Problem 587: Erect the Fence

Problem Information

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

Problem Description

You are given an array

trees

where

trees[i] = [x

i

, y

i

]

represents the location of a tree in the garden.

Fence the entire garden using the minimum length of rope, as it is expensive. The garden is well-fenced only if

all the trees are enclosed

.

Return

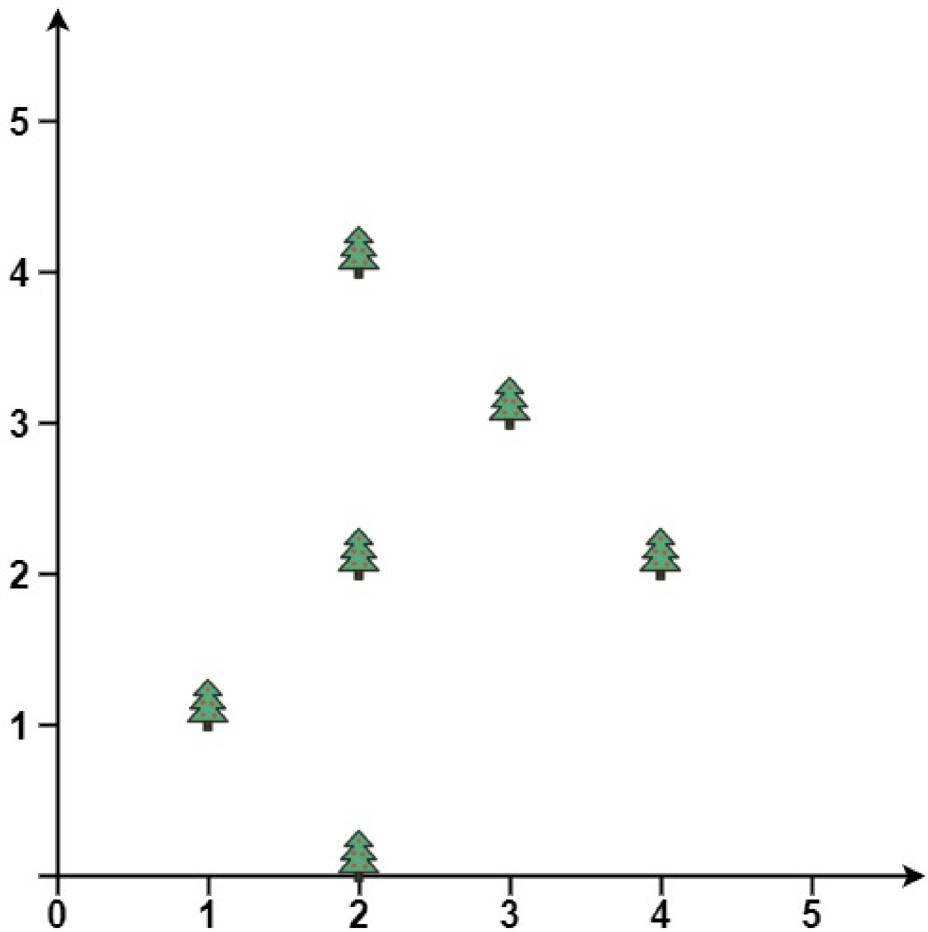
the coordinates of trees that are exactly located on the fence perimeter

. You may return the answer in

any order

.

Example 1:



Input:

trees = [[1,1],[2,2],[2,0],[2,4],[3,3],[4,2]]

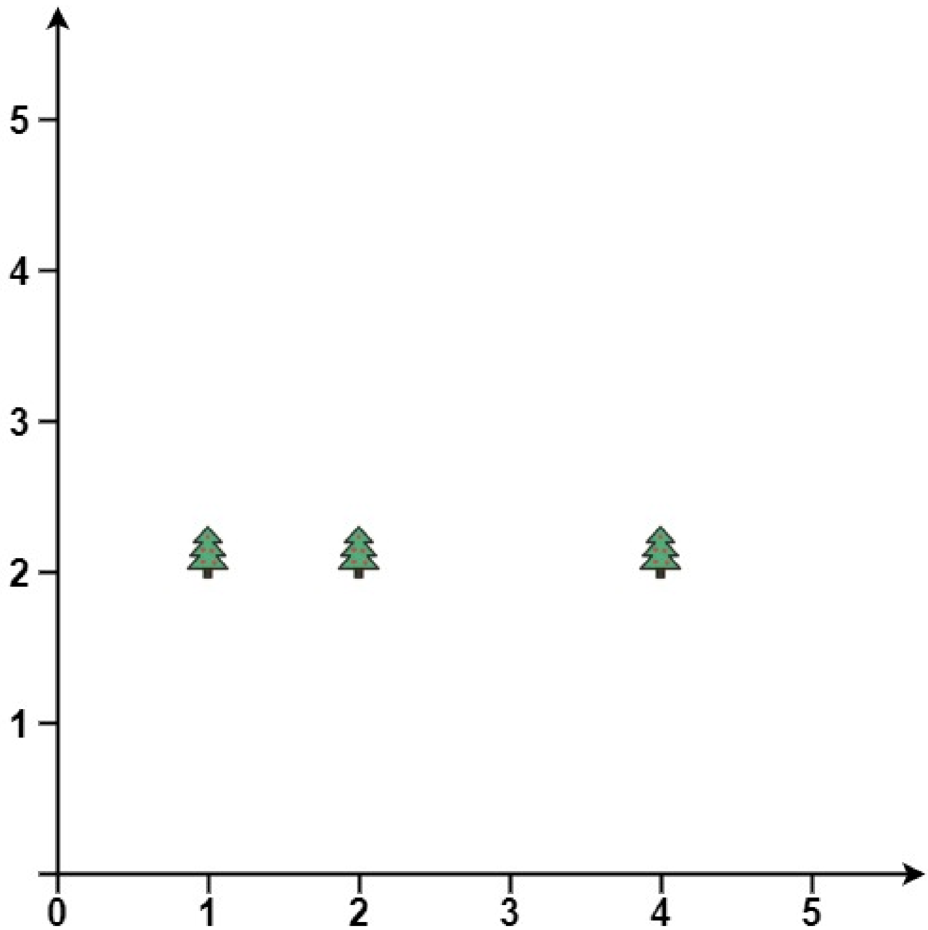
Output:

[[1,1],[2,0],[4,2],[3,3],[2,4]]

Explanation:

All the trees will be on the perimeter of the fence except the tree at [2, 2], which will be inside the fence.

Example 2:



Input:

trees = [[1,2],[2,2],[4,2]]

Output:

[[4,2],[2,2],[1,2]]

Explanation:

The fence forms a line that passes through all the trees.

Constraints:

1 <= trees.length <= 3000

trees[i].length == 2

0 <= x

i

, y

i

<= 100

All the given positions are

unique

.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> outerTrees(vector<vector<int>>& trees) {

}
};
```

Java:

```
class Solution {
public int[][] outerTrees(int[][] trees) {

}
```

```
}
```

Python3:

```
class Solution:
def outerTrees(self, trees: List[List[int]]) -> List[List[int]]:
```

Python:

```
class Solution(object):
def outerTrees(self, trees):
"""
:type trees: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[][]} trees
* @return {number[][]}
*/
var outerTrees = function(trees) {

};
```

TypeScript:

```
function outerTrees(trees: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] OuterTrees(int[][] trees) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** outerTrees(int** trees, int treesSize, int* treesColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

```
func outerTrees(trees [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun outerTrees(trees: Array<IntArray>): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func outerTrees(_ trees: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn outer_trees(trees: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} trees
# @return {Integer[][]}
def outer_trees(trees)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $trees
* @return Integer[][]
*/
function outerTrees($trees) {

}
}
```

Dart:

```
class Solution {
List<List<int>> outerTrees(List<List<int>> trees) {

}
}
```

Scala:

```
object Solution {
def outerTrees(trees: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec outer_trees(trees :: [[integer]]) :: [[integer]]
def outer_trees(trees) do

end
end
```

Erlang:

```
-spec outer_trees(Trees :: [[integer()]]) -> [[integer()]].
outer_trees(Trees) ->
.
```

Racket:

```
(define/contract (outer-trees trees)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Erect the Fence
* Difficulty: Hard
* 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:
vector<vector<int>> outerTrees(vector<vector<int>>& trees) {

}
};
```

Java Solution:

```
/**
* Problem: Erect the Fence
* Difficulty: Hard
* 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[][] outerTrees(int[][] trees) {

}
}
```

Python3 Solution:

```
"""
Problem: Erect the Fence
Difficulty: Hard
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 outerTrees(self, trees: List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def outerTrees(self, trees):
"""
:type trees: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Erect the Fence
* Difficulty: Hard
```

```
* 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[][]} trees
* @return {number[][]}
*/
var outerTrees = function(trees) {

};
```

TypeScript Solution:

```
/**
* Problem: Erect the Fence
* Difficulty: Hard
* 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 outerTrees(trees: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Erect the Fence
* Difficulty: Hard
* 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[][] OuterTrees(int[][] trees) {

}
}
```

C Solution:

```
/*
* Problem: Erect the Fence
* Difficulty: Hard
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** outerTrees(int** trees, int treesSize, int* treesColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Erect the Fence
// Difficulty: Hard
// 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
```

```go
func outerTrees(trees [][]int) [][]int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun outerTrees(trees: Array<IntArray>): Array<IntArray> {

}
}
```

Swift Solution:

```swift
class Solution {
func outerTrees(_ trees: [[Int]]) -> [[Int]] {

}
}
```

Rust Solution:

```rust
// Problem: Erect the Fence
// Difficulty: Hard
// 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 outer_trees(trees: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```ruby
# @param {Integer[][]} trees
# @return {Integer[][]}
def outer_trees(trees)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $trees
* @return Integer[][]
*/
function outerTrees($trees) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> outerTrees(List<List<int>> trees) {

}
}
```

Scala Solution:

```
object Solution {
def outerTrees(trees: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec outer_trees(trees :: [[integer]]) :: [[integer]]
def outer_trees(trees) do

end
end
```

Erlang Solution:

```erlang
-spec outer_trees(Trees :: [[integer()]]) -> [[integer()]].
outer_trees(Trees) ->
  .
```

Racket Solution:

```racket
(define/contract (outer-trees trees)
  (-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
  )
```