Algoritmos y Estructura de Datos

Douglas Leonel Barrios Gonzalez

Universidad del Valle de Guatemala

Documentación Proyecto 1

Julio Barahona M. , 141206

Rafael León , 13361

Pablo López , 14509

# Posibles Algoritmos a Implementar:

## Tremaux’s Algorithm

Algoritmo eficiente que requiere líneas en el camino, trabaja para laberintos que tienen pasajes bien definidos. Cada vez que se elige una dirección está marcada por trazar una línea en el suelo. Se elige una dirección inicial aleatoria. Al llegar a un cruce que no se ha visitado antes, elegir una dirección aleatoria que no está marcada y marcarla el camino. Al llegar a un cruce marcado y si su ruta actual se marca solamente una vez y luego dar la vuelta y caminar hacia atrás (y marcar el camino por segunda vez). Si este no es el caso, tomar la dirección con las marcas de menor cantidad (y marcarlo, como siempre). Cuando finalmente llegar a la solución, caminos marcados exactamente una vez indicarán de manera directa de nuevo al comienzo. Si no hay salida, este método le llevará de nuevo al comienzo, donde todos los caminos están marcados dos veces. En este caso, cada ruta se bajó exactamente dos veces, una vez en cada dirección. El paseo resultante se denomina bidireccional doble trazado

## Dijkstra’s Algorithm

Asignar a cada nodo un valor de distancia tentativa: ponerlo a cero para nuestro nodo inicial y hasta el infinito para todos los demás nodos. Ajuste el nodo inicial como corriente. Marcar todos los otros nodos no visitados. Crear un conjunto de todos los nodos no visitados llamados el conjunto no visitado. Para el nodo actual, tenga en cuenta todos sus vecinos no visitados y calcular sus distancias tentativas. Comparar la distancia tentativa recién calculado con el valor asignado actual y asignar la más pequeña. Por ejemplo, si el nodo actual A está marcada con una distancia de 6 y el borde de conexión con un vecino B tiene una longitud de 2, entonces la distancia a B (a través de A) será 6 + 2 = 8. Si B era previamente marcado con una distancia mayor que 8 de cambiarlo a 8. de lo contrario, mantenga el valor actual. Cuando se llevan a cabo teniendo en cuenta todos los vecinos del nodo actual, marque el nodo actual como visitado y sacarlo del conjunto no visitado. Un nodo visitado nunca se volverá a comprobarse. Si el nodo de destino ha sido marcado visitada (hora de planificar una ruta entre dos nodos específicos) o si la distancia tentativa más pequeño entre los nodos en el conjunto no visitado es infinito (hora de planificar un recorrido completo; se produce cuando no hay conexión entre el nodo inicial y la restante nodos no visitados), y luego se detiene. El algoritmo ha terminado. De lo contrario, seleccione el nodo no visitado que está marcado con la distancia más pequeña tentativa, establecerlo como el nuevo "nodo actual", y volver al paso 3.

## Breadth wide explore:

También conocido como breadth first search, es una técnica que se utiliza para recorrer una grafo finito. Se utiliza como un algoritmo para resolver el problema un laberinto y entra entre los algoritmos que resuelven un laberinto cuando el participante está dentro y no tiene un conocimiento previo del mismo. Consiste en visitar primero una esquina luego visita a las esquinas que pueden derivarse de estas y guarda el proceso en una cola. Usualmente se utiliza para encontrar el camino más corto entre un punto a otro. Entre sus aplicaciones más utilizadas están: análisis de redes, generación de algoritmos de laberintos y probar grafos para detectar si son bigrafos.

Wall follow:

Es también conocido como la regla de la mano derecha o la regla de la mano izquierda. Pertenece al grupo de algoritmos que funcionan para personas dentro del laberinto que no tienen conocimiento alguno del laberinto en el que se encuentran. Consiste en colocar una mano en una pared (izquierda o derecha) y recorrer así el laberinto. Una ventaja muy grande es que encontrar una solución para el laberinto está garantizada ya que eventualmente se habrán recorrido todas las paredes. Su desventaja principal es que solo funciona con laberintos simplemente conectados, si el laberinto no es simplemente conectado al usar este algoritmo se puede quedar encerrado en un ciclo infinito en un camino sin salida.

## Algoritmo a Utilizar:

### Wall Follow algorithm(Mano izquierda):

Se decidió utilizar el algoritmo de Seguir la Pared (Izquierda o Derecha)

* La primera razón es que es un algoritmo centrado en el participante. Este no requiere un conocimiento previo del laberinto, ni conocimiento sobre la salida para poder encontrarla. A diferencia de algoritmos basados en grafos y nodos. Que requieren saber en donde se encuentra el punto inicial y el punto final.
* La siguiente razón es que el algoritmo de la mano derecha es más fácil de implementar que algoritmos como el algoritmo de tremaux; en un robot físico. Ya que Tremaux necesita marcar paths en el laberinto para saber que lugares no recorrer de nuevo
* Recursos:
  + Tiempo: Al ser un algoritmo de exploración de profundidad. Sigue un camino hasta que se encuentre la salida.
    - En el mejor de los casos. Puede resolver el laberinto y salir rápidamente. Si la salida se encuentra cercanamente en la dirección de la pared que se sigue
    - En el peor de los casos: Puede recorrer todo el laberinto para llegar a la salida
  + Espacio: El espacio requerido es solo el necesario para las acciones básicas del robot, ya que no se guarda nada, ni se crean objetos adicionales en el mundo.

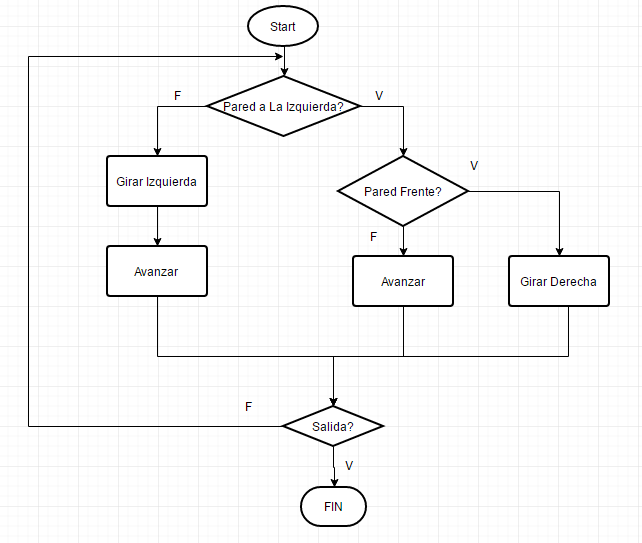
### Ventajas sobre los otros algoritmos investigados:

* No tiene que marcar caminos que ya se recorrieron
* No requiere tantos recursos como los algoritmos de Breadth Wide exploration, que llenan todo el laberinto.
* No requiere conocer el final del laberinto para encontrar una solución.

### Desventajas:

* Si la meta se encuentra en un punto central, separada de la pared, va a ser imposible de alcanzar. Se asume que por ser un laberinto físico y porque el objetivo del proyecto es “SALIR DEL LABERINTO”, este se encuentra al lado de una pared, por tanto esta desventaja es despreciable.

### Diagrama de Flujo

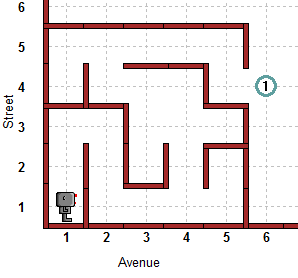


## Tests en Implementación:

Se realizaron tests al algoritmo para ver cuánto tiempo le tomaba en resolver el algoritmo desde 3 distintos puntos del mismo. Se corrió a la velocidad máxima de RUR-PLE

## Laberintos de Distinta Complejidad

### Laberinto 1:

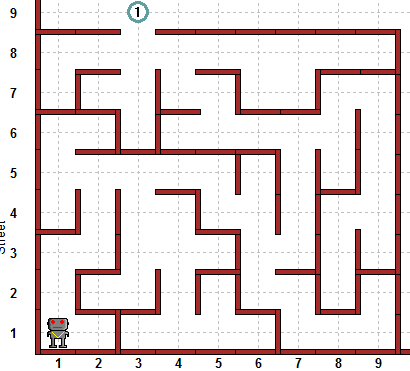


Tiempo desde (1,1): 1.05 s

Tiempo desde (5,2): 2.37 s

Tiempo desde (3,2) 1.19 s

### Laberinto 2:

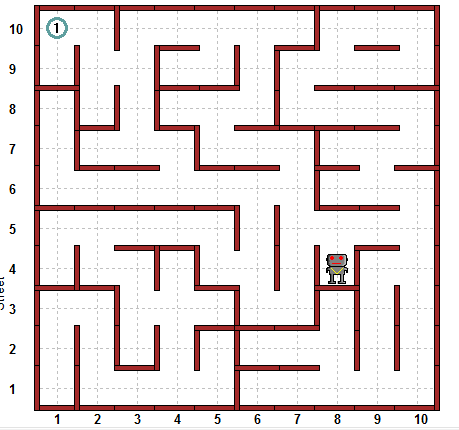


Tiempo desde (1,1): 4.04 s

Tiempo desde (6,5): 2.04 s

Tiempo desde (3,2) 4.88 s

### Laberinto 3:



Tiempo desde (8,4): 7.71 s

Tiempo desde (1,1): 3.85 s

Tiempo desde (10,10): 13.02 s

## Referencias:

<http://www.wseas.us/e-library/conferences/2008/crete/Systems/sys1-32.pdf>

<https://www.cs.bu.edu/teaching/c/tree/breadth-first/>