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

PROYECTO 1

202102996 – Anferny Jefferson Jolón Palencia

Resumen

En este ensayo se muestra el desarrollo de un sistema computacional para analizar patrones en agricultura de precisión, hecho en Python y usando estructuras de datos enlazadas. El sistema trabaja con datos de sensores agrícolas ubicados en campos de cultivo, analizando las frecuencias de detección entre estaciones base y sensores de suelo y cultivo. La aplicación usa algoritmos de agrupamiento para reconocer estaciones con patrones parecidos, logrando optimizar la distribución de recursos en los sistemas de monitoreo agrícola. El programa también tiene funciones para cargar datos XML, procesar matrices de frecuencias y patrones, generar archivos de salida optimizados y mostrar resultados en gráficas PDF. Los impactos técnicos más importantes son la optimización de redes de sensores agrícolas y la disminución de redundancia en sistemas de monitoreo. Las conclusiones prueban que las estructuras de datos enlazadas son efectivas para procesar grandes volúmenes de datos agrícolas y encontrar automáticamente patrones de comportamiento en sensores distribuidos.

Palabras clave

Estructuras de datos, análisis de patrones, sensores agrícolas, optimización.

Abstract

This essay shows the development of a computational system for pattern analysis in precision agriculture, made in Python using linked data structures. The system works with data from agricultural sensors placed in crop fields, analyzing detection frequencies between base stations and soil and crop sensors. The application uses clustering algorithms to recognize stations with similar patterns, helping to optimize resource distribution in agricultural monitoring systems. The program also has functions for XML data loading, processing of frequency and pattern matrices, generation of optimized output files, and visualization of results with PDF graphics. The main technical impacts are the optimization of agricultural sensor networks and the reduction of redundancy in monitoring systems. The conclusions prove the usefulness of linked data structures for efficient processing of large agricultural datasets and the automatic recognition of behavior patterns in distributed sensors.

Keywords

Data structures, pattern analysis, agricultural sensors, optimization.

Introducción

La agricultura de precisión es una revolución tecnológica en el área agrícola, ya que la optimización de recursos y las decisiones basadas en datos científicos influyen en la productividad y sostenibilidad de los cultivos modernos. En este escenario, los sistemas de monitoreo mediante redes de sensores se han vuelto clave para recolectar y analizar información en tiempo real.

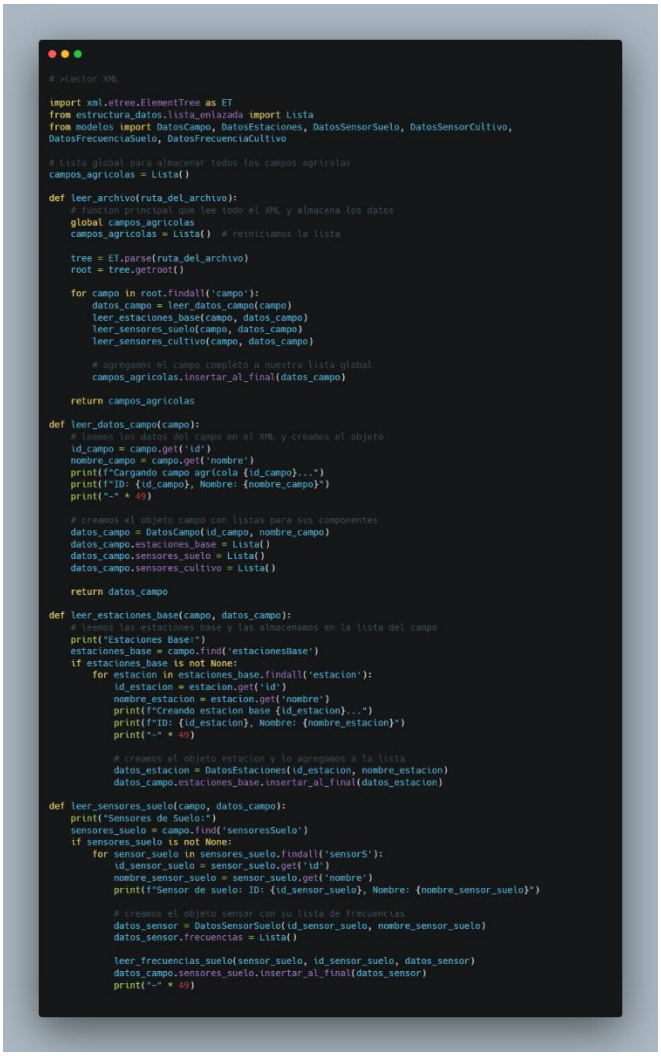
Este trabajo describe el desarrollo de un sistema computacional especializado en analizar patrones de comportamiento en redes de sensores agrícolas, usando estructuras de datos enlazadas como base. El sistema trata el problema de la redundancia en redes de sensores y la necesidad de optimizar la distribución de estaciones de monitoreo en campos agrícolas.

La importancia de este desarrollo está en su capacidad de procesar grandes cantidades de datos de sensores, detectar patrones de comportamiento entre estaciones base y generar recomendaciones para mejorar el uso de recursos en agricultura de precisión.

Desarrollo del tema

a. Arquitectura del Sistema y Estructuras de Datos:

El sistema se basa en la implementación de estructuras de datos enlazadas, en particular listas enlazadas, para manejar eficientemente la información agrícola. La arquitectura modular incluye los siguientes componentes: lector XML, procesador de matrices, generador de gráficas y sistema de menús interactivo.
La clase *Lista* implementa una estructura enlazada genérica que permite guardar información de forma dinámica sin un tamaño fijo. Esta parte es clave porque en los datos agrícolas el número de sensores, estaciones base y mediciones cambia según el campo de cultivo.
El módulo *lector_xml.py* usa un parser que convierte datos XML a objetos específicos, almacenándolos en estructuras enlazadas. Esto asegura escalabilidad y eficiencia en memoria.



```
# >Lector XML

import xml.etree.ElementTree as ET
from estructura_datos.lista_enlazada import Lista
from modelos import DatosCampo, DatosEstaciones, DatosSensorSuelo, DatosSensorCultivo,
DatosFrecuenciaSuelo, DatosFrecuenciaCultivo

# Lista global para almacenar todos los campos agricolas
campos_agricolas = Lista()

def leer_archivo(ruta_del_archivo):
    # Funcion principal que lee todo el XML y almacena los datos
    global campos_agricolas
    campos_agricolas = Lista()  # reiniciamos la lista

    tree = ET.parse(ruta_del_archivo)
    root = tree.getroot()

    for campo in root.findall('campo'):
        datos_campo = leer_datos_campo(campo)
        leer_estaciones_base(campo, datos_campo)
        leer_sensores_suelo(campo, datos_campo)
        leer_sensores_cultivo(campo, datos_campo)

        # agregamos el campo completo a nuestra lista global
        campos_agricolas.insertar_al_final(datos_campo)

    return campos_agricolas

def leer_datos_campo(campo):
    # leemos los datos del campo en el XML y creamos el objeto
    id_campo = campo.get('id')
    nombre_campo = campo.get('nombre')
    print(f"Cargando campo agrícola {id_campo}...")
    print(f"ID: {id_campo}, Nombre: {nombre_campo}")
    print("-" * 40)

    # creamos el objeto campo con listas para sus componentes
    datos_campo = DatosCampo(id_campo, nombre_campo)
    datos_campo.estaciones_base = Lista()
    datos_campo.sensores_suelo = Lista()
    datos_campo.sensores_cultivo = Lista()

    return datos_campo

def leer_estaciones_base(campo, datos_campo):
    # leemos las estaciones base y las almacenamos en la lista del campo
    print("Estaciones Base:")
    estaciones_base = campo.find('estacionesBase')
    if estaciones_base is not None:
        for estacion in estaciones_base.findall('estacion'):
            id_estacion = estacion.get('id')
            nombre_estacion = estacion.get('nombre')
            print(f"Creando estacion base {id_estacion}...")
            print(f"ID: {id_estacion}, Nombre: {nombre_estacion}")
            print("-" * 40)

            # creamos el objeto estacion y lo agregamos a la lista
            datos_estacion = DatosEstaciones(id_estacion, nombre_estacion)
            datos_campo.estaciones_base.insertar_al_final(datos_estacion)

def leer_sensores_suelo(campo, datos_campo):
    print("Sensores de Suelo:")
    sensores_suelo = campo.find('sensoresSuelo')
    if sensores_suelo is not None:
        for sensor_suelo in sensores_suelo.findall('sensorS'):
            id_sensor_suelo = sensor_suelo.get('id')
            nombre_sensor_suelo = sensor_suelo.get('nombre')
            print(f"Sensor de suelo: ID: {id_sensor_suelo}, Nombre: {nombre_sensor_suelo}")

            # creamos el objeto sensor con su lista de frecuencias
            datos_sensor = DatosSensorSuelo(id_sensor_suelo, nombre_sensor_suelo)
            datos_sensor.frecuencias = Lista()

            leer_frecuencias_suelo(sensor_suelo, id_sensor_suelo, datos_sensor)
            datos_campo.sensores_suelo.insertar_al_final(datos_sensor)
            print("-" * 40)
```

b. Algoritmos de Procesamiento de Matrices:

El módulo *procesador_matrices.py* es el núcleo del sistema e implementa algoritmos para analizar patrones en datos de sensores. El sistema crea dos tipos de matrices: matrices de frecuencias F[n,s] y F[n,t], y matrices de patrones binarios.
Las matrices de frecuencias muestran la relación entre estaciones base y sensores, donde cada valor indica la frecuencia con la que una estación detecta un sensor. El algoritmo convierte esas frecuencias en

matrices binarias: 1 si hay detección y 0 si no. Luego, el algoritmo de agrupamiento compara patrones para encontrar estaciones con comportamientos similares. Esto es muy importante porque ayuda a optimizar redes de sensores, consolidando estaciones redundantes y reduciendo costos.

```python
# Modulo que nos ayuda a procesar las matrices

from estructura_datos.lista_enlazada import Lista

class MatrizFrecuencias:
    def __init__(self, num_estaciones, num_sensores):
        # creamos una matriz usando listas enlazadas
        self.num_estaciones = num_estaciones
        self.num_sensores = num_sensores
        self.filas = Lista()  # cada fila es una lista de frecuencias

        # inicializamos la matriz con ceros
        for i in range(num_estaciones):
            fila = Lista()
            for j in range(num_sensores):
                fila.insertar_al_final(0)
            self.filas.insertar_al_final(fila)

    def establecer_valor(self, fila, columna, valor):
        # establece un valor en la posicion [fila][columna]
        if fila >= 0 and fila < self.num_estaciones and columna >= 0 and columna < self.num_sensores:
            fila_datos = self.filas.obtener_por_indice(fila)
            if fila_datos is not None:
                # tenemos que recorrer manualmente hasta la columna deseada
                actual = fila_datos.cabeza
                for i in range(columna):
                    if actual is not None:
                        actual = actual.siguiente
                if actual is not None:
                    actual.dato = valor

    def obtener_valor(self, fila, columna):
        # obtiene el valor en la posicion [fila][columna]
        if fila >= 0 and fila < self.num_estaciones and columna >= 0 and columna < self.num_sensores:
            fila_datos = self.filas.obtener_por_indice(fila)
            if fila_datos is not None:
                return fila_datos.obtener_por_indice(columna)
        return 0

    def obtener_fila_como_lista(self, num_fila):
        # retorna una fila completa como lista enlazada
        if num_fila >= 0 and num_fila < self.num_estaciones:
            return self.filas.obtener_por_indice(num_fila)
        return None

    def mostrar_matriz(self, titulo, ids_estaciones, ids_sensores):
        # imprime la matriz de forma legible
        print(f"\n{titulo}")
        print("=" * 50)

        # encabezados de columnas (sensores)
        print("Estacion\\Sensor", end="")
        for sensor in ids_sensores.iterar():
            print(f"\t{sensor}", end="")
        print()

        # filas con datos
        contador_fila = 0
        for estacion in ids_estaciones.iterar():
            print(f"{estacion}", end="")
            for j in range(self.num_sensores):
                valor = self.obtener_valor(contador_fila, j)
                print(f"\t{valor}", end="")
            print()
            contador_fila = contador_fila + 1
```

c. Visualización y Generación de Reportes:

El módulo *generador_graficas.py* genera visualizaciones con Graphviz y crea gráficas en PDF. El sistema muestra tablas de matrices de frecuencias, patrones y matrices reducidas después de la optimización.
Con tablas HTML dentro de Graphviz se logra una representación visual clara de los datos, lo que ayuda a especialistas a interpretar mejor los resultados. Además, el uso de colores diferencia tipos de sensores y valores de frecuencia, lo que hace más fácil leer los reportes.

```python
# Modulo que nos permite generar nuestras gráficas usando Graphviz

try:
    import graphviz
    GRAPHVIZ_DISPONIBLE = True
except ImportError:
    GRAPHVIZ_DISPONIBLE = False
    print("Advertencia: Graphviz no está instalado. Use: pip install graphviz")

from estructura_datos.lista_enlazada import Lista

class GeneradorGraficas:
    def __init__(self):
        self.disponible = GRAPHVIZ_DISPONIBLE

    def _crear_tabla_html(self, matriz, etiquetas_filas, etiquetas_columnas, titulo):
        """Crea una tabla HTML para usar en Graphviz"""
        html = '<<TABLE BORDER="1" CELLBORDER="1" CELLSPACING="0" CELLPADDING="4">'

        # Título de la tabla
        num_columnas = etiquetas_columnas.obtener_tamaño() + 1
        html = html + '<TR><TD COLSPAN="' + str(num_columnas) + '" BGCOLOR="lightgray"><B>' + titulo +
'</B></TD></TR>'

        # Encabezados de columnas
        html = html + '<TR><TD BGCOLOR="lightblue"><B> </B></TD>'
        for col in etiquetas_columnas.iterar():
            html = html + '<TD BGCOLOR="lightblue"><B>' + str(col) + '</B></TD>'
        html = html + '</TR>'

        # Filas con datos
        contador_fila = 0
        for fila_label in etiquetas_filas.iterar():
            html = html + '<TR><TD BGCOLOR="lightgreen"><B>' + str(fila_label) + '</B></TD>'
            for j in range(etiquetas_columnas.obtener_tamaño()):
                valor = matriz.obtener_valor(contador_fila, j)
                # Colorear las celdas según el valor
                if valor == 0:
                    color = "white"
                elif valor == 1:
                    color = "lightcyan"
                else:
                    color = "lightyellow"
                html = html + '<TD BGCOLOR="' + color + '">' + str(valor) + '</TD>'
            html = html + '</TR>'
            contador_fila = contador_fila + 1

        html = html + '</TABLE>>'
        return html

    def generar_tabla_matriz_frecuencias(self, campo, tipo_sensor="suelo"):
        """Genera la tabla de la matriz de frecuencias F[n,s] o F[n,t] como gráfica"""
        if not self.disponible:
            print("Error: Graphviz no está disponible")
            return False

        if not hasattr(campo, 'procesador'):
            print("Error: El campo no ha sido procesado")
            return False

        # Creamos el grafo
        dot = graphviz.Digraph(comment='Tabla Matriz Frecuencias - ' + campo.nombre)
        dot.attr(rankdir='TB')
        dot.attr(dpi='300')  # Alta resolución para gráfica

        # Obtenemos datos según el tipo de sensor
        if tipo_sensor == "suelo":
            sensores = campo.sensores_suelo
            matriz = campo.procesador.matriz_suelo
            titulo = "Matriz de Frecuencias - Sensores de Suelo - " + campo.nombre
        else:
            sensores = campo.sensores_cultivo
            matriz = campo.procesador.matriz_cultivo
            titulo = "Matriz de Frecuencias - Sensores de Cultivo - " + campo.nombre

        # Crear etiquetas usando listas enlazadas
        etiquetas_filas = Lista()
        for estacion in campo.estaciones_base.iterar():
            etiquetas_filas.insertar_al_final("Fe" + str(estacion.id).zfill(2))

        etiquetas_columnas = Lista()
        for sensor in sensores.iterar():
            etiquetas_columnas.insertar_al_final("Ss" + str(sensor.id).zfill(2))

        # Crear la tabla HTML
        tabla_html = self._crear_tabla_html(matriz, etiquetas_filas, etiquetas_columnas, titulo)

        # Agregar el nodo con la tabla
        dot.node('tabla', tabla_html, shape='none')

        # Guardar como gráfica
        nombre_archivo = "tabla_frecuencias_" + tipo_sensor + "_" + str(campo.id)
        try:
            dot.render(nombre_archivo, format='pdf', cleanup=True)
            print("Gráfica generada: " + nombre_archivo + ".pdf")
            return True
        except Exception as e:
            print("Error al generar gráfica: " + str(e))
            return False
```

d. Optimización y Reducción de Redundancia:

El algoritmo de reducción busca grupos de estaciones

base con el mismo patrón de comportamiento y las une en una sola estación representativa. Esto soluciona la redundancia en redes de sensores, algo común en la agricultura de precisión. Durante la reducción se conserva la información original porque las frecuencias de las estaciones se suman, manteniendo la precisión de los datos y optimizando la red de monitoreo.

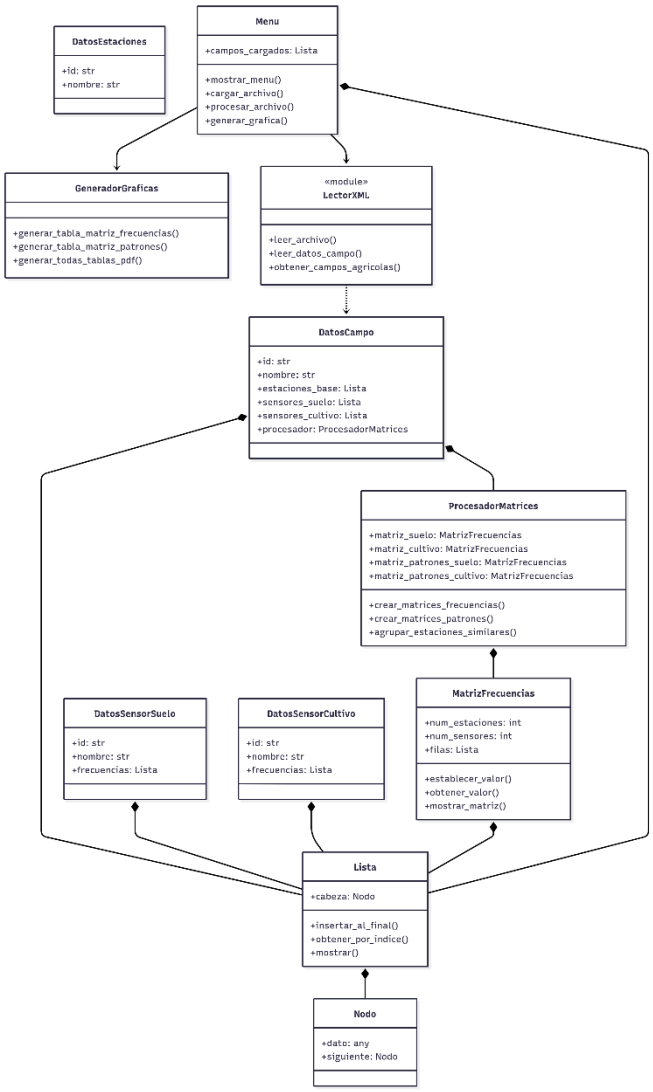


Figura 1. Diagrama de clases

Fuente: mermaidchart.com, 2025

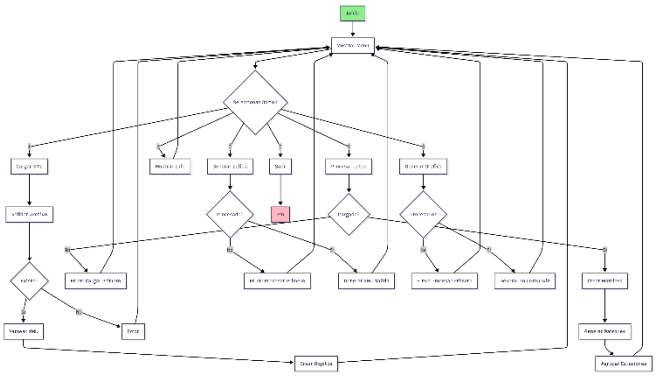


Figura 1. Diagrama de actividades

Fuente: mermaidchart.com, 2025

Tabla I.

El título de la tabla debe ser corto y conciso.

COMPONENTE	FUNCIONALIDAD PRINCIPAL
Lector XML	Parseo y carga de datos agrícolas
Procesador Matrices	Análisis de patrones y agrupamiento
Generador Gráficas	Visualización de resultados
Sistema Menús	Interfaz de usuario interactiva

Fuente: elaboración propia, 2025

El sistema además tiene validaciones de integridad de datos y manejo de excepciones, lo que le da robustez en ambientes de producción agrícola.

Conclusiones

El sistema desarrollado para analizar patrones en agricultura de precisión muestra que las estructuras de datos enlazadas son muy efectivas al procesar información agrícola compleja. Gracias a su diseño modular se facilita el mantenimiento y ampliación del sistema, mientras que los algoritmos de agrupamiento

aportan un valor real optimizando las redes de sensores.

Los resultados demuestran que el sistema puede encontrar automáticamente patrones de redundancia en sensores distribuidos, ayudando a reducir costos en agricultura de precisión. Además, la parte de visualización facilita entender resultados complejos, lo que mejora la toma de decisiones.

Las limitaciones detectadas incluyen la dependencia de archivos XML y la necesidad de optimizar más el procesamiento para grandes volúmenes de datos. En próximas versiones sería ideal usar bases de datos relacionales y algoritmos de procesamiento paralelo para trabajar mejor en escenarios a gran escala.

Referencias bibliográficas

Python Software Foundation. (2023). Python Programming Language. Python.org.

Graphviz Development Team. (2023). Graphviz - Graph Visualization Software. Graphviz.org.

Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2022). Introduction to Algorithms. MIT Press. Zhang, C., & Kovacs, J. (2021). "Precision Agriculture and IoT Sensor Networks." Journal of Agricultural Technology, 18(3), 245-267.

McKinney, W. (2022). Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter. O'Reilly Media.