Trabalho Prático 1

José Fernando Rossi, Victor Colen

[1]Instituto de Ciências Exatas e Informática (ICEI)
Pontifícia Universidade Católica (PUC Minas) – Belo Horizonte, MG – Brazil

1. Introdução

Este trabalho tem o objetivo de comparar algoritmos que acham pontes em grafos conexos, simples e não direcionados, além de computar o tempo para identificação de um caminho eureliano. Dito isso, define-se como ponte uma aresta e de um grafo $G(v, e)$ em que v são os vértices e e as arestas, que aumenta o número de componentes conexos do grafo. Já sobre o caminho eureliano, é um caminho que passa por todas as arestas de um grafo exatamente uma vez.
Dito isso, o trabalho foi desenvolvido na linguagem *Java - versão 17*, compilado e executado em um computador com 24Gb de RAM e processador de 2.1GHz.

2. Desenvolvimento

Assim, este trabalho consiste em desenvolver uma análise dos algoritmos de grafos, avaliando o tempo médio gasto nos métodos que serão apresentados a seguir em cinco grafos aleatórios (eurelianos, semi-euleriano e não-eurelianos), contendo 100, 1.000, 10.000 e 100.000 vértices. Além disso, foi definido o tempo máximo de execução para cada algoritmo de 30 minutos, caso o algoritmo exceda esse limite, o tempo de execução será definido como *Não definido*. Já em relação ao consumo de memória, foi disponibilizado um total de 2.5Gb de memória para a execução de cada algoritmo, caso ultrapasse esse limite, a execução será encerrada. A seguir, estão descritos os métodos desenvolvidos.

2.1. Algoritmo Naive

O algoritmo *Naive* é responsável pela identificação de pontes em um grafo conexo, simples e não direcionado. A implementação deste algoritmo foi feita de uma maneira caracterizada como *Força bruta*, o que significa que, basicamente, será retirada uma aresta e testará a conectividade do grafo, caso haja o aumento do número de componentes no grafo, esta aresta será uma aresta de ponte. Em **Table 1**, apresentada a seguir, observa-se os *Tempos de execução* em comparação à *Quantidade de vértices* para o algoritmo *Naive*:

Quantidade de vértices	**Tempo médio de execução**
100	$263ms$
1000	$49.985s$
10000	*Não definido*
100000	*Não definido*

Table 1. Tempo de execução por quantidade de vértices - Naive

O custo, em geral, do algoritmo implementado é $\mathcal{O}(n^4)$, já que ele percorre todas as

arestas com custo $\mathcal{O}(n+m)$, o que, em linhas gerais pode ser escrito como $\mathcal{O}(n^2)$, multiplicado pelo custo de identificação do número de componentes conexos, o qual é $\mathcal{O}(n^2)$, justificado pelo fato de ser realizado uma busca em profundida com custo $\mathcal{O}(n+m)$, ou seja, $\mathcal{O}(n^2)$.

2.2. Implementação do algoritmo de Tarjan

O método de Tarjan [Tarjan 1974] também foi utilizado para identificar pontes exitentes no grafo, porém sua implementação difere do método Naive. Sobre o custo do método de Tarjan, observa-se que o mesmo tem custo igual a $\mathcal{O}(n+m)$, podendo ser reescrito como $\mathcal{O}(n^2)$. Na **Table 2** observa-se os tempos médios de execução de acordo com a quantidade de vértices.

Quantidade de vértices	**Tempo médio de execução**
100	$8ms$
1000	$29ms$
10000	$257ms$
100000	$3.016s$

Table 2. Tempo de execução por quantidade de vértices - Tarjan

2.3. Implementação do Método de Fleury

Por fim, a implementação do método de Fleury foi desenvolvido para encontrar um caminho euleriano no grafo, ou seja, um caminho em que visita-se todas as arestas apenas uma vez, junto com as outras implementações mostradas acima. A seguir, observa-se a **Table 3**, a qual mostra os tempos de execução por quantidade de vértices do algoritmo Fleury. Percebe-se que o custo do algoritmo é alto, sendo $\mathcal{O}((v+e)^2)$, o que pode ser reescrito em linhas gerais como $\mathcal{O}(n^4)$.

Quantidade de vértices	**Tempo médio de execução**
100	$26ms$
1000	*Não definido*
10000	*Não definido*
100000	*Não definido*

Table 3. Tempo de execução por quantidade de vértices - Fleury

No caso da implementação original do algoritmo de Fleury, houve um estouro de memória em grafos com o número de vértices maiores que 100, já que o limite de 2.5Gb foi excedido, gerando um erro de *StackOverFlow*.

3. Conclusão

A partir da análise dos resultados, conclui-se que há uma diferença expressiva nos métodos para identificação de pontes. O método de força bruta (naive) não consegue executar em tempo hábil a identificação de pontes em grafos maiores que 1000 vértices, enquanto o método de *Tarjan* executa de maneira eficiente, com tempo médio de execução em um grafo de 100000 vértices igual a, aproximadamente, 3 segundos. Dito isso,

percebe-se que, em termos temporais, é melhor implementar este método. Porém em termos de legibilidade e manutenção de código, o força bruta é extremamente mais simples e intuitivo, já que consiste em apenas testar a conectividade a partir da remoção de cada aresta.

Agora, sobre o método de Fleury, observa-se que é um método extremamente ineficiente, tanto em termos de complexidade, quanto em termos de memória, já que consome muita memória e gasta um longo tempo de execução. Além disso, para os experimentos realizados, tal método não conseguiu completar a execução para grafos com número de vértices maior que 100, gerando um estouro de pilha, ou seja, utilização de mais memória do que há disponível.

References

[Tarjan 1974] Tarjan, R. (1974). A note on finding the bridges of a graph. *Information Processing Letters*.