Завдання 4
Autocomplete
(with Iterator pattern)

У вас є словник слів і на основі нього, по першим (починаючи з двох) літерам, має повертатись множина слів, що починаються з цих літер.

Ідея завдання взята з Programming Assignment: Autocomplete Me

В архіві знаходиться шаблон проекту, який містить інтерфейси та набір класів, які необхідно реалізувати.

Інструкція:

1. Скачайте архів з шаблоном проекту *Autocomplete* та розархівуйте його
2. Додайте його в локальний *Git* репозиторій
    a. Перейдіть в командному рядку до директорії *Autocomplete*
    b. Виконайте команди:
    > git init
    *> git add **
    > git commit -m "Autocomplete initial commit"
3. Створіть на своєму GitHub новий репозиторій з іменем за наступним шаблоном apps19<surname>-hw4
4. Знову перейдіть в директорію *Autocomplete* і виконайте наступну команду
    *> git remote add origin https://github.com/**<user_name>/apps19<surname>-hw4.git***
5. Виконайте команду
    > git push -u origin master

Після цього може відкрити проект в IDE і почати виконувати завдання

Важливо:

- не змінюйте структуру шаблоного проекту та не видаляйте файли *checkstyle.xml, pom.xml*
- при коміті проекту в локальний репозиторій (та на GitHub) необхідно щоб виключно були закомічені: *src/, checkstyle.xml, pom.xml*. Директорію *target/,* яка буде створена під час компіляції комітити не треба.

Завдання:

1. Реалізувати клас який пропонує авто-доповнення по першим (починаючи з двох) літерам слова.

Клас PrefixMatches повинен містити наступні методи:

```
public class PrefixMatches {

    private Trie trie;
```

```
    // Формує in-memory словник слів. Метод може приймати слово, рядок, масив слів/рядків. Якщо приходить рядок, то він має бути розділений на окремі слова (слова відокремлюються пробілами).
    // До словника мають додаватися лише слова довші за 2 символи. Передбачається, що у рядках які надходять знаки пунктуації відсутні.
    public int load(String... strings) { ... }

    // Чи є слово у словнику
    public boolean contains(String word);

    // Видаляє слово зі словника.
    public boolean delete(String word);

    // Кількість слів у словнику
    public int size();

    // Якщо pref довший або дорівнює 2ом символам, то повертається набір слів k різних довжин (починаючи з довжини префіксу або 3 якщо префікс містить дві літери) і які починаються з даного префіксу pref.
    // Приклад: задані наступні слова та їх довжини:
    // abc 3
    // abcd 4
    // abce 4
    // abcde 5
    // abcdef 6
     // Вказано префікс pref='abc',
    // - при k=1 повертається 'abc'
    // - при k=2 повертається 'abc', 'abcd', 'abce'
    // - при k=3 повертається 'abc', 'abcd', 'abce', 'abcde'
    // - при k=4 повертається 'abc', 'abcd', 'abce', 'abcde', 'abcdef'
    public Iterable<String> wordsWithPrefix(String pref, int k);

    // Якщо pref довший або дорівнює 2ом символам, то повертається усі слова які починаються з даного префіксу.
    public Iterable<String> wordsWithPrefix(String pref);

}
```

Для представлення in-memory словника, напишіть клас RWayTrie що реалізує интерфейс Trie. Як ідею для реалізації використайте R-way trie (чи R^R-way trie) на 26 елементів (опис: http://www.amazon.com/Algorithms-4th-Edition-Robert-Sedgewick/dp/032157351X).
або:
https://github.com/Zaxcoding/school/blob/master/CS%201501%20Algorithms/Algorithms%2C%204th%20edition%20-%20Robert%20Sedgewick%20and%20Kevin%20Wayne.pdf

```
public interface Trie {

      // Додає в Trie кортеж - Tuple: слово - term, і його вагу - weight.
      // У якості ваги, використовуйте довжину слова
       public void add(Tuple word);

      // Чи є слово в Trie
      public boolean contains(String word);

      // Видаляє слово з Trie
      public boolean delete(String word);

      // Ітератор по всім словам (обхід дерева в ширину)
      public Iterable<String> words();

      // Ітератор по всім словам, які починаються з pref
      public Iterable<String> wordsWithPrefix(String pref);

      // Кількість слів в Trie
      public int size();

}
```

Для зберігання проміжних даних при реалізації обходу в ***ШИРИНУ***, використовуйте власну чергу написану у попередньому завданні.

Шаблон проекту: autocomplete.zip

Для перевірки працездатності написаного застосування використайте список слів ftp://ftp.cs.princeton.edu/pub/cs226/autocomplete/words-333333.txt (взятий звідси ***http://www.cs.princeton.edu/courses/archive/fall13/cos226/assignments/autocomplete.html***)

2. Після коректної реалізації перелічених вище класів мають почати проходити інтеграційні тести
   PrefixMatchesITTest.testWordsWithPrefix_String()
   для методу
   PrefixMatches.wordsWithPrefix(pref)

   Та
   PrefixMatchesITTest.testWordsWithPrefix_String_and_K()
   для методу
   PrefixMatches.wordsWithPrefix(pref, k)

3. Аналогічно до попереднього завдання має бути створена нова Build Job на системі **CI Hudson/Jenkins** з іменем ***apps19<surname>-hw4*** та виконані вимоги до якості коду та його покриття тестами