

Problem 3017: Count the Number of Houses at a Certain Distance II

Problem Information

Difficulty: **Hard**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given three

positive

integers

n

,

x

, and

y

.

In a city, there exist houses numbered

1

to

n

connected by

n

streets. There is a street connecting the house numbered

i

with the house numbered

$i + 1$

for all

$1 \leq i \leq n - 1$

. An additional street connects the house numbered

x

with the house numbered

y

.

For each

k

, such that

$1 \leq k \leq n$

, you need to find the number of

pairs of houses

(house

1

, house

2

)

such that the

minimum

number of streets that need to be traveled to reach

house

2

from

house

1

is

k

.

Return

a

1-indexed

array

result

of length

n

where

result[k]

represents the

total

number of pairs of houses such that the

minimum

streets required to reach one house from the other is

k

.

Note

that

x

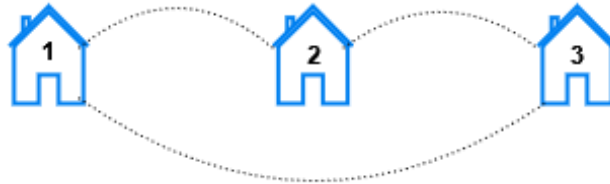
and

y

can be

equal

Example 1:



Input:

$n = 3, x = 1, y = 3$

Output:

[6,0,0]

Explanation:

Let's look at each pair of houses: - For the pair (1, 2), we can go from house 1 to house 2 directly. - For the pair (2, 1), we can go from house 2 to house 1 directly. - For the pair (1, 3), we can go from house 1 to house 3 directly. - For the pair (3, 1), we can go from house 3 to house 1 directly. - For the pair (2, 3), we can go from house 2 to house 3 directly. - For the pair (3, 2), we can go from house 3 to house 2 directly.

Example 2:



Input:

$n = 5, x = 2, y = 4$

Output:

[10,8,2,0,0]

Explanation:

For each distance k the pairs are: - For $k == 1$, the pairs are (1, 2), (2, 1), (2, 3), (3, 2), (2, 4), (4, 2), (3, 4), (4, 3), (4, 5), and (5, 4). - For $k == 2$, the pairs are (1, 3), (3, 1), (1, 4), (4, 1), (2, 5), (5, 2), (3, 5), and (5, 3). - For $k == 3$, the pairs are (1, 5), and (5, 1). - For $k == 4$ and $k == 5$, there are no pairs.

Example 3:



Input:

$n = 4, x = 1, y = 1$

Output:

[6,4,2,0]

Explanation:

For each distance k the pairs are: - For $k == 1$, the pairs are (1, 2), (2, 1), (2, 3), (3, 2), (3, 4), and (4, 3). - For $k == 2$, the pairs are (1, 3), (3, 1), (2, 4), and (4, 2). - For $k == 3$, the pairs are (1, 4), and (4, 1). - For $k == 4$, there are no pairs.

Constraints:

$2 \leq n \leq 10$

$1 \leq x, y \leq n$

Code Snippets

C++:

```
class Solution {
public:
    vector<long long> countOfPairs(int n, int x, int y) {

    }
};
```

Java:

```
class Solution {
    public long[] countOfPairs(int n, int x, int y) {

    }
}
```

Python3:

```
class Solution:
    def countOfPairs(self, n: int, x: int, y: int) -> List[int]:
```

Python:

```
class Solution(object):
    def countOfPairs(self, n, x, y):
        """
        :type n: int
        :type x: int
        :type y: int
        :rtype: List[int]
        """
```

JavaScript:

```
/**
 * @param {number} n
```

```

* @param {number} x
* @param {number} y
* @return {number[]}
*/
var countOfPairs = function(n, x, y) {

};

```

TypeScript:

```

function countOfPairs(n: number, x: number, y: number): number[] {

};

```

C#:

```

public class Solution {
    public long[] CountOfPairs(int n, int x, int y) {

    }
}

```

C:

```

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
long long* countOfPairs(int n, int x, int y, int* returnSize) {

}

```

Go:

```

func countOfPairs(n int, x int, y int) []int64 {

}

```

Kotlin:

```

class Solution {
    fun countOfPairs(n: Int, x: Int, y: Int): LongArray {

```



```
}  
}
```

Swift:

```
class Solution {  
    func countOfPairs(_ n: Int, _ x: Int, _ y: Int) -> [Int] {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn count_of_pairs(n: i32, x: i32, y: i32) -> Vec<i64> {  
  
    }  
}
```

Ruby:

```
# @param {Integer} n  
# @param {Integer} x  
# @param {Integer} y  
# @return {Integer[]}  
def count_of_pairs(n, x, y)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer $n  
     * @param Integer $x  
     * @param Integer $y  
     * @return Integer[]  
     */  
    function countOfPairs($n, $x, $y) {  
  
    }  
}
```

```
}
```

Dart:

```
class Solution {  
  List<int> countOfPairs(int n, int x, int y) {  
  
  }  
}
```

Scala:

```
object Solution {  
  def countOfPairs(n: Int, x: Int, y: Int): Array[Long] = {  
  
  }  
}
```

Elixir:

```
defmodule Solution do  
  @spec count_of_pairs(n :: integer, x :: integer, y :: integer) :: [integer]  
  def count_of_pairs(n, x, y) do  
  
  end  
end
```

Erlang:

```
-spec count_of_pairs(N :: integer(), X :: integer(), Y :: integer()) ->  
[integer()].  
count_of_pairs(N, X, Y) ->  
.
```

Racket:

```
(define/contract (count-of-pairs n x y)  
  (-> exact-integer? exact-integer? exact-integer? (listof exact-integer?))  
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Count the Number of Houses at a Certain Distance II
 * Difficulty: Hard
 * Tags: array, tree, graph
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

class Solution {
public:
    vector<long long> countOfPairs(int n, int x, int y) {

    }
};
```

Java Solution:

```
/**
 * Problem: Count the Number of Houses at a Certain Distance II
 * Difficulty: Hard
 * Tags: array, tree, graph
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

class Solution {
    public long[] countOfPairs(int n, int x, int y) {

    }
}
```

Python3 Solution:

```
"""
Problem: Count the Number of Houses at a Certain Distance II
```

Difficulty: Hard

Tags: array, tree, graph

Approach: Use two pointers or sliding window technique

Time Complexity: $O(n)$ or $O(n \log n)$

Space Complexity: $O(h)$ for recursion stack where h is height

"""

```
class Solution:
```

```
def countOfPairs(self, n: int, x: int, y: int) -> List[int]:
```

```
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
```

```
def countOfPairs(self, n, x, y):
```

```
"""
```

```
:type n: int
```

```
:type x: int
```

```
:type y: int
```

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

```
"""
```

JavaScript Solution:

```
/**
```

```
 * Problem: Count the Number of Houses at a Certain Distance II
```

```
 * Difficulty: Hard
```

```
 * Tags: array, tree, graph
```

```
 *
```

```
 * Approach: Use two pointers or sliding window technique
```

```
 * Time Complexity:  $O(n)$  or  $O(n \log n)$ 
```

```
 * Space Complexity:  $O(h)$  for recursion stack where  $h$  is height
```

```
 */
```

```
/**
```

```
 * @param {number} n
```

```
 * @param {number} x
```

```
 * @param {number} y
```

```
 * @return {number[]}
```

```

*/
var countOfPairs = function(n, x, y) {

};

```

TypeScript Solution:

```

/**
 * Problem: Count the Number of Houses at a Certain Distance II
 * Difficulty: Hard
 * Tags: array, tree, graph
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

function countOfPairs(n: number, x: number, y: number): number[] {

};

```

C# Solution:

```

/*
 * Problem: Count the Number of Houses at a Certain Distance II
 * Difficulty: Hard
 * Tags: array, tree, graph
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

public class Solution {
    public long[] CountOfPairs(int n, int x, int y) {

    }
}

```

C Solution:

```

/*
 * Problem: Count the Number of Houses at a Certain Distance II
 * Difficulty: Hard
 * Tags: array, tree, graph
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
long long* countOfPairs(int n, int x, int y, int* returnSize) {

}

```

Go Solution:

```

// Problem: Count the Number of Houses at a Certain Distance II
// Difficulty: Hard
// Tags: array, tree, graph
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func countOfPairs(n int, x int, y int) []int64 {

}

```

Kotlin Solution:

```

class Solution {
    fun countOfPairs(n: Int, x: Int, y: Int): LongArray {

    }
}

```

Swift Solution:

```

class Solution {
    func countOfPairs(_ n: Int, _ x: Int, _ y: Int) -> [Int] {

    }
}

```

Rust Solution:

```

// Problem: Count the Number of Houses at a Certain Distance II
// Difficulty: Hard
// Tags: array, tree, graph
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
    pub fn count_of_pairs(n: i32, x: i32, y: i32) -> Vec<i64> {

    }
}

```

Ruby Solution:

```

# @param {Integer} n
# @param {Integer} x
# @param {Integer} y
# @return {Integer[]}
def count_of_pairs(n, x, y)

end

```

PHP Solution:

```

class Solution {

    /**
     * @param Integer $n
     * @param Integer $x
     * @param Integer $y
     * @return Integer[]
     */
}

```

```
function countOfPairs($n, $x, $y) {

}

}
```

Dart Solution:

```
class Solution {
  List<int> countOfPairs(int n, int x, int y) {

  }
}
```

Scala Solution:

```
object Solution {
  def countOfPairs(n: Int, x: Int, y: Int): Array[Long] = {

  }
}
```

Elixir Solution:

```
defmodule Solution do
  @spec count_of_pairs(n :: integer, x :: integer, y :: integer) :: [integer]
  def count_of_pairs(n, x, y) do

  end
end
```

Erlang Solution:

```
-spec count_of_pairs(N :: integer(), X :: integer(), Y :: integer()) ->
[integer()].
count_of_pairs(N, X, Y) ->
.
```

Racket Solution:

```
(define/contract (count-of-pairs n x y)
  (-> exact-integer? exact-integer? exact-integer? (listof exact-integer?)))
```