Tasks Scheduling (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, find out if it is possible to schedule all the tasks.

Example 1:

```
Input: Tasks=3, Prerequisites=[0, 1], [1, 2]
Output: true
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 sceduling of tasks is: [0, 1, 2]
```

Example 2:

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

Example 3:

```
Input: Tasks=6, Prerequisites=[2, 5], [0, 5], [0, 4], [1, 4], [3, 2], [1, 3]
Output: true
Explanation: A possible sceduling 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 TaskScheduling {
  public static boolean isSchedulingPossible(int tasks, int[][] prerequisites) {
    // TODO: Write your code here
    return false;
  }

  public static void main(String[] args) {

    boolean result = TaskScheduling.isSchedulingPossible(3, new int[][] { new int[] { 0, 1 }, new int[] {
    System.out.println("Tasks execution possible: " + result);

    result = TaskScheduling.isSchedulingPossible(3,
        new int[][] { new int[] { 0, 1 }, new int[] { 1, 2 }, new int[] { 2, 0 } });
    System.out.println("Tasks execution possible: " + result);

    result = TaskScheduling.isSchedulingPossible(6, new int[][] { new int[] { 2, 5 }, new int[] { 0, 5 },
        new int[] { 0, 4 }, new int[] { 1, 4 }, new int[] { 3, 2 }, new int[] { 1, 3 } });
    System.out.println("Tasks execution possible: " + result);
  }
}
```

RUN SAVE RESET

Solution

This problem is asking us to find out if it is possible to find a topological ordering of the given tasks. The tasks are equivalent to the vertices and the prerequisites are the edges.

We can use a similar algorithm as described in Topological Sort to find the topological ordering of the tasks. If the ordering does not include all the tasks, we will conclude that some tasks have cyclic dependencies.

Code

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

Alien Dictionary (hard)
○ Problem Challenge 1
○ Solution Review: Problem Challenge 1
○ Problem Challenge 2
○ Solution Review: Problem Challenge 2

Miscellaneous

○ Kth Smallest Number (hard)

Conclusions

○ Where to Go from Here

Mark Course as Completed

Explore
Tracks
My Courses
Edpresso
Refer a Friend
Create

Java | Python3 | C++ | JS

```java
import java.util.*;

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

    // 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

Close

Output 1.558s

```
Tasks execution possible: true
Tasks execution possible: false
Tasks execution possible: true
```

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 can have some prerequisite courses which need to be completed before it can be taken. Given the number of courses and a list of prerequisite pairs, find if it is possible for a student to take all the courses.

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

← Back
Topological Sort (medium)

Next →
Tasks Scheduling Order (medium)

☑ Mark as Completed

Report an Issue   Ask a Question