Problem 3613: Minimize Maximum Component Cost

Problem Information

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

Problem Description

You are given an undirected connected graph with

n

nodes labeled from 0 to

n - 1

and a 2D integer array

edges

where

edges[i] = [u

i

, v

i

, w

i

]

denotes an undirected edge between node

u

i

and node

v

i

with weight

w

i

, and an integer

k

.

You are allowed to remove any number of edges from the graph such that the resulting graph has

at most

k

connected components.

The

cost

of a component is defined as the

maximum

edge weight in that component. If a component has no edges, its cost is 0.

Return the

minimum

possible value of the

maximum

cost among all components

after such removals

.

Example 1:

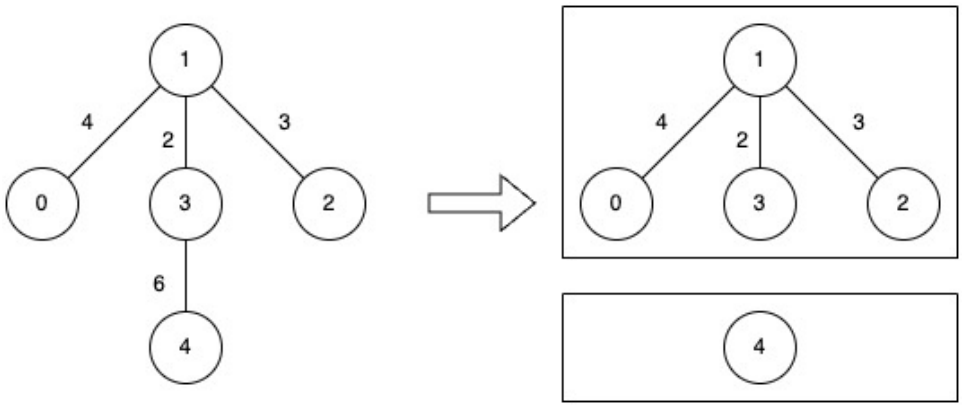
Input:

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

Output:

4

Explanation:



Remove the edge between nodes 3 and 4 (weight 6).

The resulting components have costs of 0 and 4, so the overall maximum cost is 4.

Example 2:

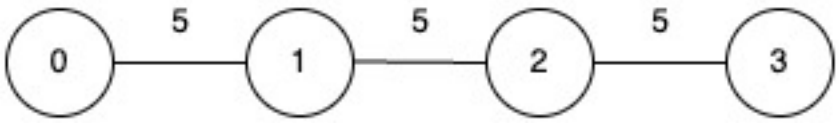
Input:

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

Output:

5

Explanation:



No edge can be removed, since allowing only one component (

k = 1

) requires the graph to stay fully connected.

That single component’s cost equals its largest edge weight, which is 5.

Constraints:

1 <= n <= 5 * 10

4

0 <= edges.length <= 10

5

edges[i].length == 3

0 <= u

i

, v

i

< n

1 <= w

i

<= 10

6

1 <= k <= n

The input graph is connected.

Code Snippets

C++:

```
class Solution {
public:
```

```
int minCost(int n, vector<vector<int>>& edges, int k) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def minCost(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 minCost = function(n, edges, k) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

```
}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimize Maximum Component Cost
* 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 minCost(int n, vector<vector<int>>& edges, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimize Maximum Component Cost
* 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 minCost(int n, int[][] edges, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimize Maximum Component Cost
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 minCost(self, n: int, edges: List[List[int]], k: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimize Maximum Component Cost
* 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 minCost = function(n, edges, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimize Maximum Component Cost
* 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 minCost(n: number, edges: number[][], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimize Maximum Component Cost
* 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 MinCost(int n, int[][] edges, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimize Maximum Component Cost
* 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 minCost(int n, int** edges, int edgesSize, int* edgesColSize, int k) {
```

```
}
```

Go Solution:

```
// Problem: Minimize Maximum Component Cost
// 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 minCost(n int, edges [][]int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimize Maximum Component Cost
// 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_cost(n: i32, edges: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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