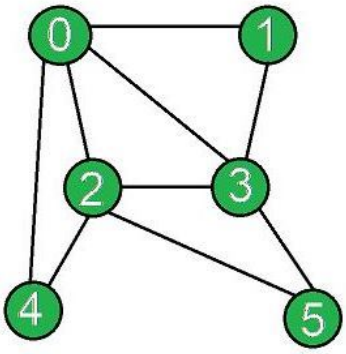
GRAPH CONCEPTS

GRAPH REPRESENTATION

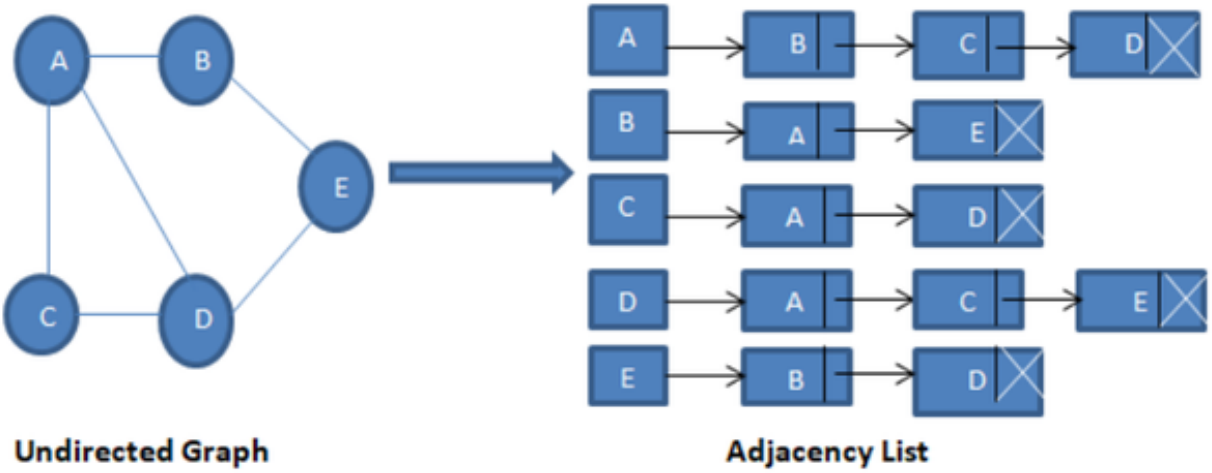
1.Adjacency Matrix



	0	1	2	3	4	5
0	0	1	1	1	1	0
1	1	0	0	1	0	0
2	1	0	0	1	1	1
3	1	1	1	0	0	1
4	1	0	1	0	0	0
5	0	0	1	1	0	0

- Undirected graph is symmetric about main diagonal.
- T.C = O(v*v) huge time complexity

2.Adjacency List



```
unordered_map<int,vector<int>>adj;
```

Just create adjacency list

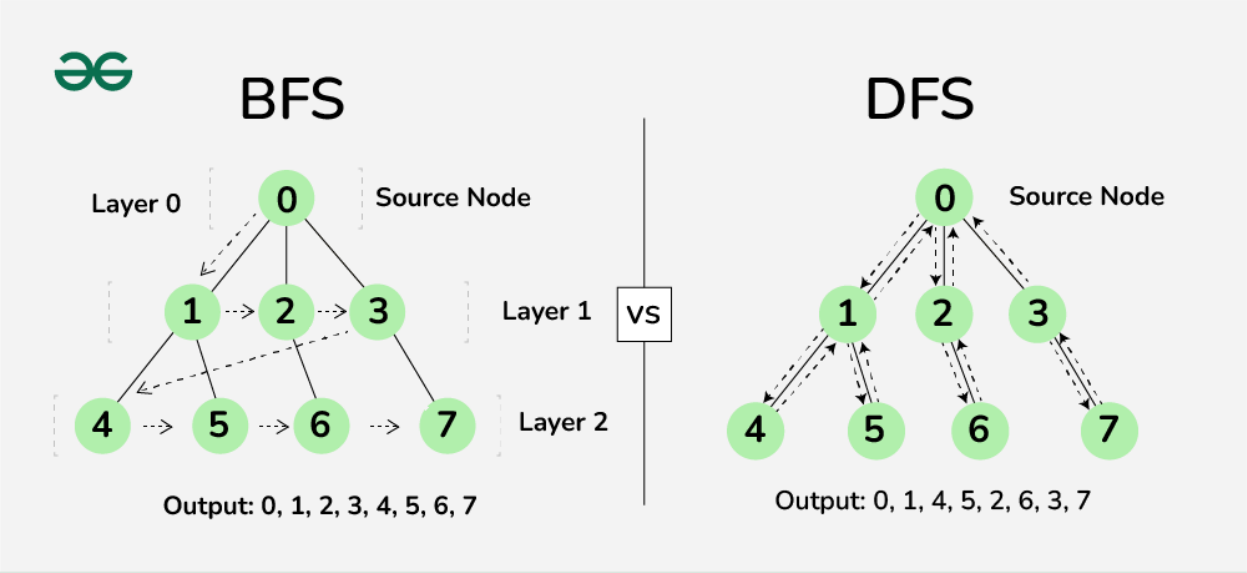
Course Schedule - LeetCode

Can you solve this real interview question? Course Schedule - There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1. You are given an array

https://leetcode.com/problems/course-schedule/description/


```
class Solution {
public:
    bool canFinish(int numCourses, vector<vector<int>>& prerequ
      //graph banao using adjacency list.
       unordered_map<int,vector<int>> adj;
       for(vector<int>& vec : prerequisites){
       int v=vec[1];
       int u=vec[0];
       adj[u].push_back(v);
      }
};
//u---> starting point
//v---->ending point
```

GRAPH TRAVERSAL



DFS (Depth First Search)

Since graphs are similar to trees but cyclic in nature the same vertices may get visited, so we create a bool array named visited[], that tells whether that node was previously visited or not.

DFS of Graph | Practice | GeeksforGeeks

You are given a connected undirected graph. Perform a Depth First Traversal of the graph.Note: Use the recursive approach to find the DFS traversal of the graph starting from the

https://www.geeksforgeeks.org/problems/depth-first-traversal-for-a-graph/1

```
class Solution {
public:
    // Function to perform DFS traversal from a given node
    void DFS(int u, vector<int> adj[], vector<int>& ans, vector<
        // Mark the current node as visited
        visited[u] = true;
        // Add the current node to the result
        ans.push_back(u);
```

```
        // Traverse all adjacent vertices
        for (int v : adj[u]) {
            // If the adjacent vertex is not visited, call DFS r
            if (!visited[v]) {
                DFS(v, adj, ans, visited);
            }
        }
    }

    // Function to return a list containing the DFS traversal of
    vector<int> dfsOfGraph(int V, vector<int> adj[]) {
        vector<bool> visited(V, false);  // Initialize the visit
        vector<int> ans;

        // Perform DFS from each component's starting node
        for (int i = 0; i < V; i++) {
            if (!visited[i]) {
                DFS(i, adj, ans, visited);
            }
        }

        return ans;
    }
};
```

T.C = O(V+E)

S.C = O(V)

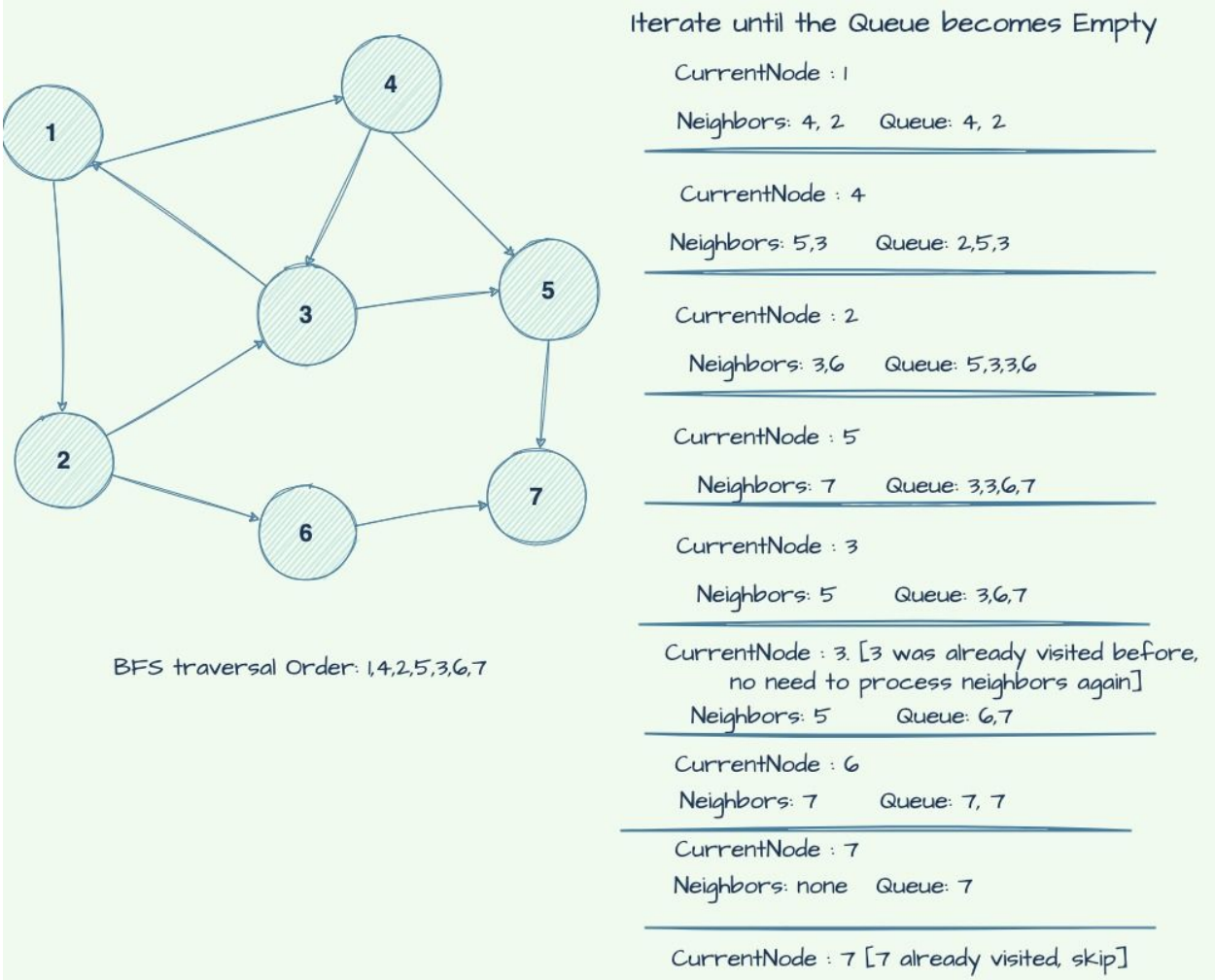
```
vector<int> adj[]>
// Agar yeh given hai question mein samjho ki woh adjacency list
```

Algorithm:

- Start with a Node and maintain a visited array.
- write dfs function DFS(0, adj, ans, visited);
- Now make that node visited and push it in vector.
- Now traverse through it adjacency list and repeat the recursive dfs function for it.Check that it is not visited.

BFS (Breadth First Search)

BFS is like corona-virus. Aaju baju ke nodes mein felta hai



Make use of Queue data structure.

ALGORITHM

- Choose the first node.
- Push the adjacent nodes in the queue by looking at the adjacency list.
- Then print that element in answer vector.
- Mark it visited in the visited array.
- Then repeat the above steps as per the elements present in the queue.
- Follow the above steps till the time the queue is not empty.

BFS of graph | Practice | GeeksforGeeks

Given a directed graph. The task is to do Breadth First Traversal of this graph starting from 0.Note: One can move from node u to node v only if there's an edge from u to v. Find

https://www.geeksforgeeks.org/problems/bfs-traversal-of-graph/1

```
class Solution {
  public:
    // Function to return Breadth First Traversal of given graph
    vector<int> bfsOfGraph(int V, vector<int> adj[]) {
        queue<int>q;
        vector<int>visited(V,false);
        int u;
        q.push(0); //as the 1st node is 0.
        visited[0]=true;
        vector<int>ans;

        while(!q.empty()){
            u=q.front();
            q.pop();
```

```
            ans.push_back(u);

            for(int v=0;v<adj[u].size();v++){ //go through the a
                if(!visited[adj[u][v]]){  //check if not visited
                    visited[adj[u][v]]=true;  //make it visited
                    q.push(adj[u][v]);  //push that node in queu
                }
            }
        }
        return ans;
    }
};
```

T.C = O(V+2E) visiting the node and then the adjacent node twice

S.C = O(V) space taken by the queue.

CYCLE DETECTION IN UNDIRECTED GRAPH

DFS

Approach:

1. **DFS Traversal**: For each node, perform DFS.
2. **Visited Array**: Keep track of visited nodes.
3. **Parent Node**: Track the parent of the current node to avoid detecting trivial cycles (a node pointing back to its parent).

Undirected Graph Cycle | Practice | GeeksforGeeks

Given an undirected graph with V vertices labelled from 0 to V-1 and E edges, check whether it contains any cycle or not. Graph is in the form of adjacency list where adj[i] contains all

https://www.geeksforgeeks.org/problems/detect-cycle-in-an-undirected-graph/1

```
class Solution {
 public:
  // Function to detect a cycle in an undirected graph using DFS
  bool DFS(int u, int parent, vector<int> adj[], vector<bool>& v
    // Mark the current node as visited
    visited[u] = true;

    // Traverse all adjacent vertices
    for (int v : adj[u]) {
      // If the adjacent vertex is not visited, call DFS recursi
      if (!visited[v]) {
        if (DFS(v, u, adj, visited)) {
          return true;  // Cycle detected
        }
      }
      // If the adjacent vertex is visited and is not the parent
      else if (v != parent) {
        return true;
      }
    }

    return false;
  }

  // Function to check if a cycle exists in the graph
  bool isCycle(int V, vector<int> adj[]) {
    vector<bool> visited(V, false);

    // Check for cycle in each component of the graph
    for (int i = 0; i < V; ++i) {
      if (!visited[i]) {
        if (DFS(i, -1, adj, visited)) {
          return true;  // Cycle found
        }
      }
```

```
    }

    return false;
  }
};
```

```
0 -> 1
1 -> 0 -> 2 -> 4
2 -> 1 -> 3
3 -> 2 -> 4
4 -> 1 -> 3
```

BFS

- Same logic as that of BFS traversal. Just use pair to store the node and its parent.
- Make use of a queue.
- If there is a node, and if its neighbor is already visited and it is not a parent then cycle is detected

```
class Solution {
  public:
    bool isCycleBFS(vector<int> adj[], int V, int start, vector<
    queue<pair<int, int>> que; // Queue to store pairs of (node,

    que.push({start, -1}); // Start with the initial node and no
    visited[start] = true; // Mark the starting node as visited

    // Perform BFS
    while (!que.empty()) {
        int curr = que.front().first;  // Current node
        int parent = que.front().second; // Parent of the curren
        que.pop(); // Remove the current node from the queue
```

```
        // Traverse all adjacent vertices
        for (auto v : adj[curr]) {
            // If the adjacent vertex is not visited
            if (!visited[v]) {
                // Mark it as visited and enqueue it with the cu
                que.push({v, curr});
                visited[v] = true;
            }
            // If the adjacent vertex is visited and is not the
            else if (v != parent) {
                return true; // Cycle detected
            }
        }
    }

    return false; // No cycle detected
}

    // Function to detect cycle in an undirected graph.
    bool isCycle(int V, vector<int> adj[]) {

        vector<bool> visited(V, false);

        for(int i = 0; i<V; i++) {
            if(!visited[i] && isCycleBFS(adj, V, i, visited)) {
                return true;
            }
        }

        return false;

    }
};
```

CYCLE DETECTION IN DIRECTED GRAPH

DFS

- If a node appears more than once in a path, then we will call it cycle.

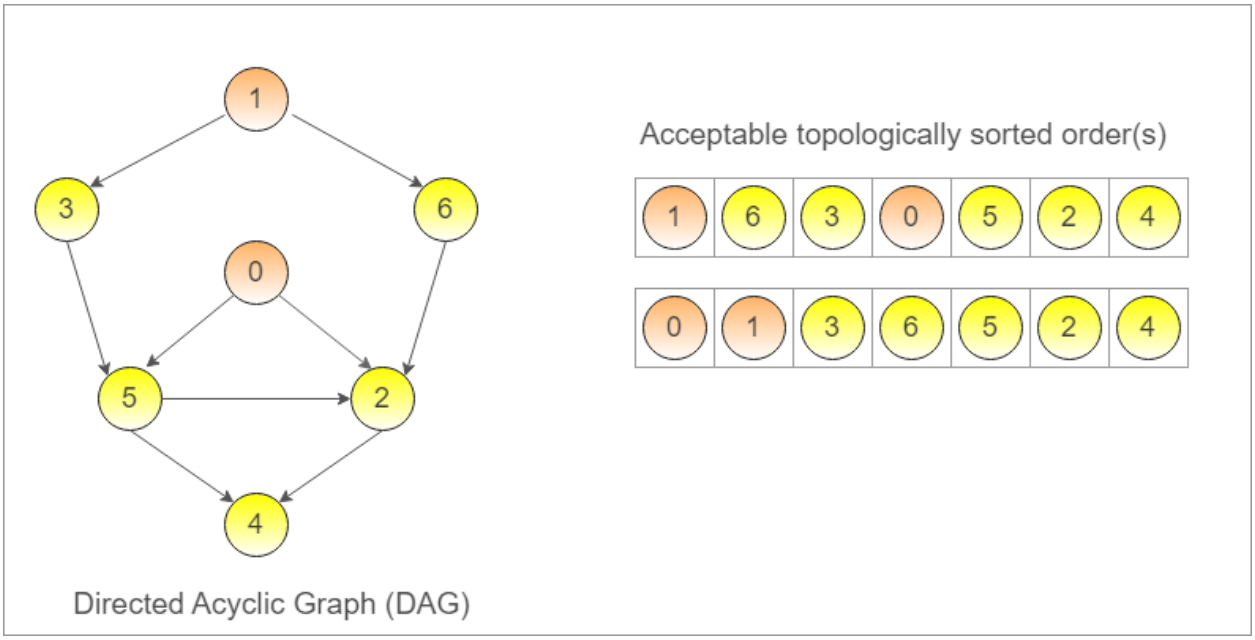
TOPOLOGICAL SORT (DFS)

- Implemented in DAG (Directed Acyclic Graph)
- There can be multiple Topological sort sequence
- Same like DFS just store the order in stack.
- In Topological sort, u —→ v, maintain this order.
-

Topological sort | Practice | GeeksforGeeks

Given an adjacency list for a Directed Acyclic Graph (DAG) where adj_list[i] contains a list of all vertices j such that there is a directed edge from vertex i to vertex j, with V

https://www.geeksforgeeks.org/problems/topological-sort/1



- same algorithm like DFS, just when for loop ends put that element in a stack.

> “pehle mere bachho(v) ko stack mein daalo uske baad mujhe(u) stack mein daalo”

```
class Solution
{
    public:
    //Function to return list containing vertices in Topological
    void DFS(vector<int> adj[], int u, vector<int>& visited, sta
        visited[u]=true;

        for(int &v: adj[u]){
            if(!visited[v]){
                DFS(adj, v, visited, st);
            }
        }
        st.push(u);
    }
```

```
    vector<int> topoSort(int V, vector<int> adj[])
    {
        vector<int>visited(V,false);
        vector<int>ans;
        stack<int>st;

        for(int i=0;i<V;i++){ //bcz the graph can be disconnecte
            if(!visited[i]){
                DFS(adj, i, visited, st);
            }
        }

        while(!st.empty()){
            ans.push_back(st.top());
            st.pop();
        }
        return ans;
    }
};
```

TOPOLOGICAL SORT (BFS) KAHN'S ALGORITHM

- make use of "indegree" concept.
- `"jiska indegree 0 hua, pehle usse print karo"`
- `"jaise hi indegree 0 ho gaya, queue mein daalo"`
- Approach:
- `Store indegree of each node in a vector`
- `Fill queue with nodes with indegree 0`
- `simple BFS`

```
//{ Driver Code Starts
#include <bits/stdc++.h>
using namespace std;

// } Driver Code Ends
class Solution
{
    public:
    //Function to return list containing vertices in Topological
    vector<int> topoSort(int N, vector<int> adj[])
    {
        vector<int>indegree(N,0);
        queue<int>q;

        //Store indegree of each node in a vector
        for(int u=0;u<N;u++){
            for(int& v:adj[u]){
                indegree[v]++;
            }
        }

        //Fill queue with nodes with indegree 0
        for(int i=0;i<N;i++){
            if(indegree[i]==0){
                q.push(i);
            }
        }

        //simple BFS
        vector<int>ans;
        while(!q.empty()){
            int u=q.front();
            ans.push_back(u);
            q.pop();
```

```
            for(int& v:adj[u]){
                indegree[v]--;

                if(indegree[v]==0){
                    q.push(v);
                }
            }
        }

        return ans;
    }
};
```

CYCLE DETECTION IN DIRECTED GRAPH

BFS

- Kahn’s Algorithm is applicable on DAG. So if we couldn't find Topological sort that means we can say that there exists a cycle.

```
//{ Driver Code Starts
#include <bits/stdc++.h>
using namespace std;

// } Driver Code Ends

// } Driver Code Ends
class Solution {
  public:
    // Function to detect cycle in a directed graph.
    bool isCyclic(int N, vector<int> adj[]) {
        vector<int> indegree(N, 0);
        queue<int> q;
        int cnt = 0;
```

```
        // Store indegree of each node in a vector
        for (int u = 0; u < N; u++) {
            for (int& v : adj[u]) {
                indegree[v]++;
            }
        }

        // Fill queue with nodes with indegree 0
        for (int i = 0; i < N; i++) {
            if (indegree[i] == 0) {
                q.push(i);
            }
        }

        // Simple BFS
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            cnt++;  // Increment cnt when processing a node

            for (int& v : adj[u]) {
                indegree[v]--;

                if (indegree[v] == 0) {
                    q.push(v);
                }
            }
        }

        // If cnt is equal to N, no cycle detected [visted all n
        return cnt != N;
    }
};
```

Number of Provinces - LeetCode

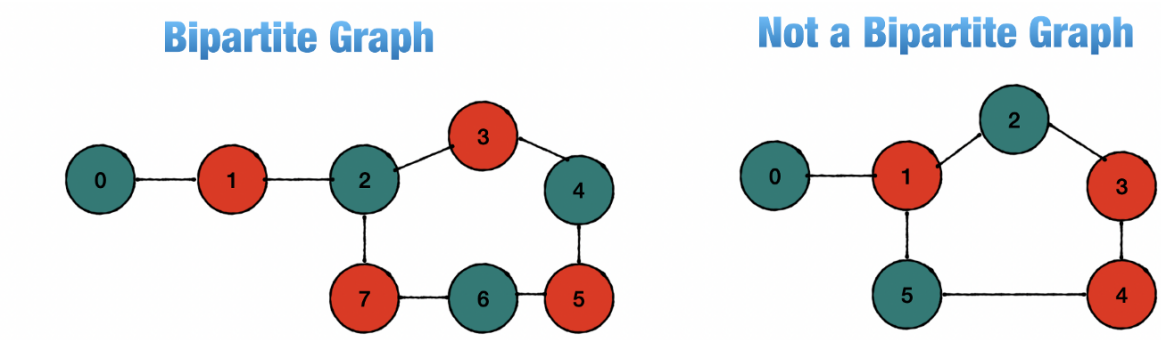
Can you solve this real interview question? Number of Provinces - There are n cities. Some of them are connected, while some are not. If city a is connected directly with city b,

https://leetcode.com/problems/number-of-provinces/description/


```
class Solution {
public:
    void dfs(unordered_map<int, vector<int>> &adj, int u, vecto
        visited[u] = true;

        //Visit neighbours
        for(int &v : adj[u]) {
            if(!visited[v]) {
                dfs(adj, v, visited);
            }
        }
    }


    int findCircleNum(vector<vector<int>>& isConnected) {
        int n = isConnected.size();

        unordered_map<int, vector<int>> adj;

        for(int i = 0; i<n; i++) {
            for(int j = 0; j<n; j++) {
                if(isConnected[i][j] == 1) {
                    adj[i].push_back(j);
                    adj[j].push_back(i);
                }
            }
        }

        vector<bool> visited(n, false);
```

```
        int count = 0;

        for(int i = 0; i<n; i++) {
            if(!visited[i]) {
                count++;
                dfs(adj, i, visited);
            }
        }

        return count;

    }
};
```

BIPARTITE GRAPH



- ALGORITHM

1. Initialize a color array for all vertices, setting each to `1` (uncolored).
2. Start DFS from any uncolored vertex and assign it color `0`.
3. For each vertex, color all its uncolored adjacent vertices with the opposite color.
4. If you find an adjacent vertex with the same color, return `false` (not bipartite).
5. Repeat the DFS for any remaining uncolored vertices in the graph.

6. If all vertices are colored without conflicts, return `true` (graph is bipartite).

- DFS

Bipartite Graph | Practice | GeeksforGeeks

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. Know more about Bipartite Graph: -

https://www.geeksforgeeks.org/problems/bipartite-graph/1

```
class Solution {
public:

    bool checkBipartiteDFS(vector<int>adj[], int curr, vector<i
        color[curr] = currColor; //color kardiya curr node ko

        //ab jaate hain adjacent nodes par
        for(int &v : adj[curr]) {

            if(color[v] == color[curr])
                return false;

            if(color[v] == -1) { //never colored (never visited

                int colorOfV = 1 - currColor;

                if(checkBipartiteDFS(adj, v, color, colorOfV) ==
                    return false;
            }

        }

        return true;
    }
```

```
    bool isBipartite(int V, vector<int>adj[]){

        vector<int> color(V, -1); //no node colored in the start

        //red = 1
        //green = 0

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

        return true;

    }

};
```

- BFS

Is Graph Bipartite? - LeetCode

Can you solve this real interview question? Is Graph Bipartite? - Level up your coding skills and quickly land a job. This is the best place to expand your knowledge and get prepared for

https://leetcode.com/problems/is-graph-bipartite/description/

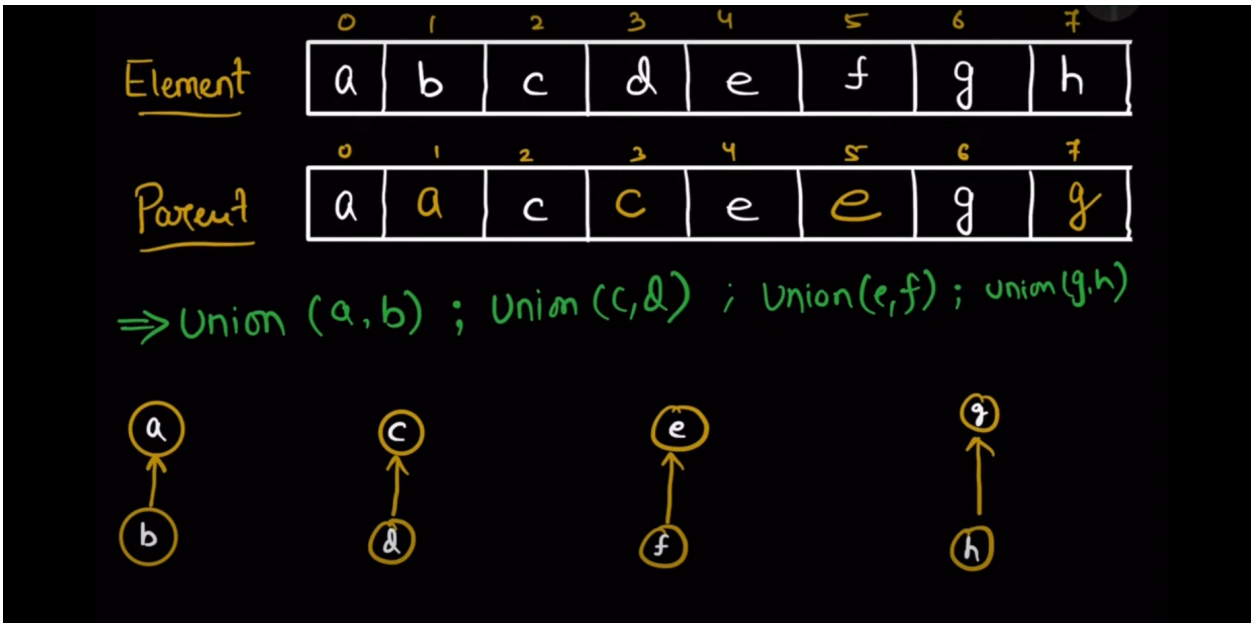

```
class Solution {
public:

    bool checkBipartiteBFS(vector<int>adj[], int curr, vector<in
```

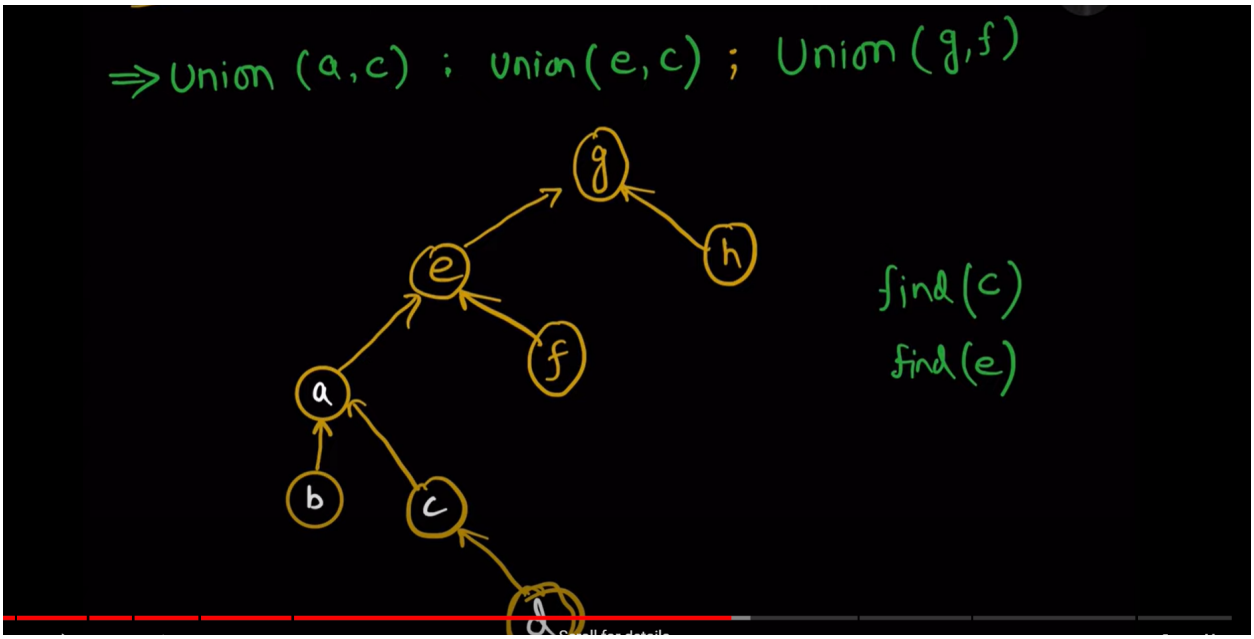
```
        color[curr] = currColor; //color kardiya curr node ko

        queue<int> que;
        que.push(curr);

        while(!que.empty()) {
            int u = que.front();
            que.pop();

            for(int &v : adj[u]) {
                if(color[v] == color[u]) {
                    return false;
                } else if(color[v] == -1) {
                    color[v] = 1 - color[u];
                    que.push(v);
                }
            }
        }

        return true;
    }

    bool isBipartite(int V, vector<int>adj[]){

        vector<int> color(V, -1); //no node colored in the start

        //red = 1
        //gree = 0

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

```
            return true;

        }

};
```

DISJOINT SET UNION (DSU)

- Operation 1 : Combine 2 given sets. [UNION]
- Operation 2 : Tell if 2 members belong to same set or not. [FIND]





Code

```
//FIND FUNCTION
int find(int i, vector<int> parent) {
  if (i == parent[i]) { //yahi leader hai
    return i;
  }
  return find(parent[i], parent);
}

//UNION FUNCTION
void union(int x, int y, vector<int> parent) {
  int x_parent = find(x, parent);
  int y_parent = find(y, parent);

  if (x_parent != y_parent) {
    parent[x_parent] = y_parent;
  }
}
```

```
//but the problem is that the tree will be large enough for larg
//so we will be using Rank and Path Compression
```

RANK AND PATH COMPRESSION

mujhe khud itna samjh nhi aaya hai 😂

```
vector<int> parent;
vector<int> rank;

int find (int x) {
    if (x == parent[x])
        return x;

    return parent[x] = find(parent[x]);
}

void Union (int x, int y) {
    int x_parent = find(x);
    int y_parent = find(y);

    if (x_parent == y_parent)
        return;

    if(rank[x_parent] > rank[y_parent]) {
        parent[y_parent] = x_parent;
    } else if(rank[x_parent] < rank[y_parent]) {
        parent[x_parent] = y_parent;
    } else {
        parent[x_parent] = y_parent;
        rank[y_parent]++;
    }
}
```

Detect Cycle using DSU