

**“AZƏRBAYCAN HAVA YOLLARI” CJSC NATIONAL AVIATION ACADEMY**

**Individual Work № 6:**

**Topic: BFS and DFS algorithms in Python**

**Subject: Obyektyönümlü proqramlaşdırma**

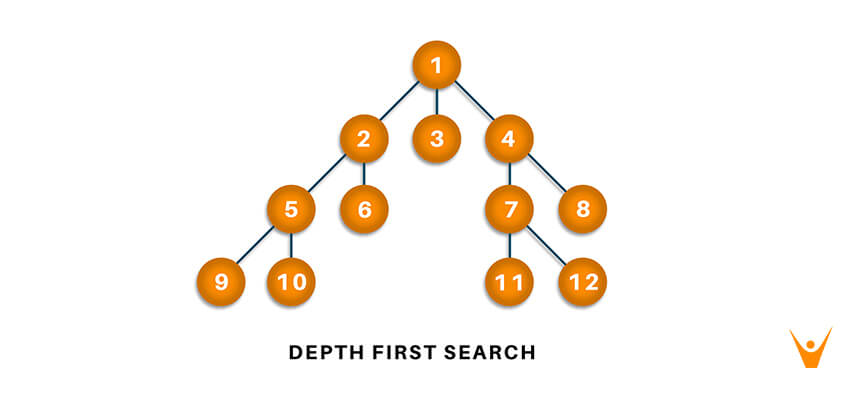
**Teacher: Mehemmed Shahmaliyev**

**Group: 1459i Student: Mammadov Ahmad**

**Date: Signature:M.A**

**Baku 2022**

# Depth First Search in Python (with Code) DFS Algorithm

Traversal means that visiting all the nodes of a graph which can be done through Depth-first search or [Breadth-first search in python](https://favtutor.com/blogs/breadth-first-search-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:

1. We will start by putting any one of the graph's vertex on top of the stack.
2. After that take the top item of the stack and add it to the visited list of the vertex.
3. 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.
4. Lastly, keep repeating steps 2 and 3 until the stack is empty.

### ****DFS pseudocode****

The pseudocode for Depth-First Search in python goes as below: In the init() function, notice that we run the DFS function on every node because many times, a graph may contain two different disconnected part and therefore to make sure that we have visited every vertex, we can also run the DFS algorithm at every node.

**DFS(G, u)**

**u.visited = true**

**for each v ∈ G.Adj[u]**

**if v.visited == false**

**DFS(G,v)**

**init() {**

**For each u ∈ G**

**u.visited = false**

**For each u ∈ G**

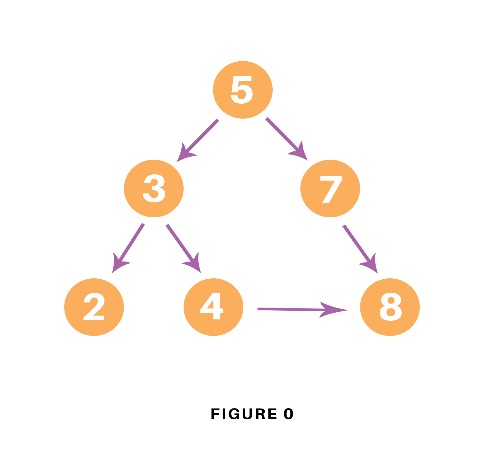
**DFS(G, u)**

**}**

## ****DFS Implementation in Python (Source Code)****

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:



# Using a Python dictionary to act as an adjacency list

graph = {

'5' : ['3','7'],

'3' : ['2', '4'],

'7' : ['8'],

'2' : [],

'4' : ['8'],

'8' : []

}

visited = set() # Set to keep track of visited nodes of graph.

**def** **dfs**(visited, graph, node): #function for dfs

**if** node **not** **in** visited:

**print** (node)

visited.add(node)

**for** neighbour **in** graph[node]:

dfs(visited, graph, neighbour)

# Driver Code**print**("Following is the Depth-First Search")

dfs(visited, graph, '5')

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.

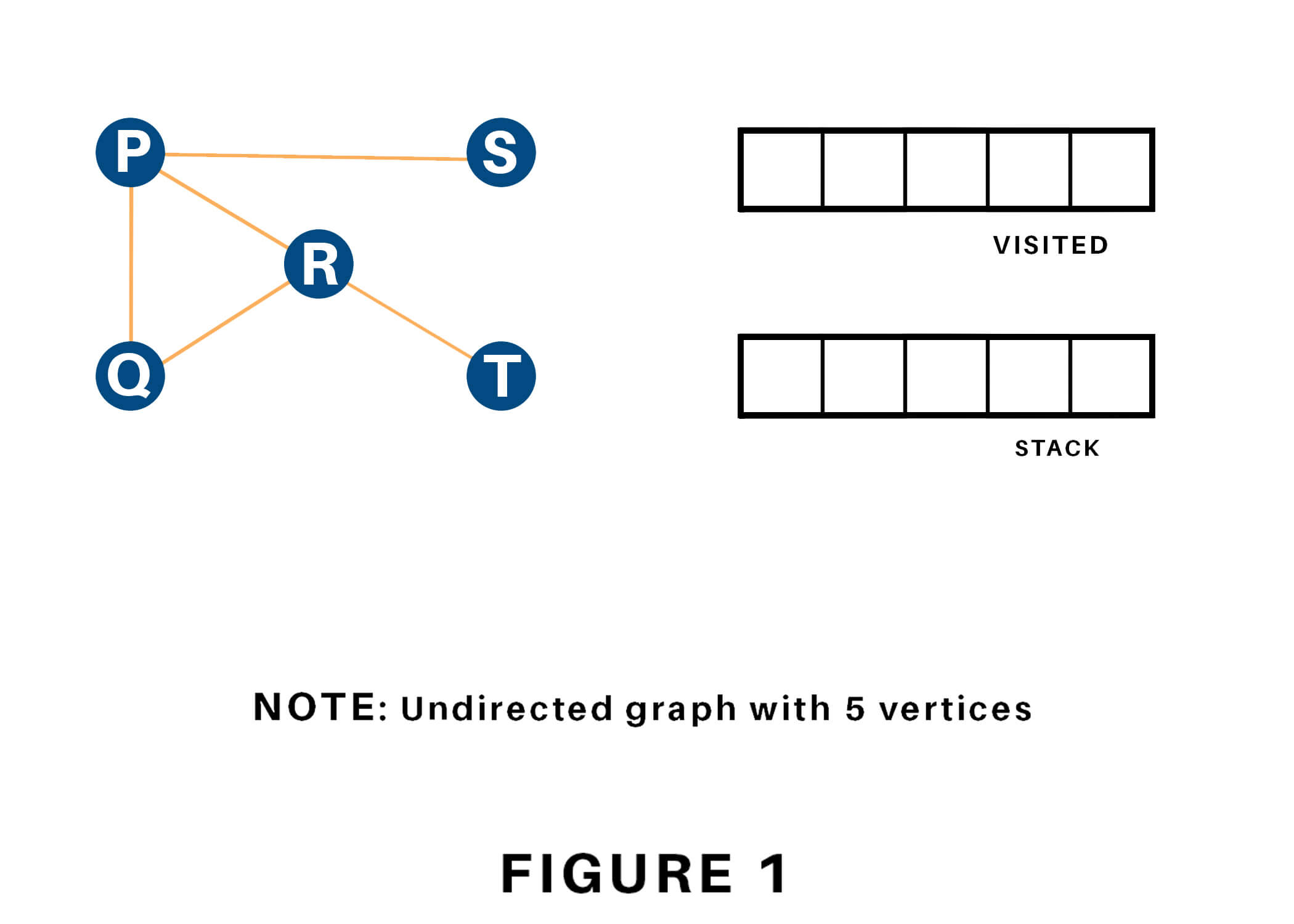
### ****Output****

The output of the above code is as follow:

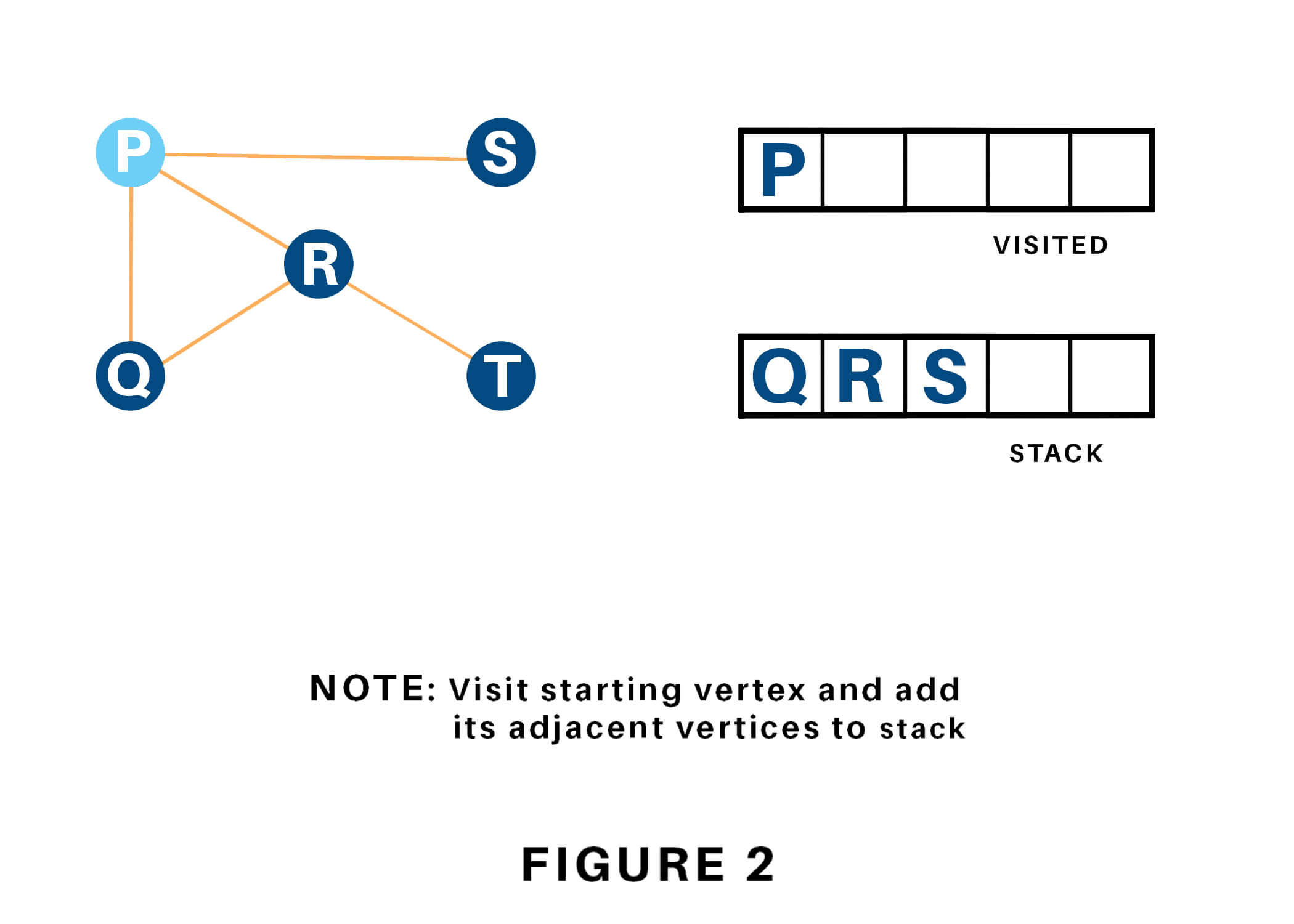
 Following is the Depth-First Search  
5 3 2 4 8 7

### ****Example****

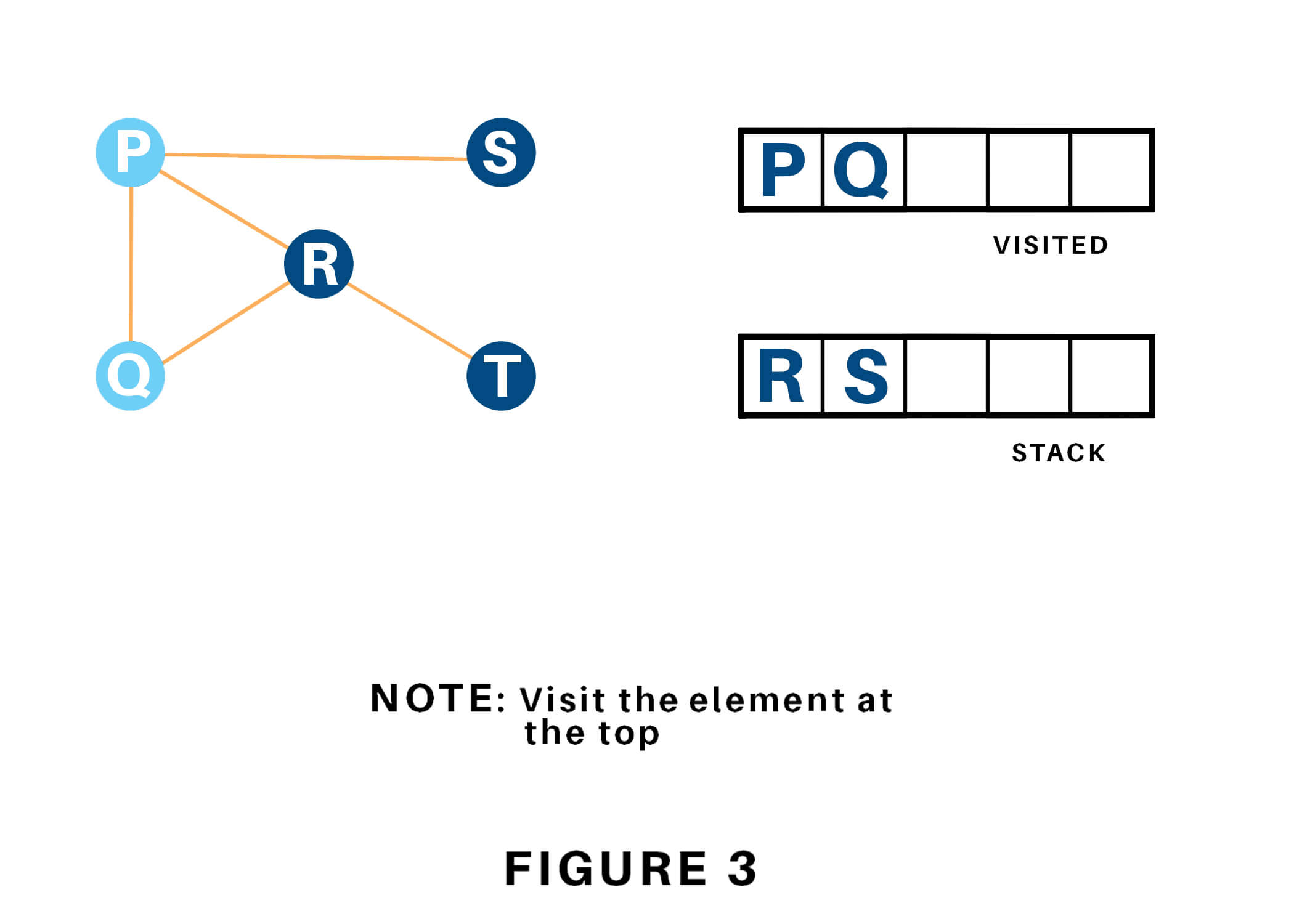
Let us see how the DFS algorithm works with an example. Here, we will use an undirected graph with 5 vertices.



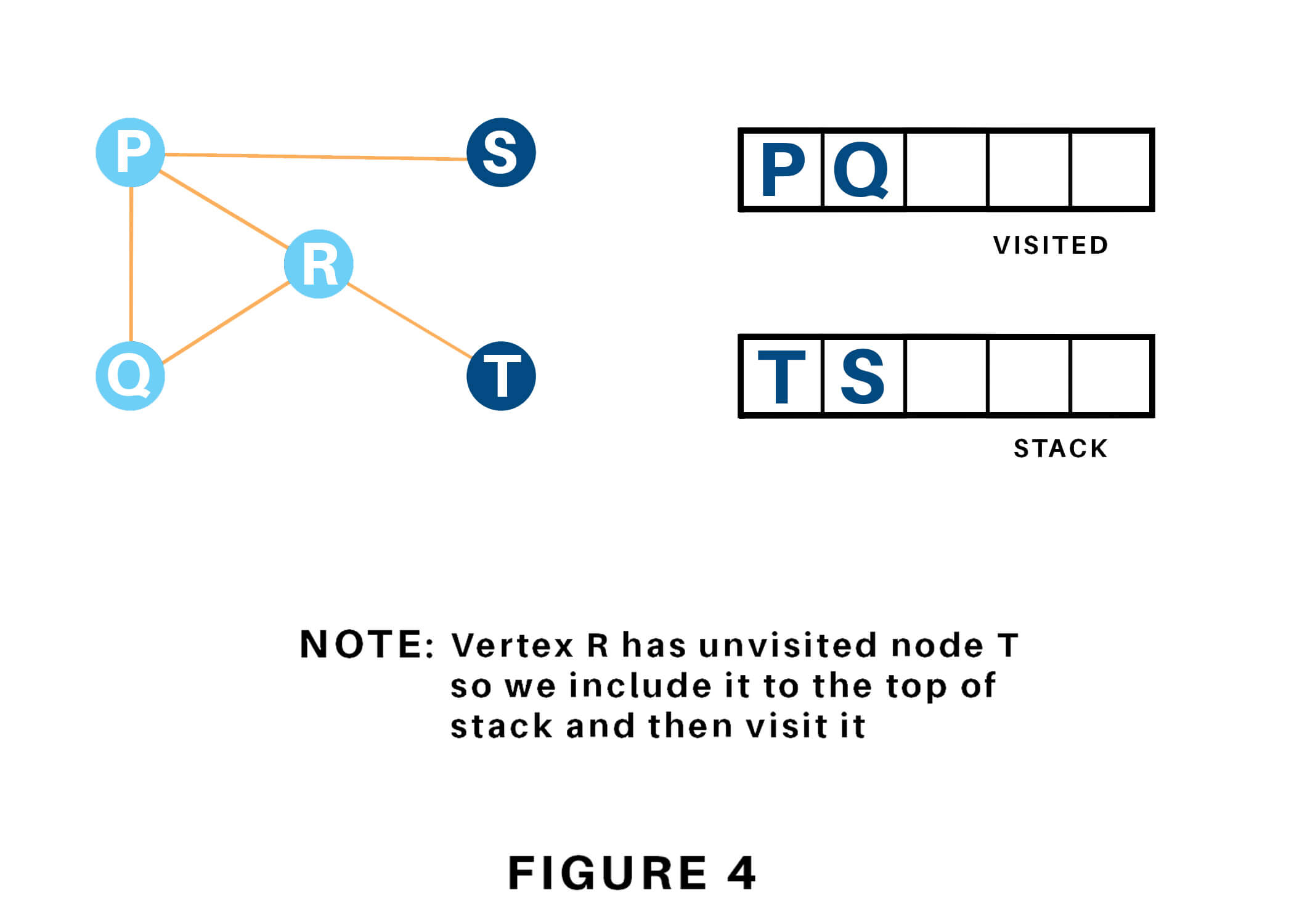
We begin from the vertex P, the DFS rule starts by putting it within the Visited list and putting all its adjacent vertices within the stack.

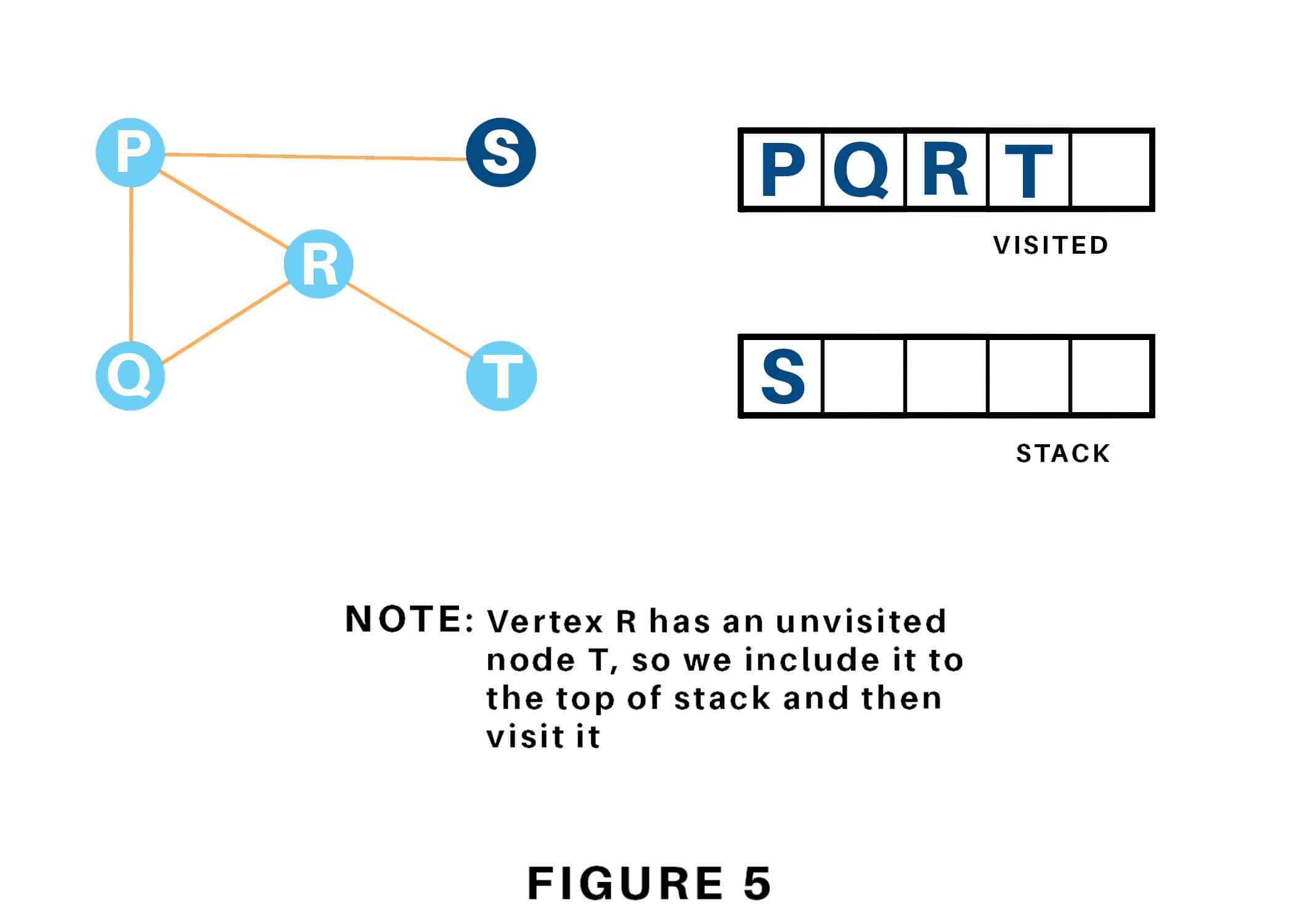


Next, we tend to visit the part at the highest of the stack i.e. Q, and head to its adjacent nodes. Since P has already been visited, we tend to visit R instead.

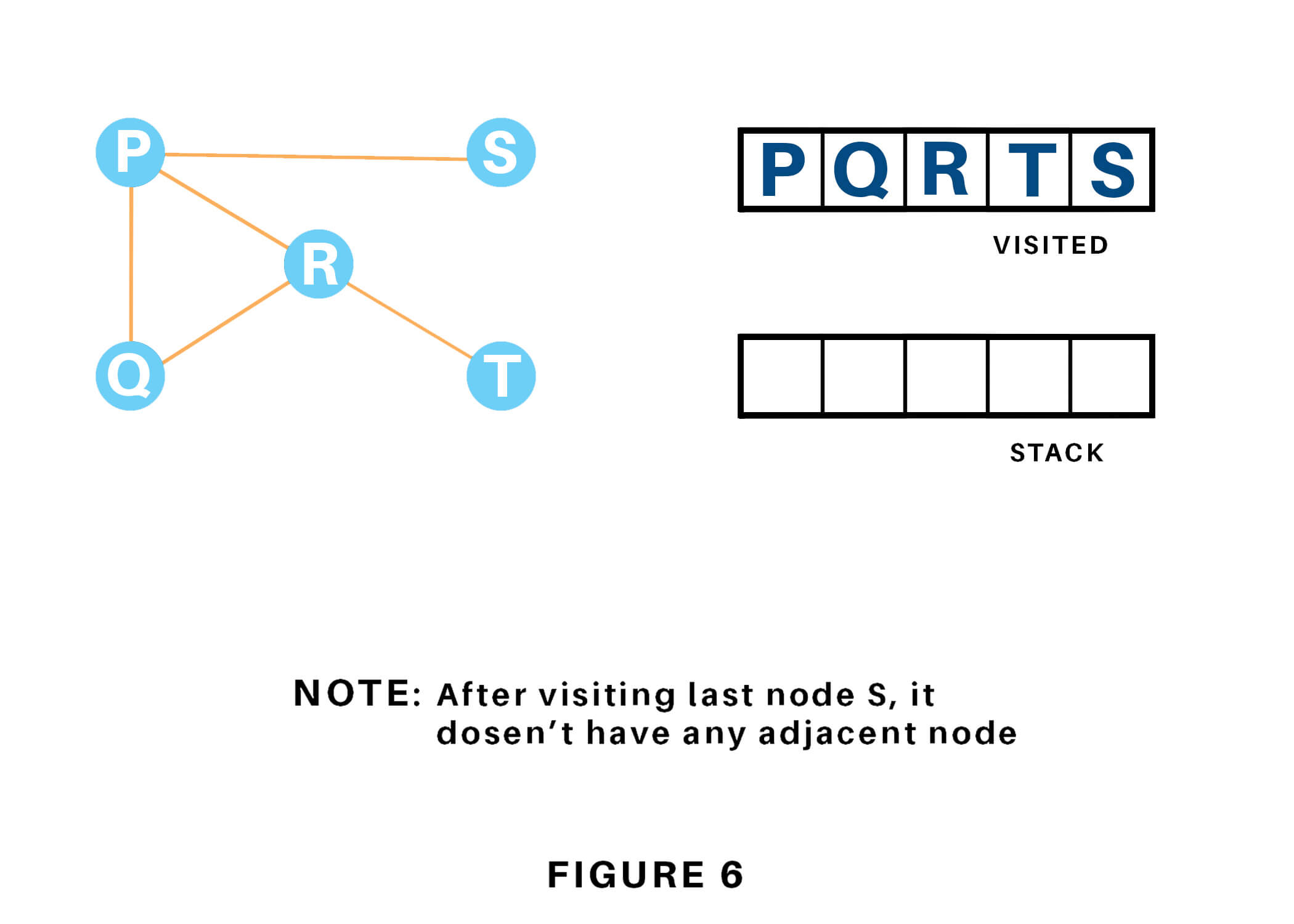


Vertex R has the unvisited adjacent vertex in T, therefore we will be adding that to the highest of the stack and visit it.





At last, we will visit the last component S, it does not have any unvisited adjacent nodes, thus we've completed the Depth First Traversal of the graph.



### ****Time Complexity****

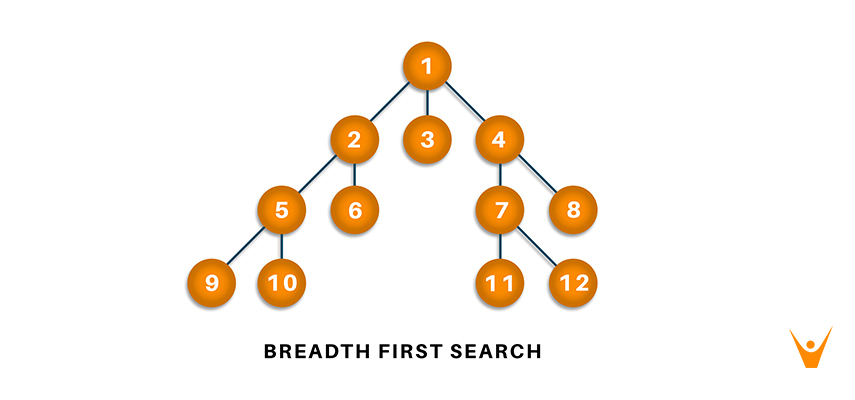
The time complexity of the Depth-First Search algorithm is represented within the sort of **O(V + E)**, where V is that the number of nodes and E is that the number of edges.  
  
The space complexity of the algorithm is **O(V).**

### ****Applications****

Depth-First Search Algorithm has a wide range of applications for practical purposes. Some of them are as discussed below:

1. For finding the strongly connected components of the graph
2. For finding the path
3. To test if the graph is bipartite
4. For detecting cycles in a graph
5. Topological Sorting
6. Solving the puzzle with only one solution.
7. Network Analysis
8. Mapping Routes
9. Scheduling a problem

# Breadth First Search in Python (with Code) BFS Algorithm



Breadth-first search and [Depth-first search in python](https://favtutor.com/blogs/depth-first-search-python) are algorithms used to traverse a graph or a tree. They are two of the most important topics that any new python programmer should definitely learn about. Here we will study what breadth-first search in python is, understand how it works with its algorithm, implementation with python code, and the corresponding output to it. Also, we will find out the application and uses of breadth-first search in the real world.

## 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:

1. Start by putting any one of the graph’s vertices at the back of the queue.
2. Now take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add those which are not within the visited list to the rear of the queue.
4. 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.

### BFS pseudocode

The pseudocode for BFS in python goes as below:

**create a queue Q**

**mark v as visited and put v into Q**

**while Q is non-empty**

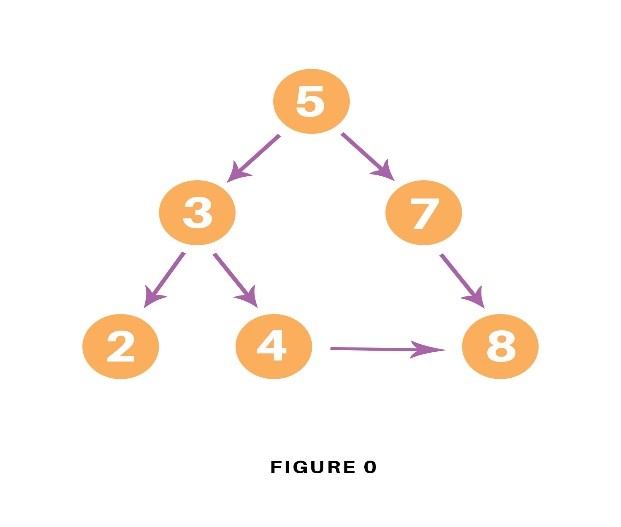
**remove the head u of Q**

**mark and enqueue all (unvisited) neighbors of u**

## BFS implementation in Python (Source Code)

Now, we will see how the source code of the program for implementing breadth first search in python.

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



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*

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.

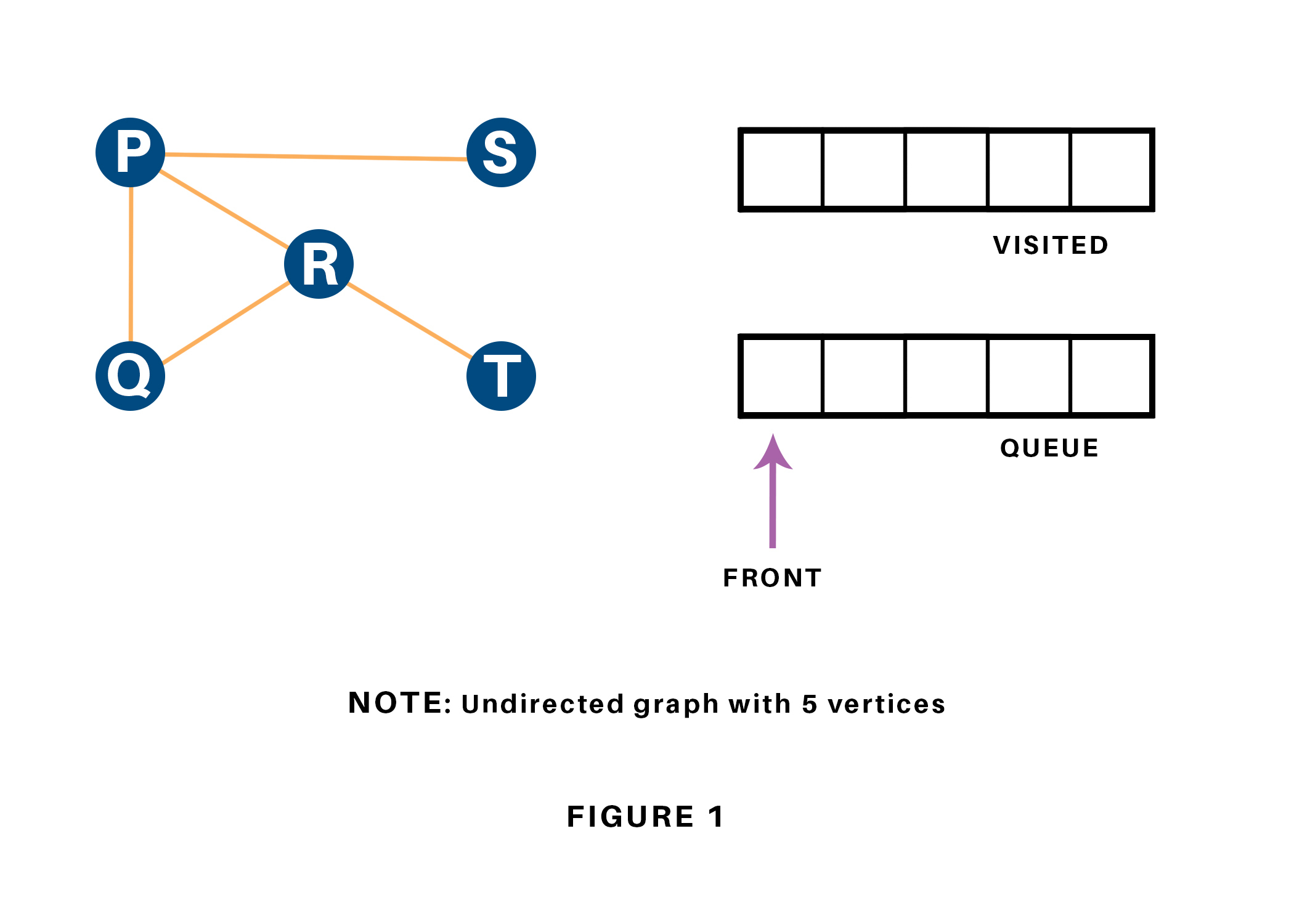
### Output

The output of the above code will be as follow:

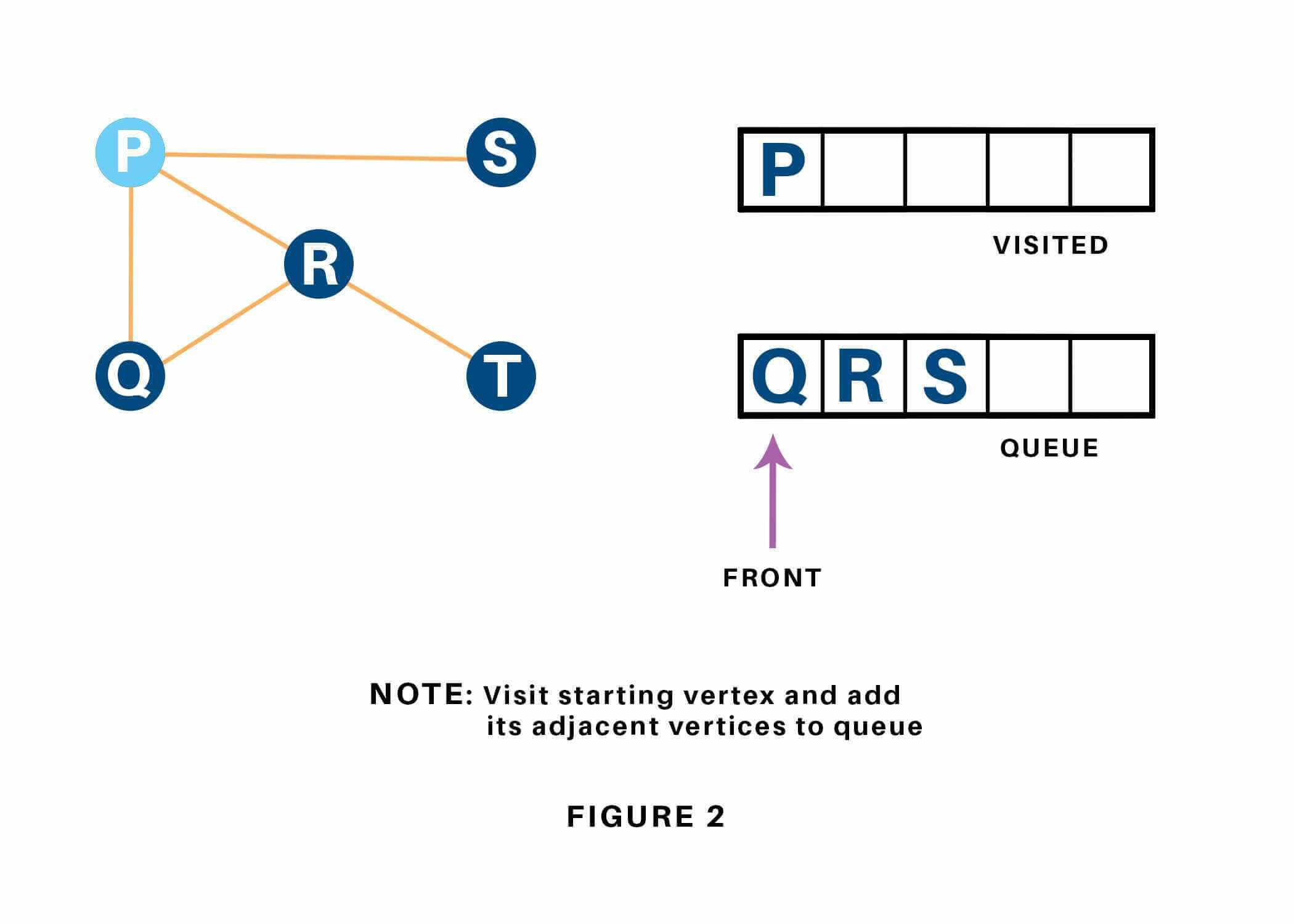
 Following is the Breadth-First Search  
 5 3 7 2 4 8

### Example

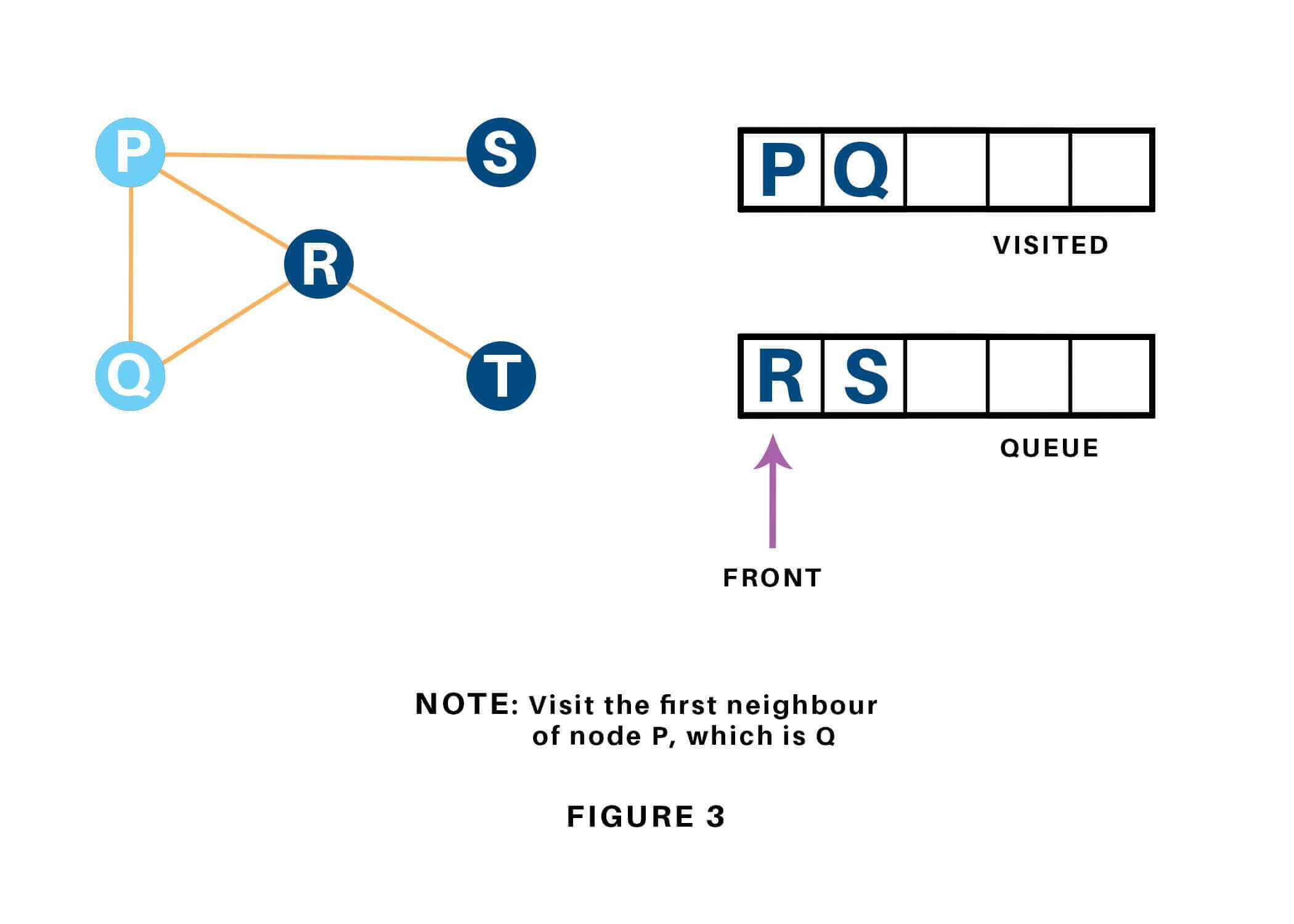
Let us see how this algorithm works with an example. Here, we will use an undirected graph with 5 vertices.



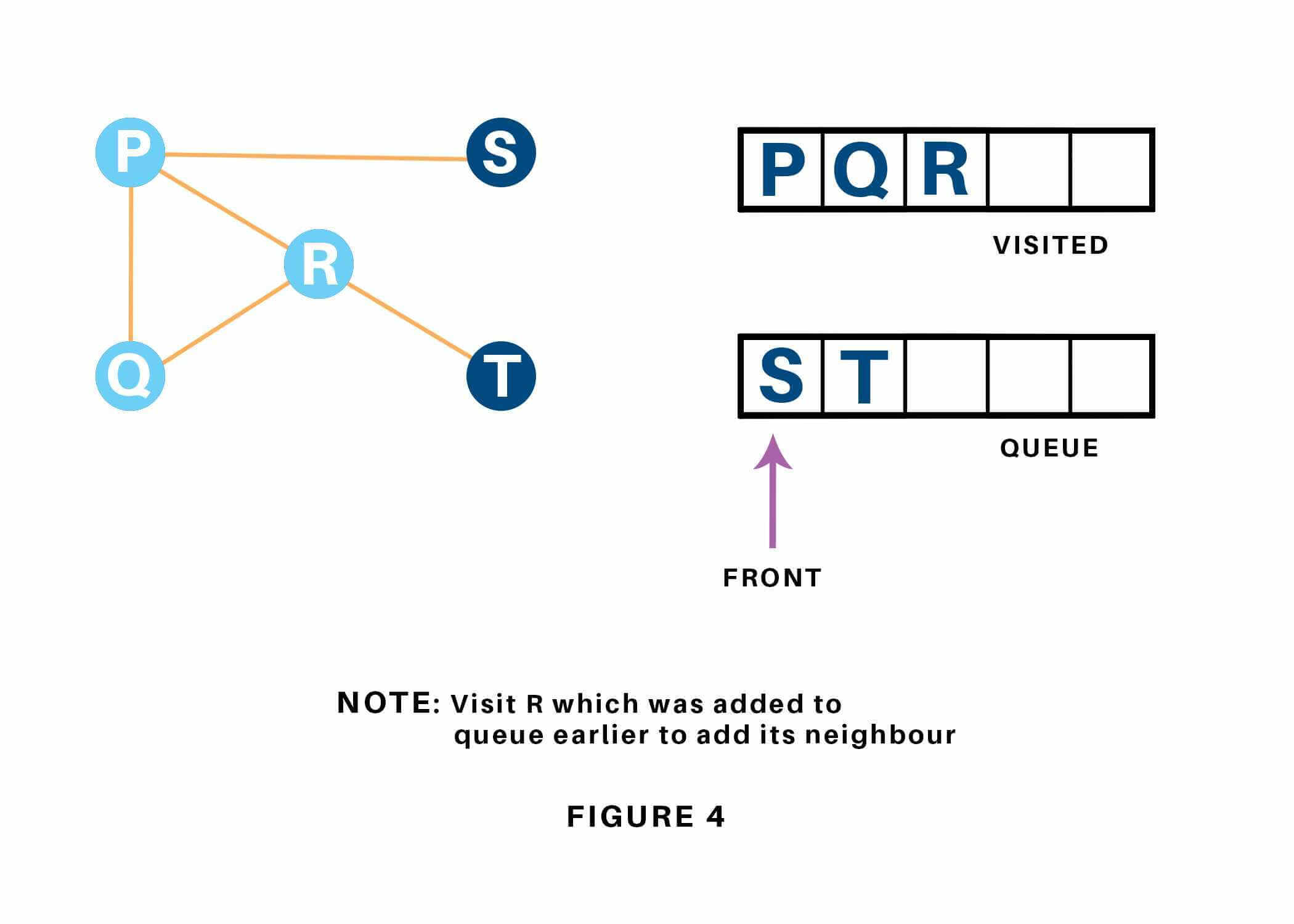
We begin from the vertex P, the BFS algorithmic program starts by putting it within the Visited list and puts all its adjacent vertices within the stack.

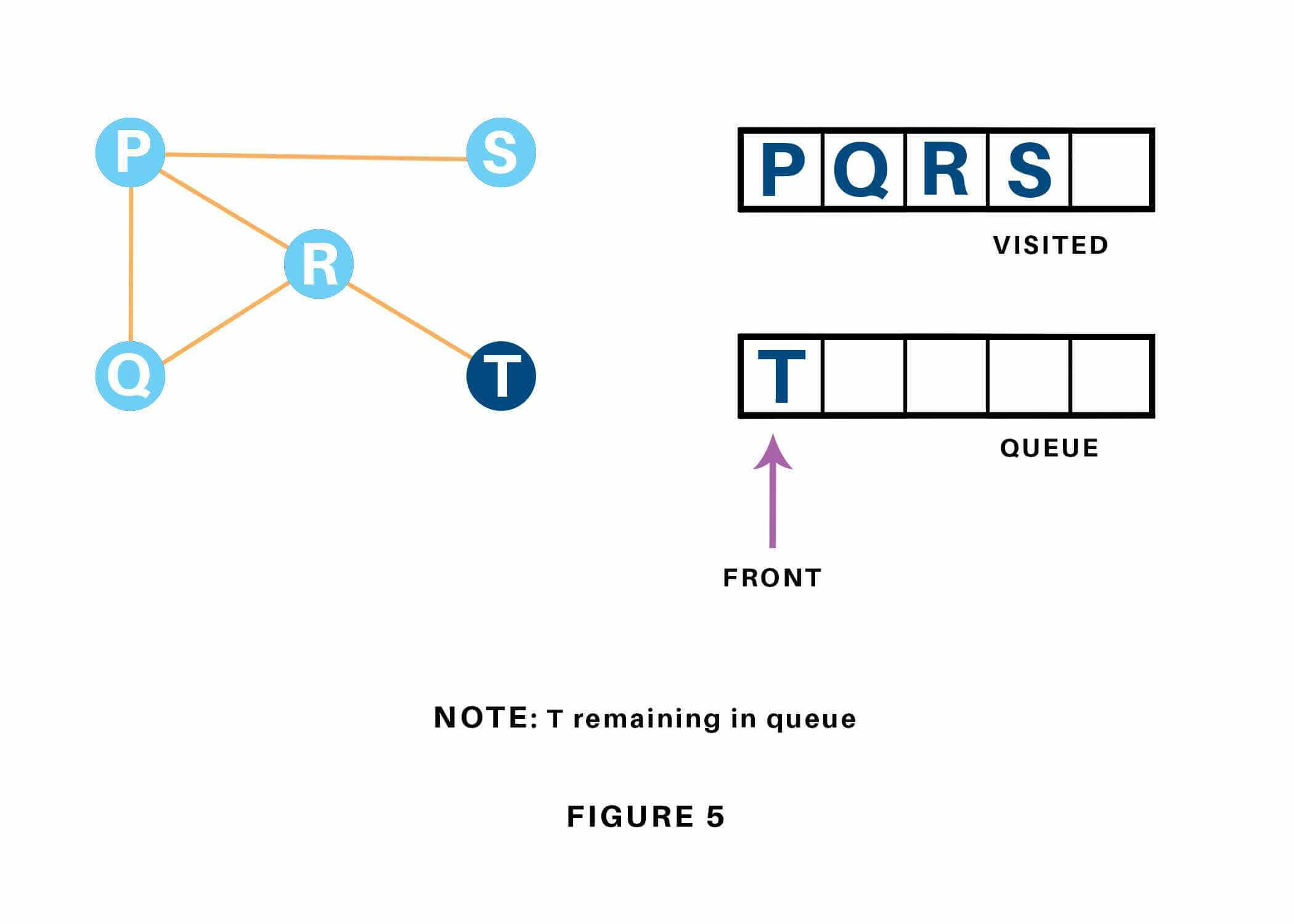


Next, we have a tendency to visit the part at the front of the queue i.e. Q and visit its adjacent nodes. Since P has already been visited, we have a tendency to visit R instead.

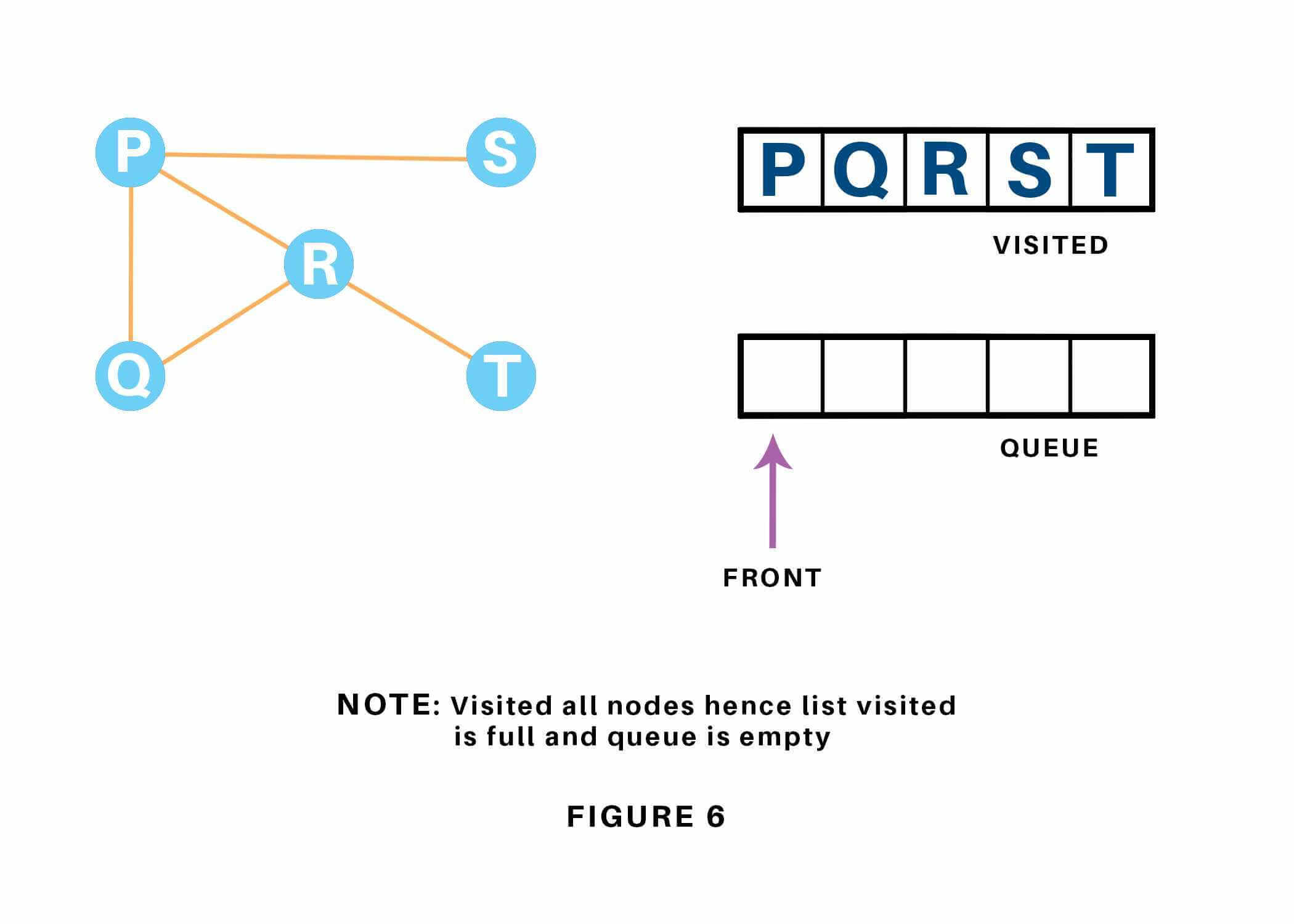


Vertex R has an unvisited adjacent vertex in T, thus we have a tendency to add that to the rear of the queue and visit S, which is at the front of the queue.





Now, only T remains within the queue since the only adjacent node of S i.e. P is already visited. We have a tendency to visit it.



Since the queue is empty, we've completed the Traversal of the graph.

### Time Complexity

The time complexity of the Breadth first Search algorithm is in the form of **O(V+E)**, where V is the representation of the number of nodes and E is the number of edges.

Also, the space complexity of the BFS algorithm is **O(V).**

### Applications

Breadth-first Search Algorithm has a wide range of applications in the real-world. Some of them are as discussed below:

1. In GPS navigation, it helps in finding the shortest path available from one point to another.
2. In pathfinding algorithms
3. Cycle detection in an undirected graph
4. In [minimum spanning tree](https://favtutor.com/blogs/prims-algorithm-python)
5. To build index by search index
6. In Ford-Fulkerson algorithm to find maximum flow in a network.