Problem 2737: Find the Closest Marked Node

Problem Information

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

Problem Description

You are given a positive integer

n

which is the number of nodes of a

0-indexed directed weighted

graph and a

0-indexed

2D array

edges

where

edges[i] = [u

i

, v

i

, w_i]

indicates that there is an edge from node u_i to node v_i with weight w_i.

You are also given a node s and a node array marked; your task is to find the minimum distance from s

to

any

of the nodes in

marked

.

Return

an integer denoting the minimum distance from

s

to any node in

marked

or

-1

if there are no paths from s to any of the marked nodes

.

Example 1:

Input:

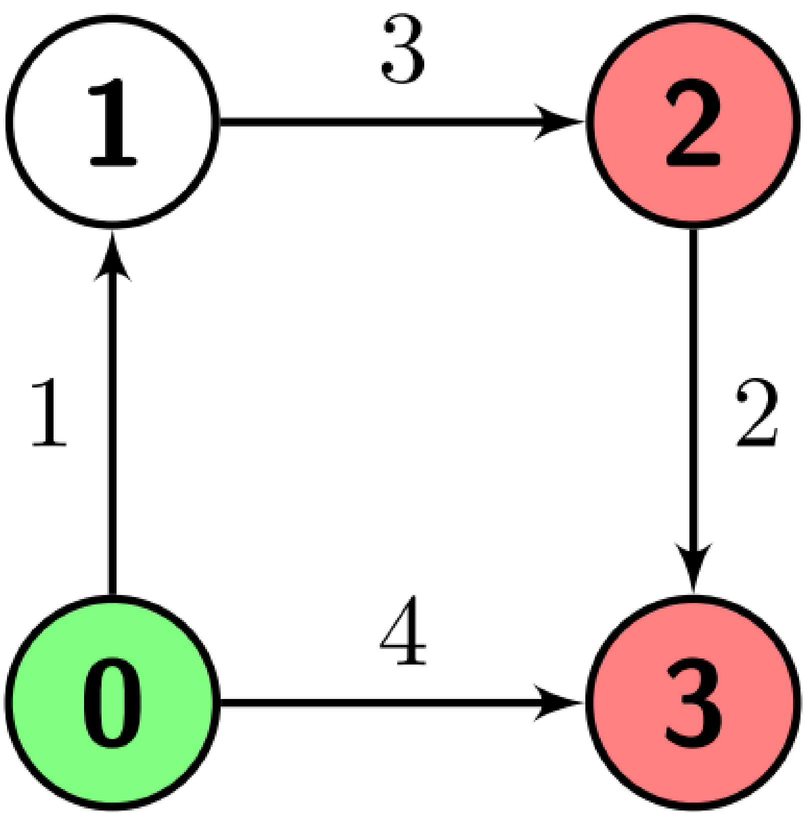
n = 4, edges = [[0,1,1],[1,2,3],[2,3,2],[0,3,4]], s = 0, marked = [2,3]

Output:

4

Explanation:

There is one path from node 0 (the green node) to node 2 (a red node), which is 0->1->2, and has a distance of 1 + 3 = 4. There are two paths from node 0 to node 3 (a red node), which are 0->1->2->3 and 0->3, the first one has a distance of 1 + 3 + 2 = 6 and the second one has a distance of 4. The minimum of them is 4.



Example 2:

Input:

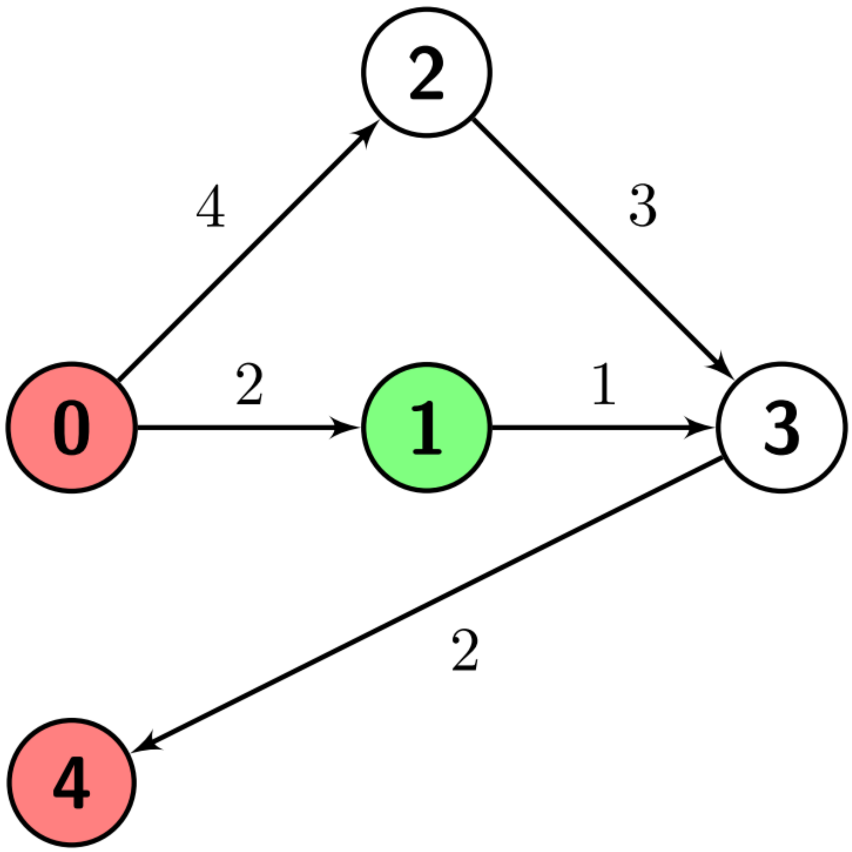
n = 5, edges = [[0,1,2],[0,2,4],[1,3,1],[2,3,3],[3,4,2]], s = 1, marked = [0,4]

Output:

3

Explanation:

There are no paths from node 1 (the green node) to node 0 (a red node). There is one path from node 1 to node 4 (a red node), which is 1->3->4, and has a distance of 1 + 2 = 3. So the answer is 3.



Example 3:

Input:

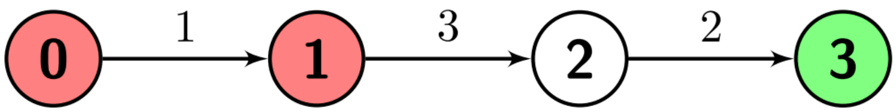
n = 4, edges = [[0,1,1],[1,2,3],[2,3,2]], s = 3, marked = [0,1]

Output:

-1

Explanation:

There are no paths from node 3 (the green node) to any of the marked nodes (the red nodes), so the answer is -1.



Constraints:

2 <= n <= 500

1 <= edges.length <= 10

4

edges[i].length = 3

0 <= edges[i][0], edges[i][1] <= n - 1

1 <= edges[i][2] <= 10

6

1 <= marked.length <= n - 1

0 <= s, marked[i] <= n - 1

s != marked[i]

marked[i] != marked[j]

for every

i != j

The graph might have

repeated edges

.

The graph is generated such that it has no

self-loops

.

Code Snippets

C++:

```
class Solution {
public:
int minimumDistance(int n, vector<vector<int>>& edges, int s, vector<int>&
marked) {

}
};
```

Java:

```
class Solution {
public int minimumDistance(int n, List<List<Integer>> edges, int s, int[]
marked) {

}
}
```

Python3:

```
class Solution:
def minimumDistance(self, n: int, edges: List[List[int]], s: int, marked:
List[int]) -> int:
```

Python:

```
class Solution(object):
def minimumDistance(self, n, edges, s, marked):
"""
```

```
:type n: int
:type edges: List[List[int]]
:type s: int
:type marked: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @param {number} s
* @param {number[]} marked
* @return {number}
*/
var minimumDistance = function(n, edges, s, marked) {

};
```

TypeScript:

```
function minimumDistance(n: number, edges: number[][], s: number, marked:
number[]): number {

};
```

C#:

```
public class Solution {
public int MinimumDistance(int n, IList<IList<int>> edges, int s, int[]
marked) {

}
}
```

C:

```
int minimumDistance(int n, int** edges, int edgesSize, int* edgesColSize, int
s, int* marked, int markedSize) {

}
```

Go:

```
func minimumDistance(n int, edges [][]int, s int, marked []int) int {

}
```

Kotlin:

```
class Solution {
fun minimumDistance(n: Int, edges: List<List<Int>>, s: Int, marked:
IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minimumDistance(_ n: Int, _ edges: [[Int]], _ s: Int, _ marked: [Int])
-> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_distance(n: i32, edges: Vec<Vec<i32>>, s: i32, marked:
Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer} s
# @param {Integer[]} marked
# @return {Integer}
def minimum_distance(n, edges, s, marked)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer $s
* @param Integer[] $marked
* @return Integer
*/
function minimumDistance($n, $edges, $s, $marked) {

}
}
```

Dart:

```
class Solution {
int minimumDistance(int n, List<List<int>> edges, int s, List<int> marked) {

}
}
```

Scala:

```
object Solution {
def minimumDistance(n: Int, edges: List[List[Int]], s: Int, marked:
Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_distance(n :: integer, edges :: [[integer]], s :: integer,
marked :: [integer]) :: integer
def minimum_distance(n, edges, s, marked) do

end
end
```

Erlang:

```
-spec minimum_distance(N :: integer(), Edges :: [[integer()]], S ::
integer(), Marked :: [integer()]) -> integer().
minimum_distance(N, Edges, S, Marked) ->
.
```

Racket:

```
(define/contract (minimum-distance n edges s marked)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer? (listof
exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Closest Marked Node
* 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 minimumDistance(int n, vector<vector<int>>& edges, int s, vector<int>&
marked) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Closest Marked Node
* 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 minimumDistance(int n, List<List<Integer>> edges, int s, int[]
marked) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Closest Marked Node
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 minimumDistance(self, n: int, edges: List[List[int]], s: int, marked:
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumDistance(self, n, edges, s, marked):
"""
:type n: int
:type edges: List[List[int]]
:type s: int
:type marked: List[int]
```

```
:rtype: int
"""
```

JavaScript Solution:

```javascript
/**
* Problem: Find the Closest Marked Node
* 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} s
* @param {number[]} marked
* @return {number}
*/
var minimumDistance = function(n, edges, s, marked) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Find the Closest Marked Node
* 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 minimumDistance(n: number, edges: number[][], s: number, marked:
number[]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Find the Closest Marked Node
* 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 MinimumDistance(int n, IList<IList<int>> edges, int s, int[]
marked) {

}
}
```

C Solution:

```
/*
* Problem: Find the Closest Marked Node
* 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
*/

int minimumDistance(int n, int** edges, int edgesSize, int* edgesColSize, int
s, int* marked, int markedSize) {

}
```

Go Solution:

```
// Problem: Find the Closest Marked Node
// 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 minimumDistance(n int, edges [][]int, s int, marked []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumDistance(n: Int, edges: List<List<Int>>, s: Int, marked:
IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumDistance(_ n: Int, _ edges: [[Int]], _ s: Int, _ marked: [Int])
-> Int {

}
}
```

Rust Solution:

```
// Problem: Find the Closest Marked Node
// 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_distance(n: i32, edges: Vec<Vec<i32>>, s: i32, marked:
```

```
Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer} s
# @param {Integer[]} marked
# @return {Integer}
def minimum_distance(n, edges, s, marked)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer $s
* @param Integer[] $marked
* @return Integer
*/
function minimumDistance($n, $edges, $s, $marked) {

}
}
```

Dart Solution:

```
class Solution {
int minimumDistance(int n, List<List<int>> edges, int s, List<int> marked) {

}
}
```

Scala Solution:

```
object Solution {
def minimumDistance(n: Int, edges: List[List[Int]], s: Int, marked:
Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_distance(n :: integer, edges :: [[integer]], s :: integer,
marked :: [integer]) :: integer
def minimum_distance(n, edges, s, marked) do

end
end
```

Erlang Solution:

```
-spec minimum_distance(N :: integer(), Edges :: [[integer()]], S ::
integer(), Marked :: [integer()]) -> integer().
minimum_distance(N, Edges, S, Marked) ->
.
```

Racket Solution:

```
(define/contract (minimum-distance n edges s marked)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer? (listof
exact-integer?) exact-integer?)
)
```