**Memoria del Trabajo Realizado**

**Proyecto: Búsqueda Aproximada de Cadenas con Distancias de Edición**

**Descripción del Proyecto**

El objetivo principal del proyecto ha sido implementar un sistema que permita realizar búsquedas aproximadas de cadenas basado en diferentes distancias de edición, integrándolo en un motor de búsqueda existente. El trabajo se ha dividido en dos partes: las **tareas obligatorias** requeridas para el funcionamiento básico y las **ampliaciones opcionales** para mejorar la funcionalidad y la flexibilidad del sistema.

**Reparto de Trabajo**

El trabajo ha sido realizado por un equipo de **4 personas**, dividiendo las responsabilidades de la siguiente manera:

1. **Parte Obligatoria**: Dos integrantes han trabajado en las implementaciones básicas.
   * **Integrante 1**: Implementación de las primeras dos tareas obligatorias:
     + Recuperar la secuencia de operaciones de edición para Levenshtein.
     + Implementación de Levenshtein con reducción de coste espacial.
   * **Integrante 2**: Implementación de las dos últimas tareas obligatorias:
     + Incorporación del umbral (threshold) en Levenshtein.
     + Uso de cotas optimistas para ahorrar cálculos.
2. **Ampliaciones Opcionales**: Los otros dos integrantes han trabajado en la extensión de las funcionalidades.
   * **Integrante 3**: Implementación de Damerau-Levenshtein Restringido.
     + Cálculo con matriz y reducción de coste espacial.
     + Recuperación de la secuencia de operaciones.
   * **Integrante 4**: Implementación de Damerau-Levenshtein Intermedia.
     + Cálculo con matriz, reducción de coste espacial y recuperación de la secuencia de operaciones.

**Tareas Desarrolladas**

**Parte Obligatoria**

1. **Recuperar la Secuencia de Operaciones de Edición para Levenshtein**
   * Se implementó un método para reconstruir la secuencia de operaciones necesarias para transformar una cadena en otra utilizando la distancia de Levenshtein. Esto incluye operaciones como inserciones, eliminaciones, sustituciones y coincidencias.
   * Decisión clave: Usar una matriz de distancias preexistente para retroceder desde el resultado final y construir la secuencia.
2. **Implementación de Levenshtein con Reducción de Coste Espacial**
   * Se optimizó la implementación de Levenshtein para usar dos vectores en lugar de una matriz completa, reduciendo significativamente el uso de memoria.
   * Decisión clave: Mantener la misma lógica de cálculo pero adaptarla a un enfoque basado en vectores.
3. **Añadir Umbral (threshold) para Reducir el Coste Computacional**
   * Se agregó la capacidad de detener el cálculo de Levenshtein si la distancia supera un umbral predefinido.
   * Decisión clave: Implementar esta funcionalidad directamente sobre la versión optimizada con reducción de coste espacial.
4. **Uso de Cotas Optimistas**
   * Se desarrolló un método para calcular una cota optimista de la distancia entre cadenas, basada en la frecuencia de caracteres. Si la cota superaba el umbral, se evitaba realizar cálculos adicionales.
   * Decisión clave: Utilizar estructuras de conteo como Counter para manejar eficientemente las diferencias de caracteres.

**Ampliaciones Opcionales**

1. **Implementación de Damerau-Levenshtein Restringido**
   * Se implementó el cálculo de Damerau-Levenshtein permitiendo transposiciones de dos caracteres consecutivos (ab ↔ ba).
   * Decisión clave: Incorporar transposiciones en el cálculo sin afectar la eficiencia del resto de las operaciones básicas.
2. **Recuperación de la Secuencia de Operaciones para Damerau-Levenshtein Restringido**
   * Se desarrolló un método para identificar las transposiciones realizadas y construir la secuencia de edición.
   * Decisión clave: Usar una estructura similar a la recuperación de Levenshtein, añadiendo casos para transposiciones.
3. **Implementación de Damerau-Levenshtein Intermedia**
   * Se extendió Damerau-Levenshtein para permitir transposiciones más complejas, como acb ↔ ba y ab ↔ bca.
   * Decisión clave: Usar una matriz de distancias y gestionar las dependencias entre múltiples caracteres.
4. **Recuperación de la Secuencia de Operaciones para Damerau-Levenshtein Intermedia**
   * Se implementó la recuperación de operaciones incluyendo transposiciones complejas. Esto permitió identificar las secuencias exactas de modificaciones.
   * Decisión clave: Utilizar un enfoque basado en matrices para rastrear transposiciones de tres caracteres.

**Decisiones Técnicas y Herramientas Utilizadas**

* **Lógica Principal**: La implementación de las distancias se basó en matrices y vectores para manejar la reducción de coste espacial.
* **Librerías Usadas**:
  + numpy para manejar las matrices y vectores en los cálculos de distancia.
  + collections.Counter para las cotas optimistas.
* **Integración**:
  + Las funciones desarrolladas se integraron en el diccionario opcionesSpell para facilitar su uso en el motor de búsqueda.
  + Se modificaron spellsuggester.py y ALT\_Searcher.py para incluir las nuevas funcionalidades, permitiendo seleccionar las distancias desde la línea de comandos.

**Resultados Obtenidos**

1. **Parte Obligatoria**:
   * El sistema ahora puede realizar búsquedas aproximadas utilizando Levenshtein optimizado.
   * Las cotas optimistas mejoran el rendimiento al evitar cálculos innecesarios.
2. **Ampliaciones Opcionales**:
   * La funcionalidad de Damerau-Levenshtein Restringido e Intermedia añade mayor precisión y flexibilidad en la búsqueda aproximada, manejando casos de transposiciones más complejas.