
 <p>GRUPO DE ESTUDOS DE POLÍTICAS MACROECONÔMICAS E CRESCIMENTO ECONÔMICO</p>	<p><b>DCOMP</b></p>	 <p>UFSJ Universidade Federal de São João del-Rei</p>
--	---------------------	--

**Grupo de Estudo de Políticas Macroeconômicas e Crescimento  
Econômico**

**Departamento de Ciência da Computação - DCOMP  
Universidade Federal de São João del-Rei – UFSJ**

# **11º Desafio da Ciência da Computação**

(Inspirado no filme Gênio Indomável do diretor Gus Van Sant - 1997)

**Data do Desafio: 15/10/2019**

**Data do Resultado: 10/11/2019**

## **FORMULÁRIO DE RESOLUÇÃO**

<b>Nome do aluno*: Bárbara Belize Moreira Boechat</b>			
<b>Período*:9º</b>	<b>Matrícula*:152050095</b>	<b>Turno(I/N)*:I</b>	<b>Campus: CTan</b>
<b>Nome do Pai**:Allen da Silva Boechat</b>			
<b>Nome da Mãe**:Sônia Moreira</b>			
<b>Time de futebol preferido**: Cruzeiro</b>			

\* Obrigatório \*\*Opcional

## **RESOLUÇÃO DO DISCENTE**

**O algoritmo proposto para realizar a simplificação de curvas poligonais possui os seguintes passos:**

### **1º Passo:**

É dada uma entrada com N pontos, para formar um vetor de pontos com que possuem coordenadas x,y. E também um valor

para tolerância, que deve ser o indicador de quais pontos serão eliminados nas demais iterações do algoritmo.

## **2º Passo:**

Após ter o vetor de pontos completo, deve-se calcular a equação da reta que passa entre o primeiro ponto do vetor, e o último ponto do vetor. E então deve-se determinar, para todos os pontos dentro do intervalo entre o primeiro e o último, sua distância até essa reta formada, assim, o ponto que possuir a maior distância é selecionado. Dessa forma, o algoritmo está pronto para de fato começar a eliminar pontos.

## **3º Passo:**

Ao encontrar o ponto de maior distância até essa reta, o algoritmo deve acontecer recursivamente nos seguintes intervalos de pontos, se a maior distância é maior que a tolerância:

1. Entre o 1º ponto do vetor, e o ponto mais distante.
2. Entre o Último ponto do vetor, e o ponto mais distante.

Dessa maneira, é gerado um vetor resultado formado pela concatenação dos resultados da recursão para **1** e **2**.

Caso a maior distância não seja maior que a tolerância, o vetor de pontos resultantes será a concatenação do 1º ponto do vetor, e do último ponto do vetor daquela chamada da função.

Segue o algoritmo (em Ruby) abaixo e em conjunto ao envio do PDF.

```
class Ponto
  attr_accessor :cord_x, :cord_y, :nome, :dist_reta
```

```

def initialize(nome, x, y)
    self.nome = nome
    self.cord_x = x
    self.cord_y = y
    self.dist_reta = 0
end
end

#distancia entre dois pontos
def dist_p2p(pA, pB)
    return Math.sqrt((pB.cord_x - pA.cord_x)**2 + (pB.cord_y -
pA.cord_y)**2 )
end

#equação da reta que passa por A e B
def eq_reta(pA, pB)
    a = pA.cord_y - pB.cord_y
    b = pA.cord_x - pB.cord_x
    c = pA.cord_x*pB.cord_y - pB.cord_x - pA.cord_y
    return a, b, c
end

def dist_ponto_reta(pA, a, b, c)
    return ((a*pA.cord_x + b*pA.cord_y + c).abs()) /
Math.sqrt(a**2 + b**2)
end

def simp_poligono(vet_pontos, tolerancia)
    maior_dist = 0
    maior_dist_i = 0
    fim = vet_pontos.length - 1

```

```

#Necessario escolher ponto inicial e final para calculo da
primeira distância
a,b,c = eq_reta(vet_pontos[0], vet_pontos[fim])
#Equação da reta entre ponto inicial e final
reta = "\nreta que passa entre #{vet_pontos[0].nome} e
#{vet_pontos[fim].nome}: #{a}x + (#{b}y) + (#{c}) = 0 \n"
#Calcular a distância entre todos os pontos a reta e escolher
escolher o ponto que está mais distante
for i in 2..fim-1 do
  vet_pontos[i].dist_reta = dist_ponto_reta(vet_pontos[i], a,
b, c)
  if(vet_pontos[i].dist_reta > maior_dist)
    maior_dist = vet_pontos[i].dist_reta
    maior_dist_i = i
  end
end
#Se a distancia é maior que a tolerancia deve-se simplificar
recursivamente
if maior_dist > tolerancia
  result1 = simp_poligono(vet_pontos[0..maior_dist_i],
tolerancia)
  result2 = simp_poligono(vet_pontos[maior_dist_i..fim],
tolerancia)

  pontos_resultantes = result1[0..fim-1]
  pontos_resultantes.concat(result2[1..fim])
else
  pontos_resultantes = vet_pontos[0], vet_pontos[fim]
end
return pontos_resultantes
end

```

```
vet_pontos = Array.new
resultado_final = Array.new
```

```
#CASO DE TESTE 1
```

```
#vet_pontos.push(Ponto.new("p1",0,1)) #0
#vet_pontos.push(Ponto.new("p2",1,2)) #1
#vet_pontos.push(Ponto.new("p3",2,2)) #2
#vet_pontos.push(Ponto.new("p4",4,4)) #3
#vet_pontos.push(Ponto.new("p5",5,3)) #4
#vet_pontos.push(Ponto.new("p6",6,3)) #5
#vet_pontos.push(Ponto.new("p7",7,1)) #6
```

```
#CASO DE TESTE 2
```

```
#vet_pontos.push(Ponto.new("p1",1,2)) #0
#vet_pontos.push(Ponto.new("p2",2,3)) #1
#vet_pontos.push(Ponto.new("p3",3,4)) #2
#vet_pontos.push(Ponto.new("p4",5,2)) #3
#vet_pontos.push(Ponto.new("p5",4,2)) #4
#vet_pontos.push(Ponto.new("p6",6,1)) #5
#vet_pontos.push(Ponto.new("p7",8,1)) #6
#vet_pontos.push(Ponto.new("p8",9,2)) #7
#vet_pontos.push(Ponto.new("p9",10,3)) #8
#tolerancia = 2
```

```
print "Quantos pontos deseja inserir? "
```

```
n_pontos = gets.chomp.to_i
```

```
for i in 0..n_pontos - 1 do
```

```
  nome = "p" + i.to_s
```

```
  print "Ponto #{i} - \n"
```

```
  print "\tCord_x: "
```

```

x = gets.chomp.to_i
print "\tCord_y: "
y = gets.chomp.to_i
vet_pontos.push(Ponto.new(nome, x, y))
nome = ""
end
print "\nQual tolerância deseja usar? "
tolerancia = gets.chomp.to_i

#Pontos iniciais
print "\n#### Pontos Iniciais ####\n"
vet_pontos.each {|x| print x.nome + "(" + (x.cord_x).to_s + "," +
(x.cord_y).to_s + ")  " }

resultado_final = simp_poligono(vet_pontos,tolerancia)

#Pontos no final:
print "\n\n#### Pontos ao Final da Operação ####\n"
resultado_final.each {|x| print x.nome + "(" + (x.cord_x).to_s +
"," + (x.cord_y).to_s + ")  " }

```



