

Nombre: José Gabriel García

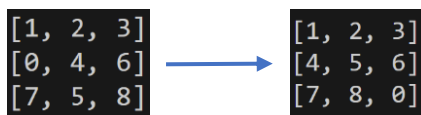
Código: 00211322

NRC:

Tarea 1: Inteligencia Artificial

Ejercicio 1: 8-tile

Para poder comparar los resultados de cada algoritmo se usa el mismo estado inicial y el mismo estado final de modo que los resultados puedan ser comparables. Sin embargo, para realizar los árboles/grafos generados en cada búsqueda se utilizan estados iniciales y finales diferentes ya que con ciertos estados iniciales y finales se obtienen estructuras demasiado grandes que se vuelven imposibles de graficar. De esta manera, el estado inicial y final para realizar una comparación son los siguientes:

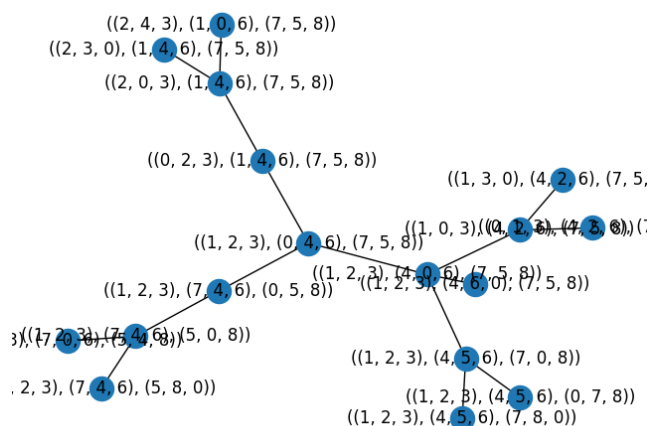


	BFS	DFS	Best-First Search (Baldosas bien ubicadas)	Best-First Search (Distancia Euclideana)	Best-First Search (Distancia Manhattan)
Elementos Visitados	17	853	4	33465	4

Al observar los estados visitados se puede observar que al estado final llegan todas las estrategias utilizadas, sin embargo, el camino que sigue cada una es evidentemente diferente ya que se tiene una diferente cantidad de estados visitados en con cada estrategia, y aunque el Best-First Search con una heurística de baldosas bien ubicadas y el que usa la distancia de Manhattan tiene el mismo numero de estados visitados, y, de hecho, el mismo path, no significa que siempre van a seguir el mismo camino ya que el valor que toma la heurística puede variar con cada estado.

BFS (se usa el mismo estado inicial y final antes mencionado)

Grafo generado



Profundidad: 4

Ancho: 8

Camino a la solución:

```
[1, 2, 3]
[0, 4, 6]
[7, 5, 8]

[1, 2, 3]
[4, 0, 6]
[7, 5, 8]

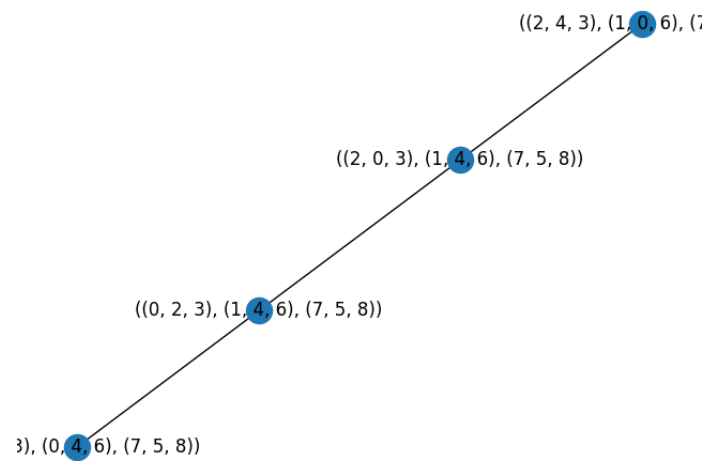
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]

[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
```

DFS: Estado inicial y final utilizados

```
[1, 2, 3]
[0, 4, 6]
[7, 5, 8] → [2, 4, 3]
               [1, 0, 6]
               [7, 5, 8]
```

Grafo generado



Profundidad: 4

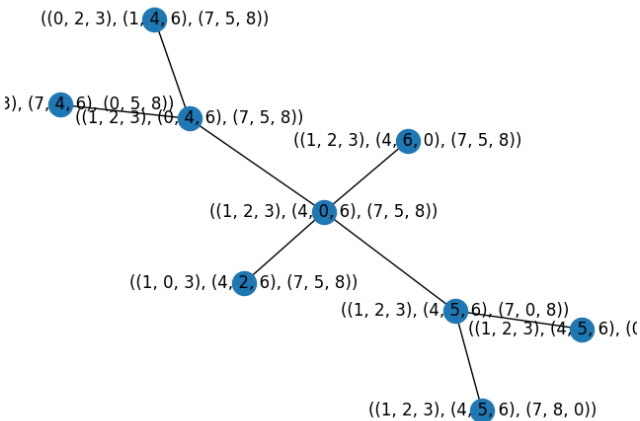
Ancho: 1

Camino a la solución:

[1, 2, 3]
[0, 4, 6]
[7, 5, 8]
[0, 2, 3]
[1, 4, 6]
[7, 5, 8]
[2, 0, 3]
[1, 4, 6]
[7, 5, 8]
[2, 4, 3]
[1, 0, 6]
[7, 5, 8]

Best-First Search con heurística de baldosas bien ubicadas (se usa el mismo estado inicial y final mencionado inicialmente)

Grafo generado



Profundidad: 4

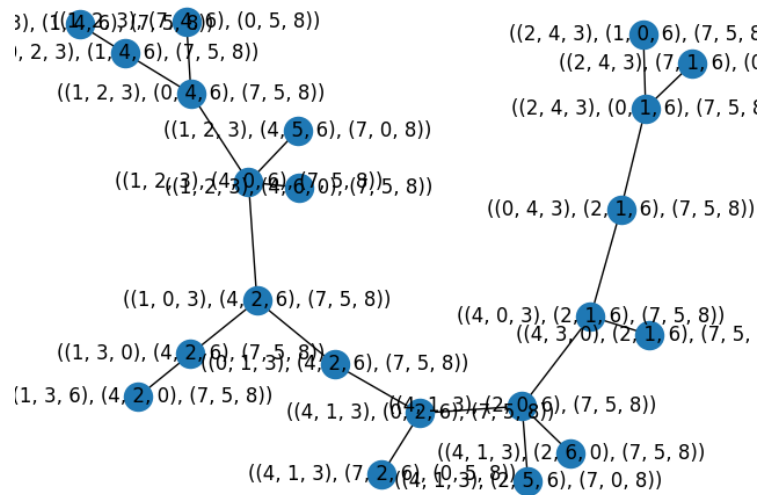
Ancho: 3

Camino a la solución:

[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]

Best-First Search con heurística de distancia Eucladiana (se usa el mismo estado inicial y final mencionado en DFS)

Grafo generado



Profundidad: 9

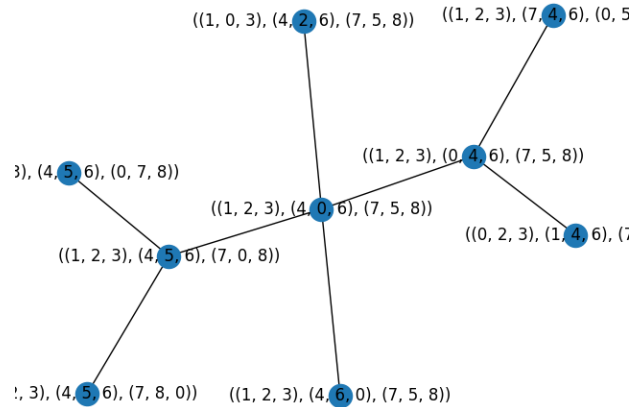
Ancho: 3

Camino a la solución:

[1, 2, 3]	[4, 1, 3]
[0, 4, 6]	[2, 0, 6]
[7, 5, 8]	[7, 5, 8]
[1, 2, 3]	[4, 0, 3]
[4, 0, 6]	[2, 1, 6]
[7, 5, 8]	[7, 5, 8]
[1, 0, 3]	[0, 4, 3]
[4, 2, 6]	[2, 1, 6]
[7, 5, 8]	[7, 5, 8]
[0, 1, 3]	[2, 4, 3]
[4, 2, 6]	[0, 1, 6]
[7, 5, 8]	[7, 5, 8]
[4, 1, 3]	[2, 4, 3]
[0, 2, 6]	[1, 0, 6]
[7, 5, 8]	[7, 5, 8]

Best-First Search con heurística de distancia Manhattan (se usa el mismo estado inicial y final mencionado inicialmente)

Grafo generado



Profundidad: 3

Ancho: 3

Camino a la solución:

```
[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]
```

Finalmente, después de realizar el ejercicio observé que el método de búsqueda Best-First Search es el mejor de todos ya que lo hace de manera más lógica, y no lo realiza ciegamente o “por fuerza bruta” como lo hace el BFS o DFS, por lo que llega a la solución más rápido, lo que lleva al uso de menos memoria y tiempo. Sin embargo, el hecho de que el Best-First Search utilice una manera para caracterizar cada estado que visita y hacer la búsqueda más rápida, la eficiencia depende de la heurística utilizada ya que como se pudo observar previamente, la heurística de distancia euclidiana generó más estados visitados que el DFS o BFS, por lo que, a pesar de ser un algoritmo de búsqueda un poco más racional, los resultados que genera son peores. Por otro lado, con este ejercicio se pudo observar que la mejor heurística es la de baldosas bien ubicadas o de la distancia de Manhattan, pero esto depende del estado final al que se quiere llegar, y el hecho de que las dos hayan obtenida los mejores resultados no indica que lo harán siempre, y pueden existir casos en los que una es mejor que la otra, y viceversa. De esta manera, se puede concluir que puede o no existir una heurística única e inmejorable,

pero esto se determinará después de realizar varias pruebas y compararla con otras. Así, el objetivo a alcanzar parece claro, y para obtener la mejor heurística se debe tener en cuenta los siguientes aspectos:

- **Admisibilidad:** hace referencia a una heurística que nunca sobrestima el costo de llegar a la meta. En el caso de las heurísticas utilizadas, puede ser que la de Distancia Euclidiana la sobrestime y es por eso le toma más tiempo en encontrar el estado objetivo.
- **Consistencia:** Este aspecto implica que, si se aplica una heurística a un nodo y luego se expande hacia un nodo sucesor, el valor de la heurística del nodo sucesor más el costo de llegar a ese sucesor no puede ser menor que la heurística del nodo original. Así, toda heurística consistente es admisible pero no viceversa.

Al asegurar estos dos aspectos, entonces la heurística usada es relevante y proporciona información importante del sistema. Finalmente, y de manera general, así como para todo el resto de los algoritmos, se debe definir una heurística que sea eficiente computacionalmente, que se pueda aplicar a cualquier estado, que perturbaciones pequeñas en los estados no la afecten, y que sea lo más sencilla de implementar.

Ejercicio 2: N Queens

Backtracking: Es un proceso para resolver problemas recursivamente que consiste en generar todos los siguientes pasos que guían a una solución y parten de un mismo estado inicial para luego analizar recursivamente cada uno, encontrando los siguientes pasos de estos, y repitiendo el proceso hasta llegar a una solución o no. En caso de no llegar a una solución, se analiza el siguiente paso de la recursividad anterior, de modo que todos los casos son analizados utilizando el concepto de recursividad. Es importante que exista un patrón de análisis ya que este tipo de solución utilizan la recursividad como elemento base.

Para resolver este ejercicio se usó una abstracción del tablero de ajedrez en forma de lista en la que los índices de la lista representan las filas, y los valores almacenados en la lista son las columnas, de modo que para cada dato en la lista se puede obtener una coordenada que es la posición en la que se debe ubicar cada reina para tener una solución. Ya que se utilizan listas, los índices de las filas y columnas del tablero empiezan por 0 y terminan en N-1.

Soluciones para un tablero de 5x5

```
[0, 2, 4, 1, 3]
[0, 3, 1, 4, 2]
[1, 3, 0, 2, 4]
[1, 4, 2, 0, 3]
[2, 0, 3, 1, 4]
[2, 4, 1, 3, 0]
[3, 0, 2, 4, 1]
[3, 1, 4, 2, 0]
[4, 1, 3, 0, 2]
[4, 2, 0, 3, 1]
```

Soluciones totales: 10

Soluciones para un tablero de 8x8

[0, 4, 7, 5, 2, 6, 1, 3]	[2, 5, 1, 6, 0, 3, 7, 4]	[3, 5, 0, 4, 1, 7, 2, 6]
[0, 5, 7, 2, 6, 3, 1, 4]	[2, 5, 1, 6, 4, 0, 7, 3]	[3, 5, 7, 1, 6, 0, 2, 4]
[0, 6, 3, 5, 7, 1, 4, 2]	[2, 5, 3, 0, 7, 4, 6, 1]	[3, 5, 7, 2, 0, 6, 4, 1]
[0, 6, 4, 7, 1, 3, 5, 2]	[2, 5, 3, 1, 7, 4, 6, 0]	[3, 6, 0, 7, 4, 1, 5, 2]
[1, 3, 5, 7, 2, 0, 6, 4]	[2, 5, 7, 0, 3, 6, 4, 1]	[3, 6, 2, 7, 1, 4, 0, 5]
[1, 4, 6, 0, 2, 7, 5, 3]	[2, 5, 7, 0, 4, 6, 1, 3]	[3, 6, 4, 1, 5, 0, 2, 7]
[1, 4, 6, 3, 0, 7, 5, 2]	[2, 5, 7, 1, 3, 0, 6, 4]	[3, 6, 4, 2, 0, 5, 7, 1]
[1, 5, 0, 6, 3, 7, 2, 4]	[2, 6, 1, 7, 4, 0, 3, 5]	[3, 7, 0, 2, 5, 1, 6, 4]
[1, 5, 7, 2, 0, 3, 6, 4]	[2, 6, 1, 7, 5, 3, 0, 4]	[3, 7, 0, 4, 6, 1, 5, 2]
[1, 6, 2, 5, 7, 4, 0, 3]	[2, 7, 3, 6, 0, 5, 1, 4]	[3, 7, 4, 2, 0, 6, 1, 5]
[1, 6, 4, 7, 0, 3, 5, 2]	[3, 0, 4, 7, 1, 6, 2, 5]	[4, 0, 3, 5, 7, 1, 6, 2]
[1, 7, 5, 0, 2, 4, 6, 3]	[3, 0, 4, 7, 5, 2, 6, 1]	[4, 0, 7, 3, 1, 6, 2, 5]
[2, 0, 6, 4, 7, 1, 3, 5]	[3, 1, 4, 7, 5, 0, 2, 6]	[4, 0, 7, 5, 2, 6, 1, 3]
[2, 4, 1, 7, 0, 6, 3, 5]	[3, 1, 6, 2, 5, 7, 0, 4]	[4, 1, 3, 5, 7, 2, 0, 6]
[2, 4, 1, 7, 5, 3, 6, 0]	[3, 1, 6, 2, 5, 7, 4, 0]	[4, 1, 3, 6, 2, 7, 5, 0]
[2, 4, 6, 0, 3, 1, 7, 5]	[3, 1, 6, 4, 0, 7, 5, 2]	[4, 1, 5, 0, 6, 3, 7, 2]
[2, 4, 7, 3, 0, 6, 1, 5]	[3, 1, 7, 4, 6, 0, 2, 5]	[4, 1, 7, 0, 3, 6, 2, 5]
[2, 5, 1, 4, 7, 0, 6, 3]	[3, 1, 7, 5, 0, 2, 4, 6]	[4, 2, 0, 5, 7, 1, 3, 6]

[4, 2, 0, 6, 1, 7, 5, 3]	[5, 2, 0, 6, 4, 7, 1, 3]	[6, 0, 2, 7, 5, 3, 1, 4]
[4, 2, 7, 3, 6, 0, 5, 1]	[5, 2, 0, 7, 3, 1, 6, 4]	[6, 1, 3, 0, 7, 4, 2, 5]
[4, 6, 0, 2, 7, 5, 3, 1]	[5, 2, 0, 7, 4, 1, 3, 6]	[6, 1, 5, 2, 0, 3, 7, 4]
[4, 6, 0, 3, 1, 7, 5, 2]	[5, 2, 4, 6, 0, 3, 1, 7]	[6, 2, 0, 5, 7, 4, 1, 3]
[4, 6, 1, 3, 7, 0, 2, 5]	[5, 2, 4, 7, 0, 3, 1, 6]	[6, 2, 7, 1, 4, 0, 5, 3]
[4, 6, 1, 5, 2, 0, 3, 7]	[5, 2, 6, 1, 3, 7, 0, 4]	[6, 3, 1, 4, 7, 0, 2, 5]
[4, 6, 1, 5, 2, 0, 7, 3]	[5, 2, 6, 1, 7, 4, 0, 3]	[6, 3, 1, 7, 5, 0, 2, 4]
[4, 6, 3, 0, 2, 7, 5, 1]	[5, 2, 6, 3, 0, 7, 1, 4]	[6, 4, 2, 0, 5, 7, 1, 3]
[4, 7, 3, 0, 2, 5, 1, 6]	[5, 3, 0, 4, 7, 1, 6, 2]	[7, 1, 3, 0, 6, 4, 2, 5]
[4, 7, 3, 0, 6, 1, 5, 2]	[5, 3, 1, 7, 4, 6, 0, 2]	[7, 1, 4, 2, 0, 6, 3, 5]
[5, 0, 4, 1, 7, 2, 6, 3]	[5, 3, 6, 0, 2, 4, 1, 7]	[7, 2, 0, 5, 1, 4, 6, 3]
[5, 1, 6, 0, 2, 4, 7, 3]	[5, 3, 6, 0, 7, 1, 4, 2]	[7, 3, 0, 2, 5, 1, 6, 4]
[5, 1, 6, 0, 3, 7, 4, 2]	[5, 7, 1, 3, 0, 6, 4, 2]	

Soluciones totales: 92

Soluciones para un tablero de 11x11

Soluciones totales: 2680

Soluciones para un tablero de 27x27

Soluciones totales: No se generaron todas, pero según fuentes de Internet el valor obtenido debería ser de más de 234×10^{15} .

Ejercicio 3: Acertijo del granjero, lobo, cabra y col.

Al utilizar el código enviado, se obtiene la siguiente salida con la siguiente query. Se utiliza el predicado length definido por PROLOG para que el proceso de búsqueda se detenga después de 7 movimientos, y en caso de no incluirlo se obtiene una recursividad que se termina por el espacio de memoria ocupado, por lo que para aplicar esta solución se debe saber que con 7 pasos se puede alcanzar la solución.


```
 length(A,7),solution([w,w,w,w],A).
```

A = [goat, nothing, wolf, goat, cabbage, nothing, goat]

Cada parte de la lista indica lo que el granjero lleva consigo en el bote en cada viaje que realiza.

Ejercicio 4: Acertijo de los misioneros y los caníbales

Al utilizar el código enviado, se obtiene la siguiente salida con la siguiente query.

```
 consulta(A).
```

A =

[dosCanibalDr, unCanibalz, dosCanibalDr, unCanibalz, dosMisionerosDr,
unMisioneroCanibalz, dosMisionerosDr, unCanibalz, dosCanibalDr, unMisionerolz,
unMisioneroCanibalDr]

Este resultado se lee desde el final hasta el inicio, y cada componente de la lista muestra un viaje realizado.

Referencias

Russell, Stuart J. (Stuart Jonathan), Artificial intelligence:, Upper Saddle River: Prentice Hall, c2010.