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

202203069 – Bruce Carbonell Castillo Cifuentes

Resumen

El objetivo general de este proyecto es desarrollar una solución integral que implemente tipos de datos abstractos (TDA) y visualización de datos utilizando Graphviz, todo ello bajo el concepto de programación orientada a objetos (POO). Los objetivos específicos incluyen la implementación de POO en Python, el uso de estructuras de programación secuenciales, cíclicas y condicionales, la visualización de TDA's a través de Graphviz y el manejo de archivos XML para la lógica y el comportamiento de la solución.

El proyecto se centra en un experimento de compresión de señales de audio realizado por el Centro de Investigaciones de la Facultad de Ingeniería. Este experimento se basa en dos parámetros fundamentales de las ondas de sonido: la frecuencia y la amplitud. La frecuencia se mide en Hertz (Hz) y representa la cantidad de ciclos por segundo, mientras que la amplitud se mide en decibelios (dB) y refleja la intensidad del sonido. Se aborda el problema como un desafío NP-Hard y se emplea una metodología de agrupamiento para resolverlo.

Palabras clave

Señales de audio, programación orientada a objetos, Matrices de frecuencia, Algoritmo de agrupación, Archivos XML

Abstract

The overall objective of this project is to develop a comprehensive solution that implements abstract data types (ADT) and data visualization using Graphviz, all under the concept of object-oriented programming (OOP). Specific objectives include the implementation of OOP in Python, the use of sequential, cyclic and conditional programming structures, the visualization of ADT's through Graphviz and the handling of XML files for the logic and behavior of the solution.

The project focuses on an audio signal compression experiment conducted by the Research Center of the School of Engineering. This experiment is based on two fundamental parameters of sound waves: frequency and amplitude. Frequency is measured in Hertz (Hz) and represents the number of cycles per second, while amplitude is measured in decibels (dB) and reflects the intensity of the sound. The problem is approached as an NP-Hard challenge and a clustering methodology is employed to solve it.

Keywords

Audio signals, object oriented programming, Frequency arrays, Grouping algorithm, XML files.

Introducción

El programa desarrollado se centra en el procesamiento y análisis de señales de audio, utilizando una sólida base de programación orientada a objetos (POO) y estructuras de datos avanzadas. Su principal objetivo es la compresión de señales de audio, centrándose en los parámetros fundamentales de frecuencia y amplitud. La frecuencia, medida en Hertz (Hz), representa la cantidad de ciclos por segundo, mientras que la amplitud, medida en decibelios (dB), describe la intensidad del sonido.

Este programa aborda un desafío NP-Hard utilizando una metodología de agrupamiento que transforma matrices de tiempo, amplitud y frecuencia en patrones de frecuencia para su posterior reducción. La implementación de archivos XML como entrada y la visualización de datos con Graphviz son aspectos destacados de esta solución.

A lo largo de este documento, se describirán en detalle los objetivos generales y específicos del programa, ejemplos de su funcionamiento y la estructura de archivos de entrada y salida. Esta herramienta representa una contribución valiosa al campo de procesamiento de señales de audio y programación orientada a objetos.

Desarrollo del tema

Durante la creación del programa, utilizamos una estructura llamada "listas enlazadas" para organizar y almacenar las señales y sus datos. Esto nos permitió diseñar el programa de manera eficiente. Creamos objetos para representar cada señal, y los dividimos en dos partes: una para la información principal y otra para datos específicos.

Este enfoque nos ayudó a mantener un programa ordenado y fácil de entender. Cada objeto señal contenía toda la información necesaria, lo que hizo que nuestro código fuera más claro y manejable.

Para lograr una implementación exitosa, también incorporamos la biblioteca Graphviz para la visualización de datos. Esta librería es esencial para la creación de gráficas que representen las señales de audio y sus matrices reducidas de manera visual y comprensible.

La interfaz del programa se desarrolló con un menú que permite al usuario elegir las acciones que desea realizar. Estas acciones incluyen cargar archivos de entrada, procesar los datos, escribir archivos de salida, mostrar datos del estudiante, generar gráficas y reiniciar el sistema.

El proceso de lectura de datos implica la importación de archivos XML que contienen la información de las señales de audio. Luego, estos datos se almacenan en las listas enlazadas mencionadas anteriormente para su procesamiento.

La fase de procesamiento de datos se encarga de analizar y agrupar las señales de audio según sus patrones de frecuencia. Esto se logra mediante el uso de matrices y algoritmos diseñados específicamente.

Finalmente, la creación de gráficas se lleva a cabo utilizando la herramienta Graphviz, lo que proporciona una representación visual de las señales de audio y sus matrices reducidas.

En las secciones siguientes, exploraremos cada uno de estos procesos con más detalle y discutiremos cómo contribuyeron al éxito de nuestro programa. Las cuales dichas pueden ser dividido en:

a. Lectura de datos
b. Almacenamiento
c. Proceso de datos
d. Graficas

Como se mencionaba anterior mente los objetos Señales almacena todo lo relacionado a la señal que se manipulara como es el caso de la matriz de frecuencia y la matriz, es así que mediante la **figura 1** podemos tener una mejor visión del objeto Señal

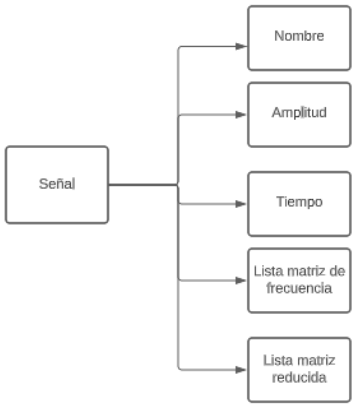

Figura 1. Distribucion de señal

Fuente: elaboración propia

La Figura 1 nos muestra que una señal está compuesta por varios elementos clave, como su nombre, amplitud y tiempo. Estos datos son esenciales para identificar y validar cada señal de manera única. El nombre de la señal la distingue de las demás, mientras que la amplitud y el tiempo son atributos importantes que determinan su comportamiento.

Además, el objeto "Señal" incluye dos listas importantes: la "Lista Matriz Frecuencia" y la "Lista Matriz Reducida". La primera lista almacena la información cruda de la señal, como su matriz de frecuencia completa. La segunda lista, en cambio, se utiliza para agrupar los tiempos que comparten la misma frecuencia. Esto es crucial para la etapa de procesamiento, donde se identifican y combinan los datos similares.

En resumen, la figura y la descripción nos permiten comprender cómo se organiza y almacena la información relacionada con las señales en el programa, lo que es esencial para su correcto funcionamiento y procesamiento posterior.

En el proceso de almacenamiento de señales, se optó por utilizar una estructura de datos fundamental conocida como listas enlazadas. Esta elección se basó en la eficiencia y versatilidad que ofrecen las listas enlazadas para organizar y gestionar datos de manera dinámica. La Figura 2 ilustra de manera conceptual cómo se lleva a cabo este almacenamiento mediante listas enlazadas

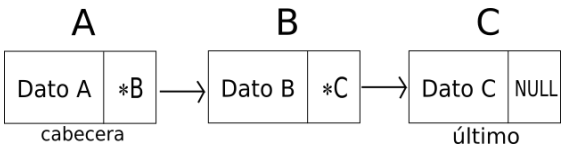

Figura 2. Lista enlazada de señales

(A, B, C representan señales respectivas)

En la Figura 2, las letras A, B y C simbolizan las señales individuales que se almacenan en el programa. Cada señal se representa como un nodo en la lista enlazada, donde cada nodo contiene tanto los datos específicos de la señal (como nombre, amplitud y tiempo) como referencias a las señales siguientes en la lista.

Este enfoque de listas enlazadas proporciona varias ventajas. En primer lugar, permite una gestión dinámica de las señales, lo que significa que podemos agregar o eliminar señales fácilmente a medida que sea necesario, sin preocuparnos por asignaciones de memoria estáticas. Además, al utilizar referencias a nodos siguientes, podemos acceder eficientemente a todas las señales almacenadas en la lista.

Es así que mediante el uso de listas enlazadas para el almacenamiento de señales es una elección inteligente, ya que proporciona flexibilidad, eficiencia y escalabilidad para gestionar las señales en el programa de manera efectiva. Esta estructura de datos es fundamental para garantizar un funcionamiento óptimo del sistema de procesamiento de señales

Lectura de datos

En el proceso de desarrollo de nuestro programa, una parte crucial se relaciona con la lectura de datos de entrada. Para lograrlo, implementamos una estructura de clases y objetos que nos permiten manejar eficientemente la información correspondiente a las señales. A través de esta sección, exploraremos en detalle cómo llevamos a cabo la lectura de datos y cómo organizamos la información para su procesamiento posterior.

Cada señal se define mediante la lectura de un archivo XML de entrada. Este archivo contiene detalles como el nombre de la señal, la dimensión de tiempo, la dimensión de amplitud y los datos respectivos. Para representar y almacenar estos datos de manera efectiva, creamos objetos de la clase Dato, definida en el archivo Dato.py, con la siguiente estructura:

```
class Dato:
    def __init__(self, grupo,  tiempo, amplitud, dato):
        self._grupo =grupo
        self._tiempo =tiempo
        self._amplitud = amplitud
        self._dato = dato
```

Figura 3. Dato.py

Fuente: elaboración propia

Cada objeto Dato almacena información sobre el grupo al que pertenece, el tiempo, la amplitud y el valor específico del dato. Estos objetos se utilizan para construir las señales.

En nuestro programa, la lectura de datos desde archivos XML es una parte crítica del proceso. La información vital de cada señal, como su nombre, dimensiones de tiempo y amplitud, y los valores de los datos, se extrae cuidadosamente de estos archivos para su posterior procesamiento.

Para lograr esta tarea, implementamos una serie de funciones y estructuras que nos permiten leer y comprender eficazmente los datos contenidos en los archivos XML. El módulo principal de lectura de datos se encuentra en el archivo **Logic.py**. Este módulo utiliza la biblioteca **xml.etree.ElementTree** para analizar el archivo XML y extraer los elementos relevantes.

El proceso de lectura de datos comienza con la identificación de la etiqueta **<senal>** en el archivo XML, que representa una señal. A continuación, se leen y almacenan los atributos de la señal, como su nombre, dimensiones de tiempo y amplitud. Estos atributos se utilizan para crear objetos de señal que posteriormente se agregan a una lista enlazada de señales.

Dentro de la etiqueta **<senal>**, se buscan las etiquetas **<dato>**, que representan los valores de datos individuales de la señal. Cada **<dato>** contiene información sobre el tiempo, la amplitud y el valor del dato. Esta información se utiliza para crear objetos de datos **(Dato)** que se almacenan en una lista enlazada de datos específica para cada señal.

Es así que el proceso de lectura de datos implica identificar y extraer información clave de los archivos XML, como el nombre de la señal, las dimensiones

de tiempo y amplitud, y los valores de los datos. Esta información se organiza en objetos de señal y datos, lo que permite una gestión efectiva de los datos de entrada para su posterior procesamiento en nuestro programa.

Almacenamiento

Una vez que hemos leído y procesado los datos desde los archivos XML, es esencial un adecuado almacenamiento para su posterior manipulación y procesamiento. En nuestro programa, hemos optado por utilizar una combinación de objetos `Señal`, objetos `Dato`, y listas enlazadas para lograr un almacenamiento organizado y eficiente de la información.

Cada objeto `Señal` almacena todos los detalles relacionados con una señal particular, incluyendo su nombre, dimensiones de tiempo y amplitud, y las listas enlazadas que contienen los datos crudos y la matriz reducida de la señal. Este enfoque orientado a objetos facilita la gestión de señales, ya que toda la información relacionada se encuentra encapsulada en una sola entidad.

Para almacenar los datos crudos de una señal, hemos implementado la clase `Dato`. Cada objeto `Dato` almacena información sobre el tiempo, la amplitud y el valor del dato. Esto nos permite tener una representación estructurada de cada punto de datos en la señal. Además, cada objeto `Dato` se inserta en una lista enlazada específica para esa señal, lo que nos permite tener un registro organizado de todos los datos de la señal.

Las listas enlazadas desempeñan un papel fundamental en nuestro sistema de almacenamiento. Hemos creado una clase **Lista_enlazada** que se utiliza tanto para la lista de datos crudos como para la lista de la matriz reducida de una señal. Esto nos proporciona una estructura de datos dinámica que puede crecer según sea necesario y mantener los datos organizados en un orden específico, para organizar los datos de manera dinámica y estructurada, implementamos listas enlazadas. En el archivo ListaEnlazada.py, definimos una clase lista_enlazada que nos permite insertar y recorrer estos objetos Dato de manera eficiente. Aquí, un extracto de la implementación:

```
class lista_enlazada:
    def __init__(self):
        self.primero = None

    def insertar(self, Dato):
        # Implementación para insertar un nuevo dato en la lis

    def recorrer(self):
        # Implementación para recorrer y mostrar los datos de
```

Figura 4. Lista enlazada

Fuente: elaboración propia

Es así que el almacenamiento de datos en nuestro programa se basa en objetos `Señal` y `Dato`, junto con listas enlazadas. Cada señal tiene su propia lista enlazada de datos crudos y matriz reducida, lo que permite un acceso y procesamiento eficientes de los datos en cada etapa del programa. Este diseño modular y estructurado ha demostrado ser esencial para la eficacia y mantenibilidad de nuestro sistema, el proceso de almacenamiento lo podemos visualizar en la **figura 5**

Figura 5. Proceso de almacenamiento

Fuente: elaboración propia

Cada señal se representa como una lista enlazada de datos, lo que nos permite mantener una estructura organizada de toda la información. Esto resulta fundamental para el procesamiento posterior de las señales,

Es así que podemos decir que la "Lectura de Datos" y el “Almacenamiento” se basa en la lectura de los archivos XML y así mismo en la creación de objetos Dato para representar los datos de las señales y su organización en listas enlazadas. Este proceso nos proporciona una base sólida para el manejo eficiente y efectivo de los datos de entrada en nuestro programa.

Proceso de datos

La etapa de procesamiento de datos en nuestro programa es crucial para la generación de matrices y patrones que se utilizarán en análisis posteriores. Hemos implementado diversos métodos para llevar a cabo esta tarea, y a continuación, se describen los procesos clave.

- **Generación de Matriz de Patrones**
  La matriz de patrones es una representación binaria de las señales, donde cada fila representa un tiempo específico y cada columna un dato. Se utiliza 1 para representar la presencia de datos y 0 para la ausencia. Esto se logra mediante la comparación de los datos en una señal y su correspondencia en la matriz de patrones.
  El método **generarMatrizPatrones** recorre la lista enlazada de datos y compara los tiempos de los datos para determinar si pertenecen a la misma fila de la matriz de patrones. Los valores binarios se generan en función de si el dato es diferente de cero. Esto crea una representación de matriz de patrones para todas las señales.

- **Generación de Matriz Reducida**
  La matriz reducida es una representación condensada de las señales que agrupa los datos con el mismo tiempo y amplitud. Esto reduce la complejidad de los datos y acelera las operaciones posteriores.
  El método **generarMatrizReducida** recorre la lista enlazada de datos crudos para una señal y compara los tiempos de los datos. Si se encuentran datos con el mismo tiempo, se realiza una suma y se agrega a la matriz reducida. Esto se realiza para todos los datos de la señal, lo que resulta en una representación condensada y eficiente.

- **Conteo de Tiempos**
  El método **contarTiempos** cuenta la cantidad de tiempos únicos presentes en las señales. Esto es esencial para determinar el tamaño de la matriz de patrones.

- **Búsqueda y Modificación de Datos**
  Se implementan métodos para buscar y modificar datos en la lista enlazada. Estos métodos son fundamentales para la manipulación de datos durante el procesamiento.

- **Eliminación de Datos Repetidos**
  Cuando se encuentran datos repetidos en una señal, se utilizan métodos para eliminar los

datos duplicados y conservar solo uno de ellos. Esto contribuye a una representación más limpia y eficiente de las señales.

Es así que mediante la etapa de procesamiento de datos en nuestro programa involucra la generación de matrices de patrones y matrices reducidas, la búsqueda y modificación de datos, y la eliminación de datos duplicados. Estos procesos son fundamentales para preparar los datos para análisis posteriores, y la implementación de listas enlazadas juega un papel crucial en la gestión de estos datos de manera eficiente.

Graficas

La integración de la librería Graphviz en nuestra aplicación es esencial para visualizar gráficamente las señales procesadas y sus representaciones. A continuación, se describen los procesos clave relacionados con la generación de gráficas utilizando Graphviz:

- **Generación de Gráfica de Frecuencias**
  La gráfica de frecuencias es una representación visual de la señal original y su matriz de patrones correspondiente. En esta gráfica, los tiempos se representan en el eje horizontal, y la amplitud se representa en el eje vertical. Cada punto en la gráfica indica la presencia (1) o ausencia (0) de datos en un tiempo específico. Esto nos permite visualizar cómo varía la señal a lo largo del tiempo.
  El método `graficar` en la clase `Graph` se encarga de generar esta gráfica. Recorre la lista enlazada de datos de la señal y crea nodos para representar cada tiempo y dato en la gráfica. Utiliza conexiones entre los nodos para indicar la presencia de datos en cada punto de tiempo. El resultado es una representación gráfica de la señal que muestra su variación a lo largo del tiempo.

- **Generación de Gráfica Reducida**
  La gráfica reducida es una representación visual condensada de la señal, que muestra la agrupación de datos con el mismo tiempo y amplitud. Esta gráfica es útil para visualizar la señal de manera más compacta y comprender fácilmente las tendencias generales.
  El método `graficarMatrizReducida` en la clase `Graph` se encarga de generar esta gráfica. Recorre la lista enlazada de datos reducidos para la señal y crea nodos para representar cada grupo de datos con el mismo tiempo y amplitud. Utiliza conexiones entre los nodos para indicar la relación entre los grupos y sus respectivos tiempos. El resultado es una representación visual que simplifica la señal y resalta las características clave.

- **Generación de Imágenes de las Gráficas**
  Una vez que se han creado los nodos y las conexiones para las gráficas, utilizamos el método `render` de Graphviz para generar imágenes de las gráficas en formato PNG. Estas imágenes se guardan en archivos para su posterior visualización y análisis.
  La combinación de estas gráficas nos permite comprender mejor las señales y sus representaciones, lo que facilita la interpretación de los datos y la toma de decisiones en análisis posteriores.

- **Creación de Nodos y Conexiones Personalizados**
  Graphviz nos brinda la flexibilidad de personalizar la apariencia de los nodos y conexiones en nuestras gráficas. En nuestra implementación, hemos configurado los nodos para representar tiempos y datos de manera clara y legible. Esto incluye etiquetas personalizadas que muestran información relevante, como el tiempo y la amplitud de cada dato. Además, hemos utilizado diferentes formas y colores para resaltar la diferencia entre los tiempos y los datos en las gráficas.
- **Organización de las Gráficas**
  Para mejorar la legibilidad y comprensión de las gráficas, hemos organizado cuidadosamente los nodos y las conexiones. En las gráficas de frecuencias, los tiempos se disponen de manera ordenada en el eje horizontal, y los datos se conectan de manera lógica para mostrar la secuencia temporal. En las gráficas reducidas, los grupos de datos con el mismo tiempo y amplitud se agrupan de manera coherente para simplificar la visualización.

- **Renderizado y Almacenamiento de Imágenes**
  Una vez que se han configurado todos los elementos de las gráficas, utilizamos la función `render` de Graphviz para generar imágenes de alta calidad en formato PNG. Estas imágenes se almacenan en archivos específicos, lo que facilita su acceso y posterior análisis. La elección del formato PNG garantiza una alta resolución y claridad en las gráficas, lo que es esencial para una interpretación precisa de los datos.

- **Visualización de Gráficas Intermedias**
  Durante el proceso de procesamiento de señales, hemos incorporado la capacidad de visualizar gráficas intermedias en cualquier momento. Esto significa que podemos observar las gráficas de frecuencias y reducidas antes y después de aplicar transformaciones o reducciones adicionales. Esta funcionalidad es útil para depurar y ajustar nuestros algoritmos de procesamiento de señales.
- **Automatización de la Generación de Gráficas**
  Hemos diseñado nuestro programa para que la generación de gráficas sea un proceso automatizado y eficiente. Una vez que se inicia el procesamiento de una señal, las gráficas se generan automáticamente, lo que ahorra tiempo y evita errores humanos. Además, el programa maneja la creación y el almacenamiento de imágenes de manera transparente, sin requerir intervención del usuario.

La integración de Graphviz en nuestra aplicación no solo nos permite crear gráficas visualmente informativas de nuestras señales, sino que también nos brinda control sobre la apariencia y organización de estas gráficas. Esto mejora la comprensión de los datos y facilita la toma de decisiones en el análisis de señales, convirtiéndose en una herramienta valiosa en nuestro flujo de trabajo.

Conclusiones

En el desarrollo de nuestra aplicación para el procesamiento y análisis de señales, hemos logrado establecer una sólida metodología que abarca desde la lectura de datos hasta la generación de gráficas informativas. A lo largo de este proceso, hemos identificado varias conclusiones y aportes clave que destacamos a continuación:

1. **Lectura de Datos Precisa**
   La etapa inicial de lectura de datos es fundamental para el éxito de todo el proceso. Hemos implementado una lectura de archivos XML que es precisa y confiable, capaz de manejar diferentes señales y validar la coherencia de los datos de entrada. Esto garantiza que trabajemos con información sólida y de calidad.

2. **Almacenamiento Eficiente**
   La elección de estructuras de datos adecuadas, como listas enlazadas, para el almacenamiento de señales y datos ha demostrado ser eficiente. Esto permite una gestión ágil y escalable de los datos, lo que es esencial para procesar señales de diversas magnitudes y duraciones.

3. **Procesamiento de Datos Completo**
   Hemos implementado una serie de métodos de procesamiento de datos que incluyen la generación de matrices de patrones, matrices reducidas, y la gestión de tiempos, amplitudes y valores. Estos procesos son esenciales para obtener una representación estructurada y simplificada de las señales.

4. **Uso de Graphviz para Visualización**
   La incorporación de la librería Graphviz ha enriquecido significativamente nuestra aplicación. Hemos logrado crear gráficas de frecuencias y gráficas reducidas de manera automatizada y personalizada. Esto facilita la comprensión visual de las señales y simplifica el análisis.

5. **Flexibilidad en la Representación Gráfica**
   Hemos aprovechado la flexibilidad de Graphviz para personalizar la apariencia y organización de las gráficas. Esto nos permite adaptar las representaciones visuales a las necesidades específicas de cada señal, mejorando la interpretación de los datos.

6. **Automatización del Proceso** Nuestra aplicación automatiza el proceso de generación de gráficas y manipulación de datos, reduciendo la carga de trabajo manual y la posibilidad de errores humanos. Esto acelera el análisis de señales y garantiza resultados coherentes.

Es así que hemos logrado construir una aplicación sólida que aborda de manera integral la lectura, almacenamiento, procesamiento y visualización de señales, sentando las bases para futuros desarrollos y análisis más avanzados en el emocionante campo del procesamiento de señales.

Referencias bibliográficas

Chazallet, S. (2016). *Python 3: los fundamentos del lenguaje*. Ediciones Eni.

Sarasa Cabezuelo, A. (2017). *Gestión de la información web usando Python*. Editorial UOC.

Srinivasa, K. G., GM, S., Srinivasa, K. G., & GM, S. (2018). Getting Started with Visualization in Python. Network *Data Analytics: A Hands-On Approach for Application Development*, *333-337*.

Kaszuba, G. (2016). *Python call graph. Internet: https://pycallgraph. readthedocs. io/en/master*.