University of Waterloo
CS240E, Winter 2023
Programming Assignment 2

Due Date: Wednesday, March 29, 2023 at 5pm

Be sure to read the assignment guidelines (`http://www.student.cs.uwaterloo.ca/~cs240e/w23/guidelines.pdf`).

Question 1 [20 marks]

These days, most smartphones use *word completion*, i.e., as you type the phone suggests commonly used words that fit the initial characters that you have typed. This assignment asks you to implement this.

Formally, implement a realization of a dictionary D that stores word-frequency pairs. Dictionary D supports the following operations:

- `access(string w)`: If $w \notin D$, then insert w with frequency 1 into D. If $w \in D$, then increase its frequency by 1.

  For example, if you executed `access("break"), access("crazy"), access("break")` onto an initially empty dictionary, and then print it, the output should be

  ```
  break, 2
  crazy, 1
  ```

- `getCompletions(string w)`: This gets the most frequent word completions as if you had typed string w one character at a time.

  Formally, assume $k = |w|$. Then for $i = 0, \ldots, k-1$, the output will be a word x that is in D and for which $w[0..i]$ is a prefix. Among all such extensions x, print the one with the largest frequency. If there are multiple extensions x that all have the largest frequency, return the lexicographically smallest among them. If there is no word that extends $w[0..i]$ then the output should be 'No extensions'.

  We illustrate the desired output-format with the following example. Assume your dictionary currently contains

  ```
  taut, 5
  teal, 3
  teamster, 4
  teatowel, 4
  total, 6
  ```

  then the output of `getCompletions(tear)` should be

  ```
  Best extension of t is total
  Best extension of te is teamster
  Best extension of tea is teamster
  No extension of tear
  ```

Implement a dictionary that supports the above two operations, as well as a print-function. Your C++ program must provide a main function that accepts the following commands from stdin:

- `p` - prints the current dictionary in the style illustrate above, i.e., prints the words (one word per line) followed by a comma and the frequency. Words must be listed in lexicographic order.
- `a word` - performs an `access` with the given word w. Nothing is printed to the output.
- `g word` - performs `getCompletions` with the given word w. In response, it writes the list of completions as illustrated above, using one line per character of w.
- `x` - terminates the program.

Evaluation. 40% of the marks will depend on having an *efficient* implementation for `access` and `wordCompletion`. We will determine efficiency by running your code on large examples, doing both `access` and `getCompletions` (with no promises about the ratio between these two types of queries) and checking whether your program times out.

Design ideas: There are very simple methods to obtain the correct answer (e.g. store all words in a dynamic array). This would be good enough to obtain the correctness marks, hence a passing grade.

To obtain the efficiency marks, you should use a dictionary for words, i.e., a trie. (The variant of trie is up to you.) Also, for any node v in the trie you must be able to find the maximum frequency among the descendants of v, so maintain this information with v (and break ties by lexicographic order). You should convince yourself that you can then do `getCompletion`(w) and `access`(w) in roughly $O(|w|)$ time, much faster than what you could do in a dynamic array. ('Roughly' hides a few terms that usually are not big, such as the time to find a child in the trie.)

Rules and Assumptions:

- All words are non-empty and use only characters in `a-z`. In particular, they do *not* contain \$.
- The words in D are not necessarily prefix-free. (It is your design-choice how to handle this in your trie.)
- There is no limit on the number of words, or the length of a word, other than that they are `int`.
- We will do `getCompletions` and print only when D contains at least one word.
- You are allowed to use `binary_search`, and `vector` from the STL. (You should not need them if you implement a trie.) You are also allowed to use `iostream, string` and `stringstream` from the STL.

- 'Printing on one line' means that the line must end with a newline. Trailing whitespace at the end of your output lines will be ignored by our test scripts.

Place your entire program in the file `wordCompletion.cpp`. Submit your solution to Marmoset. Marmoset will be set up to translate your program with `g++ -std=c++17`.