Problem 1620: Coordinate With Maximum Network Quality

Problem Information

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

Problem Description

You are given an array of network towers

towers

, where

towers[i] = [x

i

, y

i

, q

i

]

denotes the

i

th

network tower with location

(x

i

, y

i

)

and quality factor

q

i

. All the coordinates are

integral coordinates

on the X-Y plane, and the distance between the two coordinates is the

Euclidean distance

.

You are also given an integer

radius

where a tower is

reachable

if the distance is

less than or equal to

radius

. Outside that distance, the signal becomes garbled, and the tower is

not reachable

.

The signal quality of the

i

th

tower at a coordinate

(x, y)

is calculated with the formula

■q

i

/ (1 + d)■

, where

d

is the distance between the tower and the coordinate. The

network quality

at a coordinate is the sum of the signal qualities from all the

reachable

towers.

Return

the array

[c

x

, c

y

]

representing the

integral

coordinate

(c

x

, c

y

)

where the

network quality

is maximum. If there are multiple coordinates with the same

network quality

, return the lexicographically minimum

non-negative

coordinate.

Note:

A coordinate

(x1, y1)

is lexicographically smaller than

(x2, y2)

if either:

x1 < x2

, or

x1 == x2

and

y1 < y2

.

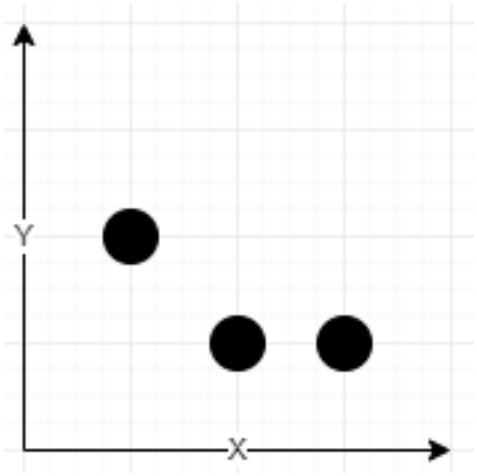
■val■

is the greatest integer less than or equal to

val

(the floor function).

Example 1:



Input:

towers = [[1,2,5],[2,1,7],[3,1,9]], radius = 2

Output:

[2,1]

Explanation:

At coordinate (2, 1) the total quality is 13. - Quality of 7 from (2, 1) results in ■7 / (1 + sqrt(0)■ = ■7■ = 7 - Quality of 5 from (1, 2) results in ■5 / (1 + sqrt(2)■ = ■2.07■ = 2 - Quality of 9 from (3, 1) results in ■9 / (1 + sqrt(1)■ = ■4.5■ = 4 No other coordinate has a higher network quality.

Example 2:

Input:

towers = [[23,11,21]], radius = 9

Output:

[23,11]

Explanation:

Since there is only one tower, the network quality is highest right at the tower's location.

Example 3:

Input:

towers = [[1,2,13],[2,1,7],[0,1,9]], radius = 2

Output:

[1,2]

Explanation:

Coordinate (1, 2) has the highest network quality.

Constraints:

1 <= towers.length <= 50

towers[i].length == 3

0 <= x

i

, y

i

, q

i

<= 50

1 <= radius <= 50

Code Snippets

C++:

```
class Solution {
public:
vector<int> bestCoordinate(vector<vector<int>>& towers, int radius) {

}
};
```

Java:

```
class Solution {
public int[] bestCoordinate(int[][] towers, int radius) {

}
}
```

Python3:

```
class Solution:
def bestCoordinate(self, towers: List[List[int]], radius: int) -> List[int]:
```

Python:

```
class Solution(object):
def bestCoordinate(self, towers, radius):
"""
:type towers: List[List[int]]
:type radius: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} towers
* @param {number} radius
* @return {number[]}
*/
var bestCoordinate = function(towers, radius) {
```

```
};
```

TypeScript:

```
function bestCoordinate(towers: number[][], radius: number): number[] {

};
```

C#:

```
public class Solution {
public int[] BestCoordinate(int[][] towers, int radius) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* bestCoordinate(int** towers, int towersSize, int* towersColSize, int
radius, int* returnSize) {

}
```

Go:

```
func bestCoordinate(towers [][]int, radius int) []int {

}
```

Kotlin:

```
class Solution {
fun bestCoordinate(towers: Array<IntArray>, radius: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func bestCoordinate(_ towers: [[Int]], _ radius: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn best_coordinate(towers: Vec<Vec<i32>>, radius: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} towers
# @param {Integer} radius
# @return {Integer[]}
def best_coordinate(towers, radius)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $towers
* @param Integer $radius
* @return Integer[]
*/
function bestCoordinate($towers, $radius) {

}
}
```

Dart:

```
class Solution {
List<int> bestCoordinate(List<List<int>> towers, int radius) {

}
```

```
}
```

Scala:

```
object Solution {
def bestCoordinate(towers: Array[Array[Int]], radius: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec best_coordinate(towers :: [[integer]], radius :: integer) :: [integer]
def best_coordinate(towers, radius) do

end
end
```

Erlang:

```
-spec best_coordinate(Towers :: [[integer()]], Radius :: integer()) ->
[integer()].
best_coordinate(Towers, Radius) ->
.
```

Racket:

```
(define/contract (best-coordinate towers radius)
(-> (listof (listof exact-integer?)) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Coordinate With Maximum Network Quality
* Difficulty: Medium
* Tags: array, graph
```

```
*
* 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> bestCoordinate(vector<vector<int>>& towers, int radius) {

}
};
```

Java Solution:

```
/**
* Problem: Coordinate With Maximum Network Quality
* Difficulty: Medium
* Tags: array, graph
*
* 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[] bestCoordinate(int[][] towers, int radius) {

}
}
```

Python3 Solution:

```
"""
Problem: Coordinate With Maximum Network Quality
Difficulty: Medium
Tags: array, graph

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 bestCoordinate(self, towers: List[List[int]], radius: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def bestCoordinate(self, towers, radius):
"""
:type towers: List[List[int]]
:type radius: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Coordinate With Maximum Network Quality
* Difficulty: Medium
* Tags: array, graph
*
* 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[][]} towers
* @param {number} radius
* @return {number[]}
*/
var bestCoordinate = function(towers, radius) {

};
```

TypeScript Solution:

```
/**
* Problem: Coordinate With Maximum Network Quality
* Difficulty: Medium
* Tags: array, graph
*
* 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 bestCoordinate(towers: number[][], radius: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Coordinate With Maximum Network Quality
* Difficulty: Medium
* Tags: array, graph
*
* 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[] BestCoordinate(int[][] towers, int radius) {

}
}
```

C Solution:

```
/*
* Problem: Coordinate With Maximum Network Quality
* Difficulty: Medium
* Tags: array, graph
*
* 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* bestCoordinate(int** towers, int towersSize, int* towersColSize, int
radius, int* returnSize) {

}
```

Go Solution:

```
// Problem: Coordinate With Maximum Network Quality
// Difficulty: Medium
// Tags: array, graph
//
// 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 bestCoordinate(towers [][]int, radius int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun bestCoordinate(towers: Array<IntArray>, radius: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func bestCoordinate(_ towers: [[Int]], _ radius: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Coordinate With Maximum Network Quality
// Difficulty: Medium
// Tags: array, graph
//
// 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 best_coordinate(towers: Vec<Vec<i32>>, radius: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} towers
# @param {Integer} radius
# @return {Integer[]}
def best_coordinate(towers, radius)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $towers
* @param Integer $radius
* @return Integer[]
*/
function bestCoordinate($towers, $radius) {

}
}
```

Dart Solution:

```
class Solution {
List<int> bestCoordinate(List<List<int>> towers, int radius) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def bestCoordinate(towers: Array[Array[Int]], radius: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec best_coordinate(towers :: [[integer]], radius :: integer) :: [integer]
def best_coordinate(towers, radius) do

end
end
```

Erlang Solution:

```
-spec best_coordinate(Towers :: [[integer()]], Radius :: integer()) ->
[integer()].
best_coordinate(Towers, Radius) ->
.
```

Racket Solution:

```
(define/contract (best-coordinate towers radius)
(-> (listof (listof exact-integer?)) exact-integer? (listof exact-integer?))
)
```