Actividad 2. Agentes Solucionadores de Problemas

Miembros del Equipo

- Paula María De Alba Barrera A01722262
- Carolina Treviño García A00835598
- Natalia Quiroga Colorado A01722353

**PSA **

Description de los estados: 4x4 TABLERO Cuadro [1 - 16]

Nuestro objetivo es limpiar todas las celdas ocupadas moviendo el caballo en forma de 'L'.

Estado inicial: el caballo se ecuentra en la celda que contenga 'H' y las celdas ocupadas con 'O'. Las celdas vacías estan representadas por '-'.

Acciones: el caballo se mueve en L . Dos espacios horizontales y uno vertical. O dos verticales y uno horizontal.

Modelo de Transición : cuando el caballo se cambia a una celda, la celda anterior se marca como vacía(-), y la nueva celda se marca ocupada (H)

Prueba de meta: La meta es limpiar el tablero, dejando solo celdas vacías en el tablero con el caballo. La prueba de meta verifica si no hay más celdas ocupadas ('O') en el tablero.

Costo de cada accion: el costo es igual para cada accion ya que no existe distinción de tiempo y esfuerzo entre ellas.

```
#PROBLEMA 1
!pip install simpleai
from simpleai.search import SearchProblem, breadth_first, astar, uniform_cost, depth_first, greedy

class HorseMovementProblem(SearchProblem):
    def actions(self, state):
        # Encontrar la posición del caballo
        horse_index = state.index('H')
        horse_row, horse_col = divmod(horse_index, 4)
        moves = [
            (-2, -1), (-2, 1),
            (-1, -2), (-1, 2),
            (1, -2), (1, 2),
            (2, -1), (2, 1),
        ]
        # El caballo solo puede moverse en forma L, ents debemos especificar eso
        valid_moves = []
        for move in moves:
            new_row, new_col = horse_row + move[0], horse_col + move[1]
            if 0 <= new_row < 4 and 0 <= new_col < 4:
                new_index = new_row * 4 + new_col
                if state[new_index] == 'O':
                    valid_moves.append(move)
        return valid_moves

    def result(self, state, action):
        new_state = list(state)
        horse_index = state.index('H')
        horse_row, horse_col = divmod(horse_index, 4)

        new_row, new_col = horse_row + action[0], horse_col + action[1]
        new_index = new_row * 4 + new_col

        # Limpiar la celda ocupada y mover el caballo
        new_state[horse_index] = '-'
        new_state[new_index] = 'H'

        return tuple(new_state)

    def is_goal(self, state):
        # Si no hay celdas ocupadas ('O'), hemos limpiado todo el tablero
        return not 'O' in state

    def heuristic(self, state):
        # Una simple heurística podría ser la cantidad de celdas ocupadas restantes
        return state.count('O')
```

```
# Estado inicial: 'H' para el caballo, 'O' para las celdas ocupadas, '-' para las celdas vacías
initial_state = (
    ('-', 'O', '-', '-',
    '-', '-', '-', 'O',
    '-', '-', 'H', '-',
    '-', '-', '-', '-',
    )
)

# Crea la instancia del problema
problem = HorseMovementProblem(initial_state)

# Buscar solución con primero en anchura
solution_breadth_first = breadth_first(problem, graph_search=True)

# Buscar solución con búsqueda A*
solution_astar = astar(problem, graph_search=True)

# Función para imprimir la solución
def print_solution(solution):
    if solution:
        print("Solución:")
        for action, state in solution.path():
            print("Mover a:", action, "Nuevo Tablero:", state)
    else:
        print("No hay solución")


# Imprimir soluciones
print("Primero en anchura:")
print_solution(solution_breadth_first)

print("\nHeurística A*:")
print_solution(solution_astar)
```

```
Collecting simpleai
  Downloading simpleai-0.8.3.tar.gz (94 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 94.4/94.4 kB 1.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: simpleai
  Building wheel for simpleai (setup.py) ... done
  Created wheel for simpleai: filename=simpleai-0.8.3-py3-none-any.whl size=100984 sha256=b0eaf9d08c4ac4ff613c2bb518ca91b7d2
  Stored in directory: /root/.cache/pip/wheels/91/0c/38/421d7910e7bc59b97fc54f490808bdb1097607d83d1a592865
Successfully built simpleai
Installing collected packages: simpleai
Successfully installed simpleai-0.8.3
Primero en anchura:
Solución:
Mover a: None Nuevo Tablero: ('-', 'O', '-', '-', '-', '-', '-', 'O', '-', '-', 'H', '-', '-', '-', '-', '-')
Mover a: (-2, -1) Nuevo Tablero: ('-', 'H', '-', '-', '-', '-', '-', 'O', '-', '-', '-', '-', '-', '-', '-', '-')
Mover a: (1, 2) Nuevo Tablero: ('-', '-', '-', '-', '-', '-', '-', 'H', '-', '-', '-', '-', '-', '-', '-', '-')

Heurística A*:
Solución:
Mover a: None Nuevo Tablero: ('-', 'O', '-', '-', '-', '-', '-', 'O', '-', '-', 'H', '-', '-', '-', '-', '-')
Mover a: (-2, -1) Nuevo Tablero: ('-', 'H', '-', '-', '-', '-', '-', 'O', '-', '-', '-', '-', '-', '-', '-', '-')
Mover a: (1, 2) Nuevo Tablero: ('-', '-', '-', '-', '-', '-', '-', 'H', '-', '-', '-', '-', '-', '-', '-', '-')
```

```
#PROBLEMA 2
from simpleai.search import SearchProblem, breadth_first, astar

class HorseMovementProblem(SearchProblem):
  def actions(self, state):
        horse_index = state.index('H')
        horse_row, horse_col = divmod(horse_index, 4)
        moves = [(-2, -1), (-2, 1), (-1, -2), (-1, 2), (1, -2), (1, 2), (2, -1), (2, 1)]

        valid_moves = []
        for move in moves:
            new_row, new_col = horse_row + move[0], horse_col + move[1]
            if 0 <= new_row < 4 and 0 <= new_col < 4:
                new_index = new_row * 4 + new_col
                if state[new_index] == 'O':
                    valid_moves.append(move)
        return valid_moves

  def result(self, state, action):
        new_state = list(state)
        horse_index = state.index('H')
        horse_row, horse_col = divmod(horse_index, 4)

        new_row, new_col = horse_row + action[0], horse_col + action[1]
        new_index = new_row * 4 + new_col

        new_state[horse_index] = '-'
        new_state[new_index] = 'H'

        return tuple(new_state)

  def is_goal(self, state):
        return not 'O' in state

  def heuristic(self, state):
        return state.count('O')

# Función para realizar la búsqueda manual y devolver el camino seguido

def manual_search(problem):
    frontier = [(problem.initial_state, [])]  # Estado inicial y camino vacío
    explored = set()

    while frontier:
        state, path = frontier.pop(0)
        explored.add(state)
        if problem.is_goal(state):
            return path  # Si el estado es el objetivo, devuelve el camino.

        for action in problem.actions(state):
            child = problem.result(state, action)
            if child not in explored and child not in [s for s, p in frontier]:
                frontier.append((child, path + [action]))  # Añade el estado hijo y el camino a la frontera.

    return path  # Devuelve el último camino si no se encontró una solución.

# Estado inicial
initial_state = (
    '-', 'O', '-', '-',
    'O', '-', '-', 'O',
    '-', '-', '-', '-',
    '-', 'O', '-', 'H'
)

# Crear la instancia del problema
problem = HorseMovementProblem(initial_state)

# Realizar la búsqueda manual
path_followed = manual_search(problem)

# Función para imprimir el camino seguido
def print_path(path, problem):
    state = problem.initial_state
    print("Estado inicial:", state)
    for action in path:
        state = problem.result(state, action)
        print("Mover a:", action, "Nuevo estado:", state)
```

```
    if problem.is_goal(state):
        print("Se ha alcanzado una solución!")
    else:
        print("No se encontró una solución completa. Se realizaron todos los movimientos posibles.")

# Imprimir el camino seguido
print_path(path_followed, problem)
```

```
Estado inicial: ('-', 'O', '-', '-', 'O', '-', '-', 'O', '-', '-', '-', '-', '-', 'O', '-', 'H')
No se encontró una solución completa. Se realizaron todos los movimientos posibles.
```

```
#PROBLEMA 3
from simpleai.search import SearchProblem, breadth_first, astar

class HorseMovementProblem(SearchProblem):
  def actions(self, state):
        horse_index = state.index('H')
        horse_row, horse_col = divmod(horse_index, 4)
        moves = [(-2, -1), (-2, 1), (-1, -2), (-1, 2), (1, -2), (1, 2), (2, -1), (2, 1)]

        valid_moves = []
        for move in moves:
            new_row, new_col = horse_row + move[0], horse_col + move[1]
            if 0 <= new_row < 4 and 0 <= new_col < 4:
                new_index = new_row * 4 + new_col
                if state[new_index] == 'O':
                    valid_moves.append(move)
        return valid_moves

  def result(self, state, action):
        new_state = list(state)
        horse_index = state.index('H')
        horse_row, horse_col = divmod(horse_index, 4)

        new_row, new_col = horse_row + action[0], horse_col + action[1]
        new_index = new_row * 4 + new_col

        new_state[horse_index] = '-'
        new_state[new_index] = 'H'

        return tuple(new_state)

  def is_goal(self, state):
        return not 'O' in state

  def heuristic(self, state):
        return state.count('O')

# Función para realizar la búsqueda manual y devolver el camino seguido

def manual_search(problem):
    frontier = [(problem.initial_state, [])]  # Estado inicial y camino vacío
    explored = set()

    while frontier:
        state, path = frontier.pop(0)
        explored.add(state)
        if problem.is_goal(state):
            return path  # Si el estado es el objetivo, devuelve el camino.

        for action in problem.actions(state):
            child = problem.result(state, action)
            if child not in explored and child not in [s for s, p in frontier]:
                frontier.append((child, path + [action]))  # Añade el estado hijo y el camino a la frontera.

    return path  # Devuelve el último camino si no se encontró una solución.

# Estado inicial
initial_state = (
    'O', 'O', '-', '-',
    'O', '-', 'H', 'O',
    '-', 'O', '-', '-',
    '-', 'O', '-', 'O'
)
```

```
# Crear la instancia del problema
problem = HorseMovementProblem(initial_state)

# Realizar la búsqueda manual
path_followed = manual_search(problem)

# Función para imprimir el camino seguido
def print_path(path, problem):
    state = problem.initial_state
    print("Estado inicial:", state)
    for action in path:
        state = problem.result(state, action)
        print("Mover a:", action, "Nuevo estado:", state)

    if problem.is_goal(state):
        print("Se ha alcanzado una solución!")
    else:
        print("No se encontró una solución completa. Se realizaron todos los movimientos posibles.")

# Imprimir el camino seguido
print_path(path_followed, problem)
```

```
Estado inicial: ('O', 'O', '-', '-', 'O', '-', 'H', 'O', '-', 'O', '-', '-', '-', 'O', '-', 'O')
Mover a: (2, 1) Nuevo estado: ('O', 'O', '-', '-', 'O', '-', '-', 'O', '-', 'O', '-', '-', '-', 'O', '-', 'H')
Mover a: (-1, -2) Nuevo estado: ('O', 'O', '-', '-', 'O', '-', '-', 'O', '-', 'H', '-', '-', '-', 'O', '-', '-')
Mover a: (-1, 2) Nuevo estado: ('O', 'O', '-', '-', 'O', '-', '-', 'H', '-', '-', '-', '-', '-', 'O', '-', '-')
Mover a: (-1, -2) Nuevo estado: ('O', 'H', '-', '-', 'O', '-', '-', '-', '-', '-', '-', '-', '-', 'O', '-', '-')
No se encontró una solución completa. Se realizaron todos los movimientos posibles.
```

Reflexión y Conclusiones:

Para esta actividad nos dimos cuenta que debíamos ser muy especificas con las instrucciones ya que tuvimos varias dificultades. Comenzamos con el primer ejercicio que era el más sencillo y todo parecía funcionar bien. Despues vimos que si al moverse el caballo no encontraba una celda ocupada arrojaba que no habia solución entonces modificamos el código para que vaya mostrando los movimientos que hacia el caballo antes de ocncluir que no había solución. En el ejercicio tres fue lo que se mostro. Se eliminaron las primeras 5 celdas ocupadas pero aun quedaban dos que ya no se pudieron eliminar entonces el código arrojo que no se encontro una solución completa. Nuestra mayor dificultad fue en el ejercicio 2 ya que desde el principio arroja que no hay movimientos ni soluciones. Esto pasa porque el caballo no tiene ni una celda ocupada a la cual pueda llegar entonces por eso el código arroja ese resultado. Despúes de realizar este código aprendimos que hay muchas maneras de intentar resolver un mismo problema y como trabajamos en equipo llegamos a tener varias diferencias de opinion. Pero al final la programación del PSA resulto ser un aprendizaje para todas. Lo más fácil fue utilizar la biblioteca simpleai, que facilitó la implementación de diferentes estrategias de búsqueda, permitiéndonos concentrarnos en los detalles específicos del problema en lugar de las complejidades de bajo nivel del algoritmo.

La búsqueda en anchura asegura que se encuentren soluciones existentes, explorando sistemáticamente todos los posibles movimientos en cada nivel de profundidad. Sin embargo, este método puede ser costoso en términos de memoria y no es necesariamente eficiente en cuanto a tiempo, especialmente en un tablero con muchas celdas vacías.

En cambio, el enfoque de la búsqueda A* se destacó por su eficiencia. La heurística de contar las celdas ocupadas restantes resultó ser útil para guiar la búsqueda hacia estados más prometedores, lo que a menudo llevaba a soluciones más rápidas. Sin embargo, la efectividad de A* depende en gran medida de qué tan buena es la heurística.

Por ultimo, el desarrollo del PSA nos enseñó la importancia de una planificación cuidadosa y un diseño algorítmico inteligente. La selección del algoritmo correcto y la heurística adecuada son fundamentales para resolver problemas de búsqueda de manera eficiente. Los retos enfrentados en el camino proporcionaron una experiencia de aprendizaje valiosa que va más allá de la simple codificación.