Problem: Our Word Finder

Given a MxN matrix of characters and a string word, your task will be to find the word in the given matrix. The rule is:

1. The word can be formed in any direction such as:
    a. Left to right
    b. Right to left
    c. Left to right diagonal
    d. Right to left diagonal
    e. Any combination above
    f. Same letter cannot be re-used to form the word
    g. Must follow any one of the above directions and their combination while forming the word.

Example:

Consider the following 3X4 matrix.

C	B	A	B
B	N	O	D
M	D	E	E

Let us try to find the following words:

"ABC" is available in the matrix:

C	**B**	**A**	B
B	N	O	D
M	D	E	E

BAOED is available in the matrix:

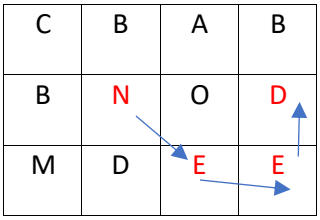
C	B	A	B
B	N	O	D
M	D	E	E

CNE is available in the matrix:

C	B	A	B
B	N	O	D
M	D	E	E

NEC is not available in the matrix

Need is available in the matrix:



NEEDON is not available in the matrix.

Input Format (Your code must read from standard inptut (no file I/o is allowed))

The first line of the input consists of 3 integers, M, N, and S. Here M = number of rows in the matrix, N = number of columns in the matrix, and S = number of words you need to find from this matrix.

Then M lines represent the rows input characters, where each line has N characters separated by space.

After M lines, there will be S lines of words. You need to find these words in the matrix.

Similarly, K test cases will be provided in the above format.

Sample input1:

```
3 4 6
C B A B
B N O D
M D E E
ABC
BAOED
CNE
NEC
NEED
NEEDON
```

Output Format (Your must use standard console output to display your result)

Please get an idea of the output format from the following output for the above input. Also, you can probably find the same word on the different places in the matrix. However, your target should be identifying the exact one we are looking for. To identify this, try to see the patterns based on the test cases from the codegrade and update your code to go the those particular directions first and match the test cases.

Sample output1 for input 1

```
Looking for ABC
[C, B, A,  ]
[ ,  ,  ,  ]
[ ,  ,  ,  ]

Looking for BAOED
[ , B, A,  ]
[ ,  , O,  ]
[ , D, E,  ]

Looking for CNE
[C,  ,  ,  ]
[ , N,  ,  ]
[ ,  , E,  ]

Looking for NEC
NEC not found!
```

```
Looking for NEED
[ ,  ,  ,  ]
[ , N,  , D]
[ ,  , E, E]

Looking for NEEDON
NEEDON not found!
```

Sample input2:

```
4 4 4
c o m p
a m a p
a p s s
x y z q
mop
comp
omo
maa
```

Sample output2

```
Looking for mop
mop not found!
Looking for comp
[c, o, m, p]
[ ,  ,  ,  ]
[ ,  ,  ,  ]
[ ,  ,  ,  ]

Looking for omo
omo not found!
Looking for maa
[ ,  ,  ,  ]
[a, m,  ,  ]
[a,  ,  ,  ]
[ ,  ,  ,  ]
```

Specific Restrictions:

Your code must incorporate the following restrictions to receive full credit:

1. **You must use backtracking strategy to solve this problem**

Submission:

Submit the following files in Codegrade. Make sure your code work in Codegrade platform.

1. **Main.java file with the code**
2. **Lastname_analysis.txt file: This file will very briefly explain the time complexity of your code**