Problem 2077: Paths in Maze That Lead to Same Room

Problem Information

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

Problem Description

A maze consists of

n

rooms numbered from

1

to

n

, and some rooms are connected by corridors. You are given a 2D integer array

corridors

where

corridors[i] = [room1

i

, room2

i

]

indicates that there is a corridor connecting

room1

i

and

room2

i

, allowing a person in the maze to go from

room1

i

to

room2

i

and vice versa

.

The designer of the maze wants to know how confusing the maze is. The

confusion

score

of the maze is the number of different cycles of length 3

.

For example,

1 → 2 → 3 → 1

is a cycle of length 3, but

1 → 2 → 3 → 4

and

1 → 2 → 3 → 2 → 1

are not.

Two cycles are considered to be

different

if one or more of the rooms visited in the first cycle is

not

in the second cycle.

Return

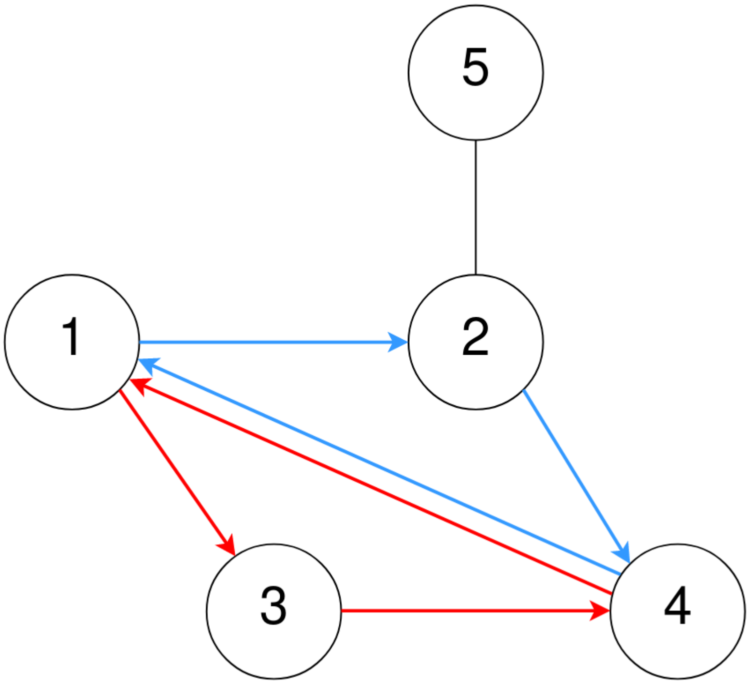
the

confusion

score

of the maze.

Example 1:



Input:

n = 5, corridors = [[1,2],[5,2],[4,1],[2,4],[3,1],[3,4]]

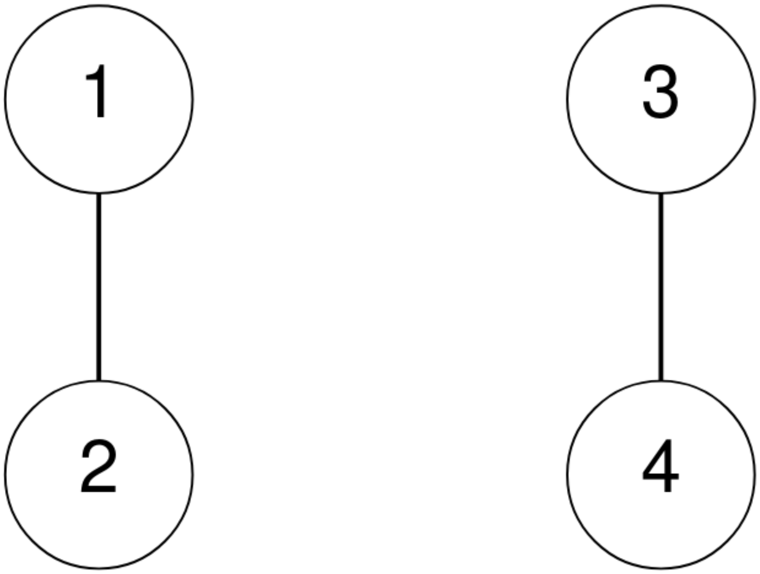
Output:

2

Explanation:

One cycle of length 3 is 4 → 1 → 3 → 4, denoted in red. Note that this is the same cycle as 3 → 4 → 1 → 3 or 1 → 3 → 4 → 1 because the rooms are the same. Another cycle of length 3 is 1 → 2 → 4 → 1, denoted in blue. Thus, there are two different cycles of length 3.

Example 2:



Input:

n = 4, corridors = [[1,2],[3,4]]

Output:

0

Explanation:

There are no cycles of length 3.

Constraints:

2 <= n <= 1000

1 <= corridors.length <= 5 * 10

4

corridors[i].length == 2

1 <= room1

i

, room2

i

<= n

room1

i

!= room2

i

There are no duplicate corridors.

Code Snippets

C++:

```
class Solution {
public:
int numberOfPaths(int n, vector<vector<int>>& corridors) {

}
};
```

Java:

```
class Solution {
public int numberOfPaths(int n, int[][] corridors) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function numberOfPaths(n: number, corridors: number[][]): number {

};
```

C#:

```
public class Solution {
public int NumberOfPaths(int n, int[][] corridors) {

}
}
```

C:

```
int numberOfPaths(int n, int** corridors, int corridorsSize, int*
corridorsColSize) {

}
```

Go:

```
func numberOfPaths(n int, corridors [][]int) int {

}
```

Kotlin:

```
class Solution {
fun numberOfPaths(n: Int, corridors: Array<IntArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} corridors
# @return {Integer}
def number_of_paths(n, corridors)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $corridors
* @return Integer
*/
function numberOfPaths($n, $corridors) {

}
}
```

Dart:

```
class Solution {
int numberOfPaths(int n, List<List<int>> corridors) {

}
}
```

Scala:

```
object Solution {
def numberOfPaths(n: Int, corridors: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec number_of_paths(n :: integer, corridors :: [[integer]]) :: integer
def number_of_paths(n, corridors) do

end
end
```

Erlang:

```
-spec number_of_paths(N :: integer(), Corridors :: [[integer()]]) ->
integer().
```

```
number_of_paths(N, Corridors) ->
.
```

Racket:

```
(define/contract (number-of-paths n corridors)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Paths in Maze That Lead to Same Room
* Difficulty: Medium
* Tags: array, graph
*
* 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 numberOfPaths(int n, vector<vector<int>>& corridors) {

}
};
```

Java Solution:

```
/**
* Problem: Paths in Maze That Lead to Same Room
* Difficulty: Medium
* Tags: array, graph
*
* 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 numberOfPaths(int n, int[][] corridors) {

}
}
```

Python3 Solution:

```
"""
Problem: Paths in Maze That Lead to Same Room
Difficulty: Medium
Tags: array, graph

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 numberOfPaths(self, n: int, corridors: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Paths in Maze That Lead to Same Room
* Difficulty: Medium
* Tags: array, graph
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Paths in Maze That Lead to Same Room
* Difficulty: Medium
* Tags: array, graph
*
* 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 numberOfPaths(n: number, corridors: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Paths in Maze That Lead to Same Room
* Difficulty: Medium
* Tags: array, graph
*
* 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 NumberOfPaths(int n, int[][] corridors) {

}
}
```

C Solution:

```
/*
* Problem: Paths in Maze That Lead to Same Room
* Difficulty: Medium
* Tags: array, graph
*
* 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
*/

int numberOfPaths(int n, int** corridors, int corridorsSize, int*
corridorsColSize) {

}
```

Go Solution:

```
// Problem: Paths in Maze That Lead to Same Room
// Difficulty: Medium
// Tags: array, graph
//
// 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 numberOfPaths(n int, corridors [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfPaths(n: Int, corridors: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Paths in Maze That Lead to Same Room
// Difficulty: Medium
// Tags: array, graph
//
// 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 number_of_paths(n: i32, corridors: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} corridors
# @return {Integer}
def number_of_paths(n, corridors)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $corridors
* @return Integer
*/
function numberOfPaths($n, $corridors) {

}
}
```

Dart Solution:

```
class Solution {
int numberOfPaths(int n, List<List<int>> corridors) {

}
}
```

Scala Solution:

```
object Solution {
def numberOfPaths(n: Int, corridors: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_of_paths(n :: integer, corridors :: [[integer]]) :: integer
def number_of_paths(n, corridors) do

end
end
```

Erlang Solution:

```
-spec number_of_paths(N :: integer(), Corridors :: [[integer()]]) ->
integer().
number_of_paths(N, Corridors) ->
```

```
.
```

Racket Solution:

```
(define/contract (number-of-paths n corridors)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```