Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-2693 Laboratorio de Algoritmos y Estructuras III
Trimestre Enero-Marzo de 2022

Resolviendo el 8-*Puzzle* con A^*

1. Introducción

El objetivo de este proyecto es el de resolver el problema del 8-*Puzzle* con el algoritmo A^*. El algoritmo A^* es un algoritmo informado de búsqueda de caminos de costo mínimo. Para este proyecto debe implementar la versión del algoritmo A^* data en la clase de teoría.

2. Descripción del problema

El problema de 8-*Puzzle* [3] consiste en un tablero $3x3$, el cual contiene 8 bloques cuadrados numerados, desde el 1 hasta el 8, y un cuadrado vacío (en nuestra representación será numerado con 0). Los cuadrados adyacentes al cuadrado vacío pueden moverse a ese espacio, y entonces dejar el espacio que ocupaba como cuadrado vacío. El objetivo es mover los cuadrados a partir de una configuración inicial del tablero, que llamamos **estado inicial**, hasta alcanzar una configuración específica que llamaremos **estado meta**. Laa Figura 1a muestra un ejemplo de un estado inicial y la Figura 1b la de un estado meta del 8-*Puzzle*.

8	1	3
4		2
7	6	5

(a) Ejemplo de un estado inicial

1	2	3
4	5	6
7	8	

(b) Ejemplo de un estado meta

Figura 1: Ejemplo de dos estados del 8-*Puzzle*

La formulación del del 8-*Puzzle* contiene los siguientes elementos:

Estados: un estado corresponde a un tablero del 8-*Puzzle*, en el cual la posición de cada uno de los ocho cuadrados es dada. La Figura 1 muestra dos estados.

Estado inicial: es el estado en el que se inicia la búsqueda.

Estado meta: es el estado meta que se quiere alcanzar en la búsqueda, a partir de un estado inicial. Para este proyecto, **el estado meta que es solución del problema**, es el presentado en Figura 1b.

Acciones: Dado un estado, las acciones posibles a ejecutar son las de mover un cuadrado hacia el espacio vacío, aplicando un movimiento del cuadrado en alguna de las siguientes direcciones: izquierda, derecha, arriba y abajo.

Función de sucesores: es una función que dado un estado, entonces retorna los estados **adyacentes válidos** que resultan de las cuatros posibles acciones indicadas anteriormente.

Grafo de estados: es un grafo en el cual los vértices son estados. Existe un lado entre un estado A y un estado B, si es posible llegar desde el estado A hasta el estado B, aplicando alguna *acción*. Es decir, el estado B es un estado sucesor del estado A que se obtuvo aplicando la **función para obtener sucesores** al estado A.

Costo del camino: el costo de un lado que une a dos estados adyacentes es **uno**. Se puede construir un camino en el **grafo de estados** que representa cada uno de los movimientos que se efectúan desde un **estado inicial** hasta un **estado final**. En costo de ese camino es igual al número de pasos (o movimientos) que se dan para alcanzar el estado meta.

Función de prueba de meta: dado un estado, retorna *cierto* si un estado es el estado meta, y *falso* en caso contrario.

En la Figura 2 se puede observar el **grafo de estados** que se obtiene al resolver el 8-*Puzzle* con el algoritmo de búsqueda en amplitud. También en la Figura 2 se muestra el camino más corto resaltado. **El objetivo en este proyecto es encontrar el camino más corto desde el estado inicial, hasta el estado meta usando el algoritmo** A^*.

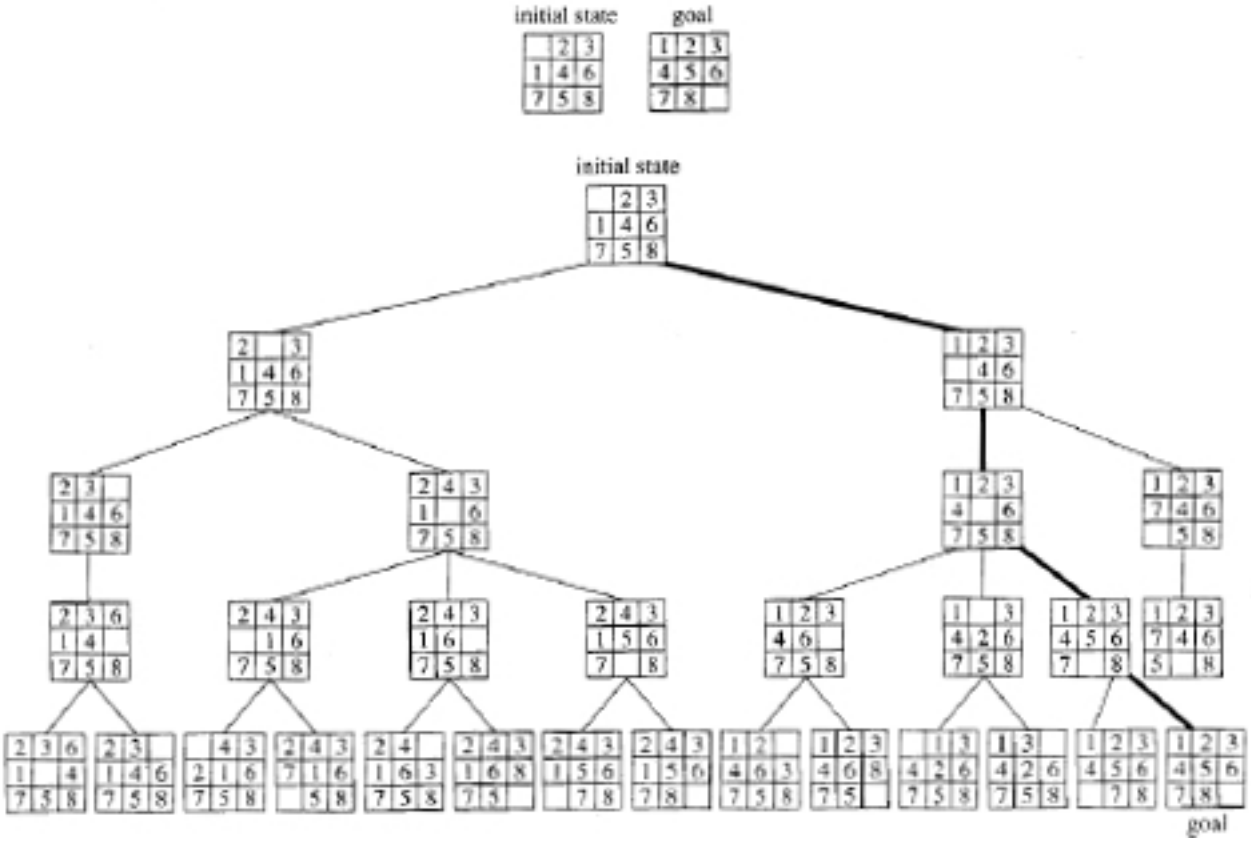

Figura 2: *Grafo de estados* del 8-*Puzzle* obtenido aplicando el algoritmo de BFS. Fuente [1]

3. Detalles de diseño e implementación

A continuación se describiran los parámetros de la realización del proyecto.

3.1. Sobre el grafo a utilizar

El grafo a usar para resolver este problema es un *grafo implícito*. Esto es, el *grafo de estados* se va construyendo a partir del *estado inicial*. Luego al estado inicial se le aplica la *función para obtener sucesores*, parta obtener los vértices adyacentes. Y así sucesivamente se va aplicando para cada uno de los estados, hasta que se alcanza el *estado meta*. Observe que no se guarda el grafo completo, que consiste tener en memoria todas las posibles combinaciones de tablero. Por el contrario, se van almacenando los estados que se van observando a medida que se expande un estado dado.

3.2. Funciones estimativas $\hat{h}$

Se debe implementar cuatro funciones estimativas $\hat{h}$ o heurísticas, que se describen a continuación:

1. **Zero**: en esta heurística todo estado da como resultado zero. Es decir, para cualquier estado n, se tiene que $\hat{h}(n) = 0$
2. **Distancia Manhattan**: es la suma de las distancias verticales 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. La idea detrás de esta heurística es que mientras más desordenado está el tablero, más difícil debe ser ordenarlo.
4. **Blanco**: distancia entre la posición en el estado meta del cuadrado 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**: 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

3.3. Evitando explorar estados visitados anteriormente

Al realizar la búsqueda con el algoritmo A^*, tenemos que es posible visitar 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 generar un estado que sea igual que el estado antecesor al estado actual o algún otro estado que ya sea parte del *grafo de estados*. La Figura 3 muestra un estado sucesor de un estado actual, el cual no debe ser abierto.

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**

3.4. Detección de tableros que no se pueden resolver

Es posible que desde un tablero inicial no sea posible alcanzar el estado meta. Su programa debe detectar si un tablero, que representa 8-*Puzzle*, 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 [2]. Por lo que se hace imperativo tener una forma eficiente de detectar si desde un estado inicial, podemos alcanzar al estado meta o no. En la sección de anexo se presenta un procedimiento para determinar si un tablero se puede resolver o no, el cual usted debe implementar.

4. Sobre el programa cliente

En programa cliente que soluciona el 8-*Puzzle*, debe implementarse en `Kotlin` en un archivo llamado `EightPuzzle.kt`. Tambien debe crear un programa Makefile que compile todos los archivos del proyecto. Los proyectos que **no se compilen o que no se ejecuten**, serán calificados con **cero**.

4.1. Entrada de los datos

El programa `EightPuzzle.kt` debe ejecutarse por medio de un *shell script*, llamado `runEightPuzzle.sh` con la siguente línea de comando:

```
>./runEightPuzzle.sh <heurística> <instancia>
```

Donde `heurística` es una letra que indica la heurística a utilizar, que corresponde a los siguientes caracteres:

- **z**: Zero.
- **d**: Número de cuadros desordenados.

- **m**: Distancia Manhattan.
- **b**: Blanco.

Se tiene que `instancia` es el nombre de un archivo con los datos del tablero, el cual es representado con tres filas y tres columnas de números desde el 0 hasta el 8, separados por un espacio en blanco. Si en la entrada no se indica una heurística y/o una instancia, entonces se debe dar un mensaje de error al usuario.

4.2. Salida de los datos

Si el tablero no se puede resolver se debe imprimir por la salida estándar la frase **"No hay solución"**. En caso de haber una solución la salida, deberá mostrar los siguientes datos: (I) la secuencia de los tableros que resuelven el *puzzle* y que componen el camino de costo mínimo. El formato del tablero es el mismo del de la entrada, y después cada tablero se separa del otro con un salto de línea, (II) sigue una línea en blanco para luego tenerse una línea con el número de estados abiertos, (III) una línea con tiempo de ejecución del algoritmo A^* en milisegundos.

4.3. Ejemplos prácticos

A continuación se presenta dos ejemplos de problemas a resolver, junto con sus respuestas correctas.

4.3.1. Ejemplo con un tablero que no tiene solución

Dado un tablero inicial almacenado en un archivo llamado `datos1.txt` con el siguiente contenido:

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

Al ejecutar el comando:

```
>./runEightPuzzle.sh z datos1.txt
```

Se debe obtener la siguiente salida:

```
No hay solución
```

4.3.2. Ejemplo con tablero que tiene solución

Dado el siguiente archivo de entrada `datos2.txt`

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

Al ejecutar el comando:

```
>./runEightPuzzle.sh m datos2.txt
```

Se muestra un ejemplo de una salida factible:

```
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 estados abiertos: 10
Tiempo: 3 ms
```

5. Informe del proyecto

Debe realizar un informe, en formato PDF, llamado `informeProyecto2.pdf` con el siguiente contenido:

Portada: debe incluir: (I) nombre de la universidad, (II) nombre del departamento, (III) nombre del curso, (IV) título del trabajo, (V) nombre y carné de los estudiantes, y (VI) nombre del docente del curso.

Plataforma de desarrollo: debe describir la plataforma usada para el desarrollo del proyecto. En la sección debe incluir: (I) el sistema de operación, (II) la versión del compilador de `Kotlin`, (III) la versión de la maquina virtual de `Java`, (IV) el modelo del CPU, y (V) la cantidad de memoria del computador usado.

Diseño de la solución: debe explicar como solucionó el problema planteado.

Detalles de la implementación: se espera que describa las estructuras de datos usadas y que explique aspectos relevantes de la implementación.

Estudio experimental: Se quiere que haga un estudio experimental sobre la eficiencia de las heurísticas. Pruebe su solución con al menos 2 casos de pruebas no triviales y *con el estado inicial más largo para resolver*, en los cuales se realizará una comparación de las heurísticas, a través del siguiente cuadro:

Estado inicial	**costo del camino**	**Número de estados abiertos**			
		Zero	**Desorden**	**Manhattan**	**Blanco**

Haga un análisis del cuadro anterior, en donde uno de los puntos que debe tratar es la respuesta a la pregunta, ¿cuál es la mejor heurística y por qué?

Estado actual de los programas: debe indicar si el programa que resuelve el problema planteado, si esta operativo o no, y si tiene errores conocidos.

Referencias bibliográficas: en caso de haber utilizado para la elaboración del proyecto.

6. Condiciones de la entrega

Los códigos del proyecto, el informe y la declaración de autenticidad debidamente firmada, deben estar contenidos en un archivo comprimido, con formato *tar.xz*, llamado *Proy2_X_Y.tar.xz*, donde X y Y son los número de carné de los estudiantes. La entrega del archivo *Proy2_X_Y.tar.xz*, debe hacerse por medio de la plataforma *Classroom* antes de las 9:30 A.M. del día viernes 22 de abril de 2022.

Referencias

[1] Berman, K., and Paul, J. *Fundamentals of Sequential and Parallel Algorithms.* PWS Publishing Co. Boston, MA, USA, 1996.

[2] Reinefeld, A. Complete Solution of the Eight-Puzzle and the Benet of Node Ordering in IDA. In *Procs. Int. Joint Conf. on AI, Chambery* (1993), pp. 248–253.

[3] Russel, S., and Norvig. *Artificial Intelligence: A Modern Approach*, 4th ed. Prentice-Hall, 2021.

Guillermo Palma / gvpalma@usb.ve / Abril 2022

A. Procedimiento que indica si un estado inicial tiene solución

```
Here is a simple rule for telling if an 8 puzzle is solvable.

Recall that our goal state is

  1 2 3
  4 5 6
  7 8 0

where the 0 represents the blank.

Now, consider the following arrangement of tiles

  1 2 3
  4 5 7
  6 8 0

The first line is in the correct order. The second
line is correct except that counter 7 comes before
counter 6. We can call this violation of the order
an "inversion".

If the number of inversions is even then the puzzle
is solvable. If the number of inversions is odd
then the puzzle is unsolvable.

Consider the puzzle

2 4 3
1 0 6
7 5 8

We have the following inversions

  2 comes before 1
  4 comes before 3 and 1
  3 comes before 1
  6 comes before 5
  7 comes before 5

for a total of 6 inversions. This puzzle is solvable.
Note that we don't need to consider the blank (0).

More examples:
```

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

  5 comes before 2 3 4
  8 comes before 2 3 4 6 7

For a total of 8 inversions. This puzzle is solvable.

  5 1 8
  0 2 3
  4 6 7

  5 comes before 1 2 3 4
  8 comes before 2 3 4 6 7

for a total of 9 inversions. This puzzle is not solvable.

  2 0 7
  8 5 4
  3 6 1

  2 comes before 1
  7 comes before 5 4 3 6 1
  8 comes before 5 4 3 6 1
  5 comes before 4 3 1
  4 comes before 3 1
  3 comes before 1
  6 comes before 1

for a total of 18 inversions. This puzzle is solvable

  8 7 6
  5 4 3
  2 1 0

  8 comes before 7 6 5 4 3 2 1
  7 comes before 6 5 4 3 2 1
  6 comes before 5 4 3 2 1
  5 comes before 4 3 2 1
  4 comes before 3 2 1
  3 comes before 2 1
  2 comes before 1

for a total of 28 inversions. This puzzle is solvable (although
```

```
you may run out of memory before finding a solution).
```