ao vértice inicial.

1.1 – COMPLEXIDADE TEMPORAL

A complexidade temporal do algoritmo de Dijkstra está diretamente relacionada à maneira de como ele escolhe o próximo nó com a menor distância e de como é atualizado as distâncias dos vizinhos desse nó.

A complexidade total do algoritmo depende tanto do número de nós V, quanto do número de arestas E. A complexidade geral será a soma das duas operações de seleção do menor nó (O (V log V)) e de atualização de distâncias (O (E log V)).

Com isso, a complexidade temporal do algoritmo de Dijkstra usando um min-heap é O ((V + E) log V), significando que ele é mais eficiente quando V (número de nós) e E (número de arestas) são grandes.

Se usar uma estrutura de dados menos eficiente, como uma lista ou grafos densos onde há muitas arestas, a complexidade poderá chegar até O(V^2), sendo muito pior.

1.2 – PRINCIPAIS DIFERENÇAS DE APLICABILIDADE E RESTRIÇÃO ENTRE DIJKSTRA E BELLMAN-FORD

O algoritmo de Dijkstra não é adequado para grafos com arestas de peso negativo. Seu resultado oferece informações completas sobre a rede, não se limitando aos vértices conectados diretamente. Não pode ser implementado facilmente de forma distribuída, entretanto é mais rápido que o algoritmo Bellman-Ford.

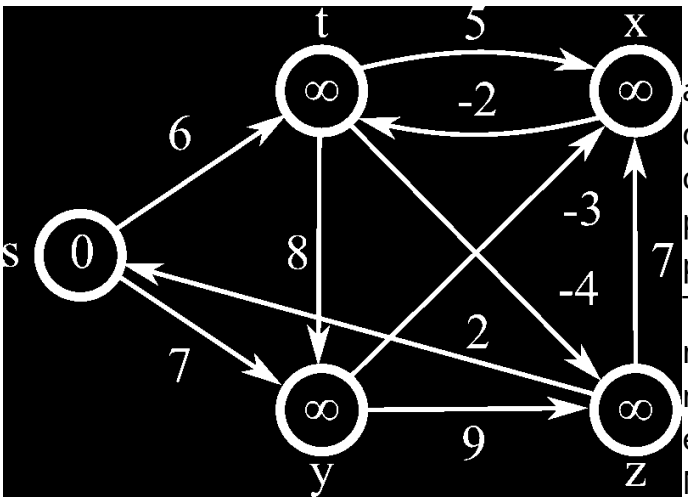
Já o algoritmo de Bellman-Ford possibilita o uso de arestas de peso negativo e a detecção de ciclos negativos. O resultado do algoritmo inclui os vértices com informações sobre os outros vértices aos quais estão conectados. Ele pode ser implementado de maneira distribuída com facilidade, embora seja mais lento que o algoritmo de Dijkstra.

Em resumo, Dijkstra supõe que todos os pesos das arestas no grafo de entrada são positivos; Já o algoritmo de Bellman-Ford permite a detecção da existência de ciclos, junto com o uso de arestas com peso negativo.

1.3 – EXEMPLO DE CENÁRIO EM QUE FLOYD-WARSALL É MAIS ADEQUADO QUE DIJKSTRA

Levando em consideração a diferença entre Djikstra e Floyd-Warshall, e tomando como exemplo um cenário de análise de rede de trânsito, o algoritmo de Floyd se torna mais útil se usado para encontrar o caminho mais curto entre todos os pares de vértices de um grafo, já que o algoritmo de Floyd pode fazer isso de forma mais eficiente.

Outro exemplo seria de uma cidade em que as ruas (nós) intercedem entre si, com distintos tempos de viagem (peso das arestas). O objetivo seria saber o tempo mais curto das viagens entre todas as interseções, para melhorar a eficiência do trânsito. Considerando este exemplo Floyd é mais eficiente, já que computaria todos os caminhos mais curtos entre todas as interseções de forma direta e eficiente.



O algoritmo de Floyd é um algoritmo que resolve o problema de determinar o caminho mais curto entre todos os pares de nós em um grafo orientado e ponderado.

Trata-se de um algoritmo que utiliza matrizes para determinar os caminhos mínimos entre todos os pares de nós da rede. No algoritmo de Floyd são feitas n iterações que corresponde ao número de nós da rede.

1.4 – PROJETO DE APLICAÇÃO DO ALGORITMO DE JOHNSON

O artigo escolhido aborda a aplicação da Teoria dos Conjuntos Aproximativos (TCA) na mineração de dados meteorológicos, com o objetivo de identificar padrões que possam indicar a ocorrência de eventos convectivos severos (fenômenos associados a chuvas e ventos fortes de curta duração, ou então de intensidade média, porém de duração prolongada que, em geral, causam sérios danos, tornando

sua previsão altamente desejável). E é utilizado o algoritmo de Johnson que é uma heurística que simplifica o cálculo das reduções.

Basicamente pegaram dados de Descargas Atmosféricas usaram o TCA (cujo enfoque é o tratamento de imprecisão e incerteza nos dados) para identificar padrões e depois usaram o algoritmo de Johnson para aplicar a redução, usando o Bellman-Ford para computar os dados de entrada e retirar os pesos negativos para então usar o Djikstra que então poderá ser utilizado, devido ao fato dos pesos terem sido colocados como positivos.

No final de tudo, os dados vão ser classificados para as três mini regiões (A, B, C), e depois é reduzido pelo algoritmo de Johnson, reduzindo os dados de cada uma das mini regiões, que depois aplica uma regra para classificação. Em seguida são geradas as matrizes dos resultados (onde incluem dados corretos e incorretos).

EX: se o dado é D e foi classificado como M então está incorreto, então se a taxa deste erro for extremamente baixa ou nula, isso demostra uma efetividade grande a respeito desta aplicação, podendo gerar resultados melhores, garantindo a efetividade do método.

3 – ALGORITMOS DE COLORAÇÃO

Algoritmos de coloração focam em atribuir cores aos vértices, de forma que a cor de um vértice deve ser diferente das cores dos seus vértices adjacentes. A coloração de grafos tem diversas aplicações no mundo real, como ajuda na coloração de mapas cartográficos e ajuda no controle de torres de telefone (GSM).

3.1 – ALGORITMO DE COLORAÇÃO GULOSA

O algoritmo de coloração gulosa (*greedy coloring*) funciona pegando uma sequência de vértices e atribuindo uma cor para cada um deles. Algoritmos de coloração gulosa podem encontrar a solução em tempo linear, mas podem não utilizar o número mínimo possível de cores.

Em um grafo, a sequência em que os vértices são processados podem gerar um resultado de coloração completamente diferentes um do outro. Isso faz com que o estudo da coloração gulosa foque em encontrar a melhor maneira de ordenar tais valores de forma a obter o melhor resultado.

Um exemplo de melhor e pior casos na ordenação da coloração gulosa pode ser encontrado na imagem abaixo. (Figura 1).

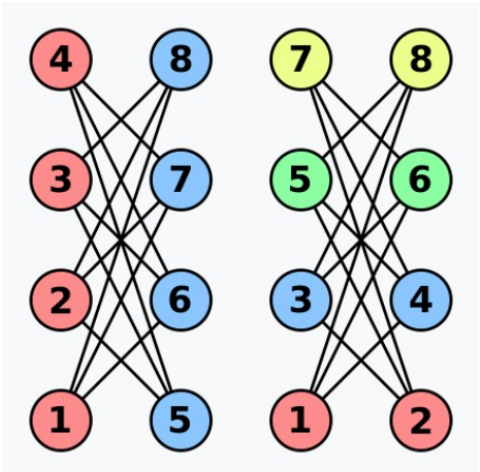


Figura 1: Melhor e pior caso da coloração gulosa

A imagem ao lado possui dois grafos isomorfos, com o da esquerda representando a melhor ordem dos vértices em que o algoritmo de coloração gulosa foi aplicado; e o da direita sendo o pior dos casos.

O número máximo de cores utilizado é igual ao número de grau máximo para qualquer vértice mais 1 (Δ+1).

No segundo grafo da imagem ao lado, pode-se observar que o número máximo de conexões que qualquer vértice possui é igual a três (grau máximo), e possui exatamente quatro (grau máximo + 1) cores sento aplicadas, comprovando o pior caso.

Existem algumas características que fazem o algoritmo de coloração gulosa não ser a melhor opção em alguns casos, tais características podem se destacar:

- Ele não garante que o mínimo de cores será usado.
- No pior dos casos, a quantidade de cores usada será igual à quantidade do número de grau máximo mais um (Δ+1), quando podia ser um valor bem menor.
- Pode não se adaptar a algumas propriedades dos grafos.

3.2 – ALGORITMO DSATUR

A maior diferença do algoritmo de coloração gulosa para o DSatur é a forma de ordenação dos vértices. Porém, similar ao algoritmo de coloração gulosa, o algoritmo DSatur também colore os vértices um após o outro, seguindo uma ordem.

Após o algoritmo colorir um vértice, ele escolhe um dos vértices que ainda não foi atrubuído uma cor. Tal escolha é decidida ao pegar o vértice sem cor que possui o maior número de vértices adjacentes com cor. Este número é determinado **grau de saturação** (d*egree of saturation*), que dá nome ao algoritmo.

Na hora da escolha do vértice com maior grau de saturação, caso ocorra de dois vértices terem o mesmo maior grau, é escolhido o que posssui o maior grau no subgrafo induzido pelos vértices sem cor.

No pior dos casos, a complexidade do algoritmo DSatur é $O(n^2)$, onde n representa o número de vértices do grafo.

DSatur é uma melhor opção em relação ao algoritmo de coloração gulosa quando se estiver trabalhando com grafos bipartidos, cíclicos e grafos de roda.

3.3 – TEOREMA DE QUATRO CORES

O teorema das quatro cores afirma que é possível, em um grafo plano, atribuir uma cor para cada um desses nós, de forma que o número máximo de cores utilizadas não passe de quatro.

Este teorema possui várias aplicações no mundo real, além de ser um teorema importante para a teoria dos grafos em geral. Ele foi difícil de se comprovar como verdadeiro, sendo necessário a utilização de máquinas para efetuar os cálculos.

Quando a primeira prova que confirmava tal teorema foi descoberta ela se demonstrava inviável de se conseguir de forma humana, com lápis e papel. Isso marcou um pilar de grande importância na coloração de grafos, e em grafos no geral, que impulsionou o estudo da área.

3.4 – PROJETO DE APLICAÇÃO DA COLORAÇÃO DE GRAFOS

Foi escolhido um artigo que detalha o uso e aplicação da coloração de grafos em uma necessidade real, mais precisamente na criação de mapas cartográficos.

Quando se fala em um mapa, é de extrema importância que ele seja organizado e de fácil leitura, com a cor que ele possui sendo um fator de extrema importância.

Imaginamos, por exemplo, o mapa do Brasil de forma que cada estado possua uma cor atribuída de forma aleatória. Este mapa seria extremamente confuso, pois teria vários estados vizinhos com cores parecidas (porém, não iguais), dificultando a leitura; e poderia utilizar uma quantidade desnecessária de cores, tornando-se visualmente carregado.

A coloração de grafos voltada para mapas cartográficos foca em criar tais mapas com a melhor distribuição de cor possível. Seguindo a teoria das quatro cores, podemos colorir um mapa completo com no máximo quatro core. Isso faz com que os mapas sejam de fácil leitura.

3.4.1 – REPRESENTAÇÃO DE MAPAS COMO GRAFOS

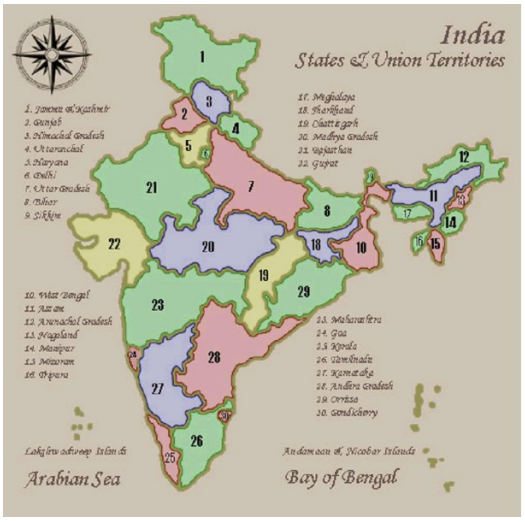


Figura 2: Mapa da índia

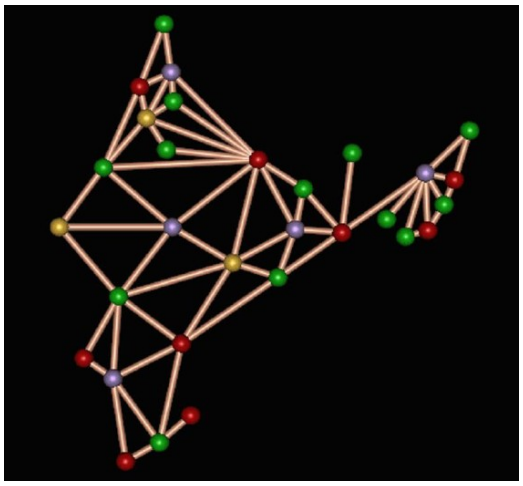


Figura 3: Mapa da índia representado como um grafo

Antes de tudo, precisamos pensar na representação de um país, por exemplo, a índia, como um grafo. Para tal, será utilizado o mesmo exemplo utilizado por *Shamim Ahmed*, no artigo ***Applications of Graph Coloring in Modern Computer Science.***

Na *Figura 2*, o mapa da índia pode ser observado, junto com uma lista de cada estado pertencente ao país.

Na representação do mapa como um grafo, podemos imaginar cada estado como sendo um nó **V** de um grafo. Caso um estado V1 possua ligação com um estado V2, tal ligação pode ser representado como uma aresta não direcionada partindo de V1 e indo até V2

Após extrair os dados do mapa e efetuar as ligações necessárias, o resultado obtido deve ser similar ao da *Figura 3*.

3.4.1.1 – APLICAÇÃO SIMILAR DE COLORAÇÃO DE GRAFOS, PORÉM VOLTADA À TORRES DE TELEFONIA MÓVEL

Outro exemplo interessante disponibilizado no material de *Shamim Ahmed*, é a utilização do algoritmo de coloração de grafos com o teorema de quatro cores para a distribuição de torres de telefonia móvel.

A *Global System for Mobile Communications* (GSM) é um padrão de comunicação voltado a dispositivos móveis. A primeira aparição do GSM é mais conhecida como 2G (sendo a evolução do 1G).

O GSM permite o funcionamento em quatro bandas distintas, com cada país diferenciando a quantidade de bandas necessárias. A *Figura 4* mostra uma lista de países e suas respectivas bandas utilizadas.

Countries that mix GSM 900/1800 and GSM 850/1900 bands

Region	Country/Territory	GSM-850	GSM-1900	GSM-900	GSM-1800
Caribbean	Aruba, Bonaire and Curacao	✗	✓	✓	✓
Caribbean	Antigua and Barbuda	✓	✓	✓	✗
Caribbean	Barbados	✓	✓	✓	✓
Caribbean	British Virgin Islands	✓	✓	✓	✓
Caribbean	Cayman Islands	✓	✓	✓	✓
Caribbean	Dominica	✓	✓	✓	✓
Caribbean	Dominican Republic	✓	✓	✓	✓
Caribbean	Grenada	✓	✓	✓	✓
Caribbean	Haiti	✓	✗	✓	✓
Caribbean	Jamaica	✓	✓	✓	✗
Caribbean	Saint Kitts and Nevis	✓	✓	✓	✓
Caribbean	Saint Lucia	✓	✓	✓	✓
Caribbean	Saint Vincent and the Grenadines	✓	✓	✓	✓
Caribbean	Turks and Caicos Islands	✓	✓	✓	✓
Central America	Costa Rica	✓	✗	✗	✓
Central America	El Salvador	✓	✓	✓	✗
Central America	Guatemala	✓	✓	✓	✗
South America	Brazil	✓	✗	✓	✓
South America	Uruguay	✓	✓	✓	✓
South America	Venezuela	✓	✓	✓	✓

Figura 4: Bandas GSM em cada país (2016)

Como pode ser observado, cada país possui ao menos duas faixas de banda em operação até 2016. De forma a distribuir cada torre de melhor forma possível, o algoritmo de coloração com teorema de quatro cores pode ser aplicado, sendo a solução perfeita.

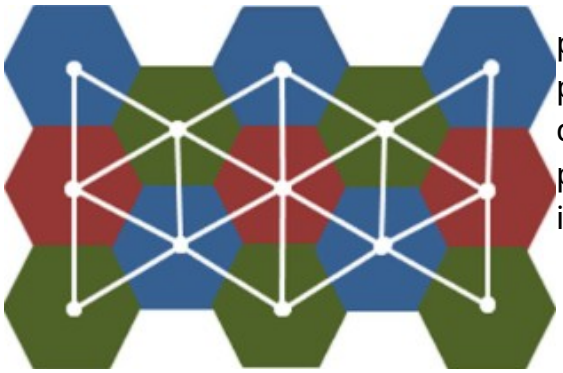


Figura 5: Exemplo de distribuição de torres de sinal GSM

Na *Figura 5* pode ser observado uma possível distribuição de três bandas de sinal para diversas torres. Tal distribuição permite que bandas iguais estejam o mais distantes o possível, reduzindo a possibilidade de interferências.

3.4.2 – APLICAÇÃO PRÁTICA DO TEOREMA DE QUATRO CORES

Para a implementação da teoria de quatro cores a fim de demonstrar o uso em um cenário real, foi feito uma aplicação para a atribuição de cores a mapas cartográficos, utilizando como exemplo o mapa do Brasil. A aplicação foi escrita na linguagem Python, que é de fácil compreensão e utilizada nos mais variados cenários.

3.4.2.1 – REPRESENTAÇÃO DE UM PAÍS EM FORMA DE GRAFO

Para a descrição do mapa do Brasil com seus estados, foi utilizado um dicionário (tipo de variável no Python), que contém o nome do estado, uma lista de estados vizinhos (adjacentes), e a cor deste estado.

Como exemplo simplificado do código vamos utilizar apenas os estados Acre e Amazonas:

```
country = {
        "Acre": {
                "adjacency": ["Amazonas", "Rondônia"],
                "color": None
        },
        "Amazonas": {
                "adjacency": ["Acre", "Roraima", "Pará", "Mato Grosso", "Rondônia"],
                "color": None
        }
}
```

A variável (do tipo dicionário) *country* (“país”, em inglês) neste exemplo, possui dois itens que representam um estado cada (Acre e Amazonas). Cada um dos estados possui uma lista de adjacência (nomes dos estados vizinhos) e a cor final que este

estado deve possuir no mapa, que inicialmente é definido como **None** (será atribuída mais a frente).

Neste exemplo foi escolhido dois estados vizinhos, com isso é possível notar que Acre está presente na lista de adjacência de Amazonas, e vice-versa. Quando dois estados não forem vizinhos, eles simplesmente não aparecem na lista de adjacência um do outro.

3.4.2.2 – VERIFICAÇÃO DE GRAFO NÃO DIRECIONADO

A primeira característica para a aplicação do teorema de quatro cores é a necessidade de que o grafo seja não direcionado. Esta função do programa verifica exatamente isso.

```
def isGraphUndirected():
        for state in country:
                for adj in country[state]["adjacency"]:
                if(state not in country[adj]["adjacency"]):
                        return False
return True;
```

Em cada estado **v**, é verificado para cada um de seus estados vizinhos (adjacentes), se tais estados vizinhos possuem o estado **v** em suas listas de adjacências.

Caso ocorra de um estado **V1** estiver em sua lista de adjacência o estado **V2**, porém **V2** não estiver em sua lista de adjacência o estado **V1** (aresta direcionada), a função retorna **False**; A função retorna **True** se não for encontrado nenhuma aresta direcionada.

3.4.2.3 – VERIFICAÇÃO DE GRAFO PLANAR

A segunda característica para a aplicação do teorema de quatro cores é a necessidade de que o grafo seja planar. Este código não implementa esta verificação devido a falta de tempo, porém deve-se enfatizar a necessidade de utilizar um grafo planar.

3.4.2.4 – PEGAR O ESTADO SEM COR ATRIBUÍDA, COM O MAIOR NÚMERO DE ESTADOS ADJACENTES QUE POSSUEM COR

O teorema de quatro cores é aplicado seguindo a ordem dos vértices sem cor atribuída, com o maior número de vértices adjacentes. Para conseguir descobrir qual é esse vértice no mapa, o seguinte código é utilizado:

```
def getHighestColoredAdjState():
        highestAdjCount = -1;
        highestAdjState = None;

        for state in country:
                if(country[state]["color"] != None):
                        continue;
                coloredCount = 0;
                for adj in country[state]["adjacency"]:
                        if(country[adj]["color"] == None):
                        coloredCount += 1;
                if coloredCount > highestAdjCount:
                        highestAdjState = state;
                        highestAdjCount = coloredCount;
        return(highestAdjState)
```

Caso não possua nenhum estado sem cor, esta função retorna **None**.

3.4.2.5 – CÓDIGO DE APLICAÇÃO DO TEOREMA DE QUATRO CORES

A função abaixo é o principal, responsável por aplica o teorema de quatro cores no grafo, definindo uma cor a cada estado.

```
def fourColorTheorem():
        while True:
                st = getHighestColoredAdjState();
                if st == None:
                        break;

                # pega a lista de cores de cada vértice adjacente
                adjColors = list();
                for adj in country[st]["adjacency"]:
                        color = country[adj]["color"];
                        if(color != None):
                                adjColors.append(color);

                # escolhe uma nova cor
                newColor = None;
                if len(adjColors) != 0:
                        for i in range(len(adjColors) + 1):
                                # caso adjColors for, por exemplo {1, 2, 4},
                                # newColor será 3.
                                if(i not in adjColors):
                                        newColor = i;
                                        break;
                else:
                        newColor = 1;

                country[st]["color"] = newColor;
```

Fontes:

https://www.researchgate.net/profile/Shamim-Ahmed-38/publication/344016939_Applications_of_Graph_Coloring_in_Modern_Computer_Science/links/5f4e3e3a458515e96d1e2e3b/Applications-of-Graph-Coloring-in-Modern-Computer-Science.pdf

https://en.wikipedia.org/wiki/Greedy_coloring

https://en.wikipedia.org/wiki/Four_color_theorem

https://en.wikipedia.org/wiki/GSM

https://en.wikipedia.org/wiki/GSM_frequency_bands#cite_note-13

https://www.researchgate.net/profile/Shamim-Ahmed-38/publication/344016939_Applications_of_Graph_Coloring_in_Modern_Computer_Science/links/5f4e3e3a458515e96d1e2e3b/Applications-of-Graph-Coloring-in-Modern-Computer-Science.pdf

https://dl.acm.org/doi/pdf/10.1145/355274.355299

https://dspace.unipampa.edu.br/bitstream/riu/7751/1/Anny%20Elise%20Santos%20Nunes%20-%202022.pdf