Problem 685: Redundant Connection II

Problem Information

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

Problem Description

In this problem, a rooted tree is a

directed

graph such that, there is exactly one node (the root) for which all other nodes are descendants of this node, plus every node has exactly one parent, except for the root node which has no parents.

The given input is a directed graph that started as a rooted tree with

n

nodes (with distinct values from

1

to

n

), with one additional directed edge added. The added edge has two different vertices chosen from

1

to

n

, and was not an edge that already existed.

The resulting graph is given as a 2D-array of

edges

. Each element of

edges

is a pair

[u

i

, v

i

]

that represents a

directed

edge connecting nodes

u

i

and

v

i

, where

u_i

is a parent of child

v_i

.

Return

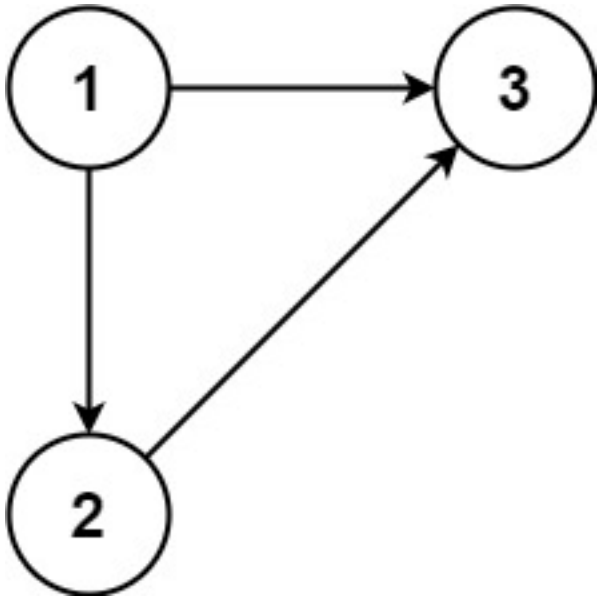
an edge that can be removed so that the resulting graph is a rooted tree of

n

nodes

. If there are multiple answers, return the answer that occurs last in the given 2D-array.

Example 1:



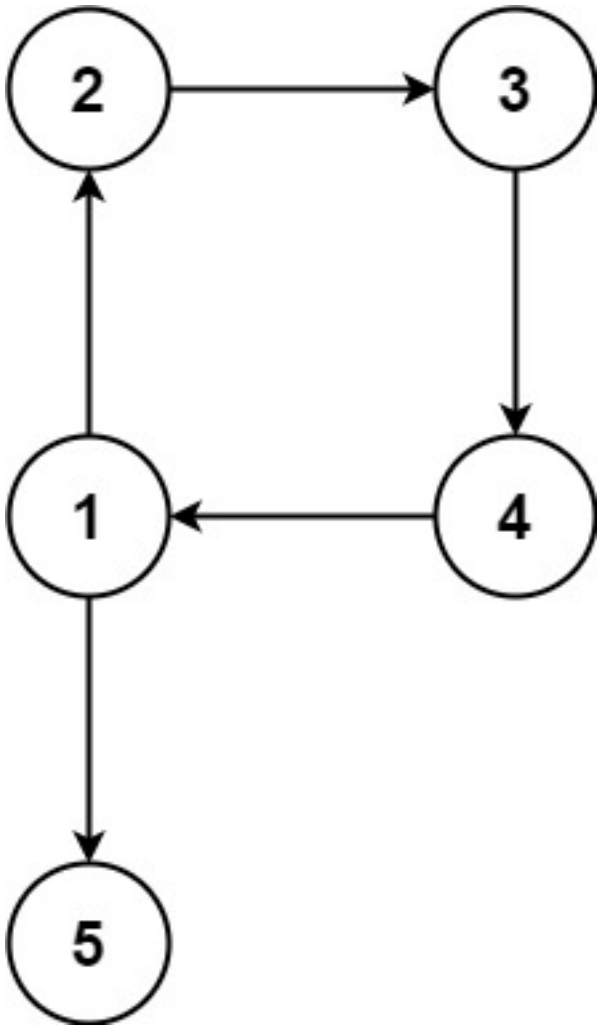
Input:

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

Output:

[2,3]

Example 2:



Input:

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

Output:

[4,1]

Constraints:

n == edges.length

3 <= n <= 1000

edges[i].length == 2

1 <= u

i

, v

i

<= n

u

i

!= v

i

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int[] findRedundantDirectedConnection(int[][] edges) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findRedundantDirectedConnection(edges: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] FindRedundantDirectedConnection(int[][] edges) {

}
}
```

C:

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

```
edgesColSize, int* returnSize) {

}
```

Go:

```
func findRedundantDirectedConnection(edges [][]int) []int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<int> findRedundantDirectedConnection(List<List<int>> edges) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Redundant Connection II
* Difficulty: Hard
* 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> findRedundantDirectedConnection(vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Redundant Connection II
* Difficulty: Hard
* 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[] findRedundantDirectedConnection(int[][] edges) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Redundant Connection II
Difficulty: Hard
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 findRedundantDirectedConnection(self, edges: List[List[int]]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Redundant Connection II
* Difficulty: Hard
* 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
```

```
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Redundant Connection II
* Difficulty: Hard
* 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 findRedundantDirectedConnection(edges: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Redundant Connection II
* Difficulty: Hard
* 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[] FindRedundantDirectedConnection(int[][] edges) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Redundant Connection II
* Difficulty: Hard
* 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* findRedundantDirectedConnection(int** edges, int edgesSize, int*
edgesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Redundant Connection II
// Difficulty: Hard
// 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 findRedundantDirectedConnection(edges [][]int) []int {

}
```

Kotlin Solution:

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

```
}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Redundant Connection II
// Difficulty: Hard
// 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_directed_connection(edges: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
```

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

}
}
```

Dart Solution:

```
class Solution {
List<int> findRedundantDirectedConnection(List<List<int>> edges) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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