Report of Program Finding Anagrams

Julia Sheng

02/28/2018

Section 1: Solution Overview

Section 1.1: Algorithm

Read data from dictionary file

Create a new Anagram class store data

Key🡨 processed data

If key exists

Add that word to the corresponding value

Else

Create the key and store word into the value

Scan target word

Input🡨processed word

Get value where key is equal to input

Eliminate the original word from value

If value is not empty

Print value

Else print no anagrams

Section 1.2: Block Diagram

Section 2: Complexity

Space complexity is O(n), n is the number of line(words) in the dictionary file, so Total Space=n(there will be no more than n keys)+n(there will n words)+1(input);

Time complexity is O(n), n is the number of line(words) in the dictionary file and m is the number of inputs, so Total Time=3\*n(scan, sort and format each words)+n(see whether key already exists and if not, store key, no more than n search)+n(store words)+3\*m(scan, sort and format input)+n\*m(search whether input is a key, no more than n search)+1(eliminate original word)+1(print);

|  |  |
| --- | --- |
| Space complexity | Time complexity |
| O(n) | O(n) |

Section 3: Data Structures

Hashtable: static Map<String,Anagrams> words =new HashMap<String,Anagrams>();

-store the formatted and sorted word as key, anagrams as value

-get(key): will return the value which is mapped by this key

-put(key,value): will add key and corresponding value to the hashtable

Class (Anagrams): Anagrams a = new Anagrams(String ans);

-contains only one parameter

-store value of parameter into ans(which represents all anagrams)

-addWords(Anagrams a): adds the ans of a to ans in this Anagrams, separated by “, ”

-getAns(): return ans

-toString(): return ans, therefore when print Anagrams a, it will print ans

Section 4: Sample Input and Output

|  |  |
| --- | --- |
| Input | Output |
| plekic | pickle |
| diapers | praised, despair, aspired |
| teardrop | prorated, predator, parroted |
| nameless | salesmen, maleness, lameness |
| allergy | regally, largely, gallery |
| deepak | peaked |
| impressions | permissions |
| restrain | trainers, terrains, strainer, retrains |
| calligraphy | graphically |
| nepal | plane, penal, panel |
| stale | least, letas, slate, stael, steal, tales, teals, tesla |
| parliaments | paternalism |
| sucrose | sources, crusoes, croesus, courses |
| persist | stripes, sprites, spriest, priests, esprits |
| disintegration | disorientating |

Section 5: Discussion

I choose Java to program this homework assignment because java is the only language that I know how to implement all structures. What’s more, we did a place and zip search program in data structure class using Java which shares a lot of similarities with this program. So that, I could look back to see how to implement the algorithms and data structures.

There are two parts of difficulties in this program, first is how to find the anagrams and second is how to store them so that we can call them whenever we want.

For the first difficulties, I find two very useful functions from oracle which are: toCharArray() and Arraysort(). The first one will separate the characters from a word and store them in a character array. Second function will help us sort the character array in alphabetical order. And then we can cast this character array back to a string using new String (char[]). After these three steps, all anagrams will become the same word, for example, “abc”, “acb”, “bac” and “cba” will all become “abc”. Then we can distinguish the anagrams by whether they produce the same word after these three steps. One thing needs to be careful here is that we should format the word first before these three steps of sorting, that is to take the punctuation and space out from the words (Julia’s becomes Julias) and make all character to lower case (Julias becomes julias), because format of the word does not influence whether it is an anagram. I use .toLowerCase() to set all character to lowercase and .replaceAll(“ ’ ”, “ ”) to eliminate the punctuation and space.

As for the second part, I used the ArrayList at the beginning. I simply stored each line of the dictionary file into an ArrayList. Each time, I took one element from the ArrayList and compare it with target word using the method in part one. And if they are anagrams, I would store the position of that element into another ArrayList called result. Thus after finishing the looping, result will contain the position of all anagrams in the ArrayList. However, after discussed with professor, I found this method is extremely inefficient since we need to sort all the words again once we have a new input which is a wasting of time. And I was reminded to use a structure, Hashtable, which we also used in the zip search program. We are also using class in Java, where the class name is Anagrams. Every time, we read a word from the dictionary file, I create a new class Anagrams and it has only one parameter ans which contains only the word right now. I set the formatted and sorted word to key, and value as this class Anagrams. If key does not exist then create a key and store corresponding value in hashtable. When key is already in the table, I just add the word of current Anagrams to the ans of key’s corresponding (Value) Anagrams. Therefore, in the hashtable, key is the formatted sorted word and in value are the anagrams of the key word, i.e. each key is map to a group of anagrams. Then every time we have a new input, we only need to check whether the sorted and formatted input are a key in the hashtable, if so, the mapped value will return all anagrams of input. I would say that hashtable is crucial to this program since it organizes the data in a readable and easy to use way and makes the whole search process efficient.

But there is another problem, that is the group of anagrams will contain the word itself but it is not correct to say the word has an anagram when anagram is only the word itself like Julia has anagrams: Julia. Thus we should eliminate the original word from the result. Since I separate each word by “, ” in ans in class Anagrams, I would replace all input+ “, ” (when input is not last in the group), “, ”+input(when input is the last in the group) and input(anything left) to null(“”). After that, we are only left the anagrams different to the original input word.

In Conclusion, the overall process will be using BufferedReader to read dictionary file first, and create a new FileReader when reading the directory. Then we use the method in the first part to find groups of anagrams. And store them into a hashtable called words as mentioned in part two, sorted and formatted words are key, values are in form of class Anagrams which returns a group of anagrams. If the formatted and sorted input word is equal to a key in hashtable, then we print the whole corresponding value except the original word itself. If it is not equal or value only contains the input word, then there are not anagrams for the input.

Section 6: Code

package lab5;

import java.io.\*;

import java.util.Arrays;

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class anagram {

static Map<String,Anagrams> words = new HashMap<String,Anagrams>();

public static void main(String[] args) {

try {

readData();

} catch (FileNotFoundException e) {

e.printStackTrace();

}

Scanner cw = new Scanner(System.in);

System.out.print("Looking for words:");

String input = cw.nextLine();

while (input != null) {

input = input.toLowerCase().replaceAll(" ' ", "").trim();

char[] target=input.toCharArray();

Arrays.sort(target);

String sinput=new String (target);

try {

String result=words.get(sinput).getAns();

String Result=result.replace(", "+input,"").replace(input+", " , "").replace(input,"");

if (!Result.trim().isEmpty())

System.out.print("The word has anagrams: "+Result);

else

System.out.print("The word has no anagram.");

} catch(NullPointerException e) {

System.out.print("The word has no anagrams.");

}

System.out.println("\nLooking for words:");

input = cw.nextLine();

} // while

cw.close();

}

public static void readData() throws FileNotFoundException {

BufferedReader dictionary;

try {

dictionary = new BufferedReader(new FileReader("/usr/share/dict/words"));

String line;

while ((line = dictionary.readLine()) != null) {

line= line.toLowerCase().replaceAll(" ' ", "").trim();

Anagrams a = new Anagrams(line);

char[] temp =line.toCharArray();

Arrays.sort(temp);

String key=new String(temp);

if(words.containsKey(key)) {

words.get(key).addWords(a);

}else {

words.put(key,a);

}

}

dictionary.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

package lab5;

public class Anagrams {

private String ans;

public Anagrams(String \_ans) {

ans = \_ans;

}//constructer

public String getAns() {

return ans;

}

public void addWords(Anagrams a) {

if(!ans.contains(a.ans))

ans= ans+", "+a.ans;

}

public String toString() {

return ans;

}

}// place()