**Ejercicio de Clase: Best first Search en un Laberinto**

**Yomin Jaramillo Múnera**

**Descripción del Problema:**

Imagina que eres el gerente de logística de una empresa de comercio electrónico que necesita optimizar la distribución de productos a diferentes clientes en una ciudad. La ciudad está representada como una cuadrícula, donde cada celda puede ser una calle transitable o una zona bloqueada (como construcciones o áreas restringidas).

El objetivo es encontrar la ruta más rápida para entregar un pedido desde el **centro de distribución** hasta un **cliente específico**, minimizando la distancia de entrega.

**Reglas:**

1. El laberinto está representado como una lista de listas en Python, donde:

* " " representa un espacio vacío.
* "#" representa una pared.
* "S" es el centro de distribucion
* "E" es el cliente.

1. El repartidos va en carro, por lo que puede moverse en las cuatro direcciones cardinales: arriba, abajo, izquierda y derecha.
2. Tu objetivo es implementar la función find\_exit() que encuentre el camino más corto desde "S" hasta "E" utilizando el algoritmo de Best First Search.

**EJECUCIÓN:**

El profesor le suministrará un código base con el fin de que usted haga las modificaciones pertinentes. Sin embargo se le aconseja seguir las siguientes instrucciones:

1. **Revise de manera detallada el algoritmo de Bucharest:**

* Entienda la definición de la clase Nodo y Problem
* Revisa como se hace el cálculo del costo de las acciones y como son definidas la mismas

1. **Definir la Representación del Problema**:

* Define un nodo que represente la posición del robot en la cuadrícula.
* Define las acciones disponibles (moverse arriba, abajo, izquierda, derecha).

1. **Implementar la Función de Evaluación**:

* La función de evaluación f(nodo) podría ser la distancia de Manhattan desde la posición actual del robot hasta la salida.

1. **Implementar el Algoritmo de Búsqueda del Mejor Primero**:

* Usa una cola de prioridad para la frontera, igual que en el ejemplo de rumania.

1. **Devolver la Solución**:

* Reconstruye el camino desde la posición inicial hasta la salida si se encuentra una solución.

**EJEMPLO DE ENTRADA Y SALIDA:**

