Tarea 3 IA

Roberto Vásquez Martínez
Profesor: Arturo Hernández Aguirre

17/Noviembre/2021

1. Búsqueda Informada con algoritmo A^*

El objetivo en esta tarea será mejor notablemente la complejidad tanto en tiempo como en memoria la búsqueda ciega implementada en la Tarea 2. Lo que haremos será resolver el 8-Imposible con el algoritmo A^*. La función de costo que utilizaremos para el algoritmo A^* será de la forma

$$f(n) = g(n) + h'(n).$$

para todo nodo n. Aquí g representa la profunidad del nodo n y h' estimará la distancia del nodo n a la solución, la cual especificaremos posteriormente.

Denotaremos por $h_M(n)$ a la distancia Manhattan del nodo n a la solución y por $h_C(n)$ el número de fichas en posición incorrecta del tablero n respecto al tablero solución; a h_C también se le conoce como *distancia de Hamming*.

Por lo tanto, lo que haremos será implementar el algoritmo A^* considerando primero la función de costo $f_M(n) = g(n) + h_M(n)$ y luego la función $f_C(n) = g(n) + h_C(n)$ para resolver el 8-imposible.

En este notebook, al igual que en la Tarea 2, importamos las funciones y clases que nos permitirán obtener los resultados del archivo heuristic_search.py que se adjunta junto con un README.md en un zip, esto con el fin de darle prioridad a las observaciones y hacer más sucinto este documento.

El archivo heuristic_search.py y el notebook de Jupyter Tarea_3_IA.ipynb deben estar en el mismo directorio si se desea ejecutar el notebook de Jupyter.

1.1. Heurísticas Admisibles

Sabemos que

$$h_C(n) \leq h_M(n) \leq h^*(n) \tag{1}$$

para todo estado n y donde h^* es el verdadero coste para llegar de n al objetivo.

Como h_C y h_M con admisibles entonces por el Teorema 4.2.2 de Ginsberg, 1993 el algoritmo A^* con estas heurísticas nunca regresará una solución subóptima, por lo que el resultado obtenido con el algoritmo A^* será la solución óptima, es decir, la más corta de la configuración inicial a la final.

Por lo tanto, anticipamos que con la distancia Manhattan y con la distancia de Hamming obtendremos soluciones con la misma cantidad de movimientos, pues en ambos casos la solución es óptima.

1.1.1. Caso 1

En este caso probaremos con la configuración inicial sugerida

$$\begin{pmatrix} 3 & 2 & 1 \\ 6 & 5 & 4 \\ 8 & 7 & 0 \end{pmatrix},$$

mientras que la configuración final es

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix}.$$

Representamos al hueco del tablero como el número 0.

[1],
```
import sys
import numpy as np
sys.path.append('..')
from heuristic_search import A_star
from heuristic_search import Manhattan_dist
from heuristic_search import Counting_dist
# Case 1
init_value_1=np.array([[3,2,1],[6,5,4],[8,7,0]])
final_value_1=np.array([[1,2,3],[4,5,6],[7,8,0]])
# Heuristic search algorithm
A_star_1=A_star(init_value_1,final_value_1,Counting_dist)
A_star_1.main(Counting_dist)
```

```
No se puede alcanzar el nodo final
```

A partir de lo anterior vemos que a partir de la configuración incial proporcionada no es alcanzable la configuración inicial. La heurística empleada fue considerar como base la configuración

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix},$$

esta se puede puede poner en forma de vector

$$(1, 2, 3, 4, 5, 6, 7, 8, 0)$$

Cualquier configuración la podemos poner en forma de vector como la anterior. En cualquier configuración del tablero, una inversión entre las entradas i, j , con $i < j$, del vector asociado a la configuración sucede cuando el valor de la celda en la posición i es mayor que el valor en la posición j.

Usamos el hecho de que para poder alcanzar el estado final a partir de un estado inicial la paridad del número de inversiones del estado inicial y del estado final debe ser la misma.

Para poder alcanzar la solución lo que haremos será intercambiar dos fichas de lugar en la configuración inicial de forma que la paridad del número de inversiones cambie y sea la misma que en la configuración final. Por lo que usaremos como configuración inicial el siguiente tablero

$$\begin{pmatrix} 3 & 2 & 1 \\ 6 & 5 & 4 \\ 7 & 8 & 0 \end{pmatrix}.$$

Distancia de Hamming Mostraremos los resultados correspondientes al algoritmo A^* considerando la distancia de Hamming. A continuación presentamos el código que nos brinda la solución con los resultados.

[2],
```
# Case 1 (Hamming distance)
init_value_1=np.array([[3,2,1],[6,5,4],[7,8,0]])
final_value_1=np.array([[1,2,3],[4,5,6],[7,8,0]])
# Heuristic search algorithm
A_star_1=A_star(init_value_1,final_value_1,Counting_dist)
A_star_1.main(Counting_dist)
```

```
Numero de movimientos de la solución: 24
Numero de nodos visitados: 12117
Numero de nodos por visitar: 6444
Numero de nodos expandidos: 18561
```

El tiempo de ejecución en este caso fue de 9m36s y el número de nodos expandidos fue 18561.

Los caminos completos de la configuración inicial a la configuración final para cada caso se encuentras en el apéndice A

Distancia Manhattan A continuación mostramos los resultados obtenidos con el algoritmo A^* utilizando la distancia Manhattan.

[3],
```
# Case 1 (Manhattan distance)
init_value_1=np.array([[3,2,1],[6,5,4],[7,8,0]])
final_value_1=np.array([[1,2,3],[4,5,6],[7,8,0]])
# Heuristic search algorithm
A_star_1=A_star(init_value_1,final_value_1,Manhattan_dist)
A_star_1.main(Manhattan_dist)
```

```
Numero de movimientos de la solución: 24
Numero de nodos visitados: 1628
Numero de nodos por visitar: 875
Numero de nodos expandidos: 2503
```

El tiempo de ejecución en este caso fue de 12.3s. Por (1) sabemos que el algoritmo A^* que usa la distancia Manhattan es *más informado* que el que usa la distancia de Hamming, lo que quiere decir que si un nodo es expandido con la distancia de Manhattan es necesariamente expandido por el algoritmo A^* con la distancia de Hamming.

Por lo tanto, el número de nodos expandidos por el algoritmo A^* con la distancia Manhattan es menor o igual que el número de nodos expandidos por el algoritmo A^* con la distancia de Ham-

ming, por lo que con la distancia Manhattan la búsqueda es más eficiente en tiempo y memoria lo que se puede corroborar con el ejemplo anterior, pues al algoritmo A^* con la distancia de Hamming le toma 9m26s y 18561 nodos expandidos hallar la solución mientras que al algoritmo A^* con la distancia de Manhattan le toma 12.3s y 2503 nodos expandidos.

Sin embargo, al ser ambas distancias admisibles, la solución en ambos casos es óptima, como lo habíamos anticipado, que en este caso es una solución en 24 movimientos para ambas heurísticas.

1.1.2. Caso 2

Como lo solicita el problema haremos dos pruebas más. Para este caso usaremos la configuración inicial

$$\begin{pmatrix} 6 & 2 & 8 \\ 4 & 0 & 5 \\ 1 & 7 & 3 \end{pmatrix},$$

y como configuración final el siguiente tablero

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 0 & 5 \\ 6 & 7 & 8 \end{pmatrix}.$$

Distancia de Hamming Corremos el algoritmo A^* con la pareja de tableros anterior considerando la distancia de Hamming.

[4],
```
# Case 2 (Hamming distance)
init_value_2=np.array([[6,2,8],[4,0,5],[1,7,3]])
final_value_2=np.array([[1,2,3],[4,0,5],[6,7,8]])
# Heuristic search algorithm
A_star_2=A_star(init_value_2,final_value_2,Counting_dist)
A_star_2.main(Counting_dist)
```

```
Numero de movimientos de la solución: 24
Numero de nodos visitados: 18786
Numero de nodos por visitar: 9378
Numero de nodos expandidos: 28164
```

El tiempo de ejecución con la distancia de Hamming en este caso fue de 22m46s y el número de nodos expandidos fue de 28164. Ahora contrastaremos estos resultados con los obtenidos por la distancia Manhattan.

Distancia Manhattan Para el par de estados inicial y final anterior buscamos la solución utilizando la distancia Manhattan.

[5],
```
# Case 2 (Hamming distance)
init_value_2=np.array([[6,2,8],[4,0,5],[1,7,3]])
final_value_2=np.array([[1,2,3],[4,0,5],[6,7,8]])
# Heuristic search algorithm
A_star_2=A_star(init_value_2,final_value_2,Manhattan_dist)
A_star_2.main(Manhattan_dist)
```

```
Numero de movimientos de la solución: 24
Numero de nodos visitados: 1548
Numero de nodos por visitar: 894
Numero de nodos expandidos: 2442
```

Al igual que con la configuración sugerida tenemos un tiempo de 11.9s y 2442 nodos implementados, en este caso la diferencia de eficiencia entre el algorimo A^* con la distancia de Hamming y con la distancia Manhattan es todavía más notable. Estos dos ejemplos nos enseñan el beneficio de una heurística admisible más informada. De igual forma que en el caso 1, ambas heurísticas nos dan la solución óptima.

1.1.3. Caso 3

Finalmente, correremos un último par de tableros. La configuración inicial será

$$\begin{pmatrix} 1 & 2 & 3 \\ 8 & 0 & 4 \\ 6 & 5 & 7 \end{pmatrix},$$

mientras que la configuración final será

$$\begin{pmatrix} 1 & 2 & 3 \\ 8 & 0 & 4 \\ 7 & 6 & 5 \end{pmatrix}.$$

Distancia de Hamming Los resultados con la distancia de Hamming son los siguientes

[9],
```
# Case 3 (Hamming distance)
init_value_3=np.array([[1,2,3],[8,0,4],[6,5,7]])
final_value_3=np.array([[1,2,3],[8,0,4],[7,6,5]])
# Heuristic search algorithm
A_star_3=A_star(init_value_3,final_value_3,Counting_dist)
A_star_3.main(Counting_dist)
```

```
Numero de movimientos de la solución: 16
Numero de nodos visitados: 525
Numero de nodos por visitar: 362
Numero de nodos expandidos: 887
```

La idea con esta configuración fue empezar con una configuración con distancia de Hamming menor que en los casos 1 y 2. Mientras que en los casos 1 y 2 la distancia de Hamming es de 4, en este caso es de 3, lo que verificamos en el siguiente código.

[10],
```
print('Distancia de Hamming Caso 1, %d '␣
 ↪%(Counting_dist(init_value_1,final_value_1)))
print('Distancia de Hamming Caso 2, %d '␣
 ↪%(Counting_dist(init_value_2,final_value_2)))
print('Distancia de Hamming Caso 3, %d '␣
 ↪%(Counting_dist(init_value_3,final_value_3)))
```

```
Distancia de Hamming Caso 1: 4
Distancia de Hamming Caso 2: 4
Distancia de Hamming Caso 3: 3
```

El resultado que obtuvimos fue que en el Caso 3 el algoritmo A^* con la distancia de Hamming tiene un mejor desempeño obtienendo la solución óptima de 16 movimientos en 1.5s y expandiendo 887 nodos.

Distancia Manhattan A continuación analizamos el desempeño del algoritmo A^* con la distancia Manhattan con la configuración del Caso 3, esperamos que sea muy rápida dado el resultado con la distancia de Hamming.

[11],
```
# Case 3 (Hamming distance)
init_value_3=np.array([[1,2,3],[8,0,4],[6,5,7]])
final_value_3=np.array([[1,2,3],[8,0,4],[7,6,5]])
# Heuristic search algorithm
A_star_3=A_star(init_value_3,final_value_3,Manhattan_dist)
A_star_3.main(Manhattan_dist)
```

```
Numero de movimientos de la solución: 16
Numero de nodos visitados: 165
Numero de nodos por visitar: 111
Numero de nodos expandidos: 276
```

En este caso, le tomó 0.2s llegar a la solución y 276 nodos expandidos.

Observamos que cuanto más cerca estemos de h^* mayor será la tendencia de explorar en profundidad y viajar directamente por el camino óptimo. Esta es la razón por la que el número de nodos expandidos se ve reducido drásticamente usando h_M que usando h_C, pues con h_M estamos más cerca de la búsqueda en profundidad que converge directo a la solución, que como vimos en la Tarea 2 es más eficiente en memoria, en consecuencia al expandir menos nodos se reduce el tiempo de cómputo.

Sin embargo una limitación de la distancia Manhattan es que considera a cada ficha independiente de las demás, mientras que en realidad una ficha puede interferir en el camino de las demás. Hay una mejora a la distancia Manhattan llamada *Conflicto Lineal* (Kondekar, 2014, pp. 423), que mejoraría aún más la eficiencia del algoritmo A^*.

1.2. Heurística no admisible

Para el tablero n consideramos P_n el número de inversiones del tablero; las inversiones las hemos definido antes. La heurística que proponemos, considerando a f como el tablero final, es

$$h_P(n) = |P_n - P_f|.$$

Notamos que para el tablero

$$n = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 0 \\ 7 & 8 & 6 \end{pmatrix},$$

y tablero final

$$f = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix},$$

se tiene que $h_P(n)$=2, lo que podemos verificar con el siguiente código

[13],
```
from heuristic_search import Inversion_dist
n=np.array([[1,2,3],[4,5,0],[7,8,6]])
f=np.array([[1,2,3],[4,5,6],[7,8,0]])
print('h_P(n)= %d' %(Inversion_dist(n,f)))
```

```
h_P(n)= 2
```

No es díficil ver que podemos obtener f a partir de n con un sólo movimiento, desplazando el 6 al espacio vacío en n, luego $h^*(n) = 1$ por lo que en este caso $h_P(n) > h^*(n)$.

Por lo tanto h_P es una heurística no admisible.

A continuación probaremos cada par de configuraciones en los Casos 1, 2 y 3 con esta heurística h_P para analizar el efecto que tiene una heurística no admisible en el algoritmo A^*

1.2.1. Caso 1

A continuación presentamos el desempeño del algoritmo A^* con la heurística h_P.

[14],
```
# Case 1 (Inversion distance)
init_value_1=np.array([[3,2,1],[6,5,4],[7,8,0]])
final_value_1=np.array([[1,2,3],[4,5,6],[7,8,0]])
# Heuristic search algorithm
A_star_1=A_star(init_value_1,final_value_1,Inversion_dist)
A_star_1.main(Inversion_dist)
```

```
Numero de movimientos de la solución: 24
Numero de nodos visitados: 4547
Numero de nodos por visitar: 2428
Numero de nodos expandidos: 6975
```

El algoritmo A^* en este caso le tomo 1m30s obtener la solución y expandío 6975 nodos. Como hemos visto h_P en ocasiones sobreestima el costo real h^* lo que puede provocar que que el algoritmo A^* con h_P sea más rápido, en este caso fue más rápido que considerando la heurística h_C pero al menos aquí no fue mejor que el algoritmo A^* con h_M.

1.2.2. Caso 2

Ahora probaremos el desempeño del algoritmo A^* con h_P y las configuraciones del caso 2.

[15],
```
# Case 2 (Hamming distance)
init_value_2=np.array([[6,2,8],[4,0,5],[1,7,3]])
final_value_2=np.array([[1,2,3],[4,0,5],[6,7,8]])
```

```
# Heuristic search algorithm
A_star_2=A_star(init_value_2,final_value_2,Inversion_dist)
A_star_2.main(Inversion_dist)
```

```
Numero de movimientos de la solución: 24
Numero de nodos visitados: 1219
Numero de nodos por visitar: 786
Numero de nodos expandidos: 2005
```

En este caso le tomó al algoritmo A^* con la heurística h_P 8.1s completar el cómputo y necesitó expandir 2005 nodos. Este comportamiento fue incluso un poco mejor que usando la distancia Manhattan donde se ocupó 11.9s en el cómputo y se expandieron 2442 nodos.

1.2.3. Caso 3

Finalmente analizaremos el comportamiento del algoritmo A^* con la heurística h_P y las configuraciones del caso 3.

[16],
```
# Case 3 (Hamming distance)
init_value_3=np.array([[1,2,3],[8,0,4],[6,5,7]])
final_value_3=np.array([[1,2,3],[8,0,4],[7,6,5]])
# Heuristic search algorithm
A_star_3=A_star(init_value_3,final_value_3,Inversion_dist)
A_star_3.main(Inversion_dist)
```

```
Numero de movimientos de la solución: 16
Numero de nodos visitados: 2531
Numero de nodos por visitar: 1643
Numero de nodos expandidos: 4174
```

En este caso el cómputo tomó 32.6s y se expendieron 4174 nodos, un compartamiento peor que la distancia de Hamming donde el cómputo se completó en 1.5s y 887 nodos expandidos y peor que la distancia Manhattan que ocupó de 0.2s y 276 nodos expandidos.

Podemos concluir que en promedio el algoritmo A^* con la heurística h_P se comparta mejor que con la heurística h_C, pero es peor que con la heurística h_M. Esta proposición se basa sólo en los casos analizados.

Cabe destacar que a pesar de que h_P no es admisible en cada uno de los casos se encontró una solución óptima, pero esto no está garantizado para todo par de configuraciones, es decir, el algoritmo A^* con la heurística h_P puede regresar soluciones subóptimas pues h_P no es admisible.

A. Salidas

Para no interrumpir la lectura del reporte con la sucesión de movimientos que lleva a la configuración final se concentran las soluciones en este apéndice.

En este apéndice se considera a los tableros en forma de vector como se ha dicho antes y el orden de los movimientos es de izquierda a derecha en las filas y moviéndose de arriba a abajo entre filas.

A.1. Solución Caso 1 Distancia de Hamming

(3 , 2 , 1, 6 , 5 , 4, 7 , 8 , 0) (3 , 2 , 1, 6 , 5 , 4, 7 , 0 , 8) (3 , 2 , 1, 6 , 0 , 4, 7 , 5 , 8) (3 , 2 , 1, 0 , 6 , 4, 7 , 5 , 8)

(0 , 2 , 1, 3 , 6 , 4, 7 , 5 , 8) (2 , 0 , 1, 3 , 6 , 4, 7 , 5 , 8) (2 , 6 , 1, 3 , 0 , 4, 7 , 5 , 8) (2 , 6 , 1, 3 , 4 , 0, 7 , 5 , 8)

(2 , 6 , 0, 3 , 4 , 1, 7 , 5 , 8) (2 , 0 , 6, 3 , 4 , 1, 7 , 5 , 8) (2 , 4 , 6, 3 , 0 , 1, 7 , 5 , 8) (2 , 4 , 6, 0 , 3 , 1, 7 , 5 , 8)

(0 , 4 , 6, 2 , 3 , 1, 7 , 5 , 8) (4 , 0 , 6, 2 , 3 , 1, 7 , 5 , 8) (4 , 3 , 6, 2 , 0 , 1, 7 , 5 , 8) (4 , 3 , 6, 2 , 1 , 0, 7 , 5 , 8)

(4 , 3 , 0, 2 , 1 , 6, 7 , 5 , 8) (4 , 0 , 3, 2 , 1 , 6, 7 , 5 , 8) (4 , 1 , 3, 2 , 0 , 6, 7 , 5 , 8) (4 , 1 , 3, 0 , 2 , 6, 7 , 5 , 8)

(0 , 1 , 3, 4 , 2 , 6, 7 , 5 , 8) (1 , 0 , 3, 4 , 2 , 6, 7 , 5 , 8) (1 , 2 , 3, 4 , 0 , 6, 7 , 5 , 8) (1 , 2 , 3, 4 , 5 , 6, 7 , 0 , 8)

(1 , 2 , 3, 4 , 5 , 6, 7 , 8 , 0)

A.2. Solución Caso 1 Distancia Manhattan

(3 , 2 , 1, 6 , 5 , 4, 7 , 8 , 0) (3 , 2 , 1, 6 , 5 , 4, 7 , 0 , 8) (3 , 2 , 1, 6 , 0 , 4, 7 , 5 , 8) (3 , 2 , 1, 0 , 6 , 4, 7 , 5 , 8)

(0 , 2 , 1, 3 , 6 , 4, 7 , 5 , 8) (2 , 0 , 1, 3 , 6 , 4, 7 , 5 , 8) (2 , 6 , 1, 3 , 0 , 4, 7 , 5 , 8) (2 , 6 , 1, 3 , 4 , 0, 7 , 5 , 8)

(2 , 6 , 0, 3 , 4 , 1, 7 , 5 , 8) (2 , 0 , 6, 3 , 4 , 1, 7 , 5 , 8) (2 , 4 , 6, 3 , 0 , 1, 7 , 5 , 8) (2 , 4 , 6, 0 , 3 , 1, 7 , 5 , 8)

(0 , 4 , 6, 2 , 3 , 1, 7 , 5 , 8) (4 , 0 , 6, 2 , 3 , 1, 7 , 5 , 8) (4 , 3 , 6, 2 , 0 , 1, 7 , 5 , 8) (4 , 3 , 6, 2 , 1 , 0, 7 , 5 , 8)

(4 , 3 , 0, 2 , 1 , 6, 7 , 5 , 8) (4 , 0 , 3, 2 , 1 , 6, 7 , 5 , 8) (4 , 1 , 3, 2 , 0 , 6, 7 , 5 , 8) (4 , 1 , 3, 0 , 2 , 6, 7 , 5 , 8)

(0 , 1 , 3, 4 , 2 , 6, 7 , 5 , 8) (1 , 0 , 3, 4 , 2 , 6, 7 , 5 , 8) (1 , 2 , 3, 4 , 0 , 6, 7 , 5 , 8) (1 , 2 , 3, 4 , 5 , 6, 7 , 0 , 8)

(1 , 2 , 3, 4 , 5 , 6, 7 , 8 , 0)

A.3. Solución Caso 2 Distancia de Hamming

(6 , 2 , 8, 4 , 0 , 5, 1 , 7 , 3) (6 , 2 , 8, 0 , 4 , 5, 1 , 7 , 3) (0 , 2 , 8, 6 , 4 , 5, 1 , 7 , 3) (2 , 0 , 8, 6 , 4 , 5, 1 , 7 , 3)

(2 , 8 , 0, 6 , 4 , 5, 1 , 7 , 3) (2 , 8 , 5, 6 , 4 , 0, 1 , 7 , 3) (2 , 8 , 5, 6 , 4 , 3, 1 , 7 , 0) (2 , 8 , 5, 6 , 4 , 3, 1 , 0 , 7)

(2 , 8 , 5, 6 , 0 , 3, 1 , 4 , 7) (2 , 0 , 5, 6 , 8 , 3, 1 , 4 , 7) (2 , 5 , 0, 6 , 8 , 3, 1 , 4 , 7) (2 , 5 , 3, 6 , 8 , 0, 1 , 4 , 7)

(2 , 5 , 3, 6 , 0 , 8, 1 , 4 , 7) (2 , 5 , 3, 0 , 6 , 8, 1 , 4 , 7) (2 , 5 , 3, 1 , 6 , 8, 0 , 4 , 7) (2 , 5 , 3, 1 , 6 , 8, 4 , 0 , 7)

(2 , 5 , 3, 1 , 0 , 8, 4 , 6 , 7) (2 , 0 , 3, 1 , 5 , 8, 4 , 6 , 7) (0 , 2 , 3, 1 , 5 , 8, 4 , 6 , 7) (1 , 2 , 3, 0 , 5 , 8, 4 , 6 , 7)

(1 , 2 , 3, 4 , 5 , 8, 0 , 6 , 7) (1 , 2 , 3, 4 , 5 , 8, 6 , 0 , 7) (1 , 2 , 3, 4 , 5 , 8, 6 , 7 , 0) (1 , 2 , 3, 4 , 5 , 0, 6 , 7 , 8)

(1 , 2 , 3, 4 , 0 , 5, 6 , 7 , 8)

A.4. Solución Caso 2 Distancia Manhattan

(6 , 2 , 8, 4 , 0 , 5, 1 , 7 , 3) (6 , 2 , 8, 0 , 4 , 5, 1 , 7 , 3) (0 , 2 , 8, 6 , 4 , 5, 1 , 7 , 3) (2 , 0 , 8, 6 , 4 , 5, 1 , 7 , 3)

(2 , 8 , 0, 6 , 4 , 5, 1 , 7 , 3) (2 , 8 , 5, 6 , 4 , 0, 1 , 7 , 3) (2 , 8 , 5, 6 , 4 , 3, 1 , 7 , 0) (2 , 8 , 5, 6 , 4 , 3, 1 , 0 , 7)

(2 , 8 , 5, 6 , 0 , 3, 1 , 4 , 7) (2 , 8 , 5, 0 , 6 , 3, 1 , 4 , 7) (2 , 8 , 5, 1 , 6 , 3, 0 , 4 , 7) (2 , 8 , 5, 1 , 6 , 3, 4 , 0 , 7)

(2 , 8 , 5, 1 , 0 , 3, 4 , 6 , 7) (2 , 0 , 5, 1 , 8 , 3, 4 , 6 , 7,) (2 , 5 , 0, 1 , 8 , 3, 4 , 6 , 7) (2 , 5 , 3, 1 , 8 , 0, 4 , 6 , 7)

(2 , 5 , 3, 1 , 0 , 8, 4 , 6 , 7) (2 , 0 , 3, 1 , 5 , 8, 4 , 6 , 7) (0 , 2 , 3, 1 , 5 , 8, 4 , 6 , 7) (1 , 2 , 3, 0 , 5 , 8, 4 , 6 , 7)

(1 , 2 , 3, 4 , 5 , 8, 0 , 6 , 7) (1 , 2 , 3, 4 , 5 , 8, 6 , 0 , 7) (1 , 2 , 3, 4 , 5 , 8, 6 , 7 , 0) (1 , 2 , 3, 4 , 5 , 0, 6 , 7 , 8)

(1 , 2 , 3, 4 , 0 , 5, 6 , 7 , 8)

A.5. Solución Caso 3 Distancia de Hamming

(1 , 2 , 3, 8 , 0 , 4, 6 , 5 , 7) (1 , 2 , 3, 8 , 5 , 4, 6 , 0 , 7) (1 , 2 , 3, 8 , 5 , 4, 0 , 6 , 7) (1 , 2 , 3, 0 , 5 , 4, 8 , 6 , 7)

(1 , 2 , 3, 5 , 0 , 4, 8 , 6 , 7) (1 , 2 , 3, 5 , 6 , 4, 8 , 0 , 7) (1 , 2 , 3, 5 , 6 , 4, 8 , 7 , 0) (1 , 2 , 3, 5 , 6 , 0, 8 , 7 , 4)

(1 , 2 , 3, 5 , 0 , 6, 8 , 7 , 4) (1 , 2 , 3, 0 , 5 , 6, 8 , 7 , 4) (1 , 2 , 3, 8 , 5 , 6, 0 , 7 , 4) (1 , 2 , 3, 8 , 5 , 6, 7 , 0 , 4)

(1 , 2 , 3, 8 , 0 , 6, 7 , 5 , 4) (1 , 2 , 3, 8 , 6 , 0, 7 , 5 , 4) (1 , 2 , 3, 8 , 6 , 4, 7 , 5 , 0) (1 , 2 , 3, 8 , 6 , 4, 7 , 0 , 5)

(1 , 2 , 3, 8 , 0 , 4, 7 , 6 , 5)

A.6. Solución Caso 3 Distancia Manhattan

(1 , 2 , 3, 8 , 0 , 4, 6 , 5 , 7) (1 , 2 , 3, 8 , 4 , 0, 6 , 5 , 7) (1 , 2 , 3, 8 , 4 , 7, 6 , 5 , 0) (1 , 2 , 3, 8 , 4 , 7, 6 , 0 , 5)

(1 , 2 , 3, 8 , 4 , 7, 0 , 6 , 5) (1 , 2 , 3, 0 , 4 , 7, 8 , 6 , 5) (1 , 2 , 3, 4 , 0 , 7, 8 , 6 , 5) (1 , 2 , 3, 4 , 7 , 0, 8 , 6 , 5)

(1 , 2 , 3, 4 , 7 , 5, 8 , 6 , 0) (1 , 2 , 3, 4 , 7 , 5, 8 , 0 , 6) (1 , 2 , 3, 4 , 0 , 5, 8 , 7 , 6) (1 , 2 , 3, 0 , 4 , 5, 8 , 7 , 6)

(1 , 2 , 3, 8 , 4 , 5, 0 , 7 , 6) (1 , 2 , 3, 8 , 4 , 5, 7 , 0 , 6) (1 , 2 , 3, 8 , 4 , 5, 7 , 6 , 0) (1 , 2 , 3, 8 , 4 , 0, 7 , 6 , 5)

(1 , 2 , 3, 8 , 0 , 4, 7 , 6 , 5)

A.7. Solución Caso 1 Heurística no admisible

(3 , 2 , 1, 6 , 5 , 4, 7 , 8 , 0) (3 , 2 , 1, 6 , 5 , 4, 7 , 0 , 8) (3 , 2 , 1, 6 , 0 , 4, 7 , 5 , 8) (3 , 0 , 1, 6 , 2 , 4, 7 , 5 , 8)

(0 , 3 , 1, 6 , 2 , 4, 7 , 5 , 8) (6 , 3 , 1, 0 , 2 , 4, 7 , 5 , 8) (6 , 3 , 1, 2 , 0 , 4, 7 , 5 , 8) (6 , 0 , 1, 2 , 3 , 4, 7 , 5 , 8)

(6 , 1 , 0, 2 , 3 , 4, 7 , 5 , 8) (6 , 1 , 4, 2 , 3 , 0, 7 , 5 , 8) (6 , 1 , 4, 2 , 0 , 3, 7 , 5 , 8) (6 , 0 , 4, 2 , 1 , 3, 7 , 5 , 8)

(0 , 6 , 4, 2 , 1 , 3, 7 , 5 , 8) (2 , 6 , 4, 0 , 1 , 3, 7 , 5 , 8) (2 , 6 , 4, 1 , 0 , 3, 7 , 5 , 8) (2 , 0 , 4, 1 , 6 , 3, 7 , 5 , 8)

(2 , 4 , 0, 1 , 6 , 3, 7 , 5 , 8) (2 , 4 , 3, 1 , 6 , 0, 7 , 5 , 8) (2 , 4 , 3, 1 , 0 , 6, 7 , 5 , 8) (2 , 0 , 3, 1 , 4 , 6, 7 , 5 , 8)

(0 , 2 , 3, 1 , 4 , 6, 7 , 5 , 8) (1 , 2 , 3, 0 , 4 , 6, 7 , 5 , 8) (1 , 2 , 3, 4 , 0 , 6, 7 , 5 , 8) (1 , 2 , 3, 4 , 5 , 6, 7 , 0 , 8)

(1 , 2 , 3, 4 , 5 , 6, 7 , 8 , 0)

A.8. Solución Caso 2 Heurística no admisible

(6 , 2 , 8, 4 , 0 , 5, 1 , 7 , 3) (6 , 2 , 8, 0 , 4 , 5, 1 , 7 , 3) (6 , 2 , 8, 1 , 4 , 5, 0 , 7 , 3) (6 , 2 , 8, 1 , 4 , 5, 7 , 0 , 3)

(6 , 2 , 8, 1 , 4 , 5, 7 , 3 , 0) (6 , 2 , 8, 1 , 4 , 0, 7 , 3 , 5) (6 , 2 , 0, 1 , 4 , 8, 7 , 3 , 5) (6 , 0 , 2, 1 , 4 , 8, 7 , 3 , 5)

(6 , 4 , 2, 1 , 0 , 8, 7 , 3 , 5) (6 , 4 , 2, 1 , 3 , 8, 7 , 0 , 5) (6 , 4 , 2, 1 , 3 , 8, 7 , 5 , 0) (6 , 4 , 2, 1 , 3 , 0, 7 , 5 , 8)

(6 , 4 , 2, 1 , 0 , 3, 7 , 5 , 8) (6 , 4 , 2, 0 , 1 , 3, 7 , 5 , 8) (0 , 4 , 2, 6 , 1 , 3, 7 , 5 , 8) (4 , 0 , 2, 6 , 1 , 3, 7 , 5 , 8)

(4 , 1 , 2, 6 , 0 , 3, 7 , 5 , 8) (4 , 1 , 2, 6 , 5 , 3, 7 , 0 , 8) (4 , 1 , 2, 6 , 5 , 3, 0 , 7 , 8) (4 , 1 , 2, 0 , 5 , 3, 6 , 7 , 8)

(0 , 1 , 2, 4 , 5 , 3, 6 , 7 , 8) (1 , 0 , 2, 4 , 5 , 3, 6 , 7 , 8) (1 , 2 , 0, 4 , 5 , 3, 6 , 7 , 8) (1 , 2 , 3, 4 , 5 , 0, 6 , 7 , 8)

(1 , 2 , 3, 4 , 0 , 5, 6 , 7 , 8)

A.9. Solución Caso 3 Heurística no admisible

(1 , 2 , 3, 8 , 0 , 4, 6 , 5 , 7) (1 , 2 , 3, 0 , 8 , 4, 6 , 5 , 7) (1 , 2 , 3, 6 , 8 , 4, 0 , 5 , 7) (1 , 2 , 3, 6 , 8 , 4, 5 , 0 , 7)

(1 , 2 , 3, 6 , 8 , 4, 5 , 7 , 0) (1 , 2 , 3, 6 , 8 , 0, 5 , 7 , 4) (1 , 2 , 3, 6 , 0 , 8, 5 , 7 , 4) (1 , 2 , 3, 6 , 7 , 8, 5 , 0 , 4)

(1 , 2 , 3, 6 , 7 , 8, 0 , 5 , 4) (1 , 2 , 3, 0 , 7 , 8, 6 , 5 , 4) (1 , 2 , 3, 7 , 0 , 8, 6 , 5 , 4) (1 , 2 , 3, 7 , 8 , 0, 6 , 5 , 4)

(1 , 2 , 3, 7 , 8 , 4, 6 , 5 , 0) (1 , 2 , 3, 7 , 8 , 4, 6 , 0 , 5) (1 , 2 , 3, 7 , 8 , 4, 0 , 6 , 5) (1 , 2 , 3, 0 , 8 , 4, 7 , 6 , 5)

(1 , 2 , 3, 8 , 0 , 4, 7 , 6 , 5)

Referencias

Béjar, J. (2021). Búsqueda Heurística.

Ginsberg, M. (1993). *Essentials of Artificial Intelligence* (M. B. Morgan, Ed.). Morgan Kaufmann Publishers, Inc.

Kondekar, R. P. (2014). Implementation and Analysis of Iterative MapReduce Based Heuristic Algorithm for Solving N-Puzzle. *Journal of Computers*, *9*(2).