Problem 847: Shortest Path Visiting All Nodes

Problem Information

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

Problem Description

You have an undirected, connected graph of

n

nodes labeled from

0

to

n - 1

. You are given an array

graph

where

graph[i]

is a list of all the nodes connected with node

i

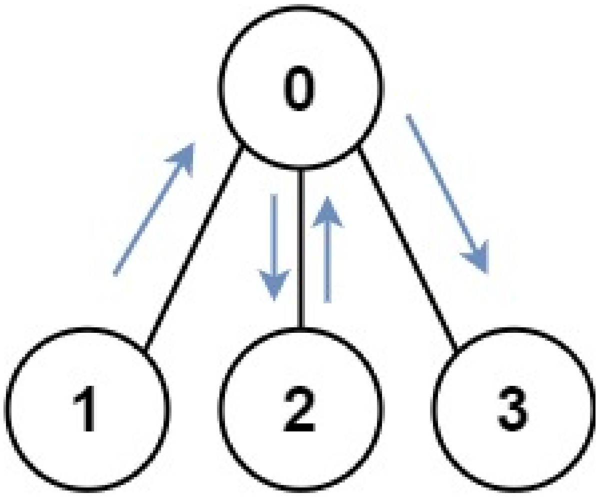
by an edge.

Return

the length of the shortest path that visits every node

. You may start and stop at any node, you may revisit nodes multiple times, and you may reuse edges.

Example 1:



Input:

graph = [[1,2,3],[0],[0],[0]]

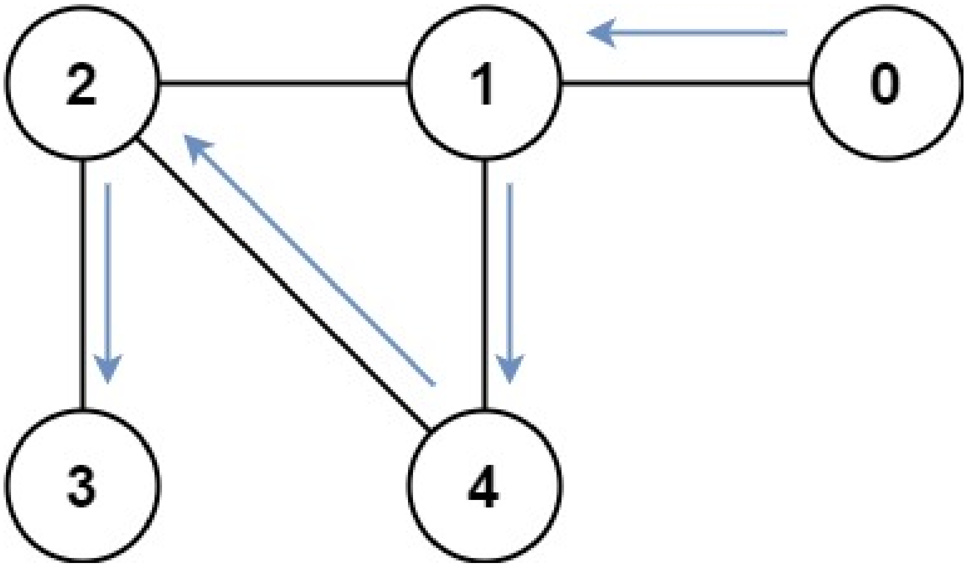
Output:

4

Explanation:

One possible path is [1,0,2,0,3]

Example 2:



Input:

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

Output:

4

Explanation:

One possible path is [0,1,4,2,3]

Constraints:

n == graph.length

1 <= n <= 12

0 <= graph[i].length < n

graph[i]

does not contain

i

.

If

graph[a]

contains

b

, then

graph[b]

contains

a

.

The input graph is always connected.

Code Snippets

C++:

```
class Solution {
public:
int shortestPathLength(vector<vector<int>>& graph) {

}
};
```

Java:

```
class Solution {
public int shortestPathLength(int[][] graph) {

}


}
```

Python3:

```
class Solution:
def shortestPathLength(self, graph: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function shortestPathLength(graph: number[][]): number {

};
```

C#:

```
public class Solution {
public int ShortestPathLength(int[][] graph) {

}
}
```

C:

```
int shortestPathLength(int** graph, int graphSize, int* graphColSize) {

}
```

Go:

```
func shortestPathLength(graph [][]int) int {

}
```

Kotlin:

```
class Solution {
fun shortestPathLength(graph: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func shortestPathLength(_ graph: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn shortest_path_length(graph: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} graph
# @return {Integer}
def shortest_path_length(graph)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $graph
* @return Integer
*/
function shortestPathLength($graph) {

}
}
```

Dart:

```
class Solution {
int shortestPathLength(List<List<int>> graph) {

}
}
```

Scala:

```
object Solution {
def shortestPathLength(graph: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec shortest_path_length(graph :: [[integer]]) :: integer
def shortest_path_length(graph) do

end
end
```

Erlang:

```
-spec shortest_path_length(Graph :: [[integer()]]) -> integer().
shortest_path_length(Graph) ->
.
```

Racket:

```
(define/contract (shortest-path-length graph)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Path Visiting All Nodes
* Difficulty: Hard
* Tags: array, graph, dp, search
*
* 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 shortestPathLength(vector<vector<int>>& graph) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Path Visiting All Nodes
* Difficulty: Hard
* Tags: array, graph, dp, search
*
* 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 shortestPathLength(int[][] graph) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest Path Visiting All Nodes
Difficulty: Hard
Tags: array, graph, dp, search

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 shortestPathLength(self, graph: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shortest Path Visiting All Nodes
* Difficulty: Hard
* Tags: array, graph, dp, search
*
* 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[][]} graph
* @return {number}
*/
var shortestPathLength = function(graph) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Path Visiting All Nodes
* Difficulty: Hard
* Tags: array, graph, dp, search
*
* 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 shortestPathLength(graph: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Shortest Path Visiting All Nodes
* Difficulty: Hard
* Tags: array, graph, dp, search
*
* 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 ShortestPathLength(int[][] graph) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Path Visiting All Nodes
* Difficulty: Hard
* Tags: array, graph, dp, search
*
* 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
*/

int shortestPathLength(int** graph, int graphSize, int* graphColSize) {

}
```

Go Solution:

```
// Problem: Shortest Path Visiting All Nodes
// Difficulty: Hard
// Tags: array, graph, dp, search
//
// 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 shortestPathLength(graph [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun shortestPathLength(graph: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func shortestPathLength(_ graph: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Shortest Path Visiting All Nodes
// Difficulty: Hard
// Tags: array, graph, dp, search
//
// 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 shortest_path_length(graph: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} graph
# @return {Integer}
def shortest_path_length(graph)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $graph
* @return Integer
*/
function shortestPathLength($graph) {

}
}
```

Dart Solution:

```
class Solution {
int shortestPathLength(List<List<int>> graph) {

}
}
```

Scala Solution:

```
object Solution {
def shortestPathLength(graph: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec shortest_path_length(graph :: [[integer]]) :: integer
def shortest_path_length(graph) do

end
end
```

Erlang Solution:

```
-spec shortest_path_length(Graph :: [[integer()]]) -> integer().
shortest_path_length(Graph) ->
.
```

Racket Solution:

```
(define/contract (shortest-path-length graph)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```