Problem 1192: Critical Connections in a Network

Problem Information

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

Problem Description

There are

n

servers numbered from

0

to

n - 1

connected by undirected server-to-server

connections

forming a network where

connections[i] = [a

i

, b

i

]

represents a connection between servers

a

i

and

b

i

. Any server can reach other servers directly or indirectly through the network.

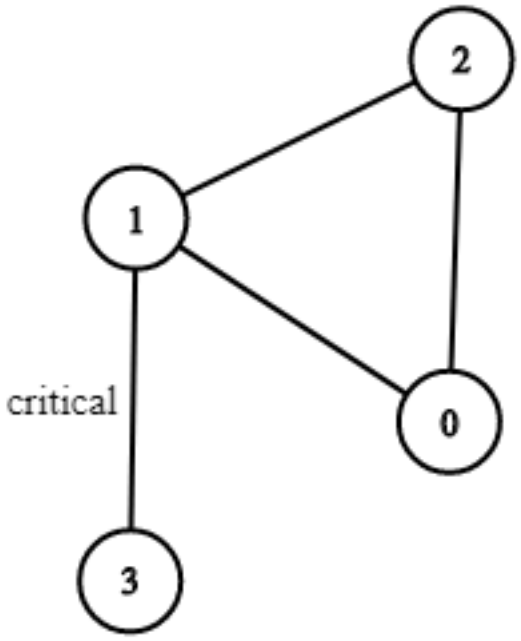
A

critical connection

is a connection that, if removed, will make some servers unable to reach some other server.

Return all critical connections in the network in any order.

Example 1:



Input:

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

Output:

[[1,3]]

Explanation:

[[3,1]] is also accepted.

Example 2:

Input:

n = 2, connections = [[0,1]]

Output:

[[0,1]]

Constraints:

2 <= n <= 10

5

n - 1 <= connections.length <= 10

5

0 <= a

i

, b

i

<= n - 1

a

i

!= b

i

There are no repeated connections.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> criticalConnections(int n, vector<vector<int>>&
connections) {

}
};
```

Java:

```
class Solution {
public List<List<Integer>> criticalConnections(int n, List<List<Integer>>
connections) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function criticalConnections(n: number, connections: number[][]): number[][]
{

};
```

C#:

```
public class Solution {
public IList<IList<int>> CriticalConnections(int n, IList<IList<int>>
connections) {

}
}
```

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** criticalConnections(int n, int** connections, int connectionsSize, int*
connectionsColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func criticalConnections(n int, connections [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun criticalConnections(n: Int, connections: List<List<Int>>):
List<List<Int>> {

}
}
```

Swift:

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

}
```

```
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} connections
# @return {Integer[][]}
def critical_connections(n, connections)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $connections
* @return Integer[][]
*/
function criticalConnections($n, $connections) {

}
}
```

Dart:

```
class Solution {
List<List<int>> criticalConnections(int n, List<List<int>> connections) {

}
}
```

Scala:

```
object Solution {
def criticalConnections(n: Int, connections: List[List[Int]]):
List[List[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec critical_connections(n :: integer, connections :: [[integer]]) ::
[[integer]]
def critical_connections(n, connections) do

end
end
```

Erlang:

```
-spec critical_connections(N :: integer(), Connections :: [[integer()]]) ->
[[integer()]].
critical_connections(N, Connections) ->
.
```

Racket:

```
(define/contract (critical-connections n connections)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Critical Connections in a Network
* Difficulty: Hard
* Tags: graph, search
*
```

```
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<vector<int>> criticalConnections(int n, vector<vector<int>>&
connections) {

}
};
```

Java Solution:

```
/**
* Problem: Critical Connections in a Network
* Difficulty: Hard
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public List<List<Integer>> criticalConnections(int n, List<List<Integer>>
connections) {

}
}
```

Python3 Solution:

```
"""
Problem: Critical Connections in a Network
Difficulty: Hard
Tags: graph, search

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
```

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

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Critical Connections in a Network
* Difficulty: Hard
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Critical Connections in a Network
* Difficulty: Hard
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function criticalConnections(n: number, connections: number[][]): number[][]
{

};
```

C# Solution:

```
/*
* Problem: Critical Connections in a Network
* Difficulty: Hard
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public IList<IList<int>> CriticalConnections(int n, IList<IList<int>>
connections) {

}
}
```

C Solution:

```
/*
* Problem: Critical Connections in a Network
* Difficulty: Hard
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
```

```
* Time Complexity: O(n) to O(n^2) depending on approach
* 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** criticalConnections(int n, int** connections, int connectionsSize, int*
connectionsColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Critical Connections in a Network
// Difficulty: Hard
// Tags: graph, search
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func criticalConnections(n int, connections [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun criticalConnections(n: Int, connections: List<List<Int>>):
List<List<Int>> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Critical Connections in a Network
// Difficulty: Hard
// Tags: graph, search
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn critical_connections(n: i32, connections: Vec<Vec<i32>>) ->
Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} connections
# @return {Integer[][]}
def critical_connections(n, connections)

end
```

PHP Solution:

```
class Solution {

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

```
}
}
```

Dart Solution:

```
class Solution {
List<List<int>> criticalConnections(int n, List<List<int>> connections) {

}
}
```

Scala Solution:

```
object Solution {
def criticalConnections(n: Int, connections: List[List[Int]]):
List[List[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec critical_connections(n :: integer, connections :: [[integer]]) ::
[[integer]]
def critical_connections(n, connections) do

end
end
```

Erlang Solution:

```
-spec critical_connections(N :: integer(), Connections :: [[integer()]]) ->
[[integer()]].
critical_connections(N, Connections) ->
.
```

Racket Solution:

```
(define/contract (critical-connections n connections)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)))
)
```