CS201-Project Report
Team Vectors (8)

Nampally Pranav
190010026

Karthik Kancharla
190020020

T Satwik
190030043

1 Description of the Algorithm

Our basic idea was to minimize the total area covered by the ranges of the factories. Since the discontent of each house is unknown and is equally likely to have any value between [0, 255], we can say that expected value of the discontent caused by a factory is same everywhere inside the grid. So to minimize the total area covered by the ranges of the factories, we can place the factories at corners and the edges of the grid. By placing the factories in the corners, we can reduce the area covered by 75% and by placing them in sides we can reduce the area by 50%.

When the corners and sides are completely occupied by the factories without overlaps, we can again place remaining factories at the corners, though it leads to overlaps, the discontent is reduced by 75%, hence the total discontent might be less. However if the number of overlaps are sufficiently large, then the total discontent can increase as well. Hence we have run the code multiple times to find the optimal number of factories that can be placed at the corners, for each test case.

To do this, we have defined 2 functions calculate() and flag_increment(). The calculate function gave us the discontent produced by the factory when placed at some coordinates and the flag_increment gave a 2-D list, which contains the number of factories whose effect is on that particular coordinate. This 2-D list acted as visual representation of how the factories are being placed.

Algorithm:
Let n be the number of factories, whenever we finalize the coordinates of a factor, we insert the coordinates to a list named coord_list at appropriate index.
If $n \leq 4$, then we only place them in the corners, We start with the factory with the largest range, calculate the discontent value, and place it at the corner which has the minimum value. Then we do the same thing for the next factories also, by checking that the corner is not repeated.

If $4 < n \leq 8$, then we only place them in the corners and in midpoints of the sides,
We start by placing the largest 4 factories in the corners as we did in the above case. After Placing them, we place the remaining factories in the midpoints in the same way as we did for corners.

If $8 < n$, then we place the factories in the corners then in all the edges and then the remaining factories in the corners again,
We start by placing the largest 4 factories in the corners(as done above),
Then we alternatively place the factories in the top and the bottom edges. We did this by using while loop and a

variable(x) which traverse through the first line and we append the coordinates to our coordinates list, whenever we place a factory, this while loop breaks the x reaches to the end of the grip (i.e when x reaches or exceeds w). Then we alternatively placed the factories in the left and right edges too (like we have done above).

Then, we placed the remaining factories uniformly distributed in the four corners using x%4 and iterating through the remaining factories.
Please refer to the code provided under the section 'Code', to understand more about our algorithm.

We have applied the above algorithm for the 5th evaluation of our project, where we got the total discontent of 170,277,475.

For the 6th evaluation of the project we have used the approach which was sort of like dynamic programming. Here, we analyzed the given test cases, found the n value and the ranges of factories for each test case, and then minimized the total discontent for each test case, by optimizing the number of factories that are placed at corners by brute force methods. Also, for each test case, we placed the biggest factories and smallest factories at the corners to minimize the overlap area at the corners and the rest were placed as usually at the 4 sides of the total area. In this way, we got the best code for each of our test cases to minimize the total discontent.

2 Approach Code:

This the Github link for the code for this project. `https://github.com/ryanpatrick8055/DSA_Project.git`
The file named "readablecode.py" contains the code with appropriate comments.

3 Complexity Analysis:

The calculate and the flag_increment() functions, have the worst case time complexity as $O(w \cdot h)$, where w and h are the given width and height respectively.
The worst case is when a range is greater than or equal to w or h (whichever is highest), In this case, we need to iterate over each of house in the given grid, hence the complexity is $O(w \cdot h)$
We call these functions at most n times, hence the worst case time complexity for our code is $O(n \cdot w \cdot h)$, where n is number of factories, w is the width of the grid and h is the height of the grid.

The average execution time for the final code in the optil server = 3.75 seconds.

4 Approach for each Week

4.1 Week 1

For the first evaluation of the project we were supposed get a valid output, not necessarily the best one.
Hence we have used the python random module, to generate a set of coordinates for each of the given factory and print them as output.
Though it wasn't required, we have tried to slightly optimize the output by not repeating the same coordinates.
This was done by using a while loop which iterates until a new coordinate which is not already present in the coordinate list is generated.

By using this approach of random coordinates, we have achieved a total discontent of 525,452,745.00

4.2 Week 2

For the Second Evaluation, we tried to write a code which generates custom test cases as they were not provided in the optil platform and to improve the discontent we have placed the factories with largest discontent in the corners.

We used the python random module to generate the number of factories, width, height within the bounds mentioned in the question. We have also generated a list of ranges for factories and 2-dimensional list of discontent. We have then written all these variables and lists to text file in the same format as mentioned in the question. This test cases generator code has provided us with a lot of flexibility in testing our algorithm in different cases. We also declared and used the calculate() function, which takes the discontent list, coordinate and the range as arguments and then returns the discontent caused by the factory having that range when placed at that particular coordinate.

We then tried to place the biggest factories in corners because, The factories with more ranges(radius) will have more area, hence covers more number of houses and leads to higher discontent. Moreover the overlaps are also more when the factories with large radius are placed in center(because of the increased occupied space). Hence to avoid this we have placed the factories with more range at corners, by applying this strategy we also can reduce 75% of the discontent caused by them.(considering that the discontent of each house is unknown)
Here, our idea was to sort the given factories based on their ranges and then, place the 4 biggest factories at the 4 corners of the total area which is a rectangle in this problem.
We optimized the way of placing factories using the following approach:

We calculated the discontent for each factory if it was placed at each corner(using the calculate function) and then placed that factory at the corner where we get the least discontent. Also, we made sure that, after placing a factory at one corner, the further factories will not be placed at that corner to avoid overlap which increases discontent.

So finally we have placed the first 4 of the largest factories in the 4 corners in such a way that it has the least discontent.

Then, we placed the remaining factories using the random module in python following the same process as in Week 1.

By using this approach of placing the factories in corners, we have achieved a total discontent of 303,476,263.00

4.3 Week 3

For the Third Evaluation, our idea was to find a way to visually interpret how the factories are placed and and to improve the discontent we placed the factories in the top and bottom edges after placing the largest factories in the corners.
To do this, we have declared a 2-Dimensional list of the same size as the given rectangle. This list has been initialized to 0 and we have declared a function called flag_increment(), which increments the value in the coordinates corresponding to those houses that fall in the range of a factory. (that is whenever we place a factory we increment the values corresponding to all the house which are within the range of that factory.) We can then print/write this 2-D list into a file and see how the factories are being placed. This flag_list is another form of heat map which we used to get the density of factories at each point instead of the colour. This has been useful for testing our code, and gave us a confidence that our code is doing exactly what we intended it to do. This function also gives the number of factories that are in the neighbourhood of a given house, hence we can get an idea about the number of overlaps. We have then placed the largest factories at the corners as we have done for the previous evaluation. and then we have tried to place the factories along the edges(top and bottom). We have placed the factories in order of their size, i.e larger factory is placed first). This strategy is useful because we don't know about the discontent at each point, hence if we consider that it is random(for a general case), by placing a factory on the edge rather than the center we can reduce the discontent by 50%. So we have placed the factories in the top and the bottom edges one by one, without overlaps, and rest of the factories have been placed randomly as done in the previous evaluation. When the number of edges is less than or equal to 8, then we have applied an algorithm similar to that in Evaluation2, but now we have placed the factories in the midpoints of all the edges after placing them in the corners.

By using this approach of placing the factories in corners and in the top and bottom edges, we have achieved a total discontent of 229,116,164.00

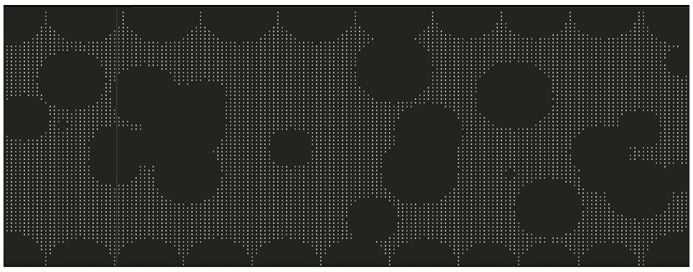

Figure 1: Visual Representaion of how factories are placed Week 3

4.4 Week 4

For the fourth evaluation, we improved our previous approach. Here, as done in 3rd evaluation, for any test case having more than 4 factories, we place the remaining factories in the top and bottom sides of the total area which is considered a rectangle, after that, remaining factories were placed in the left and right sides of the rectangle which will also decrease the discontent of each factory by half, considering that initially the random module placed the factories somewhere in the middle of the rectangle. After this we found a significant decrease of about 30 million in the total discontent.

After placing the factories at corners, then at the top, bottom, left and right sides of the rectangle, now all the remaining factories were again placed at the corners using random module in python. We know that these factories may overlap with initial factories at the corners and in between themselves, but this overlap happens at the max in 25% of the area of each factory, but considering the fact that discontent of each factory placed at the corner is decreased by 75% of it's initial discontent, also there will be lesser number of factories left to be placed at the corners using random module, as we already placed most of them in the sides of the rectangle. Using this, we observed a decrease of about 25 million in the total discontent. By using this approach of placing the factories in corners, we have achieved a total discontent of 176,851,713.00

This week's final code still has a bit of randomness while placing the factories at the corners after placing most of them already in the sides. So, there would be a variation of about 4-5 million in the total discontent when we run the code in optil.

4.5 Week 5

In the fifth evaluation, a small modification was done to the previous evaluation's code to lower the total discontent. Here, just as in fourth evaluation, the factories were now placed in the coordinates of first and last lines alternatively and then the remaining factories were placed in the right and left sides alternatively, so as to minimize the intersections if any.

This was done just by initializing a variable to a value before the while loops for first and last and run the code for placing factories in first line if the variable was odd, or else run the code to place the factories in the last line. And the value of the variable was incremented in each iteration. This helped in placing the factories alternatively. Similar approach was used for placing the factories alternatively in right and left side.

By using this approach of placing the factories in corners, we have achieved a total discontent of 170,277,475.00

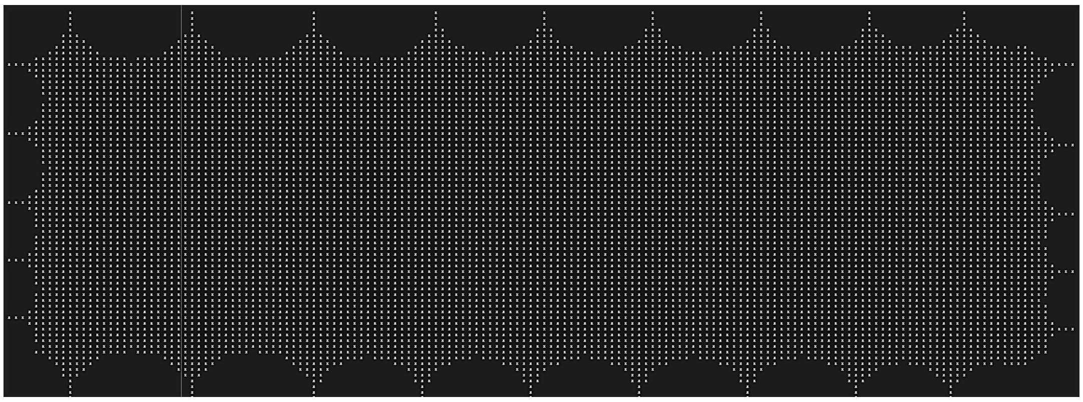

Figure 2: Visual Representaion of how factories are placed - Week 5

4.6 Week 6

Considering the 6th evaluation is the final evaluation with maximum percentage, our objective was to give our best outcome and reduce the discontent as much as we can.

We have run over 200 optil runs with various approaches until now, hence we had a lot of data. We have copied all the data into a excel sheet and analyzed how the discontent values changes for each algorithm for each of test cases.

To minimize the total discontent, we have minimized the discontent produced by each test case. We have done this by finding the n values for test cases and used them to specifically use different approach for different test cases.

For example we have observed that for the 4th test case, the discontent produced by using the algorithm used in week 5 is about 17 million, but by using another algorithm where we didn't place the factories in the sides and placed them at corners, the discontent reduced for this test case, but it wasn't a good algorithm for other test cases, Hence we have specifically used that algorithm for 4th test case specifically, hence bringing its discontent down to 11 million and then also optimizing the number of factories to be placed at the corners made it's discontent to 7 million.

Similarly, for some other test cases too, we have optimized the number of factories which are to be placed in corners so as to reduce the number of overlaps, and produce the the least discontent.

Also, in this submission, we have removed the randomness of the code which places the left out factories at corners randomly, after placing the remaining factories at all sides of the total area. Instead of using random module for the left out factories, we placed one factory after the other at each corner which not only removes randomness, also makes sure that our left out factories were equally distributed among the corners which also means that excessive overlapping doesn't occur at a specific corner.

We have optimized the discontent for each test case, and collectively reduced the total discontent to 151,779,784.00.

4.7 Total Discontent for each week

Week	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
Total Discontent	525,452,745	303,476,263	229,116,164	176,851,713	170,277,475	151,779,784

Table 1: Total Discontent in all Weeks

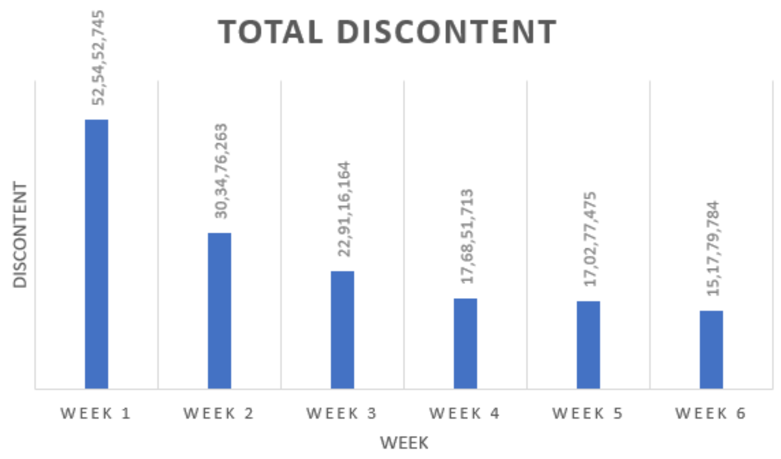

Figure 3: Variation of Total Discontent

5 Choice of data structures

We have used python lists for almost all the the tasks in our code. This is because lists are dynamic and can be resized easily, and the time complexity to access, append and set the values is O(1), hence it makes the code very efficient. Python list are bad for insertions of elements as it take O(n) time, but we haven't used any insertions into lists and hence Python lists are optimal choice for this code.

6 Theoretical observation on the problem

This is a problem involving Dynamic Programming as it tries to optimize the problem by breaking it down into simpler sub problems and utilizing the fact that the optimal solution to the overall problem depends upon the optimal solution to its sub problems. We can analyse this problem as finding the best possible position for each factory, thereby reducing the discontent caused by it. When we reduce the discontent of each factory, up to some extent we can reduce the the total discontent also, hence it can be thought of as a case of dynamic programming.
To visualize the distribution of discontent, a 2-D grid using lists could be created to make a heatmap or something similar to get the density of factories at each coordinate.

7 Ideas for further optimization of the code

1)We have placed some factories in sides of the rectangle. Our idea was to calculate all the permutations of the list of factories on each side and then place the factories on each side such that it gives the minimum discontent, but the time complexity for this approach would be very high, as it would take a lot of time to compute the permutations, We tried to implement this approach by using heap's algorithm for generating permutations, but it still gave an RTE, as the n was more and and time required was very large.

2)Another idea would be to take factories placed at the corners which have been overlapped and place that factory at a point at the central area of the rectangle where the discontent of people is minimum. We need to follow this until the total discontent decreases from initial.