Instituto Tecnológico de Costa Rica

Victoria Molina Martínez

Jose Gabriel Pérez Tullocks

Análisis de Algoritmos

Kenneth Obando Rodríguez

Proyecto I

Grupo3

23/10/2025

1. Algoritmos implementados

1. Backtracking

Se trata de una técnica para resolver problemas de forma recursiva probando todas las posibles soluciones y devolviéndose cuando un medio parcial no sea un resultado válido. Pero cuando se dice que “se devuelve”, es que termina una parte del proceso recursivo para volver a la línea anterior; es decir, vuelve a ejecutarse en el punto de decisión anterior para probar otra alternativa disponible.

1.1. Características

1.1.1. Depth-First Search (DFS):

Se refiere a un algoritmo utilizado para recorrer árboles o grafos, que busca ir lo más lejos posible en cada rama antes de tener que devolverse. Asimismo, utiliza una estrategia LIFO (Last in, First out); es decir, que utiliza una pila implícita o explícita.

Su complejidad para el peor caso es O(V), donde la V se refiere a los vértices del árbol o grafo. Sin embargo, es importante destacar que el DFS no busca optimización, sino dar una solución.

1.1.2. Pruning (Poda):

Es una técnica de optimización que consiste en eliminar ramas completas del espacio de búsqueda cuando se determina que no se puede obtener una solución válida de ese camino.

1.1.3. Solución incremental:

Cuando se habla a una estrategia incremental, se refiere a la estrategia para construir una solución paso a paso, verificando que cada paso de la solución si sea un camino vigente.

1.2. Tipos de problemas para usar Backtracking

1.2.1. Problemas de decisión

1.2.2. Problemas de optimización

1.2.3. Problemas de enumeración

2. DFS puro

Anteriormente se menciona este algoritmo de búsqueda, pero ahora vamos a referirnos a su forma más básica y fundamental, sin optimizaciones, sin poda y sin heurísticas; es decir, que explora el espacio de manera sistemática y ciega. En este caso la estrategia que se trabaja es ir profundo en el árbol primero y luego la anchura; en otras palabras, se explora el árbol lo más posible por cada rama, cuando llega al final retrocede al último punto de decisión y prueba la siguiente alternativa favorable.

2.1. Características

2.1.1. A ciegas y sistemático

Cuando se habla de que trabaja a ciegas, se refiere a que no se tiene conocimiento del problema, lo que implica que no va a ir evaluando si un camino es “superior” a otro, si no que sigue un orden predefinido.

2.1.2. Completo, pero no óptimo

Es completo porque si hay solución la va a encontrar, pero no es óptimo, porque en un problema largo puede que tome mucho tiempo en encontrar una solución.

2.2. Tipos de problemas en que se puede usar DFS

2.2.1. Recorrido de Grafos simple

2.2.2. Conteo de componentes conexas

2.2.3. Detección de ciclos

3. Breadth-First Search (BFS)

Se trata de un algoritmo de búsqueda que explora un árbol o un grafo nivel por nivel; es decir, que empieza desde un nodo raíz y explora todos los nodos vecinos hasta tener que pasar al siguiente nivel.

3.1. Características

3.1.1. Estrategia FIFO

Esta estrategia trabaja con el principio de “First in, First out”; en otras palabras, trabaja con una cola.

3.1.2. Camino más corto en grafos no ponderados

Esta propiedad se refiere a un teorema que establece que BFS, en grafos no ponderados, encuentra el camino más corto en términos del número de niveles (aristas) desde un nodo raíz, hasta terminar todo el grafo.

4. A* (A-Star)

Se refiere a un algoritmo de búsqueda, que en grafos encuentra el camino más corto entre dos nodos, por ejemplo: Encontrar el camino más corto desde un nodo raíz hasta una meta (un punto de llegada), tomando en cuenta “el costo” de cada vía tomada.

4.1. Características

4.1.1. Best-First Search (BFS)

Es un algoritmo heurístico que consiste en seleccionar el nodo más prometedor para expandir, basándose en una función de evaluación. Prioriza los nodos dentro del espacio de búsqueda utilizando una función heurística que estima su potencial para llegar a una solución óptima, dando preferencia a aquellos con mejor valoración.

La función de evaluación es f(n)=g(n)+h(n), donde g(n) es la función de certeza y h(n) es la predicción. Se balancea porque se utiliza lo que ya se tiene más lo que se predice que puede pasar.

4.1.2. Cola de prioridad

Se utiliza una cola de prioridad logra que siempre se procese el nodo con f(n) más bajo primero.

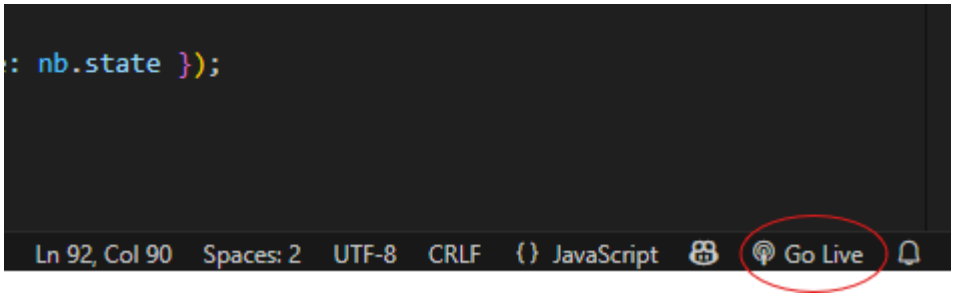
4.1.3. Optimo y completo

Este algoritmo es completo y óptimo porque siempre va a encontrar la solución más corta, siempre y cuando exista una.

2. Guía de uso de la aplicación

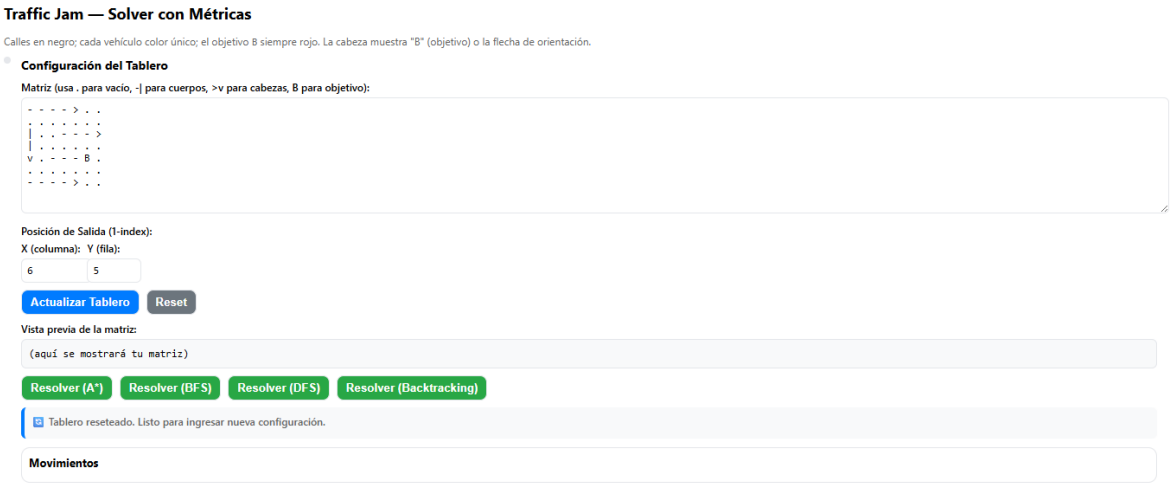
1. Abrir la App Web

Para ello, se carga el folder del proyecto en Visual Studio Codo. Una vez está el proyecto abierto, se busca la opción de “Go Live” (asumiendo que se tiene instalada la extensión de Live Share), se da click en esa opción, y el programa se abrirá en su navegador web predeterminado.



2. Utilizar la aplicación

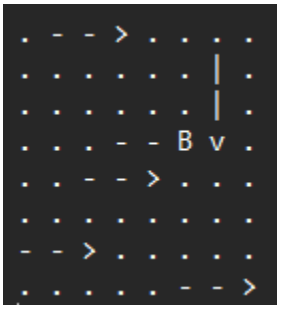
Una vez, el navegador web presente, la siguiente interfaz:



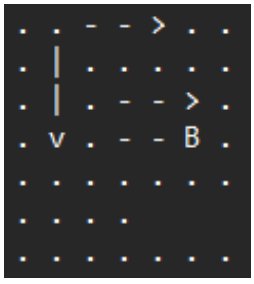
En la parte que dice “Configuración del tablero”, el usuario puede ingresar la matriz a la que le desea aplicar cada uno de los algoritmos presentes en las opciones. Siga el ejemplo propuesto como una guía para evitar errores, o pegue una matriz previamente creada, y no se olvide escribir las coordenadas de donde se encuentra el carro que desea buscar (refiriéndose a la serie de líneas que terminan en B); donde “**x” se refiere a las columnas**, y **“y” a las filas de la matriz**.

2.1. Ejemplos de matrices válidas:

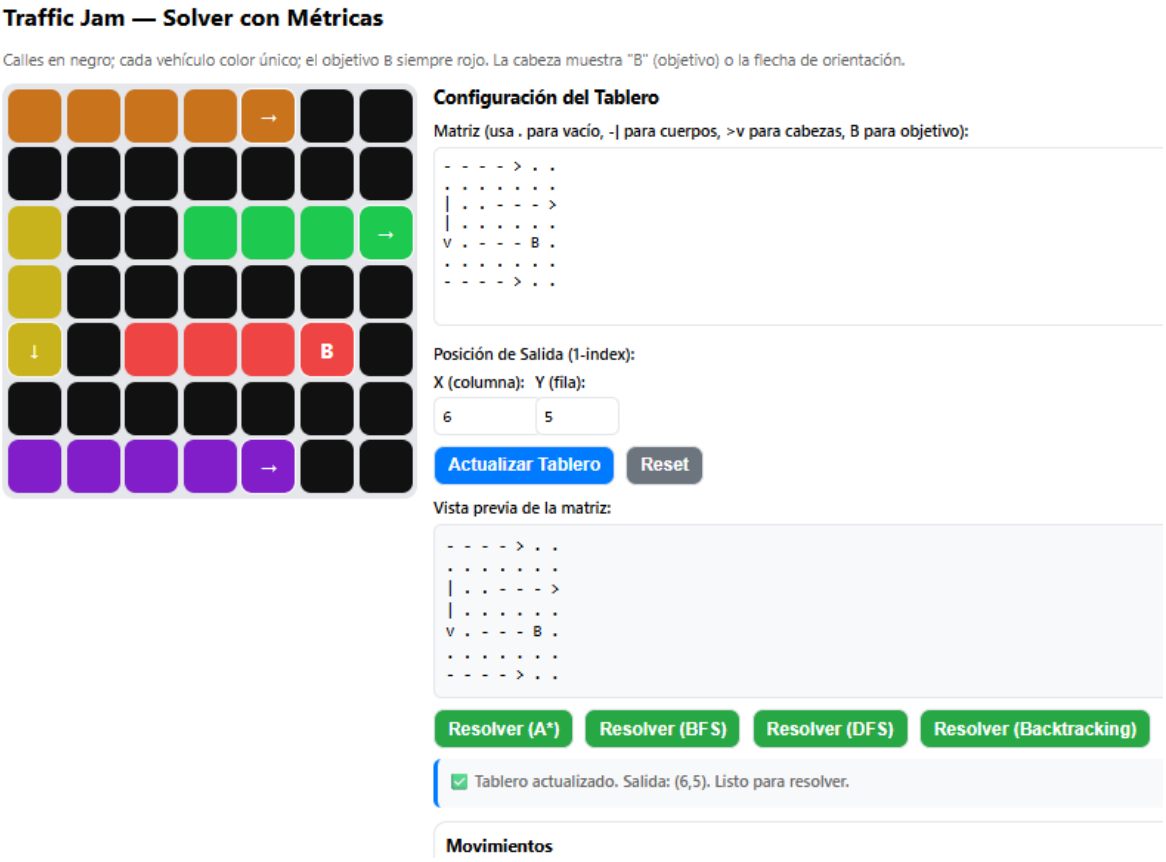
2.1.1. Con x=8 , y=4



2.1.2. Con x=7 , y=4

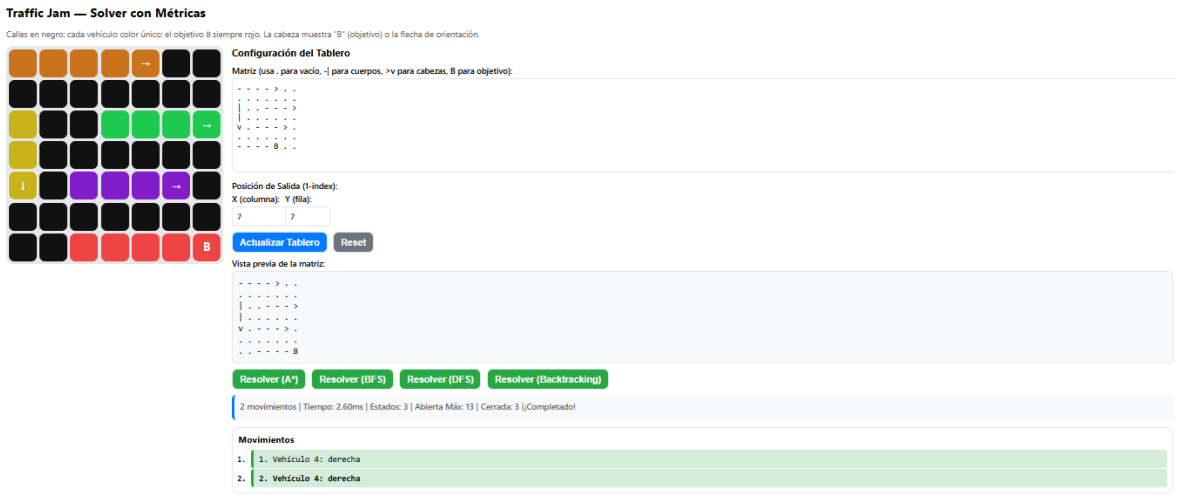


Después de tener la matriz deseada, se da click en el botón que indica “Actualizar tablero”, la interfaz va a cambiar de la siguiente forma:



Una vez se vea así, se debe elegir una de las opciones para probar en la matriz indicada. Apenas se le da click a la opción deseada, todos los carritos se van a mover hasta brindar el resultado, con la información obtenida durante la corrida.

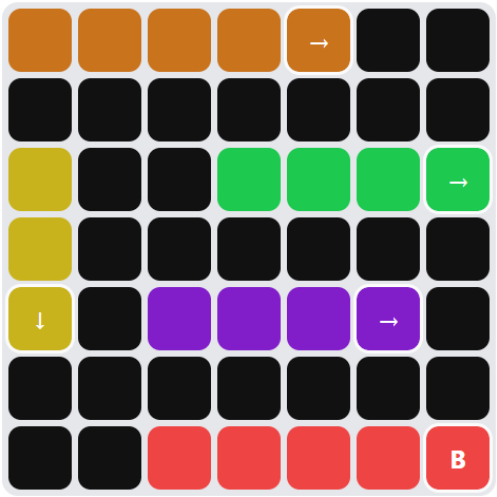
2.2. Ejemplo usando A*



Cuando finaliza el proceso, el usuario puede seleccionar otra opción sin tener que resetear la matriz de cero; en caso, de que desee cambiar la matriz por completo, dé click en el botón que dice “Resetear”.

3. Análisis comparativo de rendimiento

Al comparar cada uno de los algoritmos implementados, se estará usando la siguiente matriz de ejemplo:



La comparación entre los algoritmos se realiza mediante el tiempo, los movimientos, los estados, caminos abiertos y caminos cerrados; dando la posibilidad de poder verificar cuál es el óptimo de cada uno de los algoritmos utilizados.

Usando la matriz de ejemplo, se ha creado la siguiente tabla donde se desglosan los resultados obtenidos de cada algoritmo.

Tabla de comparación					
Algoritmo	Movimientos	Tiempo	Estados	Caminos abiertos	Caminos cerrados
Bactracking	2	0.70ms	3	x	x
DFS puro	13	0.90ms	14	x	x
BFS	2	4.20ms	28	x	x
A*	2	0.50ms	3	13	3

Los resultados muestran que **A*** es el algoritmo con mejor rendimiento general. Aunque obtiene la misma cantidad mínima de movimientos que Backtracking y BFS, lo hace en **menos tiempo (0.50 ms)** y con **menos estados explorados (3)**, gracias a su uso de heurísticas y al manejo eficiente de caminos abiertos y cerrados.

Backtracking, aunque logra una solución óptima en movimientos, explora muy pocos estados y no emplea estructuras como listas abiertas/cerradas; esto lo hace rápido en este caso, pero **no garantiza eficiencia en tableros más complejos**, ya que depende de búsqueda ciega.

DFS puro presenta el peor desempeño en movimientos, ya que encuentra una solución mucho más larga (**13 movimientos**) y explora más estados que Backtracking y A*, lo que evidencia su carácter no óptimo y su tendencia a profundizar sin evaluar costo.

BFS, por su parte, garantiza soluciones óptimas en movimientos, pero su **tiempo de ejecución (4.20 ms)** y el **número de estados explorados (28)** lo convierten en una opción **correcta pero costosa** en recursos frente a A*.

En conjunto, se observa que **A*** logra el mejor balance entre rapidez, óptimo número de movimientos y bajo costo computacional, mientras que los demás algoritmos sacrifican eficiencia o calidad de solución en diferentes niveles.