Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 1er. Semestre 2025

PROYECTO 1: GESTOR PARA LA AGRICULTURA DE PRECISIÓN

201318644 – Mynor Alejandro Cifuentes Velásquez

Resumen

La gestión eficiente de datos en la agricultura de precisión representa una tendencia innovadora a nivel nacional como internacional. La integración de sistemas de información mediante estructuras de datos enlazadas, como matrices y listas simples, permite el procesamiento de grandes volúmenes de información provenientes de sensores y estaciones distribuidas en los campos. Esta tecnología facilita la toma de decisiones fundamentadas, optimizando el uso de recursos y mejorando el rendimiento agrícola. Los impactos técnicos incluyen la mejora en la visualización y análisis de datos; en el ámbito económico, se traduce en un incremento de la productividad y reducción de costos; socialmente, contribuye a la seguridad alimentaria; y ambientalmente, promueve prácticas sostenibles.

Palabras clave

Agricultura de precisión, gestión de datos, estructuras enlazadas, sensores, sostenibilidad

Abstract

Efficient data management in precision agriculture stands out as an innovative trend both nationally and internationally. The integration of information systems using linked data structures, such as matrices and simple lists, enables the processing of large volumes of information collected from sensors and stations distributed throughout agricultural fields. This technology supports informed decision-making, optimizing resource use and enhancing crop performance. On a technical level, it improves data visualization and analysis; economically, it leads to increased productivity and reduced costs; socially, it contributes to food security; and environmentally, it encourages sustainable farming practices.

Keywords

Precision agriculture, data management, linked data structures, sensors, sustainability

Introducción

La agricultura de precisión se ha convertido en una herramienta fundamental para enfrentar los retos actuales de la producción agrícola, tales como la optimización de recursos, el aumento de la productividad y la sostenibilidad ambiental. En este contexto, la gestión eficiente de datos provenientes de sensores y estaciones distribuidas en los campos adquiere una relevancia estratégica, pues permite transformar grandes volúmenes de información en conocimiento útil para la toma de decisiones

Desarrollo del tema

La gestión avanzada de datos en agricultura de precisión utiliza algoritmos y estructuras enlazadas para optimizar la toma de decisiones en campo. El procesamiento de información proveniente de sensores y estaciones permite analizar variables clave para la productividad agrícola. En este contexto, el uso de matrices enlazadas y listas simples constituye una base teórica sólida para el manejo eficiente y flexible de grandes volúmenes de datos.

Estructuras de datos y Algoritmos

El sistema implementa las clases ListaSimple, ListaDoble, Celda y Matriz, que permiten almacenar y recorrer datos de manera eficiente. Por ejemplo, la función para completar ceros en la matriz garantiza que todos los cruces estación-sensor tengan un valor, mejorando la integridad del análisis

```
def completar_ceros(self):
    temp_est = self.estaciones.primero
    while temp_est is not None:
        idEstacion = temp_est.dato
        nodo_fila = self.filas.buscar(idEstacion)
        temp_sens = self.sensores.primero
        while temp_sens is not None:
            idSensor = temp_sens.dato
            celda_existente = nodo_fila.fila.buscar(idSensor, "idSensor")
            if celda_existente is None:
                nueva_celda = Celda(idSensor=idSensor, idEstacion=idEstacion, frecuencia="0")
                nodo_fila.fila.agregarUltimo(nueva_celda)
            temp_sens = temp_sens.siguiente
        temp_est = temp_est.siguiente
```

Las listas simples permiten almacenar elementos en una estructura lineal donde cada nodo apunta al siguiente, facilitando operaciones de inserción, búsqueda y recorrido. Se utilizan para representar conjuntos de sensores, estaciones y celdas en la matriz.

```
class Nodo:
    def __init__(self, dato=None):
        self.dato = dato
        self.siguiente = None

class ListaSimple:
    def __init__(self):
        self.primero = None
        self.ultimo = None
        self.tamanio = 0
```

Inserción al final de la lista

La función agregarUltimo permite insertar elementos al final de la lista, asegurando el mantenimiento del orden de llegada de datos.

```
def agregarUltimo(self, dato):
    nuevo = Nodo(dato=dato)
    if self.estaVacia():
        self.primero = self.ultimo = nuevo
    else:
        self.ultimo.siguiente = nuevo
        self.ultimo = nuevo
    self.tamanio += 1
```

Búsqueda de Elementos

La búsqueda en la lista puede realizarse por valor simple o por atributo, especialmente útil al buscar sensores (por su ID) o celdas (por el atributo idSensor).

```
def buscar(self, valor, atributo=None):
    temp = self.primero
    while temp is not None:
        if atributo is None:
            if temp.dato == valor:
                return temp
        else:
            if hasattr(temp.dato, atributo):
                if getattr(temp.dato, atributo) == valor:
                    return temp
        temp = temp.siguiente
    return None
```

Recorrido y Visualización

El recorrido de la lista se realiza de manera secuencial, permitiendo visualizar los elementos o realizar operaciones como el conteo o sumatoria.

Carga y visualización de datos

```
def recorrer(self):
    temp = self.primero
    while temp is not None:
        print(str(temp.dato), end=" | ")
        temp = temp.siguiente
    print()
```

Listas Doblemente Enlazadas: Filas de la Matriz

Las listas doblemente enlazadas son empleadas para representar las filas de la matriz, donde cada nodo corresponde a una estación y contiene una lista simple de celdas (valores de frecuencia por sensor).

Definición de NodoEncabezado y ListaDoble

```
class NodoEncabezado:
    def __init__(self, idEstacion=None):
        self.idEstacion = idEstacion
        self.siguiente = None
        self.anterior = None
        self.fila = ListaSimple()
```

La clase ListaDoble administra la colección de nodos encabezados (estaciones):

```
class ListaDoble:
    def __init__(self):
        self.primero = None
        self.ultimo = None
        self.tamanio = 0
```

Inserción y Búsqueda

```
def agregarUltimo(self, idEstacion):
    nuevo = NodoEncabezado(idEstacion=idEstacion)
    if self.estaVacio():
        self.primero = self.ultimo = nuevo
    else:
        nuevo.anterior = self.ultimo
        self.ultimo.siguiente = nuevo
        self.ultimo = nuevo
    self.tamanio += 1
    return nuevo

def buscar(self, idEstacion):
    temp = self.primero
    while temp is not None:
        if temp.idEstacion == idEstacion:
            return temp
        temp = temp.siguiente
    return None
```

Recorrido y Visualización

El recorrido por filas permite visualizar los valores de frecuencia por estación, facilitando el análisis de patrones.

```
def recorrer(self, sensores):
    temp = self.primero
    while temp is not None:
        print(f"Estación {temp.idEstacion}: ", end="")
        temp.fila.recorrer()
        temp = temp.siguiente
```

La clase Gestor permite la carga de archivos XML con datos agrícolas, llenando las matrices y generando reportes visuales en consola y gráficos PNG:

```
def leer_archivo(self, ruta_entrada):
    # Procesa el archivo XML y llena matrices
    campo_obj.matriz_suelo.completar_ceros()
    campo_obj.matriz_cultivo.completar_ceros()
```

Análisis de Archivos XML con ElementTree

La gestión eficiente de datos inicia con la correcta interpretación de los archivos XML, que contienen la información de sensores, estaciones y frecuencias.

Carga y Procesamiento del XML

El módulo xml.etree.ElementTree permite la carga y análisis estructurado del archivo XML.

```
import xml.etree.ElementTree as ET

def leer_archivo(self, ruta_entrada):
    self.limpiar()
    try:
        tree = ET.parse(ruta_entrada)
        root = tree.getroot()
        for campo_elem in root.findall("campo"):
            id_campo = campo_elem.get("id")
            nombre_campo = campo_elem.get("nombre")
            campo_obj = Campo(id_campo, nombre_campo)
            self.campos.agregarUltimo(campo_obj)

            estaciones_base = campo_elem.find("estacionesBase")
            if estaciones_base is not None:
                for estacion_elem in estaciones_base.findall("estacion"):
                    id_est = estacion_elem.get("id")
                    campo_obj.matriz_suelo.agregar_estacion(id_est)
                    campo_obj.matriz_cultivo.agregar_estacion(id_est)

            sensores_suelo = campo_elem.find("sensoresSuelo")
            if sensores_suelo is not None:
                for sensorS_elem in sensores_suelo.findall("sensorS"):
                    id_sensorS = sensorS_elem.get("id")
                    campo_obj.matriz_suelo.agregar_sensor(id_sensorS)

            sensores_cultivo = campo_elem.find("sensoresCultivo")
            if sensores_cultivo is not None:
                for sensorT_elem in sensores_cultivo.findall("sensorT"):
                    id_sensorT = sensorT_elem.get("id")
                    campo_obj.matriz_cultivo.agregar_sensor(id_sensorT)

            # Llena valores del XML
            if sensores_suelo is not None:
                for sensorS_elem in sensores_suelo.findall("sensorS"):
                    id_sensorS = sensorS_elem.get("id")
                    for frecuencia_elem in sensorS_elem.findall("frecuencia"):
                        id_estacion = frecuencia_elem.get("idEstacion")
                        frecuencia = frecuencia_elem.text
                        campo_obj.matriz_suelo.agregar_valor(id_estacion, id_sen

            if sensores_cultivo is not None:
                for sensorT_elem in sensores_cultivo.findall("sensorT"):
                    id_sensorT = sensorT_elem.get("id")
                    for frecuencia_elem in sensorT_elem.findall("frecuencia"):
                        id_estacion = frecuencia_elem.get("idEstacion")
                        frecuencia = frecuencia_elem.text
                        campo_obj.matriz_cultivo.agregar_valor(id_estacion, id_s

            # Completa con ceros si falta algún dato
            campo_obj.matriz_suelo.completar_ceros()
            campo_obj.matriz_cultivo.completar_ceros()

        self.cargado = True
        print("Archivo cargado y procesado correctamente.")
    except Exception as e:
        print("Error al procesar archivo:", e)
```

Gráfica de Matrices con Graphviz

Visualizar la información de forma gráfica es esencial para el análisis y toma de decisiones en la agricultura de precisión. El sistema utiliza Graphviz para convertir las matrices en imágenes tipo PNG, facilitando la interpretación visual.

Generación del Archivo DOT

El método graficar de la matriz crea un archivo DOT que representa la matriz como una tabla HTML.

```
def graficar(self, nombre_archivo, nombre_img, sensores):
    carpeta = os.path.dirname(nombre_archivo)
    if carpeta and not os.path.exists(carpeta):
        os.makedirs(carpeta)
    contenido = """
graph G {
  node [shape=plaintext];
  matriz [label=<
    <TABLE BORDER="1" CELLBORDER="1" CELLSPACING="0" >
"""
    temp = self.primero
    while temp is not None:
        contenido += "<TR>"
        contenido += f'<TD WIDTH="50" HEIGHT="50">{temp.idEstacion}</TD>'
        contenido += temp.fila.graficarCelda(sensores)
        contenido += "</TR>\n"
        temp = temp.siguiente
    contenido += """
    </TABLE>
  >];
}
"""
    with open(nombre_archivo, "w", encoding="utf-8") as file:
        file.write(contenido)
```

Gráfica de Celdas

La función graficarCelda recorre los sensores y agrega las frecuencias de cada estación como celdas de la tabla.

```
def graficarCelda(self, sensores):
    cadena = ""
    temp_sens = sensores.primero
    while temp_sens is not None:
        idSensor = temp_sens.dato
        temp_celda = self.buscar(idSensor, "idSensor")
        freq = temp_celda.dato.frecuencia if temp_celda else "0"
        cadena += f'<TD WIDTH="50" HEIGHT="50">{freq}</TD>\n'
        temp_sens = temp_sens.siguiente
    return cadena
```

Conversión a PNG

Luego de generar el archivo DOT, se utiliza el comando de Graphviz para convertirlo en una imagen PNG

```
os.system(f"dot -Tpng {nombre_archivo} -o {nombre_img}")
print("Gráfica generada exitosamente en la carpeta Salidas.")
```

Conclusiones

La implementación de Tipos de Datos Abstractos, particularmente las listas enlazadas y matrices construidas a partir de ellas, ha demostrado ser fundamental para la gestión eficiente de grandes volúmenes de información en la agricultura de precisión. Estas estructuras permitieron modelar datos complejos provenientes de sensores y estaciones, facilitando su almacenamiento, recorrido, visualización y sus respectivas gráficas en Graphviz.

Referencias bibliográficas

Máximo 5 referencias en orden alfabético.

Sierra, J. (2017). Algoritmos y estructuras de datos en Python. Marcombo.

Joyanes Aguilar, L. (2015). Fundamentos de programación: Algoritmos, estructuras de datos y objetos (3ra ed.). McGraw-Hill Interamericana.

Extensión: de cuatro a siete páginas como máximo

Adicionalmente, se pueden agregar apéndices con modelos, tablas, etc. Que complementan el contenido del trabajo.