```java
 * Student's name:

package dataStructures.trie;

import dataStructures.dictionary.AVLDictionary;

public class DictionaryStringTrie<V> {
  protected static class Node<V> {
    V value;
    Dictionary<Character, Node<V>> children;

    Node() {
      this.value = null;
      this.children = new AVLDictionary<>();
    }
  }

  protected Node<V> root;

 /**************************************************************************
  * DO NOT WRITE ANY CODE ABOVE
  *************************************************************************/

  // | = Exercise a - constructor
  public DictionaryStringTrie() {
      root = new Node<>();
  }

  // | = Exercise b - isEmpty
  public boolean isEmpty() {
      return root.children.isEmpty();
  }


  // | = Exercise c - sizeValue
  protected static <V> int sizeValue(V value) {
      return (value==null)? 0 : 1;
  }

  // | = Exercise d - size
  public int size() {
      return size(root);
  }

  protected static <V> int size(Node<V> node) {
      int res = 0;
      if(node == null) {
          return res;
      } else if(sizeValue(node.value) == 1){
          res++;
          if(!node.children.isEmpty()) {
              for(Node<V> n : node.children.values()) {
                  res += size(n);
              }
          }
      } else {
          if(!node.children.isEmpty()) {
              for(Node<V> n : node.children.values()) {
                  res += size(n);
              }
          }
      }
      return res;
  }
```

```java

// | = Exercise e - childOf
protected static <V> Node<V> childOf(char c, Node<V> node) {
    if(node.isEmpty() || !node.children.isDefinedAt(c)) {
        return null;
    } else {
        return node.children.valueOf(c);
    }
}

// | = Exercise f - search
public V search(String str) {
    return search(str, root);
}

protected static <V> V search(String str, Node<V> node) {
    V valor = null;

    if(node.isEmpty()) {
        return null;
    } else if(str.isEmpty()) {
        valor = node.value;
    } else {
        Char c = str.charAt(0);
        String suffix = str.substring(1);
        for(Character x : node.children.keys()) {
            if(c == x) {
                valor = search(suffix, childOf(x, node));
            }
        }
    }

    return valor;
}

// | = Exercise g - insert
public void insert(String str, V value) {
    return insert(str, value, root);
}

protected static <V> Node<V> insert(String str, V value, Node<V> node) {
    if(node.isEmpty()) {
        node = new Node<>();
    } else {
        if(str.isEmpty()) {
            node.value = value;
        } else {
            Char c = str.charAt(0);
            String suffix = str.substring(1);
            Node<V> child = childOf(c, node);

            if(child == null)) {
                child = new Node<>();
                node.children.insert(c, child);
            } else {
                insert(suffix, value, child);
            }
        }
    }

    return node;
}

/*****************************************************************************
```

```
142   * ONLY FOR PART TIME STUDENTS
143   *****************************************************************************/
144
145  // | = Exercise e1 - strings
146  public List<String> strings() {
147      return strings(root);
148  }
149
150  protected static <V> List<String> strings(Node<V> node) {
151      List<String> result = new LinkedList<>();
152
153        if (node.isEmpty()) {
154            return result;
155        }
156
157        if (node.value != null) {
158            result.add("");
159        }
160
161        for (Tuple2<Character, Node<V>> par : node.children.keysValues()) {
162            char c = par._1();
163            Node<V> child = par._2();
164            List<String> childStrings = strings(child);
165
166            for (String s : childStrings) {
167                result.add(c + s);
168            }
169        }
170
171        return result;
172  }
173
174  // | = Exercise e2 - fromList
175  public static DictionaryStringTrie<Integer> fromList(List<String> list) {
176      DictionaryStringTrie<Integer> trie = new DictionaryStringTrie<>();
177
178      for (String word : list) {
179          Integer currentValue = trie.search(word);
180
181          if (currentValue != null) {
182                // La palabra ya existe en el Trie, incrementar el valor actual
183                trie.insert(word, currentValue + 1);
184          } else {
185                // La palabra no existe en el Trie, insertar con valor 1
186                trie.insert(word, 1);
187          }
188      }
189
190     return trie;
191  }
192
193  /*****************************************************************************
194   * DO NOT WRITE ANY CODE BELOW
195   *****************************************************************************/
196
197  public String toString() {
198    StringBuilder sb = new StringBuilder();
199    if (root != null) {
200      sb.append(root.getClass().getSimpleName());
201      sb.append(' ');
202      sb.append(root.value);
203      sb.append('\n');
204      toString(sb,1, root);
205    }
```

```
270     Dictionary<Character, Node<Integer>> d3 = n3.children;
271     Node<Integer> n4 = new Node<>();
272     Dictionary<Character, Node<Integer>> d4 = n4.children;
273     Node<Integer> n5 = new Node<>();
274     Dictionary<Character, Node<Integer>> d5 = n5.children;
275     Node<Integer> n6 = new Node<>();
276     Dictionary<Character, Node<Integer>> d6 = n6.children;
277     Node<Integer> n7 = new Node<>();
278     Dictionary<Character, Node<Integer>> d7 = n7.children;
279     Node<Integer> n8 = new Node<>();
280     Node<Integer> n9 = new Node<>();
281     Node<Integer> n10 = new Node<>();
282     Node<Integer> n11 = new Node<>();
283     d0.insert('b',n1);
284     d0.insert('c',n2);
285     d0.insert('t',n3);
286     d1.insert('a',n4);
287     d1.insert('e',n5);
288     d2.insert('a',n6);
289     d3.insert('o',n7);
290     d4.insert('t',n8);
291     d5.insert('d',n9);
292     n5.value = 1;
293     d6.insert('t',n10);
294     d7.insert('e',n11);
295     n7.value = 4;
296     n8.value = 0;
297     n9.value = 2;
298     n10.value = 3;
299     n11.value = 5;
300     trie.root = n0;
301     return trie;
302   }
303
304   public static DictionaryStringTrie<Integer> sampleTrie1() {
305     // a -> 3  b -> 2  c -> 1
306     DictionaryStringTrie<Integer> trie = new DictionaryStringTrie<>();
307     Node<Integer> n0 = new Node<>();
308     Node<Integer> n1 = new Node<>();
309     n1.value = 3;
310     Node<Integer> n2 = new Node<>();
311     n2.value = 2;
312     Node<Integer> n3 = new Node<>();
313     n3.value = 1;
314     n0.children.insert('a', n1);
315     n0.children.insert('b', n2);
316     n0.children.insert('c', n3);
317     trie.root = n0;
318     return trie;
319   }
320
321   public static DictionaryStringTrie<Integer> sampleTrie2() {
322     // a -> 1  ab -> 2  abc -> 3  abd -> 4  acdef -> 5
323     DictionaryStringTrie<Integer> trie = new DictionaryStringTrie<>();
324     Node<Integer> n0 = new Node<>();
325     Node<Integer> n1 = new Node<>();
326     n1.value = 1;
327     Node<Integer> n2 = new Node<>();
328     n2.value = 2;
329     Node<Integer> n3 = new Node<>();
330     n3.value = 3;
331     Node<Integer> n4 = new Node<>();
332     n4.value = 4;
333     Node<Integer> n5 = new Node<>();
```

```
334     Node<Integer> n6 = new Node<>();
335     Node<Integer> n7 = new Node<>();
336     Node<Integer> n8 = new Node<>();
337     n8.value = 5;
338     n0.children.insert('a', n1);
339     n1.children.insert('b', n2);
340     n1.children.insert('c', n5);
341     n2.children.insert('c', n3);
342     n2.children.insert('d', n4);
343     n5.children.insert('d', n6);
344     n6.children.insert('e', n7);
345     n7.children.insert('f', n8);
346     trie.root = n0;
347     return trie;
348   }
349
350   public static DictionaryStringTrie<Integer> sampleTrie3() {
351     // abcd -> 1
352     DictionaryStringTrie<Integer> trie = new DictionaryStringTrie<>();
353     Node<Integer> n0 = new Node<>();
354     Node<Integer> n1 = new Node<>();
355     Node<Integer> n2 = new Node<>();
356     Node<Integer> n3 = new Node<>();
357     Node<Integer> n4 = new Node<>();
358     n4.value = 1;
359     n0.children.insert('a', n1);
360     n1.children.insert('b', n2);
361     n2.children.insert('c', n3);
362     n3.children.insert('d', n4);
363     trie.root = n0;
364     return trie;
365   }
366
367   public static DictionaryStringTrie<Integer> sampleTrie4() {
368     // abcd -> 1  def -> 2
369     DictionaryStringTrie<Integer> trie = new DictionaryStringTrie<>();
370     Node<Integer> n0 = new Node<>();
371     Node<Integer> n1 = new Node<>();
372     Node<Integer> n2 = new Node<>();
373     Node<Integer> n3 = new Node<>();
374     Node<Integer> n4 = new Node<>();
375     n4.value = 1;
376     Node<Integer> n5 = new Node<>();
377     Node<Integer> n6 = new Node<>();
378     Node<Integer> n7 = new Node<>();
379     n7.value = 2;
380     n0.children.insert('a', n1);
381     n0.children.insert('d', n5);
382     n1.children.insert('b', n2);
383     n2.children.insert('c', n3);
384     n3.children.insert('d', n4);
385     n5.children.insert('e', n6);
386     n6.children.insert('f', n7);
387     trie.root = n0;
388     return trie;
389   }
390 }
391
```