# Atividade Prática

Análise de Algoritmos

Prof. Paulo Henrique Ribeiro Gabriel

## 1 Algoritmos de Aproximação em Grafos

Algoritmos de aproximação são algoritmos eficientes (em geral) que encontram soluções aproximadas para problemas de otimização, apresentando garantias sobre a distância da solução retornada para a ótima[1]. O projeto e a análise de algoritmos de aproximação envolvem crucialmente uma prova matemática que certifica a qualidade das soluções retornadas no pior caso. Essa característica os distingue das heurísticas, como *hill climbing* ou algoritmos genéticos, que encontram soluções razoavelmente boas em algumas entradas, mas não fornecem nenhuma indicação clara no início sobre quando eles podem ter sucesso ou falhar.

Muitos problemas oriundos da Teoria dos Grafos encontram-se nessa categoria de problemas de otimização complexos. Nesses casos, comumente, são implementados algoritmos gulosos que oferecem alguma garantia sobre a qualidade da solução. Nas próximas seções, dois desses problemas são discutidos.

### 1.1 Coloração de Vértices

O problema de coloração de grafos consiste em atribuir cores a certos elementos de um grafo sujeito a certas restrições. A **coloração de vértices** é o problema de coloração mais comum: dadas $m$ cores, encontre uma maneira de colorir os vértices de um grafo de forma que dois vértices adjacentes não sejam coloridos com a mesma cor. Nesse contexto, o menor número de cores necessárias para colorir um grafo $G$ é chamado de **número cromático**. Por exemplo, o grafo da Figura 1 foi colorido com três cores, que é o o valor mínimo.

O problema para encontrar o número cromático de um dado grafo é NP-Difícil[2], de modo que se faz necessário o projeto de um algoritmo de aproximação para resolvê-lo. O Algoritmo 1 adota a estratégia gulosa para a atribuição de cores. Esse algoritmo não garante o uso do menor número de cores; em vez disso, ele provê um *limite superior* no número de cores[3]. Nesse caso, ele nunca usa mais do que $d+1$ cores, sendo que $d$ é o grau máximo do grafo $G$. No caso do grafo da Figura 1, poderíamos usar até *quatro* cores, pois o maior grau é $d = 3$; no entanto, esse é o valor máximo: o algoritmo pode, sim, encontrar o número exato, ou seja, três.

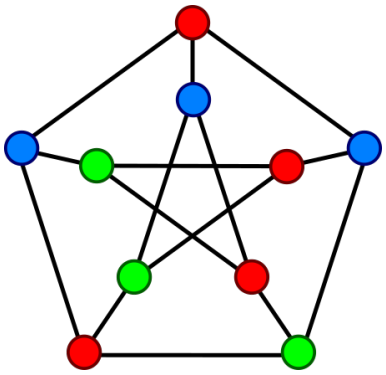

Figura 1: Exemplo de coloração de vértices. (Fonte: Wikipedia.)

**Algoritmo 1:** Algoritmo guloso básico de coloração de vértices.

**Entrada:** Um grafo $G$ e uma lista de cores $C$
**Saída:** Uma coloração aproximada de $G$

```
1 para cada vértice v_i ∈ V(G) faça
2 |   atribua a v_i a primeira cor disponível em C que ainda não foi atribuída a
  |     nenhum de seus vizinhos já coloridos de v_i;
3 fim
```

## 1.2 Problema do Caixeiro Viajante

Dado um conjunto de cidades e a distância entre cada par de cidades, o problema do Caixeiro Viajante consiste em encontrar a rota *mais curta* possível que visite *todas* as cidades exatamente uma vez e retorne ao ponto de partida. Note que o grafo é sempre completo, ou seja, sempre haverá um caminho (hamiltoniano) passando por todas as cidade; o desafio é encontrar o caminho de menor custo, ou seja, aquele que apresenta a menor soma de pesos de arestas. Por exemplo, para o grafo da Figura 2, o caminho de menor custo é $\{1, 2, 4, 3, 1\}$, cujo custo é $10 + 25 + 30 + 15 = 80$.

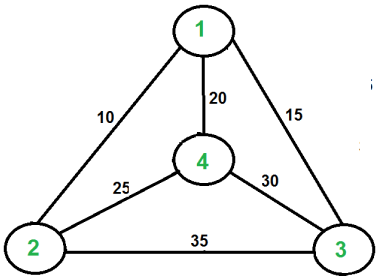


Figura 2: Exemplo de grafo completo. (Fonte: GeeksforGeeks.)

Não existe uma solução conhecida eficiente para este problema[2]. No entanto, existem algoritmos de aproximação. Nesse caso, a única exigência é que as arestas do grafo respeitem a chamada desigualdade triangular, descrita a seguir:

**Desigualdade triangular:** O caminho mais curto para alcançar um vértice $j$ a partir de $i$ é sempre alcançar $j$ diretamente de $i$, ao invés de seguir por algum outro vértice intermediário $k$. Em outras palavras, $dist(i,j)$ é sempre menor ou igual a $dist(i,k) + dist(k,j)$.

No grafo da Figura 2, a desigualdade triangular é respeitada: por exemplo, a menor distância entre os vértices 1 e 2 é 10, que é o peso da aresta que liga esses vértices; qualquer outro caminho (passando por 3 ou por 4) é mais longo.

Quando a função de custo satisfaz a desigualdade triangular, podemos projetar um algoritmo aproximado para o problema do Caixeiro Viajante que encontra um passeio cujo custo nunca é maior do que o dobro do custo de menor passeio possível. Por exemplo, se a solução ótima do problema for 100, nosso algoritmo encontrará um valor no intervalo $[100, 200]$ (note que isso pode incluir o valor ótimo, ou seja, 100, dependendo da estrutura do grafo). A ideia é usar o conceito de árvore geradora mínima do grafo, conforme mostrado no Algoritmo 2.

**Algoritmo 2:** Algoritmo de aproximação para o problema do Caixeiro Viajante.

**Entrada:** Um grafo $G$
**Saída:** Uma passeio de custo aproximadamente mínimo.
1 Seja $v_1$ o vértice inicial e final para o vendedor;
2 Construa a árvore geradora mínima com raiz $v_1$ usando o Algoritmo de Prim;
3 Liste os vértices da árvore resultante em pré-ordem e adicione $v_1$ ao final;

No caso do grafo da Figura 2, teríamos a árvore mostrada na Figura 3, que, percorrida em pré-ordem, nos leva à seguinte sequência de vértices: $1, 2, 4, 3$. Assim, o caminho hamiltoniano gerado será $1, 2, 4, 3, 1$, cujo custo é 80 (por coincidência, o menor custo possível).

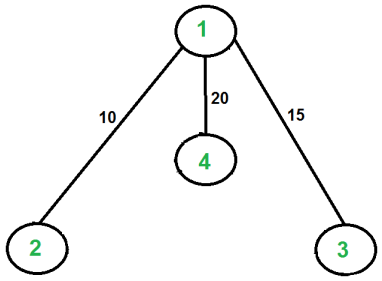


Figura 3: Árvore geradora mínima feita a partir do grafo da Figura 2. (Fonte: GeeksforGeeks.)

## 2 Tarefa

Implemente **ao menos** um dos algoritmos descritos nas seções anteriores. Para isso, você deve gerar um conjunto de grafos (**no mínimo cinco**) e aplicar o algoritmo sobre esses grafos. Em seguida, você deve comparar o resultado do seu algoritmo com o melhor

resultado para o problema em questão. Por exemplo, se você está resolvendo o problema de coloração (Seção 1.1), mostre a coloração obtida por seu algoritmo e compare-a com melhor coloração possível. (Note que ambas podem ser iguais, como já explicado.)

Os grafos podem ser gerados aleatoriamente, ou obtidos de outros trabalhos ou sites (desde que seja citada a fonte). Os grafos gerados devem ter diferentes tamanhos, mas evite grafos muito pequenos (menos de 5 vértices). Meça, também, o tempo de execução do seu algoritmo.

### Observações

Há várias maneiras de medir o tempo de execução de um programa de computador. Muitas linguagens de programação proveem funções que, quando invocadas, retornam o horário do sistema, em *ticks* de relógio da CPU: assim, basta calcular a diferença entre o número de *ticks* depois e antes da execução de cada função e converter o valor para segundos. Pesquise qual a melhor forma de fazer isso para a linguagem de programação escolhida!

**Lembre-se:** quando lidamos com análise experimental, devemos sempre nos preocupar com a confiabilidade do experimento. Em termos gerais, não podemos simplesmente comparar dois algoritmos uma única vez. Devemos replicar o experimento diversas vezes, com diversos valores de entrada. No caso dessa atividade específica, devemos utilizar grafos com diferentes dimensões e executar os algoritmos repetidas vezes para cada uma dessas dimensões. (Geralmente, considera-se que 30 repetições é um bom número — porém, caso o tempo de cada execução esteja muito alto, pode-se reduzir esse valor). Em seguida, compute a média e o desvio-padrão dessas 30 medições; os valores obtidos são utilizados na comparação[1].

## 3 Entrega e Avaliação

O trabalho deve ser desenvolvido em grupos de **dois a três** integrantes. O prazo de conclusão é o dia 07/06/2021. Dúvidas serão discutidas via MS Teams e no Fórum de Dúvidas desta atividade.

A entrega desta atividade deverá ser feita por meio de um link para um repositório (GitHub, Bitbucket, etc.) contendo o código-fonte dos programas e um breve relatório (em PDF) sobre os resultados. O link deverá ser encaminhado via Moodle, na tarefa correspondente a esta atividade. Além disso, cada grupo deverá gravar um breve vídeo, de até 10 minutos, descrevendo brevemente o código-fonte e mostrando ao menos uma execução do programa (discutindo o resultado).

O vídeo deve ser disponibilizado no Microsoft Stream ou no Youtube, como link privado e compartilhado com o professor. **O professor não compartilhará os vídeos com ninguém, exceto com prévia autorização dos autores!** No caso do código-fonte, remova todos os executáveis e demais arquivos gerados pela compilação, se houver. Finalmente, com relação ao relatório, o mesmo deve conter as seguintes informações:

[1] Um ótimo material sobre medição de desempenho de algoritmo foi disponibilizado pelo Prof. Pedro Henrique, da UFMG. Acesse-o neste link.

1. Nome completo e matrícula de todos os integrantes da equipe;
2. Descrição dos experimentos realizados, ou seja:
   (a) Qual linguagem de programação foi adotada e por quê?
   (b) Quais os grafos que foram gerados? Quais as dimensões de cada um? (Mostre as matrizes de adjacências desses grafos.)
   (c) Qual o resultado de cada algoritmo sobre cada grafo?
3. Gráficos e/ou tabelas comparando o desempenho dos dois algoritmos, utilizando valores como média e desvio-padrão. Cada gráfico e tabela deve vir acompanhado de uma breve explicação, discutindo os resultados.

A avaliação deste trabalho levará em consideração a qualidade do relatório e da apresentação. Caso o grupo opte por implementar ambos os problemas, receberá um bônus na nota. Nesse caso, a apresentação pode ser mais longa (até 15 minutos) e deve mostrar os dois algoritmos sendo executados (além de descrever ambos no relatório).

Caso seja detectado plágio, todos os envolvidos receberão nota zero nessa atividade.

# Referências

1 WILLIAMSON, D. P.; SHMOYS, D. B. *The Design of Approximation Algorithms*. Cabridge, MA: Cambridge University Press, 2010. 500 p.

2 KARP, R. M. Reducibility among combinatorial problems. In: MILLER, R. E.; THATCHER, J. W.; BOHLINGER, J. D. (Ed.). *Complexity of Computer Computations*. New York: Plenum, 1972. p. 85–103.

3 MITCHEM, J. On various algorithms for estimating the chromatic number of a graph. *The Computer Journal*, v. 19, n. 2, p. 182–183, 1976.