Problem Statement

Mrs. Qiu is planning to practice orienteering. The area where she'll practice is a rectangular field divided into unit squares. You are given its description as a String[] field. Each character in field is '.' (a period), '*' (an asterisk), or '#' (a number sign). Each '.' represents a passable square without a checkpoint, each '*' represents a passable square with a checkpoint, and each '#' represents an impassable obstacle. It is guaranteed that all passable squares (i.e., all '.'s and '*'s) form a 4-connected tree (see notes for formal definition). The number of checkpoints is at most 300.

In order to practice, Mrs. Qiu chooses K of the checkpoints uniformly at random. Afterwards, she will find the shortest sequence of squares that passes through all chosen checkpoints. The sequence can start at any square, end at any square (possibly other than the starting one), and visit each square any number of times. Each pair of consecutive squares in the sequence must have a common side. The length of the sequence is the number of moves Mrs. Qiu will have to make. (So, for example, a sequence that consists of 7 squares has length 6.)

You are given the String[] field and the int K. Return the expected length of Mrs. Qiu’s sequence.

Definition

Method signature: def expected_length(field: List[str], K: int) -> float

Notes

- A set S of squares is said to form a 4-connected tree if for any two squares A and B from S, there exists exactly one way to walk from A to B while visiting only the squares from S and not visiting the same square more than once. From a given square, it is possible to walk into any square that shares a common side with it.

Constraints

- field will contain between 1 and 50 elements, inclusive.

-    Each element of field will contain between 1 and 50 characters, inclusive.
-    Each element of field will contain the same number of characters.
-    Each character in field will be '*', '.', or '#'.
-    '*' and '.' form a 4-connected tree.
-    K will be between 2 and 300, inclusive.
-    field will contain between K and 300 '*', inclusive.

Examples

Example #1:

```
field = [
 "*#..#",
 ".#*#.",
 "*...*"]
K = 2
Returns: 3.8333333333333353

Explanation:
Let (i,j) be the square represented by the j-th character of the i-th
element of field (both numbers are 0-based).

If she chooses (0,0) and (1,2), one of the optimal sequences is (0,0)
-> (1,0) -> (2,0) -> (2,1) -> (2,2) -> (1,2).
If she chooses (0,0) and (2,0), one of the optimal sequences is (0,0)
-> (1,0) -> (2,0).
If she chooses (0,0) and (2,4), one of the optimal sequences is (0,0)
-> (1,0) -> (2,0) -> (2,1) -> (2,2) -> (2,3) -> (2,4).
If she chooses (1,2) and (2,0), one of the optimal sequences is (1,2)
-> (2,2) -> (2,1) -> (2,0).
If she chooses (1,2) and (2,4), one of the optimal sequences is (1,2)
-> (2,2) -> (2,3) -> (2,4).
If she chooses (2,0) and (2,4), one of the optimal sequences is (2,0)
-> (2,1) -> (2,2) -> (2,3) -> (2,4).
If she chooses (0,0), (1,2), and (2,4)
So the expected length of her sequences is:
  (5 + 2 + 6 + 3 + 3 + 4) / 6 = 23 / 6 = 3.8333333333333353
```

Example #2:

```
field = [
 "*#..#",
 ".#*#.",
 "*...*"]
K = 4
Returns: 8.0

Explanation:
Mrs. Qiu chooses all four checkpoints. One of the shortest sequences
is (0,0) -> (1,0) -> (2,0) -> (2,1) -> (2,2) -> (1,2) -> (2,2) ->
(2,3) -> (2,4).
```

Example #3:

```
field = [
 "#.#**",
 "....#",
 "#*#**",
 "**#*#",
 "#..##",
 "*#..#",
 ".#.#.",
 "....*"]
K = 3
Returns: 10.825000000000024
```

Example #4:

```
field = [
 "###################",
 "#*###############*#",
 "#.....#######.....#",
 "#*###*.#.*.#.*###*#",
 "#*####*.*#*.*####*#",
 "#*#####*###*#####*#",
 "###################"]
K = 9
Returns: 30.272233648704244
```

Example #5:

```
field = [
 "**###*.**#..#.*...*#...*#..#.##..#..#.#*...#.##*##.",
 ".#..###..#..#.#.##..#.#.*#.*..#..#.#*..##.#*...*..",
"..#.....###.#*.##..#.#.#*..#.#..#....#..#...#*####",
".#.##*#.*#..#*#*.#.#...*.#.*#.#.##.#*.##.#.#..*...",
"..*.*#*.###.#..#.#..##.##.*#..#.....#.....#..#.#.#",
".#.##.#..##..*#..#.#...#*##*#*..#.#.#.#.##.##.#.#*",
"..##....#..#.#*#...*.##...#.#.####...#.#*.....#...",
".#.*#.##.*#*.#*.#.#.#..#.#..#.#*#.###..##.##.#.##*",
".*.#*..*.#.#...#.*##.#.**.#.*...**..*#..#.#.#*.#..",
".#*.#*##....##.#.#*..*.###.#.##.##.#.#.#....#.#*.#",
"*.#..#*#.#*#*....#.#.#..*#**...##.#.#.**#*##.*.#..",
".#*.##..##..##.#.#..#.#.###.###...#...#*#..##*#.#.",
"#..#*.#..*.###..#.#...#.###.#.#*#.#.#**##.#...*.#*",
"..#..#.#.##.#..#.**.##*#.#**.**..#.#..#...#.##*#..",
".#*#.#.*..#.*#...#.#...#...#.##.#..*#*.##*....###.",
".*.#.#.#*#..*##.**.##*##..#.*#.#*###..*.#.##.#..",
".#......#...#.#.*#.#.#..#..#.#*#....#*.#*#.*#..*.#",
"#..####..#*#...#*.#..#.###...#.#.#.###*#..##*##.#.",
".#.*..#.#...#.#..#.##...#..#.#.#.#.###..##..*.*.*.",
".#.#.#.#..##.*..#.*.#.##.#..##*...#.#..#.#.##.#.##",
".#..#*.#.#..#.##..##..#.*..#.*#.#...##....#...###.",
".#.#.#.#*.#.#..#.#..#..#.#.*#...#.##...#.##.##.*..",
".#...#.#.##.#.#..*#.*#..###..#.#.#*###.##...#*.##.",
".#.##.*.......*.#.*#.#.#*###..*...*..#.*.##.#.#..#",
"...###*#####*#.#..##*...#..#..##.#.#.#..##*#*.*.*#.",
"#.#.#....*#..#.#.#.#.##..#*.#...#..#.#*#...#.##.*.",
"..*.#*##.#.#*#.###...#..##.#.#.#*####*#.*#.#.*###.#",
"##*##..##...#.....##.#.#.**#..#*.....##.#..#*.#.*.",
".....#.*.##..##.##*.*#...#.#.#.##.#*#.**..#..#.#.#",
"##.#.#*##.#.#.*.*.#.#*#.#.#....*...#*##*##.#....#.",
"*.**#**....*..##.#*.*.**..##.###.##.....##...##.**",
"#.####.##*#*##..#.*#*#.##*...#.##..#.##....#*..##.",
"....#...##.#...#*.#..##.##.#*..*.#....##.#.*##...#",
"#.#..*###*..#.#..#..#..#*....#.##..##.#*##.##.*##..",
"..#.#*.*.##.#.#*#.#*##.###.##...#...........#*.#.",
"#.#.##.#....*....*..##..*#.#.#.###.#.#.#.###..#..#",
".#**..#*#.#*#*#.#.#...*##....##.#*..#..#*..*#..#..",
"...#*#.....#..#.#..#*#.*##.#..#.#.##..#.*#*#.#...#",
".#*.###.#.#.#.#.*#*##.##..#.#*..#...#.#.#..#*.*#..",
"#*.#.#.#..#..#..#....*#.*##..##.#.#..#...##.#.#..#",
"*.#..#..#...#..##.#*#..#.#*#.#.#.###..#.#*...#.#..",
"#...#.#...#.#.#..#.*.#*.....**.*..#*##.#*.##....##",
"#*#....#*#..#.*.###*#..#*##.##.#.#...#.*.##.##.##.",
"..##*##*..#*#.#..#*.*##*.##.#...#.#.#.#.#..*#.##..",
"#...#*##.#*#**.##.*#.*.##..*.#*#**....#**##...*.*#",
 "*#.##......*#.##.#.#.##**.#.#.#.#.#.##..#...#*#*#*",
 "*....##.#.#..#.....#..##.#....*....#.#.##.#.#.##**",
 "#.##*#...#..#.#.##..#..##.##.##.##........##.#*#.#",
 "..#...#.#*#*..*#..*#.*#.#......##.#.#.#*#..#..****",
 ".###.#..#...#.#..#..#.#...#.#.#...**.#..*#*.*###*."]
K = 150
Returns: 1309.4951033725558
```