**Endika Salgueiro Barquin**

**Javier Prada Leizea**

**SI-02**

**Sistemas Inteligentes**

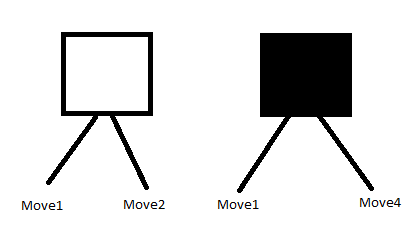
**WALKING OVER A LINE OF BLACK & WHITE SQUARES**

**Descripción del problema como un problema de búsqueda:**

* **BreathFS:**

Factores de búsqueda que emplea este algoritmo:

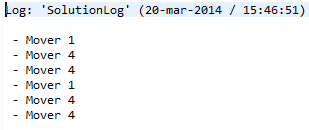
1. Branching factor “b”: La traducción literal sería factor de ramificación. Con este factor medimos el número de ramas de nuestro árbol de búsqueda. En nuestro problema, estemos en la casilla que estemos, siempre tendremos dos opciones (si estamos en una casilla blanca podremos movernos 1 o 2 posiciones; mientras que, si estamos en una casilla negra podremos movernos 1 o 4 posiciones). Es por eso, que nuestro factor de ramificación tendría el valor de 2.



1. Maximun Depth “m”: Este factor mediría el número de posiciones que tendría nuestro problema. En nuestro caso sería 19 (19 casillas).



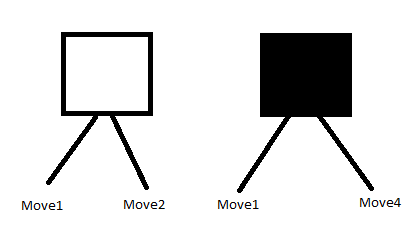
1. Shortest Depth “d”: Este factor mediría el mínimo coste de la solución, es decir, el mínimo número de movimientos que tenemos que hacer para resolverlo. En nuestro caso, sería 6.



* **DepthFS:**

Factores de búsqueda que emplea este algoritmo:

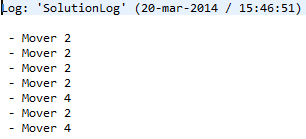
1. Branching factor “b”: La traducción literal sería factor de ramificación. Con este factor medimos el número de ramas de nuestro árbol de búsqueda. En nuestro problema, estemos en la casilla que estemos, siempre tendremos dos opciones (si estamos en una casilla blanca podremos movernos 1 o 2 posiciones; mientras que, si estamos en una casilla negra podremos movernos 1 o 4 posiciones). Es por eso, que nuestro factor de ramificación tendría el valor de 2.



1. Maximun Depth “m”: Este factor mediría el número de posiciones que tendría nuestro problema. En nuestro caso sería 19 (19 casillas).

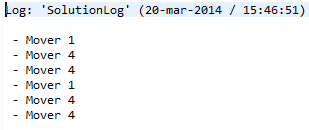


1. Shortest Depth “d”: Este factor mediría el mínimo coste de la solución, es decir, el mínimo número de movimientos que tenemos que hacer para resolverlo. En nuestro caso, sería 7.

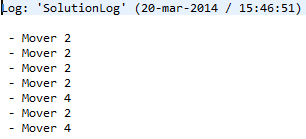


* **Conclusión:**

Utilizando el algoritmo de búsqueda a lo ancho (BreathFS) tardamos más en encontrar la solución pero no el coste es menor; es decir, nos encuentra la solución en menos movimientos.



Utilizando el algoritmo de búsqueda en profundidad (DepthFS) tardamos menos en encontrar la solución pero el coste es mayor; es decir, nos encuentra la solución en más movimientos.



Por lo tanto, el algoritmo de búsqueda a lo ancho (BreathFS) es más completo y óptimo.

**Análisis del rendimiento de nuestro problema:**

* El orden en el que cargamos los operadores en la lista influye directamente en el orden en el que se van a ejecutar. Por lo tanto si tenemos varios operadores y colocamos uno que no utilicemos muy a menudo el primero, perderíamos eficacia. Lo que tendríamos que hacer es cargar la lista de los operadores en el orden que nos gustaría que se ejecutasen.
* Los tiempos de búsqueda de los dos algoritmos son más que aceptables ya que sólo tardan unas milésimas de segundos. En caso de querer un tiempo más rápido, nos tendríamos que decantar por el algoritmo de búsqueda en profundidad. De esta forma, mejoraríamos el rendimiento de nuestro sistema.
* Para mejorar la calidad de nuestra solución, tendríamos que añadirle al algoritmo de búsqueda una Graph Search.