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 to f
inish
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:

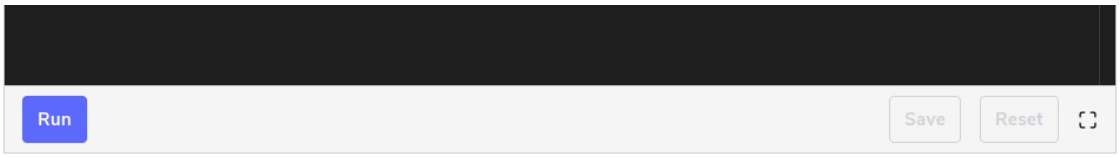
```
Input: Tasks=6, Prerequisites=[2, 5], [0, 5], [0, 4], [1, 4], [3, 2], [1, 3]
Output: [0 1 4 3 2 5]
Explanation: A possible scheduling of tasks is: [0 1 4 3 2 5]
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```python
def find_order(tasks, prerequisites):
  sortedOrder = []
  # TODO: Write your code here
  return sortedOrder


def main():
  print("Is scheduling possible: " + str(find_order(3, [[0, 1], [1, 2]])))
  print("Is scheduling possible: " +
        str(find_order(3, [[0, 1], [1, 2], [2, 0]])))
  print("Is scheduling possible: " +
        str(find_order(6, [[2, 5], [0, 5], [0, 4], [1, 4], [3, 2], [1, 3]])))


main()

```



Solution

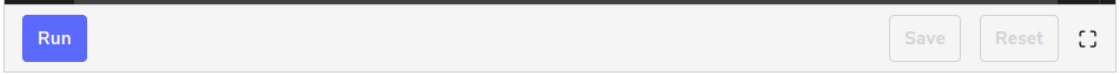
This problem is similar to Tasks Scheduling, the only difference being that we need to find the best ordering of tasks so that it is possible to schedule them all.

Code

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

Java | Python3 | C++ | JS

```python
from collections import deque


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

  # 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
  if len(sortedOrder) != tasks:
    return []

  return sortedOrder


def main():
  print("Is scheduling possible: " + str(find_order(3, [[0, 1], [1, 2]])))
  print("Is scheduling possible: " +
        str(find_order(3, [[0, 1], [1, 2], [2, 0]])))
  print("Is scheduling possible: " +
        str(find_order(6, [[2, 5], [0, 5], [0, 4], [1, 4], [3, 2], [1, 3]])))


main()
```



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.

← Back

Tasks Scheduling (medium)

Next →

All Tasks Scheduling Orders (hard)

Completed

Report an Issue Ask a Question