29 DE OCTUBRE DE 2019

MEMORIA PRÁCTICA 1

INTELIGENCIA ARTIFICIAL 2019-2020

PEDRO TAMARGO ALLUÉ - 758267

Contenido

Problema de los Caníbales y los Misioneros

El problema de los caníbales y los misioneros consiste en trasladar 3 caníbales y 3 misioneros de una orilla del río a otra, contando con la restricción de que no pueden producirse estados donde el número de caníbales de una orilla es mayor que el de los misioneros de esa orilla.

Para la resolución del problema se plantea el siguiente estado:

$$(nCan_{izq}, nMis_{izq}, pos_{bote}, nCan_{der}, nMis_{der})$$

Dónde, $nCan_x$ es el número de caníbales en la orilla del río (izq, der), $nMis_x$ es el número de misioneros en la orilla del río (izq, der) y pos_{bote} es la posición del bote en el estado, si $pos_{bote} = 0 \rightarrow bote\ en\ orilla\ izquierda$, en caso contrario, el bote se encuentra en la orilla derecha.

Por lo tanto, el estado inicial sería:

$$(3,3,0,0,0)$$

Para definir las transiciones posibles se plantea el siguiente predicado auxiliar:

$$noPeligroso(s) = \left(nCan_{izq} \leq nMis_{izq}\ or\ nMis_{izq} = 0\right)\ and\ (nCan_{der} \leq nMis_{der}\ or\ nMis_{der} = 0)$$

Siendo s el estado resultante de realizar el movimiento, y $noPeligroso(s)$ el predicado que indica si un estado es peligroso o no, en base a la restricción planteada por el problema.

Los movimientos posibles para este problema serían:

$$(MOVER1C, MOVER2C, MOVER1M, MOVER2M, MOVER1C1M)$$

Por lo tanto, se definen las condiciones para poder realizar un movimiento de la siguiente manera:

$$canMoveBoat(\text{MOVER1C}) = \left(bote_{izq}\ and\ nCan_{izq} \geq 1\ and\ noPeligroso(s_1)\right)\ or\ \left(not\ bote_{izq}\ and\ nCan_{der} \geq 1\ and\ noPeligroso(s_2)\right)$$

Siendo $bote_{izq}$ cierto si $pos_{bote} = 0$, es decir, si el bote está a en la orilla izquierda, s_1 el estado que se alcanzaría después de mover un caníbal de la orilla izquierda a la derecha y s_2 el estado que se alcanzaría después de mover un caníbal de la orilla derecha a la izquierda.

Sería análogo para el resto de movimientos posibles $(MOVER2C, MOVER1M, MOVER2M, MOVER1C1M)$.

El estado final para este problema sería que todos los caníbales, los misioneros y el bote estuvieran en la orilla derecha $(0,0,1,3,3)$.

Los ficheros pueden encontrarse en el paquete “aima.core.environment.Canibales” y en el fichero “CanibalesDemoPract1.java” del paquete “aima.gui.demo.search”

Comparación de ejecución del problema 8-puzzle con distintos algoritmos de búsqueda

Tabla comparativa de tiempos entre distintos algoritmos

Problema	Profundidad	Expand	Q.Size	MasQS	tiempo
BFS-G-3	3	5	4	5	33
BFS-T-3	3	6	9	10	1
DFS-G-3	59123	120491	39830	42913	2836
DFS-T-3	---	---	---	---	(1)
DLS-9-3	9	10	0	0	4
DLS-3-3	3	4	0	0	0
IDS-3	3	9	0	0	0
UCS-G-3	3	16	9	10	31
UCS-T-3	3	32	57	58	1
BFS-G-9	9	288	198	199	5
BFS-T-9	9	5821	11055	11056	38
DFS-G-9	44665	141452	32012	42967	1690
DFS-T-9	---	---	---	---	(1)
DLS-9-9	9	5474	0	0	36
DLS-3-9	0	12	0	0	0
IDS-9	9	9063	0	0	20
UCS-G-9	9	385	235	239	4
UCS-T-9	9	18070	31593	31594	72
BFS-G-30	30	181058	365	24048	1059
BFS-T-30	---	---	---	---	(2)
DFS-G-30	62856	80569	41533	41534	2332
DFS-T-30	---	---	---	---	(1)
DLS-9-30	0	4681	0	0	7
DLS-3-30	0	9	0	0	0
IDS-30	---	---	---	---	(1)
UCS-G-30	30	181390	49	24209	1040
UCS-T-30	---	---	---	---	(2)

Tabla comparativa entre Profundidad, Nodos Expandidos, Tamaño de la frontera, Máximo tamaño de la frontera y tiempo entre los distintos algoritmos evaluados.

(1) Este algoritmo no termina porque el algoritmo ha encontrado un bucle, esto puede porque se trataba de una búsqueda en árbol, ya que no guardan lista de nodos explorados.

(2) Este algoritmo no termina porque la máquina virtual de Java se queda sin memoria dinámica para representar los nodos.

Explicación de cada algoritmo

- **BFS-G:** Búsqueda en anchura en grafo, podemos apreciar que para una solución de 3 movimientos necesita expandir 5 nodos, y encuentra una solución a profundidad 3.
- **BFS-T:** Búsqueda en anchura en árbol, podemos apreciar que para una solución de 3 movimientos necesita expandir 6 nodos, y encuentra una solución a profundidad 3.
- **DFS-G:** Búsqueda en profundidad en grafo, podemos apreciar que para una solución de 3 movimientos necesita expandir 120491 nodos, y encuentra una solución a profundidad 59123, esto es debido a que busca por el primer hijo, en lugar de buscar por todos los hijos (por niveles) como la búsqueda en anchura.
- **DFS-T:** Búsqueda en profundidad en árbol, podemos apreciar que para ninguno de los casos (3 movimientos, 9 movimientos o 30 movimientos), consigue alcanzar una solución, esto es debido a que es un algoritmo de búsqueda en profundidad y en árbol, lo cual significa que, al no haber una lista de nodos explorados, el algoritmo ha podido entrar en un bucle.
- **DLS-9:** Búsqueda en profundidad limitada a 9 niveles, podemos apreciar que para la solución al tablero de 3 movimientos encuentra una solución a profundidad 9, habiendo expandido 10 nodos, sin embargo, para el tablero cuya solución está a 9 movimientos encuentra la solución a profundidad 9 expandiendo 10 nodos, pero para el tablero de 30 movimientos no encuentra solución, habiendo expandido 4681 nodos.
- **DLS-3:** Búsqueda en profundidad limitada a 3 niveles, podemos apreciar que para la solución al tablero de 3 movimientos encuentra una solución a profundidad 3, habiendo expandido 4 nodos, sin embargo, para el tablero cuya solución está a 9 movimientos encuentra no la solución habiendo expandiendo 12 nodos y, para el tablero de 30 movimientos no encuentra solución, habiendo expandido 9 nodos.
- **IDS:** Búsqueda en profundidad iterativa, para los tableros de 3 y 9 movimientos encuentra una solución a profundidad 3 y 9, expandiendo 9 y 9063 nodos, respectivamente, para el tablero de 30 movimientos, la búsqueda no terminaba debido a que se terminaba la memoria dinámica.
- **UCS-G:** Búsqueda de coste uniforme en grafo, para el tablero de 3 movimientos encuentra una solución a profundidad 3, habiendo expandido 16 nodos, para el de 9 movimientos, una solución a profundidad 9, expandiendo 385 nodos, y para el de 30, una solución a profundidad 30, expandiendo 181390 nodos.
- **UCS-T:** Búsqueda de coste uniforme en árbol, para el tablero de 3 movimientos, encuentra una solución a profundidad 3, expandiendo 32 nodos, para el tablero de 9 movimientos, encuentra una solución a profundidad 9, expandiendo 18070 nodos, pero para el tablero de 30 movimientos no encuentra solución ya que la búsqueda no acaba debido a que se termina la memoria dinámica.

Definición del problema del 15-puzzle

El problema del 15-puzzle se define de la misma manera que el del 8-puzzle, con la salvedad de que en este caso serían 15 fichas y 1 hueco en blanco, distribuidos en un tablero 4 x 4, que se desplazaría de la misma manera que en el 8-puzzle.

El estado inicial es análogo al del 8-puzzle, un conjunto de los 15 primeros números naturales (empezando por el 0), en una lista, de tal manera que cada número positivo es la casilla con dicho número, y el 0 es el hueco, que puede moverse.

Las funciones usadas en el 8-puzzle son similares a las utilizadas en el 15-puzzle, con la diferencia que debemos comparar con 16 valores en lugar que 8.

Se ha creado un paquete “aima.core.environment.fifteenpuzzle” que contiene las clases FifteenPuzzleBoard, FifteenPuzzleGoal y FifteenPuzzleFunctionFactory.