1620. Coordinate With Maximum Network Quality

Medium Array **Enumeration**

Problem Description

with three elements - the x-coordinate, the y-coordinate, and the quality factor. We have to find a point on an X-Y plane that has the maximum sum of the signal qualities from all the reachable towers.

In this problem, we are given the locations and quality factors of several network towers. Each tower is represented as an array

A tower is considered reachable if it is within a given radius. The quality of the signal from a tower decreases as the distance from the tower increases, and it is calculated using the formula q_i / (1 + d), where q_i is the tower's quality factor and d is the

Euclidean distance from the tower to the point of interest. We round down the calculated quality to the nearest integer. We need to find the coordinates (cx, cy) where the network quality is the highest. If there are multiple such coordinates with the

those are the same, then the one with the smallest non-negative y-coordinate.

To solve this problem, we perform exhaustive search – basically, we iterate over all the possible coordinates and calculate the

same quality, we return the smallest lexicographically coordinate, one that has the smallest non-negative x-coordinate, and if

network quality at each point by summing up the signal quality from each tower that is within the given radius.

coordinates corresponding to mx. We compute the network quality at every point by iterating over a predefined range (coordinates from (0, 0) to (50, 50) in this case, assuming these bounds are set by the problem context, i.e., the given x's and

We initiate a variable mx to keep track of the maximum network quality found so far, and a variable ans to store the best

y's are expected to lie within this range). For each point (i, j), we iterate over all towers checking their reachability. If a tower is reachable (distance d is less than or equal to radius), we calculate its contribution to the signal quality using the provided formula, rounding down to the nearest integer, and add it to the current network quality t.

If t exceeds mx, we update mx with t and set ans to the current coordinates (i, j). After checking all possible coordinates, ans will contain the coordinates with the highest possible network quality. **Solution Approach**

The solution provided uses a brute-force approach to find the best coordinate with the maximum network quality. For this, we use two nested loops to iterate over all possible coordinate points within the given constraints. In this case, it

iterates through (0, 0) to (50, 50) as the solution only examines points with integral coordinates within this range.

We initiate a variable mx to track the highest network quality observed and a list ans to store the corresponding coordinates of

this quality.

network quality.

Example Walkthrough

Inside the inner loop, for every point (i, j), another nested loop iterates through the list of towers. For each tower, it

- calculates the Euclidean distance d from the current coordinate (i, j) using the formula $sqrt((x i)^2 + (y j)^2)$. If the calculated distance is less than or equal to the radius, the tower is reachable, and its quality contribution is computed using the floor division formula q / (1 + d). This is because signal quality is integer-valued after flooring.
- The variable t represents the total signal quality at the point (i, j) and is calculated by summing up the contributions from all reachable towers.
- how the loops are structured (starting from (0, 0)). The function floor() is used to perform the floor division operation to ensure that the quality is an integer. •

Finally, after all the coordinates are checked, the function returns ans, which points to the coordinate with the highest

If at any point (i, j), the total signal quality t is greater than mx, mx is updated to t, and ans is updated to [i, j]. If t is equal

to the maximum found so far, it automatically retains the earliest (and therefore lexicographically smallest) coordinates due to

- This approach does not use any sophisticated algorithmic pattern but relies on a simple exhaustive search. It is feasible when the range of coordinates is small, as the computational complexity is O(n * m * t), where n and m are the ranges of x and y coordinates to be checked, and t is the number of towers.
- Consider the following small example to illustrate the solution approach:

Suppose we have two towers and the radius is given to be 2 units. The towers have the following properties:

• Tower 1 has coordinates (1, 2) and a quality factor of 5. • Tower 2 has coordinates (3, 4) and a quality factor of 10. To simplify, let's consider our grid extends from (0, 0) to (5, 5).

We would iterate from (0, 0) to (5, 5) and calculate the total network quality at each point. For example, at point (0, 0):

• Compute the distance from Tower 1: $d1 = sqrt((1 - 0)^2 + (2 - 0)^2) = sqrt(1 + 4) = sqrt(5)$ • Since d1 is greater than the radius of 2, Tower 1 is unreachable and contributes 0 to the network quality.

• Since d2 is also greater than the radius of 2, Tower 2 is unreachable and contributes 0 to the network quality. • The total network quality at point (0, 0) is 0.

corresponding coordinates. After we check all points:

Continuing this process for each point, assume we reach coordinate (2, 3):

• Compute the distance from Tower 1: $d1 = sqrt((1-2)^2 + (2-3)^2) = sqrt(1+1) = sqrt(2)$

We continue this for all points on the grid from (0, 0) to (5, 5), keeping track of the maximum network quality and the

The reason for choosing such a small grid is to explain the brute force approach clearly without diving deep into a complex

calculation that would involve many iterations. This step-by-step progression through the solution approach provides insight into

• The total network quality at point (2, 3) is the sum of quality contributions from both towers.

• Tower 2 is reachable as d2 <= 2, so the quality contribution from Tower 2 is floor(10 / (1 + sqrt(2))).

• Tower 1 is reachable as d1 <= 2, so the quality contribution from Tower 1 is floor(5 / (1 + sqrt(2))).

• Compute the distance from Tower 2: $d2 = sqrt((3-2)^2 + (4-3)^2) = sqrt(1+1) = sqrt(2)$

• Compute the distance from Tower 2: $d2 = sqrt((3 - 0)^2 + (4 - 0)^2) = sqrt(9 + 16) = sqrt(25)$

• Given the highest quality, ans would be [2, 3].

• Let's say the highest network quality found was 12 at point (2, 3). This would be our mx.

the methodology of searching for the best signal quality on a grid.

Initialize max signal quality and answer coordinates.

if distance <= coverage_radius:</pre>

if total_quality > max_quality:

best_coordinate = [i, j]

max_quality = total_quality

for x, y, q in towers:

return best_coordinate

Finally, the coordinates (2, 3) with the highest network quality 12 would be returned.

Solution Implementation

def bestCoordinate(self, towers: List[List[int]], coverage_radius: int) -> List[int]:

Calculate the signal quality from each tower at the current point.

Add quality to the total if inside the coverage radius.

If the total signal quality at this point is greater than the max,

After checking all points, return the coordinates with the highest signal quality.

distance = ((x - i) ** 2 + (y - j) ** 2) ** 0.5

total_quality += floor(q / (1 + distance))

update max_quality and best_coordinate to this point.

// Function to find the best coordinate with the maximum signal quality

int maxSignalQuality = 0; // To store the maximum signal quality

// Iterate through each possible x coordinate within the grid limit

// Iterate through each possible y coordinate within the grid limit

// Iterate through each tower to calculate its contribution

bestCoord = $\{x, y\}$; // Update the best coordinate

// Return the best coordinate after checking all possible points

int currentSignal = 0; // Signal quality at the current coordinate

// Calculate Euclidean distance from the current coordinate to the tower

double distance = sqrt((x - tower[0]) * (x - tower[0]) + (y - tower[1]) * (y - tower[1]));

// If the distance is within the effective radius, add the tower's signal quality

currentSignal += static_cast<int>(floor(tower[2] / (1 + distance)));

// Check if the current signal quality is greater than the max found so far

maxSignalQuality = currentSignal; // Update max signal quality

vector<int> bestCoord = {0, 0}; // To store the best coordinate

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

for (int x = 0; x < 51; ++x) {

for (int y = 0; y < 51; ++y) {

for (auto& tower : towers) {

if (distance <= radius) {</pre>

if (maxSignalQuality < currentSignal) {</pre>

Calculate euclidean distance from the tower to the current point.

max_quality = 0 best_coordinate = [0, 0] # Iterate through each possible point on the grid. for i in range(51): for j in range(51): total_quality = 0 # Sum of signal qualities from all towers for this point.

Java

public:

class Solution {

Python

from math import floor

class Solution:

```
public int[] bestCoordinate(int[][] towers, int radius) {
        int maxSignal = 0; // to keep track of the highest signal quality
        int[] bestCoordinates = new int[]{0, 0}; // to hold the best coordinates
        // Loop through each possible coordinate on the grid up to 50x50
        for (int x = 0; x < 51; x++) {
            for (int y = 0; y < 51; y++) {
                int signalQuality = 0; // to calculate the total signal quality at the point (x, y)
                // Check each tower's contribution to the signal quality at the point (x, y)
                for (int[] tower : towers) {
                    // Calculate the distance between tower and point (x, y)
                    double distance = Math.sqrt(Math.pow(x - tower[0], 2) + Math.pow(y - tower[1], 2));
                    // Add the tower's signal contribution if it is within radius
                    if (distance <= radius) {</pre>
                        signalQuality += Math.floor(tower[2] / (1 + distance));
                // Update the maximum signal quality and the best coordinates if the current point is better
                if (maxSignal < signalQuality) {</pre>
                    maxSignal = signalQuality;
                    bestCoordinates = new int[]{x, y};
        // Return the coordinates with the best signal quality
        return bestCoordinates;
C++
#include <vector>
#include <cmath>
using namespace std;
class Solution {
```

};

TypeScript

return bestCoord;

type Tower = [number, number, number];

```
// Function to calculate Euclidean distance between two points
  function calculateDistance(x1: number, y1: number, x2: number, y2: number): number {
    return Math.sqrt(Math.pow((x1 - x2), 2) + Math.pow((y1 - y2), 2));
  // Function to find the best coordinate with the maximum signal quality
  function bestCoordinate(towers: Tower[], radius: number): [number, number] {
    let maxSignalQuality: number = 0; // To store the maximum signal quality
    let bestCoord: [number, number] = [0, 0]; // To store the best coordinate
    // Iterate through each possible x coordinate within the grid limit
    for (let x = 0; x < 51; ++x) {
      // Iterate through each possible y coordinate within the grid limit
      for (let y = 0; y < 51; ++y) {
        let currentSignal: number = 0; // Signal quality at the current coordinate
        // Iterate through each tower to calculate its contribution
        for (let tower of towers) {
          // Calculate Euclidean distance from the current coordinate to the tower
          let distance: number = calculateDistance(x, y, tower[0], tower[1]);
          // If the distance is within the effective radius, add the tower's signal quality
          if (distance <= radius) {</pre>
            currentSignal += Math.floor(tower[2] / (1 + distance));
        // Check if the current signal quality is greater than the max found so far
        if (maxSignalQuality < currentSignal) {</pre>
          maxSignalQuality = currentSignal; // Update max signal quality
          bestCoord = [x, y]; // Update the best coordinate
    // Return the best coordinate after checking all possible points
    return bestCoord;
from math import floor
class Solution:
   def bestCoordinate(self, towers: List[List[int]], coverage_radius: int) -> List[int]:
       # Initialize max signal quality and answer coordinates.
        max quality = 0
        best_coordinate = [0, 0]
       # Iterate through each possible point on the grid.
        for i in range(51):
            for j in range(51):
                total_quality = 0 # Sum of signal qualities from all towers for this point.
               # Calculate the signal quality from each tower at the current point.
                for x, y, q in towers:
                    # Calculate euclidean distance from the tower to the current point.
                    distance = ((x - i) ** 2 + (y - j) ** 2) ** 0.5
                    # Add quality to the total if inside the coverage radius.
                    if distance <= coverage_radius:</pre>
                        total_quality += floor(q / (1 + distance))
               # If the total signal quality at this point is greater than the max,
                # update max_quality and best_coordinate to this point.
                if total_quality > max_quality:
```

Time Complexity

complexity.

return best_coordinate

Time and Space Complexity

• The first two loops iterate over a fixed range of 51 each, resulting in 51 * 51 iterations. • The innermost loop iterates over the list of towers. If n is the total number of towers, this loop will execute n times for each iteration of the first two loops.

The given code runs three nested loops:

max_quality = total_quality

After checking all points, return the coordinates with the highest signal quality.

best_coordinate = [i, j]

Therefore, the time complexity can be determined by multiplying the number of iterations in each loop. This gives us 51 * 51 * n, which simplifies to O(n) since the 51 * 51 is a constant.

Thus, the overall time complexity is O(n). **Space Complexity**

The space complexity of the given code is quite straightforward. There are a few integers (mx, i, j, t, x, y, q, d) and a list ans of

size 2 being used to store the answer, which do not depend on the input size. Therefore, since no additional space is used that scales with the input size, the space complexity is 0(1), or constant space