Programming Question: Honeycomb Word Search

Do Not Distribute - Property of Quantcast Corporation

Background

A tessellation of hexagonal cells forms a honeycomb pattern where each cell connects with up to 6 neighboring cells. Suppose that we have a radially symmetric honeycomb of cells containing letters. Figure 1 (below) depicts one such honeycomb consisting of 61 cells.

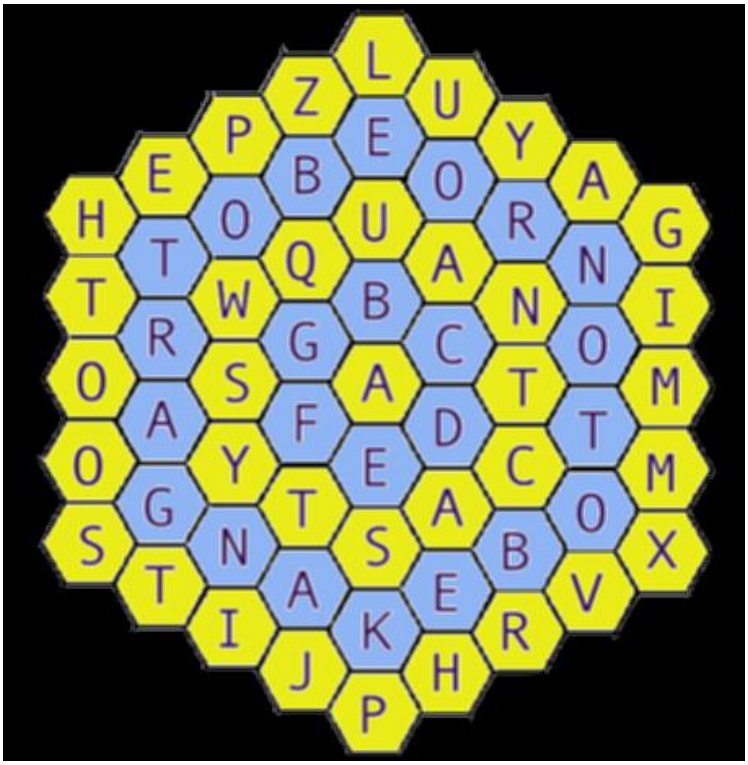

Figure 1 - Example 61-Cell Honeycomb

If you look at the example honeycomb provided in Figure 1, you will notice that the center cell contains the letter A. We define the centermost cell to be Layer 0. As we add a layer of cells (B, C, D, E, F, and G) to Layer 0, we form Layer 1, which increases the total cell count by 6. And, when we add yet another layer of cells (U, A, N, T, C, A, S, T, Y, S, W, Q), we form Layer 2, which increases the total cell count by 12. Each new layer added (Layer N: N > 0) will increase the total cell count by 6N.

Note that we describe each new layer starting with the topmost cell in that layer and fill in the rest of that layer in a clockwise fashion. In our example board, we have a total of 5 layers (in alternating colors), giving us a total of 61 tiles.

Problem Description

Your task is to write a program that efficiently finds a set of words in the honeycomb structure. Both the structure and set of words to attempt to find will be provided as explained below.

Your submission will be graded based on the correctness of the code's results, the speed of your code's execution and the readability of your code and its design. Your code will be tested against both small and large data sets.

Your program should take two command line input arguments. The first is the path of a text file containing the number of layers in the honeycomb followed by carriage-return delimited sets of uppercase letters that make up each layer of cells (Honeycomb input file). The second is the path of a text file containing carriage-return delimited valid strings with 2 or more uppercase letters (Dictionary input file). Example inputs for both of these files can be found in the next section.

Your program should find all valid strings (from the dictionary input file) that can be found by starting at an arbitrary cell and building a word by moving to adjacent connected cells one at a time and print them all to `stdout` in alphabetical order. Cells cannot be used twice in the formation of a single string, so on our example board, SAD, STING, and MOTTO can be found, but SADDEST and STINGS are not found (see Figure 2 below for these example words). If a valid string can be found via multiple paths, please print that string just once.

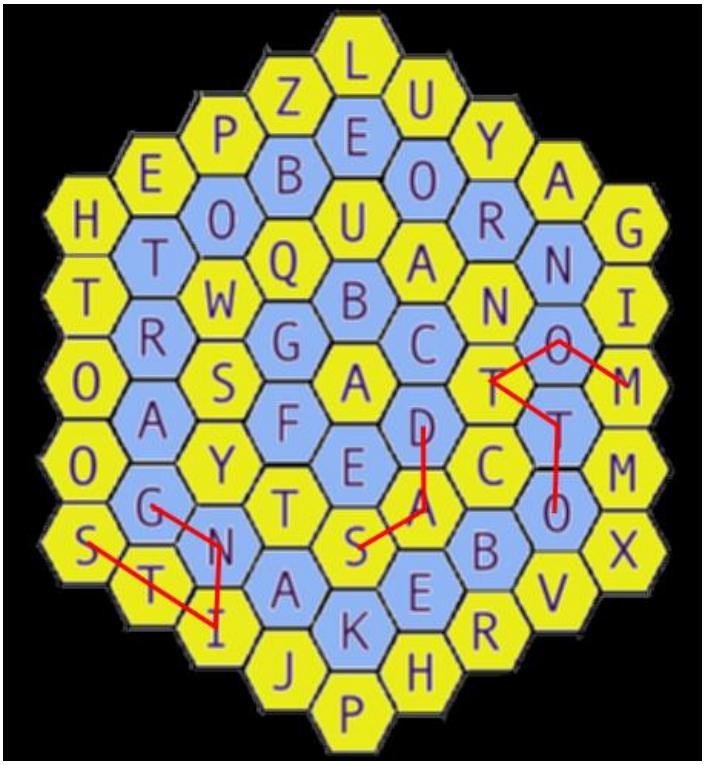

Figure 2 - Example Words Found

quantcast

Example Inputs, and Expected Output

Here are the contents of a Honeycomb input file describing our example board. The first line is simply a number denoting the total number of layers. The next line should contain exactly one letter, which forms Layer 0. Note that Layer 1 should have exactly 6 letters, and each subsequent layer should have 6 more letters than the previous one.

```
5
A
BCDEFG
UANTCASTYSWQ
EORNOTOBEKANGARTOB
LUYAGIMMXVRHPJITSOOTHEPZ
```

Once again, please note that each layer always starts at its topmost cell. Make sure that you understand what this honeycomb input file represents before proceeding.

Here are the valid strings contained in a sample Dictionary input file:

```
SAD
STING
MOTTO
SADDEST
STINGS
QUANTCAST
FADE
CASTE
CAST
CASTED
CASTLE
BAGS
BAG
GAG
DEFEAT
FADED
DEFACTO
EAST
```

Provided the two input files above, the expected output of your program should be the list consisting of all the valid strings that can be found in the example board as shown here (note that the output is alphabetically sorted):

```
BAG
BAGS
CAST
CASTE
CASTED
DEFACTO
EAST
FADE
MOTTO
QUANTCAST
SAD
STING
```


quantcast

Additional Comments

- C and C++ programs should compile with the GNU C/C++ compiler using C++03 language syntax and constructs.
  - C++ solutions must use only those standard libraries included with gcc. For example, they may use STL, but may not use Boost.
  - We will use the following command to build your code:
    ```
    g++ -Wall -Werror *.cpp
    ```
    If you require build options more sophisticated than that, you must include a Makefile.
  - We will run your code like this:
    ```
    ./a.out honeycomb.txt dictionary.txt
    ```

- Java programs should compile using the standard Oracle JDK, version 1.7 or newer.
  - Java solutions should only use the standard Java libraries.
  - We will use the following command to build your code:
    ```
    javac *.java
    ```
    If you require build options more sophisticated than that, you must include an Ant build.xml file.
  - We will run your code like this:
    ```
    java YourMainClass honeycomb.txt dictionary.txt
    ```

- Python programs should be compatible with Python v2.7.2 or later.
  - Python solutions should not use any non-standard Python libraries
  - We will run your code like this:
    ```
    python YourFile.py honeycomb.txt dictionary.txt
    ```

- For programs written in any other languages, please include basic instructions on how you would like us to compile and execute your code.

- Your program will not be fed improperly formatted or otherwise invalid inputs. If you do decide to add additional error checks, please report any errors in a sensible manner to `stderr`, and exit with a non-zero exit code.

- All letters in the input files are uppercase letters from A through Z. And, the only numeric value will be found in the first line of the Honeycomb input file.