1. Introducción

Este documento describe la arquitectura, el diseño y la implementación del sistema de riego y fertilización de invernaderos utilizando Python, programación orientada a objetos y TDAs propios.

2. Objetivo General

Modelar, documentar e implementar un sistema que optimice el riego y fertilización de un invernadero usando Python, POO y TDAs propios.

3. Arquitectura del Sistema

El sistema está organizado en módulos:

- app.py: Aplicación Flask que sirve como interfaz web.
- models/: Clases principales (Drone, Planta, Invernadero) y TDAs propios.
- services/: Servicios para parsear XML, simular el plan y generar gráficos.
- templates/: Plantillas HTML para la interfaz.
- static/: Archivos estáticos.

4. Diseño de Clases

Se han diseñado las siguientes clases principales:

• Planta: Almacena información de cada planta (hilera, posición, litros, gramos).
• Drone: Almacena datos de cada dron asignado a una hilera.
• Invernadero: Contiene hileras, plantas y drones.
• Simulador: Ejecuta la simulación de riego siguiendo el plan.
• TDAs: ListaSimple, Cola, Pila implementadas sin usar estructuras nativas.

5. Algoritmo de Simulación

El algoritmo recorre el plan de riego (cola), mueve el dron correspondiente hasta la planta indicada, aplica agua y fertilizante, y registra las instrucciones y tiempos. Solo un dron riega a la vez. Se calcula el tiempo óptimo y el uso total de recursos por dron y global.

6. Interfaz Web (Flask)

La interfaz permite cargar un archivo XML, seleccionar invernadero y plan, ejecutar la simulación y mostrar los resultados. También permite generar reportes HTML y graficar el estado de los TDAs en un tiempo t usando Graphviz.

7. Reportes HTML y Graphviz

Por cada invernadero se genera un reporte HTML con:
- Tabla de drones por hilera.
- Tabla de instrucciones por tiempo.
- Totales de agua y fertilizante.
El grafo de Graphviz representa el estado de los TDAs en tiempo t.

8. Control de Versiones

Se utiliza GitHub para el control de versiones. El repositorio se llama IPC2_Proyecto2_<Carnet>. Se deben realizar cuatro releases mínimos, siendo el cuarto el final.

9. Conclusión

El sistema propuesto cumple con los requerimientos del enunciado. Se basa en POO, TDAs propios y Flask, asegurando modularidad, mantenibilidad y capacidad de extender la funcionalidad.

10. Diagramas

A continuación se presentan los diagramas de clases y de actividad del sistema:

Diagrama de Clases

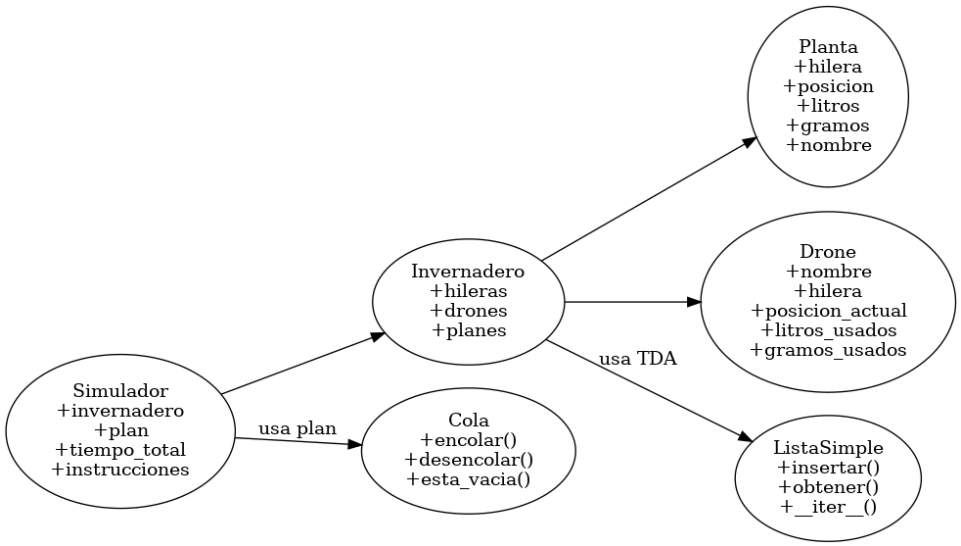

Diagrama de Actividad