Day 19 / 100 :

Topic - DFS, Linked list

1 Problem statement: Course Schedule (Medium)

There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1. You are given an array prerequisites where prerequisites[i] = [ai, bi] indicates that you must take course bi first if you want to take course ai.
For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1.
Return true if you can finish all courses. Otherwise, return false.

Example 1:
Input: numCourses = 2, prerequisites = [[1,0]]
Output: true
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0. So it is possible.

Example 2:
Input: numCourses = 2, prerequisites = [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible.

Solutions :
Approach 1 -DFS

Approach: DFS Approach
Run a for loop from the first node to the last node and check if the node is visited. If it is not visited then call the function recursively which goes into the depth as known as DFS search and if you find a cycle you can say that there is a cycle in the graph.

1. Basically calling the dfs function with number of nodes and passing the graph
2. Traversing from 0 to number of numsCourses and checking for every node if it is unvisited
3. If the node is unvisited then call a function dfs, that checks if there is a cycle and returns true if there is a cycle.
4. Now the function dfs has the node. It will also have the visited array, curent visited array and the graph that has adjacency list
5. Mark it as visited in both visit and currVisited, then traverse for its adjacency list using a for loop.

6. Calling DFS traversal if that node is unvisited call recursive function that checks if its a cycle and returns true
7. else If the current visited node is true we can say there is cycle again and will return true
8. Now if you have traveled for all adjacent nodes and all the DSF have been called and it never returned true that means we have done the DSF call entirely and mark current visited node as false and now we can return false, that mean there is no DSF cycle

```
class Solution {
public:
    bool dfs(int src,vector<bool>& vis,vector<bool>&
currVis,vector<vector<int>>& adj){
        vis[src] = true;
        currVis[src]=true;

        for (auto it: adj[src]) {
            if(!vis[it]){
                if(dfs(it, vis,currVis, adj))
                    return true;
            }
            else{
                if(currVis[it])
                    return true;
            }
        }
        currVis[src]=false;
        return false;
    }
    bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {
        vector<vector<int>> adj(numCourses);
        vector<bool> vis(numCourses+1,false);
        vector<bool> currVis(numCourses+1,false);

        for(int i=0;i<prerequisites.size();i++){
            adj[prerequisites[i][0]].push_back(prerequisites[i][1]);
        }
        for(int i=0;i<numCourses;i++){
            if(!vis[i]){
                if(dfs(i, vis,currVis, adj))
                    return false;
            }
        }
        return true;
    }};
```

Solutions :
Approach 2 -BFS

First compute the indegree array (how many edges toward a node) and take topo vector. Use a queue to push all nodes with indegree 0. Start from a front node on the queue, push front node in topo vector and subtract by 1 of all the nodes on its adjacency list. If after subtraction, the node has indegree 0, push it into queue. Do this recursively for all nodes on queue until no nodes are on queue (you pop a node after subtracting by 1 of all nodes on its adjacency list). At the end if topo vector size not equal to number of edge, then graph has a cycle return false else return false.
~Time Complexity O(V+E)
~Space Complexity O(V)

```
class Solution {
public:
    bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {
        vector<int> topo;
        vector<vector<int>> adj(numCourses);
        vector<int> indregree(numCourses,0);
        for(auto x: prerequisites){
            int u=x[1];
            int v=x[0];
            adj[u].push_back(v);
            indregree[v]++;
        }

        queue<int> q;
        for(int i=0;i<numCourses;i++){
            if(indregree[i]==0){
                q.push(i);
            }
        }
        while(!q.empty()){
            int temp=q.front();
            q.pop();
            topo.push_back(temp);
            for(int x: adj[temp]){
                indregree[x]--;
                if(indregree[x]==0){
                    q.push(x);
                }
            }
        }
        if(topo.size()==numCourses){
            return true;
        }
        return false;
    }
};
```

2 Problem statement: **multiply 2 number represented by linked List**
(Medium)

Given two numbers represented by linked lists, write a function that returns the multiplication of these two linked lists.

Examples:

Input : 9->4->6
        8->4
Output : 79464

Input : 3->2->1
        1->2
Output : 3852

Solutions :
Approach 1 - Traversing

Traverse both lists and generate the required numbers to be multiplied and then return the multiplied values of the two numbers.
Algorithm to generate the number from linked list representation:

1) Initialize a variable to zero
2) Start traversing the linked list
3) Add the value of first node to this variable
4) From the second node, multiply the variable by 10
   and also take modulus of this value by 10^9+7
   and then add the value of the node to this
   variable.
5) Repeat step 4 until we reach the last node of the list.

```
long long multiplyTwoLists (Node* first, Node* second)
{
    long long N= 1000000007;
    long long num1 = 0, num2 = 0;
    while (first || second){

        if(first){
            num1 = ((num1)*10)%N + first->data;
            first = first->next;
        }

        if(second)
        {
            num2 = ((num2)*10)%N + second->data;
            second = second->next;
        }

    }
    return ((num1%N)*(num2%N))%N;
}
```