AUTOMATIZACION DE INSTALACION Y MODIFICACION DE PISOS ARTESANALES.

202006353 – Juan Josue Zuleta Beb

Resumen

Se presenta una solución para el manejo de archivos con extensión XML en el lenguaje de programación Python con el uso de la librería MiniDom. Implementa programación orientada a objetos para el manejo de la memoria estática y se utilizan clases Nodo y Lista enlazada para el manejo de la memoria dinámica. Para el desarrollo del proyecto se utilizó estructuras tradicionales de ordenamientos de datos para generar matrices con los datos, se generó gráficos por medio de la herramienta Graphviz, presentando los pisos en forma de matriz, para que el usuario tenga una mejor idea de lo que son los patrones extraídos del archivo XML. El dominio total del proyecto se basó en el paradigma de programación orientado a objetos y la manipulación de estructuras de datos por medio de la implementación de tipo de datos abstractos TDAs. En conclusión, se puede decir que este proyecto es una introducción a las estructuras de datos y a la implementación de tipos de datos abstractos dentro del paradigma de programación orientada a objetos.

Palabras clave

- XML
- MiniDom
- Nodo
- Lista Enlazada
- TDAs (Tipo de Datos Abstractos)
- Graphviz

Abstract

A solution for handling files with XML extension in the Python programming language with the use of the MiniDom library is presented. It implements object-oriented programming for static memory management and Node and LinkedList classes are used for dynamic memory management. For the development of the project, traditional data ordering structures were used to generate matrices with the data, graphs were generated through the Graphviz tool, presenting the floors in the form of a matrix, so that the user has a better idea of what they are. the patterns extracted from the XML file. The total domain of the project was based on the object-oriented programming paradigm and the manipulation of data structures through the implementation of abstract data types ADTs. In conclusion, it can be said that this project is an introduction to data structures and the implementation of abstract data types within the object-oriented programming paradigm.

Keywords

- XML
- MiniDom
- Node
- Linked List
- ADTs (Abstract Data Type)
- Graphviz

Introducción

A continuación, se presentará la solución del problema del proyecto, para la cual se utilizó el lenguaje de programación Python y el paradigma de programación orientado objetos, y para el manejo de memoria se utilizó EDD (Estructura de datos) con implementación de tipos de datos abstractos TDAs.
Estas herramientas de programación permitieron manejar la memoria del archivo y poder manipularla en diferentes clases, sabiendo que ya que se usó el paradigma orientado a objetos esto facilitó el orden y la comunicación de datos entre clases y así evitar la perdida de datos enviados y recibidos entre las mismas. Para las distintas clases se utilizaron las importaciones nativas de Python las cuales nos permiten manejar las clases dentro del mismo archivo evitando así importaciones circulares y también falta de importaciones en el manejo de las distintas clases.

Desarrollo del tema

El problema de diseño de distribución de pisos en distintos patrones consiste generalmente en un problema de determinar los costos mínimos del intercambio de patrones a través de dos movimientos simples, intercambiar y voltear, tomando en cuenta que el piso es reversible y que puede tomar dos colore los cuales son blanco y negro esto nos desafía a determinar el alojamiento de datos de forma que los costos de volteo y de intercambio sean 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. Iniciando por extraer los datos del archivo XML y almacenándolos en los objetos de las clases, esta tarea la realizamos por medio de la librería que implementa Python llamada MiniDom, la cual nos permite buscar fuentes de datos por medio de nombres de etiquetas, de esta manera separamos los nombre de los pisos y a cada nombre le asignamos sus atributos coste de volteo coste de intercambio así como sus dimensiones en forma de matriz para la impresiona la cual se basa en filas y columnas, y lo mas importante se le asignaron los patrones de colores que podrá implementar coda uno de los pisos.

Una vez extraídos los datos y almacenados en las siguientes clases:

- Nodos pisos
- Lista pisos
- Nodo patrón
- Lista patrones
- Nodo color
- Lista colores

```
#-------------------------------------------------------------------------
def MiniDom(ruta, linked_list):
    mydoc = minidom.parse(ruta)
    decks = mydoc.getElementsByTagName('piso')
    for deck in decks:

        deck_name = deck.attributes['nombre'].value
        rows = deck.getElementsByTagName('R')
        columns = deck.getElementsByTagName('C')
        flip = deck.getElementsByTagName('F')
        slide = deck.getElementsByTagName('S')

        for a in rows:
            rows = a.firstChild.data
        for b in columns:
            columns = b.firstChild.data
        for c in flip:
            flip_price = c.firstChild.data
        for d in slide:
            slide_price = d.firstChild.data

        decks = linked_list.add_to_end(deck_name,rows,columns,flip_price,slide_price)
```

Figura 1. Extracción de datos.

Fuente: Elaboración propia.

La manipulación de estas se llevo a cabo por medio de un método llamado menú principal, por medio del cual accedimos a todas las listas y nodos y manejamos ampliamente y a voluntad nuestros datos almacenados.

Para iniciar con la ejecución se debe ingresar los datos para lo cual se debe cargar un archivo de extensión XML que cumpla con la estructura inicialmente planteada siguiendo las mismas distribuciones de nombre y raíces, por consiguiente podremos observar por separado cada uno de los pisos con sus atributos separados y sus patrones listos para ser impresos, por lo cual realizamos un algoritmo tradicional para contar filas y columnas y generar posiciones para una matriz de m x n dimensiones y con la ayuda de la herramienta graphviz podemos iniciar a escribir un archivo txt por medio de la librería neato, generamos matrices graficas como esta:

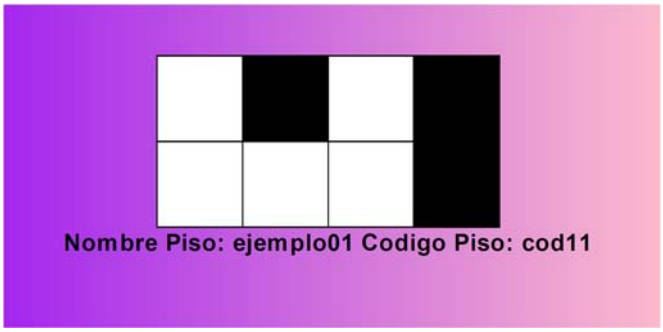

Figura 2. Impresión de pisos.

Fuente: Elaboración propia.

De forma mas especifica podemos realizar una inspección minuciosa a la generación grafica en forma de matriz quedando a luz alguna inquietudes como de donde salen las posiciones o como se almacenan los colores, esto se realiza una vez generado la lectura del archivo XML al extraer los patrones, antes de guardarlos en memoria se recorren los patrones y se asignan posiciones en coordenadas (X , Y) de esta forma: Si el carácter corresponde a W entonces el color de ese nodo será Blanco se le asigna el código de color blanco y si el carácter corresponde a B entonces el color de ese nodo será Negro y se le asigna una posición en memoria con dígitos del 1 en adelante. Esto ayudara a graphviz a no confundir nodos del mismo color, para las posiciones comparamos si las columnas son mayores que las filas mientras esto se cumpla se podrá avanzar en las columnas con contadores inicializados en 0,0 para que las coordenadas estén en este orden, una luego se verifica que las filas sean mas grandes que las columnas de esta forma podremos insertar valores en las columnas y una vez que el valor de la columna actual sea menor que el valor de la columna de la matriz entonces se cambiara de fila para poder seguir llenando la matriz con contadores, el contador de columnas se regresa a cero cada vez que se cambia de fila y de esta forma se logra asignar coordenadas a los nodos de colores, una vez asignadas, se envían a la clase Boxes_list, donde se genera un constructor para añadir el nodo a la lista con el método .add_to end()

```
patternses = deck.getElementsByTagName('patrones')

for patterns in patternses:
    patterns = deck.getElementsByTagName('patron')
    for e in patterns:
        pattern_code = e.attributes['codigo'].value
        pattern = (e.firstChild.data)
        codes = decks.code_patterns.add_to_end(pattern_code, pattern)

        pos_x = 0
        pos_y = 0
        contador = 1

        Black = '#000000'
        Withe = '#FFFFFF'
        Font_W = 'white'
        Font_B = 'black'

        for color in pattern:
            count_rows = 0
            while int(rows) > count_rows:
                count_cols = 0
                if color == 'W':
                    color = 'W{}'.format(str(contador))
                    codes.color_patterns.add_to_end(color,pos_x,pos_y,Withe,Font_W)
                    contador += 1
                    pos_x += 1
                elif color == 'B':
                    color = 'B{}'.format(str(contador))
                    codes.color_patterns.add_to_end(color,pos_x,pos_y,Black,Font_B)
                    contador = contador +1
                    pos_x += 1
                while int(columns) > count_cols:
                    count_cols += 1
                if color == 'W':
                    color = 'W{}'.format(str(contador))
                    codes.color_patterns.add_to_end(color,pos_x,pos_y,Withe,Font_W)
                    contador = contador +1
                    pos_x += 1
                elif color == 'B':
                    pos_x += 1
                    color = 'B{}'.format(str(contador))
                    codes.color_patterns.add_to_end(color,pos_x,pos_y,Black,Font_B)
                    pos_x += 1
                    contador += 1
                    pos_y += 1
                count_rows = count_rows + 1
                if pos_x >= int(columns):
                    pos_y += 1
                    pos_x = 0
```

Figura 3. conversión de XML a matriz de colores.

Fuente: Elaboración propia.

Una vez que tenemos cada nodo como un objeto de nuestra lista con cada uno de sus atributos definidos, podemos pasar a manipularlos en cualquier clase y en cualquier método, de esta premisa se sugiere que lo mejor sea manipularlos en la misma lista enlazada en donde fueron creados, por lo cual en la lista Boxes_list se crea el método de impresión que recibe por parámetros el nombre y el código de la gráfica que se imprimirá y este recorre los nodos hasta el final de la lista y por cada uno se toma el color y sus posiciones en X y en Y para ubicarlos por medio de la herramienta graphviz con lo que nuestro método de impresión queda así:

```
def show_Boxes(self, name , code):
    tmp = self.boxes_head

    strGrafica = 'digraph G { \n graph [pad="1" bgcolor="purple:pink" style="filled" margin="0"  penwidth="3"] \n'
    strGrafica += 'label = "Nombre Piso: {} Codigo Piso: {}" fontname="Arial Black" fontsize="20pt" \n'.format(name,code)
    strGrafica += 'node [style = filled shape = box height="1" width="1"] \n'

    while tmp != None:
        strGrafica += '{}[fillcolor= "{}" fontcolor = "{}" pos="{},-{}!"] \n',
        format(tmp.get_Box(),tmp.getColor_Box(),tmp.getFont_Box(),tmp.getPosX_Box(),tmp.getPosY_Box())
        tmp = tmp.getNext_Box()

    strGrafica += ' } '
    documentotxt = 'textoplano.txt'
    with open(documentotxt,'w') as grafica:
        grafica.write(strGrafica)
    pdf = 'G{}{}.pdf'.format(name,code)
    os.system( 'neato -Tpdf ' + documentotxt + ' -o ' + pdf )
    webbrowser.open(pdf)
```

Figura 4. Método de impresión en forma de matriz.

Fuente: Elaboración propia.

utilizando el método ‘os.system’, para realizar renderizado se escribe un archivo ‘*.txt, el cual luego es procesado por el software graphviz, generando en este caso un archivo pdf, es decir un ‘*.pdf, el cual luego por medio de la función ‘webbrowser.open(), es abierto por un programa instalado en el sistema, que tenga la capacidad de ejecutarlo.

De esta forma podemos manejar en general nuestros nodos de colores por lo cual para calcular los valore de los cambios únicamente debemos comparar, el precio de intercambio o de volteo y comparar en que posiciones estamos y cuáles son nuestras referencias de llegada ósea, el código de colorea hacia el cual deseamos llegar realizando validaciones de pociones en (x,y) para un mejor control de las posiciones, también podemos aclarar que para los precios se debe definir cuál es más caro y cual más barato, según los datos obtenidos al cargar el XML por lo cual siempre se dará prioridad a las operaciones que tengas un menor impacto en el precio final y en la menor cantidad de movimientos posibles.

Figura 5. Algoritmo de cálculo de costos.

Fuente: Elaboración propia.

Conclusiones

Se puede concluir que la programación orientada a objetos es fundamental para el manejo y la implementación de estructuras de datos y también para la implementación de tipos de datos abstractos.

También se puede afirmar que las estructuras de datos son la base fundamental para poder construir cualquier tiempo de algoritmo que necesite almacenar datos y manejarlos a voluntad.

Se concluye que los tipos de datos abstractos como las listas enlazadas y las listas doblemente enlazadas y su manejo mediante nodos son la mejor forma de aprender a manejar realmente la programación orientada a objetos y todas sus características poco resaltadas al utilizar librerías nativas en los lenguajes de programación.

Referencias bibliográficas

- https://ebooksonline.es/leer-un-archivo-xml-de-ejemplo-minidom-elementtree/
- https://www.graphviz.org/documentation/

Anexos

Diagrama de clases: