Text Prediction

Computer Science I Program #4: Tries

Please check Webcourses for the Due Date

Read all the pages before starting to write your code

Most cellphones are equipped with word completion. If you type in the starting part of the word, suggestions are made to complete the word, based on the frequency with which words are used.

For example, if you've used "computer" 45 times, "complain" 15 times, and "complex" 99 times, if you type "comp", your phone may suggest that you complete the word as "complex", since is the word you've used the most which starts with "comp".

Unfortunately, you often accidentally hit the button to complete the word, and in the instances where this was the word you meant to type, you have lots of erasing to do, potentially.

Thus, you'd like to create your own text completion program which simply suggests the most likely next letter. In the example above, if the three words shown were the only ones that started with "comp", then the most likely next letter would be "l", which appeared 15 + 99 = 114 in the past, whereas "u" only appeared 45 times in the past. It is possible that multiple letters have an equal chance of being the next letter. In this case, you would like your program to list every one of these letters that are "most likely", with equal probability.

Your program will handle two types of commands:

(1) Adding a word to the dictionary of used words a particular number of times.

(2) A query with a prefix, for which your program must produce the most likely next letter, or list of most likely next letters. If the given query is not a prefix of any word previously added to the dictionary, then your program must detect this instead.

The Problem

Given a list of words being used and their frequencies, as well as queries of prefixes at various points in time, answer each of the queries with the most likely next letter (or list of most likely next letters), or answer "unknown word", if there is no word in the dictionary for which this is a proper prefix. (Note: A proper prefix is a prefix of a word that is strictly shorter than it.)

The Input (to be read from standard input) - Your Program Will Be Tested on Multiple Files

The first line of the input contains a single positive integer, *n*, representing the number of commands for your program to execute. Each of the commands follow, one per line, on the following *n* lines.

There are two possible commands.

The first command starts with the integer 1, representing an insert into the dictionary of used words. This is followed by a space and *s*, a string of lowercase letters, representing the word to be inserted. This is followed by a space and *f*, an integer, representing the number of times that word is to be inserted. The command means that the user has used the word *s* an additional *f* times. (Thus, it's possible that the same word may appear more than once in the set of input commands, representing that the user used that word some, and then used it at a later time some more.)

The second command starts with the integer 2, representing a query. This is followed by a space and *p*, a string of lowercase letters, representing the prefix for the query. Note that a query will NOT make any modifications to the dictionary keeping track of used words.

The total number of letters in all of the input words will not exceed two million. The sum of the frequencies of the added words will not exceed one billion.

The Output (to be printed to standard out)

Output will only printed for queries. For each query, a single line of output is printed. If the prefix queried is the proper prefix for any word in the dictionary at that point in time, then print out each letter, in alphabetical order, which is the most likely letter to come next. There should be NO SPACES between the letters. If the prefix queried is NOT a proper prefix for any word in the dictionary, print out "unknown word" on a line by itself.

Sample Input

```
15
1 cap 15
2 ca
2 cap
2 pen
1 cat 20
2 ca
2 c
1 act 10
1 able 10
2 a
1 ace 2
2 a
2 ab
2 ac
2 ace
```

Sample Output

```
p
unknown word
unknown word
t
a
bc
c
l
t
unknown word
```

Implementation Restrictions

You must create a trie to store all of the inputted words and to use to answer each of the queries. Your trie node struct should store the following items of information:

1) The frequency of the node itself (so how many copies the string upto this node has in the dictionary)

2) The sum of frequencies for which this string is a prefix of words in the dictionary, including itself.

3) The current maximum frequency of any child node (thus, if this node represents "ca", and at the current time there are 20 words starting with "cat" and 15 starting with "cap", then 20 should be stored here.

4) An array of size 26 representing pointers to the next possible letters. These pointers should be set to NULL if there are no words in the dictionary that follow these paths. (In most cases, some of the 26 will be set to NULL and some will not.)

Commands of type 1 should be executed using the trie's insert function. Note that since a word can appear more than once in the input, insert should be written in such a way that in some cases, no new nodes are created.

Commands of type 2 should be executed as a function that takes the trie in as a parameter, but that isn't exactly one of the standard trie functions. (This function will look similar to the search function in a regular trie, but with extra code added to execute the desired request.)

Deliverables

You must submit a two files over WebCourses:

1) A source file, *predict.c*. Please use stdin, stdout. There will be an automatic 10% deduction if you don't do so.

2) Your Process Document, which should be named LASTNAME_PROCESS.doc (or .docx or .pdf or .txt)