C++

Program Of

Word Prediction Application

A Project Report

Submitted by

Nimit Kumar Jain – 18BCE0328
Mansi Saxena – 18BCE0307
Ayesha Moulana – 18BCE0373
MG Shantanu – 18BCE0362

Data Structures And Algorithms (CSE2003)

Project Superviser

Prof. Boominathan

VIT®
UNIVERSITY
(Estd. u/s 3 of UGC Act 1956)
VELLORE ▪ CHENNAI
www.vit.ac.in

ABSTRACT

In this world, with technology developing exponentially, time management and efficieny are important factors in anyone's lives. Technology is being developed each day to *increase the efficiency* and *extend the potency*. In this front, *word predictor* is a minor step which helps in increases our efficiency multifold times. Word predictor has applications in varied fields like texting, search engine and so on. It helps students to enhance their *reading, writing and listening skills*. It also helps in building the *vocabulary* for people of all age groups. It is used in various applications and websites. To build a word predictor program code we use the data structure "*Trie*". This data structure has a time complexity of O(k), where *k* is length of the *partial word* typed by the user. Our program utilizes a file of stored words to predict the word, using the partial word typed, which the user wants to type.

Word Completion or autocomplete is a feature within which an application predicts the remainder of a word, using the sub-part of the word the user has typed. In graphical user interfaces, users can usually press the tab key to simply accept a suggestion or the down arrow key to accept one in many options.

Autocomplete increases the time efficiency and decreases the buffer time of human-computer interactions by predicting the words that the user intends to enter after a few initial characters of the desired word are input. It works best in domains with a restricted number of possible or attainable words, for instance, in command line interpreters, when some words are used a lot more frequently than the others. Addressing an e- mail, or writing a structured and predictable text are examples where word predictor is very helpful.

Many autocomplete algorithms also learn new words once the user has written them several times, and thus can suggest alternatives based on the learned habits of the individual user.

INTRODUCTION

Word prediction is one of the most useful feature built in the word processors that is useful to increase the speed of word. when user types a letter, this program displays the words that start with this letter. When the next letter of the word is typed, an another set of words is displayed. For example, the user wants to type "which". By typing "A" the user may see "apple", "ample", "amber", "axe", and "animal". Then, by adding an "n" to the existing "a", the user may see, "ant", "and", "animal", "antenna", and "analysis". Just by giving number 3, the user will be able to place the word "animal" on the line, that is followed by an automatic space.

This word prediction feature generally makes it easier for a person with a physical disability to type a particular word. It can also give you cues for spelling of the respective words correctly for the ones who are poor in writing the proper spelling of the word. It can correct a user to write "e" at the end of "where". A word prediction program encourages the user in many other ways. First of all, it gives cues for words which it predicts to come in the next. When the word prediction also involves vocal output, some of the programs can be written in such a way to speak the words in the list if you could select them. Therefore, In this way, the user will be able to hear the word list to ensure if that is the word Second, it gives a cue for that incorrect spelling when there are no words in the prediction list or none of the words that had appeared matches the users choice. And also, when young and novice writers need ideas, the word prediction list provides a suggestions of words that can be used in their writing. In Juel's Simple View (1988), is the generation and organization of ideas short listed is the major part of one's writing skill.

However, users with first letter of the word spelling skill find it more useful and use it more successfully. Word prediction gives you an assistive technology for written communication and augmentative communication both, and it is used in both contexts as well. Where as many other great potential writers or augmentative communicators have this skill and some do not.

BASIC PRINCIPLE BEHIND WORKING OF AUTOCOMPLETE

Autocomplete or word completion works in such a way that when the writer writes the *initial few letters* of a word, the program code predicts one or more possible words as options. If the word he intends to write appears in the list. He can select his choice from the displayed options by using the number keys or any other method implemented in the program code.

If the word that the user desires is not predicted and displayed in the list, the writer must enter the *succeeding letter* of the word. Now, the word *list displayed is altered* such that the words provided begin with the same letters as those that have been selected. When the desired word that the user wants appears in the displayed word list, it is selected, and the word is inserted into the text that the user is working on. In another type of word prediction, the words most likely to follow and come after the just written one are predicted, based on recent word pairs that user uses *frequently* in his writing.

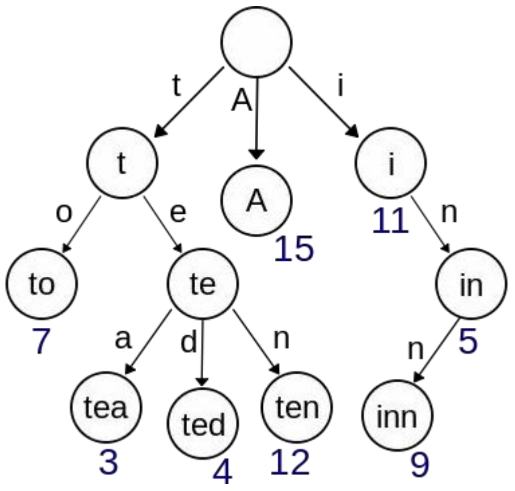
The program code of word prediction utilizes *language modeling*, where within a collection of vocabulary the words, the one ones that are most likely to appear are calculated. In conjunction with language modeling, basic level word *prediction* on AAC devices is often combined with a recency model, where words used more frequently by the *AAC* user are more likely to be predicted and displayed in the list. The word prediction software also permits the user to enter their own words into the word prediction *dictionaries* either directly, or by "learning" words that have been written.

DATA STRUCTURE TRIE

In this program code, Trie data structure is being implemented to search the data in an ordered fashion.

In the field of computer science, a trie, which is also referred to as a *digital tree or sometimes a radix tree or prefix tree* (as they can be searched by prefixes), is a type of search tree. A *search tree* is an ordered tree data structure that is implemented in storing a dynamic set or an associative array, where the keys are usually strings. It works unlike the case of a *binary search tree*. Here, not a single node in the tree stores the key that it is associated with that node. Instead of this, the *position* of the node in the tree defines the key to which it is associated to. The descendants of the particular node have a common prefix of the word desired by the user associated with it. The root is associated with an empty string. Values are not necessarily associated with every node. Instead, values only tend to be associated with *leaf nodes*, and with some inner nodes that correspond to keys of interest according to the user. Refer to a compact prefix tree, for space-optimized presentation of a prefix tree.

In the above example displayed, the keys are shown within the nodes and their values are shown below them. For every complete english word in the file, there is an *arbitrary integer* value associated with it. Thus, a trie data structure can be seen as a tree-shaped *deterministic finite automaton*. So, we see that each finite language is generated by a trie data structure automaton, and each trie data structure can be compressed into a *deterministic acyclic finite state automaton*.



TRIE DATA STRUCTURE ILLUTRATION

LITERATURE SURVEY

Facilitating The Ease of Written Work Using Two Computer Program Codes - Word Processing and Word Prediction

The purpose of this study was to find out whether or not *therapy intervention* that targeted on teaching youngsters to use word processing along with word prediction. Results displayed these two programs were effective in improving the *written communication skills* of children with *learning disabilities and handwriting issues*.

A single-subject, alternating treatment design scenario was replicated across three children studying in grades four and five. In the baseline part or phase, the children were asked to write some stories by hand, while in the intervention phase, children were asked to write stories, alternating among handwriting, and word processing with word prediction. The dependent variables focused on percentages of legible words, percentages of correctly spelled words, total amount written, and rate of writing in their stories. Data was analyzed by *visual inspection*.

The results displayed varied for all 3 children and were variable. While two children had shown clear improvements in their legibility when using either word processing with word prediction. They also demonstrated clear improvements in spelling when using word prediction, from their stories.

Occupational therapy, involving word processing along with word prediction, improves the *legibility, orthography* and spelling of written assignments. These results were completed by some youngsters with learning disabilities and handwriting issues.

EFFECTS OF WORD PREDICTION ON WRITING FLUENCY FOR CHILDERN WITH PHYSICAL DISABILITIES

Many children suffering with *physical disabilities* have trouble in their writing fluency, due to their *motor limitations*. One type of technology, that can help in being assistive to such children, and has been developed to improve writing speed and accuracy, is the word prediction software. However, there is a scarcity of research supporting its use for people with physical disabilities. This study uses an alternating treatment design structure, between word prediction versus word processing. It examines the fluency, accuracy, and passage length by making children write draft papers. These children have physical disabilities. The obtained results indicated that word prediction had negligible or almost no effectiveness in increasing writing speed for all of the students in this research. However, it does show a promising decrease in spelling mistakes and typographical errors.

Writing is a complex, multifaceted task that facilitates interaction between physical as well as cognitive skills. Persons who suffer with physical disabilities differ in terms of both their *physical and cognitive abilities*. generally they should overcome one or more barriers so that they can engage in writing task. Reducing or eliminating obstacles is necessary because opportunities are more for the one's who can effectively communicate their ideas through writing. Assistive technology (AT) is becoming a most effective solution to increase typing fluency. The main purpose of this study is to check if word prediction software, a generally used software program with individuals with learning disabilities, will be useful for those with physical disabilities to increase typing rate and decrease spelling mistakes (fluency). Data is collected for *words correct per minute* (WCPM) and errors (e.g., spelling). Four middle or high school aged people with various physical disabilities will be recruited in this single subject, alternating treatment design. Participants will be allowed to type for three-minute timed sessions through either a standard word processor or Co:Writer 4000, a word prediction software program. Particular research questions are: (a) till what extent will students with physical and health disabilities produce greater WCPM when writing a draft paper on a common topic using word prediction rather than word processing.

ADVANTAGES AND DISADVANTAGES

Writing with a word processor has a number of potential advantages over writing by hand. It is easy to revise text produced on a word processor, as ideas can easily be *added, modified, deleted, and moved*. Word processing *enhances students' motivation* to write. Support to the writer includes *spelling and grammar* checkers, software for formatting text, *speech synthesis* (typed text is converted to speech), *speech recognition* (writers' speech is converted to typed text), planning and outlining software, along with a modern software that provides us with detailed *feedback* on specific aspects of our written text.

An important question then is: *Does word processing have a positive impact on writing for students in general?* The answer to this question appears to be *yes*, as the available scientific evidence supports the contention that word processing is advantageous for students in general.

Did word processing in its various forms enhance the writing performance and motivation for weaker writers/readers? As expected, word processing had a positive impact on the quality and length of text produced by weaker writers/readers. However, it has little effect in increasing motivation.

Contrary to the expectations, the average weighted effective size for grammatical correctness, although positive, was not statistically different. There were no obvious outliers or overly influential observations.

The average weighted effect size for mechanical correctness were statistically different from zero, but effects varied considerably. Influence analyses identified three effects impacting variance estimates, but no effects with excessive leverage that unduly influenced the mean estimate. Improvement in development/organization and mechanical correctness was also noted. Contrary to the expectations, vocabulary in students' writing was not enhanced as a result of using word processing.

Significant gains were obtained when developing and weaker writers were provided with help planning, drafting, or revising text. Students preferred word processing to handwriting and students who wrote via word processing evidenced greater motivation to write.The most intriguing finding from this review was that word processing programs that provide feedback about the quality of text or prompt students to engage in specific activities that supported planning, drafting, or revising of text are particularly effective.

The obvious implication of these findings is that weaker writers/readers should use word processing as their primary tool for composing, since it increases such students' motivation to write and produces superior writing outcomes when compared to writing by hand.

AUTOCOMPLETE AS A RESEARCH TOOL: A STUDY ON PROVIDING SEARCH SUGGESTIONS

Since the library website and its online searching options become the primary branch most of the users come for their research, methods for providing automated, context-sensitive research assistance must be developed so that we provide a guide unmediated searching for the most appropriate results. This study checks one such method, the use of autocompletion in search interfaces, by conducting usability tests on its use in proper academic research sessions. The study reports findings on user preference for autocomplete features and suggests more appropriate practices for its implementation.

OUTPUT-SENSITIVE AUTOCOMPLETION SEARCH

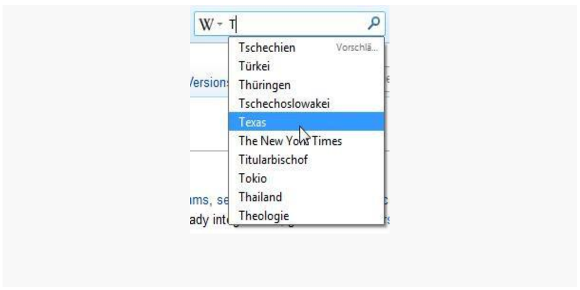
Let us have a look at the following auto completion search scene: imagine a user who is using a search engine to type a query; then every keystroke display those completions of the last query word that would take you to the best hits, and also display the best such hits. The given problem is at the core of this feature: for a fixed document collection, given a set D of documents, and an alphabetical range W of words, calculate the set of all word-in- document pairs (w, d) from the collection so that w ∈ W and d ∈ D. We display a new data structure with the help of which such autocompletion queries can be looked after, on average, in time linear in the input and also output size. Actual query processing times on a large test collection correlate perfectly with theoretical bound.

A PREDICTIVE TEXT COMPLETION SOFTWARE IN PYTHON

Predictive text completion is a technology that extends the traditional autocompletion and text replacement techniques. It helps to reduce key strokes needed in text input and serves as a more affordable assistive technology for computer users who are dyslexic or has learning/reading difficulty/disability, as compared to more expensive speech-to-text technology or special input devices. Python is used for prototyping, rapid R&D and testing of advanced features, while AutoHotKey scripting language can be used to code the regular stable release.

APPLICATION

- **In web browsers**



In web browsers, wordcomplete is done in the address bar and in text boxes on most widely used pages, just like search engine's search box. autocomplete for web addresses is mainly convenient because the full addresses are often long and very difficult to type properly. HTML5 has an autocomplete form attribute.

➢ **In e-mail programs**

In e-mail programs autocomplete is generally used to fill in the e-mail addresses of the intended recipients. Generally, there are a small number of frequently used e-mail addresses, hence it is comparitively easy to use autocomplete to select among them. Just Like web addresses, e- mail addresses are often long, hence typing them completely is pretty difficult.

For example, Microsoft Outlook Express will find addresses based on the name that is used in the address book. Google's Gmail will find addresses by any string that mostly occurs in the address.

➢ **In search engines**

Generally in most if the search engines, autocomplete user interface options provide users with suggested questions or answers as they type their question in the search box. This is also called as autosuggest or incremental search. This type of search often depends on matching algorithms that ignore entry errors such as phonetic Soundex algorithms or the language independent Levenshtein algorithm. The challenge left is to search large indices or popular query lists in under a few milliseconds so that the user sees results pop up while typing.

Autocomplete can effect adversly on individuals and businesses when negative search words are suggested when a search takes place.

Autocomplete has now become a part of reputation management because companies linked to negative words such as scam, complaints and fraud tend to alter the results. Google particularly has listed some of the factors that affect how their algorithm works, but this is an area which is open to manipulation.

➢ **In source code editors**

In a source code editor autocomplete is simplified by the regular structure of the programming languages. There are generally only a limited number of words meaningful in the current context or namespace, such as names of B to choose the right one. This is mainly useful in oops because generally the programmer will not know exactly what members a specific class has. Therefore, autocomplete then serves as a form of convenient documentation and also as an input

method. Also the most useful option in autocomplete for source code is that it encourages the programmers to use it for longer, more descriptive variable names having both lower and upper case letters (DataStructure), hence making the source code more easy to read. Typing lengthy words with many mixed cases like "numberOfwordsINparagraph" can be difficult, but Autocomplete allows you to complete typing the word using a fraction of the keystrokes.

➢ **In database query tools**

Autocompletion in a Database Query Tools facilitates the user to autocomplete the names in SQL statement and column names of the tables referenced in the SQL statement. As soon as the text is typed into the editor, the context of the cursor within the SQL statement gives an indication of if the user needs a table completion or a table column completion.

The table completion gives you a list of tables available in the database server the user is connected to. The column completion provides a list of columns for only tables mentioned in the SQL statement.

➢ **In word processors**

In many word processing programs, autocompletion decreases the time spent in typing repetitive words and phrases. The source file for autocompletion is either collected from the left part of the current document or from a list of common words defined by the user. Right now Apache OpenOffice, Calligra Suite, KOffice, LibreOffice and Microsoft Office are providing a strong support autocompletion, as do advanced text editors such
as Emacs and Vim.

Apache OpenOffice Writer and LibreOffice Writer have a working word completion program that shows words previously typed in the text, other than from the whole dictionary Microsoft Excel spreadsheet application has a working word completion program that gives words previously typed in upper cells

➢ **In command-line interpreters**

In a command-line interpreter, such as Unix's sh or

bash, or Windows's cmd.exe or PowerShell, or in similar command line interfaces, autocomplete of command names and file names may be accomplished by keeping track of all the possible names of things the user may access. Here autocomplete is generally done by pressing
the TAB key after typing the first some letters of the word. For example, if the only file in the current directory that starts with x is xLongFileName, the

user may prefer to type x and autocomplete to the complete name. If there were another file name starting with x the user would type more letters or press the Tab key repeatedly to select the appropriate text.

OTHER SEARCHING ALGORITHMS AND THEIR TIME COMPLEXITIES

OTHER SEARCHING ALGORITHMS	**TIME COMPLEXITY**
LINEAR	O(n)
BINARY	O(log n)
JUMP	O(sqrt. n)
INTERPOLATION	O(log log n)
EXPONENTIAL	O(log n)
SUBLIST	O(n)
FIBONACCI	O(log n)
UBIQUITOUS BINARY	O(log n)
TRIE	O(k)

Where,
k is the length of partial word, and
n is the number of words in the dictionary.

PSEUDOCODE

```
class Node:
        Private char mContent
      Private bool mMarker
        Private vector <Node*> mChildren
        Public Node( )
        Public ~Node( )
        Public char content( )
        Public void setContent(char c)
        Public bool wordMarker()
        Public void setWordMarker()
        Public void appendChild(Node* child)
        Public vector<Node*> children( )
        Public Node* findChild(char c)

Node:: Node( ):
        mContent ← ‘ ’
        mMarker ← False

Node::~Node( ):

char Node::content( ):
        return mContent

void Node:: setContent(char c):
        mContent ← c

bool Node:: wordMarker( ):
         return mMarker

void Node:: setWordMarker( ):
        mMarker ← true

void Node::appendChild(Node* child):
        mChildren→push_back(child)

vector<Node*> children( ):
        return mChildren

Node* Node::findChild(char c)
         For  i ← 0 to size of mChildren
        Node* tmp  =  mChildren→at(i)
                If  tmp→content( ) = c
                      return tmp
                End If
        End For
       return NULL
End findChild()

class Trie:
        Private Node* root;
        Public Trie()
        Public ~Trie()
        Public void addWord(string)
        Public bool autoComplete(string, vector<string>)
        Public void parseTree(Node*, char*,  vector<string>, bool)

Trie:: Trie( )
```

```
        Root← new Node( )

Trie::~Trie()

void Trie::addWord(string s):
        Node* current ← root
       If   s→length( ) = 0
                Current→ setWordMarker( )
                 Return
        End If

         For i ← 0 to length of string s:
                 Node* child = current->findChild(s[i])
                If  child ~= NULL
                        current = child
                Else
                 Node* tmp = new Node()
                         Tmp→ setContent(s[i])
                Current→ appendChild(tmp)
                current ← tmp

                If   i = length of string s - 1
                       current → setWordMarker()
                End If
        End For
End addWord()

bool Trie::autoComplete(s, res):
        Node* current ← root
        For i ← 0 to length of string s:
                Node* tmp ← current→ findChild(s[i])
                If temp=NULL:
                        return False
                End If
                current ← tmp
        c ← s
        loop ← True
        Trie::parseTree(current,c,res,loop)
        return True
End autoComplete()

void Trie::parseTree(current, s, res, loop):
        k←new Array
        a← new Array
        If loop=True:
                If current~=NULL:
                        If current → wordMarker()==True:
                                Insert s to res
                                If size of res >20:
                                        Loop← False
                                End if
                                child ← current→children()
                                i←0
                                do until loop=True and i<sizeof(child):
                                        k←s
                                        a[0]←child[i]→content()
                                        a[1]←'\0'
                                        k←k+a
                                        If loop=True:
                                                Trie::parseTree(child[i], k, res, loop)
                                        End if
                                        i ← i+1
                                End do
                        End if
                End if
        End if
End parseTree()


bool  loadDictionary(trie, filename)
         ifstream words
         ifstream input
        words→open(filename→c_str( ))
        If  ~words→is_open( )
             Display "Dictionary file Not Open"
             return False
        End If
        While ~words→eof()
                char array s
                words Read s
                trie → addWord(s)
        End While
        return True
End loadDictionary()

void WriteNewWord(Trie *trie):
        display "Enter the word : "
        input NewWord
        onlyAlpha ← true
        for i is a character in NewWord:
                if i is not an alphabet:
                        onlyAplha← false
                        exit from loop
        End for
        If onlyAlpha is true:
                convert NewWord to lowercase
                 trie→complete(NewWord,ListOfWords)
                If size of ListOfWords is greater than 0:
                        display "NewWord already exists in the disctionary"
                Else:
                        ofstream out
                        out.open("wordlist.txt",ios::app);
                        if out is not open :
                                display "Sorry!\nCould not open the dictionay"
                                out.close()
                        Else:
                                write NewWord in the "wordlist.txt" file using "out" object
                                display "Successfully loaded in the disctionary"
                                out.close()
                                trie→addWord(NewWord)
                                exit from this function
                        End if
                End if
        Else:
                Display "Not a Valid word!"
                Exist from the function
        End if
End WriteNewWord()

int main():
        Trie* trie ← new Trie()
        mode←"
        display "Loading dictionary"
        loadDictionary(trie, "wordlist.txt")
        WHILE True:
                display “Interactive mode, press”
                display “1. Auto complete feature”
                display “2. Enter the new word into the dictionary “
                display “3. Quit”
                input mode
                if mode is an alphabet:
                        display ”Invalid Input!”
                        display “Enter either 1 or 2..”
                        continue the loop
                end if
                CASE mode:
                        ‘1’:
                                display “Enter the partial word : “
                                input s
                                convert s to lowercase
                                autoCompleteList← new array of strings
                                trie→autoComplete(s, autoCompleteList)
                                IF size of array autoCompleteList=0:
                                        display “No Suggestions”
                                        display “Do you want to enter this word into the memory?(y/n)”
                                        read pp
                                        If pp==’y’ or p==’Y’:
                                                WriteNewWord(trie)
                                        End if

                                ELSE:
                                        Display “Autocomplete reply :”
                                        FOR i←0 to sizeof(autoCompleteList):
                                                Display autoCompleteList[i]
                                        End FOR
                                End IF
                                Continue the loop
                        ‘2’:
                                WriteNewWord(trie)
                                Continue the loop
                        ‘3’:
                                Delete trie
                                Return 0
                        Others :
                                Display “Invalid Input!”
                                Continue the loop
                        End CASE
End main()
```

CONCLUSION

Word predictor is utilized in messaging applications like WhatsApp, Web Search Engines, Word processors, command like interpreters and so on. The original need or purpose of the word prediction software was to help people of all ages, children, students and adults, who are suffering with physical disabilities in increasing their typing speed. It also helps them in decreasing the number of keystrokes needed in order to complete a word or a sentence.

Thus, in this manner, we developed our own program using Trie data structures to implement the concept of word predictor, which definitely increases efficiency of the user by at least 10%.

REFERENCES

- ✓ Dottie Handley-More; Jean Deitz; Felix F. Billingsley; Truman E. Coggins
- ✓ American Journal of Occupational Therapy, March/April 2003, Vol. 57, 139-151. doi:10.5014/ajot.57.2.139
- ✓ Effects of word prediction on writing fluency for students with physical disabilities peter j.mezei and kathryn wolff heller Georgia State University
- ✓ Autocomplete as a Research Tool: A Study on Providing Search Suggestions David Ward, Jim Hahn, and Kirsten Feist https://ejournals.bc.edu/ojs/index.php/ital/article/download/1930/pdf
- ✓ Output-Sensitive Autocompletion Search-- Holger Bast , Christian W. Mortensen , and Ingmar Weber https://people.mpi-inf.mpg.de/~bast/papers/autocompletion-spire.pdf
- ✓ Samuel Pouplin, Johanna Robertson, Jean-Yves Antoine, Antoine Blanchet, Jean Loup
- ✓ A Predictive Text Completion Software in Python -- Wong Jiang Fung

  http://ojs.pythonpapers.org/index.php/tppm/article/download/132/13
- ✓ Evaluating the Benefits of Displaying Word Prediction Lists on a Personal Digital Assistant at the Keyboard Level Cynthia Tam BScOT MSc &David Wells PhD
- ✓ Heidi Horstmann Koester, Sajay Arthanat. (2018) Effect of diagnosis, body site and experience on text entry rate of individuals with physical disabilities: a systematic review. Disability and Rehabilitation: Assistive Technology 13:3, pages 312-322.
- ✓ Yaakov HaCohen-Kerner, Asaf Applebaum, Jacob Bitterman. (2017) Improved Language Models for Word Prediction and Completion with Application to Hebrew. Applied Artificial Intelligence 31:3, pages 232-250.
- ✓ Stephen Ryan, Mary-Beth Sophianopoulos. (2017) Measurement of Assistive Technology Outcomes Associated with Computer-Based Writing Interventions for Children and Youth with Disabilities. Technologies 5:2, pages 19.
- ✓ Gerd Berget, Frode Eika Sandnes. (2016) Do autocomplete functions reduce the impact of dyslexia on information-searching behavior? The case of Google. Journal of the Association for Information Science and Technology 67:10, pages 2320-2328.
- ✓ Samuel Pouplin, Nicolas Roche, Isabelle Vaugier, Antoine Jacob, Marjorie Figere, Sandra Pottier, Jean-Yves Antoine, Djamel Bensmail. (2016) Influence of the Number of Predicted Words on Text Input Speed in Participants With Cervical Spinal Cord Injury. Archives of Physical Medicine and Rehabilitation 97:2, pages 259-265.
- ✓ Guilherme Matheus M.A. Ramos, Victor O.N. Sales, Cesar A.C. Teixeira. (2016) LETRAS. Proceedings of the 22nd Brazilian Symposium on Multimedia and the Web -Webmedia '16, pages 215-218.
- ✓ Nethanel Gelernter, Amir Herzberg. 2016. Autocomplete Injection Attack. Computer Security – ESORICS 2016, pages 512-530.

AUTHOR REFERENCES

This report was submitted on the 8th of April, 2019, on Monday.

N.K.J Author

(nimitkumar.jain2018@vitstudent.ac.in) is a BTech CSE student studying in Vellore Institute of Technology and is in IEEE-VIT and ACM-VIT chapters.

M.S Author (mansi.saxena2018@vitstudent.ac.in) is a BTech CSE student studying in Vellore Institute of Technology and is in the Dance Club and Leo Club.

A.M Author (ayeshahruksaar.2018@vitstudent.vit.ac.in) is a BTech CSE student studying in Vellore Institute of Technology and is in the Leo Club.

M.G.S Author (shantanum.g2018@vitstudent.ac.in) is a BTech CSE student studying in Vellore Institute of Technology and is in the chapter Ventursity-VIT.

INDEX TERMS

AAC user
Algorithm
Assistive technology
Associative array
Binary search tree
Buffer time
Cognitive abilities
Data structure
Descendants
Dictionaries
Digital tree
Dynamic set
Efficieny
Grammar
Handwriting issues
Language modeling
Leaf nodes
Learning disabilities
Legibility
Length
Mechanical correctness
Motivation
Motor limitations
Node
Occupational therapy
Orthography
Partial Word
Physical abilities
Physical disabilities
Predicts
Prefix tree
Quality
Radix tree
Real-time program
Root
Search tree
Spelling
Trie
Visual inspection
Vocabulary
Word predictor
Words correct per minute
Writing fluency
Deterministic acyclic finite state automaton
Deterministic finite automaton
Therapy intervention
Written communication skills