INTELIGENCIA ARTIFICIAL
Puzzle 3 x 3 con 3 Heurísticas

El juego del puzzle se representa, sobre un tablero de 3x3 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, y el 0 representa a la casilla vacía, que permite los movimientos de las fichas. El objetivo del problema es partiendo de un tablero inicial con las fichas desordenadas alcanzar un tablero en el que todas las fichas estén ordenadas en orden creciente, dejando el hueco en la primer casilla del tablero. El número total de estados o tableros posibles que se pueden generar en el juego del puzzle es 9! = 362,880 estados.

Tu labor es resolver este problema empleando 3 Heurísticas, **Piezas fuera de lugar**, **Suma de distancias fuera de lugar** y **2 por número de regresos directos.**

El estado al que se desea llegar para cualquier puzzle dado siempre será:

0	1	2
3	4	5
6	7	8

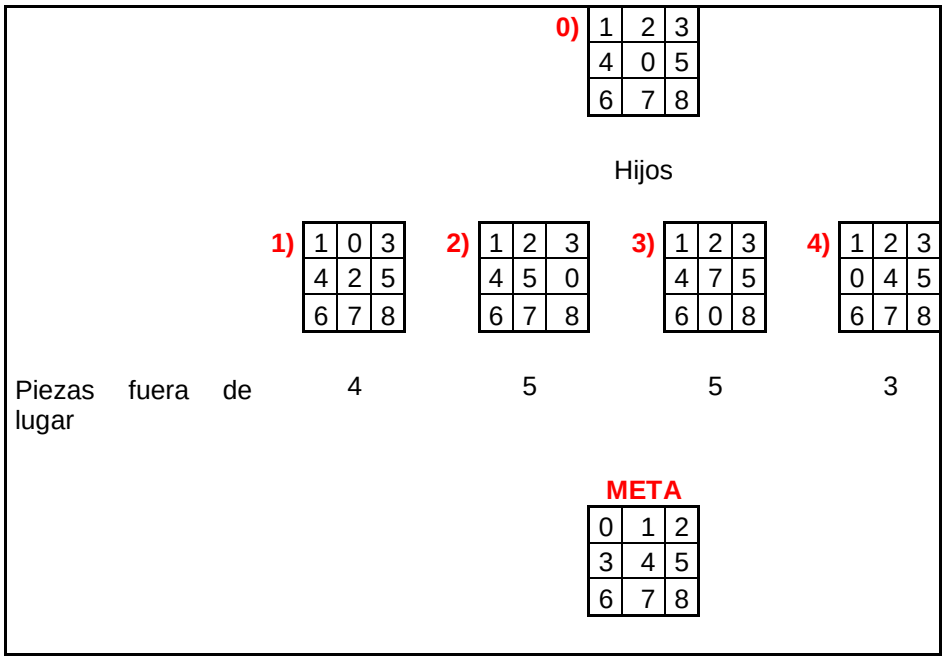
A continuación se describe como debe de funcionar cada heurística.

Primera opción del menú: Piezas fuera de lugar

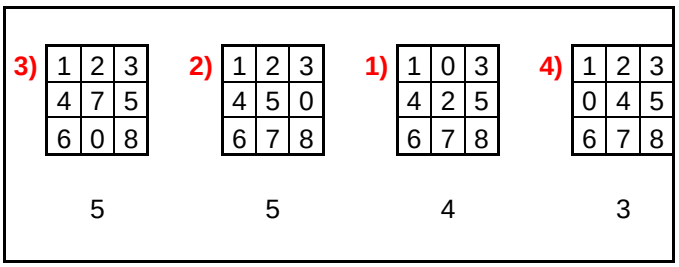
Dado un puzzle inicial, lo insertarás en un pila de estados, si dicho estado aún no es la solución, lo sacarás del tope de la pila, pero debes tener en cuenta que no se vale volver a meter a dicho estado en la pila (si es que eventualmente se vuelve a generar), es decir no debes meter estados en la pila que alguna vez ya insertaste, una vez sacado dicho estado del tope de la pila, deberás generar los hijos de este, considerando sólo a aquellos nunca han sido insertados en la pila, y deberás contar el número **g(n)** de **piezas fuera de lugar** de cada estado, una vez que tengas estos valores, los ordenaras de mayor a menor (por el número de piezas fuera de lugar) y en ese orden los insertarás en la pila, es decir, primero insertarás el que tenga el mayor número de piezas fuera de lugar, luego el segundo y así sucesivamente, en caso de empates, insertarás primero el de mayor orden lexicográfico.

El proceso continuará hasta que se llegue a la solución.

Ejemplo:



Como puedes observar, en este ejemplo hay empate según el número de piezas fuera de lugar (entre **2)** y **3)**), por lo que deberás ordenarlos e insertarlos en la pila de la siguiente forma:



Ya que el **3)** tiene menor orden lexicográfico que el **2)** (recuerda que lo puedes observar mejor si ves a las matrices como vectores), y por ende primero se inserta en la pila el **3)**, luego el **2)**, después el **1)** y finalmente el **4)**, que al quedar en el tope de la pila, será el siguiente estado a evaluar (ya que como se puede ver, este no es la solución), y nuevamente el proceso se repite.

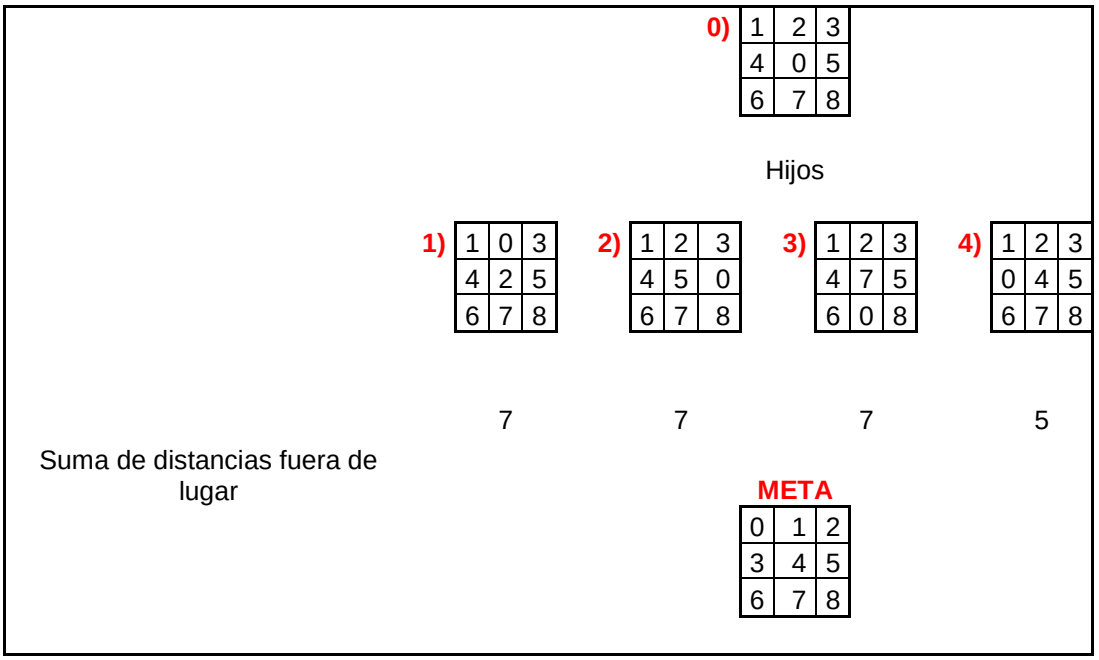
Nota: Nótese que el número 0 (blanco), no cuenta como una pieza fuera de lugar.

Segunda opción del menú: Suma de distancias fuera de lugar

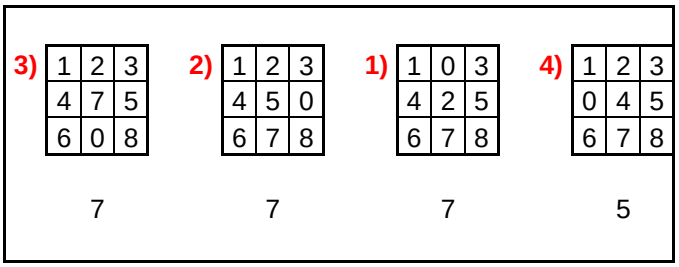
Dado un puzzle inicial, lo insertarás en una pila de estados, si dicho estado aún no es la solución, lo sacarás del tope de la pila, pero debes tener en cuenta que no se vale volver a meter a dicho estado en la pila (si es que eventualmente se vuelve a generar), es decir no debes meter estados en la pila que alguna vez ya insertaste, una vez sacado dicho estado del tope de la pila, deberás generar los hijos de este, considerando sólo a aquellos nunca han sido insertados en la pila, y deberás obtener la **sumatoria h(n) de las distancias de las piezas fuera de lugar** (es decir, cuantas casillas se aleja una pieza, de donde debería encontrarse), una vez que tengas estos valores, los ordenaras de mayor a menor (o sea la suma calculada) y en ese orden los insertarás en la pila, es decir, primero insertarás el que tenga la mayor sumatoria, luego el segundo y así sucesivamente, en caso de empates, insertarás primero el de mayor orden lexicográfico.

El proceso continuará hasta que se llegue a la solución.

Ejemplo:



Como puedes observar, en este ejemplo hay triple empate según la sumatoria de distancias de las piezas fuera de lugar (entre **1)**, **2)** y **3)**), y al ordenarlos e insertarlos en la pila como se te pide quedarían así:



Ya que el **3)** tiene menor orden lexicográfico que el **2)** (recuerda que lo puedes observar mejor si ves a las matrices como vectores), y por ende primero se inserta en la pila el **3)**, luego el **2)**, después el **1)** y finalmente el **4)**, que al quedar en el tope de la pila, será el siguiente estado a evaluar (ya que como se puede ver, este no es la solución), y nuevamente el proceso se repite.

Tercera opción del menú: 2 por número de regresos directos.

Este algoritmo involucrará a las dos primeras heurísticas y además el número de regresos directos.

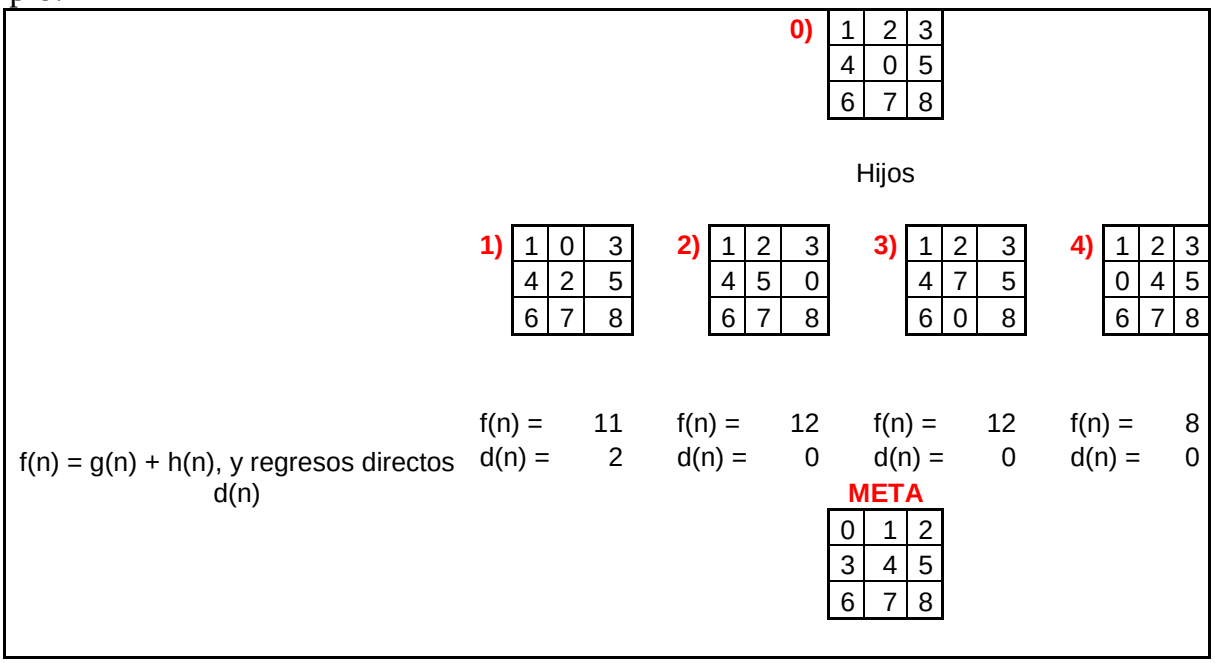
Dado un puzzle inicial, lo insertarás en una pila de estados, si dicho estado aún no es la solución, lo sacarás del tope de la pila, pero debes tener en cuenta que no se vale volver a meter a dicho estado en la pila (si es que eventualmente se vuelve a generar), es decir no debes meter estados en la pila que alguna vez ya insertaste, una vez sacado dicho estado del tope de la pila, deberás generar los hijos de este, considerando sólo a aquellos nunca han sido insertados en la pila, y deberás obtener la **suma f(n)** que es igual a la suma entre **g(n)** **(numero de piezas fuera de lugar)** + **h(n)** (**sumatoria de las distancias de las piezas fuera de lugar)**, una vez que tengas el valor **f(n)**, deberás calcular el número de regresos directos **d(n)** entre el puzzle generado y el destino.

Solo para este caso, al 0 (vacio) también lo debes de considerar para los intercambios directos.

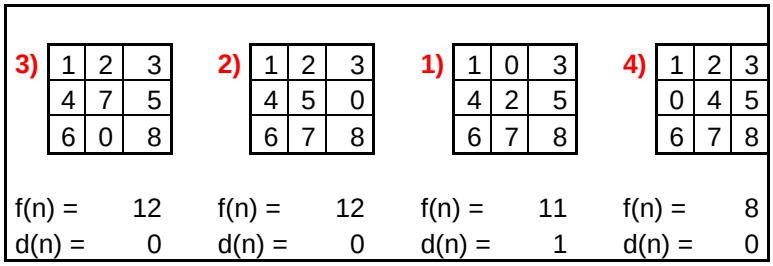
Un regreso directo cuenta sí y solo sí, basta con intercambiar dos fichas adyacentes horizontal o verticalmente, para que ambas quedasen en su lugar. Si al intercambiar dos fichas adyacentes horizontal o verticalmente solo una queda en su lugar, esto no cuenta como intercambio directo. El número de intercambios directos se multiplica por dos y ese es el valor de d(n).

Una vez que tengas **f(n)** y **d(n),** ordenarás a dichos hijos de la siguiente manera: ordenarás de mayor a menor respecto a f(n) y en ese orden los insertarás en la pila, es decir, primero insertarás el que tenga la mayor sumatoria, luego el segundo y así sucesivamente, en caso de empates, insertarás primero el que tenga mas pequeño **d(n)**, es decir si uno o más hijos empatan respecto de **f(n)**, el primero en ingresar en la pila de esos que empatan será el **de menor valor de d(n),** y si aún así hubiese empates (es decir tienen el mismo f(n) y d(n)), el primero en entrar a la pila sería el de mayor orden lexicográfico.

Ejemplo:



Como puedes observar, en este ejemplo hay doble empate según la función f(n) (entre **2)** y **3)**), y a su vez estos empatan en cuanto a d(n) (si no empataran en d(n), primero iría el de menor valor de d(n)), por lo que el único criterio de desempate que queda es el lexicográfico, de los cuales el puzzle **3)** es de mayor orden, por lo que al ordenarlos bajo estos criterios, e insertarlos en la pila como se te pide quedarían así:



Por lo tanto el siguiente estado al tope de la pila es el **4)** el cual no es la solución y por lo que ahora de éste, hay que evaluar sus descendientes. Nuevamente el proceso se repite hasta encontrar la solución.

Dado que estamos ocupando un tanto la idea de Hill Climbing (ya que insertamos en una pila a todos los descendientes cada vez), siempre se deberá encontrar la solución, por lo que no hay ningún tipo de salida especial en caso de no encontrar la solución, ya que esto no puede suceder.

Tu labor será mostrar en un archivo llamado puzzle3x3heuristicas.txt todo lo realizado por tu algoritmo según lo que se te indique en las especificaciones de salida.

Tu programa fuente deberá llamarse puzzle3x3heuristicas.(pas, for, c, cpp, java, etc).

Especificaciones de la entrada

Para este caso la entrada es más compleja.

Primero deberás leer un entero op tal que 1 <= op <= 5, donde cada valor de op significa lo siguiente:

1. Indica que debes leer una nueva matriz inicial de 3 x 3, por lo que seguido del 1 siempre deberás poder leer 9 enteros que indican la nueva matriz inicial.
2. Indica que debes aplicarle la heurística de "**piezas fuera de lugar**" a la matriz capturada.
3. Indica que debes aplicarle la heurística de "**suma de distancias fuera de lugar**" a la matriz capturada.
4. Indica que debe aplicarle que debes aplicarle la heurística de "**2 por numero de regresos directos**" a la matriz capturada.
5. Salir del programa. Puedes suponer que esta opción siempre te será dada.

Todos los valores dados pueden estar separados por uno o más espacios (incluyendo saltos de línea o tabulaciones), es decir, la entrada podría estar de las siguientes maneras:

Ejemplo 1:
```
1
1 2 3
4 0 5
6 7 8
2
3
4
5
```

Ejemplo 2:
```
1 1          2     3     4     0 5 6 7 8
2 3 4 5
```

Ejemplo 3:
```
1 1 2 3 4 0 5 6 7 8
1 0 1 2 3 4 5 6 8 7
5
```

Ejemplo 4:
```
2 3 4 5
```

Especificaciones de la salida (puzzle3x3heuristicas.txt)

El programa deberá generar un archivo denominado "puzzle3x3heuristicas.txt" (sin las comillas), el cual, dependiendo de la opción leída, contendrá lo siguiente:

Si la opción capturada es 1, no deberá escribirse en el archivo de salida, es decir se mantiene igual.

Si la opción capturada es 2, 3 o 4, y antes no se había leído una matriz inicial, deberás escribir en el archivo un -1 seguido de dos saltos de línea, que indica que no se puede ejecutar la heurística. En caso de que previamente si se haya leído una matriz inicial, deberás escribir en el archivo de salida el titulo de la heurística (PIEZAS, SUMAS o REGRESOS, tal y como esta aquí escrito, y según sea el caso), seguido de dos saltos de línea, y todos los pasos por los que pasa dicha heurística hasta llegar a la solución (finalizando cada vez con un salto de línea), así como la cuenta de los mismos (igualmente finalizando con un salto de línea).

Si la opción capturada es 5, deberás finalizar el programa y cerrar el archivo de salida.

Nótese que dependiendo de las opciones dadas el archivo de salida podría ser muy grande o incluso vacio, pero siempre debe de existir.

Obviamente al inicio del programa el archivo siempre debe estar vacio.

Cada programa se probará con 10 casos de entrada (como los mostrados en los ejemplos de entrada 1, 2 y 3), cada salida correcta* vale 1 punto.

Fecha de entrega: **Miércoles 11 de noviembre de 2009**.

No se reciben programas fuera de esta fecha, más que para derecho a exámen, pero ya sin derecho a calificación en el programa.

Modo de entrega: enviar un correo con el código fuente puzzle3x3heuristicas.(pas, for, c, cpp, java, etc), al mail **sergio10barca@gmail.com**.

Ejemplo de entrada 1

```
2
3
4
2
1
1 2 0
3 4 5
6 7 8
2
3
4
5
```

Ejemplo de salida 1 (puzzle3x3heuristicas.txt)

```
-1

-1

-1

-1

PIEZAS

1 2 0
3 4 5
6 7 8

1 0 2
3 4 5
```

```
6 7 8

0 1 2
3 4 5
6 7 8

3

SUMAS

1 2 0
3 4 5
6 7 8

1 0 2
3 4 5
6 7 8

0 1 2
3 4 5
6 7 8

3

REGRESOS

1 2 0
3 4 5
6 7 8

1 0 2
3 4 5
6 7 8

0 1 2
3 4 5
6 7 8

3
```

Ejemplo de entrada 2

```
1
1 2 5
3 4 8
6 7 0
2
2
3
4
5
```

Ejemplo de salida 2 (puzzle3x3heuristicas.txt)

```
PIEZAS

1 2 5
3 4 8
6 7 0

1 2 5
3 4 0
6 7 8

1 2 0
3 4 5
6 7 8

1 0 2
3 4 5
6 7 8

0 1 2
3 4 5
6 7 8

5
```

```
PIEZAS

1 2 5
3 4 8
6 7 0

1 2 5
3 4 0
6 7 8

1 2 0
3 4 5
6 7 8

1 0 2
3 4 5
6 7 8

0 1 2
3 4 5
6 7 8

5

SUMAS

1 2 5
3 4 8
6 7 0

1 2 5
3 4 0
6 7 8

1 2 0
3 4 5
6 7 8

1 0 2
3 4 5
6 7 8

0 1 2
3 4 5
6 7 8

5

REGRESOS

1 2 5
3 4 8
6 7 0

1 2 5
3 4 0
6 7 8

1 2 0
3 4 5
6 7 8

1 0 2
3 4 5
6 7 8

0 1 2
3 4 5
6 7 8

5
```

Ejemplo de entrada 3

```
1 1 2 5 3 4 8 6 0 7
2 4
1 1 2 3 4 0 5 6 7 8
4
```

```
1 1 2 3 4 5 6 7 8 0
5
```

Ejemplo de salida 3 (puzzle3x3heuristicas.txt)

```
PIEZAS

1 2 5
3 4 8
6 0 7

1 2 5
3 4 8
6 7 0

1 2 5
3 4 0
6 7 8

1 2 0
3 4 5
6 7 8

1 0 2
3 4 5
6 7 8

0 1 2
3 4 5
6 7 8

6

REGRESOS

1 2 5
3 4 8
6 0 7

1 2 5
3 4 8
6 7 0

1 2 5
3 4 0
6 7 8

1 2 0
3 4 5
6 7 8

1 0 2
3 4 5
6 7 8

0 1 2
3 4 5
6 7 8

6

REGRESOS

1 2 3
4 0 5
6 7 8

1 2 3
0 4 5
6 7 8

0 2 3
1 4 5
6 7 8

2 0 3
1 4 5
6 7 8
```

```
2 3 0
1 4 5
6 7 8

2 3 5
1 4 0
6 7 8

2 3 5
1 0 4
6 7 8

2 0 5
1 3 4
6 7 8

0 2 5
1 3 4
6 7 8

1 2 5
0 3 4
6 7 8

1 2 5
3 0 4
6 7 8

1 2 5
3 4 0
6 7 8

1 2 0
3 4 5
6 7 8

1 0 2
3 4 5
6 7 8

0 1 2
3 4 5
6 7 8

15
```

* **NOTA:** Se considera salida correcta aquella que es igual en su totalidad a la salida que el evaluador genera, en caso de imprimir valores o caracteres demás, o los datos incompletos, todo el caso de prueba se considerará como incorrecto y tendrás 0 puntos para ese caso. Además para cada caso de prueba tu programa no debe tardar más de 1 minuto en arrojar la respuesta.

ANA LILIA C. LAUREANO CRUCES, SERGIO LUIS PÉREZ PÉREZ,
UAM AZCAPOTZALCO, 2009