

# Laberinto Proyecto Aplicativo

---

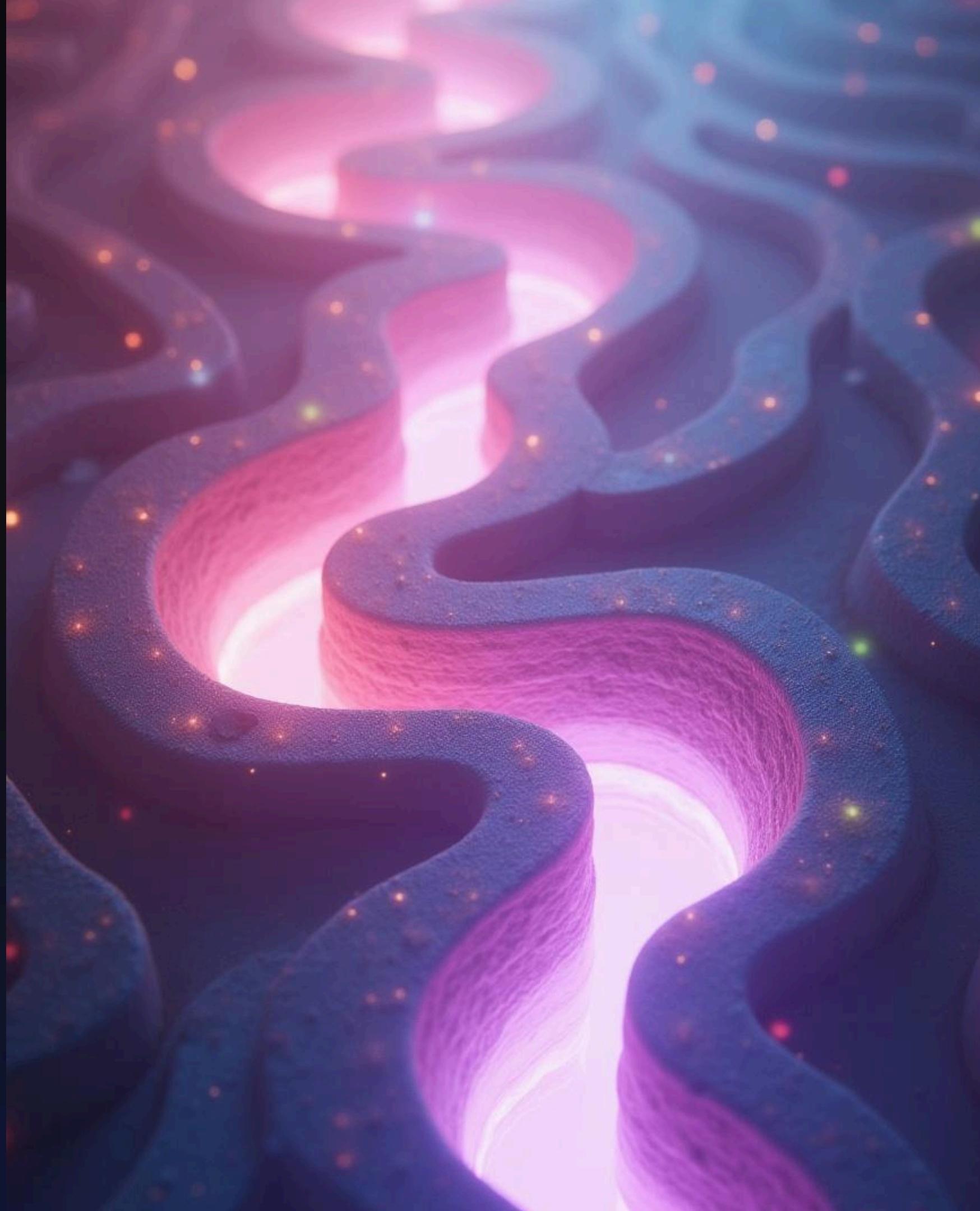
Juan Sebastian Sanchez, Karol Torres



# Introducción

Este proyecto muestra cómo resolver un laberinto NxN usando **algoritmos BFS, DFS y AI**. Aprenderemos a transformar el laberinto en grafo, aplicar heurística Manhattan para A, y reconstruir la ruta solución.

Exploraremos los desafíos comunes y cómo manejarlos para entender mejor la inteligencia artificial aplicada a búsqueda.





01

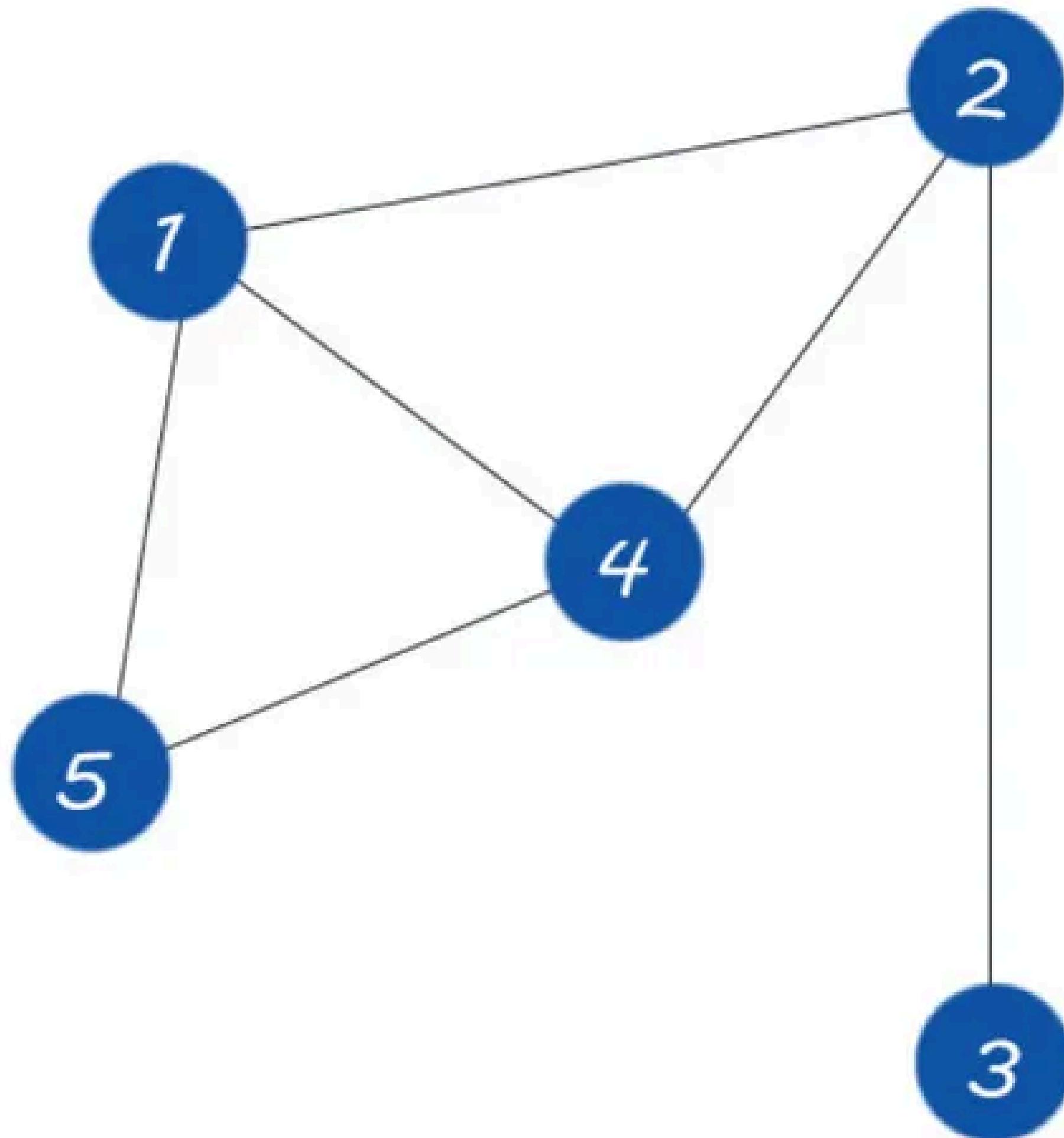
# Planteamiento y modelado del laberinto

# Representación de la matriz NxN y codificación de celdas

El laberinto se representa como una matriz donde cada celda tiene un valor: para libre, **1** para pared, **2** punto de inicio, y **3** meta. Este esquema permite identificar permisos y obstáculos fácilmente, y es la base para modelar el problema.

Se entrega la matriz junto con posiciones inicial y final para resolver.

2	1	1	1	0	0	1	1	1	1
0	0	0	1	1	1	0	0	1	0
1	1	0	0	0	1	0	1	1	1
0	1	1	1	0	1	0	1	0	1
1	1	0	1	0	1	0	0	1	1
0	0	0	1	0	3	1	0	1	0
1	1	1	1	1	0	1	0	1	1
1	0	0	0	0	1	1	1	0	1
1	1	1	1	1	1	0	1	0	1
1	0	1	0	1	0	1	1	1	1
1	0	1	0	1	0	1	1	1	1



## Modelo del estado: nodos y movimientos en 4 direcciones

Cada nodo es una celda (fila, columna) transitables. Los movimientos posibles son *arriba*, *abajo*, *izquierda*, *derecha*. Se valida que no se crucen límites ni paredes para asegurar que solo se exploran movimientos válidos. Así se garantiza que la búsqueda funcione solo sobre caminos válidos.



# Validación de límites y paredes en matriz y grafo

Para asegurar movimientos válidos, se comprueba que la próxima celda esté dentro del rango de la matriz y que no sea una pared (valor 1). Este control evita errores y ciclos infinitos en la búsqueda. En el grafo, los enlaces solo se crean entre nodos transitables vecinos que cumplen estas condiciones, garantizando rutas factibles.



# 01

## Algoritmos de búsqueda y reconstrucción de ruta



# Transformación matriz a grafo: nodos, enlaces y pesos uniformes

Cada celda libre se convierte en un nodo. Se crean enlaces hacia los nodos vecinos válidos (arriba, abajo, izquierda, derecha), con **peso 1** para cada conexión. La representación es con diccionario o lista de adyacencia, facilitando la exploración eficiente y el seguimiento de estados visitados y padres.

```
# Movimientos posibles: abajo, arriba, derecha, izquierda
directions = [(1,0), (-1,0), (0,1), (0,-1)] # 4-dir
```

# Descripción y estructura de BFS y DFS

BFS usa una **cola (deque)** para explorar niveles equitativos, garantizando encontrar la ruta más corta en distancias uniformes. DFS usa una **pila**, explorando profundidad y dependiendo del orden de vecinos cambia el camino. Ambos mantienen listas de visitados y padres para reconstruir la ruta.

```
) BFS (Búsqueda en anchura)

bfs(graph, start, goal):
    cola = deque([start])
    visited = set([start])
    parent = {start: None}

    while cola:
        u = cola.popleft()
        # Si u es la meta reconstruirmos la ruta
        if u == goal:
            return reconstruct_path(parent, goal)
        # Recorremos vecinos de u, si no han sido vistos
        for v, cost in graph.get(u, []):
            if v not in visited:
                visited.add(v)
                parent[v] = u
                cola.append(v)

    return None
```

# Heurística Manhattan y funcionamiento básico del algoritmo A\*

La heurística Manhattan calcula la distancia estimada usando la fórmula  $|r - rg| + |c - cg|$ , apropiada para movimientos en 4 direcciones. A utiliza la función  $f(n) = g(n) + h(n)$ , donde  $g(n)$  es el costo desde inicio y  $h(n)$  la heurística. Usa una cola de prioridad (heap) para elegir el nodo con menor  $f$ , optimizando la búsqueda.

# Conclusiones

Los algoritmos BFS y DFS son simples y útiles dependiendo de la necesidad: BFS para rutas cortas en costos iguales, DFS para exploración profunda. A mejora eficiencia al incorporar heurística. Transformar la matriz a grafo facilita aplicar búsquedas. El manejo correcto de límites, paredes y reconstrucción de ruta es clave para resultados exitosos.



# Gracias