Problem 3419: Minimize the Maximum Edge Weight of Graph

Problem Information

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

Problem Description

You are given two integers,

n

and

threshold

, as well as a

directed

weighted graph of

n

nodes numbered from 0 to

n - 1

. The graph is represented by a

2D

integer array

edges

, where

edges[i] = [A

i

, B

i

, W

i

]

indicates that there is an edge going from node

A

i

to node

B

i

with weight

W

i

.

You have to remove some edges from this graph (possibly

none

), so that it satisfies the following conditions:

Node 0 must be reachable from all other nodes.

The

maximum

edge weight in the resulting graph is

minimized

.

Each node has

at most

threshold

outgoing edges.

Return the

minimum

possible value of the

maximum

edge weight after removing the necessary edges. If it is impossible for all conditions to be satisfied, return -1.

Example 1:

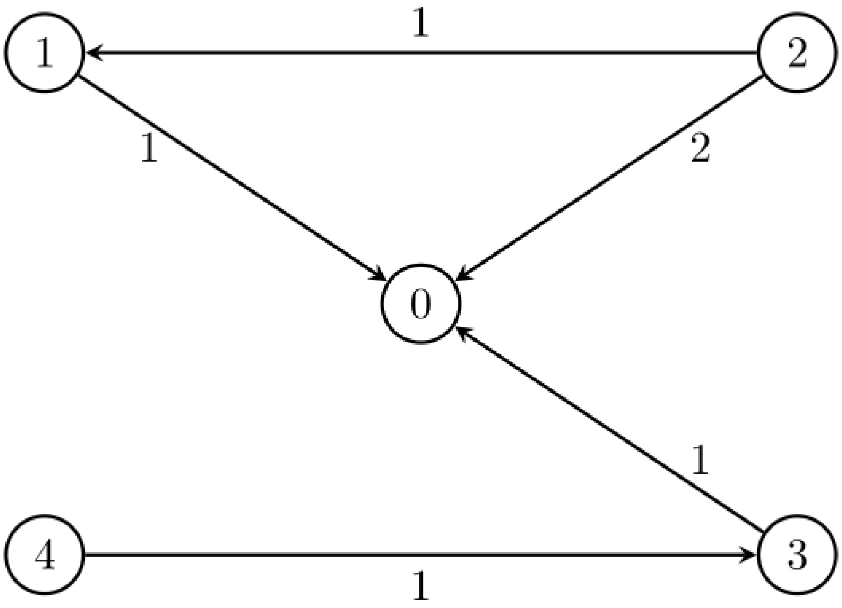
Input:

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

Output:

1

Explanation:



Remove the edge

2 -> 0

. The maximum weight among the remaining edges is 1.

Example 2:

Input:

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

Output:

-1

Explanation:

It is impossible to reach node 0 from node 2.

Example 3:

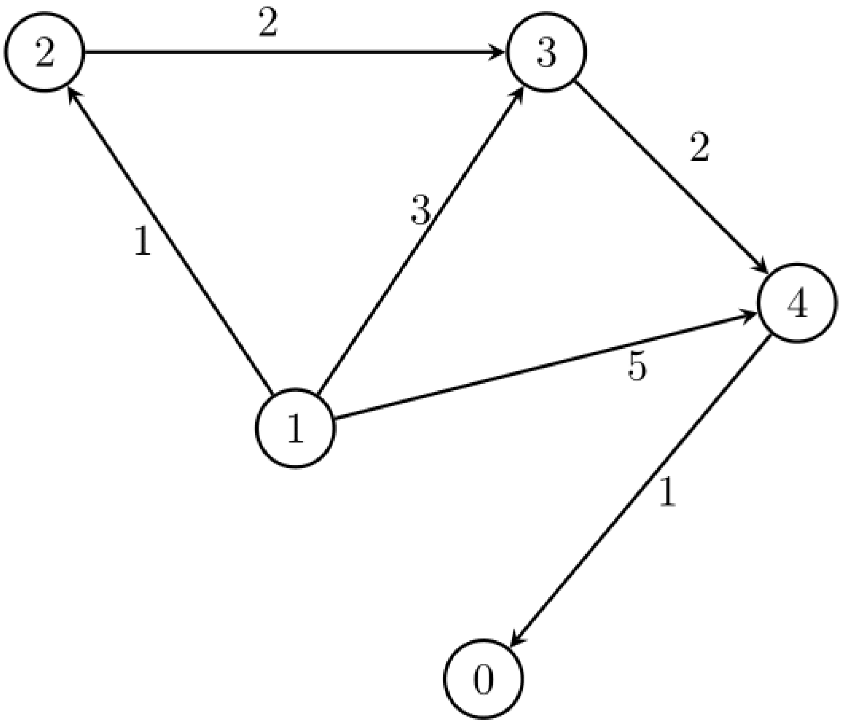
Input:

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

Output:

2

Explanation:



Remove the edges

1 -> 3

and

1 -> 4

. The maximum weight among the remaining edges is 2.

Example 4:

Input:

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

Output:

-1

Constraints:

2 <= n <= 10

5

1 <= threshold <= n - 1

1 <= edges.length <= min(10

5

, n * (n - 1) / 2).

edges[i].length == 3

0 <= A

i

, B

i

< n

A

i

!= B

i

1 <= W

i

<= 10

6

There

may be

multiple edges between a pair of nodes, but they must have unique weights.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int minMaxWeight(int n, int[][] edges, int threshold) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minMaxWeight(n: number, edges: number[][], threshold: number):
number {

};
```

C#:

```
public class Solution {
public int MinMaxWeight(int n, int[][] edges, int threshold) {

}
}
```

C:

```
int minMaxWeight(int n, int** edges, int edgesSize, int* edgesColSize, int
threshold) {

}
```

Go:

```
func minMaxWeight(n int, edges [][]int, threshold int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int minMaxWeight(int n, List<List<int>> edges, int threshold) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec min_max_weight(N :: integer(), Edges :: [[integer()]], Threshold ::
integer()) -> integer().
min_max_weight(N, Edges, Threshold) ->
.
```

Racket:

```
(define/contract (min-max-weight n edges threshold)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimize the Maximum Edge Weight of Graph
* Difficulty: Medium
* Tags: array, graph, search
*
* 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 minMaxWeight(int n, vector<vector<int>>& edges, int threshold) {
}
};
```

Java Solution:

```
/**
* Problem: Minimize the Maximum Edge Weight of Graph
* Difficulty: Medium
* Tags: array, graph, search
*
* 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 minMaxWeight(int n, int[][] edges, int threshold) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimize the Maximum Edge Weight of Graph
Difficulty: Medium
Tags: array, graph, search

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 minMaxWeight(self, n: int, edges: List[List[int]], threshold: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimize the Maximum Edge Weight of Graph
* Difficulty: Medium
* Tags: array, graph, search
*
* 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} threshold
* @return {number}
*/
var minMaxWeight = function(n, edges, threshold) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimize the Maximum Edge Weight of Graph
* Difficulty: Medium
* Tags: array, graph, search
*
* 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 minMaxWeight(n: number, edges: number[][], threshold: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimize the Maximum Edge Weight of Graph
* Difficulty: Medium
* Tags: array, graph, search
*
* 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 MinMaxWeight(int n, int[][] edges, int threshold) {

}
}
```

C Solution:

```
/*
* Problem: Minimize the Maximum Edge Weight of Graph
* Difficulty: Medium
* Tags: array, graph, search
*
* 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
*/

int minMaxWeight(int n, int** edges, int edgesSize, int* edgesColSize, int threshold) {

}
```

Go Solution:

```
// Problem: Minimize the Maximum Edge Weight of Graph
// Difficulty: Medium
// Tags: array, graph, search
//
// 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 minMaxWeight(n int, edges [][]int, threshold int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimize the Maximum Edge Weight of Graph
// Difficulty: Medium
// Tags: array, graph, search
//
// 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 min_max_weight(n: i32, edges: Vec<Vec<i32>>, threshold: i32) -> i32 {
}
}
```

Ruby Solution:

```ruby
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer} threshold
# @return {Integer}
def min_max_weight(n, edges, threshold)

end
```

PHP Solution:

```php
class Solution {

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

}
}
```

Dart Solution:

```dart
class Solution {
int minMaxWeight(int n, List<List<int>> edges, int threshold) {

}
}
```

Scala Solution:

```scala
object Solution {
def minMaxWeight(n: Int, edges: Array[Array[Int]], threshold: Int): Int = {


}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec min_max_weight(N :: integer(), Edges :: [[integer()]], Threshold ::
integer()) -> integer().
min_max_weight(N, Edges, Threshold) ->
.
```

Racket Solution:

```
(define/contract (min-max-weight n edges threshold)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer?)
)
```