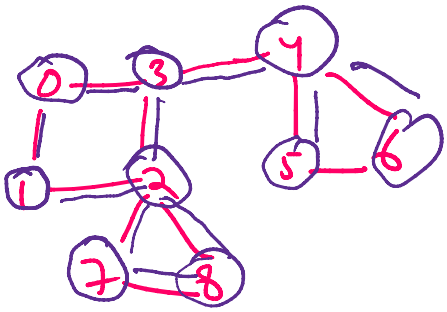
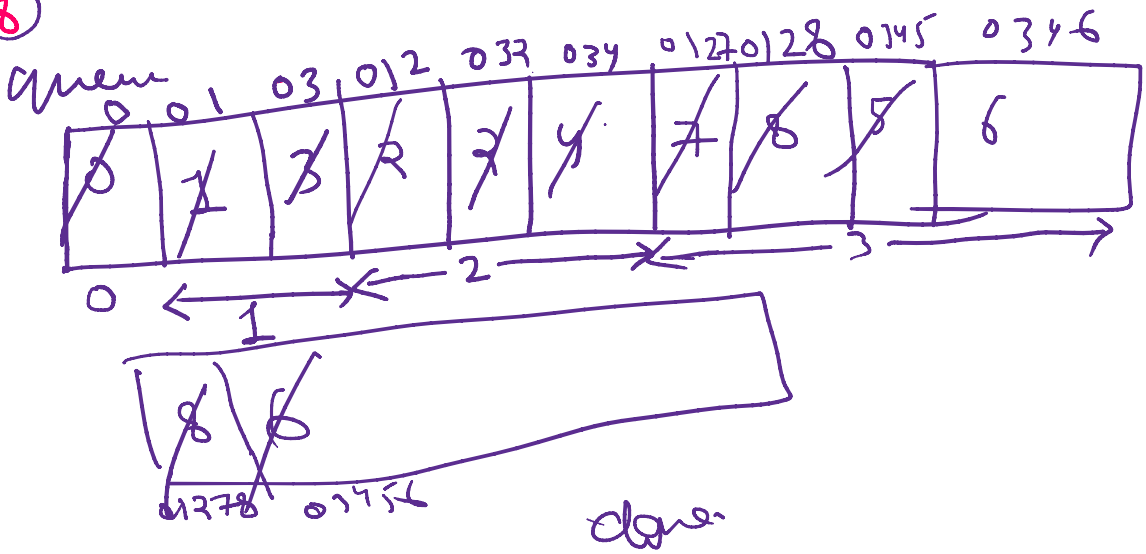
Graph

26 July 2022 20:59

BFS → Jab cycle se kuch kam krana ho.





Complexity :→ O(E).

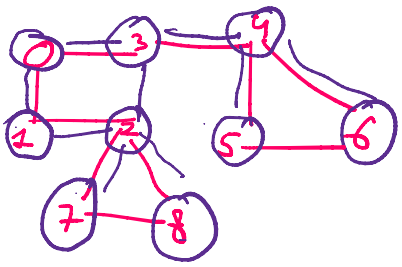
if any vertex in queue came twice that means there is a cycle in the Graph.

BFS try and run

```
void BFS(vector<vector<int>> &Graph,int src){
    queue<int> PendingNodes;
    PendingNodes.push(src);
```

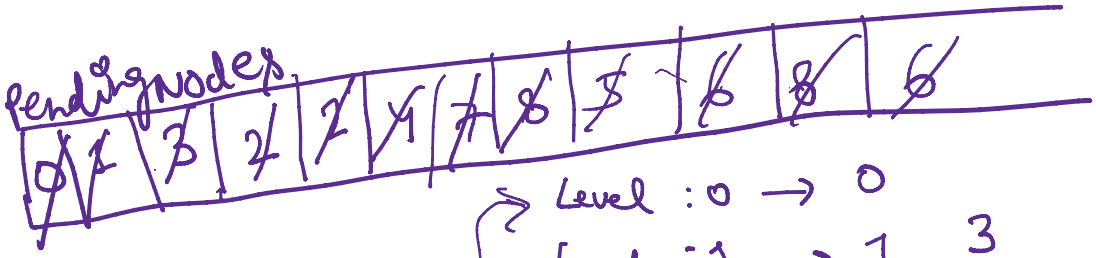
BFS try and run

```
void BFS(vector<vector<int>> &Graph,int src){
    queue<int> PendingNodes;
    PendingNodes.push(src);
    vector<int> visited(Graph.size(),false);
    PendingNodes.push(src);
    int level = 0;
    while(!PendingNodes.empty()){
        int size = PendingNodes.size();
        cout<<"Level : "<<level<<"--->";
        while(size-->0){
            int Front = PendingNodes.front();
            PendingNodes.pop();
            if(!visited[Front]){
                cout<<" "<<Front;
                visited[Front] = true;
                for(int i = 0 ; i < Graph[Front].size() ; i++){
                    int v = Graph[Front][i];
                    if(!visited[v]){
                        PendingNodes.push(v);
                    }
                }
            }
        }
        cout<<"\n";
        level++;
    }
}
```



level= 3
size= ~~4~~ 3 2 1 0
front= ~~7~~ 8 ~~8~~ 6

pendingnodes



Level : 0 → 0
Level : 1 → 1 3
Level : 2 → 2 4
Level : 3 → 7 8 5 6
Level : 4 →

BFS when no need of cycle.

```
void BFS_without_cycle(vector<vector<int>> &Graph,int src){
    queue<int> PendingNodes;
    PendingNodes.push(src);
    vector<int> visited(Graph.size(),false);
    PendingNodes.push(src);
    visited[src] = true;
    int level = 0;
    while(!PendingNodes.empty()){
        int size = PendingNodes.size();
        cout<<"Level : "<<level<<"--->";
        while(size-->0){
            int Front = PendingNodes.front();
            PendingNodes.pop();
            cout<<" "<<Front;
            visited[Front] = true;
            for(int i = 0 ; i < Graph[Front].size() ; i++){
                int v = Graph[Front][i];
                if(!visited[v]){
                    PendingNodes.push(v);
                    visited[v] = true;
                }
            }
        }
        cout<<"\n";
        level++;
    }
}
```

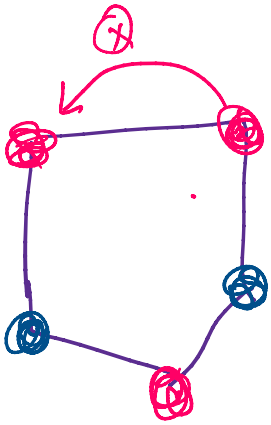
In this case at max O(v) nodes will be there in the queue.

Bipartite Graph

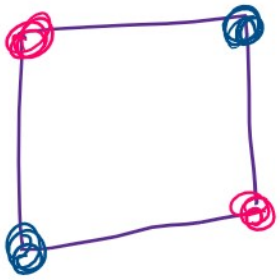
Set A

Set B.

if there is not any edge between elements of same set then graph is bipartite.

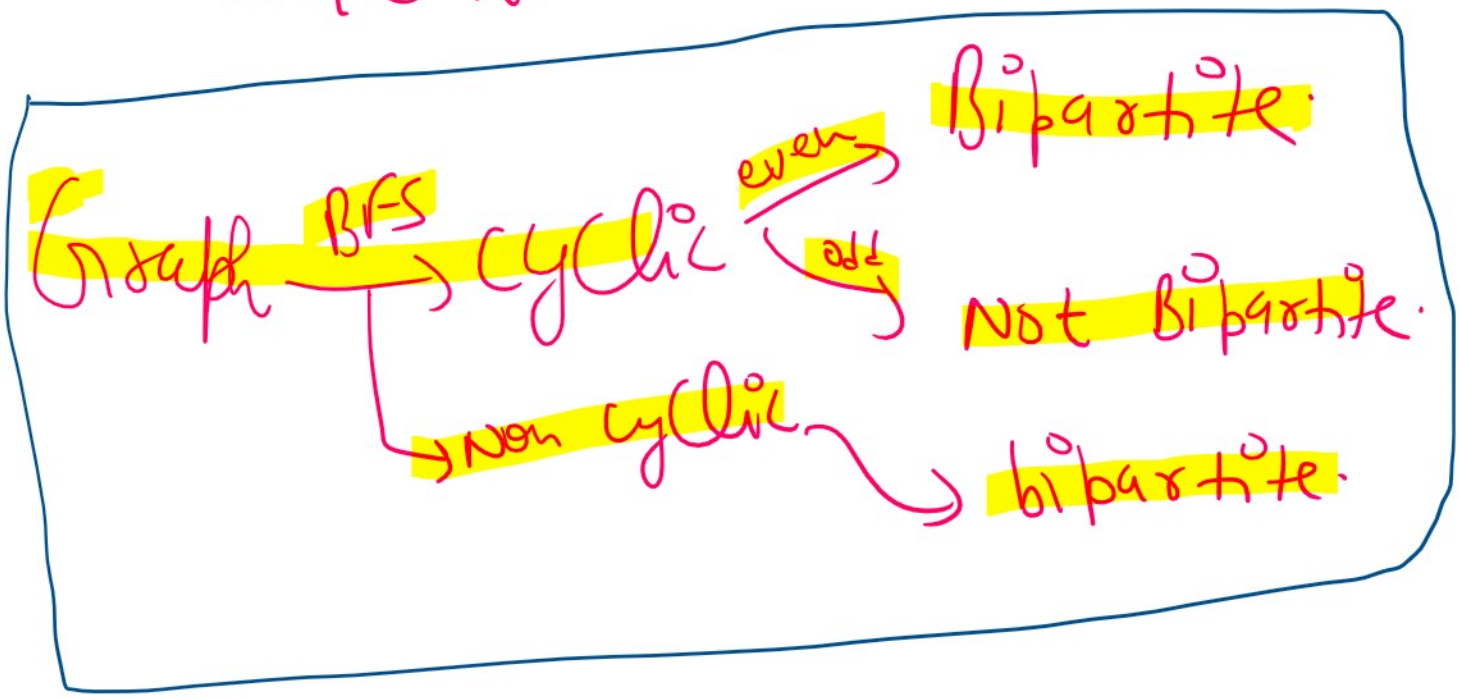


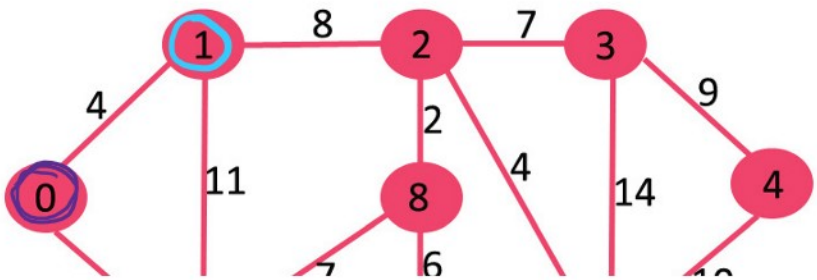
→ not a bipartite Graph

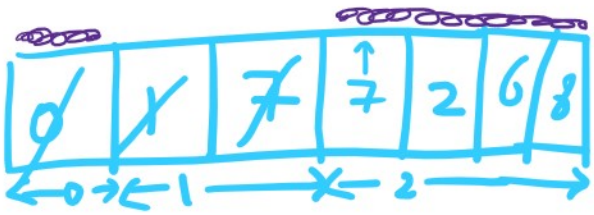


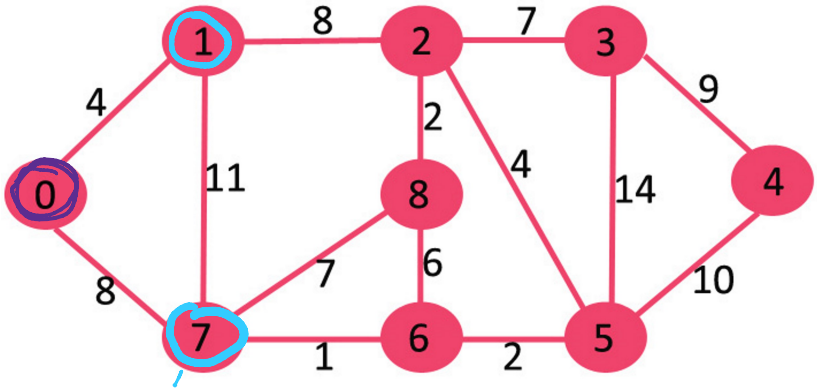
→ Graph is bipartite

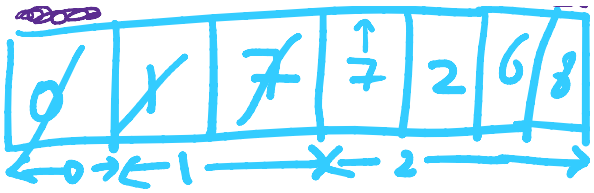
Means if cycle length is even, Graph is bipartite else no -





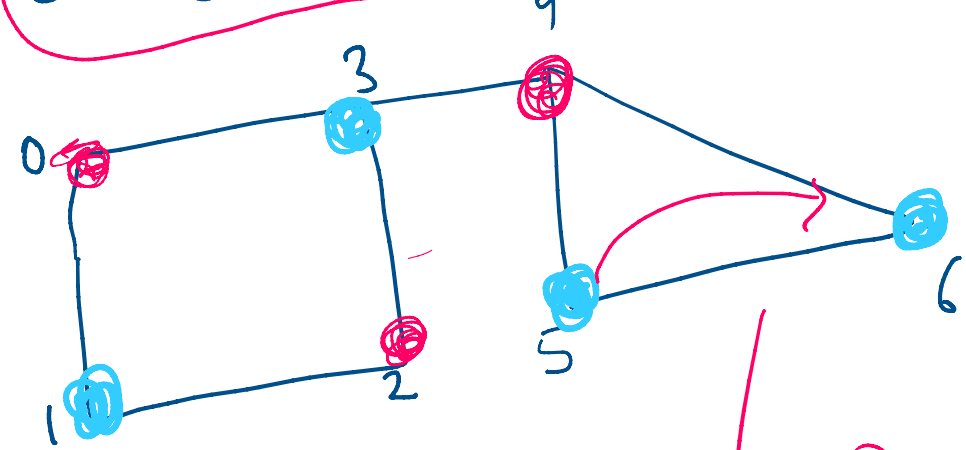






Different Colour.

Check for Bipartite



Not Bipartite.

Because Node with

Because Node with Same Colour are adjacent to each other:

```cpp
bool isBipartite(vector<vector<int>>& graph) {
    int n = graph.size();
    vector<int> visited(n,0);
    queue<int> PendingNodes;
    int Colour = 1;
        for(int i = 0 ; i < n ; i++){
        if(!visited[i])
            PendingNodes.push(i);
        while(!PendingNodes.empty()){
            int size = PendingNodes.size();
            while(size-->0){
                int Front = PendingNodes.front();
                PendingNodes.pop();
                if(visited[Front]){
                    if(visited[Front] != Colour) return false;
                    continue;
                }
                visited[Front] = Colour;
                for(int i = 0 ; i < graph[Front].size() ; i++){
                    if(!visited[graph[Front][i]])
                        PendingNodes.push(graph[Front][i]);
                }
            }
            Colour = 3 - Colour;
        }
    }
    return true;
}
```