Tasks Scheduling Order (medium)

Problem Statement

There are 'N' tasks, labeled from '0' to 'N-1'. Each task can have some prerequisite tasks which need to be completed before it can be scheduled. Given the number of tasks and a list of prerequisite pairs, write a method to find the ordering of tasks we should pick to finish all tasks.

Example 1:

```
Input: Tasks=3, Prerequisites=[0, 1], [1, 2]
Output: [0, 1, 2]
Explanation: To execute task '1', task '0' needs to finish first. Similarly, task '1' needs t
o finish
before '2' can be scheduled. A possible scheduling of tasks is: [0, 1, 2]
```

Example 2:

```
Input: Tasks=3, Prerequisites=[0, 1], [1, 2], [2, 0]
Output: []
Explanation: The tasks have cyclic dependency, therefore they cannot be scheduled.
```

Example 3:

```
Input: Tasks=6, Prerequisites=[2, 5], [0, 5], [0, 4], [1, 4], [3, 2], [1, 3]
Output: [0 1 4 3 2 5]
Explanation: A possible scheduling of tasks is: [0 1 4 3 2 5]
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
const find_order = function(tasks, prerequisites) {
  sortedOrder = [];
  // TODO: Write your code here
  return sortedOrder;
};


console.log(`Is scheduling possible: ${find_order(3, [[0, 1], [1, 2]])}`)
console.log(`Is scheduling possible: ${find_order(3, [[0, 1], [1, 2], [2, 0]])}`)
console.log(`Is scheduling possible: ${find_order(6, [[2, 5], [0, 5], [0, 4], [1, 4], [3, 2], [1, 3]])}`)

```

RUN SAVE RESET

Solution

This problem is similar to Tasks Scheduling, the only difference being that we need to find the best ordering of tasks so that it is possible to schedule them all.

Code

Here is what our algorithm will look like (only the highlighted lines have changed):

Java | Python3 | C++ | JS

```
const Deque = require('./collections/deque'); //http://www.collectionsjs.com

function find_order(tasks, prerequisites) {
  sortedOrder = [];
  if (tasks <= 0) {
    return sortedOrder;
  }

  // a. Initialize the graph
  const inDegree = Array(tasks).fill(0); // count of incoming edges
  const graph = Array(tasks).fill(0).map(() => Array()); // adjacency list graph

  // b. Build the graph
```

Mark Course as Completed


```
prerequisites.forEach((prerequisite) => {
  let parent = prerequisite[0],
    child = prerequisite[1];
  graph[parent].push(child); // put the child into it's parent's list
  inDegree[child]++; // increment child's inDegree
});


// c. Find all sources i.e., all vertices with 0 in-degrees
const sources = new Deque();
for (i = 0; i < inDegree.length; i++) {
  if (inDegree[i] === 0) {
    sources.push(i);
  }
}
```

RUN SAVE RESET

Close

Output 4.568s

```
Is scheduling possible: 0,1,2
Is scheduling possible:
Is scheduling possible: 0,1,5,4,3,2
```

Time complexity

In step 'd', each task can become a source only once and each edge (prerequisite) will be accessed and removed once. Therefore, the time complexity of the above algorithm will be $O(V + E)$, where 'V' is the total number of tasks and 'E' is the total number of prerequisites.

Space complexity

The space complexity will be $O(V + E)$, since we are storing all of the prerequisites for each task in an adjacency list.

Similar Problems

Course Schedule: There are 'N' courses, labeled from '0' to 'N-1'. Each course has some prerequisite courses which need to be completed before it can be taken. Given the number of courses and a list of prerequisite pairs, write a method to find the best ordering of the courses that a student can take in order to finish all courses.

Solution: This problem is exactly similar to our parent problem. In this problem, we have courses instead of tasks.