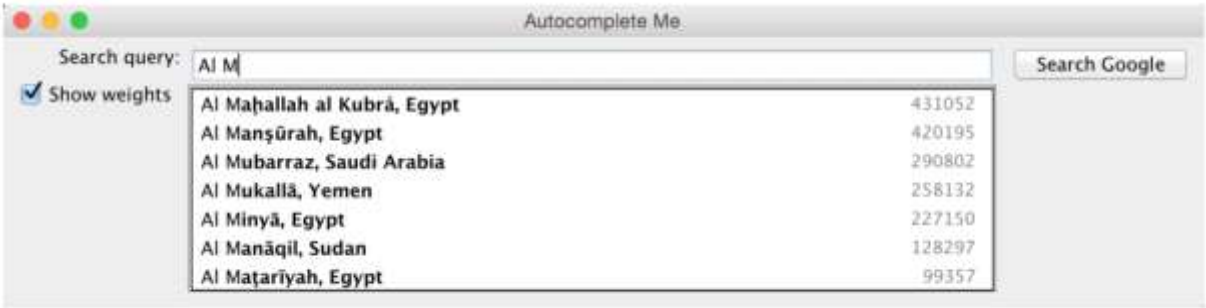
Text Autocomplete

Sample Input/Output

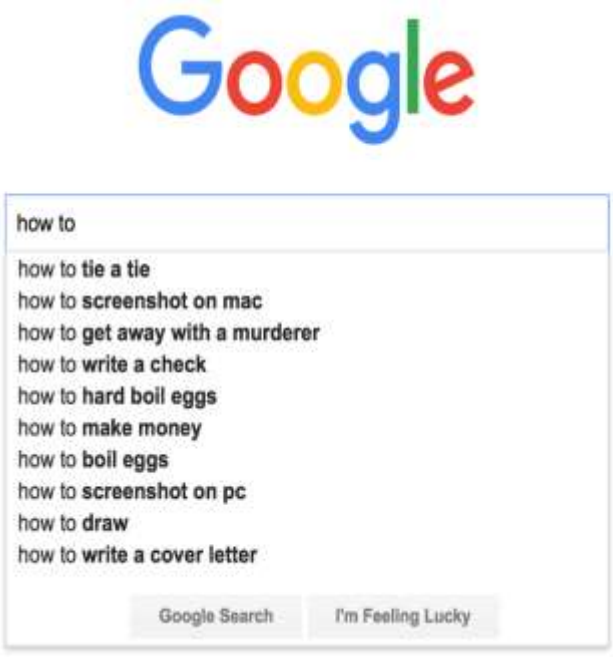


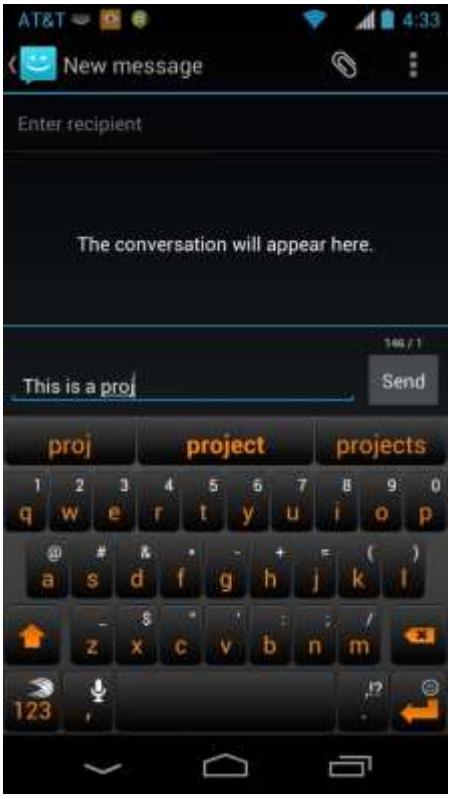
Problem Definition

Text Autocomplete

Autocomplete is pervasive in modern applications. As the user types, the program predicts the complete query (typically a word or phrase) that the user intends to type. Autocomplete is most effective when there are a limited number of likely queries. For example: search engines use it to display suggestions as the user enters web search queries; cell phones use it to speed up text input.

For example:





In this project, you are asked to implement autocomplete for a given set of N terms, where a term is a string and an associated nonnegative weight. That is, given a prefix, find all terms that start with the given prefix, in descending order of weight.

In case of not found prefix, you are asked to find the most similar terms to the input query. Sorted in descending order of weight. The similarity between 2 strings is measured using edit distance.

Related Text Terminologies

String prefix

A prefix of a string **S** is a substring of **S** that occurs at the beginning of it.

Example:

Ban is a prefix of Banana

Edit distance

Edit distance is a way of quantifying how dissimilar two strings (e.g., words) are to one another by counting the minimum number of operations required to transform one string into the other.

Example:

The edit distance between "kitten" and "sitting" is 3. A minimal edit script that transforms the former into the latter is:

- kitten → sitten (substitution of "s" for "k")
- sitten → sittin (substitution of "i" for "e")
- sittin → sitting (insertion of "g" at the end).

Project Requirements

Required Implementation

Requirement	Performance
1. Find strings that has the input query as a prefix	**Time:** should be **bounded by** **O(|Q|.|T|.N)** for each query, where: - **|Q|** is the length of query string - **|T|** is the length of the longest term - **N** is the number of terms in the given data
2. Find most similar terms to the input query (in case if it's not a prefix of any term)	**Time:** should be **bounded by** **O(|Q|.|T|.N)** for each query, where: - **|Q|** is the length of query string - **|T|** is the length of the longest term - **N** is the number of terms in the given data
3. Arranging the results in ascending order of their weight	**Time:** should be **bounded by** **O(N logN . |T|)**. For each query, where: - **N** is the count of matched results - **|T|** is the length of the longest term

Input & Output

Input:

The input file contains the following:

1. The number of queries N
2. N queries, each consists of a query string **Q**. Each query is in separate line

Output

$$\begin{cases} The\ count\ of\ terms\ that\ has\ the\ query\ string\ Q\ as\ a\ prefix \quad (if\ Q\ is\ not\ a\ prefix\ of\ any\ term) \\ Followed\ by\ those\ terms\ sorted\ by\ their\ weights\ (from\ largest\ to\ samllest) \\ \\ The\ count\ of\ most\ similar\ terms\ to\ the\ query\ string\ Q \quad (if\ Q\ is\ not\ a\ prefix\ of\ any\ term) \\ Followed\ by\ the\ most\ similar\ terms\ sorted\ by\ their\ weights\ (from\ largest\ to\ smallest) \end{cases}$$

The output MUST be saved in a FILE.

Test Cases

Sample Test: **Wikitionary.txt**

Complete Test: TBA…

Deliverables

Implementation (70%)

1. Finding terms that has the query as a prefix, sorted in ascending way of their weight
2. Finding the most similar terms to the query string (in case of query string not a prefix of any term)
3. Save the output in a file

Document (30%)

1. Code for finding the terms that has the query as a prefix.
2. Code for Finding the most similar terms to the query string.
3. Detailed analysis of the above codes.

BONUSES

1. Implementation of extra string similarity measure (in addition to edit distance)
2. User Friendly GUI
3. Performance Optimization (complexity better than the given boundaries)