INSTITUTO POLITÉCNICO NACIONAL

ESCOM

INTELIGENCIA ARTIFICIAL

LAB 3: BUSQUEDA NO INFORMADA P2

MARTÍNEZ CHÁVEZ JORGE ALEXIS

6CV3

26 SEPTIEMBRE 2024

Introducción

En esta práctica realizamos la continuación de la practica anterior donde debemos de resolver laberintos y el puzzle de 4 piezas, pero ahora debemos de hacer por medio de búsqueda en anchura. Como lo veremos a continuación ambos ejercicios los realizaremos a partir de una cola donde en donde debe de tener nodos cada uno de los nodos en los estados del ejercicio.

Como podemos ver son los mismo ejercicios, pero la diferencia es que ahora deberá de hacerlos por búsqueda en anchura, este nos beneficia cuando las soluciones estan muy cercanas, o son pocos movimientos para llegar a la misma, esto ya que primero revisara todos los hijos de nivel 1 antes de pasar a los del siguiente nivel, por tanto mas cercana este la solución será mas eficiente este tipo de algoritmo.

Algoritmos

Biblioteca

En la primera parte debemos de hacer la biblioteca para poder trabajar con las pilas, colas y listas.

Primero vamos a trabajar la pila en donde el ultimo objeto en entrar es el primer objeto en salir, para ello creamos un nodo en donde tiene como parámetros el valor y el siguiente objeto o nodo, que en el caso de las pilas será el nodo que se encuentra en la cima. Le creamos la función para ver si esta vacía, para ello simplemeten comparamos si esta None, entonces nos dicia que esta vacía y nos regresara el booleano de cual esta el estado.

```
class Pila:
    def __init__(self):
        self.cima = None

    def esta_vacia(self):
        return self.cima is None
```

Ahora le creamos la función para poder insertar nuevos objetos, recibe como parametros a la misma pila, y el nuevo nodo que estamos creado. Creamos el nuevo nodo y se le toma el parametro que esta recibiendo, y a nodo siguiente se le asigna el valor del nodo que esta en la cima de la misma pila.

```
    def push(self, objeto):
        nuevo_nodo = Nodo(objeto)
        nuevo_nodo.siguiente = self.cima
        self.cima = nuevo_nodo
```

Ahora para poder sacar un elemento primero debemos de comprobar si no esta vacía, para ello mandamos llamar la función que hemos creado, en caso de que no este vacía entonces sacara el ultimo elemento creado o la cima. Entonces simplemente guardaremos temporalmente el nodo cima como nodo removido, y al nodo cima le vamos asignar el nodo siguiente del nodo removido.

```
    def pop(self):
        if self.esta_vacia():
            return None
        nodo_removido = self.cima
        self.cima = self.cima.siguiente
        return nodo_removido.valor
```

Ahora simplemente para poder consultar haremos dos funciones, la primera para poder consultar cual es la cima y la otra para poder consultar toda la pila. En el caso de la cima retornaremos la cima y simplemete la imprimiremos. En el caso de toda la pila haremos un ciclo while en donde haremos un nodo de ayuda que es el actual al cual le asignaremos la cima, en caso de que ya no haya cima es que ya recorrimos toda la pila entonces terminaremos e iremos ir imprimiendo cada uno de los nodos.

```
    def ver_cima(self):
        if self.esta_vacia():
            return None
        return self.cima.valor

    def mostrar_pila(self):
        actual = self.cima
        elementos = []
        while actual:
            elementos.append(actual.valor)
            actual = actual.siguiente
        return elementos
```

Ahora hacemos una prueba, primero creamos la pila. Una vez creada le incertamos 3 objetos, despues pedimos que nos imprima toda la pila, depues hacemos un pop para poder sacar el ultimo objeto y volvemos a mostrar toda la pila para comprobar que realmente se hizo.

```
# Creando una pila
pila = Pila()
pila.push("objeto 1")
pila.push("objeto 2")
pila.push("objeto 3")
print("Pila:", pila.mostrar_pila())
print("Pop:", pila.pop())
print("Pila:", pila.mostrar_pila())
```

```
Pila: ['objeto 3', 'objeto 2', 'objeto 1']
Pop: objeto 3
Pila: ['objeto 2', 'objeto 1']
```

Ahora vamos a trabajar las colas en donde el primero entrar es el ultimo en salir, igual debemos de iniciar la pila en donde parametros es el primer nodo y el ultimo. Ahora hacemos el metodo para poder comprobar si la cola esta vacía para ello es simplemente hacer un booleano que regrese si esta vacía o no.

```
class Cola:
    def __init__(self):
        self.frente = None
        self.ultimo = None

    def esta_vacia(self):
        return self.frente is None
```

Ahora para poder insertar se recibe a la misma cola y al nuevo nodo que vamos a insertar con el valor que tiene, y comprobaremos que no este vacía, el caso de que si entonces al nodo frente y a ultimo le vamos a asignar el nuevo nodo que estamos recibiendo. En el caso de que no este vacía, entonces al valor de nodo siguiente del ultimo nodo le vamos a asignar el valor del nuevo nodo y el valor de ultimo nodo lo igualaremos con el del nodo recibido ya que ahora este es el ultimo.

```
def insertar(self, objeto):
    nuevo_nodo = Nodo(objeto)
    if self.esta_vacia():
        self.frente = self.ultimo = nuevo_nodo
    else:
        self.ultimo.siguiente = nuevo_nodo
        self.ultimo = nuevo_nodo
```

Ahora para eliminar un objeto, se elimina lo que este gurdado como el nodo frente, para ello primero comprobamos que no este vacío, luego a nodo removido le asignamos el nodo frente y al nodo frente ahora le asignaremos el nodo siguiente que tenia guardado este nodo frente.

```
def insertar(self, objeto):
    nuevo_nodo = Nodo(objeto)
    if self.esta_vacia():
        self.frente = self.ultimo = nuevo_nodo
    else:
        self.ultimo.siguiente = nuevo_nodo
        self.ultimo = nuevo_nodo
```

Ahora hacemos el recorrido de la cola para poderla imprimirla, iremos nodo por nodo y lo iremos almacenando en un arreglo y luego simplemente imprimirla.

```
def recorrer(self):
    actual = self.frente
    elementos = []
    while actual:
        elementos.append(actual.valor)
        actual = actual.siguiente
    return elementos
```

Ahora hacemos una prueba en donde hacemos la incerción de 3 objetos, imprimir la cola, depues eliminar el elemento que en este caso es el primer ingresado y depues volver a imprimir la cola para comprobar que se realizo la operación.

```
# Creando una cola
cola = Cola()
cola.insertar("objeto A")
cola.insertar("objeto B")
cola.insertar("objeto C")
print("Cola:", cola.recorrer())
print("Quitar:", cola.quitar())
print("Cola:", cola.recorrer())
```

```
Cola: ['objeto A', 'objeto B', 'objeto C']
Quitar: objeto A
Cola: ['objeto B', 'objeto C']
```

Ahora para la lista general unicamente tendra como parametro el nodo primero, creamos la función para poder comprobar que no este vacía la lista simplemete con el booleano de que el nodo de inicio no sea vacío.

```
class ListaGenerica:
    def __init__(self):
        self.primero = None

    def esta_vacia(self):
        return self.primero is None
```

Ahora para poder insertar un nuevo objeto primero en el caso de que este vacía la lista simplemente al nodo primero le vamos a asignar el valor del nuevo nodo, pero en caso de que no este vacía entonces nos ayudamos de un nodo que llamamos actula y almacenamos el valor del nodo primero e iremos recorriendo toda la lista hasta llegar al final esto con un bucle hasta que no haya un siguiente, entonces a este ultimo nodo en el espacio de nodo siguiente le asignaremos el valor del nuevo nodo.

```
    def insertar(self, objeto):
        nuevo_nodo = Nodo(objeto)
        if self.esta_vacia():
            self.primero = nuevo_nodo
        else:
            actual = self.primero
            while actual.siguiente:
                actual = actual.siguiente
            actual.siguiente = nuevo_nodo
```

Ahora para poder eliminar un objeto primero comprobamos que no este vacía, despues debemos de hacer un recorrido para eliminar, ya que ahora podemos eliminar un nodo que este en cualquier posición. En caso de que el objeto a eliminar sea el primero objeto simplemente el nodo que esta la posición de siguiente nodo del nodo primero, se lo asignamos al nodo primero ya que este pasara a ser el nuevo nodo primero. En

caso de que el nodo se encuntre en otra posición entonces vamos a ayudarnos de un nuevo que llamaremos como anterior, cuando hayamos encontrado el nodo entonces a la variable de siguiente del nodo anterior le asignaremos el siguiente del actual, es decir como una cadena uniremos los eslabones del que quitamos.

```
    def eliminar(self, objeto):
        if self.esta_vacia():
            return False

        if self.primero.valor == objeto:
            self.primero = self.primero.siguiente
            return True

        actual = self.primero
        anterior = None
        while actual:
            if actual.valor == objeto:
                anterior.siguiente = actual.siguiente
                return True
            anterior = actual
            actual = actual.siguiente

        return False
```

Ahora haremos la prueba en donde insertamos 3 objetos, imprimiremos la lista eliminamos el de en medio y volvemos a imprimir la lista para comprobar que se realizo la operación.

```
# Creando una lista genérica
lista = ListaGenerica()
lista.insertar("elemento 1")
lista.insertar("elemento 2")
lista.insertar("elemento 3")
print("Lista:", lista.recorrer())
print("Eliminar 'elemento 2':", lista.eliminar("elemento 2"))
print("Lista:", lista.recorrer())
```

```
Lista: ['elemento 1', 'elemento 2', 'elemento 3']
Eliminar 'elemento 2': True
Lista: ['elemento 1', 'elemento 3']
```

4-Puzzle

Para resolver el puzzle primero debemos de poder crear la estructura, en este caso vamos a crear nuestros nodos, el cual estará conformado por los parámetros del estado de la matriz, el nodo padre, y el movimiento que se realizo para poder llegar a ese estado.

```
# Clase para definir un nodo del puzzle
class Nodo:
    def __init__(self, estado, padre=None, movimiento=None):
        self.estado = estado
        self.padre = padre
        self.movimiento = movimiento
```

Ahora ejecutamos la solución del problema en donde de inicio madnara a llamar la otra función la cual le regresara como parametro la solución, en caso de que no se haya regresado nada es que no se encontro una solución. En caso de que si entonces entrara en un ciclo en donde imprimira el estado y el movimiento de cada uno de los nodos que se tomarón para poder llegar a la solución.

```
def ejecutar_resolucion():
    camino_solucion = resolver_4_puzzle()

    if camino_solucion:
        print("\nCamino para resolver el puzzle:")
        for movimiento, estado in camino_solucion:
            print(f"Movimiento: {movimiento}")
            imprimir_tablero(estado)
    else:
        print("No se encontró una solución para el puzzle.")
```

Esta función lo que hace es crear primero una variable que es el nodo inicial, en donde de forma aleatoria generamos el primer estado del nodo, y este lo ingresamos dentro de la cola, la cual la ocuparamos con la librería “deque”, tambien creamos la lista de

nodos visitados para no repetir nodos cada que se creen los posibles movimientos.

```
def resolver_4_puzzle():
    estado_inicial = generar_estado_inicial()
    print("Estado inicial:")
    imprimir_tablero(estado_inicial)

    nodo_inicial = Nodo(estado_inicial)
    cola = deque([nodo_inicial])
    visitados = set()
```

Ahora inica el ciclo en la cola, en donde va a tomar el “primer nodo ingresado” o el nodo de la izquerda, y este lo almacenaremos en una variable que llamaremos nodo actual, comprobamos que sea la solución en caso de que si, al nodo actual le madamos reconstruir el camino solución y este es el que regresaremos para impresión; en caso contrario, entonces debemos ubicar en donde esta la posición de movimiento, y a este mandar llamar la función para crearle los posibles movimientos. Una vez que nos haya regresado los movimeintos, ahora los comprobaremos que esos estados que se creen no hayan sido ya visitados, para ellos hacemos otro ciclo en donde comparamos cada uno de estos movimetos de que no esten dentro de la lista de visitados; en caso de que no se hayan visitados, entonces creamos el nodo, en donde le agreamos el estado, el nodo padre es el nodo actual, y el movimeto que se realizo para llegar a ese estado, por

utimo, simplemenete agregamos ese nodo a la cola.

```
while cola:
    nodo_actual = cola.popleft()
    estado_actual = nodo_actual.estado

    if es_estado_objetivo(estado_actual):
        print("¡Se encontró la solución!")
        camino_solucion = reconstruir_camino(nodo_actual)
        return camino_solucion

    posicion_vacia = obtener_posicion_vacia(estado_actual)
    movimientos_validos = obtener_movimientos_validos(posicion_vacia)

    for movimiento in movimientos_validos:
        nuevo_estado = mover(estado_actual, posicion_vacia, movimiento)
        estado_tupla = tuple(tuple(fila) for fila in nuevo_estado)

        if estado_tupla not in visitados:
            visitados.add(estado_tupla)
            nuevo_nodo = Nodo(nuevo_estado, nodo_actual, movimiento)
            cola.append(nuevo_nodo)

return None
```

Para poder crear los movimetos hacemos dos funciones, la primera para poder hacer los posibles movimientos, que es si esta arriba solo se puede mover para abajo, si esta abajo solo se puede mover para arrriba, si esta a la derecha solo se puede mover a la izquierda y por ultimo si esta a la izquierda, solo s epuede mover a la derecha.

```
def obtener_movimientos_validos(posicion_vacia):
    fila, columna = posicion_vacia
    movimientos = []

    if fila > 0:
        movimientos.append("arriba")
    if fila < 1:
        movimientos.append("abajo")
    if columna > 0:
        movimientos.append("izquierda")
    if columna < 1:
        movimientos.append("derecha")

    return movimientos
```

Una vez que obtengamos el posible movimeto entonces lo realizamos, haciendo la respectiva suma o resta de posición del espacio vacío. Y ya que creamos la posición entonces regresamos cada uno de los posibles nuevos estados.

```
def mover(estado, posicion_vacia, direccion):
    nueva_estado = copy.deepcopy(estado)
    fila, columna = posicion_vacia

    if direccion == "arriba":
        nueva_estado[fila][columna], nueva_estado[fila - 1][columna] = nueva_estado[fila - 1][columna
    elif direccion == "abajo":
        nueva_estado[fila][columna], nueva_estado[fila + 1][columna] = nueva_estado[fila + 1][columna
    elif direccion == "izquierda":
        nueva_estado[fila][columna], nueva_estado[fila][columna - 1] = nueva_estado[fila][columna - 1
    elif direccion == "derecha":
        nueva_estado[fila][columna], nueva_estado[fila][columna + 1] = nueva_estado[fila][columna + 1

    return nueva_estado
```

Para poder hacer la regresión en el camino solución, hacemos la función donde se recibe el nodo solución, y hara un bucle en donde por cada nodo ira buscado hacia su nodo padre, y todos estos nodos los iran gurdando en un arreglo, mismo que regresaremos como solución para su impresión.

```
def reconstruir_camino(nodo):
    camino = []
    while nodo.padre is not None:
        camino.append((nodo.movimiento, nodo.estado))
        nodo = nodo.padre
    camino.reverse()
    return camino
```

Por ultimo hacemos una ejecución del mismo para ver que si haga la solución del problema.

```
PS C:\Users\jorge\OneDrive\Documentos\JORGE\ESCOM\9_SEMESTRE> & C:/Users/jorge/AppData/Local/Programs/Python/Python310/python
.exe c:/Users/jorge/OneDrive/Documentos/JORGE/ESCOM/9_SEMESTRE/IA/LABORATORIO/3_BUSQUEDA_NO_INFORMADA_P2/4puzzle.py
Estado inicial:
[0, 3]
[2, 1]

¡Se encontró la solución!

Camino para resolver el puzzle:
Movimiento: abajo
[2, 3]
[0, 1]

Movimiento: derecha
[2, 3]
[1, 0]

Movimiento: arriba
[2, 0]
[1, 3]

Movimiento: izquierda
[0, 2]
[1, 3]

Movimiento: abajo
[1, 2]
[0, 3]

Movimiento: derecha
[1, 2]
[3, 0]
```

Laberinto

Primero iniciamos con la creación del laberinto, el cual es una matriz. También indicamos cuales son las posiciones de inicio y la posición final.

```
laberinto = [
    [1, 0, 1, 1, 1],
    [1, 0, 0, 0, 1],
    [1, 1, 1, 0, 1],
    [1, 0, 0, 0, 0],
    [1, 1, 1, 1, 1]
]

# Posición de inicio y salida
inicio = (0, 1)
fin = (3, 4)
```

Para los moviemitnos vamos a hacerlos a partir de la creación de nodos el cual tiene como parametro la posición en donde se encuentra asi como el nodo padre, que es la posición anterior al movimiento.

```
class Nodo:
    def __init__(self, posicion, padre=None):
        self.posicion = posicion
        self.padre = padre
```

Iniciamos la ejecución del programa, en donde se manda a llamar la función de resolver laberinto, en caso de que tenga solución entonces mandaremos llamar la función de imprimir el laberinto, en caso contrario entonces no se encontro solución.

```
def ejecutar_resolucion():
    camino_solucion = resolver_laberinto()

    if camino_solucion:
        print("\nCamino final para resolver el laberinto:")
        imprimir_laberinto(laberinto, camino_solucion)
        print(f"El camino encontrado es: {camino_solucion}")
    else:
        print("No se encontró ninguna solución para el laberinto.")
```

En la función solución iniciamos creando una varibale para el nodo inicial el cual le vamos a asignar un nuevo nodo que tiene como información la posición de inicio, creamos la colo y le incertamos este nuevo nodo creado. Tambien creamos la lista de

los nodos visitados para no repetir movimientos.

```
def resolver_laberinto():
    nodo_inicial = Nodo(inicio)
    cola = deque([nodo_inicial])
    visitados = set()
    visitados.add(nodo_inicial.posicion)
```

Ahora iniciamos el ciclo en donde cremamos un nodo que llamamos nodo actual y le agremos el valor del nodo que saquemos de la izquierda de la pila, y obtendremos la posición actual la cual sacaremos del nodo actual, e imprimimos el laberinto. En caso de que se haya llegado a la solución, entonces deberemos de regresa la reconstrucción del camino, en caso contrario debemos de crear los posibles movimeintos los cuales los haremos con la ayuda de otra función, asi como la comprobación de que sean movimetos que no se hayan visitado previamente.

```
while cola:
    nodo_actual = cola.popleft()
    posicion_actual = nodo_actual.posicion

    camino_actual = reconstruir_camino(nodo_actual)
    imprimir_laberinto(laberinto, camino_actual)
    print(f"Visitando: {posicion_actual}")

    if posicion_actual == fin:
        print("¡Se encontró la solución!")
        return reconstruir_camino(nodo_actual)

    fila, columna = posicion_actual
    movimientos = [(fila - 1, columna), (fila + 1, columna), (fila, columna - 1), (fila, columna -

    for movimiento in movimientos:
        if es_movimiento_valido(laberinto, movimiento) and movimiento not in visitados:
            visitados.add(movimiento)
            nuevo_nodo = Nodo(movimiento, nodo_actual)
            cola.append(nuevo_nodo)

print("No se encontró una solución para el laberinto.")
return None
```

Para imprimir el laberinto debemos de hacer un doble ciclo para poder ir imprimiedo posición por posición 1 para las paredes, 0 para el camino. Pero tambien se recibe

como parametro la lista del camino visitado, para que agruegue una P a ese camino.

```
def imprimir_laberinto(laberinto, camino_actual=[]):
    for i in range(len(laberinto)):
        for j in range(len(laberinto[i])):
            if (i, j) in camino_actual:
                print("P", end=" ")
            else:
                print(laberinto[i][j], end=" ")
        print()
    print()
```

Ahora la función para crear los movimientos, simplemente sera arriba, abajo, derecha izquieda, con la comprobación de que no exista una pared que limite el movimiento.

```
def es_movimiento_valido(laberinto, posicion):
    fila, columna = posicion
    if 0 <= fila < len(laberinto) and 0 <= columna < len(laberinto[0]) and laberinto[fila][columna]
        return True
    return False
```

La función para poder reconstruir ese camino de solución, debemos de hacer un ciclo que vaya por donde haya encontrado la solución, esto lo hara en forma de tuplas de filas y columnas, estas posiciones las agregara en el arreglo y lo invertira para poder imprimir la solución en el orden adecuado.

```
def ejecutar_resolucion():
    camino_solucion = resolver_laberinto()

    if camino_solucion:
        print("\nCamino final para resolver el laberinto:")
        imprimir_laberinto(laberinto, camino_solucion)
        print(f"El camino encontrado es: {camino_solucion}")
    else:
        print("No se encontró ninguna solución para el laberinto.")
```

Por ultimo ponemos a prueba la ejecución para comprobar que este funcione.

```
PS C:\Users\jorge\OneDrive\Documentos\JORGE\ESCOM\9_SEMESTRE> & C:/Users/jorge/AppData/Local/Programs/Python/Python310/python
.exe c:/Users/jorge/OneDrive/Documentos/JORGE/ESCOM/9_SEMESTRE/IA/LABORATORIO/3_BUSQUEDA_NO_INFORMADA_P2/laberinto.py
1 P 1 1 1
1 0 0 0 1
1 1 1 0 1
1 0 0 0 0
1 1 1 1 1
```

```
Visitando: (0, 1)
1 P 1 1 1
1 P 0 0 1
1 1 1 0 1
1 0 0 0 0
1 1 1 1 1

Visitando: (1, 1)
1 P 1 1 1
1 P P 0 1
1 1 1 0 1
1 0 0 0 0
1 1 1 1 1

Visitando: (1, 2)
1 P 1 1 1
1 P P P 1
1 1 1 0 1
1 0 0 0 0
1 1 1 1 1

Visitando: (1, 3)
1 P 1 1 1
1 P P P 1
1 1 1 P 1
1 0 0 0 0
1 1 1 1 1

Visitando: (2, 3)
1 P 1 1 1
1 P P P 1
1 1 1 P 1
1 0 0 P 0
1 1 1 1 1
```

```
Visitando: (3, 3)
1 P 1 1 1
1 P P P 1
1 1 1 P 1
1 0 P P 0
1 1 1 1 1

Visitando: (3, 2)
1 P 1 1 1
1 P P P 1
1 1 1 P 1
1 0 0 P P
1 1 1 1 1

Visitando: (3, 4)
¡Se encontró la solución!

Camino final para resolver el laberinto:
1 P 1 1 1
1 P P P 1
1 1 1 P 1
1 0 0 P P
1 1 1 1 1

El camino encontrado es: [(0, 1), (1, 1), (1, 2), (1, 3), (2, 3), (3, 3), (3, 4)]
```

Enlace

A continuación, agregamos el link correspondiente al git en donde se encuentran todas las practicas. Los códigos y reporte de esta practica se encuentra dentro de la carpeta de LAB_3_BUSQUEDA_NO_INFORMADA_P2: https://github.com/Jorge300403/IA_6CV3_MartinezChavez/tree/main/LAB_3_BUSQUEDA_NO_INFORMADA_P2