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.

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.

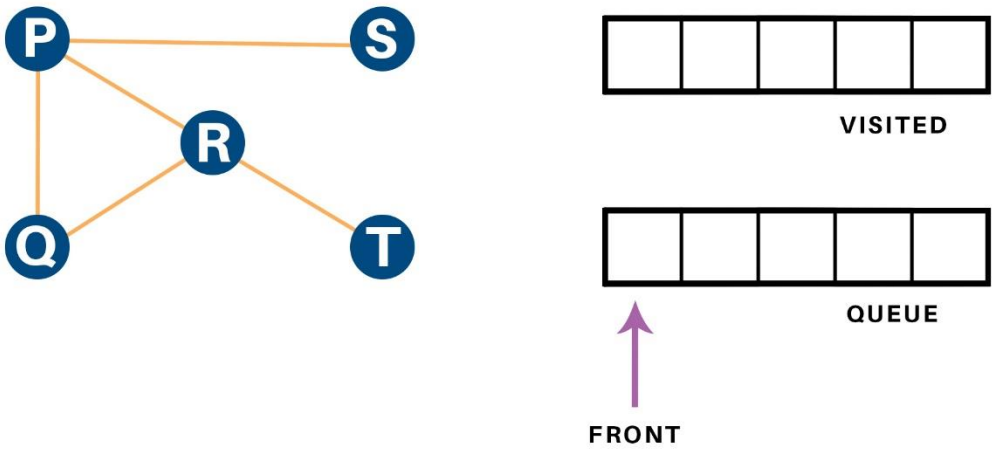
The steps of the algorithm work as follow:

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.

Example

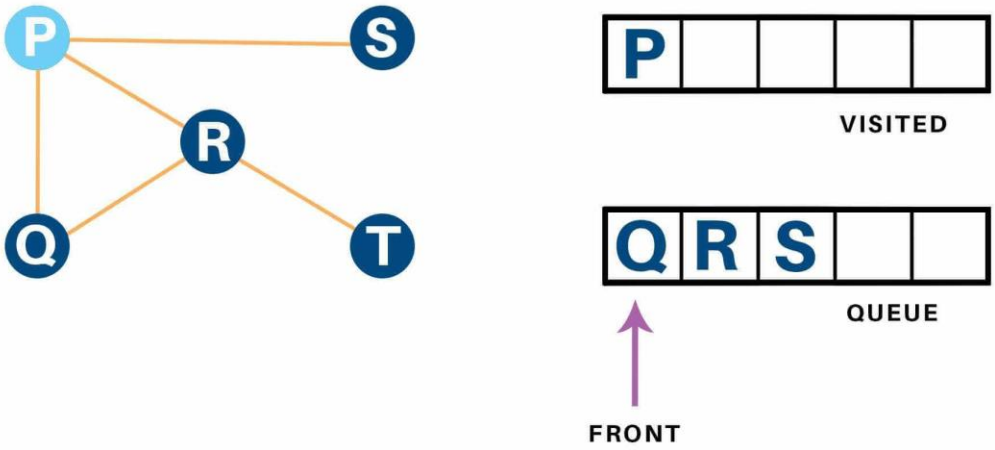
Let us see how this algorithm works with an example. Here, we will use an undirected graph with 5 vertices.



NOTE: Undirected graph with 5 vertices

FIGURE 1

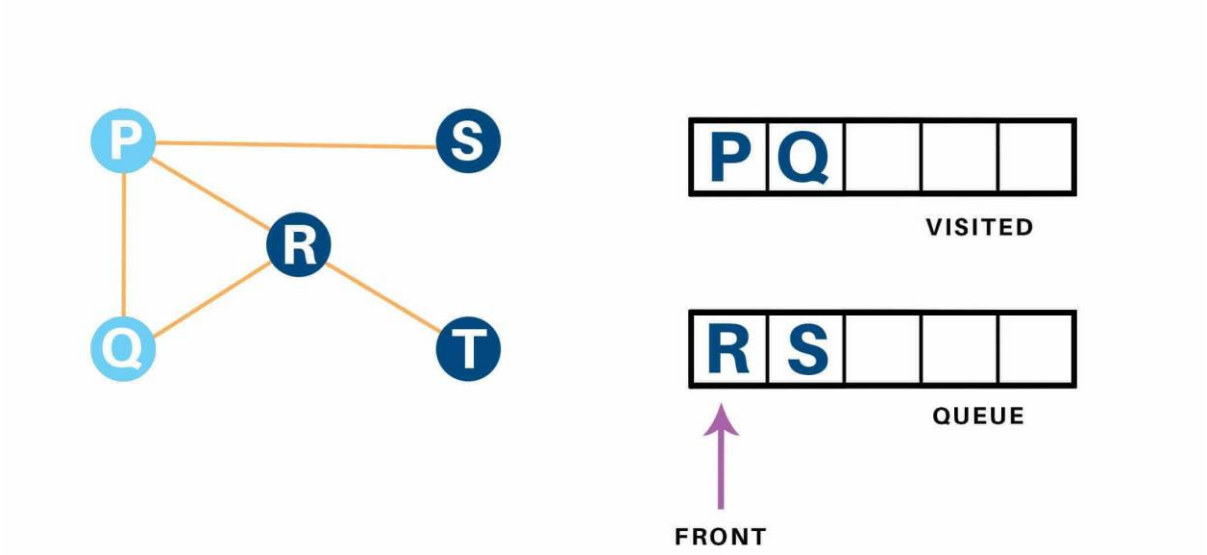
We begin from the vertex P, the BFS algorithmic program starts by putting it within the Visited list and puts all its adjacent vertices within the queue.



NOTE: Visit starting vertex and add its adjacent vertices to queue

FIGURE 2

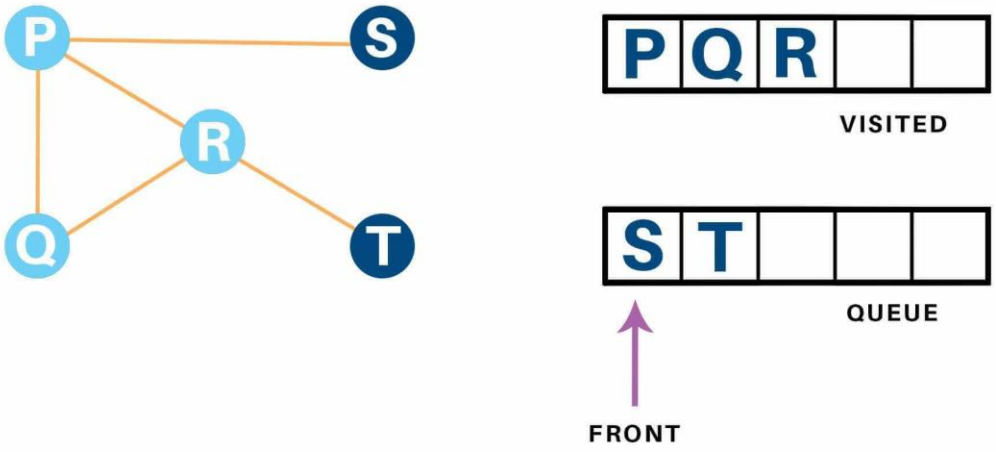
Next, we have a tendency to visit the part at the front of the queue i.e. Q and visit its adjacent nodes. Since P has already been visited, we have a tendency to visit R instead.



NOTE: Visit the first neighbour of node P, which is Q

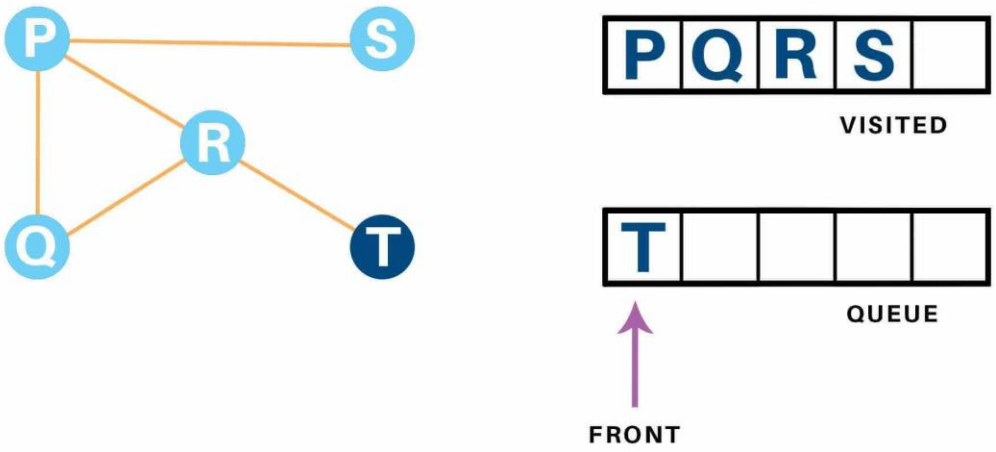
FIGURE 3

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.



NOTE: Visit R which was added to queue earlier to add its neighbour

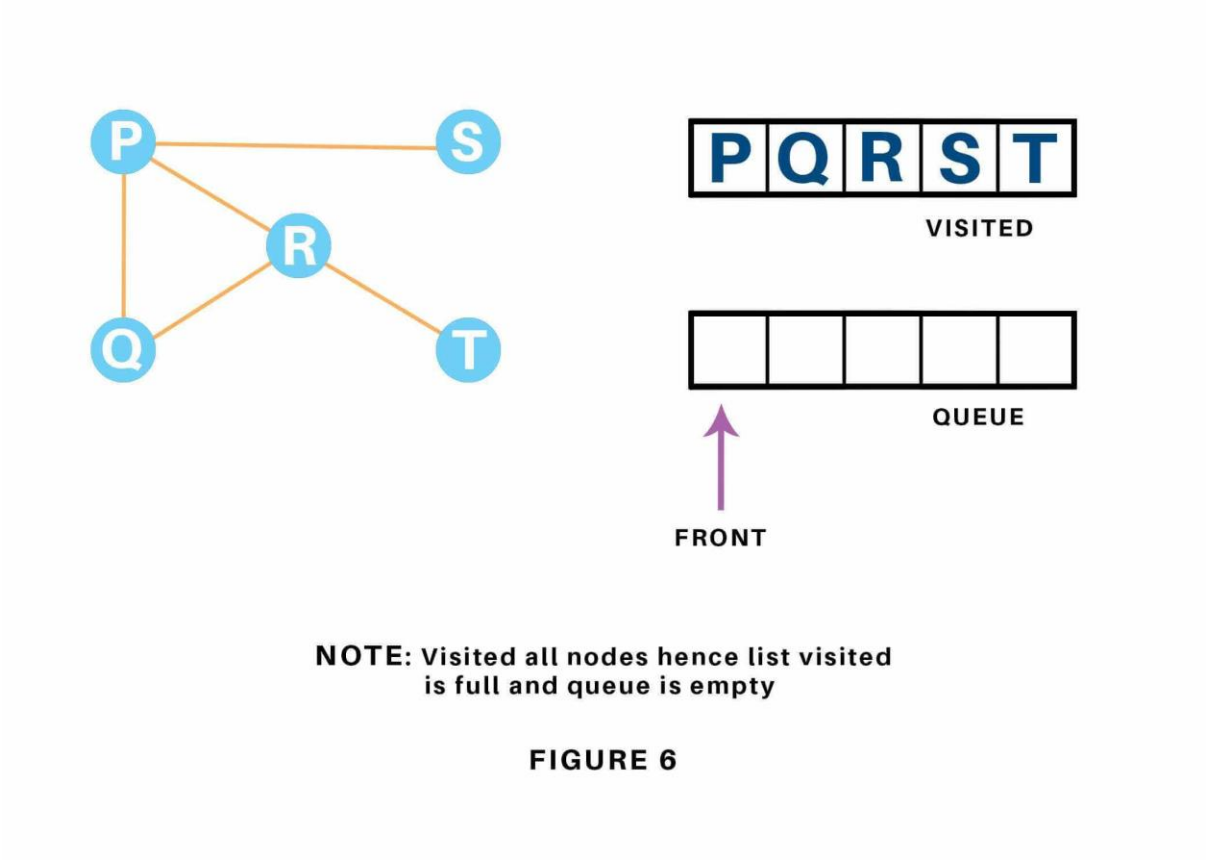
FIGURE 4



NOTE: T remaining in queue

FIGURE 5

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.



NOTE: Visited all nodes hence list visited is full and queue is empty

FIGURE 6

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

Complexity Analysis

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 of BFS Algorithm

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
5. To build index by search index
6. In Ford-Fulkerson algorithm to find maximum flow in a network.

```
def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)

  while queue:
    m = queue.pop(0)
    print (m, end = " ")

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


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

visited = []
queue = []


print("Following is the Breadth-First Search")
bfs(visited, graph, '5')
```