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 AGRICULTURA DE PRECISIÓN CON OPTIMIZACIÓN DE ESTACIONES BASE MEDIANTE AGRUPAMIENTO DE PATRONES

**202400023 – Alison Melysa Pérez Blanco**

## Resumen

Este proyecto desarrolla un sistema de agricultura de precisión basado en Programación Orientada a Objetos (POO) y estructuras de datos personalizadas, con el fin de optimizar el uso de estaciones base mediante agrupamiento por patrones de frecuencia. Se implementan listas enlazadas simples anidadas para representar matrices de frecuencia, patrones y reducidas, sin usar estructuras nativas de Python. El sistema procesa archivos XML que contienen información de campos agrícolas, sensores y frecuencias, y genera un archivo de salida con estaciones base reducidas. El algoritmo identifica estaciones con patrones idénticos en sus matrices de frecuencia (suelo y cultivo), las agrupa y suma sus datos, reduciendo la infraestructura sin pérdida de información. Se utiliza Graphviz para visualizar gráficamente las matrices, facilitando el análisis del sistema. La solución cumple con los principios de TDA, modularidad y reutilización de código. Los resultados demuestran una reducción efectiva del número de estaciones, mejorando la eficiencia del sistema de monitoreo agrícola.

## Palabras clave

Agricultura de precisión, POO, listas enlazadas, Graphviz, optimización.

## *Abstract*

*This project develops a precision agriculture system based on Object-Oriented Programming (OOP) and custom data structures to optimize the use of base stations through frequency pattern clustering. Nested singly linked lists are implemented to represent frequency, pattern, and reduced matrices, without using native Python structures. The system processes XML files containing information about agricultural fields, sensors, and transmission frequencies, and generates an output file with reduced base stations. The algorithm identifies base stations with identical patterns in their frequency matrices (soil and crop), groups them, and sums their data, reducing infrastructure without loss of information. Graphviz is used to graphically visualize the matrices, facilitating system analysis. The solution adheres to the principles of Abstract Data Types (ADT), modularity, and code reusability. Results show an effective reduction in the number of base stations, improving the efficiency of the agricultural monitoring system.*

## *Keywords*

*Precision agriculture, OOP, linked lists, Graphviz, optimization.*

## Introducción

Brindar un panorama general del tema desarrollado, su importancia y trascendencia, incluir aspectos relevantes que permitan comprender el contexto en el cual se plantea, bases teóricas o perspectivas adoptadas y otros referentes que sustenten la argumentación.

Comunicar claramente los propósitos del ensayo, evidenciando los aportes que se brindan al lector.

Pueden plantearse interrogantes cuya respuesta se construirá mediante el desarrollo del tema.

La función principal de esta sección es familiarizar rápidamente al lector con el contenido del tema a tratar.

Debe contener un máximo de 150 palabras.

## Desarrollo del tema

a. Diseño del sistema con Programación Orientada a Objetos (POO)

El sistema fue diseñado bajo el paradigma de Programación Orientada a Objetos (POO), lo cual permitió modelar de forma natural los componentes del sistema de agricultura de precisión como entidades con atributos y comportamientos bien definidos. Cada clase representa un elemento real del sistema: campos agrícolas, estaciones base, sensores y registros de frecuencia. Esta abstracción facilita la comprensión del sistema, mejora la modularidad y permite una fácil extensión futura, como la incorporación de nuevos tipos de sensores o algoritmos de optimización.

Las principales clases implementadas son:

- CampoAgricola: Representa un campo agrícola con sus estaciones, sensores y matrices de frecuencia. Es la unidad principal de procesamiento.
- EstacionMonitoreo: Modela una estación base con identificador y nombre. Cada estación puede recibir datos de múltiples sensores.
- SensorMedicion: Representa un sensor de suelo (sensorS) o de cultivo (sensorT), junto con sus registros de frecuencia de transmisión.
- RegistroFrecuencia: Almacena el valor de frecuencia y la estación que lo generó, permitiendo vincular datos con su origen.
- RepresentacionMatriz: Implementa una matriz mediante listas enlazadas anidadas, utilizada para representar frecuencias, patrones y matrices reducidas.
- ListaEnlazadaSimple y Nodo: Estructuras de datos personalizadas que sustituyen el uso de listas nativas de Python, cumpliendo con el principio de TDA.

Este diseño permite una clara separación de responsabilidades, facilitando el mantenimiento, la depuración y la escalabilidad del sistema.

b. Estructuras de datos personalizadas: Listas enlazadas anidadas

Una de las restricciones más importantes del proyecto fue no utilizar estructuras nativas de Python como listas, diccionarios o tuplas. Para cumplir con este requisito, se desarrollaron dos clases fundamentales: Nodo y ListaEnlazadaSimple.

La clase Nodo encapsula un contenido (por ejemplo, una estación o un registro) y un enlace al siguiente nodo, formando una cadena. La clase ListaEnlazadaSimple gestiona esta cadena, permitiendo operaciones como agregar, buscar y recorrer elementos mediante punteros.

A partir de esta estructura, se construyó una matriz anidada, donde cada fila es una ListaEnlazadaSimple, y el conjunto de filas forma otra ListaEnlazadaSimple. Esta representación permite

acceder a cualquier celda mediante índices y realizar operaciones como asignación, extracción y construcción de patrones, simulando el comportamiento de una matriz tradicional sin depender de estructuras nativas.

Este enfoque demuestra que es posible implementar estructuras complejas desde cero, promoviendo un entendimiento más profundo de los Tipos de Datos Abstractos (TDA) y fomentando buenas prácticas de programación.

c. Procesamiento de datos desde XML

El sistema recibe su entrada desde un archivo XML que contiene la configuración de múltiples campos agrícolas. Se utilizó el módulo xml.dom.minidom para parsear el archivo, extrayendo:

- Campos agrícolas
- Estaciones base
- Sensores de suelo y cultivo
- Frecuencias de transmisión

Cada campo se carga en memoria como una instancia Cada campo se carga en memoria como una instancia de CampoAgricola, y los datos se organizan en matrices de frecuencia. Este proceso es transparente para el usuario, quien solo debe indicar la ruta del archivo. El formato XML permite una representación clara y estructurada de los datos, facilitando su lectura y validación.

d. Algoritmo de agrupamiento por patrones

El proceso de optimización se basa en transformar las matrices de frecuencia en matrices de patrones, donde cada celda contiene un 1 si hay frecuencia mayor a cero, o 0 en caso contrario. Este paso convierte datos numéricos en patrones binarios que representan la conectividad entre estaciones y sensores.

Luego, se comparan las filas (estaciones) para identificar aquellas con el mismo patrón en ambas matrices (suelo y cultivo). Las estaciones con patrones idénticos se agrupan en una nueva estación base reducida, cuyo nombre combina los nombres originales (por ejemplo, "r1: Estacion 01, Estacion 03"). Las frecuencias se suman para mantener la integridad de la información, asegurando que no se pierda ningún dato.

Este algoritmo tiene una complejidad de $O(n^2)$ en el peor caso, pero es eficiente para instancias de tamaño moderado, como las presentadas en el proyecto. El resultado es una reducción efectiva del número de estaciones base sin pérdida de información, mejorando la eficiencia del sistema.

e. Visualización con Graphviz

Para facilitar el análisis del sistema, se implementó la generación de gráficas mediante Graphviz. El usuario puede elegir visualizar:

- Matriz de frecuencias (suelo y cultivo)
- Matriz de patrones (suelo y cultivo)
- Matriz reducida (suelo y cultivo)

Cada gráfica se genera como una tabla HTML embebida en un grafo, mostrando claramente la relación entre estaciones y sensores. El código .dot se guarda en la carpeta Graficasdot/, y la imagen .png se genera en Graficas/. Esta funcionalidad permite al usuario verificar visualmente el comportamiento del sistema, validar el proceso de agrupamiento y asegurar que las matrices reducidas sean correctas.

f. Menú y flujo de usuario

El sistema presenta un menú en consola con seis opciones:

1. Cargar Archivo: Permite ingresar la ruta y nombre del archivo XML.

2. Procesar Archivo: Muestra mensajes paso a paso del procesamiento (carga de campos, creación de estaciones, sensores, etc.).

3. Escribir Archivo de Salida: Genera un XML con estaciones base reducidas y frecuencias sumadas.

4. Mostrar Datos del Estudiante: Muestra información del autor del proyecto.

5. Generar Gráfica: Permite seleccionar un campo y tipo de matriz para graficar.

6. Salir: Finaliza la ejecución del programa.

Este flujo es intuitivo y guía al usuario a través del proceso completo, desde la carga de datos hasta la generación de resultados.

## Conclusiones

1. El sistema desarrollado cumple con el objetivo de optimizar el uso de estaciones base mediante el agrupamiento de patrones, reduciendo costos operativos sin pérdida de información.

2. La implementación de estructuras de datos personalizadas (listas enlazadas anidadas) demuestra que es posible desarrollar soluciones complejas sin depender de estructuras nativas de Python, cumpliendo con los principios de TDA.

3. El uso de POO permite un diseño modular, claro y mantenible, facilitando la extensión del sistema a nuevos tipos de sensores o algoritmos de agrupamiento.

4. La visualización con Graphviz mejora la comprensión del comportamiento del sistema, permitiendo validar el proceso de reducción.

5. El procesamiento de XML como entrada y salida garantiza la interoperabilidad con otros sistemas y cumple con el formato especificado.

Este proyecto no solo resuelve un problema técnico, sino que también sirve como ejemplo práctico de cómo aplicar conceptos teóricos de POO, TDA y algoritmos en un escenario real. Futuras mejoras podrían incluir el uso de inteligencia artificial para predecir patrones o la integración con una interfaz gráfica.

## Referencias bibliográficas

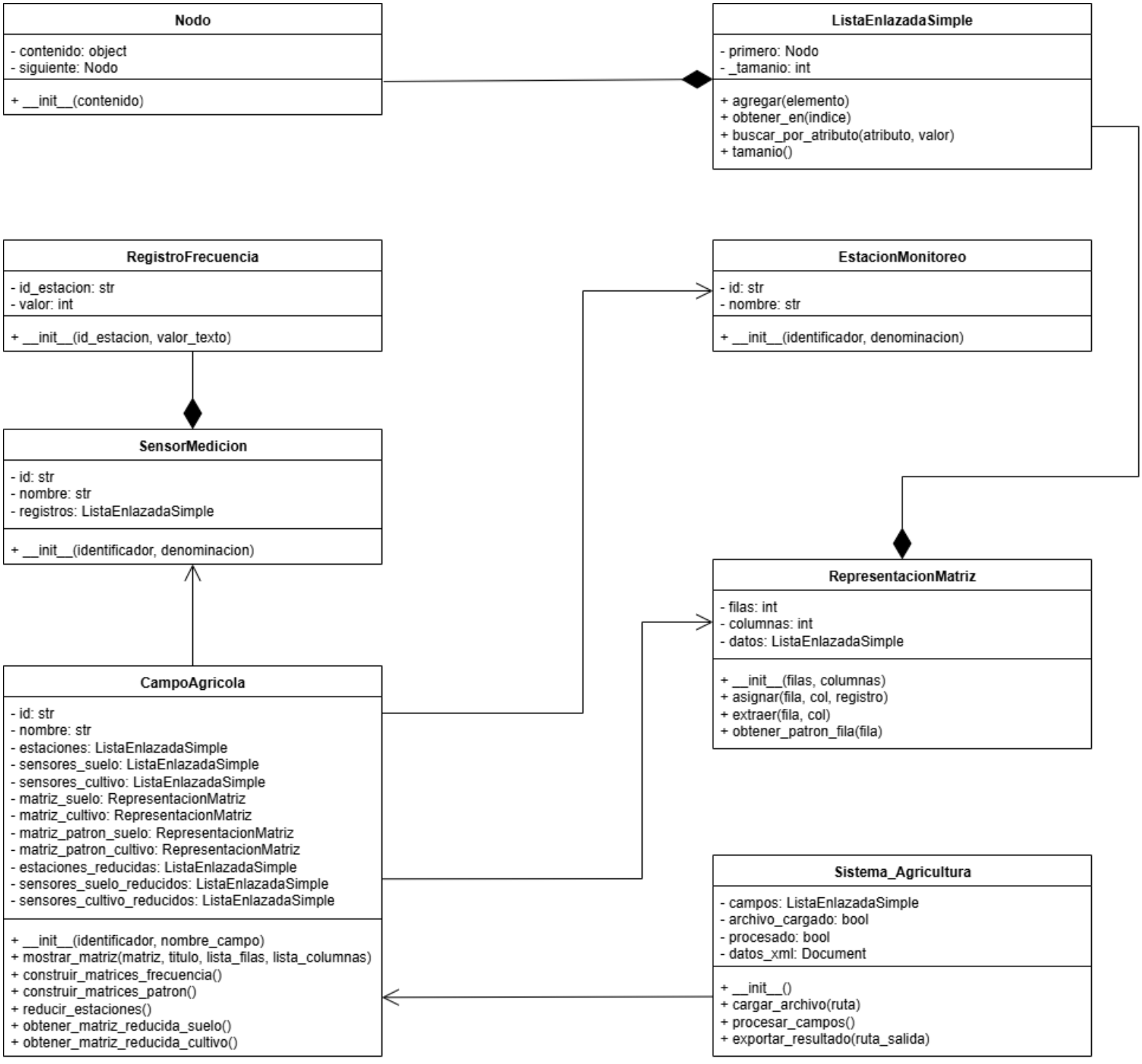
C. J. Date, (1991). *An introduction to Database Systems*. Addison-Wesley Publishing Company, Inc.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. Goodrich, M. T., & Tamassia, R. (2015). *Data Structures and Algorithms in Python*. Wiley.

McKinney, W. (2017). *Python for Data Analysis*. O'Reilly Media.

Pressman, R. S. (2014). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education.

# Apéndices


Nodo
- contenido: object
- siguiente: Nodo
+ __init__(contenido)
ListaEnlazadaSimple
- primero: Nodo
- _tamanio: int
+ agregar(elemento)
+ obtener_en(indice)
+ buscar_por_atributo(atributo, valor)
+ tamanio()
RegistroFrecuencia
- id_estacion: str
- valor: int
+ __init__(id_estacion, valor_texto)
EstacionMonitoreo
- id: str
- nombre: str
+ __init__(identificador, denominacion)
SensorMedicion
- id: str
- nombre: str
- registros: ListaEnlazadaSimple
+ __init__(identificador, denominacion)
RepresentacionMatriz
- filas: int
- columnas: int
- datos: ListaEnlazadaSimple
+ __init__(filas, columnas)
+ asignar(fila, col, registro)
+ extraer(fila, col)
+ obtener_patron_fila(fila)
CampoAgricola
- id: str
- nombre: str
- estaciones: ListaEnlazadaSimple
- sensores_suelo: ListaEnlazadaSimple
- sensores_cultivo: ListaEnlazadaSimple
- matriz_suelo: RepresentacionMatriz
- matriz_cultivo: RepresentacionMatriz
- matriz_patron_suelo: RepresentacionMatriz
- matriz_patron_cultivo: RepresentacionMatriz
- estaciones_reducidas: ListaEnlazadaSimple
- sensores_suelo_reducidos: ListaEnlazadaSimple
- sensores_cultivo_reducidos: ListaEnlazadaSimple
+ __init__(identificador, nombre_campo)
+ mostrar_matriz(matriz, titulo, lista_filas, lista_columnas)
+ construir_matrices_frecuencia()
+ construir_matrices_patron()
+ reducir_estaciones()
+ obtener_matriz_reducida_suelo()
+ obtener_matriz_reducida_cultivo()
Sistema_Agricultura
- campos: ListaEnlazadaSimple
- archivo_cargado: bool
- procesado: bool
- datos_xml: Document
+ __init__()
+ cargar_archivo(ruta)
+ procesar_campos()
+ exportar_resultado(ruta_salida)