Problem 3311: Construct 2D Grid Matching Graph Layout

Problem Information

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

Problem Description

You are given a 2D integer array

edges

representing an

undirected

graph having

n

nodes, where

edges[i] = [u

i

, v

i

]

denotes an edge between nodes

u

i

and

v

i

.

Construct a 2D grid that satisfies these conditions:

The grid contains

all nodes

from

0

to

n - 1

in its cells, with each node appearing exactly

once

.

Two nodes should be in adjacent grid cells (

horizontally

or

vertically

)

if and only if

there is an edge between them in

edges

.

It is guaranteed that

edges

can form a 2D grid that satisfies the conditions.

Return a 2D integer array satisfying the conditions above. If there are multiple solutions, return

any

of them.

Example 1:

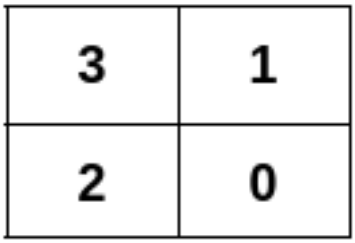
Input:

n = 4, edges = [[0,1],[0,2],[1,3],[2,3]]

Output:

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

Explanation:



Example 2:

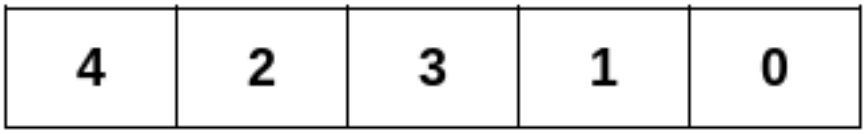
Input:

n = 5, edges = [[0,1],[1,3],[2,3],[2,4]]

Output:

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

Explanation:



Example 3:

Input:

n = 9, edges = [[0,1],[0,4],[0,5],[1,7],[2,3],[2,4],[2,5],[3,6],[4,6],[4,7],[6,8],[7,8]]

Output:

[[8,6,3],[7,4,2],[1,0,5]]

Explanation:

8	6	3
7	4	2
1	0	5

Constraints:

2 <= n <= 5 * 10

4

1 <= edges.length <= 10

5

edges[i] = [u

i

, v

i

]

0 <= u

i

< v

i

< n

All the edges are distinct.

The input is generated such that

edges

can form a 2D grid that satisfies the conditions.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> constructGridLayout(int n, vector<vector<int>>& edges) {

}
};
```

Java:

```
class Solution {
public int[][] constructGridLayout(int n, int[][] edges) {

}
}
```

Python3:

```
class Solution:
def constructGridLayout(self, n: int, edges: List[List[int]]) ->
List[List[int]]:
```

Python:

```
class Solution(object):
def constructGridLayout(self, n, edges):
"""
:type n: int
:type edges: List[List[int]]
:rtype: List[List[int]]
```

```
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @return {number[][]}
*/
var constructGridLayout = function(n, edges) {

};
```

TypeScript:

```
function constructGridLayout(n: number, edges: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] ConstructGridLayout(int n, int[][] edges) {

}
}
```

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** constructGridLayout(int n, int** edges, int edgesSize, int*
edgesColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func constructGridLayout(n int, edges [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun constructGridLayout(n: Int, edges: Array<IntArray>): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func constructGridLayout(_ n: Int, _ edges: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn construct_grid_layout(n: i32, edges: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @return {Integer[][]}
def construct_grid_layout(n, edges)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
```

```
* @param Integer[][] $edges
* @return Integer[][]
*/
function constructGridLayout($n, $edges) {

}
}
```

Dart:

```
class Solution {
List<List<int>> constructGridLayout(int n, List<List<int>> edges) {

}
}
```

Scala:

```
object Solution {
def constructGridLayout(n: Int, edges: Array[Array[Int]]): Array[Array[Int]]
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec construct_grid_layout(n :: integer, edges :: [[integer]]) ::
[[integer]]
def construct_grid_layout(n, edges) do

end
end
```

Erlang:

```
-spec construct_grid_layout(N :: integer(), Edges :: [[integer()]]) ->
[[integer()]].
construct_grid_layout(N, Edges) ->
.
```

Racket:

```
(define/contract (construct-grid-layout n edges)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Construct 2D Grid Matching Graph Layout
* Difficulty: Hard
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<vector<int>> constructGridLayout(int n, vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Construct 2D Grid Matching Graph Layout
* Difficulty: Hard
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
```

```
public int[][] constructGridLayout(int n, int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Construct 2D Grid Matching Graph Layout
Difficulty: Hard
Tags: array, graph, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def constructGridLayout(self, n: int, edges: List[List[int]]) ->
List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def constructGridLayout(self, n, edges):
"""
:type n: int
:type edges: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Construct 2D Grid Matching Graph Layout
* Difficulty: Hard
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} n
* @param {number[][]} edges
* @return {number[][]}
*/
var constructGridLayout = function(n, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Construct 2D Grid Matching Graph Layout
* Difficulty: Hard
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function constructGridLayout(n: number, edges: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Construct 2D Grid Matching Graph Layout
* Difficulty: Hard
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
public class Solution {
public int[][] ConstructGridLayout(int n, int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Construct 2D Grid Matching Graph Layout
* Difficulty: Hard
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* 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** constructGridLayout(int n, int** edges, int edgesSize, int*
edgesColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Construct 2D Grid Matching Graph Layout
// Difficulty: Hard
// Tags: array, graph, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func constructGridLayout(n int, edges [][]int) [][]int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun constructGridLayout(n: Int, edges: Array<IntArray>): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func constructGridLayout(_ n: Int, _ edges: [[Int]]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Construct 2D Grid Matching Graph Layout
// Difficulty: Hard
// Tags: array, graph, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn construct_grid_layout(n: i32, edges: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @return {Integer[][]}
def construct_grid_layout(n, edges)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @return Integer[][]
*/
function constructGridLayout($n, $edges) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> constructGridLayout(int n, List<List<int>> edges) {

}
}
```

Scala Solution:

```
object Solution {
def constructGridLayout(n: Int, edges: Array[Array[Int]]): Array[Array[Int]]
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec construct_grid_layout(n :: integer, edges :: [[integer]]) ::
[[integer]]
def construct_grid_layout(n, edges) do

end
```

```
end
```

Erlang Solution:

```
-spec construct_grid_layout(N :: integer(), Edges :: [[integer()]]) ->
[[integer()]].
construct_grid_layout(N, Edges) ->
.
```

Racket Solution:

```
(define/contract (construct-grid-layout n edges)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)))
)
```