

REPORTE TAREA 2 y 3

ALGORITMOS Y COMPLEJIDAD

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

Carlos Vera Quezada

17 de noviembre de 2024

15:25

Resumen

Un resumen es un breve compendio que sintetiza todas las secciones clave de un trabajo de investigación: la introducción, los objetivos, la infraestructura y métodos, los resultados y la conclusión. Su objetivo es ofrecer una visión general del estudio, destacando la novedad o relevancia del mismo, y en algunos casos, plantear preguntas para futuras investigaciones. El resumen debe cubrir todos los aspectos importantes del estudio para que el lector pueda decidir rápidamente si el artículo es de su interés.

En términos simples, el resumen es como el menú de un restaurante que ofrece una descripción general de todos los platos disponibles. Al leerlo, el lector puede hacerse una idea de lo que el trabajo de investigación tiene para ofrecer [3].

La extensión del resumen, para esta entrega, debe ser tal que la totalidad del índice siga apareciendo en la primera página. Recuerde que NO puede modificar el tamaño de letra, interlineado, márgenes, etc.

Índice

1. Introducción	2
2. Diseño y Análisis de Algoritmos	3
3. Implementaciones	8
4. Experimentos	9
5. Conclusiones	16
6. Condiciones de entrega	17
A. Apéndice 1	18

1. Introducción

"La algoritmia es una de las bases fundamentales de ciencias de la computación. Aunque otras áreas han ganado terreno últimamente e.g., ciencia de datos, inteligencia artificial y deep learning, la algoritmia sigue siendo fundamental para proveer soluciones eficientes a muchos de los problemas que aparecen en esas áreas. Es, de alguna manera, un área transversal a ciencias de la computación."

Algoritmos Discretos: Análisis y Diseño [2]

En este informe se estudiará la distancia de edición extendida o también conocida como **Optimal String Alignment (OSA)**, este algoritmo nos permite calcular el número mínimo de operaciones para transformar una cadena de caracteres en otra, sin modificar más de una vez alguna de sus subcadenas. Las operaciones corresponden a **Sustitución, Inserción, Eliminación y Transposición**. Donde cada una tiene un costo asociado y el cual se busca minimizar.

El algoritmo tiene diversas aplicaciones, dentro de ellas está la búsqueda sobre documentos mediante escaneo, búsqueda en textos antiguos, búsqueda en bases de datos biológicas y corrección ortográfica. Lo cual hace que este algoritmo sea igual de importante para la ciencia que para la vida cotidiana. Además, es importante mencionar que debido a la cantidad de datos a analizar en los distintos campos, la eficiencia del algoritmo es algo que se ha ganado importancia a través del tiempo. [2]

Con este informe se busca implementar el algoritmo mediante dos metodologías de diseño distintas, fuerza bruta y programación dinámica. Las cuales en la teoría, poseen distintas complejidades temporales y espaciales. Esto se hará con el fin de comparar empíricamente las dos implementaciones y contrastarlo con la teoría, para así determinar cuál de las dos es mejor.

Teóricamente, la implementación del algoritmo mediante fuerza bruta posee una complejidad temporal perteneciente a $O(4^n)$, donde el 4 proviene de la cantidad de operaciones a probar y el n corresponde al tamaño de la cadena más larga, la complejidad espacial del algoritmo pertenece a $O(m)$ (Que corresponde a la profundidad de la pila de recursión). Por otra parte, la implementación mediante programación dinámica posee una complejidad temporal y espacial perteneciente a $O(n * m)$ donde n y m corresponden al largo de las cadenas.

Sobre el papel, a medida que crece el tamaño de la entrada, el algoritmo bajo programación dinámica debería ser mucho mejor, por otra parte, el algoritmo de fuerza bruta debería utilizar menor cantidad de espacio adicional, por lo cual dependiendo del escenario, uno podría ser mejor que el otro. Es significativo realizar estas comparaciones ya que puede ocurrir que en asintóticamente una implementación sea mejor que otra pero en la práctica, por ejemplo, solo se cumpla para entradas muy grandes. Por lo cual, nos podemos preguntar ¿Esto se refleja en escenarios reales?

Para realizar las comparaciones se medirá el tiempo de ejecución de cada implementación al igual que el consumo de memoria RAM para las mismas entradas, con el fin de reducir desvíos en las mediciones, se usará el promedio de varias ejecuciones para dar un resultado más significativo.

2. Diseño y Análisis de Algoritmos

2.1. Fuerza Bruta

1) Solución diseñada:

Se diseñó un algoritmo recursivo de fuerza bruta para calcular la distancia mínima de edición entre dos cadenas, S1 y S2. Este algoritmo utiliza índices i y j para comparar caracteres de las cadenas y una lista para registrar las operaciones óptimas que minimizan el costo de transformación.

El algoritmo explora todas las posibles combinaciones de operaciones (inserción, eliminación, sustitución, y transposición) y selecciona la que tiene el menor costo, basado en una matriz de costos. La solución diseñada fue una iteración sobre al algoritmo de fuerza bruta recursiva propuesto en el blog AfterAcademy [1], donde se modificó para agregar la operación de transposición y llevar el registro de operaciones óptimas.

2) Pseudocódigo:

```

1 Function FUERZABRUTA( $s1, s2, i, j$ ):
2    $m \leftarrow |s1|, n \leftarrow |s2|$ 
3   if  $i = m$  and  $j = n$  then
4     return 0
5   if  $i = m$  then
6     return CostoInsertar( $s2[j]$ ) + FUERZABRUTA( $s1, s2, i, j + 1$ )
7   if  $j = n$  then
8     return CostoEliminar( $s1[i]$ ) + FUERZABRUTA( $s1, s2, i + 1, j$ )
9   if  $i < m$  and  $j < n$  and  $s1[i] = s2[j]$  then
10    return FUERZABRUTA( $s1, s2, i + 1, j + 1$ )
11  else
12     $ins \leftarrow$  CostoInsertar( $s2[j]$ ) + FUERZABRUTA( $s1, s2, i, j + 1$ )
13     $del \leftarrow$  CostoEliminar( $s1[i]$ ) + FUERZABRUTA( $s1, s2, i + 1, j$ )
14     $sub \leftarrow$  CostoSustituir( $s1[i], s2[j]$ ) + FUERZABRUTA( $s1, s2, i + 1, j + 1$ )
15     $trans \leftarrow$  INT_MAX
16    if  $i + 1 < m$  and  $j + 1 < n$  and  $s1[i] = s2[j + 1]$  and  $s1[i + 1] = s2[j]$  then
17       $trans \leftarrow$  CostoTransponer( $s1[i], s1[i + 1]$ ) + FUERZABRUTA( $s1, s2, i + 2, j + 2$ )
18     $costo\_min \leftarrow$  min( $\{ins, del, sub, trans\}$ )
19    return  $costo\_min$ 

```

3) Ejecución del algoritmo

Para ejemplificar el funcionamiento del algoritmo, usaremos las cadenas abba y baba, además, todos los valores de las operaciones serán 1, excepto por la operación de transposición que tendrá valor 2, los pasos son los siguientes:

- 1) Como primer paso, se definen los valores de m y n , además se comprueba si alguno de los índices ha llegado al final, como no es el caso se sigue la ejecución.

- 2) Luego, se comparan $S1[0]='a'$ y $S2[0]='b'$, al ser distintos estos ingresan al bloque ELSE donde se calcularan el de todas las operaciones, este calculo se compone de llamar la función de costo respectiva y realizar la llamada recursiva para continuar con el resto de caracteres, probando las 4 operaciones por cada indice.
- 3) Al devolverse la llamada recursiva a la llamada raíz, se calcula el mínimo entre las operaciones iniciales, se guarda el valor y se procede a agregar las acciones a la lista. esto ocurre en cada llamada recursiva.
- 4) Como resultado de la ejecución, el programa determina que la distancia mínima de edición que transforma la cadena 'abba' en 'baba' es de 1, lo cual corresponde a realizar una transición del primer y segundo carácter. En el caso que el valor de transponer fuera 3, el coste mínimo de edición pasaría a ser producido por la combinación de inserciones y eliminaciones.

4) Complejidad temporal y espacial

Si consideramos m y n como el tamaño de las cadenas $S1$ y $S2$, respectivamente, tenemos que el algoritmo propuesto posee una complejidad temporal perteneciente a $O(4^{\min(n,m)})$, esto se debe a que el algoritmo tiene que calcular todas las posibles combinaciones, donde por cada carácter de alguna de las cadenas, existen 4 opciones factibles, además, se usa el mínimo entre los dos tamaños ya que al llegar al final de una de las cadenas, se sigue completando los caracteres restantes con operaciones de inserción o eliminación, lo cual nos lleva a tiempo lineal.

Al ser un algoritmo que utiliza recursion y además solo almacena variables de manera constante, la complejidad espacial del algoritmo pertenece a $O(\min(n, m))$ donde este mínimo corresponde a la altura del árbol de recursion.

Además, se podría considerar la memoria que se usa para almacenar las operaciones que producen la distancia de edición mínima, donde el máximo de operaciones por cada par de cadenas se define como $m+n$, por lo tanto la memoria necesaria para almacenar las operaciones en el peor caso pertenece a $O(m + n)$

5) Transposiciones y costos variables

En el algoritmo implementado, la complejidad depende directamente de la cantidad de operaciones disponibles, al agregar la operación de transposición, la complejidad aumenta desde $O(3^{\min(n,m)})$ a $O(4^{\min(n,m)})$, por otra parte, los costos variables modifican las operaciones que producen la distancia de edición mínima, pero no aumentan el orden de la complejidad, ya que estos costos están almacenados en memoria y recuperar su valor tiene un costo de $O(1)$.

2.2. Programación Dinámica

1) Solución diseñada

Como solución diseñada, se optó por implementar un algoritmo de programación dinámica que utiliza el método iterativo bottom up, el cual va construyendo la solución a medida que resuelve

los subproblemas que lo componen. En este caso, el algoritmo recibe como entrada las cadenas S_1 y S_2 a comparar y devuelve el costo de edición mínimo, además se implementó una matriz auxiliar la cual va almacenando las operaciones que producen el costo mínimo.

Para calcular la distancia de edición, el algoritmo calcula todos los subproblemas posibles y almacena los que producen el costo mínimo en una matriz, donde al completarla, el resultado estará en la última posición.

La solución diseñada fue una iteración sobre el algoritmo de programación dinámica iterativo propuesto en el blog AfterAcademy [1], donde se modificó para agregar la operación de transposición y llevar el registro de operaciones óptimas.

2) Pseudocódigo

```

1 Function PROGRAMACIONDINAMICA( $S_1, S_2$ ):
2    $m \leftarrow |S_1|, n \leftarrow |S_2|$ 
3   Matriz  $dp[m][n] = 0$ ; Para todas las casillas de la matriz
4   for  $i = 1$  to  $m$  do
5      $dp[i][0] = dp[i-1][0] + \text{costo\_eliminacion}(S_1[i-1])$ 
6   for  $j = 1$  to  $n$  do
7      $dp[0][j] = dp[0][j-1] + \text{costo\_insercion}(S_2[j-1])$ 
8   for  $i = 1$  to  $m$  do
9     for  $j = 1$  to  $n$  do
10       $\text{costoIns}, \text{costoDel}, \text{costoSub} = \text{costo\_insercion}(S_2[j-1]), \text{costo\_eliminacion}(S_1[i-1])$ 
11       $\text{costo\_sustitucion}(S_1[i-1], S_2[j-1])$ 
12       $dp[i][j] = dp[i-1][j-1] + \text{costoSub}$ 
13      if  $dp[i][j-1] + \text{costoIns} < dp[i][j]$  then
14         $dp[i][j] = dp[i][j-1] + \text{costoIns}$ 
15      if  $dp[i-1][j] + \text{costoDel} < dp[i][j]$  then
16         $dp[i][j] = dp[i-1][j] + \text{costoDel}$ 
17      if  $i + 1 < m$  and  $j + 1 < n$  and  $S_1[i] = S_2[j + 1]$  and  $S_1[i + 1] = S_2[j]$  then
18        if  $dp[i-2][j-2] + \text{costo\_transposicion}(S_1[i-2], S_1[i-1]) < dp[i][j]$  then
19           $dp[i][j] = dp[i-2][j-2] + \text{costo\_transposicion}(S_1[i-2], S_1[i-1])$ 
20   return  $dp[m][n]$ 

```

3) Ejecución del algoritmo

Para ejemplificar el funcionamiento del algoritmo, usaremos las cadenas abba y baba, además, todos los valores de las operaciones serán 1, excepto por la operación de transposición que tendrá valor 2, los pasos son los siguientes:

- 1) Como primer paso, se definen los valores de m , n y la matriz dp la cual almacena los resultados parciales, esta se inicializa en tamaño $[m+1][n+1]$, para poder manejar casos con cadenas vacías.

- 2) Luego, se llenan la primera columna y fila de la matriz, con las operaciones de eliminar y insertar para manejar los casos con cadenas vacías.
- 3) Seguido se entra al bucle principal, donde para $S_1[0]='a'$ y $S_2[0]='b'$ se calcularán todas las operaciones llamando a las funciones de costos, para luego tomar como base la operación de sustitución con la cual se realizarán las comparaciones para determinar cuál de todas las operaciones produce el menor costo, modificando el valor en la matriz dp.
- 4) Para el ejemplo propuesto, una vez llenada la matriz dp, se retornará la distancia de edición mínima, la cual corresponde a transponer los primeros dos caracteres y su costo es 1. En el caso que el valor de transponer fuera 3, el costo mínimo de edición pasaría a ser producido por la combinación de inserciones y eliminaciones.

4) Complejidad temporal y espacial

Si consideramos m y n como el tamaño de las cadenas S_1 y S_2 , respectivamente, tenemos que el algoritmo propuesto posee una complejidad temporal perteneciente a $O(m * n)$, esto se debe a que el algoritmo realiza $m \times n$ operaciones con costo constante, lo que corresponde a cada combinación de caracteres de las cadenas de entrada, además, por su naturaleza de programación dinámica, aprovecha los cálculos previos almacenados en la matriz para evitar recalcular algunos costos.

Al ser un algoritmo almacena los valores parciales en una matriz, la complejidad espacial del algoritmo pertenece a $O(m * n)$ ya que tiene que almacenar $m+1 \times n+1$ resultados, esto ocurre en todos los casos ya que siempre se llenará la matriz al completo.

Además, se podría considerar la memoria que se usa para almacenar las operaciones que producen la distancia de edición mínima, la cual funciona igual a la matriz dp, por lo tanto se necesitaría $O(m * n)$ espacio adicional, por lo cual la complejidad espacial del algoritmo se mantiene igual.

5) Transposiciones y costos variables

La inclusión de la operación de transposición no afecta a la complejidad, ya que esta depende solamente del tamaño de la entrada, por otro lado, los costos variables tampoco afectan a la complejidad, ya que al estar almacenados en un vector, el costo de consultar los valores es constante.

2.2.1. Descripción de la solución recursiva

El problema de distancia de edición extendida se puede resolver mediante la resolución y composición de subproblemas, ya que al resolver estos subproblemas obtenemos su óptimo local, lo cual al volver a componer la solución general nos entrega una solución la cual también es óptima. En específico, si tenemos $S_1[0:i]$ y $S_2[0:j]$, para resolver el problema necesitamos conocer cómo se transforma $S_1[0:i-1]$ a $S_2[0:j-1]$, así sucesivamente.

Esta solución se demuestra mediante el principio de optimalidad de Bellman [4], el cual describe lo mencionado.

2.2.2. Relación de recurrencia

Dada dos cadenas S_1 y S_2 , donde $S_1 = S_1[1..m]$ y $S_2 = S_2[1..n]$, la distancia de edición extendida $DE(i, j)$, para (i, j) hasta (m, n) , se define como:

$$DE(i, j) = \begin{cases} 0 & \text{si } i = 0 \text{ y } j = 0, \\ j \cdot \text{costo_insercion}(S_2[j-1]) & \text{si } i = 0, \\ i \cdot \text{costo_eliminacion}(S_1[i-1]) & \text{si } j = 0, \\ \min \{ DE(i-1, j-1) + \text{costo_replace}(S_1[i-1], S_2[j-1]), \\ DE(i, j-1) + \text{costo_insert}(S_2[j-1]), \\ DE(i-1, j) + \text{costo_delete}(S_1[i-1]), \\ DE(i-2, j-2) + \text{costo_trans}(S_1[i-2], S_1[i-1]) \} & \text{si } i > 0 \text{ y } j > 0. \end{cases}$$

2.2.3. Identificación de subproblemas

Los subproblemas los podemos categorizar con las 4 operaciones, entonces:

1. Inserción: Para resolver el costo de inserción con (i, j) , se necesita calcular la distancia entre los primeros i caracteres de S_1 y los primeros $j-1$ caracteres de S_2 para luego sumarle el costo de insertar el carácter $S_2[j-1]$.
2. Eliminación: Para resolver el costo de eliminación con (i, j) , se necesita calcular la distancia entre los primeros $i-1$ caracteres de S_1 y los primeros j caracteres de S_2 para luego sumarle el costo de eliminar el carácter $S_1[i-1]$.
3. Sustitución: Para resolver el costo de sustitución con (i, j) , se necesita calcular la distancia entre los primeros $i-1$ caracteres de S_1 y los primeros $j-1$ caracteres de S_2 para luego sumarle el costo de sustituir el carácter $S_1[i-1]$ por el carácter $S_2[j-1]$.
4. Transposición: Para resolver el costo de transposición con (i, j) , se necesita calcular la distancia entre los primeros $i-2$ caracteres de S_1 y los primeros $j-2$ caracteres de S_2 para luego sumarle el costo de transponer los caracteres $S_1[i-2]$ y $S_1[i-1]$.

Con esto, podemos ver que los subproblemas se solapan, lo que nos da la oportunidad de optimizar el algoritmo evitando recalculer ciertos resultados.

2.2.4. Estructura de datos y orden de cálculo

Como estructura de datos se utilizó una matriz para almacenar los resultados parciales óptimos, los cuales compondrán el resultado final, además esta matriz es llenada desde izquierda a derecha y arriba hacia abajo, esto, ya que como definimos en los subproblemas, se necesitan los resultados anteriores o con menor índice para componer la solución general.

3. Implementaciones

La extensión máxima para esta sección es de 1 página.

Los algoritmos fueron implementados en el lenguaje de programación c++, los archivos son los siguientes:

1. bf.cpp: archivo que contiene el algoritmo de fuerza bruta, además de toda la lógica para la ejecución del algoritmo, como la lectura de las matrices de costos, funciones de costos, función principal y almacenamiento de la salida en el archivo output_bf.txt. La salida por consola del programa muestra las cadenas las cuales fueron comparadas, la distancia mínima de edición, el tiempo de ejecución y si el resultado coincide con la distancia de edición esperada. El código y sus instrucciones de ejecución se pueden encontrar aquí.
2. dp.cpp: archivo que contiene el algoritmo de fuerza bruta, además de toda la lógica para la ejecución del algoritmo, como la lectura de las matrices de costos, funciones de costos, función principal y almacenamiento de la salida en el archivo output_dp.txt. La salida por consola del programa muestra las cadenas las cuales fueron comparadas, la distancia mínima de edición, el tiempo de ejecución y si el resultado coincide con la distancia de edición esperada. El código y sus instrucciones de ejecución se pueden encontrar aquí.

4. Experimentos

Para los experimentos, se utilizo el siguiente entorno:

1. Hardware

- Procesador Ryzen 7 5700x3D, 96MB Cache L3, Base clock 3.0GHz , Boost clock 4.05GHz
- 16GB Ram DDR4 3200MT/s Dual-Channel
- SSD NVMe PCIe-3.0

2. Software

- Subsistema de Windows para Linux basado en Ubuntu 22.04.5 LTS
- 16 Núcleos Virtuales asignados
- 8GB de RAM asignados
- 2GB de memoria Swap
- g++ 11.4.0
- valgrind 3.18.1

4.1. Dataset (casos de prueba)

La extensión máxima para esta sección es de 2 páginas.

Se diseñaron 10 Datasets con casos de prueba, cada dataset contiene 20 pares de cadenas y su distancia de edición esperada, la cual fue calculada con la librería de python weighted-levenshtein, que permite trabajar con el calculo de OSA y costos personalizados

Cuando se habla de matrices de costo estándar, esto hace referencia a que los costos de las operaciones son los siguientes:

- Inserción: 1, Para cualquier carácter
- Eliminación: 1, Para cualquier carácter
- Sustitución: 2, Para cualquier par de caracteres distintos
- Transposición: 1, Para cualquier par de caracteres distintos

1. Datasets

- Transposiciones con matrices de costo estándar: este dataset cuenta con cadenas de largo 8 , donde son necesarias las transposiciones, con matrices de costo estándar.
- Transposiciones con matrices de costo con valores modificados para la transposición: este dataset cuenta con cadenas de largo 8, donde son necesarias las transposiciones, con matrices de costo con valores mas elevados para la operación de transposición.

- Cadenas vacías: dataset con una cadena vacía y otra de largo entre 5 y 8, con matrices de costo estándar.
- Caracteres repetidos: dataset con par de cadenas con caracteres repetidos hasta 4 veces, de largo 8, con matrices de costo estándar.
- Palabras aleatorias con matrices de costo estándar: diversos datasets con cadenas de tamaño menor a 2, 3 a 4, 7, 8 y 12, con matrices de costo estándar, estas palabras pertenecen al diccionario ingles.
- Palabras aleatorias con matrices de costo modificado: dataset con cadenas de largo 8, estas palabras pertenecen al diccionario ingles, donde las operaciones tienen los costos: Inserción=2, Eliminación=2, Sustitución=3, Transposición=2.

La documentación y resultados de los casos de prueba se encuentran en el siguiente recurso.

4.2. Resultados

Los resultados de los casos de prueba se encuentran organizados en un recurso dividido por carpetas, donde se almacenan los archivos de salida de los programas **bf.cpp**, **dp.cpp** y el dataset correspondiente. Estos archivos contienen las cadenas comparadas, la distancia calculada, la distancia esperada correcta, un indicador con la correctitud del resultado, el tiempo de ejecución, y las operaciones que producen la distancia mínima de edición junto con sus costos. Además, al final de cada archivo, se muestra el tiempo promedio de ejecución del dataset procesado.

Para obtener los resultados a continuación, es necesario ejecutar los programas mencionados. El tiempo fue medido utilizando la biblioteca **chrono** de C++. También se considera el tiempo promedio de ejecución como medida representativa, salvo que se indique lo contrario.

El uso de memoria RAM se determinó utilizando la herramienta **Massif** de Valgrind, que toma muestras del uso de memoria durante la ejecución del programa y genera un gráfico con estas métricas.

Las instrucciones detalladas para la ejecución de los programas se encuentran aquí.

1. Transposiciones con matrices de costo estándar y costo modificado

Tipo/Data	Transposiciones costo estándar	Transposiciones costo modificado
Fuerza Bruta (ms)	188.489	193.375
Programación Dinámica (ms)	0.0956651	0.0860588
Memoria FB (KB)	103.2	105.5
Memoria PD (KB)	134.4	142.1

Cuadro 1: Tabla con valores de prueba para transposiciones

Para las pruebas de transposiciones con costo estándar, podemos observar que al algoritmo de fuerza bruta le toma mucho mas tiempo en resolver la consulta en comparación al algoritmo de programación dinámica, además el uso de memoria es mayor para el algoritmo de programación dinámica, pero esta diferencia en uso de memoria no es proporcional a la diferencia en tiempo de ejecución.

Para las pruebas de transposiciones con costo modificado, se sigue la misma tendencia en términos de tiempo de ejecución y uso de memoria que para las transposiciones con costo estándar.

No obstante, lo que si cambió fueron las operaciones las cuales producen la distancia de edición mínima, para las transposiciones con costo estándar, siempre se selecciono la operación de transposición (Esto ocurre para ambos algoritmos). Por otro lado, para las transposiciones con costo modificado, al aumentar el costo de realizar una transposición, el algoritmo determinó que se podía llegar a la distancia de edición mínima mediante otras operaciones. Ejemplo:

```
Strings: ijklnop , jilknmpo
Distancia calculada: 4, Distancia esperada: 4
Resultado correcto
Tiempo de ejecución: 187.242 ms
Operaciones óptimas:
- Transponer o y p (Costo: 1)
- Transponer m y n (Costo: 1)
- Transponer k y l (Costo: 1)
- Transponer i y j (Costo: 1)
```

(a) Transposiciones con costo estándar.

```
Strings: ijklnop , jilknmpo
Distancia calculada: 8, Distancia esperada: 8
Resultado correcto
Tiempo de ejecución: 190.811 ms
Operaciones óptimas:
- Eliminar p (Costo: 1)
- Eliminar n (Costo: 1)
- Insertar p (Costo: 1)
- Eliminar l (Costo: 1)
- Insertar n (Costo: 1)
- Eliminar j (Costo: 1)
- Insertar l (Costo: 1)
- Insertar j (Costo: 1)
```

(b) Transposiciones con costo modificado.

Figura 1: Ejemplo ejecución de transposiciones.

Por lo tanto, al modificar los costos de las operaciones, no se modifica en gran medida el tiempo de ejecución de los algoritmos al igual con el uso de memoria, pero si las operaciones las cuales producen la distancia de edición mínima. Por lo tanto, se evidencia que los costos de de las operaciones no afectan a la complejidad temporal y espacial.

2. Cadenas vacías y caracteres repetidos

Tipo/Data	Cadenas vacías	Caracteres repetidos
Fuerza Bruta (ms)	0.0033952	183.942
Programación Dinámica (ms)	0.0099436	0.0952444
Memoria FB (KB)	97.15	106.5
Memoria PD (KB)	100.7	143.9

Cuadro 2: Tabla con valores de prueba para cadenas vacías y caracteres repetidos

Para el dataset de cadenas vacías podemos observar que al algoritmo de fuerza bruta le toma menos tiempo en resolver la consulta con respecto al de programación dinámica. Por otro lado, el uso de memoria es similar en los dos casos.

Estos resultados se deben a las complejidades temporales y espaciales de cada algoritmo, para el algoritmo de fuerza bruta, su tiempo de ejecución es exponencial con respecto al tamaño mas pequeño de la entrada, al tener una entrada con largo 0, el algoritmo resuelve el problema muy eficientemente, podríamos decir que le toma tiempo $4^0 = 1$.

Por el lado del algoritmo de programación dinámica, su tiempo de ejecución se modela según los tamaños de las dos entradas mas 1, por lo tanto, podríamos decir que le toma tiempo $1 * 9 = 9$, lo cual es mayor que el algoritmo de fuerza bruta.

Ademas, los usos de memoria son similares ya que el algoritmo de programación dinámica basa su uso de memoria en los tamaños de las dos entradas, al tener una de las entradas pequeñas, el uso de memoria disminuye.

Sobre el dataset de caracteres repetidos, podemos observar que el algoritmo de fuerza bruta es mas ineficiente con respecto al de programación dinámica, no obstante, el uso de memoria es mayor en este ultimo.

Con esto, podemos evidenciar en la practica que las complejidades temporales y espaciales para cada algoritmo se cumplen.

3. Palabras aleatorias con matrices de costo estándar y modificado

Data/Tipo	Fuerza Bruta (ms)	Programación Dinámica (ms)	Memoria FB (KB)	Memoria PD (KB)
Menor a 2	0.00890745	0.0101063	97.55	98.02
3 a 4	0.15021	0.0225789	99.37	104
7	33.0569	0.0653561	105.2	131.1
8	180.548	0.0875144	108.2	145.6
8 Modificado	181.627	0.0992815	105.5	145.2
12	172570	0.219571	No medido	252.5

Cuadro 3: Tabla con valores de prueba para cadenas aleatorias

En esta sección, los resultados obtenidos, como las gráficas o tablas, deben estar respaldados por los datos generados durante la ejecución de sus programas. Es fundamental que, junto con el informe, se adjunten los archivos que contienen dichos datos para permitir su verificación. Además, se debe permitir y especificar como obtener esos archivos desde una ejecución en otro computador (otra infraestructura para hacer los experimentos).

No es necesario automatizar la generación de las gráficas, pero sí es imprescindible que se pueda confirmar que las visualizaciones presentadas son producto de los datos generados por sus algoritmos, aunque la trazabilidad de los datos hasta las visualizaciones es esencial para garantizar que su validez: describa cómo se generaron los datos, cómo se procesaron y cómo se visualizaron de manera que pueda ser replicado por quien lea su informe.

Agregue gráficas que muestren los resultados de sus experimentos. La cantidad de páginas es limitada, por lo tanto escoja las gráficas más representativas y que muestren de manera clara los resultados obtenidos. Esta elección es parte de lo que se evaluara en la sección de presentación de resultados. Referencie las figuras en el texto, describa lo que se observa en ellas y por qué son relevantes.

En la [fig. 2](#) se muestra un scatterplot hecho con [TikZ](#) con el tamaño ideal cuando se incluyen dos figuras. Queda a criterio de usted el decidir qué figuras incluir.

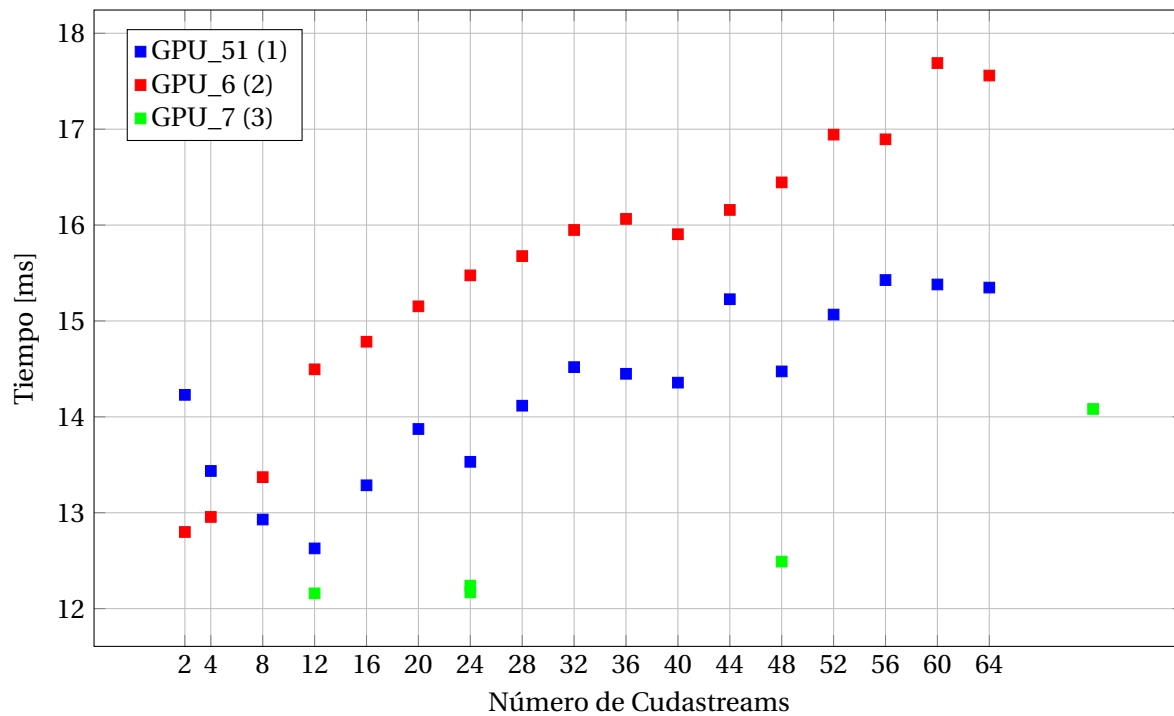


Figura 2: Ejemplo de scatterplot hecho con tikz. Tamaño ideal 1.

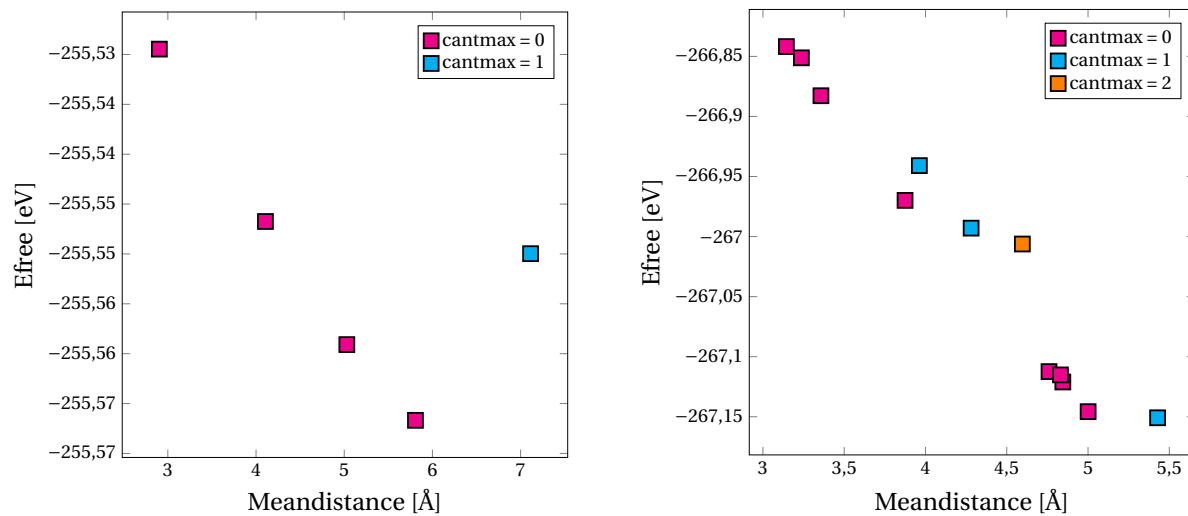


Figura 3: Ejemplo de scatterplot hecho con tikz. Tamaño ideal 2.

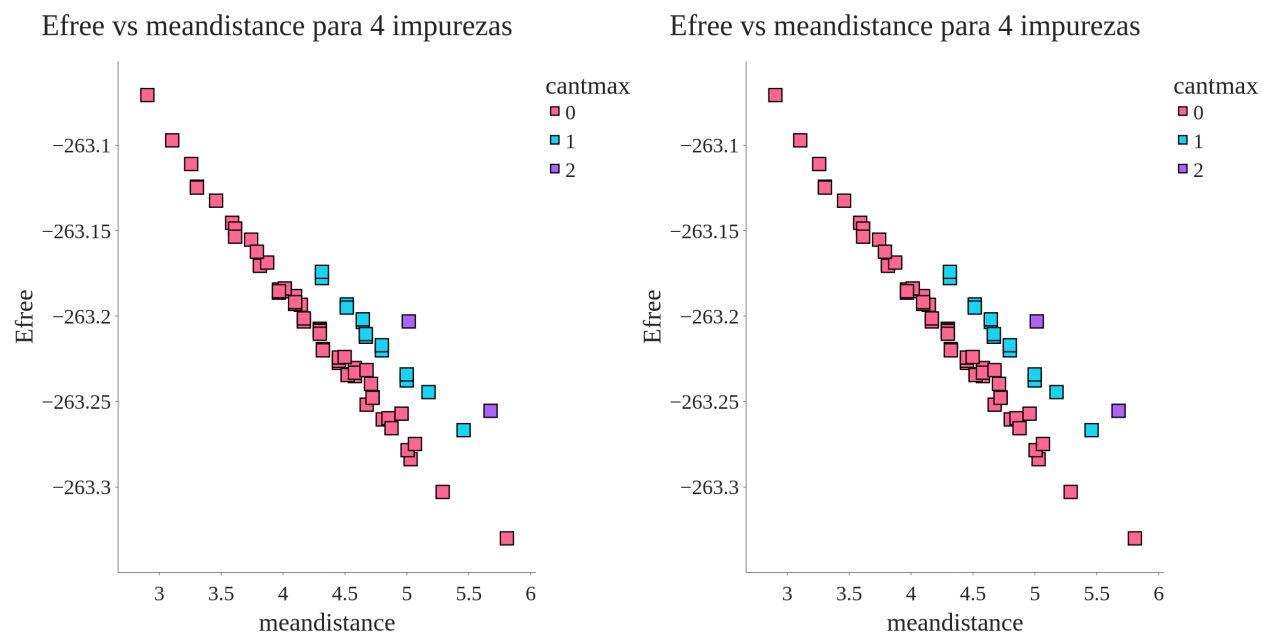


Figura 4: Ejemplo de scatterplot hecho con matplotlib.

Recuerde que es imprescindible que se pueda replicar la generación de las gráficas, por lo que usted debe incluir cómo generó esos datos y cómo podría generarlos la persona que revisa su entrega y ejecuta sus programas. Por ejemplo, si genera un scatterpolot con Tikz, usted debe explicar cómo obtener la tupla de valores que se usaron para generar la gráfica.

5. Conclusiones

La extensión máxima para esta sección es de 1 página.

La conclusión de su informe debe enfocarse en el resultado más importante de su trabajo. No se trata de repetir los puntos ya mencionados en el cuerpo del informe, sino de interpretar sus hallazgos desde un nivel más abstracto. En lugar de describir nuevamente lo que hizo, muestre cómo sus resultados responden a la necesidad planteada en la introducción.

- No vuelva a describir lo que ya explicó en el desarrollo del informe. En cambio, interprete sus resultados a un nivel superior, mostrando su relevancia y significado.
- Aunque no debe repetir la introducción, la conclusión debe mostrar hasta qué punto logró abordar el problema o necesidad planteada en el inicio. Reflexione sobre el éxito de su análisis o experimento en relación con los objetivos propuestos.
- No es necesario restablecer todo lo que hizo (ya lo ha explicado en las secciones anteriores). En su lugar, centre la conclusión en lo que significan sus resultados y cómo contribuyen al entendimiento del problema o tema abordado.
- No deben centrarse en sí mismos o en lo que hicieron durante el trabajo (por ejemplo, evitando frases como "primero hicimos esto, luego esto otro...").
- Lo más importante es que no se incluyan conclusiones que no se deriven directamente de los resultados obtenidos. Cada afirmación en la conclusión debe estar respaldada por el análisis o los datos presentados. Se debe evitar extraer conclusiones generales o excesivamente amplias que no puedan justificarse con los experimentos realizados.

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

Aquí puede agregar tablas, figuras u otro material que no se incluyó en el cuerpo principal del documento, ya que no constituyen elementos centrales de la tarea. Si desea agregar material adicional que apoye o complemente el análisis realizado, puede hacerlo en esta sección.

Esta sección es solo para material adicional. El contenido aquí no será evaluado directamente, pero puede ser útil si incluye material que será referenciado en el cuerpo del documento. Por lo tanto, asegúrese de que cualquier elemento incluido esté correctamente referenciado y justificado en el informe principal.

Referencias

- [1] AfterAcademy. *Edit Distance*. <https://afteracademy.com/blog/edit-distance-problem/>. Abr. de 2022.
- [2] Diego Arroyuelo. *Algoritmos Discretos: Análisis y Diseño*. Mar. de 2022.
- [3] Elsevier. *Differentiating between an introduction and abstract in a research paper*. Accessed: 2024-10-02, 2024. URL: <https://scientific-publishing.webshop.elsevier.com/manuscript-preparation/differentiating-between-and-introduction-research-paper/>.
- [4] Rossel Keila. *Optimización con programación dinámica*. Ene. de 2022.