Notes on Interview question:

On my first read I discovered that:

* I have to create an object for cell phone towers with attributes: latitude, and longitude.
* I will have to calculate the distance between the cell phone towers using Haversine formula.
* I will have to allocate frequencies to each of these cell phone towers.
* Frequency range is from 110 to 115 (6 unique frequencies)
* The same frequency can’t be applied to a cell close to another cell (causes interference)
* Frequencies must be allocated in the most efficient manner such that cells farthest to another cell can have the same frequency to reduce interference.
* Produce a frequency allocation plan for each of these cells.
* A generic program should be coded to allow a new set of cells to be provided.

After creating a Class for tower Objects , I made another class which serves as a graph for them…

This graph has several attributes, including:  
- minFrequency  
- maxFrequency  
- maxDistanceThreshold (this attribute serves as a constant to determine which nodes are close to each other… this value can be adjusted to fit to a specific scenario and to accommodate a different set of towers based on their distances apart or even signal ranges)  
- the list of nodes of type tower,  
- and the list of connections of towers that are within a certain range of each other

Its constructor is used to initialize the graph with a list of tower nodes and to create a list of all connections for each node which are in a proximity of 0.269002km(Found this to be the best threshold for each node to not have more than 5 neighbors, since there are 6 unique numbers to allocate) from one another.

A buffered Reader is used to read the tower data from a text file and add it to an array list of object: Towers. The attributes of each tower is split up with “ “(spacebar) to distinguish between them.

A new instance of a Graph is then created and its parameter which requires a list is then provided as this list of towers created by the buffered reader.

After doing a little more research, I’ve found that to calculate the distance between two points on a sphere (earth) I will have to use the Haversine’s Formula…

I used this formula to calculate the distance between each of these cell phone towers.

After having established a method to calculate the distances, I tested the calculations by implementing a method to display the distance between each set of Towers…

All is well so far (The towers seem to be awkwardly close to each other but I’m just going to ignore it and assume this is a miniature simulation built on a small scale)

The establishConnection method is then used to iterate through each pair of towers and to create a connection between towers that are within the maximumDistanceThreshold of each other (as stated earlier, this method is used to determine which nodes are close to one another so I can assign different frequencies to them)

After establishing all connections, I wrote a displayConnections method to see how many connections each tower has, and then tuned the maxDistanceThreshold so that the max number of connections a tower can have is 5 (6 participants in the relationship/ 6 unique numbers)

Here is where it gets tricky:

I created an assignFrequencies method that is responsible for assigning frequencies to each tower in the graph based on the average distance to its neighbors  
- First the method sorts the towers based on their average distance to its neighbors to ensure that towers with similar distances are assigned frequencies together.  
- It then iterates through the sorted towers and assigns frequencies while considering interference with neighbors.  
- The method maintains a set of usedNumbers to keep track of frequencies already assigned to avoid repetition in towers nearby…  
- If all available frequencies are used, it clears the set to reuse the frequencies.

The getNeighbors method gets the list of towers connected to the given tower (connectedTowers)  
It iterates through the connected towers and adds their frequencies to a set (neighborFrequencies)  
It then returns the set of frequencies assigned to neighbors.

The “findAvailableNumber” method finds the smallest available frequency not used by neighbors.  
- It initializes a candidate frequency to the minimum frequency.  
- It then iterates until finding a frequency that is not used by its neighbors or in the set of used numbers  
- It handles wrapping around to the minimum frequency if it exceeds the maximum frequency.

The calculateAverageDistance method calculates the average distance of a tower to its neighbor  
- It gets the list of towers connected to the given tower (connectedTowers)  
- It iterates through connected towers and calculates the total distance  
- It then returns the average distance by dividing the total distance by the number of connected towers

In summary, the assignFrequencies method intelligently assigns frequencies to towers, taking into account the average distance to its neighbors and avoiding interference . The supporting methods (getNeighborFrequencies, findAvailableNumber, and calculateAverageDistance) contribute to the efficiency and correctness of this process…

Overall, this problem was somewhat of a challenge that I enjoyed solving.

It would be an honor to join the team if I get accepted 😊