



Breadth First Search or BFS for a Graph

[Read](#)

[Discuss\(160+\)](#)

[Courses](#)

[Practice](#)

[Video](#)

*The **Breadth First Search (BFS)** algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.*

Relation between BFS for Graph and Tree traversal:

[Breadth-First Traversal \(or Search\)](#) for a graph is similar to the [Breadth-First Traversal of a tree](#).

The only catch here is, that, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we divide the vertices into two categories:

- Visited and
- Not visited.

A boolean visited array is used to mark the visited vertices. For simplicity, it is assumed that all vertices are reachable from the starting vertex. BFS uses a [queue data structure](#) for traversal.



How does BFS work?

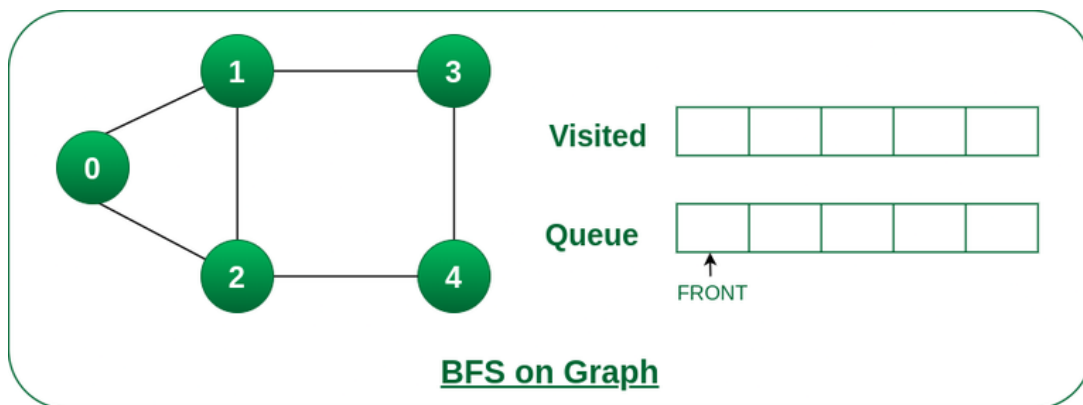
Starting from the root, all the nodes at a particular level are visited first and then the nodes of the next level are traversed till all the nodes are visited.

To do this a queue is used. All the adjacent unvisited nodes of the current level are pushed into the queue and the nodes of the current level are marked visited and popped from the queue.

Illustration:

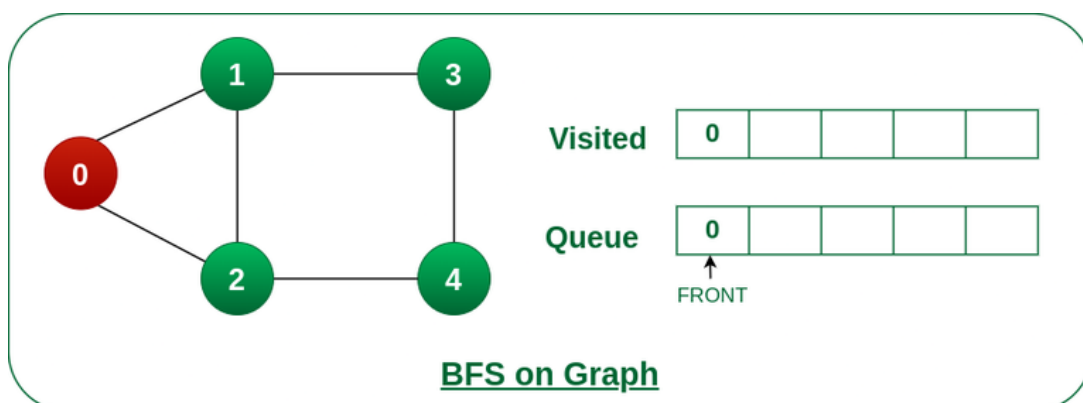
Let us understand the working of the algorithm with the help of the following example.

Step1: Initially queue and visited arrays are empty.



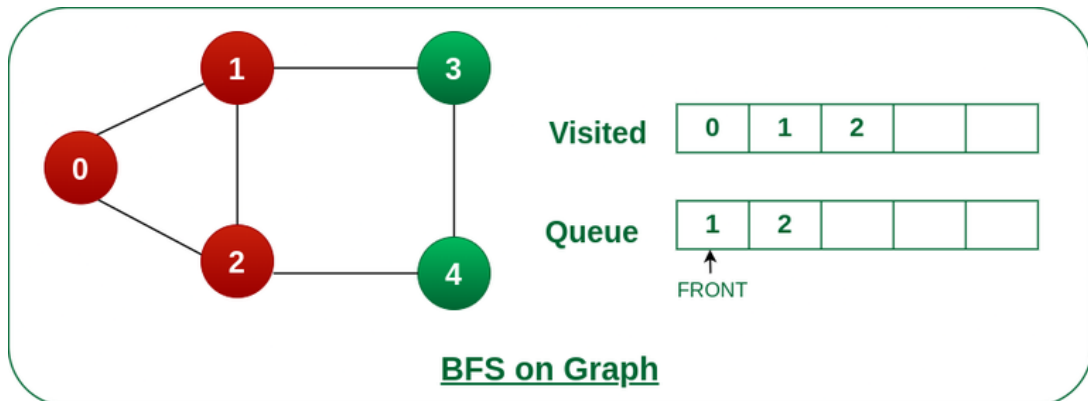
Queue and visited arrays are empty initially.

Step2: Push node 0 into queue and mark it visited.



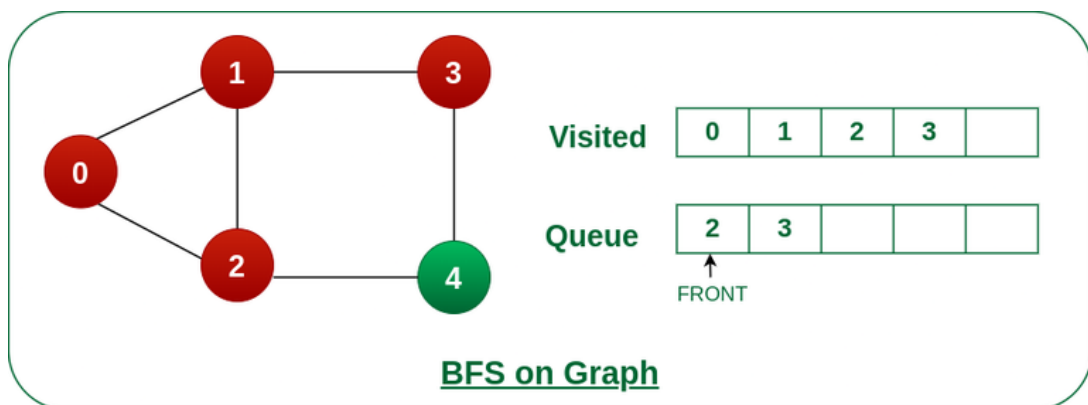
Push node 0 into queue and mark it visited.

Step 3: Remove node 0 from the front of queue and visit the unvisited neighbours and push them into queue.



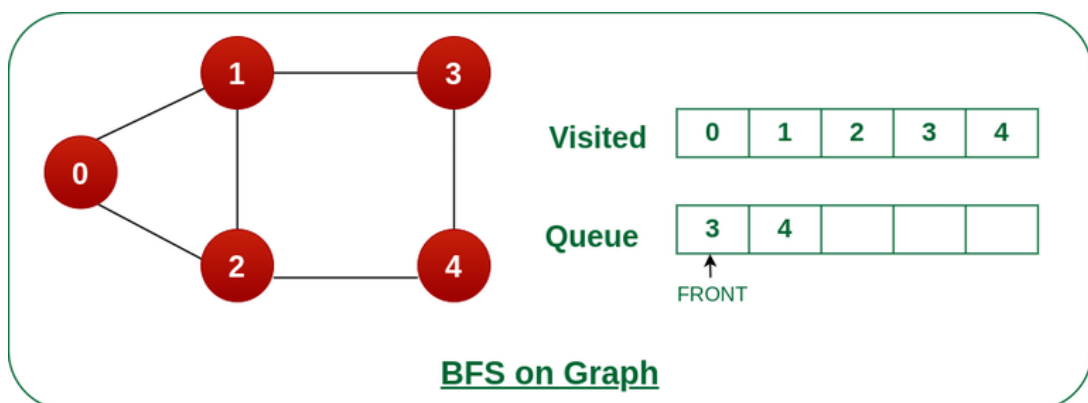
Remove node 0 from the front of queue and visited the unvisited neighbours and push into queue.

Step 4: Remove node 1 from the front of queue and visit the unvisited neighbours and push them into queue.



Remove node 1 from the front of queue and visited the unvisited neighbours and push

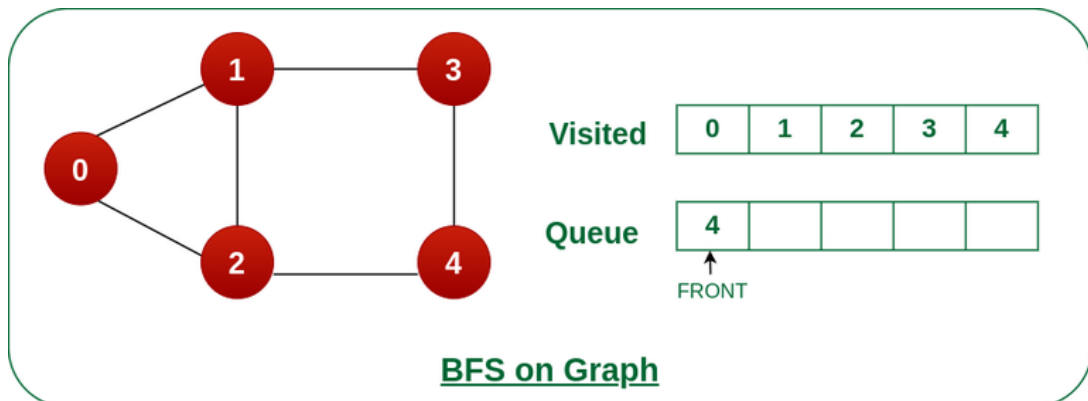
Step 5: Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.



Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.

Step 6: Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.

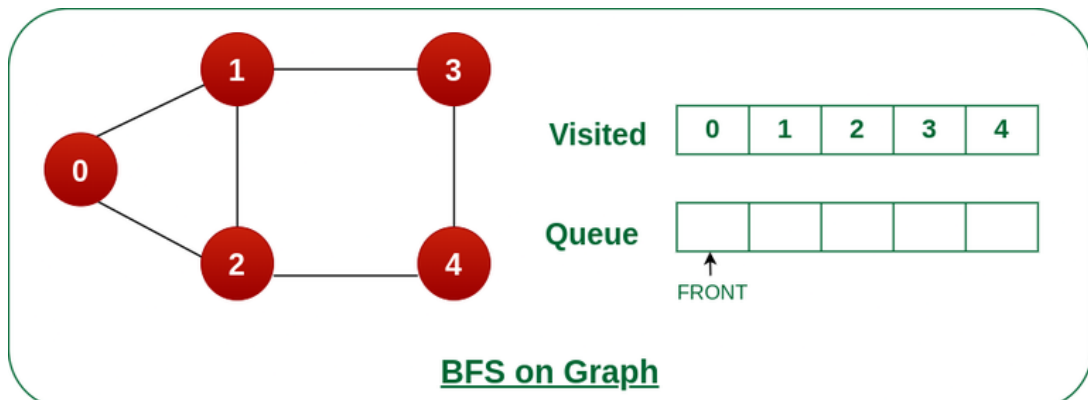
As we can see that every neighbours of node 3 is visited, so move to the next node that are in the front of the queue.



Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.

Steps 7: Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue.

As we can see that every neighbours of node 4 are visited, so move to the next node that is in the front of the queue.



Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue.

Now, Queue becomes empty, So, terminate these process of iteration.

Implementation of BFS for Graph using Adjacency List:

C

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#define MAX_VERTICES 50

// This struct represents a directed graph using
// adjacency list representation
typedef struct Graph_t {

    // No. of vertices
    int V;
    bool adj[MAX_VERTICES][MAX_VERTICES];
} Graph;

// Constructor
Graph* Graph_create(int V)
{
    Graph* g = malloc(sizeof(Graph));
    g->V = V;

    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            g->adj[i][j] = false;
        }
    }

    return g;
}

// Destructor
void Graph_destroy(Graph* g) { free(g); }

// Function to add an edge to graph
void Graph_addEdge(Graph* g, int v, int w)
{
    // Add w to v's list.
    g->adj[v][w] = true;
}

// Prints BFS traversal from a given source s
void Graph_BFS(Graph* g, int s)
{
    // Mark all the vertices as not visited
    bool visited[MAX_VERTICES];
    for (int i = 0; i < g->V; i++) {
```

```

        visited[i] = false;
    }

    // Create a queue for BFS
    int queue[MAX_VERTICES];
    int front = 0, rear = 0;

    // Mark the current node as visited and enqueue it
    visited[s] = true;
    queue[rear++] = s;

    while (front != rear) {

        // Dequeue a vertex from queue and print it
        s = queue[front++];
        printf("%d ", s);

        // Get all adjacent vertices of the dequeued
        // vertex s.
        // If an adjacent has not been visited,
        // then mark it visited and enqueue it
        for (int adjacent = 0; adjacent < g->V;
            adjacent++) {
            if (g->adj[s][adjacent] && !visited[adjacent]) {
                visited[adjacent] = true;
                queue[rear++] = adjacent;
            }
        }
    }
}

// Driver code
int main()
{
    // Create a graph
    Graph* g = Graph_create(4);
    Graph_addEdge(g, 0, 1);
    Graph_addEdge(g, 0, 2);
    Graph_addEdge(g, 1, 2);
    Graph_addEdge(g, 2, 0);
    Graph_addEdge(g, 2, 3);
    Graph_addEdge(g, 3, 3);

    printf("Following is Breadth First Traversal "
           "(starting from vertex 2) \n");
    Graph_BFS(g, 2);
    Graph_destroy(g);

    return 0;
}

```

```
}
```

C++



```
// C++ code to print BFS traversal from a given
// source vertex

#include <bits/stdc++.h>
using namespace std;

// This class represents a directed graph using
// adjacency list representation
class Graph {

    // No. of vertices
    int V;

    // Pointer to an array containing adjacency lists
    vector<list<int> > adj;

public:
    // Constructor
    Graph(int V);

    // Function to add an edge to graph
    void addEdge(int v, int w);

    // Prints BFS traversal from a given source s
    void BFS(int s);
};

Graph::Graph(int V)
{
    this->V = V;
    adj.resize(V);
}

void Graph::addEdge(int v, int w)
{
    // Add w to v's list.
    adj[v].push_back(w);
}

void Graph::BFS(int s)
{
    // Mark all the vertices as not visited
    vector<bool> visited;
    visited.resize(V, false);
```

```

// Create a queue for BFS
list<int> queue;

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

while (!queue.empty()) {

    // Dequeue a vertex from queue and print it
    s = queue.front();
    cout << s << " ";
    queue.pop_front();

    // Get all adjacent vertices of the dequeued
    // vertex s.
    // If an adjacent has not been visited,
    // then mark it visited and enqueue it
    for (auto adjacent : adj[s]) {
        if (!visited[adjacent]) {
            visited[adjacent] = true;
            queue.push_back(adjacent);
        }
    }
}

// Driver code
int main()
{
    // Create a graph given in the above diagram
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Breadth First Traversal "
         << "(starting from vertex 2) \n";
    g.BFS(2);

    return 0;
}

```



```

// Java program to print BFS traversal from a given source
// vertex. BFS(int s) traverses vertices reachable from s.

import java.io.*;
import java.util.*;

// This class represents a directed graph using adjacency
// list representation
class Graph {

    // No. of vertices
    private int V;

    // Adjacency Lists
    private LinkedList<Integer> adj[];

    // Constructor
    Graph(int v)
    {
        V = v;
        adj = new LinkedList[V];
        for (int i = 0; i < v; ++i)
            adj[i] = new LinkedList();
    }

    // Function to add an edge into the graph
    void addEdge(int v, int w) { adj[v].add(w); }

    // prints BFS traversal from a given source s
    void BFS(int s)
    {
        // Mark all the vertices as not visited(By default
        // set as false)
        boolean visited[] = new boolean[V];

        // Create a queue for BFS
        LinkedList<Integer> queue
            = new LinkedList<Integer>();

        // 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 print it
            s = queue.poll();
            System.out.print(s + " ");
        }
    }
}

```

```

        // Get all adjacent vertices of the dequeued
        // vertex s.
        // If an adjacent has not been visited,
        // then mark it visited and enqueue it
        Iterator<Integer> i = adj[s].listIterator();
        while (i.hasNext()) {
            int n = i.next();
            if (!visited[n]) {
                visited[n] = true;
                queue.add(n);
            }
        }
    }
}

// Driver code
public static void main(String args[])
{
    Graph g = new Graph(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    System.out.println(
        "Following is Breadth First Traversal "
        + "(starting from vertex 2)");

    g.BFS(2);
}
}

// This code is contributed by Aakash Hasiya

```

Python3

```

# Python3 Program to print BFS traversal
# from a given source vertex. BFS(int s)
# traverses vertices reachable from s.

from collections import defaultdict

# This class represents a directed graph
# using adjacency list representation

```

```

class Graph:

    # Constructor
    def __init__(self):

        # Default dictionary to store graph
        self.graph = defaultdict(list)

    # Function to add an edge to graph
    def addEdge(self, u, v):
        self.graph[u].append(v)

    # Function to print a BFS of graph
    def BFS(self, s):

        # Mark all the vertices as not visited
        visited = [False] * (max(self.graph) + 1)

        # Create a queue for BFS
        queue = []

        # Mark the source node as
        # visited and enqueue it
        queue.append(s)
        visited[s] = True

        while queue:

            # Dequeue a vertex from
            # queue and print it
            s = queue.pop(0)
            print(s, end=" ")

            # Get all adjacent vertices of the
            # dequeued vertex s.
            # If an adjacent has not been visited,
            # then mark it visited and enqueue it
            for i in self.graph[s]:
                if visited[i] == False:
                    queue.append(i)
                    visited[i] = True

# Driver code
if __name__ == '__main__':

    # Create a graph given in
    # the above diagram
    g = Graph()

```

```

g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)

print("Following is Breadth First Traversal"
      " (starting from vertex 2)")
g.BFS(2)

# This code is contributed by Neelam Yadav

```

C#

```

// C# program to print BFS traversal from a given source
// vertex. BFS(int s) traverses vertices reachable from s.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

// This class represents a directed graph
// using adjacency list representation
class Graph {

    // No. of vertices
    private int _V;

    // Adjacency Lists
    LinkedList<int>[] _adj;

    public Graph(int V)
    {
        _adj = new LinkedList<int>[ V ];
        for (int i = 0; i < _adj.Length; i++) {
            _adj[i] = new LinkedList<int>();
        }
        _V = V;
    }

    // Function to add an edge into the graph
    public void AddEdge(int v, int w)
    {
        _adj[v].AddLast(w);
    }
}

```

```

// Prints BFS traversal from a given source s
public void BFS(int s)
{

    // Mark all the vertices as not
    // visited(By default set as false)
    bool[] visited = new bool[_V];
    for (int i = 0; i < _V; i++)
        visited[i] = false;

    // Create a queue for BFS
    LinkedList<int> queue = new LinkedList<int>();

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

    while (queue.Any()) {

        // Dequeue a vertex from queue
        // and print it
        s = queue.First();
        Console.Write(s + " ");
        queue.RemoveFirst();

        // Get all adjacent vertices of the
        // dequeued vertex s.
        // If an adjacent has not been visited,
        // then mark it visited and enqueue it
        LinkedList<int> list = _adj[s];

        foreach(var val in list)
        {
            if (!visited[val]) {
                visited[val] = true;
                queue.AddLast(val);
            }
        }
    }
}

// Driver code
static void Main(string[] args)
{
    Graph g = new Graph(4);
    g.AddEdge(0, 1);
    g.AddEdge(0, 2);
    g.AddEdge(1, 2);
}

```

```

        g.AddEdge(2, 0);
        g.AddEdge(2, 3);
        g.AddEdge(3, 3);

        Console.WriteLine("Following is Breadth First "
                          + "Traversal(starting from "
                          + "vertex 2) \n");

        g.BFS(2);
    }
}

// This code is contributed by anv89

```

Javascript

```

// Javascript Program to print BFS traversal from a given
// source vertex. BFS(int s) traverses vertices
// reachable from s.

// This class represents a directed graph using
// adjacency list representation
class Graph
{
    // Constructor
    constructor(v)
    {
        this.V = v;
        this.adj = new Array(v);
        for(let i = 0; i < v; i++)
            this.adj[i] = [];
    }

    // Function to add an edge into the graph
    addEdge(v, w)
    {
        // Add w to v's list.
        this.adj[v].push(w);
    }

    // Prints BFS traversal from a given source s
    BFS(s)
    {
        // Mark all the vertices as not visited(By default
        // set as false)
        let visited = new Array(this.V);
        for(let i = 0; i < this.V; i++)
            visited[i] = false;
    }
}

```

```

    // Create a queue for BFS
    let queue=[];

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

    while(queue.length>0)
    {
        // Dequeue a vertex from queue and print it
        s = queue[0];
        console.log(s+" ");
        queue.shift();

        // Get all adjacent vertices of the dequeued
        // vertex s.
        // If an adjacent has not been visited,
        // then mark it visited and enqueue it
        this.adj[s].forEach((adjacent,i) => {
            if(!visited[adjacent])
            {
                visited[adjacent]=true;
                queue.push(adjacent);
            }
        });
    }
}

// Driver program to test methods of graph class

// Create a graph given in the above diagram
g = new Graph(4);
g.addEdge(0, 1);
g.addEdge(0, 2);
g.addEdge(1, 2);
g.addEdge(2, 0);
g.addEdge(2, 3);
g.addEdge(3, 3);

console.log("Following is Breadth First Traversal " +
           "(starting from vertex 2) ");

g.BFS(2);

// This code is contributed by Aman Kumar.

```

Output

Following is Breadth First Traversal (starting from vertex 2)

2 0 3 1

Time Complexity: $O(V+E)$, where V is the number of nodes and E is the number of edges.

Auxiliary Space: $O(V)$

Problems related to BFS:

S.no	Problems	Practice
1.	Find the level of a given node in an Undirected Graph	Link
2.	Minimize maximum adjacent difference in a path from top-left to bottom-right	Link
3.	Minimum jump to the same value or adjacent to reach the end of an Array	Link
4.	Maximum coin in minimum time by skipping K obstacles along the path in Matrix	Link
5.	Check if all nodes of the Undirected Graph can be visited from the given Node	Link
6.	Minimum time to visit all nodes of a given Graph at least once	Link
7.	Minimize moves to the next greater element to reach the end of the Array	Link
8.	Shortest path by removing K walls	Link

S.no	Problems	Practice
9.	Minimum time required to infect all the nodes of the Binary tree	Link
10.	Check if destination of given Matrix is reachable with required values of cells	Link

What else you can read?

- [Recent Articles on BFS](#)
- [Depth First Traversal](#)
- [Applications of Breadth First Traversal](#)
- [Applications of Depth First Search](#)

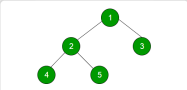
Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Feeling lost in the world of random DSA topics, wasting time without progress? It's time for a change! Join our DSA course, where we'll guide you on an exciting journey to master DSA efficiently and on schedule.


Ready to dive in? Explore our Free Demo Content and join our DSA course, trusted by over 100,000 geeks!

- [DSA in C++](#)
- [DSA in Java](#)
- [DSA in Python](#)
- [DSA in JavaScript](#)


Similar Reads




Level Order Traversal (Breadth First Search or BFS) of Binary Tree



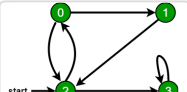
Breadth First Traversal (BFS) on a 2D array



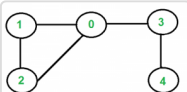
Applications, Advantages and Disadvantages of...



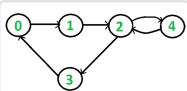
Implementing Water Supply Problem using Breadth First Search




Breadth First Search without using Queue



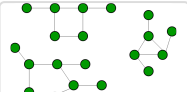
Detect cycle in an undirected graph using BFS




Detect Cycle in a Directed Graph using BFS



Print the lexicographically smallest BFS of the...




Islands in a graph using BFS




When to use DFS or BFS to solve a Graph problem?


Related Tutorials




Mathematical and Geometric Algorithms - Data Structure and...



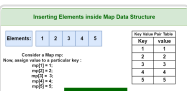
Learn Data Structures with Javascript | DSA Tutorial



Introduction to Max-Heap - Data Structure and Algorithm Tutorials



Introduction to Set - Data Structure and Algorithm Tutorials



Introduction to Map - Data Structure and Algorithm Tutorials

Previous

Difference between graph and tree

Next

Depth First Search or DFS for a Graph

Article Contributed By :



GeeksforGeeks

Vote for difficulty

Current difficulty : [Easy](#)

[Easy](#)[Normal](#)[Medium](#)[Hard](#)[Expert](#)

Improved By : [akjlucky4all](#), [aayushi2402](#), [itskawal2000](#), [anvudemy1](#),
[atharvacp20](#), [simranarora5sos](#), [ajaymakvana](#), [devendrasalunke](#),
[surinderdawra388](#), [punamsingh628700](#), [manishmandal9734](#),
[qwerty_gfg](#), [rkbhola5](#), [animeshdey](#), [abhijeet19403](#), [akashish__](#),
[vladozaric](#), [harendrakumar123](#), [mitalibhola94](#), [amankr0211](#),
[janardansthox](#), [anikettchavan](#), [patilsanik89c8](#), [_ani_s37](#)

Article Tags : [BFS](#) , [graph-basics](#) , [DSA](#) , [Graph](#) , [Queue](#)

Practice Tags : [BFS](#), [Graph](#), [Queue](#)

[Improve Article](#)[Report Issue](#)

Sanchhaya Education Private Limited

A-143, 9th Floor, Sovereign Corporate
Tower, Sector-136, Noida, Uttar Pradesh -
201305

feedback@geeksforgeeks.org



Company

About Us
Legal
Careers
In Media
Contact Us
Advertise with us
Placement Training Program
Apply for Mentor

Languages

Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial

DSA Roadmaps

DSA for Beginners
Basic DSA Coding Problems
DSA Roadmap by Sandeep Jain
DSA with JavaScript
Top 100 DSA Interview Problems
All Cheat Sheets

Computer Science

GATE CS Notes

Explore

Job-A-Thon Hiring Challenge
Hack-A-Thon
GfG Weekly Contest
Offline Classes (Delhi/NCR)
DSA in JAVA/C++
Master System Design
Master CP

DSA Concepts

Data Structures
Arrays
Strings
Linked List
Algorithms
Searching
Sorting
Mathematical
Dynamic Programming

Web Development

HTML
CSS
JavaScript
Bootstrap
ReactJS
AngularJS
NodeJS
Express.js
Lodash

Python

Python Programming Examples

Operating Systems

Computer Network

Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

Data Science & ML

Data Science With Python

Data Science For Beginner

Machine Learning Tutorial

Maths For Machine Learning

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Deep Learning Tutorial

Competitive Programming

Top DSA for CP

Top 50 Tree Problems

Top 50 Graph Problems

Top 50 Array Problems

Top 50 String Problems

Top 50 DP Problems

Top 15 Websites for CP

Interview Corner

Company Wise Preparation

Preparation for SDE

Experienced Interviews

Internship Interviews

Competitive Programming

Aptitude Preparation

Commerce

Django Tutorial

Python Projects

Python Tkinter

OpenCV Python Tutorial

Python Interview Question

DevOps

Git

AWS

Docker

Kubernetes

Azure

GCP

System Design

What is System Design

Monolithic and Distributed SD

Scalability in SD

Databases in SD

High Level Design or HLD

Low Level Design or LLD

Top SD Interview Questions

GfG School

CBSE Notes for Class 8

CBSE Notes for Class 9

CBSE Notes for Class 10

CBSE Notes for Class 11

CBSE Notes for Class 12

English Grammar

UPSC

Accountancy
Business Studies
Economics
Human Resource Management (HRM)
Management
Income Tax
Finance
Statistics for Economics

SSC/ BANKING

SSC CGL Syllabus
SBI PO Syllabus
SBI Clerk Syllabus
IBPS PO Syllabus
IBPS Clerk Syllabus
Aptitude Questions
SSC CGL Practice Papers

Polity Notes
Geography Notes
History Notes
Science and Technology Notes
Economics Notes
Important Topics in Ethics
UPSC Previous Year Papers

Write & Earn

Write an Article
Improve an Article
Pick Topics to Write
Write Interview Experience
Internships