Problem 3244: Shortest Distance After Road Addition Queries II

Problem Information

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

Problem Description

You are given an integer

n

and a 2D integer array

queries

.

There are

n

cities numbered from

0

to

n - 1

. Initially, there is a

unidirectional

road from city

i

to city

i + 1

for all

0 <= i < n - 1

.

queries[i] = [u

i

, v

i

]

represents the addition of a new

unidirectional

road from city

u

i

to city

v

i

. After each query, you need to find the

length

of the

shortest path

from city

0

to city

n - 1

.

There are no two queries such that

queries[i][0] < queries[j][0] < queries[i][1] < queries[j][1]

.

Return an array

answer

where for each

i

in the range

[0, queries.length - 1]

,

answer[i]

is the

length of the shortest path

from city

0

to city

n - 1

after processing the

first

i + 1

queries.

Example 1:

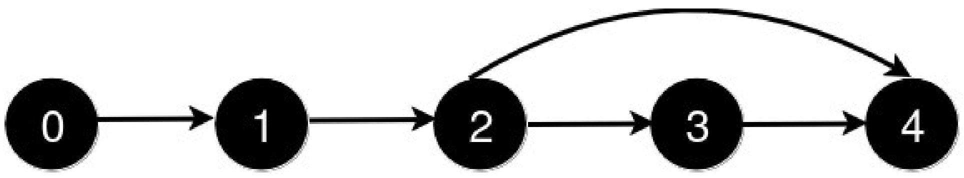
Input:

n = 5, queries = [[2,4],[0,2],[0,4]]

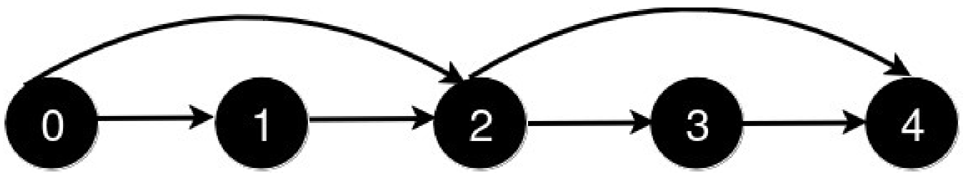
Output:

[3,2,1]

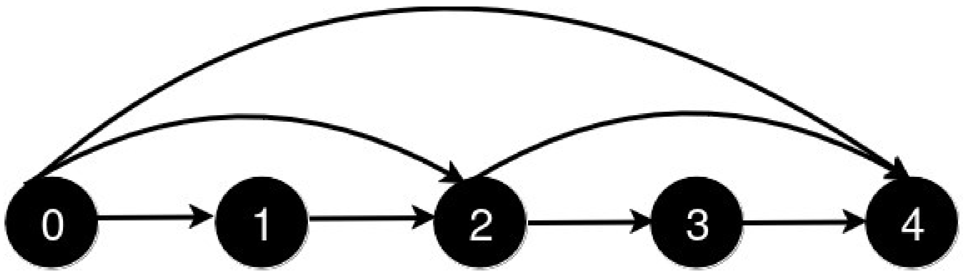
Explanation:



After the addition of the road from 2 to 4, the length of the shortest path from 0 to 4 is 3.



After the addition of the road from 0 to 2, the length of the shortest path from 0 to 4 is 2.



After the addition of the road from 0 to 4, the length of the shortest path from 0 to 4 is 1.

Example 2:

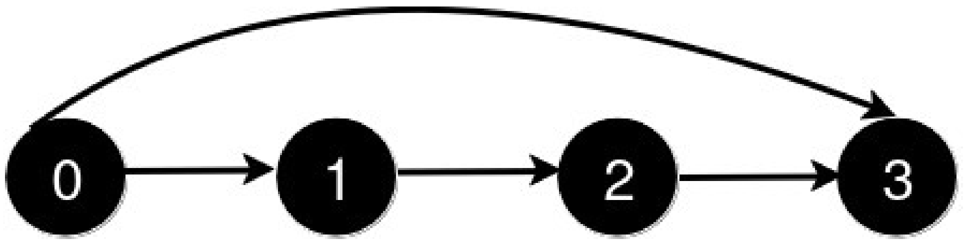
Input:

n = 4, queries = [[0,3],[0,2]]

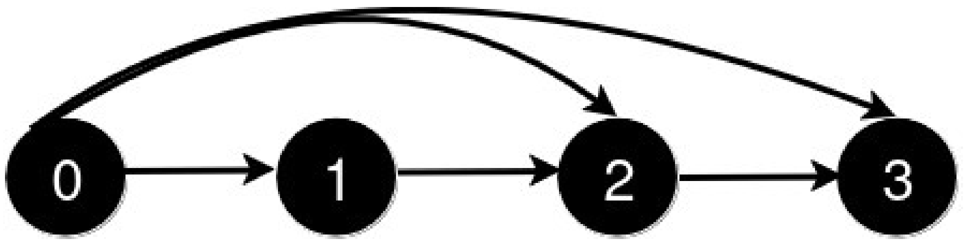
Output:

[1,1]

Explanation:



After the addition of the road from 0 to 3, the length of the shortest path from 0 to 3 is 1.



After the addition of the road from 0 to 2, the length of the shortest path remains 1.

Constraints:

3 <= n <= 10

5

1 <= queries.length <= 10

5

queries[i].length == 2

0 <= queries[i][0] < queries[i][1] < n

1 < queries[i][1] - queries[i][0]

There are no repeated roads among the queries.

There are no two queries such that

i != j

and

queries[i][0] < queries[j][0] < queries[i][1] < queries[j][1]

.

Code Snippets

C++:

```
class Solution {
public:
vector<int> shortestDistanceAfterQueries(int n, vector<vector<int>>& queries)
{

}
};
```

Java:

```
class Solution {
public int[] shortestDistanceAfterQueries(int n, int[][] queries) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function shortestDistanceAfterQueries(n: number, queries: number[][]):
number[] {

};
```

C#:

```
public class Solution {
public int[] ShortestDistanceAfterQueries(int n, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* shortestDistanceAfterQueries(int n, int** queries, int queriesSize, int*
queriesColSize, int* returnSize) {

}
```

Go:

```
func shortestDistanceAfterQueries(n int, queries [][]int) []int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<int> shortestDistanceAfterQueries(int n, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def shortestDistanceAfterQueries(n: Int, queries: Array[Array[Int]]):
Array[Int] = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (shortest-distance-after-queries n queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Distance After Road Addition Queries II
* Difficulty: Hard
* Tags: array, graph, greedy
*
* 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:
vector<int> shortestDistanceAfterQueries(int n, vector<vector<int>>& queries)
{

}
};
```

Java Solution:

```
/**
* Problem: Shortest Distance After Road Addition Queries II
* Difficulty: Hard
* Tags: array, graph, greedy
*
* 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[] shortestDistanceAfterQueries(int n, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest Distance After Road Addition Queries II
Difficulty: Hard
Tags: array, graph, greedy
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shortest Distance After Road Addition Queries II
* Difficulty: Hard
* Tags: array, graph, greedy
*
* 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[][]} queries
* @return {number[]}
*/
var shortestDistanceAfterQueries = function(n, queries) {
};
```

TypeScript Solution:

```
/**
* Problem: Shortest Distance After Road Addition Queries II
* Difficulty: Hard
* Tags: array, graph, greedy
*
* 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 shortestDistanceAfterQueries(n: number, queries: number[][]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Shortest Distance After Road Addition Queries II
* Difficulty: Hard
* Tags: array, graph, greedy
*
* 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[] ShortestDistanceAfterQueries(int n, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Distance After Road Addition Queries II
* Difficulty: Hard
* Tags: array, graph, greedy
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* shortestDistanceAfterQueries(int n, int** queries, int queriesSize, int*
queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Shortest Distance After Road Addition Queries II
// Difficulty: Hard
// Tags: array, graph, greedy
//
// 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 shortestDistanceAfterQueries(n int, queries [][]int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Shortest Distance After Road Addition Queries II
// Difficulty: Hard
// Tags: array, graph, greedy
//
// 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 shortest_distance_after_queries(n: i32, queries: Vec<Vec<i32>>) ->
Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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


}
}
```

Dart Solution:

```
class Solution {
List<int> shortestDistanceAfterQueries(int n, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def shortestDistanceAfterQueries(n: Int, queries: Array[Array[Int]]):
Array[Int] = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (shortest-distance-after-queries n queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```