



# All Tasks Scheduling Orders (hard)



### 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.
```

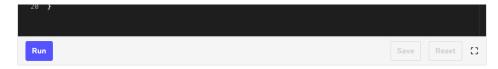
#### Example 3:

```
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:



# 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++
                                      ıs JS
🍨 Java
     lmport java.util.*:
    class AllTaskSchedulingOrders {
      public static void printOrders(int tasks, int[][] prerequisites) {
        List<Integer> sortedOrder = new ArrayList<>();
         if (tasks <= 0)
        HashMap<Integer, Integer> inDegree = new HashMap⇔(); // count of incoming edges for every vertex
        HashMap<Integer, List<Integer>> graph = new HashMap<>(); // adjacency list graph
         for (int i = 0; i < tasks; i++) {
           inDegree.put(i, 0);
          graph.put(i, new ArrayList<Integer>());
         for (int i = 0; i < prerequisites.length; i++) {</pre>
          int parent = prerequisites[i][0], child = prerequisites[i][1];
          graph.get(parent).add(child); // put the child into it's parent's list
           inDegree.put(child, inDegree.get(child) + 1); // increment child's inDegree
        Queue<Integer> sources = new LinkedList<>();
         for (Map.Entry<Integer, Integer> entry : inDegree.entrySet()) {
  if (entry.getValue() == 0)
             sources.add(entry.getKey())
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.

