Problem 2039: The Time When the Network Becomes Idle

Problem Information

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

Problem Description

There is a network of

n

servers, labeled from

0

to

n - 1

. You are given a 2D integer array

edges

, where

edges[i] = [u

i

, v

i

]

indicates there is a message channel between servers

u

i

and

v

i

, and they can pass

any

number of messages to

each other

directly in

one

second. You are also given a

0-indexed

integer array

patience

of length

n

.

All servers are

connected

, i.e., a message can be passed from one server to any other server(s) directly or indirectly through the message channels.

The server labeled

0

is the

master

server. The rest are

data

servers. Each data server needs to send its message to the master server for processing and wait for a reply. Messages move between servers

optimally

, so every message takes the

least amount of time

to arrive at the master server. The master server will process all newly arrived messages

instantly

and send a reply to the originating server via the

reversed path

the message had gone through.

At the beginning of second

0

, each data server sends its message to be processed. Starting from second

1

, at the

beginning

of

every

second, each data server will check if it has received a reply to the message it sent (including any newly arrived replies) from the master server:

If it has not, it will

resend

the message periodically. The data server

i

will resend the message every

patience[i]

second(s), i.e., the data server

i

will resend the message if

patience[i]

second(s) have

elapsed

since the

last

time the message was sent from this server.

Otherwise,

no more resending

will occur from this server.

The network becomes

idle

when there are

no

messages passing between servers or arriving at servers.

Return

the

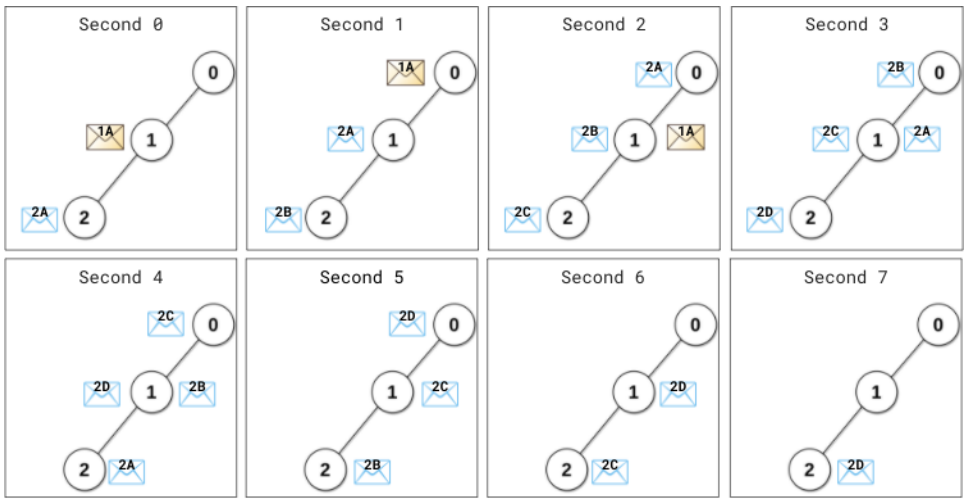
earliest second

starting from which the network becomes

idle

.

Example 1:



Input:

edges = [[0,1],[1,2]], patience = [0,2,1]

Output:

8

Explanation:

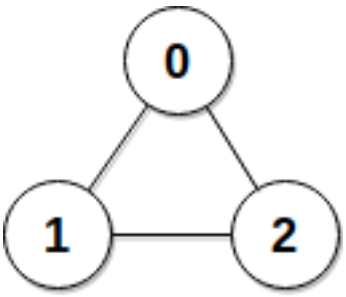
At (the beginning of) second 0, - Data server 1 sends its message (denoted 1A) to the master server. - Data server 2 sends its message (denoted 2A) to the master server.

At second 1, - Message 1A arrives at the master server. Master server processes message 1A instantly and sends a reply 1A back. - Server 1 has not received any reply. 1 second (1 < patience[1] = 2) elapsed since this server has sent the message, therefore it does not resend the message. - Server 2 has not received any reply. 1 second (1 == patience[2] = 1) elapsed since this server has sent the message, therefore it resends the message (denoted 2B).

At second 2, - The reply 1A arrives at server 1. No more resending will occur from server 1. - Message 2A arrives at the master server. Master server processes message 2A instantly and sends a reply 2A back. - Server 2 resends the message (denoted 2C). ... At second 4, - The reply 2A arrives at server 2. No more resending will occur from server 2. ... At second 7, reply 2D arrives at server 2.

Starting from the beginning of the second 8, there are no messages passing between servers or arriving at servers. This is the time when the network becomes idle.

Example 2:



Input:

edges = [[0,1],[0,2],[1,2]], patience = [0,10,10]

Output:

3

Explanation:

Data servers 1 and 2 receive a reply back at the beginning of second 2. From the beginning of the second 3, the network becomes idle.

Constraints:

n == patience.length

2 <= n <= 10

5

patience[0] == 0

1 <= patience[i] <= 10

5

for

1 <= i < n

1 <= edges.length <= min(10

5

, n * (n - 1) / 2)

edges[i].length == 2

0 <= u

i

, v

i

< n

u

i

!= v

i

There are no duplicate edges.

Each server can directly or indirectly reach another server.

Code Snippets

C++:

```
class Solution {
public:
int networkBecomesIdle(vector<vector<int>>& edges, vector<int>& patience) {

}
};
```

Java:

```
class Solution {
public int networkBecomesIdle(int[][] edges, int[] patience) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function networkBecomesIdle(edges: number[][], patience: number[]): number {

};
```

C#:

```
public class Solution {
public int NetworkBecomesIdle(int[][] edges, int[] patience) {

}
}
```

C:

```
int networkBecomesIdle(int** edges, int edgesSize, int* edgesColSize, int*
patience, int patienceSize) {

}
```

Go:

```
func networkBecomesIdle(edges [][]int, patience []int) int {

}
```

Kotlin:

```
class Solution {
fun networkBecomesIdle(edges: Array<IntArray>, patience: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[][]} edges
# @param {Integer[]} patience
# @return {Integer}
def network_becomes_idle(edges, patience)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer[] $patience
* @return Integer
*/
function networkBecomesIdle($edges, $patience) {

}
}
```

Dart:

```
class Solution {
int networkBecomesIdle(List<List<int>> edges, List<int> patience) {

}
}
```

Scala:

```
object Solution {
def networkBecomesIdle(edges: Array[Array[Int]], patience: Array[Int]): Int =
{
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec network_becomes_idle(edges :: [[integer]], patience :: [integer]) ::
integer
def network_becomes_idle(edges, patience) do

end
end
```

Erlang:

```
-spec network_becomes_idle(Edges :: [[integer()]], Patience :: [integer()])
-> integer().
network_becomes_idle(Edges, Patience) ->
.
```

Racket:

```
(define/contract (network-becomes-idle edges patience)
(-> (listof (listof exact-integer?)) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: The Time When the Network Becomes Idle
* Difficulty: Medium
* 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 networkBecomesIdle(vector<vector<int>>& edges, vector<int>& patience) {

}
};
```

Java Solution:

```
/**
* Problem: The Time When the Network Becomes Idle
* Difficulty: Medium
* 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 networkBecomesIdle(int[][] edges, int[] patience) {

}
}
```

Python3 Solution:

```
"""
Problem: The Time When the Network Becomes Idle
Difficulty: Medium
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 networkBecomesIdle(self, edges: List[List[int]], patience: List[int]) ->
int:
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: The Time When the Network Becomes Idle
* Difficulty: Medium
* 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[][]} edges
* @param {number[]} patience
* @return {number}
*/
var networkBecomesIdle = function(edges, patience) {

};
```

TypeScript Solution:

```
/**
* Problem: The Time When the Network Becomes Idle
* Difficulty: Medium
* 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 networkBecomesIdle(edges: number[][], patience: number[]): number {

};
```

C# Solution:

```
/*
* Problem: The Time When the Network Becomes Idle
* Difficulty: Medium
* 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 NetworkBecomesIdle(int[][] edges, int[] patience) {

}
}
```

C Solution:

```
/*
* Problem: The Time When the Network Becomes Idle
* Difficulty: Medium
* 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 networkBecomesIdle(int** edges, int edgesSize, int* edgesColSize, int*
patience, int patienceSize) {
```

```
}
```

Go Solution:

```
// Problem: The Time When the Network Becomes Idle
// Difficulty: Medium
// 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 networkBecomesIdle(edges [][]int, patience []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun networkBecomesIdle(edges: Array<IntArray>, patience: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: The Time When the Network Becomes Idle
// Difficulty: Medium
// 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 network_becomes_idle(edges: Vec<Vec<i32>>, patience: Vec<i32>) -> i32
{

}
}
```

Ruby Solution:

```
# @param {Integer[][]} edges
# @param {Integer[]} patience
# @return {Integer}
def network_becomes_idle(edges, patience)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer[] $patience
* @return Integer
*/
function networkBecomesIdle($edges, $patience) {

}
}
```

Dart Solution:

```
class Solution {
int networkBecomesIdle(List<List<int>> edges, List<int> patience) {

}
}
```

Scala Solution:

```
object Solution {
def networkBecomesIdle(edges: Array[Array[Int]], patience: Array[Int]): Int =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec network_becomes_idle(edges :: [[integer]], patience :: [integer]) ::
integer
def network_becomes_idle(edges, patience) do

end
end
```

Erlang Solution:

```
-spec network_becomes_idle(Edges :: [[integer()]], Patience :: [integer()])
-> integer().
network_becomes_idle(Edges, Patience) ->
.
```

Racket Solution:

```
(define/contract (network-becomes-idle edges patience)
(-> (listof (listof exact-integer?)) (listof exact-integer?) exact-integer?)
)
```