Trabalho Prático N.01

Teoria dos Grafos - ICEI - PUC Minas

Pedro Henrique Moreira Caixeta Ferreira[1]
Gabriel Vitor de Oliveiro Morais[2]
Júlia Borges Araújo Silva[3]

Resumo

Este relatório apresenta a execução bem-sucedida do Trabalho Prático 1 em Teoria dos Grafos e Computabilidade, realizado no curso de Engenharia de Software da PUC Minas. Desenvolvemos dois programas para identificar pontes em um grafo: um método ingênuo e o método de Tarjan (1974). Adicionalmente, implementamos a identificação de caminhos eulerianos usando o método de Fleury. Realizamos experimentos abrangentes em grafos aleatórios, avaliando o desempenho para diferentes tamanhos, de 100 a 100.000 vértices. O relatório, formatado em TeX conforme as diretrizes, detalha implementações, experimentos e resultados.

Abstract

This report outlines the successful completion of Practical Work 1 in Graph Theory and Computability within the Software Engineering program at PUC Minas. We developed two programs to identify bridges in a graph: a naive method and Tarjan's method (1974). Additionally, we implemented the identification of Eulerian paths using Fleury's method. Comprehensive experiments were conducted on random graphs, assessing performance across various sizes, ranging from 100 to 100,000 vertices. The report, formatted in TeX as per the guidelines, details implementations, experiments, and results.

[1]Estudante bacharel em Engenharia de Software, Brasil– phmcf13@gmail.com
[2]Estudante bacharel em Engenharia de Software, Brasil– gabrielvitor0309@gmail.com
[3]Estudante bacharel em Engenharia de Software, Brasil– juliaborgesas01@gmail.com

1 INTRODUÇÃO

Este relatório descreve o desenvolvimento de um programa em Java para a análise de grafos não-direcionados. O programa implementa duas estratégias distintas para identificar pontes em um grafo: o Método Naive e o Método de Tarjan. Além disso, o software é capaz de encontrar um caminho euleriano em um grafo utilizando o Método de Fleury.

O estudo de grafos e algoritmos associados é fundamental em diversas áreas, proporcionando soluções para uma variedade de problemas. Neste contexto, a identificação de pontes em um grafo é um problema relevante, com aplicações em diversas situações práticas. A ponte, definida como uma aresta cuja remoção torna o grafo desconexo, é crucial para entender a conectividade e a estrutura do grafo.

O objetivo principal deste projeto é realizar experimentos e avaliar o desempenho das estratégias implementadas em grafos aleatórios com diferentes números de vértices (100, 1.000, 10.000 e 100.000). Os métodos serão avaliados em termos de eficiência temporal, proporcionando insights sobre a complexidade de cada abordagem.

O relatório aborda os desafios enfrentados durante o desenvolvimento, as soluções adotadas, bem como as ferramentas e tecnologias utilizadas. O código fonte do programa está em conformidade com as boas práticas de programação e é apresentado de forma clara e estruturada.

2 DESENVOLVIMENTO

O programa desenvolvido utiliza as implementações dos métodos Naive e de Tarjan para encontrar pontes em um grafo. Além disso, a abordagem de Fleury é empregada para determinar um caminho euleriano no grafo, contribuindo para a compreensão da estrutura e da conectividade do mesmo.

Durante o desenvolvimento do software, foram conduzidas reuniões para compreender os requisitos. Tres programas, "Naive", "Tarjan"e "Fleury", foram desenvolvidos separadamente. O "Naive"é um algoritmo que tem como objetivo classificar os dados, ja o tarjan é usado para resolver o problema das componentes fortes em um grafo e o fleury é utilizado para a construção ou identificação de um ciclo euleriano em um grafo euleriano.

A implementação foi influenciada por orientações da monitoria da PUC-Minas, insights do ChatGPT e fontes de pesquisa como stackoverflow para busca de implementação dos métodos. Essas fontes foram cruciais para esclarecer dúvidas e guiar o desenvolvimento.

A integração dos programas no código fornecido resultou em uma solução abrangente. O tempo de execução de cada método foi registrado para avaliar o desempenho do software, podendo assim avaliar seu desempenho máximo na hora de executar o sistema e compreender todos os paradigmas de quando se constrói um programa.

2.1 Métodos Implementados

Método Naive para encontrar pontes (bridgeNaive):

Este método utiliza força bruta para encontrar pontes em um grafo. Itera sobre todas as arestas uma por uma, remove cada aresta e verifica se a remoção desconecta o grafo. Se sim, a aresta removida é uma ponte. Esse método possui uma complexidade alta, principalmente para grafos grandes.

Método de Tarjan para encontrar pontes (bridgeTarjan):

Este método utiliza o algoritmo de Tarjan para encontrar pontes em um grafo. O algoritmo de Tarjan é mais eficiente do que o método Naive. Ele mantém informações sobre os tempos de descoberta e o tempo de término de cada vértice durante uma busca em profundidade. A análise desses tempos ajuda a identificar as pontes no grafo.

Método de Fleury para encontrar um caminho euleriano (fleury):

Este método implementa o algoritmo de Fleury para encontrar um caminho euleriano em um grafo. O caminho euleriano visita cada aresta do grafo exatamente uma vez. O algoritmo escolhe arestas de forma cuidadosa para garantir que o caminho seja euleriano, sempre escolhendo arestas não pontes enquanto estiverem disponíveis. Este método é adequado apenas para grafos que atendem aos critérios de Euler..

2.2 Requisitos

Implementação dos Métodos:

- Implementar dois métodos para identificação de pontes em um grafo simples não-direcionado G = (V, E):
- Método Naive: Testar a conectividade após a remoção de cada aresta.
- Método de Tarjan: Utilizar o algoritmo proposto por Tarjan (1974) para encontrar pontes.
- Encontrar Caminho Euleriano: Descrição do método para encontrar um Caminho Euleriano.

Encontrar Caminho Euleriano:

- Implementar o método de Fleury para encontrar um caminho euleriano em um grafo.
- Aplicar o método de Fleury para cada uma das estratégias de identificação de pontes (Naive e Tarjan).

Experimentos:

- Realizar experimentos para avaliar o tempo médio gasto pelas duas estratégias (Naive e Tarjan) aplicadas a grafos aleatórios.

- Utilizar grafos eulerianos, semi-eulerianos e não eulerianos. Os grafos devem ter 100, 1.000, 10.000 e 100.000 vértices..

3 CÓDIGOS IMPLEMENTADOS

Elementos inseridos no texto como imagens, tabelas, algoritmos etc. Recomenda-se a colocação das ilustrações de forma centralizada, dentro das margens. Caso não seja possível, em **??**) recomenda-se utilizar recursos como: a) utilizar letras com tamanho menor ao padrão do texto; a) imprimir a ilustração no sentido vertical; c) imprimir em folha A3 ou superior e dobrá-la até atingir o tamanho da folha A4.

Nas normas da PUC é afirmado a necessidade de se observar que todos os elementos flutuantes inseridos devem ter a formatação básica:

a) Título centralizado localizado na parte superior;

a) Fonte em tamanho 10 na parte inferior;

c) Devem ser inseridas o mais próximos do texto que as referenciam.

3.1 Método Naive

Figura 1 – Método implementado correspondente ao Método Naive

```java
// Método Naive para encontrar pontes
public void bridgeNaive() {
    // Percorre todas as arestas uma por uma
    for (int u = 0; u < V; u++) {
        for (int v : adj[u]) {
            // Remove aresta da lista de adjacências
            adj[u].remove((Integer) v);
            adj[v].remove((Integer) u);

            // Verifica se o grafo ainda é conexo
            boolean[] visited = new boolean[V];
            int numVisited = 0;
            LinkedList<Integer> queue = new LinkedList<Integer>();
            queue.add((u + 1) % V); // Começa a busca em um vértice diferente de u
            visited[(u + 1) % V] = true;
            while (!queue.isEmpty()) {
                int s = queue.poll();
                numVisited++;
                for (int i : adj[s]) {
                    if (!visited[i]) {
                        visited[i] = true;
                        queue.add(i);
                    }
                }
            }

            // Se o número de vértices visitados é menor que V, então a aresta é uma ponte
            if (numVisited < V)
                System.out.println(u + " " + v);

            // Adiciona a aresta de volta à lista de adjacências
            adj[u].add(v);
            adj[v].add(u);
        }
    }
}
```

Este trecho de código implementa o método Naive para encontrar pontes em um grafo não-direcionado. O método percorre todas as arestas do grafo, e para cada aresta (u, v), remove temporariamente essa aresta do grafo, verifica se a remoção torna o grafo desconexo e, se sim, imprime essa aresta como uma ponte. Em seguida, a aresta é adicionada de volta ao grafo para que o próximo par de vértices seja analisado.

3.1.1 Análise de Complexidade

A complexidade do código *bridgeNaive* é elevada, e pode ser aproximadamente $O(V^4)$, onde V é o número de vértices do grafo.

3.2 Tarjan

O método bridgeTarjan implementa o algoritmo de Tarjan para encontrar pontes em um grafo não-direcionado. Esse algoritmo utiliza a busca em profundidade (DFS) para explorar as arestas do grafo e identificar as pontes. Aqui está uma explicação detalhada do código:

Figura 2 – Implementação do Código Tarjan

```
public void bridgeTarjan() {
    boolean visited[] = new boolean[V];
    int disc[] = new int[V];
    int low[] = new int[V];
    int parent[] = new int[V];

    // Inicializa os arrays de visitados e pais
    for (int i = 0; i < V; i++) {
        parent[i] = -1;
        visited[i] = false;
    }

    // Chama a função recursiva para encontrar pontes
    for (int i = 0; i < V; i++)
        if (visited[i] == false)
            bridgeTarjanUtil(i, visited, disc, low, parent);
}
```

3.2.1 Análise de Complexidade

A ordem de complexidade do algoritmo de Tarjan para encontrar pontes em um grafo é $O(V + E)$, onde V é o número de vértices e E é o número de arestas.

3.3 Fleury

O método fleury implementa o algoritmo de Fleury para encontrar um caminho euleriano em um grafo. Um caminho euleriano é um caminho que passa por todas as arestas de um grafo exatamente uma vez.

Figura 4 - Implementação código Fleury

```java
// Método de Fleury para encontrar um caminho euleriano
public void fleury(int start) {
    // Cria uma pilha para armazenar o caminho euleriano
    Stack<Integer> stack = new Stack<>();
    stack.push(start);

    // Cria uma lista para armazenar o caminho euleriano final
    List<Integer> path = new ArrayList<>();

    while (!stack.isEmpty()) {
        int current = stack.peek();
        if (adj[current].size() > 0) {
            // Se o vértice atual tem arestas, remove a primeira aresta e empilha o vértice
            // adjacente
            int next = adj[current].get(index:0);
            adj[current].remove((Integer) next);
            adj[next].remove((Integer) current);
            stack.push(next);
        } else {
            // Se o vértice atual não tem arestas, adiciona-o ao caminho euleriano e
            // remove-o da pilha
            path.add(current);
            stack.pop();
        }
    }

    // Imprime o caminho euleriano
    for (int i : path) {
        System.out.print(i + " ");
    }
    System.out.println();
}
```

3.3.1 Análise de Complexidade

A complexidade do algoritmo de Fleury para encontrar um caminho euleriano em um grafo é $O(E^2)$, onde E é o número de arestas no grafo.

4 TESTES

Atendendo à orientação do professor, procedemos com experimentos para analisar a média de tempo despendida pelos dois softwares em grafos de caráter aleatório, variando de 100 a 100.000 vértices. Importante destacar que, devido à ausência de requisitos específicos quanto ao número de arestas nos grafos de teste, o número de arestas é equiparado ao número de vértices, acrescido de um, para todos os cenários de teste mencionados.

Tabela 1 – Testes do Método Naive

Dados de Entrada	Ações Executadas	Tempo Médio
100	Método Naive aplicado a grafos com 100 vértices.	1:22:18
1000	Método Naive aplicado a grafos com 1000 vértices.	1:22:17
10000	Método Naive aplicado a grafos com 10000 vértices.	1:24:25
100000	Método Naive aplicado a grafos com 100000 vértices.	∞

4.1 Método Naive

4.2 Método Tarjan

Tabela 2 – Testes do Método Tarjan

Dados de Entrada	Ações Executadas	Tempo Médio
100	Método Tarjan aplicado a grafos com 100 vértices.	1327200ns
1000	Método Tarjan aplicado a grafos com 1000 vértices.	37.812.5003ns
10000	Método Tarjan aplicado a grafos com 10000 vértices.	2.3s
100000	Método Tarjan aplicado a grafos com 100000 vértices.	∞

4.3 Método Fleury

Tabela 3 – Testes do Método Fleury

Dados de Entrada	Ações Executadas	Tempo Médio
100	Método Fleury aplicado a grafos com 100 vértices.	138002ns
1000	Método Fleury aplicado a grafos com 1000 vértices.	69300ns
10000	Método Fleury aplicado a grafos com 10000 vértices.	1612900ns
100000	Método Fleury aplicado a grafos com 100000 vértices.	∞

4.4 Conclusão dos Testes

Os testes realizados evidenciam que o Método Naive apresenta um desempenho insatisfatório, tornando-se inviável para grafos de maior escala, como aqueles com 10000 vértices ou mais, onde o tempo de execução torna-se impraticável ∞.

Em contrapartida, o Método de Tarjan demonstra uma eficiência notável, mesmo em grafos mais extensos. O tempo de execução permanece razoável, escalando de maneira controlada.

Quanto ao Método de Fleury, sua aplicabilidade é restrita a grafos eulerianos. Para grafos que atendem a essa condição, o método exibe um desempenho satisfatório, destacando-se pela rapidez na identificação de caminhos eulerianos.

Conclui-se que, ao escolher um método para análise de grafos, é crucial considerar não apenas a corretude, mas também a eficiência, especialmente em cenários de grandes volumes de dados. O Método de Tarjan emerge como uma opção robusta e escalável, enquanto o Método Naive mostra-se impraticável para grafos extensos. O Método de Fleury destaca-se quando aplicável, oferecendo eficiência na identificação de caminhos eulerianos.

5 CONCLUSÃO

O relatório apresenta um programa em Java para análisar grafos não-direcionados, focando na identificação de pontes usando Métodos Naive e de Tarjan, além da busca de caminhos eulerianos com o Método de Fleury. Os métodos foram implementados e testados em grafos de tamanhos variados.

O Método Naive, apesar de simples, tem complexidade alta, tornando-se menos eficiente para grafos grandes. O Método de Tarjan, mais eficiente, mostra-se adequado para grafos maiores. O Método de Fleury é destacado na busca de caminhos eulerianos.

Os testes revelam as diferenças temporais entre os métodos, evidenciando a eficácia do Método de Tarjan em grafos maiores. A escolha do método dependerá das características específicas do problema. O artigo contribui para a compreensão prática desses métodos, sugerindo otimizações futuras.

Referências

BARTHA, Miklos; KRESZ, Miklos. A depth-first algorithm to reduce graphs in linear time. In: **2009 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing**. [S.l.: s.n.], 2009. p. 273–281.

CHAITANYA, Meher; KOTHAPALLI, Kishore. A simple parallel algorithm for biconnected components in sparse graphs. In: **2015 IEEE International Parallel and Distributed Processing Symposium Workshop**. [S.l.: s.n.], 2015. p. 395–404.

KIM, Taeheung; LEE, Jong-Seok. Exponential loss minimization for learning weighted naive bayes classifiers. **IEEE Access**, v. 10, p. 22724–22736, 2022.

ZHANG, Guangxiao; TONG, Xiaoyang. The last circuit breakers identification in hybrid ac/dc power grids based on improved tarjan algorithm. **IEEE Transactions on Power Delivery**, v. 35, n. 6, p. 2992–3002, 2020.

(KIM; LEE, 2022) (ZHANG; TONG, 2020) (BARTHA; KRESZ, 2009) (CHAITANYA; KOTHAPALLI, 2015)