Problem 2392: Build a Matrix With Conditions

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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

.

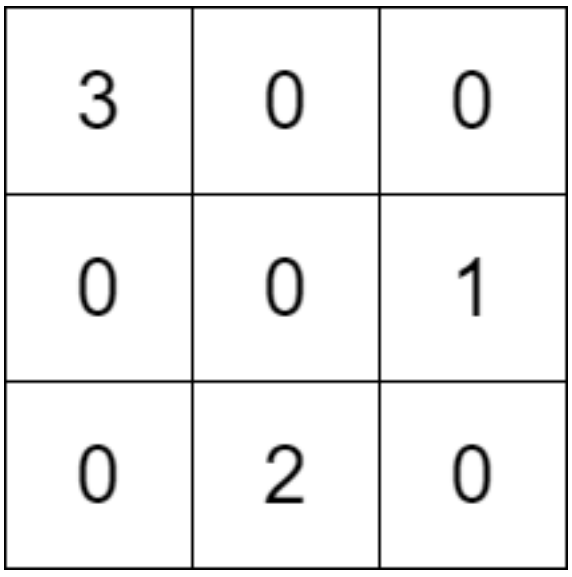
Return

any

matrix that satisfies the conditions

. If no answer exists, return an empty matrix.

Example 1:



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

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> buildMatrix(int k, vector<vector<int>>& rowConditions,
vector<vector<int>>& colConditions) {

}
};
```

Java:

```
class Solution {
public int[][] buildMatrix(int k, int[][] rowConditions, int[][]
colConditions) {

}
```

```
}
```

Python3:

```
class Solution:
def buildMatrix(self, k: int, rowConditions: List[List[int]], colConditions:
List[List[int]]) -> List[List[int]]:
```

Python:

```
class Solution(object):
def buildMatrix(self, k, rowConditions, colConditions):
"""
:type k: int
:type rowConditions: List[List[int]]
:type colConditions: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number} k
* @param {number[][]} rowConditions
* @param {number[][]} colConditions
* @return {number[][]}
*/
var buildMatrix = function(k, rowConditions, colConditions) {

};
```

TypeScript:

```
function buildMatrix(k: number, rowConditions: number[][], colConditions:
number[][]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] BuildMatrix(int k, int[][] rowConditions, int[][]
```

```
colConditions) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** buildMatrix(int k, int** rowConditions, int rowConditionsSize, int*
rowConditionsColSize, int** colConditions, int colConditionsSize, int*
colConditionsColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func buildMatrix(k int, rowConditions [][]int, colConditions [][]int) [][]int
{

}
```

Kotlin:

```
class Solution {
fun buildMatrix(k: Int, rowConditions: Array<IntArray>, colConditions:
Array<IntArray>): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func buildMatrix(_ k: Int, _ rowConditions: [[Int]], _ colConditions:
[[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn build_matrix(k: i32, row_conditions: Vec<Vec<i32>>, col_conditions:
Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer} k
# @param {Integer[][]} row_conditions
# @param {Integer[][]} col_conditions
# @return {Integer[][]}
def build_matrix(k, row_conditions, col_conditions)

end
```

PHP:

```
class Solution {

/**
* @param Integer $k
* @param Integer[][] $rowConditions
* @param Integer[][] $colConditions
* @return Integer[][]
*/
function buildMatrix($k, $rowConditions, $colConditions) {

}
}
```

Dart:

```
class Solution {
List<List<int>> buildMatrix(int k, List<List<int>> rowConditions,
List<List<int>> colConditions) {

}
}
```

Scala:

```
object Solution {
def buildMatrix(k: Int, rowConditions: Array[Array[Int]], colConditions:
Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec build_matrix(k :: integer, row_conditions :: [[integer]],
col_conditions :: [[integer]]) :: [[integer]]
def build_matrix(k, row_conditions, col_conditions) do

end
end
```

Erlang:

```
-spec build_matrix(K :: integer(), RowConditions :: [[integer()]],
ColConditions :: [[integer()]]) -> [[integer()]].
build_matrix(K, RowConditions, ColConditions) ->
.
```

Racket:

```
(define/contract (build-matrix k rowConditions colConditions)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Build a Matrix With Conditions
* Difficulty: Hard
* Tags: array, graph, sort
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<vector<int>> buildMatrix(int k, vector<vector<int>>& rowConditions,
vector<vector<int>>& colConditions) {

}
};
```

Java Solution:

```
/**
* Problem: Build a Matrix With Conditions
* Difficulty: Hard
* Tags: array, graph, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int[][] buildMatrix(int k, int[][] rowConditions, int[][]
colConditions) {

}
}
```

Python3 Solution:

```
"""
Problem: Build a Matrix With Conditions
Difficulty: Hard
Tags: array, graph, sort

Approach: Use two pointers or sliding window technique
```

```
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def buildMatrix(self, k: int, rowConditions: List[List[int]], colConditions:
List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def buildMatrix(self, k, rowConditions, colConditions):
"""
:type k: int
:type rowConditions: List[List[int]]
:type colConditions: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Build a Matrix With Conditions
* Difficulty: Hard
* Tags: array, graph, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} k
* @param {number[][]} rowConditions
* @param {number[][]} colConditions
* @return {number[][]}
*/
var buildMatrix = function(k, rowConditions, colConditions) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Build a Matrix With Conditions
* Difficulty: Hard
* Tags: array, graph, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function buildMatrix(k: number, rowConditions: number[][], colConditions:
number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Build a Matrix With Conditions
* Difficulty: Hard
* Tags: array, graph, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int[][] BuildMatrix(int k, int[][] rowConditions, int[][]
colConditions) {

}
}
```

C Solution:

```
/*
* Problem: Build a Matrix With Conditions
* Difficulty: Hard
* Tags: array, graph, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** buildMatrix(int k, int** rowConditions, int rowConditionsSize, int*
rowConditionsColSize, int** colConditions, int colConditionsSize, int*
colConditionsColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Build a Matrix With Conditions
// Difficulty: Hard
// Tags: array, graph, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func buildMatrix(k int, rowConditions [][]int, colConditions [][]int) [][]int
{

}
```

Kotlin Solution:

```
class Solution {
fun buildMatrix(k: Int, rowConditions: Array<IntArray>, colConditions:
Array<IntArray>): Array<IntArray> {
```

```
}
}
```

Swift Solution:

```
class Solution {
func buildMatrix(_ k: Int, _ rowConditions: [[Int]], _ colConditions:
[[Int]]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Build a Matrix With Conditions
// Difficulty: Hard
// Tags: array, graph, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn build_matrix(k: i32, row_conditions: Vec<Vec<i32>>, col_conditions:
Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer} k
# @param {Integer[][]} row_conditions
# @param {Integer[][]} col_conditions
# @return {Integer[][]}
def build_matrix(k, row_conditions, col_conditions)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $k
* @param Integer[][] $rowConditions
* @param Integer[][] $colConditions
* @return Integer[][]
*/
function buildMatrix($k, $rowConditions, $colConditions) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> buildMatrix(int k, List<List<int>> rowConditions,
List<List<int>> colConditions) {

}
}
```

Scala Solution:

```
object Solution {
def buildMatrix(k: Int, rowConditions: Array[Array[Int]], colConditions:
Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec build_matrix(k :: integer, row_conditions :: [[integer]],
col_conditions :: [[integer]]) :: [[integer]]
def build_matrix(k, row_conditions, col_conditions) do

end
end
```

Erlang Solution:

```
-spec build_matrix(K :: integer(), RowConditions :: [[integer()]],
ColConditions :: [[integer()]]) -> [[integer()]].
build_matrix(K, RowConditions, ColConditions) ->
.
```

Racket Solution:

```
(define/contract (build-matrix k rowConditions colConditions)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) (listof (listof exact-integer?)))
)
```