INTELIGENCIA ARTIFICIAL

PRÁCTICA 1

Rubén Rodríguez Esteban 737215

ÍNDICE DE CONTENIDOS

1. Introducción

Esta práctica ha consistido en la familiarización con código en java para resolver problemas de búsqueda. Para ello se han aplicado diferentes técnicas de búsqueda para el problema de los Caníbales y Misioneros, y el 8-puzzle. Para dichos problemas se han aplicado distintos algoritmos de búsqueda ciega o no informada, para lo que ha sido necesario la definición del problema, y resolverlo con los algoritmos de búsqueda. Una vez que se han aplicado los algoritmos a estos problemas se ha procedido a evaluar los resultados obtenidos y a extraer conclusiones.

2. Familiarización con el código Java

En primer lugar se ha procedido a analizar el código escrito en lenguaje java facilitado por el profesorado de la asignatura. En los fuentes proporcionados se encuentra un paquete denominado *aima.gui.demo.search* donde se encuentran definidos diversos problemas como el 8-puzzle y las N-reinas, aunque en esta práctica solo nos hemos centrado en el primero, es decir, el problema del 8-puzzle. La definición del problema (estado inicial, acciones y estado objetivo) han sido definidas en el paquete denominado *aima.core.environment.eightpuzzle* mediante la clase *EightPuzzleBoard*, la clase *EightPuzzleFunctionFactory* y la clase *EightPuzzleGoalTest.*

Una vez localizadas y estudiadas estas clases se ha procedido a ejecutar la clase EightPuzzleDemo, localizado en el paquete *aima.gui.demo.seacrh*, donde se han lanzado para ese problema un conjunto de algoritmos de búsqueda no informada, visualizándose los resultados por la terminal de Eclipse.

3. Pasos de los algoritmos

Al ejecutar los algoritmos se puede observar que solamente se muestra el estado final que se alcanza cuando finaliza el problema, pero no se conoce en ningún momento los estados intermedios que se han seguido para llegar a la solución. Para poder hacer esto se ha realizado dentro del paquete *aima.gui.demo.search* una clase generica conocida como claseGenerica.java. En dicha clase se ha definido un método conocido como *executeActions* que recibe por parámetros la lista de acciones generada y el problema que se ha querido resolver, dicho método puede ser ejecutado por cualquier problema que lo implemente, en este caso el problema del ocho-puzzle. De este modo, una vez finalizado el método se ha ejecutado el archivo EightPuzzleDemo para observar los resultados obtenidos.

4. Algoritmo de los misioneros y los caníbales

A continuación se ha procedido a resolver el problema de los misioneros y los caníbales. Para ello, se ha procedido a generar una clase denominada *Canibales.java*, la clase CanibalesFunctionsFactory.java y la clase CanibalesGoalTest.java donde se han definido el estado inicial, el estado final y las posibles acciones o movimientos. Dichos ficheros han sido creados dentro del paquete *aima.core.environment.Canibales*.

El problema de los caníbales y los misioneros se ha modelado por medio de una representación estática, es decir, un vector de cinco componentes donde se representa el número de misioneros a la izquierda del río, los caníbales a la izquierda del río, los misioneros a la derecha del río, los caníbales a la derecha del río y por último la ribera en la que se encuentra el bote.

Dado que el problema comienza con los tres caníbales y los tres misioneros a la izquierda junto con el boe, el estado inicial es {3,3,0,0,0} y el estado final es {0,0,3,3,1}. La posición del bote en el margen izquierda del río se representa con un cero, y en el margen derecha con un uno.

En el fichero Canibales.java se han especificado tanto la estructura interna del tipo de dato como la colección de métodos de la clase que ayudan a gestionar el algoritmo. El estado final se ha definido en el fichero CanibalesGoaltTest.java.

Las operaciones definidas en este problema permiten consultar la ribera en la que se encuentra el bote, mover el bote, consultar cuántos caníbales y misioneros hay en la orilla izquierda, cuántos caníbales y misioneros en la orilla derecha, mover un misionero, un caníbal, dos misioneros, dos caníbales , o un misionero y un caníbal.

Una vez efectuados estos pasos, se ha procedido a crear un fichero dentro de la clase *aima.gui.demo.search* denominado CanibalesDemo.java donde se ha probado el problema lazando varios algoritmos de búsqueda no informada. Concretamente se han lanzado los algoritmos BFS, IDLS-11, y DLS.

A continuación se muestra para cada algoritmo los resultados obtenidos en terminal para cada uno de los algoritmos anteriores respectivamente.

```
Misioneros y Canibales BFS -->
queueSize : 1
maxQueueSize : 3
pathCost : 11.0
nodesExpanded : 13
Tiempo:149mls

SOLUCION
GOAL STATE
RIBERA-IZQ                  --RIO-- BOTE M M M C C C RIBERA-DCHA
CAMINO ENCONTRADO
ESTADO INICIAL
                      RIBERA-IZQ M M M C C C BOTE --RIO--                  RIBERA-DCHA
   Action[name==M2C]  RIBERA-IZQ M M M     C      --RIO-- BOTE         C C RIBERA-DCHA
   Action[name==M1C]  RIBERA-IZQ M M M   C C BOTE --RIO--                C RIBERA-DCHA
   Action[name==M2C]  RIBERA-IZQ M M M           --RIO-- BOTE       C C C RIBERA-DCHA
   Action[name==M1C]  RIBERA-IZQ M M M     C BOTE --RIO--              C C RIBERA-DCHA
   Action[name==M2M]  RIBERA-IZQ     M     C      --RIO-- BOTE   M M   C C RIBERA-DCHA
 Action[name==M1M1C]  RIBERA-IZQ   M M   C C BOTE --RIO--          M     C RIBERA-DCHA
   Action[name==M2M]  RIBERA-IZQ         C C      --RIO-- BOTE M M M     C RIBERA-DCHA
   Action[name==M1C]  RIBERA-IZQ       C C C BOTE --RIO--      M M M       RIBERA-DCHA
   Action[name==M2C]  RIBERA-IZQ           C      --RIO-- BOTE M M M   C C RIBERA-DCHA
   Action[name==M1M]  RIBERA-IZQ     M     C BOTE --RIO--        M M   C C RIBERA-DCHA
 Action[name==M1M1C]  RIBERA-IZQ                  --RIO-- BOTE M M M C C C RIBERA-DCHA



Misioneros y Canibales DLS(11) -->
pathCost : 11.0
nodesExpanded : 2705
Tiempo:123mls

SOLUCION
GOAL STATE
RIBERA-IZQ                  --RIO-- BOTE M M M C C C RIBERA-DCHA
CAMINO ENCONTRADO
ESTADO INICIAL
                      RIBERA-IZQ M M M C C C BOTE --RIO--                  RIBERA-DCHA
   Action[name==M2C]  RIBERA-IZQ M M M     C      --RIO-- BOTE         C C RIBERA-DCHA
   Action[name==M1C]  RIBERA-IZQ M M M   C C BOTE --RIO--                C RIBERA-DCHA
   Action[name==M2C]  RIBERA-IZQ M M M           --RIO-- BOTE       C C C RIBERA-DCHA
   Action[name==M1C]  RIBERA-IZQ M M M     C BOTE --RIO--              C C RIBERA-DCHA
   Action[name==M2M]  RIBERA-IZQ     M     C      --RIO-- BOTE   M M   C C RIBERA-DCHA
 Action[name==M1M1C]  RIBERA-IZQ   M M   C C BOTE --RIO--          M     C RIBERA-DCHA
   Action[name==M2M]  RIBERA-IZQ         C C      --RIO-- BOTE M M M     C RIBERA-DCHA
   Action[name==M1C]  RIBERA-IZQ       C C C BOTE --RIO--      M M M       RIBERA-DCHA
   Action[name==M2C]  RIBERA-IZQ           C      --RIO-- BOTE M M M   C C RIBERA-DCHA
   Action[name==M1M]  RIBERA-IZQ     M     C BOTE --RIO--        M M   C C RIBERA-DCHA
 Action[name==M1M1C]  RIBERA-IZQ                  --RIO-- BOTE M M M C C C RIBERA-DCHA
```

```
Misioneros y Canibales IDLS -->
pathCost : 11.0
nodesExpanded : 9010
Tiempo:453mls

SOLUCION
GOAL STATE
RIBERA-IZQ                  --RIO-- BOTE M M M C C C RIBERA-DCHA
CAMINO ENCONTRADO
ESTADO INICIAL
                      RIBERA-IZQ M M M C C C BOTE --RIO--                  RIBERA-DCHA
   Action[name==M2C]  RIBERA-IZQ M M M     C      --RIO-- BOTE         C C RIBERA-DCHA
   Action[name==M1C]  RIBERA-IZQ M M M   C C BOTE --RIO--                C RIBERA-DCHA
   Action[name==M2C]  RIBERA-IZQ M M M           --RIO-- BOTE       C C C RIBERA-DCHA
   Action[name==M1C]  RIBERA-IZQ M M M     C BOTE --RIO--              C C RIBERA-DCHA
   Action[name==M2M]  RIBERA-IZQ     M     C      --RIO-- BOTE   M M   C C RIBERA-DCHA
 Action[name==M1M1C]  RIBERA-IZQ   M M   C C BOTE --RIO--          M     C RIBERA-DCHA
   Action[name==M2M]  RIBERA-IZQ         C C      --RIO-- BOTE M M M     C RIBERA-DCHA
   Action[name==M1C]  RIBERA-IZQ       C C C BOTE --RIO--      M M M       RIBERA-DCHA
   Action[name==M2C]  RIBERA-IZQ           C      --RIO-- BOTE M M M   C C RIBERA-DCHA
   Action[name==M1M]  RIBERA-IZQ     M     C BOTE --RIO--        M M   C C RIBERA-DCHA
 Action[name==M1M1C]  RIBERA-IZQ                  --RIO-- BOTE M M M C C C RIBERA-DCHA
```

Tal y como se puede en las salidas anteriores los resultados son muy aclarativos. Si se observa la devolución del algoritmo BFS se puede observar que concuerda con los resultados del guión. Se puede observar que la secuencia de estados por los que pasa el algoritmo es diferente a la del guión debido a que el orden en el que se aplican las operaciones es distinto. No obstante la expansión de los nodos es el misma por lo que el tamaño maximo de la frontera y los nodos que quedan en la frontera son los mismos.

Sin embargo esto no ocurre en el caso del DLS con límte de profundidad 11 y con el IDLS, se puede observar que al igual que el algoritmo BFS, el camino consta de 11 pasos pero la expansión de los nodos no es la misma. La razón de esta variación estriba en que los algoritmos de búsqueda en profundidad se expanden los nodos en órdenes distintos pero no tiene ninguna importancia dado que si se comparan los tiempos de ejecución las diferencias son prácticamente despreciables.

5. Evaluación de métricas y análisis del ocho-puzzle

Una vez finalizado el algoritmo de los misioneros y los caníbales se ha procedido a manipular el fichero *EightPuzzleDemo.java.* Dicho archivo ha sido copiado y renombrado con el nombre *EightPuzzlePract1.java.* En dicho fichero se ha creado un método llamado eightPuzzleSearch al que se le ha pasado como parámetro el tipo de búsqueda y el estado inicial. Este método muestra por pantalla distintas métricas (nodos expandidos, profundidad de la solución, tiempo de ejecución, tamaño de la frontera y tamaño máximo de la frontera) del algoritmo de búsqueda no informada que se le ha pasado como parámetro.

Posteriormente se han probado 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. A continuación se muestra una tabla con los resultados obtenidos.

Problema	Profundidad	Expand	Q.Size	MaxQS	tiempo
BFS-G-3	3	5	4	5	0
BFS-T-3	3	6	9	10	0
DFS-G-3	59123	120491	39830	42913	1266
DFS-T-3	---	---	---	---	(1)
DLS-9-3	3	4	0	0	0
DLS-3-3	9	10	0	0	0
IDS-3	3	9	0	0	0
UCS-F-3	3	16	9	10	0
UCS-T-3	3	32	57	58	0
BFS-G-9	9	288	198	199	16
BFS-T-9	9	5821	11055	11056	78
DFS-G-9	44665	141452	32012	42967	609
DFS-T-9	---	---	---	---	(1)
DLS-9-3	0	12	0	0	0
DLS-9-9	9	5474	0	0	0
IDS-9	9	9063	0	0	16
UCS-9-3	9	385	235	239	0
UCS-T-9	9	18070	31593	31594	47
BFS-G-30	30	181058	365	24048	579
BFS-G-30	---	---	---	---	(2)
DFS-G-30	62856	80569	41533	41534	423
DFS-T-30	---	---	---	---	(1)
DLS-30-3	0	9	0	0	0
DLS-30-9	0	4681	0	0	0
IDS-30	---	---	---	---	(2)
UCS-F-30	---	---	---	---	(2)
UCS-T-30	---	---	---	---	(1)

Con estos resultados se pueden observar algunos hechos que son interesantes. Por ejemplo el algoritmo DepthFisrtSearch en árbol no se soluciona por coste en memoria debido a que el tamaño de la frontera es bastante grande aunque el coste en espacio que supone este algoritmo es lineal. Ésto ocurre en el DFS para la solución que se encuentra tanto en tres, como en nueve como en treinta pasos.

También se puede observar que el algoritmo IDS con la solución en treinta pasos tampoco encuentra la solución. Este algoritmo es completo, es decir, si existe una solución la va a encontrar, sin embargo en éste caso no la encuentra porque es muy costoso ya que la complejidad en tiempo es exponencial frente a la de espacio que es lineal. El algoritmo de coste uniforme en 30 pasos no lo soluciona en ninguno de los casos debido al coste en tiempo y en memoria debido a que como en casos anteriores el tamaño de la frontera es muy grande. El resto de los algoritmos si que encuentran la solución aunque en tiempos muy variados como es el caso por ejemplo de los algoritmos DLS en 30 pasos que encuentran la solución en un tiempo inmediato, cero segundos, debido a que el coste de la solución en profundidad es cero o muy pequeño como es el caso del UCS en 9 pasos. El resto de los algoritmos ofrecen unas métricas de ejecución muy razonables.