Autómatas y Lenguajes Formales

Proyecto de prácticas
Curso 2023-2024 - Convocatoria de Diciembre

Profesor : Juan Antonio Sánchez Laguna

Grupo de prácticas: 2.3

Autores:
Juan Jesús Ortiz García
Gonzalo Vicente Pérez

Índice

1. Manual de usuario

La aplicación consiste en un silabeador con múltiples opciones. Primero se muestra un mensaje de bienvenida junto a las opciones disponibles (1: Silabear, 2: Clasificar entonación, 3: Obtener rimas, 666: Salir y guardar).

La primera opción, que se puede activar si se pulsa 1, permite al usuario ingresar una palabra, para luego descomponerla en sus respectivas sílabas.

La segunda opción, activada con el número 2, clasifica la palabra según su entonación en aguda, llana o esdrújula, según donde recaiga la vocal tónica en la palabra.

La tercera opción, seleccionada con la tecla del número 3, sirve para encontrar palabras en el diccionario que rimen con la palabra ingresada, ofreciendo dos tipos de rimas: asonante y consonante.

Además, el programa tiene la capacidad de guardar y recuperar información previamente procesada, permitiendo al usuario acceder a un historial de palabras analizadas y sus respectivas rimas. Esto se realiza mediante un archivo *salida.csv*, que se actualiza con cada nueva ejecución del programa.

Para usar la aplicación, se deben seguir los siguientes pasos:

Si se elige la opción 1, 2 o 3, se pedirá ingresar una palabra. Después de ingresarla y presionar Enter, el programa procesará la solicitud y mostrará los resultados.

Para salir del programa y guardar los datos en 'salida.csv', se debe escribir el número 666. Esto cerrará el programa y guardará automáticamente el historial de palabras en el archivo mencionado anteriormente.

2. Aspectos principales

2.1. Resolución del silabeo

Para lograr hacer la función `silabear`, usamos expresiones regulares para dividir las palabras en sílabas siguiendo las reglas fonéticas del boletín. Para cada caso (y subcaso), hemos definido una regla de modo que se busque la presencia de una de estas reglas en una palabra. Es importante el orden empleado en la regla compuesta, porque no sigue el orden del boletín, sino que comprueba antes la existencia de la regla 6 (triptongos) para evitar que el programa se meta a la regla de diptongos antes (puesto que en un triptongo también existen dos vocales juntas).

Sin embargo, encontramos un problema con el tratamiento de la 'y', pues hay veces que actúa como vocal y otras como consonante. Primero intentamos meterla en las expresiones regulares, pero vimos que había veces que se trataba erróneamente. Por ello, definimos una función llamada `y_vocal`, que se aplica a toda palabra para el tratamiento de las "y" que se comportan como una vocal, sustituyéndo la "y" con comportamiento vocal por una C cedilla (pues este caracter no está en las reglas, y así nos aseguramos que no hacía *match*).

Una vez tuvimos las reglas definidas, tuvimos problemas para conseguir trocear la palabra en sílabas. Finalmente, con la ayuda de regex101, lo que hicimos fue, buscar los números de los grupos que hacían *match* con el final de la sílaba y el inicio de la siguiente sílaba en cada regla, y poner un guión entre ambos grupos, de manera que el guión simbolizaba la separación silábica (es decir, nuestra función si recibe "uno", la transforma en "u-no"). Así, en la próxima iteración del bucle, la palabra analizada tendrá un guión entre sílabas, y el programa no conseguirá hacer *match* con la sílaba ya analizada, pasando a la siguiente. En el caso de los diptongos, como teníamos que modificar la palabra, lo que hicimos fue insertar una coma, pues así conseguimos modificar la sílaba para que no hiciera *match*, pero creando una diferencia, pues en este caso no son sílabas diferentes.

Una vez que se termina el análisis, la palabra realmente tiene guiones y comas de por medio, quizá incluso una 'ç' por lo que definimos una función llamada `guiones_a_lista`, que recibe la cadena y devuelve la lista de sílabas. Lo que hace es eliminar las comas (pues era la separación para diptongos, que no son sílabas diferentes), cambiar la 'ç' por 'y' otra vez, y eliminar el guión que se pueda llegar a quedar al final de la palabra. Así, la palabra ya solo tiene guiones, que simbolizan la separación entre sílabas, por lo que se puede recorrer la cadena e introducir las sílabas en una lista, que es lo que devuelve la función principal.

2.2. Resolución de la entonación

Respecto a la tarea de encontrar la vocal tónica, dividimos el problema en hacer dos funciones. Una de ellas, `buscar_tilde`, recibe la lista de sílabas y busca con una sencilla expresión regular si hay tildes en la palabra (de modo que no haya que aplicar ninguna regla más, pues el problema se reduce simplemente a encontrar la tilde), y en caso de haberlas pone la letra en mayúscula y devuelve “true” en señal de que la palabra tenía tilde.

En caso de no haberlas, será la función `vocal_tonica` la que buscará la tónica (eso sí, dentro de una sílaba). Por tanto, para que esta función pueda ejecutarse correctamente hicimos otra función, `busca_silaba`, que encuentra la sílaba tónica y devuelve el número de su sílaba. Como todas las palabras esdrújulas llevan tilde, entonces esta regla simplemente comprueba si la palabra termina en a, e, i, o, u, s, n, en cuyo caso estamos ante una palabra llana (porque si fuera aguda llevaría tilde) y aguda en el caso contrario (no definimos expresiones regulares aquí puesto que era una comprobación muy sencilla). Una vez pasada la sílaba tónica, entonces se detecta mediante las reglas del Anexo 2 cuál es la vocal tónica dentro de la sílaba. De nuevo en estas reglas vuelve a importar mucho el orden de la regla compuesta, yendo primero la que incluye triptongos para evitar que se entre en la regla de los diptongos y en la de vocal solitaria.

Estas tres funciones previas se juntan en una única función llamada `entonacion`. Esta función llama primero a `buscar_tilde`, y si la palabra tiene tilde entonces finaliza. De no tenerla, entonces se llama a `busca_silaba` y finalmente a `vocal_tonica` con el resultado de esta, se cambia la vocal tónica por su mayúscula, y se devuelve la lista otra vez. Cabe destacar que estas funciones tratan bien el caso en el que la palabra es simplemente una “y”, pues en ese caso la letra sería la vocal tónica, por mucho que sea una consonante.

Por último, la función `clasifica_entonacion`, recibe la lista con la tónica en mayúscula, y dependiendo de en qué posición de la lista esté la mayúscula, devuelve si es aguda, llana, o esdrújula.

2.3. Resolución de la rima

Para la resolución de la rima, hicimos dos funciones, que básicamente reciben la lista que devuelve `entonacion` (con la vocal tónica ya en mayúsculas) y calculan la terminación de la palabra según el tipo de rima. `rimalizer_asonante`, una vez encuentra la mayúscula, calcula la entonación empezando por la mayúscula, y añadiendo solo las vocales. `rimalizer_consonante`, la calcula empezando por la mayúscula, y añadiendo todo lo que haya detrás de ella. Así, sabremos que una palabra rima si sus terminaciones coinciden. El manejo de las terminaciones en el diccionario se explica en el siguiente apartado.

2.4. Programa principal, manejo del .csv y demás

El programa principal comienza con una serie de prints, que simbolizan el menú principal. Luego se declaran 3 diccionarios vacios, uno en el que se almacena toda la información de una palabra (silabas, entonación, y terminaciones) y dos para las rimas, en el que las terminaciones serán las claves, y los valores las palabras (si hay varias palabras con la misma terminación aparecen separadas por comas).

Posteriormente se trata de leer el .csv, si lo hubiera. Para ello, dentro de un try-except, se abre en modo lectura el fichero “salida.csv”, y va recorriendo línea a línea su contenido. Para recuperar los 4 campos del diccionario, definimos expresiones regulares, pues era la manera más directa de recuperar el contenido que queríamos. Posteriormente, vamos insertando en D (el diccionario principal), la información de cada palabra (la clave se recupera troceando la línea usando el ‘:’ como separador, pues la clave lleva detrás ese caracter). Además, con los campos ‘term_aso’ y ‘term_conso’ del diccionario, se reconstruyen también los diccionarios de rimas, en los que las claves son las terminaciones y los valores las palabras. En caso de que no haya .csv, el except incluye una línea que informa de ello. Posteriormente se entra en el bucle, en el que se pide la opción, y si es 1, 2 ó 3, analiza la palabra por completo y guarda la información en los diccionarios. Posteriormente, según la opción elegida, muestra la información pedida (aunque realmente se calcula todo). Si la opción es 666, se abre el fichero “salida.csv” en modo escritura, y se guarda en cada línea la clave con el diccionario correspondiente. Se imprime un ¡Adiós! y se sale del bucle.

3. Conclusiones

Ha sido un proyecto que hemos encontrado entretenido de hacer, porque realmente hemos visto las importantes aplicaciones que pueden tener las expresiones regulares, algo de lo que a simple vista y sin este proyecto no nos hubiéramos dado cuenta. No ha habido ningún momento de la práctica en el que nos hayamos sentido muy atascados, sino que más bien lo hemos visto como pequeños retos que ir superando de forma constante, pues los ejercicios con expresiones regulares a veces se hacen largos y pesados de controlar, pero no difíciles. Quizás donde más atascados llegamos a estar fue en la función de silabeo, donde tuvimos algún problema con las expresiones regulares, el orden de estas y como conseguir separar la palabra en sílabas, pero que pudimos llegar a superar sin mayor problema.
Nuestra experiencia con Python, que era bastante escasa antes de este proyecto, tampoco ha sido mala del todo, y si bien en este tiempo que llevamos en la carrera hemos estado acostumbrados a usar lenguajes fuertemente tipados, no ha sido muy difícil adaptarse y el hecho de no conocer el lenguaje en profundidad no nos ha supuesto ningún problema a la hora de llevar a cabo el proyecto.

Un aspecto que creemos que podría cambiarse es quizás alargar las sesiones guiadas una o dos semanas más, de modo que sólo hubiese un ejercicio optativo, porque aunque es verdad que hasta el fin de las sesiones guiadas no hemos tenido grandes incompatibilidades con otros exámenes, sí que una vez terminadas no hemos tenido tiempo para implementar las dos mejoras (que es lo que hubiéramos querido) por tener parciales y otras entregas. Además, creemos que hay algunas sesiones, como en la que usamos EditPad Pro, que podrían acortarse para dar más importancia a las que realmente son más difíciles. No obstante, en general, ha sido un trabajo que no ha sido agobiante, del que no tenemos muchas pegas y que nos ha gustado bastante hacer.