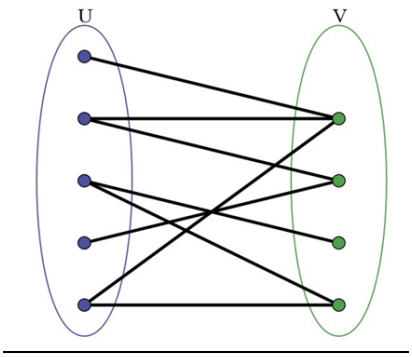
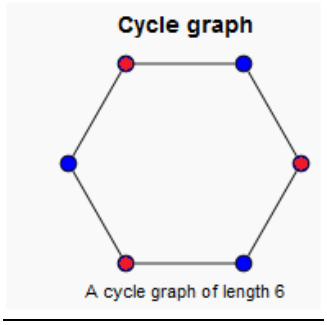
Check whether a given graph is Bipartite or not

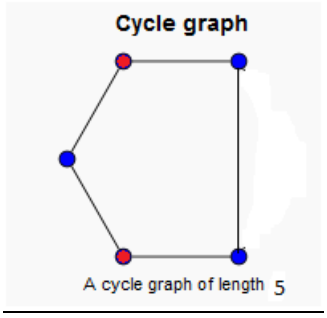
A Bipartite Graph is a graph whose vertices can be divided into two independent sets, U and V such that every edge (u, v) either connects a vertex from U to V or a vertex from V to U. In other words, for every edge (u, v), either u belongs to U and v to V, or u belongs to V and v to U. We can also say that there is no edge that connects vertices of same set.



A bipartite graph is possible if the graph coloring is possible using two colors such that vertices in a set are colored with the same color. Note that it is possible to color a cycle graph with even cycle using two colors. For example, see the following graph.



It is not possible to color a cycle graph with odd cycle using two colors.



Algorithm to check if a graph is Bipartite:

One approach is to check whether the graph is 2-colorable or not using backtracking algorithm m coloring problem.

Following is a simple algorithm to find out whether a given graph is Birpartite or not using Breadth First Search (BFS).

1. Assign RED color to the source vertex (putting into set U).
2. Color all the neighbors with BLUE color (putting into set V).
3. Color all neighbor's neighbor with RED color (putting into set U).
4. This way, assign color to all vertices such that it satisfies all the constraints of m way coloring problem where m = 2.

5. While assigning colors, if we find a neighbor which is colored with same color as current vertex, then the graph cannot be colored with 2 vertices (or graph is not Bipartite)

```
// C++ program to find out whether a given graph is Bipartite or not
#include <iostream>
#include <queue>
#define V 4
using namespace std;

// This function returns true if graph G[V][V] is Bipartite, else false
bool isBipartite(int G[][V], int src)
{
    // Create a color array to store colors assigned to all veritces. Vertex
    // number is used as index in this array. The value '-1' of  colorArr[i]
    // is used to indicate that no color is assigned to vertex 'i'.  The value
    // 1 is used to indicate first color is assigned and value 0 indicates
    // second color is assigned.
    int colorArr[V];
    for (int i = 0; i < V; ++i)
        colorArr[i] = -1;

    // Assign first color to source
    colorArr[src] = 1;

    // Create a queue (FIFO) of vertex numbers and enqueue source vertex
    // for BFS traversal
    queue <int> q;
    q.push(src);

    // Run while there are vertices in queue (Similar to BFS)
    while (!q.empty())
    {
        // Dequeue a vertex from queue ( Refer http://goo.gl/35oz8 )
        int u = q.front();
        q.pop();

         // Find all non-colored adjacent vertices
        for (int v = 0; v < V; ++v)
        {
            // An edge from u to v exists and destination v is not colored
            if (G[u][v] && colorArr[v] == -1)
            {
                // Assign alternate color to this adjacent v of u
                colorArr[v] = 1 - colorArr[u];
                q.push(v);
            }

            //  An edge from u to v exists and destination v is colored with
            // same color as u
            else if (G[u][v] && colorArr[v] == colorArr[u])
                return false;
        }
    }

    // If we reach here, then all adjacent vertices can be colored with
    // alternate color
    return true;
}

// Driver program to test above function
int main()
{
    int G[][V] = {{0, 1, 0, 1},
        {1, 0, 1, 0},
        {0, 1, 0, 1},
        {1, 0, 1, 0}
    };
```

```
    isBipartite(G, 0) ? cout << "Yes" : cout << "No";
    return 0;
}
```

Run on IDE

Output:

```
Yes
```

Refer this for C implementation of the same.

Time Complexity of the above approach is same as that Breadth First Search. In above implementation is O(V^2) where V is number of vertices. If graph is represented using adjacency list, then the complexity becomes O(V+E).

Exercise:

1. Can DFS algorithm be used to check the bipartite-ness of a graph? If yes, how?

2. The above algorithm works if the graph is strongly connected. Extend above code to work for graph with more than one component.

References:

http://en.wikipedia.org/wiki/Graph_coloring

http://en.wikipedia.org/wiki/Bipartite_graph