O Problema das Sete Pontes de Königsberg

Higor da Silva Camelo, Erick Gabriel Ferreira Gaspar, Pedro Henrique Santos Moreira

1

Resumo. *Este trabalho aborda o problema das Pontes de Königsberg, uma questão clássica da teoria dos grafos, e explora dois algoritmos fundamentais utilizados para encontrar ciclos Eulerianos: o algoritmo de Fleury e o algoritmo de Hierholzer. A primeira parte deste trabalho apresenta uma introdução ao problema das Pontes de Königsberg, bem como uma análise teórica das condições necessárias para a existência de um ciclo Euleriano em um grafo. Em seguida, são detalhados os procedimentos e as características dos algoritmos de Fleury e de Hierholzer, com ênfase em suas diferenças operacionais e respectivas eficiências computacionais.*

1. Introdução

O problema das pontes de Königsberg constitui um dos problemas mais significativos na gênese da Teoria dos Grafos, introduzido no século XVIII pelo matemático Leonhard Euler. A cidade de Königsberg, localizada na antiga Prússia Oriental (atualmente Kaliningrado, Rússia), era atravessada pelo rio Pregel, que dividia a cidade em quatro regiões de terra conectadas por sete pontes. O questionamento que se fazia a época era o seguinte: seria possível percorrer um trajeto que cruzasse todas as pontes exatamente uma vez e retornasse ao ponto de partida?

A relevância desse problema não se restringe à sua solução direta, mas reside na abstração matemática realizada por Euler. Sua resolução, apresentada em 1736, é amplamente considerada como o primeiro teorema da teoria dos grafos, um campo que desde então desempenha um papel fundamental nas ciências matemáticas e da computação.

2. Contextualização Histórica e Formulação do Problema

O problema em questão consistia na seguinte situação: a cidade de Königsberg era dividida em quatro regiões distintas de terra, denominadas A, B, C e D, como ilustrado na Figura 1. Os habitantes da cidade se perguntavam se seria possível realizar um passeio que cruzasse todas as pontes exatamente uma vez, retornando ao ponto de partida.

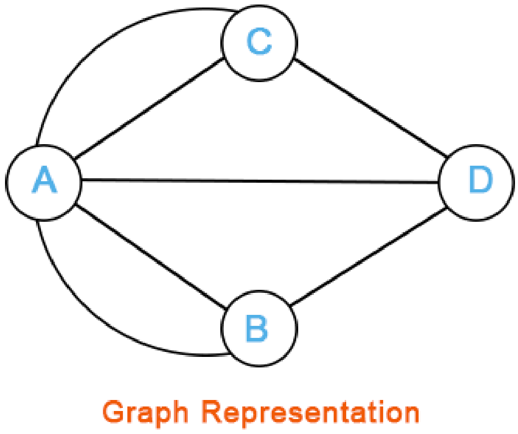

Figure 1. Representação das Pontes de Königsberg em grafo

Euler transformou esse problema em um modelo matemático ao representar as regiões de terra como vértices de um grafo e as pontes como arestas conectando esses vértices. Assim, o problema foi transcrito para um contexto puramente topológico, em que o objetivo era determinar a existência de um *circuito Euleriano*, ou seja, um ciclo no grafo que percorresse cada aresta exatamente uma vez, retornando ao vértice inicial.

3. Propriedades das Entradas e Saídas do Problema

Para que os algoritmos propostos possam ser aplicados corretamente e garantir a existência de um ciclo Euleriano, é necessário que determinadas condições sejam satisfeitas. Essas condições podem ser descritas em termos de pré-condições e pós-condições.

3.1. Pré-condições

As pré-condições estabelecem os requisitos que o grafo $G = (V, E)$ deve satisfazer:

- **Conectividade**: O grafo G deve ser conexo, isto é, para quaisquer dois vértices $u, v \in V$, deve existir um caminho que os conecte. Formalmente, para todo $u, v \in V$, existe uma sequência de vértices $u = v_1, v_2, \ldots, v_n = v$ tal que $\{v_i, v_{i+1}\} \in E$, para $i = 1, \ldots, n-1$. Isso garante que todas as arestas do grafo podem ser percorridas durante o processo de busca por um ciclo Euleriano.
- **Grau par dos vértices**: Todos os vértices do grafo devem ter grau par. Se $\deg(v)$ denota o grau de um vértice $v \in V$, então $\deg(v)$ deve ser par para todo $v \in V$. Isto é, o número de arestas incidentes a cada vértice deve ser par, pois, para a existência de um ciclo Euleriano, cada vez que um vértice é visitado, deve haver uma aresta de entrada e uma de saída.
- **Grafo não-direcionado**: O grafo G deve ser não-direcionado, ou seja, as arestas não possuem orientação. A presença de orientação nas arestas implica na necessidade de condições adicionais específicas para grafos direcionados, as quais não são consideradas neste problema.

3.2. Pós-condições

As pós-condições definem o comportamento esperado do algoritmo aplicado ao grafo G:

- **Ciclo Euleriano**: Se o grafo G satisfizer as pré-condições, o algoritmo retornará um ciclo Euleriano C, que é uma sequência de vértices $v_1, v_2, \ldots, v_n$ tal que cada aresta $\{v_i, v_{i+1}\} \in E$ é percorrida exatamente uma vez, e $v_n = v_1$, ou seja, o ciclo retorna ao vértice inicial.
- **Inexistência de ciclo Euleriano**: Caso o grafo G não satisfaça as pré-condições, o algoritmo deve concluir que não existe um ciclo Euleriano. Por exemplo, se houver vértices com grau ímpar ou se o grafo não for conexo, será impossível percorrer todas as arestas exatamente uma vez.
- **Preservação da estrutura do grafo**: Durante a execução do algoritmo, o grafo G deve permanecer inalterado de maneira permanente. Embora o algoritmo possa percorrer as arestas e, conceitualmente, ”remover” arestas temporariamente durante o processo, a estrutura original do grafo não deve ser modificada após a conclusão do algoritmo.

4. Exemplos de Saídas

Para ilustrar a aplicação das propriedades definidas previamente, consideram-se dois exemplos de grafos: um que satisfaz as condições necessárias para a existência de um ciclo Euleriano e outro que não as atende.

O primeiro exemplo, ilustrado na Figura 2, apresenta um grafo simples e conexo, no qual todos os vértices possuem grau par. Neste caso, segundo as pré-condições estabelecidas, o grafo possui um ciclo Euleriano, ou seja, é possível percorrer todas as arestas exatamente uma vez, retornando ao vértice inicial.

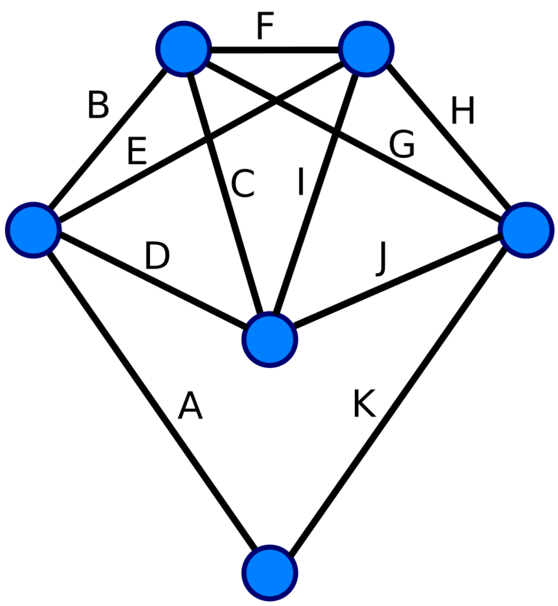

Figure 2. Exemplo de grafo com ciclo Euleriano

No entanto, no segundo exemplo, representado pela Figura 3, observa-se que há vértices com grau ímpar. Neste caso, conforme previsto pelas condições necessárias, o grafo não possui um ciclo Euleriano, impossibilitando a travessia de todas as arestas exatamente uma vez sem repetir caminhos.

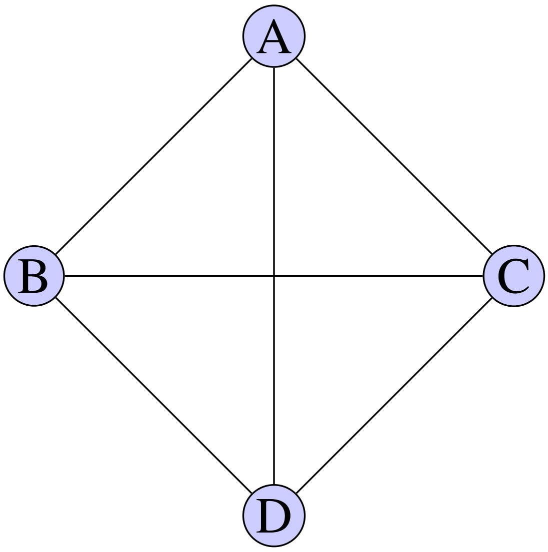

Figure 3. Exemplo de um grafo não Euleriano.

Assim, ao analisar as propriedades de um grafo em relação ao problema das pontes de Königsberg, possibilita-se identificar se a estrutura atende às pré-condições necessárias para a existência de um ciclo Euleriano. Grafos que satisfazem as condições de conectividade e de grau par em todos os vértices são exemplos de estruturas que permitem a solução do problema de Euler, enquanto grafos que violam essas condições não permitem tal solução.

5. Algoritmos

5.1. Algoritmo de Fleury

O algoritmo de Fleury é fundamentado na construção de um ciclo Euleriano, priorizando a não utilização de arestas classificadas como pontes, sempre que possível. A ideia principal é garantir que o grafo permaneça conexo durante a exploração, assegurando que todas as arestas sejam percorridas sem comprometer sua conectividade.

5.1.1. Funcionamento

Dado um grafo não direcionado, o algoritmo segue os seguintes passos:

1. Escolhe-se um vértice de início. Caso o grafo possua 0 vértices de grau ímpar, a busca pode começar em qualquer vértice. Se houver 2 vértices de grau ímpar, deve-se iniciar a busca por um deles.
2. A partir do vértice inicial, segue-se por uma aresta adjacente. Se essa aresta for uma ponte, ela deve ser escolhida apenas se não houver outra alternativa.
3. A aresta selecionada é removida do grafo.
4. O algoritmo termina quando não restarem mais arestas a serem percorridas.

5.1.2. Pseudocódigo

Algorithm 1 temPonte

```
temPonte(G, u, v)
remove aresta u − v de G
visitados ← []
para i ← 0 até V faça
    visitados[i] ← false
fim para
DFS( v ):
visitados[v] ← true
para cada vértice w ajacente a v faça
    se visitados[w] = false então
        DFS(w)
    fim se
fim para
DFS(u)
devolvevisitados[v]
=0
```

Algorithm 2 Algoritmo de Fleury

```
Fleury( G, u )
arestas ← número de arestas de G
C ← []
adiciona u para C
enquanto arestas > 0 faça
    para cada vértice v adjacente a u faça
        se número de vértices adjacentes a u > 1 and temPonte(G, u, v) então
            continue
        fim se
        remove aresta u − v de G
        adiciona v para C
        u ← v
        arestas ← arestas − 1
        break
    fim para
fim enquanto
devolveC
=0
```

5.1.3. Exemplos

Considere o grafo com quatro vértices e quatro arestas mostrado na Figura 4. A busca pelo ciclo de Euler será iniciada no vértice 2.

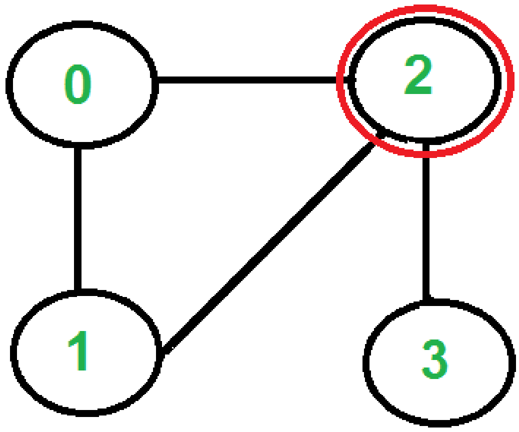

Figure 4. Grafo exemplo

A partir do vértice 2, há três opções de caminhos a seguir. Como o objetivo do algoritmo é evitar, sempre que possível, atravessar pontes, a aresta que leva ao vértice 3 será evitada, preservando a conectividade do grafo. Assim, opta-se por seguir para o vértice 0.

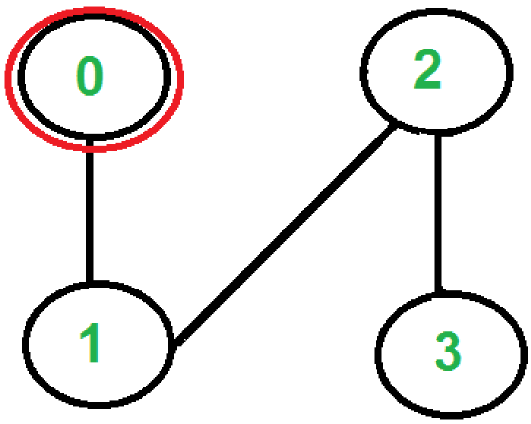

Figure 5. Vértice 0 selecionado

O vértice 0 possui apenas uma aresta adjacente, o que obriga a seguir por ela. Essa situação se repetirá até que todas as arestas sejam percorridas.

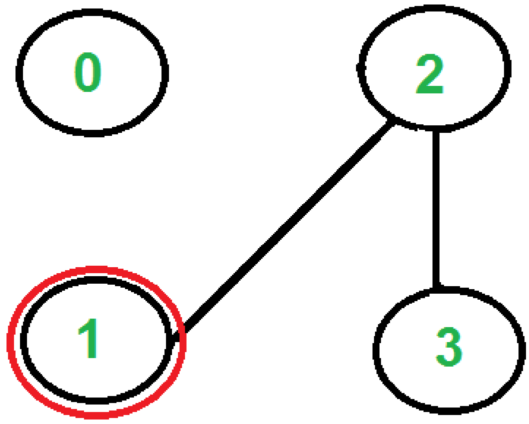

Figure 6. Vértice 1 selecionado

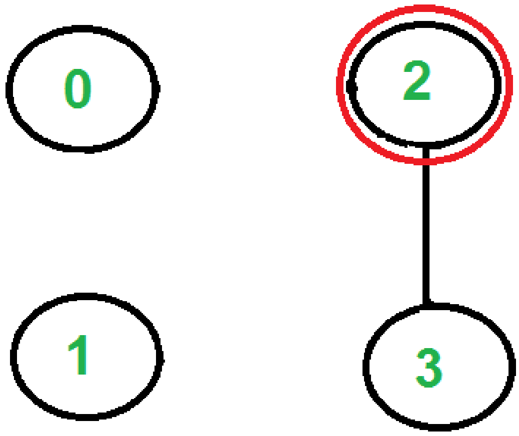

Figure 7. Vértice 2 selecionado

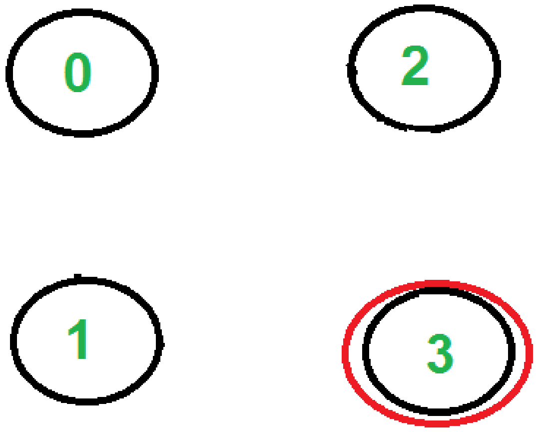

Figure 8. Vértice 3 selecionado

Ao final, todas as arestas foram percorridas e removidas, encerrando a execução do algoritmo. O ciclo resultante contém as arestas: $(2-0),(0-1),(1-2),(2-3)$. Observa-se que o vértice de início da busca é diferente do vértice de término, indicando que o grafo não possui um ciclo Euleriano.

5.1.4. Corretude

Considerando o laço de repetição das linhas 6 a 15, a invariante e sua demonstração são apresentadas a seguir:

Invariante1: No início de cada iteração i-ésima, o vértice inicial u possui um vértice v tal que possa formar uma aresta a ser removida.

- **Inicialização:** Antes da primeira iteração, o vértice inicial não possui nenhum vértice adjacente a ser comparado. Neste ponto, nenhuma aresta pode ser formada ou percorrida, portanto, validando a invariante.
- **Manutenção:** Supondo que na t-ésima iteração a invariante seja válida. Assim, verifica-se cada vértice ajacente ao vértice inicial. Caso o número de vértices ajacentes a u for maior que 1 e existe uma ponte na aresta formada pelo vértice inicial e o vértice atual da comparação, nada é feito, caso contrário, é removida esta aresta do grafo G, em seguida, adiciona-se o vértice final dessa aresta ao vetor C e, por fim, reduzido o valor da variável $arestas$ em uma unidade. Dado que uma aresta foi formada, a invariante se mantém.
- **Finalização:** Ao término do laço, todas as arestas do grafo terão sido removidas. A condição de repetição falhará quando o valor da variável $arestas$ for igual a 0. Nesse ponto, dado que uma aresta foi removida, é sinal de que foi formada anteriormente. Portanto, a invariante é válida.

Considerando o laço de repetição das linhas 5 a 16, a invariante e sua demonstração são apresentadas a seguir:

Invariante2: No início de cada iteração i-ésima, o vetor C possui todos os vértices percorridos no algoritmo.

- **Inicialização:** Antes da primeira iteração, o vértice inicial é adicionado ao vetor C. Neste ponto, nenhuma aresta foi percorrida e o vetor possui apenas o vértice inicial. Portanto, validando a invariante.
- **Manutenção:** Suponha que na t-ésima iteração a invariante seja válida. Assim, a variável $arestas$ possui o valor x. Em seguida, iremos iterar sob o vértice adjacentes a u e comparamos se as arestas formadas são pontes ou não, e caso não, iremos remover esta aresta do vetor, adicionar o vértice final para C e diminuir em uma unidade o valor da variável $arestas$. Assim, C possuirá o caminho do ciclo formado até agora. Dado exposto, a invariante se mantém.
- **Finalização:** Ao término do laço, todas as arestas do grafo terão sido removidas. A condição de repetição falhará quando o valor da variável $arestas$ for igual a 0,

o que indica que não há vértices adjacentes mutuamente. Portanto, o vetor conterá todos os vértices percorridos, validando a invariante.

5.1.5. Complexidade

A seguir, analisaremos a complexidade do algoritmo no pior caso. Para um grafo $G = (V, E)$, as seguintes considerações são feitas:

- **Linhas 1 - 4:** Operações de atribuição básicas com complexidade $O(1)$.
- **Linha 5:** Iteração sobre o número de arestas, com complexidade $O(E)$.
- **Linha 6:** Iteração sobre os adjacentes ao vértice inicial, com complexidade $O(V)$.
- **Linha 7:** Verificação se a aresta é uma ponte. Como usamos a lógica do DFS, temos uma complexidade $O(V + E)$.
- **Linhas 8:** Complexidade $O(1)$.
- **Linhas 10 - 14:** Operações de atribuição básicas, com complexidade $O(1)$.

Complexidade total: A complexidade geral do algoritmo é dada pela combinação das operações mais significativas, resultando em $O(E \cdot V)$ no pior caso.

5.2. Algoritmo de Hierholzer

O algoritmo de Hierholzer é uma abordagem eficiente para encontrar um ciclo Euleriano em um grafo Euleriano. A estratégia baseia-se na construção iterativa do ciclo Euleriano por meio de subciclos. A principal ideia é explorar subcaminhos de forma que todas as arestas sejam percorridas, garantindo que o ciclo final contenha todas as arestas sem repetições ou omissões.

5.2.1. Funcionamento

Dado um grafo não direcionado, o algoritmo de Hierholzer pode ser descrito pelos seguintes passos:

1. Seleciona-se um vértice arbitrário do grafo como ponto inicial.
2. A partir desse vértice, percorrem-se arestas ainda não visitadas até que um ciclo fechado seja formado, isto é, até retornar ao vértice de origem. Durante a execução, se ainda existirem arestas não exploradas, seleciona-se um vértice pertencente ao ciclo atual que possua arestas não visitadas, e um novo ciclo é construído a partir desse vértice.
3. O novo ciclo encontrado é inserido no ciclo previamente identificado.
4. O algoritmo termina quando todas as arestas do grafo tiverem sido percorridas.

5.2.2. Pseudocódigo

Algorithm 3 Hierholzer

Hierholzer(G, u)
para $v \in V(G)$ **faça**
  $d(v) \leftarrow adj(v).tamanho$
fim para
$S \leftarrow [u]$
$C \leftarrow []$
enquanto $S \neq vazia$ **faça**
  $u \leftarrow topo(S)$
  se $d(u) > 0$ **então**
    Selecione $v \in adj(u)$
    $RemoverAresta(u, v, G)$
    $d(u) \leftarrow d(u) - 1$
    $d(v) \leftarrow d(v) - 1$
    $Empilha(v, S)$
  senão
    $Desempilha(u, S)$
    $Adiciona(u, C)$
  fim se
fim enquanto
devolve C
=0

5.2.3. Exemplos

Considerando um grafo com quatro vértices, conforme ilustrado na Figura 9. O algoritmo inicia a execução pelo vértice 0. Este vértice é empilhado em S, e o algoritmo procede a percorrer as arestas ainda não visitadas.

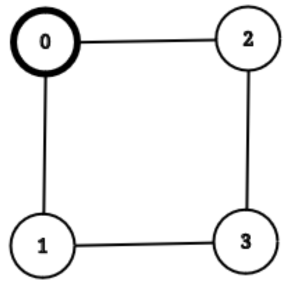

Figure 9. Grafo exemplo no estado inicial

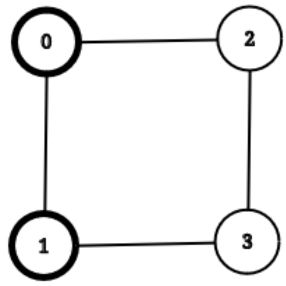

Figure 10. Procura o vértice adjacente de 0

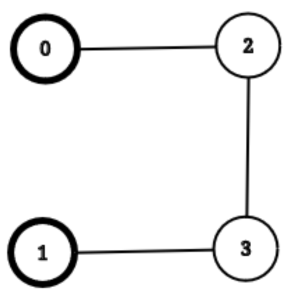

Figure 11. Remove aresta (0, 1)

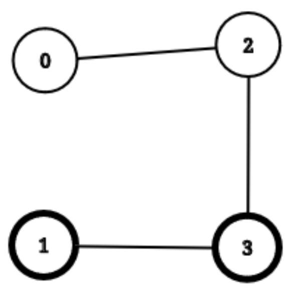

Figure 12. Seleciona o vértice 3

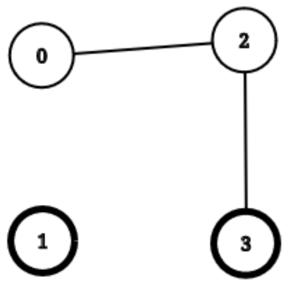

Figure 13. Remove aresta (1,3)

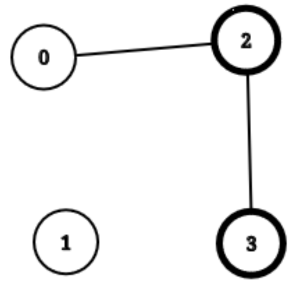

Figure 14. Seleciona o vértice 2

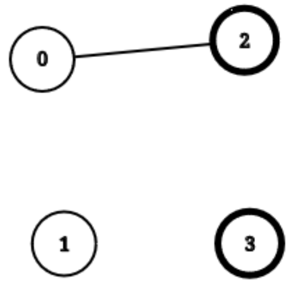

Figure 15. Remove aresta (3,2)

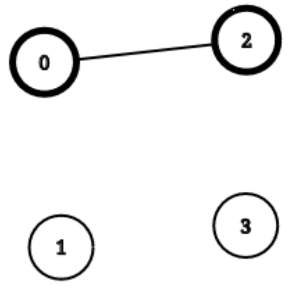

Figure 16. Seleciona o vértice 0

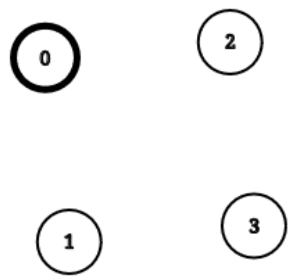

Figure 17. Remove a aresta (2,0)

Ao final da execução do algoritmo, todas as arestas foram percorridas e o estado da pilha S é [0,1,3,2,0], formando um ciclo completo. O laço final do algoritmo remove os vértices da pilha e os adiciona ao ciclo C. O ciclo Euleriano resultante para o grafo apresentado é $C = [0, 2, 3, 1, 0]$.

5.2.4. Corretude

Invariante 1: No início de cada iteração do laço principal (linha 7), a pilha S contém o caminho atual sendo explorado, e o ciclo C contém o ciclo Euleriano parcial ou completo construído até o momento.

- **Inicialização:** Antes da primeira iteração, a pilha S contém apenas o vértice inicial u, e o ciclo C está vazio. Assim, nenhum ciclo foi construído até este ponto, o que garante que a invariante é válida no início da execução.
- **Manutenção:** Durante a execução do laço principal, o algoritmo verifica se o vértice u no topo da pilha ainda possui arestas não visitadas. Caso existam arestas não percorridas, uma delas é escolhida, e a aresta correspondente é removida do grafo, atualizando-se os graus dos vértices u e v. Em seguida, o vértice v é adicionado à pilha, expandindo o caminho atualmente explorado. Quando o vértice no topo da pilha não possui mais arestas não visitadas, ele é removido da pilha e inserido no ciclo C, indicando que faz parte de um ciclo fechado. Como o algoritmo percorre as arestas de maneira a retornar ao vértice de origem, um ciclo é formado. O ciclo C cresce à medida que vértices são removidos da pilha, garantindo que, ao final do processo, o ciclo final será Euleriano, visto que todas as arestas terão sido visitadas uma única vez.
- **Término:** O algoritmo termina quando a pilha S estiver vazia, ou seja, quando todas as arestas do grafo tiverem sido percorridas e todos os ciclos foram fechados e incorporados ao ciclo Euleriano C. Isso assegura a validade da invariante até a finalização da execução.

5.2.5. Complexidade

A análise de complexidade no pior caso para o algoritmo, dado um grafo Euleriano $G(V, E)$, é apresentada a seguir:

- **Linhas 2-4:** A complexidade é **O(V)**, pois o algoritmo percorre todos os vértices $V(G)$, e a operação $u \leftarrow adj[u]$ ocorre em tempo $O(1)$.
- **Linha 5:** A complexidade é **O(1)**, pois a pilha S é inicializada com o vértice u.
- **Linha 6:** A complexidade é **O(1)**, pois o ciclo C é inicializado como vazio.
- **Linhas 7-19:** A complexidade é **O(E)**, visto que o laço principal executa enquanto a pilha S não estiver vazia. Cada aresta é processada uma única vez, pois o algoritmo remove a aresta do grafo logo após percorrê-la. Como o grafo contém E arestas, esta parte do laço principal executa no máximo $O(E)$ vezes. As operações das linhas 8 a 14 incluem a seleção de uma aresta ($O(1)$), a remoção de arestas ($O(1)$), a inserção de vértices na pilha ($O(1)$), e a atualização dos graus dos vértices u e v ($O(1)$). Nas linhas 15 a 17, as operações de desempilhamento ($O(1)$) e a adição de vértices ao ciclo C são realizadas nas arestas já percorridas, ocorrendo uma vez para cada vértice.

Complexidade total: A complexidade total do algoritmo no pior caso é **O(V + E)**.

Referências

1. Biggs, N. L., Lloyd, E. K., & Wilson, R. J. (1976). *Graph Theory 1736-1936*. Clarendon Press, Oxford.
2. Fleischner, H. (1991). *Eulerian Graphs and Related Topics: Part 1*, volume 2 of Annals of Discrete Mathematics. Elsevier.
3. GeeksforGeeks. (n.d.). *Fleury's algorithm for printing eulerian path*. Disponível em: `https://www.geeksforgeeks.org/fleurys-algorithm-for-printing-eulerian-path/`. Acesso em [Data de acesso].
4. Taylor, P. (2000). *What Ever Happened to Those Bridges?* Australian Mathematics Trust. Consultado em 12 de abril de 2010. Arquivado do original em 19 de março de 2012.