REPORTE TAREA 2 y 3
ALGORITMOS Y COMPLEJIDAD

«Explorando la Distancia entre Cadenas, una Operación a la Vez»

Francisco Rebolledo C.

18 de noviembre de 2024

02:54

Resumen

En este reporte se presenta un análisis de algoritmos sobre el problema de la distancia mínima de edición, haciendo especial énfasis en la diferencia de la complejidad temporal y en memoria que pueden presentar los enfoques de Fuerza Bruta y Programación Dinámica dentro de un mismo problema, esto mediante la directa implementación de dichos enfoques en una infraestructura de programas que abordan distintos tipos de casos con la misión de obtener los resultados de la forma más precisa y limpia posible para comprobar de forma práctica la importancia que tiene el análisis y diseño de algoritmos en la resolución de problema complejos mediante las ciencias de la computación y como la forma de dar solución a un problema mediante distintos enfoques de algoritmo puede cambiar su complejidad.

Índice

1. Introducción

Hace años que la computación es una de las mejores herramientas para la resolución de problemas en diversas áreas debido a que nos permite procesar grandes volúmenes de datos rápidamente, automatizar tareas y encontrar soluciones optimas que manualmente resultaría imposible. Dentro de las ciencias de la computación una de las ramas mas importantes es el **Análisis y Diseño de Algoritmos**, ya que permite desarrollar métodos eficientes para resolver problemas complejos.

Este trabajo se enfoca en comparar estos dos enfoques, considerando su complejidad temporal y espacial, y analizando cómo la naturaleza del problema y las características de las cadenas de entrada, como su simetría o asimetría, afectan el rendimiento de los algoritmos. El análisis de estos dos enfoques no solo proporciona una visión clara sobre sus diferencias fundamentales, sino también sobre cómo cada uno se comporta al enfrentar el problema bajo distintas condiciones y restricciones.

2. Diseño y Análisis de Algoritmos

El problema de la **Distancia Mínima de Edición** tiene varias formas de ser abordado, las cuales pueden tener una gran diferencia en la eficiencia de la memoria o los tiempos de ejecución de los algoritmos, por lo mismo, aquí se presentan dos aproximaciones a los algoritmos que dan solución a dicha problemática, en especifico para los paradigmas **Fuerza Bruta** y **Programación Dinámica** junto a su respectivo análisis teórico de la complejidad y cómo su estructura afecta a la eficiencia del mismo. Por lo mismo, antes de entrar directamente en el análisis de los algoritmos resulta importante definir cada paradigma para tener en cuenta cuáles son sus principales diferencias y como esto puede llegar a afectar a la solución que dan al problema.

El enfoque de **Fuerza Bruta** consiste en resolver un problema evaluando todas las posibles soluciones de manera exhaustiva, sin intentar optimizar el proceso de búsqueda. Es una estrategia directa que garantiza encontrar la solución correcta, aunque no sea eficiente en términos de tiempo y recursos. Mientras que, la **Programación Dinámica** utiliza un enfoque diseñado para resolver problemas que pueden descomponerse en subproblemas más pequeños y que exhiben las propiedades de solapamiento de estos para llegar a una solución del problema general.

Estos enfoques buscaran abordar el problema de la **Distancia Mínima de Edición** una letra a la vez pero se separan de tal forma que, por fuerza bruta el algoritmo deberá calcular y almacenar cada una de las 4 operaciones fundamentales (insertar, borrar, reemplazar y transponer) por cada letra, creando un crecimiento exponencial al buscar cada combinación probable, mientras que el algoritmo con enfoque de programación dinámica identificará y almacenará los costos de edición de cadenas mas cortas dentro de la cadena principal para armar en base a estas el costo de la cadena principal.

Por último, también resulta importante analizar la naturaleza del problema, pues el hecho de que cada operación tenga **costos variables** en base al o los caracteres involucrados agrega una capa de complejidad a los algoritmos que deben trabajar para decidir qué operación es la más adecuada en cada caso, lo que puede aumentar el tiempo de ejecución de los mismos dadas las operaciones necesarias para el procedimiento. Por lo mismo, agregar la **operación de transposición** aumenta las posibilidades de los caminos a tomar y variables que tener en cuenta, lo que incrementa aun más la complejidad del problema pero especialmente representa una carga en el enfoque de Fuerza Bruta al considerar exhaustivamente todas las formas de modificar la cadena.

2.1. Fuerza Bruta

2.1.1. Descripción de la solución

El algoritmo diseñado funciona de forma recursiva para abordar una cantidad mayor de casos sin colapsar la memoria del computador pero sin abandonar el principio de fuerza bruta al buscar exhaustivamente entre todas las posibilidades generadas en el árbol recursivo, la base del algoritmo es ir letra por letra evaluando las 4 operaciones y repetir por cada una para la letra siguiente hasta llegar a transformar la cadena1 en la cadena2 y luego buscar cual tuvo el mínimo costo.

2.1.2. Complejidad Temporal y Espacial

Dada la forma del problema, las complejidades están directamente relacionadas al largo de las cadenas, por lo que **a** y **b** son respectivamente el número de caracteres para cadena1 y cadena2. Definidos estos valores, **n** representa todos los posibles valores, por lo que se asume que sera la cadena más larga. Para el análisis de la complejidad temporal y espacial, se verifica que la función posee un factor de recurrencia de 4 y va guardando todos los posibles valores, por lo que se obtienen los valores:

Complejidad Temporal	**Complejidad Espacial**
$T(n) = O(4^n)$	$E(n) = O(n))$

2.1.3. Pseudocódigo del algoritmo utilizando fuerza bruta

Algoritmo 1: DME_Fuerza_Bruta

Procedure DME_Fuerza_Bruta(cadena1, cadena2, i, j)
if $i == 0$ **then**
    return $j \cdot$ costo_ins(cadena2[$j-1$])
if $j == 0$ **then**
    return $i \cdot$ costo_del(cadena1[$i-1$])
if *cadena1*[$i-1$] == *cadena2*[$j-1$] **then**
    return DME_Fuerza_Bruta(cadena1, cadena2, $i-1, j-1$)
$insertar \leftarrow$ DME_Fuerza_Bruta(cadena1, cadena2, $i, j-1$) + costo_ins(cadena2[$j-1$])
$eliminar \leftarrow$ DME_Fuerza_Bruta(cadena1, cadena2, $i-1, j$) + costo_del(cadena1[$i-1$])
$sustituir \leftarrow$
    DME_Fuerza_Bruta(cadena1, cadena2, $i-1, j-1$) + costo_sub(cadena1[$i-1$], cadena2[$j-1$])
$transponer \leftarrow \infty$
if $i > 1$ $j > 1$ *cadena1*[$i-1$] == *cadena2*[$j-2$] *cadena1*[$i-2$] == *cadena2*[$j-1$] **then**
    $transponer \leftarrow$
     DME_Fuerza_Bruta(cadena1, cadena2, $i-2, j-2$) + costo_trans(cadena1[$i-2$], cadena1[$i-1$])
return mín($\{insertar, eliminar, sustituir, transponer\}$)

2.2. Programación Dinámica

2.2.1. Descripción de la solución

De forma similar, pero en complejidad mucho menor al enfoque de fuerza bruta, la solución es plantear la recursividad de la función principal pero con la diferencia de guardar los casos intermedios . Construye una matriz de tamaño (m+1)×(n+1), donde m y n son las longitudes de las cadenas, para almacenar los costos mínimos de transformar los primeros i caracteres de una cadena en los primeros j de la otra. Inicializa los casos base para manejar transformaciones desde o hacia cadenas vacías, y luego llena la matriz evaluando las operaciones de insertar, eliminar, reemplazar y, si es posible, transponer caracteres. El costo mínimo para cada operación se calcula acumulativamente, evitando cálculos redundantes. Finalmente, el valor en la celda (m,n) de la matriz representa el costo mínimo total.

2.2.2. Relación de recurrencia

$$matriz[i][j] = \text{mín} \begin{cases} matriz[i-1][j] + \text{costo de eliminación} \\ matriz[i][j-1] + \text{costo de inserción} \\ matriz[i-1][j-1] + \text{costo de sustitución} \\ matriz[i-2][j-2] + \text{costo de transposición} \end{cases}$$

2.2.3. Identificación de subproblemas

los subproblemas son las distancias mínimas de edición entre todos los prefijos de las dos cadenas. Específicamente, los subproblemas corresponden a calcular la distancia mínima de edición entre los primeros i caracteres de la primera cadena y los primeros j caracteres de la segunda cadena. La solución final se obtiene calculando la distancia entre las cadenas completas, es decir, entre los primeros m caracteres de la primera cadena y los primeros n caracteres de la segunda cadena.

2.2.4. Complejidad Temporal y Espacial

Para obtener las complejidades con un enfoque de programación dinámica resulta relevante el largo de las 2 cadenas para el tamaño de la matriz de memoria que se creará en el proceso, en especifico m es la longitud de la primera cadena y n es la longitud de la segunda cadena. El algoritmo llena una matriz de dimensiones m+1 por n+1, y cada celda de la matriz se calcula en tiempo constante de forma que cada celda almacena un subproblema. Así se puede verificar que:

Complejidad Temporal	**Complejidad Espacial**
$T(n) = O(m * n)$	$E(n) = O(m * n))$

2.2.5. Algoritmo utilizando programación dinámica

Algoritmo 2: DME_Programacion_Dinamica

```
1  Procedure DME_Programacion_Dinamica(cadena1, cadena2)
2  m ← len(cadena1)
3  n ← len(cadena2)
4  Crear matriz matriz de tamaño (m+1) × (n+1) inicializada con ceros
5  for i ← 0 to m do
6      matriz[i][0] ← i · costo_del(cadena1[i−1])
7  for j ← 0 to n do
8      matriz[0][j] ← j · costo_ins(cadena2[j−1])
9  for i ← 1 to m do
10     for j ← 1 to n do
11         if cadena1[i−1] == cadena2[j−1] then
12             matriz[i][j] ← matriz[i−1][j−1]
13         else
14             matriz[i][j] ← mín(
15                 matriz[i−1][j] + costo_del(cadena1[i−1]),
16                 matriz[i][j−1] + costo_ins(cadena2[j−1]),
17                 matriz[i−1][j−1] + costo_sub(cadena1[i−1], cadena2[j−1])
18             )
19         if i > 1 j > 1 cadena1[i−1] == cadena2[j−2] cadena1[i−2] == cadena2[j−1] then
20             matriz[i][j] ← mín(
21                 matriz[i][j],
22                 matriz[i−2][j−2] + costo_trans(cadena1[i−1], cadena1[i−2])
23             )
24 return matriz[m][n]
```

3. Implementaciones

Todos los códigos y dependencias utilizados se encuentran en el siguiente enlace de Github, además, las instrucciones para trabajar con los programas deben realizarse en dicho directorio por consola.

https://github.com/sPyKeRT1/FranciscoRebolledo_Tarea2_3/tree/main/codigos

Para trabajar de mejor forma los programas de C++ y facilitar su uso junto al generador de casos de prueba en Python se utiliza la herramienta Makefile bajo las siguientes instrucciones:

- **make all**: Compila los archivos de C++ para posteriormente ejecutarlos junto a los demás.
- **make run**: Ejecuta los programas en el orden Generador->Programación Dinámica->Fuerza Bruta.
- **make clear**: Elimina los archivos objetivo necesarios para ejecutar los programas de C++.

La estructura general que da soporte para implementar las funcionalidades esperadas para los algoritmos consta de 5 archivos de texto para guardar por separado las palabras en **'cadenas.txt'** y las 4 tablas de costos para las operaciones insertar en **'cost_insert.txt'**, borrar en **'cost_delete.txt'**, reemplazar en **'cost_replace.txt'** y transponer en **'cost_transpose.txt'**; además, se implementa **'generador.py'** para crear un caso de prueba en base a los parámetros que se le entreguen por entrada estándar y junto a esto, se agregan los archivos **'funcostos.h'** y **'funcostos.cpp'** donde se definen e implementan las funciones que obtienen los costos directamente desde las tablas antes mencionadas para los algoritmos.

Al implementar los algoritmos en **'progfuerzabruta.cpp'** y **'progdinamica.cpp'** se busca mantener los algoritmos lo más limpios posible para que no deban realizar tareas de entrada o salida de datos fuera del llamado a las funciones de costo, por lo que, primero leen desde el archivo 'cadenas.txt' las cadenas de caracteres a trabajar y las pasan como parámetro a las funciones **DME_Fuerza_Bruta(cadena1, cadena2, tamaño_cadena1, tamaño_cadena2)** y **DME_Programacion_Dinamica(cadena1, cadena2)** de forma respectiva, las cuales como se mostró anteriormente ejecutan el algoritmo correspondiente y devuelven la distancia mínima de edición para que después la función main se encargue de mostrar los datos obtenidos del proceso por pantalla y finalizar su ejecución.

4. Experimentos

Para la replicación de este experimento es crucial la reproducibilidad de los resultados por lo que a continuación se detallan las especificaciones del equipo utilizado a nivel de hardware y software:

- Procesador: Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz 2.50 GHz
- RAM: 16,0 GB DDR4
- Almacenamiento: SK hynix BC511 HFM512GDJTNI-82A0A (SSD)
- Sistema Operativo: Edición Windows 10 Home Single Language, Versión 22H2, Compilación del SO 19045.5131, Experiencia Windows Feature Experience Pack 1000.19060.1000.0
- Compilador: g++.exe (Rev3, Built by MSYS2 project) 14.1.0

4.1. Dataset (casos de prueba)

Para verificar y probar correctamente la funcionalidad de los algoritmos ya sea de Fuerza Bruta o Programación Dinámica aparte de usar los casos aleatorios normales es importante obtener resultados con casos que posean ciertas características distintivas o limites respecto a los demás, es por ello que los 5 principales serán casos con cadenas con caracteres vacíos, cadenas con caracteres repetidos, cadenas simétricas, cadenas asimétricas y donde las matrices tienen un mismo valor para todas las operaciones. (Estos datasets pueden ser encontrados en sus respectivas carpetas dentro del repositorio o en el archivo)

1. Caso con cadenas vacías:

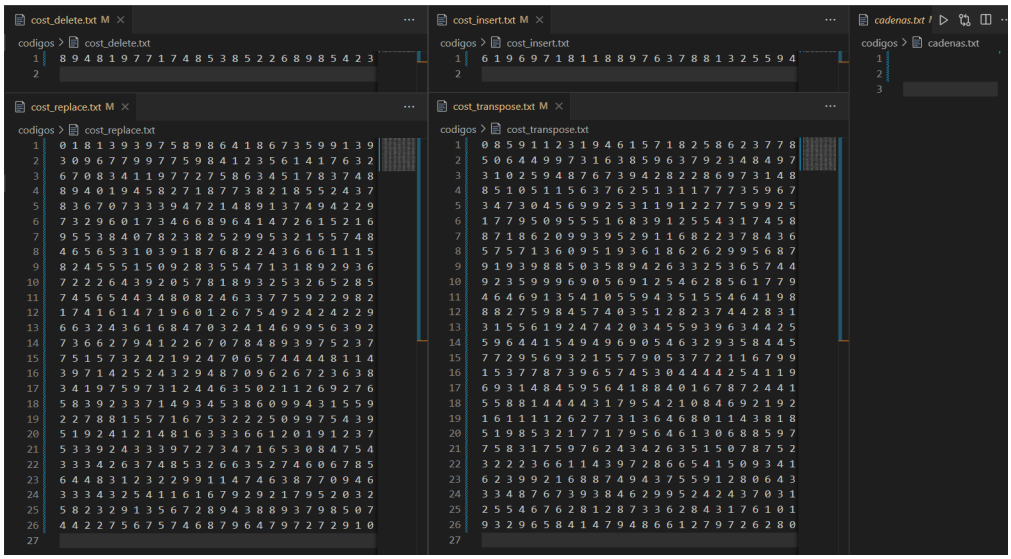

Figura 1: Archivos con las tablas y cadenas necesarias para el caso limite 1

2. Caso con cadenas repetidas:

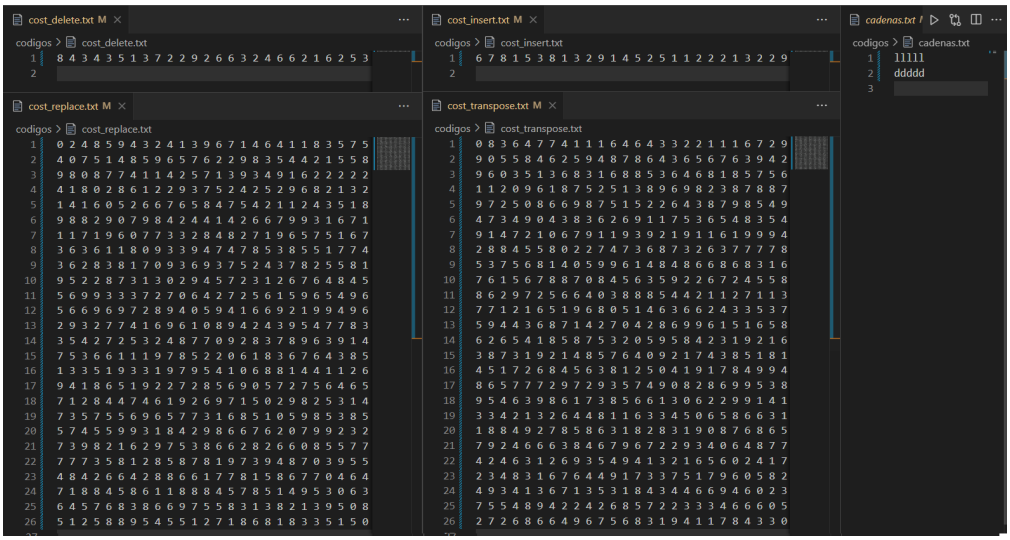

Figura 2: Archivos con las tablas y cadenas necesarias para el caso limite 2

3. Caso con cadenas simétricas:

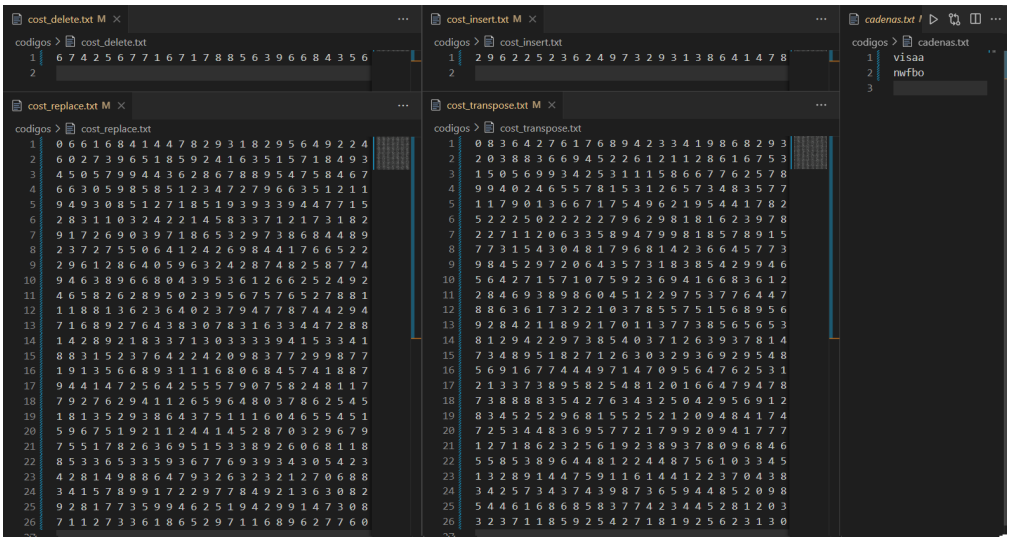

Figura 3: Archivos con las tablas y cadenas necesarias para el caso limite 3

4. Caso con cadenas asimétricas:

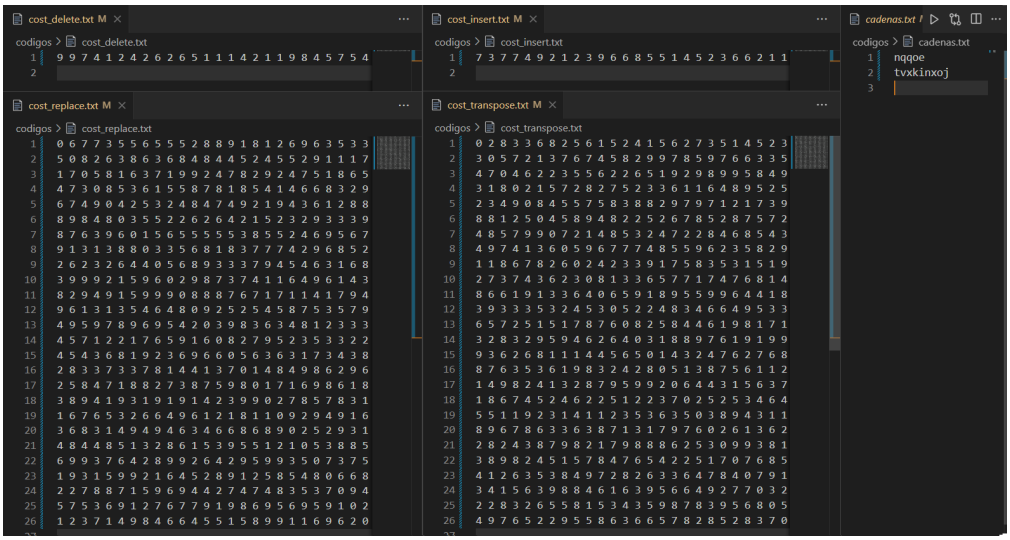

Figura 4: Archivos con las tablas y cadenas necesarias para el caso limite 4

5. Caso con matrices con valores iguales:

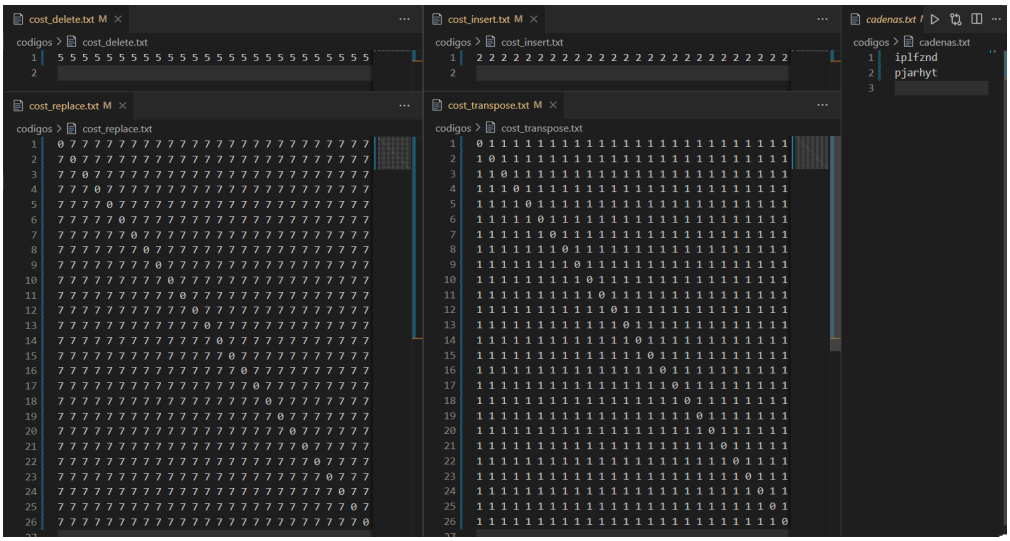

Figura 5: Archivos con las tablas y cadenas necesarias para el caso limite 5

4.2. Resultados

Primero y siguiendo con el orden de los datasets mostrados serán presentados los resultados referentes a los casos limite y luego como aumentan respecto al aumento de caracteres de forma simétrica:

1. Caso con cadenas vacías:

```
Ejecutando progdinamica:
La distancia minima de edicion es: 0
Tiempo de ejecucion: 179 microsegundos

Ejecutando progfuerzabruta:
La distancia minima de edicion es: 0
Tiempo de ejecucion: 100 microsegundos
```

Figura 6: Resultados para el caso limite 1

2. Caso con cadenas repetidas:

```
Ejecutando progdinamica:
La distancia minima de edicion es: 15
Tiempo de ejecucion: 5187 microsegundos

Ejecutando progfuerzabruta:
La distancia minima de edicion es: 15
Tiempo de ejecucion: 258936 microsegundos
```

Figura 7: Resultados para el caso limite 2

3. Caso con cadenas simétricas:

```
Ejecutando progdinamica:
La distancia minima de edicion es: 17
Tiempo de ejecucion: 7075 microsegundos

Ejecutando progfuerzabruta:
La distancia minima de edicion es: 17
Tiempo de ejecucion: 261545 microsegundos
```

Figura 8: Resultados para el caso limite 3

4. Caso con cadenas asimétricas:

```
Ejecutando progdinamica:
La distancia minima de edicion es: 18
Tiempo de ejecucion: 9615 microsegundos

Ejecutando progfuerzabruta:
La distancia minima de edicion es: 18
Tiempo de ejecucion: 1889464 microsegundos
```

Figura 9: Resultados para el caso limite 4

5. Caso con matrices con valores iguales:

```
Ejecutando progdinamica:
La distancia minima de edicion es: 42
Tiempo de ejecucion: 10216 microsegundos

Ejecutando progfuerzabruta:
La distancia minima de edicion es: 42
Tiempo de ejecucion: 5079571 microsegundos
```

Figura 10: Resultados para el caso limite 5

6. Progresión respecto al aumento de caracteres:

n° caracteres	Tiempo ProgDinamica	Tiempo ProgFuerzaBruta
1	489 ms	415 ms
5	5479 ms	252955 ms
10	19758 ms	indefinido
15	148322 ms	indefinido

Figura 11: Resultados para la progresión con aumento de caracteres

4.3. Analisis de Resultados

Es posible ver para la Fuerza Bruta que a medida que incrementa el número de caracteres en las cadenas de entrada, la cantidad de llamadas recursivas y los subproblemas a resolver aumenta drásticamente, lo que provoca una crecimiento exponencial del tiempo de ejecución. En experimentos prácticos, se observa que, para cadenas de longitud moderada, el tiempo de ejecución aumenta significativamente, y se vuelve inviable para cadenas más largas (por ejemplo, más de 10-12 caracteres). Esto se debe a que el número de subproblemas crece exponencialmente con el tamaño de las cadenas, lo que hace que el algoritmo sea muy lento.

La Programación Dinámica mejora considerablemente la eficiencia del algoritmo, ya que solo resuelve cada subproblema una vez. Este enfoque es significativamente más rápido en comparación con la fuerza bruta, especialmente cuando las cadenas tienen longitudes mayores. Los resultados experimentales muestran que incluso con cadenas de 100 o más caracteres, el algoritmo de Programación Dinámica es mucho más rápido y sigue siendo manejable en términos de tiempo de ejecución.

En el caso de cadenas simétricas, el algoritmo de Fuerza Bruta sigue siendo relativamente lento y peor que el de Programación Dinámica, ya que evalúa todas las combinaciones posibles de operaciones. Sin embargo, debido a la regularidad de la estructura de las cadenas, el número de subproblemas realmente distintos a resolver puede ser menor, lo que reduce parcialmente el tiempo de ejecución. Por otro lado, en el caso de cadenas asimétricas, la Fuerza Bruta se ve seriamente afectada. Debido a la falta de regularidad en la estructura de las cadenas, el número de subproblemas a resolver crece de manera exponencial.

5. Conclusiones

En base a los resultados obtenidos, se puede concluir que el enfoque de Programación Dinámica ofrece una solución mucho más eficiente al problema de la Distancia Mínima de Edición que el enfoque de Fuerza Bruta, especialmente cuando se trata de cadenas de caracteres largas. La Fuerza Bruta presenta un crecimiento exponencial en el tiempo de ejecución a medida que se incrementan los caracteres, lo que la convierte en una opción inviable para entradas de mayor tamaño debido a su complejidad. En cambio, la Programación Dinámica optimiza este proceso al dividir el problema en subproblemas más pequeños y reutilizar los resultados previamente calculados, lo que reduce significativamente los tiempos de ejecución y hace que el algoritmo sea escalable.

Además, al considerar cadenas simétricas y asimétricas, se observó que la Fuerza Bruta se ve gravemente afectada en el caso de cadenas asimétricas, donde la cantidad de combinaciones posibles de operaciones es mucho mayor. En comparación, el algoritmo de Programación Dinámica se comporta de manera mucho más eficiente independientemente de la simetría de las cadenas, demostrando la robustez de este enfoque. En resumen, la Programación Dinámica es claramente la mejor opción para resolver el problema de la Distancia Mínima de Edición en términos de tiempo y eficiencia, especialmente cuando se manejan cadenas de texto grandes o complejas. En general, se verifica la hipótesis preliminar respecto a los enfoques y como estos pueden cambiar la resolucion de un problema.

Una posible mejora sería optimizar el uso de memoria mediante la reducción del espacio de almacenamiento necesario. En lugar de utilizar una matriz completa para almacenar los resultados de todos los subproblemas, se podría emplear una estructura de memoria más compacta, como una matriz unidimensional o una técnica de compresión de los resultados intermedios. De todos modos, independiente de los resultados obtenidos se evidencio que existe una falencia en la comprobación de la complejidad espacial, es decir, el uso de memoria lo cual debe ser mejorado y abarcado de mejor forma en posteriores investigaciones.

6. Condiciones de entrega

- La tarea se realizará individualmente (esto es grupos de una persona), sin excepciones.
- La entrega debe realizarse vía `http://aula.usm.cl` en un tarball en el área designada al efecto, en el formato `tarea-2 y 3-rol.tar.gz` (`rol` con dígito verificador y sin guión).

  Dicho tarball debe contener las fuentes en LaTeX 2ε (al menos `tarea-2 y 3.tex`) de la parte escrita de su entrega, además de un archivo `tarea-2 y 3.pdf`, correspondiente a la compilación de esas fuentes.
- Si se utiliza algún código, idea, o contenido extraído de otra fuente, este **debe** ser citado en el lugar exacto donde se utilice, en lugar de mencionarlo al final del informe.
- Asegúrese que todas sus entregas tengan sus datos completos: número de la tarea, ramo, semestre, nombre y rol. Puede incluirlas como comentarios en sus fuentes LaTeX (en TeX comentarios son desde % hasta el final de la línea) o en posibles programas. Anótese como autor de los textos.
- Si usa material adicional al discutido en clases, detállelo. Agregue información suficiente para ubicar ese material (en caso de no tratarse de discusiones con compañeros de curso u otras personas).
- No modifique `preamble.tex`, `tarea_main.tex`, `condiciones.tex`, estructura de directorios, nombres de archivos, configuración del documento, etc. Sólo agregue texto, imágenes, tablas, código, etc. En el códigos funte de su informe, no agregue paquetes, ni archivos .tex (a excepción de que agregue archivos en `/tikz`, donde puede agregar archivos .tex con las fuentes de gráficos en `TikZ`).
- La fecha límite de entrega es el día 10 de noviembre de 2024.

NO SE ACEPTARÁN TAREAS FUERA DE PLAZO.

- Nos reservamos el derecho de llamar a interrogación sobre algunas de las tareas entregadas. En tal caso, la nota de la tarea será la obtenida en la interrogación.

NO PRESENTARSE A UN LLAMADO A INTERROGACIÓN SIN JUSTIFICACIÓN PREVIA SIGNIFICA AUTOMÁTICAMENTE NOTA 0.

A. Apéndice 1