Predictive Text Input and Phonetic Naming Indexing System

Sebastián Patiño Barrientos
Student
Eafit University
spatino6@eafit.edu.co

Luis Miguel Arroyave Quiñones
Student
Eafit University
larroy13@eafit.edu.co

José Luis Montoya Pareja
Teacher
Eafit University

ABSTRACT

In this paper, we describe the data structures and algorithms used in the final project of data structures.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features – *data types and structures.*

General Terms

Algorithms.

Keywords

Autocomplete, Algorithm, Data Structures, Predictive Text, TST, Ternary Search Tree, AVL Tree, Trie, Dictionary.

1. INTRODUCTION

The humanity always has pursued big challenges in order to make life easier in every aspect. One of these challenges is to make machines interact with humans in a way that can help people finish their jobs quickly and with minor effort.

Here, we make a brief explanation of the data structures that we used to develop a system that predicts text by input and indexes the words by their pronunciation, also we perform an analysis of the algorithms complexity with the objective to show the system's efficiency.

2. DATA STRUCTURES

2.1 TST with AVL Tree proprieties

The data structure that we used in our project is a hybrid within the Ternary Search Tree and the AVL tree.

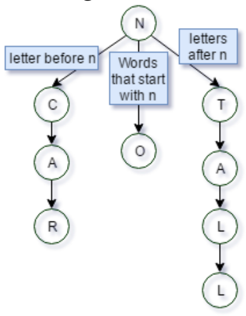
We choose the TST because its time efficient and for the reason that it requires less space than a trie. Also, this structure is faster than hashing for some search operations like misses searches and support operations like sorting.

So, why make a hybrid between TST and an AVL tree? Because the property of self-balance and because the ability to balance itself allows the TST to reduce the time it takes to access the furthest word in the data structure.

2.2 Brief description of a TST

A TST is a type of trie that uses far less memory than the usual trie and is faster than a BST. In a TST the nodes are settled like a BST, but the difference is that the nodes can have up to three children. Each node has a character and his links represents the order of the characters, if the character is greater than, equal to or less than the actual node's character.

In the figure 1 we want to add the word "cat", we have to evaluate



T 1. TST example.

the first node, the first character of the word is "c", the character of the first node is n, so we move to the child at the left side, the character of this node is "c", now we have to evaluate the child in the middle, this node characters is "a", then we evaluate the next node, the character of this node is "r", the character "t" is after "r" in the alphabet. So we add a link at the right side to a new node with the character "r". This is the way that the TST tree inserts a word. To visualize this in a better way look at the figure 2.

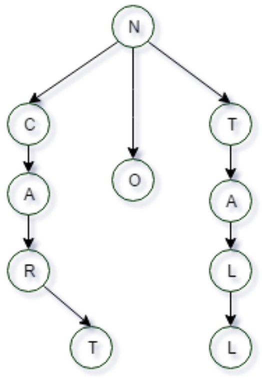


Figure 2. TST insert example.

3. ALGORITHMS

3.1 Algorithms used in the project

In our project we (usamos) the following algorithms:

1. Modified TST search algorithm.
2. TST insert algorithm.
3. Metaphone, a phonetic algorithm.

3.2 Algorithms Complexity Analysis

The complexity of the search and insert algorithms in the TST data structure are O(ln N + L) where L is the length of the search word.

The complexity of the phonetic search algorithm with the TST data structure is O(ln N + W) on average, but in worst of the cases the complexity is O(N+W). Where W is the set of words that are phonetically similar.

3.3 Execution Time

TST search algorithm: 6ms average.

TST insert algorithm: 7ms average.

TST autocomplete operation: 20ms average.

TST phonetic search algorithm: 9ms average.

3.4 Memory Used

TST search algorithm: 170MB.

TST insert algorithm: 168MB.

TST autocomplete operation: 178MB

TST phonetic search algorithm: 176MB

4. SOLUTIONS TO SIMILAR PROBLEMS

4.1 IntelliComplete

"IntelliComplete is an intelligent text productivity software aimed at boosting your typing speed. It helps you type a lot faster by predicting what you want to type based on the prefix you've already typed and prompting you with the best matching guesses. You can have IntelliComplete autocomplete the rest of the word/phrase for you with a single keystroke. What is more, you can type predefined shorthands / abbreviations and have IntelliComplete automatcially expand it into the full text version for you."[1]

4.2 Apple predictive text

"You can type and complete entire sentences with just a few taps.

[…]

As you type, you'll see choices for words and phrases you'd probably type next, based on your past conversations and writing style."[2]

4.3 JetBrains Auto-Completing Code

"Basic code completion helps you complete the names of classes, methods, fields, and keywords within the visibility scope. When you invoke code completion, IntelliJ IDEA analyses the context and suggests the choices that are reachable from the current position of the caret."[3]

4.4 New York State Identification and Intelligence System

The NYSIIS is a phonetic algorithm made in 1970. It is similar to the Soundex algorithm, but a greater accuracy of 2.7%.[4]

5. REFERENCES

[1] FlashPeak Inc. Austin, Texas, U.S.A. Medical Transcription Software Free Download – *IntelliComplete* from http://intellicomplete.com/

[2] Apple Inc. *Apple Predictive Text.* https://support.apple.com/en-eg/ht202178#predictive

[3] JetBrains. Auto-Completing Code. *How to auto-complete code in IntelliJ* https://www.jetbrains.com/help/idea/2016.2/auto-completing-code.html

[4] Rajkovic, P.; Jankovic, D. 2007, *Adaptation and Application of Daitch-Mokotoff Soundex Algorithm on Serbian Names* https://web.archive.org/web/20110827085111/http://sites.dmi.pmf.uns.ac.rs/events/2006/prim2006/Papers/pdf/193-Rajkovic-Jankovic.pdf

Wikipedia. *NYSIIS* https://en.wikipedia.org/wiki/New_York_State_Identification_and_Intelligence_System