Examen 2020-01-25

Examen teórico

1 Ejercicio 3

Se quiere implementar un programa que calcule la envolvente convexa de un conjunto de puntos en el plano. La envolvente convexa es el polígono convexo más pequeño que contiene todos los puntos del conjunto.

Para ello, se dispone de un fichero de texto que contiene las coordenadas de los puntos. Cada línea del fichero contiene dos números reales separados por espacios en blanco, representando las coordenadas x e y de un punto.

Se pide implementar en Python un programa que: 1. Lea los puntos del fichero y los almacene en una lista. 2. Calcule la envolvente convexa utilizando el algoritmo de Graham. 3. Muestre por pantalla los puntos que forman la envolvente convexa en orden.

Solución

```python
from dataclasses import dataclass
from typing import List, Tuple

@dataclass
class Punto:
    x: float
    y: float

def orientacion(p: Punto, q: Punto, r: Punto) -> int:
    val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y)
    if val == 0:
        return 0
    elif val > 0:
        return 1
    else:
        return 2

def graham(puntos: List[Punto]) -> List[Punto]:
    n = len(puntos)
    if n < 3:
        return puntos

    # Encontrar el punto más bajo
    punto_mas_bajo = min(puntos, key=lambda p: (p.y, p.x))
    puntos.remove(punto_mas_bajo)

    # Ordenar puntos por ángulo polar
    puntos.sort(key=lambda p: (
          (p.x - punto_mas_bajo.x) / ((p.x - punto_mas_bajo.x)**2 + (p.y -
```

```
punto_mas_bajo.y)**2)**0.5,
          (p.y - punto_mas_bajo.y) / ((p.x - punto_mas_bajo.x)**2 + (p.y -
punto_mas_bajo.y)**2)**0.5
    ))

    # Construir la envolvente convexa
    ch = [punto_mas_bajo]
    for p in puntos:
        while len(ch) >= 2 and orientacion(ch[-2], ch[-1], p) != 2:
            ch.pop()
        ch.append(p)

    return ch

def main():
    # Leer puntos del fichero
    puntos = []
    with open('puntos.txt', 'r') as f:
        for linea in f:
            x, y = map(float, linea.strip().split())
            puntos.append(Punto(x, y))

    # Calcular envolvente convexa
    ch = graham(puntos)

    # Mostrar resultado
    print("Puntos de la envolvente convexa:")
    for p in ch:
        print(f"({p.x:.2f}, {p.y:.2f})")

if __name__ == "__main__":
    main()
```

2 Ejercicio 3.5

Se quiere implementar un sistema de mensajería entre procesos. El sistema debe permitir que los procesos envíen y reciban mensajes entre sí. Cada mensaje tiene un identificador único y un contenido.

Se pide implementar en Python un programa que: 1. Defina las estructuras de datos necesarias para representar mensajes y procesos. 2. Implemente las operaciones de envío y recepción de mensajes. 3. Muestre por pantalla el estado del sistema después de cada operación.

Solución

```
from dataclasses import dataclass
from typing import List, Optional

@dataclass
class Mensaje:
    id: int
    contenido: str
```

```
    emisor: int
    receptor: int

class Proceso:
    def __init__(self, id: int):
        self.id = id
        self.mensajes: List[Mensaje] = []

    def recibir_mensaje(self, mensaje: Mensaje):
        self.mensajes.append(mensaje)

    def procesar_mensaje(self) -> Optional[Mensaje]:
        if self.mensajes:
            return self.mensajes.pop()
        return None

class Sistema:
    def __init__(self, num_procesos: int):
        self.procesos = [Proceso(i) for i in range(num_procesos)]

    def enviar_mensaje(self, emisor: int, receptor: int, contenido: str):
        mensaje = Mensaje(
            id=len(self.procesos[receptor].mensajes) + 1,
            contenido=contenido,
            emisor=emisor,
            receptor=receptor
        )
        self.procesos[receptor].recibir_mensaje(mensaje)
        print(f"Mensaje enviado de proceso {emisor} a proceso {receptor}")

    def recibir_mensaje(self, proceso: int):
        mensaje = self.procesos[proceso].procesar_mensaje()
        if mensaje:
            print(f"Proceso {proceso} recibe mensaje:")
            print(f"ID: {mensaje.id}")
            print(f"Contenido: {mensaje.contenido}")
            print(f"Emisor: {mensaje.emisor}")
        else:
            print(f"Proceso {proceso} no tiene mensajes pendientes")

def main():
    # Inicializar sistema
    sistema = Sistema(3)

    # Ejemplo de uso
    sistema.enviar_mensaje(0, 1, "Hola proceso 1")
    sistema.enviar_mensaje(1, 2, "Hola proceso 2")
    sistema.recibir_mensaje(1)
    sistema.recibir_mensaje(2)

if __name__ == "__main__":
    main()
```

3 Ejercicio 3.5

Se quiere implementar un juego de bingo. El juego debe permitir a los jugadores marcar números en sus cartones y verificar si han ganado.

Se pide implementar en Python un programa que: 1. Genere cartones de bingo aleatorios para cada jugador. 2. Permita a los jugadores marcar números en sus cartones. 3. Verifique si un jugador ha ganado (línea, bingo, etc.).

Solución

```python
from dataclasses import dataclass
from typing import List, Set
import random

@dataclass
class Carton:
    numeros: List[List[int]]
    marcado: List[List[bool]]

class Jugador:
    def __init__(self, id: int):
        self.id = id
        self.carton = self._generar_carton()

    def _generar_carton(self) -> Carton:
        numeros = []
        usado = set()

        for _ in range(5):
            fila = []
            for _ in range(5):
                while True:
                    num = random.randint(1, 90)
                    if num not in usado:
                        usado.add(num)
                        fila.append(num)
                        break
            numeros.append(fila)

      return Carton(numeros=numeros, marcado=[[False] * 5 for _ in range(5)])

    def marcar_numero(self, numero: int):
        for i in range(5):
            for j in range(5):
                if self.carton.numeros[i][j] == numero:
                    self.carton.marcado[i][j] = True

    def verificar_linea(self) -> bool:
        # Verificar líneas horizontales
        for i in range(5):
            if all(self.carton.marcado[i]):
                return True
```

```
        # Verificar líneas verticales
        for j in range(5):
            if all(self.carton.marcado[i][j] for i in range(5)):
                return True

        return False

    def verificar_bingo(self) -> bool:
        return all(all(fila) for fila in self.carton.marcado)

class Juego:
    def __init__(self, num_jugadores: int):
        self.jugadores = [Jugador(i) for i in range(num_jugadores)]
        self.numeros_sacados: Set[int] = set()

    def sacar_numero(self) -> int:
        while True:
            num = random.randint(1, 90)
            if num not in self.numeros_sacados:
                self.numeros_sacados.add(num)
                return num

    def jugar(self):
        while True:
            # Sacar número
            num = self.sacar_numero()
            print(f"Número sacado: {num}")

            # Marcar números en cartones
            for jugador in self.jugadores:
                jugador.marcar_numero(num)

            # Verificar ganadores
            for jugador in self.jugadores:
                if jugador.verificar_linea():
                    print(f"¡Jugador {jugador.id} ha cantado línea!")
                if jugador.verificar_bingo():
                    print(f"¡Jugador {jugador.id} ha cantado bingo!")
                    return

def main():
    random.seed()
    juego = Juego(3)
    juego.jugar()

if __name__ == "__main__":
    main()
```