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.

PROYECTO 1: implementación de tipos de datos abstractos (TDA) y visualización de datos

202200002 – Julio Samuel Isaac Lima Donis

Resumen

El Centro de Investigaciones de la Facultad de Ingeniería se dedica a la compresión de señales de audio y se enfoca en los parámetros de Frecuencia y Amplitud. La Frecuencia representa la cantidad de ciclos por segundo (medidos en Hertz), mientras que la Amplitud se refiere a la intensidad del sonido. Estos parámetros son fundamentales para describir una señal de audio en función del tiempo. Para resolver este problema, se utiliza una metodología de agrupamiento.

La metodología de agrupamiento implica la creación de una matriz que contiene información de tiempo, amplitud y frecuencia para diferentes señales de audio. Luego, se transforma en una matriz de patrones de frecuencia y se agrupan las tuplas con patrones similares. Esta matriz de patrones de frecuencia indica cuándo y a qué amplitud se encuentran las frecuencias en la señal de audio. Este enfoque de agrupamiento se utiliza para la compresión de señales de audio en el proyecto. En temas de programación, se trabajó con Python haciendo uso el Paradigma de POO(programación orientada a objetos), para este proyecto también se implementaron estructuras de datos directamente creadas por el desarrollador evitando el uso de estructuras nativas de Python, para una visualización grafica de resultados se optó por usar Graphviz mediante tablas. Y por último para mayor comodidad los datos se recibieron mediante un archivo XML y los resultados igualmente pueden ser impresos en un archivo XML.

Palabras clave

1. Python
2. Graphviz
3. XML
4. Paradigma
5. POO

Abstract

The Research Center of the Faculty of Engineering is dedicated to the compression of audio signals and focuses on the parameters of Frequency and Amplitude. The Frequency represents the number of cycles per second (measured in Hertz), while the Amplitude refers to the intensity of the sound. These parameters are essential to describe an audio signal as a function of time.
To solve this problem, a clustering methodology is used.

The clustering methodology involves creating a matrix containing time, amplitude, and frequency information for different audio signals. It is then transformed into a matrix of frequency patterns, and tuples with similar patterns are grouped together. This frequency pattern matrix indicates when and at what amplitude frequencies are present in the audio signal. This clustering approach is used for audio signal compression in the project. In terms of programming, Python was used, employing the Object-Oriented Programming (OOP) paradigm. Additionally, data structures directly created by the developer were implemented, avoiding the use of native Python data structures. For graphical visualization of results, Graphviz was chosen using tables. Finally, for convenience, the data was received through an XML file, and the results can also be printed to an XML file.

Keywords

1. *Python*
2. *Graphviz*
3. *XML*
4. *Paradigm*
5. *POO*

Introducción

Se buscó resolver la necesidad del análisis sobre unos datos de audio, para plantear la solución a este problema se analizaron las necesidades exactas del cliente. El primer análisis se hizo sobre un archivo de texto con los principales detalles del proyecto, estos eran bastante claro lo que necesitaban, aun así, gracias a la implementación de un foro y la constante comunicación con el cliente se llevaron a cabo los siguientes sistemas para la solución: Lectura y Escritura de archivos XML, Visualización de datos en archivos de imagen y Creación de Estructuras de Datos no Nativas.

Desarrollo del tema

El Centro de Investigaciones de la Facultad de Ingeniería se dedica a la compresión de señales de audio y se enfoca en los parámetros de Frecuencia y Amplitud. La Frecuencia representa la cantidad de ciclos por segundo (medidos en Hertz), mientras que la Amplitud se refiere a la intensidad del sonido (medida en decibelios). Estos parámetros son fundamentales para describir una señal de audio en función del tiempo.

El problema de compresión de señales de audio se aborda como un problema combinatorio NP-Hard, que presenta desafíos significativos en términos de tiempo y recursos. Para resolver este problema, se utiliza una metodología de agrupamiento.

La metodología de agrupamiento implica la creación de una matriz que contiene información de tiempo, amplitud y frecuencia para diferentes señales de audio. Luego, se transforma en una matriz de patrones de frecuencia y se agrupan las tuplas con patrones similares. Esta matriz de patrones de frecuencia indica cuándo y a qué amplitud se encuentran las frecuencias en la señal de audio. Este enfoque de agrupamiento se utiliza para la compresión de señales de audio en el proyecto

Funcionalidades importantes para el Proyecto:

a. XML

XML, que significa "Extensible Markup Language" en inglés, es un lenguaje de marcado utilizado para representar datos en un formato legible. XML se utiliza ampliamente en la industria de la tecnología de la información para el intercambio de datos estructurados entre sistemas heterogéneos. Sus principales características:

Sintaxis de XML: XML utiliza etiquetas para definir elementos y atributos para describir características de esos elementos. Una etiqueta XML consiste en un nombre entre ángulos (<>) que define el elemento y puede contener contenido y atributos. Por ejemplo:

Documentos XML bien formados: Un documento XML debe cumplir con ciertas reglas para ser considerado "bien formado". Debe tener un elemento raíz que contenga todos los demás elementos y las etiquetas deben estar correctamente anidadas y cerradas. Además, los atributos deben estar entre comillas. Un documento XML bien formado no contiene errores sintácticos.

Documentos XML válidos: La validación en XML se realiza mediante un esquema o un documento de tipo de documento (DTD) que define la estructura y las restricciones de los datos en el documento. Un documento XML válido es un documento bien formado que cumple con las reglas especificadas en su esquema o DTD.

Espacio de nombres (Namespaces): Los espacios de nombres permiten evitar conflictos de nombres al definir elementos y atributos en documentos XML. Esto es especialmente útil cuando se combinan datos de múltiples fuentes. Los espacios de nombres se definen mediante la declaración xmlns en los elementos raíz o en elementos específicos.

Procesamiento de XML: Para procesar datos XML, se utilizan lenguajes y tecnologías como XPath y XSLT para buscar y transformar datos, DOM (Modelo de Objeto de Documento) y SAX (Simple API for XML) para manipular documentos XML programáticamente.

Codificación de caracteres: XML admite una amplia gama de codificaciones de caracteres, lo que permite representar texto en varios idiomas y caracteres especiales. La codificación de caracteres se especifica en la declaración XML:

b. Estructura de Datos

Una estructura de datos muy común y útil en programación es la "lista enlazada". Una lista enlazada es una colección de elementos donde cada elemento se almacena en un "nodo", y cada nodo contiene dos partes6 principales: el valor del elemento y una referencia (o enlace) al siguiente nodo en la lista. La última referencia de la lista suele ser nula, indicando el final de la lista.

Hay varios tipos de listas enlazadas, pero aquí se presenta una descripción básica de una "lista enlazada simple":

Nodo: Cada nodo en una lista enlazada consta de dos partes:
Valor: Almacena el dato que queremos guardar en la lista enlazada.
Referencia: Un enlace al siguiente nodo en la lista. Esta referencia es nula si el nodo es el último en la lista.

Ventajas y desventajas:

Ventajas:

Flexibilidad en tamaño: No es necesario especificar el tamaño de la lista enlazada de antemano.

Eficiente para agregar o eliminar elementos en cualquier posición.

Desventajas:

El acceso a elementos individuales puede ser menos eficiente que en estructuras de datos contiguas como los arrays.

Requiere más memoria debido a las referencias adicionales en cada nodo.

c. Graphviz

Graphviz es una suite de herramientas de código abierto desarrollada por AT&T Research que se utiliza para crear visualizaciones de grafos y diagramas. Está diseñado para representar de manera efectiva estructuras de datos complejas y relaciones en una variedad de aplicaciones, como visualización de redes, diagramas de flujo, representación de árboles jerárquicos y más.

Las herramientas principales de Graphviz que usaremos son:

Dot: Dot es el motor principal de Graphviz. Toma una descripción textual de un grafo en el lenguaje DOT (Graph Description Language) y genera visualizaciones de ese grafo en varios formatos de salida, como PNG, PDF, SVG y otros. DOT es un lenguaje de descripción de gráficos simple que permite definir nodos, aristas, atributos y más.

Funcionalidad

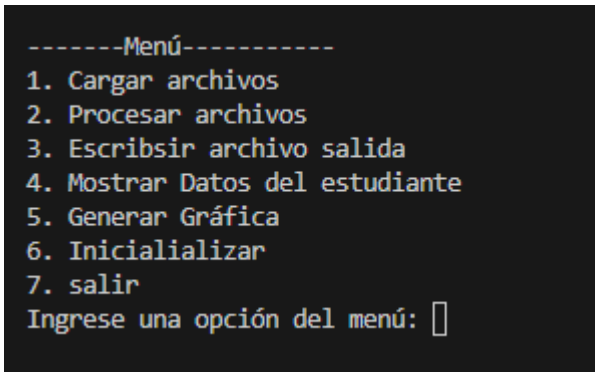

Figura 1. Menu de Aplicación.

Fuente: elaboración propia.

Cargar archivos: Cuando el usuario selecciona la opción "1" en el menú, se llama a la función cargador(). Esta función permite al usuario seleccionar un archivo XML y cargarlo en el programa para su procesamiento.

Procesar archivos: Si el usuario elige la opción "2", Esta función procesa los datos del archivo XML cargado previamente y crea estructuras internas para su posterior análisis.

Escribir archivo salida: Cuando el usuario selecciona la opción "3", se llama a la función generar_xml_salida(). Esta función permite al usuario especificar una ruta de archivo donde se generará un nuevo archivo XML con los datos procesados.

Mostrar Datos del estudiante: Si el usuario selecciona la opción "4", se llama a la función mostrar_Documentarion(), que simplemente muestra información sobre el estudiante que escribió el código.

Generar Gráfica: Cuando el usuario elige la opción "5", Esta función permite al usuario generar diferentes tipos de gráficos a partir de los datos procesados en función de sus preferencias.

Inicialializar: Si el usuario selecciona la opción "6", se llama a la función Inicialializar(). Esta función reinicia todas las estructuras de datos utilizadas en el programa, lo que permite al usuario comenzar de nuevo con nuevos archivos sin necesidad de reiniciar el programa.

Salir: Si el usuario selecciona la opción "7", el programa muestra un mensaje de despedida y se cierra.

El desarrollo del proyecto inicio después de haber entendido al cliente, el sistema de carpetas de uso de la siguiente manera:

a. clases

Contiene las clases **Cargador**, **Dato**, y **Senal**. Estas clases están relacionadas con la estructuración y manipulación de datos relacionados con señales y patrones.

1. **Archivo "Cargador.py"**

   Contiene la definición de la clase Cargador, que parece estar relacionada con la carga y manipulación de datos de archivos.

2. Archivo "Dato.py"

   Contiene la definición de la clase Dato, que se utiliza para representar datos relacionados con señales o patrones de sonido.

3. Archivo "Senal.py"

   Contiene la definición de la clase Senal, que esta relacionada con la representación de datos de señales, incluyendo listas de datos y patrones.

b. Listas

Contiene las clases Lista_dato, Lista_senales, lista_patrones, lista_grupos, y otras clases relacionadas

con el almacenamiento y procesamiento de datos en listas enlazadas.

1. Archivo "Lista_dato.py"

   Contiene la definición de la clase Lista_dato, que está relacionada con la creación y gestión de listas enlazadas que almacenan datos.

- __init__(self): Este es el método constructor de la clase Lista_dato. Se utiliza para inicializar una instancia de la clase. Crea un objeto de tipo Lista_dato con dos atributos iniciales: self.primero, que se establece en None, y self.contador_celdas, que se establece en 0. self.primero se usa para rastrear el primer elemento en la lista enlazada, y self.contador_celdas se usa para llevar la cuenta del número de elementos en la lista.
- insertar_dato(self, dato): Esta función se utiliza para insertar un nuevo dato en la lista. Si la lista está vacía (es decir, self.primero es None), crea un nuevo nodo con el dato proporcionado y lo asigna como el primer elemento de la lista. Si la lista no está vacía, recorre la lista hasta encontrar el último nodo y agrega el nuevo nodo al final.
- insertar_dato_ordenado(self, dato): Esta función inserta un dato en la lista en orden ascendente según el valor de dato.t y dato.A. Si la lista está vacía, el nuevo dato se convierte en el primer elemento. Si el nuevo dato debe insertarse en medio de la lista, se recorre la lista para encontrar la posición correcta y se inserta allí.
- recorrer_e_imprimir_lista(self): Esta función recorre la lista y muestra en la consola los valores de dato.t, dato.A, y dato.num de cada nodo. Esto se hace para imprimir los datos almacenados en la lista.
- devolver_patrones_por_nivel(self, lista_patrones_nivel): Esta función recorre la lista y agrupa los elementos con el mismo valor de dato.t en un patrón, luego inserta estos patrones en otra lista (lista_patrones_nivel) y la devuelve. Se utiliza para organizar los datos en la lista original en patrones según el nivel.
- generar_grafica(self, nombre_carcel, niveles, celdas_por_nivel): Esta función genera una representación gráfica de la lista utilizando Graphviz. Crea un archivo DOT que describe la estructura del grafo y utiliza Graphviz para generar una imagen PNG del grafo. Se proporcionan varios argumentos que se utilizan para personalizar la apariencia del grafo.
- devolver_cadena_del_grupo(self, grupo): Esta función toma un grupo de dígitos

(presumiblemente representando niveles) como entrada y devuelve una cadena que contiene los números asociados a esos niveles en la lista.

- **eliminar**(self): Esta función se utiliza para eliminar todos los elementos de la lista. Recorre la lista y elimina cada nodo uno por uno hasta que la lista esté vacía.

2. Archivo "Lista_senales.py"

Contiene la definición de la clase Lista_senales, que está relacionada con la creación y gestión de listas enlazadas que almacenan señales.

- **__init__(self):** Este es el método constructor de la clase Lista_senales. Se utiliza para inicializar una instancia de la clase. Crea un objeto de tipo Lista_senales con dos atributos iniciales: self.primero, que se establece en None, y self.contador_senales, que se establece en 0. self.primero se usa para rastrear el primer elemento en la lista enlazada, y self.contador_senales se usa para llevar la cuenta del número de elementos en la lista.

- **Incertar_dato(self, senal):** Esta función se utiliza para insertar una nueva señal en la lista. Si la lista está vacía (es decir, self.primero es None), crea un nuevo nodo con la señal proporcionada y lo asigna como el primer elemento de la lista. Si la lista no está vacía, recorre la lista hasta encontrar el último nodo y agrega el nuevo nodo al final.

- **recorrer_e_imprimir_lista(self):** Esta función recorre la lista y muestra en la consola los atributos de cada nodo, incluyendo el nombre de la señal (senal.nombre), senal.t y senal.A. Además, llama a las funciones recorrer_e_imprimir_lista de las listas de datos y patrones de cada señal para mostrar sus contenidos.

- grafica_mi_lista_original(self, nombre_senal): Esta función busca una señal por su nombre en la lista y, si la encuentra, llama a la función generar_grafica de la lista de datos de la señal para crear una representación gráfica de esa lista.

- **grafica_mi_lista_de_patrones(self, nombre_senal):** Similar a la función anterior, esta busca una señal por su nombre en la lista y, si la encuentra, llama a la función generar_grafica de la lista de patrones de datos de la señal para crear una representación gráfica de esa lista.

- grafica_mi_lista_Reducida(self, nombre_senal): Busca una señal por su nombre en la lista y, si la encuentra, llama a la función generar_graficaBIN de la lista de grupos de la señal para crear una representación gráfica de esa lista.

- calcular_los_patrones(self, nombre_senal): Busca una señal por su nombre en la lista y, si la encuentra, realiza un cálculo de patrones en función de los datos y patrones almacenados en la señal. Luego, crea una representación gráfica de los grupos de datos y patrones resultantes.

- **generar_xml_salida(self, ruta)**: Crea un archivo XML que contiene información sobre las señales almacenadas en la lista, sus nombres, tiempos y grupos de datos. La función recorre la lista y utiliza la biblioteca ElementTree para generar el archivo XML en la ruta especificada.

- **xml_arreglado(self, element, indent=' '):** Esta función es utilizada para dar formato al XML generado. Añade espacios en blanco y saltos de línea para que el XML sea más legible.

- **eliminar(self):** Esta función se utiliza para eliminar todos los elementos de la lista. Recorre la lista y elimina cada nodo uno por uno hasta que la lista esté vacía.

3. Archivo "lista_patrones.py"

Contiene la definición de la clase lista_patrones, que se utiliza para crear y gestionar listas enlazadas que almacenan patrones relacionados con los datos de señales.

- **__init__(self):** Este es el método constructor de la clase lista_patrones. Se utiliza para inicializar una instancia de la clase. Crea un objeto de tipo lista_patrones con dos atributos iniciales: self.primero, que se establece en None, y self.contador_patrones, que se establece en 0. self.primero se usa para rastrear el primer elemento en la lista enlazada de patrones, y self.contador_patrones se usa para llevar la cuenta del número de elementos en la lista.

- **insertar_dato(self, patron):** Esta función se utiliza para insertar un nuevo patrón en la lista. Si la lista está vacía (es decir, self.primero es None), crea un nuevo nodo con el patrón proporcionado y lo asigna como el primer elemento de la lista. Si la lista no está vacía, recorre la lista hasta encontrar el último nodo y agrega el nuevo nodo al final.

- **recorrer_e_imprimir_lista(self):** Esta función recorre la lista de patrones y muestra en la consola los atributos de cada nodo, incluyendo patron.t y patron.cadena_patron. Esto se hace para imprimir los patrones almacenados en la lista.

- **eliminar(self, nivel):** Esta función elimina un patrón específico de la lista de patrones en función de su nivel (nivel). Recorre la lista y busca el nodo con el nivel coincidente. Si encuentra un nodo con el nivel deseado, lo elimina de la lista. Si no encuentra el nodo, no se realiza ninguna acción.

- **encontrar_coincidencias(self):** Esta función busca patrones en la lista que tengan la misma cadena de patrones y los agrupa. Luego, elimina los patrones duplicados de la lista. Funciona recorriendo la lista principal de patrones y comparando la cadena de patrones de cada patrón con la cadena de patrones del primer patrón en la lista. Si encuentra coincidencias, agrega los niveles correspondientes a un resultado temporal (temp_niveles) y luego elimina los patrones duplicados de la lista principal. Finalmente, devuelve un resultado que contiene los niveles agrupados en cadenas separadas por "--".

- **eliminar(self):** Esta función se utiliza para eliminar todos los elementos de la lista. Recorre la lista y elimina cada nodo uno por uno hasta que la lista esté vacía. Tienes dos funciones eliminar con el mismo nombre en esta clase, lo cual podría causar problemas. Deberías considerar eliminar uno de ellos para evitar confusiones.

4. Archivo "lista_grupos.py"

Contiene la definición de la clase lista_grupos, que está relacionada con la creación y gestión de listas enlazadas que almacenan grupos de datos.

- **__init__(self):** Este es el método constructor de la clase lista_grupos. Se utiliza para inicializar una instancia de la clase. Crea un objeto de tipo lista_grupos con dos atributos iniciales: self.primero, que se establece en None, y self.contador_grupos, que se establece en 0. self.primero se usa para rastrear el primer elemento en la lista enlazada de grupos, y self.contador_grupos se usa para llevar la cuenta del número de elementos en la lista.

- **insertar_dato(self, grupo)**: Esta función se utiliza para insertar un nuevo grupo en la lista. Si la lista está vacía (es decir, self.primero es None), crea un nuevo nodo con el grupo proporcionado y lo asigna como el primer elemento de la lista. Si la lista no está vacía, recorre la lista hasta encontrar el último nodo y agrega el nuevo nodo al final.

- **recorrer_e_imprimir_lista(self):** Esta función recorre la lista de grupos y muestra en la consola los atributos de cada nodo, incluyendo grupo.el_grupo y grupo.cadena_grupo. Esto se hace para imprimir los grupos almacenados en la lista.

- **generar_graficaBIN(self, nombre_carcel, niveles, celdas_por_nivel):** Esta función genera una representación gráfica de la lista de grupos utilizando Graphviz. Crea un archivo DOT que describe la estructura del grafo y utiliza Graphviz para generar una imagen PNG del grafo. Se proporcionan varios argumentos que se utilizan para personalizar la apariencia del grafo.

- **eliminar(self):** Esta función se utiliza para eliminar todos los elementos de la lista. Recorre la lista y elimina cada nodo uno por uno hasta que la lista esté vacía.

5. Archivo "Lista_Cargador.py"

Contiene la definición de la clase Cargador_Dao, , que está relacionada con la creación y gestión de listas enlazadas que almacenan cargadores.

- **__init__(self, root):** Este es el método constructor de la clase Cargador_Dao. Se utiliza para inicializar una instancia de la clase. Toma un argumento root, que parece no utilizarse en esta función. En el constructor, se crea un objeto de tipo Cargador_Dao con un atributo inicial self.primero que se establece en None.

- **crear_producto(self, c):** Esta función se utiliza para crear y agregar un nuevo objeto de tipo Nodo_c a la lista enlazada. Toma un argumento c, que es el objeto que se desea agregar a la lista. Si la lista está vacía (es decir, self.primero es None), crea un nuevo nodo con el objeto c proporcionado y lo asigna como el primer elemento de la lista. Si la lista no está vacía, recorre la lista hasta encontrar el último nodo y agrega el nuevo nodo al final. También imprime "Se agrego un nuevo archivo" como mensaje informativo.

- **devolver(self):** Esta función se utiliza para devolver el primer nodo de la lista enlazada. Retorna self.primero, que es una referencia al primer nodo de la lista.

- **eliminar(self):** Esta función se utiliza para eliminar todos los elementos de la lista enlazada. Recorre la lista y elimina cada nodo uno por uno hasta que la lista esté vacía.

6. Archivo "nodo_patron.py"

Contiene la definición de la clase nodo_patron, que se utiliza para representar nodos en listas enlazadas que almacenan patrones.

7. Archivo "Patron.py"

Contiene la definición de la clase Patron, que está relacionada con la representación de datos de patrones.

8. Archivo "grupo.py"

   Contiene la definición de la clase grupo, que está relacionada con la representación de datos de grupos.

c. nodos

1. Archivo "nodo_c.py"

   Contiene la definición de la clase Nodo_c, que esta relacionada con la representación de nodos en listas enlazadas.

2. Archivo "Nodo_dato.py"

   Contiene la definición de la clase Nodo_dato, que esta relacionada con la representación de nodos en listas enlazadas.

3. Archivo "nodo_grupo.py"

   Contiene la definición de la clase nodo_grupo, que esta relacionada con la representación de nodos en listas enlazadas.

4. Archivo "Nodo_senal.py"

   Contiene la definición de la clase Nodo_senal, que esta relacionada con la representación de nodos en listas enlazadas.

d. Reportes

Acá se almacenan las Imágenes.

e. Lectura_xml

Acá se encuentra un lugar opcional para imprimir el xml de respuesta.

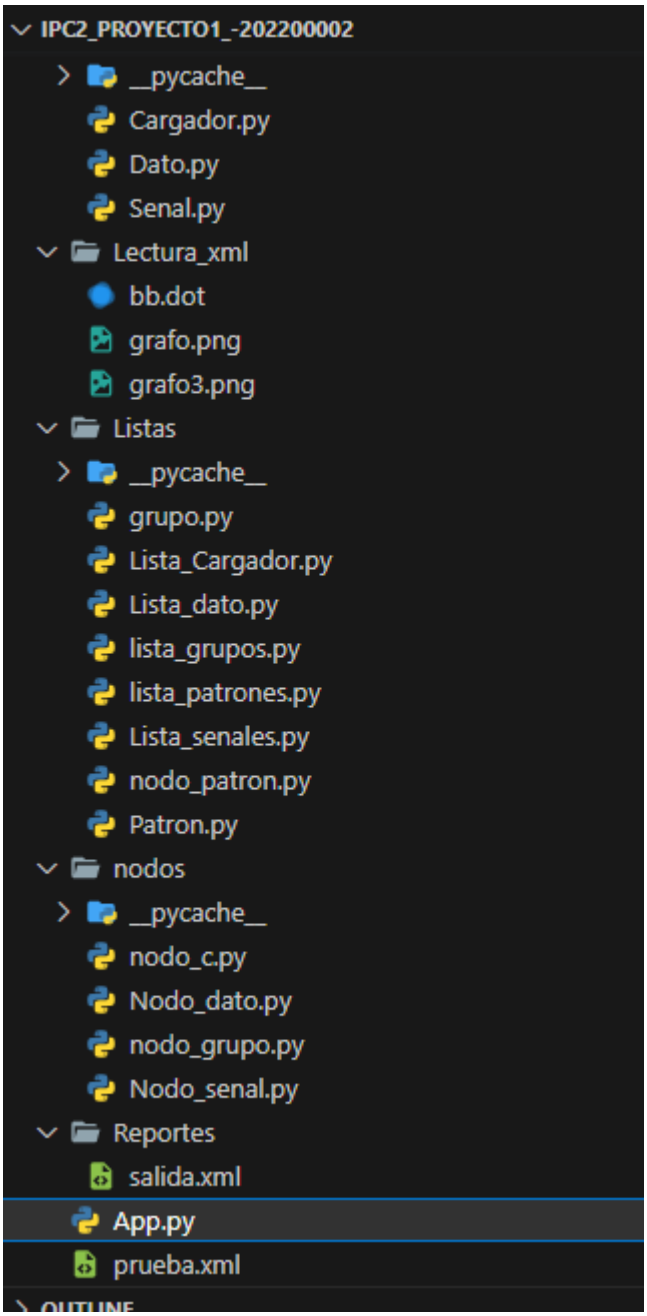

Figura 2. Archivos.

Fuente: elaboración propia

Conclusiones

Las señales de lograran ser leídas y procesadas correctamente, al llevar un flujo correcto al momento de trabajar y montar el proyecto. La POO fue de gran ayuda al ayudarnos a la reducción de código gracias a la gran reutilización que le da al código. Las estructuras de datos creadas por nosotros mismo, aunque no facilitaron el trabajo ayudaron a entrar en conciencia sobre su importancia y su funcionalidad, que, aunque en el futuro no lleguemos a usarlas de esta manera, los entendimientos más expandidos del uso de datos en Python nos harán encontrar soluciones más eficientes. La lectura y el procesamiento de los archivos xml, nos dio un aprendizaje considerable sobre el manejo de archivos en una aplicación tanto como su lectura como su edición. Y finalmente Graphviz fue imprescindible para la creación de imágenes.

Referencias bibliográficas

Máximo 5 referencias en orden alfabético.

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

Anexos:

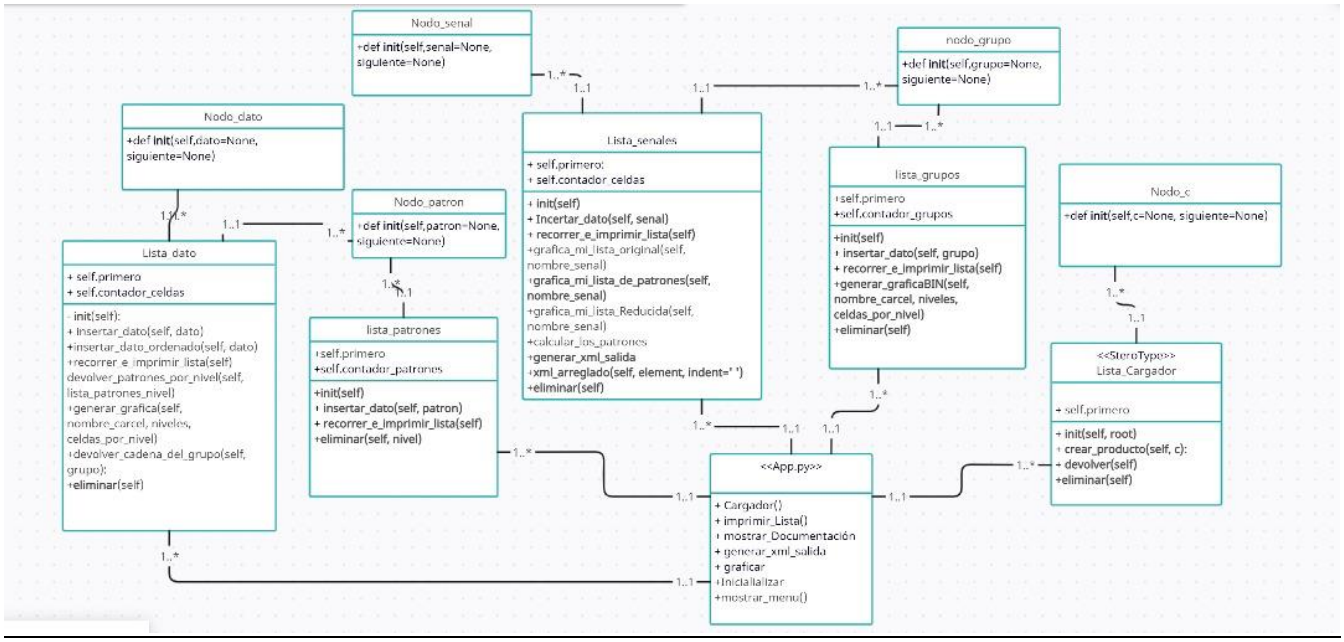

Figura 3. Diagrama de Clases.

Fuente: elaboración propia