

# Problem 1782: Count Pairs Of Nodes

## Problem Information

**Difficulty:** Hard

**Acceptance Rate:** 0.00%

**Paid Only:** No

## Problem Description

You are given an undirected graph defined by an integer

$n$

, the number of nodes, and a 2D integer array

edges

, the edges in the graph, where

$\text{edges}[i] = [u$

$i$

$, v$

$i]$

indicates that there is an  
undirected

edge between

u

i

and

v

i

. You are also given an integer array

queries

.

Let

incident(a, b)

be defined as the

number of edges

that are connected to

either

node

a

or

b

.

The answer to the

j

th

query is the

number of pairs

of nodes

(a, b)

that satisfy

both

of the following conditions:

$a < b$

$\text{incident}(a, b) > \text{queries}[j]$

Return

an array

answers

such that

$\text{answers.length} == \text{queries.length}$

and

$\text{answers}[j]$

is the answer of the

j

th

query

Note that there can be

multiple edges

between the same two nodes.

Example 1:

Node Pair	incident(a, b)	Edges
(1, 2)	5	[1,2], [2,4], [1,3], [2,3], [2,1]
(1, 3)	4	[1,2], [1,3], [2,3], [2,1]
(1, 4)	4	[1,2], [2,4], [1,3], [2,1]
(2, 3)	5	[1,2], [2,4], [1,3], [2,3], [2,1]
(2, 4)	4	[1,2], [2,4], [2,3], [2,1]
(3, 4)	3	[2,4], [1,3], [2,3]

Input:

n = 4, edges = [[1,2],[2,4],[1,3],[2,3],[2,1]], queries = [2,3]

Output:

[6,5]

Explanation:

The calculations for  $\text{incident}(a, b)$  are shown in the table above. The answers for each of the queries are as follows: -  $\text{answers}[0] = 6$ . All the pairs have an  $\text{incident}(a, b)$  value greater than 2. -  $\text{answers}[1] = 5$ . All the pairs except (3, 4) have an  $\text{incident}(a, b)$  value greater than 3.

Example 2:

Input:

$n = 5$ , edges = [[1,5],[1,5],[3,4],[2,5],[1,3],[5,1],[2,3],[2,5]], queries = [1,2,3,4,5]

Output:

[10,10,9,8,6]

Constraints:

$2 \leq n \leq 2 * 10$

4

$1 \leq \text{edges.length} \leq 10$

5

$1 \leq u$

i

, v

i

$\leq n$

u

i

$\neq v$

i

$1 \leq \text{queries.length} \leq 20$

$0 \leq \text{queries}[j] < \text{edges.length}$

## Code Snippets

### C++:

```
class Solution {  
public:  
vector<int> countPairs(int n, vector<vector<int>>& edges, vector<int>&  
queries) {  
  
}  
};
```

### Java:

```
class Solution {  
public int[] countPairs(int n, int[][] edges, int[] queries) {  
  
}  
}
```

### Python3:

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

### Python:

```
class Solution(object):  
def countPairs(self, n, edges, queries):  
"""  
:type n: int  
:type edges: List[List[int]]  
:type queries: List[int]
```

```
:rtype: List[int]
"""

```

### JavaScript:

```
/**
 * @param {number} n
 * @param {number[][]} edges
 * @param {number[]} queries
 * @return {number[]}
 */
var countPairs = function(n, edges, queries) {
};


```

### TypeScript:

```
function countPairs(n: number, edges: number[][], queries: number[]): number[] {
};


```

### C#:

```
public class Solution {
    public int[] CountPairs(int n, int[][] edges, int[] queries) {
        }
}
```

### C:

```
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* countPairs(int n, int** edges, int edgesSize, int* edgesColSize, int*
queries, int queriesSize, int* returnSize) {
}


```

### Go:

```
func countPairs(n int, edges [][]int, queries []int) []int {  
}  
}
```

### Kotlin:

```
class Solution {  
    fun countPairs(n: Int, edges: Array<IntArray>, queries: IntArray): IntArray {  
        }  
    }  
}
```

### Swift:

```
class Solution {  
    func countPairs(_ n: Int, _ edges: [[Int]], _ queries: [Int]) -> [Int] {  
        }  
    }  
}
```

### Rust:

```
impl Solution {  
    pub fn count_pairs(n: i32, edges: Vec<Vec<i32>>, queries: Vec<i32>) ->  
    Vec<i32> {  
        }  
    }  
}
```

### Ruby:

```
# @param {Integer} n  
# @param {Integer[][]} edges  
# @param {Integer[]} queries  
# @return {Integer[]}  
def count_pairs(n, edges, queries)  
  
end
```

### PHP:

```
class Solution {
```

```

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

}
}

```

### Dart:

```

class Solution {
List<int> countPairs(int n, List<List<int>> edges, List<int> queries) {

}
}

```

### Scala:

```

object Solution {
def countPairs(n: Int, edges: Array[Array[Int]], queries: Array[Int]): Array[Int] = {

}
}

```

### Elixir:

```

defmodule Solution do
@spec count_pairs(n :: integer, edges :: [[integer]], queries :: [integer])
:: [integer]
def count_pairs(n, edges, queries) do
end
end

```

### Erlang:

```

-spec count_pairs(N :: integer(), Edges :: [[integer()]], Queries :: [integer()]) -> [integer()].

```

```
count_pairs(N, Edges, Queries) ->
.
```

## Racket:

```
(define/contract (count-pairs n edges queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
(listof exact-integer?))
)
```

# Solutions

## C++ Solution:

```
/*
 * Problem: Count Pairs Of Nodes
 * Difficulty: Hard
 * Tags: array, graph, hash, sort, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

class Solution {
public:
vector<int> countPairs(int n, vector<vector<int>>& edges, vector<int>&
queries) {

}
};
```

## Java Solution:

```
/**
 * Problem: Count Pairs Of Nodes
 * Difficulty: Hard
 * Tags: array, graph, hash, sort, search
 *
 * Approach: Use two pointers or sliding window technique
```

```

* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

```

```

class Solution {
public int[] countPairs(int n, int[][] edges, int[] queries) {
}
}

```

### Python3 Solution:

```

"""
Problem: Count Pairs Of Nodes
Difficulty: Hard
Tags: array, graph, hash, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
    def countPairs(self, n: int, edges: List[List[int]], queries: List[int]) -> List[int]:
        # TODO: Implement optimized solution
        pass

```

### Python Solution:

```

class Solution(object):
    def countPairs(self, n, edges, queries):
        """
        :type n: int
        :type edges: List[List[int]]
        :type queries: List[int]
        :rtype: List[int]
        """

```

### JavaScript Solution:

```

    /**
 * Problem: Count Pairs Of Nodes
 * Difficulty: Hard
 * Tags: array, graph, hash, sort, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

    /**
 * @param {number} n
 * @param {number[][]} edges
 * @param {number[]} queries
 * @return {number[]}
 */
var countPairs = function(n, edges, queries) {

};


```

### TypeScript Solution:

```

    /**
 * Problem: Count Pairs Of Nodes
 * Difficulty: Hard
 * Tags: array, graph, hash, sort, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

function countPairs(n: number, edges: number[][][], queries: number[]): number[] {

};


```

### C# Solution:

```

/*
 * Problem: Count Pairs Of Nodes
 * Difficulty: Hard

```

```

* Tags: array, graph, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
public class Solution {
public int[] CountPairs(int n, int[][] edges, int[] queries) {
}
}

```

### C Solution:

```

/*
* Problem: Count Pairs Of Nodes
* Difficulty: Hard
* Tags: array, graph, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
/***
* Note: The returned array must be malloced, assume caller calls free().
*/
int* countPairs(int n, int** edges, int edgesSize, int* edgesColSize, int*
queries, int queriesSize, int* returnSize) {

}

```

### Go Solution:

```

// Problem: Count Pairs Of Nodes
// Difficulty: Hard
// Tags: array, graph, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)

```

```

// Space Complexity: O(n) for hash map

func countPairs(n int, edges [][]int, queries []int) []int {
}

```

### Kotlin Solution:

```

class Solution {
    fun countPairs(n: Int, edges: Array<IntArray>, queries: IntArray): IntArray {
        }
    }
}

```

### Swift Solution:

```

class Solution {
    func countPairs(_ n: Int, _ edges: [[Int]], _ queries: [Int]) -> [Int] {
        }
    }
}

```

### Rust Solution:

```

// Problem: Count Pairs Of Nodes
// Difficulty: Hard
// Tags: array, graph, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
    pub fn count_pairs(n: i32, edges: Vec<Vec<i32>>, queries: Vec<i32>) -> Vec<i32> {
        }
    }
}

```

### Ruby Solution:

```

# @param {Integer} n
# @param {Integer[][][]} edges
# @param {Integer[]} queries
# @return {Integer[]}
def count_pairs(n, edges, queries)

end

```

### PHP Solution:

```

class Solution {

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

    }
}

```

### Dart Solution:

```

class Solution {
List<int> countPairs(int n, List<List<int>> edges, List<int> queries) {
}
}

```

### Scala Solution:

```

object Solution {
def countPairs(n: Int, edges: Array[Array[Int]], queries: Array[Int]): Array[Int] = {
}
}

```

### Elixir Solution:

```
defmodule Solution do
@spec count_pairs(n :: integer, edges :: [[integer]], queries :: [integer])
:: [integer]
def count_pairs(n, edges, queries) do
end
end
```

### Erlang Solution:

```
-spec count_pairs(N :: integer(), Edges :: [[integer()]], Queries :: [integer()]) -> [integer()].
count_pairs(N, Edges, Queries) ->
.
```

### Racket Solution:

```
(define/contract (count-pairs n edges queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
(listof exact-integer?))
)
```