**COMP 271 Data Structures and Algorithms**

**Finals**

**Due in Sakai by 1 pm May 1st, 2018.**

**Instructions:**

1. **Follow strict honor code. No human interaction is allowed except with me. You will not share this finals in entirety or in part, in any way electronic or otherwise.  Raising questions on public or private forums electronic or otherwise regarding the finals is also a violation of the policy, even if you don’t ultimately benefit from it.  Violation will result in a penalty grade of F in the course.**
2. **Open notes, book, publicly available code are all ok with proper attribution.**
3. **Pay attention to efficiency as indicated.  You will lose up to 20% of the credit if your underlying algorithm does not meet the stated O() time-complexity**.
4. **Submit only your .java code as a single zip file to Saka. Do not submit helper files unless you made changes to them.**

**You must type your name below as a way to indicate you will abide by the honor policy and other instructions above. Your submission will not be accepted if you fail to type in your name below.**

**Name: Andrew Littleton**

1. Write a Java program that **extends the LinkedList<E> class to ExtLinkedList<E>** containing the following methods. For time-complexity, assume the size of the list to be n elements. (20 points)
   * + - 1. public int countOccurences(E element) that returns the number of occurrences of “element” in the LinkedList if the “element” is found otherwise returns -1. Also, handle the condition when the list is empty. The underlying algorithm of your code should run in O(n) time.
         2. public boolean pairPresence(E element1, E element2) that checks if element1, and element2 are present in the linked list as a consecutive pair and if so returns true, if not, returns false. If there is more than one consecutive pair, you will return the first pair starting from the head of the list. Also, the code should return false when the list is empty or if it has only one element. You can use .equals() method for checking equality of elements. The underlying algorithm of your code should run in O(n) time.
2. import java.util.\*;
3. public class ExtendsLinkedList<E> extends LinkedList {
4. int count;
5. static ExtendsLinkedList<Integer> intList = new ExtendsLinkedList<Integer>();
6. public int countOccurences(E element){
7. if (this.length > 0){
8. for (int i = 0; i < this.size(); i++){
9. if (this.get(i) == element){
10. count++;
11. }
12. else{
13. return -1;
14. }
15. }
16. }
17. else{
18. System.out.println("Structure is empty");
19. }
20. return count;
21. count = 0;
22. }
23. public boolean pairPresence (E element1, E element2){
24. if (this.length > 1){
25. for (int i = 0; i < this.size(); i++){
26. if (this.get(i).equals(element1) && this.get(i+1).equals(element2)){
27. System.out.println(element1 + " " + element2);
29. return true;
30. }
31. }
32. }
33. else{
34. System.out.println("Structure is empty or too small to perform function");
35. return false;
36. }
37. }
38. public static void main (String [] args){
39. intList.add(1);
40. intList.add(4);
41. intList.add(6);
42. intList.add(1);
43. intList.add(3);
44. intList.add(1);
45. intList.add(1);
46. intList.add(7);
47. intList.add(15);
48. intList.add(9);
49. intList.add(12);
50. intList.add(28);
51. intList.countOccurences(1);
52. intList.pairPresence(4,6);
53. }
54. }
55. You are given two arrays of Strings – (i) array *soccer* consists of names of students who play soccer in a large school, and (ii) array *volleyball* consists of names of students who play volleyball in the school. Notice that there may be students who play both sports. Names can be one word, two words or three words consisting of alphabetic characters. Write methods to perform the following actions: (20 points)
56. *soccerOnly* (..) that returns students who play only soccer via a suitable data structure.
57. *volleyBallOnly(..)* that returns students who play only volleyball via a suitable data structure.
58. *bothSports*(..) that returns students who play both sports via a suitable data structure. No duplicates allowed.

Note that (..) means that you may or may not use any parameters for the methods, it would be up to you to decide. Secondly, you can use other data structures to copy the data into, to make the methods efficient. The methods will each return a suitable data structure containing the result. All three methods should run in *O(n)* time where *n* is equal to the sum of the sizes of the initial two arrays.

import java.util.\*;

public class SportsArrays{

static String[] soccerPlayers;

static String[] volleyballPlayers;

static ArrayList<String> arrListSoccer = new ArrayList<String>();

static ArrayList<String> arrListVolleyball = new ArrayList<String>();

static ArrayList<String> allPlayers = new ArrayList<String>();

public static void ArrayListCoversionSoccer (String[] arr){

for (int i = 0; i < arr.length; i++){

arrListSoccer.add(arr[i]);

}

}

public static void ArrayListCoversionVolleyball (String[] arr){

for (int i = 0; i < arr.length; i++){

arrListVolleyball.add(arr[i]);

}

}

public static ArrayList<String> soccerOnly(){

ArrayList<String> soccerOnly = new ArrayList<String>();

ArrayListCoversionVolleyball(volleyballPlayers);

for (int i = 0; i < soccerPlayers.length; i++){

if(!arrListVolleyball.contains(soccerPlayers[i])){

soccerOnly.add(soccerPlayers[i]);

}

}

return soccerOnly;

}

public static ArrayList<String> volleyballOnly(){

ArrayList<String> volleyballOnly = new ArrayList<String>();

ArrayListCoversionSoccer(soccerPlayers);

for (int i = 0; i < volleyballPlayers.length; i++){

if (!arrListSoccer.contains(volleyballPlayers[i])){

volleyballOnly.add(volleyballPlayers[i]);

}

}

return volleyballOnly;

}

public static ArrayList<String> bothSports(){

ArrayList<String> bothSports = new ArrayList<String>();

ArrayListCoversionSoccer(soccerPlayers);

ArrayListCoversionVolleyball(volleyballPlayers);

for (int i = 0; i < allPlayers.size(); i++){

if (arrListSoccer.contains(allPlayers.get(i)) && arrListVolleyball.contains(allPlayers.get(i))){

bothSports.add(allPlayers.get(i));

}

}

return bothSports;

}

public static void printOut (ArrayList<String> arrList){

if (arrList.size() == 0){

System.out.println("Structure is empty");

}

for (int i = 0; i < arrList.size(); i++){

System.out.println(arrList.get(i));

}

}

public static void main (String [] args){

soccerPlayers = new String[] {"John", "Gavin", "Adam", "Bella", "Andrew", "Annie", "Isabella", "Mark", "Geoff"};

volleyballPlayers = new String[] {"John", "Geoff", "Isabella", "Adrian", "Mark", "Lance", "Parker", "Troy"};

for (int i = 0; i < soccerPlayers.length; i++){

if (!allPlayers.contains(soccerPlayers[i])){

allPlayers.add(soccerPlayers[i]);

}

}

for (int i = 0; i < volleyballPlayers.length; i++){

if (!allPlayers.contains(volleyballPlayers[i])){

allPlayers.add(volleyballPlayers[i]);

}

}

System.out.println("Soccer Only: ");

printOut(soccerOnly());

System.out.println();

System.out.println("Volleyball Only: ");

printOut(volleyballOnly());

System.out.println();

System.out.println("Both Sports: ");

printOut(bothSports());

System.out.println();

}

}

1. Use the text file (in UTF-8 format) Automotive\_5.txt for this exercise. Your first goal is to find all words in the text file that are good in the dictionary (ignore the rest) and also count the frequency of occurrence of each such word in the text file. You will convert every word into lowercase and get rid of punctuations if present. Your code thus far, should run in O(n) time where n is the number of words in the txt file. Your helper files are EnglishWordList.txt and SpellCheck.java. You can do this exercise by making changes/additions to SpellCheck.java code. But you only need to make small number of changes to the code. So don’t overthink this. Add code for the additional method *public void printStats()* that prints the word with the highest frequency, the word with the lowest frequency, the frequency counts of each and the total number of words. For example, your output could be like this:

“*word*” occurs with the highest frequency of *“count*” times.

“*word*” occurs with the lowest frequency of “*count*” times.

Total number of good words is equal to “*count*”

(20 points)

/\*\*

\* Write a description of class ReadFiles here.

\*

\* @author (your name)

\* @version (a version number or a date)

\*/

import java.util.\*;

import java.io.\*;

public class SpellCheck

{

static HashSet<String> dictionary = new HashSet<String>();

static ArrayList<String> story = new ArrayList<String>();

static ArrayList<String> words = new ArrayList<String>();

static HashMap<Integer,String> wordMap = new HashMap<Integer,String>();

public void storeDictionaryAndStory() {

Scanner reader=null;

try { reader = new Scanner(new File("EnglishWordList.txt"));

}

catch ( FileNotFoundException ex)

{System.out.println(ex+" file not found ");

}

while (reader.hasNext()){

String str = reader.next();

str = str.replaceAll("[\\[\\]\_:\"'`?;\\â€0-9â€”;â€œ()-/.,\*! ]", "").toLowerCase();

dictionary.add(str);

}

try {

reader = new Scanner(new File("Automotive\_5.txt"));

}

catch ( FileNotFoundException ex)

{System.out.println(ex+" file not found ");

}

while (reader.hasNext()){

String str = reader.next();

str = str.replaceAll("[\\[\\]\_:\"'`?;\\â€0-9â€”;â€œ()-/.,\*! ]", "").toLowerCase();

story.add(str);

}

}

public static void spellCheck () {

for (String s:story)

if (dictionary.contains(s)){

words.add(s);

}

}

public static void frequency(){

int greatestCount = 0;

int lowestCount = 10;

int tempCount;

for (int i = 0; i < words.size(); i++){

tempCount = Collections.frequency(words, words.get(i));

wordMap.put(tempCount, words.get(i));

if ( tempCount > greatestCount){

greatestCount = tempCount;

}

if ( tempCount < lowestCount){

lowestCount = tempCount;

}

}

System.out.println(wordMap.get(greatestCount) + " has the highest frequency of " + greatestCount);

System.out.println(wordMap.get(lowestCount) + " has the lowest frequency of " + lowestCount);

}

public void spellCheckNoDuplicates() {

HashSet<String> badWords = new HashSet<String>();

for (String s:story)

if (!dictionary.contains(s))

badWords.add(s);

System.out.println(badWords);

System.out.println(badWords.size());

}

public static void main (String [] args){

spellCheck();

frequency();

}

}