Problem 2699: Modify Graph Edge Weights

Problem Information

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

Problem Description

You are given an

undirected weighted

connected

graph containing

n

nodes labeled from

0

to

n - 1

, and an integer 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

.

Some edges have a weight of

-1

(

w

i

= -1

), while others have a

positive

weight (

w

i

> 0

).

Your task is to modify

all edges

with a weight of

-1

by assigning them

positive integer values

in the range

[1, 2 * 10

9

]

so that the

shortest distance

between the nodes

source

and

destination

becomes equal to an integer

target

. If there are

multiple

modifications

that make the shortest distance between

source

and

destination

equal to

target

, any of them will be considered correct.

Return

an array containing all edges (even unmodified ones) in any order if it is possible to make the shortest distance from

source

to

destination

equal to

target

, or an

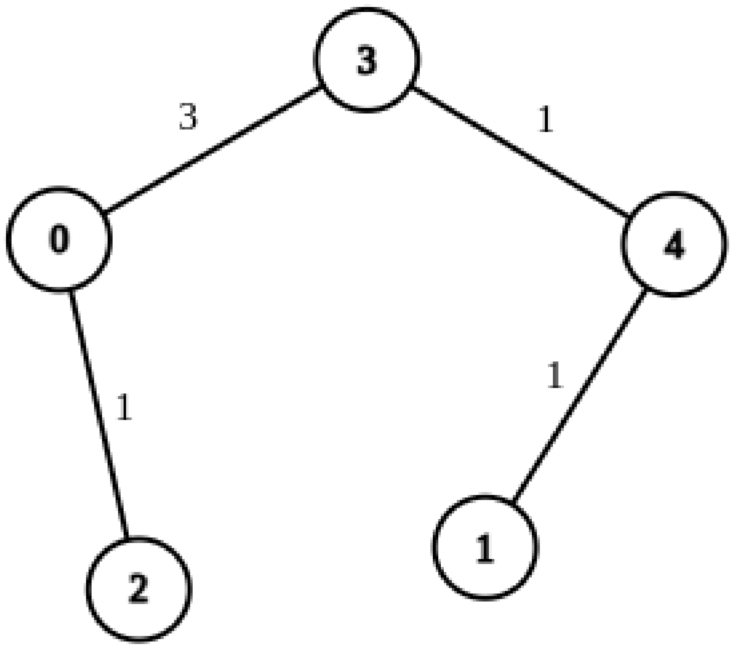
empty array

if it's impossible.

Note:

You are not allowed to modify the weights of edges with initial positive weights.

Example 1:



Input:

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

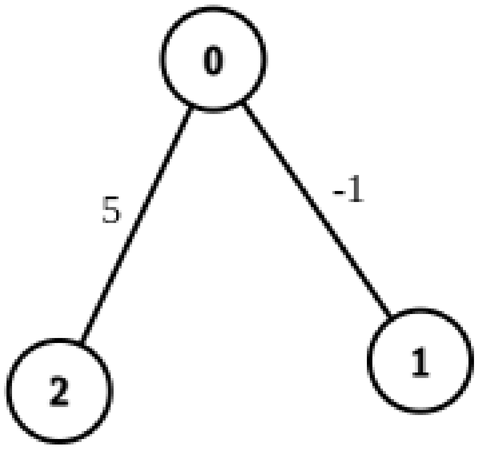
Output:

[[4,1,1],[2,0,1],[0,3,3],[4,3,1]]

Explanation:

The graph above shows a possible modification to the edges, making the distance from 0 to 1 equal to 5.

Example 2:



Input:

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

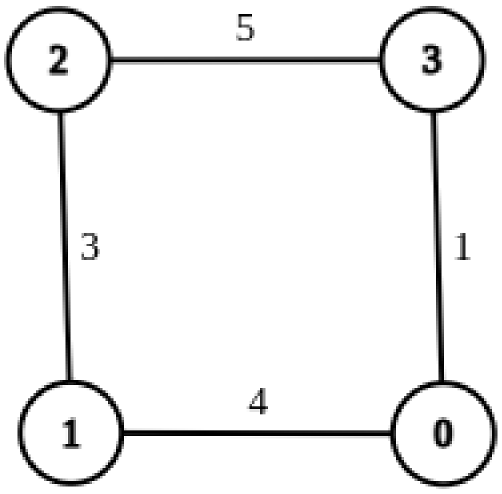
Output:

[]

Explanation:

The graph above contains the initial edges. It is not possible to make the distance from 0 to 2 equal to 6 by modifying the edge with weight -1. So, an empty array is returned.

Example 3:



Input:

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

Output:

[[1,0,4],[1,2,3],[2,3,5],[0,3,1]]

Explanation:

The graph above shows a modified graph having the shortest distance from 0 to 2 as 6.

Constraints:

1 <= n <= 100

1 <= edges.length <= n * (n - 1) / 2

edges[i].length == 3

0 <= a

i

, b

i

< n

w

i

= -1

or

1 <= w

i

<= 10

7

a

i

!= b

i

0 <= source, destination < n

source != destination

1 <= target <= 10

9

The graph is connected, and there are no self-loops or repeated edges

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int[][] modifiedGraphEdges(int n, int[][] edges, int source, int
destination, int target) {

}
}
```

Python3:

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

Python:

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

```
:type destination: int
:type target: int
:rtype: List[List[int]]
"""
```

JavaScript:

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

};
```

TypeScript:

```
function modifiedGraphEdges(n: number, edges: number[][], source: number,
destination: number, target: number): number[][] {

};
```

C#:

```
public class Solution {
public int[][] ModifiedGraphEdges(int n, int[][] edges, int source, int
destination, int target) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
```

```
*/
int** modifiedGraphEdges(int n, int** edges, int edgesSize, int*
edgesColSize, int source, int destination, int target, int* returnSize, int**
returnColumnSizes) {

}
```

Go:

```
func modifiedGraphEdges(n int, edges [][]int, source int, destination int,
target int) [][]int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<List<int>> modifiedGraphEdges(int n, List<List<int>> edges, int source,
int destination, int target) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (modified-graph-edges n edges source destination target)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer? exact-integer? (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Modify Graph Edge Weights
* Difficulty: Hard
* 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:
vector<vector<int>> modifiedGraphEdges(int n, vector<vector<int>>& edges, int
source, int destination, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Modify Graph Edge Weights
* Difficulty: Hard
* 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[][] modifiedGraphEdges(int n, int[][] edges, int source, int
destination, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Modify Graph Edge Weights
Difficulty: Hard
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 modifiedGraphEdges(self, n: int, edges: List[List[int]], source: int,
destination: int, target: int) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Modify Graph Edge Weights
* Difficulty: Hard
* 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} source
* @param {number} destination
* @param {number} target
* @return {number[][]}
*/
var modifiedGraphEdges = function(n, edges, source, destination, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Modify Graph Edge Weights
* Difficulty: Hard
* 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 modifiedGraphEdges(n: number, edges: number[][], source: number,
destination: number, target: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Modify Graph Edge Weights
* Difficulty: Hard
* 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[][] ModifiedGraphEdges(int n, int[][] edges, int source, int
destination, int target) {

}
}
```

C Solution:

```
/*
* Problem: Modify Graph Edge Weights
* Difficulty: Hard
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** modifiedGraphEdges(int n, int** edges, int edgesSize, int*
edgesColSize, int source, int destination, int target, int* returnSize, int**
returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Modify Graph Edge Weights
// Difficulty: Hard
// 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 modifiedGraphEdges(n int, edges [][]int, source int, destination int,
target int) [][]int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Modify Graph Edge Weights
// Difficulty: Hard
// 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 modified_graph_edges(n: i32, edges: Vec<Vec<i32>>, source: i32,
destination: i32, target: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> modifiedGraphEdges(int n, List<List<int>> edges, int source,
int destination, int target) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (modified-graph-edges n edges source destination target)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer? exact-integer? (listof (listof exact-integer?)))
)
```