Spellchecker: Proyecto Final del Taller de programación Java

En este proyecto deberán implementar un corrector ortográfico sencillo, el cual estará basado en un diccionario de palabras válidas.

Correctores ortográficos

Implemente los distintos correctores ortográficos de acuerdo a las especificaciones del archivo.

Objetivo

El objetivo final es que todos los test pasen sin ser modificados.
Implementar los tests requeridos.

Entrega

La entrega consistirá en:

- El proyecto completo, con todas las implementaciones requeridas y funcionando correctamente.
- Breve informe detallando decisiones de implementación tomadas y comparando la eficiencia de las estructuras de datos utilizadas. No es necesario que el informe cuente con introducción ni conclusiones.

No hay fecha límite para la entrega del trabajo. Una vez que decidan en qué mesa de final quieren presentar el trabajo, pedimos que lo entreguen al menos **15 días** antes de la misma para dar tiempo a las correcciones y a alguna potencial reentrega.

Introducción

El proyecto base provee un esqueleto sobre el cual se desarrollará la solución. Para esto se deben modificar los siguientes archivos:

1. `TokenScanner`. Debe parsear un archivo de texto separandolo en tokens clasificados en palabra o no palabra.
2. `Dictionary`. Debe parsear un archivo de texto con palabras validas, es decir, sin errores de ortografía.
3. `FileCorrector`, `SwapCorrector` and `Levenshtein`. Dada una palabra mal escrita, debe retornar una lista de sugerencias.
4. `SpellChecker`. Debe vincular el `TokenScanner`, `Dictionary` y algún Corrector para realizar toda la operación.

IMPORTANTE: Detalles del comportamiento esperado de cada clase se encuentran dentro de la clase.

Para cada clase, además se provee un caso de test para verificar su correcto funcionamiento.

Tokenizer

Implemente el tokenizador que separe todas las secuencias de letras.
Por ejemplo, dado el string: “`They aren't brown, are they?`” se debe generar la siguiente cadena de tokens: `"They", " ", "aren't", " ", "brown", ", ", "are", " ", "they" , "?"`.
Importante: No ignorar los espacios en blanco ni los saltos de linea. Considere que distintos sistemas operativos utilizan distintos saltos de línea. Por ejemplo: “\n” y “\r\n” son posibles saltos de línea.

Otro ejemplo:

```
It's time
2 e-mail!
```
Debe generar: `"It's", " ", "time", "\n2 ", "e", "-", "mail", "!"`

Importante: el tokenizador no es perfecto, por ejemplo, separa e-mail en 2 tokens. Esto es correcto.

Test a agregar en MyTests:

Agregue los test necesarios para los siguientes casos:

- La entrada es vacia.
- La entrada tiene una solo token palabra.
- La entrada tiene un solo token no-palabra.
- La entrada tiene los dos tipos de tokens, y termina en un token palabra.
- La entrada tiene los dos tipos de tokens, y termina en un token no palabra.

Diccionario

Implemente el diccionario. Considere que estructura de datos utilizará para almacenar el diccionario y para buscar. Por ejemplo: `ArrayList`, `HashSet`, `HashMap`, `TreeSet`. Considere sus implicancias.

De forma adicional, implemente el diccionario como un árbol de prefijos (https://en.wikipedia.org/wiki/Trie). Compare ambas implementaciones.

Test a agregar en MyTests:

Agregue los test necesarios para los siguientes casos:

- Chequear por una palabra que está en el diccionario.
- Chequear por una palabra que NO está en el diccionario.
- Preguntar por el número de palabras en el diccionario.
- Verificar que el String vacio “” no sea una palabra.
- Chequear que la misma palabra con distintas capitalizaciones esté en el diccionario.

Correctores

Implemente los tres correctores de acuerdo a las especificaciones.

- `FileCorrector`. Tiene un diccionario de palabras mal escritas y sus potenciales correcciones. Si la palabra mal escrita no está en el diccionario de correcciones, no se retorna ninguna sugerencia.
- `SwapCorrector`. Dado un diccionario, se retornan como correcciones todas las palabras que solo tienen dos letras consecutivas en distinto orden. Por ejemplo, para la palabra mal escrita “haet”, las posibles correcciones serían “hate”, “heat”
- `Levenshtein`. Dada una palabra mal escrita, sugiere palabras que están a edit distance de uno.

Tips

- Considere técnicas como Local-Sensitive Hashing para reducir el espacio de búsqueda (SwapCorrector, Levenshtein). Ver: https://en.wikipedia.org/wiki/Locality-sensitive_hashing
- Levenshtein: https://en.wikipedia.org/wiki/Levenshtein_distance

Test a agregar en MyTests:

Agregue los test necesarios para los siguientes casos para el `FileCorrector`:

- Probar un archivo con espacios extras en alrededor de las líneas o alrededor de las comas. (Se debe crear un nuevo archivo de diccionario)

- Pedir correcciones para una palabra sin correcciones.
- Pedir correcciones para una palabra con múltiples correcciones.
- Probar correcciones para palabras con distintas capitalizaciones, e.g., PaLaBar, palABAR, palabra, o PALABAR.

Agregue los test necesarios para los siguientes casos para el `SwapCorrector`:

- Proveer un diccionario null.
- Pedir correcciones para una palabra que está en el diccionario.
- Pedir correcciones para una palabra con distintas capitalizaciones.

Implemente el SpellChecker

Implemente el `SpellChecker` y pruébelo corriendo el `SpellCheckerRunner` desde el IDE.