Problem 2285: Maximum Total Importance of Roads

Problem Information

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

Problem Description

You are given an integer

n

denoting the number of cities in a country. The cities are numbered from

0

to

n - 1

.

You are also given a 2D integer array

roads

where

roads[i] = [a

i

, b

i

]

denotes that there exists a

bidirectional

road connecting cities

a

i

and

b

i

.

You need to assign each city with an integer value from

1

to

n

, where each value can only be used

once

. The

importance

of a road is then defined as the

sum

of the values of the two cities it connects.

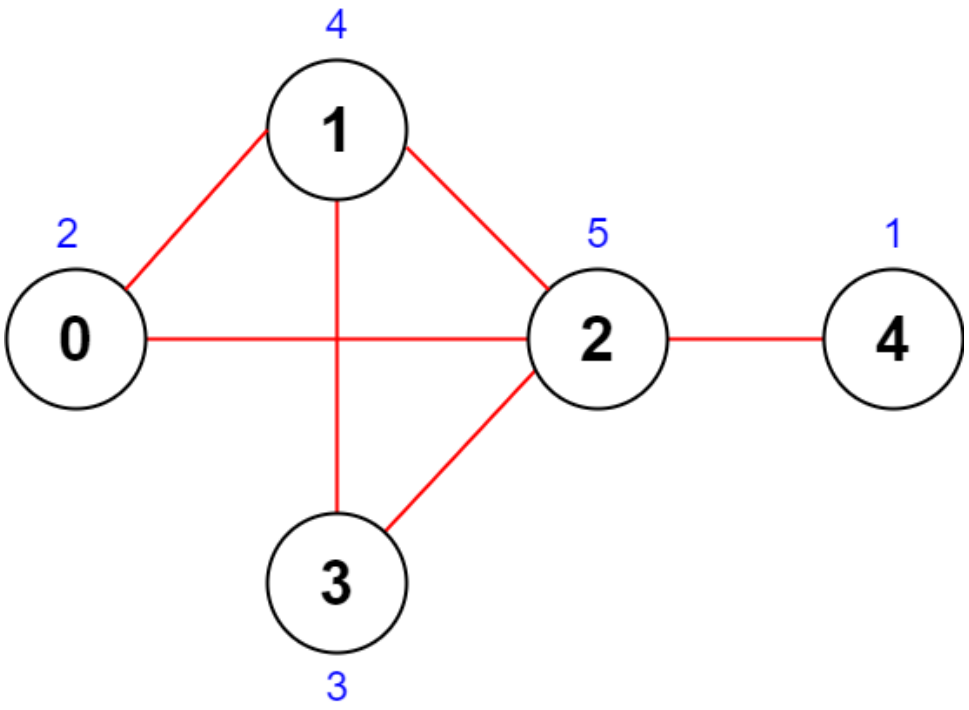
Return

the

maximum total importance

of all roads possible after assigning the values optimally.

Example 1:



Input:

n = 5, roads = [[0,1],[1,2],[2,3],[0,2],[1,3],[2,4]]

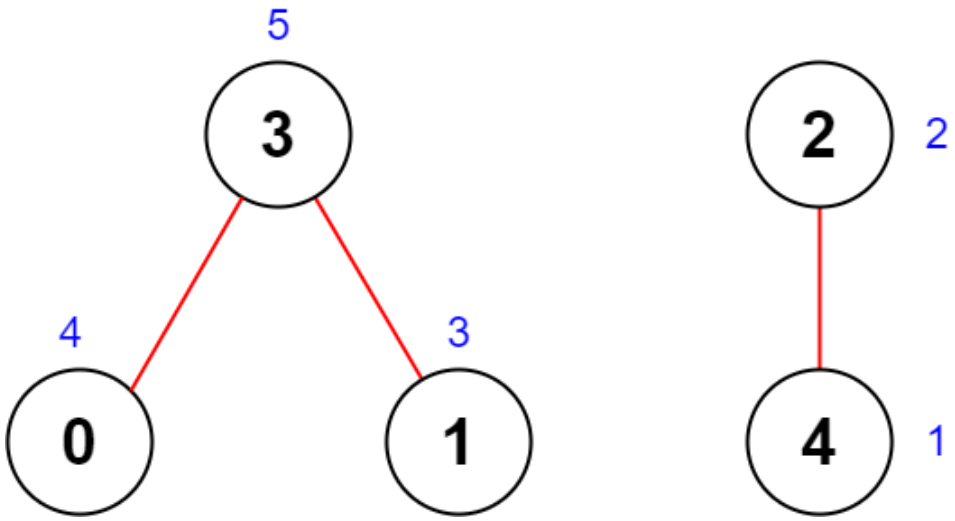
Output:

43

Explanation:

The figure above shows the country and the assigned values of [2,4,5,3,1]. - The road (0,1) has an importance of 2 + 4 = 6. - The road (1,2) has an importance of 4 + 5 = 9. - The road (2,3) has an importance of 5 + 3 = 8. - The road (0,2) has an importance of 2 + 5 = 7. - The road (1,3) has an importance of 4 + 3 = 7. - The road (2,4) has an importance of 5 + 1 = 6. The total importance of all roads is 6 + 9 + 8 + 7 + 7 + 6 = 43. It can be shown that we cannot obtain a greater total importance than 43.

Example 2:



Input:

n = 5, roads = [[0,3],[2,4],[1,3]]

Output:

20

Explanation:

The figure above shows the country and the assigned values of [4,3,2,5,1]. - The road (0,3) has an importance of 4 + 5 = 9. - The road (2,4) has an importance of 2 + 1 = 3. - The road (1,3) has an importance of 3 + 5 = 8. The total importance of all roads is 9 + 3 + 8 = 20. It can be shown that we cannot obtain a greater total importance than 20.

Constraints:

2 <= n <= 5 * 10

4

1 <= roads.length <= 5 * 10

4

roads[i].length == 2

0 <= a

i

, b

i

<= n - 1

a

i

!= b

i

There are no duplicate roads.

Code Snippets

C++:

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

```
}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
long long maximumImportance(int n, int** roads, int roadsSize, int*
roadsColSize) {

}
```

Go:

```
func maximumImportance(n int, roads [][]int) int64 {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn maximum_importance(n: i32, roads: Vec<Vec<i32>>) -> i64 {

}
```

```
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (maximum-importance n roads)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Total Importance of Roads
* Difficulty: Medium
* Tags: array, graph, greedy, sort, queue, heap
*
* 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:
long long maximumImportance(int n, vector<vector<int>>& roads) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Total Importance of Roads
* Difficulty: Medium
* Tags: array, graph, greedy, sort, queue, heap
*
* 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 long maximumImportance(int n, int[][] roads) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Total Importance of Roads
Difficulty: Medium
Tags: array, graph, greedy, sort, queue, heap

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Total Importance of Roads
* Difficulty: Medium
* Tags: array, graph, greedy, sort, queue, heap
*
* 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 maximumImportance = function(n, roads) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Total Importance of Roads
* Difficulty: Medium
* Tags: array, graph, greedy, sort, queue, heap
*
* 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 maximumImportance(n: number, roads: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Total Importance of Roads
* Difficulty: Medium
* Tags: array, graph, greedy, sort, queue, heap
*
* 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 long MaximumImportance(int n, int[][] roads) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Total Importance of Roads
* Difficulty: Medium
* Tags: array, graph, greedy, sort, queue, heap
*
* 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
*/

long long maximumImportance(int n, int** roads, int roadsSize, int*
roadsColSize) {

}
```

Go Solution:

```
// Problem: Maximum Total Importance of Roads
// Difficulty: Medium
// Tags: array, graph, greedy, sort, queue, heap
//
// 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 maximumImportance(n int, roads [][]int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Total Importance of Roads
// Difficulty: Medium
// Tags: array, graph, greedy, sort, queue, heap
//
// 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 maximum_importance(n: i32, roads: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} roads
```

```
# @return {Integer}
def maximum_importance(n, roads)

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (maximum-importance n roads)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```