Problem 2492: Minimum Score of a Path Between Two Cities

Problem Information

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

Problem Description

You are given a positive integer

n

representing

n

cities numbered from

1

to

n

. You are also given a

2D

array

roads

where

roads[i] = [a

i

, b

i

, distance

i

]

indicates that there is a

bidirectional

road between cities

a

i

and

b

i

with a distance equal to

distance

i

. The cities graph is not necessarily connected.

The

score

of a path between two cities is defined as the

minimum

distance of a road in this path.

Return

the

minimum

possible score of a path between cities

1

and

n

.

Note

:

A path is a sequence of roads between two cities.

It is allowed for a path to contain the same road

multiple

times, and you can visit cities

1

and

n

multiple times along the path.

The test cases are generated such that there is

at least

one path between

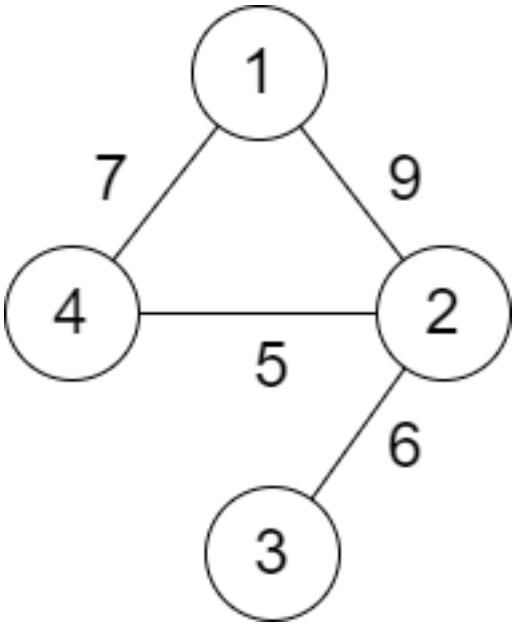
1

and

n

.

Example 1:



Input:

n = 4, roads = [[1,2,9],[2,3,6],[2,4,5],[1,4,7]]

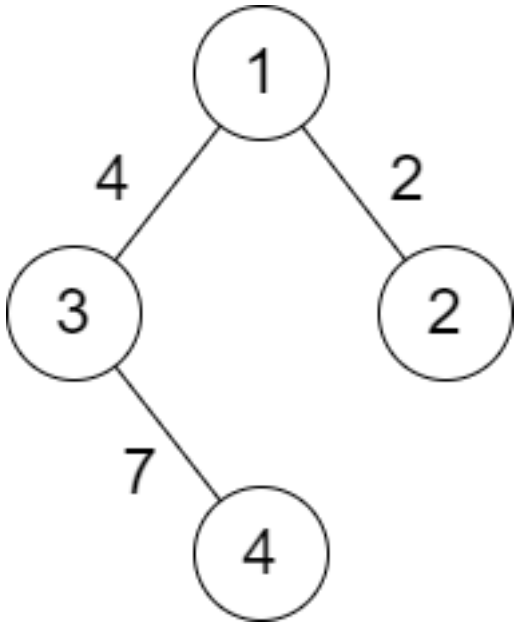
Output:

5

Explanation:

The path from city 1 to 4 with the minimum score is: 1 -> 2 -> 4. The score of this path is min(9,5) = 5. It can be shown that no other path has less score.

Example 2:



Input:

n = 4, roads = [[1,2,2],[1,3,4],[3,4,7]]

Output:

2

Explanation:

The path from city 1 to 4 with the minimum score is: 1 -> 2 -> 1 -> 3 -> 4. The score of this path is min(2,2,4,7) = 2.

Constraints:

2 <= n <= 10

5

1 <= roads.length <= 10

5

roads[i].length == 3

1 <= a

i

, b

i

<= n

a

i

!= b

i

1 <= distance

i

<= 10

4

There are no repeated edges.

There is at least one path between

1

and

n

.

Code Snippets

C++:

```
class Solution {
public:
int minScore(int n, vector<vector<int>>& roads) {

}
};
```

Java:

```
class Solution {
public int minScore(int n, int[][] roads) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def minScore(self, n, roads):
```

```
"""
:type n: int
:type roads: List[List[int]]
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function minScore(n: number, roads: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinScore(int n, int[][] roads) {

}
}
```

C:

```
int minScore(int n, int** roads, int roadsSize, int* roadsColSize) {

}
```

Go:

```
func minScore(n int, roads [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minScore(n: Int, roads: Array<IntArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} roads
# @return {Integer}
def min_score(n, roads)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $roads
* @return Integer
*/
function minScore($n, $roads) {
```

```
}
}
```

Dart:

```
class Solution {
int minScore(int n, List<List<int>> roads) {

}
}
```

Scala:

```
object Solution {
def minScore(n: Int, roads: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_score(n :: integer, roads :: [[integer]]) :: integer
def min_score(n, roads) do

end
end
```

Erlang:

```
-spec min_score(N :: integer(), Roads :: [[integer()]]) -> integer().
min_score(N, Roads) ->
.
```

Racket:

```
(define/contract (min-score n roads)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Score of a Path Between Two Cities
* 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 minScore(int n, vector<vector<int>>& roads) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Score of a Path Between Two Cities
* 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 minScore(int n, int[][] roads) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Score of a Path Between Two Cities
```

```
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 minScore(self, n: int, roads: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Score of a Path Between Two Cities
* 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[][]} roads
* @return {number}
*/
var minScore = function(n, roads) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Minimum Score of a Path Between Two Cities
* 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 minScore(n: number, roads: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Score of a Path Between Two Cities
* 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 MinScore(int n, int[][] roads) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Score of a Path Between Two Cities
```

```
* 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 minScore(int n, int** roads, int roadsSize, int* roadsColSize) {

}
```

Go Solution:

```
// Problem: Minimum Score of a Path Between Two Cities
// 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 minScore(n int, roads [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minScore(n: Int, roads: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Score of a Path Between Two Cities
// 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_score(n: i32, roads: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} roads
# @return {Integer}
def min_score(n, roads)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $roads
* @return Integer
*/
function minScore($n, $roads) {

}
}
```

Dart Solution:

```
class Solution {
int minScore(int n, List<List<int>> roads) {

}
}
```

Scala Solution:

```
object Solution {
def minScore(n: Int, roads: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_score(n :: integer, roads :: [[integer]]) :: integer
def min_score(n, roads) do

end
end
```

Erlang Solution:

```
-spec min_score(N :: integer(), Roads :: [[integer()]]) -> integer().
min_score(N, Roads) ->
.
```

Racket Solution:

```
(define/contract (min-score n roads)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```