TOLENTINO ROMERO GERARDO

Analisis y Diseño de Algoritmos

Andres Garcia Floriano

Practica 4

Para esta practica se busca determinar el par de puntos más cercanos de un conjunto de puntos.

En la primera parte implementé el siguiente algoritmo.

```
import random, math

puntos = []
puntosMasCercanos = [None] * 2;
distanciaMasCorta = 10000
for i in range (10):
    punto = [random.randint(-10,10),random.randint(-10,10)]
    puntos.append(punto)

for i in range(len(puntos)):
    for j in range(i+1,len(puntos)):
        x = math.sqrt((puntos[i][0] - puntos[j][0])**2 + (puntos[i][1] - puntos[j][1])**2 )
        if( x < distanciaMasCorta ):
            distanciaMasCorta = x
            puntosMasCercanos[0] = puntos[i]
            puntosMasCercanos[1] = puntos[j]

print(puntos)
print(puntosMasCercanos)
```

Con el algoritmo euclidiano se determina la distancia entre dos puntos y se compara con una cantidad inicial para la primera distancia encontrada, la cual se convierte en la distancia más corta hasta encontrar una distancia menor, a su vez cuando se encuentra una distancia más corta se guarda el par de puntos para saber cuales son los más cercanos.

Resultados para 10 puntos

```
Puntos generados: [[6, 7], [9, -1], [-7, 1], [-3, 0], [2, 9], [-4, -5], [-7, 6], [-7, -10], [6, -1], [-9, 9]]
Puntos más cercanos [[9, -1], [6, -1]]
El tiempo de ejecución para 10 puntos son 0.0 segundos
```

Resultados para 100 puntos

```
Puntos generados: [[-9, 3], [-10, -8], [6, 9], [-10, -2], [-2, -9], [2, 7], [-1, 6], [-5, 4], [0, 6], [1, 3], [-
8, 9], [-2, -5], [1, -2], [6, -9], [-3, -9], [4, -3], [-1, 6], [2, 8], [-5, -10], [-7, 1], [8, -8], [-1, -2], [1
0, 1], [4, 9], [5, 5], [3, 6], [-9, 6], [-4, 0], [-6, -9], [-8, 7], [8, 1], [-10, -6], [-3, -4], [9, 8], [-7, 0]
, [0, -10], [-1, -7], [-4, -4], [-2, 1], [2, -2], [0, -10], [5, 10], [-6, -7], [-6, 4], [5, -5], [-2, -5], [5, -
1], [-8, -8], [-9, -7], [-7, 2], [-2, 0], [2, -6], [-6, 1], [-4, 0], [-9, 5], [-3, -1], [3, -9], [2, -2], [1, 2]
, [4, 7], [-3, 0], [2, -9], [-2, 1], [-3, -1], [-5, -4], [-8, -9], [-9, 4], [-6, -3], [-6, 8], [4, 5], [5, 2], [
10, 5], [-8, -5], [8, 9], [-2, 5], [1, 10], [-6, 10], [-5, 6], [-10, -7], [8, -3], [3, -9], [4, 10], [9, -2], [-
7, 10], [1, 4], [-8, -5], [-5, -7], [-7, -9], [2, -10], [2, -5], [0, 1], [10, 1], [-4, -7], [-7, 6], [9, -10], [
-1, 10], [-9, 9], [-4, 9], [0, -5], [-10, 4]]
Puntos más cercanos [[-1, 6], [-1, 6]]
El tiempo de ejecución para 100 puntos son 0.003999948501586914 segundos
```

Resultados para 1000 puntos

```
Puntos más cercanos [[-6, 8], [-6, 8]]
El tiempo de ejecución para 1000 puntos son 0.3003525733947754 segundos
```

Resultados para 10000 puntos:

```
Puntos más cercanos [[8, -6], [8, -6]]
El tiempo de ejecución para 10000 puntos son 29.058475017547607 segundos
```

Para 100 000 puntos pasaron poco más de 5 minutos y no se obtuvo el resultado, por el bien de mi PC decidí parar el programa.

Con el algoritmo descrito en la practica, el cual implementa divide and conquer se obtiene el siguiente código.

```
import math

import time

import random


# Genera puntos aleatorios
```

```
def generate_random_points(num_points, coord_range=(-100, 100)):
    return [[random.randint(*coord_range), random.randint(*coord_range)] for _ in range(num_points)]

# Encuentra el par de puntos más cercanos y mide el tiempo de ejecución
def closest_pair_brute_force(points):
    min_distance = float('inf')
    closest_pair = (None, None)
    start_time = time.time()

    for i in range(len(points)):
        for j in range(i + 1, len(points)):
            distance = math.sqrt((points[i][0] - points[j][0])**2 + (points[i][1] - points[j][1])**2)
            if distance < min_distance:
                min_distance = distance
                closest_pair = (points[i], points[j])

    end_time = time.time()
    execution_time = end_time - start_time
    return closest_pair, execution_time

# Ejemplo de uso con puntos aleatorios
```

```
num_points = 10  # Número de puntos a generar
points = generate_random_points(num_points)
# Encuentra el par de puntos más cercanos y el tiempo de ejecución
closest_pair, execution_time = closest_pair_brute_force(points)

print("El par de puntos más cercano es:", closest_pair)
print(f"Tiempo de ejecución: para {num_points} puntos generados es {execution_time} segundos")
```

Tiempo para 10 puntos

```
El par de puntos más cercano es: ([7, 33], [10, 53])
Tiempo de ejecución: para 10 puntos generados es 0.0 segundos
```

Tiempo para 100 puntos

```
El par de puntos más cercano es: ([2, 44], [4, 43])
Tiempo de ejecución: para 100 puntos generados es 0.0 segundos
```

Tiempo para 1000 puntos

```
El par de puntos más cercano es: ([69, -50], [69, -50])
Tiempo de ejecución: para 1000 puntos generados es 0.2899332046508789 segundos
```

Tiempo para 10000 puntos

```
El par de puntos más cercano es: ([-12, -32], [-12, -32])
Tiempo de ejecución: para 10000 puntos generados es 31.14832615852356 segundos
```