# Problem 1615: Maximal Network Rank

## Problem Information

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

## Problem Description

There is an infrastructure of

n

cities with some number of

roads

connecting these cities. Each

roads[i] = [a

i

, b

i

]

indicates that there is a bidirectional road between cities

a

i

and

b

i

.

The

network rank

of

two different cities

is defined as the total number of

directly

connected roads to

either

city. If a road is directly connected to both cities, it is only counted

once

.

The

maximal network rank

of the infrastructure is the

maximum network rank

of all pairs of different cities.

Given the integer

n

and the array

roads

, return

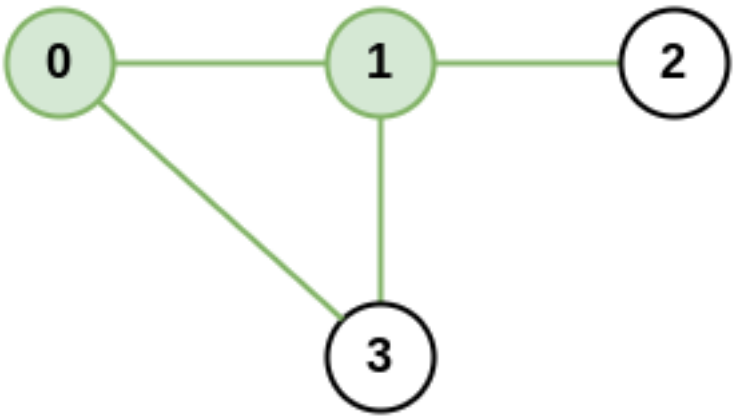
the

maximal network rank

of the entire infrastructure

.

Example 1:




Input:

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

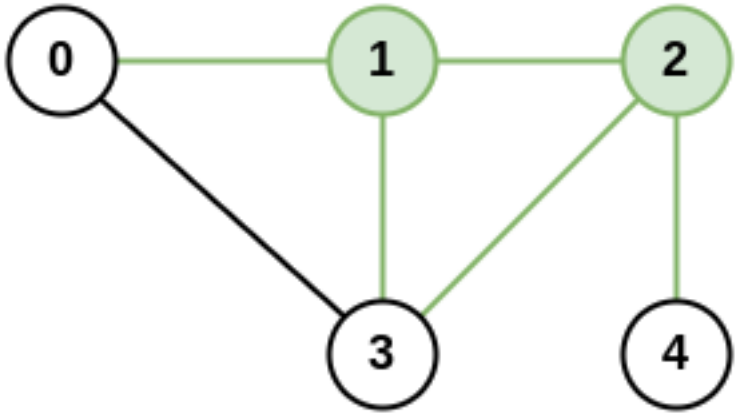
Output:

4

Explanation:

The network rank of cities 0 and 1 is 4 as there are 4 roads that are connected to either 0 or 1. The road between 0 and 1 is only counted once.

Example 2:




Input:

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

Output:

5

Explanation:

There are 5 roads that are connected to cities 1 or 2.

Example 3:

Input:

n = 8, roads = [[0,1],[1,2],[2,3],[2,4],[5,6],[5,7]]

Output:

5

Explanation:

The network rank of 2 and 5 is 5. Notice that all the cities do not have to be connected.

Constraints:

2 <= n <= 100

0 <= roads.length <= n * (n - 1) / 2

roads[i].length == 2

0 <= a

i

, b

i

<= n-1

a

i

!= b

i

Each pair of cities has

at most one

road connecting them.

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

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

};
```

**C#:**

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

}
}
```

**C:**

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

}
```

**Go:**

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

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

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

}
}
```

**Scala:**

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

```
}
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

**Racket:**

```
(define/contract (maximal-network-rank n roads)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Maximal Network Rank
* Difficulty: Medium
* Tags: array, graph
*
* 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 maximalNetworkRank(int n, vector<vector<int>>& roads) {

}
};
```

### Java Solution:

```
/**
* Problem: Maximal Network Rank
* Difficulty: Medium
* Tags: array, graph
*
* 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 maximalNetworkRank(int n, int[][] roads) {

}
}
```

### Python3 Solution:

```
"""
Problem: Maximal Network Rank
Difficulty: Medium
Tags: array, graph

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

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Maximal Network Rank
* Difficulty: Medium
* Tags: array, graph
*
* 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 maximalNetworkRank = function(n, roads) {

};
```

### TypeScript Solution:

```
/**
* Problem: Maximal Network Rank
* Difficulty: Medium
* Tags: array, graph
*
* 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 maximalNetworkRank(n: number, roads: number[][]): number {

};
```

### C# Solution:

```
/*
* Problem: Maximal Network Rank
* Difficulty: Medium
* Tags: array, graph
*
* 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 MaximalNetworkRank(int n, int[][] roads) {

}
}
```

### C Solution:

```
/*
* Problem: Maximal Network Rank
* Difficulty: Medium
* Tags: array, graph
*
* 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 maximalNetworkRank(int n, int** roads, int roadsSize, int* roadsColSize)
{

}
```

### Go Solution:

```
// Problem: Maximal Network Rank
// Difficulty: Medium
// Tags: array, graph
//
// 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 maximalNetworkRank(n int, roads [][]int) int {

}
```

### Kotlin Solution:

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

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Maximal Network Rank
// Difficulty: Medium
// Tags: array, graph
//
// 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 maximal_network_rank(n: i32, roads: Vec<Vec<i32>>) -> i32 {

}
```

```
}
```

### Ruby Solution:

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

end
```

### PHP Solution:

```
class Solution {

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

}
}
```

### Dart Solution:

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

}
}
```

### Scala Solution:

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

}
}
```

### Elixir Solution:

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

end
end
```

### Erlang Solution:

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

### Racket Solution:

```
(define/contract (maximal-network-rank n roads)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```