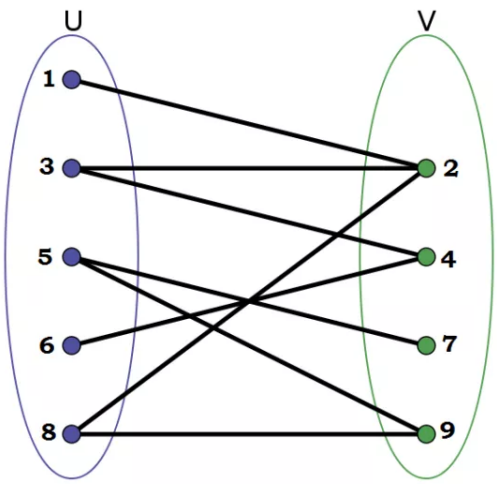
Determine if a given graph is Bipartite Graph using DFS

Given a graph, determine if given graph is bipartite graph using DFS. A bipartite graph (or bigraph) is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V.

Below graph is a Bipartite Graph as we can divide it into two sets U and V with every edge having one end point in set U and the other in set V



It is possible to test whether a graph is bipartite or not using Depth-first search algorithm. There are two ways to check for Bipartite graphs –

1. A graph is bipartite if and only if it is 2-colorable.
2. A graph is bipartite if and only if it does not contain an odd cycle.

In previous post, we have checked if the graph contains an odd cycle or not using BFS. Now using DFS, we will check if the graph is **2-colorable** or not.

The main idea is to assign to each vertex the color that differs from the color of its parent in the depth-first search tree, assigning colors in a preorder traversal of the depth-first-search tree. If there exists an edge connecting current vertex to a previously-colored vertex with the same color, then we can say that the graph is not bipartite.

C++ implementation –

```
#include <bits/stdc++.h>
using namespace std;
// Number of vertices in the graph
#define N 10

```

```cpp
// data structure to store graph edges
struct Edge {
    int src, dest;
};

// class to represent a graph object
class Graph
{
public:
    // A array of vectors to represent adjacency list
    vector<int> adjList[N];

    // Constructor
    Graph(vector<Edge> edges)
    {
        // add edges to the undirected graph
        for (int i = 0; i < edges.size(); i++)
        {
            int src = edges[i].src;
            int dest = edges[i].dest;

            adjList[src].push_back(dest);
            adjList[dest].push_back(src);
        }
    }
};

// Perform DFS on graph starting from vertex v
bool DFS(Graph const &graph, int v, vector<bool> &discovered,
        vector<int> &color)
{
    // do for every edge (v -> u)
    for (int u : graph.adjList[v])
    {
        // if vertex u is explored for first time
        if (discovered[u] == false)
        {
            // mark current node as discovered
            discovered[u] = true;

            // set color as opposite color of parent node
            color[u] = !color[v];

            // if DFS on any subtree rooted at v
            // we return false
            if (DFS(graph, u, discovered, color) == false)
                return false;
        }
        // if the vertex is already been discovered and color of ver
        // u and v are same, then the graph is not Biparte
        else if (color[v] == color[u])
            return false;
    }

    return true;
}

// Determine if a given graph is Bipartite Graph using DFS
int main()
{
    // vector of graph edges as per above diagram
    vector<Edge> edges = {
        {1, 2}, {2, 3}, {2, 8}, {3, 4}, {4, 6}, {5, 7},
        {5, 9}, {8, 9}, {2, 4}
        // if we remove 2->4 edge, graph is becomes Biparte
    };

    // create a graph from edges
    Graph graph(edges);

    // stores vertex is discovered or not
    vector<bool> discovered(N);

    // stores color 0 or 1 of each vertex in BFS
    vector<int> color(N);

    // mark source vertex as discovered and
    // set its color to 0
    discovered[0] = true, color[0] = 0;

    // start DFS traversal from any node as graph
    // is connected and undirected
    if (DFS(graph, 1, discovered, color))
        cout << "Biparte Graph";
    else
        cout << "Not a Biparte Graph";

    return 0;
```

```
94 }
```

Download Run Code

Output:

```
Not a Biparte Graph
```

The time complexity of above solution is $O(n + m)$ where n is number of vertices and e is number of edges in the graph. Please note that $O(m)$ may vary between $O(1)$ and $O(n^2)$, depending on how dense the graph is.

References: https://en.wikipedia.org/wiki/Bipartite_graph

Thanks for reading.

Please use ideone or C++ Shell or any other online compiler link to post code in comments.
Like us? Please spread the word and help us grow. Happy coding 🙂

Graph
DFS

← Find Minimum and Maximum element in an array by doing minimum comparisons

Cryptography | DES implementation in C →

Leave a Reply