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 2023.

PROYECTO 1 IPC2

202000549 – LUIS DANIEL SALÁN LETONA

Resumen

Aplicación creada para el Centro de Investigaciones de la Facultad de Ingeniería con el objetivo de analizar y comprimir señales de audio y generar una matriz de “m” filas por “n” columnas que representa la Frecuencia y Amplitud de la señal.

La frecuencia es la cantidad de ciclos que se realizan en un segundo y se mide en Hertz (Hz), mientras que la amplitud representa la altura de la onda y hace referencia a la intensidad del sonido, la amplitud se mide en decibelios (Db).

Distintas señales de diferentes tamaños en una muestra, mas filas o mas columnas que otras, al final todas se reducirán haciendo una matriz de código binario y agrupándose por grupos de patrones según sea sus datos.

Palabras clave

Lista Enlazada, Nodo, Gráfico, Muestras, Señales, Frecuencia, Amplitud.

Abstract

Application developed for the Faculty of Engineering's Research Center with the goal of analyzing and compressing audio signals and generating a matrix of "m" rows by "n" columns that represents the signal's frequency and amplitude.

The frequency is the number of cycles per second and is measured in Hertz (Hz), whereas the amplitude is the height of the wave and relates to the intensity of the sound and is measured in decibels (Db).

Different signals of varying sizes in a sample, with more rows or columns than others, will all be reduced in the end by creating a binary code matrix and grouping by pattern groupings based on their data.

Keywords

Linked list, Node, Graph, Samples, Audio signal, Frequency, Amplitude.

Introducción

Inicialmente la aplicación muestra un Menú con opciones varias en la que le solicita al usuario identificarse con una de ellas. La primera solicita cargar un archivo tipo .XML, y haciendo uso de la librería ElementTree, se extrae la información, guardando en una lista, los datos de las señales existentes que tiene la muestra, el tiempo y la amplitud en cuanto a filas y columnas y el dato que es valor de cada una de ellas.

Una vez cargado el archivo con la información de cada Señal, el usuario puede interactuar con las múltiples opciones que contiene el menú. La gráfica se realiza mediante la librería Graphviz con los datos previamente cargados.

Desarrollo del tema

La aplicación inicia con un menú principal, desplegando múltiples opciones las cuales son:

1. Cargar archivo.
2. Procesar archivo.
3. Escribir archivo de salida.
4. Mostrar datos del estudiante.
5. Generar gráfica.
6. Inicializar sistema.
7. Salir

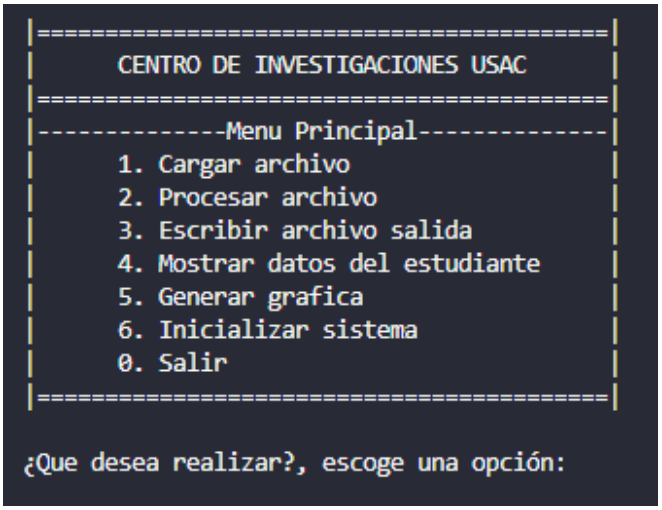

Figura 1. Menú Principal.

Al seleccionar la opción dos, se procesa el archivo y está listo para poder ser interactuado con las múltiples opciones.

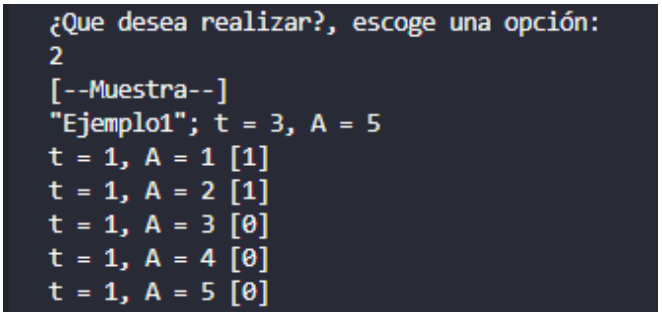

Figura 2. Muestra cargada.

Seleccionando la opción seis del menú anterior se reinicia todo el programa, quedando asi sin muestras cargadas, todo tendrá que empezar desde cero.

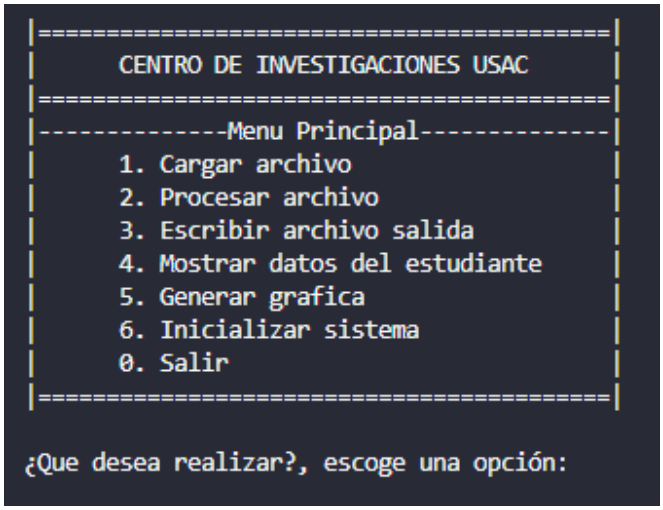

Figura 3. Sistema reiniciado.

Se utilizó una lista tipo TDA (enlazada) que funcionó como una lista preconfigurada de Python.

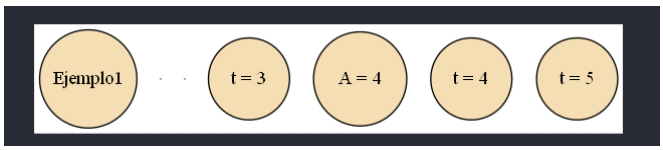

Figura 4. Gráfico de la Señal Analizada.

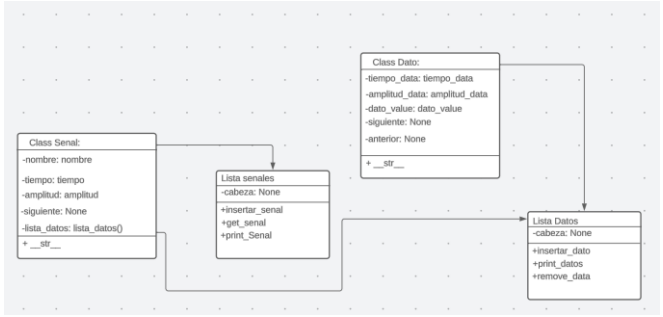

Figura 5. Diagrama de Clases.

Para poder realizar esta aplicación, fue necesario implementar estructuras de datos TDA, tanto para manejar la información de los datos de la investigación, y para poder generar la gráfica de la señal analizada con sus respectivos tiempos y amplitudes de cada dato.

Se utilizaron distintas listas enlazadas, para manejar la información de los tipos de señales de la muestra, la cantidad de filas y columnas que contiene cada muestra, siendo una más grande que la otra en cuantos a filas y columnas respectivamente.

a. Lista datos.

b. Lista senales.

```python
import graphviz
import os
# Se crea el nodo
class Dato:
    def __init__(self, tiempo_data, amplitud_data, dato_value):
        self.tiempo_data = tiempo_data
        self.amplitud_data = amplitud_data
        self.dato_value = dato_value
        self.siguiente = None
        self.anterior = None

    def __str__(self):
        return str(self.tiempo_data, self.amplitud_data, self.dato_value)

# Se crea la lista para guardar los datos


class lista_datos:
    def __init__(self):
        self.cabeza = None

    # Se crea el método para insertar datos a la lista
    def insertar_dato(self, tiempo_data, amplitud_data, dato_value):
        # dato se refiere al nodo creado anteriormente
        nuevo = Dato(tiempo_data, amplitud_data, dato_value)

        # si el primer nodo esta vacio entonces apuntara al nuevo nodo agreg
        if self.cabeza is None:
            self.cabeza = nuevo
        else:
            tmp = self.cabeza
            # si la cabeza no es nulo o vacio entonces apuntara al siguiente
            while tmp.siguiente is not None:
                tmp = tmp.siguiente
            tmp.siguiente = nuevo
```

Figura 6. Lista datos.

```python
class Senal:
    def __init__(self, nombre, tiempo, amplitud):
        self.nombre = nombre
        self.tiempo = tiempo
        self.amplitud = amplitud
        self.siguiente = None
        self.lista_datos = lista_datos()

    def __str__(self):
        return (self.nombre, self.tiempo, self.amplitud)

# Se crea la lista para guardar los datos


class lista_senales:
    def __init__(self):
        self.cabeza = None
        self.len = 0

    # Se crea el método para insertar datos a la lista
    def insertar_senal(self, nombre, tiempo, amplitud):
        # dato se refiere al nodo creado anteriormente
        nuevo = Senal(nombre, tiempo, amplitud)

        # si el primer nodo esta vacio entonces apuntara al nu
        if self.cabeza is None:
            self.cabeza = nuevo
        else:
            tmp = self.cabeza
            # si la cabeza no es nulo o vacio entonces apuntar
            while tmp.siguiente is not None:
                tmp = tmp.siguiente
            tmp.siguiente = nuevo
```

Figura 7. Lista señales.

Para la creación del gráfico se utilizó la estructura normal con nodos, dándole especificaciones tales como el borde y la separación entre filas y columnas desde un archivo Python, para luego darle una extensión .dot y así generar el archivo automáticamente con un método y luego mandarlo a llamar al menú principal, específicamente a la opción 5.

Conclusiones

El uso de listas enlazadas o de tipo TDA, son creadas desde cero, tienen una mayor funcionalidad y eficacia que una lista nativa o por defecto del lenguaje utilizado, en este caso, Python.

Al utilizar una herramienta para graficar es mucho más sencillo visualizar este tipo de aplicaciones para ver resultados más claros.

Referencias bibliográficas

MÉNDEZ, Mariano, (75.41), *Algoritmos y Programación II Tda Lista y sus Derivados.*

OJEDA, Luis Roberto, *Tda Programación Oriendtado a Objetos en Turbo. Universidad Nacional de Colombia.*

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 2023.