# Memoria Práctica 1 - Inteligencia Artificial

Fernando Peña Bes (756012)

3 de noviembre de 2019, Universidad de Zaragoza

## 1. Introducción

El objetivo de esta práctica es aplicar diferentes técnicas de búsqueda al problema de los Misioneros y Caníbales, el 8-puzzle y el 15-puzzle. Se van a utilizar tanto algoritmos de búsqueda no informada como informada y se evaluará su aplicación a los problemas planteados.

Para realizar la práctica se ha usado el entorno de desarrollo Eclipse para Java y el código perteneciente al libro *Artificial Intelligence: A Modern Approach* [1], en la versión 1.8 (se puede descargar en `https://github.com/aimacode/aima-java/releases/tag/aima3e-v1.8.1`).

## 2. Misioneros y Caníbales

En base a al paquete `aima.core.enviroment.eightpuzzle`, se ha creado otro similar llamado `aima.core.enviroment.canibals` que contiene el planteamiento del problema, con las clases `CanibalsBoard.java` (tablero del juego), `CanibalsFunctionFactory.java` (para generar las acciones posibles desde un estado) y `CanibalsGoalTest.java` (el test objetivo del juego). Para poder ejecutarlo, se ha creado una clase llamada `CanibalsDemo.java` dentro del paquete `aima.gui.demo.search`.

Sabemos que la solución óptima del juego está a 11 pasos del estado inicial. Se ha probado a buscar en el espacio de estados del juego con los algoritmos BFS en grafo y árbol, DFS en grafo y árbol, UCS en grafo y árbol, DLS con profundidad 11 e IDS. DLS e IDS están implementadas de forma recursiva.

A continuación se muestran los resultados obtenidos:

```
Misioneros y canibales BFS - Graph  -->
pathCost : 11.0
nodesExpanded : 13
nodesGenerated : 28
queueSize : 1
maxQueueSize : 3
9 ms
                                M C   B   M C
                                -------------
            ESTADO INICIAL      3 3 | I | 0 0
      Action[name==Move 2C]     3 1 | D | 0 2
      Action[name==Move 1C]     3 2 | I | 0 1
      Action[name==Move 2C]     3 0 | D | 0 3
      Action[name==Move 1C]     3 1 | I | 0 2
      Action[name==Move 2M]     1 1 | D | 2 2
   Action[name==Move 1M 1C]     2 2 | I | 1 1
      Action[name==Move 2M]     0 2 | D | 3 1
      Action[name==Move 1C]     0 3 | I | 3 0
      Action[name==Move 2C]     0 1 | D | 3 2
      Action[name==Move 1C]     0 2 | I | 3 1
      Action[name==Move 2C]     0 0 | D | 3 3
```

```
Misioneros y canibales BFS - Tree  -->
pathCost : 11.0
nodesExpanded : 4811
nodesGenerated : 10989
queueSize : 6177
maxQueueSize : 6178
34 ms
                                M C   B   M C
                                -------------
            ESTADO INICIAL      3 3 | I | 0 0
      Action[name==Move 2C]     3 1 | D | 0 2
      Action[name==Move 1C]     3 2 | I | 0 1
      Action[name==Move 2C]     3 0 | D | 0 3
      Action[name==Move 1C]     3 1 | I | 0 2
      Action[name==Move 2M]     1 1 | D | 2 2
   Action[name==Move 1M 1C]     2 2 | I | 1 1
      Action[name==Move 2M]     0 2 | D | 3 1
      Action[name==Move 1C]     0 3 | I | 3 0
      Action[name==Move 2C]     0 1 | D | 3 2
      Action[name==Move 1C]     0 2 | I | 3 1
      Action[name==Move 2C]     0 0 | D | 3 3
```

```
Misioneros y canibales DFS - Graph  -->
pathCost : 11.0
nodesExpanded : 11
nodesGenerated : 25
queueSize : 3
maxQueueSize : 4
1 ms
                                M C   B   M C
                                -------------
            ESTADO INICIAL      3 3 | I | 0 0
   Action[name==Move 1M 1C]     2 2 | D | 1 1
      Action[name==Move 1M]     3 2 | I | 0 1
      Action[name==Move 2C]     3 0 | D | 0 3
      Action[name==Move 1C]     3 1 | I | 0 2
      Action[name==Move 2M]     1 1 | D | 2 2
   Action[name==Move 1M 1C]     2 2 | I | 1 1
      Action[name==Move 2M]     0 2 | D | 3 1
      Action[name==Move 1C]     0 3 | I | 3 0
      Action[name==Move 2C]     0 1 | D | 3 2
      Action[name==Move 1M]     1 1 | I | 2 2
Action[name==Move 1M 1C]      0 0 | D | 3 3
```

```
Misioneros y canibales DFS - Tree  -->
pathCost : ---
nodesExpanded : ---
nodesGenerated : ---
queueSize : ---
maxQueueSize : ---
--- ms
```

```
Misioneros y canibales UCS - Graph  -->
pathCost : 11.0
nodesExpanded : 14
nodesGenerated : 30
queueSize : 0
maxQueueSize : 3
0 ms
                                M C   B   M C
                                -------------
            ESTADO INICIAL      3 3 | I | 0 0
   Action[name==Move 1M 1C]     2 2 | D | 1 1
      Action[name==Move 1M]     3 2 | I | 0 1
      Action[name==Move 2C]     3 0 | D | 0 3
      Action[name==Move 1C]     3 1 | I | 0 2
      Action[name==Move 2M]     1 1 | D | 2 2
   Action[name==Move 1M 1C]     2 2 | I | 1 1
      Action[name==Move 2M]     0 2 | D | 3 1
      Action[name==Move 1C]     0 3 | I | 3 0
      Action[name==Move 2C]     0 1 | D | 3 2
      Action[name==Move 1C]     0 2 | I | 3 1
      Action[name==Move 2C]     0 0 | D | 3 3
```

```
Misioneros y canibales UCS - Tree  -->
pathCost : 11.0
nodesExpanded : 8630
nodesGenerated : 20854
queueSize : 12224
maxQueueSize : 12225
37 ms
                                M C   B   M C
                                -------------
            ESTADO INICIAL      3 3 | I | 0 0
      Action[name==Move 2C]     3 1 | D | 0 2
      Action[name==Move 1C]     3 2 | I | 0 1
      Action[name==Move 2C]     3 0 | D | 0 3
      Action[name==Move 1C]     3 1 | I | 0 2
      Action[name==Move 2M]     1 1 | D | 2 2
   Action[name==Move 1M 1C]     2 2 | I | 1 1
      Action[name==Move 2M]     0 2 | D | 3 1
      Action[name==Move 1C]     0 3 | I | 3 0
      Action[name==Move 2C]     0 1 | D | 3 2
      Action[name==Move 1M]     1 1 | I | 2 2
Action[name==Move 1M 1C]      0 0 | D | 3 3
```

```
Misioneros y canibales DLS(11) -->
pathCost : 11.0
nodesExpanded : 2199
nodesGenerated : 5469
23 ms
                                M C   B   M C
                                -------------
            ESTADO INICIAL      3 3 | I | 0 0
      Action[name==Move 2C]     3 1 | D | 0 2
      Action[name==Move 1C]     3 2 | I | 0 1
      Action[name==Move 2C]     3 0 | D | 0 3
      Action[name==Move 1C]     3 1 | I | 0 2
      Action[name==Move 2M]     1 1 | D | 2 2
   Action[name==Move 1M 1C]     2 2 | I | 1 1
      Action[name==Move 2M]     0 2 | D | 3 1
      Action[name==Move 1C]     0 3 | I | 3 0
      Action[name==Move 2C]     0 1 | D | 3 2
      Action[name==Move 1C]     0 2 | I | 3 1
      Action[name==Move 2C]     0 0 | D | 3 3
```

```
Misioneros y canibales IDS -->
pathCost : 11.0
nodesExpanded : 8504
12 ms
                                M C   B   M C
                                -------------
            ESTADO INICIAL      3 3 | I | 0 0
      Action[name==Move 2C]     3 1 | D | 0 2
      Action[name==Move 1C]     3 2 | I | 0 1
      Action[name==Move 2C]     3 0 | D | 0 3
      Action[name==Move 1C]     3 1 | I | 0 2
      Action[name==Move 2M]     1 1 | D | 2 2
   Action[name==Move 1M 1C]     2 2 | I | 1 1
      Action[name==Move 2M]     0 2 | D | 3 1
      Action[name==Move 1C]     0 3 | I | 3 0
      Action[name==Move 2C]     0 1 | D | 3 2
      Action[name==Move 1C]     0 2 | I | 3 1
      Action[name==Move 2C]     0 0 | D | 3 3
```

En este problema, aunque el factor de ramificación es muy pequeño (casi siempre es 1 y como máximo es 3), todas las decisiones que se toman son reversibles. Por tanto, en las búsquedas en árbol y en las recursivas se generan muchos nodos, ya que continuamente se están haciendo deshaciendo acciones. La búsqueda en profundidad en árbol, de hecho, genera tantos nodos que no acaba (seguramente por en una rama infinita donde se deshace y rehace en bucle la misma acción).

Si nos fijamos en la búsquedas en grafo, donde no se repiten estados en el árbol de exploración, vemos que la búsqueda BFS y UCS se comportan de manera muy parecida, ya que el coste de cada paso es constante. En la UCS se expanden un par más de nodos porque el test objetivo se aplica al expandir los nodos, mientras que en la BFS se aplica al generarlos.

Vemos que el algoritmo que antes encuentra la solución es el DFS en grafo, sin embargo que haya encontrado la solución óptima ha sido posible debido a la estructura del problema y a su simplicidad, ya que el algoritmo no es óptimo, podría haberse metido por un camino que no fuera el óptimo.

# 3. 8-puzzle

Entre los archivos del código de AIMA, hay una implementación de 8-puzzle. En este apartado se ha probado a resolver el problema aplicando diferentes algoritmos de búsqueda no informada e informada. Se ha adaptado el código para generar las métricas `Profundidad` (coste de la solución), `Expand` (Nodos expandidos), `Q.size` (tamaño de la frontera), `MaxQS` (tamaño máximo de la frontera) y `tiempo de ejecución`.

Los algoritmos utilizados han sido:

- Búsquedas no informadas
  - Búsqueda primero en anchura (BFS)
  - Búsqueda primero en profundidad (DFS)
  - Búsqueda en profundidad limitada
  - Búsqueda en profundización iterativa
  - Búsqueda en coste uniforme
- Búsquedas informadas (usando las heurísticas *Misplaced Tiles* y *Manhattan*)
  - Búsqueda primero el mejor voraz
  - Búsqueda A*

Todas las búsquedas se han realizado tanto en grafo como en árbol, excepto DLS e IDS, ya que están implementadas de forma recursiva. Además cada búsqueda se ha realizado tres veces. Primero con un tablero con la solución a tres movimientos, después con la solución a 9 movimientos y por último con la solución a 30 movimientos.

A continuación se muestran los resultados obtenidos. Hay algunas búsquedas que no terminan en un tiempo razonable o que guardan tantos nodos que no se pueden manejar en memoria. Las primeros están marcados con un `(1)` y las segundas con un `(2)`. Todas las pruebas se dejaron en ejecución un mínimo de 2 minutos.

Resultados 8-puzzle

| Problema | Profundidad | Expand | Q.Size | MaxQS | tiempo |
|---|---|---|---|---|---|
| BFS-G-3 | 3 | 5 | 4 | 5 | 7 ms |
| BFS-T-3 | 3 | 6 | 9 | 10 | 0 ms |
| DFS-G-3 | 59123 | 120491 | 39830 | 42913 | 537 ms |
| DFS-T-3 | --- | --- | --- | --- | (1) |
| DLS-9-3 | 9 | 10 | 0 | 0 | 0 ms |
| DLS-3-3 | 3 | 4 | 0 | 0 | 0 ms |
| IDS-3 | 3 | 9 | 0 | 0 | 0 ms |
| UCS-G-3 | 3 | 16 | 9 | 10 | 1 ms |
| UCS-T-3 | 3 | 32 | 57 | 58 | 2 ms |
| GBFS-MisTileH-G-3 | 3 | 3 | 3 | 4 | 1 ms |
| GBFS-ManhH-G-3 | 3 | 3 | 3 | 4 | 0 ms |
| GBFS-MisTileH-T-3 | 3 | 3 | 5 | 6 | 0 ms |
| GBFS-ManhH-T-3 | 3 | 3 | 5 | 6 | 0 ms |
| A*-MisTileH-G-3 | 3 | 3 | 3 | 4 | 0 ms |
| A*-ManhH-G-3 | 3 | 3 | 3 | 4 | 0 ms |
| A*-MisTileH-T-3 | 3 | 3 | 5 | 6 | 0 ms |
| A*-ManhH-T-3 | 3 | 3 | 5 | 6 | 0 ms |
| BFS-G-9 | 9 | 288 | 198 | 199 | 9 ms |
| BFS-T-9 | 9 | 5821 | 11055 | 11056 | 82 ms |
| DFS-G-9 | 44665 | 141452 | 32012 | 42967 | 428 ms |
| DFS-T-9 | --- | --- | --- | --- | (1) |
| DLS-9-9 | 9 | 5474 | 0 | 0 | 6 ms |
| DLS-3-9 | 0 | 12 | 0 | 0 | 0 ms |
| IDS-9 | 9 | 9063 | 0 | 0 | 7 ms |
| UCS-G-9 | 9 | 385 | 235 | 239 | 1 ms |
| UCS-T-9 | 9 | 18070 | 31593 | 31594 | 25 ms |
| GBFS-MisTileH-G-9 | 85 | 813 | 520 | 521 | 11 ms |
| GBFS-ManhH-G-9 | 9 | 11 | 11 | 12 | 0 ms |
| GBFS-MisTileH-T-9 | --- | --- | --- | --- | (1) |
| GBFS-ManhH-T-9 | 9 | 11 | 21 | 22 | 0 ms |
| A*-MisTileH-G-9 | 9 | 25 | 20 | 21 | 0 ms |
| A*-ManhH-G-9 | 9 | 11 | 11 | 12 | 0 ms |
| A*-MisTileH-T-9 | 9 | 33 | 56 | 57 | 0 ms |
| A*-ManhH-T-9 | 9 | 11 | 21 | 22 | 0 ms |
| BFS-G-30 | 30 | 181058 | 365 | 24048 | 502 ms |
| BFS-T-30 | --- | --- | --- | --- | (2) |
| DFS-G-30 | 62856 | 80569 | 41533 | 41534 | 362 ms |
| DFS-T-30 | --- | --- | --- | --- | (1) |
| DLS-9-30 | 0 | 4681 | 0 | 0 | 3 ms |
| DLS-3-30 | 0 | 9 | 0 | 0 | 0 ms |
| DLS-30-30 | --- | --- | --- | --- | (1) |
| IDS-30 | --- | --- | --- | --- | (1) |
| UCS-G-30 | 30 | 181390 | 49 | 24209 | 442 ms |
| UCS-T-30 | --- | --- | --- | --- | (2) |
| GBFS-MisTileH-G-30 | 116 | 803 | 525 | 526 | 4 ms |
| GBFS-ManhH-G-30 | 66 | 211 | 142 | 143 | 1 ms |
| GBFS_MisTileH-T-30 | --- | --- | --- | --- | (1) |
| GBFS_Manh-T-30 | --- | --- | --- | --- | (1) |
| A*-MisTileH-G-30 | 30 | 95920 | 23489 | 23530 | 550 ms |
| A*-ManhH-G-30 | 30 | 10439 | 5101 | 5102 | 45 ms |
| A*-MisTileH-T-30 | --- | --- | --- | --- | (1) |
| A*-MisTileH-T-30 | --- | --- | --- | --- | (1) |

Se va a realizar un análisis de los algoritmos, hay que tener en cuenta que el espacio de estados del problema es muy grande (9!/2), que en todos los estados el factor de ramificación vale en entre 2 y 4 y que todos pasos que dan se pueden deshacer.

## 3.1. Búsqueda primero en anchura (BFS)

Se puede ver como está búsqueda siempre devuelve la solución óptima, ya que la función de coste es constante. Sin embargo, conforme aumenta la profundidad de la solución, las fronteras se van haciendo muy grandes rápidamente. En el caso de la solución a 30 pasos, al buscar en grafo, la frontera llega a tener 24048 nodos y necesita procesar 181058 en total para encontrar la solución. Hay que recordar que tanto la complejidad espacial como temporal de este algoritmo es exponencial, $O(b^d)$.

Al buscar en árbol, aparecen muchos nodos redundantes y en el caso de la solución a 30, se empiezan a generar tantos nodos que la búsqueda no termina.

## 3.2. Búsqueda primero en profundidad (DFS)

La búsqueda en profundidad, como se puede, ver es especialmente mala para este problema, ya las soluciones están en la superficie, las soluciones profundas pasan continuamente por los mismos estados. A esto hay que sumarle que las acciones son reversibles y es fácil entrar en una rama infinita. Como se puede ver, ninguna búsqueda realizada con este algoritmo en árbol termina.

En el caso de las búsquedas en grafo si que encuentra soluciones pero a unas profundidades enormes. Para la prueba con solución a tres pasos encuentra una solución a distancia 59123, muy lejos de la óptima.

## 3.3. Búsqueda en profundidad limitada

Limitando la profundidad máxima de la búsqueda en profundidad, se pueden obtener resultados para la solución a distancia 3 y 9, aunque hace falta tener conocimiento del problema para poner el límite. Este algoritmo no devuelve la solución óptima, se puede ver en el ejemplo del tablero con solución a tres pasos limitando la profundidad a 9, encuentra una solución a 9 pasos. Además si el límite de profundidad es menor que la profundidad de la solución óptima, no se encuentra solución.

Como el algoritmo está implementado de forma recursiva, se asemeja a una búsqueda en árbol, pero más resistente en cuanto a memoria (a no ser que se acumulen muchas llamadas recursivas). Al poner límites la búsqueda es capaz de terminar en algunos casos, al contrario que en la búsqueda DFS en árbol. Sin embargo conforme la solución es más profunda, aumenta exponencialmente el tiempo de procesado y en el caso de la solución a 30, poniendo 30 como límite de profundidad, recorre tantos nodos que no es capaz de acabar.

## 3.4. Búsqueda en profundización iterativa

Esta búsqueda se apoya en la DLS, aunque siempre es óptima debido a que va buscando la solución con DLS poniendo un límite de profundidad cada vez mayor.

Es interesante compararla con la DLS, ya que se puede ver el *overhead* correspondiente a las llamadas a DLS anteriores a la llamada en la que se encuentra la solución. Por ejemplo, en el ejemplo de la solución a 9 pasos, el DLS con límite 9 recorre 5474, mientras que el IDS tiene que recorrer 9063 para encontrar la solución.

Se puede ver también como la DLS condiciona a la IDS en el ejemplo a profundidad 30. Como DLS no puede terminar con límite de profundidad 30, IDS no puede encontrar la solución.

## 3.5. Búsqueda en coste uniforme

Es óptima. Sin embargo tiene que recorrer muchos nodos y al realizarla en árbol se generan tantos repetidos que no puede terminar cuando la solución está a distancia 30.

Tiene que expandir más nodos que la búsqueda primero en anchura, ya que se asegura que se encuentre la solución óptima independientemente de la función de coste del problema realizando el test objetivo al expandir. Sin embargo, como la función de coste siempre es 1, no tiene sentido usar este algoritmo frente al BFS, que encontrará la solución óptima antes expandiendo menos nodos (realiza el test objetivo al generar los nodos).

### 3.6. Búsqueda primero el mejor voraz

Se ve como este algoritmo encuentra una solución muy rápido, aunque no sea la óptima, ya que va siguiendo el camino que marca la heurística.

También se ve cómo la heurística Manhattan es más informada que la Misplaced Tile, siempre consigue encontrar la solución expandiendo menos nodos.

Al buscar en árbol, igual que los algoritmos anteriores genera muchos nodos repetidos, y cuando la solución está a 30 pasos, no acaba.

### 3.7. Búsqueda A*

Este algoritmo se comporta de manera similar al BGFS, sin embargo en la estrategia al elegir que nodo expandir, incluye el coste real de los caminos a los nodos. Como las dos heurísticas usadas son admisibles y consistentes, siempre encuentra la solución óptima tanto en grafo como en árbol, aunque tenga que recorrer más nodos. En la búsqueda en árbol con la solución a 30 pasos igual que todos los algoritmos anteriores, no acaba.

## 4. 15-puzzle

Para terminar la práctica, se implementó el problema del 15-puzzle de forma similar a como estaba programado el 8-puzzle. Se modificaron también las dos heurísticas para poder realizar búsquedas informadas y se buscaron tableros con solución a 3, 9 y 30 pasos.

Las pruebas fueron similares a las realizadas en el problema anterior. A continuación, se muestran los resultados obtenidos:

Resultados 15-puzzle

| Problema | Profundidad | Expand | Q.Size | MaxQS | tiempo |
|---|---|---|---|---|---|
| BFS-G-3 | 3 | 9 | 10 | 11 | 5 ms |
| BFS-T-3 | 3 | 11 | 21 | 22 | 1 ms |
| DFS-G-3 | --- | --- | --- | --- | (2) |
| DFS-T-3 | --- | --- | --- | --- | (1) |
| DLS-9-3 | 9 | 26 | 0 | 0 | 1 ms |
| DLS-3-3 | 3 | 11 | 0 | 0 | 0 ms |
| IDS-3 | 3 | 16 | 0 | 0 | 0 ms |
| UCS-G-3 | 3 | 10 | 13 | 14 | 1 ms |
| UCS-T-3 | 3 | 13 | 28 | 29 | 4 ms |
| GBFS-MisTileH-G-3 | 3 | 4 | 5 | 6 | 1 ms |
| GBFS-ManhH-G-3 | 3 | 3 | 5 | 6 | 0 ms |
| GBFS-MisTileH-T-3 | 3 | 4 | 8 | 9 | 0 ms |
| GBFS-ManhH-T-3 | 3 | 3 | 7 | 8 | 0 ms |
| A*-MisTileH-G-3 | 3 | 4 | 5 | 6 | 0 ms |
| A*-ManhH-G-3 | 3 | 3 | 5 | 6 | 0 ms |
| A*-MisTileH-T-3 | 3 | 4 | 8 | 9 | 0 ms |
| A*-ManhH-T-3 | 3 | 3 | 7 | 8 | 0 ms |
| BFS-G-9 | 9 | 1117 | 1208 | 1209 | 15 ms |
| BFS-T-9 | 9 | 14392 | 32177 | 32178 | 51 ms |
| DFS-G-9 | --- | --- | --- | --- | (2) |
| DFS-T-9 | --- | --- | --- | --- | (1) |
| DLS-9-9 | 9 | 14235 | 0 | 0 | 28 ms |
| DLS-3-9 | 0 | 13 | 0 | 0 | 0 ms |
| IDS-9 | --- | --- | --- | --- | (1) |
| UCS-G-9 | 9 | 2247 | 2370 | 2371 | 11 ms |
| UCS-T-9 | 9 | 28273 | 63422 | 63423 | 74 ms |
| GBFS-MisTileH-G-9 | 9 | 9 | 13 | 14 | 0 ms |
| GBFS-ManhH-G-9 | 9 | 9 | 13 | 14 | 0 ms |
| GBFS-MisTileH-T-9 | 9 | 9 | 21 | 22 | 0 ms |
| GBFS-ManhH-T-9 | 9 | 9 | 21 | 22 | 0 ms |
| A*-MisTileH-G-9 | 9 | 9 | 13 | 14 | 0 ms |
| A*-ManhH-G-9 | 9 | 9 | 13 | 14 | 0 ms |
| A*-MisTileH-T-9 | 9 | 9 | 21 | 22 | 0 ms |
| A*-ManhH-T-9 | 9 | 9 | 21 | 22 | 0 ms |
| BFS-G-30 | --- | --- | --- | --- | (2) |
| BFS-T-30 | --- | --- | --- | --- | (2) |
| DFS-G-30 | --- | --- | --- | --- | (2) |
| DFS-T-30 | --- | --- | --- | --- | (1) |
| DLS-9-30 | 0 | 14747 | 0 | 0 | 16 ms |
| DLS-3-30 | 0 | 13 | 0 | 0 | 0 ms |
| DLS-30-30 | --- | --- | --- | --- | (1) |
| IDS-30 | --- | --- | --- | --- | (1) |
| UCS-G-30 | --- | --- | --- | --- | (2) |
| UCS-T-30 | --- | --- | --- | --- | (2) |
| GBFS-MisTileH-G-30 | 194 | 10536 | 10710 | 10711 | 256 ms |
| GBFS-ManhH-G-30 | 120 | 761 | 862 | 863 | 19 ms |
| GBFS-MisTileH-T-30 | --- | --- | --- | --- | (1) |
| GBFS-ManhH-T-30 | --- | --- | --- | --- | (1) |
| A*-MisTileH-G-30 | 30 | 668049 | 613060 | 613061 | 7477 ms |
| A*-ManhH-G-30 | 30 | 606 | 600 | 601 | 5 ms |
| A*-MisTileH-T-30 | --- | --- | --- | --- | (1) |
| A*-ManhH-T-30 | --- | --- | --- | --- | (1) |

Como se puede ver en los resultados, el comportamiento de los algoritmos es similar al del problema del 8-puzzle. El factor de ramificación es el mismo, pero el número de estados posibles se incrementa, así que ahora los algoritmos tienen muchos problemas para encontrar soluciones sobretodo al buscar el árbol. Se puede ver como ya con la solución a 30 pasos, ninguno de las búsquedas no informadas encuentran solución, se hace necesario usar las búsquedas informadas.

# Referencias

[1] Stuart Rusell and Peter Norvig, *Artificial Intelligence: A Modern Approach*.

[2] *Apuntes de la asignatura Inteligencia Artificial*, Curso 2019-20.