

Problema del 8-puzzle

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

Goal State

Formalización del 8-puzzle

- **Estados:** Tableros o puzzles generados al intercambiar la ficha vacía por otra ficha
- **Estado inicial:** cualquier configuración de fichas
- **Test objetivo:** comprobar si la disposición de las fichas de un estado coincide con el estado final
- **Sucesores:**
 - **esValido:** Si hay una ficha adyacente a la ficha vacía para poder realizar el intercambio Arriba, Abajo, a la Izquierda o a la Derecha, cada movimiento tiene unas reglas específicas
 - **aplicaOperador:** Intercambiar la ficha vacía con otra ficha que se encuentre en una posición adyacente válida. Existen 4 operadores.
- **Solución:** Movimientos para llegar desde estado inicial al estado objetivo
- **Coste del camino:** coste de cada paso

Estados del 8 puzle

1	2	3
4	5	6
	8	7

Matriz que representa el tablero

celdas

	1	2	3	4	5	6	7	8	9
fila	1	1	1	2	2	2	3	3	3

	1	2	3	4	5	6	7	8	9
col	1	2	3	1	2	3	3	2	1

Vectores que guardan las posiciones de cada ficha:
(en principio sólo se necesita guardar la posición del hueco, pero para codificar funciones heurísticas en el próximo tema será útil conocer la posición de cada ficha)

Operadores

- Cada nombre de operador tiene asignado un número constante:

Algoritmo 8-Puzle

const

N = 3 // Tablero 3x3

ARRIBA = 1

ABAJO = 2

IZQUIERDA = 3

DERECHA = 4

NUM_OPERADORES=4

Tipo de datos del estado

- Se define un tipo de datos específico para representar el estado

tipo

matriz[N,N] de entero: mat

vector[N*N] de entero: vect

registro: tEstado

mat: celdas

vect: fila, col

fin_registro

Mantendremos siempre una estructura tEstado, pero su contenido cambiará de acuerdo al problema concreto

Así podremos reutilizar el código de búsqueda con independencia del problema

Tipo de datos del estado

■ Matrices para construir los estados Inicial y Final

```
puzle_inicial[N][N] =  
{  
  {1,2,3},  
  {0,4,5},  
  {7,8,6}  
}
```

```
puzle_final[N][N] =  
{  
  {1,2,3},  
  {8,0,4},  
  {7,6,5}  
}
```

Función crearEstado

tEstado: **función** crearEstado(E mat: M)

var

tEstado: estado

entero: i, j, ficha

inicio

desde i \leftarrow 1 **hasta** N **hacer**

desde j \leftarrow 1 **hasta** N **hacer**

ficha \leftarrow M[i][j]

estado.celdas[i][j] \leftarrow ficha

estado.fila[ficha] \leftarrow i

estado.col[ficha] \leftarrow j

fin_desde

fin_desde

devolver estado

fin_función

Función testObjetivo

lógico: **función** testObjetivo(E tEstado: estado)

Inicio

tEstado: estadoFinal

estadoFinal ← crearEstado(puzle_final)

devolver iguales(estados, final)

fin_función

- Iguales es una función que debe ser también codificada

Función testObjetivo

lógico: **función** iguales(E tEstado estado, E tEstado: final)

var

entero: i, j

lógico: iguales

inicio

iguales \leftarrow verdadero

i \leftarrow 1

mientras (iguales) \wedge (i \leq N) **hacer**

j \leftarrow 1

mientras (iguales) \wedge (j \leq N) **hacer**

iguales \leftarrow (estado.celdas[i][j] = final.celdas[i][j])

j \leftarrow j+1

fin_mientras

i \leftarrow i+1

fin_mientras

devolver iguales

fin_función

Función esValido

lógico: **función** esValido(E entero: op, E tEstado e)

var

lógico: valido

entero: NCel

Inicio

$N_{cel} \leftarrow N * N$

según_sea (op) **hacer**

ARRIBA: $\text{valido} \leftarrow e.\text{fila}[N_{cel}] > 1$

ABAJO: $\text{valido} \leftarrow e.\text{fila}[N_{cel}] < N$

IZQDA: $\text{valido} \leftarrow e.\text{col}[N_{cel}] > 1$

DECHA: $\text{valido} \leftarrow e.\text{col}[N_{cel}] < N$

en_caso_contrario: $\text{valido} \leftarrow \text{falso}$

fin_según

devolver valido

fin_función

Función aplicaOperador

tEstado **función** aplicaOperador(E entero: op, E tEstado: estado)

var

tEstado: nuevo

entero: ficha, Ncel, fOld, fNew, cOld, cNew

Inicio

NCel \leftarrow N*N

nuevo \leftarrow estado // SIEMPRE se crea una nueva copia del estado

según_sea (op) **hacer**

ARRIBA: nuevo.fila[Ncel] \leftarrow nuevo.fila[NCel]-1

ABAJO: nuevo.fila[Ncel] \leftarrow nuevo.fila[NCel]+1

IZQDA: nuevo.col[Ncel] \leftarrow nuevo.col[NCel]-1

DECHA: nuevo.col[Ncel] \leftarrow nuevo.col[NCel]+1

fin_según

Función aplicaOperador

```
// Intercambio de fichas una vez fijada la nueva posición del hueco
// fila y columna nueva del hueco están ya en nuevo.fila[NCel] y en
// en nuevo.col[NCel], pero en el tablero no está actualizado, aún
// está la ficha:
```

```
fOld←estado.fila[Ncel]   fNew←nuevo.fila[Ncel]
```

```
cOld←estado.col[Ncel]   cNew←nuevo.col[Ncel]
```

```
ficha ← nuevo.celdas[fNew][cNew]
```

```
//se mueve el hueco en el tablero
```

```
nuevo.celdas[fNew][cNew] ← NCel
```

```
//la ficha pasa a la antigua posición del hueco
```

```
nuevo.celdas[fOld][cOld] ← ficha
```

```
//Nuevas posiciones para ficha
```

```
nuevo.col[ficha] ← cOld
```

```
nuevo.fila[ficha] ← fOld
```

```
devolver nuevo
```

```
fin_función
```