2392. Build a Matrix With Conditions

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.
```

2392. Build a Matrix With Conditions

```
/*Time complexity: O(n+m+klogk)
  Extra space complexity: O(n+m+k)*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;

class Solution {
  public:
    vi* above_graph=nullptr;
    vi* left_graph=nullptr;

  public:
    void build_graph(int k,vvi& edges,vi*& graph){
      graph=new vi[k];
      for(auto& edge: edges){
        int u=edge[0],v=edge[1];
        graph[u].push_back(v);
      }
    }

    vi top_sort(int k,vi*& graph){
      vi indegree(k,0);
      for(int node=1;node<k;++node){
        for(auto& u: graph[node]){
          indegree[u]++;
        }
      }

      std::queue<int> q;

      for(int node=1;node<k;++node){
        if(indegree[node]==0) q.push(node);
      }

      vi ans;
      while(!q.empty()){
        int node=q.front();
        q.pop();

        ans.push_back(node);

        for(auto& u: graph[node]){
          indegree[u]--;
          if(indegree[u]==0) q.push(u);
        }
      }

      for(int node=1;node<k;++node){
        if(indegree[node]!=0) return {};
      }
      return ans;

    }
```

```
    vvi buildMatrix(int k, vvi& rowConditions, vvi& colConditions){
        k++;

        build_graph(k,rowConditions,above_graph);
        build_graph(k,colConditions,left_graph);

        vi top_sort_above=top_sort(k,above_graph);
        vi top_sort_left=top_sort(k,left_graph);

        if(top_sort_above.empty() || top_sort_left.empty() ) return {};

        k--;

        std::unordered_map<int,int> row_pos;
        for(int i=0;i<k;++i) row_pos[top_sort_above[i]]=i;


        vvi ans(k,vi(k,0));
        for(int i=0;i<k;++i) ans[row_pos[top_sort_left[i]]][i]=top_sort_left[i];

        return ans;
    }
};
```