Problem 2203: Minimum Weighted Subgraph With the Required Paths

Problem Information

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

Problem Description

You are given an integer

n

denoting the number of nodes of a

weighted directed

graph. The nodes are numbered from

0

to

n - 1

.

You are also given a 2D integer array

edges

where

edges[i] = [from

i

, to

i

, weight

i

]

denotes that there exists a

directed

edge from

from

i

to

to

i

with weight

weight

i

.

Lastly, you are given three

distinct

integers

src1

,

src2

, and

dest

denoting three distinct nodes of the graph.

Return

the

minimum weight

of a subgraph of the graph such that it is

possible

to reach

dest

from both

src1

and

src2

via a set of edges of this subgraph

. In case such a subgraph does not exist, return

-1

.

A

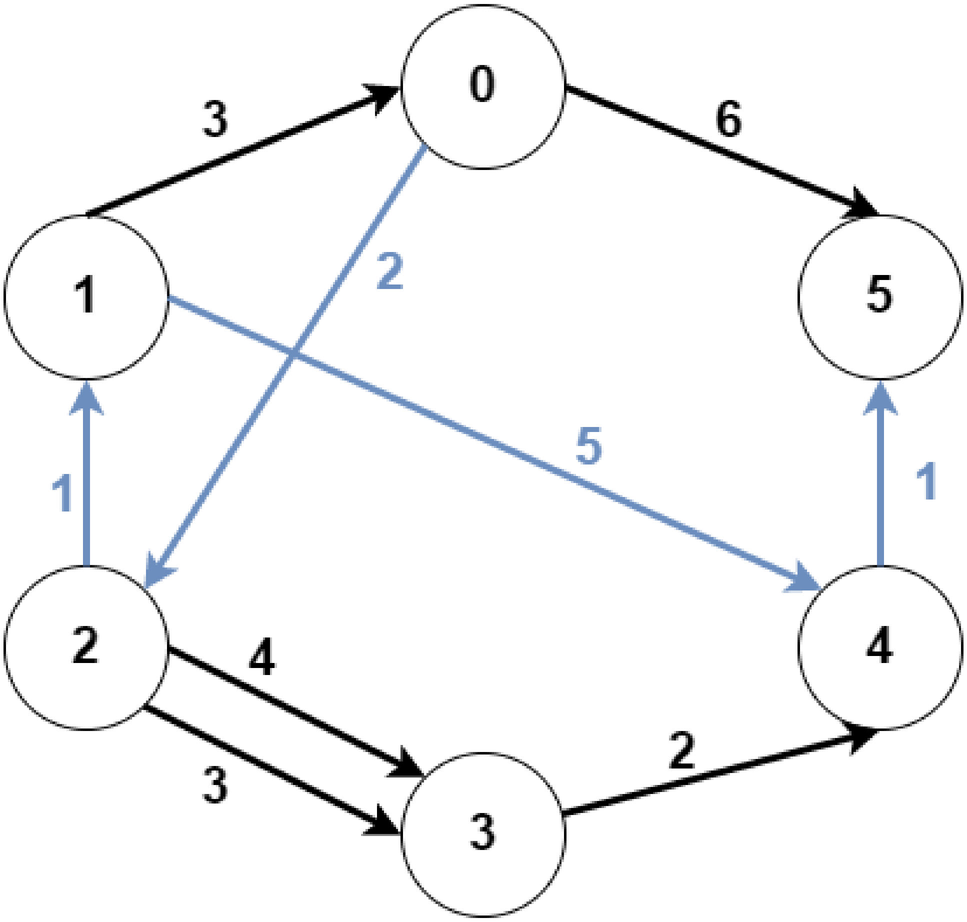
subgraph

is a graph whose vertices and edges are subsets of the original graph. The

weight

of a subgraph is the sum of weights of its constituent edges.

Example 1:



Input:

n = 6, edges = [[0,2,2],[0,5,6],[1,0,3],[1,4,5],[2,1,1],[2,3,3],[2,3,4],[3,4,2],[4,5,1]], src1 = 0, src2 = 1, dest = 5

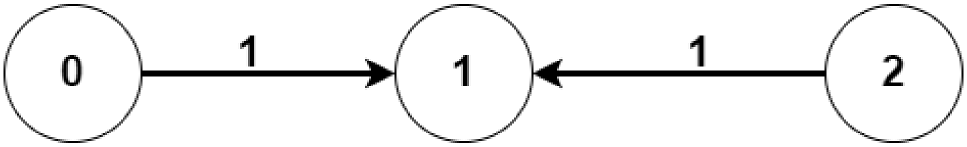
Output:

9

Explanation:

The above figure represents the input graph. The blue edges represent one of the subgraphs that yield the optimal answer. Note that the subgraph [[1,0,3],[0,5,6]] also yields the optimal answer. It is not possible to get a subgraph with less weight satisfying all the constraints.

Example 2:



Input:

n = 3, edges = [[0,1,1],[2,1,1]], src1 = 0, src2 = 1, dest = 2

Output:

-1

Explanation:

The above figure represents the input graph. It can be seen that there does not exist any path from node 1 to node 2, hence there are no subgraphs satisfying all the constraints.

Constraints:

3 <= n <= 10

0 <= edges.length <= 10

5

edges[i].length == 3

0 <= from

i

, to

i

, src1, src2, dest <= n - 1

from

i

!= to

i

src1

,

src2

, and

dest

are pairwise distinct.

1 <= weight[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
long long minimumWeight(int n, vector<vector<int>>& edges, int src1, int
src2, int dest) {

}
};
```

Java:

```
class Solution {
public long minimumWeight(int n, int[][] edges, int src1, int src2, int dest)
{

}
}
```

Python3:

```
class Solution:
def minimumWeight(self, n: int, edges: List[List[int]], src1: int, src2: int,
dest: int) -> int:
```

Python:

```
class Solution(object):
def minimumWeight(self, n, edges, src1, src2, dest):
"""
:type n: int
:type edges: List[List[int]]
:type src1: int
:type src2: int
:type dest: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @param {number} src1
* @param {number} src2
* @param {number} dest
* @return {number}
*/
var minimumWeight = function(n, edges, src1, src2, dest) {

};
```

TypeScript:

```
function minimumWeight(n: number, edges: number[][], src1: number, src2:
number, dest: number): number {

};
```

C#:

```
public class Solution {
public long MinimumWeight(int n, int[][] edges, int src1, int src2, int dest)
{

}
}
```

C:

```
long long minimumWeight(int n, int** edges, int edgesSize, int* edgesColSize,
int src1, int src2, int dest) {

}
```

Go:

```
func minimumWeight(n int, edges [][]int, src1 int, src2 int, dest int) int64
{

}
```

Kotlin:

```
class Solution {
fun minimumWeight(n: Int, edges: Array<IntArray>, src1: Int, src2: Int, dest:
Int): Long {

}
}
```

Swift:

```
class Solution {
func minimumWeight(_ n: Int, _ edges: [[Int]], _ src1: Int, _ src2: Int, _
dest: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_weight(n: i32, edges: Vec<Vec<i32>>, src1: i32, src2: i32,
dest: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer} src1
# @param {Integer} src2
# @param {Integer} dest
# @return {Integer}
def minimum_weight(n, edges, src1, src2, dest)

end
```

PHP:

```
class Solution {
```

```
/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer $src1
* @param Integer $src2
* @param Integer $dest
* @return Integer
*/
function minimumWeight($n, $edges, $src1, $src2, $dest) {

}
}
```

Dart:

```
class Solution {
int minimumWeight(int n, List<List<int>> edges, int src1, int src2, int dest)
{

}
}
```

Scala:

```
object Solution {
def minimumWeight(n: Int, edges: Array[Array[Int]], src1: Int, src2: Int,
dest: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_weight(n :: integer, edges :: [[integer]], src1 :: integer,
src2 :: integer, dest :: integer) :: integer
def minimum_weight(n, edges, src1, src2, dest) do

end
end
```

Erlang:

```
-spec minimum_weight(N :: integer(), Edges :: [[integer()]], Src1 ::
integer(), Src2 :: integer(), Dest :: integer()) -> integer().
minimum_weight(N, Edges, Src1, Src2, Dest) ->
.
```

Racket:

```
(define/contract (minimum-weight n edges src1 src2 dest)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Weighted Subgraph With the Required Paths
* Difficulty: Hard
* Tags: array, graph
*
* 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:
long long minimumWeight(int n, vector<vector<int>>& edges, int src1, int
src2, int dest) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Weighted Subgraph With the Required Paths
* Difficulty: Hard
* Tags: array, graph
*
```

```
* 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 long minimumWeight(int n, int[][] edges, int src1, int src2, int dest)
{

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Weighted Subgraph With the Required Paths
Difficulty: Hard
Tags: array, graph

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 minimumWeight(self, n: int, edges: List[List[int]], src1: int, src2: int,
dest: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumWeight(self, n, edges, src1, src2, dest):
"""
:type n: int
:type edges: List[List[int]]
:type src1: int
:type src2: int
:type dest: int
:rtype: int
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Weighted Subgraph With the Required Paths
* Difficulty: Hard
* Tags: array, graph
*
* 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} src1
* @param {number} src2
* @param {number} dest
* @return {number}
*/
var minimumWeight = function(n, edges, src1, src2, dest) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Weighted Subgraph With the Required Paths
* Difficulty: Hard
* Tags: array, graph
*
* 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 minimumWeight(n: number, edges: number[][], src1: number, src2:
number, dest: number): number {
};
```

C# Solution:

```
/*
* Problem: Minimum Weighted Subgraph With the Required Paths
* Difficulty: Hard
* Tags: array, graph
*
* 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 long MinimumWeight(int n, int[][] edges, int src1, int src2, int dest)
{

}
}
```

C Solution:

```
/*
* Problem: Minimum Weighted Subgraph With the Required Paths
* Difficulty: Hard
* Tags: array, graph
*
* 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
*/

long long minimumWeight(int n, int** edges, int edgesSize, int* edgesColSize,
int src1, int src2, int dest) {

}
```

Go Solution:

```
// Problem: Minimum Weighted Subgraph With the Required Paths
// Difficulty: Hard
// Tags: array, graph
//
// 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 minimumWeight(n int, edges [][]int, src1 int, src2 int, dest int) int64
{

}
```

Kotlin Solution:

```
class Solution {
fun minimumWeight(n: Int, edges: Array<IntArray>, src1: Int, src2: Int, dest:
Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func minimumWeight(_ n: Int, _ edges: [[Int]], _ src1: Int, _ src2: Int, _
dest: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Weighted Subgraph With the Required Paths
// Difficulty: Hard
// Tags: array, graph
//
// 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 minimum_weight(n: i32, edges: Vec<Vec<i32>>, src1: i32, src2: i32,
dest: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer} src1
# @param {Integer} src2
# @param {Integer} dest
# @return {Integer}
def minimum_weight(n, edges, src1, src2, dest)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer $src1
* @param Integer $src2
* @param Integer $dest
* @return Integer
*/
function minimumWeight($n, $edges, $src1, $src2, $dest) {

}
}
```

Dart Solution:

```
class Solution {
int minimumWeight(int n, List<List<int>> edges, int src1, int src2, int dest)
{
}
}
```

Scala Solution:

```
object Solution {
def minimumWeight(n: Int, edges: Array[Array[Int]], src1: Int, src2: Int,
dest: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_weight(n :: integer, edges :: [[integer]], src1 :: integer,
src2 :: integer, dest :: integer) :: integer
def minimum_weight(n, edges, src1, src2, dest) do

end
end
```

Erlang Solution:

```
-spec minimum_weight(N :: integer(), Edges :: [[integer()]], Src1 ::
integer(), Src2 :: integer(), Dest :: integer()) -> integer().
minimum_weight(N, Edges, Src1, Src2, Dest) ->
.
```

Racket Solution:

```
(define/contract (minimum-weight n edges src1 src2 dest)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer? exact-integer? exact-integer?)
)
```