Sandoval Resendiz Juan Carlos

- Puzzle

```
nodo_inicial=[1,2,4,3]
solucion=[1,2,3,4]
nodos_frontera=[]
nodos_visitados=[]
nodos_frontera.append(nodo_inicial)
```

Se declaro el nodo_inicial, la solución del puzzle, los nodos_frontera como una lista pero será manejada bajo las reglas de una cola, los nodos_visitados que si serán tratados como una lista y se añadira el nodo_inicial a nodos_frontera

```
def mover_izquierda(nodo_actual):
    lista_temp=nodo_actual.copy()
    lista_temp[0],lista_temp[1]=lista_temp[1],lista_temp[0]
    return lista_temp

def mover_centro(nodo_actual):
    lista_temp=nodo_actual.copy()
    lista_temp[1],lista_temp[2]=lista_temp[2],lista_temp[1]
    return lista_temp

def mover_derecha(nodo_actual):
    lista_temp=nodo_actual.copy()
    lista_temp[2],lista_temp[3]=lista_temp[3],lista_temp[2]
    return lista_temp

movimientos=[mover_derecha, mover_centro, mover_izquierda]
```

Se definieron las funciones que serán los movimientos a realizar en el puzzle lo que se hará dentro de ellos es recibir el nodo_actual en el momento de la iteración.

Se guardara una copia de la lista en lista_temp para no hacer modificación directa de nodo_actual

Se hará un intercambio de las posiciones dependiendo del movimiento a realizar y se hará el intercabion referenciando su posición en la lista

Se devolverá el return de lista_temp que se almacera próximamente en un nodo_hijo

Estas funciones se guardaran en una lista para su próximo uso en una iteración

```
while True:

    nodo_actual=nodos_frontera.pop(0)

    if nodo_actual==solucion:
        print(nodos_visitados)
        break

    nodos_visitados.append(nodo_actual)

    for i in movimientos:
        nodo_hijo=i(nodo_actual)

        if nodo_hijo not in nodos_frontera and nodo_hijo not in nodos_visitados:
            nodos_frontera.append(nodo_hijo)
```

Bucle principal el cual estará en ciclo hasta encontrar el resultado final. Se empezara el bucle sacando de la cola el nodo que haya entrado primero en la cola por eso agregamos que haga el pop en el 0 que es el primer valor en entrar y almacenándolo en nodo_actual

La condición del if revisa si nodo_actual es igual a la solución de ser así imprime los nodos_visitados para llegar al resultado y rompe el bucle de no ser así pasa de largo y se agrega el nodo_acutal a nodos_visitados

Pasamos al ciclo for que hará uso de la lista de movimientos para iterarlas y llamarlas para realizar su función dentro del ciclo se guarda el resultado de llamar a la función en nodo_hijo y pasamos a un if que verifica si el nodo_hijo no se encuentra en nodos_frontera y en nodos_visitados de ser que no se encuentren el nodo_hijo se guarda en la pila de nodo_frontera

- Resultados

```
C:\Users\juanc\Documents\practica_IA>C:/Users/juanc/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/juanc/Documents/practica_IA/busqueda_no_informada_p2/puzzleP.py
[[1, 2, 4, 3]]
```

Sandoval Resendiz Juan Carlos

- Laberinto

```
import numpy as np

maze=[
    [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]
]
start=[0,1]
end=[3,4]
nodos_frontera=[]
nodos_visitados=[]
nodos_frontera.append(start)
```

Se declara el maze el cual será nuestro laberinto que es una lista de listas start que será nuestra lista con la posición del inicio del laberinto, end con una lista con la posición de la salida, una cola de nodos_frontera, una lista de nodos_visitados

Se agrega start en nodos_frontera

Sandoval Resendiz Juan Carlos

```python
def moverse_izquierda(nodo_actual):
    posicion=nodo_actual.copy()
    if posicion[-1]!=0:
        posicion[-1]-=1
        return posicion
    else:
        return posicion


def moverse_abajo(nodo_actual):
    posicion=nodo_actual.copy()
    if posicion[0]!=len(maze)-1:
        posicion[0]+=1
        return posicion
    else:
        return posicion

def moverse_derecha(nodo_actual):
    posicion=nodo_actual.copy()
    if posicion[-1]!=len(maze)-1:
        posicion[-1]+=1
        return posicion
    else:
        return posicion

def moverse_arriba(nodo_actual):
    posicion=nodo_actual.copy()
    if posicion[0]!=0:
        posicion[0]-=1
        return posicion
    else:
        return posicion
```

Sandoval Resendiz Juan Carlos

Se definen las funciones de los movimientos a realizar moverse_arriba y moverse_izquierda recibirán el nodo_actual y harán una copia que guardaran en posición hace una condicional que si la fila en la posición 0 en el caso de moverse_arriba y la columna que se encuentra en la posición -1 que es el ultimo elemento de la lista en su caso para mover_izquierda su valor es diferente a 0 se le restara 1 al valor en esa posición para obtener una nueva posición y retornarla de no cumplirse la condición solo devolverá la posición que le había llegado

En el caso de moverse_abajo y moverse_derecha recibirán el nodo_actual y harán una copia que guardaran en posición hace una condicional que si la fila en la posición 0 en el caso de moverse_abajo y la columna que se encuentra en la posición -1 que es el ultimo elemento de la lista en su caso para mover_derecha su valor es diferente a longitud de maze -1 para saber los limites del laberinto se le sumara 1 al valor en esa posición para obtener una nueva posición y retornarla de no cumplirse la condición solo devolverá la posición que le había llegado

```
while True:

    nodo_actual=nodos_frontera.pop(0)

    if nodo_actual==end:
        print(nodos_visitados)
        break

    nodos_visitados.append(nodo_actual)

    for i in movimientos:
        nodo_hijo=i(nodo_actual)

        if nodo_hijo not in nodos_frontera and nodo_hijo not in nodos_visitados:
            if maze[nodo_hijo[0]][nodo_hijo[1]]==0:
                nodos_frontera.append(nodo_hijo)
```

Se guardan los movimientos a realizarse e iterarlos depues

Bucle principal el cual estará en ciclo hasta encontrar el resultado final. Se empezara el bucle sacando de la cola el nodo que haya entrado primero en la cola por eso agregamos que haga el pop en el 0 que es el primer valor en entrar y almacenándolo en nodo_actual

Sandoval Resendiz Juan Carlos

La condición del if revisa si nodo_actual es igual a la solución de ser así imprime los nodos_visitados para llegar al resultado y rompe el bucle de no ser así pasa de largo y se agrega el nodo_acutal a nodos_visitados

Pasamos al ciclo for que hará uso de la lista de movimientos para iterarlas y llamarlas para realizar su función dentro del ciclo se guarda el resultado de llamar a la función en nodo_hijo y pasamos a un if que verifica si el nodo_hijo no se encuentra en nodos_frontera y en nodos_visitados de ser que no se encuentren el nodo_hijo se pasa a un if que revisara que la posición que se este dando sea igual a 0 ya que es por donde se puede caminar y no por los 1 de ser así se guarda en la pila de nodo_frontera

- Resultados

```
C:\Users\juanc\Documents\practica_IA>C:/Users/juanc/AppDa
ents/practica_IA/busqueda_no_informada_p2/laberintoP.py
[[0, 1], [1, 1], [1, 2], [1, 3], [2, 3], [3, 3]]
```