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
rerequisites.
```

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:

C++

```
using namespace std;

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

class AllTaskSchedulingOrders {
 public:
  static void printOrders(int tasks, vector<vector<int>> &pr
    vector<int> sortedOrder;
    // TODO: Write your code here
  }
};

int main(int argc, char *argv[]) {
  vector<vector<int>> vec = {{0, 1}, {1, 2}};
  AllTaskSchedulingOrders::printOrders(3, vec);
  cout << endl;

  vec = {{3, 2}, {3, 0}, {2, 0}, {2, 1}};
  AllTaskSchedulingOrders::printOrders(4, vec);
  cout << endl;
```

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:

C++

```
using namespace std;

#include <algorithm>
```

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

class AllTaskSchedulingOrders {
 public:
  static void printOrders(int tasks, vector<vector<int>> &prerequisites) {
    vector<int> sortedOrder;
    if (tasks <= 0) {
      return;
    }

    // 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++) {
```

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

Next →

Tasks Scheduling Order (medium)

ic (h

✓ Mark as Completed

Report an Issue