Problem 3123: Find Edges in Shortest Paths

Problem Information

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

Problem Description

You are given an undirected weighted graph of

n

nodes numbered from 0 to

n - 1

. The graph consists of

m

edges represented by a 2D array

edges

, where

edges[i] = [a

i

, b

i

, w

i

]

indicates that there is an edge between nodes

a

i

and

b

i

with weight

w

i

.

Consider all the shortest paths from node 0 to node

n - 1

in the graph. You need to find a

boolean

array

answer

where

answer[i]

is

true

if the edge

edges[i]

is part of

at least

one shortest path. Otherwise,

answer[i]

is

false

.

Return the array

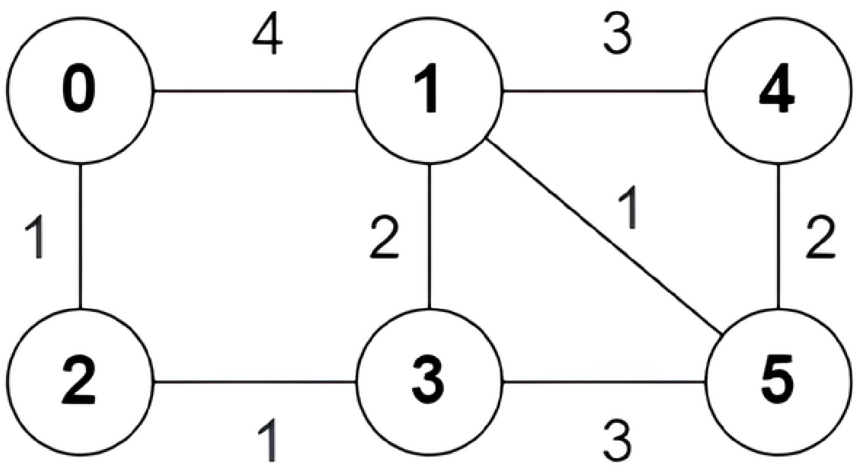
answer

.

Note

that the graph may not be connected.

Example 1:



Input:

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

Output:

[true,true,true,false,true,true,true,false]

Explanation:

The following are

all

the shortest paths between nodes 0 and 5:

The path

0 -> 1 -> 5

: The sum of weights is

4 + 1 = 5

.

The path

0 -> 2 -> 3 -> 5

: The sum of weights is

1 + 1 + 3 = 5

.

The path

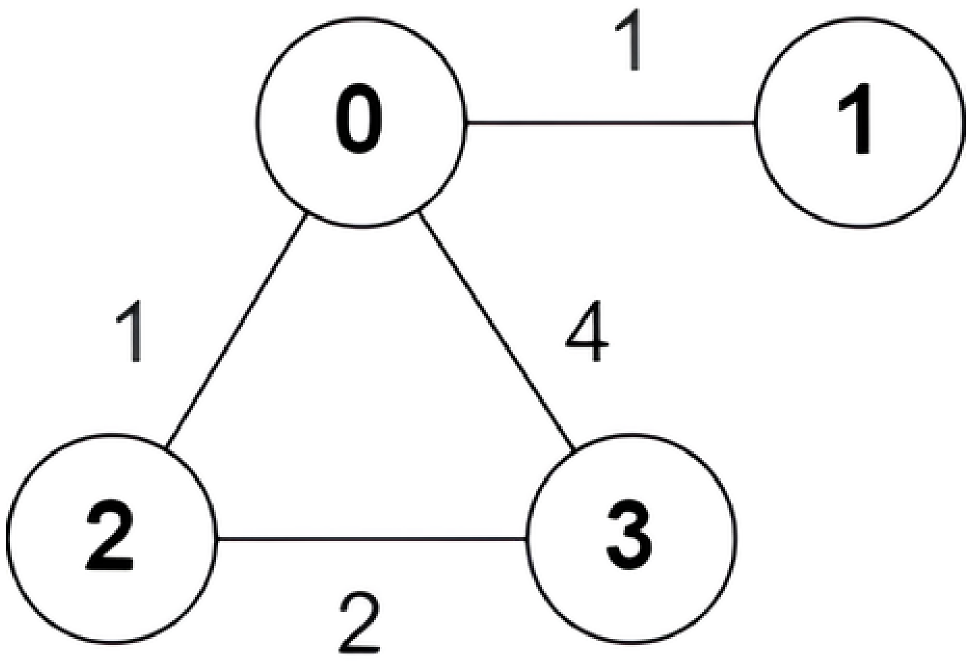
0 -> 2 -> 3 -> 1 -> 5

: The sum of weights is

1 + 1 + 2 + 1 = 5

.

Example 2:



Input:

n = 4, edges = [[2,0,1],[0,1,1],[0,3,4],[3,2,2]] Output:

[true,false,false,true]

Explanation:

There is one shortest path between nodes 0 and 3, which is the path

0 -> 2 -> 3

with the sum of weights

1 + 2 = 3

.

Constraints:

2 <= n <= 5 * 10

4

m == edges.length

1 <= m <= min(5 * 10

4

, n * (n - 1) / 2)

0 <= a

i

, b

i

< n

a

i

!= b

i

1 <= w

i

<= 10

5

There are no repeated edges.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean[] findAnswer(int n, int[][] edges) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findAnswer(n: number, edges: number[][]): boolean[] {

};
```

C#:

```
public class Solution {
public bool[] FindAnswer(int n, int[][] edges) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
bool* findAnswer(int n, int** edges, int edgesSize, int* edgesColSize, int*
returnSize) {

}
```

Go:

```
func findAnswer(n int, edges [][]int) []bool {

}
```

Kotlin:

```
class Solution {
fun findAnswer(n: Int, edges: Array<IntArray>): BooleanArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @return {Boolean[]}
```

```
def find_answer(n, edges)

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<bool> findAnswer(int n, List<List<int>> edges) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec find_answer(N :: integer(), Edges :: [[integer()]]) -> [boolean()].
find_answer(N, Edges) ->
.
```

Racket:

```
(define/contract (find-answer n edges)
(-> exact-integer? (listof (listof exact-integer?)) (listof boolean?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Edges in Shortest Paths
* Difficulty: Hard
* Tags: array, graph, search, 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:
vector<bool> findAnswer(int n, vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Find Edges in Shortest Paths
* Difficulty: Hard
* Tags: array, graph, search, 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 boolean[] findAnswer(int n, int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Edges in Shortest Paths
Difficulty: Hard
Tags: array, graph, search, 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 findAnswer(self, n: int, edges: List[List[int]]) -> List[bool]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Edges in Shortest Paths
* Difficulty: Hard
* Tags: array, graph, search, 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
* @return {boolean[]}
*/
var findAnswer = function(n, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Edges in Shortest Paths
* Difficulty: Hard
* Tags: array, graph, search, 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 findAnswer(n: number, edges: number[][]): boolean[] {

};
```

C# Solution:

```
/*
* Problem: Find Edges in Shortest Paths
* Difficulty: Hard
* Tags: array, graph, search, 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 bool[] FindAnswer(int n, int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Find Edges in Shortest Paths
* Difficulty: Hard
* Tags: array, graph, search, 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
bool* findAnswer(int n, int** edges, int edgesSize, int* edgesColSize, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Find Edges in Shortest Paths
// Difficulty: Hard
// Tags: array, graph, search, 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 findAnswer(n int, edges [][]int) []bool {
}
```

Kotlin Solution:

```
class Solution {
fun findAnswer(n: Int, edges: Array<IntArray>): BooleanArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find Edges in Shortest Paths
// Difficulty: Hard
// Tags: array, graph, search, 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 find_answer(n: i32, edges: Vec<Vec<i32>>) -> Vec<bool> {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
List<bool> findAnswer(int n, List<List<int>> edges) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec find_answer(N :: integer(), Edges :: [[integer()]]) -> [boolean()].
find_answer(N, Edges) ->
.
```

Racket Solution:

```
(define/contract (find-answer n edges)
(-> exact-integer? (listof (listof exact-integer?)) (listof boolean?))
)
```