

Grokking the Coding Interview: Patterns for Coding Questions

94% completed



Introduction ▾

Pattern: Sliding Window ▾

Pattern: Two Pointers ▾

Pattern: Fast & Slow pointers ▾

Pattern: Merge Intervals ▾

Pattern: Cyclic Sort ▾

Pattern: In-place Reversal of a LinkedList ▾

Pattern: Tree Breadth First Search ▾

Pattern: Tree Depth First Search ▾

Pattern: Two Heaps ▾

Pattern: Subsets ▾

Pattern: Modified Binary Search ▾

Pattern: Bitwise XOR ▾

Pattern: Top 'K' Elements ▾

Pattern: K-way merge ▾

Pattern : 0/1 Knapsack (Dynamic Programming) ▾

Pattern: Topological Sort (Graph) ▴

- Introduction
- Topological Sort (medium)
- Tasks Scheduling (medium)
- Tasks Scheduling Order (medium)
- All Tasks Scheduling Orders (hard)
- Alien Dictionary (hard)

All Tasks Scheduling Orders (hard)

We'll cover the following ▴

- Problem Statement
- Try it yourself
- Solution
 - Code
 - Time and Space Complexity

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:

JavaPython3JS C++

```
1 import java.util.*;
2
3 class AllTaskSchedulingOrders {
4     public static void printOrders(int tasks, int[][] prerequisites) {
5         // TODO: Write your code here
6     }
7
8     public static void main(String[] args) {
9         AllTaskSchedulingOrders.printOrders(3, new int[][] { new int[] { 0, 1 }, new int[] { 1, 2 } });
10        System.out.println();
11
12        AllTaskSchedulingOrders.printOrders(4,
13            new int[][] { new int[] { 3, 2 }, new int[] { 3, 0 }, new int[] { 2, 0 }, new int[] { 2, 1 } });
14        System.out.println();
15
16        AllTaskSchedulingOrders.printOrders(6, new int[][] { new int[] { 2, 5 }, new int[] { 0, 5 }, new int[] { 0, 4 },
17            new int[] { 1, 4 }, new int[] { 3, 2 }, new int[] { 1, 3 } });
18        System.out.println();
19    }
20 }
```

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.

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Miscellaneous

Kth Smallest Number (hard)

Conclusions

Where to Go from Here

Mark Course as Completed

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

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:

JavaPython3C++JS

```
1 import java.util.*;
2
3 class AllTaskSchedulingOrders {
4     public static void printOrders(int tasks, int[][] prerequisites) {
5         List<Integer> sortedOrder = new ArrayList<>();
6         if (tasks <= 0)
7             return;
8
9         // a. Initialize the graph
10        HashMap<Integer, Integer> inDegree = new HashMap<>(); // count of incoming edges for every vertex
11        HashMap<Integer, List<Integer>> graph = new HashMap<>(); // adjacency list graph
12        for (int i = 0; i < tasks; i++) {
13            inDegree.put(i, 0);
14            graph.put(i, new ArrayList<Integer>());
15        }
16
17        // b. Build the graph
18        for (int i = 0; i < prerequisites.length; i++) {
19            int parent = prerequisites[i][0], child = prerequisites[i][1];
20            graph.get(parent).add(child); // put the child into it's parent's list
21            inDegree.put(child, inDegree.get(child) + 1); // increment child's inDegree
22        }
23
24        // c. Find all sources i.e., all vertices with 0 in-degrees
25        Queue<Integer> sources = new LinkedList<>();
26        for (Map.Entry<Integer, Integer> entry : inDegree.entrySet()) {
27            if (entry.getValue() == 0)
28                sources.add(entry.getKey());
```

RUN

SAVE

RESET

↺

Close

1.663s

Output

[0, 1, 2]

[3, 2, 0, 1]
[3, 2, 1, 0]

[0, 1, 4, 3, 2, 5]
[0, 1, 3, 4, 2, 5]
[0, 1, 3, 2, 4, 5]
[0, 1, 3, 2, 5, 4]

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.

Interviewing soon? We've partnered with [Hired](#) so that companies apply to you instead of you applying to them. [See how](#) Ⓞ

← Back

Next →

Tasks Scheduling Order (medium)Alien Dictionary (hard)

✓

 Mark as Completed

Report an Issue Ask a Question