CS/COE 0445 Spring 2015 Assignment 3

Online: Sunday, February 15, 2015

Due: All source (.java) and data files, plus a completed Assignment Information Sheet zipped into a single .zip file and submitted to the proper directory in the submission site by **11:59PM on Monday, March 2, 2015. (Note:** see the submission information page for submission details)

Late Due Date: 11:59PM on Wednesday, March 4, 2015

Purpose: We have discussed recursion and in particular backtracking algorithms (such as Eight Queens). In this assignment you will get some practice at recursive programming by writing a backtracking algorithm to find phrases in a puzzle.

Idea: "Word search" puzzles are common in newspapers, on restaurant placemats and even in their own collections within books. These vary in format from puzzle to puzzle, but one common format is as follows: A user is presented with a 2-dimensional grid of letters and a list of words. The user must find all of the words from the list within the grid, indicating where they are by drawing ovals around them. Words may be formed in any direction – up, down, left or right (we will not allow diagonals even though most puzzles do allow them) but all of the letters in a word must be in a straight line.

This idea can be extended to allow phrases to be embedded in the grids. However, requiring an entire phrase to be in a straight line on the board is not practical, as the dimensions of the board would need to be overly large. Therefore, we will still require the letters in each word to be in a straight line, but we are allowed to change direction between words.

For example, we might be given the following 10x10 grid of letters:

```
a b s t r a c t i j
a t a d t d a t a j
t b c d c a g h t j
y b c d a t g h y j
p b c d r a g h p j
e b c d t f l h e j
s a r e s f o h s j
a r e d b f o h a j
r b c d a f c h r j
e r e a l l y r e j
```

and the following phrases:

```
abstract
abstract data types
abstract data types are really cool
abstract data types are really awesome
```

Upon searching, assuming the grid starts in the upper left corner with position (0,0), and that the row is the first coordinate:

`abstract` would be found in two places:
[(0,0) to (0,7)] and
[(8,4) to (1,4)]
`abstract data types` would also be found in two places:
[(8,4) to (1,4)], [(1,5) to (1,8)], [(2,8) to (6,8)] and
[(8,4) to (1,4)], [(1,3) to (1,0)], [(2,0) to (6,0)]
`abstract data types are really cool` would be found at:
[(8,4) to (1,4)], [(1,3) to (1,0)], [(2,0) to (6,0)], [(7,0) to (9,0)], [(9,1) to (9,6)], [(8,6) to (5,6)]
`abstract data types are really awesome` would not be found

Details: Your task is to write a Java program to read in a grid of letters from a file, and then interactively allow a user to enter phrases until he or she wants to quit. For each phrase your

Input Details:
The grid of letters for your program will be stored in a text file formatted as follows:
Line 1: Two integers, R and C, separated by a single blank space. These will represent the number of rows, R and columns, C, in the grid.
Remaining lines: R lines containing C lower case characters each

The user input will be phrases of words, with a single space between each word and no punctuation. Each phrase will be entered on a single line. The user may enter either upper or lower case letters, but the string should be converted to lower case before searching the grid. The program will end when the user enters no data for the current phrase (i.e. hits <Enter> without typing any characters beforehand).

Output Details:
If a phrase is not found in the grid the output should simply state that fact.
If a phrase is found in the grid, your program must find **one occurrence of the phrase**, and the output must indicate this fact in two ways:
1) Show the coordinates of each word in the phrase as a pair of (row, column) pairs.
2) Show the grid with the letters of the phrase indicated in upper case
For example, for the grid above and the phrase "abstract data types" your output would be:

```
abstract: (8,4) to (1,4)
data: (1,5) to (1,8)
types: (2,8) to (6,8)
a b s t r a c t i j
a t a d T D A T A j
t b c d C a g h T j
y b c d A t g h Y j
p b c d R a g h P j
e b c d T f l h E j
s a r e S f o h S j
a r e d B f o h a j
r b c d A f c h r j
```

```
e r e a l l y r e j
```

Algorithm Details:
Your search algorithm must be a recursive, backtracking algorithm. Note that you do not need recursion to match the letters within individual words (although you may do this recursively if you prefer). Where the recursion is necessary is when moving from one word to the next – since it is here where you may change direction. To make the program more consistent (and easier to grade), we will have the following requirements for the recursive process:

1) No letter may appear more than one time in any part of a solution.

2) When given a choice of directions, the options must be tried in the following order: right, down, left, up. Given this ordering and the grid above, the solution for "abstract data" would be [(8,4) to (1,4)], [(1,5) to (1,8)] rather than [(8,4) to (1,4)], [(1,3) to (1,0)], since the "right" direction is tried before the "left" direction.

3) If the last letter in a word is at location (i, j), the first letter of the next word must be at one of locations (i, j+1), (i+1, j), (i, j-1), or (i-1, j).

4) The direction chosen to find the first letter of a word is the same direction that must be used for all of the letters of the word. For example, in the grid shown above, for the phrase "abstract data", the following would NOT be a valid solution: [(8,4) to (1,4)], [(1,5) to (4,5)]. This solution is not legal because we proceeded right from the "T" of "abstract" to find the "D" in "data", but then proceeded down to find the remaining letters in "data". Similarly, also in the grid shown above, for the phrase "abstract data types are", the following would NOT be a valid solution: [(8,4) to (1,4)], [(1,3) to (1,0)], [(2,0) to (6,0)], [(7,0) to (7,2)]. This solution is not legal because we proceeded down from the "S" of "types" to find the "A" in "are", but then proceeded right to find the remaining letters in "are".

Test Files:
The file "sample.txt" is already online. Test it with the phrases above – your results should match the results shown in file "sample-out.txt". I may also be putting a couple more test files online in the next few days. Look for these and some test phrases to go with them.

Help:
If you are having trouble with recursion and backtracking, the program FindWord.java may be of help to you. This program finds individual words in a grid of characters using recursion and backtracking. It recurses on individual characters rather than on words, but the recursive process is similar in both programs. See also test file findWord1.txt.

Submission:
Make sure you submit all of the following in **a single .zip file**:
1) All source files that you either wrote or utilized (ex: from the author's files). Call your main program file **Assig3.java**.
2) All data files
3) Your completed Assignment Information Sheet

As with Assignments 1 and 2, the idea from your submission is that your TA can compile and run your programs WITHOUT ANY additional files, so be sure to test them thoroughly before submitting them. **If you use an IDE for development, make sure your program runs from the command line without the IDE before submitting it.** If you cannot get the program working as specified, clearly indicate any changes you made and clearly indicate why, so that the TA can best give you partial credit.

Extra Credit Idea:
Combine the ideas in FindWord.java and Assignment 3 to allow both the characters in the words and the words themselves to change directions – recursing on both the letters and the words. If you do this, make a second main program (Assign3B.java) and clearly explain how to run it in your Assignment Information Sheet. Also make up and include one or more data files that test the added functionality of this program.