Exercise 12: Scrabble Cheater Deluxe

Due on Monday, July 11, 2016

Schütz, Ulc, Krajewski, Fellmann

Contents

Exercise 1

Since we pretty much remained in our groups, there were not many options. We were satisfied with our solution from last week, so we took that as a base for this weeks project.

Exercise 2

We extracted a method from last weeks getWords method, that normalizes two words and compares them, checking if they are permutations. The reason we don't use this method in our code is, that it would have normalize the same word for every entry in the collision chain, when it checks for actual permutations. We could give the method the normalized word as parameter, but that would be kind of confusing.

Listing 1: method isPermutation

```java
private boolean isPermutation(String word1, String word2) {
          if (normalize(word1).equals(normalize(word2))) {
               return true;
          } else
               return false;
     }
```

Exercise 3

To generate seven random letters, we just generate 7 random numbers in between 0 and 26, add 97 to them and cast them to char. The result of that are seven random small letters from a-z. The method adds them all to a String and returns the String.

Listing 2: method randomSevenLetters

```java
private String randomSevenLetters() {
          Random rnd = new Random();
          String returnString = "";
          for (int i = 0; i <= 7; i++) {
               char c = (char) (rnd.nextInt(26) + 97);
               returnString += c;
          }
          return returnString;
     }
```

Exercise 4

This Exercise was a little bit more interesting. One problem we had to deal with was generating the distribution list, that contains how often one letter is available, in a reasonable amount of time. Another problem was, that one letter tile can only be taken once, so it had to be marked as taken somehow. Firstly, we implemented an array of 100 chars, that represents all available letter tiles. We then implemented an array of 26 integers, that contains how often a letter tile is available. The first entry for example is 9, since there are nine "a" letter tiles in the scrabble distribution. All we had to do then, was to iterate through the array of frequencies, and write the right amount of letters in the distribution table. To get seven random tiles out of that array, we used the same method as for the normal seven letters, but additionally we check if the tile was used already, by changing the entry of that tile to 64 and rolling a new number whenever the entry at the current index is 64.

Exercise 4 continued on next page. . .

Listing 3: method englishDistributionSevenLetters

```
private String englishDistributionSevenLetters() {
          char[] distributionTable = new char[100];
          int counter = 0;
          String outputString = "";
          int[] frequencies = { 9, 2, 2, 4, 12, 2, 3, 2, 9, 1, 1, 4, 2, 6, 8, 2,
              1, 6, 4, 6, 4, 2, 2, 1, 2, 1 }; //frequencies of the letters in
              alphabetical order
          for (int i = 0; i < frequencies.length; i++) {
               for (int j = 0; j < frequencies[i]; j++) {
                    distributionTable[counter] = (char) (i + 97);
                    counter++;
               }
          }
          Random rnd = new Random();
          for (int i = 0; i < 7; i++) {
               int randomIndex = rnd.nextInt(100);
               while (distributionTable[randomIndex] == 64) {
                    randomIndex = rnd.nextInt(100);
               }
               outputString += distributionTable[randomIndex];
               distributionTable[randomIndex] = 64;//set the entry to 64, to mark
                   the position as taken
          }
          return outputString;
     }
```

Exercise 5

To generate all the substrings of the word, we chose to write a recursive function, that uses a HashSet as data structure. We are using a HashSet because we don't wont to have a combination twice, which would be the case for example with the word java, which contains two "a", so our method would generate the combination "aj" twice. Our method is a variation of the standart recursive method, that generates all permutations of a string. Instead of going through the whole word before outputting the permutation, we changed it, so that it outputs every step on the way, by giving it the current index as a parameter and calling the function for every newly generated Prefix.

Listing 4: method getSubstrings

```
private void getSubstrings(String prefix, String string, HashSet<String>
    permutations, int currentIndex){
          for(int i = currentIndex + 1; i < string.length(); i++){
               String newPrefix = prefix + string.charAt(i);
               if(newPrefix.length() > 1){
                    permutations.add(newPrefix);
               }
               getSubstrings(newPrefix, string, permutations, i);
          }
     }
```

Exercise 6

This exercise was basically just putting it all together. We changed our run method, so that it first generates an ArrayList of all the substrings and then performs the procedure from last week for every of these substrings.

Source Code

Listing 5: class ScrabbleCheater

```java
package src;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.Random;

public class ScrabbleCheater {
    private final int hashTableSize = 9000;
    private LinkedList<String>[] hashTable;
    private int maxColisioncounter = 0;
    private int maxColisionposition = 0;

    @SuppressWarnings("unchecked")
    public ScrabbleCheater() {
        hashTable = new LinkedList[hashTableSize];
        for (int i = 0; i < hashTable.length; i++) {
            hashTable[i] = new LinkedList<String>();
        }
    }

    public static void main(String[] args) throws FileNotFoundException {
        ScrabbleCheater scrabbleCheater = new ScrabbleCheater();
        scrabbleCheater.readFile();
        scrabbleCheater.run();
    }

    public void run() {
        String input = "";
        BufferedReader userInput = new BufferedReader(new InputStreamReader(
            System.in));
        while (!input.equals("quit")) {
            try {
                input = userInput.readLine();
            } catch (IOException e) {
            }
```

Source Code continued on next page...

```java
            ArrayList<String> result = getSubstringWords(input);
            printResult(result);
        }
    }

    private void printResult(ArrayList<String> input) {
        for (String s : input) {
            System.out.println(s);
        }
    }

    private ArrayList<String> getWords(String input) {
        ArrayList<String> output = new ArrayList<>();
        String normalizedInput = normalize(input);
        int hash = generateHash(normalizedInput);
        String current = "";
        LinkedList<String> listOfPotentialWords = hashTable[hash];
        ListIterator<String> itr = listOfPotentialWords.listIterator();
        while (itr.hasNext()) {
            current = itr.next();
            if (normalize(current).equals(normalizedInput)) {
                output.add(current);
            }
        }
        return output;
    }

    private ArrayList<String> getSubstringWords(String input){
        ArrayList<String> substrings = getSubstrings(input);
        ArrayList<String> output = new ArrayList<>();
        for(String word:substrings){
            output.addAll(getWords(word));
        }
        return output;
    }

    private ArrayList<String> getSubstrings(String input){
        String normalizedInput = normalize(input);
        char[] chars = normalizedInput.toCharArray();
        ArrayList<String> output = new ArrayList<>();
        for(int i = 0; i < chars.length ; i++){
            for(int j = i+2; j< chars.length +1; j++){//start at i+2, to exclude single characters;continue to chars.length, to include the last character
                output.add(normalizedInput.substring(i,j));
            }
        }
        return output;
    }

    private String englishDistributionSevenLetters() {
        char[] distributionTable = new char[100];
        int counter = 0;
        String outputString = "";
        int[] frequencies = { 9, 2, 2, 4, 12, 2, 3, 2, 9, 1, 1, 4, 2, 6, 8, 2,
            1, 6, 4, 6, 4, 2, 2, 1, 2, 1 };
        for (int i = 0; i < frequencies.length; i++) {
            for (int j = 0; j < frequencies[i]; j++) {
                distributionTable[counter] = (char) (i + 97);
                counter++;
            }
        }
        Random rnd = new Random();
        for (int i = 0; i < 7; i++) {
            int randomIndex = rnd.nextInt(100);
            while (distributionTable[randomIndex] == 64) {
                randomIndex = rnd.nextInt(100);
            }
            outputString += distributionTable[randomIndex];
            distributionTable[randomIndex] = 64;
        }
        return outputString;

    }

    private String randomSevenLetters() {
        Random rnd = new Random();
        String returnString = "";
        for (int i = 0; i <= 7; i++) {
            char c = (char) (rnd.nextInt(26) + 97);
            returnString += c;
        }
        return returnString;
    }

    private boolean isPermutation(String word1, String word2) {
        if (normalize(word1).equals(normalize(word2))) {
            return true;
        } else
            return false;
    }

    public void readFile() throws FileNotFoundException {
        File dictionaryFile = new File("./src/wordslong.txt");
        String currentInput = "";
        BufferedReader fileReader = null;
        int count = 0;
            fileReader = new BufferedReader(new InputStreamReader(new
                FileInputStream(dictionaryFile)));

        try {
            currentInput = fileReader.readLine();
        } catch (IOException e1) {
            e1.printStackTrace();
        }
```

```java
        while (currentInput != null) {
            try {
                currentInput = fileReader.readLine();
            } catch (IOException e) {
                e.printStackTrace();
            }
            if (currentInput != null) {
                String normalized = normalize(currentInput);
                int hash = generateHash(normalized);
                putInHashTable(currentInput, hash);
            }
            // System.out.println(currentInput);
            count++;
        }

        try {
            fileReader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("Words total: " + count);
        System.out.println("Max colision: " + maxColisioncounter);
        System.out.println("Max colision position " + maxColisionposition);
    }

    private String normalize(String original) {
        char[] originalArray = original.toCharArray();
        Arrays.sort(originalArray);
        String sorted = new String(originalArray);
        return sorted;
    }

    private void putInHashTable(String word, int hash) {
        if (hash > 0 && hash < hashTable.length) {
            hashTable[hash].add(word);
            int colisioncounter = hashTable[hash].size();
            if (colisioncounter > maxColisioncounter) {
                maxColisioncounter = colisioncounter;
                maxColisionposition = hash;
            }
        }
    }

    private int generateHash(String word) {
        char[] chars = word.toCharArray();
        long polynom = 0;
        for (int i = 0; i < chars.length; i++) {
            int asciiPosition = chars[i] - 96; // a maps to 1
            polynom += Math.pow(asciiPosition, i);
        }
        return (int) (polynom % hashTableSize);
    }
}
```