TP2

August 21, 2025

1 Temas Tratados en el Trabajo Práctico 2

- Conceptos de Búsqueda no Informada y Búsqueda Informada.
- Concepto de Heurística.
- Abstracción de Problemas como Gráficos de Árbol.
- Estrategias de Búsqueda no Informada: Primero en Amplitud, Primero en Profundidad y Profundidad Limitada.
- Estrategias de Búsqueda Informada: Búsqueda Voraz, Costo Uniforme, A*.

1.1 Ejercicios Teóricos

1.1.1 1. ¿Qué diferencia hay entre una estrategia de búsqueda Informada y una estrategia de búsqueda No Informada?

Búsqueda No Informada (o Ciega)

- **Descripción:** Estas estrategias de búsqueda no utilizan ninguna información adicional sobre el problema (como la distancia al objetivo) para decidir qué camino explorar. Recorren el espacio de búsqueda de manera sistemática, pero "a ciegas".
- **Características:** Son exhaustivas y garantizan encontrar una solución (si existe), pero pueden ser muy lentas e ineficientes para problemas grandes. No discriminan entre un camino que se acerca al objetivo y uno que se aleja.

Búsqueda Informada (o Heurística)

- **Descripción:** Estas estrategias utilizan una **función heurística** para estimar lo "cerca" que está un estado del objetivo. La heurística guía la búsqueda, priorizando los caminos que parecen más prometedores.
- **Características:** Son mucho más eficientes y rápidas que las no informadas, ya que evitan explorar ramas innecesarias. Sin embargo, no siempre garantizan encontrar la solución óptima si la heurística no es buena.

1.1.2 2. ¿Qué es una heurística y para qué sirve?

Las heurísticas son fundamentales en inteligencia artificial, especialmente en las búsquedas informadas, para resolver problemas demasiado complejos o grandes para ser resueltos con métodos exhaustivos. Su propósito principal es guiar la búsqueda hacia estados que tienen más probabilidades de conducir a una solución, reduciendo significativamente el tiempo y los recursos computacionales.

En esencia, una heurística te permite encontrar una buena solución de forma eficiente, incluso si no es la mejor solución posible.

1.1.3 3. ¿Es posible que un algoritmo de búsqueda no tenga solución?

Sí, esto puede ocurrir por dos razones principales:

1. **El problema no tiene solución:** El objetivo simplemente no es alcanzable desde el estado inicial. Si el algoritmo de búsqueda es completo (es decir, garantiza encontrar una solución si existe), la búsqueda terminará después de explorar todo el espacio de estados sin éxito.
2. **El algoritmo no es completo:** Algunos algoritmos de búsqueda no están diseñados para explorar todo el espacio de estados. Por ejemplo, una búsqueda en profundidad (DFS) puede quedar atrapada en un bucle infinito en un camino sin solución y nunca encontrar la respuesta, incluso si existe en otra rama del árbol.

1.1.4 4. Describa en qué secuencia será recorrido el Árbol de Búsqueda representado en la imagen cuando se aplica un Algoritmo de Búsqueda con la estrategia:

`4.1 Primero en Amplitud.`

Estado Actual	↓					
A	D	F	I			
D	F	I	H	J		
F	I	H	J	C	E	
I	H	J	C	E		
H	J	C	E	P	O	
J	C	E	P	O	K	
C	E	P	O	K	Z	W
E	P	O	K	Z	W	
P	O	K	Z	W		
O	K	Z	W			
K	Z	W	B			
Z	W	B				
W	B					
B						

`4.2 Primero en Profundidad.`

Estado Actual	↓				
A	D	F	I		
D	H	J	F	I	
H	P	O	J	F	I
P	O	J	F	I	
O	J	F	I		
J	K	F	I		

Estado Actual	↓			
K	B	F	I	
B	F	I		
F	C	E	I	
C	Z	W	E	I
Z	W	E	I	
W	E	I		
E	I			
I				

4.3 Primero en Profundidad con Profundidad Limitada Iterativa (comenzando por un nivel de p

Estado Actual	↓											
A	D	F	I									
D	F	I										
F	I											
I												
A	D	H	J	F	C	E	I					
D	H	J	F	C	E	I						
H	J	F	C	E	I							
J	F	C	E	I								
F	C	E	I									
C	E	I										
E	I											
I												
A	D	H	P	O	J	K	F	C	Z	W	E	I
D	H	P	O	J	K	F	C	Z	W	E	I	
H	P	O	J	K	F	C	Z	W	E	I		
P	O	J	K	F	C	Z	W	E	I			
O	J	K	F	C	Z	W	E	I				
J	K	F	C	Z	W	E	I					
K	F	C	Z	W	E	I						
F	C	Z	W	E	I							
C	Z	W	E	I								
Z	W	E	I									
W	E	I										
E	I											
I												
A	D	H	P	O	J	K	B	F	C	Z	W	E	I
D	H	P	O	J	K	B	F	C	Z	W	E	I	
H	P	O	J	K	B	F	C	Z	W	E	I		

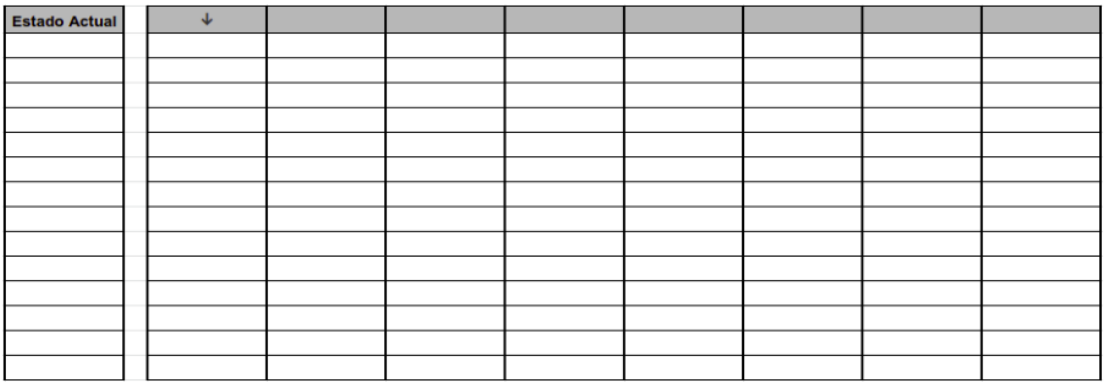
Estado Actual	↓									
P	O	J	K	B	F	C	Z	W	E	I
O	J	K	B	F	C	Z	W	E	I	
J	K	B	F	C	Z	W	E	I		
K	B	F	C	Z	W	E	I			
B	F	C	Z	W	E	I				
F	C	Z	W	E	I					
C	Z	W	E	I						
Z	W	E	I							
W	E	I								
E	I									
I										

[12]:
```
import requests
from PIL import Image
from io import BytesIO
import matplotlib.pyplot as plt

# URL directa de Google Drive
url = "https://drive.google.com/uc?
 ↪export=view&id=1IJDEKWhfMEzXnzr28RgTNOuKBER2NsuP"

# Descargar la imagen
response = requests.get(url)
img = Image.open(BytesIO(response.content))

# Mostrar la imagen
plt.imshow(img)
plt.axis('off')  # Ocultar ejes
plt.show()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[12], line 1
----> 1 import requests
      2 from PIL import Image
      3 from io import BytesIO

ModuleNotFoundError: No module named 'requests'
```

Muestre la respuesta en una tabla, indicando para cada paso que da el agente el nodo que evalúa actualmente y los que están en la pila/cola de expansión según corresponda.

```
[ ]: url = "https://drive.google.com/uc?
      ↪export=view&id=1fW_BgT5muzQffMVRcIiB2mM2Zf66Nb-m"
     response = requests.get(url)
     img = Image.open(BytesIO(response.content))

     # Mostrar con figura más grande
     plt.figure(figsize=(img.width / 80, img.height / 80))  # ajustá el divisor␣
      ↪según densidad deseada
     plt.imshow(img)
     plt.axis('off')
     plt.show()
```



1.2 Ejercicios de Implementación

1.2.1 5. Represente el tablero mostrado en la imágen como un árbol de búsqueda y a continuación programe un agente capaz de navegar por el tablero para llegar desde la casilla I a la casilla F utilizando:

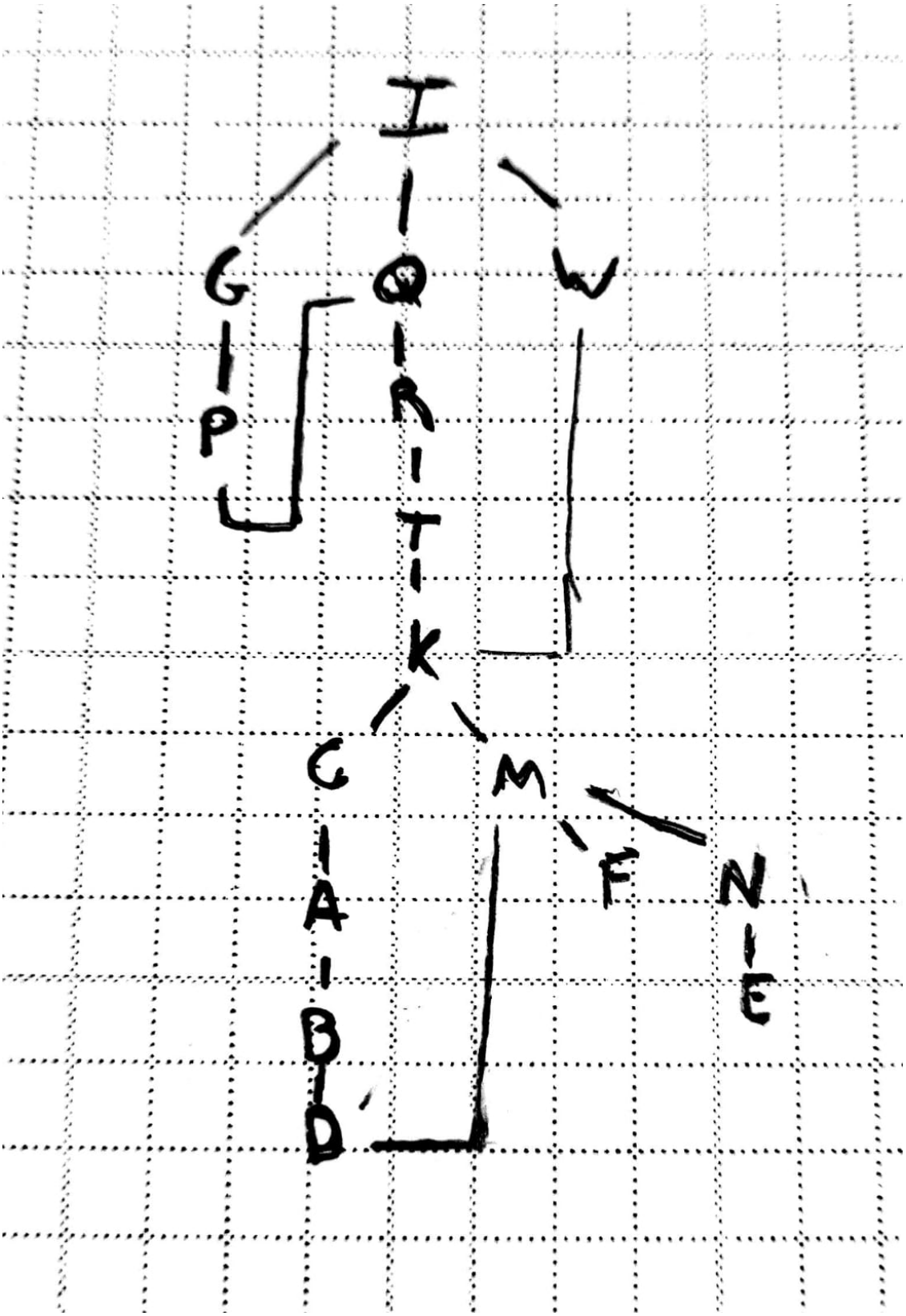
5.1 La estrategia Primero en Profundidad.

5.2 La estrategia Avara.

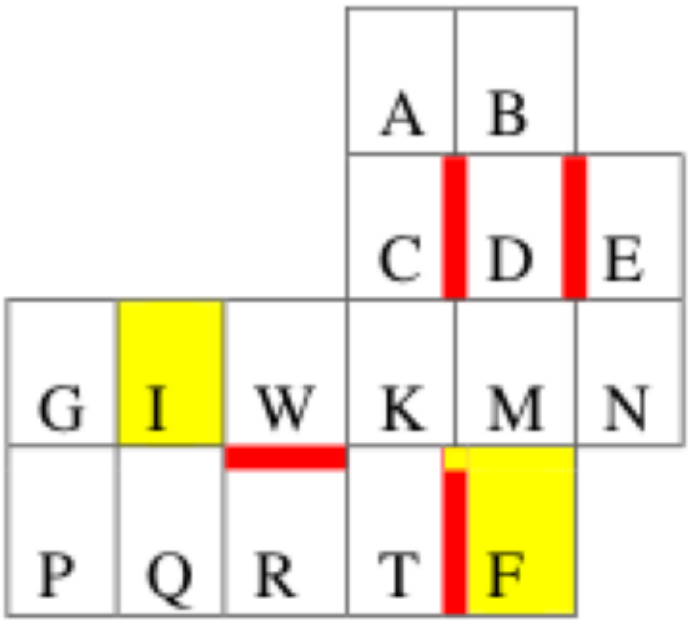
5.3 La estrategia A*.

Considere los siguientes comportamientos del agente:

- El agente no podrá moverse a las casillas siguientes si las separa una pared.
- La heurística empleada en el problema es la Distancia de Manhattan hasta la casilla objetivo (el menor número de casillas adyacentes entre la casilla actual y la casilla objetivo).
- El costo de atravesar una casilla es de 1, a excepción de la casilla W, cuyo costo al atravesarla es 30.
- En caso de que varias casillas tengan el mismo valor para ser expandidas, el algoritmo eligirá en orden alfabético las casillas que debe visitar.


I
G
Q
W
P
R
I
T
I
K
C
M
A
F
N
I
B
E
D

[ ]:
```
url = "https://drive.google.com/uc?
  ↪export=view&id=1FajYiBQ507o6yiE7MndL-PQXyoyELtuD"
response = requests.get(url)
img = Image.open(BytesIO(response.content))
plt.imshow(img)
plt.axis('off')
plt.show()
```



[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from collections import deque
from heapq import heappush, heappop

# 1. Representación del tablero como un grafo de adyacencias
# Las líneas rojas son barreras.
grafo = {
    'A': ['B', 'C'],
    'B': ['A', 'D'],
    'C': ['A', 'K'],
    'D': ['B', 'M'],
    'E': ['N'],
    'F': ['M'],
    'G': ['I', 'P'],
```

```
    'I': ['G', 'W', 'Q'],
    'K': ['W', 'M', 'C', 'T'],
    'M': ['K', 'N', 'D', 'F'],
    'N': ['M', 'E'],
    'P': ['G', 'Q'],
    'Q': ['P', 'I', 'R'],
    'R': ['Q', 'T'],
    'T': ['R', 'K'],
    'W': ['I', 'K']
}

# 2. Heurística: Distancia de Manhattan al objetivo 'F'
# Se define manualmente para este problema.
heuristica = {
    'A': 6, 'B': 6, 'C': 5, 'D': 5, 'E': 5,
    'F': 0, 'G': 5, 'I': 4, 'K': 3, 'M': 2, 'N': 3,
    'P': 4, 'Q': 3, 'R': 2, 'T': 1, 'W': 4
}

# Costo de atravesar una casilla
def costo(nodo):
    """Retorna el costo de atravesar una casilla."""
    return 30 if nodo == 'W' else 1

# 3. Agente de Búsqueda
# 3.1 Estrategia Primero en Profundidad (DFS)
def dfs(inicio, fin):
    """
    Realiza una búsqueda en profundidad para encontrar un camino del inicio al␣
 ↪fin.
    """
    stack = deque([(inicio, [inicio])])
    visitados = set()
    while stack:
        nodo, camino = stack.pop()
        if nodo == fin:
            return camino
        if nodo not in visitados:
            visitados.add(nodo)
            # Ordenar vecinos alfabéticamente para romper empates
            for vecino in sorted(grafo[nodo], reverse=True):
                if vecino not in visitados:
                    stack.append((vecino, camino + [vecino]))
    return None

# 3.2 Estrategia Avara (Greedy Best-First)
def greedy(inicio, fin):
```

```
    """
    Realiza una búsqueda avara para encontrar un camino óptimo.
    Utiliza una cola de prioridad basada en la heurística.
    """
    # La cola de prioridad almacena tuplas: (heuristica, nodo, camino)
    pq = [(heuristica[inicio], inicio, [inicio])]
    visitados = set()
    while pq:
        _, nodo, camino = heappop(pq)
        if nodo == fin:
            return camino
        if nodo not in visitados:
            visitados.add(nodo)
            for vecino in sorted(grafo[nodo]):
                if vecino not in visitados:
                    heappush(pq, (heuristica[vecino], vecino, camino +␣
 ↪[vecino]))
    return None

# 3.3 Estrategia A*
def a_star(inicio, fin):
    """
    Realiza una búsqueda A* para encontrar el camino con el costo más bajo.
    Utiliza una cola de prioridad basada en f(n) = g(n) + h(n).
    """
    # La cola de prioridad almacena tuplas: (f_costo, g_costo, nodo, camino)
    pq = [(heuristica[inicio], 0, inicio, [inicio])]
    visitados = {}  # Almacena el nodo y el costo g más bajo encontrado
    while pq:
        f, g, nodo, camino = heappop(pq)
        if nodo == fin:
            return camino
        # Si ya visitamos el nodo con un costo menor o igual, lo saltamos
        if nodo in visitados and visitados[nodo] <= g:
            continue
        visitados[nodo] = g
        # Ordenar vecinos alfabéticamente para romper empates
        for vecino in sorted(grafo[nodo]):
            g_nuevo = g + costo(vecino)
            f_nuevo = g_nuevo + heuristica[vecino]
            heappush(pq, (f_nuevo, g_nuevo, vecino, camino + [vecino]))
    return None

# 4. Pruebas y Resultados
inicio, fin = 'I', 'F'
print("DFS:   ", dfs(inicio, fin))
print("Greedy:", greedy(inicio, fin))
```

```
print("A*:    ", a_star(inicio, fin))
```

```
DFS:    ['I', 'G', 'P', 'Q', 'R', 'T', 'K', 'C', 'A', 'B', 'D', 'M', 'F']
Greedy: ['I', 'Q', 'R', 'T', 'K', 'M', 'F']
A*:     ['I', 'Q', 'R', 'T', 'K', 'M', 'F']
```

[ ]:
```
DFS:    ['I', 'G', 'P', 'Q', 'R', 'T', 'K', 'C', 'A', 'B', 'D', 'M', 'F']
Greedy: ['I', 'Q', 'R', 'T', 'K', 'M', 'F']
A*:     ['I', 'Q', 'R', 'T', 'K', 'M', 'F']
```

1.2.2 6. Desarrolle un agente que emplee una estrategia de búsqueda A* para ir de una casilla a otra evitando la pared representada, pudiendo seleccionar ustedes mismos el inicio y el final. Muestre en una imagen el camino obtenido.

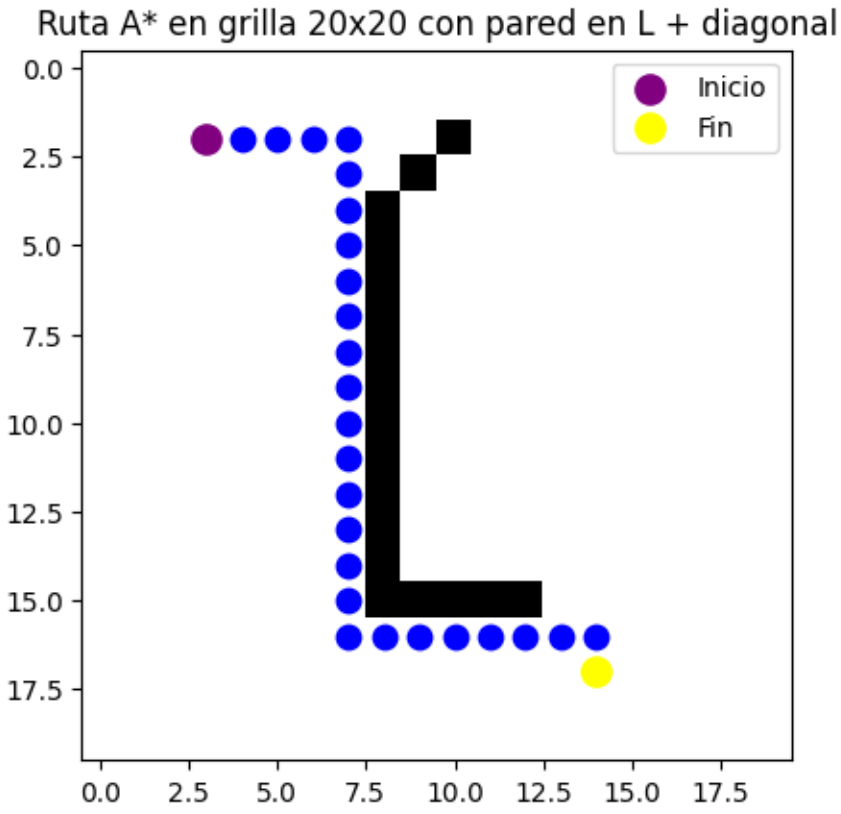
[ ]:
```
url = "https://drive.google.com/uc?
 ↪export=view&id=1fD2Ws5oqFU9_RTj-yX9BIvslXJiqcLCZ"
response = requests.get(url)
img = Image.open(BytesIO(response.content))
plt.imshow(img)
plt.axis('off')
plt.show()
```

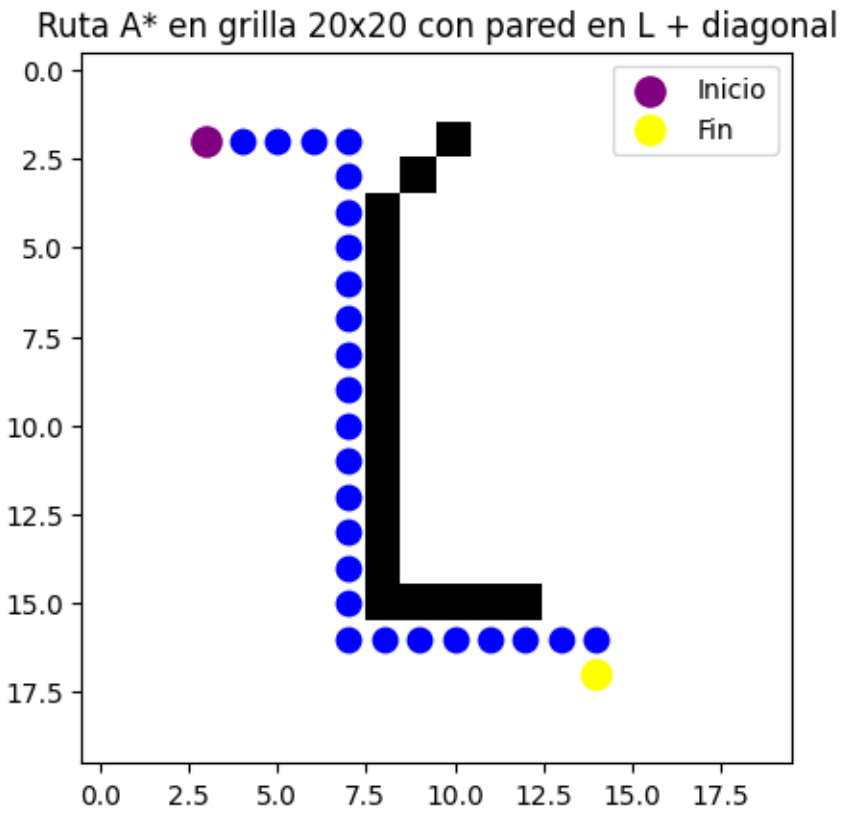
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[11], line 2
      1 url = "https://drive.google.com/uc?
 ↪export=view&id=1fD2Ws5oqFU9_RTj-yX9BIvslXJiqcLCZ"
----> 2 response = requests.get(url)
      3 img = Image.open(BytesIO(response.content))
      4 plt.imshow(img)

NameError: name 'requests' is not defined
```

[ ]:
```
import heapq
import matplotlib.pyplot as plt
import numpy as np

# Crear grilla vacía 20x20 (0 = libre, 1 = pared)
grid = np.zeros((20,20), dtype=int)

# Pared vertical (columna 7, filas 5 a 15)
for i in range(4,15):
    grid[i,8] = 1

# Pared horizontal (fila 15, columnas 3 a 7)
for j in range(8,13):
```

```
    grid[15,j] = 1

# Pared diagonal de 2 cuadros (ejemplo en filas 13-14, columnas 6-7)
grid[2,10] = 1
grid[3,9] = 1

# Heurística Manhattan
def h(a,b):
    return abs(a[0]-b[0]) + abs(a[1]-b[1])

# Algoritmo A*
def astar(grid, start, goal):
    vecinos = [(1,0),(-1,0),(0,1),(0,-1)]
    open_set = []
    heapq.heappush(open_set, (h(start,goal),0,start,[start]))
    visitados = {}
    while open_set:
        f,g,nodo,path = heapq.heappop(open_set)
        if nodo == goal:
            return path
        if nodo in visitados and visitados[nodo] <= g:
            continue
        visitados[nodo] = g
        for dx,dy in vecinos:
            nx,ny = nodo[0]+dx, nodo[1]+dy
            if 0<=nx<grid.shape[0] and 0<=ny<grid.shape[1]:
                if grid[nx,ny] == 0:   # libre
                    g2 = g+1
                    f2 = g2+h((nx,ny),goal)
                    heapq.heappush(open_set,(f2,g2,(nx,ny),path+[(nx,ny)]))
    return None

# ------------------------------
# Elegí el inicio y fin (fila,columna)
inicio = (2,3)
fin    = (17,14)
# ------------------------------

ruta = astar(grid, inicio, fin)

# Dibujar
plt.imshow(grid, cmap="gray_r")  # blanco = libre, negro = pared
for (x,y) in ruta:
    plt.scatter(y, x, c="blue", s=80)
plt.scatter(inicio[1], inicio[0], c="purple", s=120, label="Inicio")
plt.scatter(fin[1], fin[0], c="yellow", s=120, label="Fin")
plt.legend()
```

```
plt.title("Ruta A* en grilla 20x20 con pared en L + diagonal")
plt.show()
```





2 Bibliografía

Russell, S. & Norvig, P. (2004) *Inteligencia Artificial: Un Enfoque Moderno.* Pearson Educación S.A. (2a Ed.) Madrid, España

Poole, D. & Mackworth, A. (2023) *Artificial Intelligence: Foundations of Computational Agents.* Cambridge University Press (3a Ed.) Vancouver, Canada