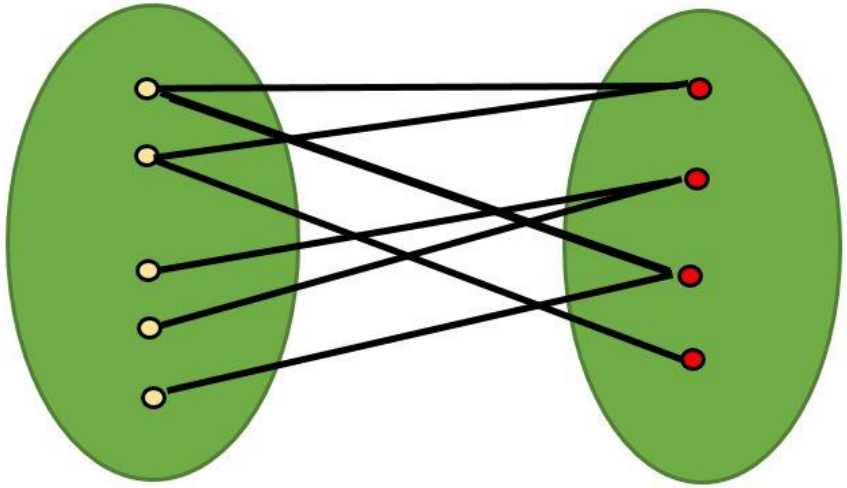
An Application of BFS

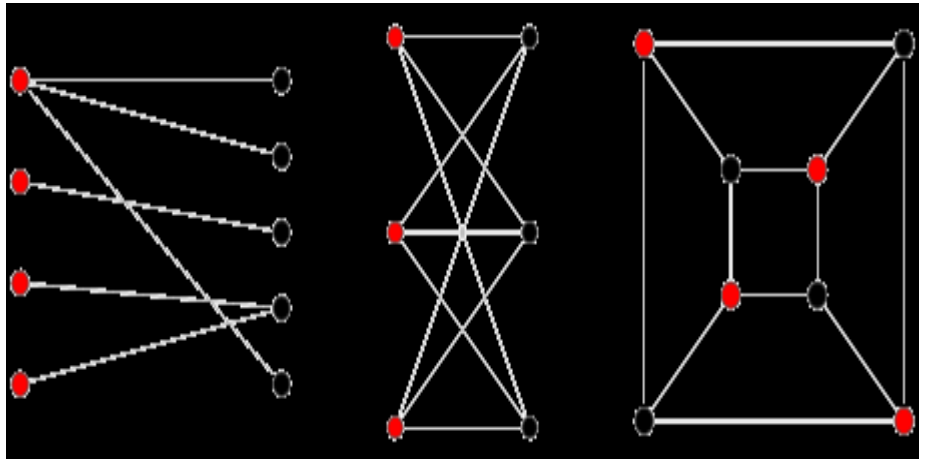
Testing Bipartiteness

Bipartite Graph

- 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, also called a bigraph, is a set of graph vertices decomposed into two disjoint set such that no two graph vertices within the same set are adjacent.





Bipartite Testing

Problem :Determine if a graph G is bipartite

How can we test if G is bipartite?

A simple algorithm using **BFS** to check if a graph is bipartite or not .

Properties:

1. If a graph is a bipartite graph then it'll never contain odd cycles.
2. The subgraphs of a bipartite graph are also bipartite.
3. A bipartite graph is always 2-colorable
4. In an undirected bipartite graph, the degree of each vertex partition set is always equal.

Bipartite Testing

- Bipartite Testing algorithm uses the concept of graph coloring and BFS to determine a given graph is a bipartite or not.
- This algorithm takes the graph and a starting vertex as input. The algorithm returns either the input graph is bipartite or the graph is not a bipartite graph.

The steps of this algorithm are:

•Assign a red color to the starting vertex

•Find the neighbors of the starting vertex and assign a blue color

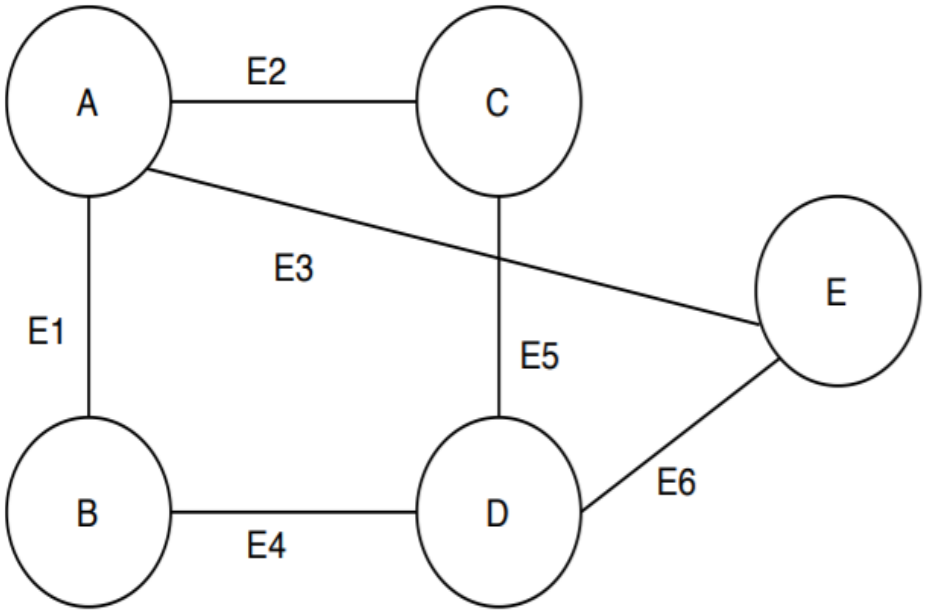
•Find the neighbor's neighbor and assign a red color

•Continue this process until all the vertices in the graph assigned a color

•During this process, if a neighbor vertex and the current vertex have the same color then the algorithm terminates. The algorithm returns that the graph is not a bipartite graph

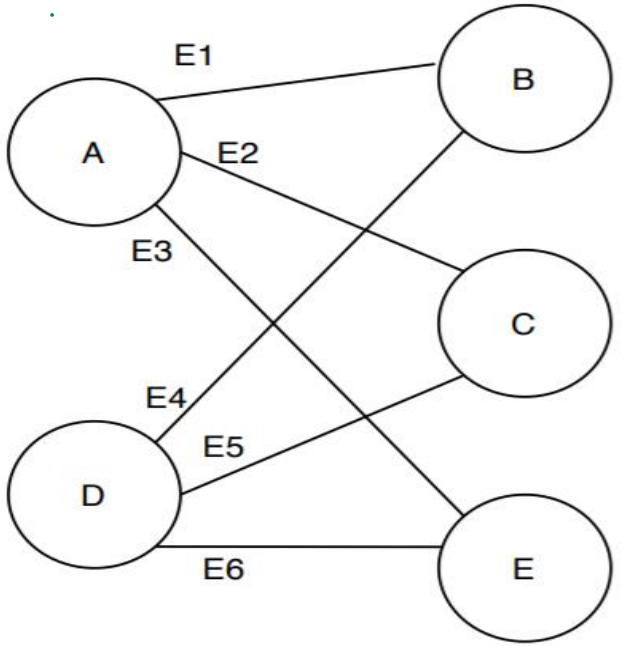
We've used a queue data structure to store and manage all the neighbor vertices.

Bipartite Testing

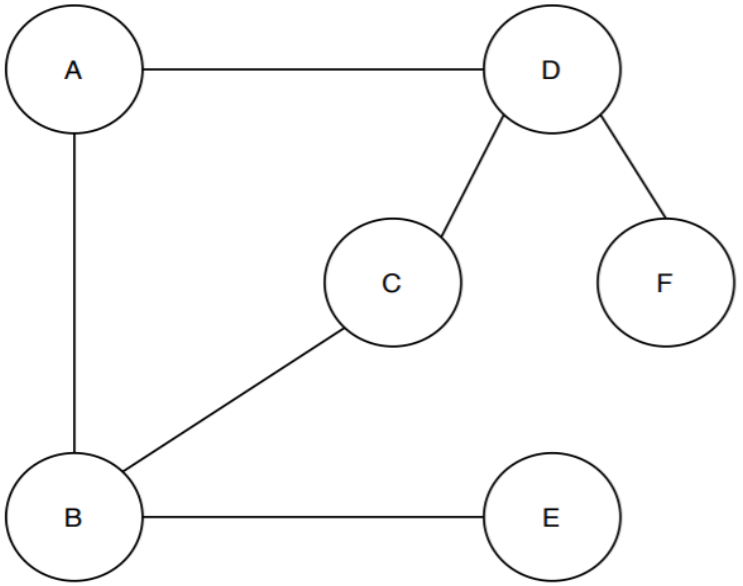


We've taken a sample graph G and vertex set has V={A,B,C,D,E,F} vertices.
Now, to satisfy the definition of a bipartite graph, the vertex set needs to be partitioned into two sets such that each edge joins the two vertex sets.

This is the same graph G, just with a different representation. Here, we partition the vertex set V=(A, B, C, D, E) into two disjoint vertex sets V_1 = (A, D) and V_2 = (B, C, E).

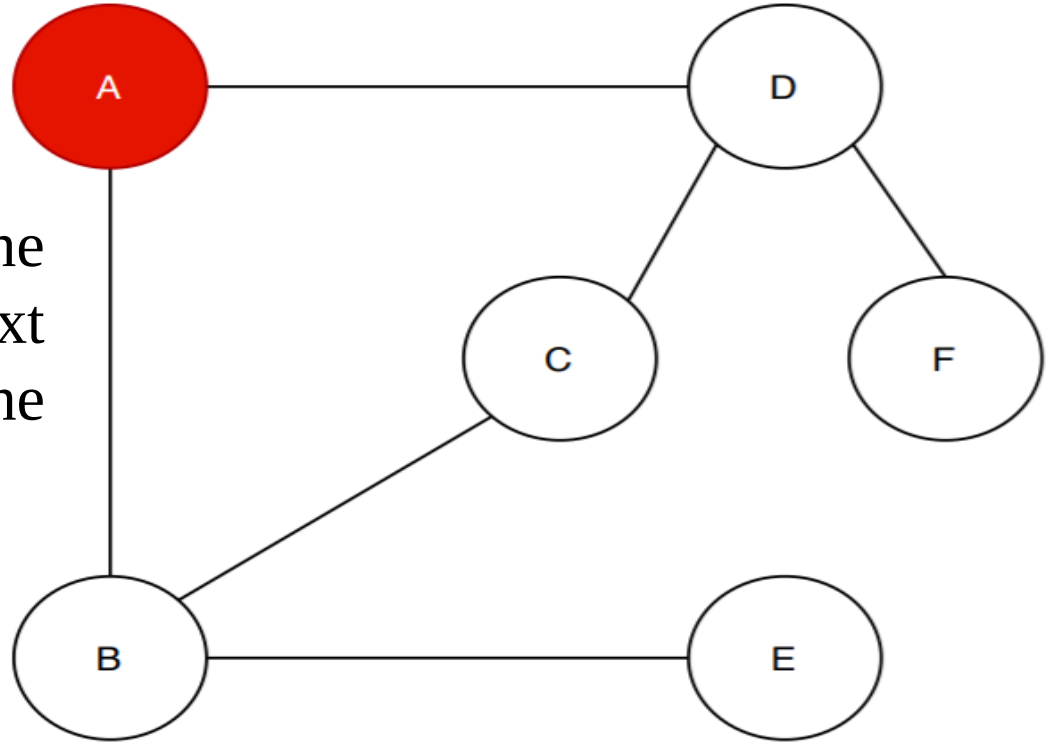


Bipartite Testing

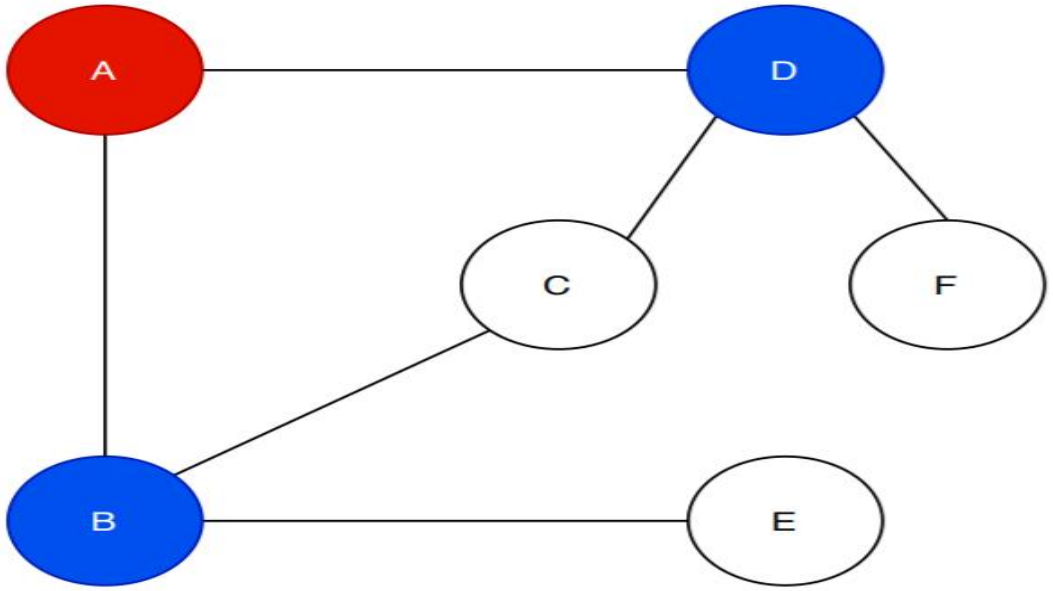


We've taken a sample graph G and vertex set V = (A, B, C, D, E, F) has 6 vertices. Here we're picking the vertex A as the starting vertex. So let's start the first step:

So the first step of the algorithm is to fill the starting vertex with the red color. The next step is to find the neighbor vertices of the vertex A and fill them with the color blue:

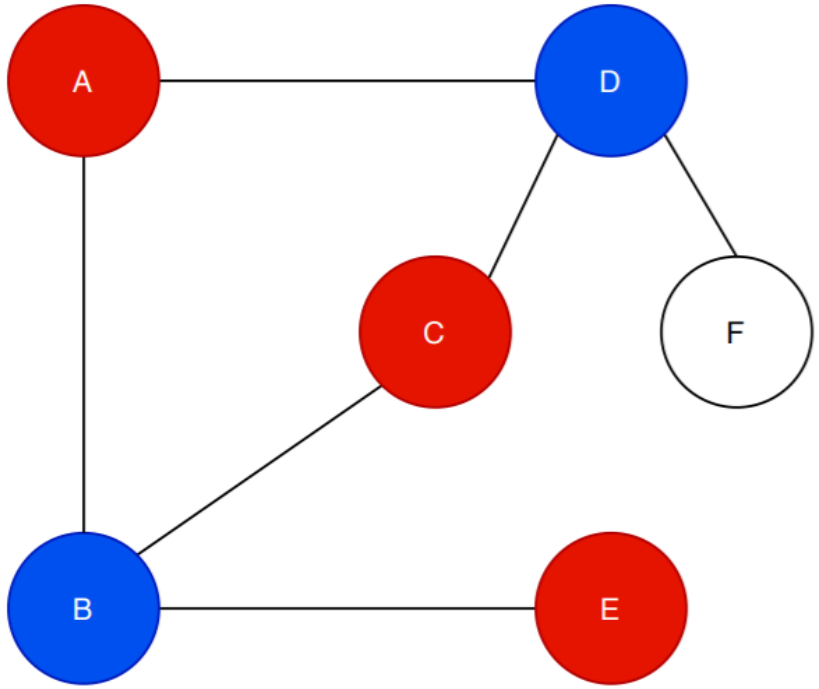


Bipartite Testing

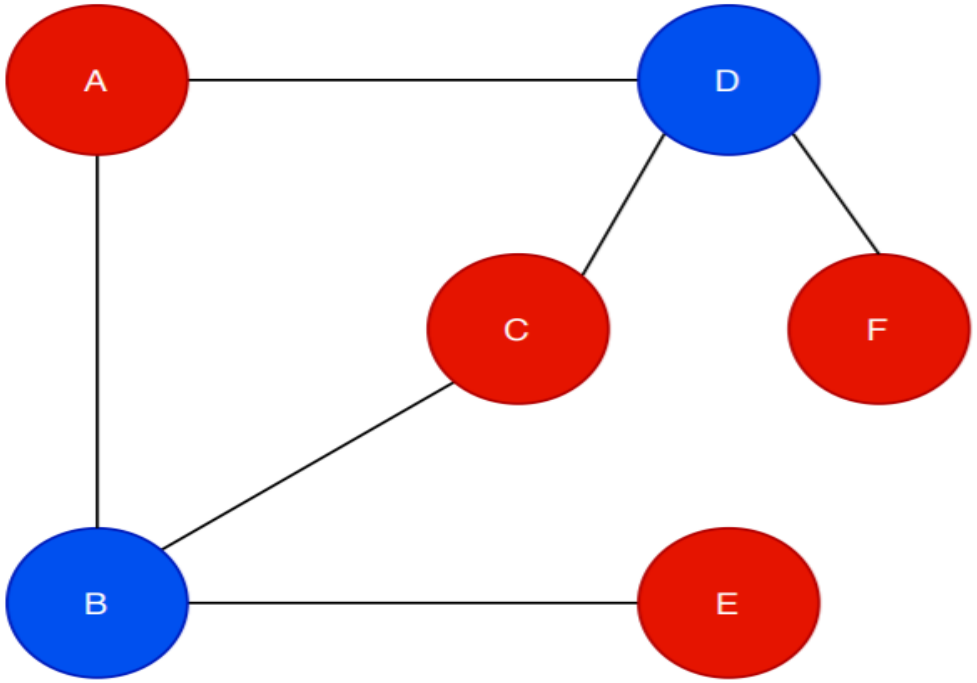


The vertex A has two adjacent vertices: B and D. The algorithm first checks if the adjacent vertices are already filled with color. In our case, they are not filled. The algorithms fill the vertex B and D with the blue color.

We then choose any of the newly filled vertices and find the neighbors. Let's choose the vertex B: the vertex B is adjacent to vertex C and E. Now the algorithm first checks if the vertices are already filled with some color. In our case, the vertices are not filled with color. Next, the algorithm checks the color of the vertex B. As the color of B is blue, so the algorithm fills C and E with red



Bipartite Testing



Checks the adjacent vertices for the D. The vertex D has two adjacent vertices: C and F. Again, the algorithm checks whether the vertices already filled with color or not. In this case, the vertex C is already filled with color. But the vertex F is not filled yet.

The algorithm checks the color of the vertex D. As the vertex D is filled with the blue color, the algorithm fills the vertex F with the red color. The algorithm continues this process until it checks all the vertices and it's neighbors once.

Now, we can see that in the graph G, no adjacent vertices have the same color. Also, we can see two clear partitions of the vertex set: V_1 = (B, D) and V_2 = (A, C, E, F).

Time Complexity Analysis

In the algorithm, first, we traverse each vertex once. Then for each vertex, we visit each neighbor of a vertex once. The algorithm uses BFS for traversing. BFS takes O(V + E) time.

For storing the vertices, we can either use an adjacency matrix or an adjacency list.

Now if we use an adjacency matrix, then it takes O(V ^2) to traverse the vertices in the graph. So, if we use an adjacency matrix, the overall time complexity of the algorithm would be O(V ^2).

On the other hand, an adjacency list takes O(V + E) time to traverse all the vertices and their neighbors in the graph. Therefore, if we use an adjacency list in the algorithm then the overall time complexity will be O(V + E).