Problem 1971: Find if Path Exists in Graph

Problem Information

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

Problem Description

There is a

bi-directional

graph with

n

vertices, where each vertex is labeled from

0

to

n - 1

(

inclusive

). The edges in the graph are represented as a 2D integer array

edges

, where each

edges[i] = [u

i

, v

i

]

denotes a bi-directional edge between vertex

u

i

and vertex

v

i

. Every vertex pair is connected by

at most one

edge, and no vertex has an edge to itself.

You want to determine if there is a

valid path

that exists from vertex

source

to vertex

destination

.

Given

edges

and the integers

n

,

source

, and

destination

, return

true

if there is a

valid path

from

source

to

destination

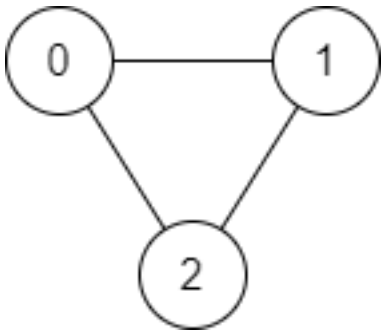
, or

false

otherwise

.

Example 1:



Input:

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

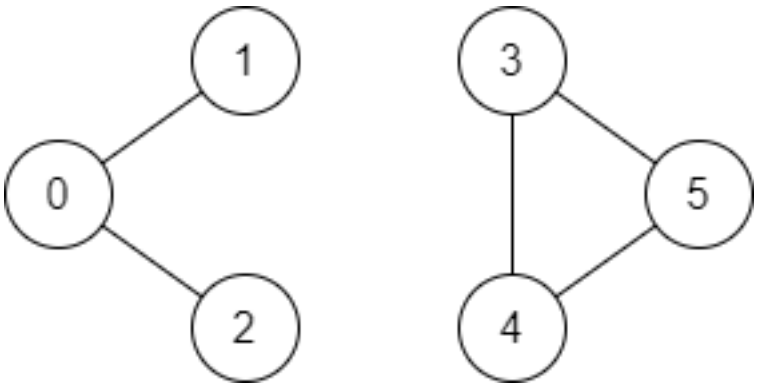
Output:

true

Explanation:

There are two paths from vertex 0 to vertex 2: - 0 → 1 → 2 - 0 → 2

Example 2:



Input:

n = 6, edges = [[0,1],[0,2],[3,5],[5,4],[4,3]], source = 0, destination = 5

Output:

false

Explanation:

There is no path from vertex 0 to vertex 5.

Constraints:

1 <= n <= 2 * 10

5

0 <= edges.length <= 2 * 10

5

edges[i].length == 2

0 <= u

i

, v

i

<= n - 1

u

i

!= v

i

0 <= source, destination <= n - 1

There are no duplicate edges.

There are no self edges.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def validPath(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 validPath = function(n, edges, source, destination) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn valid_path(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 valid_path(n, edges, source, destination)

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Find if Path Exists in Graph
* Difficulty: Easy
* Tags: array, graph, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Find if Path Exists in Graph
* Difficulty: Easy
* Tags: array, graph, dp, search
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Find if Path Exists in Graph
Difficulty: Easy
Tags: array, graph, dp, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def validPath(self, n: int, edges: List[List[int]], source: int, destination:
int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find if Path Exists in Graph
* Difficulty: Easy
* Tags: array, graph, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Find if Path Exists in Graph
* Difficulty: Easy
* Tags: array, graph, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: Find if Path Exists in Graph
* Difficulty: Easy
* Tags: array, graph, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Find if Path Exists in Graph
* Difficulty: Easy
* Tags: array, graph, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

Go Solution:

```
// Problem: Find if Path Exists in Graph
// Difficulty: Easy
// Tags: array, graph, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table
```

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

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find if Path Exists in Graph
// Difficulty: Easy
// Tags: array, graph, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn valid_path(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 valid_path(n, edges, source, destination)

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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