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

Software Solution GT

201503846 – Mario Noel Giron Maldonado

Resumen

En este proyecto se planteó representar un campo agrícola con sensores y sus estaciones para el cultivo como una matriz usando estructuras de datos propias, las estaciones son las filas y los sensores las columnas y los datos de cada casilla de la matriz es la frecuencia de cada sensor, con esto se buscaba hacer una resta de filas con el mismo patrón para reducir la matriz y representar usando graphviz

Palabras clave

TDA, POO, matriz, listas, nodos

Abstract

In this project, it was proposed to represent an agricultural field with sensors and its seasons for cultivation as a matrix using its own data structures, the seasons are the rows and the sensors the columns and the data of each box of the matrix is the frequency of each sensor, with this, it was sought to do a subtraction of rows with the same pattern to reduce the matrix and represent it using graphviz

Keywords

TDA, OOP, matrix, list, node

Introducción

En la programación orientada a objetos, su ventaja es poder plantear soluciones a problemas, dividiendo el problema principal en pequeños problemas y solucionándolo hasta tener un programa que es fácil de modificar para adaptarse o reutilizarse según la situación que se necesita, el problema de los campos se planteó como una matriz porque se deben realizar una reducción, este proceso en la matriz es una resta de filas, con esto aclarado se realizó una estructura de datos para almacenar el número de sensores y estaciones con sus frecuencias una lista doble, cada elemento de la lista contendrá el ID y nombre del campo, dentro de este campo se guardara la información de las estaciones y sensores, con esto ya se podrá operar la matriz de cada campo y sus respectivos valores

Desarrollo del tema

Las estructuras principales con las que se manejaron este proyecto son las listas. Las listas son un conjunto de datos apuntando a otro dato

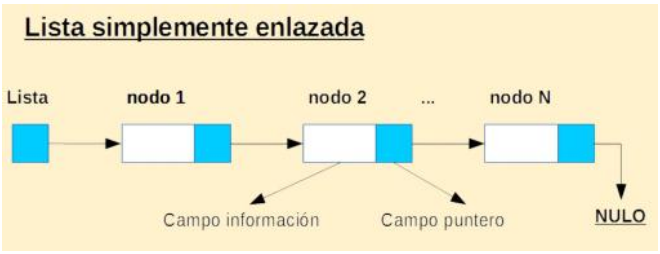

Figura 1. Lista enlazada.
Fuente: elaboración propia.

este conjunto de datos se les llama Nodos la ventaja es que puede almacenar datos o, en nuestro caso, clases que pueden contener datos y funciones.

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

Figura 2. Nodo.
Fuente: elaboración propia.

lo siguiente es la estructura que tendrán los nodos. La lista toma cada nodo y los conecta entre sí, para esto importamos la clase nodo a la clase lista. Esta inicia con los siguientes datos: primero y size que es el tamaño que tiene la lista. Su primer método es para identificar si la lista está vacía. Esto lo hace viendo nuestro primer dato que, al momento de crearse, está vacío, ya que no posee ningún dato.

El siguiente método es el de insertar. En este toma como parámetro un dato el que sé y lo primero que hace es crear un nodo y hacer una comprobación, si el primer dato de nuestra lista es None, este pasa a tener el valor de nodo que se creó.

Si nuestro primero dato de la lista resulta no ser None crea una variable temporal y entrará en un bucle en donde pasara al siguiente nodo y revisar su puntero siguiente hasta encontrar uno que se None y allí insertar el nuevo nodo que se creó al principio con esta se crea la estructura de una lista con los nodos

```
from nodo_le import nodo_le

You, 1 hour ago | 1 author (You)
class lista_enlazada:
    def __init__(self):
        self.primero = None
        self.size = 0

    def lista_vacia(self):
        return self.primero == None

    def insertar(self, dato):
        nuevo = nodo_le(dato)

        if self.primero == None:
            self.primero = nuevo

        else:
            actual = self.primero
            while actual.siguiente != None:
                actual = actual.siguiente
            actual.siguiente = nuevo
        self.size += 1
```

Figura 3. Lista enlazada.
Fuente: elaboración propia.

Las siguientes estructuras son muy parecidas, ya que salen de estas mismas clases la lista doble y su nodo doble.

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

Figura 4. Nodo doble
Fuente: elaboración propia.

En este el nodo de una lista doble, su diferencia es que ahora tiene otro apuntador. En la lista simple, los datos solo apuntaban a un dato al que le seguía . En este, puede apuntar tanto al dato que le sigue como al que le precede

```python
import os
from nodo_ld import nodo_ld
You, 2 hours ago | 1 author (You)
class lista_doble:
    def __init__(self):
        self.primero = None
        self.size = 0

    def lista_vacia(self):
        return self.primero == None

    def insertar(self, dato):
        nuevo = nodo_ld(dato)

        if self.primero == None:
            self.primero = nuevo

        else:
            actual = self.primero
            while actual.siguiente != None:
                actual = actual.siguiente
            actual.siguiente = nuevo
            nuevo.anterior = actual
        self.size += 1

    def agregar(self, dato):
        nuevo = nodo_ld(dato)

        if self.primero == None:
            self.primero = nuevo

        else:
            nuevo.siguiente = self.primero
            self.primero.anterior = nuevo
            self.primero = nuevo

        self.size += 1
```

Figura 5. Lista doble
Fuente: elaboración propia.

La estructura de la lista doble es idéntica a la lista enlazada, tanto los métodos como lista vacía. El cambio que vemos en el de insertar es que ahora, al insertar un nuevo dato, también tenemos que indicarle al apuntador anterior qué dato viene antes que él.
La función agregar es un método que nos permite ingresar un dato al inicio de nuestra lista. Lo que hace es que el apuntador del nuevo nodo apunte al primer nodo de nuestra lista y ahora él apuntador anterior del primer nodo apunta al nuevo.

```python
from lista_doble import lista_doble
from celda import celda
You, 2 hours ago | 1 author (You)
class campo_agricola:
    def __init__(self, id_campo, nombre_campo):
        self.id_campo = id_campo
        self.nombre_campo = nombre_campo
        self.matriz = lista_doble()

    def matriz_frecuencias(self, estaciones, sensores):
        for y in range(estaciones):
            encabezado = self.matriz.insertar(celda(n = 0, s = 1))
            for x in range(sensores):
                encabezado.fila.insertar(celda(n = x, s = y))      You, 7 hours a
        print("Se creo el campo agricola")
```

Figura 6. Clase campo
Fuente: elaboración propia.

La clase campó agrícola contendrá la información de los campos, cada campo tiene un, id y un nombre único también cada campo tendrá un atributo matriz que es una lista doble está lista contendrá en cada nodo una lista simple hace es como crearemos la matriz cada lista simple será una fila y la lista dobles eran las columnas
La función matriz frecuencias creará la matriz de cada campo usando como parámetros la cantidad de estaciones y sensores que posee cada uno, para esto nuestro primer for recorrerá el número de estaciones e insertará en el atributo matriz este atributo al ser una lista doble tiene todos sus métodos y luego entrará en otro for para insertar los sensores como definimos en los nodos de la lista doble cada uno tiene una lista simple en el atributo fila

```python
class celda:
    def __init__(self, frecuencia="0", s = -1, n = -1):
        self.frecuencia = frecuencia
        self.s = s
        self.n = n      You, 7 hours ago • se agrego a la
```

Figura . Clase celda
Fuente: elaboración propia.

La clase celda es llamada en la función matriz y nos ayuda a guardar las coordenadas de la matriz usando el contador x y en los for así cada celda tendrá su posición en la matriz. Cuando busquemos un dato, lo haremos con las coordenadas.

```python
def cargarArchivo():
    ruta = input("Ingrese la ruta del archivo: ")
    leerArchivo(ruta)

def leerArchivo(rutaArchivo):
    tree = ET.parse(rutaArchivo)
    root = tree.getroot()

    for elementos_campo in root.findall('campo'):
        id = elementos_campo.get('id')
        nombre = elementos_campo.get('nombre')
        campo = campo_agricola(id, nombre)

        for elementos_sensor in elementos_campo.find('estacionesBase').findall('estacion'):
            id = elementos_sensor.get('id')
            nombre = elementos_sensor.get('nombre')
            estacion = nodo_ld(nombre)
            campo.fila.insertar(estacion)

        columnasXml.insertar(campo)
    columnasXml.imprimir()      You, yesterday • se agrego un metodo str a la clase campo para i…
    campo.fila.imprimir()
```

Figura 6. Función para leer un archivo
Fuente: elaboración propia.

En el archivo principal en donde se ejecutan todas las clases y funciones, tendremos una función para leer un archivo XML de donde extraeremos todos los datos de los campos.
Esto lo haremos con la librería de XML element tree para leer y escribir archivos, le pediremos la ruta del archivo al usuario y recorreremos cada elemento del archivo por medio de sus etiquetas. todos estos valores los capturamos en todas las clases vistas anteriormente.

Conclusiones

Aunque se lograron plantear algunas modelos y clases para el proyecto, debido al tiempo no se lograron implementar todo lo que el proyecto solicitaba, como el uso de Graphviz para representar los campos de cultivo y la resta de filas de la matriz del campo

Referencias bibliográficas

Larman, C. (2004). *Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development* (3rd ed.). Prentice Hall.

Python aplicado a la POO (Muy claro y práctico):

*Phillips, D. (2014). *Python 3 object-oriented programming* (2nd ed.). Packt Publishing.*