INTELIGENCIA ARTIFICIAL

Práctica 1

2018/2019

Jesús Aguas Acín -- 736935@unizar.es

Introducción

El objetivo de esta práctica es resolver distintos problemas, con un estado inicial y una serie de paso necesarios para llegar a un estado objetivo, para ello se utilizan varios algoritmos de búsqueda, y se comparan sus resultados para analizar cuales son los más eficientes dependiendo el tipo de problema y su complejidad.

Primero se utilizará el juego del ocho-puzzle (ya implementado) con 3 niveles distintos de dificultad, esto son 3 distintos estados iniciales de modo que cada uno necesita un número distinto de pasos para llegar al estado objetivo.

Después se implementará el problema de los Misioneros y los Caníbales para después ser resuelto mediante búsquedas no informadas.

Ficheros entregados:

- *EightPuzzlePract1.java*: Contiene el main, que utilizará distintos algoritmos para resolver el juego del ocho-puzzle.
- *CanibalesPract1.java*: Contiene el main, que utilizará distintos algoritmos para resolver el problema de los Caníbales.
- *CanibalesBoard.java*: Implementa las funciones que utilizará el problema de los Caníbales.
- *CanibalesFunctionFactory.java*: Define el funcionamiento del problema de los Caníbales.
- *CanibalesGoalTest.java*: Define el estado objetivo del problema de los Caníbales.

EightPuzzle

El funcionamiento del EightPuzzle ya estaba implementado en *aima.core.environment.eightpuzzle*, por lo que solo hacía falta encontrar la solución con distintos algoritmos de búsqueda, para ello se ha creado la función *EightPuzzleSearch* con parámetros: el tipo de búsqueda, el estado inicial, un string para especificar el tipo de búsqueda, un booleano que si vale false no ejecutará la búsqueda, sino que mostrará el comentario especificado en el último parámetro de la función. Este comentario será (1) si el problema no se puede resolver en un tiempo razonable, y (2) si es por falta de memoria.

Se han recopilado variables como el tiempo total del algoritmo, la profundidad, los nodos expandidos.. todo para formatear la salida tal y como la muestra el guión de la práctica.
El resultado es el siguiente:

Problema	Profundidad	Expandi	Q.Size	MaxQS	tiempo
BFS-G-3	3	5	4	5	10
BFS-T-3	3	6	9	10	1
DFS-G-3	59123	120491	39830	42913	1274
DFS-T-3	---	---	---	---	(1)
DLS-9-3	9	10	0	0	1
DLS-3-3	3	4	0	0	0
IDS-3	3	9	0	0	0
UCS-G-3	3	16	9	10	4
UCS-T-3	3	32	57	58	0
BFS-G-9	9	288	198	199	3
BFS-T-9	9	5821	11055	11056	15
DFS-G-9	44865	141452	32012	42967	605
DFS-T-9	---	---	---	---	(1)
DLS-9-9	9	5474	0	0	61
DLS-3-9	0	12	0	0	0
IDS-9	9	9063	0	0	19
UCS-G-9	9	385	235	239	7
UCS-T-9	9	180701	31593	31594	106
BFS-G-30	30	181058	365	24048	664
BFS-T-30	---	---	---	---	(2)
DFS-G-30	62856	80569	41533	41534	594
DFS-T-30	---	---	---	---	(1)
DLS-9-30	0	4681	0	0	3
DLS-3-30	0	9	0	0	0
IDS-30	---	---	---	---	(1)
UCS-G-30	30	181390	49	24209	926
UCS-T-30	---	---	---	---	(2)

A partir de los resultados podemos obtener varias conclusiones, la primera es que el algoritmo de búsqueda DFS (DepthFirstSearch) es la menos eficiente en tiempo para resolver el problema en cualquiera de los estados iniciales, si hacemos la búsqueda en árbol, vemos que no es completa y que no puede alcanzar el estado objetivo en un tiempo razonable (1), esto es debido a que la búsqueda en árbol, a diferencia de la búsqueda en grado, puede dar lugar a bucles infinitos ya que no comprueba que haya estados repetidos.
También podemos observar que la búsqueda en anchura(BFS) se complica a medida que el estado objetivo se encuentra más profundo, debido a que el número de nodos frontera para cada nivel de profundidad es mayor, hasta que punto de que no se pueden guardar todos los estados en memoria (2). Lo mismo pasa con la búsqueda de coste uniforme, ya que al ser el coste de todos los nodos el mismo es los mismo que hacer una búsqueda en anchura.
El IDS tarda demasiado en tiempo ya que debe recorrer todos los nodos de cada nivel en profundidad hasta llegar al 30, que es donde se halla la solución, por ello (1)

Misioneros y Caníbales

Para implementar el problema de los Misioneros y Caníbales se han utilizado los ficheros *CanibalesBoard.java, CanibalesFunctionFactory.java* y *CanibalesGoalTest.java*.
Estos se obtienen del juego del EightPuzzle, para los dos últimos ficheros bastó con cambiar varios nombres para que funcionase para este problema, además del estado objetivo en *CanibalesGoalTest.java.*

Sin embargo para implementar el *CanibalesBoard.java* fue necesario cambiar totalmente las funciones para definir las acciones que se podían realizar para cambiar de estado y las precondiciones que restringían dichas acciones.

El estado del problema se ha definido con un vector de 5 enteros, siempre inicializado (Estado inicial) a {3,3,0,0,0} esto significa que hay 3 caníbales y 3 misioneros en la orilla izquierda, el bote se encuentra en la orilla izquierda(0) y hay 0 caníbales y 0 misioneros en la orilla derecha.

Se han diseñado varias funciones privadas para acceder al vector del estado del problema como getMLeft() o getBote(), que devuelve el número de misioneros en la orilla izquierda o la posición del bote, respectivamente. De este modo conseguir capas de abstracción para acceder a la estructura interna de la clase, por si en el futuro pudiera cambiar.

Además se ha reimplementado el método toString() para conseguir la salida por pantalla mostrada en el guión de la práctica.

Una vez definido el problema de los Caníbales se ha creado el fichero *CanibalesPract1.java* que contiene el *main*, desde el que se probarán distintos algoritmos de búsqueda para hallar la solución. En este caso lo probaremos con 3, el BFS(Búsqueda en anchura en Grafo), el DLS(Búsqueda en profundidad limitada a 11) y el IDLS (Búsqueda en profundidad iterativa).
Para ello se ha creado la función *CanibalesSearch* con parámetros: el algoritmo de búsqueda y un comentarios para especificar el tipo de búsqueda utilizada.
Además se extraen datos interesantes de la búsqueda, por ejemplo, el tiempo transcurrido, los nodos expandidos, la profundidad alcanzada (pathCost) que debería ser 11 si todo ha salido bien.

También se ha modificado la salida para que se muestre igual que se especifica en el guión:

```
Misioneros y canibales BFS -->
queueSize : 1
maxQueueSize : 3
pathCost : 11.0
nodesExpanded : 13
Tiempo:0mls

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==MOV2C] RIBERA-IZQ M M M     C       --RIO-- BOTE         C C  RIBERA-DCH
Action[name==MOV1C] RIBERA-IZQ M M M   C C  BOTE --RIO--                C  RIBERA-DCH
Action[name==MOV2C] RIBERA-IZQ M M M             --RIO-- BOTE       C C C  RIBERA-DCH
Action[name==MOV1C] RIBERA-IZQ M M M     C  BOTE --RIO--                C  RIBERA-DCH
Action[name==MOV2M] 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==MOV2M] RIBERA-IZQ         C C       --RIO-- BOTE M M M     C  RIBERA-DCH
Action[name==MOV1C] RIBERA-IZQ       C C C  BOTE --RIO--      M M M        RIBERA-DCH
Action[name==MOV2C] RIBERA-IZQ           C       --RIO-- BOTE M M M   C C  RIBERA-DCH
Action[name==MOV1C] RIBERA-IZQ         C C  BOTE --RIO--      M M M     C  RIBERA-DCH
Action[name==MOV2C] RIBERA-IZQ                   --RIO-- BOTE M M M C C C  RIBERA-DCH
```

Misioneros y Caníbales

```
Misioneros y canibales DLS (11) -->
pathCost : 11.0
nodesExpanded : 2180
Tiempo:43mls

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==MOV2C]     RIBERA-IZQ M M M     C        --RIO-- BOTE         C C  RIBERA-DCH
Action[name==MOV1C]     RIBERA-IZQ M M M   C C  BOTE --RIO--                 C  RIBERA-DCH
Action[name==MOV2C]     RIBERA IZQ M M M                --RIO-- BOTE       C C C  RIBERA DCH
Action[name==MOV1C]     RIBERA-IZQ M M M     C  BOTE --RIO--                 C C  RIBERA-DCH
Action[name==MOV2M]     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==MOV2M]     RIBERA-IZQ         C C        --RIO-- BOTE M M M     C  RIBERA-DCH
Action[name==MOV1C]     RIBERA-IZQ       C C C  BOTE --RIO--       M M M       RIBERA-DCH
Action[name==MOV2C]     RIBERA-IZQ           C        --RIO-- BOTE M M M   C C  RIBERA-DCH
Action[name==MOV1C]     RIBERA-IZQ         C C  BOTE --RIO--       M M M     C  RIBERA-DCH
Action[name==MOV2C]     RIBERA-IZQ                  --RIO-- BOTE M M M C C C  RIBERA-DCH
```

```
Misioneros y canibales IDLS -->
pathCost : 11.0
nodesExpanded : 8485
Tiempo:46mls

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==MOV2C]     RIBERA-IZQ M M M     C        --RIO-- BOTE         C C  RIBERA-DCH
Action[name==MOV1C]     RIBERA-IZQ M M M   C C  BOTE --RIO--                 C  RIBERA-DCH
Action[name==MOV2C]     RIBERA-IZQ M M M                --RIO-- BOTE       C C C  RIBERA-DCH
Action[name==MOV1C]     RIBERA-IZQ M M M     C  BOTE --RIO--                 C C  RIBERA-DCH
Action[name==MOV2M]     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==MOV2M]     RIBERA-IZQ         C C        --RIO-- BOTE M M M     C  RIBERA-DCH
Action[name==MOV1C]     RIBERA-IZQ       C C C  BOTE --RIO--       M M M       RIBERA-DCH
Action[name==MOV2C]     RIBERA-IZQ           C        --RIO-- BOTE M M M   C C  RIBERA-DCH
Action[name==MOV1C]     RIBERA-IZQ         C C  BOTE --RIO--       M M M     C  RIBERA-DCH
Action[name==MOV2C]     RIBERA-IZQ                  --RIO-- BOTE M M M C C C  RIBERA-DCH
```

Así se puede comprobar que el pathCost es 11 en todos los algoritmos de búsqueda, ese decir que la forma más rápida de llegar al estado objetivo es con 11 pasos y que la más eficiente en tiempo es la BFS (Búsqueda en Anchura en Grafo), esto es debido a que la solución se encuentra en un número reducido de pasos, si fuese una solución con un gran número de paso dejaría de ser viable una búsqueda en anchura y sería más lógico optar por una en profundidad.