Tasks Scheduling Order (medium)

We'll cover the following

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++

```java
import java.util.*;

class TaskSchedulingOrder {
  public static List<Integer> findOrder(int tasks, int[][] prerequisites) {
    List<Integer> sortedOrder = new ArrayList<>();
    // TODO: Write your code here
    return sortedOrder;
  }

  public static void main(String[] args) {
    List<Integer> result = TaskSchedulingOrder.findOrder(3, new int[][] { new int[] { 0, 1 }, new int[] {
    System.out.println(result);

    result = TaskSchedulingOrder.findOrder(3,
        new int[][] { new int[] { 0, 1 }, new int[] { 1, 2 }, new int[] { 2, 0 } });
    System.out.println(result);

    result = TaskSchedulingOrder.findOrder(6, new int[][] { new int[] { 2, 5 }, new int[] { 0, 5 }, new i
        new int[] { 1, 4 }, new int[] { 3, 2 }, new int[] { 1, 3 } });
    System.out.println(result);
  }
}
```

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

```java
import java.util.*;

class TaskSchedulingOrder {
  public static List<Integer> findOrder(int tasks, int[][] prerequisites) {
    List<Integer> sortedOrder = new ArrayList<>();
    if (tasks <= 0)
      return sortedOrder;

    // a. Initialize the graph
    HashMap<Integer, Integer> inDegree = new HashMap<>(); // count of incoming edges for every vertex
    HashMap<Integer, List<Integer>> graph = new HashMap<>(); // adjacency list graph
    for (int i = 0; i < tasks; i++) {
      inDegree.put(i, 0);
      graph.put(i, new ArrayList<Integer>());
    }

    // b. Build the graph
    for (int i = 0; i < prerequisites.length; i++) {
      int parent = prerequisites[i][0], child = prerequisites[i][1];
      graph.get(parent).add(child); // put the child into it's parent's list
      inDegree.put(child, inDegree.get(child) + 1); // increment child's inDegree
    }

    // c. Find all sources i.e., all vertices with 0 in-degrees
    Queue<Integer> sources = new LinkedList<>();
    for (Map.Entry<Integer, Integer> entry : inDegree.entrySet()) {
      if (entry.getValue() == 0)
        sources.add(entry.getKey());
```

Run | Save | Reset

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.