

Problem 2204: Distance to a Cycle in Undirected Graph

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a positive integer

n

representing the number of nodes in a

connected undirected graph

containing

exactly one

cycle. The nodes are numbered from

0

to

$n - 1$

(

inclusive

).

You are also given a 2D integer array

edges

, where

edges[i] = [node1

i

, node2

i

]

denotes that there is a

bidirectional

edge connecting

node1

i

and

node2

i

in the graph.

The distance between two nodes

a

and

b

is defined to be the

minimum

number of edges that are needed to go from

a

to

b

.

Return

an integer array

answer

of size

n

, where

answer[i]

is the

minimum

distance between the

i

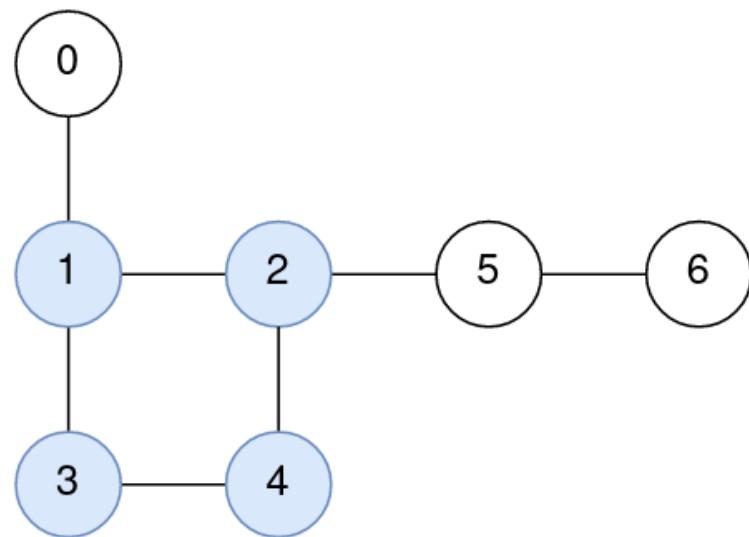
th

node and

any

node in the cycle.

Example 1:



Input:

$n = 7$, edges = [[1,2],[2,4],[4,3],[3,1],[0,1],[5,2],[6,5]]

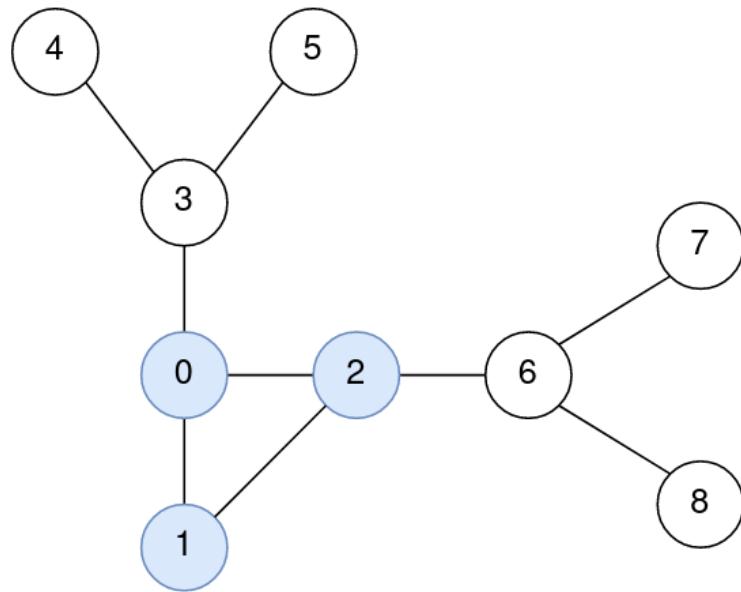
Output:

[1,0,0,0,0,1,2]

Explanation:

The nodes 1, 2, 3, and 4 form the cycle. The distance from 0 to 1 is 1. The distance from 1 to 1 is 0. The distance from 2 to 2 is 0. The distance from 3 to 3 is 0. The distance from 4 to 4 is 0. The distance from 5 to 2 is 1. The distance from 6 to 2 is 2.

Example 2:



Input:

$n = 9$, edges = [[0,1],[1,2],[0,2],[2,6],[6,7],[6,8],[0,3],[3,4],[3,5]]

Output:

[0,0,0,1,2,2,1,2,2]

Explanation:

The nodes 0, 1, and 2 form the cycle. The distance from 0 to 0 is 0. The distance from 1 to 1 is 0. The distance from 2 to 2 is 0. The distance from 3 to 1 is 1. The distance from 4 to 1 is 2. The distance from 5 to 1 is 2. The distance from 6 to 2 is 1. The distance from 7 to 2 is 2. The distance from 8 to 2 is 2.

Constraints:

$3 \leq n \leq 10$

edges.length == n

edges[i].length == 2

0 <= node1

i

, node2

i

<= n - 1

node1

i

!= node2

i

The graph is connected.

The graph has exactly one cycle.

There is at most one edge between any pair of vertices.

Code Snippets

C++:

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

Java:

```
class Solution {  
    public int[] distanceToCycle(int n, int[][] edges) {  
  
    }  
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function distanceToCycle(n: number, edges: number[][]): number[] {  
  
};
```

C#:

```
public class Solution {  
    public int[] DistanceToCycle(int n, int[][] edges) {  
  
    }  
}
```

C:

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

Go:

```
func distanceToCycle(n int, edges [][]int) []int {  
  
}
```

Kotlin:

```
class Solution {  
    fun distanceToCycle(n: Int, edges: Array<IntArray>): IntArray {  
  
    }  
}
```

Swift:

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

Rust:

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

```
}
```

```
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][][]} edges
# @return {Integer[]}
def distance_to_cycle(n, edges)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer $n
     * @param Integer[][] $edges
     * @return Integer[]
     */
    function distanceToCycle($n, $edges) {

    }
}
```

Dart:

```
class Solution {
List<int> distanceToCycle(int n, List<List<int>> edges) {

}
```

Scala:

```
object Solution {
def distanceToCycle(n: Int, edges: Array[Array[Int]]): Array[Int] = {

}
```

Elixir:

```
defmodule Solution do
  @spec distance_to_cycle(n :: integer, edges :: [[integer]]) :: [integer]
  def distance_to_cycle(n, edges) do

  end
end
```

Erlang:

```
-spec distance_to_cycle(N :: integer(), Edges :: [[integer()]]) ->
[inode()].
distance_to_cycle(N, Edges) ->
.
```

Racket:

```
(define/contract (distance-to-cycle n edges)
  (-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Distance to a Cycle in Undirected Graph
 * Difficulty: Hard
 * Tags: array, graph, search
 *
 * 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:
vector<int> distanceToCycle(int n, vector<vector<int>>& edges) {

}
```

Java Solution:

```
/**  
 * Problem: Distance to a Cycle in Undirected Graph  
 * Difficulty: Hard  
 * Tags: array, graph, search  
 *  
 * 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[] distanceToCycle(int n, int[][] edges) {  
        }  
    }  
}
```

Python3 Solution:

```
"""  
Problem: Distance to a Cycle in Undirected Graph  
Difficulty: Hard  
Tags: array, graph, search  
  
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 distanceToCycle(self, n: int, edges: List[List[int]]) -> List[int]:  
        # TODO: Implement optimized solution  
        pass
```

Python Solution:

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

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

```

JavaScript Solution:

```
/**
 * Problem: Distance to a Cycle in Undirected Graph
 * Difficulty: Hard
 * Tags: array, graph, search
 *
 * 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[][]} edges
 * @return {number[]}
 */
var distanceToCycle = function(n, edges) {

};


```

TypeScript Solution:

```
/**
 * Problem: Distance to a Cycle in Undirected Graph
 * Difficulty: Hard
 * Tags: array, graph, search
 *
 * 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 distanceToCycle(n: number, edges: number[][]): number[] {

};


```

C# Solution:

```

/*
 * Problem: Distance to a Cycle in Undirected Graph
 * Difficulty: Hard
 * Tags: array, graph, search
 *
 * 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[] DistanceToCycle(int n, int[][] edges) {
        return new int[n];
    }
}

```

C Solution:

```

/*
 * Problem: Distance to a Cycle in Undirected Graph
 * Difficulty: Hard
 * Tags: array, graph, search
 *
 * 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
 */

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

}

```

Go Solution:

```

// Problem: Distance to a Cycle in Undirected Graph
// Difficulty: Hard
// Tags: array, graph, search
//

```

```

// 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 distanceToCycle(n int, edges [][]int) []int {
}

```

Kotlin Solution:

```

class Solution {
    fun distanceToCycle(n: Int, edges: Array<IntArray>): IntArray {
        return IntArray(n)
    }
}

```

Swift Solution:

```

class Solution {
    func distanceToCycle(_ n: Int, _ edges: [[Int]]) -> [Int] {
        return []
    }
}

```

Rust Solution:

```

// Problem: Distance to a Cycle in Undirected Graph
// Difficulty: Hard
// Tags: array, graph, search
//
// 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 distance_to_cycle(n: i32, edges: Vec<Vec<i32>>) -> Vec<i32> {
        let mut distances = vec![n + 1; n];
        let mut visited = vec![false; n];
        let mut stack = Vec::new();
        let mut current = 0;
        let mut index = 0;

        while !stack.contains(&current) {
            if visited[current] {
                break;
            }

            visited[current] = true;
            stack.push(current);

            for &neighbor in edges[current].iter() {
                if !visited[neighbor] {
                    distances[neighbor] = distances[current] + 1;
                    index = neighbor;
                }
            }

            current = index;
        }

        distances
    }
}

```

Ruby Solution:

```

# @param {Integer} n
# @param {Integer[][]} edges
# @return {Integer[]}
def distance_to_cycle(n, edges)

end

```

PHP Solution:

```

class Solution {

    /**
     * @param Integer $n
     * @param Integer[][] $edges
     * @return Integer[]
     */
    function distanceToCycle($n, $edges) {

    }
}

```

Dart Solution:

```

class Solution {
List<int> distanceToCycle(int n, List<List<int>> edges) {
}
}

```

Scala Solution:

```

object Solution {
def distanceToCycle(n: Int, edges: Array[Array[Int]]): Array[Int] = {
}
}

```

Elixir Solution:

```

defmodule Solution do
@spec distance_to_cycle(n :: integer, edges :: [[integer]]) :: [integer]
def distance_to_cycle(n, edges) do

```

```
end  
end
```

Erlang Solution:

```
-spec distance_to_cycle(N :: integer(), Edges :: [[integer()]]) ->  
[integer()].  
distance_to_cycle(N, Edges) ->  
. . .
```

Racket Solution:

```
(define/contract (distance-to-cycle n edges)  
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))  
)
```