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

202200075 – Dominic Juan Pablo Ruano Pérez

Resumen

Utilizando Tipos de Datos Abstractos (TDA), la herramienta Graphviz para la generación de gráficos y la Programación Orientada a Objetos (POO), hemos desarrollado un sistema capaz de procesar archivos con extensión ".XML" y almacenar los datos de las tablas obtenidas en listas simples. Estas listas se han implementado mediante una clase personalizada de listas simples, creada por el usuario, y haciendo uso de una clase nodo.
La estructura de la lista simple se ha diseñado de manera secuencial para optimizar el acceso a los datos. Además, hemos creado una clase que importa la librería Graphviz, la cual se enfoca en diversas funciones para la creación de nuevos nodos y la generación de gráficos de las diferentes señales presentes en los archivos procesados.
Mediante la lectura y análisis de archivos con extensión ".XML", hemos recopilado datos que, gracias a la aplicación de lógica, bucles y estructuras de control, se han empleado para la generación de gráficas representativas de las distintas señales contenidas en los archivos procesados.

Palabras clave

1. Desarrollo
2. Procesamiento
3. Listas
4. Gráficas

Abstract

Using Abstract Data Types (ADT), the Graphviz tool for graph generation, and Object-Oriented Programming (OOP), we have developed a system capable of processing files with ".XML" extension and storing data from obtained tables into simple lists. These lists have been implemented using a custom class for simple lists, created by the user, and making use of a node class.
The structure of the simple list has been designed sequentially to optimize data access. Furthermore, we have created a class that imports the Graphviz library, which focuses on various functions for creating new nodes and generating graphs of different signals present in the processed files.
By reading and analyzing files with ".XML" extension, we have collected data that, thanks to the application of logic, loops, and control structures, has been used to generate representative graphs of the various signals contained in the processed files.

Keywords

1. *Development*
2. *Processing*
3. *Lists*
4. *Charts*

Introducción

La creación de un sistema de gestión de archivo y generación de graficas de senales obtenidas por medio de la lectura de archivos los cuales deben ser únicamente leídos y no modificados en ningún momento, la creación de un mockup en consola para poder realizar todo lo necesario para el buen funcionamiento del sistema.

Correcta gestión de listas simples donde se guardarán distintos tipos de datos, donde se guardarán senales la cuales contaran con amplitudes, tiempos y a su vez estos contendrán datos los cuales se utilizaran para genera matrices gemelas binarios y consigo una suma de tuplas con las que se obtendrá una tercera matriz llamada matriz reducida sobre la cual se centra este proyecto.

Se debe poder generar graficas de las senales requeridas en ese momento y un archivo de salida el cual será escrito en .XML y debe contener los datos anteriormente generados de la matriz reducida.

Desarrollo del tema

Para de este proyecto se creo una lista con las distintas funcionalidades que se requieren para la entrega.

a) Se genero de manera simple una interfaz para que se más fácil la navegación entre las opciones distintas del proyecto.

Código:

Para la creación del menú se usó un while el cual mostrara en todo momento el menú con la opciones a excepción de cuando se selecciona una opción, utilizando la sentencia if, elif y else. Por medio del input del usuario se puede elegir una de las opciones teniendo en cuenta que se debe hacer todo en secuencia y no se puede generar una grafica sin leer datos o leer datos sin cargar archivo.

Consola

```
while True:
    print("Menu Principal:")
    print("     1. Cargar Archivo")
    print("     2. Procesar Archivo")
    print("     3. Escribir Archivo Salida")
    print("     4. Mostrar Datos del Estudiante")
    print("     5. Generar Grafica")
    print("     6. Inicializar Sistema")
    print("     7. Salir")

    opcion = input("Ingrese una opcion: ")

    if opcion == "1":···

    elif opcion == "2":···

    elif opcion == "3":···

    elif opcion == "4":···

    elif opcion == "5":···

    elif opcion == "6":···

    elif opcion == "7":···

    else:···
```

Figura 1. Código de menú principal .
Fuente: elaboración propia.

Aquí podemos observar cómo se ve el menú en consola.

```
Menu Principal:
     1. Cargar Archivo
     2. Procesar Archivo
     3. Escribir Archivo Salida
     4. Mostrar Datos del Estudiante
     5. Generar Grafica
     6. Inicializar Sistema
     7. Salir
Ingrese una opcion:
```

Figura 2. Vista de consola del menú principal.
Fuente: elaboración propia.

b) Se implemento una clase lista simple la cual cuenta con un constructor y algunas funciones entre ellas algunas de las siguiente.

El constructor.

el cual crea los objetos.

```
class ListaSimple():
    id = 0
    def __init__(self):
        self.nodoInicio = None
        self.nodoFinal = None
        self.size = 0
```

Figura 3. Clase y Constructor de clase.
Fuente: elaboración propia.

Agregar final.

Que nos agrega un nodo al final de la lista.

```
    def agregarFinal(self, dato):
        nuevo = Nodo(self.id, dato)
        self.id += 1
        if self.estaVacia():
            self.nodoInicio = nuevo
            self.nodoFinal = nuevo
        else:
            self.nodoFinal.setSiguiente(nuevo)
            self.nodoFinal = nuevo
        self.size += 1
```

Figura 4. Función agregarFinal.
Fuente: elaboración propia.

Imprimir.

Imprimiendo los datos de la lista.

```
    def imprimir(self):
        tmp = self.nodoInicio
        while tmp != None:
            print(tmp.getDato())
            tmp = tmp.getSiguiente()
```

Figura 5. Función imprimir.
Fuente: elaboración propia.

c) Haciendo uso de la clase leerXML, se lee el archivo y devuelve al objeto con los atributos de lista de encabezado la cual es una lista con todas las señales que se encuentran en el archivo, además de generar las listas de lista de datos y lista de datos binario y dichas listas son atributos del mismo objeto.

El constructor

Hace la creación de los objetos.

```
class leerXML:
    def __init__(self, path) -> None:
        self.root = ET.parse(path).getroot()
        self.listaEncabezados = ListaSimple()
        self.listaDatos = ListaSimple()
        self.listaBinaria = ListaSimple()
```

Figura 7. Clase y constructor de clase.
Fuente: elaboración propia.

getsenal.

esta función cumple una de las funciones mas importantes de todo el programa, generando las listas de encabezados o mejor dicho una lista donde se encuentran listadas todas las senales que se pueden leer en el archivo .xml.
también realiza la lectura de las amplitudes y tiempos máximos de cada señal.
Generando una llamada lista de datos donde se obtiene los datos de cada dato junto con indicadores de amplitud y tiempo además de la señal a la que pertenecen para poder ubicarlo de forma correcta.
Y por último la creación de una lista binaria con la cual más adelante se espera poder hacer una matriz reducida.

```
def getSenal(self) -> None:
    for a in self.root.findall("senal"):
        senal1 = a.get("nombre")
        Tmax =  a.get("t")
        Amax = a.get("A")
        tmpS = senal(senal1, Tmax, Amax)
        self.listaEncabezados.agregarFinal(tmpS)
        for e in a.findall("dato"):
            tiempo = e.get("t")
            amplitud = e.get("A")
            dato2 = e.text
            tmpD = dato(tiempo,amplitud,dato2, senal1)
            self.listaDatos.agregarFinal(tmpD)
            if int(dato2) > 0:
                tmpB = dato(tiempo,amplitud,"1", senal1)
                self.listaBinaria.agregarFinal(tmpB)
            else:
                tmpB = dato(tiempo,amplitud,"0", senal1)
                self.listaBinaria.agregarFinal(tmpB)

    senguar = self.listaEncabezados.getInicio()
```

Figura 8. Función getsenal.
Fuente: elaboración propia.

Getlista.

Esta función únicamente retorna la lista de senales que se han leído del archivo.

```
    def getLista(self):
        return self.listaEncabezados
```

Figura 9. Función getlista.
Fuente: elaboración propia.

Graficar.

Las dos funciones graficar son en resumen lo mismo con la única diferencia de que graficar2 genera graficas de los datos obtenidos de la

lectura sin modificar nada y graficar3 genera una grafica de la lista binaria.
Haciendo uso de una complicada lógica se generan las graficas en base a las distintas listas.

```
def graficar2(self, valor):
    valor1 = 1
    graph = Graph()
    tmp = self.listaEncabezados.nodoInicio
    primerNodo = self.listaDatos.nodoInicio
    segundoNodo = primerNodo.getSiguiente()
    for a in range(self.listaEncabezados.size):
        if a == valor:
            graph.addEncabezado(tmp)
            primerNodoC = primerNodo
            segundoNodoC = segundoNodo
            for amplitud in range(1, int(tmp.getDato().getAmax()) + 1):
                for amplitud2 in range(1, int(self.listaDatos.size) + 1):
                    if segundoNodoC:
                        if tmp.getDato().getNombre() == primerNodoC.getDato().getSenal():
                            if primerNodoC.getDato().getSenal() == segundoNodoC.getDato().getSenal():
                                if primerNodoC.getDato().getAmplitud() == segundoNodoC.getDato().getAmplitud():
                                    graph.addNodo(primerNodoC, segundoNodoC, valor1)
                                    valor1 += 1
                                    primerNodoC = segundoNodoC
                                    segundoNodoC = segundoNodoC.getSiguiente()
                                else:
                                    segundoNodoC = segundoNodoC.getSiguiente()
                            else:
                                primerNodoC = primerNodo.getSiguiente()
                                segundoNodoC = primerNodoC.getSiguiente()
                                primerNodo = primerNodo.getSiguiente()
                                valor1 += 2
                    primerNodoC = primerNodo.getSiguiente()
                    segundoNodoC = primerNodoC.getSiguiente()
                    primerNodo = primerNodo.getSiguiente()
                    valor1 += 2
            graph.generar2(a)
            return
        tmp = tmp.getSiguiente()
```

Figura 10. Función graficar2.
Fuente: elaboración propia.

generarArchivo

esta función genera un archivo de salida en formato xml de la lista binaria.

```
def generarArchivo(self):
    with open("salida.xml", "w") as f:
        f.write("<?xml version=\"1.0\"?>\n")
        f.write("<SenalesBinarios>\n")
        tmp = self.listaEncabezados.nodoInicio
        while tmp:
            print(tmp.getDato().getNombre())
            f.write("\t<senal nombre=\"{}\" t=\"{}\" A=\"{}\">\n".format(tmp.g
            tmp2 = self.listaBinaria.nodoInicio
            while tmp2:
                print(tmp2.getDato().getSenal())
                if tmp.getDato().getNombre() == tmp2.getDato().getSenal():
                    f.write("\t\t<dato t=\"{}\" A=\"{}\">{}</dato>\n".format(t
                tmp2 = tmp2.getSiguiente()
            f.write("\t</senal>\n")
            tmp = tmp.getSiguiente()
        f.write("</SenalesBinarios>\n")
```

Figura 11. Función generarArchivo.
Fuente: elaboración propia.

d) Para hacer la carga de un archivo es simple como ingresar el nombre del archivo sin la extensión.

```
Opcion cargar archivos:
Ingrese el nombre/path del archivo (sin .xml): nombre
```

Figura 12. Vista de consola de cargar archivo.
Fuente: elaboración propia.

Siendo una función bastante básica para obtener el path del archivo ya sea que este en la carpeta raíz o fuera.

```
def cargarArchivo() -> str:
    print("Opcion cargar archivos:")
    nombre = input("Ingrese el nombre/path del archivo (sin .xml): ") + ".xml"

    if input("¿seguro que el nombre {} es correcto? (s/n): ".format(nombre)) == "s":
        return nombre

    else:
        return cargarArchivo()
```

Figura 13. Función CargarArchivo.
Fuente: elaboración propia.

e) Imprimir datos del estudiante.

Simplemente imprimiendo los datos.

```
elif opcion == "4":
    os.system("cls")
    print("Mostrar Datos del Estudiante:")
    print(">    Dominic juan pablo Ruano Perez")
    print(">    202200075")
    print(">    Introduccion a la Programacion y Computacion 2 seccion \"A\"")
    print(">    Ingenieria en Ciencias y Sistemas")
    print(">    4to Semestre")
    #mostrarDatosEstudiante()
```

Figura 14. Código de la opción 4, imprimir datos del estudiante.
Fuente: elaboración propia.

f) Inicializar sistema.

Debido al uso de POO utilizando el objeto “obj” para obtener toda la información podemos de una forma sumamente fácil inicializar a none el objeto y de esta forma tener nuestro sistema listo para un nuevo uso.
Además de limpiar el path del archivo a usar.

```
elif opcion == "6":
    os.system("cls")
    print("Inicializar Sistema:")
    nombreArchivo = None
    obj = None
    print("Sistema inicializado con exito!")
```

Figura 15. Código de la opción 6, inicializa el sistema.
Fuente: elaboración propia.

g) Salir.

Con el simple uso de un break salimos del bucle principal dando fin a nuestro sistema.

```
elif opcion == "7":
    os.system("cls")
    print("Saliendo...")
    break
```

Figura 16. Código de la opción 7, sale del programa.
Fuente: elaboración propia.

Esta sería la estructura principal del proyecto podemos observar el de POO programación orientada a objetos la creación de listas simples la cuales son recorridas nodo por nodo y estas siendo atributos del objeto único que se empleo para la lectura de los datos del xml.
Se explicaros todas las funcionalidades del menú, pero ahora se procederán a explicar algunas clases las cuales apenas de tener una gran importancia para no alargar demasiado algunas explicaciones se omitieron.

La clase senal

esta cuenta con 4 atributo. Nombre amplitud máxima,

```
class senal:
    def __init__(self, nombre, Tmax, Amax) -> None:
        self.nombre = nombre
        self.tmax = Tmax
        self.amax = Amax
        self.tiempos = ListaSimple()
        self.listatiempo()

    def getNombre(self):
        return self.nombre

    def getTmax(self):
        return self.tmax

    def getAmax(self):
        return self.amax

    def listatiempo(self):
        for i in range(1, int(self.tmax) + 1):
            objT = tiempo(i, self.amax)
            self.tiempos.agregarFinal(objT)
```

tiempo máximo y una lista de tiempos.

Figura 17. Clase señal con su constructor y sus funciones.
Fuente: elaboración propia.

Además de contar con los getters para acceder a todos los atributos de este objeto.
También cuenta con una función lista tiempo la cual se declara en el constructor para que sea creada la lista de tiempos.

La clase dato.

Esta clase es la más importante y por lo tanto mas usada en este sistema porque guarda todos los datos del archivo teniendo los siguientes 4 atributos. Tiempo, amplitud, dato, señal a la que pertenece de cada dato distinto.

Esta clase se crea para poder almacenar toda la información posible de cada dato y de esta forma poder saber su ubicación.

```
class dato:
    def __init__(self, tiempo, amplitud, dato, senal) -> None:
        self.tiempo = tiempo
        self.amplitud = amplitud
        self.dato = dato
        self.senal = senal

    def getTiempo(self):
        return self.tiempo

    def getAmplitud(self):
        return self.amplitud

    def getDato(self):
        return self.dato

    def getSenal(self):
        return self.senal

    def print(self):
        print("___SENAL: {} | Tiempo: {} | Amplitud: {} | Dato: {} ___".format( self
```

Figura 18. La clase dato, su constructor y sus funciones.
Fuente: elaboración propia.

Se puede ver que cuenta con las siguientes funciones.
Los getters de todos los atributos del objeto.
Una función de imprimir la cual fue implementada por motivo de debuggeo.

La clase tiempo

Esta clase forma parte del la lista tiempos de la clase señal.
Cuenta con tiempo, amplitud y una lista llamada amplitudes como atributos.

```
class tiempo:
    def __init__(self, tiempo, amplitud) -> None:
        self.tiempo = tiempo
        self.amplitud = amplitud
        self.amplitudes = ListaSimple()
        self.listaAmplitudes()

    def getTiempo(self):
        return self.tiempo

    def getAmplitud(self):
        return self.amplitud

    def listaAmplitudes(self):
        for a in range(1, int(self.amplitud) + 1):
            tmp = amplitud(a)
            self.amplitudes.agregarFinal(tmp)
```

Figura 19. Clase tiempo, su constructor y sus funciones.
Fuente: elaboración propia.

Se observa que como únicas funciones se tiene a los getters y una función que se ejecuta desde el constructor llamada lista simple donde se obtienen las amplitudes de la señal y se agregan a la lista llamada amplitudes.

Conclusiones

A través de la ejecución de este proyecto, he logrado adquirir un valioso conjunto de habilidades relacionadas con la manipulación de archivos XML. En particular, he profundizado en la comprensión de las estructuras internas de estos archivos, aprendiendo a acceder y procesar de manera precisa sus señales y datos. Además, he adquirido la capacidad de realizar modificaciones en estos archivos estando los datos en memoria volátil y de escribirlos de nuevo con las alteraciones necesarias. Este conocimiento no solo me ha permitido abordar con éxito casi todos los desafíos presentados por este proyecto específico, sino que también ha enriquecido habilidades generales en el manejo de datos estructurados, lo que resultará beneficioso en futuros proyectos y desafíos relacionados.

Este proyecto brindó una valiosa oportunidad para aprender el funcionamiento de la biblioteca Graphviz. Durante este proceso, adquirir un profundo entendimiento de cómo generar gráficos, especialmente a partir de los datos extraídos de archivos XML. Este conocimiento no solo resultó esencial para cumplir con los requisitos de este proyecto en particular, sino que también como una herramienta para posibles desafíos futuros que puedan surgir.

Me permitió representar con precisión la estructura de los datos de manera similar a los ejemplos de referencia. Este conjunto de habilidades se convierte, sin lugar a duda, en un recurso valioso que poder aprovechar en un futuro próximo. Estoy seguros de que el conocimiento adquirido será de gran utilidad y apreciado en diversas situaciones por venir.

Referencias bibliográficas

Szapeta, S. (2023). Proyecto1_IPC2. GitHub. https://github.com/szapeta/Proyecto1_IPC2

programmerclick.com (2023 – año de consulta) [python] Empezando con el módulo de python graphviz https://programmerclick.com/article/32021840155/

Anexos

Diagrama de clases:

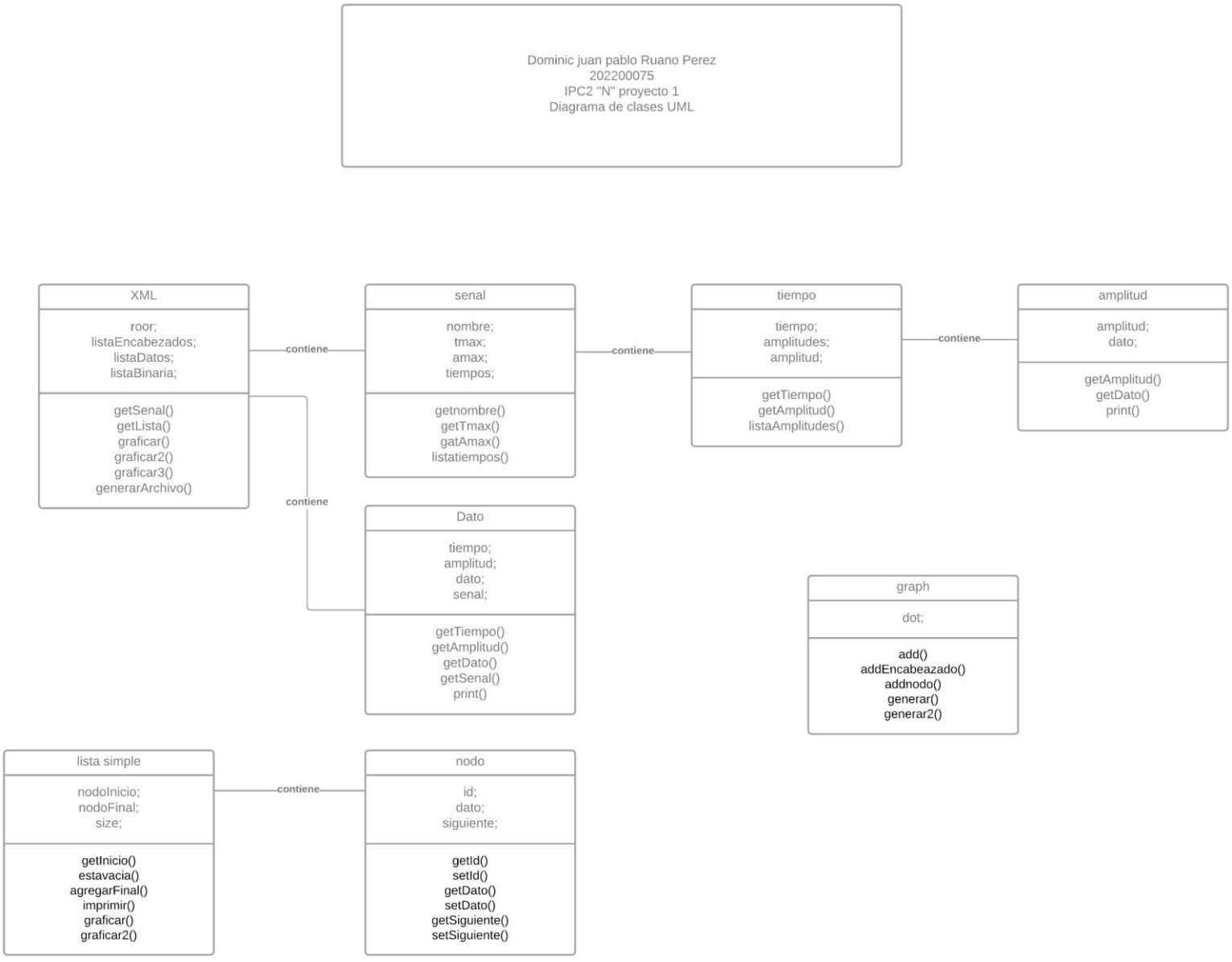

Figura 20. Diagrama de clases.
Fuente: elaboración propia.