Find the town judge - QOTD 23 Jan 23

Leetcode Link : CLick

★ $\checkmark ® Approach-1$ (using unordered map)

Time : $O(n*m)$

> where n is number of rows, m is no of col$

Space : $O(n^2)$

> in worst case there can be n pairs in map, and each pair can have at max n-1 elements$

approach/steps :

```
/* ✔⭐Approach - 1 (using unordered map)

    explanation :-

            // -> Main function

                step 0: exception case - if there is only 1 person then he is the
judge coz all knows judge, judge knows noone

                step 1 : create map(int,vector<int>) where int represents each element
and vector represents all elements that the element knows

                step 2 :insert each element and the elements it knows into map
                step 2.2 : exception case : when n is more then total number of people
that know each other(total entries in map) then return -1, coz that means their is
more then 1 person that doesnt know any one

                step 3 : create a judge (int) and traverse whole map and whatever
element has a empty vector as value, then it is the judge, if there is no judge, then
return -1

                step 4 : return -1 if (no judge is found or if there are more then 1
person that doesnt know any one)
                step 5 : now check if the judge we have found is known by every other
element, if all knows hi, then return the judge at the end, if at least 1 element
doesnt knows him then return -1
                    step 5.1 :if atleast 1 element doesnt knows the judge then return
-1, else keep going
                    step 5.2 : dont check the judge, he knows himself we already know

                step 6 : when all knows the judge, we found our judge


 n = number of elements in total matrix

   ☑T : O(n*m)  -> where n is number of rows, m is no of col
   S : O(n^2)  -> in worst case there can be n pairs in map, and each pair can have at
max n-1 elements


*/
```

Code :

```
public:// -> Main function

    int findJudge(int n, vector<vector<int>>& arr) {

        // exception case - if there is only 1 person then he is the judge coz all
knows judge, judge knows noone

        if(n==1) return 1;

        int rowSize = arr.size();

        int colSize = 2;

        // step 1 : create map(int,vector<int>) where int represents each element and
vector represents all elements that the element knows

        unordered_map<int,vector<int>> map;

        // step 2 :insert each element and the elements it knows into map

        for(int i = 0; i < rowSize; i++){

            for(int j = 0; j < colSize; j++){

                map[arr[i][j]];

            }

            (map[arr[i][0]]).push_back(arr[i][1]);

        }

        // exception case : when n is more then total number of people that know each
other(total entries in map) then return -1, coz that means their is more then 1 person
that doesnt know any one

        if(map.size() < n) return -1;

        // step 3 : create a judge (int) and traverse whole map and whatever element
has a empty vector as value, then it is the judge, if there is no judge, then return
-1

        int judge = -999;

        int judgeCount = 0;

        for(auto i:map){

            if((i.second).size() == 0 ){
                judge = i.first;
                judgeCount++;
            }
        }
        // return -1 if (no judge is found or if there are more then 1 person that doesnt know any one)
        if(judgeCount > 1 || judge == -999) return -1;
        // now check if the judge we have found is known by every other element, if all knows hi, then return the judge at the end, if at least 1 element doesnt knows him then return -1
        for(auto i:map){
            int element = i.first;
            vector<int> knowsThem = i.second;
            int JudgeisPresent = false;
            for(auto j:knowsThem){
                if(j == judge) JudgeisPresent = true;
            }
            // if atleast 1 element doesnt knows the judge then return -1, else keep going
            if(i.first == judge) continue; // dont check the judge, he knows himself we already know
            if(JudgeisPresent == false) return -1;
        }
        // when all knows the judge, we found our judge
        return judge;
    }
};
```

Pictures for reference (Dry run) :-

App-1 (updation)

```
                    ele   knows these ele
                    (          )
// create a map<int, vector<int>>

unordered_map<int, vector<int>>.

// traverse whole input 2D array & insert
   each element into vector of their known
                                       ones.

for ______
[  for ______
[  [    map[arr[i][j]];
   map[arr[i][0]].push_back(arr[i][1]);


// now traverse whole map & find
   if there is a entry whose value is
   empty vector. if yes then that can be
   the judge.
int
judge;  for (auto i : map).
            vector = i.second;
            int ele = i.first;
            if( vector.size() == 0). judge = ele;

                              comple
// again traverse the map & check if
          judge is present in all vector.

for(auto i : map).
   vec = i.second,   bool ispresent = false;
   for (auto i; vector) if (i == judge) ispresent = true
   if(ispresent == false) return -1;
return judge.
```

dry run - 1

[[1, 2]
  [2, 3]]

map.

ind	1	2	3
vector<int>	2,	3,	.

judgeCanbe = 3.

judge is not present in vector
∴ return -1;

[1, 2] pass
[2, 3]
[3, 4]

n = 4 (1, 2, 3, 4)
n = 5 (1, 2, 3, 4, 5) ?

dry run - 2

[1, 2] 3, 4. ✗
[3, 2] 2

n = 1
arr = [ ]

if (row col <= n) return -1.

edge case

* when total no of people are more the row & col

$n=4$ (1,2,3,4)

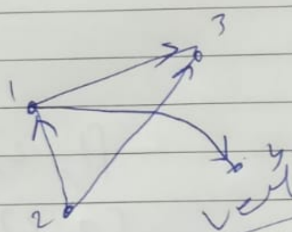
~~1~~ 3	2
3	4
4	3

6

$n=2$

[[1,2]]

1	2
< >	< >



5	2
1	2
3	2
4	3
1	4.
1	

i/p	vector.
1	3, 4, 2.
2,	1, 3, 4
3,	1, 2, 4
4 →	< >

← judge

in worst case there will be ~ n pairs in map ~ L in each pair at max there can be n ele.

$\boxed{n \times n}$