Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento Ciencia de la Computación y TI
Algoritmos y Estructuras de Datos
Catedrático: Douglas Barrios
Auxiliares:

Proyecto #1
Fase 1: Construcción de un algoritmo

María Fernanda Estrada 14198
Luis Abadia 13300
Jorge Súchite 15293
Agosto 3, de 2016

Diferentes algoritmos que existen para salir de un laberinto

Desde tiempo antiguos existen los tan famosos y divertidos “laberintos”. Los vemos, en los periódicos, en la primaria, en jardines y hasta en aquel laberinto que a diario resolvemos a lo largo del tiempo llamado vida.

No es de sabios ni de letrados que puede ser un entretenimiento o hasta un reto para los extremistas; tanto así, que existen algoritmos para poder resolverlos de la mejor manera y en menor tiempo. Entre los más comunes están:

Algoritmos aleatorios

Este tipo de algoritmos tienen como ideal el explorar de manera “aleatoria” el laberinto. Es decir, puede elegir irse por la izquierda como irse por la derecha; puede irse para adelante o para atrás; también puede detenerse un tiempo determinado así como recorrer una distancia cualesquiera.

El robot no tiene una inteligencia artificial, así que está propenso a chocar con las paredes y dirigirse a otro lado donde no choque. Además, el robot puede irse por una dirección contraria a la que se desea. Este algoritmo resuelve el laberinto, pero en un tiempo indeterminado, el tiempo que se tarde en explorar es demasiado tedioso y longevo.

Algoritmo Tremaux

Este algoritmo data desde el siglo XIX y es llamado así por su inventor Charles Trémaux, quien relata y da estos simples pasos los cuales le ayudaron a él para resolver un laberinto.

Las instrucciones son tan sencillas que cualquiera puede utilizarlas. No seguir el mismo camino dos veces y si se llegase a un cruce nuevo esto no importará si lo toma o no. Tanto así que si llegásemos a un camino viejo o, en el peor de los casos, un callejón sin salida solo tendríamos que regresar a la entrada del camino que nos llevó hasta allí. Y si un camino viejo lo lleva a un cruce ya conocido es mejor tomar un camino nuevo para encontrar algo nuevo.

Son cuatro pasos sencillos pero sabemos que este algoritmo nos puede llevar horas y horas pero de que es funcional lo es.

Algoritmo “Ruta Predeterminada”

El objetivo de este algoritmo es presentarle al robot una ruta inicial para que el pueda seguirla, la cual puede cambiar en el transcurso del recorrido y así, el poder trazar otra ruta nueva en el punto donde se pudo haber quedado.

Algoritmos de procesamiento de información

Este algoritmo es uno de los más efectivos puesto que se toman absolutamente todos los datos que se tienen a ese momento. Se toma el problema como una ruta predeterminada a alcanzar y así, cuando el robot se tope en el camino con obstáculos pueda mapear otra vez y tener una ruta predeterminada más exacta para no volver a quedarse en el mismo obstáculo.

Este algoritmo funciona puesto que siempre tendrá información nueva la cual se estará actualizando el mapa con los obstáculos encontrados hasta el momento. El robot, al momento de volver a entrar en el laberinto tendrá una ruta más conveniente para él.

Algoritmo de Backtracking

Este tipo de algoritmo es útil para aquellos problemas que de por sí, tienen una solución completa (que si se puede resolver) en los cuales no nos interesa el orden que tengan los elementos de nuestro problema. Nosotros podemos utilizar las variables a nuestro parecer pero sin olvidarnos de las restricciones que se tienen al momento de que se plantea el problema.

El término de “backtracking” fue introducido a nuestro léxico por el matemático Derrick Henry Lehmer a mitad del siglo XX y diez años más tarde fue más formalizado para su utilización en el curso de “Algoritmos y Estructuras de Datos”.

Como se había mencionado antes, este algoritmo es utilizado cuando en el problema es necesaria una serie de decisiones (críticas) las cuales deben ser las necesarias y las adecuadas para poder llegar a la resolución del problema. Estas decisiones deben satisfacer tanto las restricciones como el tiempo requerido; la solución no solo debe darnos el resultado sino debe ser la más eficiente tanto en el tiempo como en el espacio. Al igual, dicho algoritmo utiliza una funciones llamadas "recursivas" en las que se deben asignar un valor determinado para cada una de las posibilidades.

Las ventajas de este algoritmo son tantas que si existe una solución, este calcula la más eficiente, es flexible y/o viable a características de cualquier problema. No obstante, si el rango de la solución se define como infinito, aunque exista la solución este algoritmo, nunca la encontrará. Con respecto a las funciones recursivas, estas conllevan un espacio considerable de memoria puesto que este algoritmo tiene costos exponenciales en la mayoría de los casos.

Algoritmo "Divide y vencerás"

Este algoritmo (en nuestro contexto) es un eficiente diseño el cual consiste en dividir un problema y a partir de sus subdivisiones resolver el problema en partes pequeñas. Cabe notar que si los subproblemas todavía son un poco grandes se aplica otra vez el algoritmo para poder alcanzar una subdivisión tal que se puedan resolver directamente. Para poder utilizar este tipo de algoritmos se debe de tomar en cuenta las siguientes observaciones:

El problema tiene que poder dividirse en pequeñas porciones de él mismo, al igual que cada subdivisión de tenga que resolver individualmente o bien, de forma recursiva y por último, que el problema, luego de haber resuelto sus subdivisiones pueda volver a unirse a un problema como tal pero ya resuelto.

En términos de eficiencia, podemos decir que sus subdivisiones se comportan de forma logarítmica o lineal pero una desventaja característica es que deteriora el código resultante después de la resolución del problema. No olvidemos que, los algoritmos diseñados con la ideología de "Divide y vencerás" heredan sus características. Son robustos, legibles, claros y concisos y fácil de depurar los códigos basura. Se tiene desventaja en lo material puesto que conlleva mucho espacio en la memoria.

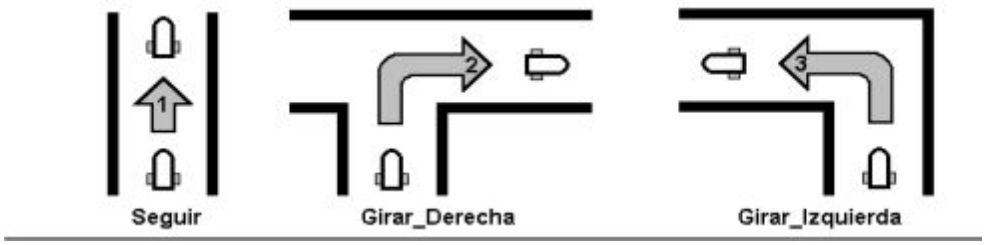
Algoritmo seguidor pared

Este tipo de algoritmo es uno de los más comunes en la robótica aplicada. Este algoritmo se le es aplicado al robot para que siga a la pared (como su nombre lo menciona) de diferentes formas y circunstancias.

Este tipo de algoritmo lo rigen unas restricciones las cuales son:

- Si el robot tuviera una pared a su derecha y no tuviera una adelante, este seguiría en línea recta.
- Si el robot tiene una pared enfrente y ninguna hacia la derecha entonces, seguiría caminando pero esta vez a la derecha.
- En cambio, si el robot tuviera una pared adelante y otra pared a su derecha; la única salida sería irse por la izquierda

Ejemplo:



Algoritmo de la mano derecha

Este algoritmo es uno de los más sencillos, puesto que la dinámica es poner; ya sea la mano derecha o izquierda en la pared y avanzar hasta llegar a la salida. No podemos decir que en todos los laberintos es funcional puesto que si las paredes no están conectadas nos veríamos en un gran problema. Es más, el laberinto también tiene que ser simple para poder utilizarlo.

Algoritmo de Pledge

Este algoritmo se centra en apuntar a la misma dirección en la cual se inició manteniéndolo hasta el final. Como es uno de los tipo de mano derecha o izquierda,

éste hereda una desventaja. No es útil en todos los laberintos ya que podría quedar en un ciclo de cualquier camino.

Algoritmo Lee

Este algoritmo se divide en dos partes. La primera es tener un mapa del laberinto en la memoria mediante recorridos sucesivos y cuando el robot está haciendo dichos recorridos va guardando las paredes que tiene cada que va pasando por ellas.

Si en algún caso el robot visita un lugar donde no tenga pared y la única que tuviera fuera hacia el norte; el robot automáticamente le asigna un valor predeterminado (8). No obstante, si también pasa por un lugar donde exista una pared al este se le ha de asignar un valor (10). A las paredes del este se le asigna un valor 2.

Como se puede deducir, esta parte ha de consumir mucha memoria por lo que se necesita una de gran tamaño para poder utilizar dicho algoritmo ya que, al momento de tener mapeado el laberinto se le ha de asignar un valor a cada celda del laberinto para que se determine cuál es el camino más corto.

Algoritmo de Tarry

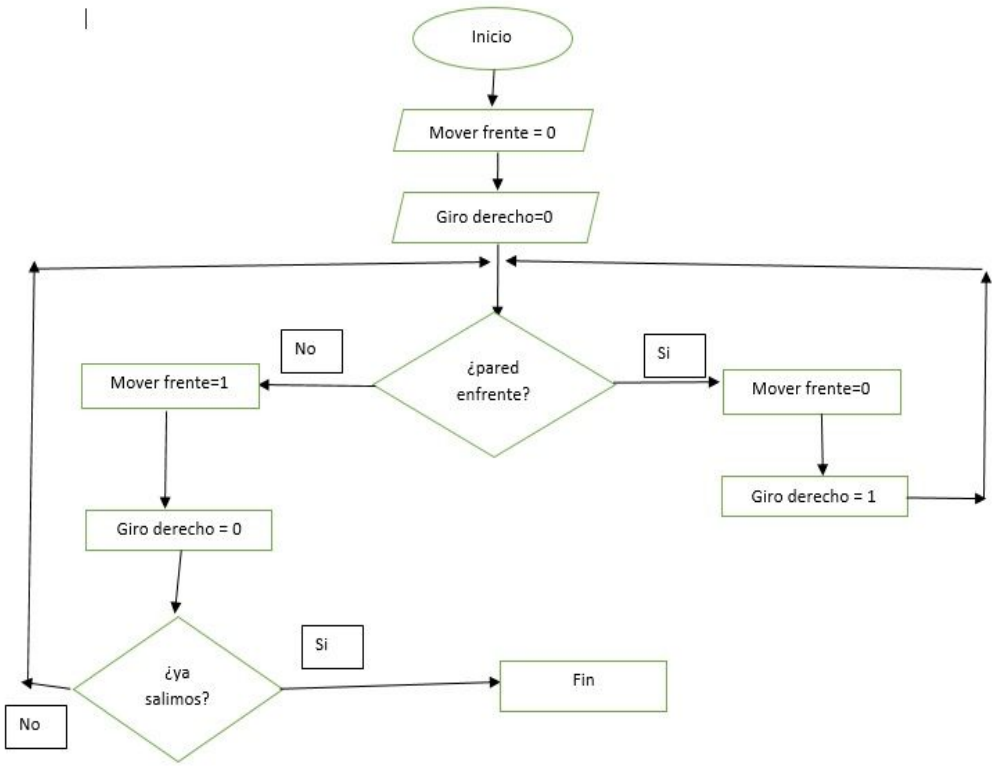
Este algoritmo data del año 1895 proponiendo un camino cíclico en el laberinto a resolver: pasando por cada lugar una sola vez en todas las direcciones, determinando una posición inicial para luego reconocerla y seguir su camino, si llegase a pasar por el mismo cruce este igual lo marca según lo indicado. El robot terminará el laberinto donde empezó.

Razones por las cuales se eligió el algoritmo

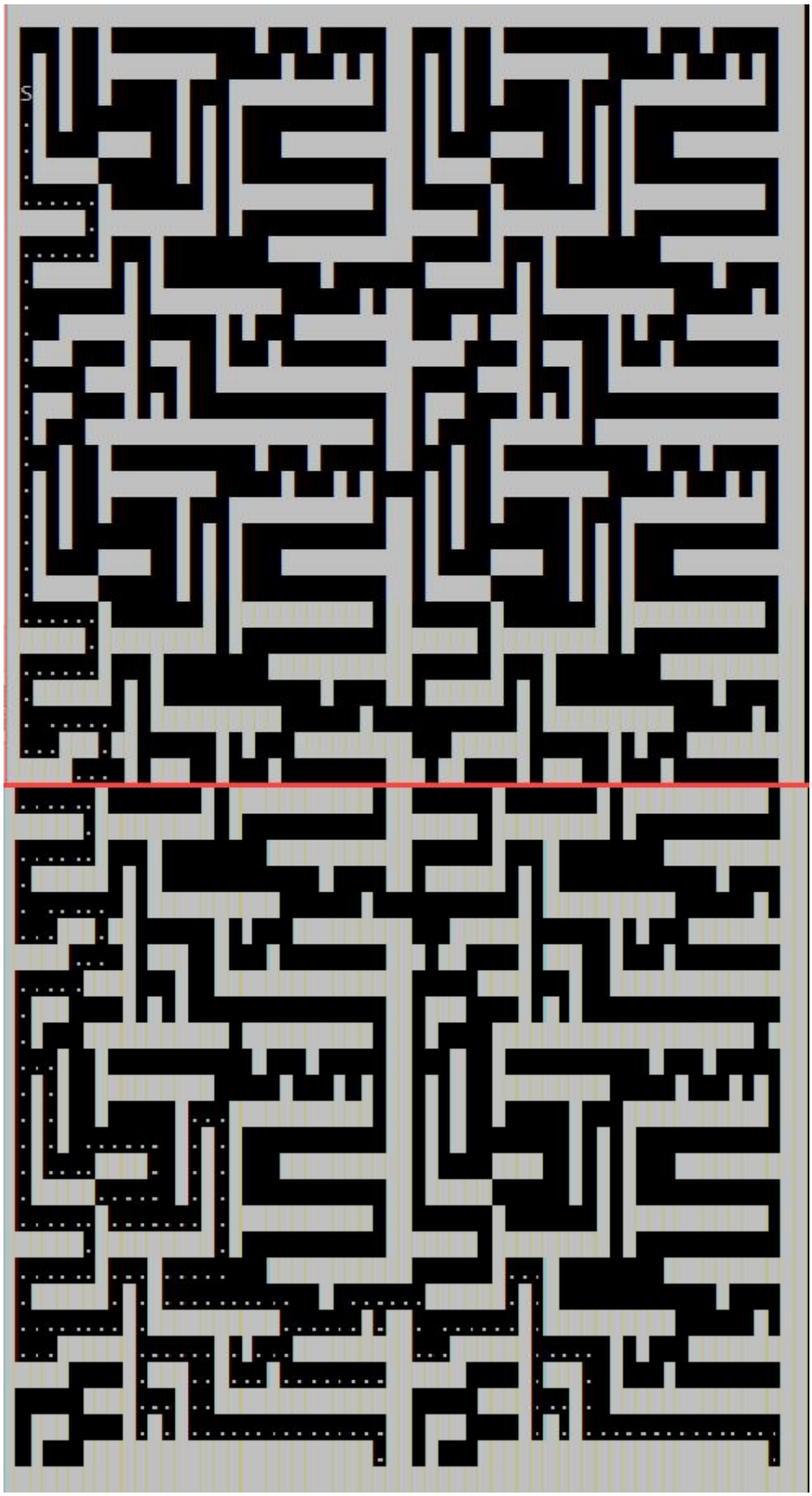
De manera unánime se decidió que se utilizará el algoritmo de la mano derecha puesto que no se cuenta con el tiempo y espacio necesario como para probar el de backtracking que es el segundo algoritmo que tendría la posibilidad de poder ser implementado en la segunda fase del proyecto.

Además, como el diseño el laberinto es variable y se desconoce, se optó por un algoritmo que en su mayoría depende del sensor de la derecha (o izquierda). Almacenar cada trayectoria u obstáculo implica una gran cantidad de memoria que se debe utilizar; la mayoría de los algoritmos encontrados presentan esta desventaja.

Diagrama de Flujo



Simulación


S

Github: https://github.com/JorgeSuc/Proyecto-Brutality

El video está en el repositorio

Bibliografía

Diaz, I. (2013). Algoritmo de backtrackingrecursivo y no recursivopara la resolución de unlaberinto y su aplicación enSDL. Buenos Aires, Argentina: Universidad Tecnológica Nacional Facultad Regional Bahía Blanca.

Paredes,E. Regalado,C. (2006). Diseño y construcción de un robot móvil que pueda desplazarse dentro de un laberinto. Quito,Ecuador: Escuela Politécnica Nacional.