Cell Towers Assignment

GROUP Q

AHMET HAKAN AFŞIN

UĞURCAN ÇATAK

KEVSER NUR AYAR

ERDEM EFE PEYNIRCI

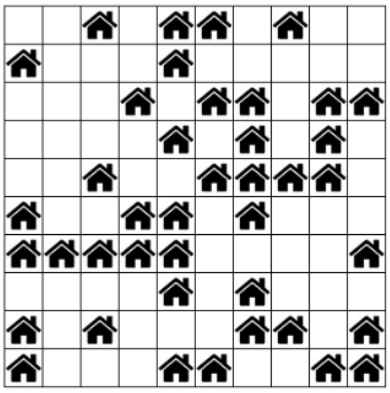
IE120: Industrial Engineering: Overview and Orientation

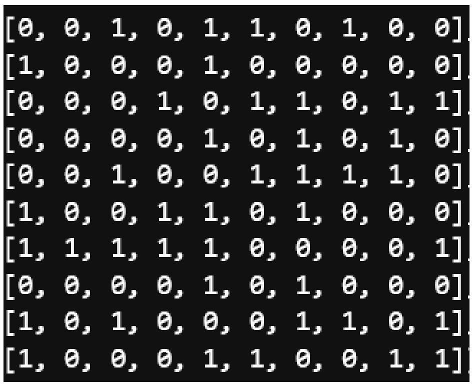
Taner Bilgiç

April 4, 2022

Our Approach

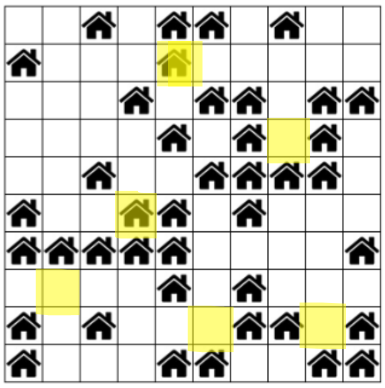
In order to solve the problem with programming, we decided to create two 10x10 matrices, one contains the houses, ant the other contains the positions of the towers. Later, by using the towers matrix, we can create a kernel that represents the service and apply that kernel in a addition operation between matrices.

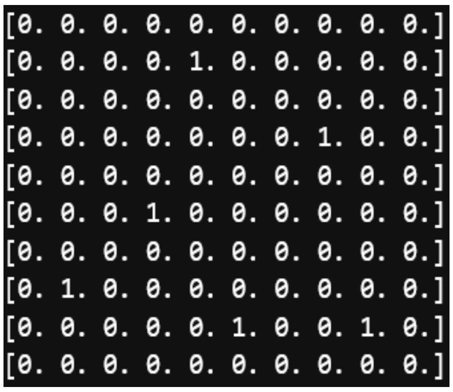




```
[0, 0, 1, 0, 1, 1, 0, 1, 0, 0]
[1, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 1, 1, 0, 1, 1]
[0, 0, 0, 0, 1, 0, 1, 0, 1, 0]
[0, 0, 1, 0, 0, 1, 1, 1, 1, 0]
[1, 0, 0, 1, 1, 0, 1, 0, 0, 0]
[1, 1, 1, 1, 1, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 0, 1, 0, 0, 0]
[1, 0, 1, 0, 0, 0, 1, 1, 0, 1]
[1, 0, 0, 0, 1, 1, 0, 0, 1, 1]
```

The house matrix contains the value “1” at the positions where there is a house. All other positions contain the value “0”

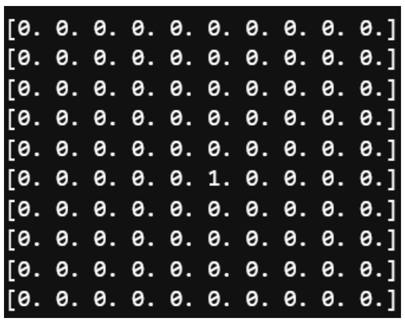




```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 1. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

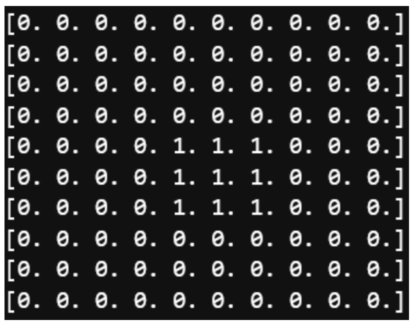
The towers matrix represents the locations of towers to be placed. The value of a cell is one if it is containing a tower, it is zero otherwise.

Later, on a copy of the house matrix, we apply a 3x3 kernel (a matrix full of “1” s for the first situation, and full of 0.5 s except the central cell which is containing a “1” for the second situation). We simply create that three-by-three matrix (if the tower is located at the edges or corners, the shape of the matrix changes according to the situation) and then, we subtract that from the house matrix, but limiting its minimum values to “0”. The result is the houses that do not receive full service.



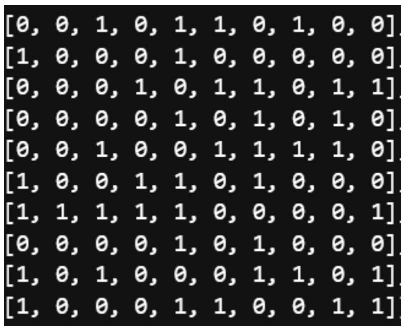
```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Example towers matrix



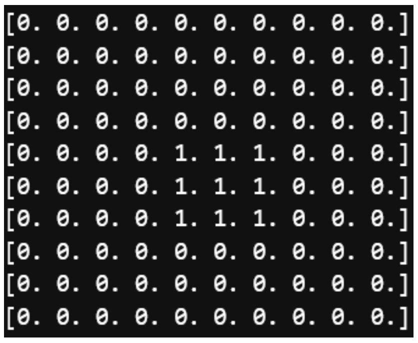
```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 1. 1. 0. 0. 0.]
[0. 0. 0. 0. 1. 1. 1. 0. 0. 0.]
[0. 0. 0. 0. 1. 1. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

The resulting kernel



```
[0, 0, 1, 0, 1, 1, 0, 1, 0, 0]
[1, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 1, 1, 0, 1, 1]
[0, 0, 0, 0, 1, 0, 1, 0, 1, 0]
[0, 0, 1, 0, 0, 1, 1, 1, 1, 0]
[1, 0, 0, 1, 1, 0, 1, 0, 0, 0]
[1, 1, 1, 1, 1, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 0, 1, 0, 0, 0]
[1, 0, 1, 0, 0, 0, 1, 1, 0, 1]
[1, 0, 0, 0, 1, 1, 0, 0, 1, 1]
```

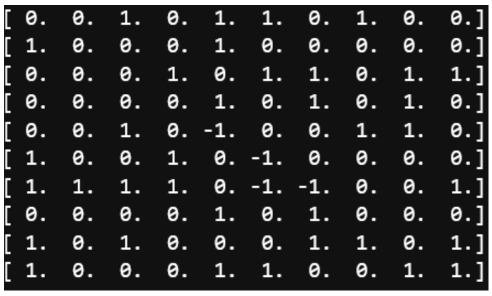
−



```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 1. 1. 0. 0. 0.]
[0. 0. 0. 0. 1. 1. 1. 0. 0. 0.]
[0. 0. 0. 0. 1. 1. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

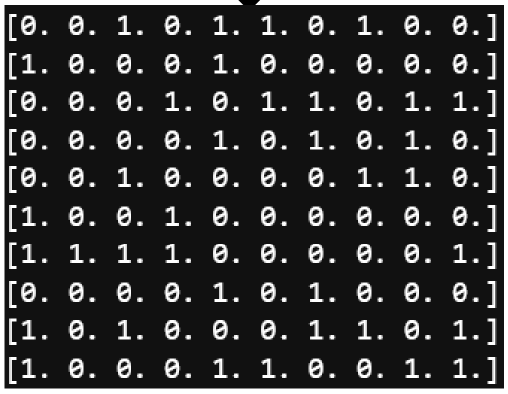
House Matrix

=



```
[ 0.  0.  1.  0.  1.  1.  0.  1.  0.  0.]
[ 1.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  1.  0.  1.  1.  0.  1.  1.]
[ 0.  0.  0.  0.  1.  0.  1.  0.  1.  0.]
[ 0.  0.  1.  0. -1.  0.  0.  1.  1.  0.]
[ 1.  0.  0.  1.  0. -1.  0.  0.  0.  0.]
[ 1.  1.  1.  1.  0. -1. -1.  0.  0.  1.]
[ 0.  0.  0.  0.  1.  0.  1.  0.  0.  0.]
[ 1.  0.  1.  0.  0.  0.  1.  1.  0.  1.]
[ 1.  0.  0.  0.  1.  1.  0.  0.  1.  1.]
```

Clipping the minimum value to zero

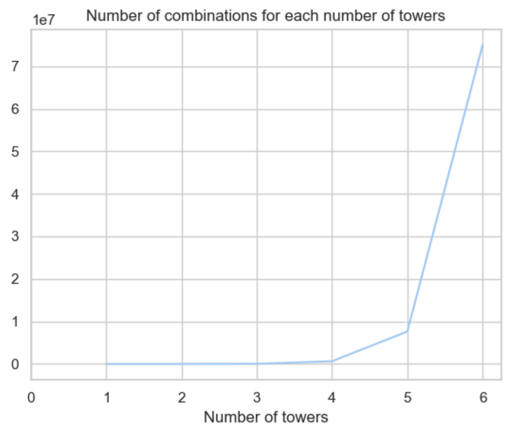


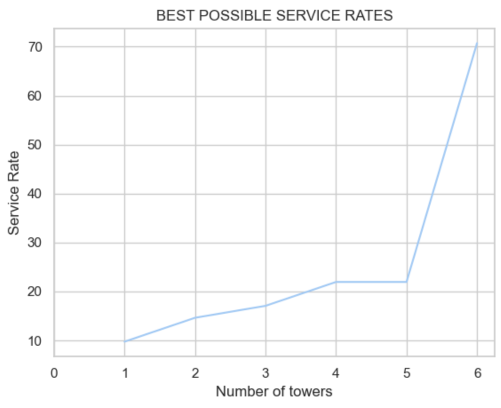
```
[0. 0. 1. 0. 1. 1. 0. 1. 0. 0.]
[1. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 0. 0. 1. 0. 1. 1. 0. 1. 1.]
[0. 0. 0. 0. 1. 0. 1. 0. 1. 0.]
[0. 0. 1. 0. 0. 0. 0. 1. 1. 0.]
[1. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
[1. 1. 1. 1. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 1. 0. 1. 0. 0. 0.]
[1. 0. 1. 0. 0. 0. 1. 1. 0. 1.]
[1. 0. 0. 0. 1. 1. 0. 0. 1. 1.]
```

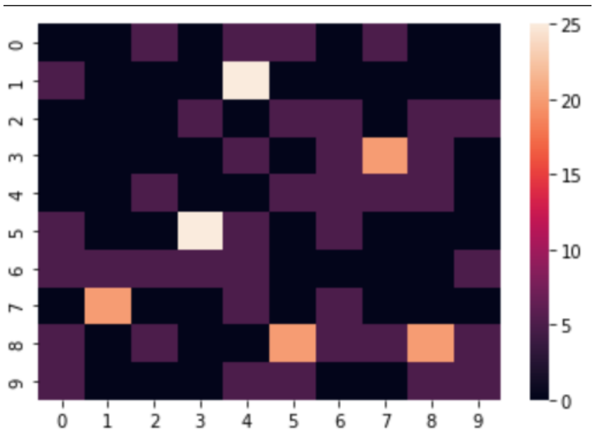
Result matrix containing the remaining houses

Finding the minimum number of towers by brute forcing

In order to find the minimum number of towers needed for the requirements, we decided simply to try to brute force the problem and try every possible case. After programming this, we realized that it consumes a huge amount of memory even with a small number of towers like six, since we were trying to place towers at every coordinate except the edges, our computer was only able to produce a result for six towers. We tried to optimize the process as much as possible by using 8-bit unsigned integers, but it did not solve our problem since it roughly takes $C\frac{64}{n}$ (n combination of 64) where n is the number of towers. Only answer we could receive by that method was the solution of problem one, which is surely six since our program tried every combination. We also visualized the results of this approach in order to make it easier to understand.







The white cells represent the cells with a house and a tower.

Brown represents towers.

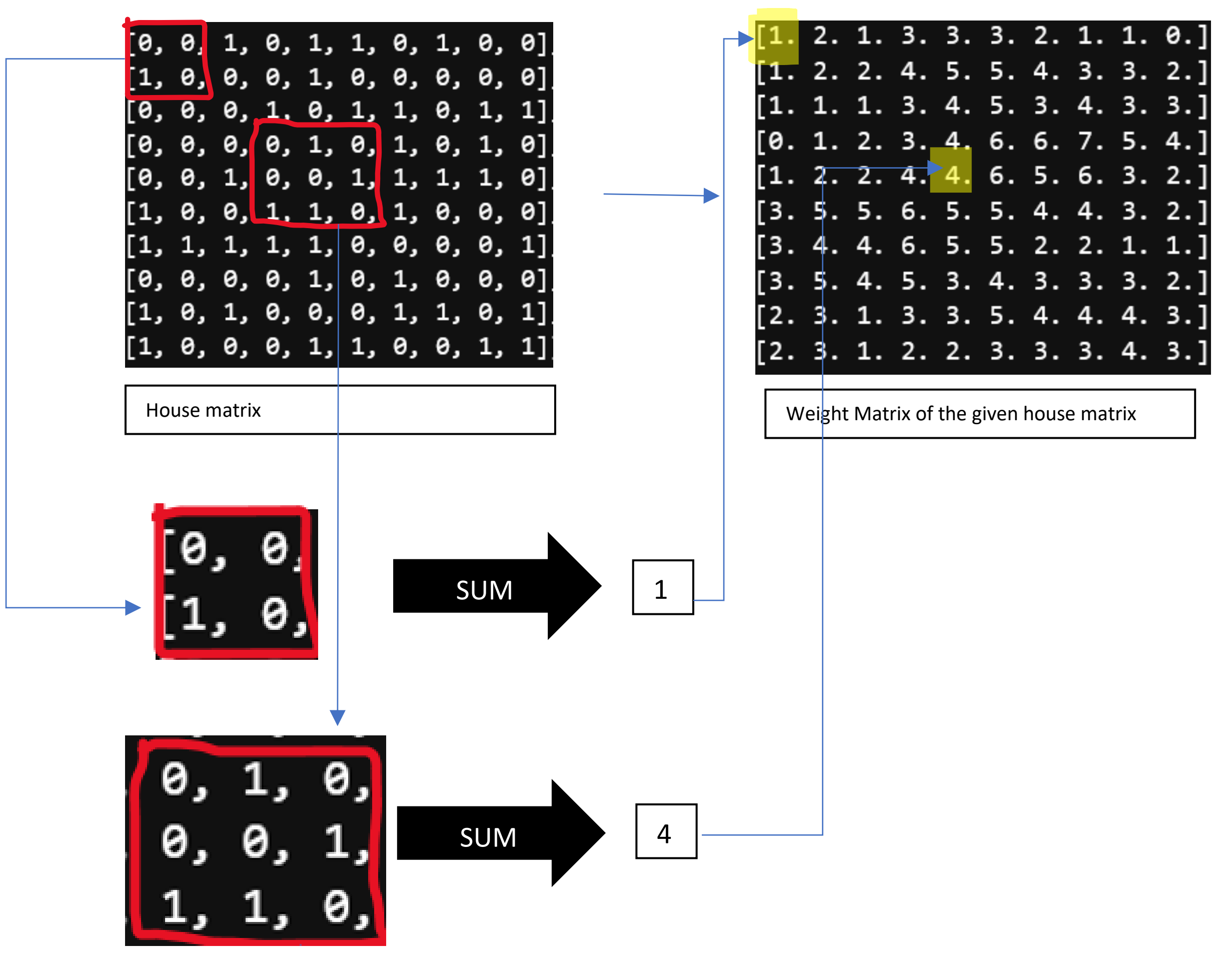
Purple cells represent houses

Finding the minimum number of towers by using weight matrices

After the results of our first approach, we realized that even if we could solve this problem by optimizing further, it would not be applicable to any other real-life situation which would definitely be a lot bigger. So, we decided to find an innovative approach and after considering how a human would try to solve this problem, we produced the solution we named weight matrices. In this approach, we simply give each cell a weight value, which is the sum of the service rates it would provide to houses and placing towers at the highest weighted cells.

Creating the weight matrices

To create the weight matrix, we traverse every cell of the house matrix and for the first situation, we set the value at the given index of the weight matrix to the sum of the surrounding cells. For the second situation, if its central cell contains a house, we add one, and for every cell around it that contains a house, we add 0.5.



```
[0, 0, 1, 0, 1, 1, 0, 1, 0, 0]
[1, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 1, 1, 0, 1, 1]
[0, 0, 0, 0, 1, 0, 1, 0, 1, 0]
[0, 0, 1, 0, 0, 1, 1, 1, 1, 0]
[1, 0, 0, 1, 1, 0, 1, 0, 0, 0]
[1, 1, 1, 1, 1, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 0, 1, 0, 0, 0]
[1, 0, 1, 0, 0, 0, 1, 1, 0, 1]
[1, 0, 0, 0, 1, 1, 0, 0, 1, 1]
```

House matrix

```
[1. 2. 1. 3. 3. 3. 2. 1. 1. 0.]
[1. 2. 2. 4. 5. 5. 4. 3. 3. 2.]
[1. 1. 1. 3. 4. 5. 3. 4. 3. 3.]
[0. 1. 2. 3. 4. 6. 6. 7. 5. 4.]
[1. 2. 2. 4. 4. 6. 5. 6. 3. 2.]
[3. 5. 5. 6. 5. 5. 4. 4. 3. 2.]
[3. 4. 4. 6. 5. 5. 2. 2. 1. 1.]
[3. 5. 4. 5. 3. 4. 3. 3. 3. 2.]
[2. 3. 1. 3. 3. 5. 4. 4. 4. 3.]
[2. 3. 1. 2. 2. 3. 3. 3. 4. 3.]
```

Weight Matrix of the given house matrix

```
[0, 0,
[1, 0,
```

```
0, 1, 0,
0, 0, 1,
1, 1, 0,
```

Creating the new house matrix

After creating the weight matrix, we simply select the coordinate that contains the highest weight in the weight matrix and if there is no tower at the tower matrix at that coordinate, we add a tower to the towers matrix and make the corresponding subtraction operation on the house matrix. This subtraction operation is subtracting “1” from the cells that contain a house if the service provided is “1”. If the service provided is “0.5”, we subtract that value from that cell.

The Loop

We simply repeat this procedure until we reach the point where the number of cells that contain non-zero values in the house matrix is lower than

$$1 - (n * r)$$

Where n is the total number of towers and r is the minimum service rate

After the loop ends, the sum of the tower matrix gives us the number of towers needed for that solution

Multiple Maximum Weights Case

There are cases where the weight matrix has multiple occurrences of its maximum value. In that case, we simply get the first one that do not contain a tower. This is not a good approach and we would create another inner loop there that calculates the best solution for each of them and comparing their results, simply creating a tree model and computing their result and selecting the best solution among them, but since the time is short and we were not sure we could handle it, we decided to go for a simpler solution on that case. So we are not sure if our program produces the best solution but even if it does not, it produces a solution very near to the best since if those two maximums do not intersect, the other maximum would still be the maximum on the next step of the loop.

The Resulting Matrices

After the loop, we get a new matrix that contains the houses that do not have full service. Houses are represented in their coordinates, and their values are "1" if they do not get any service and "0.5" if they only get half service. Apart from that, we have a towers matrix that contains the value "1" on the coordinates of placed towers. The solution found in that case is not only the minimum towers needed for the case, but also is one of the best solutions with that number of towers.

Visualization

To visualize the results, we created heatmaps of sum of towers and houses, we simply multiplied the towers by twenty and houses by five to make it clearly visible. We also created another heatmap for the houses that do not receive full service. Our solutions can be examined from those maps easily.

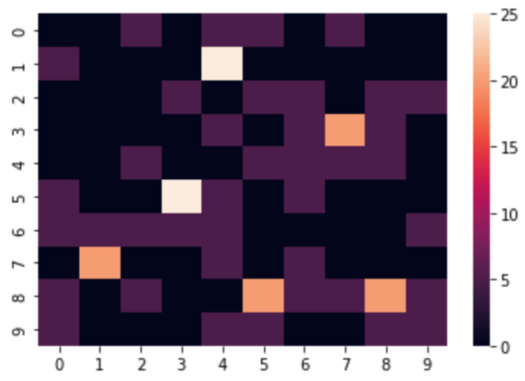
Results

First Situation

A cell tower provides full service to all adjacent cells and the cell its placed on, so it provides service to nine cells. We were asked to find the least numbers of towers needed to provide service to seventy percent and ninety percent of the houses. So we need to provide service to at least 29 houses to acquire seventy percent, and 37 houses to acquire ninety percent. Simply resulting houses matrix needs to have at most 12 houses for seventy percent and 4 houses for the ninety percent rate.

First Situation with seventy percent service rate

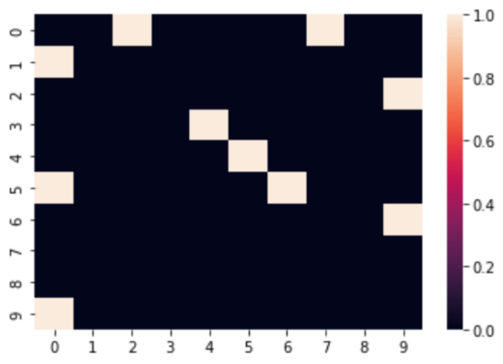
We have already found out that least number of towers needed to provide seventy percent service rate in the first scenario is six in our first approach, second approach provided the same result.



White cells represent the cells that contain a house and a tower

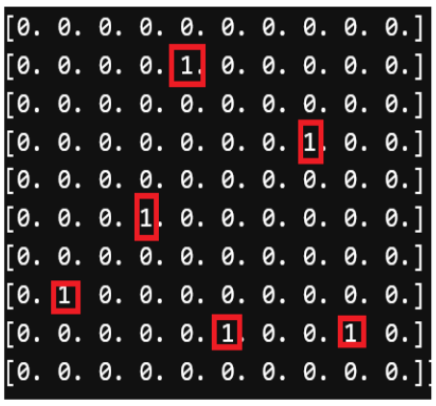
Orange cells represent towers

Purple cells represent houses



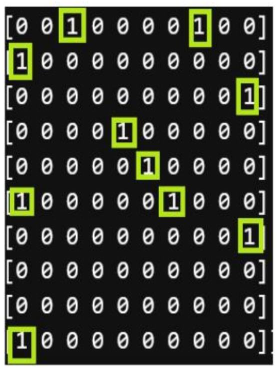
White cells represent houses that do not receive full service

Towers

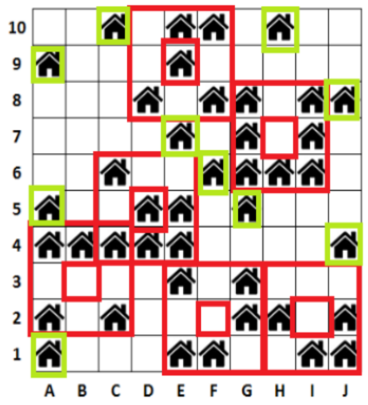


```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 1. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

Houses without service

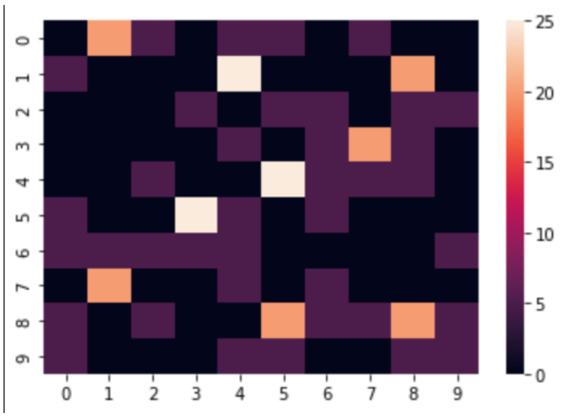


```
[0 0 1 0 0 0 0 1 0 0]
[1 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0]
[1 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0]
[1 0 0 0 0 0 0 0 0 0]]
```



First situation with ninety percent service rate

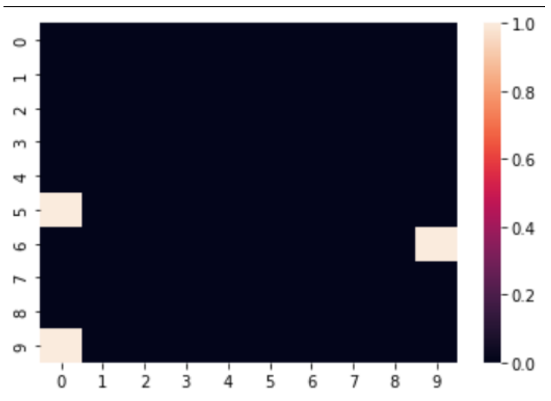
We have found out that least number of towers needed to provide ninety percent service rate in the first scenario is nine. It was not possible for our first approach to find that since it would take days to compute even if we had enough memory (the computer we were working on had 32GB's of memory and even that was not enough.). There are 27 540 584 510 different tower combinations for that number of towers and even if we were using 8 bit integers to store the coordinates of towers (it would take at least two integers to store a coordinate), we would need nearly 55 Gigabytes of ram only for the coordinates, even without considering the memory needed for matrices and operations, it's a huge amount of memory for such a small problem and it would take days of computation. With our second approach, it took seconds to provide a result.



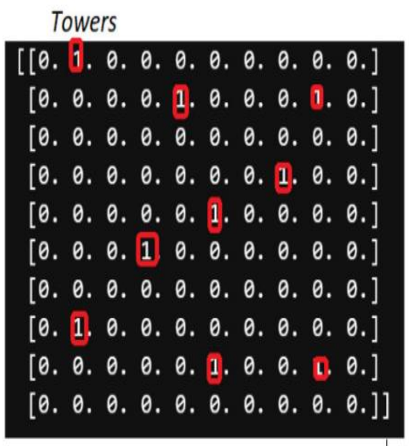
White cells represent cells that contain a house and a tower

Orange cells represent towers

Purple cells represent houses

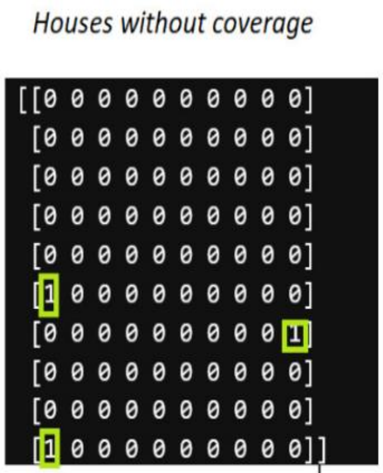


White cells represent houses that do not receive full service



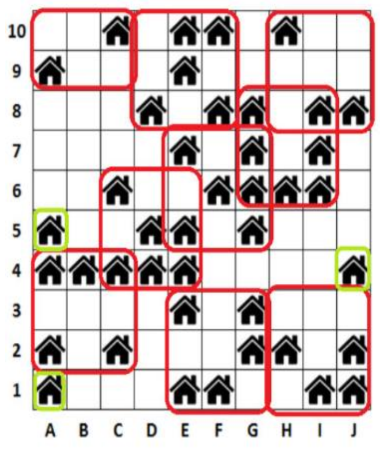
Towers

```
[[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```



Houses without coverage

```
[[0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0]]
```

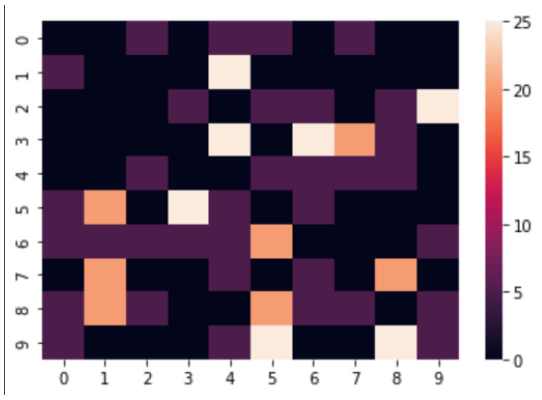


Second Situation

A cell tower provides full service to the cell it was placed, other 8 cells adjacent to it get only half service. We were asked to find least numbers of towers to provide full service to seventy percent of the neighbourhood and ninety percent of the neighbourhood. So we need to provide full service to at least 29 houses to acquire seventy percent, and 37 houses to acquire ninety percent. Simply resulting houses matrix needs to have at most twelve houses for seventy percent and 4 houses for the ninety percent rate.

Second situation with seventy percent service rate:

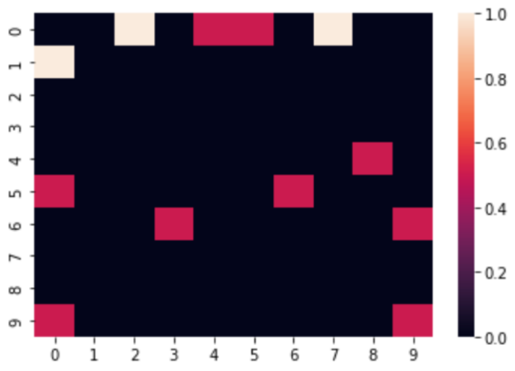
We have found out that least number of towers needed to provide full service to at least seventy percent of the houses is fourteen. We reached this solution the approach we described above, we simply edited how our program calculates weights and how it changes the houses matrix after adding a tower. It subtracts 0.5 instead of one from the cells around the central cell that contains the tower.



White cells represent the cells that contain a house and a tower

Orange cells represent towers

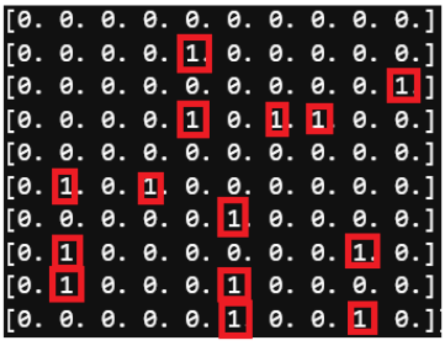
Purple cells represent houses



White cells represent houses that do not receive any service

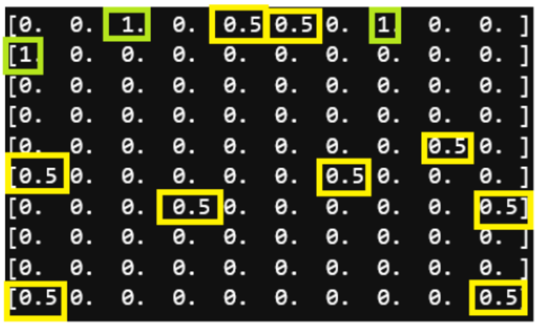
Red cells represent houses that receive only fifty percent service

Towers

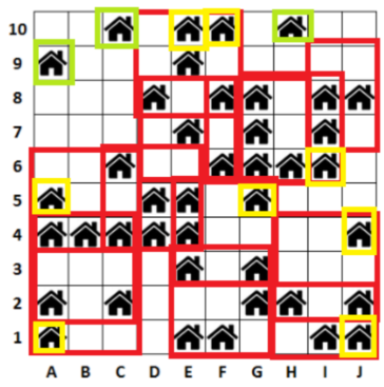


```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 1. 0. 1. 1. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 1. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 1. 0. 0. 0. 1. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 1. 0.]]
```

Houses without full service

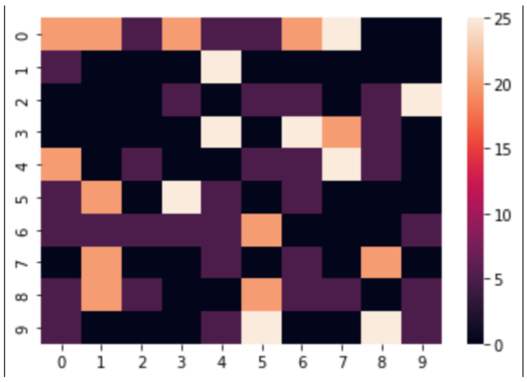


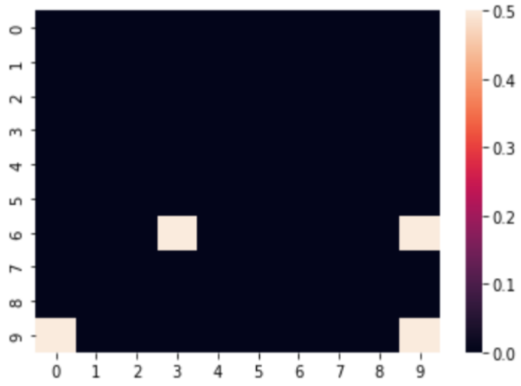
```
[0.  0.  1.  0.  0.5 0.5 0.  1.  0.  0. ]
[1.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
[0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
[0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
[0.  0.  0.  0.  0.  0.  0.  0.  0.5 0. ]
[0.5 0.  0.  0.  0.  0.  0.5 0.  0.  0. ]
[0.  0.  0.  0.5 0.  0.  0.  0.  0.  0.5]
[0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
[0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
[0.5 0.  0.  0.  0.  0.  0.  0.  0.  0.5
```



Second situation with ninety percent service rate

We have found out that least number of towers needed to solve this problem is twenty-one by using the same method. It again only took seconds for the program to provide this result. It would take eighty-two petabytes of memory if we only decided to save all the possible coordinates and that would cost almost 41 000 2TB hard disks to try to store that in a hard disk. That would cost around two million dollars to buy them. So, our second solution, even though it has some flaws which will be described, has a huge advantage against the brute force approach.





White cells represent cells that contain a house and a tower

Orange cells represent towers

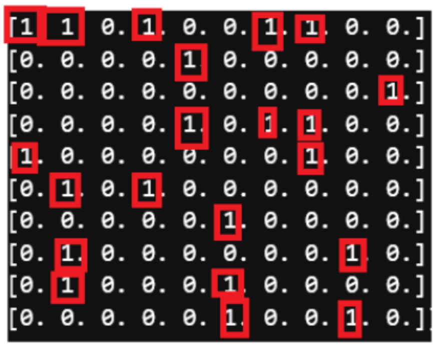
Purple cells represent houses

White cells represent houses that do not receive any service

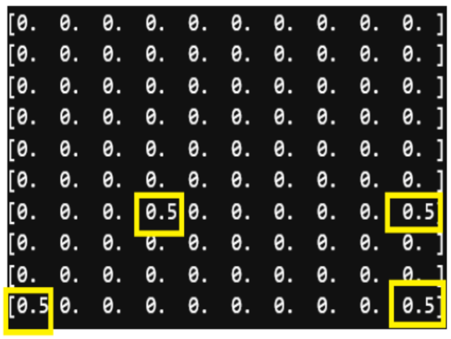
Red cells represent houses that receive only half service

Houses without full service

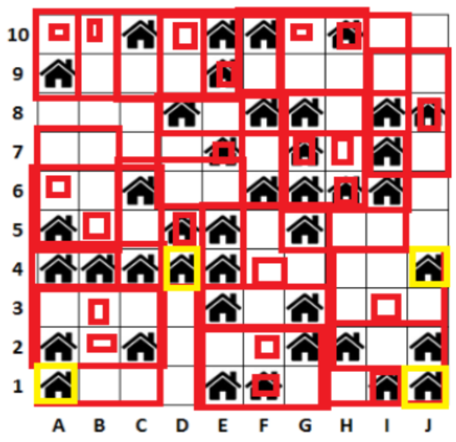
Towers



```
[1  1  0. 1. 0. 0. 1. 1. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 1. 0. 1. 1. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
[0. 1. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 1  0. 0. 0. 1. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 1. 0.]]
```



```
[0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
[0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
[0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
[0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
[0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
[0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
[0.  0.  0.  0.5 0.  0.  0.  0.  0.  0.5]
[0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
[0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
[0.5 0.  0.  0.  0.  0.  0.  0.  0.  0.5]
```

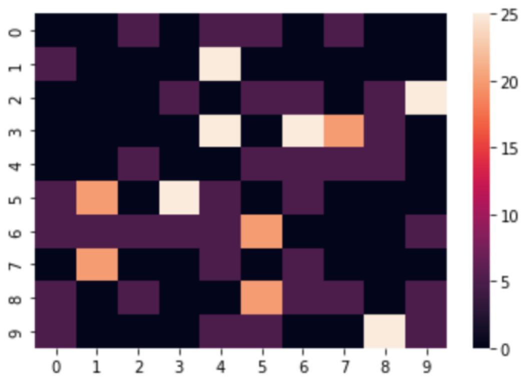


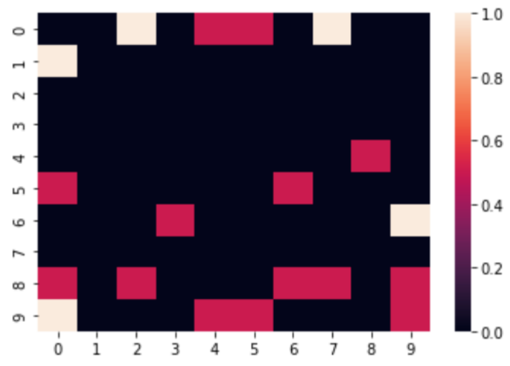
Third Situation

There was a misunderstanding in the second situation, we are not sure if the houses that receive fifty percent service should be considered as 0.5's so two houses that have half service should be considered as one house or not. We assumed that they do not and created our program on that assumption. In case we were wrong, we have edited our program and considered this as third situation. In that situation, our solution needs to provide seventy percent and ninety percent to the all neighbourhood. So sum of our resulting hoses matrix needs to be at most 12 for the seventy percent rate and 4 for the ninety percent rate

Third situation with seventy percent service rate:

We found out that least number of towers needed to provide seventy percent service rate to the neighbourhood is eleven.





White cells represent cells that contain a house and a tower

Orange cells represent towers

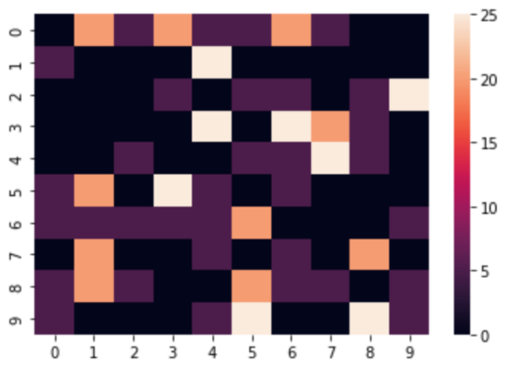
Purple cells represent houses

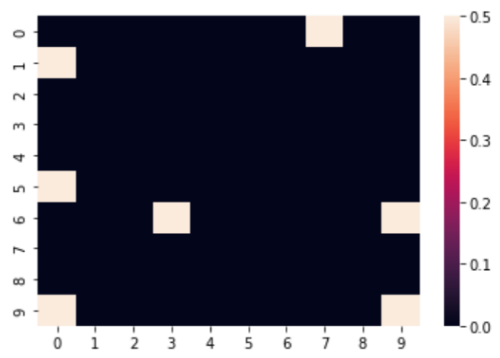
White cells represent houses that do not receive any service

Red cells represent houses that receive only half service

Third situation with ninety percent service rate:

We found out that least number of towers needed to provide ninety percent service rate to all of the neighbourhood is 18.





White cells represent cells that contain a house and a tower

Orange cells represent towers

Purple cells represent houses

White cells represent houses that receive fifty percent service

Possible flaws of our solution

Since we did not have the time or resources to evaluate our approach, we are not completely sure that our solutions are correct (except the first one since we tried every combination). A flaw we could imagine is at the point where we select the cell to place a tower by using the highest weights. We are not sure if it is possible but there might be cases that placing towers at the cells with highest weights from the beginning might not provide the best result, but we are not sure if its possible or how to define that clearly.

Unconsidered Aspects in This Problem

While this problem could act as a great model for a real-life scenario, there are many unconsidered aspects of this problem in real world. For example, buildings in a neighbourhood would not share the same importance for a cell phone company since they might have different number of houses inside them, some of them could be apartment buildings that contain multiple houses while others could be single houses. In that scenario, in the calculation step, we would give different numbers to the buildings due to their population. Also, in a real-world scenario, probably we would not directly have a least number of houses that needs to be served, we would have a cost for building a cell tower and we would try to find an optimal point according to that price, so we would not place a cell tower to serve to a single house to get the minimum rate we want, we would instead have a cost function and another function that would try to numerically explain our gain by providing service to a house. That function could be revenue if we were trying to maximize profit. It also could be the number of houses or people who got served if we are trying to maximize that instead of revenue for a different purpose. Also, the fact that the signal power would decrease after passing through a body of a building or a tree and would travel further if it is travelling through an empty cell and this is not considered eighter. Even though there are many unconsidered aspects, this model could be improved to simulate those other aspects and it could be used in a real-life scenario after those improvements.

Source Code

The source code of our solution can be examined on https://github.com/1942Spectre/IE120-GROUP-Q-Cell-Towers-Problem

References

- Harris, C.R., Millman, K.J., van der Walt, S.J. et al. *Array programming with NumPy*. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2. (Publisher link).

- J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.

- Waskom, M. L., (2021). seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021, https://doi.org/10.21105/joss.03021