Cycle Detection

We can use both BFS & DFS to detect cycle. to detect the cycle is to walk in the graph and keep track of all nodes that have been visited & a vector to store the nodes which are in the current path. Once a node is visited for the second time & is present in the vector of current path, we can conclude cycle exist and that node is the first node in the cycle. This method works in O(V+E) time and also uses O(V) memory.

DFS

1. Create a DFS function to traverse the Graph.
2. There are two boolean vectors to store the visited nodes & nodes in the path.

```
//To keep track of visited node
vector<bool>visited(n,0);
//To keep track of node in the path
vector<bool>inStack(n,0);
```

3. Mark the current node as visited and also mark the index in inStack.
4. Visit the vertices which are not visited and are adjacent to the current node.
5. Explore the neighbor's of current node recursively till all node visited

   If
   adjacent vertices are already visited and present in inStack then return true.
   else
   return false

The complete implementation is :

```
#include<bits/stdc++.h>
using namespace std;

bool dfsHelper(vector<vector<int>>adj,vector<bool>& visited,
                                vector<bool>& inStack,int i){
    //Mark the node in the path as visited & push it inStack

    visited[i]=1;
    inStack[i]=1;
    //Explore the neighbours
    for(auto neighbours:adj[i]){
        //Two things can happens
        /*1.If current node is not visited but its further branch lead to a
cycle.
          2.If the current node is already visited and its a node in the path*/
        if((!visited[neighbours] && dfsHelper(adj,visited,inStack,neighbours))
            ||inStack[neighbours]){
            return true;
        }
    }
    //Removes the element from stack
    inStack[i]=0;
    return false;
```

```
}

bool dfs(vector<vector<int>>adj,int n){
    //To keep track of visited node
    vector<bool>visited(n,0);
    //To keep track of node in the path
    vector<bool>inStack(n,0);

    //In directed graph it is not neccessary that all nodes are connected
    for(int i=1;i<n;i++){
        if(!visited[i]){
            bool isCyclePresent=dfsHelper(adj,visited,inStack,i);
            if(isCyclePresent){
                return true;
            }
        }
    }
    return false;
}


int main(){
    int n;
    cin>>n;
    vector<vector<int>>adj(n+1);
    int edge;
    cin>>edge;
    for(int i=0;i<edge;i++){
        int u,v;
        cin>>u>>v;
        adj[u].push_back(v);
    }
    if(dfs(adj,n+1)){
        cout<<"yes cycle found";
    }else{
        cout<<"No ";
    }
}
```

BFS

1. Calculate the indegree of each node

```
 //Calculation Indegree of each node
for(int i=1;i<n;i++){
   for(auto v:adj[i]){
       inDegree[v]++;
   }
}
```

2. If the indegree of each node is positive, then there exist a cycle.
3. If the indegree of any node is zero, then we cant say about cycle.
    - Create an queue and enqueue all vertices with in degree 0.
4. Initialize a variable count and Iterate till queue isn't empty.

```
        int count=1;
        while(queue is not empty){
            Extract front of queue. front = q.front();
             Pop the front of queue & increment the count by 1.
               Relax all the adjacent nodes connected to queue.
                for(auto u : adj[front]){
                   if(- - inDegree[u] == 0)
                           q.push(u);
                     }
                }
        }
```

5. If count of nodes with indegree 0 is **not** equal to the number of nodes in the graph has cycle, otherwise not.

The complete implementation is :

```
#include<bits/stdc++.h>
using namespace std;

void bfs(vector<vector<int>>adj,int n){
    vector<int>inDegree(n,0);

    //Calculation Indegree of each node
    for(int i=1;i<n;i++){
        for(auto v:adj[i]){
            inDegree[v]++;
        }
    }
    queue<int>q;
    //Create an queue and enqueue all vertices with inDegree 0
    for(int i=1;i<n;i++){
        if(inDegree[i]==0){
            q.push(i);
        }
    }
    int count=1;
    while(!q.empty()){
```

```
        int front=q.front();
        q.pop();
        count++;
        for(auto u:adj[front]){
            if(--inDegree[u]==0){
                q.push(u);
            }
        }
    }
    if(count==n){
        cout<<"Cycle doesn't exist"<<endl;
    }else{
        cout<<"Cycle exist"<<endl;
    }

}

int main(){
    int n;
    cin>>n;
    vector<vector<int>>adj(n+1);
    int edge;
    cin>>edge;
    for(int i=0;i<edge;i++){
        int u,v;
        cin>>u>>v;
        adj[u].push_back(v);
    }
    bfs(adj,n+1);
}
```

Marking the Nodes or Coloring

It is also using internally DFS. But instead of keep track of instack and visited elements. It simply mark them.

Colors	Description
0	Not Visited
1	Visited
2	Visited & Instack

The implementation of above logic is :

```
#include<bits/stdc++.h>
using namespace std;

#define ll long long int


bool detectCycle(vector<ll>adj[],vector<ll>& color,ll u){
    color[u]=2;
    for(auto v :adj[u]){
        if(color[v]==2){
            return true;
        }
        if(color[v]==0 && detectCycle(adj,color,v)){
            return true;
        }
    }
    color[u]=1;
    return false;
}


int main(){
    ll n,m;
    cin>>n>>m;
    vector<ll>adj[n+1];
    vector<ll>color(n+1,0);
    for(ll i=0;i<m;i++){
        ll u,v;
        cin>>u>>v;
        adj[u].push_back(v);
    }
    if(detectCycle(adj,color,1)){
        cout<<"Yes"<<endl;
    }else{
        cout<<"No"<<endl;
    }
}
```