IMPLEMENTACIÓN DEL PROBLEMA DE LAS JARRAS

David Mallasén Quintana e Iván Prada Cazalla
Doble grado en Ingeniería Informática y Matemáticas

TABLA DE CONTENIDO

1 INFORMACIÓN

Para cambiar los casos de prueba hay que cambiar las variables ***CASO*** y ***CASO_OBJETIVO*** de la clase ***GarrafasDemo.*** Resaltar que en **CASO** la garrafa de mayor capacidad debe ser la segunda, como se indica en los comentarios del código.

1 HEURÍSTICA

Para la heurística hemos utilizado una función que comprueba si estamos a distancia 0 o 1 de la solución, y en caso de esto se nos devuelven los respectivos valores. Si no, devuelve 2. Hemos elegido esta ya que queríamos una admisible y consistente.

1.1 PROPIEDADES DE LA HEURÍSTICA

La heurística es **admisible**. Siempre que estemos en un estado, tendremos tres posibilidades: Que sea ya la solución, en cuyo caso la estimación da lo mismo que el coste real. Que esté a distancia uno de la solución, entonces también nos está dando el coste del camino. Y por último, si no estamos en los dos casos anteriores, como mínimo faltan 2 pasos para llegar a la solución, luego 2 es inferior al coste real. No haría falta estudiar esto ya que, como veremos a continuación, la heurística es también consistente.

Para ver que la heurística es **consistente** baste darse cuenta de que el coste de pasar de un nodo a otro es siempre 1 y la heurística como mucho disminuye en 1 al aplicar un operador. De esta forma nunca tendremos que el valor de la heurística en un nodo sea mayor que el coste de llegar a otro nodo más la heurística en el otro nodo.

Hemos buscado una heurística consistente, y por tanto admisible, para obtener siempre la solución óptima al usar el algoritmo A*.

2 Tabla comparativa

2.1 Caso 3-5-4

Garrafa1: 3 Garrafa2: 5 Objetivo: 4	TreeSearch			GraphSearch				
	BFS	Coste Uniforme	A*	BFS	DFS	Coste Uniforme	Voraz	A*
pathCost	6	6	6	6	10	6	6	6
maxQueueSize	362	5	136	4	9	5	8	5
queueSize	362	1	135	2	8	1	7	1
nodesExpanded	189	12	73	12	10	12	10	11
exTime (micro s)	18150	1050	2919	738	953	555	572	323

2.2 Caso 3-7-1

Garrafa1: 3 Garrafa2: 7 Objetivo: 1	TreeSearch			GraphSearch				
	BFS	Coste Uniforme	A*	BFS	DFS	Coste Uniforme	Voraz	A*
pathCost	4	4	4	4	16	4	4	4
maxQueueSize	44	5	19	4	12	5	5	5
queueSize	44	1	18	2	11	1	4	2
nodesExpanded	23	8	10	8	16	8	6	7
exTime (micro s)	8503	611	555	809	1108	438	400	340

2.3 Caso 12-3-1

Garrafa1: 12 Garrafa2: 3 Objetivo: 1	TreeSearch			GraphSearch				
	BFS	Coste Uniforme	A*	BFS	DFS	Coste Uniforme	Voraz	A*
pathCost	-	0	-	0	0	0	0	0
maxQueueSize	-	5	-	4	8	5	6	5
queueSize	-	0	-	0	0	0	0	0
nodesExpanded	-	10	-	10	10	10	10	10
exTime (micro s)	-	1402	-	5978	1428	861	528	526

3 Análisis de los resultados obtenidos

Como estudio de los casos 3-5-4 y 3-7-1 observamos que los mejores resultados son los obtenidos en A* en GraphSearch, tanto en tiempo como en memoria. Vemos que el algoritmo voraz mejora ligeramente el uso de la memoria en los dos casos, pero produce peores resultados en el tiempo. Además, vemos que el algoritmo DFS no nos da la solución óptima ya

que, como sabemos, al no tratarse de un recorrido por niveles, la solución que produce no siempre será la óptima.

El caso 12-3-1 lo estudiamos aparte debido a que no existe solución. Esto lo podemos razonar viendo que una garrafa es múltiplo de la otra y el resultado no es múltiplo de la pequeña, luego nunca se podrá alcanzar la configuración buscada.

Para este caso el BFS con treeSearch no termina, pues no se tienen en cuenta repetidos y por lo tanto entra en bucle y sigue expandiendo ramas de forma indefinida. Lo mismo le pasa a A* con TreeSearch, que al no haber un mecanismo de eliminación de repetidos no se consigue vaciar nunca el conjunto de nodos abiertos.

En el resto de algoritmos nos da como resultado coste cero ya que nos quedamos sin abiertos en la cola y sin haber llegado a la solución, lo que produce que el algoritmo termine.