REPORTE TAREA 2 y 3
ALGORITMOS Y COMPLEJIDAD

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

Sebastián Muñoz

17 de noviembre de 2024

21:36

Resumen

Objetivo General implementar y analizar dos algoritmos que calculen la distancia mínima de edición entre dos cadenas, utilizando:

- *Fuerza Bruta: Evaluar todas las posibles transformaciones para determinar el costo mínimo.*
- *Programación Dinámica: Diseñar una solución eficiente aprovechando la reutilización de subproblemas.*

El cálculo incorpora costos variables para las operaciones de edición y considerando transposiciones como una operación adicional.

Índice

1. Introducción

La extensión máxima para esta sección es de 2 páginas.

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

La extensión máxima para esta sección es de 5 páginas.

Diseñar un algoritmo por cada técnica de diseño de algoritmos mencionada en la sección de objetivos. Cada algoritmo debe resolver el problema de distancia mínima de edición extendida, dadas dos cadenas `S1` y `S2`, utilizando las operaciones y costos especificados.

- Describir la solución diseñada.
- Incluir pseudocódigo (ver ejemplo **??**)
- Proporciones un ejemplo paso a paso de la ejecución de sus algoritmos que ilustren cómo sus algoritmos manejan diferentes escenarios, particularmente donde las transposiciones o los costos variables afectan el resultado. Haga referencias a los programas expresados en psudocódigo (además puede hacer diagramas).
- Analizar la Complejidad temporal y espacial de los algoritmos diseñados en términos de las longitudes de las cadenas de entrada $S1$ y $S2$
- Discute cómo la inclusión de transposiciones y costos variables impacta la complejidad.

Los pseudocódigos los he diseñado utilizando el paquete *Algorithm2e documentation* [2] para la presentación de algoritmos. Se recomienda consultar *Algorithm2e on CTAN* [3] y *Writing Algorithms in LaTeX* [4].

Todo lo correspondiente a esta sección es, digamos, en "lapiz y papel", en el sentido de que no necesita de implementaciones ni resultados experimentales.

Recuerde que lo importante es diseñar algoritmos que cumplan con los paradigmas especificados.

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.

2.1. Fuerza Bruta

Primero, se cargan matrices y vectores de los costos, se abren los archivos con las palabras a trasnformar, y procesa cada par de palabras estas estando divididas por un delimitador ("|"). El cálculo del costo se realiza mediante un algoritmo que considera transposiciones, sustituciones, inserciones y eliminaciones, usando funciones auxiliares que manejan matrices y vectores. El diseño incluye manejo de errores, como fallas al abrir archivos o al procesar datos, y asegura que los resultados se guarden correctamente, proporcionando robustez y funcionalidad para su uso en análisis lingüísticos o similares.

La complejidad de este algoritmo es:

- Temporal: O(n*l), donde n es la cantidad de pares de palabras y l el largo promedio de ellas.

- Espacial: O(L), donde L es el largo de la palabra más larga.

Algoritmo 1: Estructura del algoritmo para procesar pares de palabras y calcular costos mínimos.

```
1  Procedure MAIN()
2      CARGARMATRICES()
3      archivo ← abrir "palabras.txt.en modo lectura
4      archivo_salida ← abrir resultado.txt.en modo escritura
5      if archivo no está abierto then
6          mostrar "No se pudo abrir el archivo."
7          return 1
8      if archivo_salida no está abierto then
9          mostrar "No se pudo abrir el archivo de salida."
10         return 1
11     archivo → leer N
12     archivo → ignorar hasta nueva línea
13     for i ← 0 to N − 1 do
14         línea ← leer línea desde archivo
15         dividir línea en palabra1 y palabra2 usando '|' como delimitador
16         costo1 ← CALCULARCOSTO(palabra1, palabra2)
17         costo2 ← CALCULARCOSTO(palabra2, palabra1)
18         costoMin ← mín(costo1, costo2)
19         archivo_salida → escribir costo1, costo2, costoMin
20     cerrar archivo
21     cerrar archivo_salida
22     return 0
```

Algoritmo 2: Algoritmo para calcular el costo entre dos palabras considerando sustituciones, transposiciones, inserciones y eliminaciones.

```
1  Procedure CALCULARCOSTO(palabra1, palabra2)
2      costoTotal ← 0
3      largoP1 ← tamaño de palabra1
4      largoP2 ← tamaño de palabra2
5      n ← mín(largoP1, largoP2)
6      i ← 0
7      while i < n do
8          if i + 1 < n y palabra1[i + 1] = palabra2[i] y palabra1[i] = palabra2[i + 1] then
9              costoTotal ← costoTotal + COSTO_TRANS(palabra1[i], palabra1[i + 1])
10             i ← i + 2
11         else
12             costoTotal ← costoTotal + COSTO_SUB(palabra1[i], palabra2[i])
13             i ← i + 1
14     for i ← i to largoP2 − 1 do
15         costoTotal ← costoTotal + COSTO_INS(palabra2[i])
16     for i ← i to largoP1 − 1 do
17         costoTotal ← costoTotal + COSTO_DEL(palabra1[i])
18     return costoTotal
```

Algoritmo 3: Algoritmo para leer una matriz 26×26 desde un archivo.

```
1  Procedure LEERMATRIZ(nombreArchivo)
2      archivo ← abrir nombreArchivo en modo lectura
3      matriz ← matriz de 26 × 26 inicializada en ceros
4      if archivo no está abierto then
5          mostrar .Error al abrir el archivo - nombreArchivo
6          return matriz
7      for i ← 0 to 25 do
8          for j ← 0 to 25 do
9              archivo → leer matriz[i][j]
10             if lectura falla then
11                 mostrar .Error al leer el archivo en la posición (- i + ", - j + ")"
12                 cerrar archivo
13                 return matriz
14     cerrar archivo
15     return matriz
```

Algoritmo 4: Algoritmo para leer un vector de tamaño 26 desde un archivo, con manejo de errores.

```
1  Procedure LEERVECTOR(nombreArchivo)
2      archivo ← abrir nombreArchivo en modo lectura
3      if archivo no está abierto then
4          lanzar error "No se pudo abrir el archivo."
5      vector ← vector de tamaño 26
6      for i ← 0 to 25 do
7          archivo → leer vector[i]
8          if lectura falla then
9              lanzar error .Error al leer el archivo de vector."
10     if tamaño de vector ≠ 26 then
11         lanzar error .El archivo no contiene exactamente 26 números."
12     cerrar archivo
13     return vector
```

Ejemplo paso a paso:

- 1. Se toman el par de palabras S1=.ABAA*z* S2="BAAA*z* se empieza a leer letra a latra el S1.
- 2. Se comprueba si S1[0]==S2[1] y si S1[1]==S2[0].
- 3. Si es así, se hace una trasposición y se avanza en 2 el lector de la palabra. En este caso si es igual por lo que hay transposición.
- 4. En caso contrario se hace una sustitución y se avanza en 1 el lector de la palabra.
- 5. Se repite hasta el fin de la palabra para posteriormente ajustar la palabra con ins o del segun corresponda. En este caso no es necesario.
- 6. Se repiten todos los pasos pero son S2 y se toma el costo mínimo. En este caso los costos quedan igual.

2.2. Programación Dinámica

Dynamic programming is not about filling in tables. It's about smart recursion!

Erickson, 2019 [1]

2.2.1. Descripción de la solución recursiva

2.2.2. Relación de recurrencia

2.2.3. Identificación de subproblemas

2.2.4. Estructura de datos y orden de cálculo

2.2.5. Algoritmo utilizando programación dinámica

Algoritmo 5: Algoritmo de programación dinámica para calcular el costo mínimo de edición con inserción, eliminación, sustitución y transposición.

```
1  Procedure PROGRAMACIONDINAMICA(S1, S2, cost_insert, cost_delete, cost_replace, cost_transpose)
2      m ← longitud de S1
3      n ← longitud de S2
4      dp ← matriz de tamaño (m+1) × (n+1) inicializada en 0
5      for i ← 1 to m do
6          dp[i][0] ← dp[i−1][0] + cost_delete[S1[i−1] − 'a']
7      for j ← 1 to n do
8          dp[0][j] ← dp[0][j−1] + cost_insert[S2[j−1] − 'a']
9      for i ← 1 to m do
10         for j ← 1 to n do
11             if S1[i−1] == S2[j−1] then
12                 dp[i][j] ← dp[i−1][j−1]
13             else
14                 coste_sustitucion ← cost_replace[S1[i−1] − 'a'][S2[j−1] − 'a'] + dp[i−1][j−1]
15                 coste_insercion ← cost_insert[S2[j−1] − 'a'] + dp[i][j−1]
16                 coste_eliminacion ← cost_delete[S1[i−1] − 'a'] + dp[i−1][j]
17                 dp[i][j] ← mín(coste_sustitucion, coste_insercion, coste_eliminacion)
18                 if i > 1 and j > 1 and S1[i−1] == S2[j−2] and S1[i−2] == S2[j−1] then
19                     coste_transposicion ← cost_transpose[S1[i−2] − 'a'][S1[i−1] − 'a'] + dp[i−2][j−2]
20                     dp[i][j] ← mín(dp[i][j], coste_transposicion)
21     return dp[m][n]
```

3. Implementaciones

La extensión máxima para esta sección es de 1 página.

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.

Consideraciones: La operacion trasnponer solo se realiza si al realizarlo ambas letras quedan correspondientes a las letras de la otra palabra.

Para la progrmación dinámica: El programa lee un archivo de texto (palabras.txt) que contiene pares de cadenas separadas por un delimitador |. Para cada par, calcula la distancia mínima de edición utilizando la función minEditDistance, que llena una matriz de programación dinámica dp en la que cada elemento dp[i][j] representa el costo mínimo de convertir los primeros i caracteres de la cadena S1 en los primeros j caracteres de la cadena S2.

La función de distancia mínima de edición se calcula considerando los costos de sustitución (, inserción y eliminación. Además, existe una verificación para transponer caracteres adyacentes si es posible, lo que puede reducir el costo total.

Al final, los resultados se escriben en un archivo de salida, donde se muestra el costo mínimo de edición entre cada par de cadenas leídas.

4. Experimentos

La extensión máxima para esta sección es de 6 página.

Para evaluar el rendimiento de los algoritmos de fuerza bruta y programación dinámica, se realizaron experimentos en un sistema con las siguientes características:

- 1. Procesador: AMD Ryzen 5 3500U, 2.10 GHz
- 2. Memoria RAM: 8GB
- 3. Almacenamiento SSD
- 4. Sistema Operativo: Windows 11 Home Single Language
- 5. Compilador: g++ versión 14.1.0

4.1. Dataset (casos de prueba)

La extensión máxima para esta sección es de 2 páginas.

Se diseñaron conjuntos de prueba para analizar el comportamiento de ambos algoritmos bajo diferentes condiciones. Estos incluyen: cadenas vacías, cadenas idénticas, cadenas con caracteres repetidos, cadenas que requieren transposiciones, cadenas largas con costos variados. Aqui el dataset usado:

```
20
cat|bat
apple|applle
kitten|sitting
hello|hallo
abcd|abce
flame|flame
testing|tasting
banana|bananaa
flight|
book|look
star|start
mail|fail
moon|soon
light|night
train|grain
travel|traveling
dog|dig
close|clothes
```

chair|hair

open|pen

4.2. Resultados

La extensión máxima para esta sección es de 4 páginas.

Para la toma de datos se uso la función chrono de C++

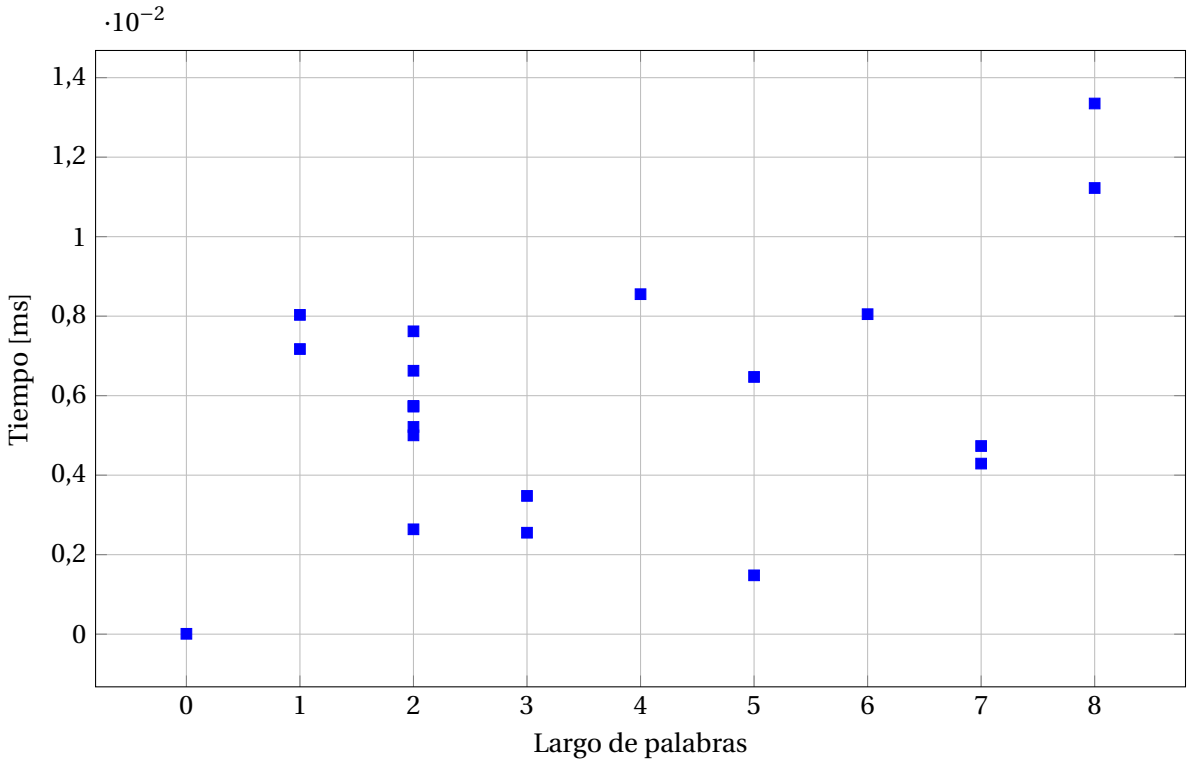

Figura 1: Grafico de tiempo en función del largo de las palabras

5. Conclusiones

La extensión máxima para esta sección es de 1 página.

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.

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

Link al repositorio donde se encuentran los códigos https://github.com/smunoz3/Tareas-Algoco/tree/main/Tareas/Tarea-2-3

Referencias

[1] Jeff Erickson. *Algorithms*. Jun. de 2019. ISBN: 978-1-792-64483-2.

[2] Christophe Fiorio. *Algorithm2e documentation*. http://ctan.math.illinois.edu/macros/latex/contrib/algorithm2e/doc/algorithm2e.pdf. 2023.

[3] Christophe Fiorio. *Algorithm2e on CTAN*. https://ctan.org/pkg/algorithm2e. 2023.

[4] Overleaf. *Writing Algorithms in LaTeX*. https://www.overleaf.com/learn/latex/Algorithms. 2023.