

**Trabajo Práctico Cuatrimestral**

**2C 2018**

**Programación III**

**Trabajo Práctico**

|  |  |
| --- | --- |
| Profesores | Cancela, Julio |

|  |  |  |
| --- | --- | --- |
| GRUPO 7 | Nombre | L.U. |
|  | Calisi, Lucas Pablo | 1069703 |
|  | Tarantino, Jose Luis | 1072978 |

## Introducción

El problema consiste en encontrar el camino mínimo entre dos puntos que se definen sobre un mapa. En el mapa además se establecen áreas (puntos) donde el costo de atravesarlas será mayor o incluso no será posible atravesar (infranqueables). El algoritmo debe encontrar un camino de puntos contiguos que unan el origen con el destino (si es posible), pasando por las áreas de menor peso y evitando las infranqueables. El algoritmo utilizará la librería proporcionada por la cátedra para graficar el camino en pantalla.

## Resolución

La estrategia está basada en el algoritmo de BFS (Breadth-First search), que parte de un nodo origen y expande todos los nodos adyacentes, repitiendo hasta llegar al nodo destino.

Para la resolución, creamos una estructura Nodo para contener la información del punto de partida, del de destino, y calcular el peso y la distancia al destino.

Manejamos dos estructuras tipo lista:

Hashmap: Para incluir los nodos que se expandieron. La clave de la hashtable será el punto del mapa, y su valor el nodo, esto permite acceder rápidamente a los nodos expandidos.

PriorityQueue: Para incluir los nodos que restan expandir. De esta manera siempre sale de la cola el de menor peso. Esto se hace sobrescribiendo el método CompareTo de la clase nodo, que es usado por la cola para determinar qué elemento debe salir. Dentro del método CompareTo, se compara el peso y la distancia a destino, de esta manera se asegura el camino mínimo.

Iniciamos creando un nodo para el punto origen (primer punto agregado al mapa) y se expanden todos los nodos adyacentes. Para expandir, se toma en cuenta no haber salido de los límites del mapa y la densidad del punto a expandir (descartando los infranqueables). Se repite el proceso hasta encontrar el nodo destino entre los expandidos.

Una vez que se encontró el nodo destino, el camino se arma de atrás hacia adelante (desde destino hasta el origen, siguiendo la referencia al nodo anterior de cada nodo), agregando cada nodo del camino en una lista. El nodo origen es el que tiene anterior en nulo.

## Análisis de complejidad

Método CmcResolucion: O(1) = cte.

Método obtenerNodoDestino: O(n2)

Método expandir: 2(2n+1) \* n2 = 4n + 2 \* n2 => O(n2)

Método puedoExpandir: O(1) = cte.

Método generarCamino: O(n2)

* Complejidad O(n2), cuadrático.

## Conclusiones

**PROS**: Breadth First Search (BFS) garantiza visitar todos los nodos a distancia 1 antes que los de distancia 2, y así sucesivamente. Deepth First Search (DFS) puede tomar caminos demasiado largos en profundidad antes de encontrar el destino. Para el caso de caminos mínimos, es más adecuado (rápido) explorar primero los vecinos inmediatos antes de extenderse a nodos que están a mayor distancia, dado que se podría estar profundizando caminos que no lleguen al destino.

**CONTRAS**: BFS puede utilizar más memoria que DFS, porque BFS mantiene múltiples rutas (vecinos) en memoria, mientras que DFS solo tiene que mantener una rama.

**BFS vs DFS**: BFS encuentra un camino mínimo, mientras que DFS sólo encuentra un camino.

En nuestro caso, utilizamos un hashtable para tener acceso rápido a los nodos ya visitados, y una cola de prioridad para acceder siempre al de menor peso.

BIBLIOGRAFIA:

1. Apuntes de la materia.
2. Breadth First Search Tutorials and Notes. <https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/> [Fecha de consulta: 1/Nov/2018]
3. <https://stackoverflow.com/questions/20192445/which-procedure-we-can-use-for-maze-exploration-bfs-or-dfs> [Fecha de consulta: 1/Nov/2018]
4. <https://courses.csail.mit.edu/6.006/fall10/lectures/lec12-graphs2.pdf> [Fecha de consulta: 14/Nov/2018]
5. <https://cs.stanford.edu/people/abisee/gs.pdf> [Fecha de consulta: 14/Nov/2018]