Homework 3 sample solution

Due 09/08/15

September 1, 2015

1. Consider the *word search* problem. The input to this problem consists of three integers m, n, and w, an $m \times n$ matrix of letters, and a list of w words. All of the words in the list are hidden inside the matrix of letters (forwards, backwards, up, or down, but not diagonally), and the solution to a word search consists of $w \times 2$ array containing the row and column of the first letter of each word in the list, where the coordinates appear in the same order as the words. (If one of the words in the list does not appear in the matrix of letters, this row of the output may contain any values. If it appears more than once, you may return any of its locations.)

   Design an algorithm to solve the word search problem. Take care not to read past the top, bottom, left, or right of the letter matrix. *Hint*: you may wish to use subarray notation, like letters[a..b, c], to denote words formed by scanning the matrix letters vertically (or horizontally).

   Answer: Multiple solutions are possible here, though I expect most will use some form of exhaustive search. A sample solution appears below. Note that this sample solution could be made more efficient (e.g., by skipping words whose first letter doesn't match the current position in the matrix), but we are only asked to find a correct solution.

2. Prove that your algorithm for WordSearch is correct.

   Answer:

   Proof. Let words[x] be an arbitrary word in the list words. By the description of the input, words[x] must appear somewhere in the matrix letters, forwards, backwards, up, or down. Let (y, z) be the location of the first letter of words[x] in letters. We prove that WordSearch sets row x of found to be (y, z) by cases:

   (*Case 1*: words[x] goes up) In this case, the last letter of words[x] must be at letters[y-(ℓ-1), z], where ℓ is the length of words[x]. Since this position

Input: m: number of rows in word search
Input: n: number of columns in word search
Input: w: number of words to search for
Input: letters: $m \times n$ matrix of letters to search
Input: words: list of words to search for
Output: a $w \times 2$ array containing the coordinates of the first letter of each word in words

```
Algorithm: WordSearch
Initialize found as a w × 2 array containing 0
for i = 1 to m do
    for j = 1 to n do
        for k = 1 to w do
            Let len be the length of words[k]
            len = len - 1
            if i > len then
                if letters[i..i-len, j] = words[k] then
                    found[w, 1..2] = (i, j)
                end
            end
            if j + len ≤ n then
                if letters[i, j..j+len] = words[k] then
                    found[w, 1..2] = (i, j)
                end
            end
            if i + len ≤ m then
                if letters[i..i+len, j] = words[k] then
                    found[w, 1..2] = (i, j)
                end
            end
            if j > len then
                if letters[i, j..j-len] = words[k] then
                    found[w, 1..2] = (i, j)
                end
            end
        end
    end
end
return found
```

must be inside the bounds of the array,

$$
\begin{aligned}
y - (\ell - 1) &\geq 1 \\
y - \ell + 1 &\geq 1 \\
y - \ell &\geq 0 \\
y &\geq \ell \\
y &> \ell - 1
\end{aligned}
$$

Thus, on iteration y of the outer `for` loop, z of the middle `for` loop, and x of the inner `for` loop, i = y and len = ℓ - 1, so the algorithm will find that i > len in line 8. Afterwards, it will find that letters[i..i-len,j] = words[k], and it will set row x of found to be (y, z).

(*Case 2*: words[x] goes down) In this case, the last letter of words[x] must be at letters[y+(ℓ-1), z], where ℓ is the length of words[x]. Since this position must be inside the bounds of the array, y+(ℓ-1) $\leq$ m.

Thus, on iteration y of the outer `for` loop, z of the middle `for` loop, and x of the inner `for` loop, i = y and len = ℓ - 1, so the algorithm will find that i + len $\leq$ m in line 18. Afterwards, it will find that letters[i..i+len,j] = words[k], and it will set row x of found to be (y, z).

(*Case 3*: words[x] goes left) In this case, the last letter of words[x] must be at letters[y, z-(ℓ-1)], where ℓ is the length of words[x]. Since this position must be inside the bounds of the array,

$$
\begin{aligned}
z - (\ell - 1) &\geq 1 \\
z - \ell + 1 &\geq 1 \\
z - \ell &\geq 0 \\
z &\geq \ell \\
z &> \ell - 1
\end{aligned}
$$

Thus, on iteration y of the outer `for` loop, z of the middle `for` loop, and x of the inner `for` loop, j = z and len = ℓ - 1, so the algorithm will find that j > len in line 23. Afterwards, it will find that letters[i,j..j-len] = words[k], and it will set row x of found to be (y, z).

(*Case 4*: words[x] goes right) In this case, the last letter of words[x] must be at letters[y,z+(ℓ-1)], where ℓ is the length of words[x]. Since this position must be inside the bounds of the array, z+(ℓ-1) $\leq$ n.

Thus, on iteration y of the outer `for` loop, z of the middle `for` loop, and x of the inner `for` loop, i = y and len = ℓ - 1, so the algorithm will find that j + len $\leq$ n in line 13. Afterwards, it will find that letters[i,j..j+len] = word[k], and it will set row x of found to be (y, z).

Thus, in all four cases, row x of found is set as (y, z). As a result, each row of found is set to be the location of the first letter of the corresponding word in words.

Moreover, if row x of found is set to (y, z), it must be because letters[y..y-ℓ,z], letters[y,z..z+ℓ], letters[y,z..z+ℓ], or letters[y..y-ℓ,z] equal words[x], as lines 10, 15, 20, and 25 are the only lines that modify found, and the corresponding if statements must be true before these statements execute. Thus, every row of found will be (y, z) if and only if the first letter of the corresponding word is located at (y, z), so our output will be correct. $\square$

(Note that words that do not appear in the letters matrix will have a row equal to (0, 0) in found, and those that appear multiple times will have the last location of their first letter (in left-to-right, top-to-bottom order) in the row of found.)