Dictionary and Spell Checker

Efficiency Comparison Between Various Data Structures

Hriday Baghar, Shashwat Singh, Insaf Muhammed Ali,
VIT University,
Vellore, India

***Abstract*— In this project, we have compared various data structures to compare the efficiency of different data structures for a dictionary and a spell checker algorithm. We have used array, tries , ternary search tree implementation in C as well as Java language. This aims in helping the user in catching the errors in his words. The project provides the solution by giving the most efficient data structure.**

***Index Terms*— Dictionary, Spell Checker, Tree.**

Introduction

A simple program for a spell checker for English text. It will contain a list of words of the dictionary and stored in different data structures. The code will check the word and compare it with the words in the dictionary, then check for suggestions. The dictionary will already be sorted. To get suggestions it checks based on some set of rules:

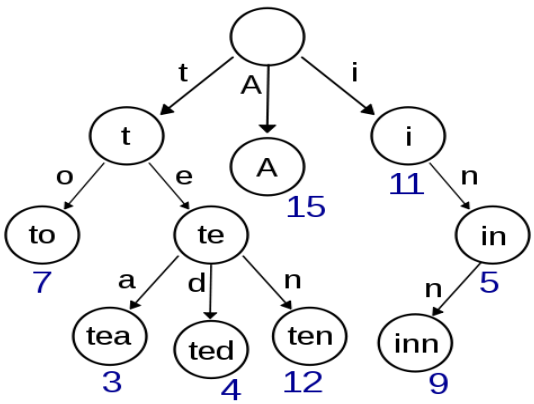
If the word and the suggestion differ by less than 2 letters then it will suggest it.

If the word is a prefix of the suggestion and vice-versa then it will suggest it.

The data structures used in this project are Arrays, Tries and Ternary Search Tree.

I. Trie

Appears pretty simple and may offer quick lookup/insertion. Approximate search using Damerau-Levenshtein algorithm, is straightforward to implement here as well. But it is space efficient as lot of overhead pointers are used for word storage.



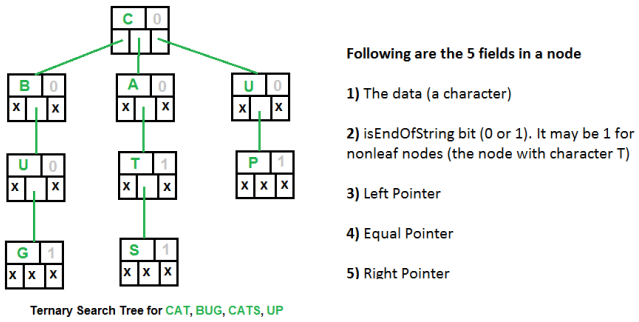
II. Ternary Seacrh Tree

A ternary search tree is a special trie data structure where the child nodes of a standard trie are ordered as a binary search tree.

1. The left pointer points to the node whose value is less than the value in the current node.
2. The equal pointer points to the node whose value is equal to the value in the current node.
3. The right pointer points to the node whose value is greater than the value in the current node.

Apart from above three pointers, each node has a field to indicate data(character in case of dictionary) and another field to mark end of a string.

The disadvantage of Trie is, it requires lots of space. If space is concern, then Ternary Search Tree can be preferred.



III. Array

In this method the words from the dictionary are store in the array in an alphabetical order and performs binary search to find if the word is correct or not.

IV. Algorithms

A The code uses a Levenshtein algorithm to compare the difference between two words. The Levenshtein algorithm (also referred to as Edit-Distance) calculates the least range of edit operations that square measure necessary to switch one string to get another string. The most common way of hard this is often by the dynamic programming approach.

A matrix is initialized measuring in the (m,n)-cell the Levenshtein distance between the m-character prefix of one with the n-prefix of the opposite word.The matrix can be crammed from the higher left to the lower right corner. Each

jump horizontally or vertically corresponds to AN insert or a delete, respectively. The cost is often set to one for every of the operations.

The diagonal jump can price either one, if the two characters within the row and column don't match or zero, if they do. Each cell perpetually minimizes the price regionally. This way the quantity within the lower right corner is that the Levenshtein distance between each words. Here is an example that options the comparison of "meilenstein" and "levenshtein".

		m	e	i	l	e	n	s	t	e	i	n
	0	1	2	3	4	5	6	7	8	9	10	11
l	1	1	2	3	3	4	5	6	7	8	9	10
e	2	2	1	2	3	3	4	5	6	7	8	9
v	3	3	2	2	3	4	4	5	6	7	8	9
e	4	4	3	3	3	3	4	5	6	6	7	8
n	5	5	4	4	4	4	3	4	5	6	7	7
s	6	6	5	5	5	5	4	3	4	5	6	7
h	7	7	6	6	6	6	5	4	4	5	6	7
t	8	8	7	7	7	7	6	5	4	5	6	7
e	9	9	8	8	8	7	7	6	5	4	5	6
i	10	10	9	8	9	8	8	7	6	5	4	5
n	11	11	10	9	9	9	8	8	7	6	5	4

Fig. 1. *Levenshtein Matrix*

ACKNOWLEDGMENT

We would like to thank our teacher to give us the opportunity to carry out this topic and giving us the freedom to choose our topic. We have learned a lot while doing this project.

Lastly we thank VIT University and the dean for giving us this wonderful opportunity to carry out our studies and projects at the university..

REFERENCES

[1] G. Michael Schneider and Steven C. Bruell, Concepts in Data Structures and Software Development, West Publishing Co., St. Paul, MN, 1991.

[2] GeeksForGeeks, Ternary Search Tree- http://www.geeksforgeeks.org/ternary-search-tree/

[3] Trie, Insert and Search- www.geeksforgeeks.org/trie-insert-and-search/

[4] Thomas H. Cormen, Charles E. Leisserson, Ronald L. Rivest and Clifford Stein, Introduction to Algorithms, Third Edition, MIT Press, 2009.