Problem 928: Minimize Malware Spread II

Problem Information

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

Problem Description

You are given a network of

n

nodes represented as an

n x n

adjacency matrix

graph

, where the

i

th

node is directly connected to the

j

th

node if

graph[i][j] == 1

.

Some nodes

initial

are initially infected by malware. Whenever two nodes are directly connected, and at least one of those two nodes is infected by malware, both nodes will be infected by malware. This spread of malware will continue until no more nodes can be infected in this manner.

Suppose

M(initial)

is the final number of nodes infected with malware in the entire network after the spread of malware stops.

We will remove

exactly one node

from

initial

,

completely removing it and any connections from this node to any other node

.

Return the node that, if removed, would minimize

M(initial)

. If multiple nodes could be removed to minimize

M(initial)

, return such a node with

the smallest index

.

Example 1:

Input:

graph = [[1,1,0],[1,1,0],[0,0,1]], initial = [0,1]

Output:

0

Example 2:

Input:

graph = [[1,1,0],[1,1,1],[0,1,1]], initial = [0,1]

Output:

1

Example 3:

Input:

graph = [[1,1,0,0],[1,1,1,0],[0,1,1,1],[0,0,1,1]], initial = [0,1]

Output:

1

Constraints:

n == graph.length

n == graph[i].length

2 <= n <= 300

graph[i][j]

is

0

or

1

.

graph[i][j] == graph[j][i]

graph[i][i] == 1

1 <= initial.length < n

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

All the integers in

initial

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int minMalwareSpread(vector<vector<int>>& graph, vector<int>& initial) {

}
};
```

Java:

```
class Solution {
public int minMalwareSpread(int[][] graph, int[] initial) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minMalwareSpread(graph: number[][], initial: number[]): number {

};
```

C#:

```
public class Solution {
public int MinMalwareSpread(int[][] graph, int[] initial) {

}
}
```

C:

```
int minMalwareSpread(int** graph, int graphSize, int* graphColSize, int*
initial, int initialSize) {

}
```

Go:

```
func minMalwareSpread(graph [][]int, initial []int) int {

}
```

Kotlin:

```
class Solution {
fun minMalwareSpread(graph: Array<IntArray>, initial: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minMalwareSpread(_ graph: [[Int]], _ initial: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_malware_spread(graph: Vec<Vec<i32>>, initial: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} graph
# @param {Integer[]} initial
# @return {Integer}
def min_malware_spread(graph, initial)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $graph
* @param Integer[] $initial
* @return Integer
*/
function minMalwareSpread($graph, $initial) {

}
}
```

Dart:

```
class Solution {
int minMalwareSpread(List<List<int>> graph, List<int> initial) {

}
}
```

Scala:

```
object Solution {
def minMalwareSpread(graph: Array[Array[Int]], initial: Array[Int]): Int = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec min_malware_spread(graph :: [[integer]], initial :: [integer]) ::
integer
def min_malware_spread(graph, initial) do

end
end
```

Erlang:

```
-spec min_malware_spread(Graph :: [[integer()]], Initial :: [integer()]) ->
integer().
min_malware_spread(Graph, Initial) ->
.
```

Racket:

```
(define/contract (min-malware-spread graph initial)
(-> (listof (listof exact-integer?)) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimize Malware Spread II
* Difficulty: Hard
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
class Solution {
public:
int minMalwareSpread(vector<vector<int>>& graph, vector<int>& initial) {

}
};
```

Java Solution:

```
/**
* Problem: Minimize Malware Spread II
* Difficulty: Hard
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int minMalwareSpread(int[][] graph, int[] initial) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimize Malware Spread II
Difficulty: Hard
Tags: array, graph, hash, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

```
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimize Malware Spread II
* Difficulty: Hard
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[][]} graph
* @param {number[]} initial
* @return {number}
*/
var minMalwareSpread = function(graph, initial) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimize Malware Spread II
* Difficulty: Hard
* Tags: array, graph, hash, search
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function minMalwareSpread(graph: number[][], initial: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimize Malware Spread II
* Difficulty: Hard
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int MinMalwareSpread(int[][] graph, int[] initial) {

}
}
```

C Solution:

```
/*
* Problem: Minimize Malware Spread II
* Difficulty: Hard
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int minMalwareSpread(int** graph, int graphSize, int* graphColSize, int*
initial, int initialSize) {
```

```
}
```

Go Solution:

```
// Problem: Minimize Malware Spread II
// Difficulty: Hard
// Tags: array, graph, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func minMalwareSpread(graph [][]int, initial []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minMalwareSpread(graph: Array<IntArray>, initial: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minMalwareSpread(_ graph: [[Int]], _ initial: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimize Malware Spread II
// Difficulty: Hard
// Tags: array, graph, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
```

```
// Space Complexity: O(n) for hash map

impl Solution {
pub fn min_malware_spread(graph: Vec<Vec<i32>>, initial: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} graph
# @param {Integer[]} initial
# @return {Integer}
def min_malware_spread(graph, initial)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $graph
* @param Integer[] $initial
* @return Integer
*/
function minMalwareSpread($graph, $initial) {

}
}
```

Dart Solution:

```
class Solution {
int minMalwareSpread(List<List<int>> graph, List<int> initial) {

}
}
```

Scala Solution:

```
object Solution {
def minMalwareSpread(graph: Array[Array[Int]], initial: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_malware_spread(graph :: [[integer]], initial :: [integer]) ::
integer
def min_malware_spread(graph, initial) do

end
end
```

Erlang Solution:

```
-spec min_malware_spread(Graph :: [[integer()]], Initial :: [integer()]) ->
integer().
min_malware_spread(Graph, Initial) ->
.
```

Racket Solution:

```
(define/contract (min-malware-spread graph initial)
(-> (listof (listof exact-integer?)) (listof exact-integer?) exact-integer?)
)
```