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 to f
inish
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++

```python
def is_scheduling_possible(tasks, prerequisites):
  # TODO: Write your code here
  return False


def main():
  print("Is scheduling possible: " +
        str(is_scheduling_possible(3, [[0, 1], [1, 2]])))
  print("Is scheduling possible: " +
        str(is_scheduling_possible(3, [[0, 1], [1, 2], [2, 0]])))
  print("Is scheduling possible: " +
        str(is_scheduling_possible(6, [[0, 4], [1, 4], [3, 2], [1, 3]])))

main()
```

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):

Java Python3 C++ JS

```python
from collections import deque


def is_scheduling_possible(tasks, prerequisites):
  sortedOrder = []
  if tasks <= 0:
    return False

  # a. Initialize the graph
  inDegree = {i: 0 for i in range(tasks)}  # count of incoming edges
  graph = {i: [] for i in range(tasks)}  # adjacency list graph

  # b. Build the graph
  for prerequisite in prerequisites:
    parent, child = prerequisite[0], prerequisite[1]
    graph[parent].append(child)  # put the child into it's parent's list
    inDegree[child] += 1  # increment child's inDegree

  # c. Find all sources i.e., all vertices with 0 in-degrees
  sources = deque()
  for key in inDegree:
    if inDegree[key] == 0:
      sources.append(key)

  # d. For each source, add it to the sortedOrder and subtract one from all of its children's in-degrees
  # if a child's in-degree becomes zero, add it to the sources queue
  while sources:
    vertex = sources.popleft()
    sortedOrder.append(vertex)
    for child in graph[vertex]:  # get the node's children to decrement their in-degrees
      inDegree[child] -= 1
      if inDegree[child] == 0:
        sources.append(child)

  # if sortedOrder doesn't contain all tasks, there is a cyclic dependency between tasks, therefore, we
  # will not be able to schedule all tasks
  return len(sortedOrder) == tasks


def main():
  print("Is scheduling possible: " +
        str(is_scheduling_possible(3, [[0, 1], [1, 2]])))
  print("Is scheduling possible: " +
        str(is_scheduling_possible(3, [[0, 1], [1, 2], [2, 0]])))
  print("Is scheduling possible: " +
        str(is_scheduling_possible(6, [[0, 4], [1, 4], [3, 2], [1, 3]])))

main()
```

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.

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)

Completed

Report an Issue Ask a Question