1136. Parallel Courses

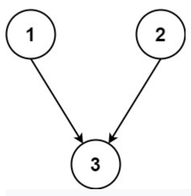
Medium 559 15 Add to List Share

You are given an integer `n`, which indicates that there are `n` courses labeled from `1` to `n`. You are also given an array `relations` where `relations[i] = [prevCourse_i, nextCourse_i]`, representing a prerequisite relationship between course `prevCourse_i` and course `nextCourse_i`: course `prevCourse_i` has to be taken before course `nextCourse_i`.

In one semester, you can take **any number** of courses as long as you have taken all the prerequisites in the **previous** semester for the courses you are taking.

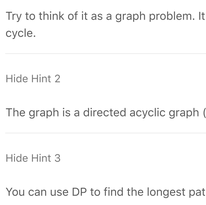
Return *the **minimum** number of semesters needed to take all courses*. If there is no way to take all the courses, return `-1`.

Example 1:




```
Input: n = 3, relations = [[1,3],[2,3]]
Output: 2
Explanation: The figure above represents the given graph.
In the first semester, you can take courses 1 and 2.
In the second semester, you can take course 3.
```

Example 2:




```
Input: n = 3, relations = [[1,2],[2,3],[3,1]]
Output: -1
Explanation: No course can be studied because they are prerequisites of each
other.
```

Constraints:

- `1 <= n <= 5000`
- `1 <= relations.length <= 5000`
- `relations[i].length == 2`
- `1 <= prevCourse_i, nextCourse_i <= n`
- `prevCourse_i != nextCourse_i`
- All the pairs `[prevCourse_i, nextCourse_i]` are **unique**.

Accepted 28,403 | Submissions 47,212

Seen this question in a real interview before? Yes No

Companies

0 ~ 6 months 6 months ~ 1 year 1 year ~ 2 years

Google | 3 Facebook | 2

Related Topics

Graph Topological Sort

Similar Questions

Course Schedule II	Medium
Parallel Courses II	Hard
Parallel Courses III	Hard

Hide Hint 1

Try to think of it as a graph problem. It will be impossible to study all the courses if the graph had a cycle.

Hide Hint 2

The graph is a directed acyclic graph (DAG). The answer is the longes path in this DAG.

Hide Hint 3

You can use DP to find the longest path in the DAG.

Java | Autocomplete

```java
class Solution {
    public int minimumSemesters(int N, int[][] relations) {
        List<List<Integer>> graph = new ArrayList<>(N + 1);
        for (int i = 0; i < N + 1; ++i) {
            graph.add(new ArrayList<Integer>());
        }
        for (int[] relation : relations) {
            graph.get(relation[0]).add(relation[1]);
        }
        int[] visited = new int[N + 1];

        int maxLength = 1;
        for (int node = 1; node < N + 1; node++) {
            int length = dfs(node, graph, visited);
            // we meet a cycle!
            if (length == -1) {
                return -1;
            }
            maxLength = Math.max(length, maxLength);
        }
        return maxLength;
    }

    private int dfs(int node, List<List<Integer>> graph, int[] visited) {
        // return the longest path (inclusive)
        if (visited[node] != 0) {
            return visited[node];
        } else {
            // mark as visiting
            visited[node] = -1;
        }
        int maxLength = 1;
        for (int endNode : graph.get(node)) {
            int length = dfs(endNode, graph, visited);
            // we meet a cycle!
            if (length == -1) {
                return -1;
            }
            maxLength = Math.max(length + 1, maxLength);
        }
        // mark as visited
        visited[node] = maxLength;
        return maxLength;
    }
}
```