Forming Words from Random Alphabets

In this problem, you are given three files:

1. dictionary_2.txt: A text file containing one word on each line. This is a list of valid words from the English dictionary.
2. score.txt: This is a text file containing a comma separated list of alphabets and the score for each alphabet.
3. test_random_string_500.txt: Each line of this test file has a string of random alphabets. Using each string as input, the goal is to output a valid word that gives the highest score. The output file will contain one line for each line in the test file. Each line will contain: the random string, the legal word, score of the word. The three entries will be separated by commas.

4. test_random_string_500.txt.result: This is the result file we expect for the above test file.

For example:
Take the score given below:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	3	3	2	1	4	2	4	1	8	5	1	3	1	1	3	10	1	1	1	1	4	4	8	4	10

Let us say one of the random string is:
eotxberx

Output for the above string will be:
eotxberx, xerox, 19

Score for the word xerox is computed as follows: 8 + 1 + 1 + 1 + 8. Many other words can be formed from the random words given above: or, to, bee, eat, beer, bore, exert and many more. However xerox is the only word printed because it has the highest score.

Additional instructions:

1. All capital letters must be converted to small letters.

2. If a letter occurs in the random string “n” times, to create a valid word, the letter can be used between 0 to n times.
   a) For example, there are two e’s in the example string “eotxberx”, but the word with best score is “xerox” and it does not use the letter “t”.
   b) For example, there is only one “r” in the example string “eotxberx”, you cannot use “r” two times in creating a word.

3. If two or more words are formed with the same score, choose the word that is smaller as per string comparison.

4. There should be no alphabets other than a to z in the random strings. If you encounter any, please let us know.

5. If a specific string cannot form a legal word, then the code needs to assign it a score of zero. The output would be: the random string, random string, 0. For example, if the random string is:

qqttee

Output for above string will be:
qqttee, qqttee, 0

6. You have been given sample code in file WordForming.cpp in the following function:

```
WordForming::processFile(std::string dictonaryPath, std::string scorePath, std::string inputPath, std::string outputFilePath)
```

Process each word in the while loop in lines 34-39 of file WordForming.cpp. Feel free to define any additional classes and functions as needed. The code can be executed from command line using the following command. Note that you would have to adjust the input and output file paths as per your directories.

```
./homework dictionary_2.txt score.txt test_random_string_500.txt /tmp/out.txt
```

This will generate two result files:

   a) /tmp/out.txt : Result file containing valid dictionary words that have maximum score.
   b) /tmp/out.txt.memory_runtime: File containing memory and runtime statistics.

7. Correctness of your results will be evaluated by running your code on a larger set of random words generated from the same dictionary (dictionary_2.txt).

8. You will only submit the two source files WordForming.cpp and WordForming.h on gradescope and canvas.

9. You cannot use std::template packages or external APIs. You cannot change the #include statements given in any of the files.

Hints:

1. Use a trie data structure to load the dictionary.
2. To identify the word that can be formed from the random string, pass through the trie while consuming alphabets from the random string.
3. Compute scores of the word being formed. Retain word that has maximum score.

Grading method:

1. If your code runs and generates an output file in the correct format for each input file, you get submission points.
2. If your method generates correct results for each test case, you get points for correctness.
3. We will measure the memory consumption in Bytes (submitted_memory). We will take the memory used by our implementation (our_memory). Memory score will be based on the ratio: (our_memory / submitted_memory) * max score for memory.
4. We will measure the run time in milliseconds (submitted_time). We will take the time used by our implementation (our_time). Your run-time score will be based on the ratio: (our_time / submitted_time) * max score for run-time.
5. The maximum scores for sbumission, correctness, memory and runtime is based on a percentage of total points. The percentage breakup is below:

Grading Rubrick

Task No	Output: Does the code run? Are the output files generated with the filenames and formats as specified in the problem?		Correctness: Is the right output generated?		Timing			Memory			Total points obtained	Total for task
	Points obtained	Total points	Points obtained	Total points	Time taken to run (ms)	Comparison with other students (divide by min of others)	Timing score	Memory consumed (Bytes)	Comparison with other students (divide by min of others)	Memory score		
1	10.00%		50.00%		20.00%			20.00%				