COP 3502C Programming Assignment#6

Topics covered: Trie

Please check Webcourses for the Due Date

Read all the pages before starting to write your code

Introduction: For this assignment, you will write a c program that utilizes the Trie data structure.

What should you submit?

Write all the code in a single .c file and upload the .c file.

Please include the following lines in the beginning of your code to declare your authorship:

```
/* COP 3502C Programming Assignment 6
This program is written by: Your Full Name */
```

Compliance with Rules: UCF Golden rules apply towards this assignment and submission. Assignment rules mentioned in syllabus, are also applied in this submission. The TA and Instructor can call any students for explaining any part of the code in order to better assess your authorship and for further clarification if needed.

Caution!!!

Sharing this assignment description (fully or partly) as well as your code (fully or partly) to anyone/anywhere is a violation of the policy. I may report such incidence to office of student conduct and an investigation can easily trace the student who shared/posted it. Also, getting a part of code from anywhere other than the class resources will be considered as cheating.

Deadline:

See the deadline in Webcourses. An assignment submitted by email will not be graded and such emails will not be replied according to the course policy.

Additional notes: The TAs and course instructor can call any students to explain their code for grading.

Assignment: Predictive Text Assistant

Introduction:

Typing efficiently is essential in today's fast-paced digital world. Many devices offer predictive text assistance, suggesting letters or words to help users type faster. However, these predictions are not always accurate, leading to frustration and errors.

Imagine you are developing a personalized text suggestion system that predicts the most likely next letter(s) based on your typing history. This system analyzes past usage to provide tailored suggestions, reducing errors and improving the overall typing experience. Unlike existing systems, which complete entire words, your system focuses on suggesting only the most probable next letter(s), ensuring better flexibility and precision.

This program will simulate a text prediction engine capable of learning from user input and responding to queries with suggestions. It offers the following functionalities:

1. Adding words and their usage frequencies to the system's dictionary.
2. Respond to queries by predicting the most likely next letter based on the provided prefix.

The program will appropriately inform the user if no words match a given prefix.

Example:

Suppose your dictionary contains the following data:

- "hello" used 50 times
- "help" used 30 times
- "heat" used 20 times
- "hero" used 10 times

If you query with the prefix, "he", the system will analyze the words starting with "he" and suggest the most likely next letter(s):

- 'l' appears in "hello" and "help" a total of 50+30=80 times.
- 'a' appears in "heat" a total of 20 times.
- 'r' appears in "hero" a total of 10 times.

Thus, the most likely next letter is l.

If you query with the prefix ho, since no word starts with ho, the program will respond appropriately, indicating that no matching prefix exists in the dictionary.

The Problem

You are tasked with designing a system that answers queries about the most likely next letter based on a list of words and their usage frequencies. Your program will process a series of operations, which include adding words to a dictionary and querying for prefixes.

For each query, your program must:

1. Return the most likely next letter(s) for the given prefix, based on the frequencies of words in the dictionary.
2. If no word in the dictionary has the given prefix as a proper prefix (i.e., the prefix is not strictly shorter than any matching word), respond with "`unrecognized prefix`".

The Input (to be read from standard console input (not file i/0. Any file i/o in the code will get zero))

The input begins with a single positive integer, ***n***, which specifies the number of commands your program needs to process. Each of the following ***n*** lines contains one command, in one of two formats:

- **Insert Command**
  A command that starts with the integer 1, followed by a word and a frequency:

```
1 s f
```

  - *s* is a string of lowercase letters representing the word to be added to the dictionary.
  - *f* is a positive integer representing the frequency with which the word was used. This command indicates that the word s has been used an additional f times. Note that the same word may appear multiple times in the input, as users can use the same word at different times. The total usage frequency is the cumulative sum of all occurrences of the word.
- **Query Command**
  A command that starts with the integer 2, followed by a prefix:

```
2 p
```

  - p is a string of lowercase letters representing the prefix for which the user wants a prediction.
    This query asks the program to find the most likely next letter(s) based on the current dictionary. A query does **not** modify the dictionary.

More input specification:

- The total number of letters in all input words will not exceed 2,000,000.
- The sum of the frequencies of all added words will not exceed 1,000,000,000.

The Output (to be printed to standard console output (no file i/o. Any file i/o in the code will get zero))

For each query command, your program must produce exactly one line of output:

1. If the prefix is a proper prefix for one or more words in the dictionary at the time of the query:
   - Print all the letters that are the most likely to come next, in alphabetical order, without space between them.
2. If the prefix is not a proper prefix for any word in the dictionary:
   - Print the string "`unrecognized prefix`" on a line by itself.

Sample Input

```
15
1 apple 40
1 application 25
2 app
1 aply 65
2 ap
1 banana 60
2 ba
1 band 20
2 abc
1 bandage 10
1 bat 50
2 bat
1 batter 30
1 apply 40
2 appl
```

Sample Output

```
l
lp
n
unrecognized prefix
unrecognized prefix
ey
```

Implementation Restrictions

You are required to implement a trie data structure to manage all input words and handle queries. Each trie node should include the following information:

1. **Node Frequency**: The frequency of the string represented by this node (i.e., how many times this specific string has been added to the dictionary).
2. **Sum Prefix Frequency**: The total frequency of all words in the dictionary that have this string as a prefix, including the string itself.
3. **Maximum Child Frequency**: The highest frequency among all child nodes of the current node. For example, if the node represents "app" and there are currently 25 occurrences of words starting with "appl" and 20 with "appy," this value should be 25.
4. **Children Pointers**: An array of 26 pointers, each representing one of the possible next letters ('a' to 'z'). A pointer should be NULL if no words in the dictionary continue along that path. Typically, only a subset of these pointers will be active.

Implementation Details

- **Type 1 Commands**: Use the trie's insert function to add words to the dictionary. Since words may appear multiple times in the input, ensure your insert function is designed to update existing nodes instead of creating duplicates unnecessarily.
- **Type 2 Commands**: Implement a custom function for handling prefix queries. While this function will be similar to the standard search function in a trie, it must include additional logic to determine and return the required output (e.g., the most likely next letter or a list of letters).

Additional Requirement:

You must read data from the standard console input and print the output in the standard console as well. No file i/o should be used in the code. The output format must match with the sample output format. Do not add additional space, extra characters, or words with the output as we will use diff command to check whether your result matches our result. Next, you must write a well-structured code. ***There will be a deduction of 10% points if the code is not well-indented and 5% for not commenting on important blocks. Also, poor choice of variable names will get a penalty.***

- As always, all the programming submissions will be graded based on the result from **Codegrade**. If your code does not work in codegrade we will conclude that your code has an error even if it works on your computer.
- Your code should not have any memory leak to receive full credit

Steps to check your output AUTOMATICALLY in shell or repl.it or online gdb or some other compilers that have command option:

Use the same strategy outlined in previous assignments.

Rubric (Subject to change):

1) A code not compiling or creating seg fault without producing any result can get zero. There may or may not be any partial credit. We will not modify your code or compile again to test it. So, make sure it works and pass test cases.
2) There is no grade for just writing the required functions. However, there will be 20% penalty for not writing insert function, 10% penalty for not writing the appropriate search function.
3) Implementing insert operation properly: 25%
4) Implementing prediction properly: 25%
5) Matching test cases: 50%. Your code will be tested against a set of test inputs
6) There is no point in well-structured, commented, and well-indented code. ***There will be a deduction of 10% points if the code is not well-indented and 5% for not commenting on important blocks, 10% missing header comment, and 2% for poor choice of variable names***

Hints:

In addition to the following hints, I will upload another pdf file with more hints and explanations

- Start programming assignments as soon as possible.
- Read the entire assignment and get a very good idea of what is expected.
- Use the sample input/output to understand it better
- Draw and simulate the sample input/output step by step by drawing the Trie step by step. Update the values of the structure accordingly while drawing.
- Based on that, work on the insert function. You can review the insert function we have discussed in class. You mainly need to update some extra variables.
- After that, start working on the prediction function. It would be somehow like a search with a lot of modification. It will start with a typical search. However, as soon as you are at the end of your search word, you need to make a decision. You might need to return NULL, or you might need to return a string with one or more characters. During this process, you need to check all the children of the current node and their frequency and need to match who has the highest frequency based on the current node's stored max frequency. If multiple children have max frequency, then you can produce a string with those characters and return that string! Don't forget to terminate your string by '\0'. Another strategy could be directly printing the characters.