Problem 2876: Count Visited Nodes in a Directed Graph

Problem Information

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

Problem Description

There is a

directed

graph consisting of

n

nodes numbered from

0

to

n - 1

and

n

directed edges.

You are given a

0-indexed

array

edges

where

edges[i]

indicates that there is an edge from node

i

to node

edges[i]

.

Consider the following process on the graph:

You start from a node

x

and keep visiting other nodes through edges until you reach a node that you have already visited before on this

same

process.

Return

an array

answer

where

answer[i]

is the number of

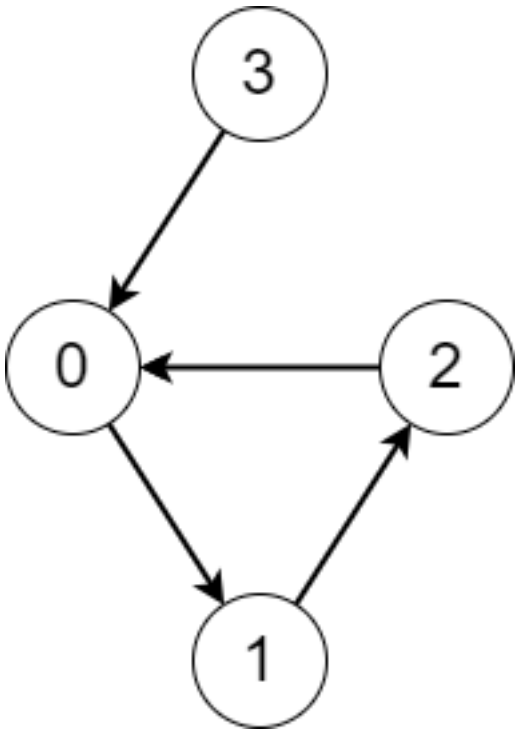
different

nodes that you will visit if you perform the process starting from node

i

.

Example 1:



Input:

edges = [1,2,0,0]

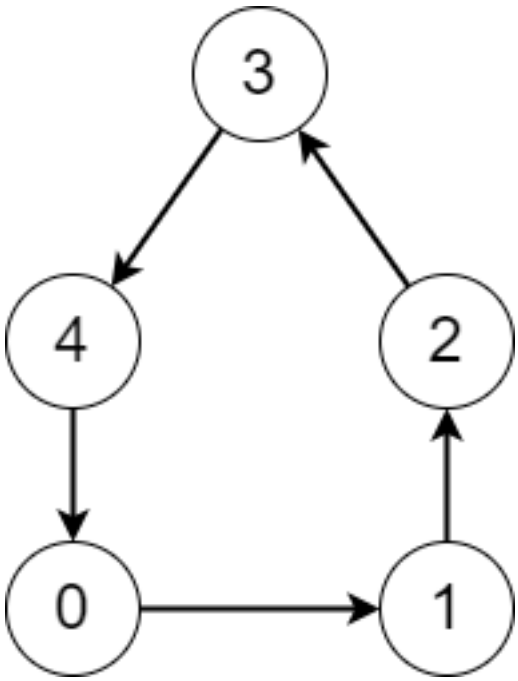
Output:

[3,3,3,4]

Explanation:

We perform the process starting from each node in the following way: - Starting from node 0, we visit the nodes 0 -> 1 -> 2 -> 0. The number of different nodes we visit is 3. - Starting from node 1, we visit the nodes 1 -> 2 -> 0 -> 1. The number of different nodes we visit is 3. - Starting from node 2, we visit the nodes 2 -> 0 -> 1 -> 2. The number of different nodes we visit is 3. - Starting from node 3, we visit the nodes 3 -> 0 -> 1 -> 2 -> 0. The number of different nodes we visit is 4.

Example 2:



Input:

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

Output:

[5,5,5,5,5]

Explanation:

Starting from any node we can visit every node in the graph in the process.

Constraints:

n == edges.length

2 <= n <= 10

5

0 <= edges[i] <= n - 1

edges[i] != i

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int[] countVisitedNodes(List<Integer> edges) {

}
}
```

Python3:

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

Python:

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

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int[] CountVisitedNodes(IList<int> edges) {

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun countVisitedNodes(edges: List<Int>): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
```

```
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (count-visited-nodes edges)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Visited Nodes in a Directed Graph
* Difficulty: Hard
* Tags: array, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Count Visited Nodes in a Directed Graph
* Difficulty: Hard
* Tags: array, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int[] countVisitedNodes(List<Integer> edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Visited Nodes in a Directed Graph
Difficulty: Hard
Tags: array, graph, dp
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Visited Nodes in a Directed Graph
* Difficulty: Hard
* Tags: array, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Count Visited Nodes in a Directed Graph
* Difficulty: Hard
* Tags: array, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function countVisitedNodes(edges: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Count Visited Nodes in a Directed Graph
* Difficulty: Hard
* Tags: array, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int[] CountVisitedNodes(IList<int> edges) {

}
}
```

C Solution:

```
/*
* Problem: Count Visited Nodes in a Directed Graph
* Difficulty: Hard
* Tags: array, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

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

}
```

Go Solution:

```
// Problem: Count Visited Nodes in a Directed Graph
// Difficulty: Hard
// Tags: array, graph, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func countVisitedNodes(edges []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun countVisitedNodes(edges: List<Int>): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Visited Nodes in a Directed Graph
// Difficulty: Hard
// Tags: array, graph, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-visited-nodes edges)
(-> (listof exact-integer?) (listof exact-integer?))
)
```