INTELIGENCIA ARTIFICIAL

Búsquedas en AIMA-Java y herramienta Search.

Descripción breve

En este document se resuelven las cuestiones relativas al 4º Homework de la asignatura de Inteligencia Artificial.

Sebastian Pedrosa Granados y Germán Alejo Domínguez

Tabla de contenido

1. Utilizar las técnicas de búsqueda no informada vistas en clases de teoría que estén incluidas en AIMA y mostrar los resultados obtenidos.

1.1. Definición del problema.

Estados: Representación de un estado

Los estados se representan mediante el uso de un array bidimensional en el que cada posición podrá tomar los siguientes valores:

- Vacío: Una posición vacía indica un camino que el agente puede recorrer.
- "%": El carácter porcentaje indica la existencia de una pared/obstáculo en el laberinto y, por tanto, no será posible recorrer dicha posición.
- "K": El carácter K corresponde al primer hito que debe alcanzar el agente.
- "G": El carácter G corresponde con el Goal u objetivo del agente, no podrá alcanzar este objetivo sin haber alcanzado primero la posición del carácter K.

Estado inicial: hay un origen, un punto intermedio y un objetivo pero no hay ningún punto recorrido en el inicio.

Acciones: Descripción de las posibles acciones disponibles para el agente

Las acciones posibles para el agente serán:

- UP: moverse hacia arriba en el laberinto, situándose dentro en la misma posición fila pero en una posición inferior de columna siempre y cuando el valor de la columna sea superior a 0.
- DOWN: moverse hacia abajo en el laberinto, situándose dentro en la misma posición fila pero en una posición superior de columna siempre y cuando el valor de la columna sea inferior al tamaño total del vector columna.
- RIGHT: moverse hacia la derecha en el laberinto, situándose dentro en la misma posición columna pero en una posición superior de fila siempre y cuando el valor de la fila sea inferior al tamaño del vector fila.
- LEFT: moverse hacia la izquierda en el laberinto, situándose dentro en la misma posición columna pero en una posición inferior de fila siempre y cuando el valor de la fila sea superior a 0.

Modelo de transición: Estado resultante de realizar una acción en un estado determinado.

En cada movimiento, la posición recorrida se marcará con un punto. De esta forma se definirá el camino que ha sido recorrido por el agente.

Coste: En nuestro ejercicio el coste de un cambio de estado vine dado por el tiempo de cruce.

Al no indicarse lo contrario, el coste es 1, por lo que el coste del camino será el número de pasos dados por el agente.

Observaciones: Primero se resolverá por búsqueda no informada empleando un algoritmo de búsqueda en anchura y un algoritmo de búsqueda en profundidad. A continuación, se identificarán dos heurísticas admisibles para resolverlo utilizando una búsqueda informada. Además, hemos considerado que los laberintos probados deben ser de tamaño NxN, es decir, que corresponden a matrices cuadradas.

1.2. Resultados obtenidos.

```
LaberintoDemo breadth -->
Action[name==up]
Action[name==up]
Action[name==up]
Action[name==right]
Action[name==left]
Action[name==down]
Action[name==down]
Action[name==down]
Action[name==right]
Action[name==right]
Action[name==right]
Action[name==up]
Action[name==up]
Action[name==up]
pathCost : 14.0
nodesExpanded : 7246
queueSize : 6673
maxQueueSize : 6673

LaberintoDemo Depth -->
Action[name==up]
Action[name==up]
Action[name==up]
Action[name==right]
Action[name==left]
Action[name==down]
Action[name==down]
Action[name==down]
Action[name==right]
Action[name==right]
Action[name==right]
Action[name==up]
Action[name==up]
Action[name==up]
pathCost : 14.0
nodesExpanded : 20
queueSize : 0
maxQueueSize : 2
```

1.3. ¿Qué criterio sigue AIMA a la hora de elegir el camino a seguir en la búsqueda en profundidad y en anchura? (Esta cuestión se refiere al orden de expansión de los nodos hijos)

En la búsqueda en profundidad el algoritmo expande los nodos hijos hasta llegar al final del árbol, mientras que en la búsqueda en anchura el algoritmo expande los nodos vecinos, comenzando desde la raíz, visitando en cada nodo los nodos vecinos de este hasta que se recorra todo el árbol.

2. Implementar dos funciones heurísticas admisibles e indicar cuál debería funcionar mejor. Además, emplearlas en las búsquedas informadas indicadas en el tema correspondiente y mostrar los resultados obtenidos.

2.1. Definición de heurísiticas.

En primer lugar, hemos considerado como una primera heurística admisible el empleo de la Distancia Manhattan cómo forma de calcular la distancia entre la posición actual del agente y el objetivo final.

Para el cálculo de la Distancia Manhattan hemos empleado lo siguiente:

Math.abs(board.getGoalx() - board.getRow() + board.getGoaly() - board.getCol());

Por otro lado, hemos considerado como una segunda heurística admisible el empleo del número de posiciones hasta el objetivo como una forma de calcular la distancia desde la posición actual hasta este.

Para el cálculo de esta distancia hemos implementado el siguiente código:

```
for(int i = 0 ; i < state.length - 1;i++){
        for(int j=0; j<state[i].length -1 ;j++){

           if(state[i][j] == ' '){
              left++;
           }

        }
     }
```

2.2. Resultados obtenidos.

```
LabyrinthDemo A* -->
Action[name==up]
Action[name==up]
Action[name==up]
Action[name==right]
Action[name==left]
Action[name==down]
Action[name==down]
Action[name==down]
Action[name==right]
Action[name==right]
Action[name==right]
Action[name==up]
Action[name==up]
Action[name==up]
pathCost : 14.0
nodesExpanded : 20
queueSize : 0
maxQueueSize : 2

LabyrinthDemo A* -->
Action[name==up]
Action[name==up]
Action[name==up]
Action[name==right]
Action[name==left]
Action[name==down]
Action[name==down]
Action[name==down]
Action[name==right]
Action[name==right]
Action[name==right]
Action[name==up]
Action[name==up]
Action[name==up]
pathCost : 14.0
nodesExpanded : 20
queueSize : 0
maxQueueSize : 2
```

2.3. Mejor heurística

A priori ambas heurísticas funcionan de forma similar, pero, al implementarse la segunda mediante un doble bucle for, la primera resulta menos costosa computacionalmente y resulta más beneficiosa.

3. Crear con la herramienta Search el espacio de búsqueda sobre el que se podrán realizar los posibles recorridos para el siguiente ejemplo de laberinto.