Problem 1059: All Paths from Source Lead to Destination

Problem Information

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

Problem Description

Given the

edges

of a directed graph where

edges[i] = [a

i

, b

i

]

indicates there is an edge between nodes

a

i

and

b

i

, and two nodes

source

and

destination

of this graph, determine whether or not all paths starting from

source

eventually, end at

destination

, that is:

At least one path exists from the

source

node to the

destination

node

If a path exists from the

source

node to a node with no outgoing edges, then that node is equal to

destination

.

The number of possible paths from

source

to

destination

is a finite number.

Return

true

if and only if all roads from

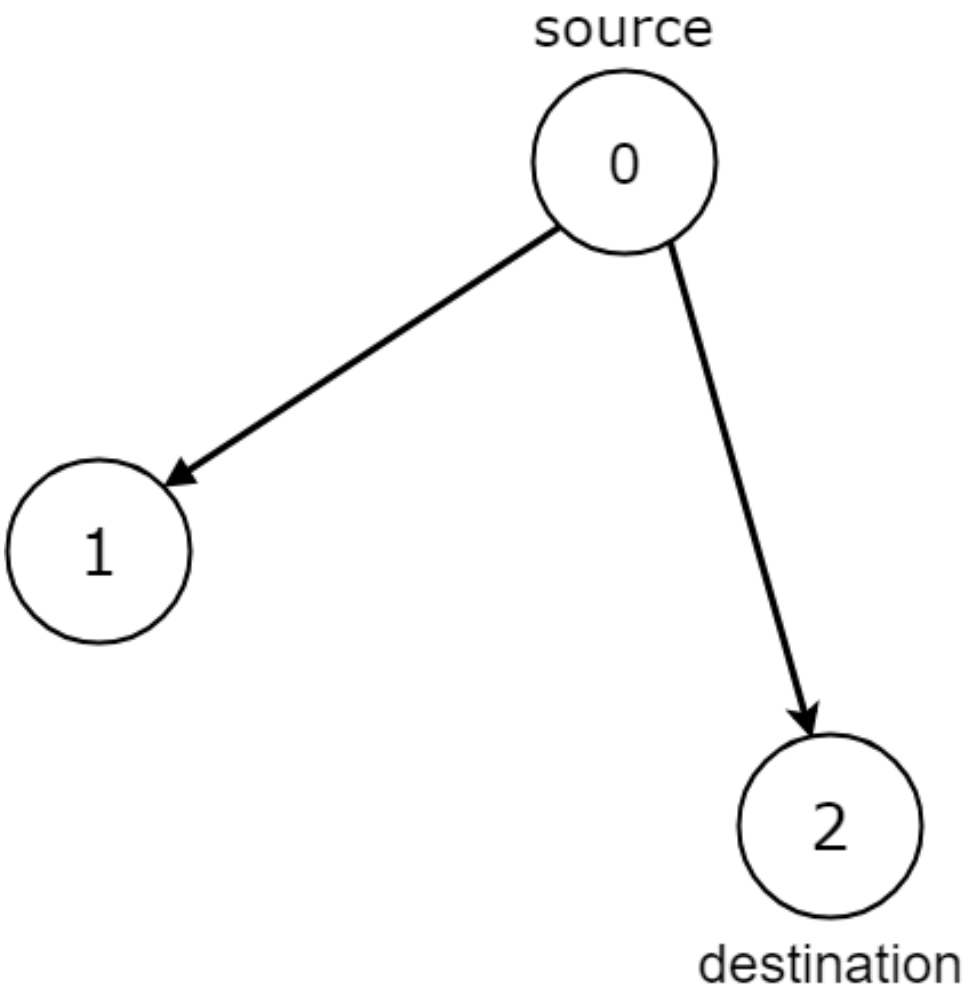
source

lead to

destination

.

Example 1:



Input:

n = 3, edges = [[0,1],[0,2]], source = 0, destination = 2

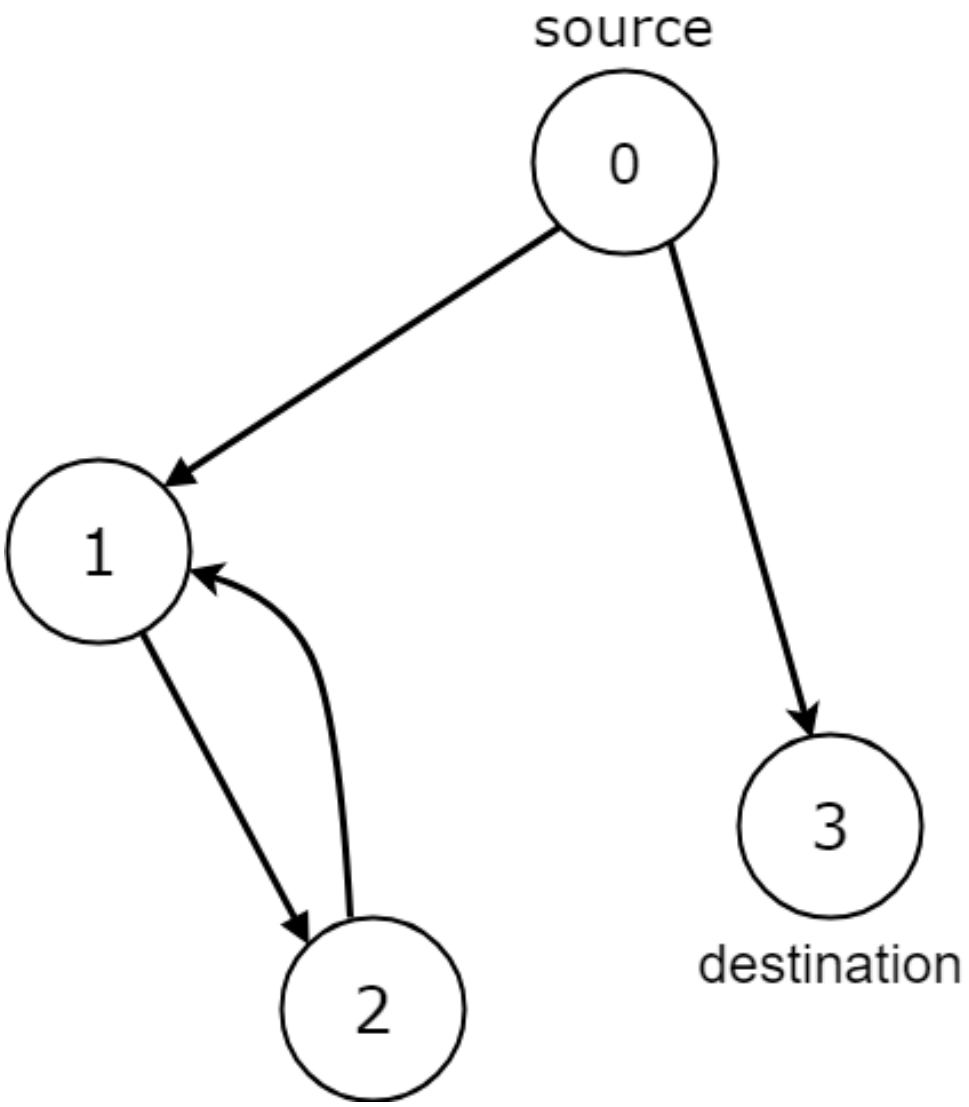
Output:

false

Explanation:

It is possible to reach and get stuck on both node 1 and node 2.

Example 2:



Input:

n = 4, edges = [[0,1],[0,3],[1,2],[2,1]], source = 0, destination = 3

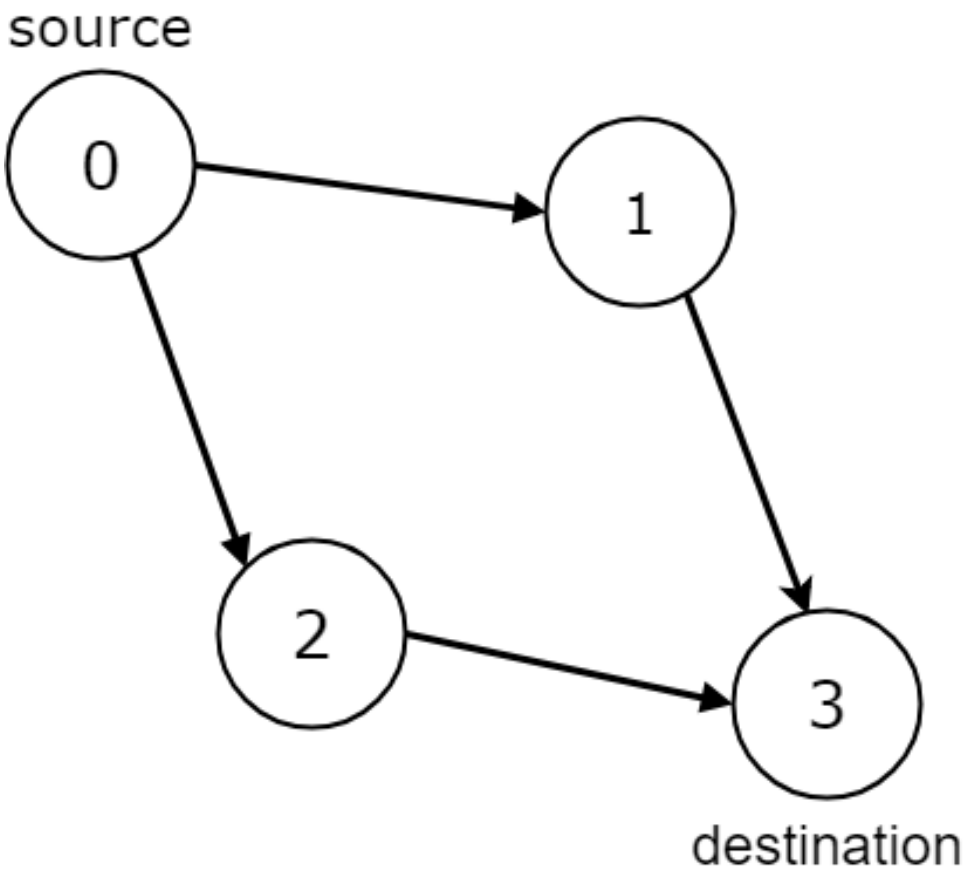
Output:

false

Explanation:

We have two possibilities: to end at node 3, or to loop over node 1 and node 2 indefinitely.

Example 3:



Input:

n = 4, edges = [[0,1],[0,2],[1,3],[2,3]], source = 0, destination = 3

Output:

true

Constraints:

1 <= n <= 10

4

0 <= edges.length <= 10

4

edges.length == 2

0 <= a

i

, b

i

<= n - 1

0 <= source <= n - 1

0 <= destination <= n - 1

The given graph may have self-loops and parallel edges.

Code Snippets

C++:

```
class Solution {
public:
bool leadsToDestination(int n, vector<vector<int>>& edges, int source, int
destination) {

}
};
```

Java:

```
class Solution {
public boolean leadsToDestination(int n, int[][] edges, int source, int
destination) {

}
}
```

Python3:

```
class Solution:
def leadsToDestination(self, n: int, edges: List[List[int]], source: int,
destination: int) -> bool:
```

Python:

```
class Solution(object):
def leadsToDestination(self, n, edges, source, destination):
"""
:type n: int
:type edges: List[List[int]]
:type source: int
:type destination: int
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @param {number} source
* @param {number} destination
* @return {boolean}
*/
var leadsToDestination = function(n, edges, source, destination) {

};
```

TypeScript:

```
function leadsToDestination(n: number, edges: number[][], source: number,
destination: number): boolean {

};
```

C#:

```
public class Solution {
public bool LeadsToDestination(int n, int[][] edges, int source, int
destination) {

}
```

```
}
```

C:

```
bool leadsToDestination(int n, int** edges, int edgesSize, int* edgesColSize,
int source, int destination) {

}
```

Go:

```
func leadsToDestination(n int, edges [][]int, source int, destination int)
bool {

}
```

Kotlin:

```
class Solution {
fun leadsToDestination(n: Int, edges: Array<IntArray>, source: Int,
destination: Int): Boolean {

}
}
```

Swift:

```
class Solution {
func leadsToDestination(_ n: Int, _ edges: [[Int]], _ source: Int, _
destination: Int) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn leads_to_destination(n: i32, edges: Vec<Vec<i32>>, source: i32,
destination: i32) -> bool {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer} source
# @param {Integer} destination
# @return {Boolean}
def leads_to_destination(n, edges, source, destination)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer $source
* @param Integer $destination
* @return Boolean
*/
function leadsToDestination($n, $edges, $source, $destination) {

}
}
```

Dart:

```
class Solution {
bool leadsToDestination(int n, List<List<int>> edges, int source, int
destination) {

}
}
```

Scala:

```
object Solution {
def leadsToDestination(n: Int, edges: Array[Array[Int]], source: Int,
destination: Int): Boolean = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec leads_to_destination(n :: integer, edges :: [[integer]], source ::
integer, destination :: integer) :: boolean
def leads_to_destination(n, edges, source, destination) do

end
end
```

Erlang:

```
-spec leads_to_destination(N :: integer(), Edges :: [[integer()]], Source ::
integer(), Destination :: integer()) -> boolean().
leads_to_destination(N, Edges, Source, Destination) ->
.
```

Racket:

```
(define/contract (leads-to-destination n edges source destination)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: All Paths from Source Lead to Destination
* Difficulty: Medium
* Tags: graph, sort
*
* 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:
bool leadsToDestination(int n, vector<vector<int>>& edges, int source, int
destination) {

}
};
```

Java Solution:

```
/**
* Problem: All Paths from Source Lead to Destination
* Difficulty: Medium
* Tags: graph, sort
*
* 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 boolean leadsToDestination(int n, int[][] edges, int source, int
destination) {

}
}
```

Python3 Solution:

```
"""
Problem: All Paths from Source Lead to Destination
Difficulty: Medium
Tags: graph, sort

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 leadsToDestination(self, n: int, edges: List[List[int]], source: int,
```

```
destination: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def leadsToDestination(self, n, edges, source, destination):
"""
:type n: int
:type edges: List[List[int]]
:type source: int
:type destination: int
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: All Paths from Source Lead to Destination
* Difficulty: Medium
* Tags: graph, sort
*
* 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} n
* @param {number[][]} edges
* @param {number} source
* @param {number} destination
* @return {boolean}
*/
var leadsToDestination = function(n, edges, source, destination) {

};
```

TypeScript Solution:

```
/**
* Problem: All Paths from Source Lead to Destination
* Difficulty: Medium
* Tags: graph, sort
*
* 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 leadsToDestination(n: number, edges: number[][], source: number,
destination: number): boolean {

};
```

C# Solution:

```
/*
* Problem: All Paths from Source Lead to Destination
* Difficulty: Medium
* Tags: graph, sort
*
* 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 bool LeadsToDestination(int n, int[][] edges, int source, int
destination) {

}
}
```

C Solution:

```
/*
* Problem: All Paths from Source Lead to Destination
* Difficulty: Medium
* Tags: graph, sort
*
* 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
*/

bool leadsToDestination(int n, int** edges, int edgesSize, int* edgesColSize,
int source, int destination) {

}
```

Go Solution:

```
// Problem: All Paths from Source Lead to Destination
// Difficulty: Medium
// Tags: graph, sort
//
// 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 leadsToDestination(n int, edges [][]int, source int, destination int)
bool {

}
```

Kotlin Solution:

```
class Solution {
fun leadsToDestination(n: Int, edges: Array<IntArray>, source: Int,
destination: Int): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func leadsToDestination(_ n: Int, _ edges: [[Int]], _ source: Int, _
destination: Int) -> Bool {

}
}
```

Rust Solution:

```
// Problem: All Paths from Source Lead to Destination
// Difficulty: Medium
// Tags: graph, sort
//
// 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 leads_to_destination(n: i32, edges: Vec<Vec<i32>>, source: i32,
destination: i32) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer} source
# @param {Integer} destination
# @return {Boolean}
def leads_to_destination(n, edges, source, destination)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer $source
* @param Integer $destination
* @return Boolean
*/
function leadsToDestination($n, $edges, $source, $destination) {

}
```

```
}
```

Dart Solution:

```
class Solution {
bool leadsToDestination(int n, List<List<int>> edges, int source, int
destination) {

}
}
```

Scala Solution:

```
object Solution {
def leadsToDestination(n: Int, edges: Array[Array[Int]], source: Int,
destination: Int): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec leads_to_destination(n :: integer, edges :: [[integer]], source ::
integer, destination :: integer) :: boolean
def leads_to_destination(n, edges, source, destination) do

end
end
```

Erlang Solution:

```
-spec leads_to_destination(N :: integer(), Edges :: [[integer()]], Source ::
integer(), Destination :: integer()) -> boolean().
leads_to_destination(N, Edges, Source, Destination) ->
.
```

Racket Solution:

```
(define/contract (leads-to-destination n edges source destination)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
```

```
exact-integer? boolean?)
)
```