CS2040S: Data Structures and Algorithms

WiFi

(Optional)

WiFi

Problem 0. Finding the Optimal WiFi Coverage You've just moved to a new street where all the houses intend to share public routers. Since these things are costly for the residents of the street, they can only afford m many routers. Your job, is to find the minimum radius that the routers need to cover, so that all houses in the street are covered.

Problem 0.a. Say for now, you had to solve an easier problem: given a radius r and some number of routers m as well as the placements of each house on the street, just determine whether it is possible for all the houses to be covered.

For example, if houses were placed at positions $1, 3, 10$ of the street, $m = 2$ and $r = 1.5$, then it definitely would be possible, since we can place one router at position 1.5 and one at 10. Note that this isn't the only possible placement of routers that covers all the houses. Another example placement is at position 2 and 8.5. Additionally, note that the routers need not be placed at the same position of where a house may be, even though they can, if need be.

On the other hand, if the houses were placed at positions $1, 3, 10$ of the street, $m = 2$ and $r = 0.5$, then it would be impossible to cover the houses, no matter the placement of the routers.

Implement `public static boolean coverable(int[] houses, int numOfAccessPoints, double distance)` that takes in an integer array `houses` which contains the positions of the houses (here in our example `houses` would be the array $\{1, 3, 10\}$), `numOfAccessPoints`, which is the number of routers to be placed, and `distance`, which is the radius of coverage of a single router. What it needs to return is `true` if it is possible to cover all the houses with such parameters, and return `false` otherwise.

Important note: The array `houses` may not come in sorted order. i.e. even $\{10, 1, 3\}$ is a possible input. You may use `Arrays.sort(houses)` to sort it first if that helps your algorithm. The `Arrays.sort()` method can be obtained by importing `java.util.Arrays`.

Problem 0.b. Assuming you have a solution to the previous part, the next question would be: How would you use this to find out the smallest possible distance? Two important questions to answer would be: (1) What happens if you consider a radius that is smaller than the minimum possible? What should the output of the previous function be? (2) What happens if you consider a radius that is larger than the minimum possible? o answer would be: (1) What happens if you consider a radius that is smaller than the minimum possible? What should the output of the previous function be?

For example, if houses were placed at positions $1, 3, 10$ of the street, $m = 2$ and $r = 1.5$, then the minimum possible radius for the routers would be 1.0. Since we can place a router at position 2.0 and a router at position 10. Furthermore, if the radius were any smaller then no placement of the routers would cover all the houses.

Implement `public static double computeDistance(int[] houses, int numOfAccessPoints)` that takes in an integer array `houses` which contains the positions of the houses (here in our example `houses` would be the array $\{1, 3, 10\}$), `numOfAccessPoints`, which is the number of routers to be placed and returns a radius that is within 0.5 of the true minimum distance.

(Hint: Use the function implemented in the previous subproblem.)