Comenzado el	martes, 24 de octubre de 2023, 16:49
Estado	Finalizado
Finalizado en	martes, 24 de octubre de 2023, 17:50
Tiempo empleado	1 hora
Puntos	3/3
Calificación	**10** de 10 (**100**%)

Pregunta **1**

Correcta

Se puntúa 1 sobre 1

A lo largo de todo este desafío se trabajará con un conjunto de objetos que representan *puntos en un plano*. Se supone que la clase que permite representar a esos puntos es la misma clase *Point* que se presentó en la *Ficha 19 [2023] Registros y Objetos* (concretamente en el problema 46 de esa ficha), pero a los efectos de este Desafío los estudiantes pueden rediseñar la clase según sus propios criterios.

La idea es que para cada punto en el plano se usen dos atributos *x* e *y* para almacenar las coordenadas de ese punto, y una descripción de tipo cadena de caracteres (que en este desafío no será de importancia). Se provee a los alumnos un archivo *puntos.csv* (que puede descargar haciendo click *aquí* o en el nombre del archivo) que contiene los datos de ***n = 5000*** (cinco mil) puntos con valores de coordenadas generados en forma aleatoria.

El archivo provisto es un archivo de texto en formato *csv*. No tiene línea de timestamp ni tampoco tiene una línea con los nombres de las descripciones de los datos. Cada línea contiene simplemente dos valores separados por una coma: el primer valor de cada línea es la coordenada *x* del punto representado por esa línea, y el segundo es el valor de la coordenada *y* del mismo punto. Note que el archivo podría contener puntos repetidos (es decir, puntos con los mismos valores de coordenadas *(x, y)*).

Su primera tarea será crear un arreglo en memoria principal, que contenga a cada uno de los n = 5000 puntos que están representados en el archivo. Es decir, deberá leer el archivo puntos.csv línea por línea, obtener cada uno de los valores que cada línea trae, convertir esos datos a un valor int (recuerde que si están en un archivo de texto, esos valores están en forma de cadena...), armar objetos de tipo Point con esos valores convertidos, y agregarlos en un arreglo de objetos.

Una vez creado ese arreglo, deberá plantear un proceso que encuentre los ***dos puntos del arreglo que se encuentren a la menor distancia***, y ***los dos puntos que se encuentren a la mayor distancia***. La clase Point de la Ficha 19 (que hemos sugerido al inicio de esta consigna) ya provee un método para hacer el cálculo de esa distancia aplicando el Teorema de Pitágoras.

Existen algoritmos rápidos para resolver el problema del encontrar los dos puntos con mínima distancia en un conjunto de *n* puntos, pero esos algoritmos están fuera del alcance de este curso. Y por otro lado, no hay algoritmos tan rápidos ni tan eficientes para la determinación de los dos puntos a *distancia máxima* en el mismo conjunto de n puntos.

En este desafío, simplemente se espera que los estudiantes sean capaces de implementar un algoritmo tipo ***fuerza bruta***, tanto para encontrar la menor distancia como la mayor distancia. Un algoritmo de *fuerza bruta* es aquel que hace todo el trabajo posible, explorando y ejecutando todas las combinaciones posibles, sin intentar ahorrar trabajo ni tiempo. Obviamente, un algoritmo de este tipo eventualmente llegará al resultado correcto, pero insumiendo normalmente ***mucho*** más tiempo que el que llevaría un algoritmo optimizado si el tamaño del conjunto de datos es muy grande.

Para el cálculo de los puntos a distancia mínima y máxima en un conjunto de *n* puntos, un algoritmo de *fuerza bruta* podría ser el siguiente (expresado como pseudocódigo):

```
0.) Cree un arreglo puntos con todos los datos del archivo puntos.csv.
1.) Sea dmax = 0  # distancia máxima inicial
2.) Sea dmin = distancia entre puntos[0] y puntos[1] # distancia mínima inicial

3.) Para i en el rango(0, n-1):
        Para j en el rango(i+1, n):
             1.1) Sea d = distancia entre puntos[i] y puntos[j]
             1.2) Si d < dmin:
                  1.2.1) dmin = d
             1.3) Si d > dmax:
                  1.3.1) dmax = d

4.) Mostrar distancia mínima dmin.
5.) Mostrar distancia máxima dmax.
```

Por lo tanto, su tarea será implementar este algoritmo, calcular las dos distancias pedidas (note que no se le pide indicar cuáles son los puntos, sino solo las distancias), y registrar luego los resultados que haya obtenido. Para simplificar, cuando calcule las distancias redondee los resultados al entero más próximo, y trabaje con esos valores enteros. Obviamente, para hacer el redondeo use la función round() que Python ya provee...

Registre ahora (en el casillero que sigue) el valor redondeado (como número entero) de la *distancia mínima* que haya encontrado en el conjunto:

Respuesta: 1 ✔

Pregunta **2**

Correcta

Se puntúa 1 sobre 1

Registre ahora (en el casillero que sigue) el valor redondeado (como número entero) de la *distancia máxima* que haya encontrado en el conjunto:

Respuesta: 27913 ✔

Pregunta **3**

Correcta

Se puntúa 1 sobre 1

Ya se indicó en las consignas que en general un algoritmo de *fuerza bruta* llegará eventualmente al resultado pedido, pero posiblemente demorando mucho tiempo si n (la cantidad de datos a procesar) es grande. ¿Cuál de las siguientes expresiones de orden es la que mejor describe el tiempo de ejecución en el peor caso del algoritmo de *fuerza bruta* que hemos indicado en este desafío para el cálculo de las distancias *mínima* y *máxima* en el conjunto de *n* puntos? (**Observación:** para responder a esta pregunta son necesarios los temas y contenidos de la Ficha 27 de la asignatura. Si esa ficha aún no ha sido desarrolladas en clases, sugerimos esperar antes de responderla).

Seleccione una:

- a. O(n)
- b. $O(n^2)$ ✔
- c. $O(n^3)$
- d. O(n * log(n))

◄ Cuestionario 26 [Temas: hasta Ficha 26] [OPCIONAL - LA CALIFICACIÓN NO SE REGISTRA]

Ir a...

Guía de Ejercicios Prácticos 26 ►