Memoria P1

Inteligencia Artificial

Grupo 2361

Autores: Guillermo Rodríguez y Javier Muñoz

**Ejercicio 1:** Distancia Coseno (1.5 ptos)

**1.1)**

**1.2)**

**1.3)**

**1.4)**

**Ejercicio 2:** Raíces de una función (1.5 ptos)

**2.1)**

**2.2)**

**2.3)**

**2.3.1)**

**Ejercicio 3:** Combinación de listas (1 pto)

**3.1)**

**3.2)**

**3.3)**

**Ejercicio 4:** Árboles de verdad en lógica proposicional (5 ptos)

**4.1)**

**4.2)**

**Ejercicio 5:** Búsqueda en anchura (1 pto)

**5.1)**

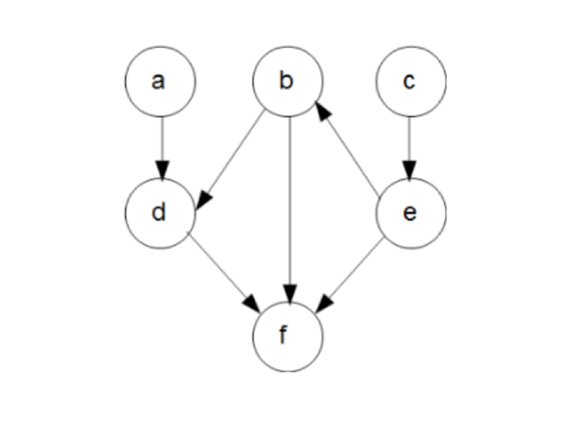
* Grafos especiales

**Explicación:**

En este caso el Algoritmo de búsqueda en anchura es tiene los siguientes pasos:

Si nuestra búsqueda comienza desde A, primero descubrirá los nodos más cercanos que tiene, en nuestro caso serían B, C y D, después visitará B, y empieza a descubrir los nodos que no están descubiertos todavía, en este caso E, posteriormente visita C y busca nodos que todavía no están descubiertos, como el único al que tiene acceso es E y ya está visitado, C es cerrado, por último en este nivel visitamos D, buscamos nuevos nodos, al no encontrar nada, lo cerramos, lo que hacemos en pasar a E. Una vez en E, buscamos nuevos nodos. Al no encontrar ninguno, cerramos E y terminamos de buscar nodos.

* Caso típico (grafo dirigido)



**Explicación:**

Para este grafo (el de el enunciado) el cual es dirigido el algoritmo varía un poco del que se usan en grafos no dirigidos.

Empezamos por ejemplo en C, y de ahí descubrimos los nodos a los que se pueden acceder, en este caso E, una vez que E es visitado, descubrimos los nodos a los que se pueden acceder desde él, que son B y F, visitamos B por ejemplo. Descubrimos los nodos que son accesibles desde B, en este caso D, el cual marcamos como descubierto. Luego vamos a F, que está al mismo nivel que B, lo marcamos como visitado y descubrimos los nodos accesibles desde él, en este caso ninguno. Ahora con marcamos D como visitado e intentamos acceder a algún nodo. Al no encontrar ninguno, buscamos otro nodo que no esté visitado, en este caso A, lo visitamos e intentamos descubrir otros nodos. Al no haber mas nodos que descubrir nuestra búsqueda a finalizado.

* Casi típico distinto al anterior

**Explicación:**

En este grafo empezaremos a recorrerlo desde A, marcando visitado como A. Una vez que hemos visitado A, descubrimos los nodos adyacentes, que son B y D. Visitamos B y descubrimos los nodos adyacentes, en este caso C. Por otra parte, tenemos D, de los cuales descubrimos los nodos F y G. Pasamos a C, visitándolo, y descubriendo los nodos a los que podemos acceder, en este caso E. Luego pasamos a los dos nodos descubiertos desde D, que son F y G, primero con F lo visitamos e intentamos descubrir mas nodos, a no encontrarlo vamos a G, lo visitamos e intentamos explorar mas nodos, al no encontrar ninguno pasamos a E. Marcamos E como visitado, al no poder descubrir mas nodos, terminamos la BFS.

**5.2)** Escribir un algoritmo correspondiente al BFS.

El pseudocódigo de BFS podría explicarse como:

Sea un Grafo G(V, E), podemos representar el BFS como:

BFS(G, v)

Marcar(v) Poner (v) en una COLA  
 Mientras la COLA no sea vacía hacer Quitar el primer elemento w de la COLA

Para cada vértice x adyacente a w hacer

si x no esta marcado

entonces marcar x

poner x en la COLA

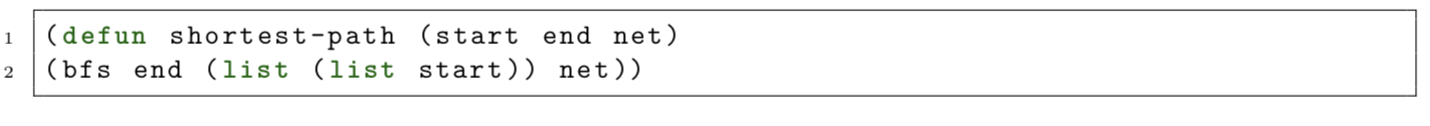
Fuente: <https://www.cs.us.es/cursos/cc-2009/material/bfs.pdf>

**5.3)** Una vez que entendemos como ha funcionado, pasamos a pegar el código y entenderlo para el apartado 4.

**5.4)** Comentamos el código que se nos ha sido proporcionado.

****

**5.5)** El código proporcionado es el siguiente:



Por lo que hemos intuido, este código lo que hace es llamar a la función de bfs principal, la cual realiza todas las operaciones de búsqueda.

Cuando usamos esta función, lo que estamos haciéndole saber a la función bfs es saber cual es su principio, y cual es su final y cual es el grafo donde tiene que descubrir.

Esto hace que muchas veces en vez de buscar por el primer nodo pasado en la lista ya sabe por donde empezar a buscar, de tal manera que muchas veces las búsquedas se hacen mas rápidas por el propio hecho de saber donde buscar.

**5.6)**

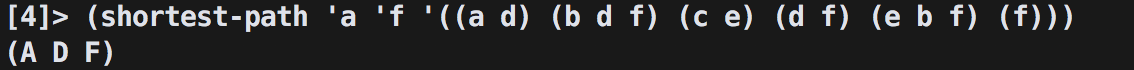
El código funciona para la siguiente entrada tal que:

Lo primero que se hace es llamar a la función de shortest-path, a la cual le pasamos los argumentos de nodo de inicio, que es el nodo desde donde se empieza a buscar, nodo final, que es el nodo que queremos buscar, y la lista de listas que representan las conexiones entre los nodos.

Sabiendo esto, shortest-path llamará a bfs, pasándole el nodo final, a encontrar, el nodo donde tiene que empezar a buscar, siendo este list (list start), y todos los nodos conectados entre si.

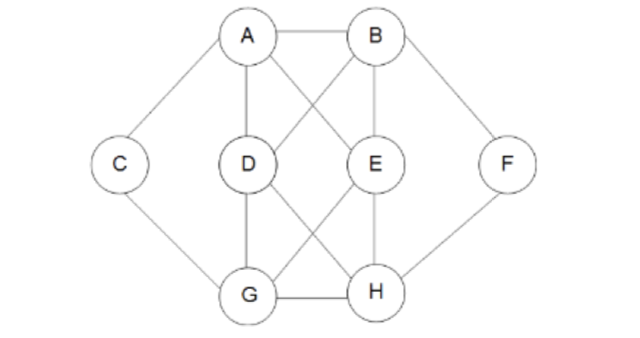
Como el argumento del nodo de inicio a partir de donde se empieza a buscar está metido ya en la queue de bfs, le indicamos cual es el primer nodo de la lista, y por tanto a partir de que nodo tiene que empezar a descubrir los demás nodos. Una vez que nuestra función bfs sabe esto, realiza exactamente el mismo proceso que hemos comentado en el apartado previo.

Obteniendo la siguiente salida:



**5.7)** Para representar el siguiente grafo, tenemos que analizar las conexiones que tiene cada uno de los nodos que lo componen con sus adyacentes, el resultado fue l siguiente:

Para el grafo:



Creamos la siguiente llamada:

(shortest-path 'a 'f '((c g a)(a c d e b)(b a f e d)(f b h)(d a b h g)(e b h a g)(g c d h e)

(h g d e f)).

El resultado que hemos obtenido es inconcreto, creemos que es dado a que al haber ciclos en este grafo, la función entra en un bucle infinito en el que no para de añadir nodos a la cola para explorar de forma que nunca “termina” de analizar todos.

**5.8)** Implementamos las funciones que se nos piden, en este caso son una nueva versión de shortest-path llamada shortest-path-improved y bfs, llamada bfs-improved.