

[Register for SDE Sheet Challenge](#)**takeUforward**[SignIn/Signup](#)[Striver's SDE Sheet](#) [Striver's A2Z DSA Course/Sheet](#)[Striver's DSA Playlists](#)[CS Subjects](#) [Interview Prep Sheets](#)[Striver's CP Sheet](#)

September 6, 2022 • Graph

Search

# Bipartite Graph | BFS Implementation

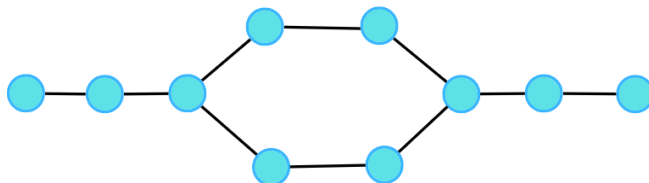
**Problem Statement:** Given an adjacency list of a graph adj of V no. of vertices having 0 based index. Check whether the graph is bipartite or not.

If we are able to colour a graph with two colours such that no adjacent nodes have the same colour, it is called a bipartite graph.

## Examples:

### Example 1:

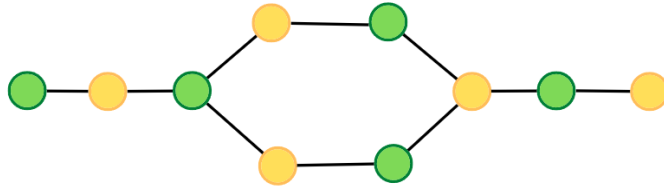
Input:



Output: 1

## Recent Posts

[2023 – Striver's SDE Sheet Challenge](#)[Top Array](#)[Interview](#)[Questions –](#)[Structured Path with Video](#)[Solutions](#)[Longest Subarray](#)[with sum K |](#)[\[Positives and](#)[Negatives\]](#)

**Explanation:****Example 2:****Input:****Output:** 0**Explanation:**

Count Subarray

sum Equals K

Binary Tree

Representation in

Java

Accolite Digital

Amazon Arcesium

arrays Bank of America

Barclays BFS Binary

Search Binary Search

Tree Commvault CPP DE

Shaw DFS DSA

Self Paced

google HackerEarth Hashing

infosys inorder Interview

Experience Java Juspay

Kreeti Technologies Morgan

Stanley Newfold Digital

Oracle post order recursion

Samsung SDE Core Sheet

SDE Sheet

Searching set-bits sorting

Strivers

A2ZDSA

Course sub-array

subarray Swiggy

takeuforward TCS TCS

CODEVITA TCS Ninja

## Solution

**Disclaimer:** *Don't jump directly to the solution, try it out yourself first.*

### Intuition:.

A bipartite graph is a graph which can be coloured using 2 colours such that no adjacent nodes have the same colour. Any linear graph with no cycle is always a bipartite graph. With cycle, any graph with an even cycle length can also be a bipartite graph. So, any graph with an odd cycle length can never be a bipartite graph.

The intuition is the brute force of filling colours using any traversal technique, just make sure no two adjacent nodes have the same colour. If at any moment of traversal, we find the adjacent nodes to have the same

colour, it means that there is an odd cycle, or it cannot be a bipartite graph.

## **Approach:**

We can follow either of the traversal techniques. In this article, we will be solving it using BFS traversal.

Breadth First Search, BFS is a traversal technique where we visit the nodes level-wise, i.e., it visits the same level nodes simultaneously, and then moves to the next level. We will be defining the BFS traversal below, but this check has to be done for every component, for that we can use the simple for loop concept that we have learnt, to call the traversals for unvisited nodes.

## **Initial configuration:**

- **Queue:** Define a queue and insert the source node initially to start with.
- **Colour array:** Instead of a visited array, we will take a colour array where all the nodes are initialised to -1 indicating they are not coloured yet.

The algorithm steps are as follows:

- For BFS traversal, we need a queue data structure and a visited array (in this case colour array).
- Take the source node and push it into the Queue. Whenever we try to put it in the queue, we assign a colour to the node. We will try to colour with 0 and 1, but you can choose other colours as well. We will start with the colour 0, you can start with 1 as well, just make sure for the adjacent node, it should be opposite of what the current node has.
- Start the BFS traversal, pop out an element from the queue every time and travel to all its uncoloured neighbours using the adjacency list.
- For every uncoloured node, initialise it with the opposite colour to that of the current node, and push it into the Q data structure, for further traversals.
- Repeat the steps either until the queue becomes empty.
- If at any moment, we get an adjacent node from the adjacency list which is

already coloured and has the same colour as the current node, we can say it is not possible to colour it, hence it cannot be bipartite. Thereby we will stop the check here, and return a false, without visiting any further nodes.

- If the queue becomes empty, the graph is coloured and no two adjacent nodes have the same colour then return value 1 indicating it is a bipartite graph.

Consider the following graph and its adjacency list.

Consider the following illustration to understand the colouring of the nodes using BFS traversal.

## Code:

---

### C++ Code

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

class Solution {
    // colors a component
private:
    bool check(int start, int V, vector<int> adj, vector<int> color) {
        queue<int> q;
        q.push(start);
        color[start] = 0;
        while(!q.empty()) {
            int node = q.front();
            q.pop();

            for(auto it : adj[node])
                // if the adjacent node is not colored
                // you will give the color
                if(color[it] == -1) {
                    color[it] = !color[node];
                    q.push(it);
                }
            // is the adjacent node already colored
            // someone did color it
            else if(color[it] == 0 || color[it] == 1)
                return false;
        }
        return true;
    }
};
```

```

        return false;
    }
}
}
return true;
}
public:
    bool isBipartite(int V, vector<int> adj) {
        int color[V];
        for(int i = 0; i < V; i++) color[i] = -1;

        for(int i = 0; i < V; i++) {
            // if not coloured
            if(color[i] == -1) {
                if(!check(i, V, adj, color))
                    return false;
            }
        }
        return true;
    }
};

void addEdge(vector<int> adj[], int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u);
}

int main() {
    // V = 4, E = 4
    vector<int> adj[4];

    addEdge(adj, 0, 2);
    addEdge(adj, 0, 3);
    addEdge(adj, 2, 3);
    addEdge(adj, 3, 1);

    Solution obj;
    bool ans = obj.isBipartite(4, adj);
    if(ans) cout << "1\n";
    else cout << "0\n";

    return 0;
}

```



**Output:** 0

**Time Complexity:**  $O(V + 2E)$ , Where  $V$  = Vertices,  $2E$  is for total degrees as we traverse all adjacent nodes.

**Space Complexity:**  $O(3V) \sim O(V)$ , Space for queue data structure, colour array and an adjacency list.

## Java Code

```
import java.util.*;

class Solution
{
    private boolean check(int start,
        ArrayList<ArrayList<Integer>>adj
        Queue<Integer> q = new Linke
        q.add(start);
        color[start] = 0;
        while(!q.isEmpty()) {
            int node = q.peek();
            q.remove();

            for(int it : adj.get(nod
                // if the adjacent n
                // you will give the
                if(color[it] == -1)

                    color[it] = 1 -
                    q.add(it);
            }
            // is the adjacent g
            // someone did color
            else if(color[it] ==
                return false;
            }
        }
    }
    return true;
}
```

```

    }

    public boolean isBipartite(int V)
    {
        int color[] = new int[V];
        for(int i = 0;i<V;i++) color[i] = -1;

        for(int i = 0;i<V;i++) {
            if(color[i] == -1) {
                if(!check(i, V, adj, color))
                    return false;
            }
        }

        return true;
    }

    public static void main(String[] args)
    {
        // V = 4, E = 4
        ArrayList<ArrayList<Integer>> adj = new ArrayList<ArrayList<Integer>>();
        for (int i = 0; i < 4; i++)
            adj.add(new ArrayList<Integer>());

        adj.get(0).add(2);
        adj.get(2).add(0);
        adj.get(0).add(3);
        adj.get(3).add(0);
        adj.get(1).add(3);
        adj.get(3).add(1);
        adj.get(2).add(3);
        adj.get(3).add(2);

        Solution obj = new Solution();
        boolean ans = obj.isBipartite(4, adj);
        if(ans)
            System.out.println("1");
        else System.out.println("0")
    }
}

```

**Output: 0**

**Time Complexity:**  $O(V + 2E)$ , Where  $V$  = Vertices,  $2E$  is for total degrees as we traverse all adjacent nodes.

**Space Complexity:**  $O(3V) \sim O(V)$ , Space for queue data structure, colour array and an adjacency list.

Special thanks to [Vanshika Singh Gour](#) for contributing to this article on takeUforward. If you also wish to share your knowledge with the takeUforward fam, [please check out this article](#). If you want to suggest any improvement/correction in this article please mail us at [write4tuf@gmail.com](mailto:write4tuf@gmail.com)

---

« Previous Post

**Bipartite Graph |  
DFS Implementation**

Next Post »

**Matrix Chain  
Multiplication | (DP-  
48)**

Load Comments

---

Copyright © 2023 takeuforward | All rights reserved