Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação
AD2 de Programação III
2° semestre de 2016

Exercício (ENTREGAR OS ARQUIVOS EM MÍDIA, PARA FINS DE TESTE, JUNTAMENTE COM A AD IMPRESSA):
Considerando as seguintes definições [1]:
(1) Um grafo é conexo se existe um caminho entre qualquer par de nós, caso contrário ele é chamado desconexo; e
(2) Um grafo é não-orientado quando não existem direções nas arestas.

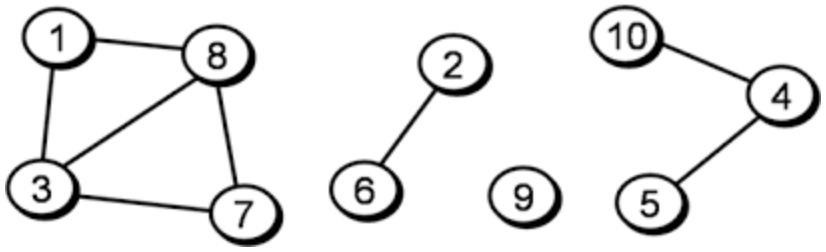
Escreva um programa em JAVA para resolver o problema de encontrar as componentes conexas de um grafo não-orientado. Dados um grafo G=(V,E) não orientado, onde V é o conjunto de nós e E é o conjunto de arestas, Os componentes conexos de um grafo são os subgrafos conexos maximais deste grafo.

Você deve desenvolver um algoritmo que, dado um arquivo de entrada (**LIDO SOMENTE UMA VEZ PELO SEU PROGRAMA**), contendo o grafo não-orientado, forneça a quantidade de componentes conexas e estas componentes.

Para o melhor entendimento do problema a ser resolvido, considere que você receba o seguinte arquivo como parâmetro de entrada:

```
1 2 3 4 5 6 7 8 9 10
1 3
1 8
2 6
3 7
3 8
4 5
4 10
```

que representa o grafo a seguir:



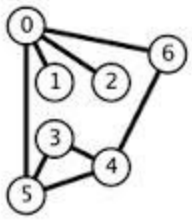
seu programa deve obter como resposta:

```
quantidade: 4
1 3 7 8
2 6
4 5 10
9
```

Para um outro exemplo de arquivo:

```
0 1 2 3 4 5 6
0 1
0 2
0 5
0 6
3 4
3 5
4 5
4 6
```

que representa o grafo a seguir:



seu programa deve obter como resposta:

```
quantidade: 1
0 1 2 3 4 5 6
```

LEMBRE-SE: SEU PROGRAMA DEVE EXECUTAR COM QUAISQUER DADOS INFORMADOS COMO PARÂMETROS DE ENTRADA. SE O SEU PROGRAMA RESOLVER SOMENTE O PROBLEMA DO EXERCÍCIO SUPRACITADO, SUA QUESTÃO SERÁ TOTALMENTE DESCONTADA.

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest e C. Stein, “Algoritmos: teoria e prática”, Editora Campus, 2012.

RESPOSTA:

```
import java.io.*;
import java.util.*;

/*
A estrutura Vizinho é composta de:
  - no vizinho
  - referência para o próximo vizinho
(para fazer o encadeamento de vizinhos)
Esta classe é formada de construtor  e
método toString. O último é usado para verificar se a
estrutura está sendo criada de maneira correta.
*/
class Vizinho{
  int no_viz;
  Vizinho prox;

  Vizinho(int c){
    no_viz = c;
    prox = null;
  }
```

```
public String toString(){ return no_viz + " "; }
}

/*
A estrutura Lista é composta de:
  - no de origem, sua cor
  - a lista de seus vizinhos
  - o proximo no do grafo
Esta estrutura é montada para ser usada no grafo
(estrutura principal criada para resolver o problema).
Esta classe tem o método construtor, um método para testar se o
vizinho já está na lista, um método para inserir vizinho na
primeira posição da lista de vizinhos, e o método toString.
*/
class Lista{
  int no, cor;
  Vizinho prox_viz;
  Lista prox_no;

  Lista(int c){
    no = c;
    cor = 0;
    prox_viz = null;
    prox_no = null;
  }

  Vizinho pertence(int no){
    Vizinho resp = prox_viz;
    while((resp != null) && (no != resp.no_viz))
      resp = resp.prox;
    return resp;
  }

  void ins_Viz(int c){
    Vizinho v = pertence(c);
    if(v != null) return;
    v = new Vizinho(c);
    v.prox = prox_viz;
    prox_viz = v;
  }

  public String toString(){
    String resp = no + "(" + cor + "): \n";
    Vizinho p = prox_viz;
    while(p != null){
      resp += p.toString();
      p = p.prox;
    }
    return resp + "\n";
  }
}
```

```
/*
A estrutura Grafo é desenvolvida para resolver o problema. Ela é
composta da referência para o primeiro nó da lista. Tem os seguintes
métodos:
  - construtor;
  - para verificar se um no existe na lista de nos do grafo;
  - para inserir nos; e
  - toString.
*/
class Grafo{
  Lista prim;

  Grafo(){  prim = null; }

  Lista pertence(int no){
    Lista resp = prim;
    while((resp != null) && (no != resp.no)) resp = resp.prox_no;
    return resp;
  }

  void insere(int no){
    Lista p = pertence(no);
    if(p == null){
      p = new Lista(no);
      Lista q = prim;
      if(q == null){
        prim = p;
        return;
      }
      while(q.prox_no != null) q = q.prox_no;
      q.prox_no = p;
    }
  }

  void insere(int no1, int no2){
    Lista p = pertence(no1);
    p.ins_Viz(no2);
    Lista q = pertence(no2);
    q.ins_Viz(no1);
  }

  public String toString(){
    String resp = "";
    Lista p = prim;
    while(p != null){
      resp += p.toString();
      p = p.prox_no;
    }
    return resp;
  }
}
```

```
public class AD2_POO_2016_2{
  public static void main(String[] args) throws IOException{
    BufferedReader in;
    in = new BufferedReader(new FileReader(args[0]));
    try {
      Grafo g = new Grafo();
      String s, vs[];
      s = in.readLine();
      vs = s.split(" ");
      for(int i = 0; i < vs.length; i++)
        g.insere(Integer.parseInt(vs[i]));
      while((s = in.readLine()) != null){
        vs = s.split(" ");
        g.insere(Integer.parseInt(vs[0]),Integer.parseInt(vs[1]));
      }
      in.close();
      contaComponentesConexas(g);
    }
    catch (Exception e){
      System.out.println("Excecao\n");
    }
  }

  static int retornaNoCorZero(Grafo g){
    Lista p = g.prim;
    while(p != null){
      if(p.cor == 0) return p.no;
      p = p.prox_no;
    }
    return -1;
  }

  static void contaComponentesConexas(Grafo g){
    int cor = 0, ind = retornaNoCorZero(g);
    while(ind != -1){
      cor++;
      Deque<Lista> nos = new ArrayDeque<Lista>();
      nos.addLast(g.pertence(ind));
      while(nos.size() != 0){
        Lista aux = nos.remove();
        aux.cor = cor;
        Vizinho viz = aux.prox_viz;
        while(viz != null){
          Lista p = g.pertence(viz.no_viz);
          if(p.cor == 0) nos.addLast(p);
         viz = viz.prox;
        }
      }
      ind = retornaNoCorZero(g);
    }
```

```
    System.out.println("quantidade: " + cor);
    ind = 1;
    while (ind <= cor){
      Lista p = g.prim;
      while(p != null){
        if(p.cor == ind) System.out.print(p.no + " ");
        p = p.prox_no;
      }
      System.out.println();
      ind++;
    }
  }
}
```