207. Course Schedule

Hint

Medium 15.3K 615

Companies

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] = [a_i, b_i]` indicates that you **must** take course `b_i` first if you want to take course `a_i`.

- 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:

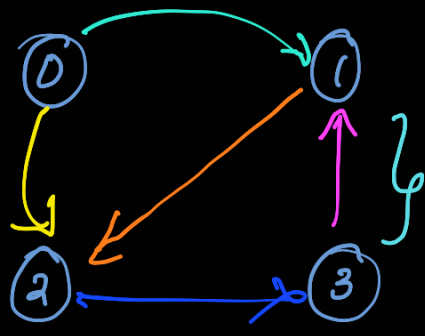
```
Input: numCourses = 2, prerequisites = [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible.
```

Constraints:

- `1 <= numCourses <= 2000`
- `0 <= prerequisites.length <= 5000`
- `prerequisites[i].length == 2`
- `0 <= a_i, b_i < numCourses`
- All the pairs prerequisites[i] are **unique**.

Accepted 1.4M | Submissions 3M | Acceptance Rate 46.3%

[(1,0) , (2,0) , (2,1) , (3,2) , (1,3)]



i.e. To do course (1), you need to complete course (3)

note: we can also interpret the edges the otherway also.

Its just about checking cycle in a directed graph.

if cycle exists, then return false

if not then return true

```
class Solution {
public:
    bool cycleDfs(vector<vector<int>> &v,int i,vector<int> &visited,vector<int> &currVisited){

        visited[i] = 1;
        currVisited[i] = 1;
        for(auto j:v[i]){
            if(!visited[j]){
                if(cycleDfs(v,j,visited,currVisited))
                    return true;
            }

            else if(currVisited[j] == 1)
                return true;
        }
        currVisited[i] = 0;
        return false;
    }

    bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {
        vector<vector<int>> v(numCourses,vector<int>(0));
        vector<int> visited(numCourses,0);
```

general dfs algo to check cycle in directed graph.

→ adjacency list used to check

```
        vector<int> currVisited(numCourses,0);

        for(auto &temp:prerequisites){
            v[temp[0]].push_back(temp[1]);
        }   v[temp[1]].push_back(temp[0])

        for(int i=0;i<numCourses;i++){
            if(!visited[i] && cycleDfs(v,i,visited,currVisited))
                return false;
        }

        return true;

    }
};
```

cycle in directed graph

} creating graph

also works

$T(n): O(m) + O(n) + O(m)$

$S(n): O(2n) + O(n) + O(n)$

where m: prerequisites length

n: numCourses