Centro Federal de Educação Tecnológica de Minas Gerais
CEFET-MG

Laboratório de Algoritmos e Estruturas de Dados II

Prática V:
Implementação em JAVA do TAD MatrizAdj

Data de entrega: 10/05/2019

Illyana Guimarães de Avelar

2019

•Código que implementa a classe JGrafo comentado

```
package Pratica05;

public class JGrafo{

  public static class Aresta{

    private int v1, v2, peso;

    public Aresta(int v1, int v2, int peso){
      this.v1 = v1; this.v2 = v2; this.peso = peso;
    }

    public int peso(){
        return this.peso;
    }
    public int v1(){
        return this.v1;
    }
    public int v2(){
        return this.v2;
    }
  }

  private int mat[][];     // matriz de pesos
  private int numVertices; //quantidade de vértices
  private int pos[];       // posicao atual ao se percorrer as adjacencias de um vertice

  //contrutor de JGrafo
```

```
 public JGrafo (int numVertices){

  this.mat = new int[numVertices][numVertices];//matriz de pesos com o numero de vertices^2
  this.pos = new int[numVertices];//lista com o numero de vertices
  this.numVertices = numVertices; //numero de vertices do grafo que será recebido
  for(int i = 0; i < this.numVertices; i++){//para cada linha da matriz e lista
   for(int j = 0; j < this.numVertices; j++)//e para cada coluna
    this.mat[i][j] = 0;//a matriz é iniciada com 0
   this.pos[i] = -1;//posicão atual recebe uma posição em que ela nunca estará ao visitar as adjacências de um
vértice
  }
 }

 public void insereAresta(int v1, int v2, int peso){
  this.mat[v1][v2] = peso;//insere o peso da aresta em sua respectiva posição
 }

 public boolean existeAresta(int v1, int v2){
  return (this.mat[v1][v2] > 0);//retorna se o valor corresponente a aresta foi inserido
 }

 public boolean listaAdjVazia(int v){
  for(int i =0; i < this.numVertices; i++)//para cada vértice
   if (this.mat[v][i] > 0)//verifica se a lista está vazia
    return false;//retorna falso caso não esteja
  return true;
 }

 public Aresta primeiroListaAdj(int v){
  //Retorna a primeira aresta que o vertice v participa
  //ou null se a lista de adjacencia de v for vazia
```

```
  this.pos[v] = -1;
  return this.proxAdj(v);
}

public Aresta proxAdj(int v){
  //Retorna a proxima aresta que o vertice v participa ou
  //retorna null se a lista de adjacencia de v estiver no fim
  this.pos[v] ++;
  while ((this.pos[v] < this.numVertices)&&(this.mat[v][this.pos[v]] == 0))
    this.pos[v]++;
  if (this.pos[v] == this.numVertices)
    return null;
  else
    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)//caso a aresta nao exista
    return null;//não faz nada e retorna nulo
  else{//caso exista, realiza a remoção
    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 numVertices(){
    return this.numVertices;
  }

  public JGrafo grafoTransposto(){
    JGrafo grafoT = new JGrafo(this.numVertices);
    for(int v = 0; v < this.numVertices; v++)
      if(!this.listaAdjVazia (v)){
        Aresta adj = this.primeiroListaAdj (v);
        while (adj != null){
          grafoT.insereAresta(adj.v2 (), adj.v1 (), adj.peso ());
          adj = this.proxAdj (v);
        }
      }
    return grafoT;
  }
}
```

•Código que implementa a classe JCiclo comentado

```
package Pratica05;
import java.util.*;

public class JCiclo {
  private ArrayList<Integer> path;//caminho percorrido no grafo
  private int cycle = 0;//variável para controlar a existência ou não de ciclos
  public static final byte branco = 0;//não foi visitado
  public static final byte cinza  = 1;//foi visitado
  public static final byte preto  = 2;//não está na fila
  private int d[], antecessor[];
  private JGrafo grafo;//instância de um grafo

  //metódo contrutor de JCiclo
  public JCiclo (JGrafo grafo){
    this.grafo = grafo;
    int n = this.grafo.numVertices();
    this.d = new int[n];
    this.antecessor = new int[n];
  }

  //Percorre todo o grafo até que não hajam elementos na fila
  private void visitaBfs (int u, int cor[]) throws Exception {
    cor[u] = cinza;//vertice foi visitado
    this.d[u] = 0;
    path.add(u);//adiciona o vértice u ao caminho
    Fila fila = new Fila();//cria uma nova fila
    fila.enfileira(new Integer (u));//enfilera o vértice que será visitado

    while (!fila.vazia ()){
```

```
    Integer aux = (Integer)fila.desenfileira();//retira um item para percorre-lo
    u = aux.intValue();//u recebe o item retirado da fila
    if(!this.grafo.listaAdjVazia(u)){//enquanto a lista de adjacência não estiver vazia
      JGrafo.Aresta a = this.grafo.primeiroListaAdj(u);//a recebe o primeiro item da lista

      while(a != null){//enquanto houverem adjacências
        for(int i : path){//para cada item do caminho
          if(i==a.v2())//se ha um ciclo na lista
            this.cycle = 1;//então cycle é alterada
        }
        int v = a.v2();
        if(cor[v] == branco){//se o vértice não foi percorrido ainda
          cor[v] = cinza;//ele será nesse momento então recebe a cor cinza
          this.d[v] = this.d[u] + 1;
          this.antecessor[v] = u;
          fila.enfileira(new Integer (v));
        }
        a = this.grafo.proxAdj (u);
      }
    }
    path.remove(path.size()-1);//remove os vertices do caminho
    cor[u] = preto;//depois de percorrer todas as adjacências do vértice, a cor muda para preto

  }
}
//realiza a busca em largura
public void buscaEmLargura () throws Exception {
  int cor[] = new int[this.grafo.numVertices ()];//vetor de cores com a quantidade de vertices
  for (int u = 0; u < grafo.numVertices (); u++){//para cada vértice
    cor[u] = branco; this.d[u] = Integer.MAX_VALUE;//inicia com nenhum vértice visitado
```

```
    this.antecessor[u] = -1;
  }
  //para todos os vértices
  for (int u = 0; u < grafo.numVertices (); u++){
    this.path = new ArrayList<Integer>();//cria um vetor para o caminho
    if (cor[u] == branco) this.visitaBfs (u, cor);//se ainda não foi visitado,
    //é mandado para o método que visita os vértices
  }
}

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

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

public void imprimeCaminho(int origem, int v){
  if(origem == v)
    System.out.println (origem);
  else if(this.antecessor[v] == -1)
    System.out.println ("Nao existe caminho de" + origem +" ate" + v);
  else{
    imprimeCaminho(origem, this.antecessor[v]);
    System.out.println (v);
  }
}
//retorna o valor de cycle que será usado na main para verificar se existe ciclo
public int cycle(){
  return this.cycle();}}
```

•Resultado obtido nos grafos do item 4.