6163. Build a Matrix With Conditions

User Accepted:	0
User Tried:	3
Total Accepted:	0
Total Submissions:	3
Difficulty:	Hard

You are given a **positive** integer `k`. You are also given:

- a 2D integer array `rowConditions` of size `n` where `rowConditions[i] = [above_i, below_i]`, and
- a 2D integer array `colConditions` of size `m` where `colConditions[i] = [left_i, right_i]`.

The two arrays contain integers from `1` to `k`.

You have to build a `k x k` matrix that contains each of the numbers from `1` to `k` **exactly once**. The remaining cells should have the value `0`.

The matrix should also satisfy the following conditions:

- The number `above_i` should appear in a **row** that is strictly **above** the row at which the number `below_i` appears for all `i` from `0` to `n - 1`.
- The number `left_i` should appear in a **column** that is strictly **left** of the column at which the number `right_i` appears for all `i` from `0` to `m - 1`.

Return ***any*** *matrix that satisfies the conditions*. If no answer exists, return an empty matrix.

Example 1:

3	0	0
0	0	1
0	2	0

```
Input: k = 3, rowConditions = [[1,2],[3,2]], colConditions = [[2,1],[3,2]]
Output: [[3,0,0],[0,0,1],[0,2,0]]
Explanation: The diagram above shows a valid example of a matrix that satisfies all the conditions.
The row conditions are the following:
- Number 1 is in row 1, and number 2 is in row 2, so 1 is above 2 in the matrix.
- Number 3 is in row 0, and number 2 is in row 2, so 3 is above 2 in the matrix.

The column conditions are the following:
- Number 2 is in column 1, and number 1 is in column 2, so 2 is left of 1 in the matrix.
- Number 3 is in column 0, and number 2 is in column 1, so 3 is left of 2 in the matrix.
Note that there may be multiple correct answers.
```

Example 2:

```
Input: k = 3, rowConditions = [[1,2],[2,3],[3,1],[2,3]], colConditions = [[2,1]]
Output: []
Explanation: From the first two conditions, 3 has to be below 1 but the third conditions needs 3 to be above 1 to be satisfied.
No matrix can satisfy all the conditions, so we return the empty matrix.
```

Constraints:

- `2 <= k <= 400`
- `1 <= rowConditions.length, colConditions.length <= 10^4`
- `rowConditions[i].length == colConditions[i].length == 2`
- `1 <= above_i, below_i, left_i, right_i <= k`
- `above_i != below_i`
- `left_i != right_i`

JavaScript

```
1 const initializeGraph = (n) => { let g = []; for (let i = 0; i < n; i++) { g.push([]); } return g; };
```

```
const packDGInDegree = (g, edges, indegree) => { for (const [u, v] of edges) { g[u-1].unshift(v-1); indegree[v-1]++; } };
const initialize2DArray = (n, m) => { let d = []; for (let i = 0; i < n; i++) { let t = Array(m).fill(0); d.push(t); }
return d; };

const buildMatrix = (k, rowConditions, colConditions) => {
    let gr = make(k, rowConditions), gc = make(k, colConditions), d = initialize2DArray(k, 2), res = initialize2DArray(k,
k);
    if (gr.length == 0 || gc.length == 0) return [];
    for (let i = 0; i < k; i++) {
        d[gr[i]][0] = i;
        d[gc[i]][1] = i;
    }
    for (let i = 0; i < k; i++) {
        let [x, y] = d[i];
        res[x][y] = i + 1;
    }
    return res;
};

const make = (n, edges) => {
    let g = initializeGraph(n), deg = Array(n).fill(0);
    packDGInDegree(g, edges, deg);
    return topologicalSort(g, deg);
};

const topologicalSort = (g, indegree) => {
    let res = [], q = [], n = g.length;
    for (let i = 0; i < n; i++) {
        if (indegree[i] == 0) q.push(i);
    }
    while (q.length) {
        let cur = q.shift();
        res.push(cur);
        for (const child of g[cur]) {
            indegree[child]--;
            if (indegree[child] == 0) q.push(child);
        }
    }
    for (let i = 0; i < n; i++) {
        if (indegree[i] > 0) return [];
    }
    return res;
};
```

Custom Testcase | Use Example Testcases

Run | Submit

Submission Result: Accepted (/submissions/detail/785291734/) | More Details > (/submissions/detail/785291734/)

Share your acceptance!