Documentação - Trabalho Prático 1
Teoria dos Grafos e Computabilidade

Vinícius Henrique Giovanini

[1]ICEI – Pontifícia Universidade Católica de Minas Gerais (PUC)
Belo Horizonte – MG – Brazil

{vgiovanini}@sga.pucminas.br

Resumo. *Desenvolvimento de grafos Eulerianos, Semi Eulerianos e Não Eulerianos, com a linguagem de programação* ***Python****, com objetivo de fazer sua identificação e também fazer a pesquisa no grafo, e para isso foi necessário implementar algoritmos para identificação de pontes.*

1. Introdução

Inicialmente, grafos Eulerianos são grafos em que todos os seus vértices possuem **grau par**, Semi Euleriano é caracterizado caso possua um limite de dois vértices com **grau impar**, e um grafo não euleriano consiste em não atender essas duas regras, então seria um grafo com mais de dois vértices com **grau impar**. O método para fazer o **caminho** ou **trajeto** nos grafos gerados consiste no Naive, e ele testa a conectividade removendo a aresta percorrida, e verificando a existência de pontes.

2. Estrutura dos Códigos

O projeto foi realizado utilizando três classes, a classe principal chamada **core**, a classe para Geração de Grafos foi chamada de **gerarGrafos**, a classe do Fleury que utiliza o Naive chamam respectivamente **fleury** e **naive**. Os grafos gerados da classe **gerarGrafos** são armazenados na pasta **data**.

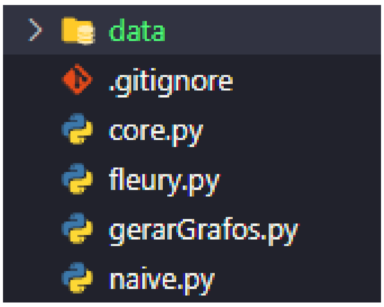
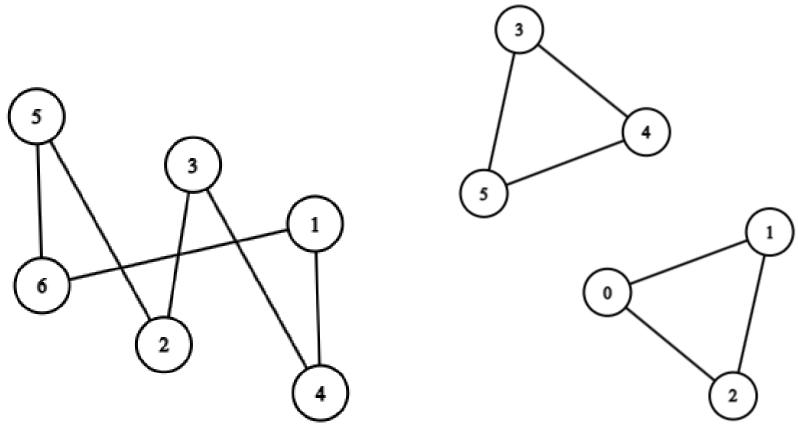


Figure 1. Estrutura dos Códigos em Python

3. Geração dos Grafos

3.1. Grafos Eulerianos

Inicialmente o desenvolvimento de ambos os três grafos teve como parâmetro a ideia de gerar grafos com destinos aleatórios, porém não foi obtido sucesso para garantir que a aleatoriedade da escolha do destino das arestas evitasse a obtenção de mais de um componente conexo. Em dois testes executados com a geração de um grafo de seis vértices, foi obtido o seguinte resultado.



(a) Ideia da Criação do Grafo Randômico (Success)

(b) Possível Geração Randomica do Grafo (Error)

Figure 2. Demonstração do Erro causado na Geração do Grafo na ideia inicial.

A solução encontrada para a geração dos grafos eulerianos foi cria-lo de com todos os vértices possuindo a base de grau dois, e de maneira sequencial, garantindo assim que o grafo teria sempre um único componente conexo.

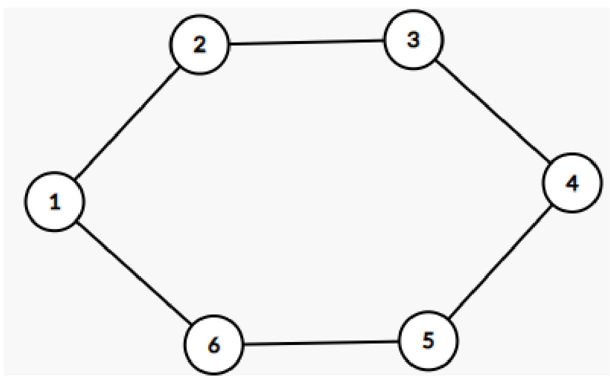


Figure 3. Grafo Euleriano - Gerados de maneira sequencial

3.2. Grafos Semi Eulerianos

Para a geração do grafo Semi euleriano, foi utiliado a base do grafo Euleriano, dessa maneira o programa gera o grafo euleriano e faz a conexão do primeiro vértice que será sempre o **um**, e o seu destino será estabelecido com o o **tamanho total da lista divido por dois**, sempre fazendo um arredondamento para baixo caso seja necessário.

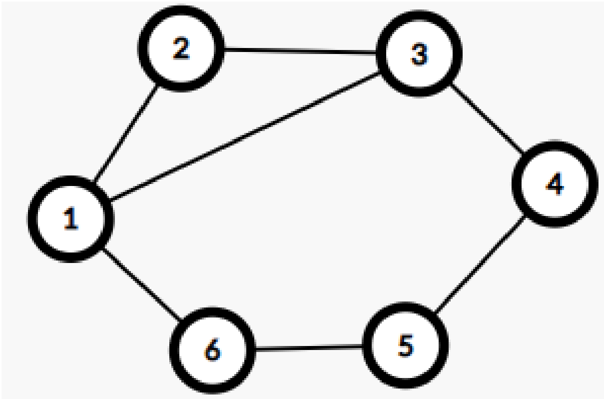


Figure 4. Grafo Semi Euleriano - Gerados de maneira sequencial

3.3. Grafos Não Eulerianos

Para a geração do grafo **não euleriano** também foi utilizado a base do grafo euleriano, e basicamente quando na geração do semi euleriano ele pega o tamanho da lista dividindo por dois para encontrar o destino, ele faz a mesma coisa para o não euleriano, porém agora quando ele termina de gerar os destinos padrões, ele adiciona mais uma ligação quando vai gerar o vértice dois, verificando sempre se já existe a ligação, e se já existir esse destino, ele vai para o destino gerado e adiciona um, e verifica se já existe um caminho do vértice dois para tal elemento, caso não existe ele conclui a conexão, fazendo assim o grafo possuir mais de dois vértices de grau impar, não atendendo mais a condição para ser considerado semi euleriano e se tornando não euleriano.

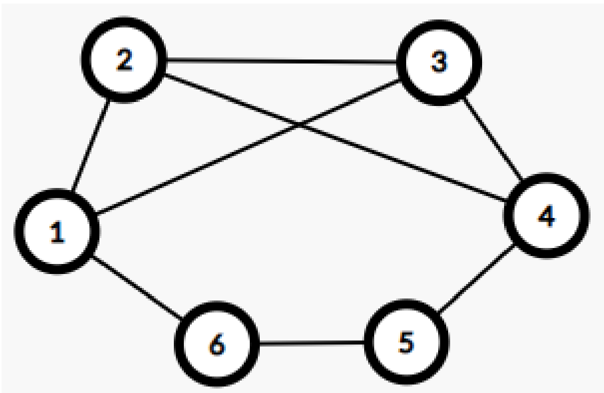


Figure 5. Grafo Não Euleriano - Gerados de maneira sequencial

4. Identificação de Grafos

Para identificar grafos foi utilizado a classe Fleury com o método **tipeGraph**, que basicamente depois da inicialização da matriz do grafo, o método vai analisar todas as posições da matriz pegar seu tamanho e calcular se é impar ou par, se for impar adiciona ao contador, e a partir disso vai identificar o tipo do grafo. Euleriano caso não exista vértice de grau impar, Semi caso exista grau impar em no máximo dois vértices e Não euleriano caso exista mais de dois vértices com grau impar.

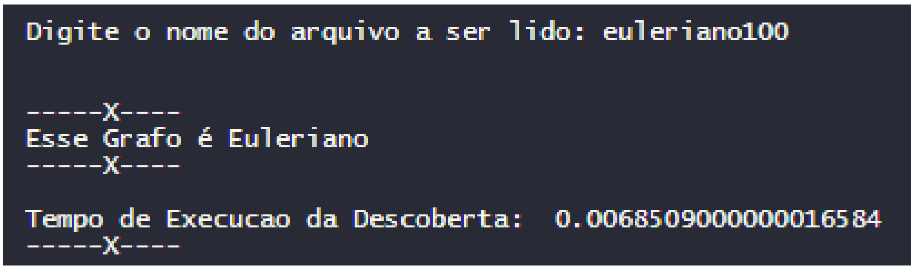


Figure 6. Identificando Grafo Euleriano

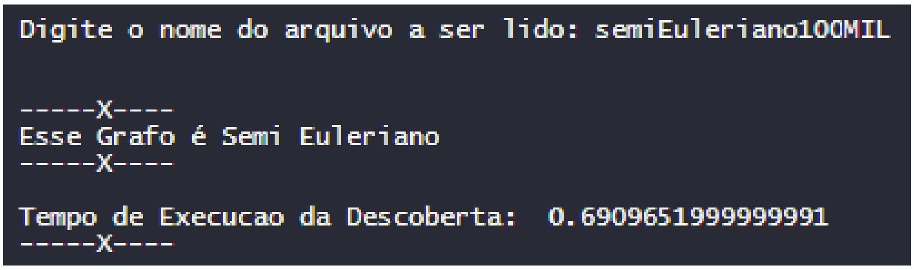


Figure 7. Identificando Grafo Semi Euleriano

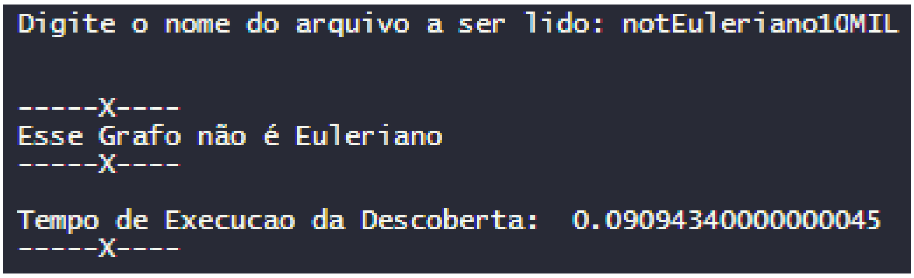


Figure 8. Identificando Grafo Não Euleriano

5. Fleury

5.1. Criação da Matriz de Vértices

O fleury foi implementado através da classe chamada **fleury**, recebendo como parâmetro o nome do arquivo desejado, e assim ele vai iniciar gerando a matriz do grafo, lendo linha por linha o arquivo e montando uma lista com esses valores, dessa maneira quando é lido a linha ele adiciona na matriz o caminho de volta, por exemplo se for lido a aresta **1 – 2** vai ser adicionado na lista na posição zero, e na posição um (referente a segunda posição) sera adicionado o caminho de volta o **2 – 1**, gerando assim a matriz completa dos elementos do grafo.

```
> 0: [[1, 2], [1, 3]]
> 1: [[2, 1], [2, 3], [2, 5], [2, 4]]
> 2: [[3, 1], [3, 2], [3, 4], [3, 6]]
> 3: [[4, 2], [4, 3], [4, 6], [4, 5]]
> 4: [[5, 2], [5, 4], [5, 6], [5, 7]]
> 5: [[6, 3], [6, 4], [6, 5], [6, 7]]
> 6: [[7, 5], [7, 6]]
  len(): 7
```

Figure 9. Matriz dos Elementos de um Grafo Euleriano com 7 Vértices

5.2. Naive

Para a identificação de pontes com o método Naive foi utilizado a **ideia base da busca em largura**, a ideia seria pegar o menor elemento dos vértices caso não exista caminho direto para o vértice inicial e analisar seus vértice em busca do caminho, basicamente procurando um **ciclo**, porém foram feitas alguma otimizações, uma delas consiste em analisar se existe caminho para um vértice que tem conexão com o vértice inicial, e não somente com o vértice analisado, dessa maneira economizando algumas etapas de processamento.

Na **figura 10** é um exemplo dessa teoria, para testar se o vértice dois é ponte ele vai para o vértice dois e verifica se tem outros vértices, eliminando o vértice na qual chegou, e testando primeiramente se tem conexão com algum vértice que o vértice um também tem conexão, e encontra no vértice três, garantido assim que não é ponte, e caso não fosse possível comprovar, ele pegaria outro menor elemento desconsiderando o três, seria o quatro, dessa maneira ele identificou que não tem ponte mesmo sem fazer o teste no vértice um para o três, colaborando para uma melhoria na performance do tempo.

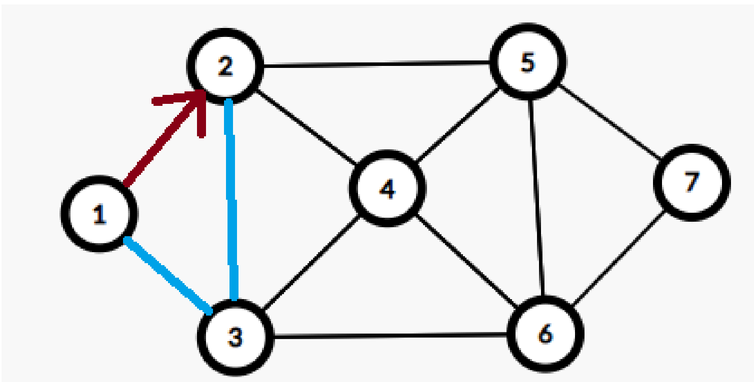


Figure 10. Executando o Naive para Identificação de Pontes

Na **figura 11** podemos perceber quando a execução já está verificando qual deve ser o destino do vértice três, e quando ele seleciona o menor elemento referente ao vértice um ele se depara com uma ponte, pois ele não consegue completar um ciclo, assim o vértice um não tem mais caminhos a não ser o que foi utilizado para chegar nele, dessa forma volta para o vértice que tem arestas disponíveis para serem percorridas, que é o proprio vertice inicial, pegando assim o próximo menor elemento que é o quatro, e nesse vértice ele faz uma verificação antes de selecionar o menor elemento do vértice quatro, ele testa se no vértice quatro existe destino para algum vértice que o três também tem destino, e encontra o seis, retornando assim que o vértice quatro não é ponte.

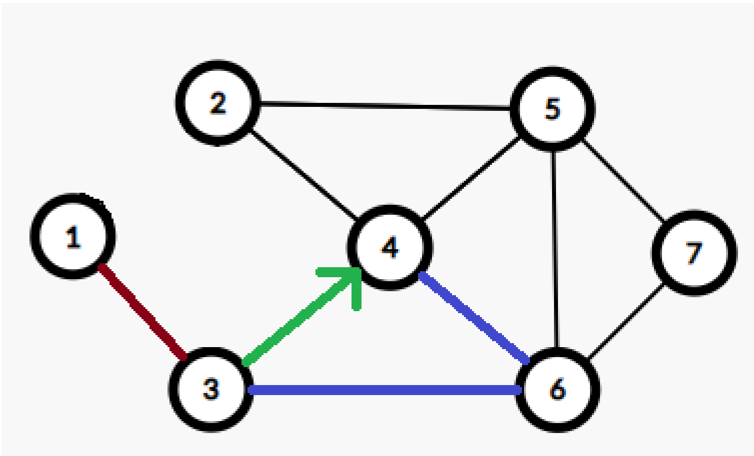


Figure 11. Executando o Naive para Identificação de Pontes - A partir do vértice 3

5.3. Naive - Tempos de Execução e Identificação Trajeto ou Ciclo

5.3.1. Tempos Grafos Eulerianos

Nome do Grafo	Quantidade de Elementos	Tempo em MS
Euleriano	100	0.001999729
Euleriano	1.000	0.019675900
Euleriano	10.000	0.9593129
Euleriano	100.000	116.80873499

5.3.2. Prints dos Testes e Determinando Trajeto ou Ciclo

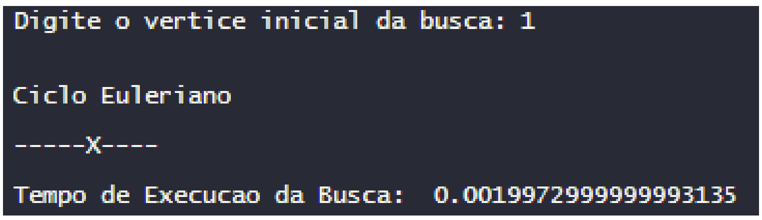


Figure 12. Tempo para descobrir o Caminho do Grafo Euleriano de 100 Vértices

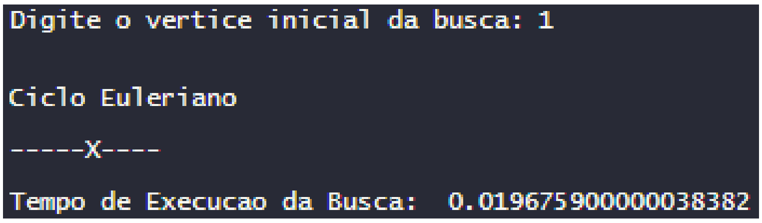


Figure 13. Tempo para descobrir o Caminho do Grafo Euleriano de 1000 Vértices

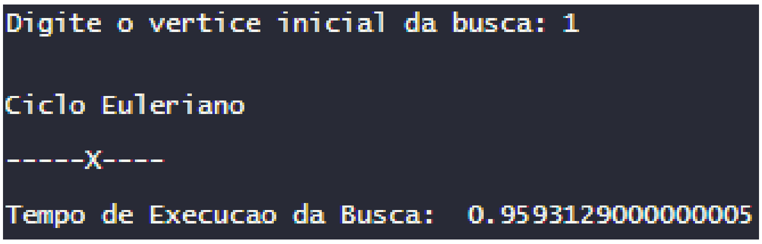


Figure 14. Tempo para descobrir o Caminho do Grafo Euleriano de 10.000 Vértices

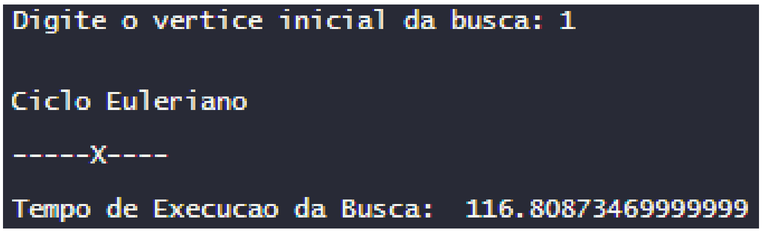


Figure 15. Tempo para descobrir o Caminho do Grafo Euleriano de 100.000 Vértices

5.3.3. Tempos Grafos Semi Euleriano

Nome do Grafo	Quantidade de Elementos	Tempo em MS
Semi Euleriano	100	0.00244200
Semi Euleriano	1.000	0.03435100
Semi Euleriano	10.000	0.561854199
Semi Euleriano	100.000	48.8132018

5.3.4. Prints dos Testes e Determinando Trajeto ou Ciclo

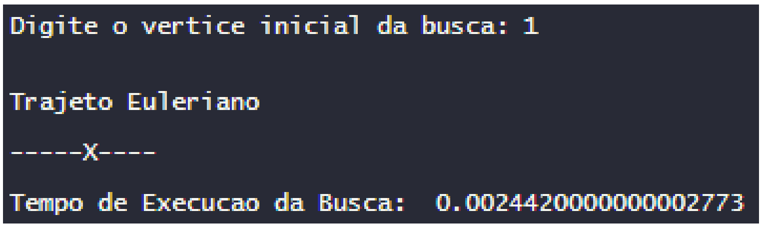


Figure 16. Tempo para descobrir o Caminho do Grafo SEMI Euleriano de 100 Vértices

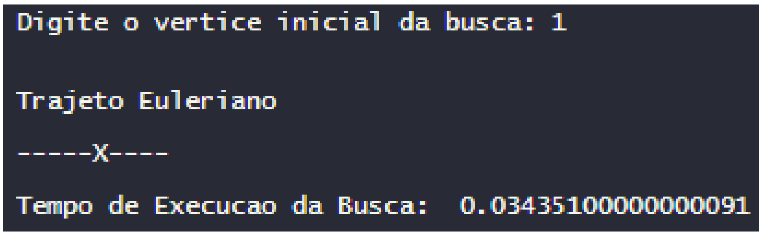


Figure 17. Tempo para descobrir o Caminho do Grafo SEMI Euleriano de 1000 Vértices

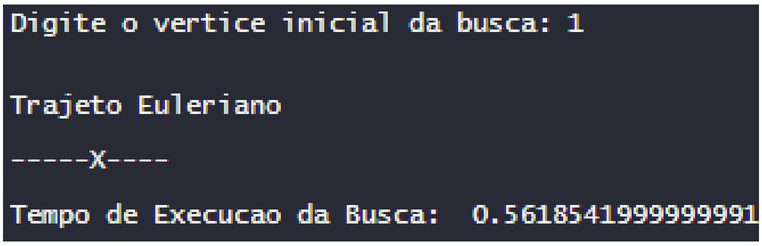


Figure 18. Tempo para descobrir o Caminho do Grafo SEMI Euleriano de 10.000 Vértices

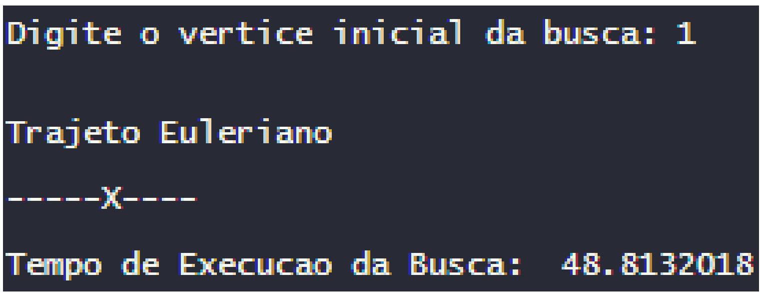


Figure 19. Tempo para descobrir o Caminho do Grafo SEMI Euleriano de 100.000 Vértices

6. Referências

Os conteudos usados para a criação do trabalho foi o material da matéria de Grafos [Júnior 2022], e o documento teve base no código criado que está presente no GitHub [Giovanini 2022]

References

Giovanini, V. H. (2022). Repositório do github de vinícius h. https://github.com/viniciushgiovanini/Grafos/tree/main/Atividade%20Avaliativas/TP01. Accessed: 2022-10-30.

Júnior, Z. K. G. D. P. (2022). Material acadêmico. https://www.pucminas.br. Accessed: 2022-10-30.