

REPORTE TAREA 2 y 3

ALGORITMOS Y COMPLEJIDAD

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

Francisco Rebolledo C.

21 de noviembre de 2024

02:06

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	2
2. Diseño y Análisis de Algoritmos	3
3. Implementaciones	7
4. Experimentos	8
5. Conclusiones	12
6. Condiciones de entrega	13
A. Apéndice 1	14

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 óptimas que manualmente resultarían imposibles de realizar. Dentro de las ciencias de la computación una de las ramas mas importantes es el **Análisis y Diseño de Algoritmos**, ya que permite desarrollar diferentes formas para resolver problemas complejos y así, encontrar el método mas eficiente para abordar distintas problemáticas.

Este trabajo se enfoca en comparar dos de los principales paradigmas para la creación de algoritmos: **Fuerza Bruta** y **Programación Dinámica**, el primero de estos 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, el segundo 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.

Dentro de este reporte será planteado el problema de la **Distancia Mínima de Edición** como caso de estudio en el que aplicar los enfoques ya mencionados. Este problema consiste en determinar el número mínimo de operaciones necesarias para transformar una cadena A en otra cadena B, mediante las funciones insertar un carácter, eliminar un carácter y reemplazar un carácter. Además, para este experimento se incorpora adicionalmente la operación de transposición, la cual corresponde al intercambio posicional de 2 caracteres adyacentes para igualar cadenas de caracteres.

Para lograr un buen experimento sobre la diferencia entre los enfoques utilizados para abordar este problema, junto a evidenciar como el Diseño y Análisis de algoritmos afecta a la obtención e identificación de una solución óptima es relevante estudiar la naturaleza del problema, las características de las cadenas de entrada, como su simetría o asimetría afectan el rendimiento de los algoritmos, su complejidad temporal y espacial, etc. Todo esto, a través del estudio de sus respectivos tiempos de ejecución y la calidad de las respuestas proporcionadas por los algoritmo para no solo tener 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, antes de entrar directamente en el análisis de los algoritmos resulta importante definir el diseño que seguirán los paradigmas para tener en cuenta cuáles son sus principales diferencias y como esto puede llegar a afectar a la solución que dan al problema.

En general, estos enfoques buscaran abordar el problema 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.

Además, 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 cadena A en la cadena B y luego buscar cual tuvo el mínimo costo dentro de todas esas posibilidades.

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 la cadena A y la cadena B. Definidos estos valores, **n** debe representar todos los posibles números de entrada, por lo que será correspondiente al valor de la cadena más larga entre a y b. Para el análisis de las complejidades, se verifica que la función posee un factor de recurrencia de 4 y va guardando todos los posibles valores en cada oportunidad, por lo que se obtienen los siguientes 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

```

1 Procedure DME_Fuerza_Bruta(cadena1, cadena2, i, j)
2 if i == 0 then
3   | return j · costo_ins(cadena2[j - 1])
4 if j == 0 then
5   | return i · costo_del(cadena1[i - 1])
6 if cadena1[i - 1] == cadena2[j - 1] then
7   | return DME_Fuerza_Bruta(cadena1, cadena2, i - 1, j - 1)
8 insertar ← DME_Fuerza_Bruta(cadena1, cadena2, i, j - 1) + costo_ins(cadena2[j - 1])
9 eliminar ← DME_Fuerza_Bruta(cadena1, cadena2, i - 1, j) + costo_del(cadena1[i - 1])
10 sustituir ←
    DME_Fuerza_Bruta(cadena1, cadena2, i - 1, j - 1) + costo_sub(cadena1[i - 1], cadena2[j - 1])
11 transponer ← ∞
12 if i > 1 j > 1 cadena1[i - 1] == cadena2[j - 2] cadena1[i - 2] == cadena2[j - 1] then
13   | transponer ←
    DME_Fuerza_Bruta(cadena1, cadena2, i - 2, j - 2) + costo_trans(cadena1[i - 2], cadena1[i - 1])
14 return mín({insertar, eliminar, sustituir, transponer})

```

2.2. Programación Dinámica

2.2.1. Descripción de la solución

De forma similar, pero con mucha menor complejidad en comparación al enfoque de fuerza bruta, la solución al problema se encuentra al plantear la recursividad de la función principal, con la diferencia esta vez de que se deben guardar los casos intermedios para utilizarlos en la construcción de la solución principal. Por lo que primero, el algoritmo construye una matriz de tamaño $(a+1) \times (b+1)$, donde a y b son las longitudes de las cadenas A y B respectivamente, con el motivo de almacenar los costos mínimos de transformar los primeros i caracteres de una cadena en los primeros j de la otra. Luego, inicializa los casos base para manejar transformaciones desde o hacia cadenas vacías y después llena la matriz evaluando las operaciones de insertar, eliminar, reemplazar y, si es posible, transponer caracteres. Así, el costo mínimo para cada operación se calcula de forma acumulativa, evitando cálculos redundantes. Finalmente, el valor en la celda (a,b) de la matriz representa el costo mínimo total.

2.2.2. Relación de recurrencia

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

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 a caracteres de la primera cadena y los primeros b 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, como se menciono anteriormente, a es la longitud de la primera cadena y b es la longitud de la segunda cadena. Con esto, el algoritmo llena una matriz de dimensiones $(a+1) \times (b+1)$, calculando cada celda de la matriz en tiempo constante y de forma que cada celda almacena un subproblema. Bajo esta descripción se puede verificar:

Complejidad Temporal	Complejidad Espacial
$T(n) = O(a * b) = O(n^2)$	$E(n) = O(a * b) = O(n^2)$

2.2.5. Algoritmo utilizando programación dinámica

Algoritmo 2: DME_Programacion_Dinamica

```

1  Procedure DME_Programacion_Dinamica(cadena1, cadena2)
2   $a \leftarrow \text{len}(\text{cadena1})$ 
3   $b \leftarrow \text{len}(\text{cadena2})$ 
4  Crear matriz matriz de tamaño  $(a + 1) \times (b + 1)$  inicializada con ceros
5  for  $i \leftarrow 0$  to  $a$  do
6     $\text{matriz}[i][0] \leftarrow i \cdot \text{costo\_del}(\text{cadena1}[i - 1])$ 
7  for  $j \leftarrow 0$  to  $b$  do
8     $\text{matriz}[0][j] \leftarrow j \cdot \text{costo\_ins}(\text{cadena2}[j - 1])$ 
9  for  $i \leftarrow 1$  to  $a$  do
10   for  $j \leftarrow 1$  to  $b$  do
11     if  $\text{cadena1}[i - 1] == \text{cadena2}[j - 1]$  then
12        $\text{matriz}[i][j] \leftarrow \text{matriz}[i - 1][j - 1]$ 
13     else
14        $\text{matriz}[i][j] \leftarrow \text{mín} \left( \right.$ 
15          $\text{matriz}[i - 1][j] + \text{costo\_del}(\text{cadena1}[i - 1]),$ 
16          $\text{matriz}[i][j - 1] + \text{costo\_ins}(\text{cadena2}[j - 1]),$ 
17          $\text{matriz}[i - 1][j - 1] + \text{costo\_sub}(\text{cadena1}[i - 1], \text{cadena2}[j - 1])$ 
18        $\left. \right)$ 
19     if  $i > 1 \wedge j > 1 \wedge \text{cadena1}[i - 1] == \text{cadena2}[j - 2] \wedge \text{cadena1}[i - 2] == \text{cadena2}[j - 1]$  then
20        $\text{matriz}[i][j] \leftarrow \text{mín} \left( \right.$ 
21          $\text{matriz}[i][j],$ 
22          $\text{matriz}[i - 2][j - 2] + \text{costo\_trans}(\text{cadena1}[i - 1], \text{cadena1}[i - 2])$ 
23        $\left. \right)$ 
24 return  $\text{matriz}[a][b]$ 

```

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 create:** Inicia Generador.py para crear el caso de prueba con ciertos parámetros.
- **make all:** Compila los archivos de C++ para posteriormente ejecutarlos junto a los demás.
- **make run:** Ejecuta los programas en el orden 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 '**cadena.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 '**cadena.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

La replicación de este experimento es crucial para la comprobación y validez de los resultados dada la gran cantidad de variables que pueden afectar en el proceso, por lo que a continuación se detallan las especificaciones del equipo utilizado en la toma de datos 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 HFM512GDJTN1-82A0A (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 con crecimiento en el número de caracteres es importante obtener resultados con casos que posean ciertas características distintivas o límites respecto a los demás, es por ello que los 5 principales serán casos con cadenas de caracteres vacíos, cadenas de caracteres repetidos, cadenas simétricas, cadenas asimétricas y donde las matrices tienen un mismo valor para todas las operaciones, casos los cuales tienen al rededor de 5 caracteres por cadena para que esta variable no afecte en los resultados. (Estos datasets pueden ser encontrados en sus respectivas carpetas dentro del repositorio de Github o como imágenes en el apéndice de este reporte)

1. Caso con cadenas vacías: En este [caso](#) se deben testear cadenas sin ningún carácter para ver el funcionamiento correcto de los algoritmos y ver cual posee mayor efectividad al lidiar con tal problemática.

2. Caso con caracteres repetidos: La característica de este [caso](#) es que las cadenas son repeticiones de un mismo carácter para ver si la repetición de un subproblema hace al algoritmo más eficiente.

3. Caso con cadenas simétricas: La intención de este [caso](#) es ver si ayuda de alguna forma a los algoritmos que las cadenas posean igual cantidad de caracteres y compararlo con las cadenas asimétricas.

4. Caso con cadenas asimétricas: Este [caso](#) va de la mano con el anterior y sirve para verificar si las cadenas de distinto largo tienen algún impacto en los tiempos de ejecución y la respuesta de los algoritmos.

5. Caso con matrices con valores iguales: Por último [caso](#), resulta interesante verificar el efecto de matrices iguales o planas en sus valores y como esto puede afectar en la calidad de sus respuestas.

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 los tiempos de ejecución respecto al aumento de caracteres, haciendo un análisis de lo que reflejan en temas de optimización y calidad de respuestas:

Caso Particular	Tiempo Fuerza bruta	Tiempo Prog. Dinámica
Cadenas Vacías	100 [ms]	179 [ms]
Caracteres Repetidos	258936 [ms]	5187 [ms]
Cadenas Simétricas	261545 [ms]	7075 [ms]
Cadenas Asimétricas	1889464 [ms]	9615 [ms]
Matrices Iguales	257996 [ms]	6824 [ms]

Cuadro 1: Resultados Casos Limites.

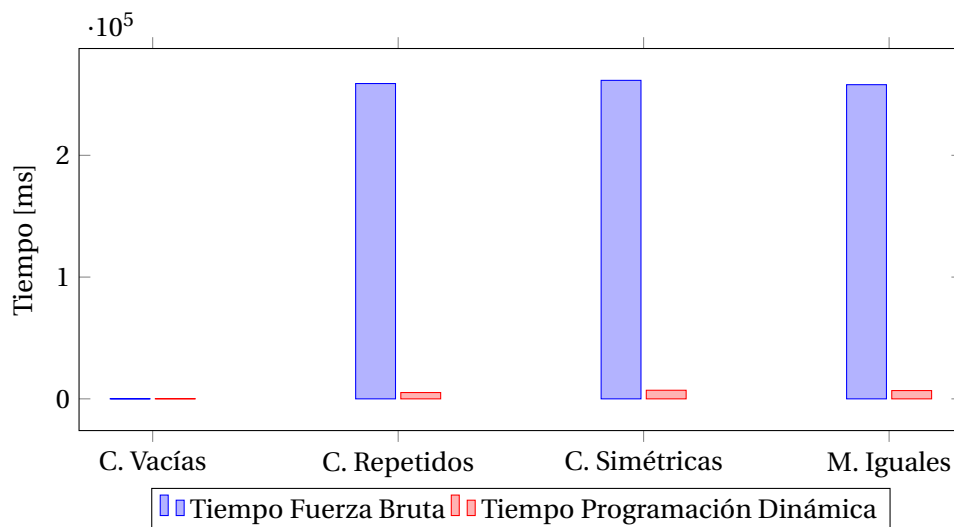


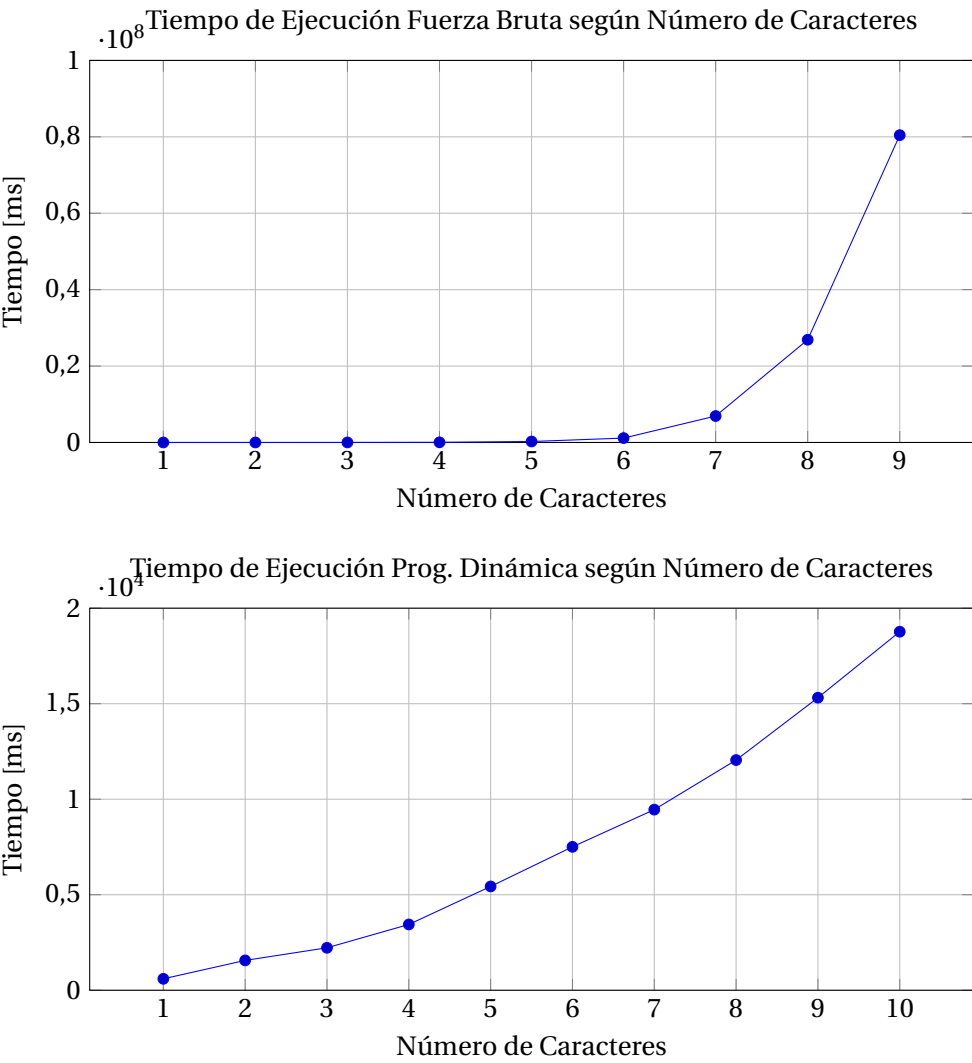
Figura 1: Comparación de los tiempos de Fuerza Bruta y Programación Dinámica.

Resulta importante destacar de este gráfico que han sido quitados los valores para el caso de **Cadenas Asimétricas** ya que para el enfoque de Fuerza Bruta, como se puede ver en la Tabla, este se escapaba por mucho de los otros valores, haciendo que fuera difícil visualizarlo junto a los demás.

De esto es posible ver como las cadenas asimétricas no afectan en gran medida pues están directamente relacionadas a la cadena de mayor tamaño mas que al contenido de las mismas. También se puede apreciar como con cadenas vacías el enfoque de Fuerza Bruta es levemente mejor al ser más directo. Además, se ve que dentro del enfoque de Programación Dinámica la repetición de caracteres ayuda a agilizar los procesos y por último, que las matrices iguales ayudan un poco pero no significativamente al procesamiento de los datos y a la eficiencia de los algoritmos.

Número Caracteres	Tiempo Fuerza bruta	Tiempo Prog. Dinámica
1	419 [ms]	599 [ms]
2	2065 [ms]	1562 [ms]
3	9582 [ms]	2223 [ms]
4	40931 [ms]	3444 [ms]
5	257123 [ms]	5434 [ms]
6	1150738 [ms]	7507 [ms]
7	6921528 [ms]	9456 [ms]
8	26891156 [ms]	12050 [ms]
9	80437376 [ms]	15315 [ms]
10	Indefinido	18769 [ms]

Cuadro 2: Resultados Aumento Progresivo de Caracteres.



4.3. Análisis 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 un 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 no se ve significativamente afectado, 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. El único caso en el que es mejor un enfoque de Fuerza Bruta es en el de cadenas vacías y se debe a la simplicidad del mismo.

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 como los enfoques pueden cambiar la resolución de un problema y que tan importante es el Diseño y Análisis de Algoritmos para encontrar las soluciones optimas a este.

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).

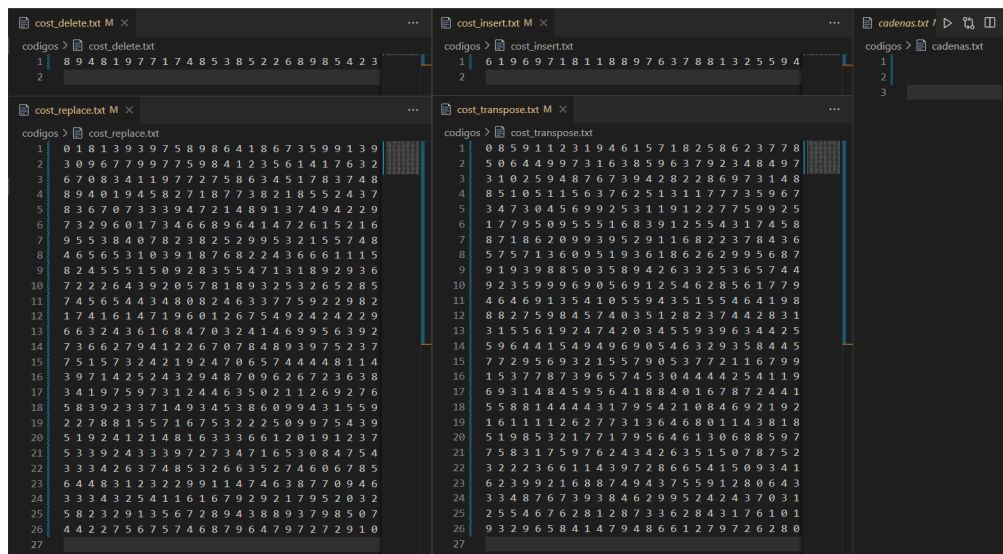
Dicho **tarball** debe contener las fuentes en \LaTeX (al menos **tarea-2 y 3.tex**) de la parte escrita de su entrega, además de un archivo **tarea-2 y 3.pdf**, correspondiente a la compilación de esas fuentes.
- Si se utiliza algún código, idea, o contenido extraído de otra fuente, este **debe** ser 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 código fuente 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.

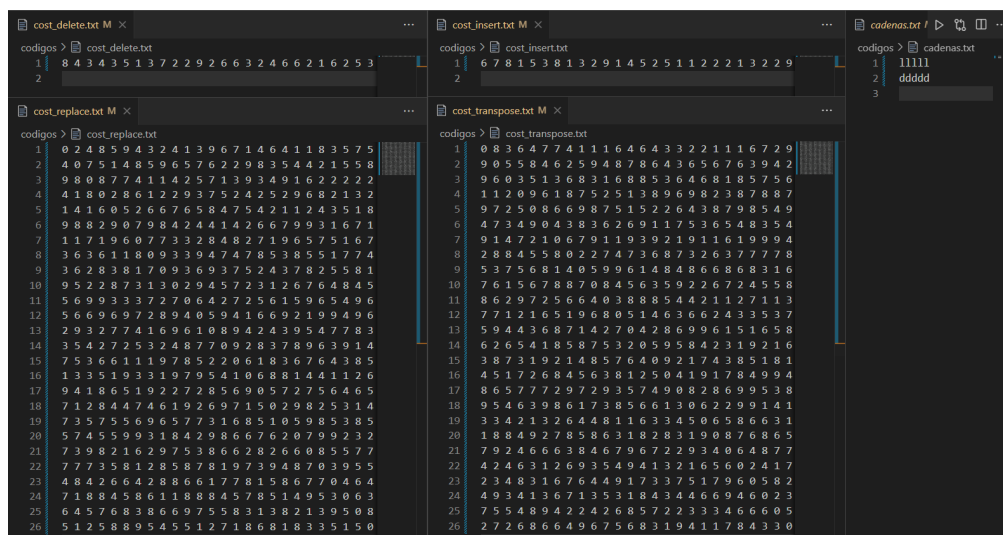
A. Apéndice 1



cost_delete.txt	cost_insert.txt	cadenas.txt
1 89481977174853852268985423	1 61969718118897637881325594	1
2	2	2
		3

cost_replace.txt	cost_transpose.txt
1 01813939758986418673599139	1 08591123194615718258623778
2 30967799775984123561417632	2 50644997316385963792348497
3 67083411977275863451783748	3 31025948767394282786973148
4 89401945827187738218552437	4 85105115637625131177735967
5 83670733394721489137494229	5 34730456992531191227759925
6 73296017346689641472615216	6 17795095551683912554317458
7 95538407823825299532155748	7 87186209939529116822378436
8 46565310391876822436661115	8 57571360951936186262995687
9 82455515092835547131892936	9 91939885035894263325365744
10 72226439205781893253265285	10 92359996905691254628561779
11 74565443480824633775922982	11 46469135410559435155464198
12 17416147196012675492424229	12 88275984574035128237442831
13 66324361684703241469956392	13 3155619247420345930634425
14 73662794122670784893975237	14 59644154949690546329358445
15 75157324219247065744448114	15 77295693215579053772116799
16 39714252432948709626723638	16 15377873965745304444254119
17 34197597312446350211269276	17 69314845956418840167872441
18 58392337149345386009431559	18 55881444431795421084692192
19 2278815571675322509975439	19 16111126277313646801143818
20 51924121481633366120191237	20 51985321771795646130688597
21 53392433397273471653084754	21 75831759762434263515078752
22 33342637485326635274606785	22 32223661143972866541509341
23 64483123229911474638770946	23 62399216887494375591280643
24 33343254116167929217952032	24 33487673938462995242437031
25 58232913567289438893798507	25 25546762812873362843176101
26 44227567574687964797272910	26 932965841147948661279726280
27	27

Figura 2: Tablas de Costo y Cadenas para el Caso con Cadenas Vacías



cost_delete.txt	cost_insert.txt	cadenas.txt
1 84343513722926632466216253	1 67815381329145251122213229	1 11111
2	2	2 dddd
		3

cost_replace.txt	cost_transpose.txt
1 024859432411396714641183575	1 08364774111646433221116729
2 40751485965762298354421558	2 90558462594878643656763942
3 98087741142571393491622222	3 96035136831688536468185756
4 41802861229375242529682132	4 11209618752513896982387887
5 14160526676584754211243518	5 97250866987515226438798549
6 98829079842441426679931671	6 47349043836269117536548354
7 11719607733284827196575167	7 91472106791193921911619994
8 3636118093394747853851774	8 2884580227473687326377778
9 36283017003693752437825581	9 53756814059961484866868316
10 95228731302945723126764845	10 76156788708456359226724558
11 56993337270642725615965496	11 86297256640388854421127113
12 56696972894059416692199496	12 77121651968051463662433537
13 29327741696108942439547783	13 59443687142704286996151658
14 35427253248770928378963914	14 62654185875320595842319216
15 75366111978522061836764385	15 38731921485764092174385181
16 13351933197954106881441126	16 45172684563812504191784994
17 94186519227285690572756465	17 86577729729357490828699538
18 71284474619269715029825314	18 95463986173856613062299141
19 73575569657731685105985385	19 33421326448116334506586631
20 57455993184298667620799232	20 18849278586318283190876865
21 73982162975386628266085577	21 79246663846796722934064877
22 77735812858781973948703955	22 42463126935494132165602417
23 48426642886617781586770464	23 23483167644917337517960582
24 71884586118884578514953063	24 49341367135318434466946023
25 64576838669755831382139508	25 75548942242685722333466605
26 51258895455127186818335150	26 27268664967568319411784330

Figura 3: Tablas de Costo y Cadenas para el Caso con Caracteres Repetidos

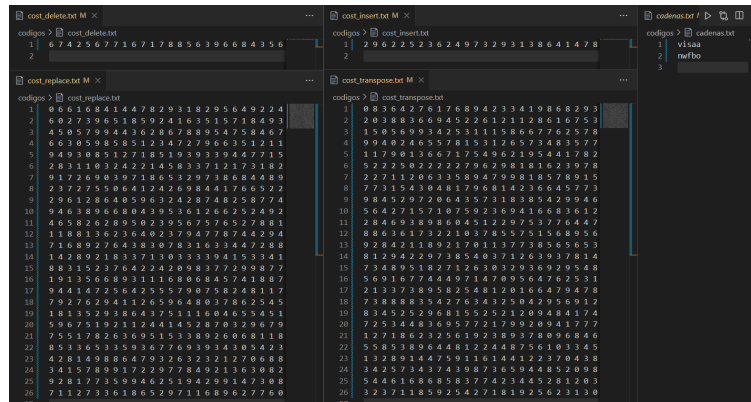


Figura 4: Tablas de Costo y Cadenas para el Caso con Cadenas Simétricas

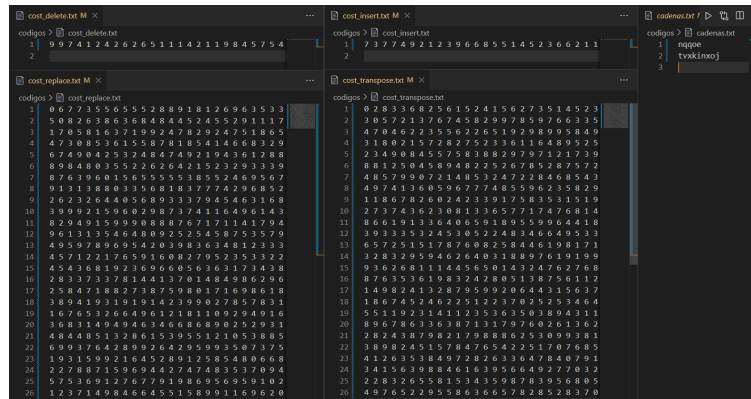


Figura 5: Tablas de costo y Cadenas para el Caso con Cadenas Asimétricas

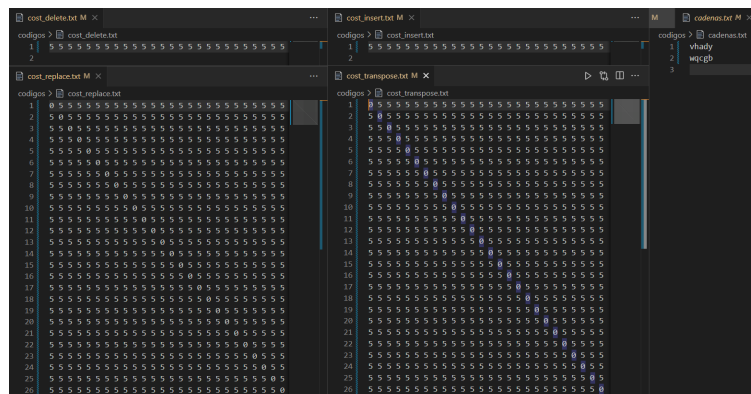


Figura 6: Tablas de Costo y Cadenas para el Caso con Matrices de Iguales Valores

```
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 7: Resultados Caso de Cadenas Vacías

```
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 8: Resultados Caso Caracteres Repetidos

```
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 9: Resultados Caso Cadenas Simétricas

```
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 10: Resultados Caso Cadenas Asimétricas

```
Ejecutando progdinamica:  
La distancia minima de edicion es: 25  
Tiempo de ejecucion: 6824 microsegundos  
  
Ejecutando progfuerzabruta:  
La distancia minima de edicion es: 25  
Tiempo de ejecucion: 257996 microsegundos
```

Figura 11: Resultados Caso Matrices Valores Iguales