Problem 2608: Shortest Cycle in a Graph

Problem Information

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

Problem Description

There is a

bi-directional

graph with

n

vertices, where each vertex is labeled from

0

to

n - 1

. The edges in the graph are represented by a given 2D integer array

edges

, where

edges[i] = [u

i

, v

i

]

denotes an edge between vertex

u

i

and vertex

v

i

. Every vertex pair is connected by at most one edge, and no vertex has an edge to itself.

Return

the length of the

shortest

cycle in the graph

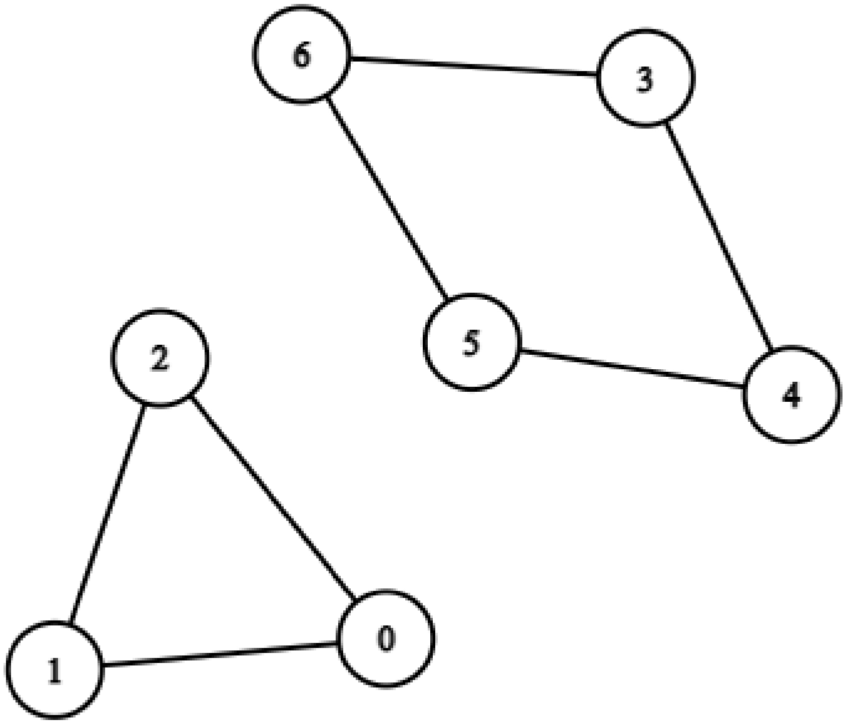
. If no cycle exists, return

-1

.

A cycle is a path that starts and ends at the same node, and each edge in the path is used only once.

Example 1:



Input:

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

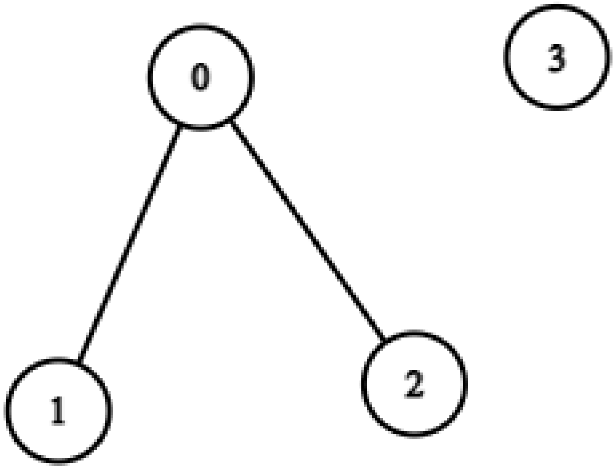
Output:

3

Explanation:

The cycle with the smallest length is : 0 -> 1 -> 2 -> 0

Example 2:



Input:

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

Output:

-1

Explanation:

There are no cycles in this graph.

Constraints:

2 <= n <= 1000

1 <= edges.length <= 1000

edges[i].length == 2

0 <= u

i

, v

i

< n

u

i

!= v

i

There are no repeated edges.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findShortestCycle(n: number, edges: number[][]): number {

};
```

C#:

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

}
}
```

C:

```
int findShortestCycle(int n, int** edges, int edgesSize, int* edgesColSize) {

}
```

Go:

```
func findShortestCycle(n int, edges [][]int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-shortest-cycle n edges)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Cycle in a 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 findShortestCycle(int n, vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Cycle in a 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 findShortestCycle(int n, int[][] edges) {
}
}
```

Python3 Solution:

```
"""
Problem: Shortest Cycle in a 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 findShortestCycle(self, n: int, edges: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shortest Cycle in a 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 findShortestCycle = function(n, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Cycle in a 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 findShortestCycle(n: number, edges: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Shortest Cycle in a 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 FindShortestCycle(int n, int[][] edges) {
}
}
```

C Solution:

```
/*
* Problem: Shortest Cycle in a 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
*/

int findShortestCycle(int n, int** edges, int edgesSize, int* edgesColSize) {

}
```

Go Solution:

```
// Problem: Shortest Cycle in a 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 findShortestCycle(n int, edges [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Shortest Cycle in a 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 find_shortest_cycle(n: i32, edges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-shortest-cycle n edges)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)

)
```