BFS Report

Dipanshu

September 2022

1 Introduction

BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.

What is Breadth-First Search?

As discussed earlier, Breadth-First Search (BFS) is an algorithm used for traversing graphs or trees. Traversing means visiting each node of the graph. Breadth-First Search is a recursive algorithm to search all the vertices of a graph or a tree. BFS in python can be implemented by using data structures like a dictionary and lists. Breadth-First Search in tree and graph is almost the same. The only difference is that the graph may contain cycles, so we may traverse to the same node again.

BFS Algorithm

Before learning the python code for Breadth-First and its output, let us go through the algorithm it follows for the same. We can take the example of Rubik's Cube for the instance. Rubik's Cube is seen as searching for a path to convert it from a full mess of colors to a single color. So comparing the Rubik's Cube to the graph, we can say that the possible state of the cube is corresponding to the nodes of the graph and the possible actions of the cube is corresponding to the edges of the graph.

As breadth-first search is the process of traversing each node of the graph, a standard BFS algorithm traverses each vertex of the graph into two parts: 1) Visited 2) Not Visited. So, the purpose of the algorithm is to visit all the vertex while avoiding cycles.

BFS starts from a node, then it checks all the nodes at distance one from the beginning node, then it checks all the nodes at distance two, and so on. So

as to recollect the nodes to be visited, BFS uses a queue.

The steps of the algorithm work as follow:

Start by putting any one of the graph's vertices at the back of the queue.

Now take the front item of the queue and add it to the visited list.

Create a list of that vertex's adjacent nodes. Add those which are not within the visited list to the rear of the queue.

Keep continuing steps two and three till the queue is empty.

Many times, a graph may contain two different disconnected parts and therefore to make sure that we have visited every vertex, we can also run the BFS algorithm at every node.

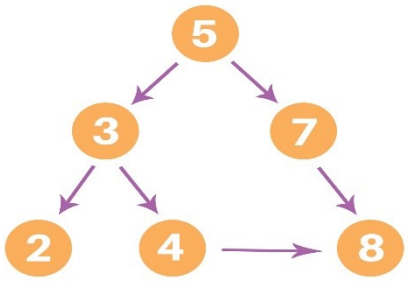

FIGURE 0

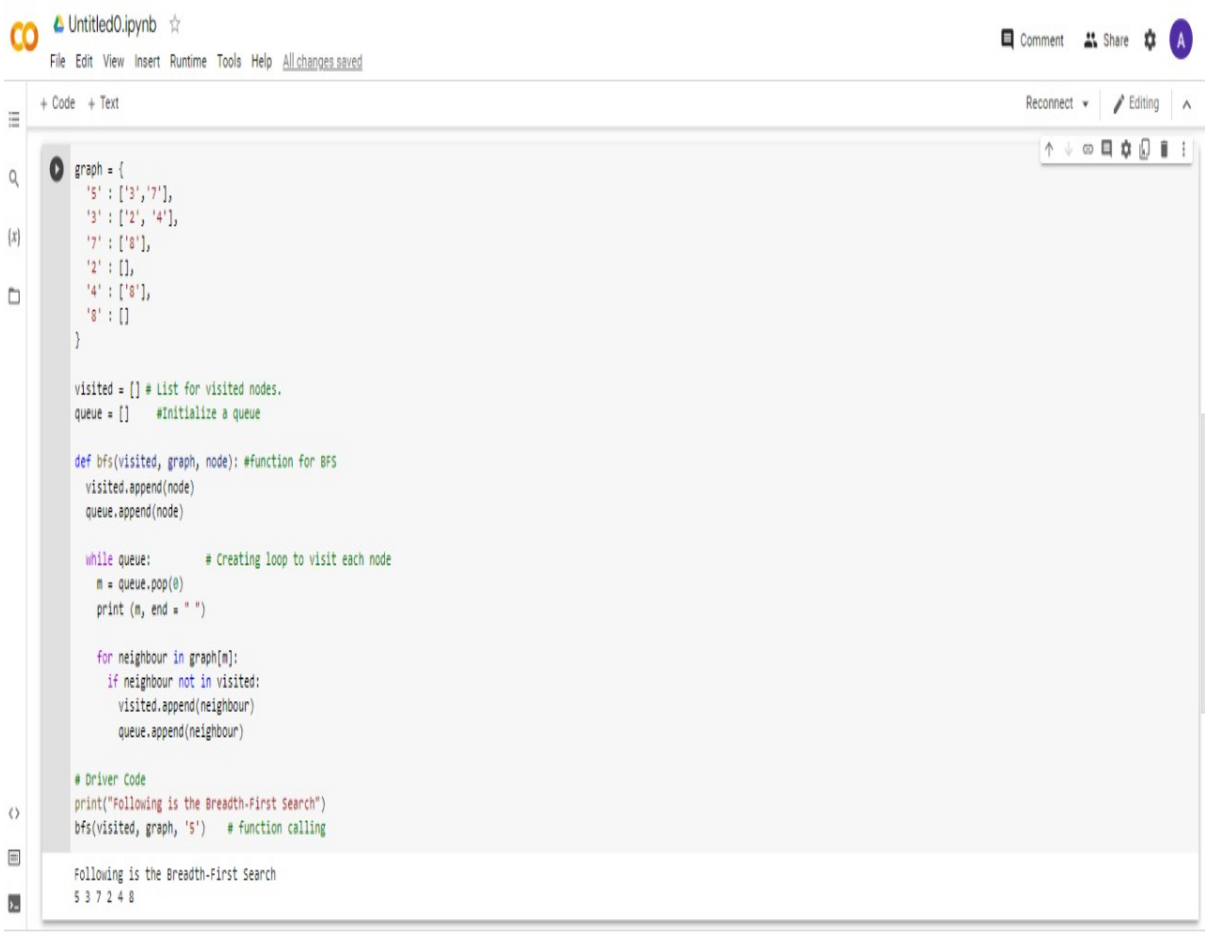
Untitled0.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Reconnect
Editing
graph = {
'5' : ['3','7'],
'3' : ['2', '4'],
'7' : ['8'],
'2' : [],
'4' : ['8'],
'8' : []
}
visited = [] # List for visited nodes.
queue = [] #Initialize a queue
def bfs(visited, graph, 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
Following is the Breadth-First Search
5 3 7 2 4 8

Figure 1: BFS

Implementation of code

In the above code, first, we will create the graph for which we will use the breadth-first search. After creation, we will create two lists, one to store the visited node of the graph and another one for storing the nodes in the queue.

After the above process, we will declare a function with the parameters as visited nodes, the graph itself and the node respectively. And inside a function, we will keep appending the visited and queue lists.

Then we will run the while loop for the queue for visiting the nodes and then will remove the same node and print it as it is visited.

At last, we will run the for loop to check the not visited nodes and then append the same from the visited and queue list.

As the driver code, we will call the user to define the bfs function with the first node we wish to visit.

The output of the above code will be as follow:

Following is the Breadth-First Search 5 3 7 2 4 8