Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

**Curso de Tecnologia em Sistemas de Computação**
**AD2 de Programação III**
**2° semestre de 2014**

**Nome:**
**Matrícula:**
**Pólo:**

**Exercício (ENTREGAR OS ARQUIVOS EM MÍDIA, PARA FINS DE TESTE, JUNTAMENTE COM A AD IMPRESSA):**
Considere que sua empresa seja contratada por uma outra companhia para resolver o problema de projeto de redes a 2-caminhos. Este problema consiste em realizar ligações de vários pares de dispositivos eletrônicos usando OU uma ligação direta, OU um outro dispositivo eletrônico entre este par, MINIMIZANDO o custo total destas ligações. Uma maneira de resolver este problema é, a partir de cada par de dispositivos, descobrir qual é a ligação mais barata possível. Neste momento, o custo das ligações que já fazem parte da solução passam a ser zero, indicando que esta ligação pode ser usada para ligar mais de um par de dispositivos.

Seu programa deve receber, como parâmetro de entrada, um arquivo contendo as ligações possíveis e os custos associados a essas ligações, e um outro arquivo de entrada indicando os pares que devem ser ligados OBRIGATORIAMENTE. Seu programa deve retornar um arquivo contendo as ligações usadas e o custo da solução. Um EXEMPLO dos arquivos de entrada neste formato seriam (OBSERVE QUE SEU PROGRAMA DEVE FUNCIONAR PARA QUAISQUER ARQUIVOS NOS FORMATOS ANTERIORMENTE DEFINIDOS):

```
//ARQUIVO DE LIGAÇÕES POSSÍVEIS
4           //NÚMERO DE DISPOSITIVOS
1 2 3       //O DISPOSITIVO 1 LIGA-SE AO DISPOSITIVO 2 COM CUSTO 3
1 3 1
1 4 2
2 3 1
2 4 3
3 4 3

//ARQUIVOS DE LIGAÇÕES QUE DEVEM SER FEITAS PELO SEU SOFTWARE
2           //NÚMERO DE LIGAÇÕES OBRIGATÓRIAS
1 2         //SEU SOFTWARE DEVE LIGAR O DIPOSITIVO 1 AO 2
3 4
```

Logo após a leitura, a solução deve ser calculada e gravada no arquivo **resp.txt**. Para o exemplo acima, o arquivo de resposta seria composto das ligações **{1-3, 2-3, 1-4}** com custo **4**. Neste caso, a ligação **1-3** foi usada duas vezes. Por questões de desempenho, seu programa deve ler os arquivos de entrada SOMENTE uma vez.

**RESPOSTA:**

```
import java.io.*;

public class AD2_2014_2{
  public static void main(String[] args) throws Exception{
    BufferedReader in_grafo, in_demanda;
    //abertura dos arquivos contendo o grafo e a demanda
    in_grafo = new BufferedReader(new FileReader(args[0]));
    in_demanda = new BufferedReader(new FileReader(args[1]));
    Grafo G = null;
    Demanda D = null;

    try{
      String s = null, vs[];
      s = in_grafo.readLine();
      //leitura do tamanho do grafo e sua alocação
      if(s != null) G = new Grafo(Integer.parseInt(s));
      while((s = in_grafo.readLine()) != null){
       vs = s.split(" ");
     //preenchimento da estrutura do grafo
      G.insereAresta(Integer.parseInt(vs[0])-1, Integer.parseInt(vs[1])-
1, Integer.parseInt(vs[2]));
      }
      in_grafo.close();

      //leitura da quantidade de demandas e sua alocação
      s = in_demanda.readLine();
      if(s != null) D = new Demanda(Integer.parseInt(s));
      while((s = in_demanda.readLine()) != null){
      vs = s.split(" ");
      //preenchimento da estrutura de demanda
    D.insere(Integer.parseInt(vs[0]) - 1, Integer.parseInt(vs[1]) - 1);
      }
      in_demanda.close();

      //criação da solução a partir do grafo e das demandas
      Solucao sol = calculaSolucao(G, D);
      BufferedWriter out;
      //escrita, no arquivo de saida, da solução e do custo
      out = new BufferedWriter(new FileWriter("resp.txt"));
      out.write(sol.toString());
      out.close();
    }catch (Exception e) { System.out.println("Excecao..."); }
  }

  //função estática que realiza a lógica do programa
  static Solucao calculaSolucao(Grafo G, Demanda D){
    //foi feita a cópia do grafo para poder alterar os pesos das
    //arestas, toda vez que a aresta participa, pela primeira vez,
    //de uma demanda.
    Grafo copia = G.copia();
    //cada solução pode ter, no máximo, o dobro do conjunto de demandas
    Solucao sol = new Solucao(D.retornaTamanho());
    int i, j, custo, aux, no_interno;
```

```
    Par par;
    for(i = 0; i < D.retornaTamanho(); i++){
      //para cada par, o programa informa que a aresta que liga os nós
      //da demanda diretamente é considerada a menor
      par = D.retornaPar(i);
      custo = copia.retornaValor(par.origem, par.destino);
      no_interno = -1;
      for(j = 0; j < G.retornaTamanho(); j++){
        if((par.origem != j) || (par.destino != j)){
         aux = copia.retornaValor(par.origem, j) + copia.retornaValor(j,
par.destino);
          //verifica-se se o custo da aresta origem-j e j-destino é
          //menor que o custo da aresta origem-destino. Se for, altera-
          //se o custo e armazena-se o nó j.
          if(aux < custo){
            no_interno = j;
            custo = aux;
          }
        }
      }

      //se o menor caminho é composto por duas arestas e estas arestas
      //ainda não foram usadas, inclua-as na solução e zere os custos
      //dela no grafo que é uma cópia do grafo original
      if(no_interno != -1){
        if(copia.retornaValor(par.origem, no_interno) != 0){
          copia.insereAresta(par.origem, no_interno, 0);
          sol.insereSolucao(new Par(par.origem, no_interno));
        }
        if(copia.retornaValor(no_interno, par.destino) != 0){
          copia.insereAresta(no_interno, par.destino, 0);
          sol.insereSolucao(new Par(no_interno, par.destino));
        }
      }
      //mesmo processo que o anterior. A diferença é o fato de usar a
      //aresta direta
      else{
        if(copia.retornaValor(par.origem, par.destino) != 0){
          copia.insereAresta(par.origem, par.destino, 0);
          sol.insereSolucao(new Par(par.origem, par.destino));
        }
      }
    }
    //calcula-se o custo da solução e retorne a solução obtida
    sol.calculaCusto(G);
    return sol;
  }
}


//classe que indica os pares de demanda.
class Par{
  int origem, destino;
```

```
  Par(int o, int d){
    origem = o;
    destino = d;
  }

  //impressão dos pares como solicitado no exercício
  public String toString(){
    String resp = (origem + 1) + "-" + (destino + 1) + " ";
    return resp;
  }
}


//classe Demanda composta do vetor de pares e o número de pares, nd.
//Posso usar vetor porque sabe-se a quantidade de demandas previamente
class Demanda{
  int cont = 0, nd;
  Par[] pares;

  Demanda(int tam){
    nd = tam;
    pares = new Par[nd];
  }

  int retornaTamanho(){ return nd; }

  Par retornaPar(int ind){ return pares[ind]; }

  void insere(int o, int d){
    if(cont < nd) pares[cont++] = new Par(o,d);
    else System.out.println("Erro de insercao de demanda...");
  }

  //impressão de todos os pares do conjunto de demandas
  public String toString(){
    String resp = "";
    for(int i = 0; i < cont; i++) resp += pares[i].toString();
    return resp + "\n";
  }
}


//classe Grafo composta da matriz de adjacências e o número de nós, ng.
//Posso usar matriz porque sabe-se a quantidade de nós previamente
class Grafo{
  int ng;
  int mat[][];


  //inicialização do grafo com todas arestas iguais a zero
  Grafo(int tam){
    ng = tam;
    mat = new int[ng][];
    int i, j;
```

```
    for(i = 0; i < ng; i++)
      mat[i] = new int[ng];
    for(i = 0; i < ng; i++)
      for(j = 0; j <= i; j++)
        mat[i][j] = mat[j][i] = 0;
  }

  void insereAresta(int l, int c, int v){
    if((l < 0) || (l >= ng) || (c < 0) || (c >= ng))
      System.out.println("Erro de indices...");
    else mat[l][c] = mat[c][l] = v;
  }

  int retornaTamanho(){ return ng; }

  int retornaValor(int l, int c){ return mat[l][c]; }

  //método para copiar o grafo original, a fim de usar a cópia no
  //método estático que calcula a solução
  Grafo copia(){
    Grafo g = new Grafo(ng);
    int i, j;
    for(i = 0; i < ng; i++)
      for(j = 0; j < ng; j++)
        g.insereAresta(i, j, mat[i][j]);
    return g;
  }

  //impressão do grafo por linha
  public String toString(){
    String resp = "";
    for(int i = 0; i < ng; i++){
      for(int j = 0; j < ng; j++) resp += mat[i][j] + " ";
      resp += "\n";
    }
    return resp;
  }
}


//classe Solucao possui o vetor de pares, cujo o tamanho máximo é o
//dobro do número de demandas. Usa-se, também, um campo que armazena
//o custo da solução
class Solucao{
  int cont, custo;
  Par[] pares_sol;

  Solucao(int n){
    cont = custo = 0;
    pares_sol = new Par[n * 2];
  }
```

```
  //método usado para calcular o custo da solução. É necessário
  //informar o grafo como parâmetro de entrada porque só esta
  //estrutura tem o valor dos custos das arestas
  void calculaCusto(Grafo g){
   custo = 0;
   for(int i = 0; i < cont; i++)
   if(pares_sol[i] != null)
    custo += g.retornaValor(pares_sol[i].origem, pares_sol[i].destino);
 }

  void insereSolucao(Par p){ pares_sol[cont++] = p; }

  //impressão da solução, de acordo com o enunciado do problema
  public String toString(){
    String resp = "{ ";
    for(int i = 0; i < cont; i++)
      if(pares_sol[i] != null) resp +=  pares_sol[i].toString();
    resp += "}, com custo igual a " + custo + "\n";
    return resp;
  }
}
```