A B A K Ó S
PUC Minas Instituto de Ciências Exatas e Informática

Análise Comparativa de Métodos para Identificação de Pontes: Abordagens Naïve e Tarjan Aplicadas ao Algoritmo de Fleury*

Comparative Analysis of Methods for Bridge Identification: Naïve and Tarjan Approaches Applied to Fleury's Algorithm

Luca Ferrari Azalim[1]
Fernando Antônio Ferreira Ibrahim[2]

Resumo

O presente trabalho tem como objetivo implementar um algoritmo para a identificação de trajetos e ciclos eulerianos em grafos, com foco na comparação de diferentes métodos para a identificação de pontes. Serão consideradas as implementações dos métodos *naïve* e *Tarjan*, que são fundamentais para a execução do algoritmo de Fleury, escolhido devido à sua popularidade e baixa complexidade computacional. O estudo visa avaliar a eficiência de ambos os métodos na identificação de pontes, elemento crucial para a construção de trajetos e ciclos eulerianos. Espera-se que este projeto contribua para uma melhor compreensão da performance de diferentes abordagens na solução de problemas relacionados à teoria dos grafos e à otimização de rotas.

Palavras-chave: Caminho euleriano, Ciclo euleriano, Algoritmo de *Fleury*, Identificação de pontes, Teoria dos grafos, Comparação de métodos.

*Artigo apresentado à Revista Abakos
[1]Graduação em Engenharia de Software PUC Minas, Brasil– lucaazalim@gmail.com
[2]Graduação em Engenharia de Software PUC Minas, Brasil– fernandofibrahim@gmail.com

Abstract

This paper aims to implement an algorithm for identifying Eulerian paths and cycles in graphs, with an emphasis on comparing different methods for bridge identification. The *naïve* and *Tarjan* methods will be considered, as they are key components for the execution of *Fleury's* algorithm, chosen for its popularity and low computational complexity. The study seeks to evaluate the efficiency of both methods in identifying bridges, a crucial element for constructing Eulerian paths and cycles. It is expected that this project will contribute to a better understanding of the performance of different approaches in solving graph theory problems and optimizing routes.

Keywords: Eulerian path, Eulerian cycle, Fleury's algorithm, Bridge identification, Graph theory, Method comparison.

1 INTRODUÇÃO

Grafos são estruturas matemáticas amplamente utilizadas para modelar diversos problemas do mundo real. Entre as várias aplicações possíveis, um dos problemas clássicos é a identificação de ciclos e trajetos eulerianos. Um trajeto euleriano é um trajeto que percorre todas as arestas de um grafo exatamente uma vez, enquanto um ciclo euleriano retorna ao ponto inicial após percorrer todas as arestas de forma única.

O objetivo deste projeto é comparar a eficiência de dois métodos distintos de identificação de pontes em um grafo, uma etapa crucial na aplicação do algoritmo de *Fleury* para encontrar ciclos eulerianos. Os métodos em questão são o método *naïve* e o método *de Tarjan*, ambos com abordagens diferentes para detectar as arestas críticas que, se removidas, desconectariam o grafo.

A análise de desempenho será conduzida em três categorias de grafos: **eulerianos**, **semi-eulerianos** e **não eulerianos**. O foco principal será comparar os tempos de execução dos métodos *naïve* e *de Tarjan* na identificação de ciclos eulerianos, a fim de determinar qual abordagem oferece melhor eficiência na identificação de pontes e, consequentemente, na resolução do problema dos ciclos eulerianos.

Os resultados dessa comparação fornecerão uma base sólida para a escolha do método mais eficiente em diferentes cenários, o que possibilitará futuras aplicações em otimização de rotas e outras áreas que dependem de grafos e algoritmos de trajeto.

2 IMPLEMENTAÇÃO

2.1 Linguagem

A linguagem de programação Java foi escolhida para a implementação e execução da solução proposta devido à sua ampla utilização, à vasta quantidade de código reutilizável disponível e ao conhecimento prévio dos integrantes do grupo, o que facilitou o desenvolvimento da solução.

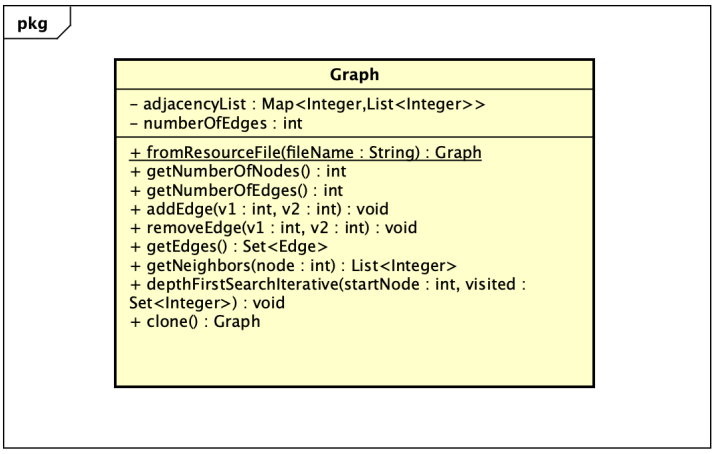
2.2 Representação do Grafo

No código desenvolvido, o grafo é representado por meio de uma classe chamada *Graph*, utilizando uma estrutura de dados do tipo `HashMap<Integer, List<Integer> >`, que funciona como uma lista de adjacência. Nesta estrutura, cada chave do mapa corresponde a um vértice do grafo, enquanto o valor associado é uma lista de inteiros que indica os vértices adjacentes a esse vértice. Essa representação permite acessar rapidamente os vizinhos de qualquer

vértice, facilitando operações e algoritmos de busca e navegação no grafo.

Tal fato pode ser vizualizado na figura 1.

Figura 1 – Classe Graph representada através de um diagarama de classes



2.3 Identificação de Trajetos e Ciclos Eulerianos

Além da estrutura do grafo, foi criada a classe `FleuryEulerTourFinder`, que é responsável por receber o grafo desejado e identificar possíveis trajetos ou ciclos eulerianos. A identificação é realizada utilizando um dos métodos de identificação de pontes, que é passado como um objeto do tipo `BridgeIdentifier` ao método construtor da classe, junto ao grafo desejado.

Inicialmente, a classe `FleuryEulerTourFinder` faz um clone do grafo enviado por parâmetro, sendo esse novo grafo um auxiliar, o qual terá as suas arestas removidas à medida que o trajeto euleriano é encontrado.

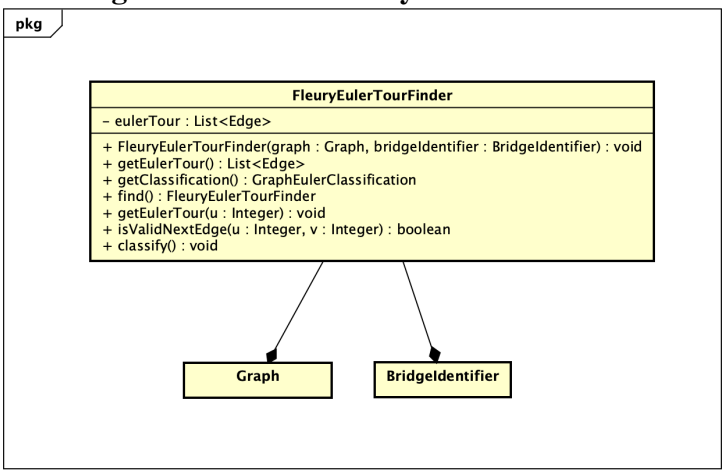
Em seguida, é executado o método `find()`, o qual dá início à busca pelo trajeto euleriano. O processo começa ao identificar um vértice inicial (`u`), sempre dando preferencia por vértices de grau impar, uma vez que o trajeto semi-euleriano precisa começar em um vértice de grau ímpar. O algoritmo, então, percorre o grafo, removendo arestas enquanto constrói o trajeto em `eulerTour`. Para realizar esse processo, chama o método auxiliar `getEulerTour(Integer)`, que explora o grafo a partir do vértice inicial e encontra o trajeto euleriano.

Durante o percurso, cada aresta é verificada pelo método `isBridge()`, da classe BridgeIdentifier, que determina se a aresta atual é uma ponte e, portanto, se sua remoção comprometeria a conectividade do grafo.

A classe também conta com o método `classify()`, responsável por verificar se o grafo é euleriano, semi-euleriano ou não eureleriano.

Tal fato pode ser vizualizado na figura 2.

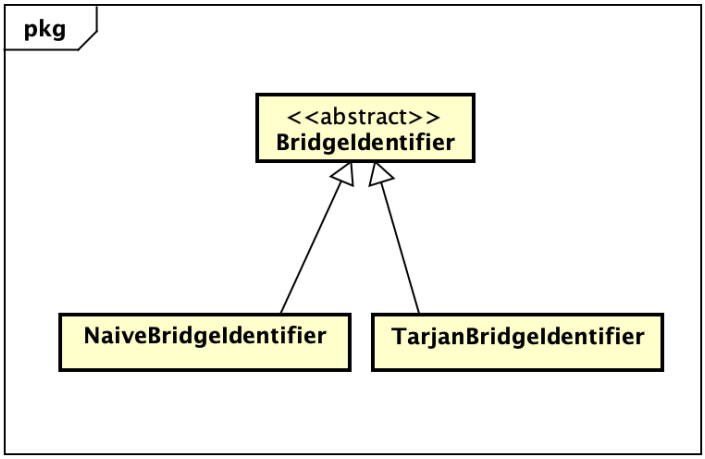
Figura 2 – Classe FleuryEulerTourFinder



2.4 Identificação de Pontes

As classes responsáveis pela identificação das pontes implementam a classe abstrata `BridgeIdentifier`, permitindo a fácil troca entre diferentes abordagens para a identificação de pontes, como os métodos *Naïve* e *Tarjan*, sem a necessidade de modificações no código. Tal fato pode ser visualizado na figura 3.

Figura 3 – Hierarquia de Classes BridgeIdentifier



2.4.1 Naïve

A classe `NaiveBridgeIdentifier` identifica pontes no grafo utilizando o método `isBridge()`. Esse método realiza a identificação de pontes removendo temporariamente a aresta selecionada e, em seguida, chamando o método `isConnected()` no grafo. O método `isConnected()` executa uma busca em profundidade (DFS) para verificar se todos os vértices ainda podem ser alcançados, mesmo com a remoção da aresta. Caso algum vértice fique inacessível, a aresta removida é identificada como uma ponte, pois sua ausência desconecta o grafo.

2.4.2 Tarjan

O algoritmo implementado na classe `TarjanBridgeIdentifier` visa identificar as pontes de um grafo utilizando a técnica de busca em profundidade (DFS) modificada.

A função principal, `isBridge()`, recebe um grafo e uma aresta como entrada. Para verificar se essa aresta é uma ponte, o algoritmo executa uma busca DFS começando a partir do vértice inicial da aresta. Durante a execução da DFS, o algoritmo mantém dois arrays auxiliares: `disc[]`, que armazena o tempo de descoberta de cada vértice, e `low[]`, que armazena o menor tempo de descoberta de um vértice alcançável a partir do vértice atual ou de seus descendentes.

Se, ao visitar um vértice v, for encontrado um vértice u (vértice pai de v) tal que `low[v] > disc[u]`, então a aresta entre u e v é identificada como uma ponte, pois a remoção dessa aresta desconectaria o grafo.

O algoritmo usa duas pilhas para realizar a DFS de forma iterativa: uma pilha para armazenar os vértices a serem visitados (`stack`) e outra para armazenar os iteradores dos vizinhos de cada vértice (`adjStack`).

2.5 Testes

Para a realização dos testes e obtenção dos resultados, foram gerados automaticamente diferentes grafos não direcionados, incluindo grafos Eulerianos, semi-Eulerianos e não Eulerianos, com tamanhos variando entre 100, 1.000, 10.000 e 100.000 vértices. Essa variação permitiu uma avaliação abrangente dos resultados dos métodos de identificação de pontes em grafos, possibilitando uma análise comparativa de desempenho entre as abordagens.

A criação dos grafos foi automatizada através do arquivo `generate_graphs.py`. O código gera os três tipos de grafos: um ciclo fechado para grafos Eulerianos, um caminho com duas arestas extras para semi-Eulerianos e um caminho com múltiplas arestas extras para não Eulerianos. Cada grafo gerado é salvo em arquivos de texto, com nomes que indicam seu tipo e tamanho.

3 RESULTADOS

Para a obtenção dos resultados, o método *Fleury* foi executado cinco vezes para cada grafo criado, com o tempo de execução de cada tentativa registrado em milissegundos. Em seguida, calcular-se-á a média aritmética desses tempos, proporcionando uma estimativa mais precisa do desempenho. Esse procedimento foi realizado utilizando dois métodos distintos para a identificação de pontes: o método *naïve* e o método *Tarjan*.

Após a execução de ambos os métodos, os tempos médios de execução serão compara-

dos, permitindo uma análise detalhada das diferenças de eficiência e desempenho entre as duas abordagens.

Tabela 1 – Tempo Médio de Execução dos Grafos Eulerianos Usando o Método Naïve para Identificação de Pontes

Grafo	Tempo Total (ms)	Média Aritmética (ms)
eulerian_100_graph	46	9
eulerian_1000_graph	566	113
eulerian_10000_graph	116610	23322
eulerian_100000_graph	5x	x > 7200000

Fonte: Dados obtidos pelo autor

Tabela 2 – Tempo Médio de execução dos grafos Eulerianos utilizando o método de Tarjan para identificação de pontes

Grafo	Tempo Total (ms)	Média Aritmética (ms)
eulerian_100_graph	40	8
eulerian_1000_graph	390	78
eulerian_10000_graph	25252	5050
eulerian_100000_graph	26262975	5252595

Fonte: Dados obtidos pelo autor

Tabela 3 – Tempo Médio de Execução dos Grafos Semi-Eulerianos Utilizando o Método Naïve para Identificação de Pontes

Grafo	Tempo Total (ms)	Média Aritmética (ms)
semi_eulerian_100_graph	44	8
semi_eulerian_1000_graph	844	168
semi_eulerian_10000_graph	127842	25380
semi_eulerian_100000_graph	5x	x > 7200000

Fonte: Dados obtidos pelo autor

Os resultados apresentados nas Tabelas 1, 2, 3, 4, 5, e 6 de tempo médio de execução mostram uma clara diferença de desempenho entre os métodos *Naïve* e *Tarjan* para a identificação de pontes em grafos Eulerianos, Semi-Eulerianos, e Não-Eulerianos. De maneira geral, o método *Tarjan* se mostrou mais eficiente em todas as categorias de grafos, apresentando tempos de execução consistentemente menores em comparação ao método *Naïve*. Por exemplo, no caso dos grafos `eulerian_100000_graph`, o método de *Tarjan* levou apenas 5050 ms, enquanto o método *Naïve* exigiu 23322 ms, conforme ilustrado nas Tabelas 1 e 2.

A razão para essa diferença de desempenho está diretamente relacionada às complexidades de ambos os métodos. O *Método Naïve* apresenta uma complexidade de **O(V * E)**, o que implica que, à medida que o grafo aumenta em tamanho, o tempo de execução cresce de maneira significativamente mais rápida. Isso ocorre porque o algoritmo realiza uma busca para cada aresta, o que se torna muito custoso para grafos grandes.

Em contraste, o *Método de Tarjan*, com complexidade **O(V + E)**, realiza a identificação de pontes em uma única varredura do grafo, sendo linear em relação ao número de vértices e arestas. Isso explica a superioridade do método *Tarjan*, especialmente para grafos com 10.000 e

Tabela 4 – Tempo Médio de Execução dos Grafos Semi-Eulerianos Usando o Método Tarjan para Identificação de Pontes

Grafo	Tempo Total (ms)	Média Aritmética (ms)
semi_eulerian_100_graph	42	8
semi_eulerian_1000_graph	378	75
semi_eulerian_10000_graph	25568	5113
semi_eulerian_100000_graph	26142521	5228504

Fonte: Dados obtidos pelo autor

Tabela 5 – Tempo Médio de Execução dos Grafos Não-Eulerianos Usando o Método Naïve para Identificação de Pontes

Grafo	Tempo Total (ms)	Média Aritmética (ms)
non_eulerian_100_graph	49	9
non_eulerian_1000_graph	840	168
non_eulerian_10000_graph	101136	20227
non_eulerian_100000_graph	5x	x > 7200000

Fonte: Dados obtidos pelo autor

100.000 vértices, onde as diferenças de tempo se tornam evidentes, como mostrado nas Tabelas 1, 2, 3, 4, 5, e 6.

CONCLUSÃO

Este estudo comparou os métodos *naïve* e *Tarjan* para a identificação de pontes em grafos, etapa essencial na execução do algoritmo de Fleury para encontrar trajetos e ciclos eulerianos. Os resultados evidenciam que o algoritmo de *Tarjan*, com complexidade $O(V+E)$, é significativamente mais eficiente que a abordagem *naïve*, que opera em $O(V \times E)$. Para grafos maiores, o algoritmo de *Tarjan* demonstrou maior praticidade, destacando a importância de algoritmos otimizados em aplicações da teoria dos grafos, como a otimização de rotas e análise de redes. Esta pesquisa reforça a necessidade de métodos eficientes na resolução de problemas baseados em grafos em diversos domínios.

Tabela 6 – Tempo Médio de Execução dos Grafos Não-Eulerianos Usando o Método de Tarjan para Identificação de Pontes

Grafo	Tempo Total (ms)	Média Aritmética (ms)
non_eulerian_100_graph	43	8
non_eulerian_1000_graph	436	87
non_eulerian_10000_graph	25829	5165
non_eulerian_100000_graph	30058036	6011607

Fonte: Dados obtidos pelo autor

Referências