Lenguajes de Programación: Trabajo Práctico III

M. Sc. Saúl Calderón Ramírez
Instituto Tecnológico de Costa Rica,
Escuela de Ingeniería en Computación,
PAttern Recognition and MAchine Learning Group (PARMA-Group)

25 de septiembre de 2016

El presente proyecto introduce el uso del lenguaje Scala, al propósito de la programación funcional, el cual trata sobre la implementación de un «resolvedor» automático para el problema del *rompecabezas deslizante*.

- **Fecha de entrega: 25 de Octubre**
- **Modo de trabajo**: Grupo de tres personas.
- **Tipo de entrega:** digital, por medio de la plataforma TEC-digital.

Parte I
Motivación

Los algoritmos de optimización son un área activa de investigación en la inteligencia articial. Tales algoritmos buscan una solución óptima (global o local) de un problema, muchas veces bajo un conjunto de restricciones y usualmente buscan minimizar el costo computacional de realizar tal búsqueda. Ejemplos de enfoques de optimización son los algoritmos basados en el gradiente, algoritmos genéticos, algoritmos simplex, etc. Uno de los enfoques más usados desde la Inteligencia artificial es el de árboles de búsqueda, y entre los algoritmos populares en este enfoque se puede mencionar el algoritmo A* (A estrella), propuesto por Nils Nilson, uno de los pioneros de la Inteligencia Artificial, en 1968 [1]. Los algoritmos de búsqueda definen un problema como un conjunto de variables, cuyos valores óptimos o «resueltos» es necesario encontrar, y muchas veces también, la secuencia de pasos para llegar a tal estado. A continuación se definen algunos conceptos básicos en los algoritmos de búsqueda.

- Un *estado* S_i se define como los valores particulares que toman un arreglo de N variables:

$$S_i = \langle x_1, x_2, \ldots x_N \rangle \, .$$

Las variables modelan un estado físico o abstracto de un problema.

- Un estado meta G_p se define como el estado objetivo a alcanzar, por lo que un problema resoluble tiene al menos un estado meta, por lo que $p > 0$. La función:
$$g(S_i) = \begin{cases} 1 & S_i = G_m \\ 0 & S_i \neq G_m \end{cases}$$
evalúa si un estado S_i es un estado meta.

- Un estado derivado, alcanzable o sucesor de un estado S_i se denota como:
$$S_{i \to j} = \langle x_1, x_2, \ldots x_N \rangle$$
y es un estado alcanzable a partir del estado S_i, el cual respeta un conjunto de restricciones $\mathcal{R}$.

- Para un problema específico, se define el espacio de estados de tamaño M, como el conjunto de estados plausibles (que respetan al conjunto de restricciones $\mathcal{R}$):
$$\Omega = \{S_1, S_2, \ldots, S_M\}.$$

- Una función sucesoria $f(S_i)$ recibe un estado S_i y tiene como salida el conjunto de estados de todos los estados alcanzables desde el estado S_i:
$$\{S_{i \to 1}, S_{i \to 2}, \ldots S_{i \to j}\} = f(S_i).$$

- Muchas veces, es necesario modelar el *costo* de transición de un estado a otro, por lo que ello se representa en la función de costo:
$$\zeta(S_i, S_j) = c_{i,j}$$

- El camino $p(S_m, S_n)$ de largo K entre un estado S_i y un estado S_j se define entonces como un arreglo de estados consecutivamente sucesorios:
$$p(S_m, S_n) = \langle S_{i_1}, S_{i_2}, \ldots, S_{i_K} \rangle \Leftrightarrow S_{i_{a+1}} = S_{i_a \to j}, S_{i_1} = S_m, S_{i_k} = S_n$$
y el costo total del camino P_k se define como la suma de los costos de transicionar entre estados sucesivos dentro de tal camino, con el funcional:
$$z(P_k) = \zeta(S_{i_1}, S_{i_2}) + \zeta(S_{i_2}, S_{i_3}) + \ldots + \zeta\left(S_{i_{K-1}}, S_{i_K}\right).$$

- Una heurísitca
$$H(S_i) = \widehat{z}(S_i, G_p)$$
es un algoritmo o método que estima el costo $\widehat{z}$ de llegar a un estado específico S_i (o cualquiera, según la heurística).

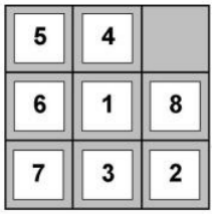

Figura 1: Arreglo de fichas para un tablero de 3×3.

La notación anterior permite formular algoritmos de búsqueda abstraídos del problema específico a resolver, facilitando la formulación de distintos algoritmos de búsqueda de soluciones. Algunos ejemplos de problemas específicos son el mundo de los cubos (analizado en el curso), juegos de tablero (ajedrez, damas chinas, etc), búsqueda de ruta óptima, etc. Para la presente tarea programada se resolverá el problema específico del rompecabezas deslizante, usando el algoritmo A*. Sin embargo, es necesario notar que tal enfoque se puede aplicar para distintos problemas específicos.

Parte II
Problema del rompecabezas deslizante

1. Representación del problema

El problema del rompecabezas deslizante es una variante de problemas particulares, usados comúnmente para estudiar distintos algoritmos de resolución de problemas en la inteligencia artificial. Consiste en un tablero de $M = N \times N$ fichas o entradas, con cada ficha etiquetada con un número $n = 0, 1, 2, \ldots, M$ (**restricción 1**). En todo el tablero la etiqueta de la ficha no se puede repetir (**restricción 2**) . La Figura 1 muestra un arreglo de fichas para un tablero de dimensiones $M = 9 = 3 \times 3$.

En la Figura 1 , la esquina superior derecha se dice que está «libre», y numéricamente se representa con la etiqueta $n = 0$. Tomemos entonces el estado de la figura anterior como el estado válido (pues cumple las restricciones 1 y 2) S_1, el cual, simbólicamente se representaría con la matriz de variables:

$$\begin{bmatrix} x_{1,1} = 5 & x_{1,2} = 4 & x_{1,3} = 0 \\ x_{2,1} = 6 & x_{2,2} = 1 & x_{2,3} = 8 \\ x_{3,1} = 7 & x_{3,2} = 3 & x_{3,3} = 2 \end{bmatrix},$$

de modo que en general, un estado para el problema del rompecabezas desli-

zante, se representa como:

$$S_i = \langle x_{1,1}, x_{1,2}, \ldots x_{N,N} \rangle$$

Para cambiar el estado del juego, el rompecabezas deslizante permite «deslizar» las fichas en un vecindario de grado 4 (horizonatales y verticales, no diagonales) (**restricción de sucesión**). Por ejemplo, para el estado S_1, es posible deslizar la ficha 8 hacia arriba, y la ficha 4 hacia la derecha, lo que significa que la función sucesoria resulta en:

$$f(S_1) = \{S_{1\to 1}, S_{1\to 2}\}$$

y etiquetando a $S_{1\to 1} = S_2$ y $S_{1\to 2} = S_3$ para simplificar la notación, se tiene que:

$$S_2 = \begin{bmatrix} x_{1,1} = 5 & x_{1,2} = 0 & x_{1,3} = 4 \\ x_{2,1} = 6 & x_{2,2} = 1 & x_{2,3} = 8 \\ x_{3,1} = 7 & x_{3,2} = 3 & x_{3,3} = 2 \end{bmatrix},$$

$$S_3 = \begin{bmatrix} x_{1,1} = 5 & x_{1,2} = 4 & x_{1,3} = 8 \\ x_{2,1} = 6 & x_{2,2} = 1 & x_{2,3} = 0 \\ x_{3,1} = 7 & x_{3,2} = 3 & x_{3,3} = 2 \end{bmatrix}.$$

El costo asociado de transicionar del estado S_1 al S_2 y del estado S_1 al S_3 es equivalente, y viene dado por la «cantidad de movimientos realizados para llevar a cabo tal transición», por lo que entonces:

$$\zeta(S_1, S_2) = \zeta(S_1, S_3) = 1.$$

El estado objetivo o meta, se puede definir arbitrariamente, pero usualmente se fija un solo estado en el que las fichas se encuentran en forma ascendente, desde la esquina superior izquierda, dejando el espacio libre en la esquina inferior derecha:

$$G_1 = \begin{bmatrix} x_{1,1} = 1 & x_{1,2} = 2 & x_{1,3} = 3 \\ x_{2,1} = 4 & x_{2,2} = 5 & x_{2,3} = 6 \\ x_{3,1} = 7 & x_{3,2} = 8 & x_{3,3} = 0 \end{bmatrix}.$$

Una solución se define entonces como el camino entre un estado inicial S_i y el estado meta G_1lo que corresponde a:

$$p(S_m, S_n) = \langle S_{i_1}, S_{i_2}, \ldots, S_{i_K} \rangle.$$

2. El algoritmo A*

Habiendo planteado la representación del problema del rompecabezas deslizante, es posible plantear distintos algoritmos para resolverlo. En el presente proyecto se implementará el algoritmo A* (A estrella o voraz), el cual desarrolla un árbol de búsqueda como el que se observa en la Figura 2.

A diferencia de algoritmos más simples de búsqueda como los algoritmos de búsqueda por ancho primero (visita y evalúa primero todos los estados de

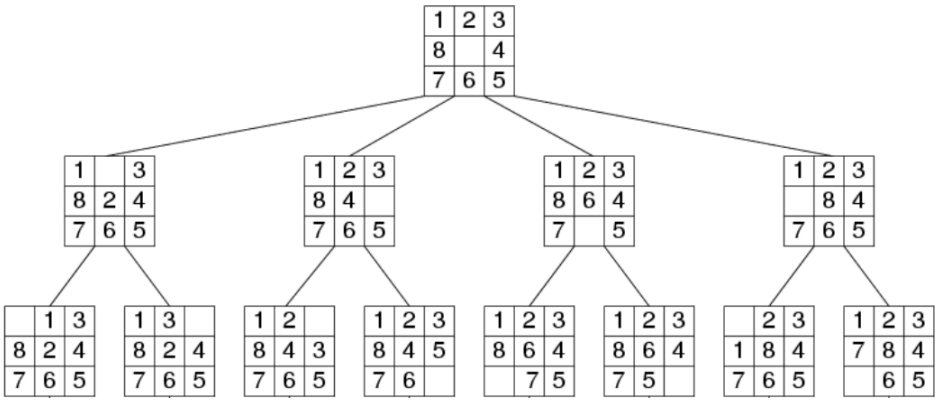

Figura 2: Arbol de búsqueda.

un nivel para averiguar si alguno es un estado meta) y por profundidad primero (la primer rama se desarrolla al máximo hasta encontrar un estado meta), el algoritmo voraz utiliza una o más heurísticas para decidir que ramificación desarrollar (posterior a realizar la evaluación de la función $g(S_i)$). Esto permite encontrar soluciones de manera más eficiente (con menos desarrollos de los nodos). El algoritmo voraz a implementar debe además acotar la búsqueda, impidiendo el desarrollo de nodos ya encontrados, para así evitar el enciclamiento del algoritmo. A continuación se presentan dos heurísticas sencillas. Puede encontrar más en el siguiente enlace: `https://heuristicswiki.wikispaces.com/N+-+Puzzle`.

2.1. Heurística de distancia de Manhattan

Suponga que existe un solo estado meta G, como el mostrado en la Figura 3.

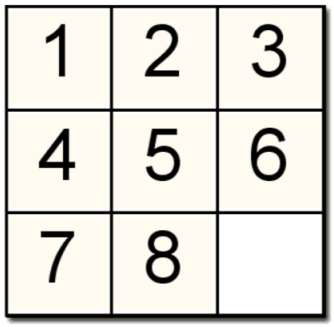

Figura 3: Estado meta G de ejemplo.

Cada ficha r_m, con $m = 0, 1, \ldots, 8$ (en el caso del tablero de 3×3 fichas) tiene una posición meta compuesta por la fila $\hat{i}_m$ y columna $\hat{j}_m$. Así por ejemplo, para la ficha r_8, en el estado meta, su posición está dada por el par ordenado posicion $(r_8, G) = \left\langle \hat{i}_8, \hat{j}_8 \right\rangle = \langle 3, 2 \rangle$. La distancia de Manhattan conceptualiza la distancia entre dos puntos $p_1 = \langle i_1, j_1 \rangle$ y $p_2 = \langle i_2, j_2 \rangle$ como el recorrido en bloques, «cuadras» o «manzanas» más corto para comunicar ambos puntos,y viene dada por:

$$d_M(p_1, p_2) = |i_1 - i_2| + |j_1 - j_2|.$$

Así por ejemplo, con un estado inicial S_1 como el presentado en la Figura 4, se tiene que: posicion $(r_8, S_1) = p_1 = \langle i_8, j_8 \rangle = \langle 1, 1 \rangle$. De esta manera, si hacemos $p_2 = \langle 3, 2 \rangle$, se arriba a la siguiente distancia de Manhattan:

$$d_M(p_1, p_2) = |1 - 3| + |1 - 2| = 2 + 1 = 3.$$

8		2
5	1	3
4	7	6

Figura 4: Estado inicial S_1.

La sumatoria de la distancia de Manhattan para todas las fichas en el tablero, tomando como posición p_1 la posición en el estado actual S_i y como posición $p_2$la posición en el estado meta G, permite formular una heurística H_M o aproximación de cuántos movimientos son necesarios para llegar al estado meta (suponiendo un tablero vacío):

$$H_M(S_i) = \sum_{m=0}^{8} \left| i_m - \hat{i}_m \right| + \left| j_m - \hat{j}_m \right|.$$

Así para el ejemplo en el que S_1 se define en la Figura 4 y el estado G se define

en la Figura 3, se tiene que:

$$
\begin{aligned}
d_M\left(\langle 1,1\rangle, \langle 2,2\rangle\right) &= |1-2| + |1-2| = 2 \\
d_M\left(\langle 1,2\rangle, \langle 1,3\rangle\right) &= |1-1| + |2-3| = 1 \\
d_M\left(\langle 1,3\rangle, \langle 2,3\rangle\right) &= |1-2| + |2-3| = 2 \\
d_M\left(\langle 2,1\rangle, \langle 3,2\rangle\right) &= |2-3| + |1-2| = 2 \\
d_M\left(\langle 2,2\rangle, \langle 2,1\rangle\right) &= |2-2| + |2-1| = 1 \\
d_M\left(\langle 2,3\rangle, \langle 3,3\rangle\right) &= |2-3| + |3-3| = 1 \\
d_M\left(\langle 3,1\rangle, \langle 3,2\rangle\right) &= |3-3| + |1-2| = 1 \\
d_M\left(\langle 3,2\rangle, \langle 1,1\rangle\right) &= |1-3| + |1-2| = 3 \\
d_M\left(\langle 3,3\rangle, \langle 1,2\rangle\right) &= |1-3| + |3-2| = 2 \\
H\left(S_1\right) &= 2+1+2+2+1+1+1+3+2 = 15
\end{aligned}
$$

Parte III

Requerimientos del programa

Requerimientos mínimos:

- El programa debe posibilitar la resolución de un tablero de 3×3.
- Se deben implementar al menos dos heurísticas distintas de su preferencia.
    - Las heurísticas deben implementarse como funciones, las cuales deben pasarse **por parámetro** al solucionador voraz.
- Se debe desplegar de manera legible la solución del problema (secuencia de estados).
- El programa debe proveer una interfaz de línea de comandos que permita inicializar un tablero de 3×3 dimensiones.

Requerimientos extra:

- El programa debe posibilitar la resolución de un tablero de $N \times N$ dimensiones.
- Implementar una o más heurísticas adicionales.
- Implementar una interfaz gráfica que permita inicializar el tablero, y visualizar la solución.
- Definir uno o más estados meta personalizados.

3. Implementación

El programa debe implementarse en su totalidad usando Scala, explotando al máximo las facilidades pertinentes del lenguaje. Tal como se mencionó, las heurísticas deben implementarse como funciones, las cuales deben pasarse **por parámetro** al solucionador voraz. Para la documentación interna y dem´sa aspectos del proyecto, se deben seguir los lineamientos estipulados en el programa del curso.

Referencias

[1] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.