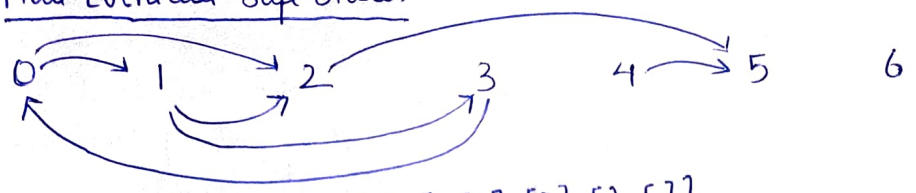
(4) Find Eventual Safe States.



i/p : graph = [[1,2],[2,3],[5],[0],[5],[],[]].

o/p : output = [2,4,5,6].

Terminal Nodes → nodes with 0 indegree

Safe Nodes → nodes having a possible path to the terminal nodes.

→ If node makes a cycle/loop in graph, that is definitely not a safe node.

→ Terminal node is also be considered as a safe node.

Approach: If we detect a loop from a node, we can say that node is not a safe node. And nodes with degree are ~~terminal~~ safe nodes.

Code:

```
void dfs(node, vis, dfs_vis, cycle, adj) {
    vis[node] = 1; dfs_vis[node] = 1;
    for (auto &it : cycle[node])
        if (!vis[it]) {
            if (dfs(it, vis, ...))
            return cycle[node] = true;
        }
        else if (vis[it] && dfs_vis[it])
            return cycle[node] = true;
    dfs_vis[node] = 0;
    return false;
}
vector<int> eventualSafeNodes(adj) {
    int v = adj.size();
    vector<int> vis(v,0), dfs_vis(v,0), ans;
    vector<bool> cycle(v, false);
    for (i → v)
        if (!vis[i])
            dfs(i, vis, dfs_vis, ...);
    for (i → v)
        if (!cycle)
            ans.push_back(i);
    return ans;
}
```

5
(5) Course Schedule

There are n courses labeled from 0 to n-1, and condition is that you have to take all the n courses but There is tricky part of this ques is that the prerequisite is also given to take a particular course.

Example: (i) n = 2 (course)

prerequisite = [[1,0]]

To take the course '1', '0' course is requisite for this

Both the courses can be complete, so return.

Approach: n = 2

preq = [[1,0], [0,1]]

1 ⇄ 0

Both are the prerequisites for each other, so return false.

Code: → Just implement the cycle/loop detection using DFS.

(6) Surrounded Regions

X	X	X	X		X	X	X	X
X	O	O	X	→	X	X	X	X
X	X	O	X		X	X	X	X
X	O	X	X		X	O	X	X

flipping 'O' to 'X' only and only if all 4 directions O are surrounded by 'X'.

All the boundary 'O' will be considered as there can not be 'X' if its over the boundary. like in the above diagram.

Approach: Here we can apply dfs like the no. of islands & flood fill ques but we have to take care of boundary 'O's and all the 'O's that are connected to that boundary wala 'O'.
So, To achieve this, first we will traverse all the boundary and start dfs if its 'O' and marks all the nodes as 'B' (which are connected to boundary wala 'O'. After that simply traverse the matrix and convert 'B' to 'O' and 'O' to 'X'. And process is completed.

Code:

```
bool isValid() {
}

void dfs() {
  board[i][j] = 'O';
  ____
  ____
  ____
  ____
}
```

→ Same as previous ques

```
void solve(board) {
  for(i → n) {
    int j = 0;
    if(board[i][j] == 'O') dfs(board, i, j, n, m);
    j = m - 1;
    if(board[i][j] == 'O') dfs(board, i, j, n, m);
  }
  for(j → m) {
    int i = 0;
    if(board[i][j] == 'O') dfs(board, i, j, n, m);
    i = n - 1;
    if(board[i][j] == 'O') dfs(board, i, j, n, m);
  }
  for(i → n)
    for(i → m)
      if(board[i][j] == 'B')
        board[i][j] = 'O';
      else if(board[i][j] == 'O')
        board[i][j] = 'X';
}
```

→ Top → Bottom (Left)

→ Top → Bottom (Right)

→ Left → Right (Top)

→ Left → Right (Bottom)

→ Last conversion
B → O
and O → X.