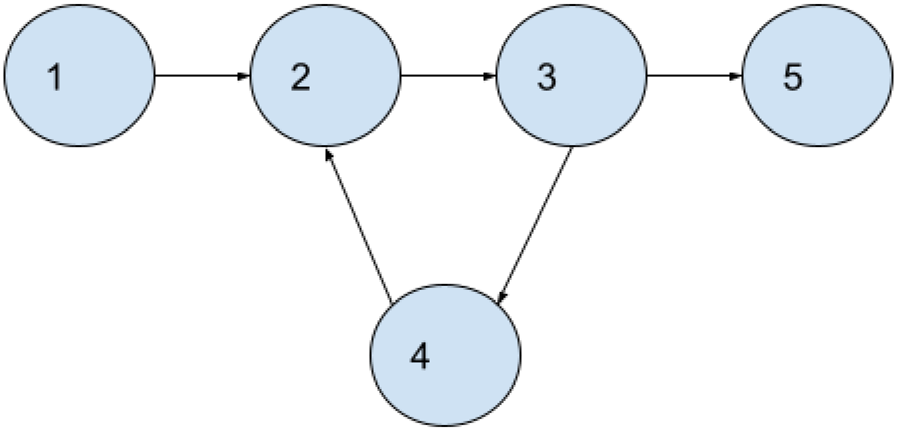
MEDIUM

Detect a cycle in a directed graph using Kahn’s Algorithm

Intuition :



Adjacency List :

1 - [2]
2 - [3]
3 - [4, 5]
4 - [2]
5 - [ ]

Kahn’s algorithm can only be applicable for Directed Acyclic graphs. It wont be applicable on any other type of graph.

1 - 0
2 - 2 - 1 = 1
3 - 1
4 - 1
5 - 1

Queue = 1 |
pop(q) - the queue becomes empty and now we are left with no element that can be traversed now. Toposort is a linear order of n elements and this can be used to detect if there is a directed graph having a cycle or not as.

- If we are able to generate toposort then we are having no cycle
- If we are not able to generate a toposort we are having a cycle

Approach :

- Declare vector for counting indegree initialized with indegree of all elements
- Create an empty queue
- Insert all elements having 0 indegree to the queue
- Create a counter variable to count the topological sort elements
- Traverse until the queue becomes empty :
    - Extract the first element of the queue
    - Pop the first element of the queue
    - Increment the counter
    - Traverse for adjacent elements :
        - Decrease the indegree of elements by 1
        - Check if the indegree of any adjacent element becomes 0 :
            - If indegree becomes 0 push it to the queue
- If the counter is equal to number of nodes then there is no cycle : return false
- Return true // there is a cycle

Function Code :

```
bool isCyclic(int n, vector<int> adj[]) {
       // Declare a indegree vector
       vector<int> indegree(n,0);

       // initializing indegree
       for(int i=0;i<n;i++)
       {
           for(auto it : adj[i])
           {
               indegree[it]++;
           }
       }

       // creating an empty queue

       queue<int> q;

       // creating a counter variable

       int counter = 0;

       // initializing the queue

       for(int i =0;i<n;i++)
```

```
        {
            if(indegree[i]==0)
            {
                q.push(i);
            }
        }

        // traversing while the queue becomes empty

        while(!q.empty())
        {
            int node = q.front();
            q.pop();
            counter++;

            // traversing through the adjacent elements
            for(int i:adj[node])
            {
                indegree[i]-=1;
                if(indegree[i]==0)
                {
                    q.push(i);
                }
            }
        }
        if(counter==n)return false;
        return true;
    }
```

Time Complexity :

O(V+E)