tubular

Programming Challenge - Word Search Optimization

The goal of this challenge is to write a command line tool that calculates the best possible play sequence for a timed word search game using an arbitrary rectangular playing board. Consider a N rows by M columns grid of letters, where N = 3 and M = 3. Words from a list of possible words may be found in the grid of letter tiles as a sequence of adjacent (vertically, horizontally, or diagonally) tiles with any given tile being used only once when forming each word, but each letter tile can be reused across different words.

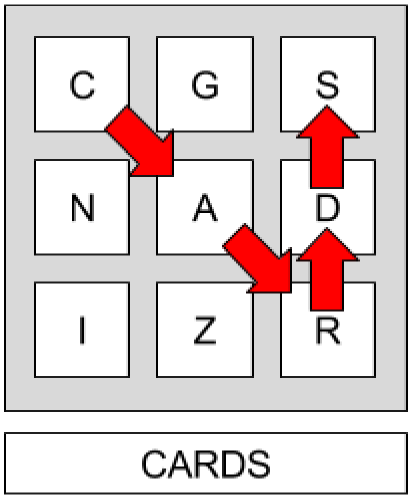
C	G	S
N	A	D
I	Z	R

Source Word List

```
air
car
card
cards
drag
drags
sad
sadden
sin
snail
zig
```

Each found word is then scored by adding up individual letter points per a given letter scoring scheme. For example, given the source word list and a scoring scheme where consonants are valued 1 and vowels are valued 2, the point values for the found words would be:

Found Word	Point Value
car	4
card	5
cards	6
drag	5
drags	6
sad	4



There is a time limit to find and identify words on the playing board. The amount of time it takes to find and identify each word can modeled by two parameters: the amount of time to find the general location of the word and its first letter on the playing board, and the amount of time it takes to identify each of the word's subsequent letter in sequence (e.g., by sliding your finger over each letter in order). Given the game's time limit, you may be only able identify a subset of the possible words on the board within the time limit, and so you want to figure out which set of found words would yield the maximum total score. Finding and identifying the word must completely fit within the game time period for it to be counted towards the total score.

For example, if in the above example it took you 15 seconds to find each word and its first letter, then it took 2 seconds to identify each subsequent letter in the word, the word "cards" would take a total of 23 seconds to identify. Given a lime limit of 55 seconds, the set of words that would yield the maximum score would be "cards" and "drags", as finding and identifying these two words would yield a total score of 12 and take a total of 46 seconds, leaving insufficient time to find and identify any other of the valid solution words.

Your assignment is to write a command line tool that solves this search and optimization problem. The command line tool should accept on stdin (standard in) a JSON object that fully describes the various input parameters for the problem. The JSON object will have the following members:

Member Key	**Member Value Type**	**Member Description**
`rowCount`	Number	The number of rows in the game board. Value is greater than or equal to one and won't be greater than 1000.
`columnCount`	Number	The number of columns in each row in the game board. Value is greater than or equal to one and won't be greater than 1000.
`gameBoard`	Array of character arrays	The playing board, which is a two-dimensional array of characters laid out in row-major order. Each cell in the two dimensional array contains one letter.
`wordList`	Array of strings	The list of words which may (or may not) be found in the game board using the search mechanics described above. Only words found in this list may be found on the game board. There is no significance to the ordering of words in this list, and this list will contain at least one word, though no listed words may be findable in the playing

		board.
`letterPoints`	Dictionary of character, number pairs	A dictionary indicating the point value for each letter. The key is a string for the letter, while the value is an integer greater than zero representing the points for the letter. If a given letter is not listed in this dictionary, it is assumed to have a point value of 1.
`wordFindTime`	Number	The number of seconds it takes to initially find each word on the game board. Value is an integer greater than or equal to zero.
`letterIdentifyTime`	Number	The number of seconds it takes to identify each letter in a found word. Value is an integer greater than or equal to zero.
`maxGameTime`	Number	The maximum number of seconds the game may be played for. Value is an integer greater than or equal to zero.

For the example problem described above, the JSON that would be used to describe the example game above would be:

```
{
    "rowCount": 3,
    "columnCount": 3,
    "gameBoard": [
            [ "C", "G", "S" ],
            [ "N", "A", "D" ],
            [ "I", "Z", "R" ]
        ],
    "wordList": [
            "air", "car", "card", "cards",
            "drag", "drags", "sad", "sadden",
            "Sin", "snail", "zig"
        ],
    "letterPoints":{
            "a":2,
            "e":2,
            "i":2,
            "o":2,
            "u":2,
            "y":2
        },
    "wordFindTime": 15,
    "letterIdentifyTime": 2,
    "maxGameTime": 55
}
```

The output of the command line tool should be printed to stdout (standard out) and simply be the alphabetized list of ***all*** words from the input that can be found on the playing board, with one word per line, and one final line contained the maximum possible score that can be attained within the given time limit. For the example problem given above, the expected output would be:

```
car
card
cards
drag
drags
sad
12
```

Comments

- C and C++ programs should be compiled using the GNU Compiler Collection (GCC) or LLVM compiler.
    - C and C++ solutions must only use the standard libraries available to the compiler. For example, in C++, the code may use STL, but should not use Boost. However, you may use this library for JSON processing:
        - https://github.com/nlohmann/json
        - Please include the source code of the JSON library you link to in your solution.
    - We will compile your code using the following unix command: "`g++ -Wall *.cpp`". If your code requires more sophisticated compiling and linking, then you should create and include a makefile for GNU Make. If you require C++11 compilation, simply indicate in your submission whether the `-std=c++11` compiler option should be enabled.
    - We will then run your code using the following command unix command: "`cat input.json | a.out | diff answer.txt - `".
- Java programs should compile using the standard Oracle SDK, Java version 1.7 or 1.8.Java solutions should only use the standard Java libraries. However, you may use this JSR 353 compliant library for JSON parsing:
        - https://jsonp.java.net/
        - Please include the jar of the JSON library you link to in your solution.
    - You should define your main method in a class file named `WordSearchGame.java`.
    - We will compile your code using the following unix command: "`javac -cp json_lib.jar *.java `", where `json_lib.jar` will be the JSON library jar you provide with your solution, if any. If you require more build options, you should include an Ant `build.xml` file.
    - We will then run your code using the following unix command: "`cat input.json | java -cp json_lib.jar:. WordSearchGame | diff`

    `answer.txt -`", where `json_lib.jar` will be the JSON library jar you provide with your solution, if any.

- Python solutions should indicate whether they depend on Python 2.7 or 3.x.
    - Python solutions should only require the standard python libraries. No external libraries should be required. That is, only libraries documented at <https://docs.python.org/library/> should be used.
    - Your main python file should be named `WordSearchGame.py`
    - We will run your code with the following unix command: "`cat input.json | python WordSearchGame.py | diff answer.txt -` ". We will use the correct version of python (`python2` or `python3`) based on your description.
- In all example unix commands, the `input.json` and `answer.txt` files are what we will use to test the correctness of your code. The input to your command line tool should only be `stdin`, and the output should be printed only to `stdout`.

- The following points should be accounted for in your program:
    - We will test your program with game boards of various sizes (large and small) and word lists of several hundred thousand words. The speed and efficiency of your code will be an important consideration in its evaluation.
    - All text operations should be case insensitive regardless if capitalization is used in any game board or word list input. The output text should be all lowercase.
    - Your code may assume that the input JSON is properly formatted and represents a valid game configuration.
    - If the word list input contains duplicate instances of a word, the subsequent instances after the first should be ignored.
    - If a word is found multiple times on a playing board, it can only be counted once when determining the solution and total score.
    - When searching for words on the playing board, the starting letter may be any letter on the playing board and each subsequent letter must be horizontally, vertically, or diagonally adjacent to the prior letter in the word. Individual game board letters may be used only once when forming a specific word, and the playing board edges should be respected. That is, if a playing board has three or more columns, the top left cell is **not** adjacent to the top right cell via some "wrap around" concept.