**Breadth First Search**

**Tabish, CSE-09/16**

**Department of Computer Science & Engineering, NIT Srinagar**

**Introduction to the Algorithm:**

BFS is a graph traversing algorithm where we start traversing the graph from a selected node and explore the graph in a layer wise manner until the required element is located within the graph. At a given time, only nodes at the same level are looked at by the algorithm. Thus, all node at level *n* are looked at before moving on to any level *n+1* or greater. Thus, BFS returns the shortest length path from the source node to the destination (search) node.

As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

1. Is current node the node being searched for? If yes, return the node.
2. Move to a node at the same level. Repeat step 1 if a node is explorable, otherwise step 3.
3. Move to the next layer and repeat from step 1

Eliminating looping:

A graph can contain cycles, which may bring you to the same node again while traversing the graph. This would lead to an infinite loop while searching the graph. To avoid such situations, we make use of Boolean array to mark the various nodes on the graph as either visited or yet to be visited. Any visited nodes will not be explored again.

To make this process easy, use a queue to store the node and mark it as 'visited' until all its neighbours (vertices that are directly connected to it) are marked. The queue follows the First-In First-Out (FIFO) queuing method, and therefore, the neighbors of the node will be visited in the order in which they were inserted in the node i.e. the node that was inserted first will be visited first, and so on.

**Algorithm:**

procedure BFS(G,s,destination)

for each vertex v ∈ V[G] do

explored[v] ← false

d[v] ← ∞

end for

explored[s] ← true

d[s] ← 0

Q:= a queue data structure, initialized with s

while Q != φ do

u ← remove vertex from the front of Q

if u.data == destination:

return u

for each v adjacent to u do

if not explored[v] then

explored[v] ← true

d[v] ← d[u] + 1

insert v to the end of Q

end if

end for

end while

end procedure

**Code:**

from collections import defaultdict

class Graph:

    def \_\_init\_\_(self):

        self.graph = defaultdict(list)

    def addEdge(self,u,v):

        self.graph[u].append(v)

    def BFS(self, s, destination):

        visited = [False] \* (len(self.graph))

        queue = []

        queue.append(s)

        visited[s] = True

        while queue:

            s = queue.pop(0)

            print (s, end = " ")

if s == destination:

print("Found")

   return

            for i in self.graph[s]:

                if visited[i] == False:

                    queue.append(i)

                    visited[i] = True

g = Graph()

g.addEdge(0, 1)

g.addEdge(0, 2)

g.addEdge(1, 2)

g.addEdge(2, 0)

g.addEdge(2, 3)

g.addEdge(3, 3)

g.addEdge(3,4)

g.addEdge(3,5)

g.addEdge(5,2)

g.addEdge(4,6)

g.addEdge(6,6)

print ("Following is Breadth First Traversal")

g.BFS(0,6)

**Output Observed for different Inputs:**

* **Input passed:**

g.BFS(0,6)

* **Output Obtained:**

Following is Breadth First Traversal

0 1 2 3 4 5 6 Found

* **Input passed:**

g.BFS(0,11)

* **Output Obtained:**

Following is Breadth First Traversal

0 1 2 3 4 5 6 Not Found

* **Input passed:**

g.BFS(5,6)

* **Output Obtained:**

Following is Breadth First Traversal

5 2 0 3 1 4 6 Found