Breadth First Traversal of a Graph

The ***Breadth First Traversal*** or ***BFS*** traversal of a graph is similar to that of the Level Order Traversal of Trees.

The BFS traversal of Graphs also traverses the graph in levels. It starts the traversal with a given vertex, visits all of the vertices adjacent to the initially given vertex and pushes them all to a queue in order of visiting. Then it pops an element from the front of the queue, visits all of its neighbours and pushes the neighbours which are not already visited into the queue and repeats the process until the queue is empty or all of the vertices are visited.

The BFS traversal uses an auxiliary boolean array say *visited[]* which keeps track of the visited vertices. That is if **visited[i] = true** then it means that the **i-th** vertex is already visited.

Complete Algorithm:

1. Create a boolean array say ***visited[]*** of size **V+1** where *V* is the number of vertices in the graph.
2. Create a Queue, mark the source vertex visited as **visited[s] = true** and push it into the queue.
3. Until the Queue is non-empty, repeat the below steps:
    - Pop an element from the queue and print the popped element.
    - Traverse all of the vertices adjacent to the vertex poped from the queue.
    - If any of the adjacent vertex is not already visited, mark it visited and push it to the queue.

Illustration:

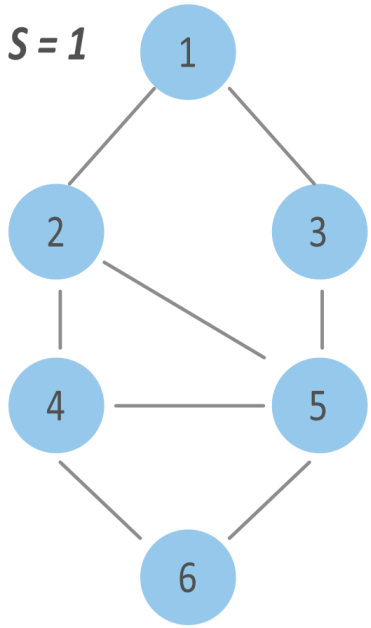
Consider the graph shown in the below Image. The vertices marked **blue** are *not-visited* vertices and the vertices

marked **yellow** are *visited*. The vertex numbered **1** is the source vertex, i.e. the BFS traversal will start from the vertex 1.

Following the BFS algorithm:

- Mark the vertex 1 visited in the visited[] array and push it to the queue.

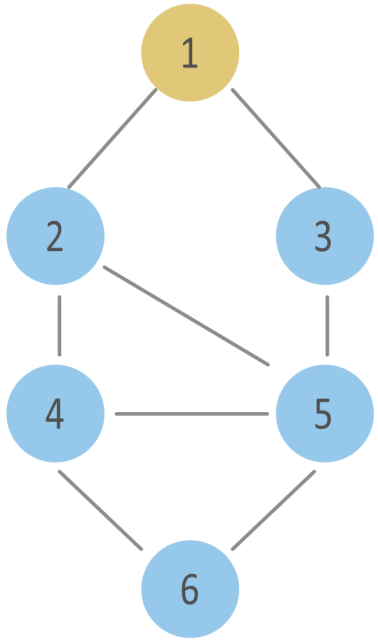
Step: 1



	1	2	3	4	5	6
Visited	0	0	0	0	0	0

Queue

Step: 2

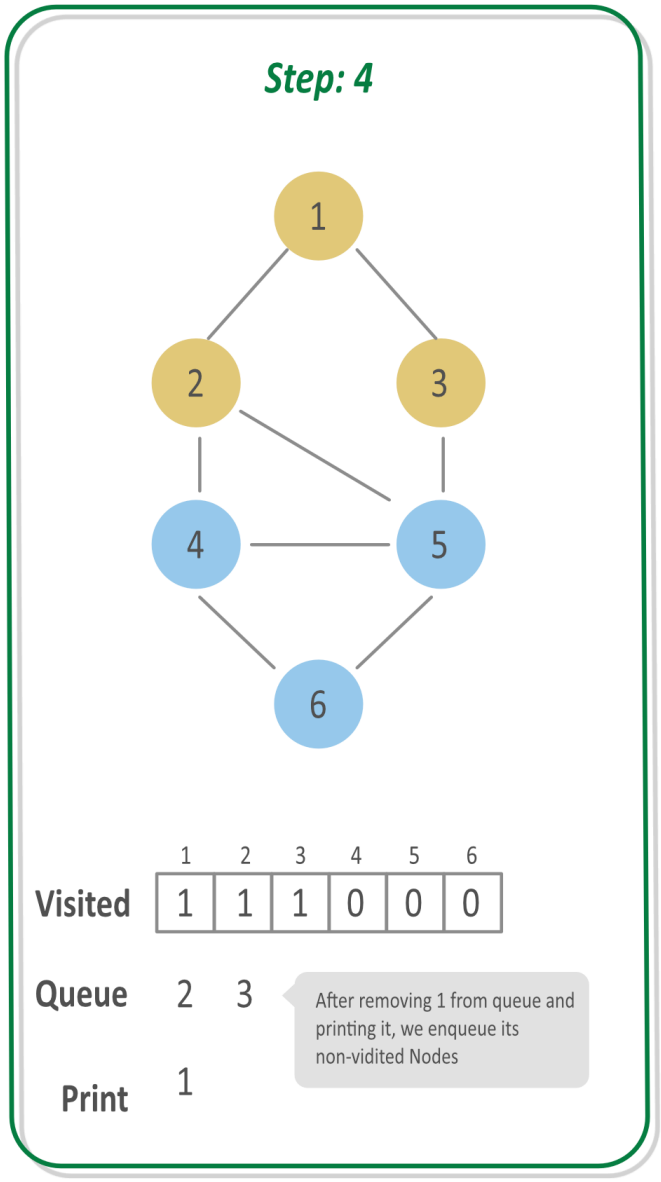


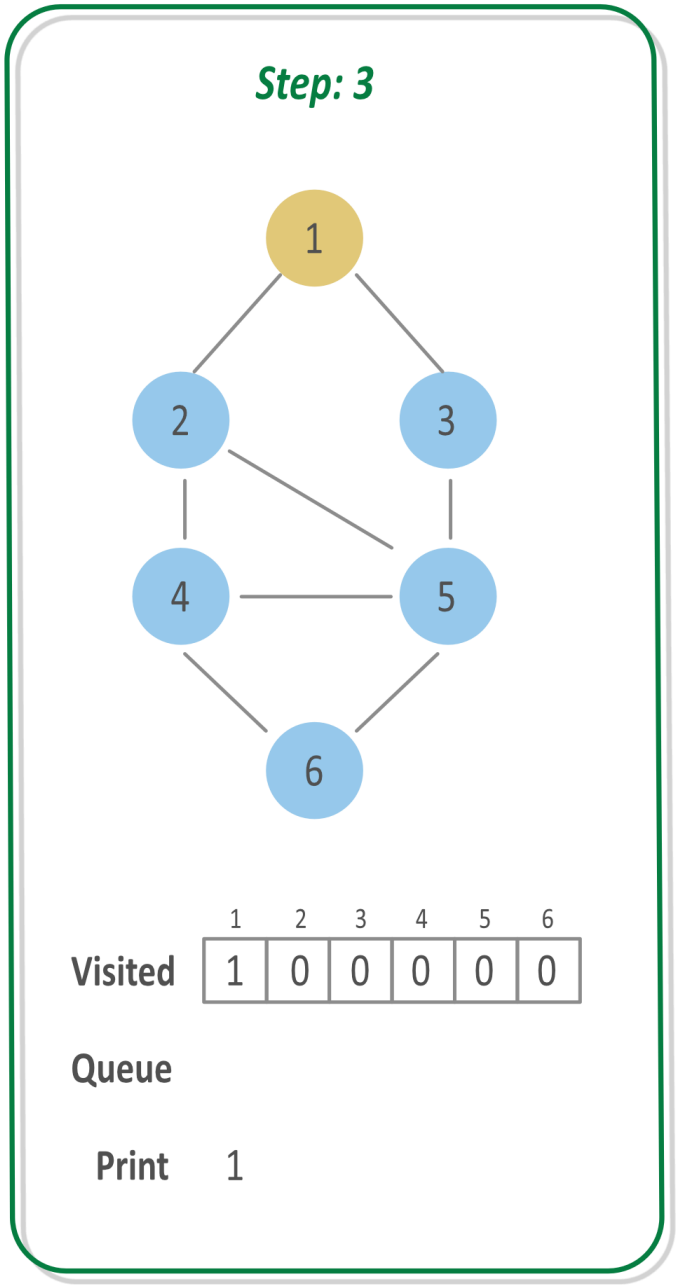
	1	2	3	4	5	6
Visited	1	0	0	0	0	0

Queue 1

Step 3: POP the vertex at the front of queue that is 1, and print it.

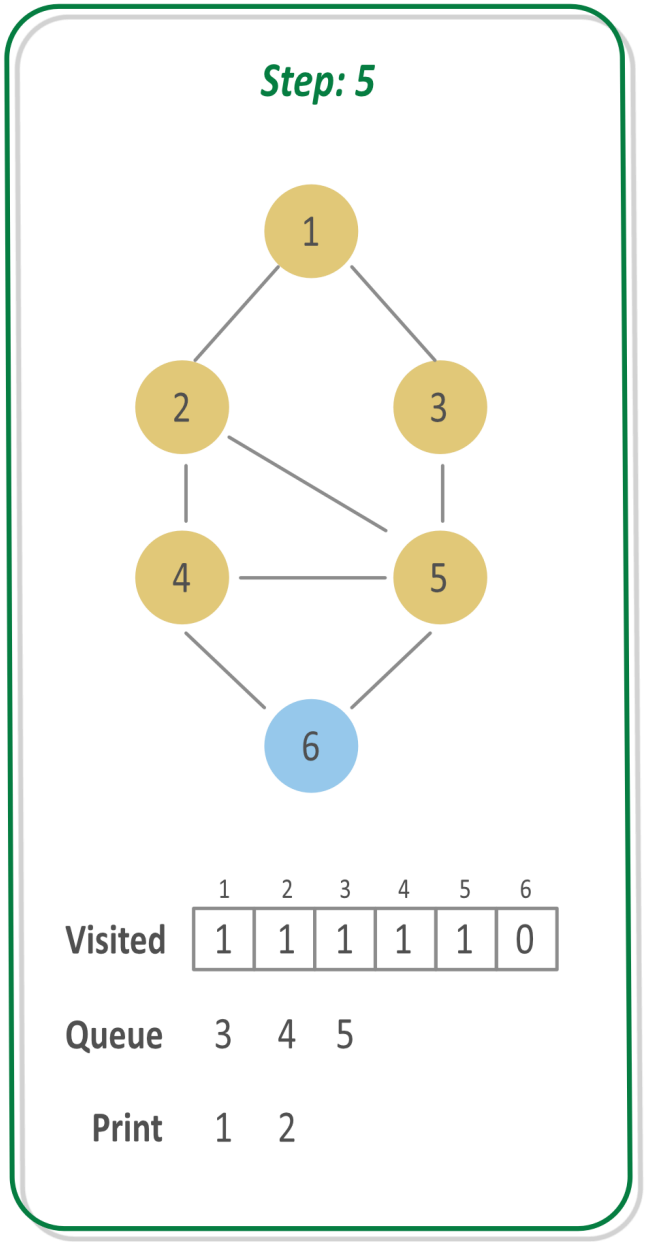
Step 4: Check if adjacent vertices of the vertex 1 are not already visited. If not, mark them visited and push them back to the queue.





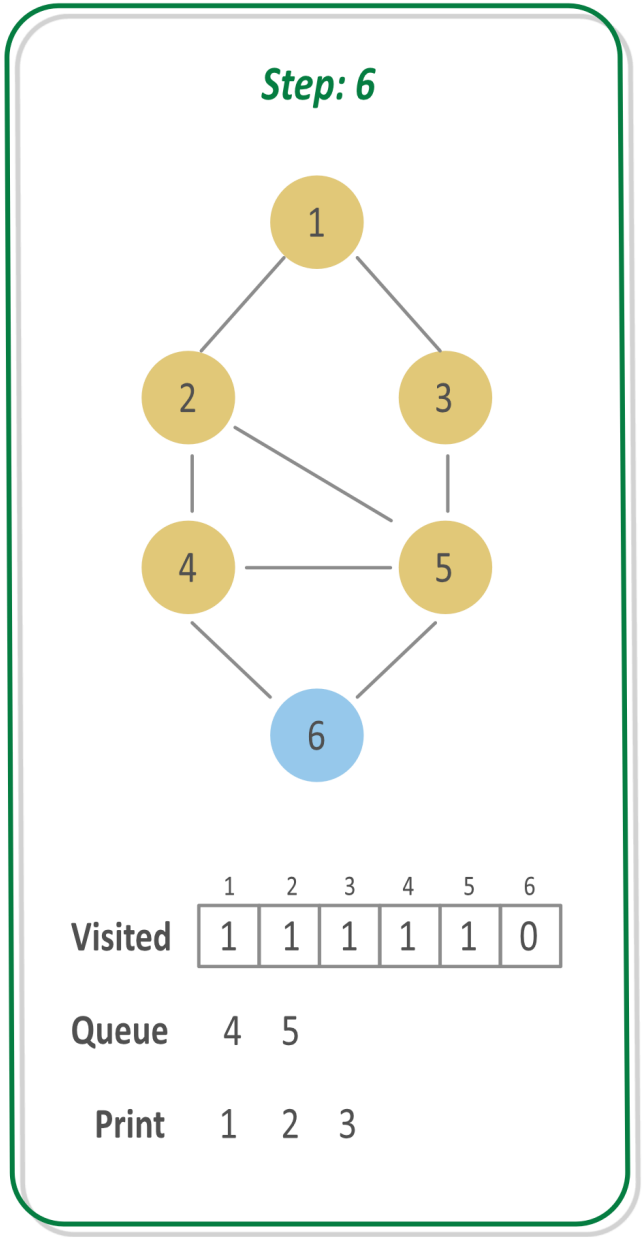
Step 5:

- POP the vertex at front that is 2 and print it.
- Check if the adjacent vertices of 2 are not already visited. If not, mark them visited and push them to queue. So, push 4 and 5.



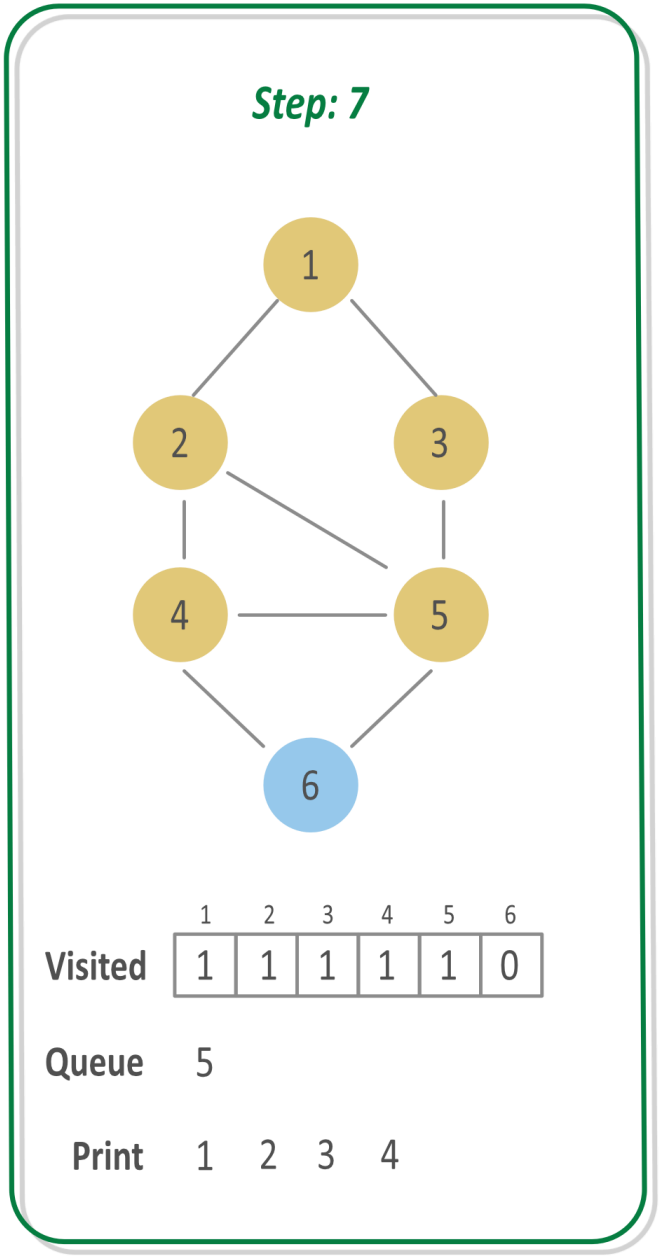
Step 6:

- POP the vertex at front that is 3 and print it.
- Check if the adjacent vertices of 3 are not already visited. If not, mark them visited and push them to queue. So, donot push anything.



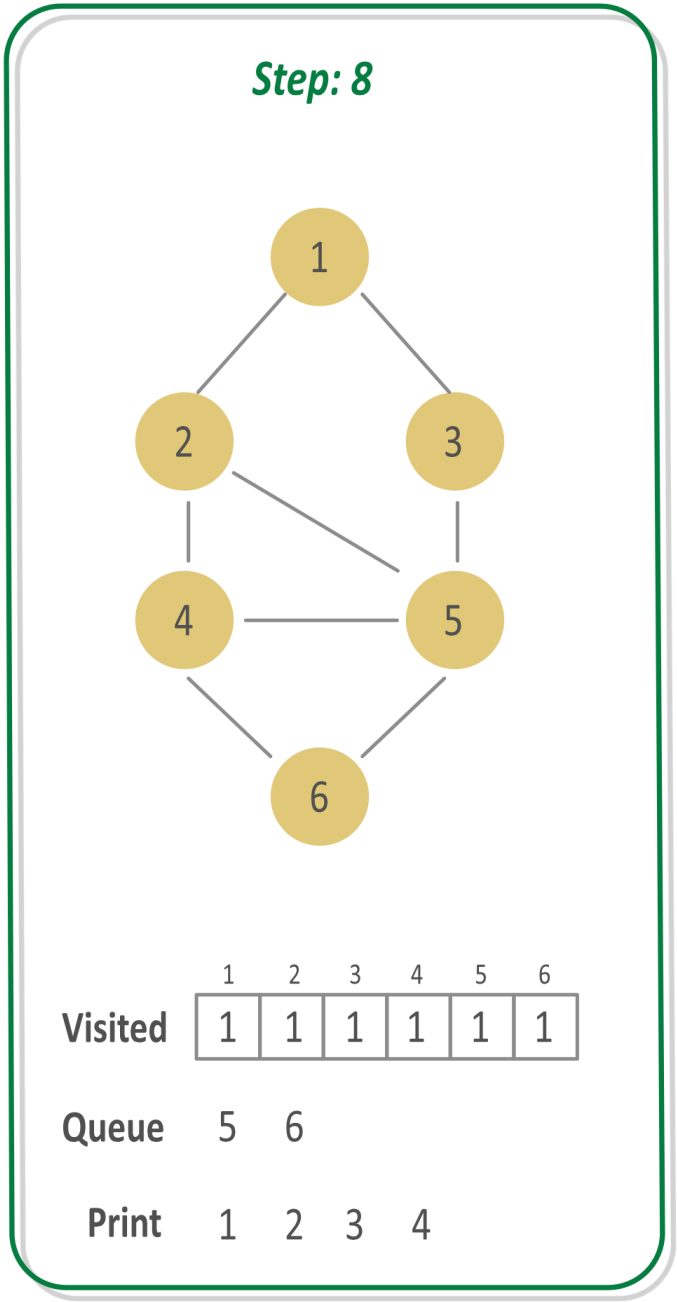
Step 7:

- POP the vertex at front that is 4 and print it.



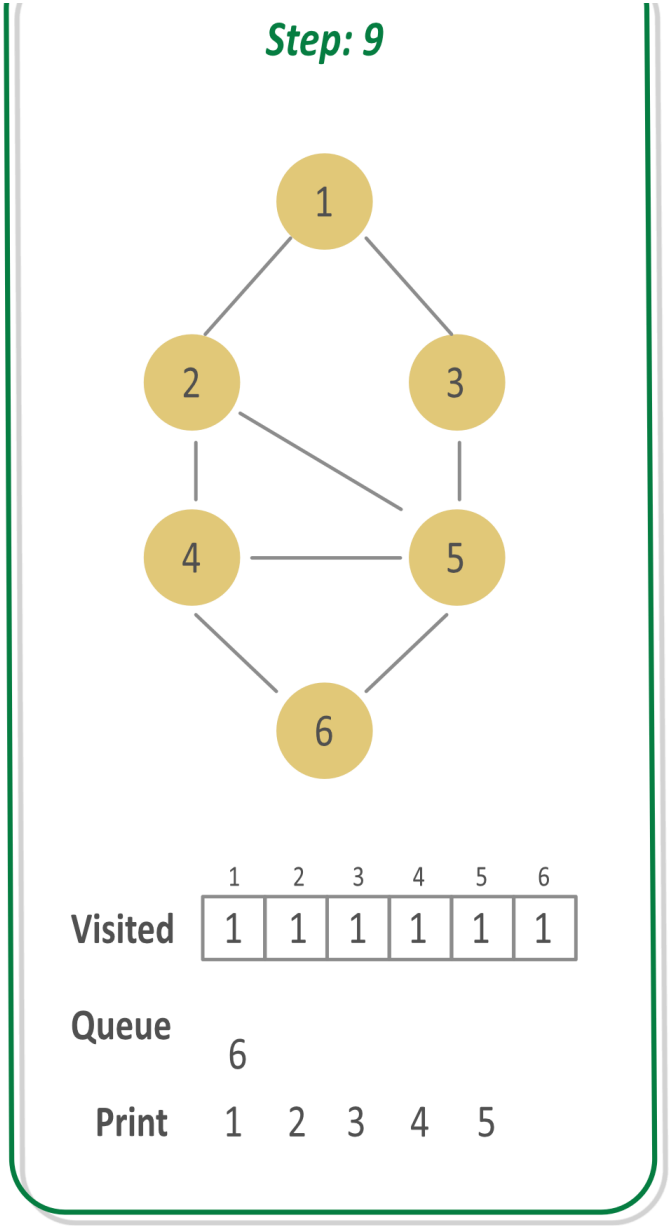
Step 8:

- Check if the adjacent vertices of 4 are not already visited. If not, mark them visited and push them to queue. So, push 6 to the queue.



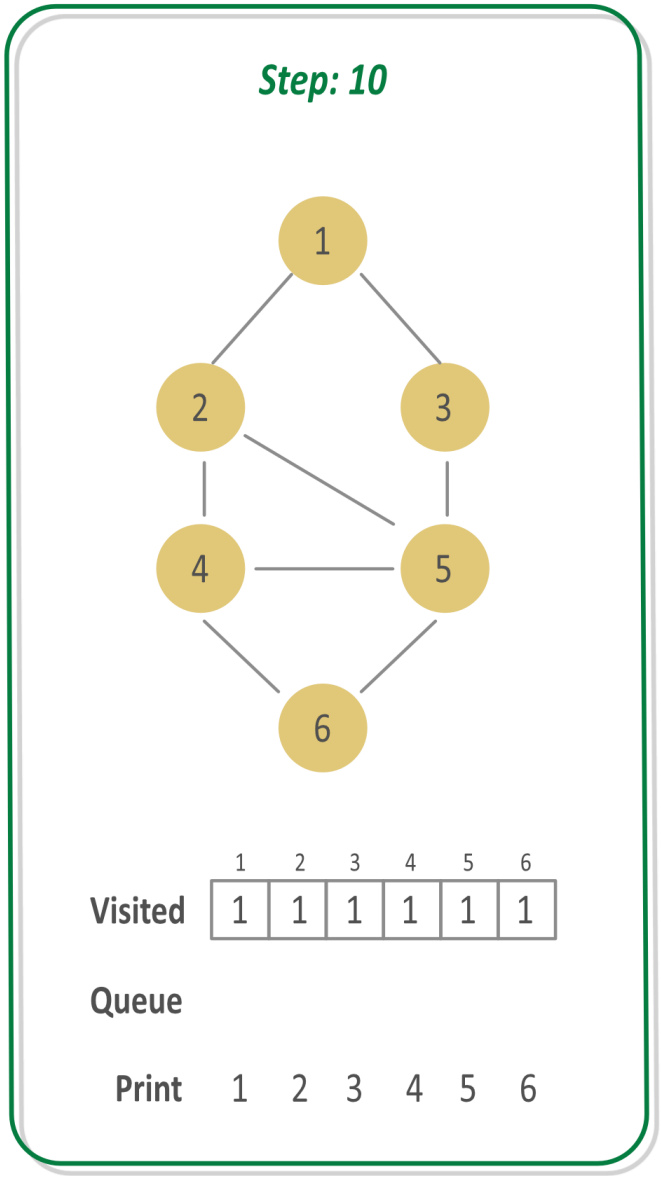
Step 9:

- POP the vertex at front, that is 5 and print it.
- Since, all of its adjacent vertices are already visited, donot push anything.



- POP the vertex at front, that is 6 and print it.
- Since, all of its adjacent vertices are already visited, donot push anything.

Dash
All
Articles
Videos
Problems
Quiz
Contest



Since the Queue is empty now, it means that the complete graph is traversed.

Implementation:

C++ Java

```
// Java code to illustrate BFS traversal
// in a Graph

import java.util.*;

class Graph {

    // A utility function to add an edge in ar
    // undirected graph
    static void addEdge(ArrayList<ArrayList<In
                        int u, int v)
    {
        adj.get(u).add(v);
        adj.get(v).add(u);
    }

    // Function to perform BFS traversal of a
    static void BFS(ArrayList<ArrayList<Intege
    {
        // Initialize a boolean array
        // to keep track of visited vertices
        boolean visited[] =  new boolean[V+1];

        // Mark all vertices not-visited init
        for (int i = 1; i <= V; i++)
            visited[i] = false;

        // Create a queue for BFS
        LinkedList<Integer> queue = new Linke
```

```
        // The start vertex or source vertex :
        int s = 1;

        // Mark the current node as
        // visited and enqueue it
        visited[s]=true;
        queue.add(s);

        while (queue.size() != 0)
        {
            // Dequeue a vertex from queue and
            s = queue.poll();
            System.out.print(s+" ");

            // Traverse the nodes adjacent to
            // poped element and push those e
            // queue which are not already vis
            for (int i = 0; i < adj.get(s).si

                // Fetch adjacent node
                int newNode = adj.get(s).get(

                // Check if it is not visited
                if(visited[newNode] == false)
                {
                    // Mark it visited
                    visited[newNode] = true;

                    // Add it to queue
                    queue.add(newNode);
                }
            }
        }
    }

    // Driver Code
    public static void main(String[] args)
    {
        // Creating a graph with 6 vertices
        int V = 6;
```

```
        ArrayList<ArrayList<Integer> > adj
                    = new ArrayList<ArrayList<

        for (int i = 0; i < V+1; i++)
            adj.add(new ArrayList<Integer>());

        // Adding edges one by one
        addEdge(adj, 1, 2);
        addEdge(adj, 1, 3);
        addEdge(adj, 2, 4);
        addEdge(adj, 2, 5);
        addEdge(adj, 3, 5);
        addEdge(adj, 4, 5);
        addEdge(adj, 4, 6);
        addEdge(adj, 5, 6);

        BFS(adj, V);
    }
}
```

Output:

```
1 2 3 4 5 6
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

If you are facing any issue on this page. Please let us know.