Implementación del algoritmo *Breadth First Search* en `python`

Francisco Gerardo Meza Fierro

1. Introducción

En esta práctica se hace un código en `python` que implementa el algoritmo *Breadth First Search (BFS)* en grafos propuestos. El código genera un archivo que, al ser leído en `gnuplot`, genera un archivo *eps* que contiene la imagen del grafo creado, mostrando a los nodos con diferentes colores, siendo éstos últimos la representación de los niveles en que los nodos fueron recorridos al emplear el algoritmo *BFS*.

2. Algoritmo *Breadth First Search*

Es un algoritmo de búsqueda que recorre y examina todos los nodos de un grafo, arrojando como resultado la distancia mínima que existe entre un nodo inicial a cualquier otro nodo en el grafo mediante los siguientes pasos:

1. Se fija un nodo inicial i y se almacena en un vector de salida *output*.
2. Los nodos vecinos a i se almacenan en *output* y en una lista *queue*.
3. Se toma el primer elemento de *queue*, se borra de la lista y se exploran sus vecinos: si éstos no se encuentran en *output*, se almacenan tanto en *output* y en *queue*, de lo contrario no se hace nada.
4. Se repite el paso 3 hasta que todos los nodos se encuentren en *output* y hasta que *queue* quede vacía al final del paso 3.
5. Se imprime *output*.

3. Creación de los grafos

Para crear los grafos con los que se probará el algoritmo BFS, primero se crearon los nodos que fueron generados de manera aleatoria con coordenadas reales entre cero y uno.

Para unir los nodos, primero se hizo un criterio para ubicar los *nodos madre*. En esta práctica un *nodo madre* es aquel que permite que algún otro nodo se conecte con éste si la distancia entre ellos es menor a una constante definida en 3. Se propuso que si la cantidad x de nodos totales en el grafo cumple que $5a < x \leq 5(a+1)$, donde a es un entero positivo, entonces ese grafo contaría con $a+1$ *nodos madre*, los cuales se eligen de manera aleatoria de entre los nodos que inicialmente se crearon.

Una vez que se ubican los *nodos madre*, se procede a unirlos de la manera que ya se mencionó: si un nodo se encuentra a una distancia menor a 3 del *nodo madre*, entonces esos nodos se conectarán mediante una arista.

4. Implementación del algoritmo BFS

Para mayor comodidad visual, en los grafos se agregaron colores a los nodos dependiendo de qué tan alejados estuviera del nodo inicial, es decir del nivel de búsqueda que tuvieran al llevar a cabo el BFS.

La figura 1 da un ejemplo de esta visualización. Cabe resaltar que el nodo inicial se toma de manera aleatoria, siendo representado con color negro. Los nodos que no están conectados son debido a que no fueron alcanzados por el radio de ningún *nodo madre*.

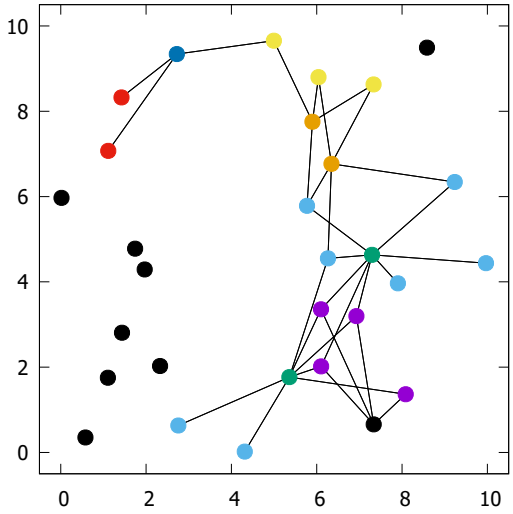


Figura 1: Ejemplo de un grafo con nivel 7.

Los colores de los nodos se detallan en el cuadro 1. Si el nivel llega a nueve, los colores se repiten.

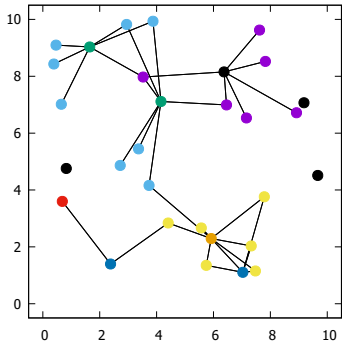
Cuadro 1: Asignación de números y niveles de búsqueda

Nivel	Color
1	morado
2	verde
3	celeste
4	naranja
5	amarillo
6	azul
7	rojo
8	negro

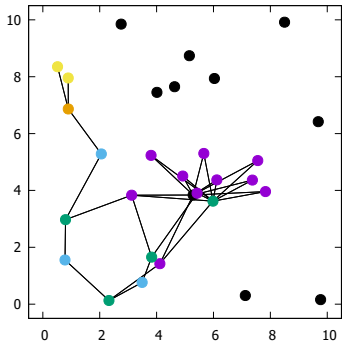
Al imprimir la función `G.bfs`, la cual, en el código creado en `python`, contiene el algoritmo BFS, arroja como resultado dos cosas:

- La coordenada de cada nodo junto con el nivel que le corresponde.
- El orden en que fueron siendo explorados y almacenados en el vector *output* (justo como se mencionó en la introducción de esta práctica).

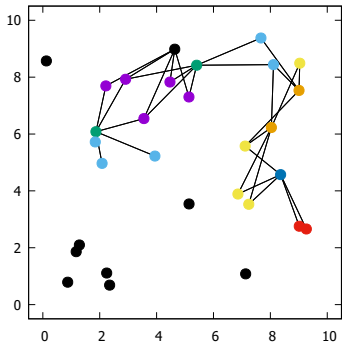
La figura 2 muestra más ejemplos de grafos creados con este código, todos con diferentes niveles de búsqueda.



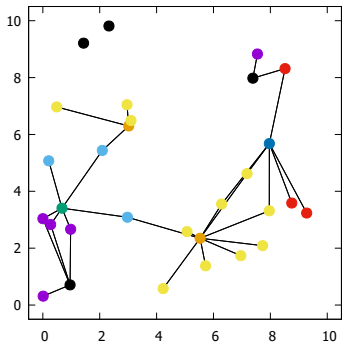
(a) Nivel 7.



(b) Nivel 5.



(c) Nivel 7



(d) Nivel 9.

Figura 2: Diferentes resultados.