Nome: Leonardo Botrel Tobias **Matrícula:** 201622040333
Data: 13/10/2017 Número: 05
Título da prática: Implementação em JAVA do TAD MatrizAdj e da busca em Profundidade.

Código da classe JGrafo:

```
public class JGrafo {

   public static class Aresta {
       private int v1, v2, peso;//Classe  Aresta com os atributos
v1, v2 e peso
       //Construtor para inicializar os atributos
       public Aresta (int v1, int v2, int peso) {
           this.peso = peso;
           this.v1 = v1;
           this.v2 = v2;
       }

       //Métodos para obter o valor dos atributos
       public int getPeso () {
           return this.peso;
       }
       public int getV1 () {
           return v1;
       }
       public int getV2 () {
           return v2;
       }
   }

   private int mat[][];// Pesos das arestas
   private int numVertices;// Atributo indicando o número de
vértices
   private int pos[];//Posição atual ao se percorrer  os adjs de um
vértice v

   //Construtor para inicializar os atributos
   public JGrafo (int numVertices) {
       //Instancialização de mat[][] e pos[]
       this.mat = new int[numVertices][numVertices];
       this.pos = new int [numVertices];
       this.numVertices = numVertices;

       // Inicialização de mat[][] e pos[]
       for (int i = 0; i < this.numVertices; i++) {
```

```
            for (int j = 0; j < this.numVertices; j++)
                this.mat[i][j] = 0;
            this.pos[i] = -1;
        }
    }

    //Inserção de uma nova aresta em mat[][]
    public void insereAresta (int v1, int v2, int peso) {
        this.mat[v1][v2] = peso;
    }

    //Verifica se existe uma aresta cujas incidências são passadas
por parâmetro
    public boolean existeAresta (int v1, int v2) {
        return (this.mat[v1][v2] > 0);
    }

    // Retorna o primeiro objeto Aresta em que o parâmetro v
participa
    public Aresta primeiroListaAdj (int v) {
        //Retorna a primeira aresta que o vértice v participa ou
        //null se a lista de adjacência de v for vazia
        this.pos[v] = -1;
        return this.proxAdj (v);// inicializa o percurso das
adjacências de v
    }

    public Aresta proxAdj (int v) {
        //Retorna a próxima aresta que o vértice v participa ou
        //null se a lista de adjacência de v estiver no fim

        this.pos[v]++;//Posiciona p[] na próxima aresta a ser
pesquisada

        //Verifica de p[v] é uma adjacência válida e Se a adjacência
existe. Se não existir verifica a próxima adjacência
        while ((this.pos[v] < this.numVertices) &&
(this.mat[v][this.pos[v]] == 0))
            this.pos[v]++;

        if (this.pos[v] == this.numVertices)//Se a adjacência for
inválida retorna null
            return null;
        else// Caso contrário retorna um objeto Aresta com a aresta
procurada
```

```
            return new Aresta(v, this.pos[v],
this.mat[v][this.pos[v]]);
    }

    public Aresta retiraAresta (int v1, int v2) {
        if (this.mat[v1][v2] == 0)//Aresta não existe
            return null;
        else {
            Aresta aresta = new Aresta(v1, v2, this.mat[v1][v2]);
            this.mat[v1][v2] = 0;
            return aresta;
        }
    }

    public void imprime () {
        System.out.print("  ");

        for (int i = 0; i < this.numVertices; i++)
            System.out.print(i + "  ");
        System.out.println();

        for (int i = 0; i < this.numVertices; i++) {
            System.out.print(i  + "  ");

            for (int j = 0; j < this.numVertices; j++)
                System.out.print(this.mat[i][j] + "  ");
            System.out.println();
        }
    }

    public int getNumVertices (){
        return this.numVertices;
    }

    public boolean listaAdjVazia (int u) {
        int n = 0;
        for (int i = 0; i < this.numVertices; i++) {
            n += this.mat[u][i];
        }

        if (n == 0) {
            return true;
        }
        else
            return false;
    }
```

```
}
```

Código da classe JCiclo:

```
public class JCiclo {

   public static final byte branco = 0;
   public static byte cinza = 1;
   public static byte preto = 2;
   private int d[] , t [] , antecessor[];
   private JGrafo grafo;
   private boolean retorno = false;

   //Construtor, inicializa os atributos
   public JCiclo (JGrafo grafo) {
        this.grafo = grafo; //inicializa o grafo
        int n = this.grafo.getNumVertices();//número de vertices

        d = new int[n];
        t = new int[n];
        antecessor = new int[n];
   }

   private int visitaDfs ( int u, int tempo, int cor[]) {
        cor[u] = cinza;//O vertice u acabou de ser descoberto
        this.d[u] = ++tempo;

        //Explora a aresta(u,v)
        if (!this.grafo.listaAdjVazia (u)) {
            JGrafo.Aresta a = this.grafo.primeiroListaAdj (u);

            while (a != null) {
                int v = a.getV2 ();
                //Caso o vértice seja branco, a aresta (u,v) para
esse vertice é uma aresta
                //de árvore e portando não indica se o grafo tem
ciclo.
                if (cor[v] == branco) {
                    this.antecessor[v] = u;
                    tempo = this.visitaDfs (v, tempo, cor);
                }
                //caso contrario, se o vertice tiver cor cinza, entao
a aresta(u,v)
                //para esse vértice é de retorno, portanto esse grafo
obrigatoriamente
```

```
                //tem ciclo, e a condição de existencia de aresta de
retorno se torna
                //verdadeira
                else if (cor[v] == cinza){
                    this.retorno = true;
                }

                a = this.grafo.proxAdj (u);
            }
        }

        cor[u] = preto;
        this. t [u] = ++tempo;
        return tempo;
    }

    //Realiza a busca em profundidade
    public void buscaEmProfundidade () {
        int tempo = 0; //Cria e inicializa o tempo
        int cor[] = new int[this.grafo.getNumVertices ()];//Cria e
inicializa o vetor de cores

        for (int u = 0; u < grafo.getNumVertices (); u++) {
            cor[u] = branco;//Atribui a cor branca aos vertices
            this.antecessor[u] = -1;//Inicializa o antecessor com -1
        }

        for (int u = 0; u < grafo.getNumVertices (); u++)
            if (cor[u] == branco)
                tempo = this.visitaDfs (u, tempo, cor);
    }

    public int d (int v) {
        return this.d[v];
    }

    public int t (int v) {
        return this. t [v];
    }

    public int antecessor ( int v) {
        return this.antecessor[v];
    }

    //Verifica se o grafo possui ciclo
    public boolean temCiclo () {
```

```
        this.buscaEmProfundidade();
        return this.retorno;
    }

}
```

Resultados obtidos nos grafos do item 4:

Há ciclo no primeiro grafo.

Não há ciclo no segundo grafo.