Spell Check and Auto Suggest Using Trie

Yuyang Liu, Raj Sundhar, Jiyu Zhang
Rutgers University
Piscataway, NJ, USA
Email: yl1158@scarletmail.rutgers.edu
rr966@scarletmail.rutgers.edu
jz644@scarletmail.rutgers.edu

***Abstract*— With the rapid growth in our modern day communication, the spelling errors we make, are increasing at a very high rate. It is essential to have a spell check before sending out an email or submitting a technical paper. With the trie algorithm this can be achieved with a time complexity in the order of O(k), where k is the size of the word. But, in order to achieve this faster rate of accessibility, we have to sacrifice on the space complexity, which is in quadratic polynomial order.**

I. Project Description

We plan to construct a spelling check application using Trie algorithm. The application is initially loaded with a collection of words scrapped from a dictionary. The user can insert, delete or query a word from this data set. The application will also have an option of spelling check, where the user will enter a sentence or a paragraph and the application will return the words that are misspelled. This project is an implementation of an algorithm that is not been covered in the class.
The trie (also called as Radix tree or index tree) is a special kind of binary search tree, where every node of trie consists of multiple branches and each node represents a character of a word. This kind of structure is useful to implement search engines and dictionaries. The most useful character of this algorithm is its time complexity. The time for insertion and querying is O(k), where k is the length of the input key. Thus it is very fast to add and lookup for words using trie. Since the time complexity is of linear order, this algorithm is novel to use in spell check and search engine technologies.

The Main Stumbling blocks of the project are:
1.To decide a most suitable data structure to implement the Trie
2.To scrap all the list of words from a dictionary and maintain it in our application
3.Design a good animation to depict the working of trie algorithm
4.To adapt this algorithm to addon search engine index suggestions

Time Line:
Due October 25, analyze the requirement and decide the basic funcitons.
Due November 1, complete the core algorithm and finish collecting data.
Due November 20, build up UI and implement animation(if necessary).
Due December 1, optimize the system and algorithms.

Major milestone:
Core code: use Java to finish the code and algorithms.
Data collection: collecting the most often used words from Google and other website.
UI: use Java UI to make a basic user interface and try to use HTML and CSS to make a cool animation.
Optimize: Try to study reconstruct the system design and adjust the data structure to optimize.

A. Stage1 - The Requirement Gathering Stage.

- **General System Description** The spelling-check feature is a widely known feature that is being used by many people. It helps users to correct the sentence in their writings. The admin user can also add or delete words in the data by insertion or deletion operation.
- **Two Access Types of Users** Teachers, Students
- **Interaction Modes** Checking Mode, Administrator Mode
- **Real World Scenarios**
  - Scenarios 1 description:
    Students who have finished their homework and need to check typos in their scripts.
    Teachers who have built their lecture notes and need to check typos before present it to their students.

    Data Input:
    Sentences.

    Input Data Types
    Set of strings.

    Data Output
    Corrected Senteces.

Output Data Types
Set of strings.

- Scenarios 2 description:
  Embedded in Office software, where user can check the spelling when typing.
  The administrator is the software engineers who can change the dataset.

  Data Input
  Words.

  Input Data Types
  Set of strings.

  Data Output
  New dictionary and Boolean Values(Ture or False).

  Output Data Types
  Set of Strings.

- **Project Time line and Divison of Labor.**
  Due October 25, analyze the requirement and decide the basic funcitons.
  Due November 1, complete the core algorithm and finish collecting data.
  Due November 20, build up UI and implement animation(if necessary).
  Due December 1, optimize the system and algorithms.

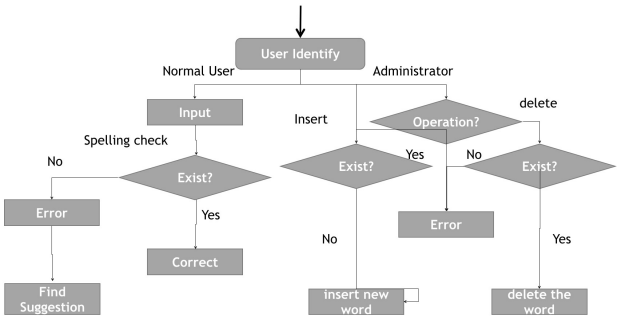
B. Stage2 - The Design Stage.

- **Short Textual Project Description.** There are three basic operations in the system: spelling check, insertion and deletion. Spelling check feature is used to check the correctness of the input's spelling. It compares each word with the trie and checks if any word is misspelled. Insertion and deletion options can be used to add or delete words from the trie. In addition, if any words are spelled wrongly it will give suggestions base on the edit distance between the input words and the words in the tree. It will return the words with small distance efficiently.

  Time Complexity:
  LookUp - O(L) where L is the length of the word
  Insertion - O(L) where L is the length of the word
  Deletion - O(L) where L is the length of the word

  Space Complexity:
  Worst Case - O(W*L)
  W - No. Of Words
  L - Length of the longest word

- **Flow Diagram.**



- **High Level Pseudo Code System Description.**

```
Define:
    Insert (node, word)
Input:
    node – The root node of a Trie,
    which is a Tree with root node as
    empty and all other nodes pointing
    to another node and the leaf node
    marked as EndOfWord

    word – The word need to be inserted
Output:
    the new trie with the word in it

    For each char in word
        If node.children contains char
            node <– node.child
            Continue;
        else
            node.add(char)
            node <– node.child
        if char is the last
            node <– EndOfWord;

Define:
    Delete(node, word)
Input:
    node – The root node of a Trie,
    which is a Tree with root node as
    empty and all other nodes pointing
    to another node and the leaf node
    marked as EndOfWord

    word – The word need to be inserted
Output:
    the new Trie without the word in it.

    For each char in word
        If node.child(char) is EndOfWord
            remove EndOfWord
            delete node.child(char)
    While node.child is null
```

```
        delete node
        set node <- node.parent

Define:
    Search(node, word)
Input:
    node - The root node of a Trie,
    which is a Tree with root node
    as empty and all other nodes pointing
    to another node and the leaf node
    marked as EndOfWord

    word - The word need to be inserted
Output:
    TRUE or FALSE - TRUE means the word
    spelled correctly, FALSE means wrong.

    For each char in word
        If char is last
            If node.child(char) is EndOfWord
                return TRUE
        else
            return FALSE
        If node.children contains char
            node <- node.child
            Continue
        else
            return FALSE


Define:
    SpellCheck (node, sentence)
Input:
    node - The root node of a Trie,
    which is a Tree with root node
    as empty and all other nodes
    pointing to another node and
    the leaf node marked as EndOfWord

    sentence - The paragraphs need to check
Output:
    the word list that mistakenly spelled.

    List <- empty
    For each word in sentence
        spellCheck <- search(word)
        If spellCheck is True
            Continue;
        else
            List.append(word)
    return List


Define:
    editDis(word, word)
Input:
    two words
output:
    the value of the edit distance between
    two words

    for i = 0, 1, 2, 3, ..., m:
        E(i,0) = i
    for j = 0, 1, 2, 3, ..., n:
        E(0,j) = j
    for i = 1, 2, ... , m:
        for j  = 1, 2, 3,..., n:
            E(i,j) = min{E(i-1, j)+1,
                         E(i, j-1) + 1,
                         E(i-1, -1) + diff(i, j)}
    return E(m, n)
```

Trie is an efficient data structure to retrieve data compared to Binary Trees and HashTables. The words are represented in a tree and each node represents a possible character of a word. The end of the word is marked by EndOfWord flag. Each level of the trie is constructed as an array with length at most 26.

- **Flow Diagram Major Constraints.**
  - Integrity Constraint.
    Insertion and Deletion operations are provided only to the administrators and normal users can't do any write operations on the data
    The administrators potentially have all the write access on the data and can effect the data

C. Stage3 - The Implementation Stage.

We used Java as our language to write codes, because it is easy to use and also powerful. The running environment is on JRE 1.8 or higher version than that. We plan to export the whole project into a JAR, and implement UI with JFrame.

- Sample small data snippet.
  "Imagination is the beginnin of creations You imagine wha you desire; you will what you imagine and at last you create what your will."
- Sample small output.
  Invalid Words are ::
  beginnin suggestion:[beginning]
  wha suggestion:[whale, what, whatever, whats]

- Working code
  For stage3 we have finished the core functions of operation of a Trie, including inserting, deleting, searching and returning suggestions. We also built the dictionary, inserting about 10000 words into our Trie.
  Now we store our data in .txt files, and have implemented the basic functions to read or write the file.
  Firstly, in order to get the correct subject need to be checked we do simple processings on the input. We delete all space at the starting and the ending of the string, and chop them into words. Secondly, we searched each word of our input in the Trie, added those cannot be found into "Invalid" wordlist. Finally, for each invalid word, we tried to find at most 5 similar words as spelling suggestion.
  The next step is building a simple UI. Using Java UI, we can show the result in a more direct and clear way.

- Demo and sample findings
  - Data size:
    Our project is to check the spellings of words by

using Trie. The dictionary we used contains 9903, which is a .txt file. Below is a sample of its data snippet: (each line is a word in the list).

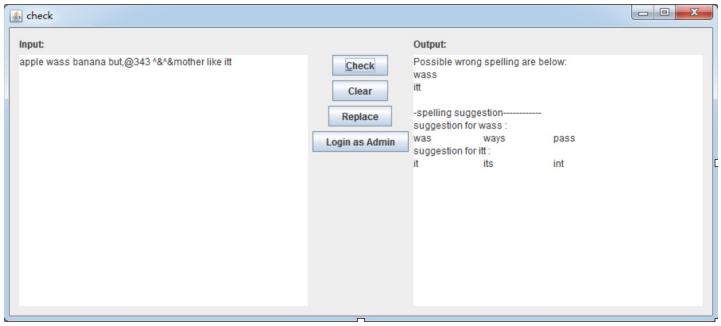
```
antiques
density
hundred
ryan
operators
strange
sustainable
philippines
statistical
beds
mention
innovation
pcs
employers
Grey
```

Then we insert each words to build the whole Trie (to insert all the words cost 17ms). We also test our program on checking a word and finding the suggest words if the word is incorrect, those two operations only takes less than 0.005ms on average.

The total size of a Trie including the whole dictionary is 4027008 Bytes.

D. Stage4 - User Interface.

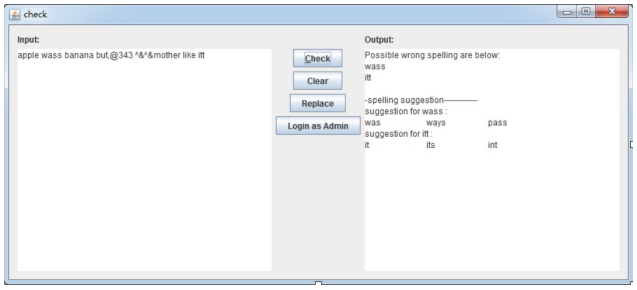
The User Interface of our Trie-checking program is basically designed for two kinds of users: the normal checking users and the administrators of our system. Below are the main interactive view of our project:



The text area on the left is the Input text, user can put their sentences here which would be check. The system will automatically leave out the irrelevant symbols like numbers or space, and it chop sentence into an array of words.

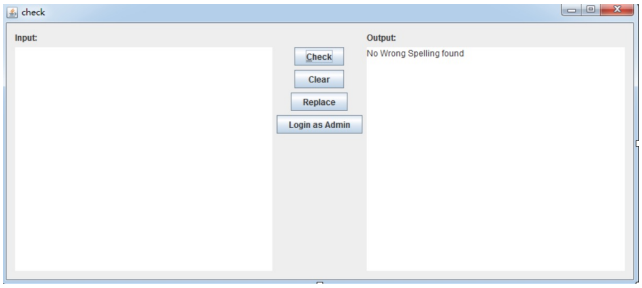
And the text area on the right is the output area. This view is not editable. It only used to show out the checking results and the spelling suggestion.

After input the text in the input area, users can click 'check' button which will show the results. The results include showing all incorrect spellings and listing at most three possible suggestion spelling for each word. Like in the picture1, the "wass" is suggested as "was", "ways" and "pass". And user can replace all the incorrect words by the first suggestion word by clicking the button "Replace". The result is like below:

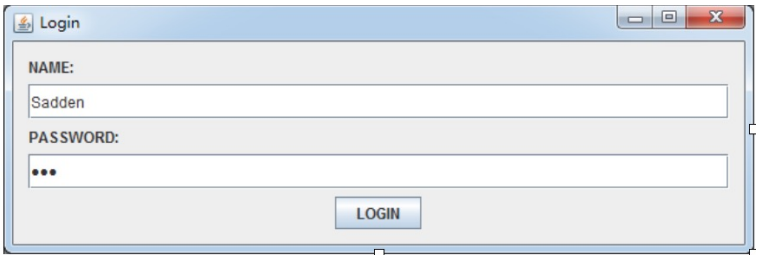


To clear up both two textboxes, user can click the "Clear" button.

If the input area is empty or the sentences are all right. The output will show " No Wrong Spelling found" like below:



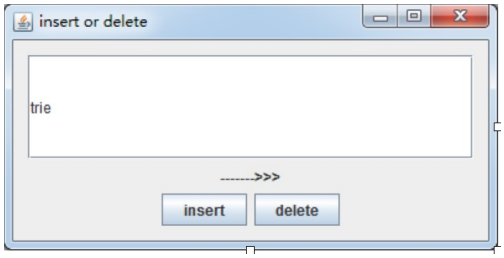
When users need to insert new words or delete a existing word from the Trie, they need to login as administrator. Click "Login as Administrator", a new Login Window will jump out and let user to input username and password. The Login window is like below:

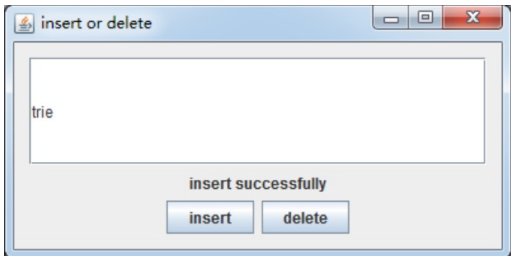


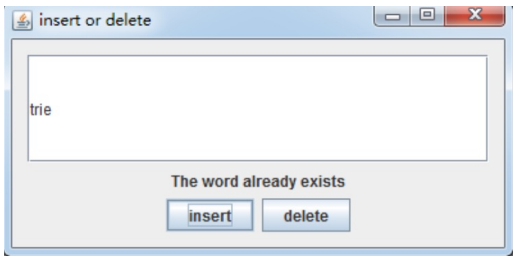
If the password matches the username, when clicking the Login button, the a new view will come out and login window will close. If either the username or the password is incorrect, the two text block will be cleared and let user input again.

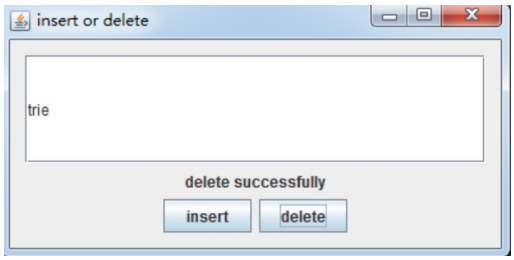
After user login, he or she can insert or delete words in the next view. User can enter the word in the text area and

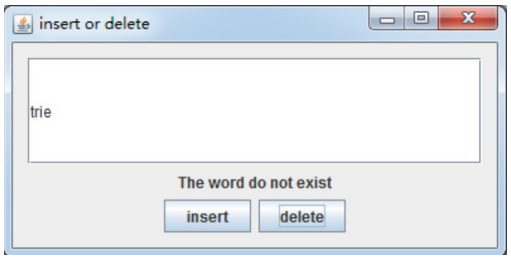
click ”insert” or ”delete” button to insert or delete it. If the insertion is successful, it will show ”insert successfully” below the input text. If the word is already in the Trie, it will show ”the word already exists”. As for deletion, if it is successful, it will show the success information. And if cannot find the word in the Trie, it will show ”The word do not exist”.











Here’s another version of our UI example of adding and removing words from the Trie