ALGORITHM ANALYSIS HW1 REPORT

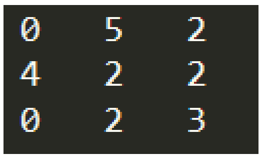
Cem Güleç – 150117828

1. Introduction:

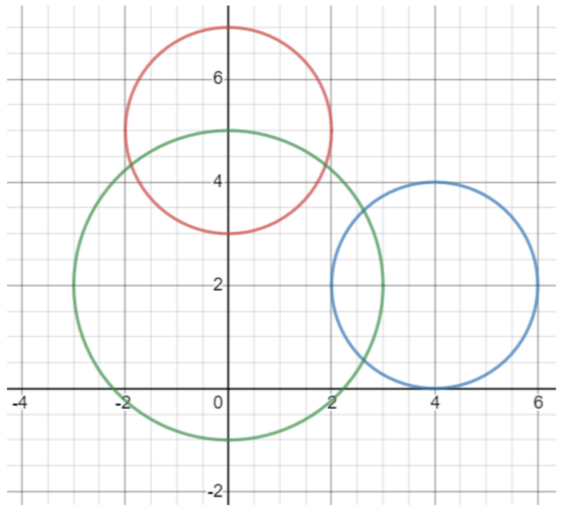
Our objective for this program is in a simple fashion checking whether laptop devices with the given coordinate locations and ranges able to reach among themselves. After knowing all the information (which one of those computers can reach each other), we use graph structure to represent it.

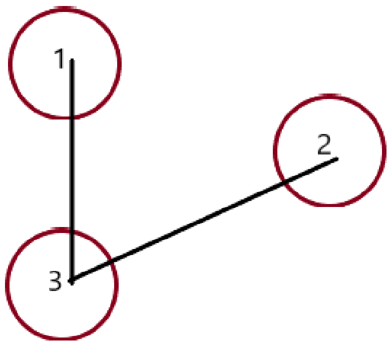
In graph, we declare edges between those which computers are in each other's range. For example, for the given input "test2.txt" ranges should look like below images:

Input format (x, y, range): **In coordinate system:** **Graph representation**



```
0   5   2
4   2   2
0   2   3
```





Here, we have 3 laptops. As we can see in the second image, we need to imagine their ranges like that. And after knowing that, we simply put edges between them to represent the structure in a graph.

Our final goal is to, traverse this graph with an algorithm which is going to work in BFS algorithm manner. Then taking the 1st laptop as the reference laptop, calculate the distance unit to go each laptop from the referenced laptop.

2.Fundamentals:

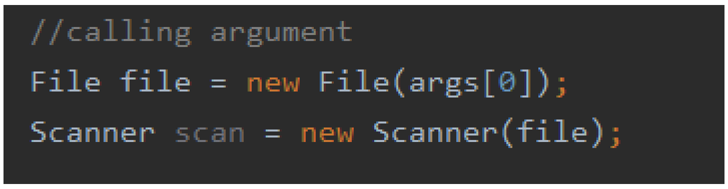
At the coding part, I would like to declare that we have 3 java classes which are separating functions among themselves.

Laptop.java: It basically is used to create instances of laptop devices. It is functional to get coordinate and range values of each laptop.

Graph.java: This function allows us to create an instance of the graph itself. In order to represent it, I have used adjacency list. And holding the number of vertices inside of a variable.

In graph function, if we required to put an edge between 2 device, we call them from adjacency list with the given index and append destination index to source location of the list.

Main.java: In here, I would like to mention that, besides functions inside of Main.java, inner main function of the Main.java calls argument as it can be seen below. It allows us to run our program with the given txt name from the command line.



```
//calling argument
File file = new File(args[0]);
Scanner scan = new Scanner(file);
```

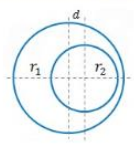
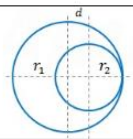
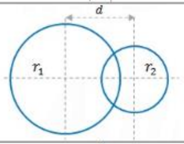
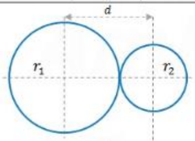
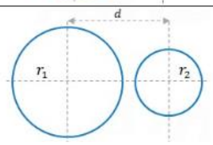
3.Functions:

read_file: It is the first called function in order to scrap laptop information from the given txt file. Here, we pass argument parameter(string) as the name of the file so that we can use the program through command line itself. There are some “#” characters which are the definition lines which are needs to be get rid of during scrapping.

calculate_distance: This function calculates and returns the distance between two points, depending on the equation below:

$$\sqrt{(\,(|\boldsymbol{x1}-\boldsymbol{x2}|)^2+(|\boldsymbol{y1}-\boldsymbol{y2}|)^2)}$$

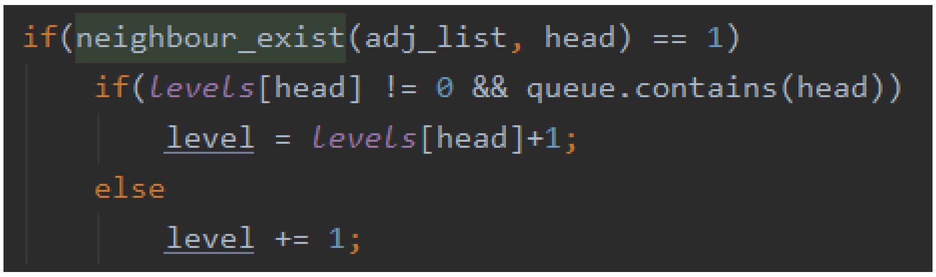
hop_distance: This function returns 1 if two laptop devices in the same range, else returns 0, depending on the mathematical formulas below with the given cases. Even though, two range have outer tangency, they still reach out to each other. So, it also needs to be counted as a returning 1 case.

Description	*Condition*
	*Circles equations:* $(x-x_1)^2+(y-y_1)^2=r_1^2$ $(x-x_2)^2+(y-y_2)^2=r_2^2$ *Distance between circles centers d is:* $d=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$ $d<\lvert r_1-r_2\rvert$
	*Inner tangency* $\lvert r_1-r_2\rvert$
	*Intersecting circles* $\lvert r_1-r_2\rvert<d<r_1+r_2$
	*Outer tangency* $d=r_1+r_2$
	*Intersecting circles* $d>r_1+r_2$

neighbour_exist: Checks whether a node with the given index has some neighbor which is not traversed. Returns 1 if there is still a neighbor, else returns 0.

BFS_traverse: This function is the most crucial one among all the others, since it has a big responsibility on the operation. Idea behind this is that, we hold a queue inside the function as a storage and as it is traversing through graph in BFS algorithm manner: whenever we put all the neighbors inside the queue we remove the head from the queue and repeat these procedures.

Tricky part with the function is that, whenever we add something to the queue we declare their unit level of distance from the referenced (1st computer), we assign of the distance to the nodes.



```
if(neighbour_exist(adj_list, head) == 1)
    if(levels[head] != 0 && queue.contains(head))
        level = levels[head]+1;
    else
        level += 1;
```

Another point is that, in the image above, if the queue head index is some node that we need to trace back, first we check it’s level assigned before and continue on from there to calculate distance using BFS.

After the algorithms completes, as I was holding these levels in an arraylist with the given indexes, we simply can reach out to it and print it.

Time Complexity: In this program, since our program’s performance depends on the BFS algoritm we can take this as referencing point. With this manner, time complexity becomes O(V+E), where V is the number of vertices and E is the number of edges on the graph structure. Complexity depends on which term is greater than which one, meaning that one of them surpass the other depending on the term value.

-> In case where number of vertices is greater than number of edges, then complexity will become O(V).

-> Otherwise, when the number of edges is greater than the number of vertices then complexity will become O(E).

Space Complexity: It will become O(V), only depending on the number of vertices, meaning that amount of place that needs to be reserved depending on the vertices (laptop device) number.