CMP-5014Y Coursework 2 - Word Auto Completion with Tries

Student number: 100238596. Blackboard ID: pxh18ksu

Thursday 14th May, 2020 03:52

Contents

1 Part 1: Form a Dictionary and Word Frequency Count **2**
1.1 Fundamental Operation . . . 2
1.2 Run time complexity function . . . 2
1.3 Worst case scenario . . . 2

2 Part 2: Implement a Trie Data Structure **3**
2.1 Add method for adding a key to a trie . . . 3
2.2 Contains method to check if key passed in is a full word or a prefix . . . 3
2.3 Output by Breadth First Search Method . . . 3
2.4 Output by Depth First Search Method . . . 3
2.5 getSubTrie method to return a trie rooted at the prefix passed . . . 4
2.6 getAllWords method to return all the words in the trie . . . 4

3 Part 3: Word Auto Completion Application **5**
3.1 Auto Completion Method . . . 5
3.2 Auto Completion Results . . . 5

4 Code Listing **7**

1 Part 1: Form a Dictionary and Word Frequency Count

The method formDictionary takes in a List of strings and returns a TreeMap¡String, Integer¿ as an output. The string is the "word" from the list, and the value is the amount of times the word appears in the list

Algorithm 1 formDictionary algorithm

```
Require: List of String words
1: for String word in words do
2:     if treeMap contains word then
3:         Add word, key + 1 to treeMap
4:     else
5:         Add word, 1 to treeMap
   return treeMap
```

1.1 Fundamental Operation

The fundamental operation for the algorithm is the "if treeMap contains" comparison as it is run every time the for loop runs

1.2 Run time complexity function

$$\sum_{i=1}^{n} n \tag{1}$$

$$\tag{2}$$

1.3 Worst case scenario

The worst case is that the words in the list passed are all words not already present within the treeMap

2 Part 2: Implement a Trie Data Structure

2.1 Add method for adding a key to a trie

Algorithm 2 add algorithm

```
Require: String key
 1: TrieNode $currentNode \leftarrow root$
 2: for every char $c$ in $key$ do
 3:     if $currentNode$ != null then
 4:         $next \leftarrow currentNode$.getOffspring(c)
 5:         if next == null then
 6:             $next \leftarrow TrieNode.newNode(c)$
 7:             $currentNode.addNode(next)$
 8:         $currentNode \leftarrow next$
 9: if $currentNode$.isComplete() then return false
10: else
11:     $currentNode$.setComplete(true)
12:     return true
```

2.2 Contains method to check if key passed in is a full word or a prefix

Algorithm 3 contains algorithm

```
Require: String $key$
 1: for char $c$ in $key$ do do
 2:     if $currentNode$.getOffspring(c) == null then return false
 3:     else
 4:         $currentNode \leftarrow currentNode$.getOffspring(c)
    return $currentNode$.isComplete()
```

2.3 Output by Breadth First Search Method

Algorithm 4 outputBreadthFirstSearch algorithm

```
Require: No Input
 1: Queue $Nodes \leftarrow EmptyQueue$
 2: ArrayList $characterArrayList \leftarrow EmptyArrayList$
 3: $nodes$.add(root)
 4: while $nodes$.isEmpty() = false do
 5:     TrieNode $next \leftarrow nodes$.poll()
 6:     if $next$.getOffspring != null then
 7:         for each TrieNode node : $next$.getOffspring() do
 8:             if $node$ != null then
 9:                 $nodes$.add($node$)
10:         $characterArrayList$.add($next$.getCharacter())
11: String $output$
12: for char $c$ in $characterArrayList$ do
13:     $output \leftarrow output + c$
14: return output
```

2.4 Output by Depth First Search Method

Algorithm 5 outputDepthFirstSearch algorithm

```
Require: TrieNode $next$, StringBuilder $stringBuilder$
1: $next.visited \leftarrow true$
2: for $i \leftarrow 1$ to $next$.offspring.length-1 do
3:     if $next$.offspring[i] != null and !$next$.offspring.visited then
4:         outputDepthFirstSearch($next$.offspring[i], $stringBuilder$)
5: $stringBuilder$.append($next$.getCharacter())
6: return result
```

2.5 getSubTrie method to return a trie rooted at the prefix passed

Algorithm 6 getSubTrie algorithm

```
Require: String $prefix$
1: TrieNode $next \leftarrow root$
2: Trie $newTrie \leftarrow newTrie()$
3: for $i \leftarrow 1$ to $prefix$.charAt(i) do
4:     $index \leftarrow 1$
5:     if $next$ == null then return null
6:     if $next.getOffspring(index)! = null$ then
7:         $newTrie$.root $\leftarrow$ $next$.getOffspring(index)
8:     $next \leftarrow next$.offspring[index]
   return $newTrie$
```

2.6 getAllWords method to return all the words in the trie

Algorithm 7 getAllWords algorithm

```
Require: String $wordSoFar$, TrieNode $currentNode$, List of Strings $nodes$
1: for TrieNode $temp$ in $currentNode$.getOffspring() do
2:     if $temp$ != null then
3:         String $currentPrefix \leftarrow wordSoFar + temp$.getCharacter()
4:         getAllWords($currentPrefix$, $temp$, $nodes$);
5: if $currentNode$.isComplete() then
6:     $nodes$.add($wordSoFar$)
   return output
```

3 Part 3: Word Auto Completion Application

3.1 Auto Completion Method

Algorithm 8 AutoCompletion algorithm

```
1: List of Strings $dictWordsList \leftarrow readWordsFromCSV(file)$
2: List of Strings $LotrQueries \leftarrow readWordsFromCSV(file)$
3: TreeMap(String, Integer) $dictWordsMap \leftarrow formDictionary(dictWordsList)$
4: Trie $dictTrie \leftarrow dictWordsList$
5: NavigableMap(String, Integer) $navMap \leftarrow newTreemap$
6: for String $s$ in $LotrQueries$ do
7:     Trie $subTrie \leftarrow dictTrie$.getSubTrie(s)
8:     List of Strings $list \leftarrow temp.getAllWords()$
9:     for String $listString$ in $list$ do
10:        String $auto \leftarrow s.trim() + listString.trim()$
11:        for every $entry$ of Map(String, Integer) in $dictWordsMap$.entrySet() do
12:            if $auto$.equals($entry$.getKey() then
13:                $navMap$.put($entry$.getKey(), $entry$.getValue())
14:    LinkedHashMap(String, Float) $tempList \leftarrow sortByFrequency(navMap)$    ▷ Described below
15:    LinkedHashMap(String, Float) $finalList$.putAll($tempList$)
16:    $navMap$.clear()
17: $DictionaryFinder$.saveToFile($finalList$, "$lotrMatches.csv$")
```

Forms a dictionary and word count of words from the file passed
Constructs a trie using this dictionary
Loads prefixes from LotrQueries into a list
For each prefix:
Recover all possible words
Choose 3 most frequent and display to standard output
Write results to file

sortByFrequency is a function created to mainly sort the list of possible words by how frequently they appear (their value in the Map). It has a running tally of frequency each time it is run, so it is possible to divide each key's value by that total to work out the probability. This uses the Java List built in sort and comparator.

Once the map has been sorted, it fetches the front 3 (most populous), adds them to a LinkedHashMap¡String, Float¿, and returns it to the main method.

3.2 Auto Completion Results

Listing 1: lotrMatches.csv

```
about=0.56666666
above=0.3
able=0.1
going=0.2777778
go=0.24074075
good=0.16666667
the=0.626703
they=0.15395096
them=0.06811989
merry=0.94736844
merrily=0.02631579
merely=0.02631579
frodo=0.4909091
from=0.43636364
front=0.07272727
great=0.1969697
ground=0.18181819
grass=0.15151516
```

```
goldberry=0.6
golden=0.4
sam=1.0
```

4 Code Listing

Listing 2: DictionaryFinder.java

```java

package com.company;

import java.io.*;
import java.util.*;

/**
 *
 * @author ajb
 */
public class DictionaryFinder {

    //String is the word, Int is the number of occurrences
    TreeMap<String, Integer> dict;

    public DictionaryFinder(){
    }
    /**
     * Reads all the words in a comma separated text document into an Array
     * @param file
     */
    public static ArrayList<String> readWordsFromCSV(String file) throws FileNotFoundException {
        Scanner sc=new Scanner(new File(file));
        sc.useDelimiter(" |,");
        ArrayList<String> words=new ArrayList<>();
        String str;
        while(sc.hasNext()){
            str=sc.next();
            str=str.trim();
            str=str.toLowerCase();
            words.add(str);
        }
        return words;
    }

    public static ArrayList<String> readWordsFromCSVNewLine(String file) throws FileNotFoundException {
        Scanner sc=new Scanner(new File(file));
        sc.useDelimiter("\n");
        ArrayList<String> words=new ArrayList<>();
        String str;
        while(sc.hasNext()){
            str=sc.next();
            str=str.trim();
            str=str.toLowerCase();
            words.add(str);
        }
        return words;
    }
    public static void saveCollectionToFile(Collection<?> c,String file) throws IOException {
        FileWriter fileWriter = new FileWriter(file);
        PrintWriter printWriter = new PrintWriter(fileWriter);
        for(Object w: c){
            printWriter.println(w.toString());
        }
        printWriter.close();
    }
    public TreeMap<String, Integer> formDictionary(List<String> words) throws
       ↪ FileNotFoundException {
        Collections.sort(words);
        dict = new TreeMap<String, Integer>();
        for(String word : words){
            //If word already exists, increment counter
            if(dict.containsKey(word)){
                dict.put(word, dict.get(word) + 1);
            }
            //Else, add word and set occurrences to 1
            else{
                dict.put(word, 1);
            }
        }
        return dict;
    }

public static void saveToFile(LinkedHashMap map,String file) throws IOException {
    FileWriter fileWriter = new FileWriter(file);
    PrintWriter printWriter = new PrintWriter(fileWriter);
    for(Object w: map.entrySet()){
        printWriter.println(w);
    }
    printWriter.close();
}

    public static void main(String[] args) throws Exception {
        DictionaryFinder df=new DictionaryFinder();
        //ArrayList<String> in=readWordsFromCSV("C:\\Teaching\\2017-2018\\Data
           ↪ Structures and Algorithms\\Coursework 2\\test.txt");
        //DO STUFF TO df HERE in countFrequencies
        //df.formDictionary(in);
        //df.saveToFile();

        Trie test = new Trie();
        System.out.println("Testing adding words to the trie - all should be T apart
           ↪ from the last");
        System.out.println(test.add("cheers"));
        System.out.println(test.add("cheese"));
        System.out.println(test.add("chat"));
        System.out.println(test.add("cat"));
        System.out.println(test.add("bat"));
        System.out.println(test.add("bat"));
        System.out.println("\n");
        System.out.println("Testing the contains method - should produce F,F,F,T,T");
        System.out.println(test.contains("chee"));
        System.out.println(test.contains("afc"));
        System.out.println(test.contains("ba"));
        System.out.println(test.contains("cheese"));
        System.out.println(test.contains("bat"));
        System.out.println("\n");
        System.out.println("Testing the breadth first search method - should produce
           ↪ bcaahttaetersse");
        System.out.println(test.outputBreadthFirstSearch());
        System.out.println("\n");
        System.out.println("Testing the depth first search method - should produce
           ↪ batcathateersse");
        System.out.println(test.outputDepthFirstSearch());
        System.out.println("\n");
        Trie subtrie = test.getSubTrie("ch");
        System.out.println("Testing the subTrie method via breadth search - should
           ↪ produce haetersse");
        System.out.println(subtrie.outputBreadthFirstSearch());
        System.out.println("\n");
        System.out.println("Testing getAllWords - should produce bat, cat, chat,
           ↪ cheers, cheese");
        System.out.println(test.getAllWords());

        System.out.println("\n");
        System.out.println("Now running AutoCompletion");
        AutoCompletion.main();

    }

}
```

Listing 3: TrieNode.java

```java
package com.company;

public class TrieNode {

    private char character;
    TrieNode[] offspring = new TrieNode[26];
    private boolean isComplete;
    Boolean visited;

    public TrieNode(){
        this.offspring = new TrieNode[26];
        visited = false;
    }

    public TrieNode(char character){
        this.character = character;
        this.offspring = new TrieNode[26];
        for(int i  = 0; i < offspring.length; i++){
            offspring[i] = null;
        }
        visited = false;
    }


    public boolean isComplete() {
        return this.isComplete;
    }

    public char getCharacter() {
        return character;
    }

    public TrieNode getNode(char c) {
        return this.offspring[getCharIndex(c)];
    }

    public void setCharacter(char character) {
        this.character = character;
    }

    public void setOffspring(TrieNode[] offspring) {
        this.offspring = offspring;
    }

    public static int getCharIndex(char c) {
        return c - 'a';
    }

    public static TrieNode newNode(char c){
        TrieNode newNode = new TrieNode();
        newNode.isComplete = false;
        newNode.character = c;
        for (int i = 0; i < newNode.offspring.length; i++){
            newNode.offspring[i] = null;
        }
        return newNode;
    }
    public void addNode(TrieNode next){
        int index = (int)next.character-97;
        offspring[index] = next;
    }

    public void setComplete(boolean complete) {
        isComplete = complete;
    }

    public TrieNode[] getOffspring(){
        return this.offspring;
    }

    public TrieNode getOffspring(char c){
        for(int i = 0; i < offspring.length; i++){
            if (offspring[i] != null && offspring[i].character == c){
                return offspring[i];
            }
        }
        return null;
    }

}
```

Listing 4: Trie.java

```java
package com.company;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

public class Trie extends TrieNode {

    private TrieNode root;

    public Trie() {
        root = new TrieNode();
    }


    /* let current node = root node
    for each letter in the key
        find the child node of current node associated with that letter
        if there is no child node associated with that letter, create a new node and
           ↪ add it to current node as a child associated with the letter
        set current node = child node
    add value to current node */
    public boolean add(String key) {
        TrieNode currentNode = root;
        for (int i = 0; i < key.length(); i++) {
            char currentChar = key.charAt(i);
            if (currentNode != null) {
                TrieNode next = currentNode.getOffspring(currentChar);
                if (next == null) {
                    next = TrieNode.newNode(key.charAt(i));
                    currentNode.addNode(next);
                }
                currentNode = next;
            }
        }

        // returns false if key is already in the trie
        if (currentNode.isComplete()) {
            return false;
        }

        currentNode.setComplete(true);
        return true;
    }


    public boolean contains(String key) {
        TrieNode currentNode = root;
        for (int i = 0; i < key.length(); i++) {
            char currentChar = key.charAt(i);
            TrieNode next = currentNode.getOffspring(currentChar);
            if (next == null) {
                return false;
            } else {
                currentNode = next;
            }
        }
        return currentNode.isComplete();
    }

    public String outputBreadthFirstSearch() {
        //Breadth first done using a queue
        Queue<TrieNode> nodes = new LinkedList();
        ArrayList<Character> characterArrayList = new ArrayList();
        nodes.add(root); //add root node to the queue
        while (!nodes.isEmpty()) { //while queue isn't empty
            TrieNode next = nodes.poll(); //set next node to item front of queue

            if (next.getOffspring() != null) {
                for (TrieNode node : next.getOffspring()) {
                    //if node is not null
                    if (node != null) {
                        //add a node to nodes linkedlist
                        nodes.add(node);

                    }
                }
                characterArrayList.add(next.getCharacter()); //add character value of
                    ↪ node to an arraylist
            }

        }

        //Adding the arraylist to a string
        String outputBreadth = "";
        for (char character : characterArrayList) {
            outputBreadth = outputBreadth + character;
        }
        return outputBreadth;
    }


    public String outputDepthFirstSearch(){
        StringBuilder builder = new StringBuilder();
        outputDepthFirstSearch(root, builder);
        return builder.toString();
    }

    private static void outputDepthFirstSearch(TrieNode next, StringBuilder str){
        next.visited = true;
        for(int i = 0; i < next.offspring.length-1; i++){
            if(next.offspring[i] != null && !next.offspring[i].visited){
                outputDepthFirstSearch(next.offspring[i],str);
            }
        }
        str.append(next.getCharacter());
    }

    public Trie getSubTrie(String prefix){
        TrieNode next = root;
        Trie newTrie = new Trie();
        for(int i = 0; i < prefix.length(); i++){
            int index = (int)prefix.charAt(i)-97;
            if(next == null){
                return null;
            }
            if(next.getOffspring(prefix.charAt(i)) != null){
                newTrie.root = next.getOffspring(prefix.charAt(i));
            }
            next = next.offspring[index];
        }
        return newTrie;
    }

    public List<String> getAllWords() {
        List<String> output = new LinkedList<>();
        getAllWords("", root, output);
        return output;
    }

    private void getAllWords(String wordSoFar, TrieNode currentNode,
                             List<String> nodes) {
        for (TrieNode temp : currentNode.getOffspring()) {
            if (temp != null) {
                String currentPrefix = wordSoFar + temp.getCharacter();
                getAllWords(currentPrefix, temp, nodes);
            }
        }

        if (currentNode.isComplete()) {
            nodes.add(wordSoFar);
        }
    }

}
```

Listing 5: AutoCompletion.java

```java
package com.company;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.*;

import static com.company.DictionaryFinder.readWordsFromCSV;
import static com.company.DictionaryFinder.readWordsFromCSVNewLine;

public class AutoCompletion {

    public static void main() throws FileNotFoundException,
            IOException {

        DictionaryFinder df = new DictionaryFinder();

        NavigableMap<String, Integer> storeStringMap
                = new TreeMap<String, Integer>();

        LinkedHashMap<String, Float> finalList = new LinkedHashMap<>();

        String auto;

        //list of words from dictionary file
        List<String> dictWordsList = readWordsFromCSV("lotr.csv");
        TreeMap<String, Integer> dictWordsMap = df.formDictionary(dictWordsList);

        // dictionary trie
        Trie dictTrie = new Trie();
        for (String s : dictWordsList){
            dictTrie.add(s);
        }


        // list of words from query file
        List<String> LotrQueries = readWordsFromCSVNewLine("lotrQueries.csv");


        //for each prefix in query file
        for(String s : LotrQueries){
            //get a subtrie rooted at the prefix
            Trie temp = dictTrie.getSubTrie(s);
            //create a list of words from the new trie - will be missing the prefix
               ↪ though as new root: e.g. eese
            List<String> list = temp.getAllWords();
            //re-add the prefix onto the list: e.g. ch + eese = cheese
            for (String listString : list){
                auto = s.trim() + listString.trim();

                for (Map.Entry<String, Integer> entry : dictWordsMap.entrySet()) {
                    //if words that are in auto equal to the words in entry map
                        if (auto.equals(entry.getKey())) {
                            //store those words in a storeAuto map
                            storeStringMap.put(entry.getKey(), entry.getValue());
                        }
                }

            }

            LinkedHashMap<String, Float> tempList  = sortByFrequency(storeStringMap);
            finalList.putAll(tempList);
            storeStringMap.clear();
        }
        DictionaryFinder.saveToFile(finalList, "lotrMatches.csv");
    }


    public static LinkedHashMap<String, Float> sortByFrequency(NavigableMap<String,
      ↪ Integer> dictionary) throws IOException {
        List<Map.Entry<String, Integer>> dictList = new
          ↪ LinkedList<>(dictionary.entrySet());
        LinkedHashMap<String, Float> map = new LinkedHashMap<>();

        dictList.sort(Comparator.comparingInt(Map.Entry::getValue));
        Collections.reverse(dictList);

        float totalFreq = 0;
        for(Map.Entry<String, Integer> item : dictList){
            totalFreq = totalFreq + item.getValue();
        }

        for(int i=0; i < 3; i++){
            try{
                float probability = (float)dictList.get(i).getValue() / totalFreq;
                map.put(dictList.get(i).getKey(), probability);
                System.out.println(dictList.get(i).getKey() + " (probability " +
                  ↪ probability + ")");
            }
            catch (Exception ignored){
            }

        }

        return map;
    }


}
```