Problem 2359: Find Closest Node to Given Two Nodes

Problem Information

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

Problem Description

You are given a

directed

graph of

n

nodes numbered from

0

to

n - 1

, where each node has

at most one

outgoing edge.

The graph is represented with a given

0-indexed

array

edges

of size

n

, indicating that there is a directed edge from node

i

to node

edges[i]

. If there is no outgoing edge from

i

, then

edges[i] == -1

.

You are also given two integers

node1

and

node2

.

Return

the

index

of the node that can be reached from both

node1

and

node2

, such that the

maximum

between the distance from

node1

to that node, and from

node2

to that node is

minimized

. If there are multiple answers, return the node with the

smallest

index, and if no possible answer exists, return

-1

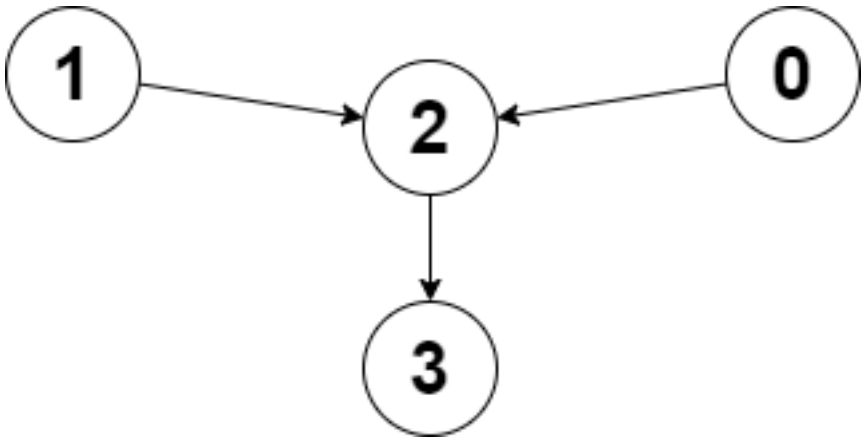
.

Note that

edges

may contain cycles.

Example 1:



Input:

edges = [2,2,3,-1], node1 = 0, node2 = 1

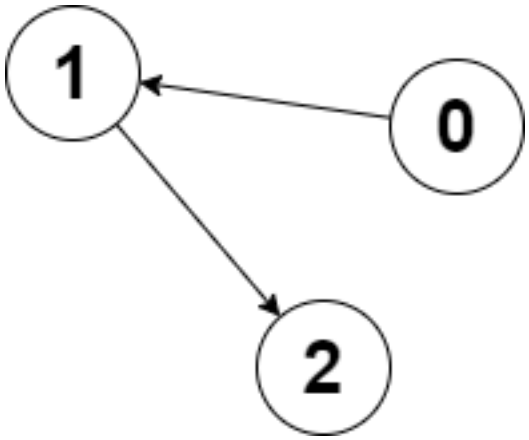
Output:

2

Explanation:

The distance from node 0 to node 2 is 1, and the distance from node 1 to node 2 is 1. The maximum of those two distances is 1. It can be proven that we cannot get a node with a smaller maximum distance than 1, so we return node 2.

Example 2:



Input:

edges = [1,2,-1], node1 = 0, node2 = 2

Output:

2

Explanation:

The distance from node 0 to node 2 is 2, and the distance from node 2 to itself is 0. The maximum of those two distances is 2. It can be proven that we cannot get a node with a smaller maximum distance than 2, so we return node 2.

Constraints:

n == edges.length

2 <= n <= 10

5

-1 <= edges[i] < n

edges[i] != i

0 <= node1, node2 < n

Code Snippets

C++:

```
class Solution {
public:
int closestMeetingNode(vector<int>& edges, int node1, int node2) {

}
};
```

Java:

```
class Solution {
public int closestMeetingNode(int[] edges, int node1, int node2) {

}
}
```

Python3:

```
class Solution:
def closestMeetingNode(self, edges: List[int], node1: int, node2: int) ->
int:
```

Python:

```
class Solution(object):
def closestMeetingNode(self, edges, node1, node2):
"""
:type edges: List[int]
:type node1: int
:type node2: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} edges
* @param {number} node1
* @param {number} node2
* @return {number}
```

```
*/
var closestMeetingNode = function(edges, node1, node2) {

};
```

TypeScript:

```
function closestMeetingNode(edges: number[], node1: number, node2: number):
number {

};
```

C#:

```
public class Solution {
public int ClosestMeetingNode(int[] edges, int node1, int node2) {

}
}
```

C:

```
int closestMeetingNode(int* edges, int edgesSize, int node1, int node2) {

}
```

Go:

```
func closestMeetingNode(edges []int, node1 int, node2 int) int {

}
```

Kotlin:

```
class Solution {
fun closestMeetingNode(edges: IntArray, node1: Int, node2: Int): Int {

}
}
```

Swift:

```
class Solution {
func closestMeetingNode(_ edges: [Int], _ node1: Int, _ node2: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn closest_meeting_node(edges: Vec<i32>, node1: i32, node2: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} edges
# @param {Integer} node1
# @param {Integer} node2
# @return {Integer}
def closest_meeting_node(edges, node1, node2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $edges
* @param Integer $node1
* @param Integer $node2
* @return Integer
*/
function closestMeetingNode($edges, $node1, $node2) {

}
}
```

Dart:

```
class Solution {
int closestMeetingNode(List<int> edges, int node1, int node2) {
```

```
}
}
```

Scala:

```
object Solution {
def closestMeetingNode(edges: Array[Int], node1: Int, node2: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec closest_meeting_node(edges :: [integer], node1 :: integer, node2 ::
integer) :: integer
def closest_meeting_node(edges, node1, node2) do

end
end
```

Erlang:

```
-spec closest_meeting_node(Edges :: [integer()], Node1 :: integer(), Node2 ::
integer()) -> integer().
closest_meeting_node(Edges, Node1, Node2) ->
.
```

Racket:

```
(define/contract (closest-meeting-node edges node1 node2)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Closest Node to Given Two Nodes
* 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 closestMeetingNode(vector<int>& edges, int node1, int node2) {

}
};
```

Java Solution:

```
/**
* Problem: Find Closest Node to Given Two Nodes
* 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 closestMeetingNode(int[] edges, int node1, int node2) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Closest Node to Given Two Nodes
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 closestMeetingNode(self, edges: List[int], node1: int, node2: int) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def closestMeetingNode(self, edges, node1, node2):
"""
:type edges: List[int]
:type node1: int
:type node2: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find Closest Node to Given Two Nodes
* 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} node1
* @param {number} node2
* @return {number}
*/
var closestMeetingNode = function(edges, node1, node2) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Find Closest Node to Given Two Nodes
* 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 closestMeetingNode(edges: number[], node1: number, node2: number):
number {

};
```

C# Solution:

```
/*
* Problem: Find Closest Node to Given Two Nodes
* 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 ClosestMeetingNode(int[] edges, int node1, int node2) {

}
}
```

C Solution:

```c
/*
* Problem: Find Closest Node to Given Two Nodes
* 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 closestMeetingNode(int* edges, int edgesSize, int node1, int node2) {

}
```

Go Solution:

```go
// Problem: Find Closest Node to Given Two Nodes
// 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 closestMeetingNode(edges []int, node1 int, node2 int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun closestMeetingNode(edges: IntArray, node1: Int, node2: Int): Int {

}
}
```

Swift Solution:

```swift
class Solution {
func closestMeetingNode(_ edges: [Int], _ node1: Int, _ node2: Int) -> Int {

}
```

```
}
```

Rust Solution:

```
// Problem: Find Closest Node to Given Two Nodes
// 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 closest_meeting_node(edges: Vec<i32>, node1: i32, node2: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} edges
# @param {Integer} node1
# @param {Integer} node2
# @return {Integer}
def closest_meeting_node(edges, node1, node2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $edges
* @param Integer $node1
* @param Integer $node2
* @return Integer
*/
function closestMeetingNode($edges, $node1, $node2) {

}
```

```
}
```

Dart Solution:

```
class Solution {
int closestMeetingNode(List<int> edges, int node1, int node2) {

}
}
```

Scala Solution:

```
object Solution {
def closestMeetingNode(edges: Array[Int], node1: Int, node2: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec closest_meeting_node(edges :: [integer], node1 :: integer, node2 ::
integer) :: integer
def closest_meeting_node(edges, node1, node2) do

end
end
```

Erlang Solution:

```
-spec closest_meeting_node(Edges :: [integer()], Node1 :: integer(), Node2 ::
integer()) -> integer().
closest_meeting_node(Edges, Node1, Node2) ->
.
```

Racket Solution:

```
(define/contract (closest-meeting-node edges node1 node2)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```