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. Simila
rly, task '1' needs
to finish before '2' can be scheduled. One possible scheduling of task
s is: [0, 1, 2]
```

Example 2:

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

Example 3:

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

Try it yourself#

Try solving this question here:

C++

```
using namespace std;

#include <iostream>
#include <queue>
#include <unordered_map>
#include <vector>

class TaskScheduling {
 public:
  static bool isSchedulingPossible(int tasks, const vector<vector<int>>& pre
    // TODO: Write your code here
    return false;
  }
};

int main(int argc, char* argv[]) {
  bool result = TaskScheduling::isSchedulingPossible(
      3, vector<vector<int>>{vector<int>{0, 1}, vector<int>{1, 2}});
  cout << "Tasks execution possible: " << result << endl;
```

```
    result = TaskScheduling::isSchedulingPossible(
        3, vector<vector<int>>{vector<int>{0, 1}, vector<int>
    cout << "Tasks execution possible: " << result << endl;

    result = TaskScheduling::isSchedulingPossible(
        6, vector<vector<int>>{vector<int>{2, 5}, vector<int>{0, 5}, vector<in
                               vector<int>{1, 4}, vector<int>{3, 2}, vector<in
```

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):

C++

```
using namespace std;

#include <iostream>
#include <queue>
#include <unordered_map>
#include <vector>

class TaskScheduling {
 public:
  static bool isSchedulingPossible(int tasks, const vector<vector<int>>& pre
    vector<int> sortedOrder;
```

```
      if (tasks <= 0) {
        return false;
      }

      // a. Initialize the graph
      unordered_map<int, int> inDegree;       // count of incoming edges for e
      unordered_map<int, vector<int>> graph;  // adjacency list graph
      for (int i = 0; i < tasks; i++) {
        inDegree[i] = 0;
        graph[i] = vector<int>();
      }

      // b. Build the graph
      for (int i = 0; i < prerequisites.size(); i++) {
        int parent = prerequisites[i][0], child = prerequisites[i][1];
        graph[parent].push_back(child);  // put the child into it's parent's l
```

Time complexity#

In step ‘d’, each task can become a source only once, and each edge (i.e., 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 lis re is 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