Q1

The first problem finally uses the idea of Dij to find the shortest path. Floyd has been tried before, but this method causes excessive space usage (mle). There are two key points in this problem: 1. To translate the problem as the shortest path problem, my understanding here is that each lattice point can be imagined to have an edge connected with the surrounding four nodes, if it is in line with the direction of the letters stored in each point, then I think its weight is 0, and the weight of the remaining edges is 1. Based on this, we look for the minimum value from i to j. 2.The second key point is how to store the distance from each cell point to i in the current case. At first, I tried to use a dictionary, but it used too much space, so I changed the dictionary to list and simplified the step of using the dictionary to store the neighbors of each node, trying to reduce the space occupation by this. As for the final solution, the specific idea is as follows: First, the input string is pieced together to facilitate the subsequent reading of the letter at this position according to the index. Then, using the Dij method in class, create a list to store the distance of each point, and create a priority queue to pop up the node with the smallest distance in the current situation. After the node is obtained, it traverses its neighbors and updates the distance. Implementing the full AC of this question, especially using python, is really not an easy thing.

Q2

For the second question, the overall idea is very clear. Let's consider creating a list and continuing to nest lists within it. For convenience, I'm going to leave out index=0. The small list of index = i stores the neighbors of vertex whose ID is i. The first initialization is to create a number of empty small lists in the large list, and read all the input, and initially fill in the small list. For each small list taken out, first expand it by k times, and then find the intersection with the original list. For the numbers contained in the intersection set, read the corresponding small list in the big list according to index, and check whether the small list contains other elements in the intersection. If it does not, increase the number of each of the two elements in the small list. Therefore, after the convenience is completed, the problem requires completion and can be output in accordance with BST.