8-Puzzle Con Heurísticas

Profesor: Angel Augusto Agudelo Zapata

Andres Felipe Gordillo Guerrero

1088348241

a.gordillo@utp.edu.co

Inteligencia artificial

Universidad tecnológica de pereira

2024

Código para Resolver el 8-Puzzle y Visualizar su Solución

Este código implementa un solucionador para el problema del 8-puzzle utilizando el algoritmo A*. El árbol de soluciones se visualiza con una interfaz gráfica que permite mover nodos y reorganizarlos en una disposición binaria. A continuación se detalla el propósito y funcionamiento de cada función en el código:

1. Funciones Heurísticas

- **`hamming(state, goal)`**
  - **Descripción:** Calcula la heurística de Hamming, que mide la cantidad de piezas que están en posiciones incorrectas en el estado actual con respecto al estado objetivo.
  - **Entrada:** `state` (estado actual), `goal` (estado objetivo)
  - **Salida:** Número de piezas en posiciones incorrectas.
- **`manhattan(state, goal)`**
  - **Descripción:** Calcula la heurística de Manhattan, que suma las distancias absolutas de cada pieza desde su posición actual a su posición en el estado objetivo.
  - **Entrada:** `state` (estado actual), `goal` (estado objetivo)
  - **Salida:** Suma de las distancias de Manhattan para todas las piezas.

2. Funciones de Estado del Puzzle

- **`is_goal(state, goal)`**
  - **Descripción:** Verifica si el estado actual es el estado objetivo.
  - **Entrada:** `state` (estado actual), `goal` (estado objetivo)
  - **Salida:** `True` si el estado actual es el objetivo, de lo contrario `False`.
- **`neighbors(state)`**
  - **Descripción:** Encuentra los posibles estados vecinos generados por los movimientos del espacio vacío en el puzzle.
  - **Entrada:** `state` (estado actual)
  - **Salida:** Lista de estados vecinos generados por movimientos válidos.

*3. Algoritmo A para Resolver el Puzzle**

- **`a_star(initial_state, goal_state, heuristic)`**
  - **Descripción:** Implementa el algoritmo A* para encontrar la solución del puzzle. Usa la heurística proporcionada (Hamming o Manhattan) para guiar la búsqueda.
  - **Entrada:** `initial_state` (estado inicial), `goal_state` (estado objetivo), `heuristic` (función heurística)
  - **Salida:** Diccionario `came_from` que rastrea el camino, y `cost_so_far` que guarda los costos asociados a cada estado.
- **`reconstruct_path(came_from, start, goal)`**

- **Descripción:** Reconstruye el camino desde el estado inicial hasta el estado objetivo utilizando el diccionario `came_from`.
- **Entrada:** `came_from` (diccionario de seguimiento de estados), `start` (estado inicial), `goal` (estado objetivo)
- **Salida:** Lista de estados que forman el camino desde el inicio hasta el objetivo.

4. Funciones de Interfaz y Animación

- **`clear_console()`**
  - **Descripción:** Limpia la consola para actualizar la visualización durante la animación.
  - **Entrada:** Ninguna
  - **Salida:** Ninguna
- **`display_grid(state)`**
  - **Descripción:** Muestra el estado del puzzle en formato de cuadrícula en la consola.
  - **Entrada:** `state` (estado del puzzle)
  - **Salida:** Impresión del estado del puzzle en la consola.
- **`animate_console_solution(solution, delay=1)`**
  - **Descripción:** Anima la solución del puzzle en la consola mostrando cada estado en la lista de soluciones con un retraso definido.
  - **Entrada:** `solution` (lista de estados de la solución), `delay` (retraso entre estados)
  - **Salida:** Animación en la consola de los estados del puzzle.

5. Funciones de Entrada y Visualización del Árbol

- **`input_puzzle()`**
  - **Descripción:** Permite al usuario ingresar manualmente el estado inicial del puzzle y valida la entrada.
  - **Entrada:** Ninguna
  - **Salida:** Estado inicial del puzzle en forma de matriz `np.array`.
- **`create_puzzle_tree(path)`**
  - **Descripción:** Crea un árbol dirigido que representa la solución del puzzle usando los estados en el camino.
  - **Entrada:** `path` (lista de estados en el camino de la solución)
  - **Salida:** Grafo dirigido (`DiGraph`) que representa el árbol de soluciones.
- **`format_state(state)`**
  - **Descripción:** Formatea un estado del puzzle en una cadena para su visualización.
  - **Entrada:** `state` (estado del puzzle)
  - **Salida:** Cadena que representa el estado del puzzle.
- **`organize_binary_tree(G, pos)`**
  - **Descripción:** Reorganiza el árbol en una disposición binaria.
  - **Entrada:** `G` (grafo dirigido del árbol), `pos` (posición actual de los nodos)

    - **Salida:** Nuevas posiciones para los nodos en una disposición binaria.
- `plot_tree(G)`
    - **Descripción:** Dibuja el árbol de soluciones usando `networkx` y `matplotlib`, permitiendo mover nodos con el mouse y reorganizar el árbol en formato binario con un botón.
    - **Entrada:** `G` (grafo dirigido del árbol)
    - **Salida:** Visualización interactiva del árbol de soluciones.

6. Función Principal

- `main()`
    - **Descripción:** Función principal que ejecuta el programa. Permite al usuario elegir el método de resolución, ejecuta el algoritmo A*, muestra la solución en la consola o como animación, y visualiza el árbol de soluciones.
    - **Entrada:** Ninguna (interacción con el usuario para elegir opciones)
    - **Salida:** Resultado de la solución del puzzle, visualización del árbol de soluciones.