

# BFS Report

Dipanshu

September 2022

## 1 Introduction

Traversal means that visiting all the nodes of a graph which can be done through Depth-first search or Breadth-first search in python. Depth-first traversal or Depth-first Search is an algorithm to look at all the vertices of a graph or tree data structure. Here we will study what depth-first search in python is, understand how it works with its bfs algorithm, implementation with python code, and the corresponding output to it.

### **What is Depth First Search?**

What do we do once have to solve a maze? We tend to take a route, keep going until we discover a dead end. When touching the dead end, we again come back and keep coming back till we see a path we didn't attempt before. Take that new route. Once more keep going until we discover a dead end. Take a come back again... This is exactly how Depth-First Search works.

The Depth-First Search is a recursive algorithm that uses the concept of backtracking. It involves thorough searches of all the nodes by going ahead if potential, else by backtracking. Here, the word backtrack means once you are moving forward and there are not any more nodes along the present path, you progress backward on an equivalent path to seek out nodes to traverse. All the nodes are progressing to be visited on the current path until all the unvisited nodes are traversed after which subsequent paths are going to be selected.

### **DFS Algorithm**

Before learning the python code for Depth-First and its output, let us go through the algorithm it follows for the same. The recursive method of the Depth-First Search algorithm is implemented using stack. A standard Depth-First Search implementation puts every vertex of the graph into one in all 2 categories: 1) Visited 2) Not Visited. The only purpose of this algorithm is to visit all the vertex of the graph avoiding cycles.

The DSF algorithm follows as:

We will start by putting any one of the graph's vertex on top of the stack.

After that take the top item of the stack and add it to the visited list of the vertex.

Next, create a list of that adjacent node of the vertex. Add the ones which aren't in the visited list of vertexes to the top of the stack.

Lastly, keep repeating steps 2 and 3 until the stack is empty.

Now, knowing the algorithm to apply the Depth-First Search implementation in python, we will see how the source code of the program works.

Consider the following graph which is implemented in the code below:

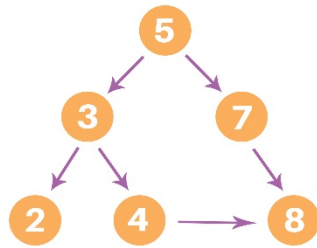
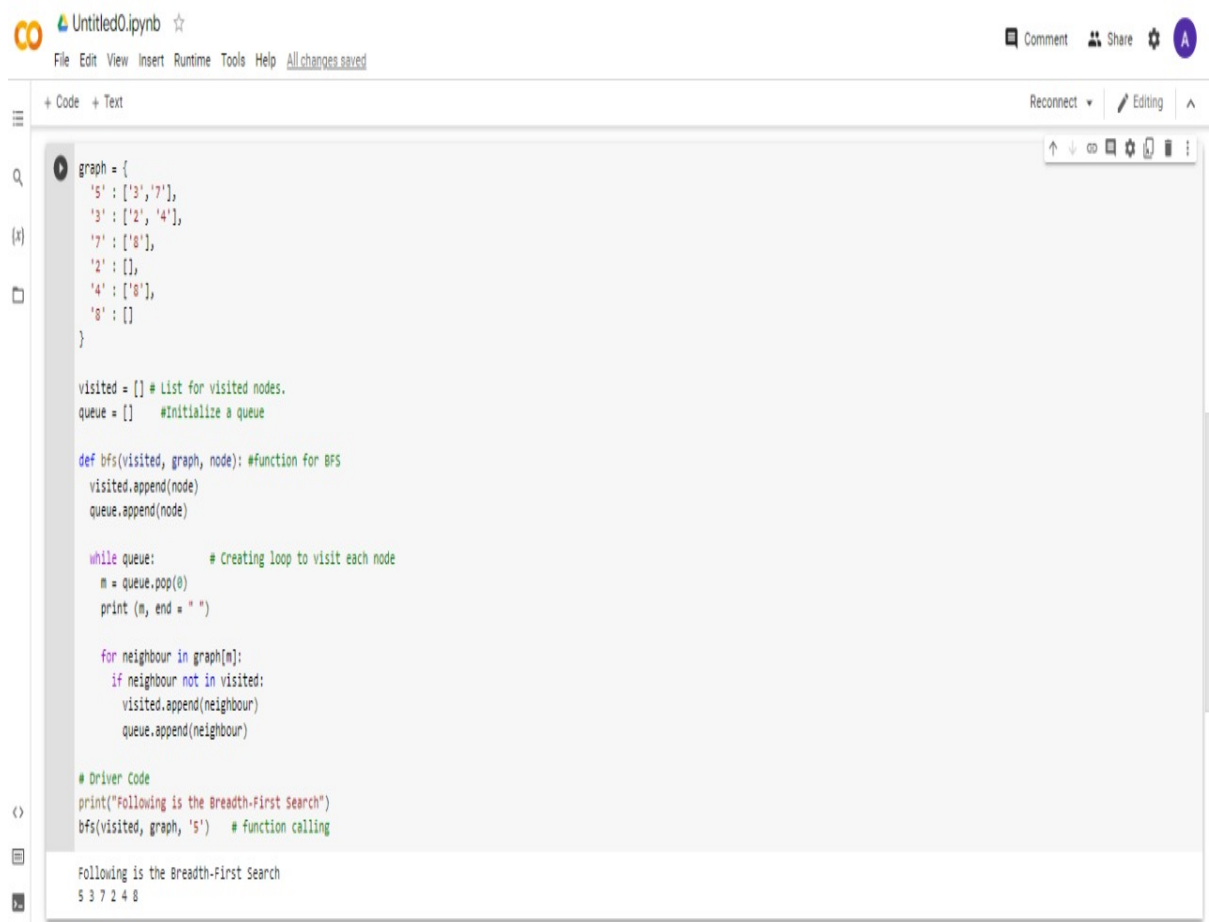


FIGURE 0



```
graph = {
    's': ['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, 's') # function calling
```

Following is the Breadth-First Search  
s 3 7 2 4 8

Figure 1: DFS

**Implementation** In the above code, first, we will create the graph for which we will use the depth-first search. After creation, we will create a set for storing the value of the visited nodes to keep track of the visited nodes of the graph.

After the above process, we will declare a function with the parameters as visited nodes, the graph itself and the node respectively. And inside the function, we will check whether any node of the graph is visited or not using the “if” condition. If not, then we will print the node and add it to the visited set of nodes.

Then we will go to the neighboring node of the graph and again call the DFS function to use the neighbor parameter.

At last, we will run the driver code which prints the final result of DFS by calling the DFS the first time with the starting vertex of the graph.