All Tasks Scheduling Orders (hard)

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 print all possible ordering of tasks meeting all prerequisites.

Example 1:

```
Input: Tasks=3, Prerequisites=[0, 1], [1, 2]
Output: [0, 1, 2]
Explanation: There is only possible ordering of the tasks.
```

Example 2:

```
Input: Tasks=4, Prerequisites=[3, 2], [3, 0], [2, 0], [2, 1]
Output:
1) [3, 2, 0, 1]
2) [3, 2, 1, 0]
Explanation: There are two possible orderings of the tasks meeting all prerequisites.
```

Example 3:

```
Input: Tasks=6, Prerequisites=[2, 5], [0, 5], [0, 4], [1, 4], [3, 2], [1, 3]
Output:
1) [0, 1, 4, 3, 2, 5]
2) [0, 1, 3, 4, 2, 5]
3) [0, 1, 3, 2, 4, 5]
4) [0, 1, 3, 2, 5, 4]
5) [1, 0, 3, 4, 2, 5]
6) [1, 0, 3, 2, 4, 5]
7) [1, 0, 3, 2, 5, 4]
8) [1, 0, 4, 3, 2, 5]
9) [1, 3, 0, 2, 4, 5]
10) [1, 3, 0, 2, 5, 4]
11) [1, 3, 0, 4, 2, 5]
12) [1, 3, 2, 0, 5, 4]
13) [1, 3, 2, 0, 4, 5]
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```python
def print_orders(tasks, prerequisites):
  # TODO: Write your code here
  print()


```

```python

def main():
  print("Task Orders: ")
  print_orders(3, [[0, 1], [1, 2]])

  print("Task Orders: ")
  print_orders(4, [[3, 2], [3, 0], [2, 0], [2, 1]])

  print("Task Orders: ")
  print_orders(6, [[2, 5], [0, 5], [0, 4], [1, 4], [3, 2], [1, 3]])


main()

```

Run

Save

Reset

Solution

This problem is similar to Tasks Scheduling Order, the only difference is that we need to find all the topological orderings of the tasks.

At any stage, if we have more than one source available and since we can choose any source, therefore, in this case, we will have multiple orderings of the tasks. We can use a recursive approach with **Backtracking** to consider all sources at any step.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```python
from collections import deque


def print_orders(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)

  print_all_topological_sorts(graph, inDegree, sources, sortedOrder)


def print_all_topological_sorts(graph, inDegree, sources, sortedOrder):
  if sources:
    for vertex in sources:
      sortedOrder.append(vertex)
      sourcesForNextCall = deque(sources)  # make a copy of sources
      # only remove the current source, all other sources should remain in the queue for the next call
      sourcesForNextCall.remove(vertex)
      # get the node's children to decrement their in-degrees
      for child in graph[vertex]:
        inDegree[child] -= 1
        if inDegree[child] == 0:
          sourcesForNextCall.append(child)

      # recursive call to print other orderings from the remaining (and new) sources
      print_all_topological_sorts(
        graph, inDegree, sourcesForNextCall, sortedOrder)

      # backtrack, remove the vertex from the sorted order and put all of its children back to consider
```

```python
      # the next source instead of the current vertex
      sortedOrder.remove(vertex)
      for child in graph[vertex]:
        inDegree[child] += 1

  # if sortedOrder doesn't contain all tasks, either we've a cyclic dependency between tasks, or
  # we have not processed all the tasks in this recursive call
  if len(sortedOrder) == len(inDegree):
    print(sortedOrder)


def main():
  print("Task Orders: ")
  print_orders(3, [[0, 1], [1, 2]])

  print("Task Orders: ")
  print_orders(4, [[3, 2], [3, 0], [2, 0], [2, 1]])

  print("Task Orders: ")
  print_orders(6, [[2, 5], [0, 5], [0, 4], [1, 4], [3, 2], [1, 3]])


main()
```

Run

Save Reset

Time and Space Complexity

If we don't have any prerequisites, all combinations of the tasks can represent a topological ordering. As we know, that there can be $N!$ combinations for 'N' numbers, therefore the time and space complexity of our algorithm will be $O(V! * E)$ where 'V' is the total number of tasks and 'E' is the total prerequisites. We need the 'E' part because in each recursive call, at max, we remove (and add back) all the edges.

← Back

Tasks Scheduling Order (medium)

Next →

Alien Dictionary (hard)

Completed

Report an Issue Ask a Question