SOLUCIÓN DE MATRIZ DE ACCESO, UTILIZANDO IMPLEMENTACIÓN DE TDA Y VISUALIZACIÓN CON SOFTWARE GRAPHVIZ

Xhunik Miguel, 201900462

Resumen

Se presenta la solución para el manejo de archivos XML, el cual se procesa maneja objetos y es necesario crear una matriz para poder manipular los datos para lograr hacer varias operaciones con el mismo archivo, para eso usaremos las herramientas de Gestión de ficheros xml que se programó en el lenguaje python utilizando la librería element tree, usando librerías que representan la información estructura en el fichero xml y cómo podemos obtener información de dicha estructura, como añadir o eliminar elementos o atributos y como modificar la información de guardada. Todo es una aplicación de consola, usando programación orientado a objetos y estructura de datos para almacenar la información y manejarla con EDD, el cual nos servirá para construir un programa y solucionar el problema a través de la programación, el cual se usaron varios paradigmas lo que nos permitió una solución rápida para la app.

Palabras clave

- Matriz
- XML
- Python

Abstract

I present the solution for handling XML files, which is processed handles objects and it is necessary to create a matrix to manipulate the data to achieve several operations with the same file, for that we will use the xml file management tools that was programmed in the python language using the element tree library, using libraries that represent the information structure in the xml file and how we can get information from that structure, how to add or delete elements or attributes and how to modify the stored information. Everything is a console application, using object-

oriented programming and data structure to store the information and manage it with EDD, which will help us to build a program and solve the problem through programming, which used several paradigms that allowed us a quick solution for the app.

Keywords

- Matriz
- XML
- Python

Introducción

A continuación, se presentará la solución del problema del proyecto, el cual se usara el lenguaje de programación Python y el paradigma de programación orientado objetos, para el manejo de memoria se usara EDD (Estructura de datos).

Junto a estas dos herramientas de programación se logro manejar la memoria del archivo y poder manipularla en diferentes clases, ya que se usó orientación a objetos lo cual nos facilito y nos ayudo a tener ordenado nuestras sentencias de código.

Para este proyecto se debió haber usado varias clases y se necesito importar las clases como un módulo de Python utilizando el archivo "__init__.py" el cual fue colocado dentro del sistema de carpetas para tener un mayor control del proyecto y poder tenerlo todo ordenado.

Desarrollo del tema

El problema que nos presenta es el siguiente: "El problema de diseño de distribución consiste en determinar el alojamiento de datos de forma que los costos de acceso y comunicación son minimizados. Como muchos otros problemas reales, es un problema combinatorio NP-Hard. Algunas de las situaciones comunes que hemos observado cuando se resuelven instancias muy grandes de un problema NP-Hard son: Fuerte requerimiento de tiempo y fuerte demanda de recursos de memoria. Un método propuesto para resolver este tipo de problemas consiste en aplicar una metodología de agrupamiento."

Para este problema se usó el lenguaje Python, el cual importamos las librerías para leer el archivo, usamos la librería de "element tree" para la manipulación de archivos XML y poder pasarlas a una matriz, el cual usamos un for para recorrer la lista y después meterla en una lista.

Para todo este proceso usamos programación orientado a objetos, el cual creamos varias clases.

Esto nos ayudo a crear nuestra estructura de datos por diferentes áreas, las clases fueron las siguientes

- Para los Nodos
- Para la importación de las clases

- Para las listas
- Para el menú principal
- Para leer el archivo
- Para la matriz del archivo
- Para exportar el diagrama

Lo que se hizo fue ingresar el archivo, el archivo se guardo en una variable, recorrimos el archivo con un for, después lo convertimos en un objeto y los pasamos a la clase donde se almacenan la matriz, desde esa clase logramos grafica.

Un ejemplo de la grafica es en la figura 1. Pero para hacer las manipulaciones de la matriz tuvimos que crear un node, el cual nos sirvió para recorrer las listas que nos pedía.

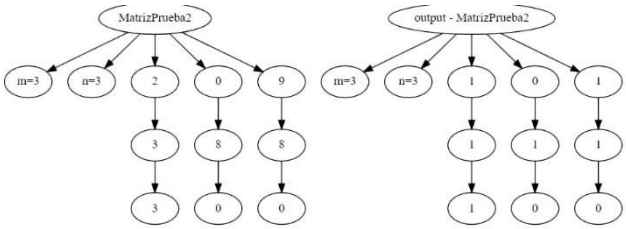

Figura 1. Grafica.

Fuente: elaboración propia.

De forma especifica para realizar la lectura del archivo se utilizo una Matriz, la cual se conforma de una lista enlazada, la cual contiene dentro de si varias listas, simulando de esta forma la estructura de datos de una matriz, para manipularse utiliza internamente 2 métodos, ‘insert’ y ‘get’, los cuales permiten interactuar con la matriz utilizando índices, simulando el funcionamiento de un array ‘bult-in’ dentro de Python, por otra parte estos métodos utilizan el método ’get_by_index’, definido para la clase Lista Enlazada, el cual permite ingresar a un elemento de la lista indicando únicamente un índice, cuyo valor puede estar comprendido en el rango [0,∞).

```
def load_file(data: ListaEnlazada):
    Tk().withdraw()
    filename = askopenfilename()
    xml_tree = ET.parse(filename)
    xml_root = xml_tree.getroot()
    for matrix in xml_root:
        matrix_attrib = matrix.attrib
        data.add_to_end(Matriz(matrix_attrib['nombre'], int(matrix_attrib['m']),
                               int(matrix_attrib['n'])))
        for obj in matrix:
            obj_attrib = obj.attrib
            data.get_last().insert(int(obj_attrib['x'])-1,
                                   int(obj_attrib['y'])-1, int(obj.text))
```

Figura 2. Lectura de archivo.

Fuente: elaboración propia.

Para realizar la manipulación de los datos, es decir para permitir que los datos sean accesibles desde todos los ámbitos, se declararon 2 listas enlazadas en el inicio del programa, las cuales se pasan como parámetros a todas las funciones que puedan manipular los datos, esto se hace aprovechando la funcionalidad de Python, que permite pasar por referencia todos los valores, de tal manera que, si un valor es modificado por una función, este realmente es modificado para todas las funciones y/o clases que requieran de su acceso.

```python
def process_file(data: ListaEnlazada, output: ListaEnlazada):
    output.clear()
    count = 0
    while data.get_size() > count:
        matrix = data.get_by_index(count)
        output_matrix = Matriz(
            'output - {}'.format(matrix.name), matrix.m, matrix.n)
        y_count = 0
        while matrix.n > y_count:
            x_count = 0
            while matrix.m > x_count:
                value = matrix.get(x_count, y_count)
                if value > 0:
                    output_matrix.insert(x_count, y_count, 1)
                else:
                    output_matrix.insert(x_count, y_count, 0)
                x_count = x_count + 1
            y_count = y_count + 1
        output.add_to_end(output_matrix)
        count = count + 1


def write_file(output: ListaEnlazada):
    def save():
        files = [('XML Files', '*.xml')]
        filename = asksaveasfilename(filetypes=files, defaultextension=files)
        return filename

    count = 0
    print('Selecciona una matriz: ')
    while output.get_size() > count:
        print('{}. {}'.format(count, output.get_by_index(count).name))
        count = count + 1

    matrix_index = int(input('> '))
    matrix = output.get_by_index(matrix_index)

    xml_root = ET.Element(
        'matriz', {'nombre': matrix.name, 'm': str(matrix.m), 'n': str(matrix.n)})

    y_count = 0
    while matrix.n > y_count:
        x_count = 0
        while matrix.m > x_count:
            matrix_element = ET.SubElement(
                xml_root, 'dato', {'x': str(x_count), 'y': str(y_count)})
            matrix_element.text = str(matrix.get(x_count, y_count))
            x_count = x_count + 1
        y_count = y_count + 1

    bin_xml = ET.tostring(xml_root)

    with open(save(), 'wb') as f:
        f.write(bin_xml)
```

Figura 3. Procesar y Escribir Archivo.

Fuente: elaboración propia.

Para realizar el renderizado utilizamos la herramienta ‘graphviz’, la cual debe encontrarse instalada dentro del ordenador como un programa, y posteriormente debe estar incluido en la variable ‘PATH’, de manera que pueda ser ejecutada por medio del interprete de comandos, desde Python utilizando el método ‘os.system’, para realizar el renderizado se escribe un archivo ‘*.dot’, el cual luego es procesado por el software graphviz, generando en este caso una imagen vectorial, es decir un ‘*.svg’, el cual luego por medio de la función ‘os.startfile’, es abierto por un programa instalado en el sistema, que tenga la capacidad de ejecutarlo.

```python
def render_graph(data: ListaEnlazada, output: ListaEnlazada):
    count = 0
    print('Selecciona una matriz: ')
    while data.get_size() > count:
        print('{}. {}'.format(count, data.get_by_index(count).name))
        count = count + 1

    matrix_index = int(input('> '))
    output_matrix = output.get_by_index(matrix_index)
    data.get_by_index(matrix_index).render_matrix(output_matrix)
```

Figura 4. Renderizar archivo

Fuente: elaboración propia.

```python
def render_matrix(self, process: Matriz):
    temp_file = open('graph.dot', 'w+')
    temp_file.write('digraph G {\n')  # begin file
    temp_file.write('name [label="{}"];\n'.format(self.name))
    temp_file.write('m [label="m={}"];\n'.format(self.m))
    temp_file.write('n [label="n={}"];\n'.format(self.n))
    temp_file.write('name -> m;')
    temp_file.write('name -> n;')

    # Gráfica entrada
    x_count = 0
    while self.m > x_count:

        y_count = 0
        while self.n > y_count:
            temp_file.write('x{}y{} [label="{}"];\n'.format(
                x_count, y_count, self.get(x_count, y_count)))

            if not (y_count + 1) >= self.n:
                temp_file.write(
                    'x{}y{} -> x{}y{};\n'.format(x_count, y_count, x_count, y_count+1))
            y_count = y_count + 1

        temp_file.write('name -> x{}y{};'.format(x_count, 0))
        x_count = x_count + 1

    # Gráfica salida procesada
    temp_file.write('proc_name [label="{}"];\n'.format(process.name))
    temp_file.write('proc_m [label="m={}"];\n'.format(process.m))
    temp_file.write('proc_n [label="n={}"];\n'.format(process.n))
    temp_file.write('proc_name -> proc_m;')
    temp_file.write('proc_name -> proc_n;')
    p_x_count = 0
    while process.m > p_x_count:
        p_y_count = 0
        while process.n > p_y_count:
            temp_file.write('px{}py{} [label="{}"];\n'.format(
                p_x_count, p_y_count, process.get(p_x_count, p_y_count)))

            if not (p_y_count + 1) >= process.n:
                temp_file.write(
                    'px{}py{} -> px{}py{};\n'.format(p_x_count, p_y_count, p_x_count, p_y_count+1))
            p_y_count = p_y_count + 1

        temp_file.write('proc_name -> px{}py{};'.format(p_x_count, 0))
        p_x_count = p_x_count + 1

    temp_file.write('}\n')  # end file
    temp_file.close()
    system('dot -Tsvg graph.dot -o output.svg')
    startfile('output.svg')
```

Figura 5. Renderizar archivo

Fuente: elaboración propia.

Conclusiones

Este proyecto se concluyó que la programación orientada a objetos es muy fundamental y es muy importante para desarrollar cualquier app, ya que por sus varias ventajas ayudan mucho a poder tener

un mejor orden del programa, pero también lo importante es que la mejor forma de manipular los datos de una matriz es a través de las estructuras de datos y junto con la herramienta Graphviz se logra graficar cualquier matriz desde las librerías.

Referencias bibliográficas

- https://rico-schmidt.name/pymotw-3/xml.etree.ElementTree/parse.html
- https://docs.python.org/3/library/xml.etree.elementtree.html
- https://www.geeksforgeeks.org/xml-parsing-python.html