

CSCI-1200 Computer Science II — Fall 2008

Homework 6 — Inverse Word Search Recursion

In this homework we will build an inverse word search program using the techniques of recursion. The goal is to construct a grid of letters that one can search to find specific words. Understanding the non-linear word search program from Lecture 12 will be helpful in thinking about how you will solve this problem. We strongly urge you to study and play with that program, including tracing through its behavior using a debugger or cout statements or both. *Please read the entire handout before beginning your implementation.*

Your Tasks

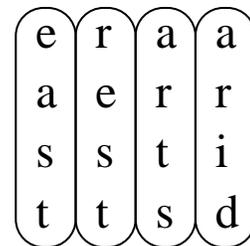
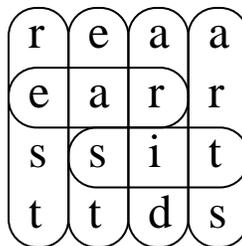
For this assignment, you will be given the dimensions (width and height) of a word search puzzle, a set of words that should appear in the grid (forwards, backwards, up, down, or along any diagonal), and optionally a set of words that should not appear anywhere in the grid. Each grid cell will be assigned one of the 26 lowercase letters. Note that unlike the non-linear word search problem we discussed in class, we will only allow words that appear in a straight line (including diagonals). Your task is to output all unique grids that satisfy the requirements. Rotations and mirroring of the board will be considered unique solutions.

Your program should expect two command line arguments, the name of the input file and the name of the output file:

```
inverse_word_search.exe puzzle2.txt out2.txt
```

Here's an example of the input file format:

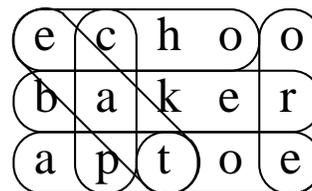
```
4 4
+ arts
+ arid
+ east
+ rest
- ear
- at
- sit
```



The first line specifies the width and height of the grid. Then each line that follows contains a character and a word. If the character is '+', then the word must *appear* in the grid. If the character is '-', then the word must *not appear* in the grid. For this first example we show an incorrect solution on the left. Though it contains the 4 required words, it also contains two of the forbidden words. The solution on the right is a fully correct solution. This particular problem has 8 solutions including rotations and reflections.

Below is a second example that specifies only positive (required) words. This puzzle has 4 solutions including rotations and reflections.

```
5 3
+ echo
+ baker
+ apt
+ toe
+ ore
+ eat
+ cap
```



Your program will output the number of solutions and an ASCII representation for each solution. See the example output on the course webpage. You should follow this output closely, however your solutions may be listed in a different order. If the puzzle is impossible your program should output “No solutions found”.

To implement this assignment, you must use recursion in your search. First you should tackle the problem of finding and outputting one legal solution to the puzzle (if one exists). Significant partial credit will be given for submissions that do this correctly. Full credit will be given to programs that find *all* of the solutions.

Algorithm Analysis

For larger, more complex examples, this is a really hard problem. Your program should be able to handle the small puzzles we have created in a reasonable amount of time. You should make up your own test cases as well to understand this complexity. Include these test cases with your submission. Summarize the results of your testing, which test cases completed successfully and the approximate “wall clock time” for completion of each test. The UNIX/cygwin `time` command can be prepended to your command line to estimate the running time:

```
time inverse_word_search.exe puzzle1.txt out1.txt
```

Once you have finished your implementation and testing, analyze the performance of your algorithm using order notation. What important variables control the complexity of a particular problem? The width & height of the grid, the number of required words, the number of forbidden words, the number of letters in each word, the number of solutions, etc.? In your *plain text* `README.txt` file, write a concise paragraph (< 200 words) justifying your answer. If you have any notes you want the grader to read, include them in this file.

Additional Information

You must use recursion for this homework submission. You may use any of the data structures and techniques we have discussed so far in this course. Do all of your work in a new folder named `hw6` inside of your CSII homeworks directory. Use good coding style when you design and implement your program. Be sure to make up new test cases to fully test your program and don't forget to comment your code! Use the template `README.txt` to list your collaborators and any notes you want the grader to read. When you are finished please zip up your `hw6` folder exactly as instructed for the previous assignments and submit it through the course webpage.

Extra Credit Contest!

For extra credit you may implement techniques to improve the performance (running time) of the basic algorithm. Describe these improvements in your `README.txt` file. Make up at least one new challenging test case that your program solves correctly and include it with your submission.

We will hold a contest to find the best inverse word search program. You must use recursion for the homework assignment, but you are not required to do so for the contest. Entries must include a plaintext `README_contest.txt` file explaining their optimizations. Extra credit will be awarded for all entries that compile and run the basic test cases. Entries will be judged based on their performance (accuracy *and* speed) on a variety of test cases. The winners will be announced during lecture (date TBA) and the winners must be present to claim their prizes.

To enter the contest, prepare your entry in a folder named `hw6_contest` and submit it on the homework submission site. The usual homework late day policy applies to the non extra credit homework submission. All contest entries are due on Saturday October 25th by 11:59pm (this will not count against your late day total).