Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 2do. Semestre 2025

Sistema de Simulación y Control de Riego con Drones en Invernaderos

202300502 – Pablo Javier Alvarez Marroquin

Resumen

El proyecto implementa un sistema de optimización de riego y fertilización para invernaderos utilizando Python y programación orientada a objetos con TDAs propios (listas enlazadas, pila y cola), cumpliendo la restricción de no usar estructuras nativas. A partir de archivos XML de entrada, se modelan drones, hileras y plantas, y se simulan planes de riego en tiempo discreto respetando reglas: un segundo por metro de movimiento, un segundo por riego, y un solo dron regando por instante en el orden especificado. La solución genera salida XML con tiempo óptimo, consumo total y por dron, e instrucciones por segundo; además produce reportes HTML por invernadero/plan y grafos de estado de TDAs con Graphviz en un tiempo t configurable desde una interfaz web desarrollada en Flask. Los resultados muestran la trazabilidad completa de la ejecución, métricas de eficiencia hídrica y de fertilizante, y visualizaciones que apoyan la evaluación del algoritmo y su desempeño operativo. El proyecto está versionado en GitHub con entregables y documentación en formato de ensayo.

Palabras clave
Optimizació, TDAs propios, Simulación, XM y Flask.

Abstract

The project implements an irrigation and fertilization optimization system for greenhouses using Python and object-oriented programming with custom data structures (linked lists, stack, and queue), complying with the restriction of not using native lists or dictionaries. Input XML files are parsed to model drones, rows, and plants, and irrigation plans are simulated in discrete time, following strict rules: one second per meter of movement, one second per watering, and only one drone watering at a time in the specified order. The solution generates an output XML with optimal time, total and per-drone consumption, and second-by-second instructions; it also produces HTML reports per greenhouse/plan and Graphviz graphs of TDA states at a configurable time t, accessible through a Flask-based web interface. Results demonstrate complete traceability of the execution, efficiency metrics for water and fertilizer, and visualizations that support the evaluation of the algorithm and its operational performance. The project is versioned in GitHub with deliverables and documentation in essay format.

Keywords
Optimization, Custom TDAs, Simulation, XML y Flask.

Introducción

La agricultura de precisión se ha convertido en una de las áreas más relevantes para enfrentar los desafíos de la seguridad alimentaria y el uso eficiente de los recursos naturales. En particular, el riego y la fertilización representan procesos críticos que, si no se gestionan adecuadamente, pueden generar desperdicio de agua, sobreuso de fertilizantes y disminución en la productividad de los cultivos.

En este contexto, el presente proyecto propone el desarrollo de un sistema de simulación y optimización de riego en invernaderos, implementado en Python bajo un enfoque de programación orientada a objetos y utilizando estructuras de datos abstractas (TDAs) diseñadas desde cero. La restricción académica de no emplear listas, diccionarios ni otras estructuras nativas de Python permitió reforzar el entendimiento de los fundamentos de las estructuras dinámicas, tales como listas enlazadas, pilas y colas, aplicadas a un problema real.

El sistema recibe como entrada archivos XML que describen drones, hileras, plantas y planes de riego. A partir de esta información, se simula el proceso en tiempo discreto, respetando reglas estrictas de operación: un segundo por metro recorrido, un segundo por riego y la restricción de que solo un dron puede regar en cada instante. Los resultados se presentan en archivos XML de salida, reportes HTML y visualizaciones gráficas de los TDAs mediante Graphviz, todo ello accesible desde una interfaz web desarrollada con Flask.

De esta manera, el proyecto no solo cumple con los objetivos académicos del curso, sino que también aporta una aproximación práctica a la gestión eficiente de recursos agrícolas, integrando conceptos de algoritmia, modelado de datos y desarrollo de aplicaciones web.

Desarrollo del tema

El desarrollo del proyecto se estructuró en varias fases, cada una orientada a cumplir con los requerimientos académicos y funcionales establecidos en el enunciado:

1. Diseño de estructuras de datos (TDAs propios)
    - Se implementaron listas enlazadas, colas y pilas desde cero, evitando el uso de estructuras nativas de Python como list, dict o set.
    - Estas estructuras se utilizaron para almacenar y manipular los elementos principales del sistema: drones, plantas, planes de riego y asignaciones.
    - El diseño modular permitió reutilizar los TDAs en diferentes partes del proyecto, garantizando consistencia y cumplimiento de las restricciones.

2. Modelado de entidades del dominio
    - Se definieron clases para representar los objetos centrales:
    - Drone: con atributos de posición, hilera asignada y consumo de recursos.
    - Plant: con requerimientos de agua y fertilizante.

- Plan y PlanEntry: que almacenan la secuencia de riego a ejecutar.

- Greenhouse: que integra plantas, drones y planes en un mismo invernadero.

Este modelado facilitó la simulación y la trazabilidad de cada acción.

3. Procesamiento de archivos XML
- Se desarrolló un parser que interpreta el archivo de entrada (entrada.xml), extrayendo la configuración de invernaderos, drones, plantas y planes de riego.

- El sistema genera un archivo de salida (salida.xml) con los resultados de la simulación: tiempo óptimo, consumo total y por dron, así como las instrucciones por segundo.

4. Motor de simulación
- Se implementó un simulador en tiempo discreto, que avanza segundo a segundo: 1 segundo por metro recorrido y 1 segundo por riego.

- Solo un dron puede regar en cada instante, siguiendo el orden del plan.

- El simulador produce un timeline con las acciones de cada dron en cada segundo, lo que permite reconstruir el proceso completo.

5. Generación de reportes
❖ Se desarrollaron dos tipos de reportes:

❖ HTML: con estadísticas de eficiencia, consumo de recursos y detalle de instrucciones.

❖ Graphviz (.dot): que representa gráficamente el estado de los TDAs en un tiempo t configurable.

❖ Estos reportes permiten evaluar tanto el desempeño del algoritmo como la correcta implementación de las estructuras de datos.

6. Interfaz web con Flask
❖ Se construyó una aplicación web que permite:

❖ Subir el archivo de entrada.

❖ Seleccionar invernadero y plan de riego.

❖ Ejecutar la simulación y visualizar resultados.

❖ Descargar reportes en HTML, XML y gráficos de TDAs.

La interfaz facilita la interacción con el sistema y la validación de los resultados.

Conclusiones

Esta sección debe orientarse a evidenciar claramente las principales ideas generadas, propuestas que deriven del análisis realizado y si existen, expresar las conclusiones o aportes que autor quiera destacar.

Enfatizando, lo importante es destacar las principales posturas fundamentadas del autor, que desea transmitir a los lectores.

Adicionalmente, pueden incluirse preguntas abiertas a la reflexión y debate, temas concatenados con el tema expuesto o recomendaciones para profundizar en la temática expuesta.

Referencias bibliográficas

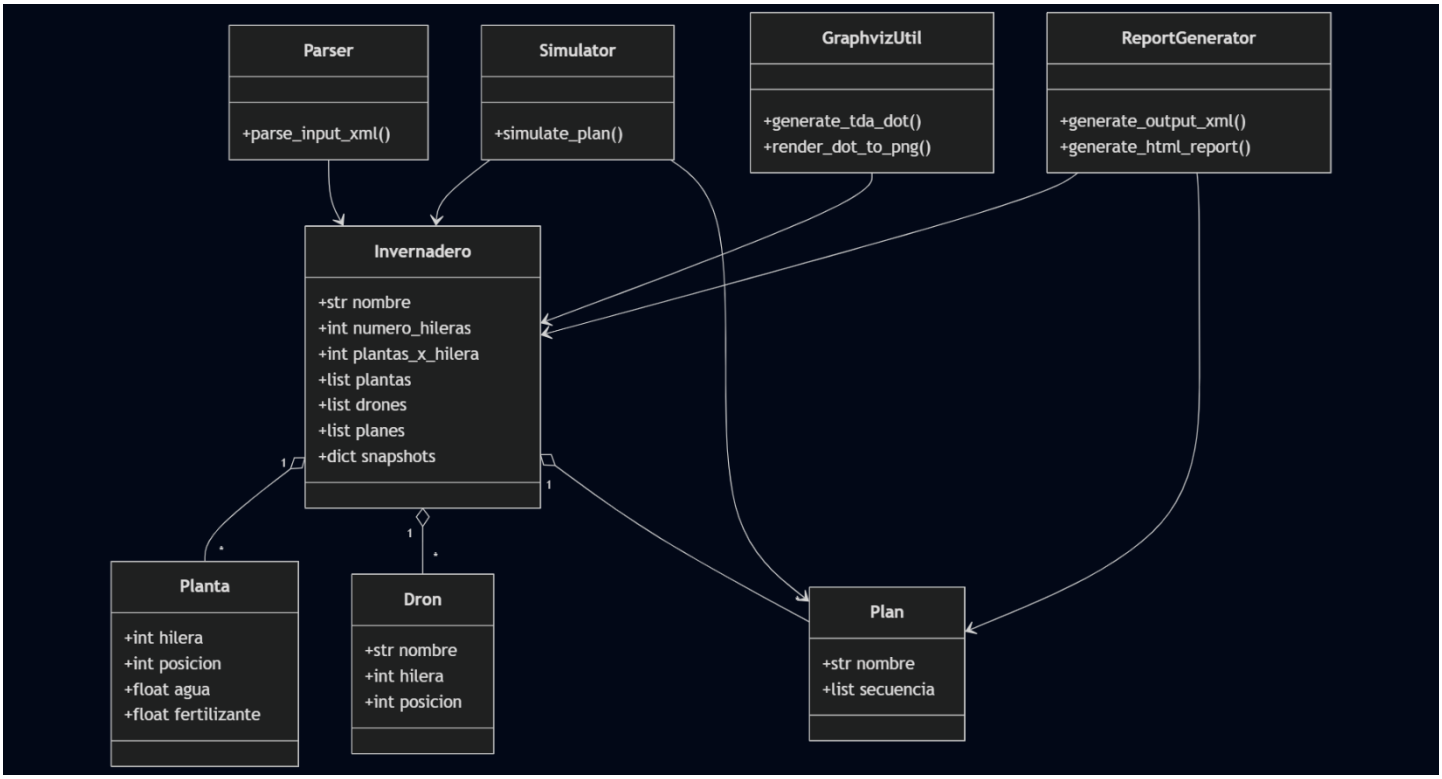
1. Documentación oficial de Flask. 2. Documentación de xml.etree.ElementTree (Python).
3. Documentación de graphviz y python-graphviz.

Extensión: de cuatro a siete páginas como máximo

app.py

```python
from flask import Flask, request, render_template, redirect, url_for, send_from_directory
from parser import parse_input_xml
from simulator import simulate_plan
from report_generator import generate_output_xml, generate_html_report
from graphviz_util import generate_tda_dot, render_dot_to_png
import os

app = Flask(__name__)
UPLOAD_FOLDER = 'uploads'
REPORTS_FOLDER = 'reports'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
os.makedirs(REPORTS_FOLDER, exist_ok=True)

_loaded_greenhouses = None

@app.route('/', methods=['GET'])
def index():
    return render_template('index.html')

@app.route('/upload', methods=['POST'])
def upload():
    global _loaded_greenhouses
    f = request.files.get('file')
    if not f:
        return "No file provided", 400
```

```python
@app.route('/upload', methods=['POST'])
def upload():
    global _loaded_greenhouses
    f = request.files.get('file')
    if not f:
        return "No file provided", 400
    path = os.path.join(UPLOAD_FOLDER, 'entrada.xml')
    f.save(path)
    _loaded_greenhouses = parse_input_xml(path)
    return redirect(url_for('select_plan'))

@app.route('/select', methods=['GET'])
def select_plan():
    global _loaded_greenhouses
    if not _loaded_greenhouses:
        return redirect(url_for('index'))
    return render_template('select_plan.html', greenhouses=_loaded_greenhouses)

@app.route('/simulate', methods=['POST'])
def simulate():
    global _loaded_greenhouses
    inv_name = request.form.get('invernadero')
    plan_name = request.form.get('plan')
    gh = None
```


Parser
+parse_input_xml()
Simulator
+simulate_plan()
GraphvizUtil
+generate_tda_dot()
+render_dot_to_png()
ReportGenerator
+generate_output_xml()
+generate_html_report()
Invernadero
+str nombre
+int numero_hileras
+int plantas_x_hilera
+list plantas
+list drones
+list planes
+dict snapshots
Planta
+int hilera
+int posicion
+float agua
+float fertilizante
Dron
+str nombre
+int hilera
+int posicion
Plan
+str nombre
+list secuencia
1
*
1
*
1
1