Leetcode May Challenge DAY: 29

1. Python

This is a classical problem showing how topological sort works. Topological sort can be implemented by a queue, and it's kind of BFS. Everytime we enqueue the vertices with indegree 0, keep a counter to count how many vertices have been sorted, if there's no cycle, the counter should be equal to be total number of courses.

```
class Solution:
    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> bool:
        # construct a graph, using adjacency list
        g = collections.defaultdict(list)
        indeg = collections.defaultdict(int)
        tot = numCourses
        for i in range(tot):
            indeg[i] = 0
        for i in range(len(prerequisites)):
            cur, pre = prerequisites[i]
            g[pre].append(cur)
            indeg[cur] += 1
        q = []          # then do topological sort
        cnt = 0
        for k in indeg:
            if indeg[k] == 0:
                q.append(k)
        while q:
            cur = q.pop(0)
            print("course: ", cur)
            cnt += 1
            for vertex in g[cur]:
                indeg[vertex] -= 1
                if indeg[vertex] == 0:
```

```
                q.append(vertex)

        # keep a counter during sorting, if counter < #vertices, there's a cycle
        return True if (cnt == tot) else False
```

2. C++

```
bool canFinish( int numCourses, vector<vector<int>>& prerequisites )
{
	/* Create adjacency list and indegree array for each course*/
	vector<int> indegree( numCourses, 0 );
	unordered_map<int, set<int> > adjList;
	for( auto prereq : prerequisites ) {
		adjList[ prereq[1] ].insert( prereq[0] );
		indegree[ prereq[0] ]++;
	}
	/* Add all the sources with indegree zero to queue for
processing */
	queue<int> sources;
	for( int i=0; i < indegree.size(); i++ )
		if( indegree[i] == 0 )
			sources.push( i );
	/* For each source visit all neighbours and reduce their
indegrees */
	int noOfCoursesTaken = 0;
	while( !sources.empty() ) {
		++noOfCoursesTaken;
		int curr = sources.front(); sources.pop();
		for( auto nbr : adjList[curr] ) {
			--indegree[nbr];
			if( indegree[nbr] == 0 )
				sources.push(nbr);
		}
	}
	return noOfCoursesTaken == numCourses;
}
```

3. JAVA

```
class Solution {
    public boolean canFinish(int numCourses, int[][] prerequisites) {

        //An array conating the list of nodes each node is connected to.
        ArrayList<Integer>[] map=new ArrayList[numCourses];
        for(int i=0;i<numCourses;i++){
            map[i]=new ArrayList();
        }

        //An array containing the indegree of each nodes
        int[] indegree=new int[numCourses];
        Arrays.fill(indegree,0);
        for(int[] p:prerequisites){
            indegree[p[0]]++;
            map[p[1]].add(p[0]);
        }

        //variable to store total number of edges
        int total_edge=prerequisites.length;

        //queue to push nodes with 0 in-degrees.
        Queue<Integer> q=new LinkedList();
        for(int i=0;i<numCourses;i++){
            if(indegree[i]==0){
                q.offer(i);
            }
        }
```

```
        int removed=0;

        //keep looping and remove edges with help of nodes having 0 in-degrees.
        //keep a count of number of edges removed. This gives a topological sorted order.
        while(!q.isEmpty()){
            int s=q.size();
            for(int i=0;i<s;i++){
                int x=q.poll();
                for(int next:map[x]){
                    indegree[next]--;
                    removed++;
                    if(indegree[next]==0){
                        q.offer(next);
                    }
                }
            }
        }

        //check if we are able to remove all the edges or not.
        if(removed==total_edge){
            return true;
        }
        else{
            return false;
        }
    }
}
```