INTELIGENCIA ARTIFICIAL: PRÁCTICA I

Daranuta Zob Cristian

55261830Y

Esquinas Fernández Guillermo

49750576N

02/11/2025

PARTE I. KIWIS AND DOGS PROBLEM

1. get_start_states()

Función: Define el estado inicial.

- Salida: Lista con estado inicial: 2 kiwis en nodos D y F; 1 perro en nodo C
- Lógica: Retorna hardcoded el estado inicial del problema

2. is_goal_state(state)

- Entrada: Estado actual
- Salida: True si todos los kiwis están en A y todos los perros en E
- Lógica: Comprueba con all() que cada animal esté en su destino

3. is_valid_state(state)

- Entrada: Estado actual
- Salida: True (siempre válido)
- Lógica: Sin restricciones de cantidad de animales por nodo

4. get_valid_moves(current_pos, state)

Función: Retorna movimientos posibles desde una posición.

- Entrada: Posición actual y estado
- Salida: Lista de tuplas (destino, costo)
- Lógica:
    1. Busca en el grafo aristas desde current_pos
    2. Evalúa condiciones ("nobody(X)" o "somebody(X)")
    3. Filtra movimientos válidos según estado actual

5. move_kiwi(state, kiwi_id, destination) & move_dog(state, dog_id, destination)

Función: Mueve un kiwi o perro a un destino.

- Entrada: Estado, ID del animal, nodo destino
- Salida: (costo, nuevo_estado) o None si inválido
- Lógica:
    1. Obtiene posición actual del animal
    2. Verifica si destino es accesible
    3. Retorna costo dinámico y nuevo estado

5. Analisis Datos

Algoritmo	Grafo/Árbol	Exitoso	Nodos Expandidos	Costo	Fringe Máx.
UCS	Grafo	✓	253	77	63
BFS	Grafo	✓	245	77	45
DFS	Grafo	✓	226	171	66
UCS	Árbol	✗	—	—	—
BFS	Árbol	✗	—	—	—
DFS	Árbol	✗	—	—	—

1. La búsqueda por Grafo permite que los algoritmos terminen correctamente, mientras que la búsqueda en Árbol falla la cantidad de nodos que expande y la RAM explota
2. Rendimiento:
   - 2.1. BFS es el algoritmo más equilibrado, ya que encuentra resultado con coste óptimo, y usa poca memoria.
   - 2.2. UCS aunque usa más memoria, asegura encontrar una solución optimal
   - 2.3. DFS aunque es el más eficiente en cuanto a expandir nodos, el coste que encuentra no es óptima

PARTE II. N QUEENS PROBLEM

1. get_start_states()

Función: Genera estado inicial aleatorio.

- Salida: Lista con una tupla de N reinas en posiciones aleatorias (0 a board_size-1)
- Lógica: random.randint() para cada reina, seed controlado

2. is_goal_state(state)

Función: Verifica si el tablero está sin conflictos.

- Entrada: Estado (tupla de posiciones de reinas)
- Salida: True si todas las reinas tienen 0 conflictos
- Lógica: Comprueba con all() que cada reina no ataque a otras

3. is_valid_state(state)

Función: Valida que todas las reinas estén dentro del tablero.

- Entrada: Estado actual
- Salida: True si todas las filas están en rango [0, n-1]
- Lógica: Itera sobre cada posición y verifica límites

4. conflicts(state, col)

Función: Cuenta conflictos de una reina específica.

- Entrada: Estado y columna de la reina
- Salida: Número de conflictos (0 si aislada)
- Lógica:
    1. Compara con todas las otras reinas
    2. Detecta ataques horizontales: state[col] == state[col2]
    3. Detecta ataques diagonales: |col - col2| == |fila - fila2|

5. move_queen(state, col, new_row)

Función: Mueve una reina a nueva fila.

- Entrada: Estado, columna de reina (0 a N-1), nueva fila (0 a N-1)
- Salida: Nuevo estado o None si movimiento innecesario
- Lógica:
    1. Verifica si la posición cambió
    2. Crea copia del estado y actualiza posición

6. RepairHeuristic.compute(state)

Función: Heurística admisible para A*.

- Entrada: Estado actual
- Salida: Total de conflictos en el tablero (suma de todos)
- Lógica: Suma conflictos de todas las reinas
- Admisibilidad: Nunca sobrestima (mínimo 1 movimiento por conflicto)

6. Analisis Datos

N	Tipo	Algoritmo	Heurística	Exitosos	Nodos Exp	Fringe
4	Tree	A*	Conflictos	5/5	2-56	23-34
4	Tree	UCS	Ninguna	5/5	107-1266	1178-12486
4	Graph	A*	Conflictos	5/5	2-20	20-41
4	Graph	UCS	Ninguna	5/5	45-127	122-128
6	Graph	A*	Conflictos	5/5	3-56	77-1146
6	Graph	UCS	Ninguna	5/5	367-31250	2600-31250
8	Tree	A*	Conflictos	5/5	4-101	221-5556
8	Tree	UCS	Ninguna	0/5	—	—
8	Graph	A*	Conflictos	5/5	4-32	198-1418
8	Graph	UCS	Ninguna	5/5	41517-900619	315805-3329571
10	Graph	A*	Conflictos	5/5	12-68	955-5189
10	Graph	UCS	Ninguna	0/5	—	—

1. A* vs UCS, la diferencia simplemente tiene un crecimiento exponencial según la N. La heurística simplemente hace que el problema sea viable.
2. Para problemas pequeños, todos los algoritmos lo pueden solucionar, pero llega un punto en que UCS simplemente pierde la batalla
3. Igual que con los Kiwis, la búsqueda en Grafo simplemente es mejor, aunque aquí se encontró alguna solución, llega un punto que es inviable
4. UCS simplemente no es una opción para resolver este problema, porque como todos los movimientos tienen el mismo coste, sin ventaja.

PARTE III. PACMAN PROBLEM

1. PacmanManhattanHeuristic

Función: Distancia Manhattan a la comida.

- Entrada: Estado (posición Pacman, posición comida)
- Salida: Número de pasos mínimos en cuadrícula
- Admisibilidad: Nunca sobrestima (camino real ≥ Manhattan)

2. PacmanEuclideanHeuristic

Función: Distancia Euclidiana a la comida.

- Entrada: Estado (posición Pacman, posición comida)
- Salida: Distancia en línea recta

- Admisibilidad: Nunca sobrestima (camino real ≥ línea recta)

3. Analisis Datos

Laberinto	Tipo	Algoritmo	Heuristica	OK	Nodos	Costo
testMaze	Graph	A*	Manhattan	✓	7	7
testMaze	Graph	A*	Euclidean	✓	7	7
testMaze	Graph	UCS	Ninguna	✓	7	7
testMaze	Graph	BFS	Ninguna	✓	7	7
testMaze	Tree	A*	Manhattan	✓	7	7
testMaze	Tree	A*	Euclidean	✓	7	7
testMaze	Tree	UCS	Ninguna	✓	43	7
testMaze	Tree	BFS	Ninguna	✓	24	7
testMaze	Tree	IDS	Ninguna	✓	93	7
smallMaze	Graph	A*	Manhattan	✓	53	19
smallMaze	Graph	A*	Euclidean	✓	56	19
smallMaze	Graph	UCS	Ninguna	✓	92	19
smallMaze	Tree	A*	Manhattan	✓	1332	19
smallMaze	Tree	A*	Euclidean	✓	3331	19
smallMaze	Tree	IDS	Ninguna	✓	4.695.373	19
smallMaze	Tree	BFS	Ninguna	✓	5.687.100	19
mediumMaze	Graph	A*	Manhattan	✓	220	68
mediumMaze	Graph	A*	Euclidean	✓	225	68
mediumMaze	Graph	UCS	Ninguna	✓	270	68
mediumMaze	Tree	A*	Manhattan	✗	—	—
mediumMaze	Tree	A*	Euclidean	✗	—	—
mediumMaze	Tree	UCS	Ninguna	✗	—	—
mediumMaze	Tree	BFS	Ninguna	✗	—	—
mediumMaze	Tree	IDS	Ninguna	✗	—	—
bigMaze	Graph	A*	Manhattan	✓	548	210
bigMaze	Graph	A*	Euclidean	✓	556	210
bigMaze	Graph	UCS	Ninguna	✓	622	210
bigMaze	Tree	A*	Manhattan	✗	—	—
bigMaze	Tree	A*	Euclidean	✗	—	—
bigMaze	Tree	UCS	Ninguna	✗	—	—
bigMaze	Tree	BFS	Ninguna	✗	—	—
bigMaze	Tree	IDS	Ninguna	✗	—	—

1. La búsqueda en Grafo sigue siendo muy superior comparada con la búsqueda por Árbol, debido a que es muy propenso a entrar en bucles infinitos. Por lo que en mazmorras grandes, simplemente no es una buena opción.
2. La heurística de Manhattan expande menos nodos que la Euclidiana.
3. Respecto a los algoritmos:
    3.1. A* es el más eficiente
    3.2. UCS y BFS trabajan bien en Grafo
    3.3. IDS simplemente no es viable, solo trabaja bien en escenarios pequeños