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 2024

SISTEMA DE MANEJO Y VISUALIZACION DE PISOS DE LA EMPRESA PISOS GUATEMALA S.A

202100543 – Harold Benjamin Oxlaj Mangandi

Resumen

Por parte de la empresa “Pisos Guatemala S.A.”, se desea optimizar el costo por operación de un robot especializado capaz de colocar azulejos para la creación de pisos de dimensiones R x C. Cada azulejo tiene un lado blanco y uno negro, por lo que el robot es capaz de voltear azulejos uno a uno o intercambiar azulejos vertical u horizontalmente.

El proyecto presentado se basa en la implementación de un sistema que permite cargar un archivo XML con los datos de los pisos que se desean construir, así como los patrones de azulejos para cada piso y la cantidad de azulejos por piso, y el costo por voltear o intercambiar un azulejo. Se incluye la visualización gráfica del piso por construir y, por último, también se implementó un algoritmo de optimización para minimizar el costo por operación del robot de la empresa.

Palabras clave
Objetos, Listas, Optimización, Algoritmos, Clases.

Abstract

The company "Pisos Guatemala S.A." wants to optimize the cost per operation of a specialized robot capable of laying tiles for the creation of floors of dimensions R x C. Each tile has a white side and a black side, so the robot is able to flip tiles one by one or exchange tiles vertically or horizontally.

The project is based on the implementation of a system that allows loading an XML file with the data of the floors to be built, as well as the tile patterns for each floor, the quantity of tiles per floor, and the cost of flipping or swapping a tile. It includes the graphical visualization of the floor to be built, and lastly, an optimization algorithm was also implemented to minimize the operating cost of the company's robot.

Keywords
Objects, Linked Lists, Optimization, Algorithms, Classes

Introducción

La construcción de pisos utilizando tecnología avanzada es una idea innovadora que facilita enormemente el trabajo, aunque resulta algo costosa en términos económicos. Por ello, la optimización de los costos es fundamental en una empresa. Una de estas empresas es “Pisos Guatemala S.A.”, que, al tener un robot especializado en construir patrones para pisos, incurre en un costo por cada movimiento de los azulejos al cambiar el patrón del piso original.

El proyecto aborda la solución al problema de la optimización mediante el uso de listas enlazadas y condicionales que deben cumplirse para lograr el costo mínimo. La utilización de listas enlazadas fue vital para la construcción del proyecto.

Desarrollo del tema

Un sistema a pesar de tener toda una lógica interna compleja debe ser simple y sencillo de entender para el usuario final con esto se garantiza que el usuario esté cómodo y pueda usarlo de manera correcta.

El primer acercamiento al usuario con el sistema será un Menú sencillo en el que se le dará la bienvenida y dos opciones, cargar un archivo XML al sistema o salir de este.

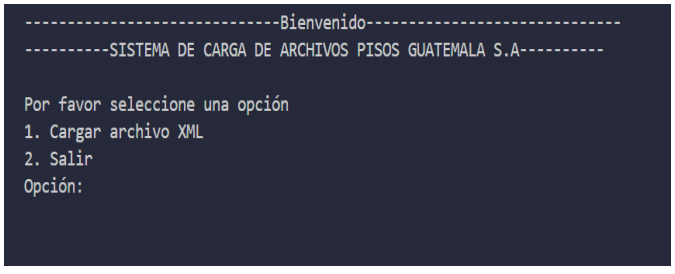

Figura 1. Menú principal

Fuente: elaboración propia 2024.

Los datos del archivo son guardados en listas enlazadas

a. Lista Enlazada para Pisos

Los datos son guardados en Objetos de tipo Piso y estos a su vez son guardados en una Lista Enlazada de Pisos con varios métodos; agregar un elemento a la lista, verificar si está vacía, imprimir los nombres de los pisos guardados y verificar por medio del nombre si el piso está disponible.

```
class Piso:
    def __init__(self, nombre, filas, columnas,flip, swap, patrones):
        self.nombre = nombre
        self.filas = filas
        self.columnas = columnas
        self.flip = flip
        self.swap = swap
        self.patrones = patrones
        self.siguiente = None
```

Figura 2. Objeto Piso

Fuente: elaboración propia 2024.

b. Lista Enlazada para Patrones

Los datos son guardados en Objetos de tipo Patrones y estos son agregados a una Lista Enlazada de Patrones para finalmente ser agregada a la Lista Enlazada de los pisos disponibles cada piso cuenta con su propia Lista Enlazada de Patrones, cuenta con varios métodos; agregar un elemento a la lista, verificar si está vacía, imprimir los códigos de los patrones guardados y verificar por medio del Código si el patrón está disponible.

```
class Patrones:
    def __init__(self, codigo, azulejos):
        self.codigo = codigo
        self.azulejos = azulejos
        self.siguiente = None
```

Figura 3. Objeto Patrones

Fuente: elaboración propia 2024.

c. Lista Enlazada para Azulejos

Los datos son guardados en objetos de tipo Azulejo, los cuales son agregados a una lista enlazada de Azulejos. Posteriormente, son incorporados a una respectiva lista enlazada de Patrones y, finalmente, agregados a la lista de pisos disponibles. La lista cuenta con varios métodos: agregar un elemento a la lista, verificar si está vacía, imprimir los azulejos en forma matricial en consola, copiar una lista enlazada a otra, modificar los datos de los azulejos y aplicar un algoritmo de ordenamiento

```
class Azulejo:
    def __init__(self, color):
        self.color = color
        self.siguiente = None

costo_total= None
```

Figura 4. Objeto Azulejo

Fuente: elaboración propia 2024.

- Algoritmo de Ordenamiento

Para la realización de este algoritmo fue necesario utilizar muchas condicionales. Primero, se evalúa si los azulejos en ambos patrones son iguales o distintos. Si son iguales, se procede al siguiente azulejo. Si no, se entra en otra condicional: si al índice del azulejo se le añade una unidad y este no es múltiplo del número de columnas del piso, se evalúan otros posibles movimientos. Uno es el intercambio por la derecha, aplicable si el siguiente azulejo en el patrón original es igual al azulejo actual en el patrón a convertir y el azulejo siguiente en el patrón a convertir es igual al azulejo actual en el patrón original; en este caso, se realiza el movimiento. Esta misma lógica se aplica al azulejo inferior al actual, si es que existe, comprobando si es igual y procediendo al cambio si es necesario. Si ninguna de estas condiciones se cumple, se procede a voltear el azulejo, continuando así hasta finalizar el patrón. Si al índice del azulejo se le añade una unidad y resulta ser múltiplo del número de columnas del piso, se elimina la posibilidad de intercambio por la derecha, ya que ese movimiento no sería posible, pero se sigue aplicando la misma lógica para los demás movimientos recorra todos los azulejos.

```
def movement_str(self, patron_objetivo,filas, columnas, costo_flip, costo_swap):
    if not self.esta_vacia():
        num_paso = 0
        aux_patron = self.copiar(self)
        objetivo_azulejo_actual = patron_objetivo.primero
        azulejo_actual = aux_patron.primero
        print("\nPatron inicial:\n")
        aux_patron.imprimir(filas, columnas)
        print("\n")
        indice = 0
        global costo_total
        costo_total = 0
        while azulejo_actual is not None and objetivo_azulejo_actual is not None:
            if azulejo_actual.color == objetivo_azulejo_actual.color:
                azulejo_actual= azulejo_actual.siguiente
                objetivo_azulejo_actual = objetivo_azulejo_actual.siguiente
                indice+=1
            elif azulejo_actual.color != objetivo_azulejo_actual.color:
                if (indice+1) % (int(columnas)) == 0:
```

Figura 5. Algoritmo de optimización parte 1

Fuente: elaboración propia 2024.

```
if (indice+1) % (int(columnas)) == 0:
    if aux_patron.get(indice+int(columnas)) == objetivo_azulejo_actual.color \
        and patron_objetivo.get(indice+int(columnas)) == azulejo_actual.color:
        num_paso+=1
        aux_patron.modificar(indice, objetivo_azulejo_actual.color)
        aux_patron.modificar(indice+int(columnas), patron_objetivo.get(indice+int(columnas)))
        costo_total+=costo_swap
        print(f"Paso {num_paso}, Intercambio por la parte inferior, Costo: Q{costo_swap}, COSTO ACUMULADO: {costo_total}" )
        print(f"Entre el Azulejo {indice+1} y el {indice+int(columnas)+1}")
        aux_patron.imprimir(filas, columnas)
        azulejo_actual= azulejo_actual.siguiente
        objetivo_azulejo_actual = objetivo_azulejo_actual.siguiente
        indice+=1
        if indice == int(filas)*int(columnas):
            return aux_patron
    else:
        num_paso+=1
        if azulejo_actual.color == "B":
            aux_patron.modificar(indice, "N")
        elif azulejo_actual.color == "N":
            aux_patron.modificar(indice, "B")
        costo_total+=costo_flip
        print(f"Paso {num_paso}, Volteo, Costo: Q{costo_flip}, , COSTO ACUMULADO: {costo_total}")
        print(f"Azulejo {indice+1}")
        aux_patron.imprimir(filas, columnas)
        azulejo_actual= azulejo_actual.siguiente
        objetivo_azulejo_actual = objetivo_azulejo_actual.siguiente
        indice+=1
        if indice == int(filas)*int(columnas): (variable) objetivo_azulejo_actual: Any
            return
```

Figura 6. Algoritmo de optimización parte 2

Fuente: elaboración propia 2024.

```
elif azulejo_actual.siguiente.color == objetivo_azulejo_actual.color \
    and azulejo_actual.color == objetivo_azulejo_actual.siguiente.color:
    num_paso+=1
    aux_patron.modificar(indice, objetivo_azulejo_actual.color)
    aux_patron.modificar(indice+1, azulejo_actual.siguiente.color)
    costo_total+=costo_swap
    print(f"Paso {num_paso}, Intercambio por la derecha, costo: {costo_swap}, , COSTO ACUMULADO: {costo_total}")
    print(f"Entre el Azulejo {indice+1} y el {indice+2}")
    aux_patron.imprimir(filas, columnas)
    azulejo_actual= azulejo_actual.siguiente
    objetivo_azulejo_actual = objetivo_azulejo_actual.siguiente
    indice+=1
elif aux_patron.get(indice+int(columnas)) == objetivo_azulejo_actual.color \
    and patron_objetivo.get(indice+int(columnas)) == azulejo_actual.color:
    num_paso+=1
    aux_patron.modificar(indice, objetivo_azulejo_actual.color)
    aux_patron.modificar(indice+int(columnas), patron_objetivo.get(indice+int(columnas)))
    costo_total+=costo_swap
    print(f"Paso {num_paso}, Intercambio por la parte inferior, Costo: Q{costo_swap}, , COSTO ACUMULADO: {costo_total
    print(f"Entre el Azulejo {indice+1} y el {indice+int(columnas)+1}")
    aux_patron.imprimir(filas, columnas)
    azulejo_actual= azulejo_actual.siguiente

    objetivo_azulejo_actual = objetivo_azulejo_actual.siguiente
    indice+=1
```

Figura 7. Algoritmo de optimización parte 3

Fuente: elaboración propia 2024.

Al cargarse un archivo XML correctamente se le enviará a otro submenú en donde podrá elegir entre mostrar todos los nombres de los pisos y sus respectivos patrones ir al menú principal o seleccionar un piso junto con un patrón.

```
Por favor seleccione una opción
1. Cargar archivo XML
2. Salir
Opción: 1
Ingrese el nombre del archivo XML en el escritorio: archivo_prueba.xml

Archivo Cargado Con exito.


Por favor seleccione una opción
1. Seleccionar Piso
2. Mostrar Todos los pisos
3. Salir
Opción:
```

Figura 8. Sub-Menú 1

Fuente: elaboración propia 2024.

Al seleccionar “Mostrar todos los pisos” se mostrarán los pisos en consola con sus respectivos códigos de patrones

```
Nombre: bggob
codigo patron: cUUVMXkrhN
codigo patron: prCs1CFZ5E
codigo patron: wnnRGXyavO

Nombre: evzex
codigo patron: CHRQ3PWia6
codigo patron: mGPI8lvtT6
codigo patron: yrScrC01Ec

Nombre: kvblc
codigo patron: FySLa2EZhe
codigo patron: rzLAsL4Mci
codigo patron: zcNsaFPPmR

Nombre: lujqf
codigo patron: 0xwzlO5SrS
codigo patron: gKgYdtAgmU
codigo patron: uyYJvPnI19
codigo patron: wxxOJiorTT
```

Figura 9. Mostrar Pisos

Fuente: elaboración propia 2024

Al querer seleccionar un piso se debe ingresar su nombre y algún patrón que se desee visualizar. A continuación, se mostrará un segundo submenú con tres opciones: Mostrar gráficamente el piso y patrón seleccionado, convertir el patrón a otro que posea el piso, o volver al submenú 1

```
Por favor seleccione una opción
1. Seleccionar Piso
2. Mostrar Todos los pisos
3. Salir
Opción: 1

Seleccione un piso por medio del nombre: yuxzh

¡Piso Encontrado con exito!

Seleccione un patron por medio de su codigo: 0Fz0v9AeqH

¡Patron Encontrado con exito!

Seleccione una opción para el patron:
1. Mostrar graficamente
2. Convertir patron a uno nuevo
4. Salir
Opción:
```

Figura 10. Submenú 2

Fuente: elaboración propia 2024

Al querer ver los patrones gráficamente se generará un archivo .pdf con la vista grafica del patrón utilizando la librería de graphiz para graficar los azulejos simulando nodos y conforme a las filas y columnas del piso deseado se forma la gráfica utilizando ciclos de i filas anidado en j columnas.

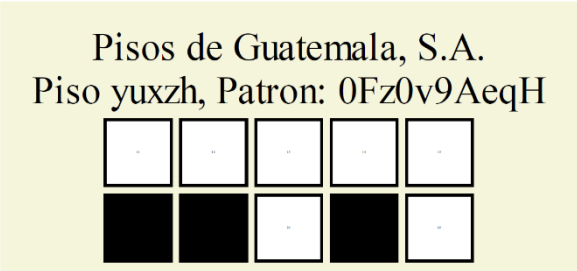

Figura 11. Vista Grafica de un patrón

Fuente: elaboración propia 2024

Al querer pasar de un patrón original a otro se necesita ingresar el código de otro patrón, luego de esto se generaran los pasos que el algoritmo siguió para llegar a ese patrón objetivo y junto con el costo total se grafica el patrón final.

```
Paso 1, Volteo, Costo: Q13, , COSTO ACUMULADO: 13
Azulejo 3
[B] [B] [N] [B] [B]
[N] [N] [B] [N] [B]


Paso 2, Volteo, Costo: Q13, , COSTO ACUMULADO: 26
Azulejo 4
[B] [B] [N] [N] [B]
[N] [N] [B] [N] [B]


Paso 3, Volteo, Costo: Q13, , COSTO ACUMULADO: 39
Azulejo 6
[B] [B] [N] [N] [B]
[B] [N] [B] [N] [B]


EL COSTO TOTAL ES: Q39
```

Figura 12. Pasos seguidos por el algoritmo

Fuente: elaboración propia 2024

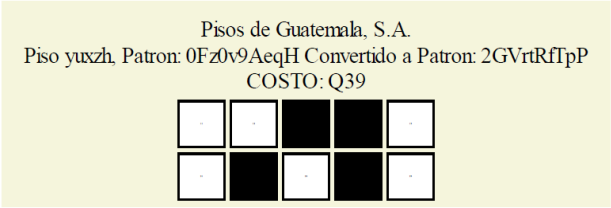

Figura 13. Grafica del patrón final

Fuente: elaboración propia 2024

Tanto las gráficas individuales como las de conversión de patrón inicial a final se guardan en la carpeta del proyecto.

Conclusiones

La implementación de listas enlazadas puede ser de gran utilidad y eficacia al ser dinámicos.

Las Listas Enlazadas son de gran ayuda para comprender y manejar la utilización de memoria del dispositivo.

Los algoritmos de optimización dependen de la lógica y estrategia del programador

Referencias bibliográficas

Facultad de Estadística e Informática.(n.d). *Listas ligadas o enlazadas: Fundamentos teóricos y aplicaciones* [PDF]. Recuperado el [15/02/2024], de https://www.uv.mx/personal/ermeneses/files/2021/08/Clase5-ListasEnlazadasFinal.pdf

Miller, B. N., & Ranum, D. L. (n.d.). *3.21. Implementación de una lista no ordenada: Listas enlazadas. Solución de problemas con algoritmos y estructuras de datos usando Python*. Recuperado el [13/02/2024], de https://runestone.academy/ns/books/published/pythoned/BasicDS/ImplementacionDeUnaListaNoOrdenadaListasEnlazadas.html

Salcedo, L. (2018, 1 de julio). *Linked List: Listas enlazadas - Implementación en Python*. Mi Diario Python. Recuperado el [13/02/2024], de https://pythondiario.com/2018/07/linked-list-listas-enlazadas.html

Anexos

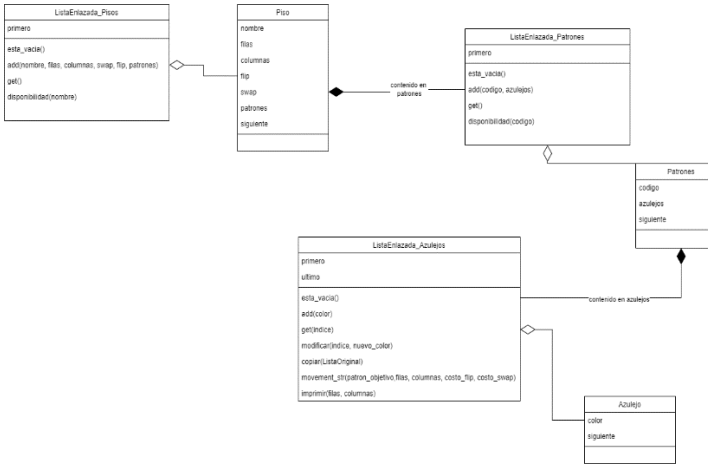

Figura 14. Diagrama de Clases

Fuente: elaboración propia 2024.