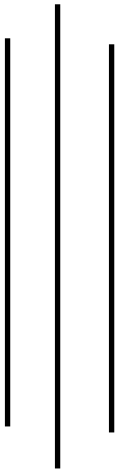
LA TROBE UNIVERSITY

Melbourne Campus

Bundoora 3086, VIC, Australia



ASSIGNMENT REPORT ON

Algorithm sand Data Structures

BY

Ishwari Prasad Khanal
ID:17488253

Submitted to La Trobe University In Partial Fulfillment of requirements
For Master Degree of Information and Communication Technology (ICT)

Submitted To

DR. Kinh Nguyen
Department of Computer Science and Engineering
La Trobe University
Melbourne VIC 3086 Australia

Contents

Introduction

Dozens of different data structures have been proposed for implementing cross word matching, phone directory, dictionaries and so on including hash tables, skip lists, and balanced/unbalanced binary search trees such as AVL tree, BST, etc. It is therefore choosing the right one can be tricky. Depending on the application, it is also a decision that can significantly impact performance. In practice, it is more important to a avoid using a bad data structure than to identify the single best option available.

Background

As we know, Tries are a multiway tree that can be used for fast text retrieval. It is an ordered tree data structure that uses strings as keys. Unlike Binary Trees, Tries do not store keys associated with the node on the tree. Any descendants of a node shares a common prefix of the key string associated with the node. Hence, trie is also called as Prefix Tree. The word 'Trie' comes from Retrieval, and it is pronounced as try.

Since this data structure is a prefix tree, trie is commonly used in Cross Word Matching, Dictionary, Phone Directories and other matching algorithms.

So I have decided to implement Trie to solve this problem as a part of the given assignment solution and created the following four classes and interface:

TrieNode : Represents a single tree node and the children of a node.
Trie: Represents the interface to insert and search nodes.
WordMatch: Represents Trie implementation .
MergeSort: Represents sorting lexicons alphabetically

Implementation

TrieNode: TrieNode represents a basic tree node which implements the methods created in interface namely Trie. It also contains its Parent node and children node. TrieNode has a key and a value. Key contains the single character and the value has the actual char value of the node. The actual key of the node will be determined by suffixing the single character to its parent's key. TrieNode has a special property called isFinalChar. This property is set to true if the kye or a value represents a complete string. This class contains an internal Set class of type TrieNode which implements the standard set methods such as containsKey(), put(), etc.

WordMatch: This class implements the Trie algorithms such as addWords and searchWords method.

```
    // search words for the input match calling recursively

    public Set<String> searchWords(String matchingString) {
        if (!isValidString(matchingString)) {
            return null;
        }
        matchingString = matchingString.toLowerCase();
        // recursive calling
        Set<String> finalSet = searchWords(trieRoot, matchingString.toCharArray(),
0, null);
        return finalSet;
    }
```

```
// to add the words if not existed in the trie otehrwise fasle
    public boolean addWords(String word) {
        if (word == null || word.isEmpty()) {
            return false;
        }
        char[] wordCharArr = word.toCharArray();
        TrieNode tempRoot = trieRoot;
        for (char charValue : wordCharArr) {
            tempRoot.addChild(charValue);
            tempRoot = tempRoot.getChild(charValue);
        }
        tempRoot.setFinalChar(true);
        return true;
    }
```

The addWords method inserts the string as one character at a time. It starts with the first character; if the first character does not already exist in the root node it adds a new node with the new character and returns true otherwise returns false. It loops until it adds the entire string. Once it reaches the last character, it marks that node as a finalChar because this node represents a complete string in the tree hierarchy.

The searchWords method is implemented by recursive function. The tree is searched until the complete string is found. Below is the part of code.

```
protected Set<String> searchWords(TrieNode curNode, char[] wordArray, int curIndex, StringBuffer wordFormed) {
        if (curNode == null || wordArray == null || curIndex < 0) {
            return null;
        }

        if (curNode.isFinalChar() && curIndex >= wordArray.length && wordFormed != null && wordFormed.length() != 0) {
            return Collections.singleton(wordFormed.toString());
        }
        if (curIndex >= wordArray.length) {
            return null;
        }
        if (wordFormed == null) {
            wordFormed = new StringBuffer();
        }
        Set<String> wordSet = new HashSet<String>();
        char curChar = wordArray[curIndex];
        if (curChar == '?') {
            Set<TrieNode> childrenNodes = curNode.getChildrenNodes();
            if (childrenNodes != null) {
                for (TrieNode node : childrenNodes) {
                    StringBuffer wordFormedClone = new StringBuffer(wordFormed);
                    wordSet = addToSetIfNotNull(wordSet,
                            searchWords(node, wordArray, curIndex + 1, wordFormedClone.append(node.getCharValue())));

                }
                return wordSet;
            }
        } else {
            assert Character.isLetter(curChar);
            TrieNode node = curNode.getChild(curChar);
            if (node != null) {
                wordSet = addToSetIfNotNull(wordSet,
```

```
                        searchWords(node,     wordArray,     curIndex     +     1,
wordFormed.append(node.getCharValue()));
                return wordSet;
            }
        }
        return null;
    }
```

MergeSort: This class represents sorting the given list based on MergeSort algorithm i.e. split the list into two sub-lists, sort them and merge them back.

- start with one list;
- break it into two lists;
- sort each list;
- merge the sorted lists;

The merge sort has the time complexity of(N Log N) for the best case, worst case, and average case. This sorting is faster than all sorting algorithms.

How the program flows?

The program drives the user in menu option after executing the file(java WordMatch.java):

Enter the character 'r', 's', 'w' or 'q'
---
r) Read in a text file and add the words in it to the lexicon
s) Search for a word
w) Write the lexican to a new file
q) Quit

Option r: User has to input the any text file, for example , fiction.txt and it automatically redirects to the main menu. And then either user can select same option ‘r’ to read another file or go to another option if needed.

Option s: User has to input the valid input i.e. only alphabetic string without any space but can use ‘?’ as a wild card. If the given input pattern matches from the set of lexicons, the output will be displayed on the screen, which is sorted if more than one words displayed.

Option w: User has to supply the valid text file name to save all the lexicons to that file.

Option q: User can quit the program by pressing q from keyboard and following the prompts shown in the screen.

Testing

Testing has been performed for the correctness and efficiency purposes.

1. Correctness Testing: Correctness was thoroughly tested by addressing the problems such as reading any text file which stores only alphabetic words in the lexicon. For example, if the word ‘don’t’ or ‘first-time’ appeared in the lines, they would be stored without punctuation ‘dont’ and ‘firsttime’. Similarly

numeric words such as 100 , 345 etc. are discarded and duplicate words are removed too. In the case of searching correctness, if the input pattern matched , the word(s) that are found would be displayed in sorted form. When input string consists wild-card character '?', it represents any single character of string.

The various input and output formats has been tabulated as follows:

S.N	Description	Input	Expected Output	Actual Output
1	Discarding redundant of words	to contribute to state schools through taxes.	contribute schools state taxes through To	contribute schools state taxes through to
2	Discarding case sensitive	ebook is the START OF THIS GUTENBERG EBOOK METAMORPHOSIS	ebook gutenberg is metamorphosis of project start the this	ebook gutenberg is metamorphosis of project start the this
3.	Discarding numeric words	Copyright (C) 2002 David Wyllie.	c copyright david wyllie	c copyright david wyllie
4	Discarding punctuation	Dr. Gregory B. Newby Chief Executive and Director gbnewby@pglaf.org	and b chief director dr executive gbnewbypglaforg gregory	and b chief director dr executive gbnewbypglaforg gregory
5.	Searching full string	executive	Executive	Executive
6.	Searching four characters-long words starting from character 'c'	c???	chief	chief
7.	Searching three characters-long words having middle character 'o'	?o?	for nor not	for nor not

2. Efficiency Testing: Efficiency of programs had been measured with respect to time how quickly the data were read from input text files and displayed on the screen based on given input pattern while searching. Similarly, the efficiency of sorting algorithm was also measured in terms of big O notation in best, worst and average case.

The following table depicts the test cases and their expected output.

S.N	Test Case Description	Input	Time
1	Reading text file	http://www.freebookbooks.com/books/pg76.txt	10 secs
2	Reading text file	http://www.freebookbooks.com/books/pg5200.txt	1 mins 10 secs
3	Reading text file	http://www.freebookbooks.com/books/pg2600.txt	45 secs
4	Searching full string word	Gutenberg	0 sec
5	Searching pattern matched words	?o?	3 sec
6	Writing the sorted words to output file	OutputFile.txt	10 secs

Conclusion

In conclusion, As this program uses the Trie data structure and MergeSort algorithm, the overall performance of program has been measured based on correctness and efficiency which is highly satisfactory as shown in the test reports above. it is more important to avoid bad data structure than finding the single best solution.

References

http://stackoverflow.com/questions/674844/trie-implementation-question/674980#674980

http://en.wikipedia.org/wiki/Trie#As_a_replacement_for_other_data_structures

http://www.toptal.com/java/the-trie-a-neglected-data-structure

Lecture Note

Lab Solutions

-----THE END-----