

Problem 684: Redundant Connection

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

In this problem, a tree is an

undirected graph

that is connected and has no cycles.

You are given a graph that started as a tree with

n

nodes labeled from

1

to

n

, with one additional edge added. The added edge has two

different

vertices chosen from

1

to

n

, and was not an edge that already existed. The graph is represented as an array

edges

of length

n

where

edges[i] = [a

i

, b

i

]

indicates that there is an edge between nodes

a

i

and

b

i

in the graph.

Return

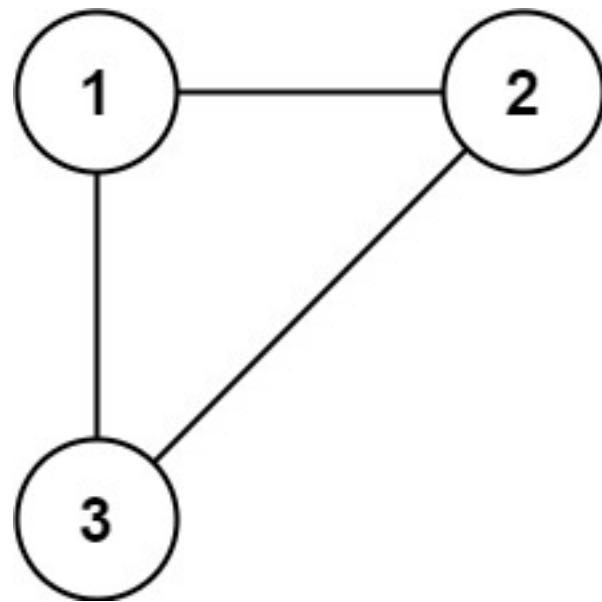
an edge that can be removed so that the resulting graph is a tree of

n

nodes

. If there are multiple answers, return the answer that occurs last in the input.

Example 1:



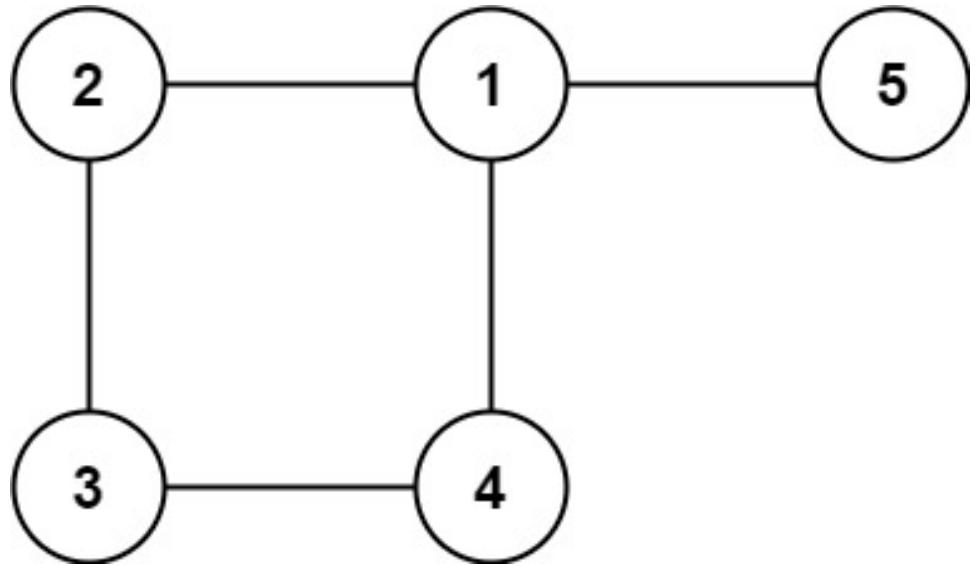
Input:

edges = [[1,2],[1,3],[2,3]]

Output:

[2,3]

Example 2:



Input:

```
edges = [[1,2],[2,3],[3,4],[1,4],[1,5]]
```

Output:

```
[1,4]
```

Constraints:

```
n == edges.length
```

```
3 <= n <= 1000
```

```
edges[i].length == 2
```

```
1 <= a
```

```
i
```

```
< b
```

```
i
```

```
<= edges.length
```

a

i

!= b

i

There are no repeated edges.

The given graph is connected.

Code Snippets

C++:

```
class Solution {  
public:  
vector<int> findRedundantConnection(vector<vector<int>>& edges) {  
  
}  
};
```

Java:

```
class Solution {  
public int[] findRedundantConnection(int[][] edges) {  
  
}  
}
```

Python3:

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

Python:

```
class Solution(object):  
def findRedundantConnection(self, edges):
```

```
"""
:type edges: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
 * @param {number[][]} edges
 * @return {number[]}
 */
var findRedundantConnection = function(edges) {
};
```

TypeScript:

```
function findRedundantConnection(edges: number[][]): number[] {
};
```

C#:

```
public class Solution {
public int[] FindRedundantConnection(int[][] edges) {
}
```

C:

```
/*
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findRedundantConnection(int** edges, int edgesSize, int* edgesColSize,
int* returnSize) {

}
```

Go:

```
func findRedundantConnection(edges [][]int) []int {  
}  
}
```

Kotlin:

```
class Solution {  
    fun findRedundantConnection(edges: Array<IntArray>): IntArray {  
        }  
    }  
}
```

Swift:

```
class Solution {  
    func findRedundantConnection(_ edges: [[Int]]) -> [Int] {  
        }  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn find_redundant_connection(edges: Vec<Vec<i32>>) -> Vec<i32> {  
        }  
    }  
}
```

Ruby:

```
# @param {Integer[][]} edges  
# @return {Integer[]}  
def find_redundant_connection(edges)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[][] $edges  
     * @return Integer[]  
     */
```

```
*/  
function findRedundantConnection($edges) {  
  
}  
}  
}
```

Dart:

```
class Solution {  
List<int> findRedundantConnection(List<List<int>> edges) {  
  
}  
}
```

Scala:

```
object Solution {  
def findRedundantConnection(edges: Array[Array[Int]]): Array[Int] = {  
  
}  
}
```

Elixir:

```
defmodule Solution do  
@spec find_redundant_connection(edges :: [[integer]]) :: [integer]  
def find_redundant_connection(edges) do  
  
end  
end
```

Erlang:

```
-spec find_redundant_connection(Edges :: [[integer()]]) -> [integer()].  
find_redundant_connection(Edges) ->  
.
```

Racket:

```
(define/contract (find-redundant-connection edges)  
  (-> (listof (listof exact-integer?)) (listof exact-integer?))  
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Redundant Connection
 * Difficulty: Medium
 * Tags: array, tree, graph, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

class Solution {
public:
vector<int> findRedundantConnection(vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
 * Problem: Redundant Connection
 * Difficulty: Medium
 * Tags: array, tree, graph, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

class Solution {
public int[] findRedundantConnection(int[][] edges) {

}
}
```

Python3 Solution:

```

"""
Problem: Redundant Connection
Difficulty: Medium
Tags: array, tree, graph, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

```

```

class Solution:

def findRedundantConnection(self, edges: List[List[int]]) -> List[int]:
    # TODO: Implement optimized solution
    pass

```

Python Solution:

```

class Solution(object):

def findRedundantConnection(self, edges):
    """
:type edges: List[List[int]]
:rtype: List[int]
"""

```

JavaScript Solution:

```

/**
 * Problem: Redundant Connection
 * Difficulty: Medium
 * Tags: array, tree, graph, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

var findRedundantConnection = function(edges) {

```

```
};
```

TypeScript Solution:

```
/**  
 * Problem: Redundant Connection  
 * Difficulty: Medium  
 * Tags: array, tree, graph, search  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(h) for recursion stack where h is height  
 */  
  
function findRedundantConnection(edges: number[][][]): number[] {  
  
};
```

C# Solution:

```
/*  
 * Problem: Redundant Connection  
 * Difficulty: Medium  
 * Tags: array, tree, graph, search  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(h) for recursion stack where h is height  
 */  
  
public class Solution {  
    public int[] FindRedundantConnection(int[][] edges) {  
  
    }  
}
```

C Solution:

```
/*  
 * Problem: Redundant Connection  
 * Difficulty: Medium
```

```

* Tags: array, tree, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

```

```

/***
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findRedundantConnection(int** edges, int edgesSize, int* edgesColSize,
int* returnSize) {

}

```

Go Solution:

```

// Problem: Redundant Connection
// Difficulty: Medium
// Tags: array, tree, graph, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func findRedundantConnection(edges [][]int) []int {
}

```

Kotlin Solution:

```

class Solution {
    fun findRedundantConnection(edges: Array<IntArray>): IntArray {
    }
}

```

Swift Solution:

```

class Solution {
    func findRedundantConnection(_ edges: [[Int]]) -> [Int] {
}

```

```
}
```

```
}
```

Rust Solution:

```
// Problem: Redundant Connection
// Difficulty: Medium
// Tags: array, tree, graph, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
    pub fn find_redundant_connection(edges: Vec<Vec<i32>>) -> Vec<i32> {
        ...
    }
}
```

Ruby Solution:

```
# @param {Integer[][]} edges
# @return {Integer[]}
def find_redundant_connection(edges)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[][] $edges
     * @return Integer[]
     */
    function findRedundantConnection($edges) {

    }
}
```

Dart Solution:

```
class Solution {  
    List<int> findRedundantConnection(List<List<int>> edges) {  
  
    }  
}
```

Scala Solution:

```
object Solution {  
    def findRedundantConnection(edges: Array[Array[Int]]): Array[Int] = {  
  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec find_redundant_connection([integer()]) :: [integer]  
  def find_redundant_connection(edges) do  
  
  end  
end
```

Erlang Solution:

```
-spec find_redundant_connection([integer()]) -> [integer()].  
find_redundant_connection(Edges) ->  
.
```

Racket Solution:

```
(define/contract (find-redundant-connection edges)  
  (-> (listof (listof exact-integer?)) (listof exact-integer?)))  
)
```