Ciclos eulerianos: uma aproximação usando algoritmo de Fleury

Leandro L. A. Vieira,[1] **Mateus P. Silva**[2]

[1]Ciência da Computação – Universidade Federal de Viçosa (UFV-caf)
– Florestal – MG – Brasil

(leandrolazaro[1], mateus.p.silva[2])@ufv.br

Resumo. *Este trabalho trata-se da implementação do algoritmo de Fleury para a solução do ciclo euleriano de um grafo.*

1. Introdução

A teoria dos grafos é um dos principais métodos de modelagem de algoritmos graças à sua simplicidade, o que a faz tangenciar a universalidade na solução dos mais variados problemas. Ademais, os avanços tecnológicos fazem com que o tamanho dos dados cresçam exponencialmente todos os anos, tornando obrigatório o estudo de formas mais eficientes de operações computacionais, especialmente as com grafos.

Um grafo euleriano apresenta um ciclo em que é possível iniciar um percurso num vértice arbitrário, passar por cada uma das arestas apenas uma vez, e retornar ao vértice inicial. Esse ciclo é chamado Ciclo Euleriano. É condição necessária que todos os vértices tenham grau par para que hajam ciclo(s) euleriano(s).

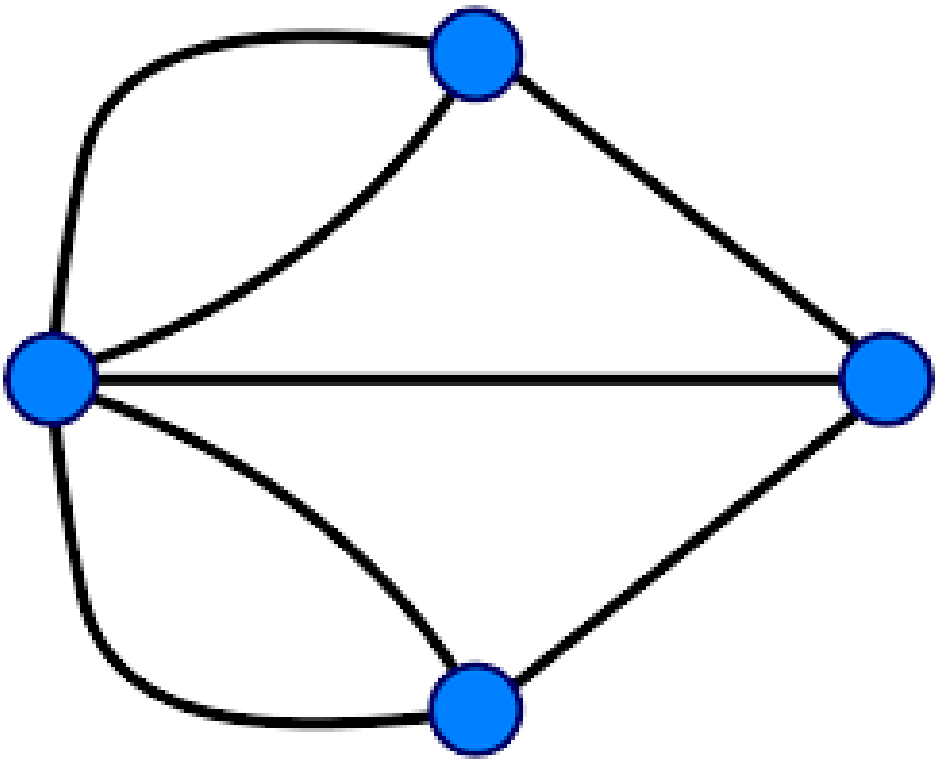

Figura 1. Grafo com ciclo euleriano

2. Implementação

Utilizamos a linguagem C++ no paradigma orientado a objetos pelo bom desempenho e quantidade de estruturas de dados prontas, como pilas e filhas. Criamos uma única classe que representa um grafo utilizando listas de adjacência. Além disso, há também um inteiro que conta quantos vértices há no grafo.

```
    class Graph
    {
        private:
            unsigned int V;
            list<int> *adj;
        public:
            Graph(string filename);
            ~Graph();

            void addEdge(int u, int v);
            void rmvEdge(int u, int v);
            void printEulerTour();
            void printEulerUtil(int s);
            int DFSCount(int v, bool visited[]);
            bool isValidNextEdge(int u, int v);
            bool isEulerian();
    [...]
    }
```

Há funções para adicionar vértice, remover, exibir um caminho euleriano, caminho euleriano até um dado vértice, contar o grau de um vértice, verificar se o próximo vértice é valido (ou seja, não é ponte) e verificar se o grafo é euleriano.

2.1. O algoritmo de Fleury

O algoritmo funciona iterando todas as arestas do vértice, e, caso ela não seja uma ponte, ela é removida, e o algoritmo é chamado recursivamente para o outro vértice conectado a essa aresta.

```
void Graph::printEulerUtil(int u)
{
    list<int>::iterator i;
    for (i = adj[u].begin(); i != adj[u].end(); ++i)
    {
        int v = *i;


        if (v != -1 && isValidNextEdge(u, v))
        {
            cout << (v+1) << " ";
            rmvEdge(u, v);
            printEulerUtil(v);
        }
    }
}
```

2.2. Criador de grafos aleatórios eulerianos

Também foi implementado um gerador muito simples de grafos eulerianos como arquivos de entrada para o algoritmo. Ele cria todas as arestas como arestas paralelas, garantindo a condição necessária para ser euleriano.

3. Resultados

3.1. Grafo da proposta

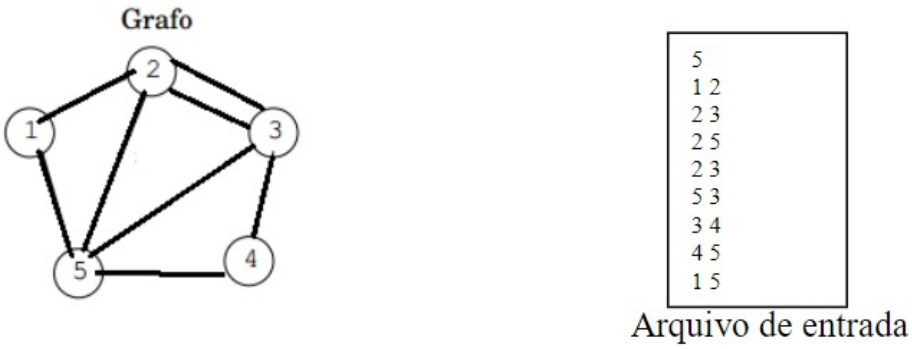

Figura 2. Grafo pedido

Utilizando o grafo da proposta, foi obtido o seguinte ciclo:

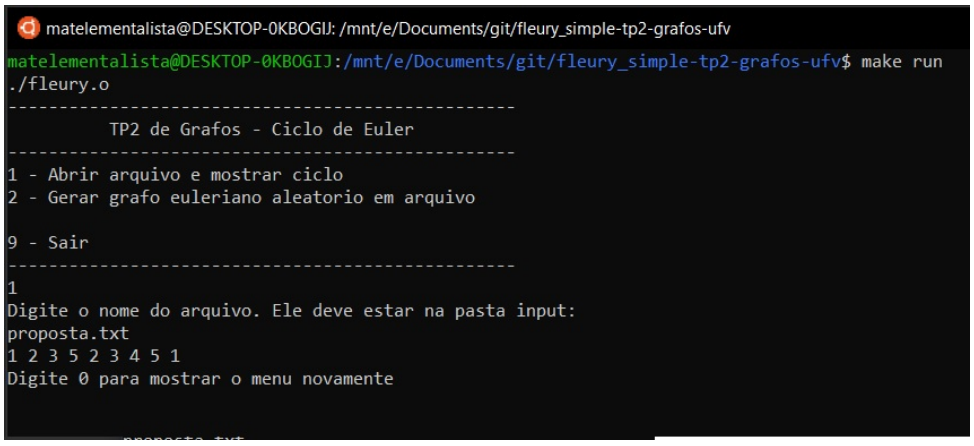

Figura 3. Ciclo obtido na execução do programa

Trata-se de um ciclo válido.

3.2. Grafo euleriano com 100 vértices e 150 arestas

Trata-se do arquivo max.txt.

100									
94	56	64	76	60	84	34	23	2	21
56	94	44	69	84	60	73	82	23	93
56	80	69	44	2	3	82	73	93	23
80	56	71	99	3	2	17	74	35	39
15	54	99	71	52	24	74	17	39	35
54	15	1	86	24	52	37	21	19	23
17	18	86	1	54	4	21	37	23	19
18	17	5	69	4	54	50	42	59	20
84	1	69	5	61	10	42	50	20	59
1	84	55	88	10	61	24	10	64	35
33	59	88	55	73	15	10	24	35	64
59	33	69	87	15	73	51	48	29	14
81	59	87	69	97	93	48	51	14	29
59	81	98	1	93	97	77	47	82	57
54	39	1	98	2	94	47	77	57	82
39	54	98	51	94	2	41	30	12	22
6	98	51	98	94	51	30	41	22	12
98	6	39	3	51	94	92	34	86	55
41	70	3	39	96	32	34	92	55	86
70	41	48	80	32	96	80	39	7	65
61	55	80	48	53	95	39	80	65	7
55	61	24	8	95	53	17	84	45	24
72	22	8	24	63	28	84	17	24	45
22	72	86	47	28	63	33	80	48	77
87	10	47	86	2	100	80	33	77	48
10	87	82	72	100	2	11	86	3	11
6	37	72	82	27	35	86	11	11	3
37	6	56	87	35	27	79	89	15	33
76	64	87	56	23	34	89	79	33	15
21	34	34	21	35	99	21	2	99	35

Tabela 1. Quantidade de vértices em negrito. Arestas demarcadas como adjacências.

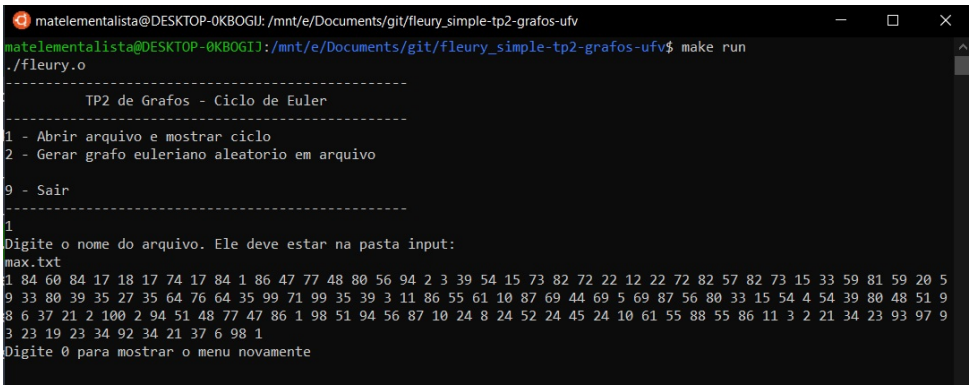

Figura 4. Ciclo obtido na execução de max.txt

3.3. Considerações finais

Através deste trabalho, foi possível implementar o algoritmo de Fleury que cria um ciclo euleriano. Foi criado o algorítmo de forma o mais simples possível, sem nenhum diferencial, porém de forma eficiente