Reflection

Summary

My approach to solve this problem was based on an improved version of greedy algorithm. My implementation could solve all inputs in less than 10 seconds, on a moderately powerful laptop.

Description of method

My method works in two phases:

- Divide all cities into groups
- For each group designate a tower.

A group of city, in this context, is a list of several cities, that the distance between each other is less than `2*coverage_radius`. In other words, between any two cities in one group, we could always use one tower to cover both of them. Simple geometry would show that we can in fact use one tower to cover ALL cities in a group. To divide cities into groups, we used greedy algorithm:

```
greedy_group(cities):
      groups = []
    while there are still cities ungrouped:
      let new_group = [first remaining city]
      for all other cities X:
           if X's distance to all other cities in new_group < 2*radius:

                add X to new_group
      groups.append(new_group)
    return groups
```

In other words, we would set the first ungrouped city as a group, and try to add all of its neighbors into this group. This is looped till all cities have a group.

Then to assign a tower for the group, we simply take the average of all the city coordinates.

Why I think this is a good approach

1. This approach is extremely fast. We only need to loop through the list of cities several times and get a solution.
2. The positioning of tower for a group of cities are calculated using average of coordinates. So the positioning of towers are more scientific than other solutions.

What other approaches I tried

Naive greedy + tower removal

I tried to modify the given naive solver that puts a tower up for each city by allowing the algorithm to scan for a second pass to see what cities are covered by more than one towers, and if so try to remove any.

This algorithm doesn't work as good as mine. Per my observation it generated approximately 15% higher penalty on average. This is due to the fact that it will have to place towers on cities.

Minimum spanning tree based

I have also considered building several MSTs across the whole map. Once the MST is built, I will try to start from the root, and cover as many cities from the root as possible. For remaining branches, they can be upgraded into MSTs themselves. And we can solve it recursively.

This method is extremely slow, and the quality of solutions found are not greater than what I presented here.

What computational resources did I use

I only used my own laptop to solve this problem.