Problem 3608: Minimum Time for K Connected Components

Problem Information

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

Problem Description

You are given an integer

n

and an undirected graph with

n

nodes labeled from 0 to

n - 1

. This is represented by a 2D array

edges

, where

edges[i] = [u

i

, v

i

, time

i

]

indicates an undirected edge between nodes

u

i

and

v

i

that can be removed at

time

i

.

You are also given an integer

k

.

Initially, the graph may be connected or disconnected. Your task is to find the

minimum

time

t

such that after removing all edges with

time <= t

, the graph contains

at least

k

connected components.

Return the

minimum

time

t

.

A

connected component

is a subgraph of a graph in which there exists a path between any two vertices, and no vertex of the subgraph shares an edge with a vertex outside of the subgraph.

Example 1:

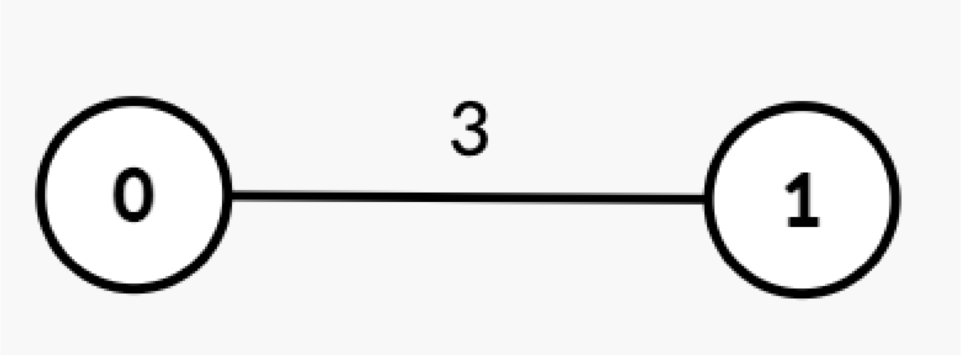
Input:

n = 2, edges = [[0,1,3]], k = 2

Output:

3

Explanation:



Initially, there is one connected component

{0, 1}

.

At

time = 1

or

2

, the graph remains unchanged.

At

time = 3

, edge

[0, 1]

is removed, resulting in

k = 2

connected components

{0}

,

{1}

. Thus, the answer is 3.

Example 2:

Input:

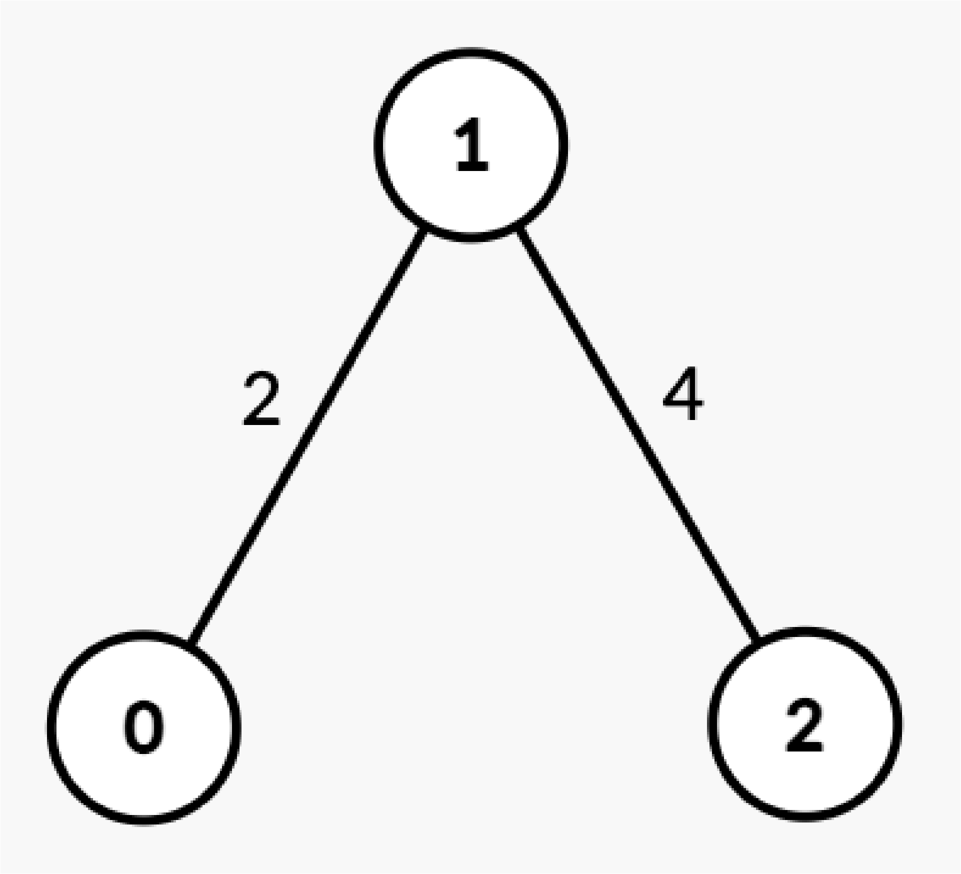
n = 3, edges = [[0,1,2],[1,2,4]], k = 3

Output:

4

Explanation:

,



Initially, there is one connected component

{0, 1, 2}

.

At

time = 2

, edge

[0, 1]

is removed, resulting in two connected components

{0}

,

{1, 2}

.

At

time = 4

, edge

[1, 2]

is removed, resulting in

k = 3

connected components

{0}

,

{1}

,

{2}

. Thus, the answer is 4.

Example 3:

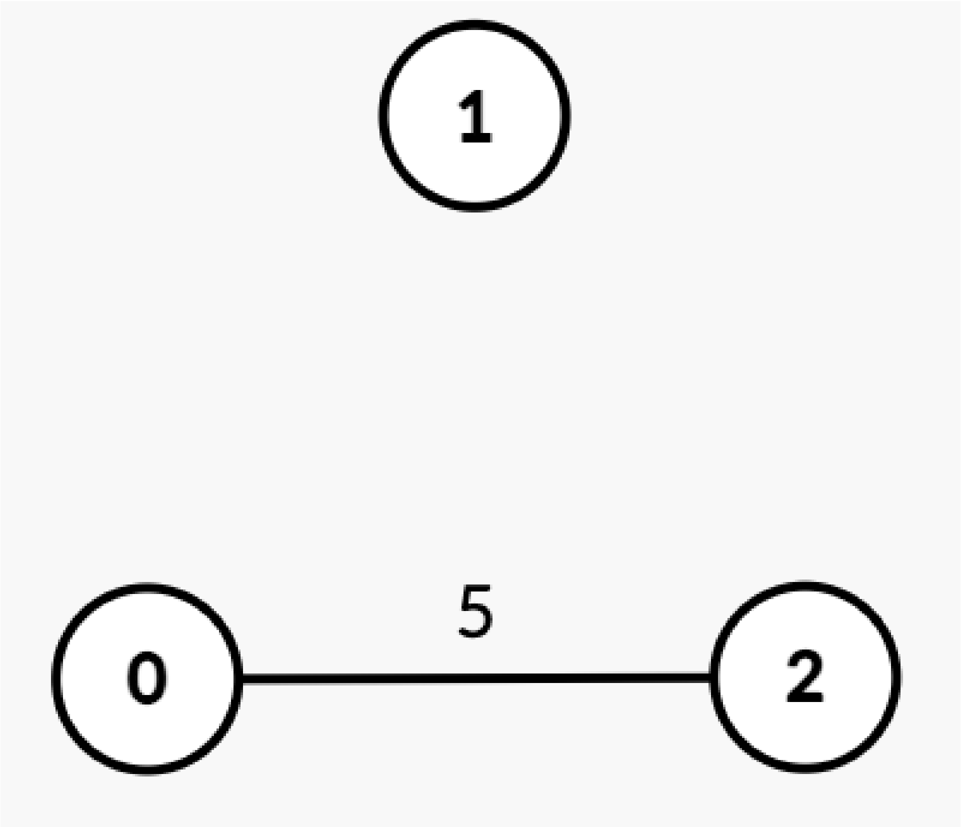
Input:

n = 3, edges = [[0,2,5]], k = 2

Output:

0

Explanation:



Since there are already

k = 2

disconnected components

{1}

,

{0, 2}

, no edge removal is needed. Thus, the answer is 0.

Constraints:

1 <= n <= 10

5

0 <= edges.length <= 10

5

edges[i] = [u

i

, v

i

, time

i

]

0 <= u

i

, v

i

< n

u

i

!= v

i

1 <= time

i

<= 10

9

1 <= k <= n

There are no duplicate edges.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int minTime(int n, int[][] edges, int k) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def minTime(self, n, edges, k):
```

```
"""
:type n: int
:type edges: List[List[int]]
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function minTime(n: number, edges: number[][], k: number): number {

};
```

C#:

```
public class Solution {
public int MinTime(int n, int[][] edges, int k) {

}
}
```

C:

```
int minTime(int n, int** edges, int edgesSize, int* edgesColSize, int k) {

}
```

Go:

```
func minTime(n int, edges [][]int, k int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int minTime(int n, List<List<int>> edges, int k) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec min_time(N :: integer(), Edges :: [[integer()]], K :: integer()) ->
integer().
min_time(N, Edges, K) ->
.
```

Racket:

```
(define/contract (min-time n edges k)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time for K Connected Components
* Difficulty: Medium
* Tags: array, graph, sort, 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 minTime(int n, vector<vector<int>>& edges, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Time for K Connected Components
* Difficulty: Medium
* Tags: array, graph, sort, 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 minTime(int n, int[][] edges, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Time for K Connected Components
Difficulty: Medium
Tags: array, graph, sort, 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 minTime(self, n: int, edges: List[List[int]], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Time for K Connected Components
* Difficulty: Medium
* Tags: array, graph, sort, 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} k
* @return {number}
*/
var minTime = function(n, edges, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time for K Connected Components
* Difficulty: Medium
* Tags: array, graph, sort, 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 minTime(n: number, edges: number[][], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Time for K Connected Components
* Difficulty: Medium
* Tags: array, graph, sort, 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 MinTime(int n, int[][] edges, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Time for K Connected Components
* Difficulty: Medium
* Tags: array, graph, sort, 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 minTime(int n, int** edges, int edgesSize, int* edgesColSize, int k) {

}
```

Go Solution:

```
// Problem: Minimum Time for K Connected Components
// Difficulty: Medium
// Tags: array, graph, sort, 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 minTime(n int, edges [][]int, k int) int {

}
```

Kotlin Solution:

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


}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Time for K Connected Components
// Difficulty: Medium
// Tags: array, graph, sort, 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_time(n: i32, edges: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int minTime(int n, List<List<int>> edges, int k) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec min_time(N :: integer(), Edges :: [[integer()]], K :: integer()) ->
integer().
```

```
min_time(N, Edges, K) ->
.
```

Racket Solution:

```
(define/contract (min-time n edges k)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer?)
)
```