**1. Introducción**

La distancia de edición es una medida que representa el número mínimo de operaciones necesarias para transformar una cadena en otra. Se utiliza ampliamente en áreas como la bioinformática, el procesamiento de texto y la inteligencia artificial, ya que permite cuantificar las diferencias entre secuencias de datos. Tradicionalmente, las operaciones permitidas para calcular la distancia de edición incluyen inserciones, eliminaciones y sustituciones, cada una con un costo asociado. En este informe, abordaremos una extensión de este problema, donde también se considera la operación de transposición (intercambio de dos caracteres adyacentes) y costos variables para cada operación.

El objetivo de esta tarea es implementar y analizar dos algoritmos que calculen la distancia de edición extendida entre dos cadenas dadas. Estos algoritmos se basan en dos enfoques diferentes:

1. **Fuerza Bruta**: Este enfoque explora exhaustivamente todas las posibles secuencias de operaciones para encontrar el costo mínimo. Aunque es efectivo en problemas pequeños, su complejidad puede crecer exponencialmente, lo que lo hace poco eficiente para entradas grandes.
2. **Programación Dinámica**: Este enfoque optimiza el cálculo de la distancia de edición almacenando soluciones parciales y evitando cálculos redundantes. La programación dinámica es particularmente útil para problemas de optimización como el de la distancia de edición, ya que permite reducir significativamente el tiempo de ejecución en comparación con la fuerza bruta.

A través de estos dos algoritmos, se busca no solo calcular la distancia mínima de edición entre dos cadenas, sino también analizar cómo la inclusión de transposiciones y costos variables afecta la complejidad y el comportamiento de los algoritmos. Para esto, se llevarán a cabo pruebas experimentales con diferentes configuraciones de cadenas de entrada y costos, lo que permitirá observar la eficiencia y precisión de cada enfoque en distintos escenarios.

La estructura de este informe es la siguiente: en la sección 2, se presenta el diseño detallado de ambos algoritmos; en la sección 3, se describe la implementación en C++; en la sección 4, se muestran los experimentos realizados y sus resultados; y finalmente, en la sección 5, se exponen las conclusiones del análisis.

2. Diseño y Análisis de Algoritmos

Procedure

3. Implementaciones

Para la fuerza bruta:

El programa necesita cost\_delete.txt, cost\_insert.txt, cost\_replace.txt y cost\_transpose.txt que serian los archivos de los costos cada operación, y al ejecutarse crea un archivo palabras.txt que contiene el costo de pasar la primera palabra a la segunda, costo de pasar la segunda a la primera y el costo mínimo.

Primero, se cargan matrices y vectores globales desde archivos dados para obtener los costos de estas operaciones. Luego, se usa una función calcularCosto que evalúa el costo de transformar una palabra en otra, considerando costos específicos de sustitución, inserción, eliminación, y transposición entre letras. Esta función primero intenta la transposición de letras, porque se toma el supuesto que hacer una trasposición es la operación menos costosa, para eso llama a costo\_trans que simplemente retorna el costo de hacer esa operación buscando la posición en la matriz. Si no se puede hacer una transposición se hace sustitución, para eso se llama a costo\_sub que busca en la matriz el costo de la operación. El proceso se hace letra a letra hasta que se alcanza el largo de la palabra más corta. Cuando finaliza se ajusta la palabra con costo\_ins o costo\_del según si se tiene que acortar o largar la palabra, estas funciones mencionadas funcionan buscando en su respectivo vector el costo de la letra la cual se quiera insertar o eliminar. Después de haber realizado el proceso de la primera palabra a la segunda se hace exactamente lo mismo, pero en sentido contrario para finalmente comparar cual es el costo mínimo entre las direcciones.

**5. Conclusiones**

En esta tarea, se implementaron y compararon dos algoritmos para calcular la distancia mínima de edición extendida entre dos cadenas: uno basado en fuerza bruta y otro en programación dinámica. Ambos algoritmos fueron diseñados para manejar costos variables en las operaciones de inserción, eliminación, sustitución y transposición, lo que añadió complejidad al problema y permitió evaluar su rendimiento en escenarios más realistas y aplicables en el mundo real.

El enfoque de **fuerza bruta** demostró ser adecuado para casos simples, permitiendo explorar todas las posibles secuencias de operaciones y garantizando la obtención de la solución óptima. Sin embargo, su complejidad exponencial lo hace inviable para cadenas de gran longitud, ya que el tiempo de ejecución aumenta drásticamente con el tamaño de las cadenas.

Por otro lado, el enfoque de **programación dinámica** resultó significativamente más eficiente en términos de tiempo de ejecución y uso de recursos, especialmente en casos de entrada de tamaño considerable. Este enfoque optimiza el cálculo almacenando soluciones de subproblemas intermedios, lo cual reduce los cálculos redundantes y permite una resolución más rápida y escalable del problema. La inclusión de transposiciones y costos variables se manejó de forma efectiva en la matriz de programación dinámica, lo cual permite que el algoritmo se adapte fácilmente a diferentes configuraciones de costos.

En resumen, los resultados muestran que la programación dinámica es el enfoque más adecuado para problemas de edición de cadenas con configuraciones complejas, debido a su balance entre precisión y eficiencia. Este trabajo destaca la importancia de seleccionar el paradigma de diseño de algoritmos adecuado según la naturaleza del problema, y aporta una base sólida para aplicaciones en las que es esencial minimizar los costos de edición en secuencias de datos.

AMD Ryzen 5 3500U, 2.10 GHz, 8GB RAM, Almacenamiento SSD

SO Windows 11 Home Single Language, g++ 14.1.0