CMP – 5015Y

MODULE ASSESSMENT EXERCISES 2

100108964

Part 1(pseudocode)

Part 2(Java Project)

1. Form Dictionary and Word Count

List<String> Dictionary

FormDictionary()
    StringBuilder sb ← create stringbuilder
    dictionaryWords ← create arraylist to hold dictionary
    String aFile := "testDocument.txt"

    For line in readWordFromCSV(aFile)
        DictionaryWords.add(line) ← adds words from file

    Set wCount ← creates HashSet of dictionaryWords
    List SortList ← new list to sort wCount
    Collections.sort(SortList)

    For word : SortList
        Print(word)
        Collections.sort(dictionaryWords) ← sort dictionaryWords

    For value : readWordsFromCSV(aFile) ←read value and append to string builder
        Sb.append(value)

    String file := sb.toString
    Return dictionaryWords

---

saveToFile()
    String fileName := file
    PrintWriter writer := Printwriter(file)
    df.formDictionary() ← create Dictionary
    writer.println(formDictionary) ← write to file
    writer.close()

---

countFrequencies()
    StringBuilder sb ← create stringbuilder
    ArrayList dictionary ← new dictionary list

    String aFile ←open file
    For line : readWordsFromCSV(aFile)
        dictionaryWords.add(line)

    Set wCount ← create hashmap
    List SortList

    For word : SortList
        Print(Collections.frequency(dictionaryWords, words) ← prints frequency of words

Algorithm Analysis

formDictionary()

1. Fundamental Operation: `dictionaryWords.add(line);`

For loop adding string to the dictionaryWords list

2. Worst case: `File not opening in readWordsFromCSV`
3. Runtime Complexity
4. Characterise RTF: Order n

---

2. Implement a Trie Data Structure

TrieNode.java

```
Boolean wordComplete
int ALPHABET := constant alphabet size
char c
TrieNode[] offspring := size of ALPHABET
TrieNode left
TrieNode right
String data
```

```
TrieNode(String d)
    Offspring := new TrieNode ← assign TrieNode offspring size ALPHABET
    this.data = d
    this.left = null;
    this.right = null
```

```
char getChar()
    return c ← return character in node
```

```
TrieNode getOffspring(char x)
    TrieNode a := new node
    For d : offspring
        If a.getchar() == x
            Return a
        return null
```

```
void add(String key)
    int level
    int length: = key.length ← length of string
    int pos ← position of char

    TrieNode node := new TrieNode

    For level := 0  level < length level++
```

```
        pos := key.charAt(level) – ‘a’ ← scan string and assign index the character at
postion of level – ‘a’ 26
        If node.offspring[pos] == null
                node.offspring[level] := new TrieNode ←if position of offspring is null return false
        node := node.offspring[pos]
        else
                node.wordComplete := true ← else set it to true
```

```
Boolean contains(String key)
        Int level
        Int length := key.length
        Int pos
        TrieNode node := new TrieNode

        For level :=0 level < length level++
                Index = key.charAt(level) – ‘a’

        If node.offspring[pos] == null
                return false ← return false if offspring is empty

        node := node.offspring[pos]

        return node != null && node.wordComplete ← set node to true
```

```
Trie.java
        TrieNode root ← root of TrieNode
        Char c

        Root := new TrieNode ← create a new TrieNode
```

```
TrieNode getNode(String key)

        TrieNode Node := root
        For I := 0 I < key.length() i++
                TrieNode offspring := Node.offspring[key.charAt – ‘a’]
                If offspring == null
                        Return null
                Node := offspring

return node
```

```
String outputBreadthFirstSearch()
        TrieNode node
        Queue<TrieNode> q := new LinkedList ← create a collection of linkedlist
        q.add(root); ← add root to queue

        String bfs := “” ← create empty string output word
        If root == null
                Print(“empty)
        q.clear();
        q.add(root)
```

```
        while(!q.isEmpty())
                node = q.remove()
                print(node.data + " ") ← prints data string in node
                bfs += node.data + " "
                if node.left != null
                        q.add(node.left) ← add character to left node if not empty
                if(node.right  != null
                        q.add(node.right) ← add node character to right node
        return bfs ← return string
```

```
String outputDepthFirstSearch()
        TrieNode node = root

        If root == null
                Print("empty)
        Stack s := new Stack ← create a new stack
        String dfs := ""
        s.push(root) ← push item onto stack
        while(s.isEmpty() == false)
                temp = s.peek ← check object at top of stack
                TrieNode c = s.pop()

                If(c.right != null)
                        s.add(c.add) ← if right is null add node c to stack
                if(c.left!=null)
                        s.add(c.left) ← if left is null add to stack
                print(" " + c.data)
                dfs += node.data + " " ← add to string to return
        return dfs
```

```
Trie getSubTrie(String prefix)
        Trie t := new Trie() ← create new Trie
        TrieNode temp: = root ← assign root to temp

        For i = 0 i < prefix.length() i++ ← scan length of string
                TrieNode word := temp.getOffspring(prefix.charAt(i)) ← word assign offspring that
is at position i
                If(word == null)
                        print("empty") ← print if null
                temp := word ← temp becomes word
                t.root := temp ← temp is assigned to trie root
        return t        ← return trie
```

```
List getAllWords
        TrieNode node := root ← assign root to node
        List<TrieNode> list ← new arraylist

        For TrieNode s : list ← scan and assign node to s
                s:=node
                list.add(s) ← add node to list
```

```
        print(list)
return list ← returns words
```

1 DictionaryFinder.java

```
1  import java.io.*;
2  import java.nio.BufferUnderflowException;
3  import java.util.*;
4
5  /**
6  //SECTION 1 - FORM DICTIONARY AND WORD COUNT
7  */
8  public class DictionaryFinder {
9
10     private List < String > Dictionary;
11
12     public DictionaryFinder(ArrayList < String > dictionary) {
13         this.Dictionary = dictionary;
14
15     }
16
17     public DictionaryFinder() {
18
19     }
20
21     /**
22      * Reads all the words in a comma separated text document into an Array
23      * @param
24      */
25     public static ArrayList < String > readWordsFromCSV(String file) throws
26     FileNotFoundException {
27         Scanner sc = new Scanner(new File(file));
28         sc.useDelimiter(" |,");
29         ArrayList < String > words = new ArrayList < > ();
30         String str;
31         while (sc.hasNext()) {
32             str = sc.next();
33             str = str.trim();
34             str = str.toLowerCase();
35             words.add(str);
36             //Collections.sort(words);
37         }
38         return words;
39     }
40     public static void saveCollectionToFile(Collection << ? > c, String file) throws
41     IOException {
42         FileWriter fileWriter = new FileWriter(file);
43         PrintWriter printWriter = new PrintWriter(fileWriter);
44         for (Object w: c) {
45             printWriter.println(w.toString());
46         }
47         printWriter.close();
48     }
49
50     public static ArrayList < String > formDictionary() throws FileNotFoundException {
51         StringBuilder sb = new StringBuilder();
52         ArrayList < String > dicitionaryWords = new ArrayList < > ();
```

```
53
54          String aFile = "src/testDocument.txt";
55
56          //add strings to arraylist
57          for (String line: readWordsFromCSV(aFile)) {
58              dicitionaryWords.add(line);
59          }
60
61          Set < String > wCount = new HashSet < > (dicitionaryWords);
62          List SortList = new ArrayList(wCount);
63          Collections.sort(SortList);
64          // int count = 0;
65          for (Object word: SortList) {
66              // Integer count = wCount.get(word);
67              // wCount.put(word, (count==null) ? 1 : count + 1);
68              System.out.println(word + ", " + Collections.frequency(dicitionaryWords, word));
69              System.out.println(word); //Collections.frequency(dicitionaryWords, word));
70              Collections.sort(dicitionaryWords);
71          }
72
73          for (String value: readWordsFromCSV(aFile)) {
74              sb.append(value);
75          }
76          String file = sb.toString();
77
78          //System.out.print(readWordsFromCSV(aFile).toString().replaceAll("(^\\[|\\]£)", ""));
79          return dicitionaryWords;
80      }
81      public void saveToFile() throws IOException {
82          ArrayList < String > Dictionary = new ArrayList < > ();
83          String fileName = "dictionary.txt";
84          DictionaryFinder df = new DictionaryFinder(Dictionary);
85
86          PrintWriter writer = new PrintWriter(new FileWriter(fileName));
87          df.formDictionary();
88          writer.println(formDictionary());
89          writer.close();
90
91      }
92
93      /**
94       * function to count frequencies using hashmap
95       * @throws FileNotFoundException
96       */
97      public void countFrequencies() throws FileNotFoundException {
98
99          StringBuilder sb = new StringBuilder();
100         ArrayList < String > dicitionaryWords = new ArrayList < > ();
101
102         String aFile = "src/testDocument.txt";
103         for (String line: readWordsFromCSV(aFile)) {
104             dicitionaryWords.add(line);
105         }
106
```

```
107         Set < String > wCount = new HashSet < > (dicitionaryWords);
108         List SortList = new ArrayList(wCount);
109         Collections.sort(SortList);
110 
111         //print frequencies of words in dictionary
112         for (Object word: SortList) {
113             System.out.println(Collections.frequency(dicitionaryWords, word));
114 
115         }
116 
117     }
118 
119     /**
120      * testing dictionary
121      * @param args
122      * @throws Exception
123      */
124     public static void main(String[] args) throws Exception {
125 
126         ArrayList < String > Dictionary = new ArrayList < > ();
127         String aFile = null;
128         DictionaryFinder df = new DictionaryFinder(Dictionary);
129         ArrayList < String > in = readWordsFromCSV("src/testDictionary.txt");
130         //DO STUFF TO df HERE in countFrequencies
131 
132         df.formDictionary();
133         df.saveToFile();
134 
135         ArrayList < String > list = new ArrayList < > ();
136 
137         // list = df.formDictionary();
138 
139         // Trie n = new Trie(Dictionary);
140         // TrieNode node = new TrieNode();
141         //  node.add("c");
142 
143         //    n.outputBreadthFirsdtSearch();
144 
145         String keys[] = {
146             "cheese",
147             "forty",
148             "nine",
149             "a",
150             "bat"
151         };
152 
153         String[] output = {
154             "not in trie",
155             "is in trie"
156         };
157         char c = 'a';
158         TrieNode root = new TrieNode();
159         Trie a = new Trie();
160 
```

```
161         //adding dictionary to arraylist and converting it to string
162         //  String[] s = df.formDictionary().toArray(new String[0]);
163 
164         /// for (int i = 0; i<s.length; i++){
165         //     System.out.println(s[i]);
166         // }
167 
168 
169         //add strings to trie
170         for (int i = 0; i < keys.length; i++) {
171             a.add(keys[i]);
172         }
173 
174         /* Trie t = new Trie();
175          t.root = new TrieNode("ARSENAL");
176          t.root.add(new TrieNode("FORTY"));
177          t.root.offspring[0].add(new TrieNode("I"));
178          t.root.offspring[1].add(new TrieNode("Bet"));
179          t.root.offspring[2].offspring[1].add(new TrieNode("Examples"));
180 
181          System.out.println(t);
182          */
183 
184         //adding nodes to the trie
185         a.root = new TrieNode("HELLO");
186         a.root.left = new TrieNode("s");
187         a.root.left.left = new TrieNode("t");
188         a.root.right = new TrieNode("c");
189         a.root.right.left = new TrieNode("a");
190         a.root.right.left.left = new TrieNode("t");
191 
192         System.out.println("order ");;
193         a.outputBreadthFirstSearch();
194         System.out.println(" ");
195 
196 
197         if (a.contain("forty") == true) {
198             System.out.println("forty--- " + output[1]);
199         } else
200             System.out.println("forty --- " + output[0]);
201 
202         if (a.contain("cheese") == true) {
203             System.out.println("cheese --- " + output[1]);
204         } else System.out.println("these --- " + output[0]);
205 
206 
207     }
208 
209 
210 }
```

2 TrieNode.java

```
1  import java.util.Queue;
2  /**
3  / SECTION 2 - IMPLEMENT TRIE DATA STRUCTURE
4  **/
5  public class TrieNode {
6      boolean wordComplete;
7      private static final int ALPHABET = 26;
8      public char c;
9      TrieNode[] offspring;
10     TrieNode left;
11     TrieNode right;
12     String data;
13
14     /**
15      * creates a new TrieNode offspring of size ALPHABET
16      */
17     public TrieNode() {
18         this.offspring = new TrieNode[ALPHABET];
19         this.wordComplete = false;
20
21     }
22
23     /**
24      * assign fields in constructor and positions of the characters in node
25      * @param d
26      */
27     public TrieNode(String d) {
28         offspring = new TrieNode[ALPHABET];
29         this.data = d;
30         this.left = null;
31         this.right = null;
32     }
33
34     /**
35      * return character
36      * @return
37      */
38     public char getChar() {
39         return c;
40     }
41
42     /**
43      * get offspring by scanning node and return character on match
44      * @param x
45      * @return
46      */
47     public TrieNode getOffspring(char x) {
48         TrieNode a = new TrieNode();
49         for (TrieNode d: offspring) {
50             if (a.getChar() == x) {
51                 return a;
52             }
```

```
53          }
54
55          return null;
56      }
57
58      // static TrieNode root = new TrieNode();
59
60      /**
61       * add string to node
62       * @param key
63       */
64      public void add(String key) {
65          int level;
66          int length = key.length();
67          int index;
68          TrieNode node = new TrieNode();
69          char fChar = key.charAt(0);
70          //        TrieNode offspring = getOffspring(fChar);
71
72          for (level = 0; level < length; level++) {
73              index = key.charAt(level) - 'a';
74              if (node.offspring[index] == null) {
75                  node.offspring[index] = new TrieNode();
76
77                  node = node.offspring[index];
78
79              } else
80                  node.wordComplete = true;
81
82          }
83
84
85      }
86
87      /**
88       * find is node contains string
89       * @param key
90       * @return
91       */
92      public boolean contain(String key) {
93          int level;
94          int length = key.length();
95          int index;
96          TrieNode node = new TrieNode();
97
98          for (level = 0; level < length; level++) {
99              index = key.charAt(level) - 'a';
100
101             if (node.offspring[index] == null)
102                 return false;
103
104             node = node.offspring[index];
105
106
```

```
107             }
108             return (node != null && node.wordComplete);
109         }
110 
111         /*
112            /**
113             * set character
114             * @param c
115             * @param node
116             */
117         /* public void setChild(char x, TrieNode node){
118                 char c;
119                 node = new TrieNode();
120                 c = x;
121                 node.add(c);
122 
123            }
124 
125            */
126 
127         /*8
128         add node delete
129          */
130         public void add(TrieNode s) {
131 
132             Trie t = new Trie();
133             t.root = s;
134 
135         }
136     }
```

3 Trie.java

```
1  import java.io.FileNotFoundException;
2  import java.io.IOException;
3  import java.util.*;
4
5  public class Trie {
6
7      public static TrieNode root;
8      public char c;
9      TrieNode left;
10     TrieNode right;
11     String data;
12
13
14     /**
15      * create empty trie node
16      */
17     public Trie() {
18
19         root = new TrieNode();
20
21     }
22
23     /**
24      * get node from the trie
25      * @param key
26      * @return
27      */
28     public TrieNode getNode(String key) {
29         TrieNode Node = root;
30         for (int i = 0; i < key.length(); i++) {
31             TrieNode offspring = Node.offspring[key.charAt(i) - 'a'];
32             if (offspring == null) {
33                 return null;
34             }
35             Node = offspring;
36         }
37         return Node;
38     }
39
40     public boolean contains(String key) {
41         TrieNode Node = getNode(key);
42         return Node != null && Node.wordComplete;
43     }
44
45
46     public boolean contain(String keys) {
47
48         int level;
49         int length = keys.length();
50         int index;
51         TrieNode node = root;
52
```

```
53          for (level = 0; level < length; level++) {
54              index = keys.charAt(level) - 'a';
55
56              if (node.offspring[index] == null)
57                  return false;
58
59              node = node.offspring[index];
60          }
61
62          return (node != null && node.wordComplete);
63      }
64
65
66      /**
67       * add string to the trie at next node
68       * @param key
69       */
70      public void add(String key) {
71          int level;
72          int length = key.length();
73          int pos;
74          TrieNode node = root;
75          char fChar = key.charAt(0);
76
77          for (level = 0; level < length; level++) {
78              pos = key.charAt(level) - 'a';
79              if (node.offspring[pos] == null) {
80                  node.offspring[pos] = new TrieNode();
81
82                  node = node.offspring[pos];
83
84              } else
85                  node.wordComplete = true;
86
87          }
88
89
90      }
91
92      /**
93       * return String containing words in traversal breadth order using queue
94       * @return
95       */
96
97      String outputBreadthFirstSearch() {
98          TrieNode node;
99
100         Queue < TrieNode > q = new LinkedList < TrieNode > ();
101         q.add(root);
102
103         String bfs = "";
104         if (root == null)
105             System.out.println("Empty");
106         q.clear();
```

```
107             q.add(root);
108             while (!q.isEmpty()) {
109 
110                 node = q.remove();
111                 System.out.print(node.data + " ");
112                 bfs += node.data + " ";
113                 if (node.left != null) {
114                     q.add(node.left);
115                 }
116                 if (node.right != null) {
117                     q.add(node.right);
118                 }
119             }
120             return bfs;
121 
122         }
123 
124         String outputDepthFirstSearch() {
125 
126             //creates an node of root
127             TrieNode node = root;
128             TrieNode temp = new TrieNode();
129 
130             //return empty
131             if (root == null) {
132                 System.out.print("empty");
133             }
134 
135             //empty stack
136             Stack < TrieNode > s = new Stack < TrieNode > ();
137             String dfs = "";
138             s.push(root);
139             while (s.isEmpty() == false) {
140                 temp = s.peek();
141                 TrieNode c = s.pop();
142 
143                 if (c.right != null) {
144                     s.add(c.right);
145                 }
146                 if (c.left != null) {
147                     s.add(c.left);
148                 }
149                 System.out.print(" " + c.data);
150                 dfs += node.data + " ";
151 
152             }
153             return dfs;
154         }
155 
156 
157         //get prefix of word
158         public Trie getSubTrie(String prefix) {
159             Trie t = new Trie();
160             TrieNode temp = root;
```

```
161
162             //searches if prefix exists in the trie
163             for (int i = 0; i < prefix.length(); i++) {
164                 TrieNode word = temp.getOffspring(prefix.charAt(i));
165                 if (word == null) {
166                     System.out.println("empty");
167                 }
168                 temp = word;
169                 t.root = temp;
170
171             }
172
173             return t;
174
175         }
176
177         //scan through node and add to list then print
178         public List getAllWords() {
179             TrieNode node = root;
180             List < TrieNode > list = new ArrayList < > ();
181             for (TrieNode s: list) {
182                 s = node;
183                 list.add(s);
184                 System.out.print(list + ", ");
185             }
186
187             return list;
188
189         }
190
191
192         /**SECTION 3 - WORD AUTO COMPLETION
193          * testing the trie
194          * @throws FileNotFoundException
195          */
196         public static void main() throws IOException {
197             ArrayList < String > Dictionary = new ArrayList < > ();
198             String aFile = null;
199             DictionaryFinder df = new DictionaryFinder(Dictionary);
200             ArrayList < String > in = DictionaryFinder.readWordsFromCSV("src/testDictionary.txt");
201             //DO STUFF TO df HERE in countFrequencies
202
203             df.formDictionary();
204             df.saveToFile();
205
206
207             String keys[] = {
208                 "arsernal",
209                 "forty",
210                 "nine"
211             };
212
213             String[] output = {
214                 "not in trie",
```

```
215             "is in trie"
216         };
217 
218         TrieNode root = new TrieNode();
219         Trie a = new Trie();
220 
221         //adding dictionary to arraylist and converting it to string
222         String[] s = df.formDictionary().toArray(new String[0]);
223 
224         /// for (int i = 0; i<s.length; i++){
225         //     System.out.println(s[i]);
226         // }
227 
228         //add keys to the trie
229         int i = 0;
230         for (i = 0; i < keys.length; i++) {
231             a.add(keys[i]);
232         }
233 
234         /* Trie t = new Trie();
235          t.root = new TrieNode("ARSENAL");
236          t.root.add(new TrieNode("FORTY"));
237          t.root.offspring[0].add(new TrieNode("I"));
238          t.root.offspring[1].add(new TrieNode("Bet"));
239          t.root.offspring[2].offspring[1].add(new TrieNode("Examples"));
240 
241          System.out.println(t);
242          */
243 
244         //test the trie
245         a.root = new TrieNode("HELLO");
246         a.root.left = new TrieNode("s");
247         a.root.left.left = new TrieNode("t");
248         a.root.right = new TrieNode("c");
249         a.root.right.left = new TrieNode("a");
250         a.root.right.left.left = new TrieNode("t");
251 
252         //output BreadthFirstSearch
253         System.out.println("order ");;
254         a.outputBreadthFirstSearch();
255         System.out.println(" ");
256 
257         if (a.contain("arsenal") == true) {
258             System.out.println("arsenal " + output[1]);
259         } else
260             System.out.println("arsenal " + output[0]);
261         if (a.contain("forty") == true) {
262             System.out.println("forty " + output[1]);
263         } else System.out.println("forty" + output[0]);
264 
265     }
266 
267 
268 }
```

4 AutoCompletion.java

```
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.util.*;
4
5  public class AutoCompletion extends DictionaryFinder {
6
7      private static ArrayList < String > dicitionaryWords;
8
9      public AutoCompletion() {
10         super();
11         dicitionaryWords = new ArrayList < > ();
12
13     };
14
15     public static ArrayList < String > readWordsFromCSV(String file) throws FileNotFoundExceptio
16         Scanner sc = new Scanner(new File(file));
17         sc.useDelimiter(" |,");
18         ArrayList < String > words = new ArrayList < > ();
19         String str;
20         while (sc.hasNext()) {
21             str = sc.next();
22             str = str.trim();
23             str = str.toLowerCase();
24             words.add(str);
25             //Collections.sort(words);
26         }
27         return words;
28
29     }
30
31     public static ArrayList < String > formDictionary(String file) throws FileNotFoundException
32         StringBuilder sb = new StringBuilder();
33         ArrayList < String > dicitionaryWords = new ArrayList < > ();
34
35         String aFile = file;
36         for (String line: readWordsFromCSV(aFile)) {
37             dicitionaryWords.add(line);
38         }
39
40         Set < String > wCount = new HashSet < > (dicitionaryWords);
41         List SortList = new ArrayList(wCount);
42         Collections.sort(SortList);
43         // int count = 0;
44         for (Object word: SortList) {
45             // Integer count = wCount.get(word);
46             // wCount.put(word, (count==null) ? 1 : count + 1);
47             System.out.println(word + ", " + Collections.frequency(dicitionaryWords, word));
48             //System.out.println(word); //Collections.frequency(dicitionaryWords, word));
49             Collections.sort(dicitionaryWords);
50         }
51
52         for (String value: readWordsFromCSV(aFile)) {
```

```
53              sb.append(value);
54          }
55          String s = sb.toString();
56
57
58          //System.out.print(readWordsFromCSV(aFile).toString().replaceAll("(^\\[|\\]£)", ""));
59          return dicitionaryWords;
60      }
61
62      public static void formTrie() throws FileNotFoundException {
63
64          DictionaryFinder d = new DictionaryFinder();
65          String file = "src/lotrQueries.csv";
66          String q2 = "src/lotrQueries.csv";
67          String gollum = "src/gollum.csv";
68          formDictionary(file);
69          formDictionary(q2);
70
71
72          formDictionary(gollum);
73
74          Trie t = new Trie();
75          TrieNode node = new TrieNode();
76
77          int i = 0;
78          for (i = 0; i < dicitionaryWords.size(); i++) {
79              t.add(dicitionaryWords.get(i));
80          }
81
82          //input words into a list
83          ArrayList < String > prefix = readWordsFromCSV(file);
84
85          //load prefix
86          for (String wordPrefix: prefix) {
87              t.getSubTrie(wordPrefix);
88          }
89
90
91
92
93
94
95
96
97
98      }
99
100
101
102 }
```