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.

ENSAYO DE MODELO PARA RESOLVER EXPRESIONES ARITMÉTICAS SIMPLES

IPC2 INNOVATIVE PROJECTS

202204496 – Rodrigo Sebastian Castro Aguilar

Resumen

El ensayo examina detalladamente un código de procesamiento y visualización de señales en Python que demuestra la capacidad de la programación para abordar desafíos prácticos. A través de una estructura organizada y la incorporación de bibliotecas especializadas, el código carga, procesa y representa señales a través de archivos XML. La implementación de clases como "Node" y "LinkedList" ilustra la reutilización de conceptos y la modularidad, mientras que la generación de gráficos mediante la librería graphviz resalta la comunicación efectiva de datos. La preocupación por la robustez y la fiabilidad mediante validaciones muestra una mentalidad responsable. En última instancia, el código encarna la programación como una herramienta creativa para resolver problemas complejos y transformar abstracciones en soluciones concretas.

Palabras clave

Programación, Procesamiento de señales, Visualización, Estructuras de datos, Bibliotecas especializadas.

Abstract

The essay takes a detailed look at signal processing and display code in Python that demonstrates the programming's ability to address practical challenges. Through an organized structure and the incorporation of specialized libraries, the code loads, processes and represents signals through XML files. The implementation of classes like "Node" and "LinkedList" illustrate concept reuse and modularity, while graph generation using the Graphviz library highlights effective data communication. Concern for robustness and reliability through validations shows a responsible mindset. Ultimately, code embodies programming as a creative tool for solving complex problems and transforming abstractions into concrete solutions.

Keywords

Programming, Signal processing, Visualization, Data structures, Specialized libraries.

Introducción

En la era actual, la programación no solo se limita a ser un medio para crear software o aplicaciones; más bien, se ha convertido en una potente herramienta para abordar desafíos complejos y resolver problemas del mundo real. El código de procesamiento y visualización de señales en Python, que es objeto de análisis en este ensayo, es un vívido ejemplo de cómo la programación puede trascender las líneas del código para modelar y solucionar problemas tangibles en diversos campos. A medida que la tecnología sigue evolucionando, se vuelve imprescindible explorar cómo la programación, al fusionar conceptos fundamentales con herramientas especializadas, puede ser una fuerza transformadora en la resolución de problemas interdisciplinarios.

En este contexto, el código de procesamiento y visualización de señales se destaca como un microcosmos de ingenio programático. Este código, meticulosamente diseñado y estructurado, no solo refleja la habilidad técnica del programador, sino que también destila los elementos esenciales de la programación que tienen una resonancia más allá de los puros códigos y algoritmos. A lo largo de este ensayo, exploraremos cómo esta implementación demuestra la versatilidad y aplicabilidad de la programación en diversos contextos, desde la ingeniería hasta la biología, y cómo encapsula conceptos como modularidad y reutilización de código.

Es fundamental comprender cómo el código en cuestión aprovecha la programación para traducir datos abstractos en resultados concretos. Esto resalta cómo la programación se convierte en una lente que transforma datos crudos en visualizaciones gráficas claras y comprensibles, que a su vez, permiten una toma de decisiones informada. Al examinar más de cerca la integración de bibliotecas como numpy y graphviz, se revela cómo la programación se nutre de una amplia gama de herramientas para potenciar su capacidad resolutiva

A medida que profundizamos en el análisis del código, no solo consideraremos sus aspectos técnicos, sino también su capacidad para inculcar mejores prácticas. La preocupación por la robustez y la fiabilidad a través de verificaciones y validaciones refleja una ética de programación responsable, subrayando cómo los programadores deben anticipar posibles problemas y construir sistemas resistentes..

Desarrollo del tema

El código presentado ejemplifica de manera elocuente cómo la programación no solo es un conjunto de instrucciones abstractas, sino también una herramienta poderosa para abordar desafíos reales de manera estructurada y creativa. Al explorar en detalle sus componentes y enfoques, es posible desentrañar varios aspectos adicionales que resaltan la versatilidad y el impacto de la programación en la resolución de problemas tecnológicos.

Una característica notable del código es la forma en que combina múltiples bibliotecas y módulos de Python para lograr una solución completa. Más allá de la ya mencionada librería numpy para la manipulación eficiente de matrices, se observa la incorporación de la biblioteca graphviz para la generación de gráficos. Esta adición refuerza el hecho de que la programación no se limita a la codificación de algoritmos, sino que también abarca la selección inteligente y el uso de herramientas específicas para cada etapa del proceso.

Además, es esencial notar cómo el código no solo se centra en la funcionalidad, sino también en la legibilidad y la modularidad. La implementación de clases como "Node" y "LinkedList" va más allá de ser un simple ejercicio académico. Estas clases encapsulan conceptos de estructuras de datos que son fundamentales en la programación y que tienen aplicaciones en una amplia gama de escenarios. Esta abstracción demuestra la habilidad del programador para reconocer patrones y conceptos reutilizables en problemas más complejos.

Al profundizar en las capacidades de procesamiento y visualización de señales, es importante resaltar cómo estas funcionalidades son esenciales en diversas disciplinas, desde la ingeniería hasta la biología y la economía. El código se convierte en un ejemplo concreto de cómo la programación puede empoderar a los profesionales de diferentes campos para analizar y comprender datos en formas significativas. La capacidad de transformar datos crudos en gráficos comprensibles y manipulables resalta cómo la programación actúa como un puente entre los datos y la toma de decisiones informadas.

Además, el código presenta un enfoque en la robustez y la fiabilidad a través de la implementación de verificaciones y validaciones. Esta preocupación por anticipar posibles problemas y manejarlos de manera adecuada refleja la mentalidad responsable y ética que los programadores deben adoptar. Garantizar que el programa no se rompa frente a entradas inesperadas o situaciones límite es un recordatorio de cómo la programación impacta directamente en la experiencia del usuario y en la calidad del producto final.

En última instancia, el código no solo resuelve un problema técnico, sino que también demuestra cómo la programación puede ser una herramienta creativa para materializar ideas y conceptos abstractos. La manera en que toma datos en forma de archivos XML y los transforma en representaciones visuales y procesables es una manifestación tangible de cómo la tecnología puede traducir abstracciones en realidades concretas. Este proceso también subraya la importancia de la iteración y la mejora continua en la programación, ya que cada iteración del código puede llevar a una solución más refinada y eficiente.

Versatilidad y Aplicabilidad Interdisciplinaria:

El código de procesamiento y visualización de señales en Python es una manifestación concreta de cómo la programación trasciende las barreras disciplinarias. A través de la capacidad de cargar, analizar y representar datos en forma de señales, este código se convierte en una herramienta esencial en una variedad de campos. Desde la ingeniería, donde puede utilizarse para el análisis de señales eléctricas, hasta la biología, donde podría aplicarse para el estudio de patrones biológicos, el código demuestra la versatilidad de la programación para abordar una amplia gama de desafíos. Esto resalta la idea de que la programación no es solo un conjunto de habilidades técnicas, sino una competencia que puede empoderar a profesionales de diversas disciplinas para tomar decisiones informadas basadas en datos.

Legibilidad y Modularidad en la Programación:

El código en cuestión no solo se distingue por su funcionalidad, sino también por su estructura organizada y su enfoque en la legibilidad y la modularidad. La implementación de clases como

"Node" y "LinkedList" no solo permite la manipulación de listas de manera eficiente, sino que también destaca la importancia de la reutilización de conceptos en la programación. Esta abstracción de conceptos en clases independientes no solo facilita la comprensión del código, sino que también fomenta prácticas de programación más sólidas. La modularidad se extiende a la elección de bibliotecas, como numpy y graphviz, que permiten al programador aprovechar herramientas existentes en lugar de reinventar la rueda. Esto ilustra cómo la programación no solo se trata de escribir código desde cero, sino de seleccionar y combinar sabiamente las herramientas disponibles para lograr soluciones efectivas.

Transformación de Conceptos en Resultados Tangibles:

Uno de los aspectos más intrigantes del código es su capacidad para traducir conceptos abstractos en resultados tangibles. Al cargar y procesar datos desde archivos XML y convertirlos en visualizaciones gráficas comprensibles, el código refleja el poder de la programación para convertir datos en información significativa. Esta transformación es esencial para la toma de decisiones informada, ya que proporciona una forma clara y concisa de analizar y comprender conjuntos de datos complejos. Además, esta capacidad de traducir datos abstractos en representaciones visuales tiene un impacto en la depuración y la comunicación del código en sí. Al mostrar procesos internos en forma de gráficos, los programadores pueden identificar más fácilmente posibles errores y optimizaciones en su implementación.

Conclusiones

- El código demuestra cómo la programación es aplicable en diversas disciplinas, al permitir el procesamiento y la visualización de señales para un análisis informado en campos tan variados como la ingeniería y la biología.
- La implementación de clases y la selección de bibliotecas especializadas destacan la importancia de la legibilidad y la modularidad en la programación, promoviendo la reutilización de conceptos y herramientas en problemas más complejos.
- El código ejemplifica la capacidad de la programación para transformar datos abstractos en representaciones visuales y procesables, lo que subraya cómo la tecnología puede materializar ideas y resolver problemas de manera efectiva.

Anexos

```
import xml.etree.ElementTree as ET
import xml.dom.minidom as minidom
from graphviz import Digraph

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None
        self.length = 0

    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
        self.length += 1

    def exists(self, data):
        current = self.head
        while current:
            if current.data == data:
                return True
            current = current.next
        return False

    def __getitem__(self, index):
        current = self.head
        count = 0
        while current:
            if count == index:
                return current.data
```

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.

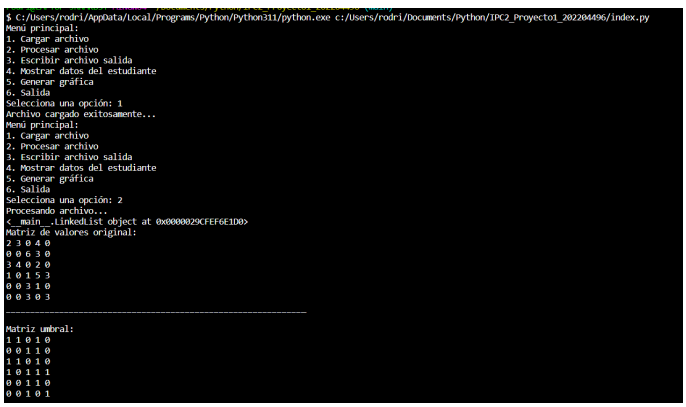
```
            if count == index:
                return current.data
            count += 1
            current = current.next
        raise IndexError("Index out of range")

    def __setitem__(self, index, data):
        current = self.head
        count = 0
        while current:
            if count == index:
                current.data = data
                return
            count += 1
            current = current.next
        raise IndexError("Index out of range")

    def __len__(self):
        return self.length

def imprimir_matriz(matriz):
    for fila in matriz:
        fila_str = ' '.join(map(str, fila))
        print(fila_str)


def imprimir_grupos(grupos):
    for grupo_num, filas in grupos.items():
        filas_str = ', '.join(map(str, filas))
        print(f"Grupo {grupo_num}: Filas [{filas_str}]")


def cargar_archivo():
    tree = ET.parse('prueba.xml')
    root = tree.getroot()

    max_amplitud = 0
    max_tiempo = 0
```

```
    max_amplitud = 0
    max_tiempo = 0

    for elemento in root.findall('senal'):
        for item in elemento.findall('dato'):
            tiempo = int(item.get('t'))
            amplitud = int(item.get('A'))

            max_amplitud = max(max_amplitud, amplitud)
            max_tiempo = max(max_tiempo, tiempo)

    matriz = LinkedList()
    for _ in range(max_tiempo):
        fila = LinkedList()
        for _ in range(max_amplitud):
            fila.insert(0)
        matriz.insert(fila)

    for elemento in root.findall('senal'):
        for item in elemento.findall('dato'):
            tiempo = int(item.get('t')) - 1
            amplitud = int(item.get('A')) - 1
            valor = int(item.text)
            fila = matriz[tiempo]
            fila[amplitud] = valor

    return matriz

def aplicar_umbral(matriz):
    matriz_umbral = LinkedList()
    for fila in matriz:
        fila_umbral = LinkedList()
        for valor in fila:
            fila_umbral.insert(1 if valor > 0 else 0)
        matriz_umbral.insert(fila_umbral)
    print(matriz_umbral)
    return matriz_umbral
```

```
def agrupar_filas(matriz_umbral):
    grupos = {}
    fila_indices = {}

    for fila_num, fila in enumerate(matriz_umbral):
        fila_str = ''.join(map(str, fila))
        if fila_str in fila_indices:
            grupo_id = fila_indices[fila_str]
            grupos[grupo_id].append(fila_num)
        else:
            nuevo_grupo_id = len(grupos) + 1
            grupos[nuevo_grupo_id] = [fila_num]
            fila_indices[fila_str] = nuevo_grupo_id

    return grupos

def sumar_valores_por_grupo(matriz_valores, grupos):
    valores_agrupados = LinkedList()
    num_columnas = len(matriz_valores[0])

    for grupo_num, filas in grupos.items():
        suma_grupo = [0] * num_columnas
        for fila_num in filas:
            suma_grupo = [a + b for a, b in zip(suma_grupo, matriz_valores[fila_num])]
        valores_agrupados.insert(suma_grupo)

    return valores_agrupados

def guardar_en_xml(filename, matriz_valores, grupos, valores_agrupados):
    root = ET.Element('senalesReducidas')

    for grupo_num, filas in grupos.items():
        atributos_grupo = {'nombre': f'Grupo {grupo_num + 1}'}
        grupo = ET.SubElement(root, 'senal', atributos_grupo)
        grupo.set('A', str(len(matriz_valores[0])))
        grupo.set('a', str(grupo_num + 1))

        grupo_element = ET.SubElement(grupo, 'grupo', g=str(grupo_num + 1))
```

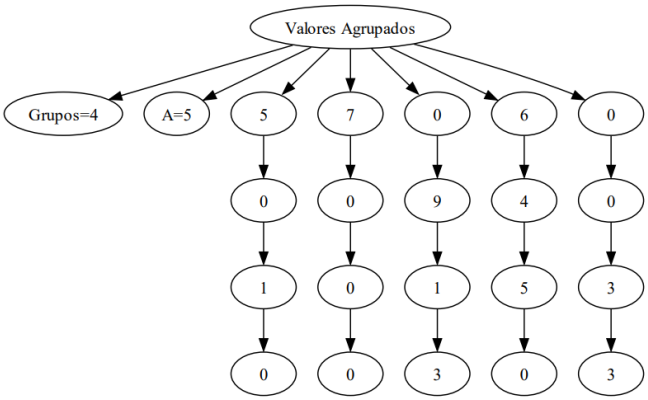
```
        grupo = ET.SubElement(root, 'senal', atributos_grupo)
        grupo.set('A', str(len(matriz_valores[0])))
        grupo.set('a', str(grupo_num + 1))

        grupo_element = ET.SubElement(grupo, 'grupo', g=str(grupo_num + 1))
        tiempos_element = ET.SubElement(grupo_element, 'tiempos')
        tiempos_element.text = ','.join(str(fila + 1) for fila in filas)

        datos_grupo_element = ET.SubElement(grupo_element, 'datosGrupo')
        for amplitud, valor in enumerate(valores_agrupados[grupo_num - 1]):
            dato_element = ET.SubElement(datos_grupo_element, 'dato', A=str(amplitud + 1))
            dato_element.text = str(valor)

    rough_string = ET.tostring(root, 'utf-8')
    reparsed = minidom.parseString(rough_string)
    pretty_xml = reparsed.toprettyxml(indent="  ")

    with open(filename, "w", encoding="utf-8") as xml_file:
        xml_file.write(pretty_xml)

def generar_grafica_matriz_original(matriz, titulo):
    dot = Digraph(titulo)
    dot.attr(rankdir='TB')

    dot.node('inicio', titulo, shape='ellipse')

    t_filas = LinkedList()
    t_filas.insert(len(matriz))
    dot.node('t_filas', label=f't={t_filas.head.data}', shape='ellipse')
    dot.edge('inicio', 't_filas')

    A_columnas = LinkedList()
    A_columnas.insert(len(matriz[0]))
    dot.node('A_columnas', label=f'A={A_columnas.head.data}', shape='ellipse')
    dot.edge('inicio', 'A_columnas')

    for i in range(A_columnas.head.data):
```

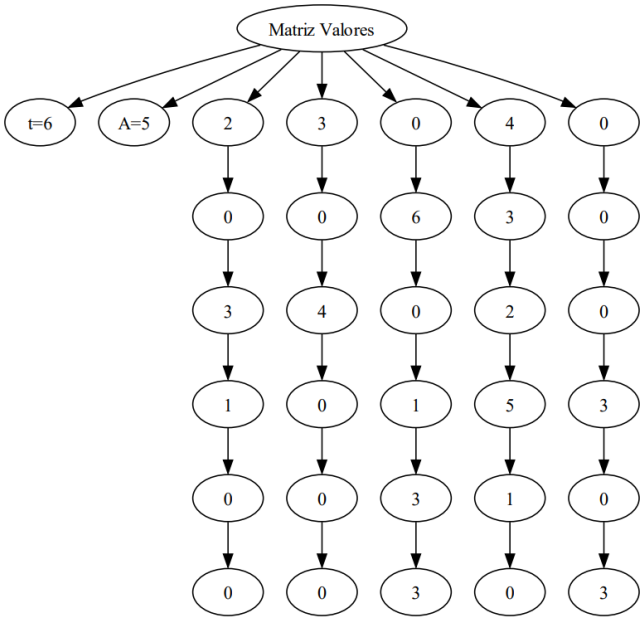
```
            node_id = f'fila_{j}_col_{i}'
            label = LinkedList()
            label.insert(str(matriz[j][i]))
            dot.node(node_id, label=label.head.data, shape='ellipse', style='filled', fillcolor='white')
            columna_node_ids.insert(node_id)

        dot.edge('inicio', columna_node_ids.head.data)

        current_node = columna_node_ids.head
        while current_node.next:
            dot.edge(current_node.data, current_node.next.data)
            current_node = current_node.next

    dot.view()

def generar_grafica_valores_agrupados(valores_agrupados, titulo):
    dot = Digraph(titulo)
    dot.attr(rankdir='TB')

    dot.node('inicio', titulo, shape='ellipse')

    t_grupos = LinkedList()
    t_grupos.insert(len(valores_agrupados))
    dot.node('t_grupos', label=f'Grupos={t_grupos.head.data}', shape='ellipse')
    dot.edge('inicio', 't_grupos')

    A_columnas = LinkedList()
    A_columnas.insert(len(valores_agrupados[0]))
    dot.node('A_columnas', label=f'A={A_columnas.head.data}', shape='ellipse')
    dot.edge('inicio', 'A_columnas')

    for i in range(A_columnas.head.data):
        columna_node_ids = LinkedList()
        for j in range(t_grupos.head.data):
            node_id = f'grupo_{j}_col_{i}'
            label = LinkedList()
            label.insert(str(valores_agrupados[j][i]))
            dot.node(node_id, label=label.head.data, shape='ellipse', style='filled', fillcolor='white')
```



```
$ C:/Users/rodri/AppData/Local/Programs/Python/Python311/python.exe c:/Users/rodri/Documents/Python/IPC2_Proyecto1_202204496/index.py
Menú principal:
1. Cargar archivo
2. Procesar archivo
3. Escribir archivo salida
4. Mostrar datos del estudiante
5. Generar gráfica
6. Salida
Selecciona una opción: 1
Archivo cargado exitosamente...
Menú principal:
1. Cargar archivo
2. Procesar archivo
3. Escribir archivo salida
4. Mostrar datos del estudiante
5. Generar gráfica
6. Salida
Selecciona una opción: 2
Procesando archivo...
<__main__.LinkedList object at 0x0000029CFEF6E1D0>
Matriz de valores original:
2 3 0 4 0
0 0 6 3 0
3 4 0 2 0
1 0 1 5 3
0 0 3 1 0
0 0 3 0 3
------------------------------------------------------------

Matriz umbral:
1 1 0 1 0
0 0 1 1 0
1 1 0 1 0
1 0 1 1 1
0 0 1 1 0
0 0 1 0 1
```

```
------------------------------------------------------------

Grupos:
Grupo 1: Filas [0, 2]
Grupo 2: Filas [1, 4]
Grupo 3: Filas [3]
Grupo 4: Filas [5]

Valores agrupados:
5 7 0 6 0
0 0 9 4 0
1 0 1 5 3
0 0 3 0 3

------------------------------------------------------------
Menú principal:
1. Cargar archivo
2. Procesar archivo
3. Escribir archivo salida
4. Mostrar datos del estudiante
5. Generar gráfica
6. Salida
Selecciona una opción: 3
Escribiendo archivo de salida...
Archivo XML 'senales_reducidas.xml' guardado exitosamente.
Menú principal:
1. Cargar archivo
2. Procesar archivo
3. Escribir archivo salida
4. Mostrar datos del estudiante
5. Generar gráfica
6. Salida
Selecciona una opción: 4
Nombre: Rodrigo Sebastian Castro Aguilar
Carné: 202204496
Curso: Introduccion a la Computacion y Programacion 2
Carrera: Ingenieria en Ciencias y Sistemas
Semestre: Cuarto Semestre
```





Referencias bibliográficas y e-grafías

- Documentación oficial de sympy: https://docs.sympy.org/
- "SymPy Tutorial" de SymPy documentation: https://docs.sympy.org/latest/tutorial/index.html