Práctica 1

Resolución de problemas y búsqueda.
Búsqueda no informada

Inteligencia artificial

3º curso, Grado de Ingeniería en Informática

28 de octubre de 2021

Leticia Sánchez Romero, 760549

Índice de contenidos

1. 8-puzzle: Análisis clases java

class EightPuzzleDemo

En primer lugar hay 3 tableros distintos definidos con tres estados iniciales (*boardWithThreeMoveSolution, random1* y *extreme*).

El método main realiza la ejecución de distintos problemas de búsqueda para el problema del 8-puzzle, no informadas e informadas. Los métodos a los que llama están definidos en la misma clase (*EightPuzzleDemo*).

Cada uno de los métodos definidos en la clase pertenecientes a búsquedas (no informadas o informadas) en su implementación hacen lo siguiente:

- ➔ Se define un problema (clase *aima.core.search.framework.Problem*) con:
  - **initialState** el estado inicial en el que comienza el agente.
  - **actionsFunction** una descripción de las posibles acciones disponibles para el agente.
  - **resultFunction** una descripción de lo que hace cada acción; el nombre formal para esto es el modelo de transición, especificado por una función RESULT (s, a) que devuelve el estado que resulta de hacer la acción a en el estado s.
  - **goalTest** la prueba que determina si un estado dado es un estado objetivo.
- ➔ Se define el tipo de búsqueda escogiendo un método de una de las siguientes clases pertenecientes a los paquetes:
  1. *aima.core.search.uninformed*: en caso de que sea un tipo de búsqueda no informada
  2. *aima.core.search.informed*: en caso de que sea un tipo de búsqueda informada

  Dentro de cada paquete se encuentran distintas clases con los nombres de los distintos tipos de búsquedas. Para realizar los ejercicios del 8-puzzle de esta práctica se necesitará hacer uso del paquete número 1.
- ➔ Se define el agente, es decir la función para solucionar el problema indicado siguiendo la estrategia indicada con el tipo de búsqueda.
- ➔ Se sacan por pantalla (stdout) las distintas acciones que se han realizado para resolver el problema con dicha estrategia (tipo de búsqueda).

class EightPuzzleBoard

Se definen el estado y las acciones que se pueden realizar sobre el tablero. El estado se define como un array de enteros de capacidad 8 en el que del tablero la casilla 1 (primera fila a la izquierda) es el elemento 0; la casilla 2 (primera fila en medio) es el elemento 1; …; la casilla 8 (tercera fila a la derecha) es el elemento 7. Además cada uno de los números queda representado por él mismo y el 0 representa al hueco vacío.

Se definen distintos métodos tanto para definir un cierto estado, como para realizar las distintas acciones sobre

class EightPuzzleFunctionFactory

Se definen métodos para definir las acciones que se van realizando y devolverlas, y para devolver el nuevo tablero tras haber realizado cierta acción.

class EightPuzzleGoalTest

Se define el tablero que tiene que quedar como resultado, es decir estado objetivo (estado final).
Se define el método que compara un estado con el estado objetivo.

2. 8-puzzle: Algoritmos no informados

Los resultados obtenidos para cada algoritmo no informado son los siguientes:

Problema	Profundidad	Expand	Q.Size	MaxQS	tiempo (mls)
BFS-G-3	3	5	4	5	8
BFS-T-3	3	6	9	10	1
DFS-G-3	59123	120491	39830	42913	1150
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-G-3	3	16	9	10	3
UCS-T-3	3	32	57	58	1
BFS-G-9	9	288	198	199	4
BFS-T-9	9	5821	11055	11056	13
DFS-G-9	44665	141452	32012	42967	782
DFS-T-9	---	---	---	---	(1)
DLS-9-9	9	5474	0	0	9
DLS-3-9	0	12	0	0	1
IDS-9	9	9063	0	0	13
UCS-G-9	9	385	235	239	4
UCS-T-9	9	18070	31593	31594	57
BFS-G-30	30	181058	365	24048	862
DFS-T-30	---	---	---	---	(2)
DFS-G-30	62856	80569	41533	41534	595
DFS-T-30	---	---	---	---	(1)
DLS-9-30	0	4681	0	0	4
DLS-3-30	0	9	0	0	0
IDS-30	---	---	---	---	(1)
UCS-G-30	30	181390	49	24209	741
UCS-T-30	---	---	---	---	(2)

(1) La ejecución del algoritmo ha sido abortada por la complejidad temporal.
(2) La ejecución del algoritmo ha sido abortada por la complejidad espacial.

El problema indica el tipo de búsqueda que se ha realizado y a cuántos pasos está la solución (3,9 o 30); la profundidad indica el coste de la solución, es decir el número de nodos que tiene el camino desde el nodo inicial hasta el objetivo; expand indica los nodos que se han expandido para llegar a la solución; Q.size indica el tamaño de la frontera cuando se ha encontrado la solución; MaxQs indica el tamaño máximo que ha llegado a tener la frontera en la búsqueda; y tiempo indica el tiempo de ejecución en milisegundos.

A continuación se va a hacer una breve explicación de cada uno de estos algoritmos, no se indicarán la explicación de todos los valores ya que se puede ver en la tabla y con la explicación del párrafo anterior.

Búsqueda Primero en anchura en grafo (BFS-G)
Se observa que es un buen algoritmo para encontrar la solución óptima ya que en los tres casos la profundidad es igual al número de pasos a los que está la solución.
En este como se ha visto en teoría se utiliza una cola FIFO y el test objetivo se aplica al generar el nodo.
Se observa que cuando la solución está a mayor número de pasos la complejidad espacial aumenta mucho ya que los nodos que se expanden, y los datos relacionados con la frontera son de gran magnitud.

Búsqueda Primero en anchura en árbol (BFS-T)
Se observa que cuando la solución está a 3 y a 9 pasos la frontera es mayor, ya que el algoritmo en árbol no tiene la información de los nodos expandidos, por lo que expande más nodos y la frontera es mayor. Por esta razón si en el caso de los 30 pasos en grafo ya era muy alta, ahora al no tener expandidos el árbol puede entrar en bucles infinitos y no terminar la búsqueda (al menos en un tiempo razonable).

Búsqueda Primero en profundidad en grafo (DFS-G)
Se observa que este algoritmo no garantiza la solución óptima ya que la profundidad en los tres casos es excesivamente elevada.

Búsqueda Primero en profundidad en árbol (DFS-T)
En este caso lo más probable es que por ser la búsqueda en árbol y además no garantizar la solución óptima, la búsqueda ha podido quedar en un bucle en el árbol por tener solo frontera y no tener expandidos. Además en el caso de solución a 30 pasos lo que ocurre es que la complejidad espacial es muy elevada.

Búsqueda en profundidad limitada a 9 niveles (DLS-9)
Se observa que para la solución a 3 pasos no se encuentra la solución óptima, ya que la profundidad es 9. Para la solución a 9 pasos sí encuentra la solución óptima ya que la profundidad es 9, pero lo hace expandiendo un gran número de nodos (5474). Para la solución de 30 pasos expande 4681 nodos y no encuentra solución.

Búsqueda en profundidad limitada a 3 niveles (DLS-3)
Con la búsqueda en árbol encuentra solución óptima para la solución a 3 pasos, pero no encuentra solución para los otros dos casos expandiendo una cantidad pequeña de nodos.

Búsqueda en profundidad iterativa (IDS)
Para las soluciones a 3 y 9 pasos encuentra la solución óptima ya que la profundidad es la misma que los pasos indicados. Para el caso de 30 pasos no es capaz de encontrar una solución en tiempo razonable por lo que se aborta la ejecución.

Búsqueda coste uniforme en grafo (UCS-G)
En los tres casos encuentra la solución óptima. Al igual que en la búsqueda primero en anchura se observa como los nodos expandidos crecen muy rápido y mucho en cuanto la solución está a más pasos.

Búsqueda coste uniforme en árbol (UCS-T)
Para las soluciones a 3 y 9 pasos encuentra la solución óptima y se observa que al ser búsqueda en árbol se expanden más nodos y la frontera es mayor. Para el caso de la solución a 30 pasos la complejidad temporal es muy elevada.

3. 8-puzzle: executeActions

Se ha modificado la clase *EightPuzzleDemo* de tal forma que se ha añadido el método indicado en el enunciado de la práctica y se ha comprobado el correcto funcionamiento de todas las demos del main de esta clase. Para poder hacerlo se ha comentado la llamada a *printActions(agent.getActions())* y se ha realizado la llamada a *executeActions(agent.getActions(), problem)*.

En la salida por pantalla (stdout) se puede ver como ahora indica qué acción se va a realizar y muestra el tablero en el estado que queda después de realizar dicha acción.

Se utiliza esta función en el problema de los misioneros y caníbales.

4. Misioneros y caníbales

Para realizar el problema de los misioneros y caníbales se han seguido las diapositivas de clase.

En primer lugar se ha definido el estado de la siguiente forma:
ESTADO = {Mi, Ci, B, Md, Cd}
B ∈ [0, 1] indica la posición de la barca, por lo que toma el valor 0 si está en el extremo izquierdo, o 1 si está en el extremo derecho.
Mi, Ci, Md, Cd ∈ [0,..., 3] indican el número de misioneros y caníbales que quedan en el extremo izquierdo y derecho del río, respectivamente.

De esta manera, el estado inicial se representa como (3, 3, i, 0, 0) y el estado final como (0, 0, f, 3, 3).

Las acciones son 5 ya que están parametrizadas siendo x la orilla en la que está e y la orilla a la que va (otra opción es poner 10 acciones y no parametrizarlas). Las acciones son las siguientes:

- Mover1C(x, y) transporta 1 caníbal desde la orilla x hasta la orilla y
- Mover2C(x, y) transporta 2 caníbales desde la orilla x hasta la orilla y
- Mover1M(x,y) transporta 1 misionero desde la orilla x hasta la orilla y
- Mover2M(x,y) transporta 2 misioneros desde la orilla x hasta la orilla y
- Mover1M1C(x,y) transporta 1 misionero y un caníbal desde x hasta y

El modelo de transiciones es el siguiente:

	Modelo de transiciones			
Acciones	Precondiciones			Resultado
Mover1C(x,y)	$C_x \geq 1$	$B = x$	$M_y \geq C_y + 1 \vee M_y = 0$	$B = y, C_x = C_x - 1, C_y = C_y + 1$
Mover2C(x,y)	$C_x \geq 2$	$B = x$	$M_y \geq C_y + 2 \vee M_y = 0$	$B = y, C_x = C_x - 2, C_y = C_y + 2$
Mover1M(x,y)	$M_x \geq 1$	$B = x$	$(M_x \geq C_x + 1 \vee M_x = 1) \wedge M_y \geq C_y - 1$	$B = y, M_x = M_x - 1, M_y = M_y + 1$
Mover2M(x,y)	$M_x \geq 2$	$B = x$	$(M_x \geq C_x + 2 \vee M_x = 2) \wedge M_y \geq C_y - 2$	$B = y, M_x = M_x - 2, M_y = M_y + 2$
Mover1M1C(x,y)	$M_x \geq 1, C_x \geq 1$	$B = x$	$M_y \geq C_y$	$B = y, M_x = M_x - 1, M_y = M_y + 1, C_x = C_x - 1, C_y = C_y + 1$

Respecto a la implementación, se ha seguido el modelo del 8-puzzle. Se ha definido la clase CanibalesBoard en la que se ha definido el estado como un array de enteros, y las 5 acciones, así

como los métodos que realizan las 5 acciones con sus respectivos parámetros, y el método que comprueba si las acciones se pueden realizar, siguiendo el modelo de transiciones.
En el código se aprecia la similitud con lo explicado anteriormente ya que se han nombrado las variables de igual o similar forma para que se entienda mejor.
También se ha definido el método *toString()* para poder realizar la tabla indicada en el enunciado con ayuda también del método *executeActions()* (perteneciente a la clase que ejecuta el main).

Además se ha definido la clase *CanibalesFunctionFactory*, la cual se ha tenido que modificar respecto al esquema del 8-puzzle por la parametrización de las acciones. Por otro lado se ha definido la clase *CanibalesGoalTest* para definir el estado objetivo y hacer la comprobación de un estado con el estado objetivo.

Por último se ha definido la clase *CanibalesPract1* para realizar las búsquedas no informadas vistas en clase.

El resultado de estas búsquedas se puede ver a continuación (comentarios tras los resultados):

```
Misioneros y canibales BFS en grafo -->
pathCost: 11.0
nodesExpanded: 13
queueSize: 1
maxQueueSize: 3
Tiempo: 11 mls
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 BFS en árbol -->
pathCost: 11.0
nodesExpanded: 4811
queueSize: 6177
maxQueueSize: 6178
Tiempo: 62 mls
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 DFS en grafo -->
pathCost: 11.0
nodesExpanded: 11
queueSize: 3
maxQueueSize: 4
Tiempo: 1 mls
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==M1M1C]   RIBERA-IZQ M M   C C        --RIO-- BOTE     M     C  RIBERA-DCH
  Action[name==M1M]   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==M1M]   RIBERA-IZQ M     C     BOTE --RIO--        M M   C C  RIBERA-DCH
Action[name==M1M1C]   RIBERA-IZQ                  --RIO-- BOTE M M M C C C  RIBERA-DCH


Misioneros y canibales DFS en árbol -->
Ejecución no realizada -> complejidad temporal


Misioneros y canibales DLS(11)-->
pathCost: 11.0
nodesExpanded: 2199
queueSize: 0
maxQueueSize: 0
Tiempo: 30 mls
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
queueSize: 0
maxQueueSize: 0
Tiempo: 41 mls
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 UCS en grafo -->
pathCost: 11.0
nodesExpanded: 14
queueSize: 0
maxQueueSize: 3
Tiempo: 1 mls
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==M1M1C]   RIBERA-IZQ M M   C C        --RIO-- BOTE     M     C  RIBERA-DCH
  Action[name==M1M]   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 UCS en arbol -->
```

```
pathCost: 11.0
nodesExpanded: 8630
queueSize: 12224
maxQueueSize: 12225
Tiempo: 59 mls
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==M1M]   RIBERA-IZQ M     C     BOTE --RIO--         M M   C C  RIBERA-DCH
Action[name==M1M1C]   RIBERA-IZQ                  --RIO-- BOTE M M M C C C  RIBERA-DCH
```

Se observa que en **BFS en grafo** la solución es óptima y los nodos expandidos solo son 2 más que el total de nodos del camino de la solución, y se refleja también el el tamaño de la frontera.
En **BFS en árbol** la solución es también óptima, pero en este caso la cantidad de nodos expandidos es muy elevada debido a que en grafo hay expandidos y frontera, y en árbol solo hay frontera.

En **DFS en grafo** se encuentra la solución óptima y el número de nodos expandidos es el mismo que el número de nodos que contiene el camino. Por el contrario en **DFS en árbol** se ha abortado la ejecución ya que no terminaba en un tiempo razonable, esto puede ser porque haya quedado en bucle en la búsqueda en árbol por solo tener frontera y no expandidos.

En **DLS con límite a 11 niveles** se encuentra la solución óptima expandiendo 2199 nodos.

En **IDLS** se encuentra la solución óptima expandiendo 8504 nodos.

En **UCS** se encuentra la solución óptima tanto **en grafo** como **en árbol**, observando lo mismo que se ha comentado anteriormente, y esto es que en árbol los nodos expandidos y la frontera tiene un mayor número de nodos.