Universidad Zaragoza

INTELIGENCIA ARTIFICIAL (30223) – Curso 19-20

Grado en Ingeniería Informática

Área de Lenguajes y Sistemas Informáticos

Dpto. Informática e Ingeniería de Sistemas

Práctica 1 : Resolución de problemas y búsqueda. Búsqueda no informada

1. Objetivo de la práctica

Familiarizarse con el código en java para resolver problemas de búsqueda. El código lo puedes encontrar en https://github.com/aimacode. Descarga la versión 1.8 https://github.com/aimacode/aima-java/releases/tag/aima3e-v1.8.1.

Si descargas la última versión oficial https://github.com/aimacode/aima-java/releases/tag/aima3e-v1.9.1 (link *Download the latest official version = 1.9.1 (Dec 19 2016)*), debes descargar el fichero aima3e-java-1.9.1-Search-and-JavaFX.zip. Se precisa JDK 1.8. En esta última versión han reorganizado el código y puede que tengas que localizar el código en otros paquetes.

En esta práctica trataremos de aplicar diferentes técnicas de búsqueda para el problema de los Canibales y Misioneros, el 8-puzzle y el 15-puzzle. Se pretende (i) utilizar distintos algoritmos de búsqueda ciega, (ii) ser capaz de definir un problema y resolverlo con los algoritmos de búsqueda, y (iii) evaluar la aplicación de los algoritmos a los problemas y extraer algunas conclusiones.

8-puzzle

El 8-puzzle es un juego de mesa para un único jugador que consiste en 8 piezas cuadradas numeradas del 1 al 8 y un espacio vacío en un tablero de 3 x 3. El objetivo del juego es, partiendo de un estado representado por una disposición determinada de las piezas (estado inicial I) realizar movimientos permitidos para alcanzar una configuración deseada (meta O) en el menor número de movimientos posible.

La descripción de un estado especifica la localización de cada una de las 8 fichas y el espacio en blanco, en cada uno de los nueve cuadrados.

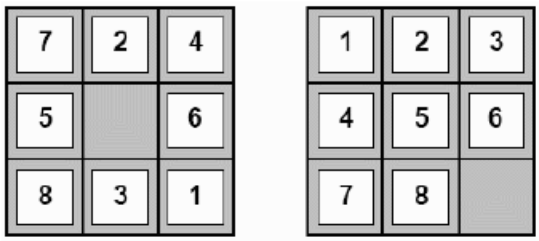

Figura 1: Ejemplos de estado inicial y estado final del 8-puzzle

Un **estado inicial** es cualquier configuración de las fichas de partida y un **estado meta**, una configuración determinada que hay que alcanzar (ver Figura 1). Para representar el problema podemos utilizar un array con los números {0,1,2,3,4,5,6,7,8} donde el cero simboliza la casilla vacía.

Misioneros y Caníbales

En el problema de los misioneros y los caníbales se plantea el siguiente problema: En la orilla de un río hay 3 misioneros y 3 caníbales y todos ellos pretenden cruzar al otro lado. La barca que se utiliza para cruzarlo solo tiene capacidad para dos personas, con lo que alguien ha de estar volviendo siempre a la orilla inicial mientras quede gente sin cruzar. Además, si en alguna ocasión y en cualquiera de las orillas se encuentran un número mayor de caníbales que de misioneros, los primeros se comerán a los segundos.

2. Tareas

1. En el paquete `aima.gui.demo.search` encontrarás el programa `EightPuzzleDemo`. Este programa muestra la utilización de distintos algoritmos para el problema del 8-puzzle. Verás que la definición del problema (estado, acciones, estado objetivo) se define en el paquete `aima.core.environment.eightpuzzle` mediante las clases: `EightPuzzleBoard`, `EightPuzzleFunctionFactory` y `EightPuzzleGoalTest`. Estudia estas clases y ejecuta el programa para comprobar el funcionamiento.
2. En las demos sólo se muestra la lista de acciones cuando se encuentra la solución. Mostrar los estados por los que se pasa al aplicar las soluciones. Define un método `executeActions`, al que pasamos como argumentos la lista de acciones solución y el problema. (Nota: El método `agent.getActions()` devuelve la lista de acciones para encontrar la solución una vez realizada la búsqueda).
3. Aplicar las nociones y métodos aprendidos en los apartados anteriores pare resolver el problema de los misioneros y los caníbales. Muestra las métricas y la traza de ejecución con distintos algoritmos.
   Como se muestra a continuación:

```
Misioneros y canibales BFS -->
pathCost : 11.0
nodesExpanded : 13
queueSize : 1
maxQueueSize : 3
Tiempo:16mls

SOLUCIÓN:
GOAL STATE
RIBERA-IZQ                  --RIO-- BOTE M M M C C C  RIBERA-DCH
CAMINO ENCONTRADO
      ESTADO INICIAL   RIBERA-IZQ M M M C C C  BOTE --RIO--                  RIBERA-DCH
   Action[name==M2C]   RIBERA-IZQ M M M     C       --RIO-- BOTE         C C  RIBERA-DCH
   Action[name==M1C]   RIBERA-IZQ M M M   C C  BOTE --RIO--                C  RIBERA-DCH
   Action[name==M2C]   RIBERA-IZQ M M M             --RIO-- BOTE       C C C  RIBERA-DCH
   Action[name==M1C]   RIBERA-IZQ M M M     C  BOTE --RIO--              C C  RIBERA-DCH
   Action[name==M2M]   RIBERA-IZQ     M     C       --RIO-- BOTE   M M   C C  RIBERA-DCH
 Action[name==M1M1C]   RIBERA-IZQ   M M   C C  BOTE --RIO--          M     C  RIBERA-DCH
   Action[name==M2M]   RIBERA-IZQ         C C       --RIO-- BOTE M M M     C  RIBERA-DCH
   Action[name==M1C]   RIBERA-IZQ       C C C  BOTE --RIO--      M M M       RIBERA-DCH
   Action[name==M2C]   RIBERA-IZQ           C       --RIO-- BOTE M M M   C C  RIBERA-DCH
   Action[name==M1C]   RIBERA-IZQ         C C  BOTE --RIO--      M M M     C  RIBERA-DCH
   Action[name==M2C]   RIBERA-IZQ                   --RIO-- BOTE M M M C C C  RIBERA-DCH
```

```
Misioneros y canibales DLS(11) -->
pathCost : 11.0
nodesExpanded : 2199
Tiempo:16mls

SOLUCIÓN:
GOAL STATE
RIBERA-IZQ                  --RIO-- BOTE M M M C C C  RIBERA-DCH
CAMINO ENCONTRADO
      ESTADO INICIAL   RIBERA-IZQ M M M C C C  BOTE --RIO--                  RIBERA-DCH
   Action[name==M2C]   RIBERA-IZQ M M M     C       --RIO-- BOTE         C C  RIBERA-DCH
   Action[name==M1C]   RIBERA-IZQ M M M   C C  BOTE --RIO--                C  RIBERA-DCH
   Action[name==M2C]   RIBERA-IZQ M M M             --RIO-- BOTE       C C C  RIBERA-DCH
   Action[name==M1C]   RIBERA-IZQ M M M     C  BOTE --RIO--              C C  RIBERA-DCH
   Action[name==M2M]   RIBERA-IZQ     M     C       --RIO-- BOTE   M M   C C  RIBERA-DCH
 Action[name==M1M1C]   RIBERA-IZQ   M M   C C  BOTE --RIO--          M     C  RIBERA-DCH
   Action[name==M2M]   RIBERA-IZQ         C C       --RIO-- BOTE M M M     C  RIBERA-DCH
   Action[name==M1C]   RIBERA-IZQ       C C C  BOTE --RIO--      M M M        RIBERA-DCH
   Action[name==M2C]   RIBERA-IZQ           C       --RIO-- BOTE M M M   C C  RIBERA-DCH
   Action[name==M1C]   RIBERA-IZQ         C C  BOTE --RIO--      M M M     C  RIBERA-DCH
   Action[name==M2C]   RIBERA-IZQ                  --RIO-- BOTE M M M C C C  RIBERA-DCH

Misioneros y canibales IDLS -->
pathCost : 11.0
nodesExpanded : 8504
Tiempo:21mls

SOLUCIÓN:
GOAL STATE
RIBERA-IZQ                  --RIO-- BOTE M M M C C C  RIBERA-DCH
CAMINO ENCONTRADO
      ESTADO INICIAL   RIBERA-IZQ M M M C C C  BOTE --RIO--                  RIBERA-DCH
   Action[name==M2C]   RIBERA-IZQ M M M     C       --RIO-- BOTE         C C  RIBERA-DCH
   Action[name==M1C]   RIBERA-IZQ M M M   C C  BOTE --RIO--                C  RIBERA-DCH
   Action[name==M2C]   RIBERA-IZQ M M M             --RIO-- BOTE       C C C  RIBERA-DCH
   Action[name==M1C]   RIBERA-IZQ M M M     C  BOTE --RIO--              C C  RIBERA-DCH
   Action[name==M2M]   RIBERA-IZQ     M     C       --RIO-- BOTE   M M   C C  RIBERA-DCH
 Action[name==M1M1C]   RIBERA-IZQ   M M   C C  BOTE --RIO--          M     C  RIBERA-DCH
   Action[name==M2M]   RIBERA-IZQ         C C       --RIO-- BOTE M M M     C  RIBERA-DCH
   Action[name==M1C]   RIBERA-IZQ       C C C  BOTE --RIO--      M M M        RIBERA-DCH
   Action[name==M2C]   RIBERA-IZQ           C       --RIO-- BOTE M M M   C C  RIBERA-DCH
   Action[name==M1C]   RIBERA-IZQ         C C  BOTE --RIO--      M M M     C  RIBERA-DCH
   Action[name==M2C]   RIBERA-IZQ                  --RIO-- BOTE M M M C C C  RIBERA-DCH
```

4. En las demos se muestran tres estados iniciales (*boardWithThreeMoveSolution, random1 y extreme)* cuya solución está a 3, 9 o 30 pasos. Prueba los algoritmos de búsqueda no informada vistas (anchura, profundidad en árbol y en grafo, búsqueda en profundidad recursiva con límite (DLS), la búsqueda iterativa en profundidad IDLS y coste uniforme) con estos estados iniciales y visualiza una tabla como la mostrada con las métricas obtenidas: Profundidad (coste de la solución), Expand. (Nodos expandidos), Q.size (tamaño de la frontera), MaxQS (Tamaño máximo frontera), tiempo de ejecución. Habrá algoritmos que no sean capaces de resolver en tiempo razonable o no tengan suficiente memoria. En esos casos pon un comentario (1), o (2) para hacer referencia a esos casos.

```
Problema|Profundidad|     Expand|     Q.Size|      MaxQS|     tiempo
 BFS-G-3|          3|          5|          4|          5|         17
 BFS-T-3|          3|          6|          9|         10|          1
 DFS-G-3|      59123|     120491|      39830|      42913|        891
 DFS-T-3|        ---|        ---|        ---|        ---|        (1)
 DLS-9-3|          9|         10|          0|          0|          1
 DLS-3-3|          3|          4|          0|          0|          1
   IDS-3|          3|          9|          0|          0|          1
 UCS-F-3|          3|         16|          9|         10|          2
 UCS-T-3|          3|         32|         57|         58|          1
 BFS-G-9|          9|        288|        198|        199|          1
 BFS-T-9|          9|       5821|      11055|      11056|         11
 DFS-G-9|      44665|     141452|      32012|      42967|        618
 DFS-T-9|        ---|        ---|        ---|        ---|        (1)
 DLS-9-9|          9|       5474|          0|          0|          7
 DLS-3-9|          0|         12|          0|          0|          0
   IDS-9|          9|       9063|          0|          0|          9
 UCS-G-9|          9|        385|        235|        239|          3
 UCS-T-9|          9|      18070|      31593|      31594|         29
BFS-G-30|         30|     181058|        365|      24048|        540
BFS-T-30|        ---|        ---|        ---|        ---|        (2)
DFS-G-30|      62856|      80569|      41533|      41534|        604
DFS-T-30|        ---|        ---|        ---|        ---|        (1)
...
```

5. Define el problema para el 15-puzzle y comprueba cómo funciona para tres casos distintos a diferentes profundidades. Experimenta como en el apartados anterior y muestra los resultados obtenidos.

NOTAS:
Para visualizas las métricas de la búsqueda puedes acceder a cada una de ellas de la siguiente forma:

```
String pathcostM =agent.getInstrumentation().getProperty("pathCost");
if (pathcostM!=null) depth = (int)Float.parseFloat(pathcostM);
else depth = 0;
if (agent.getInstrumentation().getProperty("nodesExpanded")==null) expandedNodes= 0;
else expandedNodes =
       (int)Float.parseFloat(agent.getInstrumentation().getProperty("nodesExpanded"));
if (agent.getInstrumentation().getProperty("queueSize")==null) queueSize=0;
else queueSize =   (int)Float.parseFloat(agent.getInstrumentation().getProperty("queueSize"));
if (agent.getInstrumentation().getProperty("maxQueueSize")==null) maxQueueSize= 0;
else maxQueueSize  =
       (int)Float.parseFloat(agent.getInstrumentation().getProperty("maxQueueSize"));
```

Copia `EightPuzzleDemo` como `EightPuzzlePract1`. Mira el código y crea un método eightPuzzleSearch al que pases como parámetro el tipo de búsqueda y el estado inicial y muestre por pantalla distintas métricas (nodos expandidos, profundidad de la solución, tiempo de ejecución, tamaño de la frontera y tamaño máximo de la frontera). Prueba todos los algoritmos informados explicados (excepto bidireccional) en clase mostrando una tabla de resultados y comentando los resultados, así como aquellos algoritmos con los que no has obtenido resultados por ser costosos en memoria o tiempo.

Crea el problema de las Misioneros y Caníbales en `aima.core.environment.Canibales` siguiendo el modelo del `aima.core.environment.eightpuzzle` sólo para búsquedas no informadas (no es necesario que definas ninguna heurística).

Te será útil el siguiente método `public static void executeActions(List<Action> actions, Problem problem)` para visualizar los estados por los que pasa desde el estado inicial hasta la solución encontrada y para comprobar que los operadores que has implementado funcionan correctamente.

```
	public static  void executeActions(List<Action> actions, Problem problem) {

		Object initialState = problem.getInitialState();
		ResultFunction resultFunction = problem.getResultFunction();

		Object state = initialState;
		System.out.println("INITIAL STATE");
		System.out.println(state);

		for (Action action : actions) {
			System.out.println(action.toString());
			state = resultFunction.result(state, action);
			System.out.println(state);
			System.out.println("- - -");
		}
	}
```