|  |  |  |
| --- | --- | --- |
|  | **DCOMP** |  |

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

**Departamento de Ciência da Computação - DCOMP**

**Universidade Federal de São João del-Rei – UFSJ**

**11° 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**

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

**\* 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 Ultimo 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 + ") " }