CSC 212 Programming Assignment # 3
User Search History using Trees.
Due date: 28/3/2019
Bonus: +1. Plagiarism: -5.

Guidelines:	This is an **individual** assignment. The assignment must be submitted to **Web-CAT**

Trees can be used to solve many real life problems. In this assignment, you will use a tree to implement a search history data structure that stores a set of strings (history) and allows users to get all the words that begin with a given prefix.

Example 1. *If the search history tree stores {"there","their", "cat", "case", "cater", "dog", "dorm", "doll", "do", "dad"} and the user searches for "do" then he/she must get the following suggestions "dog", "dorm", "doll".*

The search history data structure has the following interface:

```
public interface SearchHistory {

  // Return the number of words stored in the search history.
  int size();

  // Add word to the search history.
  void add(String word);

  // Search for word.
  boolean findWord(String word);

  // Search for prefix
  boolean findPrefix(String prefix);

  // Return all stored words that start with prefix. If none exists, the
      method returns an empty list. If prefix is empty, all words in the
      history are returned. The order of the words is not important.
  List<String> complete(String prefix);
}
```

This data structure can be implemented in many ways, and a simple implementation that uses a list to store the words is given to you for testing purposes (class `LinkedListSearchHistory`). Your goal is to write a more efficient implementation using trees as explained in the next section.

1 A tree implementation of SearchHistory

A tree search history is a general tree (may have more than two children) where the nodes contain letters, and the words are stored along paths that start at the root. Since some words may appear as prefixes of other words, for example, "cat" and "cater", not all words must terminate at a leaf node. Therefore, a flag stored in the nodes is used to indicated the end of a word.

Example 2. *The resulting search history tree for the set words given in the previous example should look like this:*

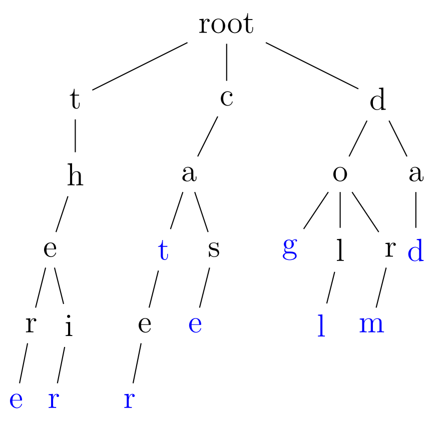

Figure 1: Example of a tree search history. The words are stored along paths that start at the root. A letter in blue indicates the end of a word.

Your goal is to implement the class `TreeSearchHistory` which uses this representation to implement the interface `SearchHistory`. Each node in search history tree is of type `THSNode` and contains a character and a list of the node's children. A Boolean is used to indicate end of word nodes (in the previous example, blue nodes are end of words).

```java
// Do not change this class
public class TSHNode {
  public List<Pair<Character, TSHNode>> children;
  public boolean end;

  public TSHNode() {
    children = new LinkedList<Pair<Character, TSHNode>>();
    end = false;
  }
}
```

Example 3. *The tree shown in Figure 1 is represented in memory using* `TSHNode` *as follows:*

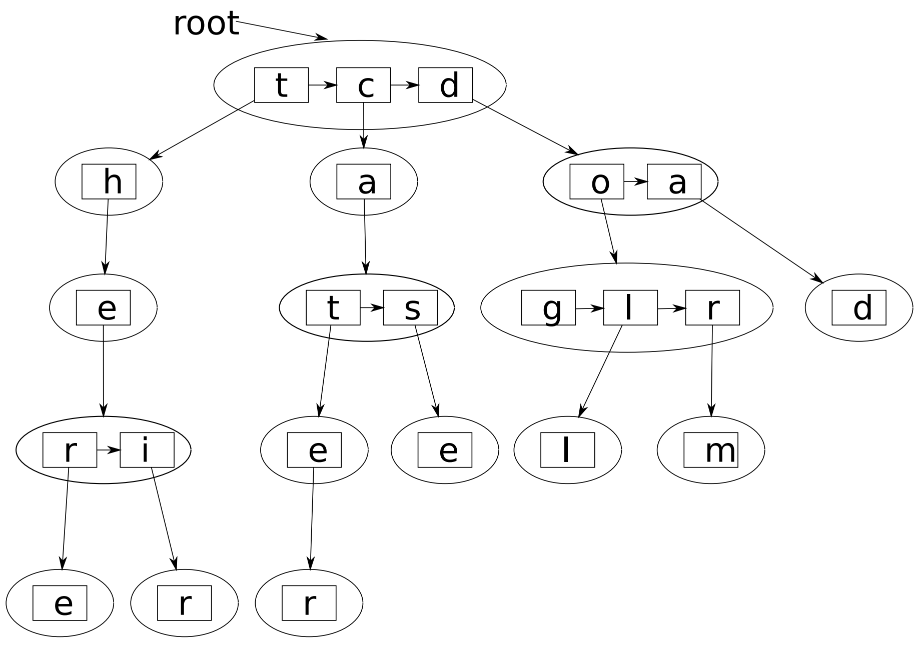

Figure 2: Representation in memory of the tree shown in Figure 1.

You must follow the representation given in the skeleton of class `TreeSearchHistory`. The member variable should remain public to allow for proper testing of your code.

```
public class TreeSearchHistory implements SearchHistory {
  public TSHNode root; // Do not change
  public int n; // Do not change

  public TreeSearchHistory() {  // Do not change
    root = new TSHNode();
    n = 0;
  }

  // Change the code staring from this point

  @Override
  public int size() {
    return 0;
  }

  @Override
  public void add(String word) {
  }

  @Override
  public boolean findWord(String word) {
    return false;
  }

  @Override
  public boolean findPrefix(String prefix) {
    return false;
  }

  @Override
  public List<String> complete(String prefix) {
    return null;
  }
}
```

2 User methods

Write a class called `SearchHistoryUtils`. In this class, you should implement the following static methods:

1. **public static int** `countWordOnly(SearchHistory h, List<String> words)`: This method takes a search history tree `h` and a list of strings `words` and returns the number of strings from `words` that appear in `h` as complete words and not as prefixes. **DO NO USE** `complete(String prefix)`;

2. **public static int** `countPrefixOnly(SearchHistory h, List<String> prefixes)`: This method takes a search history tree `h` and a list of strings `prefixes` and returns the number of strings from `prefixes` that appear in `h` as prefixes and not as complete words. **DO NO USE** `complete(String prefix)`;

3 Deliverable and rules

You must deliver:

1. Source code submission to Web-CAT. You have to upload the following classed in a zipped file:

   - `TreeSearchHistory.java`
   - `SearchHistoryUtils.java`

   Notice that you should **not upload** the interfaces `SearchHistory` and `List` and the classes `Pair`, `LinkedList` and `TSHNode`.

The submission **deadline** is: **28/03/2019**.
You have to read and follow the following rules:

1. The specification given in the assignment (**class and interface names, and method signatures**) must not be modified. Any change to the specification results in compilation errors and consequently the mark zero.

2. All data structures used in this assignment **must be implemented** by the student. The use of Java collections or any other data structures library is strictly forbidden.

3. This assignment is an individual assignment. Sharing code with other students will result in harsh penalties.

4. Posting the code of the assignment or a link to it on public servers, social platforms or any communication media including but not limited to Facebook, Twitter or WhatsApp will result in disciplinary measures against any involved parties.

5. The submitted software will be evaluated automatically using Web-Cat.

6. All submitted code will be automatically checked for similarity, and if plagiarism is confirmed penalties will apply.

7. You may be selected for discussing your code with an examiner at the discretion of the teaching team. If the examiner concludes plagiarism has taken place, penalties will apply.