Práctica 1:
"RESOLUCIÓN DE PROBLEMAS MEDIANTE BÚSQUEDA EN UN ESPACIO DE ESTADOS"

Sistemas Inteligentes, 3D1 GII –ETSINF-

Objetivo de la práctica

- *Evaluar la eficiencia, coste temporal y espacial de distintas estrategias de búsqueda aplicadas al juego del 8-puzzle.*

- Para ello se proporciona el programa puzzle1 implementado en Python.

Sesiones

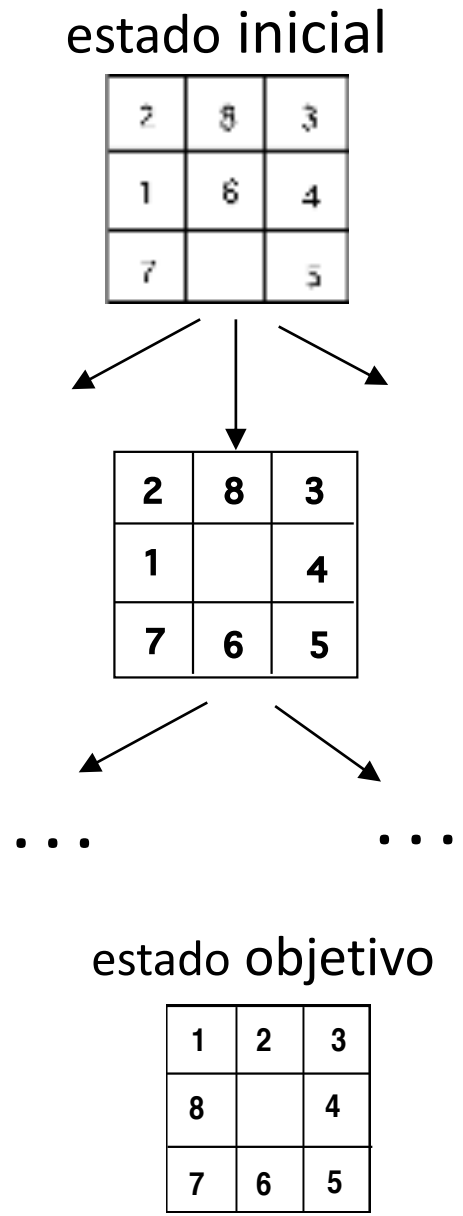
S1: 26/9
S2: 3/10
S3: 10/10
S4: 17/10
S5: 24/10 Examen Práctica

8-puzzle

- Tablero de 3*3 casillas.
- 8 de las casillas contienen una pieza o ficha que se puede deslizar a lo largo del tablero horizontal y verticalmente.
- Las fichas vienen marcadas con los números del 1 al 8.
- Hay una casilla libre en el tablero que permite los movimientos de las fichas.

8-puzzle

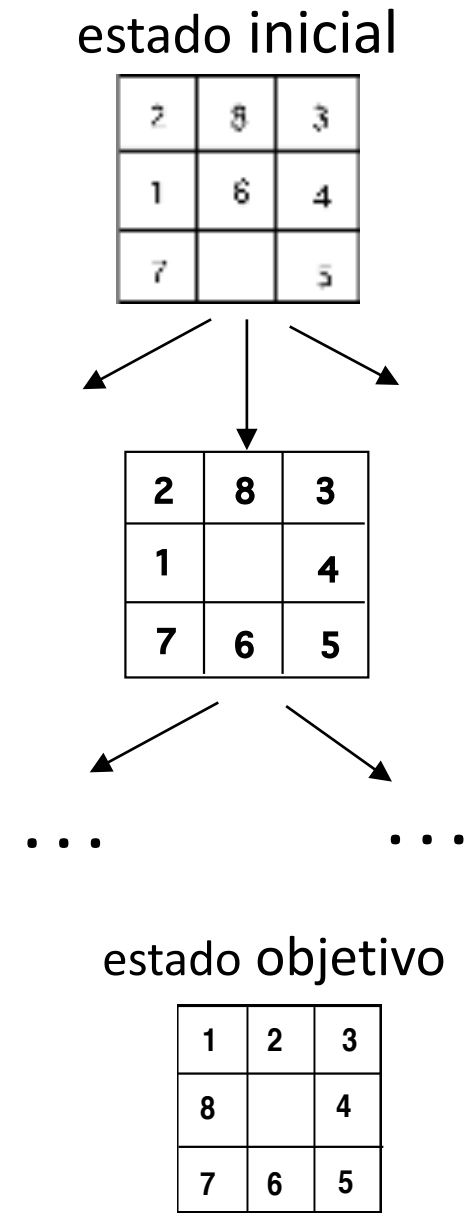
- Acciones: movimientos de las fichas (o movimientos de casilla libre) en 4 direcciones:
  - arriba, abajo, izquierda, derecha
- Función de coste de las acciones: **coste= 1** para todas las acciones del problema
- Solución: secuencia de movimientos de las fichas desde estado inicial a estado objetivo
- Coste del camino: **número de movimientos del camino**



8-puzzle

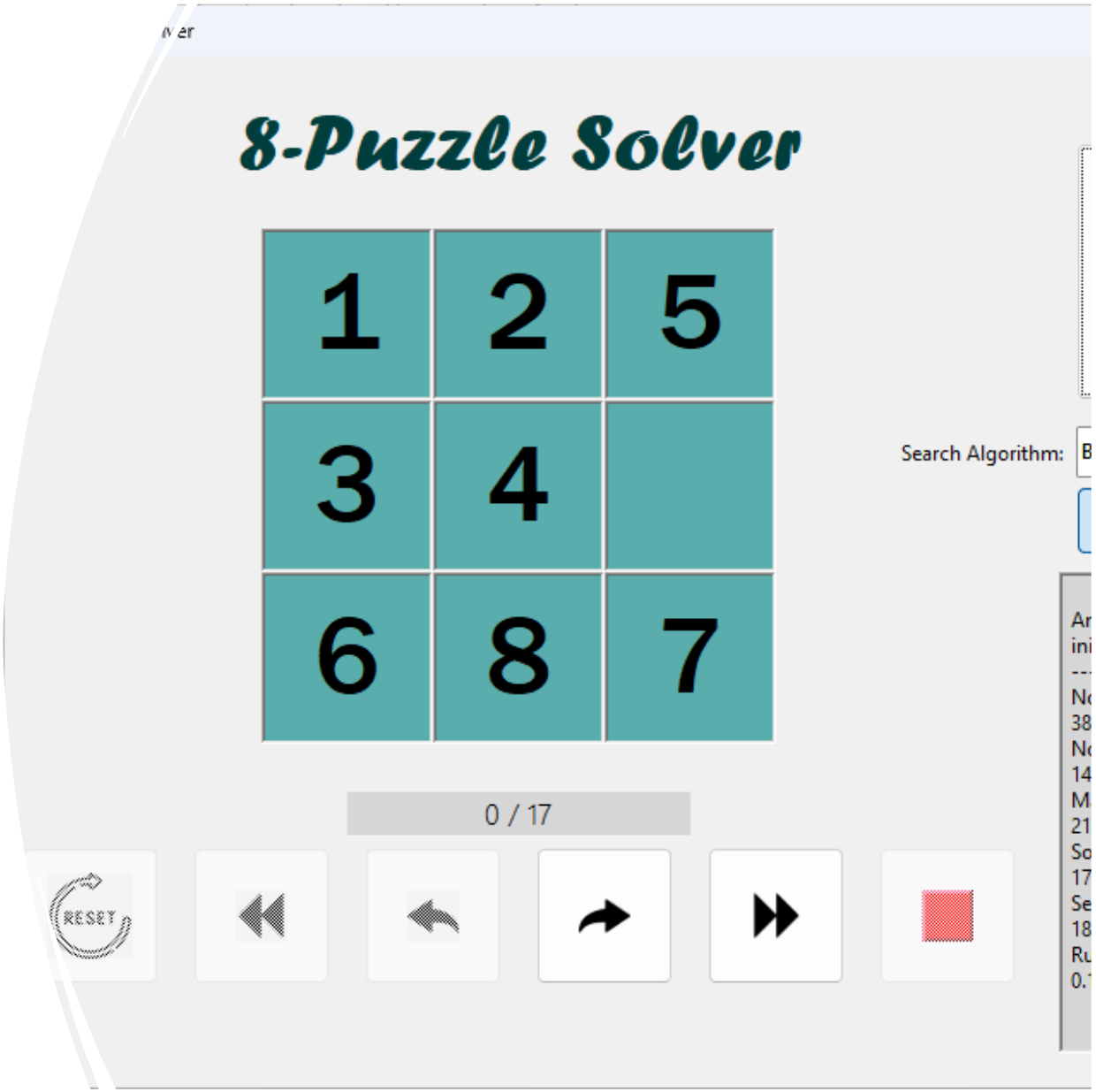
- Representación como lista de un estado:
  - {C1, C2, C3, …. C8}
  - La casilla libre se representa con un 0

- Estado objetivo: {1,2,3,8,0,4,7,6,5}



8-puzzle Solver

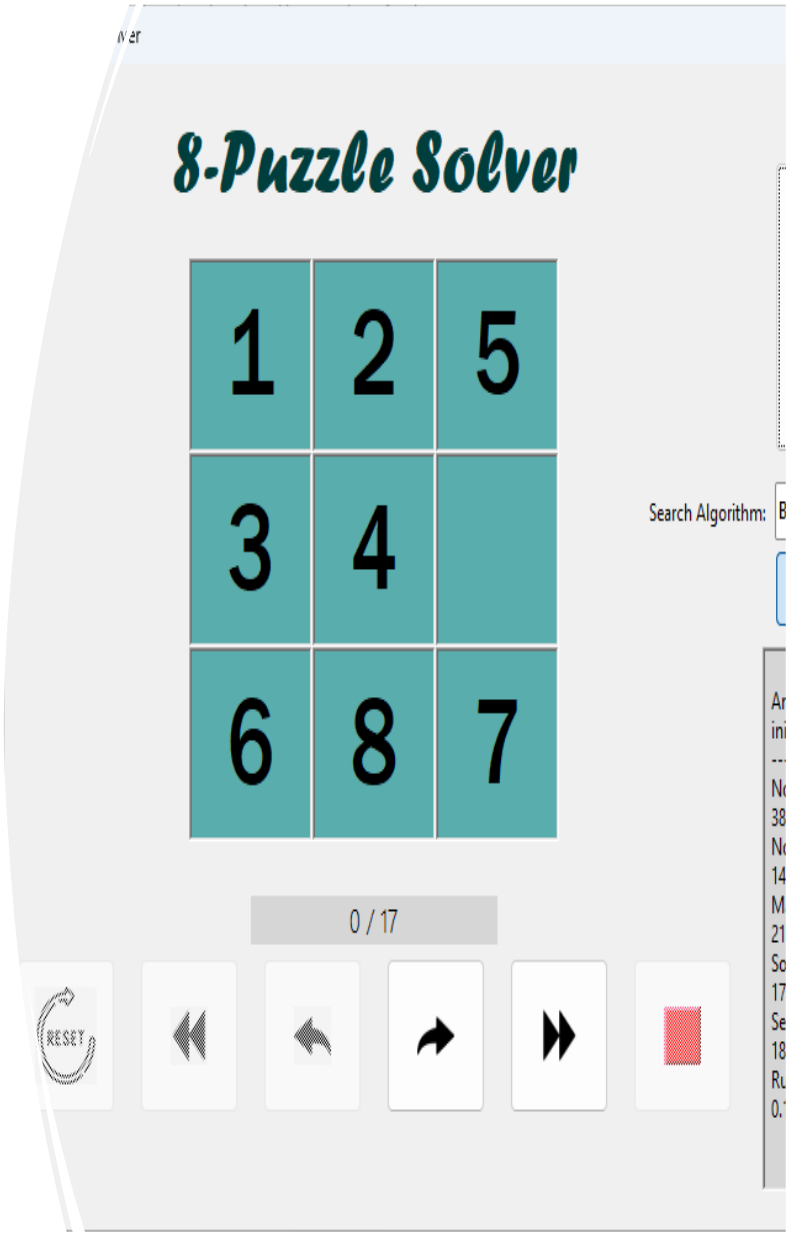
- Programa en Python que resuelve el 8-puzzle con distintas estrategias
- Se pueden programa nuevas estrategias
- Disponible en Poliformat



8-puzzle Solver

Instalación:

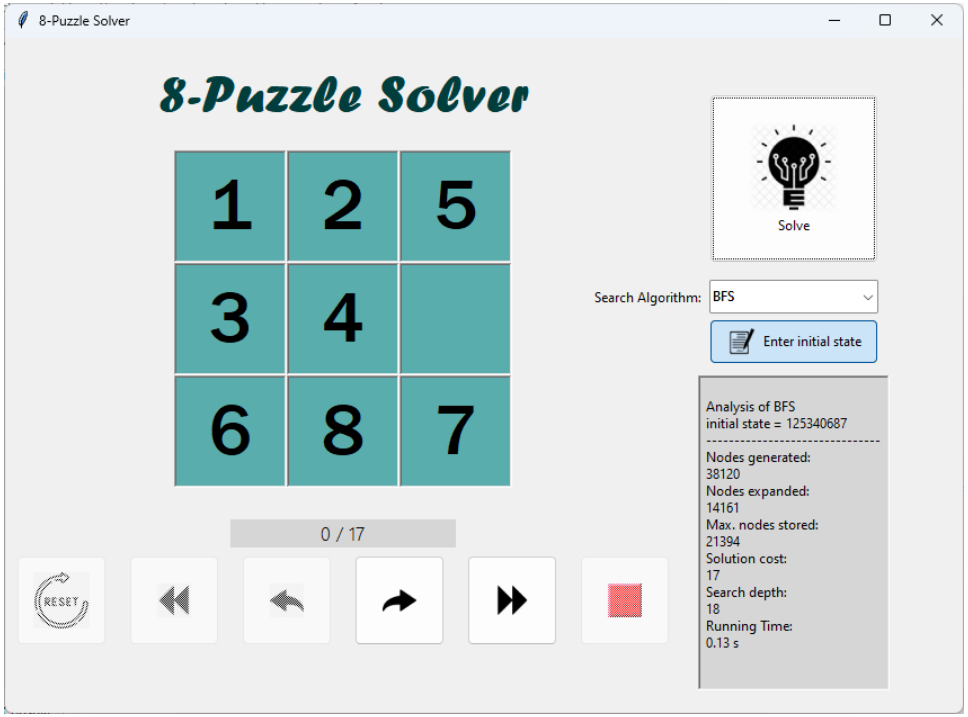
- Copiar el fichero **puzzle.zip** que está en PoliformaT
- Descomprimir el fichero**.** Al descomprimirlo se creará un directorio puzzle donde se ubican los ficheros fuente.
- Ejecución:
  - Desde un entorno de desarrollo de Python (por ejemplo **spyder**, disponible en los laboratorios)
  - Desde un terminal (si tiene Python instalado en el PATH) ejecuta el programa **interface.py**.



8-puzzle Solver

Pruebas:

- Pulsar en el botón '**Enter initial state'**
- Insertar el hecho inicial : El ejemplo sería 125340687
- Seleccionar la estrategia de resolución deseada en la pestaña desplegable '**Search Algorithm'**.
  - Si se selecciona una estrategia que requiere un nivel de corte de profundidad se abrirá una ventana donde se podrá introducir dicho valor.
- Pulsa el botón '**Solve'.**
  - Se mostrarán los datos de la ejecución del proceso de búsqueda realizado
- Utilizar los botones de debajo de la ventana del puzzle para recorrer el camino de la solución.



8-puzzle Solver (estrategias implementadas)

- **BFS.** Esta es la estrategia de **Anchura** que utiliza $f(n)=g(n)$ para la expansión de nodos, donde $g(n)$ es el factor coste asociado al nivel de profundidad del nodo n en el árbol de búsqueda.
- **DFS (Graph Search)**. Esta estrategia implementa una búsqueda en **Profundidad** GRAPH-SEARCH. Para ello, utiliza la función $f(n)=-g(n)$, donde $g(n)$ es el factor coste asociado al nivel de profundidad del nodo n en el árbol de búsqueda.
- **DFS (Bactracking)**. Esta estrategia implementa una búsqueda en **Profundidad** con backtracking o **Profundidad** TREE-SEARCH. Para ello, utiliza la función $f(n)=-g(n)$, donde $g(n)$ es el factor coste asociado al nivel de profundidad del nodo n en el árbol de búsqueda. Solo mantiene en memoria los nodos explorados que pertenecen al camino actual (lista PATH)
- **ID**. Esta estrategia implementa una búsqueda por **Profundización Iterativa** (Iterative Deepening). Se trata de realizar sucesivas búsquedas en profundidad con backtracking hasta que se alcanza la solución. Cada nueva búsqueda incrementa el nivel de profundidad de la exploración en 1.

8-puzzle Solver (estrategias implementadas)

- **Voraz (Manhattan).** Esta es una estrategia que implementa un Algoritmo voraz siguiendo la función $f(n)=D(n)$, donde $D(n)$ es la distancia de Manhattan.
- **A* Manhattan**. Esta es una búsqueda de tipo A que utiliza la distancia de Manhattan como heurística; la expansión de los nodos se realiza siguiendo la función $f(n)=g(n)+D(n)$, donde $h(n)=D(n)$ es la distancia de Manhattan.
- **A* Euclidean**: Esta es una búsqueda de tipo A que utiliza a distancia Euclídea como función heurística; la expansión de los nodos se realiza siguiendo la función $f(n)=g(n)+D(n)$, donde $h(n)=D(n)$ es la distancia Euclídea.
- **IDA* Manhattan.** Esta estrategia de búsqueda implementa una búsqueda IDA* donde el límite de la búsqueda viene determinado por el valor-f, es decir, por la función $f(n)=g(n)+D(n)$, donde $g(n)$ es el factor de coste y $h(n)=D(n)$ es la distancia de Manhattan. El límite de una iteración i del algoritmo IDA* es el valor-f más pequeño de cualquier nodo que haya excedido el límite en la iteración anterior $i-1$.
- Se pueden implementar más ….

Sesiones 2, 3 y 4

1º. Implementar tres nuevas funciones heurísticas

→ Modificar el código que os hemos pasado

2º. Comparar todas las heurísticas

¿Cómo crear heurísticas?

Fichero **interface.py**: parte de la interfaz y llamadas a los algoritmos de búsqueda

Fichero **main.py**: implementación de los algoritmos y estrategias de búsqueda

Fichero 'interface.py'

• Añadir el nombre de la nueva función de búsqueda para que aparezca en la lista desplegable de estrategias de búsqueda:

Función `def __init__(self, master=None):` **Líneas 88-89 (aprox.):**

```
self.algorithmbox.configure(cursor="hand2", state="readonly",
                            values=('BFS',    'DFS    (Graph    Search)',    'DFS
                           (Backtracking)',   'Voraz   (Manhattan)',   'ID',   'A*
                           Manhattan', 'A* Euclidean', 'IDA* Manhattan'))
```

¿Cómo crear heurísticas?

Fichero 'interface.py'

- Si necesita un nivel de profundidad máximo

Función `selectAlgorithm`, línea 248 (aprox):

```
if algorithm in ['DFS (Graph Search)', 'DFS (Backtracking)']:
 cutDepth = int(simpledialog.askstring('Cut depth', 'Please, enter your max
            depth'))
```

¿Cómo crear heurísticas?

Fichero 'interface.py'

- Realizar la llamada a la función de búsqueda. Ejemplo de A* (línea 381 aprox.)

```
elif str(algorithm) == 'A* Manhattan':
     main.graphSearch(initialState,main.function_1,main.getManhattanDistance)
     path, cost, counter, depth, runtime, nodes, max_stored,memory_rep = \
         main.graphf_path, main.graphf_cost, main.graphf_counter,
         main.graphf_depth,          main.time_graphf,          main.node_counter,
main.max_counter,
         main.max_rev_counter
```

`function_1`. Devuelve 1, coste habitual de un movimiento en el puzzle
`function_0`. Devuelve 0, se usa para la estrategia voraz
`function_N`. Devuelve -1. Usado en estrategias de profundidad

llamada a la función especifica que calcula **h(n)**

¿Cómo crear heurísticas?

Fichero 'main.py'

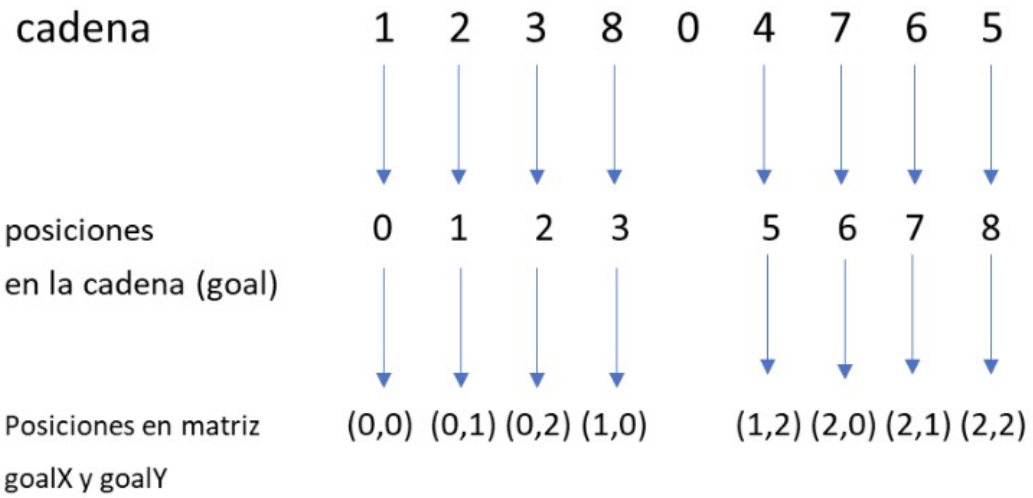
- Hay que incluir una función que implemente la nueva función heurística. Ejemplo "Distancias de Manhattan

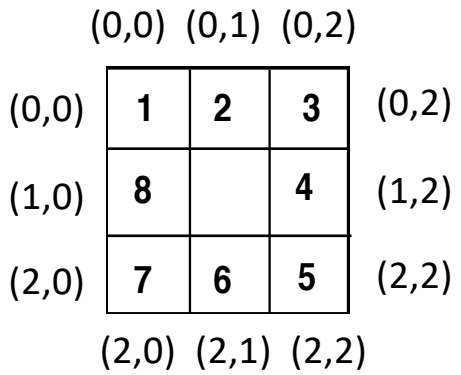
```
def getManhattanDistance(state):  #state es el estado actual a evaluar (en forma de cadena
texto)
  tot = 0
  for i in range(1,9):                    #recorre las 8 piezas
    goal = end_state.index(str(i))  #posición de la pieza i en el estado final (end_state)
    goalX = int(goal/ 3)            #Pasamos el end state de cadena a una matriz de 3x3
    goalY = goal % 3                #fila de la pieza (goalX) y columna (goalY)
    idx = state.index(str(i))       #misma operación pero para la fila (idx) y columna (idy)
   itemX = int(idx / 3)             #que ocupa esa pieza en el estado actual
    itemY = idx % 3
   tot += (abs(goalX - itemX) + abs(goalY - itemY)) #cálculo de distancias
 return tot
```

¿Cómo crear heurísticas?

Fichero 'main.py'

- Transformación de representación lineal (cadena) a matriz. Ejemplo estado objetivo.





Trabajo a realizar

Implementar tres funciones heurísticas y responder a una serie de preguntas.

- Entrega en tarea Poliformat el día del examen (fichero Python y texto de respuesta a las preguntas)

- Implementar la función heurística **PIEZAS DESCOLOCADAS** que se ha visto en clase de teoría.
- Implementar la función heurística **SECUENCIAS**
- Implementar la función heurística **FILAS_COLUMNAS**

Trabajo a realizar

Secuencias. $f(n)=g(n)+3*S(n)$, donde $g(n)$ es el factor coste asociado al nivel de profundidad del nodo n en el árbol de búsqueda y $h(n)=3*S(n)$.

$S(n)$ es la secuencia obtenida recorriendo sucesivamente las casillas del puzzle, excepto la casilla central, y anotando 2 puntos en la cuenta por cada ficha no seguida por su ficha sucesora y 0 puntos para las otras; si hay una ficha en el centro, se anota 1. Ejemplo:

2	8	3
1	6	4
7		5

Para la ficha 2:	Sumar 2 (porque no va seguida de su ficha sucesora 3)
Para la ficha 8:	Sumar 2 (porque no va seguida de su ficha sucesora 3)
Para la ficha 3:	Sumar 0 (porque sí va seguida de su ficha sucesora 4)
Para la ficha 4:	Sumar 0 (porque sí va seguida de su ficha sucesora 5)
Para la ficha 5:	Sumar 2 (porque no va seguida de su ficha sucesora 6)
Para la ficha 0:	Sumar 2 (se sumará 2 puntos siempre que no esté en la casilla central)
Para la ficha 7:	Sumar 2 (porque no va seguida de su ficha sucesora 8)
Para la ficha 1:	Sumar 0 (porque va seguida por su ficha sucesora 2)
Para la ficha 6:	Sumar 1 (porque es la ficha central).
TOTAL	**11 puntos**

Trabajo a realizar

Filas_Columnas. h(n)=FilCol(n).

Donde FilCol(n) tiene en cuenta lo siguiente: si la ficha no está en su fila destino correcta se suma 1 punto; si la ficha no está en su columna destino correcta se suma 1 punto. Por tanto, el mínimo valor de esta heurística para una ficha es 0 (cuando la ficha está colocada correctamente en su posición objetivo), y el máximo valor es 2 (cuando ni la fila ni la columna de la posición de la ficha son iguales a valor de la fila y columna de la posición objetivo).
Ejemplo:

2	8	3
1	6	4
7		5

La ficha 1 está en la columna correcta pero no en la fila: sumaría 1
La ficha 2 está en la fila correcta pero no en la fila: sumaría 1
Las fichas 3, 4 y 5 suman 0
La ficha 6 está en la columna correcta pero no en la fila: sumaría 1
La ficha 7 suma 0
La ficha 8 no está ni en la fila ni en la columna correcta: suma 2

Trabajo a realizar (resumen)

1. **Implementar** las tres funciones heurísticas (Descolocadas, Secuencias, y FilCol).
2. **Ejecutar** varias configuraciones del puzzle y anotar los resultados de todas las estrategias de búsqueda que se muestran en el programa, además de las tres nuevas funciones heurísticas implementadas.
   - A modo de ejemplo, se puede ejecutar las configuraciones propuestas (se recomienda ejecutar más configuraciones para analizar detalladamente el comportamiento de las estrategias)
   - **Rellenar** tablas

	BFS	DFS (GS)	DFS (Back)	Voraz	ID	A* Manh	A* Euclid	IDA* Manh	Desc	Sec	FilCol
Nodes generated											
Nodes expanded											
Max nodes stored											
Solution cost											
Search depth											
Running time											

3. **Contestar** de forma razonada las preguntas del boletín.

Trabajo para hoy (y ya sirve para la entrega)

- **Ejecutar dos configuraciones con los algoritmos de búsqueda no informados y rellenar las tablas con los resultados en el boletín**

	2	3
8	4	5
7	1	6

7	8	1
4		6
2	3	5

Trabajo para hoy (y ya sirve para la entrega)

- **Ejecutar dos configuraciones con los algoritmos de búsqueda (salvo los 3 últimos) y rellenar las tablas con los resultados en el boletín**

	BFS	DFS (GS)	DFS (Back)	Voraz	ID	A* Manh	A* Euclid	IDA* Manh	Desc	Sec	FilCol
Nodes generated											
Nodes expanded											
Max nodes stored											
Solution cost											
Search depth											
Running time											