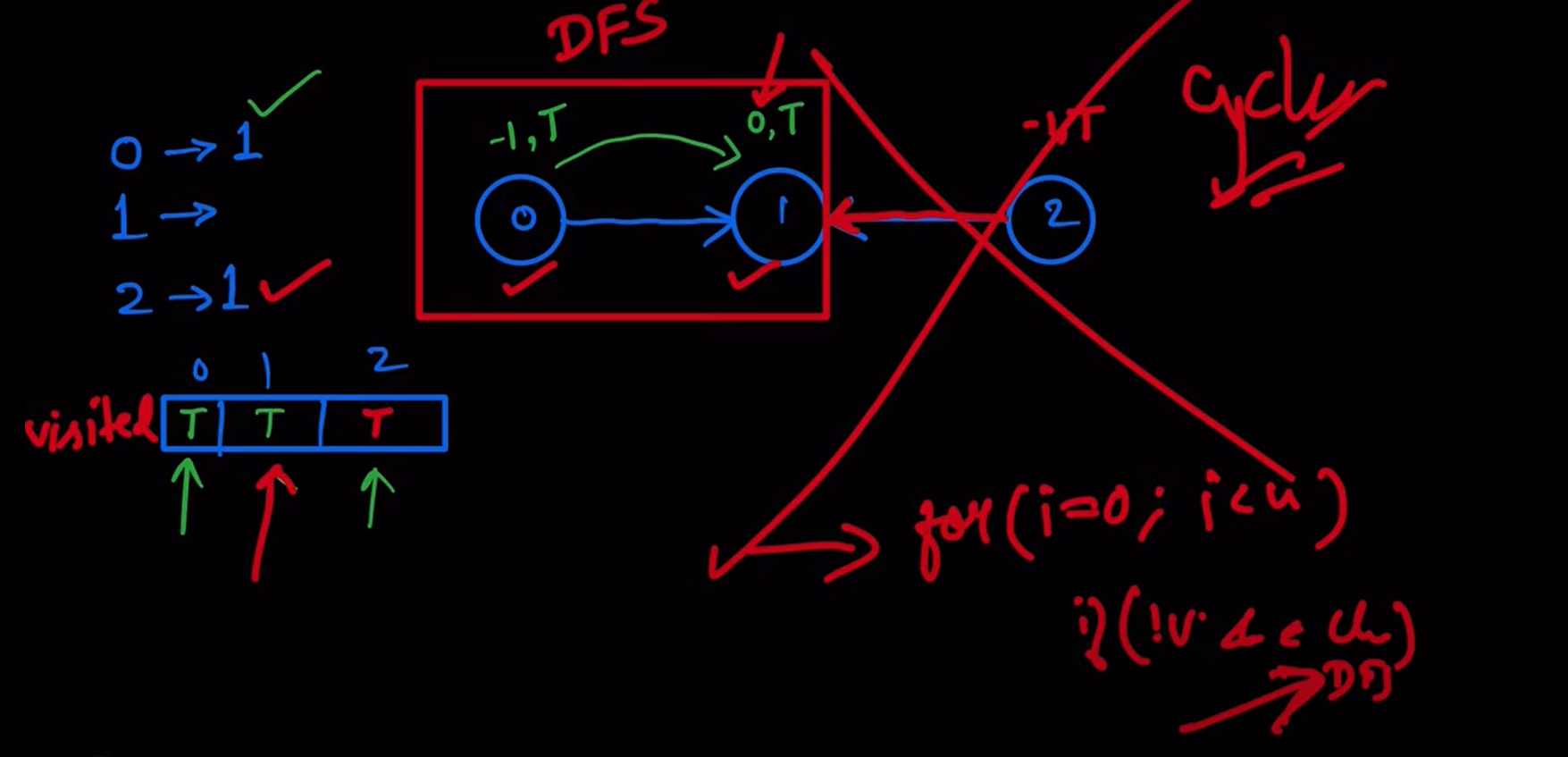
CYCLE DETECTION

We use inrecursion here for detection, inrecursion dictionary

# CYCLE IN DIRECTED GRAPH:

Course Schdule 1 is cycle detection

And course Schedule 2 is topological sort ordering question

## DFS

#User function Template for python3

from typing import List

class Solution:

**#Function to detect cycle in a directed graph.**

**def isCyclic(self, V : int , adj : List[List[int]]) -> bool :**

**def dfs(adj,node,visited,inRecur):**

**visited[node] = True**

**inRecur[node] = True**

**for child in adj[node]:**

**if child in visited and child in inRecur and inRecur[child]==True:**

**return 1**

**elif child not in visited and dfs(adj,child,visited,inRecur)==1:**

**return 1**

**inRecur[node] = False**

**return 0**

**visited = {}**

**inRecur = {}**

**for i in range(V):**

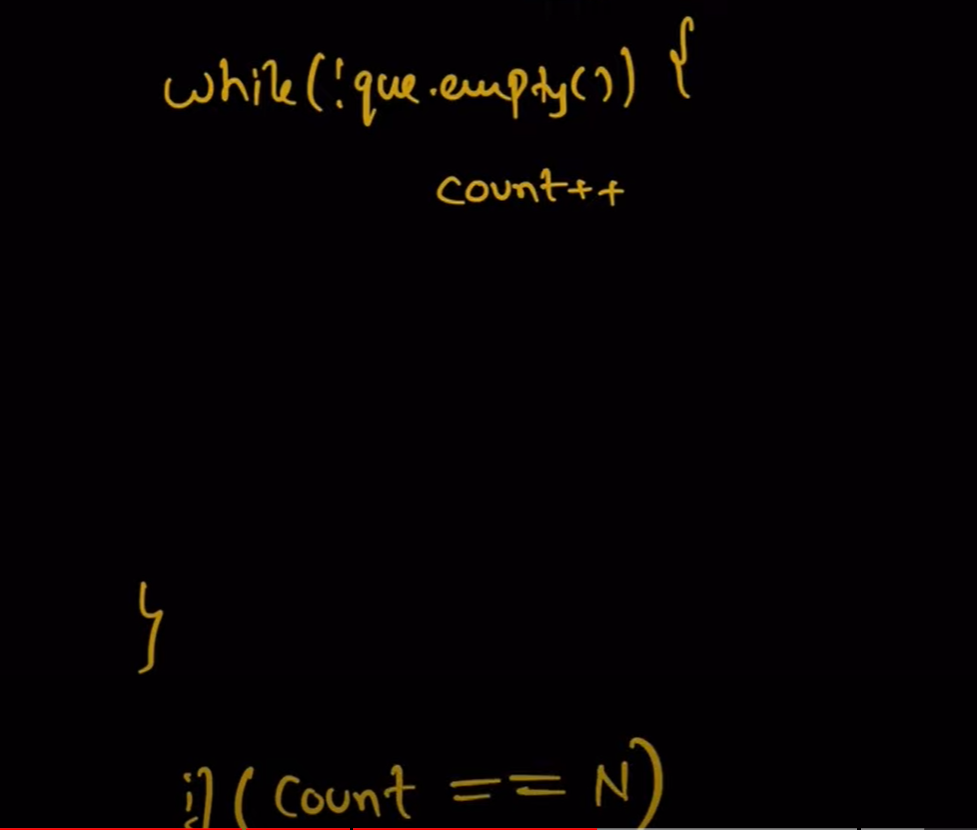
**if i not in visited and dfs(adj,i,visited,inRecur):**

**return 1**

**return 0**

## BFS

Kahns ALGo ko use krna h agr saray nodes que m add hugye or indegree jinki zero h uni ko add krna ha or at the end agr sary nodes add hujaty ha tu no cycle otherwise cycle huga, as no of nodes not equal to number of total nodes.



# CYCLE IN UNDIRECTED GRAPH:

**def bfs(adj,node,visited):**

**visited[node] = True**

**que = []**

**que.append( (node,-1))**

**while que:**

**node = que.pop(0)**

**visiting,parent = node**

**for item in adj[visiting]:**

**if item not in visited:**

**visited[item] = True**

**que.append( (item,visiting) )**

**elif item != parent:**

**return 1**

**return 0**

**def dfs(adj,node,parent,visited):**

**visited[node] = True**

**for child in adj[node]:**

**if child == parent:**

**continue**

**if child in visited and visited[child]==True:**

**return 1**

**if dfs(adj,child,node,visited)==1:**

**return 1**

**return 0**