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

201800956 – Hugo Lizandro Ramirez Siquinajay

Resumen

El presente ensayo expone el proceso de desarrollo de un proyecto en el curso de Introducción a la Programación y Computación 2, donde el objetivo principal fue la lectura, análisis y reducción de estaciones agrícolas descritas en archivos XML. Para ello se diseñaron estructuras de datos propias, evitando el uso de listas nativas, y se implementaron clases específicas para modelar campos, estaciones y sensores. A lo largo del trabajo se construyó un sistema capaz de procesar los datos, generar patrones de comportamiento, reducir la redundancia de estaciones y producir un archivo XML de salida más optimizado. También se añadió la opción de visualización mediante Graphviz, lo que permitió representar gráficamente la relación entre los elementos. Este ensayo describe paso a paso el funcionamiento de cada módulo, las decisiones de diseño y las lecciones aprendidas.

Palabras clave:

- XML
- Estructuras de datos
- Listas enlazadas
- Programación en Python
- Graphviz

.

Abstract

This essay presents the development process of a project in the course Introduction to Programming and Computing 2, where the main goal was the reading, analysis, and reduction of agricultural stations described in XML files. For this purpose, custom data structures were designed, avoiding the use of native lists, and specific classes were implemented to model fields, stations, and sensors. Throughout the project, a system was built capable of processing the data, generating behavior patterns, reducing station redundancy, and producing a more optimized output XML file. Additionally, a visualization option was added using Graphviz, which allowed to graphically represent the relationship between the elements. This essay describes step by step the operation of each module, the design decisions, and the lessons learned.

Keywords:

- *XML*
- *Data structures*
- *Linked lists*
- *Python programming*
- *Graphviz*

Introducción

El proyecto desarrollado surge como una práctica integral para consolidar los conocimientos en estructuras de datos, programación orientada a objetos y manejo de archivos XML. La temática planteada gira en torno al control de campos agrícolas y estaciones de monitoreo, cada una con sensores específicos que almacenan frecuencias de datos. El sistema debía ser capaz de leer esta información, organizarla de manera eficiente en estructuras enlazadas, aplicar un proceso de reducción de redundancia y finalmente producir resultados comprensibles tanto en archivos como en gráficos. Este ensayo explica en detalle las fases del proyecto, sus clases principales, los métodos implementados y la lógica de funcionamiento, con el fin de documentar el proceso y los aprendizajes obtenidos.

Desarrollo del tema

Lectura y análisis del archivo XML

La primera fase del proyecto consistió en implementar la clase encargada de leer el archivo XML. Este archivo contenía la información jerárquica de campos, estaciones y sensores. Se utilizó la librería ElementTree de Python para recorrer los nodos y extraer los datos relevantes. El método cargar_archivo(ruta) permitió obtener el documento desde la ruta indicada por el usuario, y procesar_archivo() se encargó de recorrer los elementos y almacenarlos en estructuras personalizadas. Esta parte fue crucial porque representó la base de datos sobre la que trabajaría el resto del sistema.

```
class XMLManager:
    def cargar_archivo(self, ruta):
        root = tree.getroot()

        for campo_xml in root.findall("campo"):
            id_campo = campo_xml.get("id")
            nombre_campo = campo_xml.get("nombre")
            campo = CampoAgricola(id_campo, nombre_campo)

            # Estaciones
            estaciones_xml = campo_xml.find("estacionesBase")
            if estaciones_xml is not None:
                for est_xml in estaciones_xml.findall("estacion"):
                    est = Estacion(est_xml.get("id"), est_xml.get("nombre"))
                    campo.agregar_estacion(est)

            # Sensores Suelo
            suelo_xml = campo_xml.find("sensoresSuelo")
            if suelo_xml is not None:
                for s_xml in suelo_xml.findall("sensorS"):
                    sensor = SensorSuelo(s_xml.get("id"), s_xml.get("nombre"))
                    for f_xml in s_xml.findall("frecuencia"):
                        sensor.agregar_frecuencia(Frecuencia(f_xml.get("idEstacion"), f_xml.text.strip()))
                    # asignar a estaciones relacionadas
                    for est in campo.estaciones.recorrer():
                        if self._existe_frecuencia(sensor, est.get_id()) == 1:
                            est.agregar_sensor_suelo(sensor)

            # Sensores Cultivo
            cultivo_xml = campo_xml.find("sensoresCultivo")
            if cultivo_xml is not None:
                for t_xml in cultivo_xml.findall("sensorI"):
                    sensor = SensorCultivo(t_xml.get("id"), t_xml.get("nombre"))
                    for f_xml in t_xml.findall("frecuencia"):
                        sensor.agregar_frecuencia(Frecuencia(f_xml.get("idEstacion"), f_xml.text.strip()))
                    for est in campo.estaciones.recorrer():
                        if self._existe_frecuencia(sensor, est.get_id()) == 1:
```

Figura 1: Método Cargar Archivo

Fuente elaboración propia

Clases y estructuras de datos implementadas

Para evitar el uso de listas nativas, se diseñaron clases de listas enlazadas apoyadas en nodos. Se crearon entidades como Campo, Estación, Sensor de Suelo y Sensor de Cultivo. Cada clase contaba con atributos específicos (nombre, tipo, valores de frecuencia) y métodos para agregar y recorrer los elementos. La clase Nodo permitió enlazar los objetos, y la clase Frecuencia modeló los valores obtenidos de cada sensor.

Por ejemplo, la clase Campo incluía un atributo lista_estaciones que no era una lista nativa, sino una lista enlazada propia. El método agregar_estacion() insertaba nuevas estaciones en dicha lista. De esta forma, se fortaleció el entendimiento de estructuras dinámicas.

```python
# clases/nodo.py
class Nodo:
    def __init__(self, dato):
        self.dato = dato
        self.siguiente = None


# Definición de la clase ListaEnlazada,
class ListaEnlazada:
    def __init__(self):
        self.primero = None
        self._tamanio = 0

```

Figura 2: Metodo Cargar Archivo

Fuente elaboración propia

Procesamiento de matrices y patrones

Una parte fundamental fue la generación de matrices F[n,s] y F[n,t], las cuales representaban las frecuencias de los sensores en cada estación. A partir de ellas se crearon patrones binarios que permitieron identificar configuraciones equivalentes entre estaciones.

El método generar_patrones() dentro del gestor XML se encargaba de convertir los datos de frecuencias en cadenas binarias que resumían el estado de cada estación. Esto facilitó el análisis posterior, ya que las estaciones con el mismo patrón podían considerarse redundantes.

Figura 2: Método Generar Patrones

Fuente Elaboración Propia

Reducción de estaciones redundantes

El paso siguiente fue aplicar la reducción. El método reducir_estaciones() agrupó las estaciones que compartían patrones idénticos, generando grupos equivalentes. Esto permitió disminuir la cantidad de información sin perder los datos más importantes.

Finalmente, el método escribir_salida() tomó los grupos reducidos y generó un nuevo archivo XML más limpio y simplificado, que reflejaba el mismo conocimiento pero con menos redundancia.

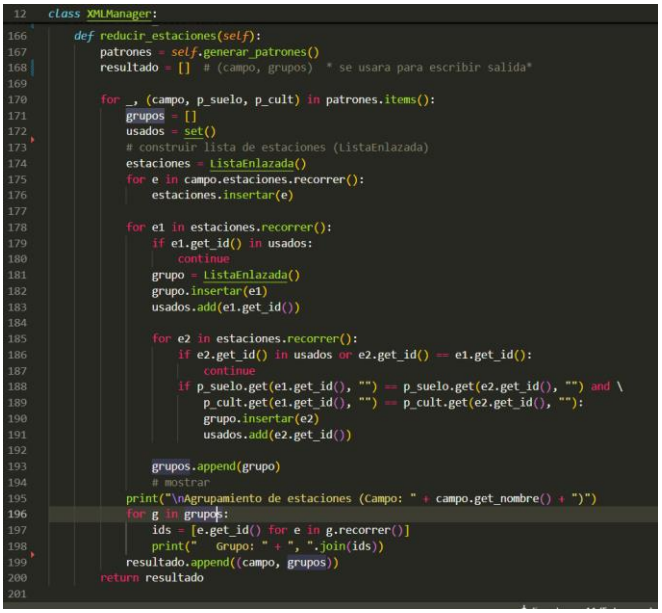

Figura 3:Método Reducir Estaciones

Fuente Elaboración Propia

Exportación y generación de gráficas con Graphviz

Para complementar la salida textual, se incluyó la opción de generar gráficas con la librería Graphviz. El módulo test_graphviz.py se encargaba de crear nodos y aristas que representaban visualmente los campos, estaciones y sensores. Gracias a esto, el usuario podía comprender de manera intuitiva la

organización de los datos.

El método principal permitía ejecutar estos procesos en orden, desde la lectura del archivo hasta la visualización final.

```python
# Método generar_grafica:
    def generar_grafica(self, archivo_salida="grafica"):
        if self.campos.esta_vacia():
            print("No hay campos cargados.")
            return

        dot = Digraph(comment="Campos Agricolas", format="png")
        dot.attr(rankdir="LR", size="8")

        for campo in self.campos.recorrer():
            campo_id = f"campo_{campo.get_id()}"
            dot.node(campo_id, f"Campo: {campo.get_nombre()}", shape="box", style="filled", color="lightblue")

            for est in campo.estaciones.recorrer():
                est_id = f"est_{est.get_id()}"
                dot.node(est_id, f"Estación: {est.get_nombre()}", shape="ellipse", style="filled", color="lightgreen")
                dot.edge(campo_id, est_id)

                # Sensores de suelo
                for sensor in est.sensores_suelo.recorrer():
                    s_id = f"suelo_{sensor.get_id()}"
                    dot.node(s_id, f"Suelo: {sensor.get_nombre()}", shape="diamond", style="filled", color="orange")
                    dot.edge(est_id, s_id)

                # Sensores de cultivo
                for sensor in est.sensores_cultivo.recorrer():
                    c_id = f"cult_{sensor.get_id()}"
                    dot.node(c_id, f"Cultivo: {sensor.get_nombre()}", shape="diamond", style="filled", color="pink")
                    dot.edge(est_id, c_id)

        output_path = dot.render(archivo_salida, cleanup=True)
        print(f"Gráfica generada: {output_path}")
```

Figura 4: Método Generar Grafica

Fuente elaboración Propia

Figura 5: Figura generada con Graphviz.

Fuente Elaboracion Propia

Conclusiones

Este proyecto permitió afianzar conocimientos en estructuras de datos, listas enlazadas, programación orientada a objetos y manejo de XML. La reducción de estaciones fue un reto interesante porque exigió pensar en términos de patrones y equivalencias. Asimismo, la integración de Graphviz mostró la importancia de la visualización en la comprensión de información compleja. El trabajo en conjunto fortaleció la lógica de programación y la capacidad de resolver problemas aplicados.

Por ultimo podemos concluir que las TDA son de las mejores opciones para el buen gestiona miento de memoria ya que ayudo con guardar y manipular de manera eficiente los datos

Referencias bibliográficas

- Graphviz Online. (s/f). Github.io. Recuperado el 4 de septiembre de 2023, de https://dreampuf.github.io/GraphvizOnline/xml.etree.ElementTree — La API XML de ElementTree. (s/f).
- Python documentation. Recuperado el 4 de septiembre de 2023, de https://docs.python.org/es/3/library/xml.etree.elementtree.html
- Graphviz. (s/f). Graphviz. Recuperado el 6 de septiembre de 2023, de https://graphviz.org/ Fernandez, R. (2017, junio 15).

- POO Programación Orientada a Objetos en Python. ▷ Cursos de 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 2023.

- Programación de 0 a Experto © Garantizados. https://unipython.com/programacion-orientada-objetos-python/

.