

## Proyecto 2: Resolviendo el 8-*Puzzle* con A\*

### 1. Objetivo

El objetivo del proyecto es el de resolver el problema del 8-*Puzzle* con un algoritmo informado de búsqueda de caminos de costo mínimo, específicamente A\*

### 2. Descripción del problema

El problema de 8-*Puzzle* [7], como lo describe Russel y Norvig [5] consiste, como se muestra la figura 1, en tablero de 3 X 3, el cual contiene 8 bloques cuadrados numerados (del 1 al 8) y un cuadrado vacío (En nuestra representación será numerado con 0). Los cuadrados adyacentes al espacio vacío pueden moverse a ese espacio. El objetivo es alcanzar un estado meta, el cual se puede observar en la tabla derecha de la figura 1. Una formulación posible del problema es la siguiente:

- **Estados:** Un estado indica la posición de cada uno de los nueve cuadrados dentro del tablero
- **Estado Inicial:** Un estado cualquiera
- **Función para obtener sucesores:** A partir de un estado se generan los estados válidos que resultan de las cuatro posibles acciones. Las acciones posibles son: mover el cuadrado a la izquierda, derecha, arriba y abajo
- **Costo del camino:** Cada movimiento o paso, tiene un costo de uno (1), por lo tanto el camino desde un estado inicial a un estado meta es igual al número de pasos (o movimientos) que se dan para alcanzar el estado meta.
- **Función de prueba de meta:** Determina si un estado es el estado meta.

En la figura 2, cortesía de Berman y Paul [1], se puede observar la arborescencia obtenida al resolver el 8-*Puzzle* con el algoritmo de Búsqueda en amplitud. Nuestro objetivo es encontrar el camino más corto, el cual en la figura 2 se muestra resaltado. Note que el *grafo para este problema es implícito*. Es decir, dado un estado inicial se obtienen los estados sucesores con una función. No se guarda todo el grafo, que es todas las posibles combinaciones del tablero, en memoria.

8	1	3
4		2
7	6	5

1	2	3
4	5	6
7	8	

Figura 1: Ejemplo de **Estado Inicial** y **Estado Final** del 8-*Puzzle*

### 3. Algoritmos informados de búsqueda de caminos mínimos

Se desea que resolver el 8-*Puzzle* con el algoritmos informado conocido como A\* [6]. El algoritmo A\* esta descrito en el libro de Meza y Ortega [3]. Tambien se encuentra una buena explicación del algoritmo A\* y del 8-*Puzzle* en libro de Rusell y Norvig [5]. En el libro *Artificial Intelligence for Games* [2] hay un pseudo código muy ilustrativo del algoritmo A\*, mostrando sus semejanzas con el algoritmo de Dijkstra y se hace un análisis sobre como implementarlo. Casi todas estas referencias se encuentran en la biblioteca de la universidad, en la sala de lectura hay algunas. También se publicará referencias en la página web del curso en la sección de *Bibliografía*.

Se debe implementar cuatro heurísticas o costos estimados a la meta  $h(\cdot)$ .

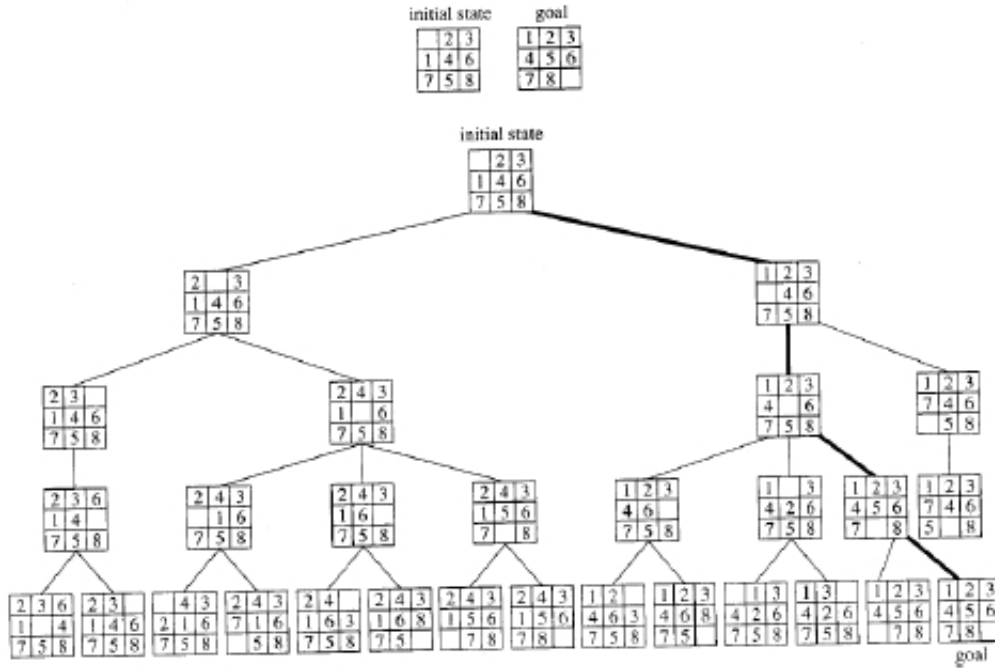


Figura 2: Arborescencia del 8-Puzzle obtenida con BFS

1. **Zero heurística:** En esta heurística todo estado da como resultado zero. Es decir, para cualquier estado  $n$ , se tiene que  $h(n) = 0$
2. **Distancia Manhattan:** Es la suma de las distancias, vérticales y horizontales, de cada uno de los bloques, a su posición en el estado meta.
3. **Número de cuadros desordenados:** Es el número de bloques en posiciones equivocadas, es decir, diferentes a las que corresponden al estado meta. Se puede interpretar que mientras más desordenado esté el tablero, más difícil debe ser ordenarlo.
4. **Blanco:** Distancia entre la posición en el estado meta del blanco y su posición actual.

Por ejemplo, dado los estados inicial y final de la figura 1, se tiene que el cálculo de las heurísticas es como sigue:

- **Zero heurística:** 0

- **Distancia Manhattan:**

Cuadro	1	2	3	4	5	6	7	8
Distancia	1	2	0	0	2	2	0	3

$$\text{total} = 1 + 2 + 2 + 2 + 3 = 10$$

- **Número de cuadros desordenados:**

Cuadro	1	2	3	4	5	6	7	8
Posiciones	1	1	0	0	1	1	0	1

$$\text{total} = 1 + 1 + 1 + 1 + 1 = 5$$

- **Blanco:** 2

### 3.1. Detalle importante de optimización

Al implementar A\* se debe notar que es posible encolar, o colocar en la lista de estados abiertos, el mismo estado o tablero varias veces. Para prevenir la exploración innecesaria de estados, cuando se considere los estados sucesores de un estado actual, se debe evitar encolar o generar un estado que sea igual que el estado antecesor al estado actual. En la figura 3 se muestra un ejemplo de un estado que no debe ser abierto o encolado

8	1	3
4		2
7	6	5

8	1	3
4	2	
7	6	5

8	1	3
4		2
7	6	5

Figura 3: Estado **antecesor**, Estado **Actual** y Estado **Sucesor no permitido**

## 4. Detección de puzzles que no se pueden resolver

Desde cualquier tablero inicial no siempre es posible alcanzar el estado meta. El programa debe detectar si un tablero se puede resolver o no. El número de tableros posibles en el 8-*Puzzle* es igual al número de permutaciones de 9 elementos, es decir,  $9! = 362880$ . Se tiene que 8-*Puzzle*, tiene la propiedad de que está compuesto por 2 grafos desconectados, cada uno conteniendo  $9!/2 = 181440$  estados [4]. Por lo que se hace imperativo tener una forma eficiente de detectar si un estado se puede resolver o no. Para ello se colocará un documento en la pagina web del curso, en la sección *Bibliografía*, en el cual se presenta un procedimiento para determinar si un tablero se puede resolver o no.

## 5. Actividad a realizar

Debe hacer un programa llamado `OchoPuzzle.java` que recibe un archivo con la configuración de un tablero inicial. El programa determina si el tablero se puede resolver o no. Si se puede resolver, retorna la solución de 8-*Puzzle*, que va a consistir en la secuencia de los tableros que componen el camino de costo mínimo, el número de movimientos necesarios para alcanzar el estado meta, el número de estados abiertos (o encolados en el proceso) y el tiempo en milisegundos que dura la búsqueda.

### 5.1. API requerido

Su programa puede requerir de varias estructura de datos. Se pide que implemente la siguiente clase que representa a un tablero, con al menos la siguiente especificación

```
public class Tablero {
    public Tablero(int[][] cuadros) // Constructor de un tablero
    public int zero() // heurística zero
    public int manhattan() // heurística distancia Manhattan
    public int desorden() // heurística del número de bloques fuera de lugar
    public int blanco() // heurística de la posición blanca
    public LinkedList<Tablero> vecinos // Todos los tableros vecinos del tablero actual
}
```

## 6. Entrada / Salida

### 6.1. Entrada

El programa llamando `OchoPuzzle.java` se ejecutará de la siguiente manera:

```
>java OchoPuzzle <heurística> <entrada>
```

Donde *heurística* indica la función heurística a utilizar y puede tomar cuatro valores:

- **z**: Zero heurística
- **d**: Número de cuadros desordenados
- **m**: Distancia Manhattan
- **b**: Blanco

También se tiene que *entrada* es el nombre del archivo donde se define el tablero, en el cual hay tres filas y tres columnas y en donde el espacio en blanco se denota como cero.

## 6.2. Salida

Si el tablero no se puede resolver debe indicarlo. En caso de haber solución la salida deberá mostrar los siguientes datos:

- La secuencia de los tableros que resuelven el *puzzle*
- El número de movimientos
- El número de estados abiertos o encolados
- El tiempo de ejecución del algoritmo A\* en milisegundos

## 6.3. Ejemplo con tablero sin solución

Dado el siguiente archivo de entrada datos.txt

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

Se obtiene la siguiente salida:

```
>java z OchoPuzzle datos.txt
```

```
El Tablero no tiene solucion
```

## 6.4. Ejemplo con tablero con solución

Dado el siguiente archivo de entrada datos.txt

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

Se obtiene la siguiente salida:

```
>java m OchoPuzzle datos.txt
```

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

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

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

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

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

```
Numero de movimientos: 4
Numero de estados abiertos: 10
Tiempo: 3 ms
```

## 7. Elementos a incluir en el informe

Su informe debe seguir la estructura dada al principio del curso. Dentro de ese mismo formato debe incluir los siguientes puntos:

- Debe explicar *brevemente* las estructuras de datos realizadas
- Explique *brevemente* como se implementó la clase `Tablero.java`
- Realice al menos 5 casos de pruebas relevantes, en los cuales se pueda hacer una comparación de las heurísticas, a través del siguiente cuadro comparativo:

Tablero	Movimientos	Estados Abiertos			
		Zero	Desorden	Manhattan	Blanco

- Haga un *breve* análisis del cuadro anterior. ¿Cuál es la mejor heurística?

### 7.1. Sobre los criterios de corrección

Se corregirá la ejecución y la calidad del código. Toda entrega que no pueda ser compilada sin errores tendrá **CERO** en la parte de ejecución.

## 8. Entrega

El día jueves de la semana 12, los alumnos de Hilmar Castro entregarán a las 9:30 am en su oficina y los de Guillermo Palma a las 1:30 pm en el Laboratorio E. Se deberá entregar un sobre cerrado debidamente identificado con nombre, carnet y profesor de laboratorio. Dentro del sobre deberán estar incluido:

- Listados (documentados) de los fuentes del TAD **Grafo** y la aplicación de prueba
- Un **CD** debidamente identificado, libre de virus y defectos físicos cuyos únicos directorios sean `bin/` (conteniendo los archivos `.class`), `src/` (conteniendo los archivos `.java`) `doc/` (conteniendo la documentación generada con `javadoc`).
- Un informe que describa el proyecto y su implementación. En la página del curso podrá encontrar la información de la estructura y formato del mismo.

No se aceptan proyectos que no tengan un respaldo impreso.

## Referencias

- [1] K.A. Berman and J. Paul. *Fundamentals of Sequential and Parallel Algorithms*. PWS Publishing Co. Boston, MA, USA, 1996.
- [2] I. Millington. *Artificial Intelligence for Games*. Morgan Kaufmann, 2006.
- [3] M. Ortega and O. Meza. *Grafos y Algoritmos*. Equinoccio, 2006.
- [4] A. Reinefeld. Complete Solution of the Eight-Puzzle and the Benet of Node Ordering in IDA. In *Procs. Int. Joint Conf. on AI, Chambery*, pages 248–253, 1993.
- [5] S. Russel and Norvig. *Artificial Intelligence*. Prentice-Hall, 2003.
- [6] Varios. A\* search algorithm. Website, 2008. [http://en.wikipedia.org/wiki/A\\*\\_search\\_algorithm](http://en.wikipedia.org/wiki/A*_search_algorithm).
- [7] Varios. Fifteen puzzle. Website, 2008. [http://en.wikipedia.org/wiki/Fifteen\\_puzzle](http://en.wikipedia.org/wiki/Fifteen_puzzle).