Determinar o número cromático de um dado grafo não orientado G

Pedro Xavier Leite Cavadas, Nº Mec. 85090, MEI

*Resumo* – Este artigo é realizado no âmbito da cadeira Algoritmos Avançados do Mestrado em Engenharia Informática da Universidade de Aveiro.

Neste artigo pretende-se apresentar uma possível solução para o problema de determinar o número cromático de um dado gráfico não orientado G, com recurso a uma algoritmo de pesquisa exaustiva.

Para isso será apresentado, detalhadamente, o problema, a estratégia de resolução utilizada, a solução obtida, bem como testes de *performance* e a análise aos mesmos.

*Abstract* – This article is written under the Advanced Algorithms course of the Master in Software Engineering, Universidade de Aveiro.

This is paper This paper aims to present a possible solution to the problem of determining the chromatic number of a given non-oriented graph G, utilizing an exhaustive search algorithm.

With this in mind, a detailed explanation of the problem will be given, as well as, the resolution strategy, the solution obtained, the results of the performance testing and their respective analysis.

# Introdução

Este artigo é realizado no âmbito da cadeira Algoritmos Avançados onde é nos proposto um problema junstamente com um método de resolução. O objetivo do trabalho proposto é implementar um algoritmo capaz de resolver o problema com base no método de resolução proposto. Além disto, é necesário realizar testes (de *performance*) e a análise dos mesmos.

O problema escolhido foi então o de determinar o número cromático de um dado grafo não orientado G, tendo como método de resolução, um algoritmo de pesquisa exaustiva.

Este projeto divide-se então em duas partes: um s*script python* para a geração de um grafo aleatório (não orientado) com *n* vertices e *m* arestass; e outro *script python* com a implementaçãp do algoritmo de pesquisa exaustiva para a resolução do problema.

Este artigo describe então alguns conceitos técnicos necessários para a resolução do problema, uma análise detalhada ao problema, à estratégia, ao algoritmo, à implementação, aos testes e à sua respetiva análise.

No final serão também feitas algumas previsões tendo como base a análise dos resultados dos testes.

# Conceitos técnicos

# Grafo Não Orientado

Um grafo é um conjuntos de elementos unidos por arcos (arestas). Um grafo não orientado é um grafo, dado por:

* Um conjunto *V* de vertices;
* Um conjunto *E* de arestas;
* Uma função *w*: *E* -> *P(V)* que associa a cada aresta um subconjunto de 2 ou de 1 elementos de *V*.

Num grafo não orientado dois vértices são adjacentes se e só se existir uma aresta *E* tal que esses dois vertices sejam os pontos terminais dessa aresta, por outras palavras, são conectados por essa aresta.

# Pesquisa Exaustiva

Um algoritmo de pesquisa exaustiva (também conhecido como algoritmo de força bruta), é um algoritmo onde todas as soluções são testadas até que se encontre a solução desejada. Este algoritmo garante sempre a resolução do problema, no entanto, na maioria das vezes o tempo de execução cresce exponencialmente (ou até mais) em relação ao tamanho do problema. Tendo isto em conta, é impensável, na maioria das vezes, utilizar esta metodologia para resolver um problema.

O algoritmo passa por gerar todas as soluções possíveis e de seguida testar quais das soluções obedecem a dadas condições (que façam da possível solução, uma solução real).

# Número cromático de um grafo

O número cromático de um grafo expressa o número mínimo de cores necessárias para colorir as arestas desse mesmo grafo, de tal forma que a vértices correspondam a cores distintas.

Pode-se denotar o número cromático de um grafo G com a notação χ(G) ou γ(G).

Duas propriedas evidentes do número cromático são as seguintes:

1. 1 ≤ χ(G) ≤ |*V*|
2. O número cromático de qualquer grafo complete com *n* vértices é *n*.

Tendo como exemplo os seguintes grafos (com o mesmo número de vertices, mas deiferente nuúmero de arestas):

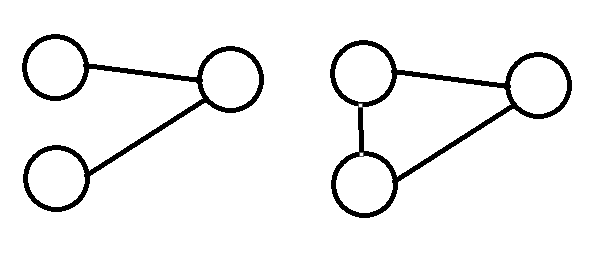


Figura - Dois grafos (não coloridos) com 3 vértices

Se calcularmos o número cromático de cada um destes grafos, obtemos 2 como número cromático do grafo à esquerda, e 3 como número cromático do grafo à direita, ou seja, são necessárias apenas 2 cores diferentes para colorir o grafo à esquerda de forma a que vertices adjacentes sejam coloridos com cores diferentes, já no da direita são necessárias 3 cores, ainda que ambos tenham o mesmo número de vértices.

Sendo assim, uma possível coloração para cada um dos grafos poderá ser:

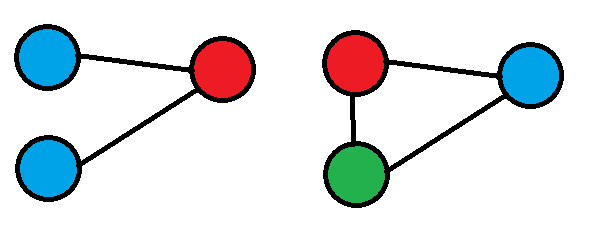


Figura - Dois grafos (coloridos, de forma a ter o número cromático) com 3 vértices

# Estratégia

Para a resolução do problema proposto, foi então aplicada pesquisa exaustiva sobre o conjunto de soluções possíveis.

Para perceber ao certo como isto será feito é preciso perceber as estrturas de dados definidas e como serão utilizadas:

* Classe *Graph*: esta classe é um *wrapper* à volta de um dicionário onde as *keys* são um *ID* de um dado vértice e os *values* uma lista com os *ID*s dos vértices adjacentes ao vértice identificado pela *key*.
* *ID* de um vértice: este *ID* começa em 0 e vai até *n* (número de vértices do grafo) – 1.
* Uma lista com os *ID*s de todos os vértices (esta lista é dada por uma propriedade da classe *Graph*)
* Uma lista com *ID*s representativos das cores. Estes *ID*s tais como os *ID*s dos vértices começam em 0 e vão até *n* – 1.

Posto isto, é então necessário gerar todas as soluções possíveis de forma a testar-las e a descobrir uma solução real. A ideia por trás de gerar estas soluções passa por calcular o produto cartesiano da lista [ 0 .. *n* – 1 ] por ela mesma com *n* repetições. Isto é equivalente a gerar todas as combinações possíveis dos *ID*s das cores com repetição Isto gera uma lista que é então ordenada com base no número de cores diferentes por ordem crescente. Exemplo para *n* = 2: [ 0, 1 ] x [ 0, 1 ] = [ (0, 0), (0, 1), (1, 0), (1, 1) ].



Figura - Linha de código responsável por gerar as combinações de cores por ordem crescente do número de cores diferentes

De seguida, para cada uma das combinações de cores geradas, é feito a correspondência com os *ID*s dos vértices com base no índice, por exemplo, para a possível solução (0, 0) é gerado o dicionário { 0: 0, 1: 0 } onde as *keys* são os *ID*s dos vértices e os *values* os *ID*s das cores.



Figura - Linha de código responsável por gerar o dicionário de associação entre vértices e cores de uma dada solução

Finalmente itera-se sobre esta lista, obtém-se a lista de adjacentes de cada vértice e verifica-se se têm todos cores diferentes (também com base na lista), caso algum vértice tenha algum adjacente da mesma cor a solução é imediatamente descartada, caso contrário, é tida como uma possível solução e é essa a utilizada para calcular o número cromático (para isto basta ver o número de *ID*s diferentes dessa solução), Isto funciona pois a lista de cores estava já ordenada por ordem crescente com base no número diferente de cores.

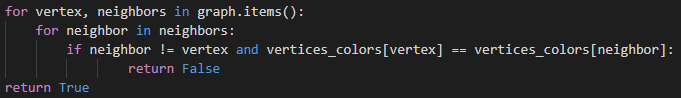


Figura - Pedaço de código responsável por verificar se uma solução é válida

Implementação completa:

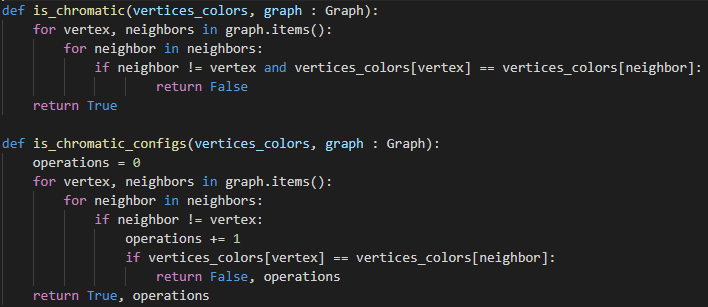


Figura - Implementação completa do algoritmo de pesquisa exaustiva sobre um grafo para o cálculo do seu número cromático

# Implementação

A implementação do algoritmo e da resolução do problema é feita na linguagem *Python3*. A seguir estão apresentados os módulos implementados e as bibliotecas utilizadas na sua implementação.

# Módulos

1. *graph.py* – módulo que contém a classe *Graph*. Esta classe possui um método estático para a geração de um grafo aleatório. Além disto implementa também alguns métodos e propriedades que auxiliam tanto na geração do grafo em si, bem como no processamento do mesmo.
2. *algorithm.py* – este módulo contém o algoritmo para a resolução do problema, bem como uma segunda implementação do mesmo com a contagem do número de operações básicas e do número de soluções testadas. O módulo implementa também uma função para contabilização do tempo gasto numa dada função e uma main onde se encontram os testes de *performance*.

# Bibliotecas Utilizadas

1. *Itertools* – módulo nativo ao *Python3* que fornece ferramentas para a geração. bem como manipulação de iteradores. Neste trabalho em particular foi utilizada para gerar o produto cartesiano da lista [ 0 .. *n* – 1 ] por ela mesma com *n* repetições.
2. *Sys* – módulo nativo ao *Python3* que fornece ferramentas de interação com o sistema nativo da máquina. Neste projeto foi utilizado para saber o valor inteiro máximo uma variável pode assumir.
3. *Math* – Também um módulo nativo ao *Python3*, que fornece ferramentas matemáticas. Utilizado para arredondamentos e cálculo da raíz quadrada.
4. *Random* – Outro módulo nativo ao *Python3*. Contém ferramentas de geração de valores aleatórios. Neste trabalho utilizado para a geração aleatório de um grafo.
5. *Time* – Último módulo nativo ao *Python3* utilizado. Disponibiliza ferramentas que permitem trabalhar com o tempo, tais como obter o *timestamp* atual, entre outros.
6. *Matplotlib* – módulo não nativo do *Python3*. Este módulo permite o *plot* de dados na forma de gráficos, *charts*, entre outros. Neste trabalho foi utilizado para a visualização dos resultados dos testes feitos aos algoritmos.

# Análise de Complexidade

Com base no que foi dito acima, chega-se à conclusão que a complexidade do algoritmo está na ordem de O(*nn*), segundo a notação big-O, isto porque gera-se o produto cartesiano de [ 0 .. *n* – 1] com *n* repetições, o que gera *nn* combinações de cores que dps deverão ser testadas.

# Testes de *Performance*

Para análise de *performance* foi executado um programa de teste que cria vários gráficos aleatórios e calcula o número de operações básicas, o tempo gasto (média de 10 execuções) e as soluções percorridas quando submetido ao algoritmo. Estes gráficos têm um número de vértices entre 1 e 8, sendo que para cada número de vértices existe um grafo com , , e do máximo possível de arestas para o dado número de vértices (no caso desse número ser inferior a 4, o número de arestas é arredondado para baixo).

Por fim, os resultados são guardados num ficheiros e apresentados em dois gráficos, um que mostra a relação entre o número de vértices e o o tempo gasto, e outro que mostra a relação entre o número de vértices e o número de operações básicas (ambos com o número máximo de arestas).

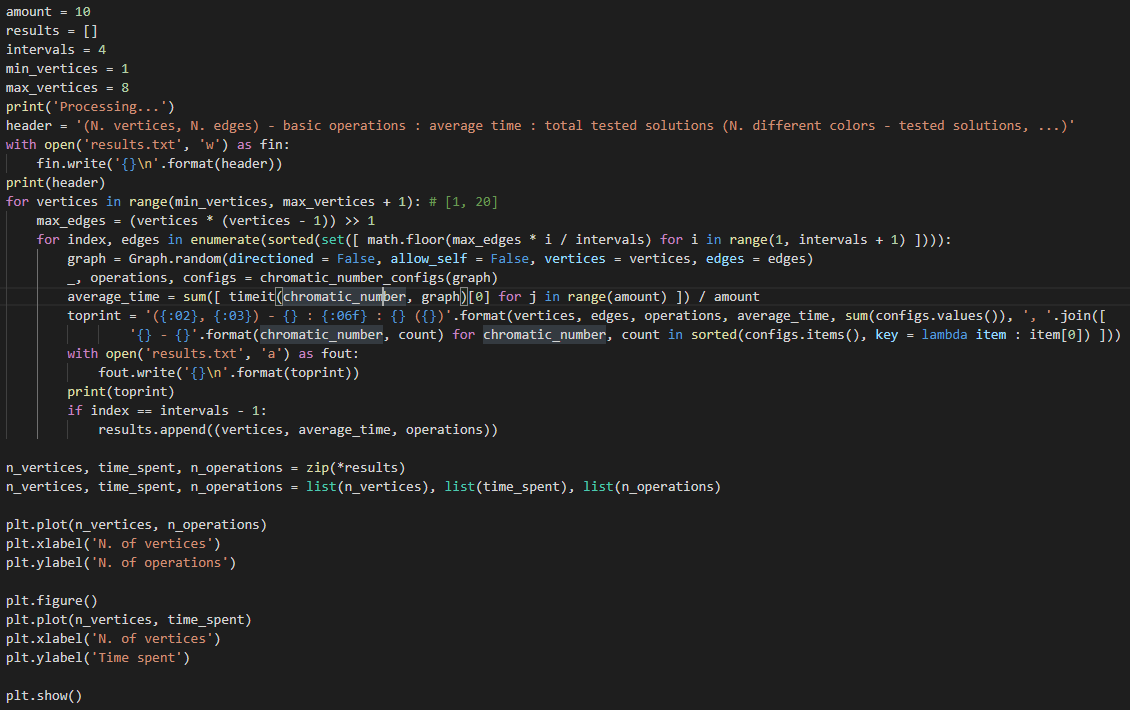


Figura - código de teste de performance

# resultados

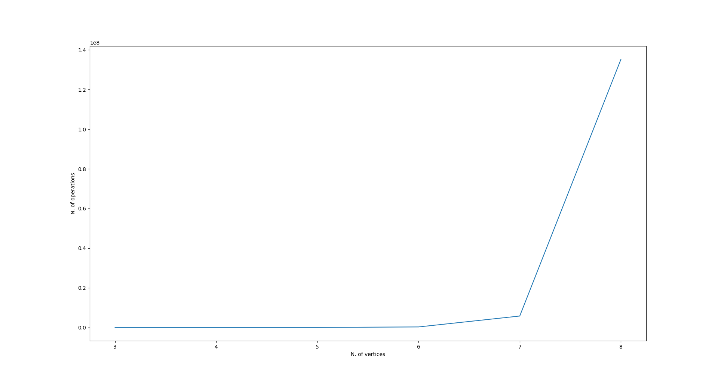


Figura - Relação entre o número de vértices e o número de operações básicas

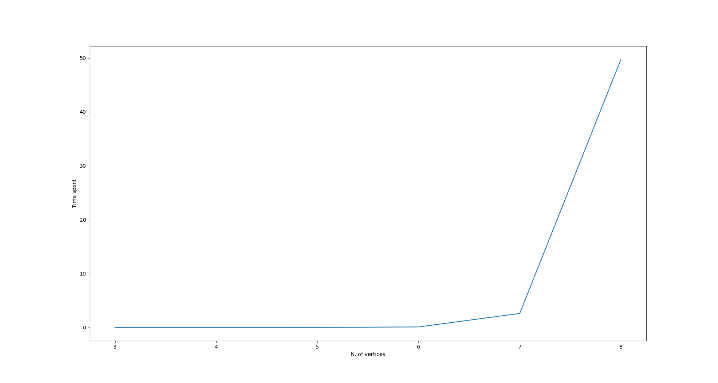


Figura - Relação entre o número de vértices e o número de operações básicas

# Análise dos Resultados

Pelo gráfico da relação entre o número de vértices e o número de operações básicas, pudemos observar que a relação entre os dois é dada por f(*n*) 1.15 \* (*n* – 1) \* *nn* (isto para o número máxismo de arestas, que é dado por .

Para conseguir prever o tempo para um dado número de vértices calculamos precisams primeiro de descobrir a relação entre o número de operações e o tempo gasto, para isto pudemos definir uma relação t(*n*) = *m* \* f(*n*), onde f(*n*) é o número de operações básicas para um dado número de vértices, t(*n*) o tempo gasto para esse mesmo número de vértices e *a* descrebe a relação entre o tempo gasto e o número de operações. Tendo 4 como o número de vértices, temos que:

0.0004 = *m* \* 820 ⬄ m = 0.0004 / 820 ⬄ m 4.88e-0.7

Com *m* calculado e a função f, pudemos finalmente calcular o tempo gasto para completar o algoritmo um dado grafp *n* número de vértices e com o número máximo de arestas para esse mesmo vértice. Por exemplo, vamos supor que *n* = 100, então o tempo gasto é dado por:

t(100) = 4.88e-0.7 \* f(*100*) 5.55e+195 segundos

Ou seja, o tempo necessário para calcular o número cromático de um grafo com 100 vértices e número máximo de arestas (4950), utilizando pesquisa exaustiva, é de aproximadamente 6.43e+190 dias.

# Conclusão

Com este trabalho pudemos concluir que utilizar algoritmos de pesquisa exaustiva para é inviável para instâncias de um problema com uma dimensão elevada. Por vezes, mesmo para uma instância com uma dimensão razoável, utilizar algoritmos de pesquisa exaustiva é impensável. Para verificar isto basta olhar o exemplo dado neste trabalho: para calcular o número cromático de um grafo com 100 vértices e número máximo de arestas (4950), utilizando pesquisa exaustiva, é de aproximadamente 6.43e+190 dias. Isto são cerca de 1.76e+188 anos; para se ter uma ideia, estima-se que o nosso universo tenha 13.7e+09 anos. Olhando para estes números torna-se bem claro que de facto o tempo que estes algoritmos demoram a executar tornam-nos inviáveis para resolver uma grande parte dos problemas.

# Referências

[1] <https://pt.wikipedia.org/wiki/Teoria_dos_grafos>

[2] <https://pt.wikipedia.org/wiki/Colora%C3%A7%C3%A3o_de_grafos>

[3] <http://www.educ.fc.ul.pt/icm/icm2001/icm33/grafosnaoorientados.htm>

[4] <https://en.wikipedia.org/wiki/Big_O_notation>

[5] <https://matplotlib.org/contents.html>