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 2021.

PROYECTO NO. 1

202004804 – José Andrés Montenegro Santos

Resumen

En el desarrollo de este proyecto se hizo uso del lenguaje de programación Python y del IDE Visual Studio Code para la elaboración de un programa en consola que leyera un archivo en formato xml (para esto se hizo uso de la librería ElementTree) y que posteriormente almacenara la información leída del documento en diferentes estructuras de datos, las cuáles incluyen listas simple y doblemente enlazadas y listas ortogonales. Posteriormente haciendo uso de listas de prioridad se realizó un algoritmo de búsqueda el cual encuentra la ruta con menor consumo de combustible en un determinado mapa (desde un nodo inicial hasta un nodo final indicado por el usuario), al finalizar esto despliega un mensaje en consola el cuál informa al usuario del gasto total de combustible que conlleva la ruta encontrada, así como de una representación de la matriz donde se resalta el corrido sugerido. Finalmente, se hizo uso de la herramienta Graphviz para elaborar una imagen de la matriz ingresada.

Palabras clave

- Cola de prioridad
- Nodo
- Puntero
- Ordenamiento
- Iteración

Abstract

In the development of this project the programming language Python and the Visual Studio Code IDE was used in the elaboration of a console program that could read a file in xml format (for this was used the ElementTree library) and later save the information read from the document in data structures, which include singly, doubly and orthogonal linked lists. Later making use of a priority queue was made a search algorithm which finds the route with less possible fuel consume y a determinate map (from an initial node to a final one indicated by the user), at the end unfolds a message in console to inform the user the total amount of fuel to use in the path found, as well as a representation of the array were the suggested route is highlighted. Finally use was made of Graphviz tool to elaborate an image of the entered array.

Keywords

- *Priority queue*
- *Node*
- *Pointer*
- *Ordering*
- *Iteration*

Introducción

El objetivo de este proyecto fue el de elaborar un programa el cuál emulara el comportamiento de un software de consola que recibe como entrada un mapa representado por una matriz escrita en una archivo en formato xml, posteriormente es capaz de hacer uso de estructuras de datos tales como listas simple y doblemente enlazadas además de listas ortogonales para almacenar esta información y, seguidamente hacer uso de un algoritmo para encontrar el camino más corto desde un nodo inicial a otro final para finalmente informarle al usuario los resultados los cuáles comprenden el valor total de combustible del camino determinado, una representación en consola de la matriz donde se detalla el camino a seguir y, finalmente, un gráfico del terreno que se procesó.

Desarrollo del tema

Clase MAIN:

- **Función menuPrincipal():**

```
def menuPrincipal():

    print("Menu Principal:")
    print("\t1. Cargar Archivo")
    print("\t2. Procesar Archivo")
    print("\t3. Escribir Archivo de salida")
    print("\t4. Mostrar Datos del estudiante")
    print("\t5. Generar gráfica")
    print("\t6. Salida")
    print("")
    print("Escriba la opción que desea realizar: ")
    respuesta = input()

    if int(respuesta) == 1:···
    elif int(respuesta) == 2:···
    elif int(respuesta) == 3:···
    elif int(respuesta) == 4:···
    elif int(respuesta) == 5:···
    elif int(respuesta) == 6:···
```

Figura 1. Función menuPrincipal().
Fuente: elaboración propia, 2020.

El objetivo del desarrollo de esta función fue el que esta sea mostrada al usuario cada vez que este ejecuta el programa, recibe la entrada que este debe ingresar (la cuál corresponde al número de una de las opciones disponibles) y posteriormente ejecuta la función que corresponde al número ingresado.

```
if int(respuesta) == 1:
    carga()
```

Figura I1. Función menuPrincipal() opción 1.
Fuente: elaboración propia, 2020.

Cuando el usuario ingresa el número correspondiente a la opción número uno se ejecuta la función **carga().**

```
if int(respuesta) == 1:
    carga()
elif int(respuesta) == 2:
    global terreno_actual
    global terreno_res
    print("Ingrese el nombre del terreno a procesar")

    #Obtención de la entrada del usuario
    terreno_res = input()
    nodo_terreno = terrenos.verificarTerreno(terreno_res)
    if nodo_terreno == None:
        print("El terreno que usted ha ingresado no se encuentra en el programa")
    terreno_actual = nodo_terreno.getMapa()
    #Algoritmo para encontrar el camino más corto
    print("************************************************")
    print(" CALCULANDO MEJOR RUTA Y SU COSTE DE COMBUSTIBLE ")
    print("************************************************")
    print("")
    print("")
    lista = Lista_prioridad()
    nodo_actual = terreno_actual.obtenerNodo(nodo_terreno.inicioX, nodo_terreno.inicioY)
    camino.insertarNodo(nodo_actual.nodo)
    while ((nodo_actual.nodo.x != nodo_terreno.finalX) or (nodo_actual.nodo.y != nodo_terreno.finalY)):
        vecinos = terreno_actual.obtenerHermanos(nodo_actual.nodo.x, nodo_actual.nodo.y)
        if vecinos.inicio is not None:
            puntero = vecinos.inicio
            while puntero is not None:
                puntero.nodo.setData(nodo_actual)
                lista.insertar(puntero.nodo)
                puntero = puntero.siguiente
        nodo_actual.nodo.visitado = True
        nodo_actual = lista.pop()
        camino.insertarNodo(nodo_actual.nodo)
    print("************************************************")
    print("          RUTA ENCONTRADA EXITOSAMENTE          ")
    print("************************************************")
    print("")
    print("EL COSTO DE COMBUSTIBLE DE LA RUTA MÁS ÓPTIMA ES: ")
    print("Coordenadas    x: ", nodo_actual.nodo.x , "y: ", nodo_actual.nodo.y, "Gasto de combustible: ", nodo_actual.nodo.data)
    print("")
    print("")
```

Figura 1II. Función menuPrincipal() opción 2.
Fuente: elaboración propia, 2020.

Al momento de presionar la opción número dos, se desplegará un mensaje en consola donde se le indicará al usuario que ingrese el nombre del terreno al que quiere que le sea encontrado el camino más corto, posteriormente se iniciará el algoritmo de búsqueda. Este algoritmo inicia con la función **obtenerNodo()** a la cual se le deben ingresar por parámetro las coordenadas del nodo inicial posteriormente comprobará si el nodo existe y si es así lo regresará como respuesta. Después se ingresa el nodo de partida en una lista que se llama **camino** y finalmente el nodo actual ingresa a un ciclo **while**, dentro de este ciclo lo que se realiza es que usa una función llamada **obtenerHermanos()**, la cual se le

envía como parámetros las coordenadas del nodo actual y posteriormente retorna como respuesta una lista con todos los nodos que son adyacentes al nodo actual siempre y cuando alguno de estos no haya sido visitado. Sin embargo, debido a esta restricción y al hecho de que debido a la posición de algunos nodos no todos van a tener 4 nodos adyacentes cabe la posibilidad de que la función regrese un valor nulo, es por esto por lo que después de obtener los nodos adyacentes al nodo actual hay una condicional **if,** la cual comprueba si la respuesta de esta función no fue nula, si no lo fue, entonces añade todos los nodos adyacentes retornados a una variable **lista** la cuál ordena todos los nodos en su interior de menor a mayor tomando como parámetro el gasto de combustible que conlleva cada uno finalmente se marca el nodo actual como visitado para que la función no lo vuelva a regresar como adyacente a otro nodo y se le asigna un nuevo valor al nodo actual el cuál es el primero que esté en la variable **lista** ya que este será el de menor gasto de combustible y se ingresa este nodo a una lista donde se guardan todos los nodos que recorrió el programa. Este proceso se repite hasta que se alcance el nodo destino. Para la impresión del camino en la matriz en consola se usa otro ciclo, en el cuál se busca el nodo destino y se va obteniendo su nodo padre (el cuál es el nodo que le sumó una menor cantidad de combustible) y esto se hace para todos los nodos hasta llegar al nodo de inicio estos se van guardando en una lista que los ordena en forma de pila llamada **ruta,** finalmente haciendo uso de las dimensiones de la matriz se imprime la misma en consola, comprobando cuáles de las posiciones están en la lista **ruta** y marcando las mismas con un número 1 y en rojo para formar el camino recorrido.

```
print("Escriba la ruta de onde quiere guardar el archivo")
ruta_archivo = input()
#Ciclos para obtener el recorrido que realizó el algoritmo
ultimo = camino.inicio
ruta = ListaCamino()
while ultimo.siguiente is not None:
    ultimo = ultimo.siguiente
combustible = ultimo.nodo.data
while ultimo != None:
    ultimo.nodo.resetData()
    ruta.insertarNodo(ultimo.nodo)
    ultimo = ultimo.nodo.nodoPadre

#Creación de la estructura xml con los datos de salida
nodo_terreno = terrenos.verificarTerreno(terreno_res)
raiz = ET.Element('terreno', nombre = nodo_terreno.dato)
inicio = ET.SubElement(raiz, "posicioninicio")
ET.SubElement(inicio, "x").text = str(nodo_terreno.inicioX)
ET.SubElement(inicio, "y").text = str(nodo_terreno.inicioY)
final = ET.SubElement(raiz, "posicionfin")
ET.SubElement(final, "x").text = str(nodo_terreno.finalX)
ET.SubElement(final, "y").text = str(nodo_terreno.finalX)
ET.SubElement(raiz, "combustible").text = str(combustible)
puntero = ruta.inicio
while puntero is not None:
    ET.SubElement(raiz, "posicion", x = str(puntero.nodo.x), y = str(puntero.nodo.y)).text = str(puntero.nodo.data)
    puntero = puntero.siguiente
archivo = ET.tostring(raiz, "unicode", "xml")
xml1 = xml.dom.minidom.parseString(archivo)
xml2 = xml1.toprettyxml()

#Escritura del archivo xml
try:
    miArchivo = open(ruta_archivo + "\salida.xml", "w")
    miArchivo.write(xml2)
    print("El archivo se ha escrito satisfactoriamente")
except:
    print("Ocurrió un error al escribir el archivo, por favor inténtelo de nuevo")
```

Figura 1V. Función menuPrincipal() opción 3.
Fuente: elaboración propia, 2020.

Al seleccionar la opción 3 se escribirá el archivo de salida, el cuál será uno de extensión xml con el siguiente formato

```
<?xml version="1.0" ?>
<terreno nombre="terreno1">
    <posicioninicio>
        <x>1</x>
        <y>1</y>
    </posicioninicio>
    <posicionfin>
        <x>5</x>
        <y>5</y>
    </posicionfin>
    <combustible>79</combustible>
    <posicion x="1" y="0">17</posicion>
    <posicion x="2" y="0">17</posicion>
    <posicion x="2" y="1">17</posicion>
    <posicion x="2" y="2">23</posicion>
    <posicion x="2" y="3">26</posicion>
    <posicion x="2" y="4">28</posicion>
    <posicion x="3" y="4">30</posicion>
    <posicion x="4" y="4">38</posicion>
    <posicion x="5" y="4">42</posicion>
    <posicion x="6" y="4">55</posicion>
    <posicion x="6" y="5">57</posicion>
    <posicion x="6" y="6">67</posicion>
    <posicion x="6" y="7">72</posicion>
    <posicion x="7" y="7">77</posicion>
    <posicion x="7" y="8">79</posicion>
</terreno>
```

Figura V. Archivo xml de salida.
Fuente: elaboración propia, 2020.

Para realizar esto se usó la librería ElementTree de Python, donde se añadieron los valores de inicio, fin y combustible por medio de variables globales o usando la lista **camino**. El camino recorrido se añadió recorriendo la lista **ruta.**

Al seleccionar la opción número 5 se ejecutará la función **generarGrafico()** el cuál usa la herramienta graphviz para generar un grafo que representa a la matriz ingresada por el usuario.

Figura VI. Función menuPrincipal() opción 5.
Fuente: elaboración propia, 2020.

Esta función construye el código necesario para generar el diagrama usando graphviz, primeramente, recorre toda la matriz seleccionada por el usuario y crea un vertice del diagrama por cada uno de los nodos de la matriz, identificándolos con un nodo único y asignándoles un grupo. Después de haber creado todos los nodos se recorre la matriz horizontalmente, esto con el objetivo de enlazar todas las filas del gráfico y posteriormente hacer **Rank (**función cuya función es que los nodos se visualicen horizontalmente), posterior a esto se recorre la matriz verticalmente enlazando los vértices del diagrama en esta dirección. Al finalizar de realizar esto se tendrá un string con todo el código del gráfico de la matriz.

```
grafico = grafico + '''}
}'''
miArchivo = open('graphviz.dot', 'w')
miArchivo.write(grafico)
miArchivo.close()
system('dot -Tpng graphviz.dot -o graphviz.png')
system('cd ./graphviz.png')
startfile('graphviz.png')
```

Figura VII. Escritura de archivo .dot y renderización a una imagen
Fuente: elaboración propia, 2020.

Para poder culminar el proceso de graficado se genera un archivo nuevo con extensión **.dot**, y se escribe en el string que contiene el código generado con anterioridad. Finalmente se hace uso de la función **System()** para escribir directamente en el cmd del computador el código necesario para renderizar el archivo **.dot** y convertirlo a uno **.png** y mostrárselo al usuario.

Función Carga():

Figura VIII. Función carga().
Fuente: elaboración propia, 2020.

Esta es la función de lectura, la cuál se encarga de procesar y almacenar toda la información que se

obtenga del archivo xml que ingrese el usuario. Primeramente, esta función solicita por medio de un mensaje en consola que el usuario ingrese la ruta de la ubicación del archivo de entrada. Si se presenta algún error al momento de la lectura de este se notificará al usuario de esto. De lo contrario se procederá a la extracción de datos del archivo, primero se creará un nodo al cuál se le mandará por parámetro el nombre del terreno ingresado, posteriormente con un ciclo for se accederán a los valores **m** y **n** de este documento y de igual manera se almacenarán en el nodo donde se encuentra el nombre del terreno. A igual que para los parámetros anteriores, se usará un ciclo for para obtener los valores de x y y iniciales y finales, los cuáles de igual manera se almacenarán en el nodo donde se encuentra el nombre del terreno. Finalmente para construir la matriz se hará uso de la función **agregarNodo()** de la clase **Matriz** y se le mandará como parámetro las coordenadas en x y y, además del valor de combustible de cada uno de los nodos. Para concluir al tener almacenados y ordenados los datos que pertenecen a la matriz, esta se almacenará en otra variable dentro del nodo donde se encuentra el nombre del terreno. Este proceso se repetirá para todos los demás terrenos.

- **Clase Matriz:**

Esta clase se creó para hacer posible el representar los datos ingresados por el usuario como un conjunto de nodos ortogonales que forman parte de un mapa. La matriz que contiene estos datos posee como atributos dos listas, una lista que sirve para llevar el control de las componentes en x de los nodos y otra lista que lleva el control de los componentes en y.
Una de las funciones más importantes de esta clase es la función **agregarNodo().**

Función agregarNodo():

```
def agregarNodo(self, dato, x, y):
    nodo_nuevo = NodoM(dato, x, y)
    encabezado_fila = self.cabecera_filas.buscar(y)

    if encabezado_fila == None:
        encabezado_fila = NodoF(y)
        encabezado_fila.acceso = nodo_nuevo
        self.cabecera_filas.nuevoNodo(encabezado_fila)
    else:
        if nodo_nuevo.x < encabezado_fila.acceso.x:
            nodo_nuevo.derecha = encabezado_fila.acceso
            encabezado_fila.acceso.izquierda = nodo_nuevo
            encabezado_fila.acceso = nodo_nuevo
        else:

            puntero = encabezado_fila.acceso
            while puntero.derecha is not None:
                if nodo_nuevo.x < puntero.derecha.x:
                    nodo_nuevo.derecha = puntero.derecha
                    nodo_nuevo.izquierda = puntero
                    puntero.derecha.izquierda = nodo_nuevo
                    puntero.derecha = nodo_nuevo
                    break
                puntero = puntero.derecha

            if puntero.derecha is None:
                puntero.derecha = nodo_nuevo
                nodo_nuevo.izquierda = puntero
```

Figura IX. Función agregarNodo().
Fuente: elaboración propia, 2020.

Esta función recibe como parámetro el valor del combustible de un nodo en específico y sus coordenadas x y y, posteriormente hace uso de la función **buscar()** para verificar si en la lista ya se encuentra registrada la fila que corresponde a la coordenada en y que le pertenece al nodo a agregar. Si no existe entonces crea esta nueva fila (ordenándose de menor a mayor junto a los números de fila que ya existan) y finalmente se inserta el nodo en este número de fila.
Si el número de fila ya existe significa que ya hay otros nodos en la fila en al cuál se desea insertar el nuevo nodo, por lo que al momento de insertarlo en su fila correspondiente se ordena de menor a mayor junto a los nodos con el mismo valor en y.

```
encabezado_columna = self.cabecera_columnas.buscar(x)

if encabezado_columna == None:
    encabezado_columna = NodoC(x)
    encabezado_columna.acceso = nodo_nuevo
    self.cabecera_columnas.nuevoNodo(encabezado_columna)
else:
    if nodo_nuevo.y < encabezado_columna.acceso.y:
        nodo_nuevo.abajo = encabezado_columna.acceso
        encabezado_columna.acceso.arriba = nodo_nuevo
        encabezado_columna.acceso = nodo_nuevo
    else:
        puntero = encabezado_columna.acceso
        while puntero.abajo is not None:
            if nodo_nuevo.y < puntero.abajo.y:
                nodo_nuevo.abajo = puntero.abajo
                nodo_nuevo.arriba = puntero
                puntero.abajo.arriba = nodo_nuevo
                puntero.abajo = nodo_nuevo
                break
            puntero = puntero.abajo
        if puntero.abajo == None:
            puntero.abajo = nodo_nuevo
            nodo_nuevo.arriba = puntero
```

Figura X. Función agregarNodo() parte 2.
Fuente: elaboración propia, 2020.

En la segunda parte de este proceso se repite el mismo procedimiento que en la parte anterior de la función, pero con las coordenadas en x, primeramente se verifica si el valor de x que corresponde al nodo a insertar ya existe, si no existe se crea y posteriormente se inserta al nodo nuevo en su columna correspondiente, si ya existe igualmente se inserta en su columna correspondiente sin embargo, al hacerlo se ordena de menor a mayor con los nodos que ya se encuentran ubicados en ese valor de x.

Función obtenerHermanos():

```
def obtenerHermanos(self, x, y):
    hermanos = ListaNodos()
    puntero = self.cabecera_filas.primero.acceso
    while puntero is not None:
        if puntero.y == y:
            while puntero is not None:
                if puntero.x == x:
                    if puntero.arriba is not None and puntero.arriba.visitado is not True:
                        hermanos.insertarNodo(puntero.arriba)
                    if puntero.abajo is not None and puntero.abajo.visitado is not True:
                        hermanos.insertarNodo(puntero.abajo)
                    if puntero.derecha is not None and puntero.derecha.visitado is not True:
                        hermanos.insertarNodo(puntero.derecha)
                    if puntero.izquierda is not None and puntero.izquierda.visitado is not Tru
                        hermanos.insertarNodo(puntero.izquierda)
                    return hermanos
                puntero = puntero.derecha
        puntero = puntero.abajo
    return None
```

Figura X1. Función obtenerHermanos().
Fuente: elaboración propia, 2020.

Esta función forma una parte fundamental del programa, ya que su propósito es el de uno de los pasos más importantes en el algoritmo para encontrar la ruta más optima. Recibe como parámetro las coordenadas de un nodo en la matriz y, posteriormente devuelve una lista con todos los nodos adyacentes a el nodo con las coordenadas enviadas. Sin embargo, tiene ciertas condiciones que cumplir ya que debido a la posición de algunos nodos puede que alguno de los cuatro punteros que posee sea nulo, por lo que no regresa estos valores, tampoco regresa nodos que ya hayan sido marcados como “visitados” por el algoritmo de búsqueda, para que este no pueda dar vueltas en círculos y por ende caer en ser in ciclo infinito.

Función obtenerNodo():

```
def obtenerNodo(self, x, y):
    puntero = self.cabecera_filas.primero.acceso
    while puntero is not None:
        if puntero.y == y:
            while puntero is not None:
                if puntero.x == x:
                    nodo = NodoP(puntero)
                    return nodo
                puntero = puntero.derecha
        puntero = puntero.abajo
    return None
```

Figura X1I. Función obtenerNodo().
Fuente: elaboración propia, 2020.

Esta función realiza algo similar a la función **obtenerHermanos()** ya que al igual que esta recibe como parámetros las coordenadas de un nodo, sin embargo esta función retorna solamente el nodo que posee las coordenadas ingresadas.

- **Clase columnas y Clase filas:**

```python
class columnas:
    def __init__ (self):
        self.primero = None
        self.ultimo = None

    def nuevoNodo(self, nodo):
        if self.primero == None:
            self.primero = nodo
            self.ultimo = nodo
        else:
            if nodo.x < self.primero.x:
                nodo.derecha = self.primero
                self.primero.izquierda = nodo
                self.primero = nodo
            else:
                puntero = self.primero
                while puntero.derecha is not None:
                    if nodo.x < puntero.derecha.x:
                        nodo.izquierda = puntero
                        nodo.derecha = puntero.derecha
                        puntero.derecha = nodo
                        puntero.derecha.izquierda = nodo
                        break
                    puntero = puntero.derecha
                if puntero.derecha == None:
                    puntero.derecha = nodo
                    nodo.izquierda = puntero
                    self.ultimo = nodo
```

Figura XII1. Clase Columnas).
Fuente: elaboración propia, 2020.

Esta clase es utilizada para cuando se desea insertar un nodo en la matriz cuya coordenada en x o y no ha sido creada todavía, de hacer esta verificación se encarga la función **agregarNodo()** mencionada anteriormente. Posteriormente al insertar el número de fila o columna deseado este se ordena de menor a mayor con los demás números que estén en su respectiva lista (lista de números de filas o listas de números de columnas).

- **Clase Lista:**

Debido a la gran variedad de datos que se estuvo manejando a lo largo del desarrollo del programa se crearon varias clases con diferentes tipos de listas que fueran capaces de poder transportar estos datos. Por lo que en el módulo **Listas** se tiene 4 diferentes tipos de listas:

```python
class Lista:
    def __init__(self):
        self.inicio = None

    def insertar(self, dato):...

    def verificarTerreno(self, terreno):...

    def asignarMapa(self, terreno, mapa):...

    def asignarMapaAux(self, terreno, mapa):...

    def recorrer(self):...

    def vacia(self):...

class ListaNodos:
    def __init__ (self):...

    def insertarNodo(self, nodo):...

    def recorrer(self):...

class ListaCamino():
    def __init__ (self):...

    def insertarNodo(self, nodo):...

    def recorrer(self):...

    def verificarPunto(self, x, y):...


class Lista_prioridad:
    def __init__ (self):...

    def analizarNodo(self, nodo):...

    def pop(self):...

    def eliminarNodo(self, nodo):...

    def existe(self, nodo):...

    def insertar(self, nodo):...

    def recorrer(self):...
```

*Figura XIV. Módulo **Listas**.*
Fuente: elaboración propia, 2020.

La primera clase llamada **Lista,** es la encargada de recibir toda la información de los terrenos que esta ingresando el usuario (nombre, punto inicial, punto final, dimensiones, la matriz con el terreno) y posteriormente con los métodos **verificarTerreno(), recorrer()** y **vacia()** es capaz de gestionar los terrenos añadidos y regresar una respuesta a la solicitud del usuario.

La clase llamada **Lista,** esta encargada de contener nodos cuya tarea es la de transportar nodos pertenecientes a la matriz. La clase **ListaCamino()** contiene solamente nodos que pertenecen al camino más corto obtenido por el algoritmo, además apoya en la creación de la matriz en consola con su función **verificarPunto()** con el que es capaz de revisar si en su interior contiene algún punto de la matriz. Finalmente la clase **lista_prioridad()** es la encargada de recibir nodos de la matriz y ordenarlos de menor a

mayor. También es la clase que contiene el método **pop()** para devolver su primer elemento. (Es el método que se usa en el algoritmo de búsqueda).
Dentro del programa existen dos módulos destinados a la creación de estructuras de tipo nodo. El primer modulo se llama **Nodo** y únicamente contiene la clase con la estructurara de los nodos que conforman la lista de todos los terrenos registrados en el programa con su respectiva información proporcionada en el archivo xml.

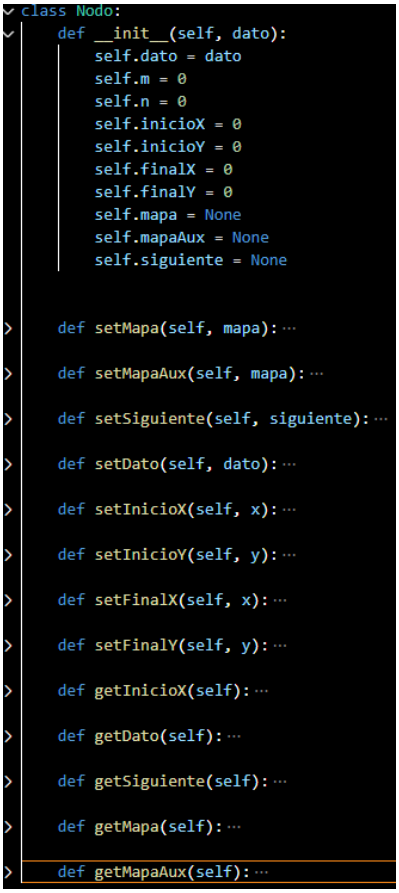


*Figura XV. Clase **Nodo**.*
Fuente: elaboración propia, 2020.

El otro módulo destinado para el fin de contener estructuras de tipo nodo lleva por nombre **Nodos** y tiene definidas 5 clases de tipo Nodo.

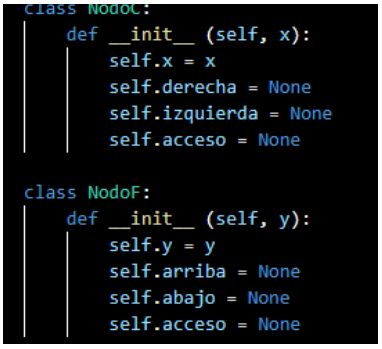


Figura XVI. Clases de Nodos de fila y Nodos de columnas.
Fuente: elaboración propia, 2020.

Las primeras clases se llaman **NodoF y NodoC**, llevan los mismos atributos, sin embargo, la clase **NodoF** esta destinada a ser para las listas de tipo encabezado **Filas** y la clase **NodoC** a formar parte de la clase encabezado **Columnas.**

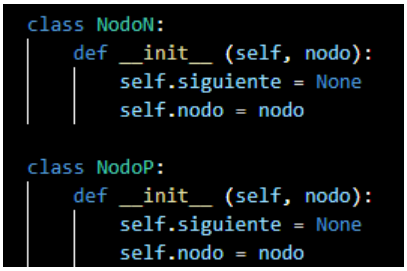


Figura XVII. Nodos para transportar Nodos
Fuente: elaboración propia, 2020.

Además de estas dos clases, el módulo posee dos clases extras de nodos destinadas para que la información que posean sea la de un nodo de la matriz.

Estos nodos son de suma importancia en el desarrollo del algoritmo de búsqueda, ya que son los que la función **regresarHermanos()** y **obtenerNodo()** devuelven como respuesta introduciendo en estos la información de los nodos solicitados.

Finalmente, uno de los nodos más importantes del programa es el nodo **NodoM,** ya que a partir de estos nodos es que está hecho el contenido de la matriz ortogonal, sin embargo, uno de sus aspectos más importantes es la contribución que hace al algoritmo del camino más corto. Ya que esta clase posee una función llamada **setData()**. La utilidad de esta función es algo confusa. Sin embargo, se puede resumir en que almacena el nodo que le haya sumado menor cantidad de combustible al nodo al cuál se le está cambiando el valor de la variable **data**. Esta funcionalidad se debe a que al momento de que se recorra la matriz existe la posibilidad de que un nodo sea ingresado dos veces en la lista, sin embargo con un gasto de combustible menor y debido a que este algoritmo se basa en la premisa de que "El camino más corto hacia un nodo cualesquiera es el conjunto de los caminos más cortos de todos los nodos intermedios" esta función gestiona esta parte, y únicamente almacena la menor suma de combustible hacia cada uno de los nodos de la matriz (y el nodo que le sumó ese valor).

```
class NodoM:
    def __init__ (self, data, x, y):
        self.data = data
        self.x = x
        self.y = y
        self.visitado = False
        self.arriba = None
        self.abajo = None
        self.derecha = None
        self.izquierda = None
        self.nodoPadre = None

    def setData(self, nodo):
        if self.nodoPadre == None:
            self.nodoPadre = nodo
            self.data = self.data + self.nodoPadre.nodo.data
        elif self.data < ((self.data - self.nodoPadre.nodo.data) + nodo.nodo.data):
            pass
        else:
            self.data = self.data - self.nodoPadre.nodo.data
            self.data = self.data + nodo.nodo.data
            self.nodoPadre = nodo

    def resetData(self):
        if self.nodoPadre is not None:
            self.data = self.data - self.nodoPadre.nodo.data

    def getData(self):
        return self.data
```

Figura XVIII. Nodos que conforman la matriz ortogonal
Fuente: elaboración propia, 2020.

Finalmente otra función fundamental es **resetData()** cuya función es restarle al valor actual de un nodo especifico el valor guardado más pequeño que encontró la función **setData()** lo que resulta en obtener el valor inicial del costo de combustible del nodo. Esta función sirve para generar el reporte en xml.

Conclusiones

Es de suma importancia el aprender las utilidades y formas de aplicar los algoritmos de búsqueda para caminos más cortos debido a sus cada vez más crecientes aplicaciones en la tecnología actual.

Es de suma importancia el saber manejar archivos de tipo **.xml** ya que estos son de suma importancia para varias formas de manejar los datos en muchas partes del mundo por lo que es prácticamente un requisito el tener conocimientos acerca de su estructura, funcionamiento y formas de manipular los datos que estos transportan.

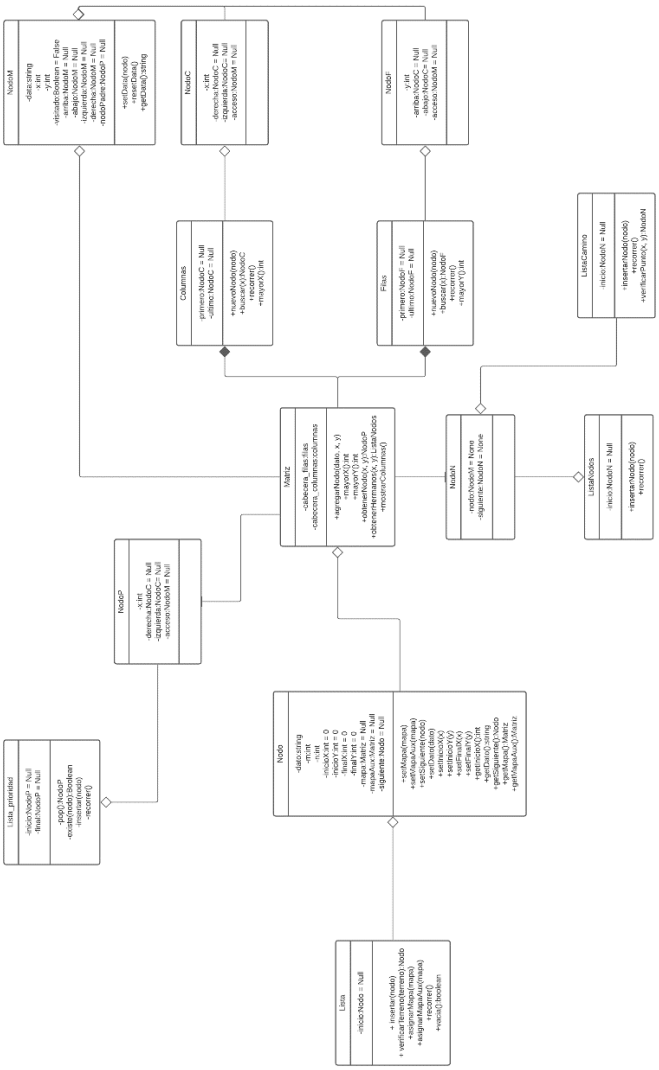
Para todos los lenguajes existen herramientas que son de mucha importancia para el programador, ya que brindar métodos con los cuáles se pueden enseñar y organizar datos de una forma mucho más gráfica y menos abstracta que el hacerlo en consola, logrando así una mayor compresión para el usuario. Este tipo de herramientas tienen bastante utilidad en el área de las estructuras de datos ya que son capaces de representar de una forma mucho más clara y concisa algo muy abstracto como lo son las listas, colas, pilas etc.

.

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 2021.

Anexos

I. DIAGRAMA DE CLASES DEL PROYECTO



II. LINK DEL DIAGRAMA DE CLASES

https://lucid.app/lucidchart/invitations/accept/inv_2a496a07-5539-4146-9c7e-61f643656670