Problem 573: Squirrel Simulation

Problem Information

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

Problem Description

You are given two integers

height

and

width

representing a garden of size

height x width

. You are also given:

an array

tree

where

tree = [tree

r

, tree

c

]

is the position of the tree in the garden,

an array

squirrel

where

squirrel = [squirrel

r

, squirrel

c

]

is the position of the squirrel in the garden,

and an array

nuts

where

nuts[i] = [nut

i

r

, nut

i

c

]

is the position of the

i

th

nut in the garden.

The squirrel can only take at most one nut at one time and can move in four directions: up, down, left, and right, to the adjacent cell.

Return

the

minimal distance

for the squirrel to collect all the nuts and put them under the tree one by one

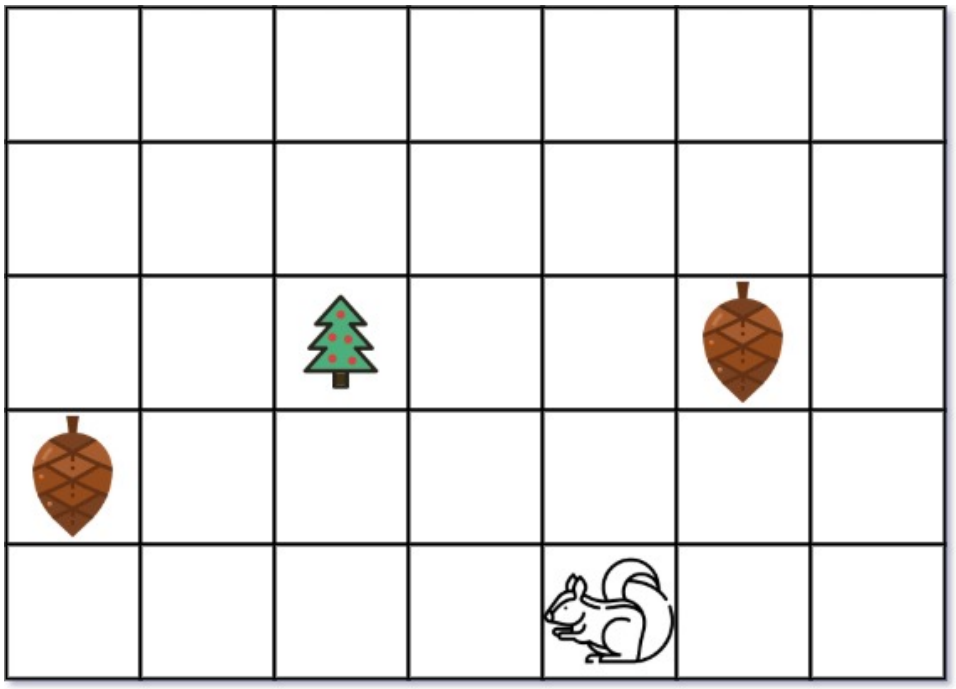
.

The

distance

is the number of moves.

Example 1:



Input:

height = 5, width = 7, tree = [2,2], squirrel = [4,4], nuts = [[3,0], [2,5]]

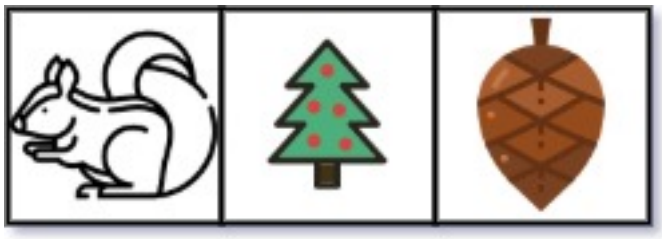
Output:

12

Explanation:

The squirrel should go to the nut at [2, 5] first to achieve a minimal distance.

Example 2:



Input:

height = 1, width = 3, tree = [0,1], squirrel = [0,0], nuts = [[0,2]]

Output:

3

Constraints:

1 <= height, width <= 100

tree.length == 2

squirrel.length == 2

1 <= nuts.length <= 5000

nuts[i].length == 2

0 <= tree

r

, squirrel

r

, nut

i

r

<= height

0 <= tree

c

, squirrel

c

, nut

i

c

<= width

Code Snippets

C++:

```
class Solution {
public:
int minDistance(int height, int width, vector<int>& tree, vector<int>&
squirrel, vector<vector<int>>& nuts) {

}
};
```

Java:

```
class Solution {
public int minDistance(int height, int width, int[] tree, int[] squirrel,
int[][] nuts) {

}
}
```

Python3:

```
class Solution:
def minDistance(self, height: int, width: int, tree: List[int], squirrel:
List[int], nuts: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def minDistance(self, height, width, tree, squirrel, nuts):
"""
:type height: int
:type width: int
:type tree: List[int]
:type squirrel: List[int]
:type nuts: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} height
* @param {number} width
* @param {number[]} tree
* @param {number[]} squirrel
* @param {number[][]} nuts
* @return {number}
*/
var minDistance = function(height, width, tree, squirrel, nuts) {

};
```

TypeScript:

```
function minDistance(height: number, width: number, tree: number[], squirrel:
number[], nuts: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinDistance(int height, int width, int[] tree, int[] squirrel,
int[][] nuts) {

}
}
```

C:

```
int minDistance(int height, int width, int* tree, int treeSize, int*
squirrel, int squirrelSize, int** nuts, int nutsSize, int* nutsColSize) {

}
```

Go:

```
func minDistance(height int, width int, tree []int, squirrel []int, nuts
[][]int) int {

}
```

Kotlin:

```
class Solution {
fun minDistance(height: Int, width: Int, tree: IntArray, squirrel: IntArray,
nuts: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func minDistance(_ height: Int, _ width: Int, _ tree: [Int], _ squirrel:
[Int], _ nuts: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_distance(height: i32, width: i32, tree: Vec<i32>, squirrel:
Vec<i32>, nuts: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} height
# @param {Integer} width
```

```
# @param {Integer[]} tree
# @param {Integer[]} squirrel
# @param {Integer[][]} nuts
# @return {Integer}
def min_distance(height, width, tree, squirrel, nuts)

end
```

PHP:

```
class Solution {

/**
* @param Integer $height
* @param Integer $width
* @param Integer[] $tree
* @param Integer[] $squirrel
* @param Integer[][] $nuts
* @return Integer
*/
function minDistance($height, $width, $tree, $squirrel, $nuts) {

}
}
```

Dart:

```
class Solution {
int minDistance(int height, int width, List<int> tree, List<int> squirrel,
List<List<int>> nuts) {

}
}
```

Scala:

```
object Solution {
def minDistance(height: Int, width: Int, tree: Array[Int], squirrel:
Array[Int], nuts: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_distance(height :: integer, width :: integer, tree :: [integer],
squirrel :: [integer], nuts :: [[integer]]) :: integer
def min_distance(height, width, tree, squirrel, nuts) do

end
end
```

Erlang:

```
-spec min_distance(Height :: integer(), Width :: integer(), Tree ::
[integer()], Squirrel :: [integer()], Nuts :: [[integer()]]) -> integer().
min_distance(Height, Width, Tree, Squirrel, Nuts) ->
.
```

Racket:

```
(define/contract (min-distance height width tree squirrel nuts)
(-> exact-integer? exact-integer? (listof exact-integer?) (listof
exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Squirrel Simulation
* Difficulty: Medium
* Tags: array, tree, math
*
* 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:
int minDistance(int height, int width, vector<int>& tree, vector<int>&
```

```
squirrel, vector<vector<int>>& nuts) {

}
};
```

Java Solution:

```
/**
* Problem: Squirrel Simulation
* Difficulty: Medium
* Tags: array, tree, math
*
* 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 int minDistance(int height, int width, int[] tree, int[] squirrel,
int[][] nuts) {

}
}
```

Python3 Solution:

```
"""
Problem: Squirrel Simulation
Difficulty: Medium
Tags: array, tree, math

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 minDistance(self, height: int, width: int, tree: List[int], squirrel:
List[int], nuts: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minDistance(self, height, width, tree, squirrel, nuts):
"""
:type height: int
:type width: int
:type tree: List[int]
:type squirrel: List[int]
:type nuts: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Squirrel Simulation
* Difficulty: Medium
* Tags: array, tree, math
*
* 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} height
* @param {number} width
* @param {number[]} tree
* @param {number[]} squirrel
* @param {number[][]} nuts
* @return {number}
*/
var minDistance = function(height, width, tree, squirrel, nuts) {

};
```

TypeScript Solution:

```
/**
* Problem: Squirrel Simulation
* Difficulty: Medium
* Tags: array, tree, math
```

```
*
* 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 minDistance(height: number, width: number, tree: number[], squirrel:
number[], nuts: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Squirrel Simulation
* Difficulty: Medium
* Tags: array, tree, math
*
* 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 int MinDistance(int height, int width, int[] tree, int[] squirrel,
int[][] nuts) {

}
}
```

C Solution:

```
/*
* Problem: Squirrel Simulation
* Difficulty: Medium
* Tags: array, tree, math
*
* 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
*/
```

```
int minDistance(int height, int width, int* tree, int treeSize, int*
squirrel, int squirrelSize, int** nuts, int nutsSize, int* nutsColSize) {

}
```

Go Solution:

```
// Problem: Squirrel Simulation
// Difficulty: Medium
// Tags: array, tree, math
//
// 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 minDistance(height int, width int, tree []int, squirrel []int, nuts
[][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minDistance(height: Int, width: Int, tree: IntArray, squirrel: IntArray,
nuts: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minDistance(_ height: Int, _ width: Int, _ tree: [Int], _ squirrel:
[Int], _ nuts: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Squirrel Simulation
// Difficulty: Medium
// Tags: array, tree, math
//
// 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 min_distance(height: i32, width: i32, tree: Vec<i32>, squirrel:
Vec<i32>, nuts: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} height
# @param {Integer} width
# @param {Integer[]} tree
# @param {Integer[]} squirrel
# @param {Integer[][]} nuts
# @return {Integer}
def min_distance(height, width, tree, squirrel, nuts)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $height
* @param Integer $width
* @param Integer[] $tree
* @param Integer[] $squirrel
* @param Integer[][] $nuts
* @return Integer
*/
function minDistance($height, $width, $tree, $squirrel, $nuts) {

}
```

```
}
```

Dart Solution:

```
class Solution {
int minDistance(int height, int width, List<int> tree, List<int> squirrel,
List<List<int>> nuts) {

}
}
```

Scala Solution:

```
object Solution {
def minDistance(height: Int, width: Int, tree: Array[Int], squirrel:
Array[Int], nuts: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_distance(height :: integer, width :: integer, tree :: [integer],
squirrel :: [integer], nuts :: [[integer]]) :: integer
def min_distance(height, width, tree, squirrel, nuts) do

end
end
```

Erlang Solution:

```
-spec min_distance(Height :: integer(), Width :: integer(), Tree ::
[integer()], Squirrel :: [integer()], Nuts :: [[integer()]]) -> integer().
min_distance(Height, Width, Tree, Squirrel, Nuts) ->
.
```

Racket Solution:

```
(define/contract (min-distance height width tree squirrel nuts)
(-> exact-integer? exact-integer? (listof exact-integer?) (listof
```

```
exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```