TP-01 Grafos

Nome Felipe Carvalho de Paula Silva

Gerador de grafos

Para 100, 1.000, vértices gerei grafos com arestas equivalentes a 2 vezes o numero de vértices; Porem para 10.000 e 100.000 vértices foi necessário diminuir o numero de arestas para 1*1 o numero de vértices pois o gerador de grafos aleatórios não estava conseguindo gerar grafos tao grandes.

Exemplos de execução

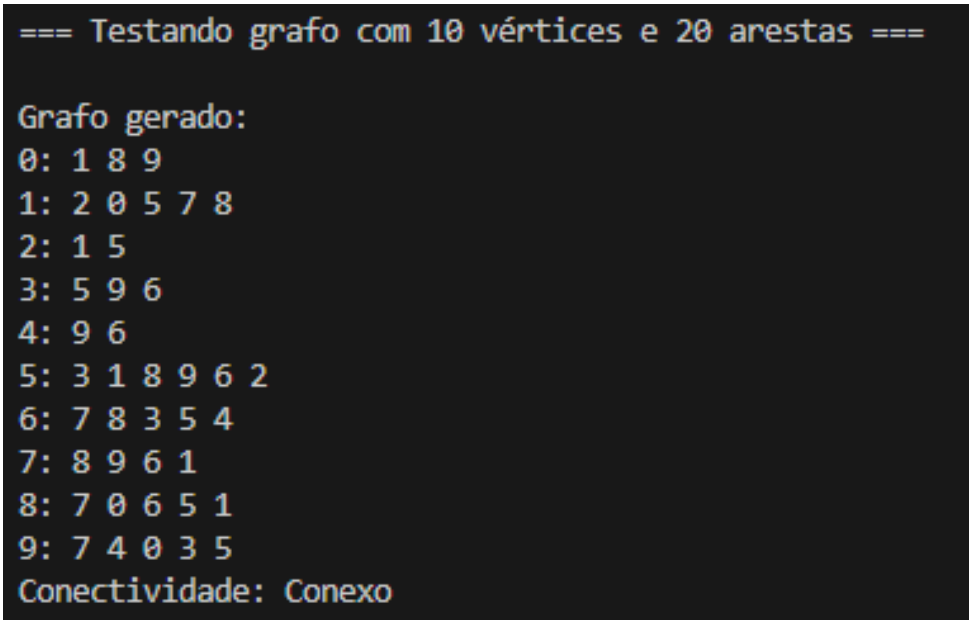

Figure 1: Exemplo de grafo gerado

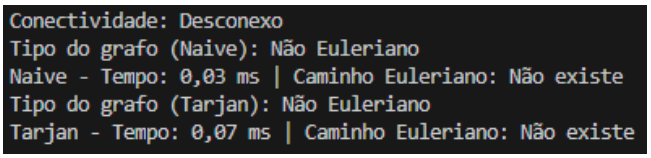

Figure 2: 100 vértices / 200 arestas

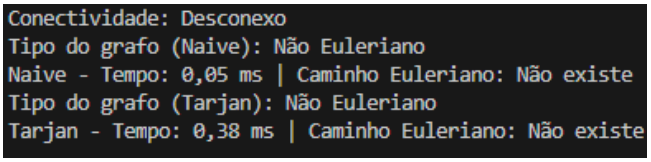

Figure 3: 1000 vértices / 2000 arestas

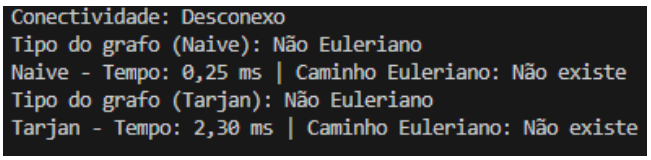

Figure 4: 10000 vértices / 10000 arestas

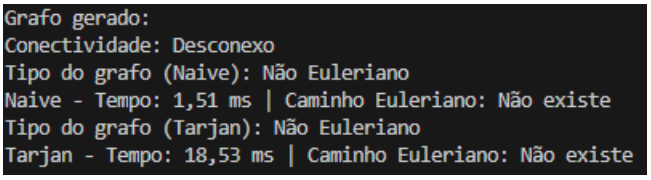

Figure 5: 100000 vértices / 55555 arestas

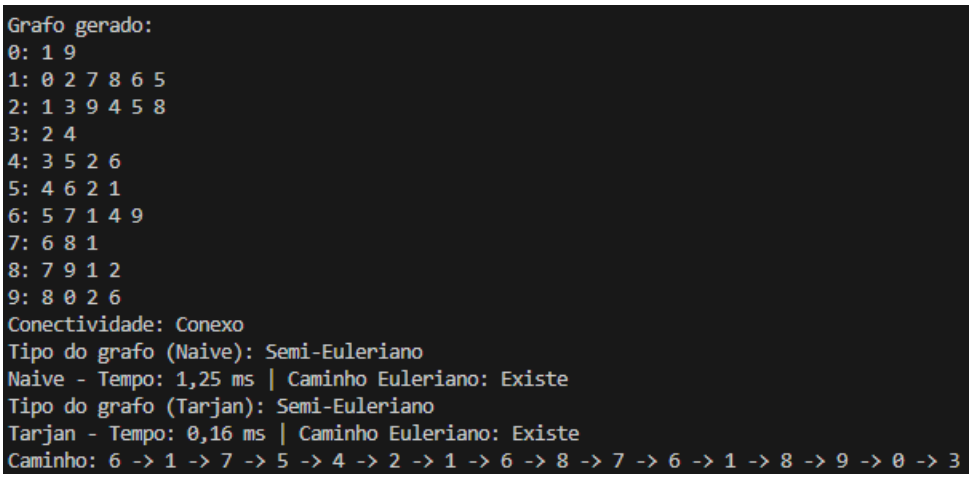

Figure 6: Caminho euleriano

Código Gerador de Grafos, Naive, Tarjan, Fleury

```java
package Grafo;

import java.util.*;

// Classe principal para gerar um grafo aleat rio e encontrar pontes
class Graph {

    private int V; // N mero de v rtices
    private List<List<Integer>> adj; // Lista de adjac ncia

    public Graph(int V) {
        this.V = V;
        adj = new ArrayList<>(V);
        for (int i = 0; i < V; i++) {
            adj.add(new ArrayList<>()); // Inicializa a lista de adjac ncia para
                cada v rtice
        }
    }

    public Graph clone() {
        Graph copia = new Graph(this.V);
        for (int i = 0; i < V; i++) {
            copia.adj.set(i, new ArrayList<>(this.adj.get(i)));
        }
        return copia;
    }

    public int getV() {
        return V;
    }

    public List<List<Integer>> getAdj() {
        return adj;
    }

    public List<Integer> getAdj(int v) {
        return adj.get(v);
    }

    public void addEdge(int u, int v) {
        adj.get(u).add(v); // Adiciona uma aresta de u para v
        adj.get(v).add(u); // Grafo    n o direcionado, adiciona de v para u tamb m
    }

    public void removeEdge(int u, int v) {
        adj.get(u).remove(Integer.valueOf(v));
        adj.get(v).remove(Integer.valueOf(u));
    }

    public void printGraph() {
        for (int i = 0; i < V; i++) {
            System.out.print(i + ": ");
            for (int neighbor : adj.get(i)) {
                System.out.print(neighbor + " ");
            }
            System.out.println();
        }
    }

    public boolean isConnected() {
        boolean[] visited = new boolean[V];
        int start = -1;

        // Verifica se h  algum v rtice isolado (sem nenhuma aresta)
```

```
64          for (int i = 0; i < V; i++) {
65              if (adj.get(i).isEmpty()) {
66                  return false; // V rtice isolado     grafo n o     conectado
67              }
68          }
69
70          // Encontra um v rtice com pelo menos uma aresta
71          for (int i = 0; i < V; i++) {
72              if (!adj.get(i).isEmpty()) {
73                  start = i;
74                  break;
75              }
76          }
77          if (start == -1)
78              return true; // Grafo vazio    considerado conectado
79
80          dfs(start, visited); // Realiza busca em profundidade
81
82          for (int i = 0; i < V; i++) {
83              if (!adj.get(i).isEmpty() && !visited[i])
84                  return false; // Se algum v rtice com arestas n o foi visitado, o
                        grafo n o    conectado
85          }
86          return true;
87      }
88
89      private boolean isConnectedForNaive() {
90          boolean[] visited = new boolean[V];
91
92          // Encontra um v rtice com pelo menos uma aresta
93          int start = -1;
94          for (int i = 0; i < V; i++) {
95              if (!adj.get(i).isEmpty()) {
96                  start = i;
97                  break;
98              }
99          }
100
101         // Grafo sem arestas    considerado conectado
102         if (start == -1)
103             return true;
104
105         // Realiza DFS a partir do v rtice encontrado
106         dfs(start, visited);
107
108         // Verifica se todos os v rtices com pelo menos uma aresta foram visitados
109         for (int i = 0; i < V; i++) {
110             if (!adj.get(i).isEmpty() && !visited[i]) {
111                 return false;
112             }
113         }
114
115         return true;
116     }
117
118     private void dfs(int v, boolean[] visited) {
119         visited[v] = true;
120         for (int neighbor : adj.get(v)) {
121             if (!visited[neighbor]) {
122                 dfs(neighbor, visited); // Chamada recursiva para os vizinhos
123             }
124         }
125     }
126
127     // Naive
128     public List<int[]> findBridgesNaive() {
```

```
129         List<int[]> bridges = new ArrayList<>();
130 
131         for (int u = 0; u < V; u++) {
132             for (int v : new ArrayList<>(adj.get(u))) {
133                 if (u < v) { // Evita checagem duplicada
134                     Graph clone = this.clone(); // Clona o grafo atual
135                     clone.removeEdge(u, v); // Remove no clone
136                     if (!clone.isConnectedForNaive()) {
137                         bridges.add(new int[] { u, v }); // Se a remo  o desconecta
                                ,    ponte
138                     }
139                 }
140             }
141         }
142 
143         return bridges;
144     }
145 
146     // Tarjan
147     public List<int[]> findBridgesTarjan() {
148         boolean[] visited = new boolean[V];
149         int[] disc = new int[V], low = new int[V];
150         List<int[]> bridges = new ArrayList<>();
151         Arrays.fill(disc, -1); // Inicializa tempos de descoberta
152         int time = 0;
153         for (int i = 0; i < V; i++) {
154             if (disc[i] == -1) {
155                 tarjanDFS(i, -1, visited, disc, low, time, bridges);
156             }
157         }
158         return bridges;
159     }
160 
161     private void tarjanDFS(int u, int parent, boolean[] visited, int[] disc, int[]
        low, int time, List<int[]> bridges) {
162         visited[u] = true;
163         disc[u] = low[u] = ++time; // Inicializa tempos de descoberta e menor
            ancestral
164 
165         for (int v : adj.get(u)) {
166             if (v == parent)
167                 continue; // Ignora aresta para o pai
168 
169             if (!visited[v]) {
170                 tarjanDFS(v, u, visited, disc, low, time, bridges);
171                 low[u] = Math.min(low[u], low[v]); // Atualiza low[u] com o menor low
                    [v]
172 
173                 if (low[v] > disc[u]) {
174                     bridges.add(new int[] { u, v }); // Ponte detectada
175                 }
176             } else {
177                 low[u] = Math.min(low[u], disc[v]); // Atualiza low[u] com tempo de
                    descoberta de v
178             }
179         }
180     }
181 
182 }
183 
184 // Gerador de grafo aleat rio
185 class GeradorGrafo {
186     public static Graph gerarGrafoAleatorio(int V, int E) {
187         Graph graph = new Graph(V);
188         Random rand = new Random();
```

```
189         Set<String> edges = new HashSet<>(); // Hashset para evitar arestas
                duplicadas
190
191         while (edges.size() < E) {
192             int u = rand.nextInt(V);
193             int v = rand.nextInt(V);
194             // Garante que n o haja loops nem arestas duplicadas
195             if (u != v && !edges.contains(u + "-" + v) && !edges.contains(v + "-" + u
                    )) {
196
197                 // Adiciona aresta ao grafo
198                 graph.addEdge(u, v);
199
200                 // Adiciona aresta ao Hashset de controle
201                 edges.add(u + "-" + v);
202             }
203         }
204
205         return graph;
206     }
207     public static Graph gerarGrafoEuleriano(int V, int E) {
208         Graph g = new Graph(V);
209         Random rand = new Random();
210
211         if (E < V || E % 2 != 0) {
212             throw new IllegalArgumentException("N mero de arestas deve ser pelo
                    menos igual ao n mero de v rtices e par");
213         }
214
215         // Garante um ciclo inicial (conectado e com graus pares)
216         for (int i = 0; i < V; i++) {
217             int next = (i + 1) % V;
218             g.addEdge(i, next);
219             E--;
220         }
221
222         // Adiciona arestas extras mantendo os graus pares
223         while (E > 0) {
224             int u = rand.nextInt(V);
225             int v = rand.nextInt(V);
226
227             if (u != v) {
228                 // Verifica se a aresta j  existe percorrendo a lista de adjac ncia
229                 boolean jaExiste = false;
230                 for (int vizinho : g.getAdj().get(u)) {
231                     if (vizinho == v) {
232                         jaExiste = true;
233                         break;
234                     }
235                 }
236
237                 if (!jaExiste) {
238                     g.addEdge(u, v);
239                     E--;
240                 }
241             }
242         }
243
244         return g;
245     }
246
247 }
248
249 class FleuryAlgorithm {
250     private Graph graph;
251
```

```
252     public FleuryAlgorithm(Graph graph) {
253         this.graph = graph;
254     }
255 
256     // Retorna tipo do grafo: 0 = n o euleriano, 1 = semi-euleriano, 2 = euleriano
257     private int tipoGrafo() {
258         int odd = 0;
259         for (int i = 0; i < graph.getV(); i++) {
260             if (graph.getAdj(i).size() % 2 != 0) {
261                 odd++;
262             }
263         }
264         if (odd > 2)
265             return 0;
266         return (odd == 2) ? 1 : 2;
267     }
268 
269     // Encontra caminho usando Fleury com verifica  o de ponte via Tarjan ou For a
270     // Bruta
271     public List<Integer> encontrarCaminhoEuleriano(boolean usarTarjan) {
272         int tipo = tipoGrafo();
273         if (tipo == 0)
274             return null; // N o existe caminho euleriano
275 
276         int start = 0;
277         if (tipo == 1) {
278             for (int i = 0; i < graph.getV(); i++) {
279                 if (graph.getAdj(i).size() % 2 != 0) {
280                     start = i;
281                     break;
282                 }
283             }
284         }
285 
286         List<Integer> caminho = new ArrayList<>();
287         dfsFleury(start, caminho, usarTarjan);
288         return caminho;
289     }
290 
291     private void dfsFleury(int u, List<Integer> caminho, boolean usarTarjan) {
292         for (int v : new ArrayList<>(graph.getAdj(u))) {
293             if (!isBridge(u, v, usarTarjan)) {
294                 graph.removeEdge(u, v);
295                 dfsFleury(v, caminho, usarTarjan);
296             }
297         }
298         caminho.add(u);
299     }
300 
301     private boolean isBridge(int u, int v, boolean usarTarjan) {
302         graph.removeEdge(u, v);
303         boolean isBridge = false;
304 
305         if (!graph.isConnected()) {
306             isBridge = true;
307         } else if (usarTarjan) {
308             for (int[] bridge : graph.findBridgesTarjan()) {
309                 if ((bridge[0] == u && bridge[1] == v) || (bridge[0] == v && bridge
                        [1] == u)) {
310                     isBridge = true;
311                     break;
312                 }
313             }
314         } else {
315             for (int[] bridge : graph.findBridgesNaive()) {
```

```
                if ((bridge[0] == u && bridge[1] == v) || (bridge[0] == v && bridge
                    [1] == u)) {
                    isBridge = true;
                    break;
                }
            }
        }

        graph.addEdge(u, v); // Reinsere a aresta
        return isBridge;
    }

    public String tipoEuleriano() {
        int tipo = tipoGrafo();
        switch (tipo) {
            case 0:
                return "N o Euleriano";
            case 1:
                return "Semi-Euleriano";
            case 2:
                return "Euleriano";
            default:
                return "Desconhecido";
        }
    }

    // imprime o caminho Euleriano
    public void printarCaminhoEuleriano(boolean usarTarjan) {
        List<Integer> caminho = encontrarCaminhoEuleriano(usarTarjan);

        if (caminho == null || caminho.isEmpty()) {
            System.out.println("N o existe caminho Euleriano.");
            return;
        }

        System.out.println("Caminho Euleriano:");
        for (int i = caminho.size() - 1; i >= 0; i--) {
            System.out.print(caminho.get(i));
            if (i != 0) System.out.print(" -> ");
        }
        System.out.println(); // quebra de linha
    }


}
```

Listing 1: Gerador de Grafos, Naive, Tarjan, Fleury

Código Main

```
package Grafo;

import java.util.*;

public class Main {

    public static void main(String[] args) {
        int[] tamanhos = { 10 }; // Numero de vertices para teste

        for (int vertices : tamanhos) {
            int arestas = vertices*2; // N mero de arestas proporcional aos
                v rtices; diminuir quando tiver muitos v rtices

            System.out.println("\n=== Testando grafo com " + vertices + " v rtices e
                 " + arestas + " arestas ===");
```

```
14
15              // Gerar grafo e imprimir conectividade
16              Graph g = GeradorGrafo.gerarGrafoAleatorio(vertices, arestas); //gerar
                    grafo euleriano
17              // Graph g = GeradorGrafo.gerarGrafoEuleriano(10, 20); //gerar grafo
                    euleriano
18              Graph salva = g.clone();
19
20              System.out.println("\nGrafo gerado:");
21              g.printGraph(); //Printa o grafo gerado
22              System.out.println("Conectividade: " + (g.isConnected() ? "Conexo" : "
                    Desconexo"));
23
24              FleuryAlgorithm fleuryNaive = new FleuryAlgorithm(g);
25              System.out.println("Tipo do grafo (Naive): " + fleuryNaive.tipoEuleriano
                    ());
26              long startNaive = System.nanoTime();
27              var caminhoNaive = fleuryNaive.encontrarCaminhoEuleriano(false);
28              long endNaive = System.nanoTime();
29              /*
30               * System.out.println("\nPontes encontradas com Na ve:");
31               * for (int[] bridge : bridgesNaive) {
32               * System.out.println(bridge[0] + " - " + bridge[1]);
33               * }
34               */
35              System.out.printf("Naive - Tempo: %.2f ms | Caminho Euleriano: %s%n",
36                      (endNaive - startNaive) / 1e6,
37                      (caminhoNaive != null ? "Existe" : "N o existe"));
38
39              // Regenerar o grafo para garantir igualdade de condi  es
40              g = salva;
41              FleuryAlgorithm fleuryTarjan = new FleuryAlgorithm(g);
42              System.out.println("Tipo do grafo (Tarjan): " + fleuryTarjan.
                    tipoEuleriano());
43              long startTarjan = System.nanoTime();
44              List<int[]> bridgesTarjan = g.findBridgesTarjan();
45              var caminhoTarjan = fleuryTarjan.encontrarCaminhoEuleriano(true);
46              long endTarjan = System.nanoTime();
47              System.out.printf("Tarjan - Tempo: %.2f ms | Caminho Euleriano: %s%n",
48              (endTarjan - startTarjan) / 1e6,
49              (caminhoTarjan != null ? "Existe" : "N o existe"));
50
51      if (caminhoTarjan != null) {
52          System.out.print("Caminho: ");
53          for (int i = caminhoTarjan.size() - 1; i >= 0; i--) {
54              System.out.print(caminhoTarjan.get(i));
55              if (i != 0) System.out.print(" -> ");
56          }
57          System.out.println();
58      }
59      System.out.print("Pontes: \n");
60              for (int[] bridge : bridgesTarjan) {
61                  System.out.println(bridge[0] + " - " + bridge[1]);
62              }
63
64          }
65      }
66  }
```

Listing 2: Main que chama todos metodos e printa o grafo gerado, juntamente com pontes encontradas e caminho euleriano (se existirem)