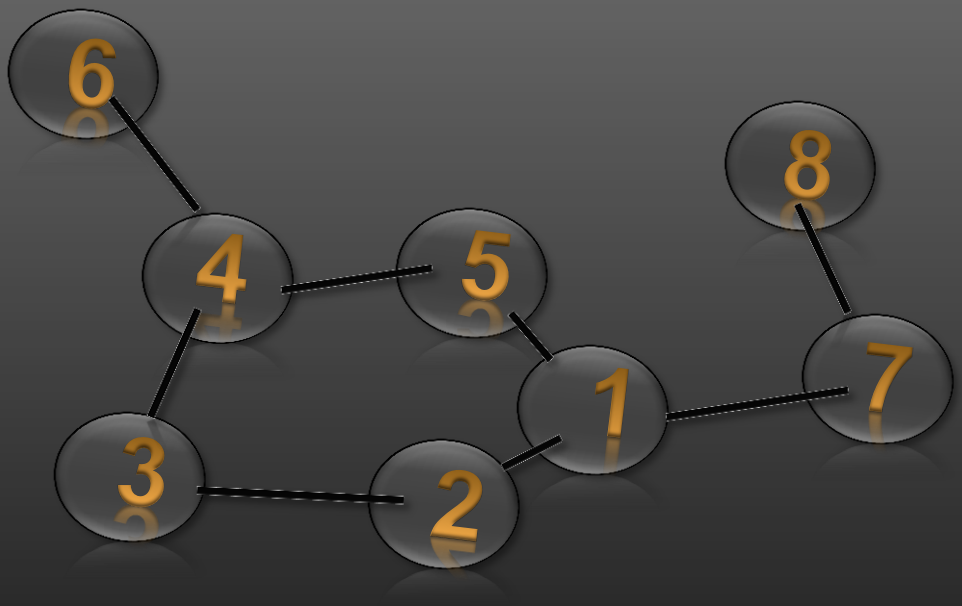


Atividade Final Laboratório de Conceitos de Linguagem de Programação

Projeto : Distância na Rede

Prof. Sidney

Nome: Lucas Biason

nº 5495-1 , Computação 2ºA – Matutino

Sumário

Problema:

Dado um conjunto de pares ordenados, representando arestas de um grafo, calcular a distância Euclidiana com relação à origem, e a distância Geodésica (a distância média de cada valor com todos os demais pontos. O ponto que tiver o menor valor de dispersão, ou seja, cuja média for a menor, será o ponto mais central dos dados).

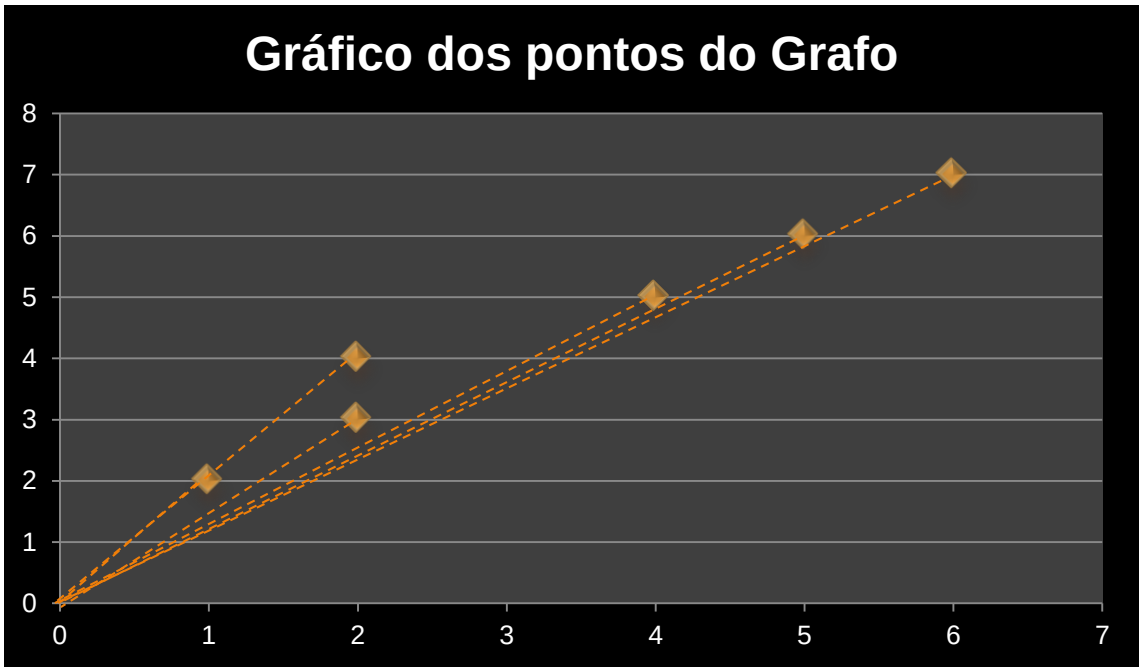
Teorias:

Distância Euclidiana da origem:

Seja A=(x_1, y_1), então a distancia a origem será igual aa raiz de x1 ao quadrado somado com y1 ao quadrado.

$$x_1{}^2 + y_1{}^2 = A^2$$
$$A = \sqrt[2]{x_1{}^2 + y_1{}^2}$$

Exemplo:



Valores X	Valores Y	Distância Euclidiana
1	2	2,23
2	3	3,60
2	4	4,45
4	5	6,40
5	6	7,81
6	7	9,21

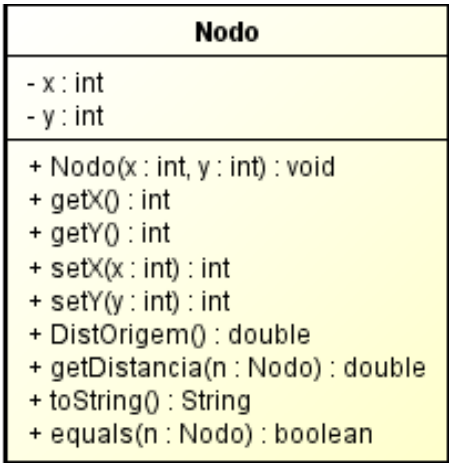
Solução:

Aplicativo utilizando a linguagem Java com orientação a objetos.

Lista de Objetos Usados:

Nodos: classe que define a estrutura das coordenadas. Onde cada uma apresenta uma coordenada x e uma coordenada y.

Define-se segundo a imagem a seguir:



Implementando:

```
public class Nodo {
    private int x;
    private int y;
    public Nodo(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public int getX() {return x;}
    public void setX(int x) {this.x=x;}
    public int getY() {return y;}
    public void setY(int y) {this.y=y;}
    public double DistOrigem() {return getDistancia(new Nodo(0, 0));}
    public double getDistancia(Nodo n) {
        return Math.sqrt(
                (Math.pow(x, 2) - Math.pow(n.getX(), 2)) +
                (Math.pow(y, 2) - Math.pow(n.getY(), 2)));
    }
```

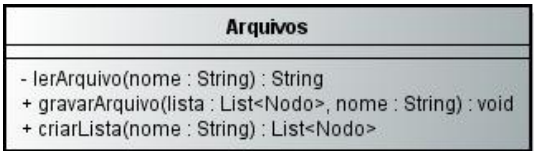
```
    @Override
    public String toString(){
        return "("+x+","+y+")";
    }
    public boolean equals(Nodo n){
        return (x == n.getX() && y == n.getY());
    }
}
```

Arquivos: classe que lê os dados de um arquivo texto, e gerar uma coleção de pares de nodos. O arquivo texto deve estar no formato nome.*txt*, tendo seu conteúdo obedecendo a seguinte forma:

$$C_n = x_1, y_1; x_2, y_2; x_3, y_3; \ldots; x_n, y_n;$$

Onde C é o conjunto das coordenadas que compõem o sistema do problema em questão (grafo).

Apresenta três métodos estáticos que definem a seguir nas imagens:



Implementação dos métodos:

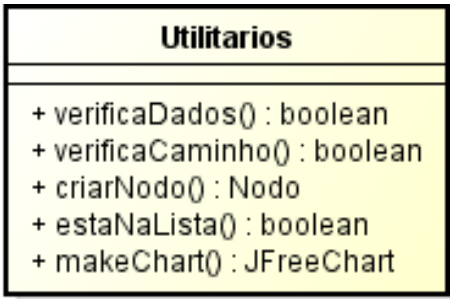
```
public class Arquivos {

    private static String lerArquivo(String nome) {
        StringBuilder sb = new StringBuilder();
        try {
            FileReader leitura = new FileReader(nome);
            int c;
            do {
                c = leitura.read();
                if (c != -1) sb.append((char) c);
            } while (c != -1);
            leitura.close();
            return sb.toString();
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null,
                    "O arquivo ainda não foi criado."
                    + "\nPara criar-lo basta inserir os manipular os dados "
                    + "normalmente, que ele será criado automaticamente.",
                    "Atenção", JOptionPane.INFORMATION_MESSAGE);
            return null;
        }
    }
```

```
public static void gravarArquivo(List<Nodo> lt,String nomeArquivo){
    String sb ="";
    try{
    FileWriter escreve = new FileWriter(nomeArquivo);
    for(Nodo aux : lt)
        sb+=aux.getX()+","+aux.getY()+";";
    escreve.write(sb);
    escreve.flush();
    escreve.close();
    JOptionPane.showMessageDialog(null,
            "Salvo com sucesso o arquivo : \""+nomeArquivo+"\"",
            "Atenção", JOptionPane.INFORMATION_MESSAGE);
    }catch(IOException e)
        {e.printStackTrace();}
}
```

```
 public static List<Nodo> criarLista(String arquivo) {
      try{
         int x, y;
         List<Nodo> lt = new LinkedList<Nodo>();
         StringTokenizer st = new StringTokenizer(lerArquivo(arquivo));
         while (st.hasMoreTokens()) {
             StringTokenizer subst = new StringTokenizer(st.nextToken(), ";");
             while (subst.hasMoreTokens()) {
                 StringTokenizer subsubst = new StringTokenizer(subst.nextToken(), ",");
                 while (subsubst.hasMoreTokens()) {
                     x = Integer.parseInt(subsubst.nextToken());
                     y = Integer.parseInt(subsubst.nextToken());
                     lt.add(new Nodo(x, y));
                 }
             }
         }
         return lt;
      }catch(NullPointerException e){return new LinkedList<Nodo>();}
   }
}
```

Utilitários: classe de métodos estáticos úteis a alguns processos do aplicativo. “verificaDados” e “verificaCaminho” validam os campos do formulario afim de evitar erros. “criarNodo” recebe o texto digitado pelo usuario e a apartir dele criar um ponto com as coordenadas x e y. “estaNaLista” verifica se o ponto já foi adicionado ao sistema, para q se evite repetições desnecessárias. “makeChart” fábrica que cria um gráfico a apartir dos pontos do sistema.



Implementando:

```
public class Utilitarios {
    public static boolean verificaDados(String texto) {
        if (texto.equals("")) {
            JOptionPane.showMessageDialog(null,
                    "Preencha corretamente o campo");
            return false;
        }
        if (!texto.contains(",")) {
            JOptionPane.showMessageDialog(null,
                    "Preencha corretamente usando a seguinte forma: x,y");
            return false;
        }
        return true;
    }
    public static boolean verificaCaminho(String texto) {
        if (texto.equals("")) {
            JOptionPane.showMessageDialog(null,
                    "Preencha corretamente o caminho");
            return false;
        }
        if (texto.contains("\\")) {
            JOptionPane.showMessageDialog(null,
                    "Favor trocar todas as barra \"\\\" por \"/\" " );
            return false;
        }
        return true;
    }
 public static Nodo criarNodo(String texto){
  try {
          String[] textot  = texto.split(",");
          int x = Integer.parseInt(textot[0]);
          int y = Integer.parseInt(textot[1]);
          return new Nodo(x,y);
      } catch (NumberFormatException e) {
          JOptionPane.showMessageDialog(null,
                  "Preencha corretamente usando a seguinte forma: x,y "
                  + "\nSendo x e y valores inteiros.");
          return null;
      }
 }
 public static boolean estaNaLista(List<Nodo> ln, Nodo n){
     for(Nodo aux : ln) if ( aux.equals(n) )return true;
     return false;
 }
```

```
public static JFreeChart makeChart(List<Nodo> ln) {

    XYSeriesCollection dataset = new XYSeriesCollection();
    XYSeries serie;
    for (Nodo aux : ln) {
        serie = new XYSeries(aux.toString());
        serie.add(aux.getX(), aux.getY());
        dataset.addSeries(serie);
    }
    JFreeChart chart = ChartFactory.createXYLineChart(
            "POints",
            "X","Y",
            dataset, // data
            PlotOrientation.VERTICAL,
            false, // include legend
            true, // tooltips
            false // urls
            );
    XYPlot plot = (XYPlot) chart.getPlot();
    plot.setAxisOffset(new RectangleInsets(5.0, 5.0, 5.0, 5.0));
    plot.setDomainGridlinePaint(Color.BLACK);
    plot.setRangeGridlinePaint(Color.BLACK);
    XYLineAndShapeRenderer renderer = (XYLineAndShapeRenderer) plot.getRenderer();
    renderer.setShapesVisible(true);
    renderer.setShapesFilled(true);
    NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();
    rangeAxis.setStandardTickUnits(NumberAxis.createIntegerTickUnits());
    return chart;
}
```

Calculos: responsável pelos cálculos das médias das distâncias euclidianas e geodésicas.

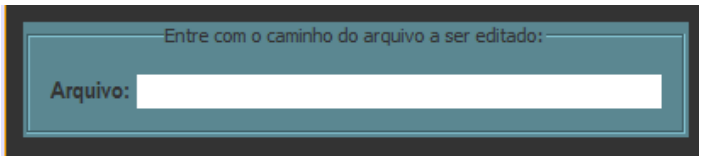
```
public class Calculos {
    public static double distancias(List<Nodo> li){
        if(li.isEmpty())return 0;
        double soma=0;
        for(Nodo aux : li)
            soma += aux.DistOrigem();
        return (soma/li.size());
    }
    public static double calculoMediaOrigem(List<Nodo> li){
        if(li.isEmpty())return 0;
        double soma = 0;
        for(Nodo aux : li)
            soma+=distGeral(aux,li);
        return soma/li.size();
    }
    private static double distGeral(Nodo base, List<Nodo> li){
        double soma = 0;
        for(Nodo aux : li)
            soma+= base.getDistancia(aux);
        return soma/(li.size() - 1);
    }
}
```

O primeiro método refere-se ao calculo da média das distâncias dos pontos a origem. Os outros dois calculam a média geodésica das médias das distancia de cada ponto perante os outros. Sendo que o segundo calcula a média e o terceiro, serve como auxiliar para calcular a média das distancias de um determinado ponto perante aos outros do sistema.

PlanoGUI:

Interface gráfica:

Utilização:



Recebe o caminho de um arquivo texto a ser editado com os pontos do sistema. Mesmo que esse arquivo não exista. O caminho e o nome são armazenados, e ao clicar em “salvar”, o arquivo texto é criado automaticamente.

Recebe um novo ponto a ser adicionado no sistema. Evita que o mesmo ponto seja adiciona duas vezes. A lista mostra os pontos atuais do sistema, e ao ser clicada, oferece a exclusão do ponto selecionado. Lembrando que as

alterações somente serão atualizadas no arquivo ao clicar no botão “salvar”.

Mostra as médias das distancias de cada ponto a origem, uma lista com as distancias a origem de cada ponto e também a distancia geodésica calculada.

Diagrama das Classes: