Tecnológico Nacional de México / Instituto Tecnológico de Culiacán.



**Ingeniería en Sistemas Computacionales.**

**Inteligencia Artificial**

Zuriel Dathan Mora Felix.

**Modulo 1**

**Documentación Puzzle-8.**

Ontiveros Sánchez Jesús Daniel (22170750).

Ramírez Ruiz Santiago (22170783).

**Horario de 9:00 a 10:00 AM.**

**Culiacán Sinaloa a 10 de septiembre del 2025**

**Descripción General del Proyecto (Puzzle-8)**

Este proyecto implementa el juego del Puzzle-8 y un algoritmo de búsqueda en anchura (BFS) para hallar la secuencia más corta de movimientos desde un estado inicial hasta el estado objetivo:

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 0 |

El tablero se representa como una matriz de 9 enteros donde 0 es el espacio vacío. BFS explora los estados por niveles (primero todos los que están a 1 movimiento, luego a 2, etc.), garantizando encontrar la solución con el menor número de movimientos cuando existe. El programa:

1. Lee el estado inicial desde consola,
2. Ejecuta BFS
3. Reconstruye el camino solución
4. Muestra cada paso del tablero.

**Instrucciones de Instalación**

Requisitos

* Python 3.8+
* No requiere dependencias externas (solo collections.deque de la biblioteca estándar).

Ejecución:   
# Clona / copia el archivo puzzle8.py (por ejemplo)

python puzzle8.py

**Instrucciones de Uso**

1. **Inicia el programa. Verás un mensaje:**

Ingresa los 9 números separados por espacios (ejemplo: 1 2 3 4 5 6 7 8 0)

1. **Escribe el estado inicial del tablero como 9 números del 0 al 8 sin repetir (el 0 es el hueco).**

Ejemplo de entrada válida: 1 2 3 4 5 6 0 7 8

1. **El programa validará la entrada y luego:**

* Imprimirá el estado inicial.
* Ejecutará BFS.
* Mostrará paso a paso la solución y el número de movimientos.

**Controles**

No hay controles en vivo: el juego no es interactivo por teclas; el algoritmo calcula y muestra la solución automáticamente.

**Objetivo**

Que el algoritmo utilizado encuentre la solución teniendo la menor cantidad de movimientos posibles. El algoritmo te indica cómo mover las fichas deslizándolas hacia el espacio vacío (arriba/abajo/izquierda/derecha) hasta alcanzar el estado de aceptación: (1,2,3,4,5,6,7,8,0).

**Documentación de Código**

from collections import deque

# Definir el estado de aceptacion del puzzle

ESTADO\_ACEPTACION = (1, 2, 3,

                     4, 5, 6,

                     7, 8, 0)

# Función para imprimir el estado del puzzle

"""

Se imprime el estado del puzzle en un formato 3x3,

reemplazando el 0 con un espacio en blanco para mayor claridad.

"""

def imprimir\_puzzle(estado):

    for i in range(0, 9, 3):

        fila = estado[i:i+3]

        print(" ".join(str(num) if num != 0 else ' ' for num in fila))

    print()

# Función para encontrar los vecinos (movimientos posibles)

    """

    Genera todos los estados vecinos de un estado dado,

    moviendo el espacio vacío (0) en las direcciones válidas:

    arriba, abajo, izquierda o derecha.

    """

def obtener\_vecinos(estado):

    vecinos = []

    indice = estado.index(0)  # Encuentra la posición del espacio vacío

    fila, columna = divmod(indice, 3) # Convierte el índice en coordenadas de fila y columna

    movimientos = [] # Lista de movimientos posibles

    if fila > 0:

        movimientos.append((-1, 0))  # Arriba

    if fila < 2:

        movimientos.append((1, 0))   # Abajo

    if columna > 0:

        movimientos.append((0, -1))  # Izquierda

    if columna < 2:

        movimientos.append((0, 1))   # Derecha

    # Genera nuevos estados para cada movimiento posible

    for movimiento in movimientos:

        nueva\_fila, nueva\_columna = fila + movimiento[0], columna + movimiento[1]

        nuevo\_indice = nueva\_fila \* 3 + nueva\_columna

        nuevo\_estado = list(estado)

        # Intercambio entre el 0 y la casilla destino

        nuevo\_estado[indice], nuevo\_estado[nuevo\_indice] = nuevo\_estado[nuevo\_indice], nuevo\_estado[indice]

        vecinos.append(tuple(nuevo\_estado))

    return vecinos

# Función para reconstruir el camino desde el estado inicial hasta el objetivo

def reconstruir\_camino(viene\_de, actual):

    """

    Reconstruye el camino desde el estado inicial hasta el estado actual,

    usando el viene\_de.

    """

    camino = [actual]

    mientras\_actual = actual

    while mientras\_actual in viene\_de:

        mientras\_actual = viene\_de[mientras\_actual]

        camino.append(mientras\_actual)

    camino.reverse() # Invierte el camino para que vaya del inicio al objetivo

    return camino

# BFS (Búsqueda en Anchura)

# Explora nivel por nivel hasta encontrar el estado de aceptación

def bfs(estado\_inicial):

    print("=== BFS (Búsqueda en Anchura) ===")

    # Cola de exploración

    frontera = deque([estado\_inicial])

    viene\_de = {}   # Guarda el padre de cada estado

    explorados = set() # Conjunto de estados ya visitados

    while frontera:

        actual = frontera.popleft() # Extrae el primer estado de la cola

        if actual == ESTADO\_ACEPTACION: # Verifica si es el estado objetivo

            camino = reconstruir\_camino(viene\_de, actual)

            return camino

        explorados.add(actual)  # Marca el estado como explorado

        for vecino in obtener\_vecinos(actual): # Genera los estados vecinos

            if vecino not in explorados and vecino not in frontera:

                frontera.append(vecino) # Añade el vecino a la cola

                viene\_de[vecino] = actual #Guarda de donde vino el vecino

    return None # Si no se encuentra solución

# Función para mostrar el camino paso a paso

"""

Muestra cada paso del camino desde el estado inicial hasta el estado objetivo,

imprimiendo el estado del puzzle en cada paso.Si no se encuentra solución, informa al usuario.

"""

def mostrar\_solucion(camino):

    if not camino:

        print("No se encontró solución.")

        return

    print(f"Se encontraron {len(camino)-1} movimientos.\n")

    for i, estado in enumerate(camino):

        print(f"Paso {i}:")

        imprimir\_puzzle(estado)

# Función principal

def main():

    print("Juego del Puzzle-8")

    print("Ingresa el estado inicial del puzzle (usa 0 para el espacio vacío):")

    estado\_inicial = []

    while True:

        entrada = input("Ingresa los 9 números separados por espacios (ejemplo: 1 2 3 4 5 6 7 8 0): ")

        partes = entrada.strip().split()

        if len(partes) != 9:

            print("Por favor, ingresa exactamente 9 números.")

            continue

        try:

            estado\_inicial = tuple(int(num) for num in partes)

            if set(estado\_inicial) != set(range(9)):

                print("Los números deben ser del 0 al 8 sin repetirse.")

                continue

            break

        except ValueError:

            print("Por favor, asegúrate de ingresar números válidos.")

    print("\nEstado Inicial:")

    imprimir\_puzzle(estado\_inicial)

    camino = bfs(estado\_inicial)

    mostrar\_solucion(camino)

if \_\_name\_\_ == "\_\_main\_\_":

    main()