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 2022.

# Proyecto 1. Robot Pisos Artesanales SA.

**Carné 202010905 – Cristian Fernando Hernández Tello**

## Resumen

El robot fue creado con el propósito de colocar los pisos de dimensiones R x C con cualquier patrón, el proyecto consiste en poder encontrar el intercambio de patrones de la forma menos costosa posible , en base al costo de volteo de azulejo y el movimiento de azulejos que tienen un costo predeterminado. Al intercambiar los patrones el programa genera un menú con las opciones de mostrar la instrucción paso a paso de como el robot fue intercambiando el patrón de la manera más optima posible.

El proyecto se trabajó en base a programación orientada a objetos en conjunto con TDA (Tipos de datos Abstractos), en el proceso se crearon Nodos y Clases necesarias para la utilización de listas enlazadas, así como también en este proyecto no se utilizó ninguna de las listas predeterminadas.

El robot en el algoritmo comparando cada nodo de la lista enlazada simple de forma horizontal y vertical, nunca en diagonal, el algoritmo fue de creación propia utilizando ciclos for y condicionales If.

## Palabras clave

- Estructura de datos.
- Grafos
- Matriz Octogonal
- Nodo
- Lista enlazada

## *Abstract*

*The robot was created with the purpose of placing the floors of dimensions R x C with any pattern, the project consists of being able to find the interchange of patterns in the least expensive way possible, based on the cost of turning the tile and the movement of tiles. They have a predetermined cost. When exchanging the patterns, the program generates a menu with the options to show the step-by-step instruction of how the robot was exchanging the pattern in the most optimal way possible.*

*The project was worked on based on object-oriented programming in conjunction with TDA (Abstract Data Types), in the process Nodes and Classes necessary for the use of linked lists were created, as well as in this project none of the default lists.*

*The robot in the algorithm comparing each node of the simple linked list horizontally and vertically, never diagonally, the algorithm was self-created using "for" loops and "If" conditionals.*

## *Keywords*

- *Data Structure*
- *Graphs*
- *Octogonal Matrix*
- *Node*
- *Linked List*

## Introducción

Para la realización de este proyecto se implementó como base el lenguaje de programación Python en su octava versión, se trabajó en base a los conceptos de programación orientada a objetos, junto con conceptos de listas enlazadas y nodos, el archivo de entrada es un lenguaje de marcado extensible o mejor conocido como XML, para la lectura del mismo se utilizó la librería de Python llamada minidom, la cual es una implementación de la interfaz de modelado de documentos de objetos, en su mayoría se utiliza listas enlazadas para ingresar la información extraída del archivo de entrada XML.

Para el grafo donde se representa el piso se utiliza la herramienta de graphviz, la cual es una herramienta para visualizar información estructural, tal y como se necesita en este proyecto.

Se trata de brindar un algoritmo de costo mínimo para el robot que estará utilizando el fabricante de azulejos, al final se generarán reportes de dichas instrucciones de cambio al usuario final.

## Desarrollo del tema

En este proyecto se trabajó con el modelo de estructura de datos que consiste en desarrollo de la creación de una ruta óptima para el recorrido del robot en los diferentes patrones de los pisos , se creó una clase Nodo Piso que contiene los parámetros necesarios para la información de cada piso, tal como el nombre, dimensión de las filas , dimensión de las columnas , y la lista de patrones.

```python
from ListaPatron import ListaPatron


class NodoPiso():
    def __init__(self, nombre, filas, columnas, costoVolteo, costoMov):
        self.nombre = nombre
        self.filas = filas
        self.columnas = columnas
        self.costoVolteo = costoVolteo
        self.costoMov = costoMov
        self.siguiente = None
        self.patrones = ListaPatron()
        #self.casillas = Matriz() #Matriz
```

*Figura 1.* Nodo Piso.

Fuente: elaboración propia, 2022.

Se creó también la clase Lista Pisos, que indicará los apuntadores de la lista enlazada de patrones y azulejos, la clase Nodo Patrones, contiene la información necesaria para cada patrón, como su código y la lista de azulejos que representa cada cuadrito del patrón de un piso a graficar.

```python
from NodoPiso import NodoPiso
from os import startfile, system
class ListaPisos():
    def __init__(self):
        self.inicio=None
        self.fin=None
        self.dimension=0

    def insertarPiso(self,nombre,filas,columnas,costoVolteo,costoMov):
        nuevoPiso=NodoPiso(nombre,filas,columnas,costoVolteo,costoMov)
        self.dimension+=1
        if self.inicio is None:
            self.inicio=nuevoPiso
            self.fin=nuevoPiso
        else:
            # actual=self.inicio
            # while actual.siguiente is not None:
            #     actual=actual.siguiente
            # actual.siguiente=nuevoPiso

            self.fin.siguiente=nuevoPiso
            self.fin=nuevoPiso
            # nuevoPiso.anterior=actual
            # self.fin=nuevoPiso
        return nuevoPiso

```

*Figura 2.* Ejemplo de funciones Lista Pisos.

Fuente: elaboración propia, 2022.

Se creó un nodo patrón que contiene la información del código del patrón y una lista de los Azulejos que lo conforman.

```python
from ListaAzulejo import ListaAzulejo


class NodoPatron():
    def __init__(self, codigo):
        self.codigo = codigo
        self.listaAzulejos = ListaAzulejo()
        self.siguiente=None
        self.anterior=None

```

*Figura 3.* Nodo patrón.

Fuente: elaboración propia, 2022.

Se creó una lista patrones que contiene funciones básicas para realizar con el patrón como insertar, mostrar, buscar y retornan el patrón por defecto ene l archivo de entrada.

```python
class ListaPatron():
    def __init__(self):
        self.inicio=None
        self.fin=None
        self.dimension=0

    def insertarPatron(self,codigo):
        nuevoPatron=NodoPatron(codigo)
        self.dimension+=1
        if self.inicio is None:
            self.inicio=nuevoPatron
            self.fin=nuevoPatron
        else:
            # actual=self.inicio
            # while actual.siguiente is not None:
            #     actual=actual.siguiente
            # actual.siguiente=nuevoPiso

            self.fin.siguiente=nuevoPatron
            nuevoPatron.anterior=self.fin
            self.fin=nuevoPatron
            # nuevoPiso.anterior=actual
            # self.fin=nuevoPiso
        return nuevoPatron

    def mostrarPatrones(self):...

    def mostrarNombrePatron(self):...

    def buscarPatron(self,codigoPatron):...

    def getPatronDefecto(self):
        actual = self.inicio
        return actual.codigo
```

*Figura 4.* Funciones Lista Patrón.

Fuente: elaboración propia, 2022.

Para la lectura del archivo se implementó un ciclo for que recorre las etiquetas de cada piso y luego extrae el nombre de cada piso, para las opciones que tiene el usuario se creó un menú donde ingresa la opción que desea realizar, en la lectura del archivo no tiene implementado si las etiquetas de XML vienen en minúsculas o mayúsculas.

```python
def cargarArchivo():
    global path
    path = input("Ingrese la ruta del archivo: ")
    try:
        file = open(path)
        global text
        parsear=file.read()
        text = parsear.lower()
    except:
        print("Ruta invalida")

def procesarArchivo():
    global pisos
    global listaP
    global filas,columnas
    listaP=ListaPisos()
    listaPatrones=ListaPatron()
    archivo=str(path)
    mydoc = minidom.parse(archivo)
    pisos = mydoc.getElementsByTagName('piso') # pisps = mydoc.getElementsByTagName('piso')
    # print()
    for x in pisos: # for piso in pisos
        global nombre
        nombre=x.attributes['nombre'].value
        filas=x.getElementsByTagName('R')[0].firstChild.data
        columnas=x.getElementsByTagName('C')[0].firstChild.data
        costoVolteo=x.getElementsByTagName('E')[0].firstChild.data
        costoMov=x.getElementsByTagName('S')[0].firstChild.data
        nuevoPiso=listaP.insertarPiso(nombre,filas,columnas,costoVolteo,costoMov)
        patrones = x.getElementsByTagName('patron')

        for item in patrones:
            nuevoPatronPiso=item.attributes['codigo'].value
            piso=nuevoPiso.patrones.insertarPatron(nuevoPatronPiso)
            azulejos=item.firstChild.data.replace('\n','')
            azulejos=azulejos.replace(' ','')
```

*Figura 5.* Carga y procesamiento del archivo XML.

Fuente: elaboración propia, 2022.

El algoritmo utilizado para desarrollar el costo óptimo para el intercambio de pisos se basó en comparar azulejo del patrón por defecto con el azulejo en la misma posición del patrón nuevo que desea cambiar, esto lo hace según el número de columnas y filas que tenga predeterminado dichos pisos en el archivo de entrada.

Cuando finalmente el intercambio de patrones culmina, al usuario se le pregunta si desea ver las series de pasos que utilizó el robot para intercambiarlos, en consola, o ya se bien quiere generar un archivo .HTML para ver de una forma más amigable la serie de pasos.

```python
def Instrucciones():
        global instrucciones,Total
        print("¿Como desea ver las instrucciones?")
        print("1---En consola")
        print("2---Un archivo HTML")
        print("3---No deseo verlas, volver al menu anterior")
        opcion=input("Ingrese la opcion: ")
        if opcion=="1":
            print("---------------INSTRUCCIONES-----------")
            print(instrucciones)
            print("---Costo Operacion---",Total)
                # comparacion=patronDefecto.compararXCuadrito()
            menuXPiso(p)
        elif opcion=="2":
            instruccionesHTML()
        elif opcion=="3":
            menuCambiarPatron()
        else:
            print("Error, intentelo de nuevo")
            Instrucciones()
```

*Figura 6.* Generación de archivo de Instrucciones.

Fuente: elaboración propia, 2022.

En este proyecto se utilizando los tipos de datos abstractos para entender de una mejor manera como las listas realmente funcionan, aplicando en el programador una abstracción de estás de una manera en la que el necesita de la implementación de una lógica más compleja para una función tan sencilla como podría ser la inserción de datos en una lista, u otras operaciones que podrían llevar un poco más de complejidad como el buscar un elemento en una lista enlazada u ordenar los nodos de esta de forma descendente y ascendente.

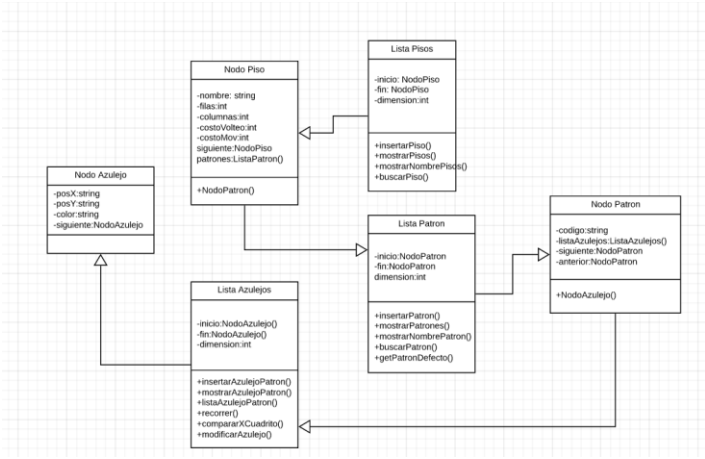


*Figura 7 Diagrama de Clases*.

Fuente: elaboración propia, 2022.

## Conclusiones

Se pudo concluir que el uso de listas enlazadas puede sustituir el uso de las listas predefinidas por Python de una buena manera en la estructura de los datos a analizar de un archivo XML.

Se demostró que puede generarse un grafo dinámico para cada uno de los diferentes pisos a analizar que recorrerá el robot de la empresa de Pisos Artesanales SA.

Con los TDA se consigue más posibilidades y variaciones de lo que podemos realizar con una lista, pudiendo así poder manejar de distintas formas y que siempre tengan un tamaño dinámico para mejorar y optimizar el uso en memoria de las mismas.

## Referencias bibliográficas


- C. J. Date, (1991). *An introduction to Database Systems.* Addison-Wesley Publishing Company, Inc.
- Craig Larman; Introducción al análisis y diseño orientado a objetos. Prentice Hall
- Fundamentos de programación, Algoritmos, Estructuras de Datos y Objetos, Luis Joyanes Aguilar, Cuarta Edición, McGraw-Hill
- https://docs.python.org/es/3.10/library/xml.dom.minidom.html
- Python para informáticos Versión 2.7.2 Charles Severance