Problem 743: Network Delay Time

Problem Information

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

Problem Description

You are given a network of

n

nodes, labeled from

1

to

n

. You are also given

times

, a list of travel times as directed edges

times[i] = (u

i

, v

i

, w

i

)

, where

u

i

is the source node,

v

i

is the target node, and

w

i

is the time it takes for a signal to travel from source to target.

We will send a signal from a given node

k

. Return

the

minimum

time it takes for all the

n

nodes to receive the signal

. If it is impossible for all the

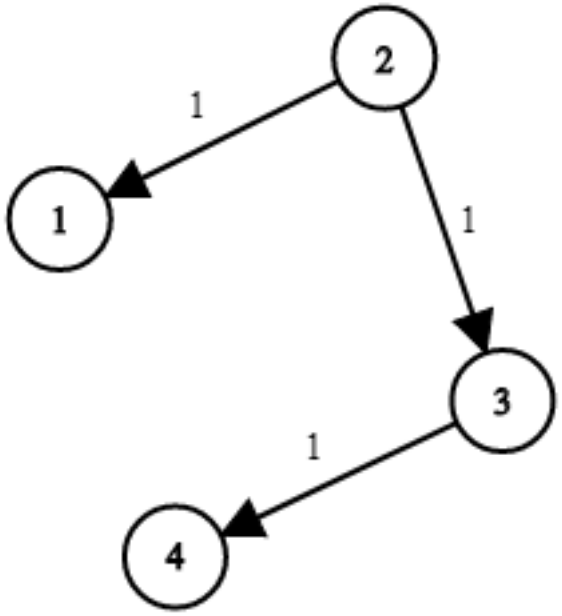
n

nodes to receive the signal, return

-1

.

Example 1:



Input:

times = [[2,1,1],[2,3,1],[3,4,1]], n = 4, k = 2

Output:

2

Example 2:

Input:

times = [[1,2,1]], n = 2, k = 1

Output:

1

Example 3:

Input:

times = [[1,2,1]], n = 2, k = 2

Output:

-1

Constraints:

1 <= k <= n <= 100

1 <= times.length <= 6000

times[i].length == 3

1 <= u

i

, v

i

<= n

u

i

!= v

i

0 <= w

i

<= 100

All the pairs

(u

i

, v

i

)

are

unique

. (i.e., no multiple edges.)

Code Snippets

C++:

```
class Solution {
public:
int networkDelayTime(vector<vector<int>>& times, int n, int k) {

}
};
```

Java:

```
class Solution {
public int networkDelayTime(int[][] times, int n, int k) {

}
}
```

Python3:

```
class Solution:
def networkDelayTime(self, times: List[List[int]], n: int, k: int) -> int:
```

Python:

```
class Solution(object):
def networkDelayTime(self, times, n, k):
"""
:type times: List[List[int]]
:type n: int
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} times
* @param {number} n
* @param {number} k
* @return {number}
*/
var networkDelayTime = function(times, n, k) {

};
```

TypeScript:

```
function networkDelayTime(times: number[][], n: number, k: number): number {

};
```

C#:

```
public class Solution {
public int NetworkDelayTime(int[][] times, int n, int k) {

}
}
```

C:

```
int networkDelayTime(int** times, int timesSize, int* timesColSize, int n,
int k){

}
```

Go:

```
func networkDelayTime(times [][]int, n int, k int) int {

}
```

Kotlin:

```
class Solution {
fun networkDelayTime(times: Array<IntArray>, n: Int, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func networkDelayTime(_ times: [[Int]], _ n: Int, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn network_delay_time(times: Vec<Vec<i32>>, n: i32, k: i32) -> i32 {

}
```

```
}
```

Ruby:

```
# @param {Integer[][]} times
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def network_delay_time(times, n, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $times
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function networkDelayTime($times, $n, $k) {

}
}
```

Dart:

```
class Solution {
int networkDelayTime(List<List<int>> times, int n, int k) {

}
}
```

Scala:

```
object Solution {
def networkDelayTime(times: Array[Array[Int]], n: Int, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec network_delay_time(times :: [[integer]], n :: integer, k :: integer) ::
integer
def network_delay_time(times, n, k) do

end
end
```

Erlang:

```
-spec network_delay_time(Times :: [[integer()]], N :: integer(), K ::
integer()) -> integer().
network_delay_time(Times, N, K) ->
.
```

Racket:

```
(define/contract (network-delay-time times n k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?
exact-integer?)

)
```

Solutions

C++ Solution:

```
/*
* Problem: Network Delay Time
* Difficulty: Medium
* Tags: graph, search, queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
```

```
int networkDelayTime(vector<vector<int>>& times, int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Network Delay Time
* Difficulty: Medium
* Tags: graph, search, queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int networkDelayTime(int[][] times, int n, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Network Delay Time
Difficulty: Medium
Tags: graph, search, queue, heap

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def networkDelayTime(self, times: List[List[int]], n: int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def networkDelayTime(self, times, n, k):
"""
:type times: List[List[int]]
:type n: int
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Network Delay Time
* Difficulty: Medium
* Tags: graph, search, queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} times
* @param {number} n
* @param {number} k
* @return {number}
*/
var networkDelayTime = function(times, n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Network Delay Time
* Difficulty: Medium
* Tags: graph, search, queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
function networkDelayTime(times: number[][], n: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Network Delay Time
* Difficulty: Medium
* Tags: graph, search, queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int NetworkDelayTime(int[][] times, int n, int k) {

}
}
```

C Solution:

```
/*
* Problem: Network Delay Time
* Difficulty: Medium
* Tags: graph, search, queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int networkDelayTime(int** times, int timesSize, int* timesColSize, int n,
int k){

}
```

Go Solution:

```
// Problem: Network Delay Time
// Difficulty: Medium
// Tags: graph, search, queue, heap
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func networkDelayTime(times [][]int, n int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun networkDelayTime(times: Array<IntArray>, n: Int, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func networkDelayTime(_ times: [[Int]], _ n: Int, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Network Delay Time
// Difficulty: Medium
// Tags: graph, search, queue, heap
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn network_delay_time(times: Vec<Vec<i32>>, n: i32, k: i32) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[][]} times
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def network_delay_time(times, n, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $times
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function networkDelayTime($times, $n, $k) {

}
}
```

Dart Solution:

```
class Solution {
int networkDelayTime(List<List<int>> times, int n, int k) {

}
}
```

Scala Solution:

```
object Solution {
def networkDelayTime(times: Array[Array[Int]], n: Int, k: Int): Int = {

}
```

```
}
```

Elixir Solution:

```
defmodule Solution do
@spec network_delay_time(times :: [[integer]], n :: integer, k :: integer) ::
integer
def network_delay_time(times, n, k) do

end
end
```

Erlang Solution:

```
-spec network_delay_time(Times :: [[integer()]], N :: integer(), K ::
integer()) -> integer().
network_delay_time(Times, N, K) ->
.
```

Racket Solution:

```
(define/contract (network-delay-time times n k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?
exact-integer?)

)
```