AUTO COMPLETE FEATURE FOR DATA STRUCTURE RELATED TERMS

A MAJOR PROJECT REPORT

Submitted by

CH.EN.U4AIE20021 HARISH B

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE

AMRITA SCHOOL OF ENGINEERING, CHENNAI

AMRITA VISHWA VIDYAPEETHAM

CHENNAI 601 103

ABSTRACT

This project is an attempt to implement the autocomplete feature for data structure-related terms using the trie data structure. Trie is an efficient information retrieval data structure. Trie data structure helps in bringing down the search complexities to an optimal level, which makes it more efficient for data retrieval. The program code is done in the Java programming language. We have also implemented a GUI with the help of the swing framework, to enhance the user interaction with the program. Apart from auto-suggestions display in the GUI window, the user has the functionality to insert and delete terms. Also, the information about the terms entered by the user will be displayed in the GUI, which has been implemented with the help of the JSoup library for web scrapping.

Keywords: Autocomplete feature, Trie data structure, GUI, Web scrapping.

1. Introduction

The auto-complete feature is one of the most used features in messaging apps and social media platforms. This feature completes the word or phrase after the user has typed the prefix of the complete word. With the help of this feature, the users can type quickly and also without any spelling errors. In this project, we have tried to implement a model of this autocomplete feature using a trie data structure. The autocomplete feature we have implemented only predicts and suggests the data structure-related terms. Also, a GUI is designed to enhance the user interactivity with the code. In the GUI window, the suggested words will be shown at the JTextArea. When the search button is clicked, a brief definition of the term will be displayed in the JTextPane.

1.1 Trie Data Structure

Trie is also called the information retrieval data structure. This data structure performs search operation at a very optimal rate with a search complexity of O(L), where L is the maximum string length. Whereas in binary search trees the time complexity will be L*log(N) where L is the maximum length of the strings and N is the number of keys in the tree. With trie, we can insert and search in O(L) time. The main drawback of using tries is that they take a lot of memory to store the strings.

Trie data structure allows us to do a prefix-based search which is necessary for our project to implement autocomplete feature. In the project, we have used HashMap to implement trie data structure rather than arrays. A HashMap is a data structure that maps certain keys to certain values. Arrays consume more memory than HashMaps, which is the reason for choosing HashMaps over arrays. In trie data structure some nodes have many branches, each representing possible characters of keys. Each node has a field that

contains a boolean value whether this node is the last character of the word or not (also known as the Leaf node).

In trie data structure the main operation includes insertion, searching, deletion. In insertion operation, we insert every character of the input word as an individual trie node. There will be children of the HashMap which is an array of references to next-level trie nodes. And the character of the input word is an index of the children. If the input word/key is new or an extension of the existing key, we need to construct non-existing nodes for those keys and mark the last character as the last node.

Searching for a key is similar to the insertion operation, but in the search operation, we compare the characters and move down. The search will terminate at the node where the end of the word is true. The search also terminates when the character doesn't match the required characters.

In the Deletion operation, we do the deletion of a word from the trie. There are many cases in the deletion operation. If the key is not in the trie, no deletion operation occurs. If the key is a unique key which means it doesn't associate with any other prefixes nor any extension of this key then we can delete all nodes related to the key. If the key is the prefix of another long key then simply unmark the leaf node as the end word.

1.2 Web Scraping

Web scraping is the process of using bots to extract content and data from a website. Web scraping extracts underlying HTML code and, along with it the data stored in a database. The scraper can then replicate the entire website content elsewhere.

JSoup is an open-source project which provides a powerful API for data extraction. We can use it to parse HTML content from URLs, files, and Strings.

It can also manipulate HTML elements or attributes. JSoup is a Java library for working with real-world HTML content. It provides a very convenient API for fetching URLs and extracting and manipulating data, using the best of HTML5 DOM methods and CSS selectors. JSoup implements the WHATWG HTML5 specification and parses HTML to the same DOM as modern browsers do.

2. Implementation

In the implementation, we have 4 classes with names TrieNode, Trie, Gui, and wiki. The UML diagram below gives the layout of our program and how it is constructed.

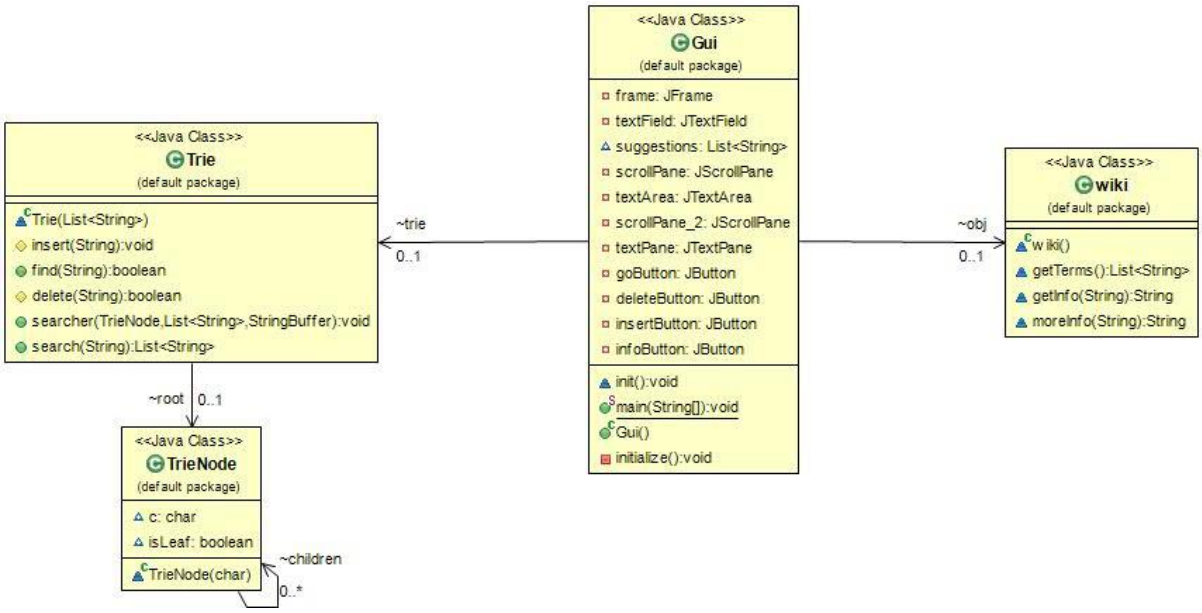

Fig.2.1. UML Diagram

2.1 TrieNode Class

TrieNode class consists of the node implementation of trie data structure. It has a HashMap variable ‘children’ which maps the key ‘Character’ (characters of the string word) to the values ‘TrieNode’. HashMap is an implementation of the Map class in java. The class also consists of instance variables, ‘c’ (char data

type) and ‘isLeaf’ (Boolean data type). Here, the isLeaf variable denotes the last character of the word. There is a constructor which initializes the ‘c’ variable and children HashMap when the objects for the class are created.

```
class TrieNode
{
	Map<Character, TrieNode> children;
	char c;
	boolean isLeaf;

	TrieNode(char c)
	{
		this.c = c;
		this.children = new HashMap<>();
	}
}
```

2.2 Trie Class

In the Trie class, there are five methods methods namely insert(), find(), delete(), search() and searcher(). In the constructor, we will be inserting the list of words into the trie which is done with the insert() method. In the insert() method we are inserting the characters of the word into the trie. First, the input word string is converted into an array of characters, and then we will check whether a node with the given character key is already present or not. If present then the program continues with the next character or else it will create a new TrieNode and is put into the children's HashMap implementation with the character as key. The program continues until it reaches the last character of the word and marks the isLeaf property of the last character as true.

In the find() method we will be finding whether a word is present in the tree or not and returns a Boolean value. The method will start from the first character and iterates till it reaches the last character and checks whether, at a particular character key, the node value becomes null or not. At last, using the isLeaf variable the find() method will confirm whether the word is present (if isLeaf is true) or not in the trie.

In the delete() method, we will delete a word from the trie after checking whether the word is present in the trie using the find() method. If the word is present then the word is converted into a character array, and using for loop we will iterate through each character until the last character is reached. Inside the for a loop, a TrieNode will store the node which has more than one child, and another char data variable stores the next character. Now when the last character node doesn't have any more child nodes, then the node is simply removed, or else the nodes' isLeaf property is marked as false.

In the search() method, a prefix word is given as an argument, and with the prefix word the for loop iterates to the last character of the word if there are no null nodes. After reaching the last node, the searcher() method is called, which returns a list of all the child of the prefix word only when the isLeaf value of the last node of the child is true.

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

class TrieNode
{
	Map<Character, TrieNode> children;
	char c;
	boolean isLeaf;

	TrieNode(char c)
	{
		this.c = c;
		children = new HashMap<>();
	}
}

public class Trie
{

	TrieNode root;
```

```
Trie(List<String> words)           //constructor
{
      root = new TrieNode('\0');
      for (String word : words)
            insert(word);
}

protected void insert(String word)
{
      if (word == null || word.isEmpty())
            return;

      TrieNode parent = root;
      for (char w : word.toCharArray())
      {
            TrieNode child = parent.children.get(w);

            if (child == null)
            {
                  child = new TrieNode(w);
                  parent.children.put(w, child);
            }
            parent = child;
      }

      parent.isLeaf = true;
}

public boolean find(String prefix)
{
   TrieNode lastNode = root;
   for (char c : prefix.toCharArray())
   {
       lastNode = lastNode.children.get(c);
       if (lastNode == null)
           return false;
   }
   return lastNode.isLeaf;
}

protected boolean delete(String word)
{
      if (word == null || word.isEmpty())
```

```
            return false;

        if (!find(word))
            return false;

        TrieNode parent = root;
        TrieNode deleteBelow = null;
        char deleteChar = '\0';

        for (char w : word.toCharArray())
        {
            TrieNode child = parent.children.get(w);

            if (parent.children.size() > 1 || parent.isLeaf)
            {
               deleteBelow = parent;
               deleteChar = w;
            }
            parent = child;
        }

        if (parent.children.isEmpty())
            deleteBelow.children.remove(deleteChar);

        else
            parent.isLeaf = false;

        return true;
    }

    public List<String> search(String prefix)
    {
       List<String> list = new ArrayList<>();
       TrieNode rootNode = root;
       StringBuffer current = new StringBuffer();
       for (char c : prefix.toCharArray())
       {
           rootNode = rootNode.children.get(c);
           if (rootNode == null)
               return list;
           current.append(c);
       }
       searcher(rootNode, list, current);
```

```
        return list;
    }

    public void searcher(TrieNode root, List<String> list,
    StringBuffer current)
    {
        if (root.isLeaf)
        {
            list.add(current.toString());
        }

        if (root.children == null || root.children.isEmpty())
            return;

        for (TrieNode child : root.children.values())
        {
            searcher(child, list, current.append(child.c));
            current.setLength(current.length() - 1);
        }
    }
}
```

2.3 Wiki Class

There are 3 methods in this class, namely, getTerms(), getInfo() and moreInfo(). In this method named getTerms(), with the help of Jsoup.connect() we connect to the Wikipedia page and with the help of getElementByTag() method, we get all the elements with the element name given as an argument. Next, we create an empty list named terms to which we add the content in the lower-case format and we return the list. The same operation is done by getInfo() and moreInfo() method, but these methods return the information on the individual terms as a string.

One of the methods used in our class is getElementsByTag(). The function is used to return all elements of the given tag name in the order they are present in

the document. It is a built-in function. This is a function of DOM element or document object, so it is called as document.getElementsByTagName().

It takes in the Tag name parameter, which is a string parameter that specifies the tag name of an element that we want to get. This parameter is required, it is not optional. The return value of this function is an array of collections of all HTML elements which match the specified tag name. The returned array stored elements in the sorted order if they are present in the document.

When the Gui is started, then the getTerms() method gets all the data structure related terms from Wikipedia URL and inserts into the trie.

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.select.Elements;

class wiki
{

      List<String> getTerms() throws IOException
     {
         Document list =
         Jsoup.connect("https://en.wikipedia.org/wiki/List_of_terms_
         relating_to_algorithms_and_data_structures").get();

        Elements content = list.getElementsByTag("li");
        List<String> terms = new ArrayList<String>();
        for (int i=28; i<content.size()-55; i++)
           terms.add(content.get(i).text().toLowerCase());
        return terms;
      }

    String getInfo(String text) throws IOException
    {
        Document doc =
        Jsoup.connect("https://en.wikipedia.org/wiki/"+text).get();

        Elements content = doc.getElementsByTag("p");
        String info = content.get(0).text();
        int i = 1;
```

```
        while(info.equals("") && i< 5)
        {
           info = content.get(i).text();
           i++;
        }
        return info;
    }

    String moreInfo(String text) throws IOException
    {
       Document doc =
       Jsoup.connect("https://en.wikipedia.org/wiki/"+text).get();

       Elements content = doc.getElementsByClass("mw-parser-
       output");
       String info = content.text();

       return info;
    }
}
```

2.4 Gui Class

a) JTextarea - JTextArea is a Java Swing component. It symbolizes a text-display area with many lines. It's used to make changes to the text. JTextArea is a descendant of JComponent. JTextArea's text can be changed to any of the available fonts and attached to new text. A text area can be tailored to the user's requirements.

b) JTextField - The javax.swing package includes JTextField. The JTextField class is a component that lets us change a single line of text. JTextField derives from JTextComponent and implements the SwingConstants interface. JTextField contains a method for determining the string that will be used as the command string for the action event.

c) JButton - The JButton class is used to construct a labeled button that may be implemented on any platform. When the button is pressed, the application

performs some action. It is a descendant of the AbstractButton class. There are four JButtons in our implementation namely, Go, Insert, Delete, and More info.

d) JScrollPane - A JscrollPane is used to make a component's view scrollable. We utilize a scroll pane to display a huge component or a component whose size can change dynamically when the screen size is constrained.

e) JTextPanel - A text component that can be marked up with attributes that are represented graphically.

```
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.util.List;
import java.awt.event.ActionEvent;
import java.awt.Font;
import java.awt.Insets;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.io.IOException;
import javax.swing.SwingConstants;
import javax.swing.JTextPane;

public class Gui
{

	wiki obj = new wiki();
	Trie trie;
	void init()
	{
		try
		{
			List<String> words = obj.getTerms();
			trie = new Trie(words);
		}

		catch (IOException e)
		{
			e.printStackTrace();
		}
	}
```

```
private JFrame frame;
private JTextField textField;
List<String> suggestions;
private JScrollPane scrollPane;
private JTextArea textArea;
private JScrollPane scrollPane_2;
private JTextPane textPane;
private JButton goButton;
private JButton deleteButton;
private JButton insertButton;
private JButton infoButton;

 /**
  * Launch the application.
  */
public static void main(String[] args)
{
	EventQueue.invokeLater(new Runnable()
	{
		public void run()
		{
			try
			{
				Gui window = new Gui();
				window.frame.setVisible(true);
			}
			catch (Exception e)
			{
				e.printStackTrace();
			}
		}
	});
 }

 /**
  * Create the application.
  */
 public Gui()
 {
	initialize();
	init();
 }

 /**
  * Initialize the contents of the frame.
  */
 private void initialize()
 {
```

```
        frame = new JFrame("Auto Complete Feature");
        frame.setBounds(100, 100, 762, 630);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(null);

        textField = new JTextField();
        textField.setFont(new Font("Open Sans", Font.PLAIN, 14));
        textField.setHorizontalAlignment(SwingConstants.LEFT);
        textField.addKeyListener(new KeyAdapter()
       {
          public void keyReleased(KeyEvent e)
         {
            String text = textField.getText().toLowerCase();
            textArea.setText("");
            if (!text.equals(""))
            {
                suggestions = trie.search(text);
                for (String s : suggestions)
                      textArea.setText(textArea.getText()+s+"\n");
            }
         }
       });

       textField.setBounds(108, 40, 414, 36);
       frame.getContentPane().add(textField);
       textField.setColumns(10);

       scrollPane = new JScrollPane();

        scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_
SCROLLBAR_AS_NEEDED);

        scrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZON
TAL_SCROLLBAR_AS_NEEDED);
        scrollPane.setBounds(108, 94, 527, 121);
        frame.getContentPane().add(scrollPane);

        textArea = new JTextArea();
        textArea.setEditable(false);
        textArea.setFont(new Font("Open Sans Semibold", Font.PLAIN,
14));
        textArea.setMargin(new Insets(10,10,10,10));
        scrollPane.setViewportView(textArea);

        goButton = new JButton("Go");
        goButton.setFont(new Font("Open Sans", Font.BOLD, 14));
        goButton.addActionListener(new ActionListener()
       {
```

```
            public void actionPerformed(ActionEvent e)
            {
                String text = textField.getText();
                if (text.isBlank())
                    return;
                try
                {
                    textPane.setText(obj.getInfo(text));
                }
                catch (IOException e1)
                {
                    textPane.setText("Page not found (404)");
                }
            }
        });

        goButton.setBounds(561, 40, 74, 36);
        frame.getContentPane().add(goButton);

        scrollPane_2 = new JScrollPane();

        scrollPane_2.setVerticalScrollBarPolicy(JScrollPane.VERTICA
L_SCROLLBAR_AS_NEEDED);

        scrollPane_2.setHorizontalScrollBarPolicy(JScrollPane.HORIZ
ONTAL_SCROLLBAR_AS_NEEDED);
        scrollPane_2.setBounds(23, 265, 691, 297);
        frame.getContentPane().add(scrollPane_2);

        textPane = new JTextPane();
        textPane.setMargin(new Insets(10,10,10,10));
        scrollPane_2.setViewportView(textPane);

        deleteButton = new JButton("Delete");
        deleteButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                String word = textField.getText();
                if (word.isBlank())
                    return;
                if (trie.delete(word.toLowerCase()))
                    textPane.setText(word+" deleted successfully");
                else
                    textPane.setText("Not deleted or not found");
            }
        });

        deleteButton.setBounds(312, 226, 105, 28);
```

```
            frame.getContentPane().add(deleteButton);

            insertButton = new JButton("Insert");
            insertButton.addActionListener(new ActionListener()
           {
                public void actionPerformed(ActionEvent e)
                {
                     String word = textField.getText();
                     trie.insert(word.toLowerCase());
                     if (!word.isBlank())
                          textPane.setText(word+" added successfully");
                }
            });

            insertButton.setBounds(171, 226, 96, 28);
            frame.getContentPane().add(insertButton);

            infoButton = new JButton("More info");
            infoButton.addActionListener(new ActionListener()
            {
                 public void actionPerformed(ActionEvent e)
                 {
                     String text = textField.getText();
                     if (text.isBlank())
                          return;
                     try
                     {
                          textPane.setText(obj.moreInfo(text));
                     }
                     catch (IOException e1)
                     {
                          textPane.setText("Page not found (404)");
                     }
                 }
            });

             infoButton.setBounds(465, 226, 96, 28);
            frame.getContentPane().add(infoButton);
        }
    }
```

3. Demonstration

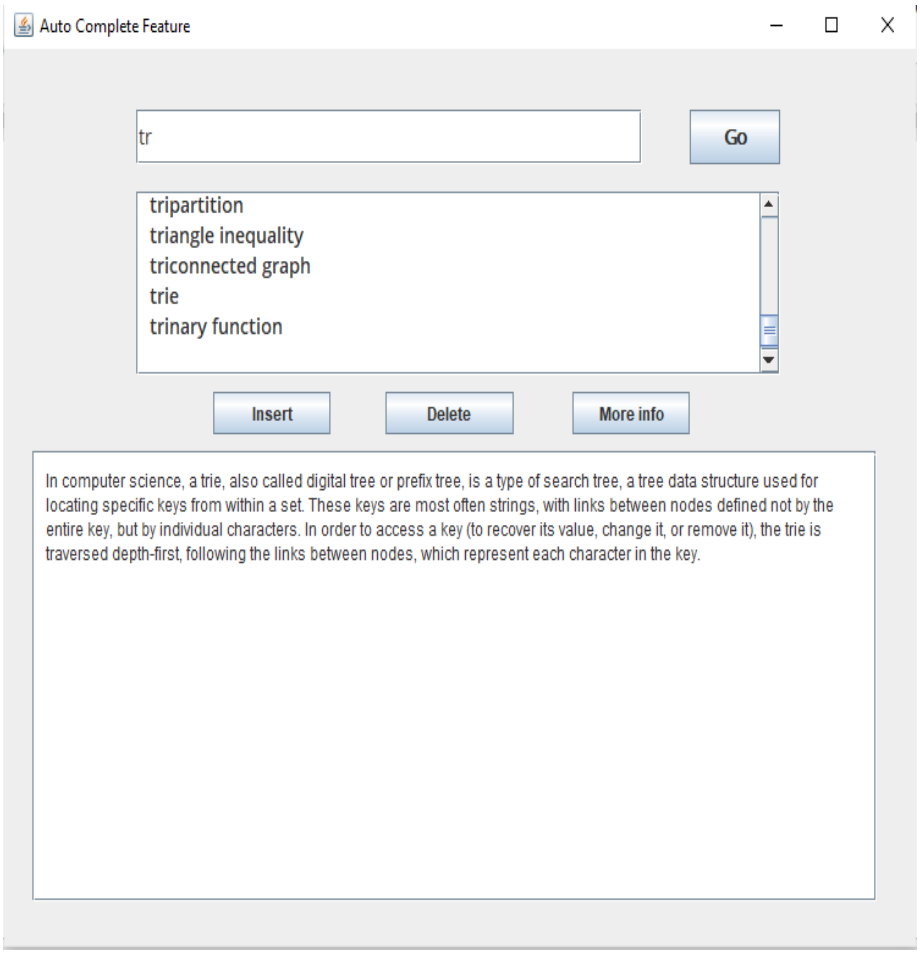

Fig.3.1 GUI Implementation

In figure 3.1. we can see that when the users enter a prefix of a complete word, then our program gives few suggestions, and also when the user clicks the Go button the definition of that term is displayed to the user. There are also options for inserting and deleting terms. Also, the user can get more detailed information about the term when the more info button is clicked.

4. Conclusion

Data retrieval and processing are some of the most important topics for programmers. Many real-time applications are based on string processing like Search Engine results optimization, Sentimental Analysis, AutoComplete Feature, and Spell Checker. The data structure that is very important for string handling is the Trie data structure which is based on the prefix of string. Trie data structure is the fastest for auto-complete suggestions. Even in the worst case, it is O(n) times faster than the alternate imperfect hash table algorithm where n is the string length. Also, Trie data structure overcomes the problem of key collisions in imperfect hash tables that cause reduction inaccuracy. The AutoComplete feature is implemented in various software applications, like Web browsers, Code Editors, and Messaging apps.

We learned many new topics in the implementation of this project like Web scrapping with Jsoup, and Gui designing with swing framework. We get to know about the different applications of Trie data structure, its importance in real-life problems, and how it is more advantageous in doing search operations when compared to other data structures.

5. Learning Outcome

- ❖ We learned many new things especially we got a nice introduction to the Jsoup.
- ❖ We learned how to create a GUI with the help of the swing framework and the role of GUI in attracting the user towards our project.
- ❖ We get to know different applications of java Trie data structure, its importance in real-life problems.
- ❖ We understood how to relate terms with the help of map data structures.
- ❖ We understood the importance of object-oriented concepts and low-level data structure which includes LinkedList in implementing the high order data structure.

Bibliography

https://docs.oracle.com/javase/7/docs/api/javax/swing

https://www.hackerearth.com/practice/data-structures/advanced-data-structures/trie-keyword-tree/tutorial/

https://jsoup.org/cookbook/