Problem 1135: Connecting Cities With Minimum Cost

Problem Information

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

Problem Description

There are

n

cities labeled from

1

to

n

. You are given the integer

n

and an array

connections

where

connections[i] = [x

i

, y

i

, cost

i

]

indicates that the cost of connecting city

x

i

and city

y

i

(bidirectional connection) is

cost

i

.

Return

the minimum

cost

to connect all the

n

cities such that there is at least one path between each pair of cities

. If it is impossible to connect all the

n

cities, return

-1

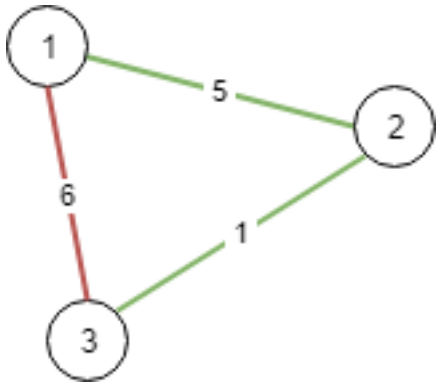
,

The

cost

is the sum of the connections' costs used.

Example 1:



Input:

n = 3, connections = [[1,2,5],[1,3,6],[2,3,1]]

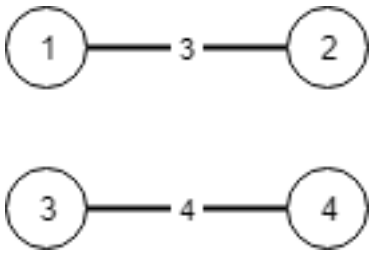
Output:

6

Explanation:

Choosing any 2 edges will connect all cities so we choose the minimum 2.

Example 2:



Input:

n = 4, connections = [[1,2,3],[3,4,4]]

Output:

-1

Explanation:

There is no way to connect all cities even if all edges are used.

Constraints:

1 <= n <= 10

4

1 <= connections.length <= 10

4

connections[i].length == 3

1 <= x

i

, y

i

<= n

x

i

!= y

i

0 <= cost

i

<= 10

5

Code Snippets

C++:

```
class Solution {
public:
int minimumCost(int n, vector<vector<int>>& connections) {

}
};
```

Java:

```
class Solution {
public int minimumCost(int n, int[][] connections) {

}
```

```
}
```

Python3:

```
class Solution:
def minimumCost(self, n: int, connections: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def minimumCost(self, n, connections):
"""
:type n: int
:type connections: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} connections
* @return {number}
*/
var minimumCost = function(n, connections) {

};
```

TypeScript:

```
function minimumCost(n: number, connections: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinimumCost(int n, int[][] connections) {

}
}
```

C:

```
int minimumCost(int n, int** connections, int connectionsSize, int*
connectionsColSize) {

}
```

Go:

```
func minimumCost(n int, connections [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minimumCost(n: Int, connections: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func minimumCost(_ n: Int, _ connections: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_cost(n: i32, connections: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} connections
# @return {Integer}
def minimum_cost(n, connections)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $connections
* @return Integer
*/
function minimumCost($n, $connections) {

}
}
```

Dart:

```
class Solution {
int minimumCost(int n, List<List<int>> connections) {

}
}
```

Scala:

```
object Solution {
def minimumCost(n: Int, connections: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_cost(n :: integer, connections :: [[integer]]) :: integer
def minimum_cost(n, connections) do

end
end
```

Erlang:

```
-spec minimum_cost(N :: integer(), Connections :: [[integer()]]) ->
integer().
minimum_cost(N, Connections) ->
.
```

Racket:

```
(define/contract (minimum-cost n connections)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Connecting Cities With Minimum Cost
* Difficulty: Medium
* Tags: array, tree, graph, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int minimumCost(int n, vector<vector<int>>& connections) {

}
};
```

Java Solution:

```
/**
* Problem: Connecting Cities With Minimum Cost
* Difficulty: Medium
* Tags: array, tree, graph, queue, heap
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int minimumCost(int n, int[][] connections) {

}
}
```

Python3 Solution:

```
"""
Problem: Connecting Cities With Minimum Cost
Difficulty: Medium
Tags: array, tree, graph, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def minimumCost(self, n: int, connections: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumCost(self, n, connections):
"""
:type n: int
:type connections: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Connecting Cities With Minimum Cost
* Difficulty: Medium
* Tags: array, tree, graph, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number} n
* @param {number[][]} connections
* @return {number}
*/
var minimumCost = function(n, connections) {

};
```

TypeScript Solution:

```
/**
* Problem: Connecting Cities With Minimum Cost
* Difficulty: Medium
* Tags: array, tree, graph, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function minimumCost(n: number, connections: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Connecting Cities With Minimum Cost
* Difficulty: Medium
* Tags: array, tree, graph, queue, heap
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int MinimumCost(int n, int[][] connections) {

}
}
```

C Solution:

```
/*
* Problem: Connecting Cities With Minimum Cost
* Difficulty: Medium
* Tags: array, tree, graph, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int minimumCost(int n, int** connections, int connectionsSize, int*
connectionsColSize) {

}
```

Go Solution:

```
// Problem: Connecting Cities With Minimum Cost
// Difficulty: Medium
// Tags: array, tree, graph, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func minimumCost(n int, connections [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumCost(n: Int, connections: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumCost(_ n: Int, _ connections: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Connecting Cities With Minimum Cost
// Difficulty: Medium
// Tags: array, tree, graph, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn minimum_cost(n: i32, connections: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} connections
# @return {Integer}
def minimum_cost(n, connections)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $connections
* @return Integer
*/
function minimumCost($n, $connections) {

}
}
```

Dart Solution:

```
class Solution {
int minimumCost(int n, List<List<int>> connections) {

}
}
```

Scala Solution:

```
object Solution {
def minimumCost(n: Int, connections: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_cost(n :: integer, connections :: [[integer]]) :: integer
def minimum_cost(n, connections) do

end
end
```

Erlang Solution:

```
-spec minimum_cost(N :: integer(), Connections :: [[integer()]]) ->
integer().
minimum_cost(N, Connections) ->
.
```

Racket Solution:

```
(define/contract (minimum-cost n connections)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```