INFORME TAREA 2 y 3
ALGORITMOS Y COMPLEJIDAD

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

Diego Sierra

17 de noviembre de 2024

Resumen

https://github.com/Daspssj/Tarea2_3

Índice

1. Introducción

2. Diseño y Análisis de Algoritmos

2.1. Fuerza Bruta

La solucion diseñada para el algoritmo de fuerza bruta se enfoca en recorrer todas las posibles combinaciones de las operaciones de inserción, eliminacion, sustitución y transposición, comparando la cadena de strings `S1` y `S2` para encontrar la distancia minima de edicción extendida, modificando unicamente una de estas 2 cadenas en cada iteración, en mi caso se modifica la primera cadena `S1`.

Como hay que explesar las complajidades en terminos de las longitudes de las cadenas de entrada $S1$ y $S2$, se tiene que n y m son las longitudes de las cadenas $S1$ y $S2$ respectivamente y se tiene que la complejidad temporal de este algoritmo es de $O(4^{\text{máx}(n,m)})$ y la complejidad espacial es de $O(\text{máx}(n,m))$.

La transposiciones y costos variables impactan significativamente en la complejidad en el caso de las transposiciones se tiene que la complejidad temporal aumenta a $O(4^{\text{máx}(n,m)})$ siendo la complejidad espacial sin la transposición de $O(3^{\text{máx}(n,m)})$ en cambio la complejidad espacial con la transposición es la misma que la vista anteriormente.

Para el ejemplo de ejecucionse tienen las cadenas $S1$ = "kittenz $S2$ = "sitting", se tiene que la distancia de edición mínima es de 1, y se puede obtener de la siguiente manera:

- **Sustituir 'k' por 's':** k $\rightarrow$s
- **Insertar 'g' al final de la cadena:** $\rightarrow$g
- **Sustituir 'e' por 'i':** e $\rightarrow$i

Algoritmo 1: Fuerza Bruta para calcular la distancia mínima de edición.

```
1  Procedure FuerzaBruta(S1, S2)
2      n ← longitud de S1
3      m ← longitud de S2
4      if n = 0 then
5          return costo de insertar todos los caracteres de S2
6      else if m = 0 then
7          return costo de eliminar todos los caracteres de S1
8      costo_sust ← CostoSust(S1[n-1], S2[m-1]) + FuerzaBruta(S1[0:n-1], S2[0:m-1])
9      costo_elim ← CostoElim(S1[n-1]) + FuerzaBruta(S1[0:n-1], S2)
10     costo_inser ← CostoInser(S2[m-1]) + FuerzaBruta(S1, S2[0:m-1])
11     costo_trans ← ∞
12     if n > 1 and m > 1 and S1[n − 1] = S2[m − 2] and S1[n − 2] = S2[m − 1] then
13         costo_trans ← CostoTrans(S1[n-2], S1[n-1]) + FuerzaBruta(S1[0:n-2], S2[0:m-2])
14     return mín(costo_sust, costo_elim, costo_inser, costo_trans)
```

2.2. Programación Dinámica

2.2.1. Descripción de la solución recursiva

Este algoritmo busca calcular el costo mínimo de transformar una cadena de caracteres $S1$ en otra cadena de caracteres $S2$. Para ello, se consideran las siguientes operaciones:

- **Inserción:** Insertar un carácter a la cadena $S1$.
- **Eliminación:** Eliminar un carácter de la cadena $S1$.
- **Sustitución:** Reemplazar un carácter de la cadena $S1$ por otro que le correspon a $S2$.
- **Transposición:** Intercambiar dos caracteres consecutivos de la cadena $S1$ y $S2$.

Se enfoca en comparar el costo de realizar cada una de estas operaciones para cada caracter para asi elejir la operación que minimice el costo total de transformar $S1$ en $S2$.

2.2.2. Relación de recurrencia

Sea dp[i][j] el costo mínimo de transformar los primeros i caracteres de S1 en los primeros j caracteres de S2, tenemos que la relación de recurrencia es la siguiente:

$$dp[i][j] = \begin{cases} j \cdot \text{costo\_inser}(s2[j-1]), & \text{si } i = 0 \\ i \cdot \text{costo\_elim}(s1[i-1]), & \text{si } j = 0 \\ \text{mín}\Big(dp[i][j-1] + \text{costo\_inser}(s2[j-1]), & \text{(insertar)} \\ dp[i-1][j] + \text{costo\_elim}(s1[i-1]), & \text{(eliminar)} \\ dp[i-1][j-1] + \text{costo\_sust}(s1[i-1], s2[j-1]), & \text{(sustituir)} \\ dp[i-2][j-2] + \text{costo\_transpos}(s1[i-1], s2[j-1]) & \text{(transponer, si aplica)}\Big) \end{cases}$$

Como se puede obeservar en la relación de recurrencia, se consideran los casos base cuando $i = 0$ y $j = 0$, que corresponden a los costos de insertar y eliminar todos los caracteres de $S2$ y $S1$, respectivamente, y tambien se incluye la operación de transposición si se cumplen las condiciones necesarias.

2.2.3. Identificación de subproblemas

La soluicon que propone este algoritmo de transformar una cadena de caracteres $S1$ en otra cadena de caracteres $S2$ (entregando los costos asociados) se puede dividir en subproblemas más pequeños, que corresponden a transformar subcadenas de $S1$ en subcadenas de $S2$, se busca transformar los prefijos inmediatos de $S1$ y $S2$ en cada iteración para luego sumar el costo de la operación que minimice el costo total de transformar $S1$ en $S2$.

2.2.4. Estructura de datos y orden de cálculo

Para resolver este problema utilizando programación dinámica, se propone utilizar una matriz dp de tamaño $(n+1) \times (m+1)$,donde n y m son las longitudes de las cadenas $S1$ y $S2$, respectivamente. El programa utiliza un programación dinamica don un enfoque de Bottom-Up. En la matriz se inicializan los valores de los casos base de manera que la primera fila es el costo de insertar caracteres en $S1$ y la primera columna respresenta el costo de eliminar caracteres de $S1$,luego se recorren las filas y columnas de la matriz para calcular el costo mínimo de transformar los prefijos de $S1$ y $S2$ en cada iteración, se llenan los valores de la matriz de izquierda a derecha y de abajo hacia arriba, de manera que al final de la ejecución el valor de $dp[n][m]$ corresponderá al costo mínimo.

2.2.5. Complejidades

La complejidad temporal y espacial de este algoritmo son las mismas y es de $O(n \times m)$, donde n y m son las longitudes de las cadenas $S1$ y $S2$, respectivamente.

2.2.6. Ejemplo de ejecución

Para las cadenas $S1$ = "kittenz $S2$ = "sitting", se tiene que la distancia de edición mínima es de 1, y se puede obtener de la siguiente manera:

- **Sustituir 'k' por 's':** k $\rightarrow$s
- **Insertar 'g' al final de la cadena:** $\rightarrow$g
- **Sustituir 'e' por 'i':** e $\rightarrow$i

2.2.7. Algoritmo utilizando programación dinámica

Algoritmo 2: Programación Dinámica para calcular la distancia mínima de edición.

```
1  Procedure ProgDinamica(S1, S2)
2      n ← longitud de S1
3      m ← longitud de S2
4      Crear matriz dp de tamaño (n+1) × (m+1)
5      for i ← 0 to n do
6          dp[i][0] ← costo de eliminar todos los caracteres hasta i en S1
7      for j ← 0 to m do
8          dp[0][j] ← costo de insertar todos los caracteres hasta j en S2
9      for i ← 1 to n do
10         for j ← 1 to m do
11             costo_inser ← dp[i][j−1] + CostoInser(S2[j-1])
12             costo_elim ← dp[i−1][j] + CostoElim(S1[i-1])
13             costo_sust ← dp[i−1][j−1] + CostoSust(S1[i-1], S2[j-1])
14             costo_trans ← ∞
15             if i > 1 and j > 1 and S1[i−1] = S2[j−2] and S1[i−2] = S2[j−1] then
16                 costo_trans ← dp[i−2][j−2] + CostoTrans(S1[i-2], S1[i-1])
17             dp[i][j] ← mín(costo_inser, costo_elim, costo_sust, costo_trans)
18     return dp[n][m]
```

3. Implementaciones

- **Estructura de archivos:** El archivo donde se encuentra la implementación de los algoritmos es Algoritmos.cpp , en este archivo se encuentran las implementaciones de los algoritmos de fuerza bruta y programación dinámica, como la funcion del main que se encarga de leer los datos de entrada y llamar a los algoritmos correspondientes.

  En esta carpeta de la Tarea 2 y 3 tambien se encuentran los costos de las operaciones de inserción, eliminación, sustitución y transposición, como tambien las distintas datasets que se utilizaron para probar los algoritmos.

  Tambien se encuentra el archivo de Creardataset.cpp que se encarga de crear los datasets que se utilizaron para probar los algoritmos.

- **Funciones:** Las funciones mas importantes son dynamic progra y brute force, que corresponden a las implementaciones de los algoritmos de programación dinámica y fuerza bruta, respectivamente.

4. Experimentos

4.1. Infraestructura utilizada

Para la realización de los experimentos, se utilizó un computador con las siguientes características:

- **Procesador:** Intel Core i7-14700KF, 5.6 GHz
- **Memoria RAM:** 32 GB DDR5
- **Almacenamiento:** 2 TB SSD NVMe
- **Sistema Operativo:** Windows 10 Pro
- **Compilador:** g++ 14.2.1
- **Librerías:** C++ Standard Library
- **Entorno de Desarrollo:** Ubuntu 14.2.0

4.2. Datasets

Los datasets utilizados para las pruebas fueron 6, y son de 20 palabras de largo variable

- **dataset1:** Contiene palabras de misma longitud que varia entra 1 y 14 caracteres
- **dataset2:** Contiene palabras donde S1 es vacia y S2 tiene longitud aleatoria
- **dataset3:** Contiene palabras donde S2 es vacia y S1 tiene longitud aleatoria
- **dataset4:** Contiene palabras donde S1 es mayor o igual que S2
- **dataset5:** Contiene palabras donde S2 es mayor o igual que S1
- **dataset6:** Contiene palabras transpuestas

La importancia de estos datasets radica en que se pueden probar los algoritmos con distintos casos de prueba, y se pueden verificar si los algoritmos son capaces de resolverlos de manera correcta en varios casos, esto nos puede dar una idea de la eficiencia y efectividad de los algoritmos. mendiantes las pruebas que veremos acontinuacion.

4.3. Resultados

Para el analisis de los resultados se tiene que hay que destacar que en el eje y estan los timepos de ejecucion en microsegundos y en el eje x se tiene el largo de las cadenas de entrada, se puede observar que en la mayoria de los casos el tiempo de ejecucion es muy bajo, esto se debe a que las cadenas de entrada son muy pequeñas, en el caso de las cadenas de entrada de longitud 14 se puede observar que el tiempo de ejecucion es mayor, esto se debe a que la complejidad del algoritmo es de $O(4^{\text{máx}(n,m)})$ y la

complejidad espacial es de $O(\text{máx}(n, m))$. Se puede observar tambien que en el caso de las cadenas de entradas para s1 y s2 vacios el tiempo de ejecucion de la fuerza bruta es mejor que el de programacion dinamica.

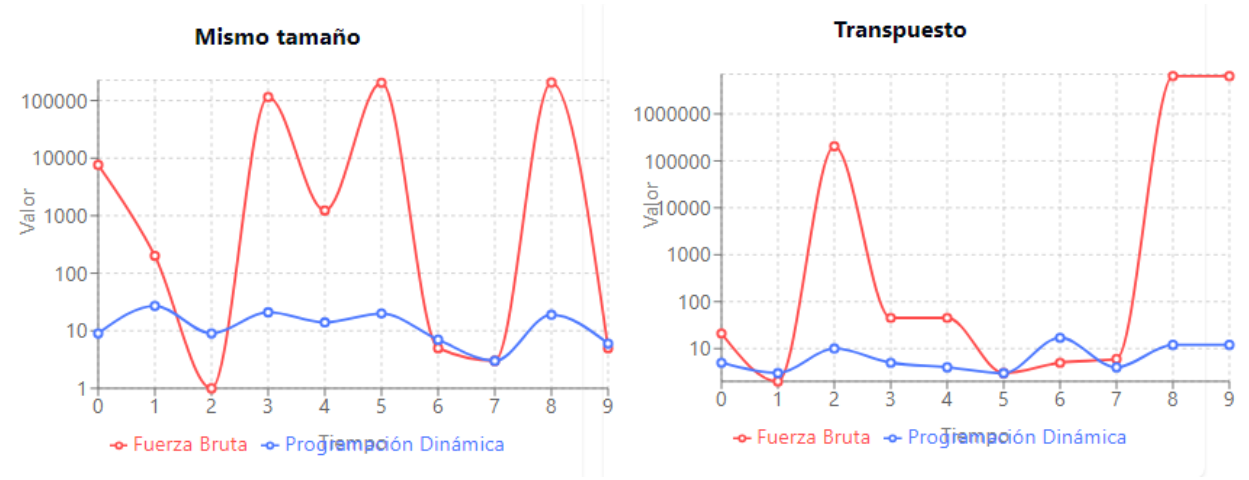

Figura 1: Tiempos de ejecucion vs largo de las cadenas

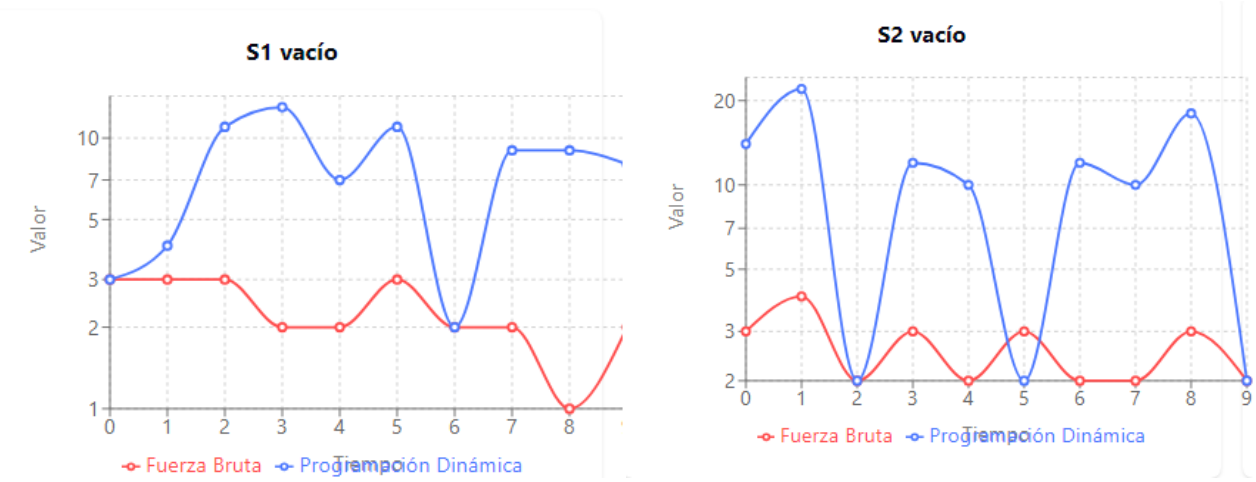

Figura 2: Tiempos de ejecucion vs largo de las cadenas

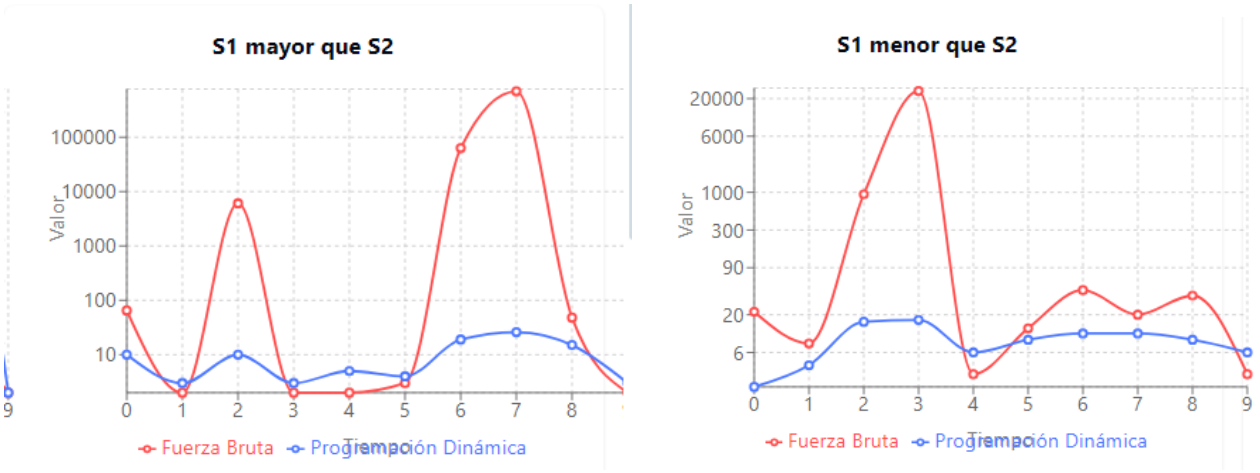

Figura 3: Tiempos de ejecucion vs largo de las cadenas

5. Conclusiones

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.