Problem 3112: Minimum Time to Visit Disappearing Nodes

Problem Information

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

Problem Description

There is an undirected graph of

n

nodes. You are given a 2D array

edges

, where

edges[i] = [u

i

, v

i

, length

i

]

describes an edge between node

u

i

and node

v

i

with a traversal time of

length

i

units.

Additionally, you are given an array

disappear

, where

disappear[i]

denotes the time when the node

i

disappears from the graph and you won't be able to visit it.

Note

that the graph might be

disconnected

and might contain

multiple edges

.

Return the array

answer

, with

answer[i]

denoting the

minimum

units of time required to reach node

i

from node 0. If node

i

is

unreachable

from node 0 then

answer[i]

is

-1

.

Example 1:

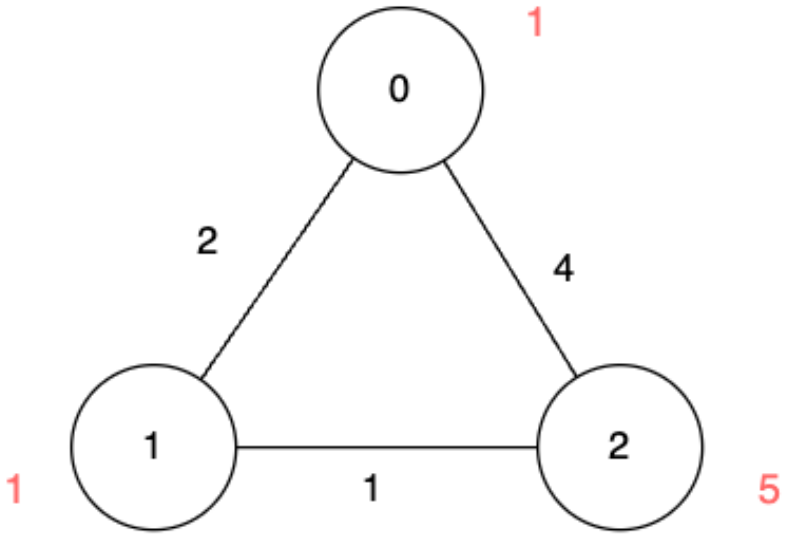
Input:

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

Output:

[0,-1,4]

Explanation:



We are starting our journey from node 0, and our goal is to find the minimum time required to reach each node before it disappears.

For node 0, we don't need any time as it is our starting point.

For node 1, we need at least 2 units of time to traverse

edges[0]

. Unfortunately, it disappears at that moment, so we won't be able to visit it.

For node 2, we need at least 4 units of time to traverse

edges[2]

.

Example 2:

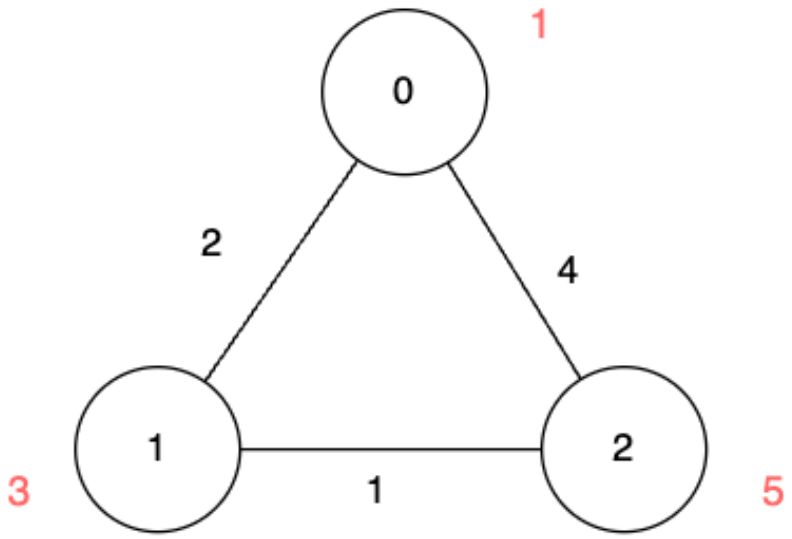
Input:

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

Output:

[0,2,3]

Explanation:



We are starting our journey from node 0, and our goal is to find the minimum time required to reach each node before it disappears.

For node 0, we don't need any time as it is the starting point.

For node 1, we need at least 2 units of time to traverse

edges[0]

.

For node 2, we need at least 3 units of time to traverse

edges[0]

and

edges[1]

.

Example 3:

Input:

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

Output:

[0,-1]

Explanation:

Exactly when we reach node 1, it disappears.

Constraints:

1 <= n <= 5 * 10

4

0 <= edges.length <= 10

5

edges[i] == [u

i

, v

i

, length

i

]

0 <= u

i

, v

i

<= n - 1

1 <= length

i

<= 10

5

disappear.length == n

1 <= disappear[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

```
func minimumTime(n int, edges [][]int, disappear []int) []int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (minimum-time n edges disappear)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
(listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time to Visit Disappearing Nodes
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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> minimumTime(int n, vector<vector<int>>& edges, vector<int>&
disappear) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Time to Visit Disappearing Nodes
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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[] minimumTime(int n, int[][] edges, int[] disappear) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Time to Visit Disappearing Nodes
Difficulty: Medium
```

```
Tags: array, graph, queue, heap

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 minimumTime(self, n: int, edges: List[List[int]], disappear: List[int])
-> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Time to Visit Disappearing Nodes
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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
* @param {number[]} disappear
* @return {number[]}
```

```
*/
var minimumTime = function(n, edges, disappear) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time to Visit Disappearing Nodes
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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 minimumTime(n: number, edges: number[][], disappear: number[]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Minimum Time to Visit Disappearing Nodes
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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[] MinimumTime(int n, int[][] edges, int[] disappear) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Time to Visit Disappearing Nodes
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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* minimumTime(int n, int** edges, int edgesSize, int* edgesColSize, int*
disappear, int disappearSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Minimum Time to Visit Disappearing Nodes
// Difficulty: Medium
// Tags: array, graph, queue, heap
//
// 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 minimumTime(n int, edges [][]int, disappear []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Time to Visit Disappearing Nodes
// Difficulty: Medium
// Tags: array, graph, queue, heap
//
// 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 minimum_time(n: i32, edges: Vec<Vec<i32>>, disappear: Vec<i32>) ->
Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer[] $disappear
* @return Integer[]
```

```
*/
function minimumTime($n, $edges, $disappear) {

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimum-time n edges disappear)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
(listof exact-integer?))
)
```