COP 3503C Programming Assignment # 2

BackTracking

Read all the pages before starting to write your code

What should you submit?

Write all the code in a single **Main.java** file and **Lastname_analysis.txt**. The analysis file will briefly discuss the time complexity of your code.

Please include the following commented lines in the beginning of your code to declare your authorship of the code:

```
/* COP 3503C Assignment 2
This program is written by: Your Full Name */
```

Compliance with Rules: UCF Golden rules apply towards this assignment and submission. Assignment rules mentioned in syllabus, are also applied in this submission. The TA and Instructor can call any students for explaining any part of the code in order to better assess your authorship and for further clarification if needed.

Caution!!!

Sharing this assignment description (fully or partly) as well as your code (fully or partly) to anyone/anywhere is a violation of the policy. I may report to office of student conduct and an investigation can easily trace the student who shared/posted it. Also, getting a part of code from anywhere will be considered as cheating.

Deadline:

See the deadline in webcourses. After that the assignment submission will be locked. An assignment submitted by email will not be graded and such emails will not be replied according to the course policy.

What to do if you need clarification on the problem?

I will create a discussion thread in webcourses and I highly encourage you to ask your question in the discussion board. Maybe many students might have same question like you. Also, other students can reply and you might get your answer faster. Also, you can write an email to the TAs and put the course teacher in the cc for clarification on the requirements.

How to get help if you are stuck?

According to the course policy, all the helps should be taken during office hours. Occasionally, we might reply in email.

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:

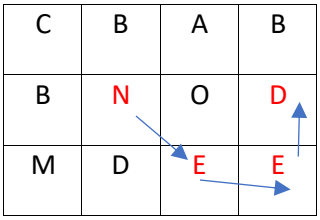
C	B	A	B
B	N	O	D
M	D	E	E

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**

Rubric (subject to change):

According to the Syllabus, the code will be compiled and tested in Codegrade Platform for grading. If your code does not compile in Codegrade, we conclude that your code is not compiling and it will be graded accordingly. We will apply a set of test cases to check whether your code can produce the expected output or not. Failing each test case will reduce some grade based on the rubric given bellow. If you hardcode the output, you will get -200% for the assignment.

1. If a code does not compile, the code may get 0. However, some partial credit maybe awarded. A code having compiler error cannot get more than 35% even most of the codes are correct
2. Coding and applying backtracking strategies: 35%
3. Passing test cases (exact output format): 55%
4. Reading data from file properly: 5%
5. Analysis file with correct content: 5%

Penalty:

6. Not using backtracking: -60%
7. Not putting header comment: -10%
8. Poor indentation in code and spacing in code: -10%
9. Not putting proper comments on important block of code: -5% (don't start commenting every line. Only important block of code)
10. Using file i/o would result in penalty : -100%

Some hints:

- **Have a wrapper function like we have seen for the maze problem.**
- **From the wrapper function, call the back tracking worker function for each box in the matrix until you find a solution or you decide there is no solution. Unlike the maze problem, we don't know the starting point of our search. So, we try each box as a starting point.**

Good Luck!