Búsqueda en amplitud (BFS)

David Ricardo Arteta, Jaime Dorian Lopez

***Resumen*—Se presenta y describe, de manera simple, el seudocódigo de la búsqueda en amplitud, sus características y funcionamiento.**

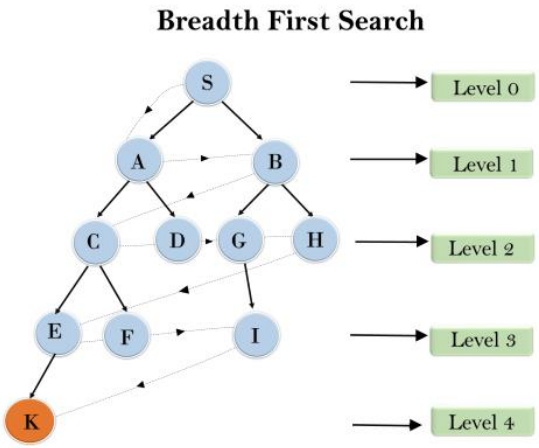
***Abstract*-- It presents and describes, in a simple way, the pseudo-code of the Breadth First Search or BFS, its characteristics and operation.**

I. Introducción

Una búsqueda en amplitud o BFS por sus siglas en ingle (Breadth First Search), es un algoritmo de búsqueda por medio del cual se recorren los nodos de un grafo comenzando por el nodo raíz o padre (o eligiendo uno en el caso de un grafo) para luego explorar todos los vecinos de este nodo, siguiente a esto, para cada uno de los vecinos del nodo se exploran sus respectivos vecinos adyacentes ; se continua ejecutando este proceso hasta recorrer todo el grafo teniendo en cuenta que si durante el proceso se encuentra el nodo que buscamos antes de recorrer todo el grafo , se concluye la búsqueda .
Entre las principales aplicaciones de la BFS podemos encontrar:

- Encontrar el camino más corto entre 2 nodos, medido por el número de nodos conectados
- Probar si un grafo de nodos es bipartito (si se puede dividir en 2 conjuntos)
- Encontrar el árbol de expansión mínima en un grafo no ponderado
- Hacer un Web Crawler
- Sistemas de navegación GPS, para encontrar localizaciones vecinas.

Representación grafica de BFS



(Fig. 1)

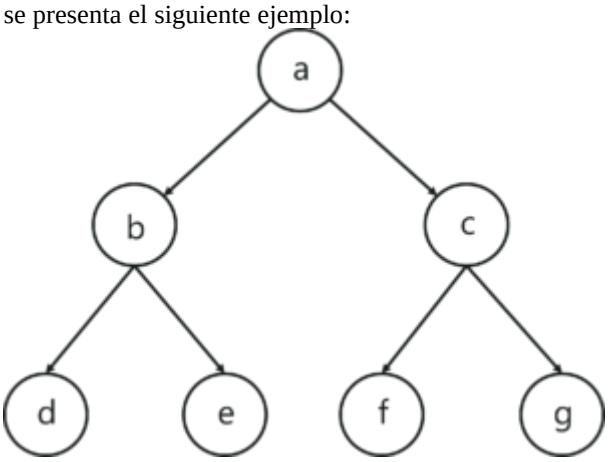
II. Pseudo Código

```
BFS (Grafo, verticeOrigen(V1), verticeBuscado(V2))
    Cola={V1}
    Marcar V1 como visitado

  Mientras Cola no este vacía:
        V =cola. removeFirst ()

      If  V es V2:
          Nodo encontrado

     Obtener "L" lista de vecinos de V
     Para todo vértice en L:
          Marcarlo como visitado
          Meterlo a la Cola

  Nodo no encontrado
```

Para llevar a cabo el procedimiento BFS se va a necesitar pasar el grafo, el vértice origen y el vértice a buscar como parámetros, seguido a esto se declara una cola compuesta por el vértice origen el cual se marca como visitado, el proceso va a continuar mientras la cola no este vacía, lo que significa que siempre va a haber vecinos del vértice a visitar ; luego se extrae el primer componente de la cola y se compara con el vértice a buscar , de ser el mismo la búsqueda se da por concluida, de lo contrario se obtiene una lista con los vecinos del vértice los cuales serán marcados como visitados y se agregaran a la cola para continuar con el proceso, en caso de recorrer todo el grafo sin encontrar el vértice a buscar el proceso de da por terminado sin coincidencias.

III. Funcionamiento

Para intentar dar claridad sobre el funcionamiento del código, se presenta el siguiente ejemplo:



Sea la cola C iniciada con el nodo raíz:

C={a}
Nodo_p= a (nodo primero sacado de la cola)
Vecinos_p={b,c}
C= C U Vecinos_p ({b,c})

C={b,c}
Nodo_p= b
Vecinos_p={d,e}
C= C U Vecinos_p

C={c,d,e}
Nodo_p= c
Vecinos_p={f,g}
C= C U Vecinos_p

C={d,e,f,g}
……..

Vale destacar que la principal característica que diferencia a la BFS de DFS en el uso de las estructuras auxiliares, mientras la DFS utiliza una estructura tipo LIFO como es la pila, la BFS se apoya en una estructura tipo FIFO como lo es la cola.

IMPLEMENTACION EN PYTHON

Por último, se deja un ejemplo de BFS implementado en python:

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = [] # List for visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node): #function for BFS
  visited.append(node)
  queue.append(node)

  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
            queue.append(neighbour)

      # Driver Code
print("Following is the Breadth-First Search")
      bfs(visited, graph, '5')    # function calling
```