Informe Taller de Programación

Nombre : Alex Pacheco

Sección : A – 1

Fecha entrega : 04 - 04

Búsqueda A\*

# Idea de solución de laberinto

## Consideraciones

Si bien es cierto que en un inicio se planteo un laberinto completamente aleatorio y en ocasiones con secciones de cúmulos de cantidad de obstáculos, al final se decidió por obstáculos aleatorios, con inicio en la coordenada (0,0) y final en (n,n), donde n es largo de la matriz del laberinto.

Bajo la siguiente suposición: la coordenada (0,0) es la esquina superior izquierda y (n,n) la esquina inferior derecha, entonces es posible notar que el camino más corto para llegar llegar al final es suficientes pasos hacia a la derecha y hacia abajo. Si en algún caso se está obligado a ir hacia la izquierda o hacia arriba entonces el camino retrocedió en alguna medida. Lo anterior sugiere priorizar caminos donde solo se baja o se va a la derecha.

## Implementación

Lo que se plantea es generar inicialmente un heap, que manejará todos los nodos en los cuales se recorrerá todas sus posibilidades y una matriz fija de nodos especiales que representarán los nodos ya visitados y contendrán el antecesor más eficiente. Posteriormente, se procede a verificar las posibilidades de movimiento desde un nodo en particular una a una. Si una de ellas está visitada o bien está bloqueada, es decir, es una pared o está fuera de los límites del laberinto, entonces no se agrega a los visitados ni tampoco al heap, en caso contrario se agregan a ambos.

Algo sumamente relevante a señalar es que el criterio de orden dentro del heap es el camino con menor cantidad de movimiento hacia arriba y abajo, es decir, un camino que ha hecho solo movimientos hacia la derecha y hacia abajo tiene mayor prioridad que uno con los movimientos contrarios, esto debido a que gracias al razonamiento anterior ir hacia la izquierda o hacia arriba es retroceder en dirección al nodo inicial.

# Imagen que contiene biombo, edificio, juego, dibujo Descripción generada automáticamenteEjemplificación

Se puede ver que iniciando en (0,0) las posibilidades con coste 0 son hacia la abajo y derecha, por tema de orden de condicionales se elige hacia abajo. Algo a considerar es que los cuadrados marcados con el rombo amarillo representa casillas ya analisadas y agrega a la cola, esto para que posteriormente no se vuelvan a agregar.

Imagen que contiene biombo, juego, edificio, dibujo

Descripción generada automáticamente

Luego de bajar se plantean dos posibilidades, abajo y derecha nuevamente, sin embargo, la diferencia cabe en que una de esas posibilidades es una pared (señalado por triángulos verdes), por ende, no se agrega a la cola ni tampoco se visita, solo se va hacia al nodo de abajo

Imagen que contiene biombo, edificio, dibujo

Descripción generada automáticamente

Luego de saltar algunos pasos se llega a un punto en el que se compara entre subir o bajar desde un nodo en particular. En este caso como subir genera más pasos para llegar a final, este camino no se puede elegir, por ende, se va hacia abajo, nodo desde el cual se llega al final.

## ¿Porque se considera que es un método eficiente?

Esto se debe a que, como se menciona en la explicación, el camino más rápido para llegar es aquellos que no retroceden y, para este laberinto en particular, subir o ir a la izquierda es retroceder. Los caminos donde se retrocede serán revisados recién cuando todos los caminos en donde se avanza de manera constante estén resueltos y no haya posibilidad de que sean la solución.

Se considera que bajo cualquier circunstancia encontrará obligatoriamente el camino más corto existente. Dentro de las búsqueda para verificar que un nodo ha sido visitado se usan matrices fijas, por ende, tienen O(1); luego, el heap presenta inserciones de O(log(n)) y para conseguir el valor mayor se emplea O(log(n)). En base a lo anterior mencionado el peor caso se puede entender como O(n^2) mientras que el mejor caso es de O(n).

## Compilación y ejecución

Inicialmente para lograr compilar todo se debe iniciar en alguna terminal el comando “make”, posteriormente existen diferentes testeos de las distintas clases: “make run”, para testear el main; “make runmaze”, para testear la clase maze; “make runcontainer“, para testear la clase container; “make runarbol”, para testear la clase árbol y, por último, “make runnode”, para testear la clase node.

## Consideraciones finales

En diferentes momentos la ejecución del programa generaba problemas de memoria a la hora de ejecutar laberintos de 1000x1000, para ello se uso un método donde se reutiliza parte de la memoria creada, se resetean los nodos y resetea el laberinto a la hora de aplicar el método generate().