Comenzado el	martes, 6 de octubre de 2020, 00:14
Estado	Finalizado
Finalizado en	martes, 6 de octubre de 2020, 00:15
Tiempo empleado	46 segundos
Puntos	3/3
Calificación	**10** de 10 (**100**%)

Pregunta **1**

Correcta

Puntúa 1 sobre 1

A lo largo de todo este desafío se trabajará con un conjunto de registros que representan *puntos en un plano*. Se supone que el tipo registro que permite representar a esos puntos es el siguiente (incluyendo la función constructora *__init__()* y la función *to_string()* para favorecer la explicación):

```
class Point:
    def __init__(self, cx, cy, desc='p'):
        self.x = cx
        self.y = cy
        self.descripcion = desc


def to_string(point):
    r = str(point.descripcion) + '(' + str(point.x) + ', ' + str(point.y) + ')'
    return r
```

Como se ve, para cada punto en el plano se suponen dos campos *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.df4* (que puede descargar haciendo click *aquí* o en el nombre del archivo) que contiene ***n = 5000*** (cinco mil) registros de tipo *Point*, con valores de coordenadas generados en forma aleatoria. El archivo puede contener puntos repetidos (es decir, puntos con los mismos valores de coordenadas *(x, y)*). El archivo además, fue generado por serialización, grabando uno a uno los 5000 puntos con la función *pickle.dump()*.

Su primera tarea será crear un arreglo en memoria principal, que contenga a cada uno de los n = 5000 puntos que están grabados en el archivo (es decir, des-serializar los 5000 puntos y almacenarlos en un arreglo.

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***. Si *p1* y *p2* son dos registros de tipo *Point*, entonces la siguiente función permite calcular la distancia entre ellos, aplicando el *Teorema de Pitágoras*:

```
def distance_between(p1, p2):
    # calcular "delta y" y "delta x"
    dy = p2.y - p1.y
    dx = p2.x - p1.x

    # Distancia entre ellos... Pitágoras...
    return math.sqrt(pow(dx, 2) + pow(dy, 2))
```

Existen algoritmos rápidos para resolver el problema del cálculo de 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 registros del archivo puntos.fd4.
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 sólo 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...

Detalle MUY importante: para el desarrollo que se le pide aquí, deberá incluir en el código fuente de su programa la definición de la clase Point tal y como está presentada en este enunciado, en el mismo archivo que contiene al código fuente de su programa y con el mismo nombre, definición de campos y función constructora. No cambie el nombre, ni incluya la clase en un módulo separado, pues en ese caso la serialización fallará al intentar leer los registros.

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

Respuesta: 2 ✔

Pregunta **2**

Correcta

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: 20836 ✔

Pregunta **3**

Correcta

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?

Seleccione una:

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

◄ Cuestionario 25 [Temas: hasta Ficha 25]

Ir a...

Guía de Ejercicios Prácticos 22 ►