COP3502

Spring 2025

Programming Assignment 3

Smart Typing System

Modern text applications use auto-completion to suggest the most probable word based on existing words. In this assignment, you will implement a Trie-based word auto-completion system that stores words with their frequency of occurrence and suggests the most frequently used word for a given prefix**.**

Unlike traditional autocomplete, which may suggest multiple possible words, your system will return only the most frequently occurring word that starts with the given prefix. You can assume that no two words will have the same highest frequency i.e., each most frequent word is unique

The words and their frequencies will be inserted into the system once and will not change over time.

The Problem

Your program will support two types of operations:

1. Inserting a word into the Trie with a frequency.
2. Querying a prefix to determine the most frequent complete word. If a prefix does not exist in the dictionary, your program should return "unknown word". (This function will look like the search function in a regular trie, but with extra code added to execute the desired request.)

Implementation Restrictions

You must use a Trie to store the words and their frequencies. The Trie node structure should contain:

1. An array of 26 child pointers, each representing a letter ('a' to 'z').
2. A flag to indicate if the node marks the end of a word.
3. A field to store the frequency of a complete word at a node.
4. A field to track the highest frequency of any word in the subtree.
5. A field to store the most frequent word at a node for efficient retrieval.

Function Definitions

To ensure consistency in implementation, students must use the following function prototypes:

1. Creating a Trie Node: This function returns a pointer to a newly allocated Trie node. It Allocates memory for a new Trie node, initializes child pointers to `NULL` and sets the frequency to `0` and maxFrequency to `0`. The function prototype is:

   ```
   TrieNode* createNode();
   ```

2. Inserting a word into the Trie: This function traverses the Trie following the characters in Word. It creates nodes for missing letters. It Marks the final node as the end of a word and updates its frequency (i.e., if the word already exists, you should update its frequency). Also this function, updates the most frequent word and its frequency for each prefix along the path. The function prototype is:

   ```
   void insertWord(TrieNode* root, const char* word, int frequency);
   ```

3. Querying the Most Frequent Word for a Prefix: This function returns a `char*` (dynamically allocated string containing the most frequent complete word for the given prefix). The function traverses the Trie to locate `prefix`. It returns the **most frequently used complete word** starting with `prefix`. If no valid words match, returns `unknown word.`

Assumptions You Can Make:

- The maximum length of any word inserted into the dictionary will not exceed 100 characters.
- No two words will have the same frequency; each word will have a unique frequency.
- There will always be exactly one possible word to complete any given prefix.
- When inserting a word into the Trie (operation 1), no output is required.

Input and Output Format

The input will be read from standard input (stdin), and the output should be written to standard output (stdout)

Input: The first line contains an integer `n`, representing the number of commands. Each of the next `n` lines contains one of the following commands:

1. Adding a word with a frequency.

   `1 word frequency` For example `1 apple 100`

2. Querying a prefix for the most frequent complete word

   `2 prefix` For example `2 ap`

Output: For each query (search), output one line that has the most frequently occurring full word that start with `prefix`. If no valid prefix exists, print `"unknown word"`. Notice that you are not printing anything if the operation is to insert (i.e., the input starts with 1). You are only print the autocompletion words if the input line starts with 2.

Example Input/Output:

Input:

```
9
1 apple 50
1 ape 30
1 bat 40
1 ball 60
2 a
2 ap
2 b
2 ba
2 c
```

Output:

```
apple
apple
ball
ball
unknown word
```

What to Submit:

You must submit your source code over WebCourses by the due date

1. A source file, name it *predict.c*. Please use stdin, stdout to handle the input and the output.
2. Your code must be tested on Eustis before submission.