REPORTE TAREA 2 y 3
ALGORITMOS Y COMPLEJIDAD

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

Carlos Vera Quezada

16 de noviembre de 2024

17:53

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

> "La algoritmia es una de las bases fundamentales de ciencias de la computación. Aunque otras areas 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 areas. Es, de alguna manera, un area 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 numero mínimo de operaciones para transformar una cadena de caracteres en otra, sin modificar mas 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 esta 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. Ademas, 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 traves 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 asi determinar cual 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 mas larga, la complejidad espacial del algoritmo pertenece a $O(m)$ (Que corresponde a la profundidad de la pila de recursion). 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 practica, 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 usara el promedio de varias ejecuciones para dar un resultado mas significativo.

2. Diseño y Análisis de Algoritmos

2.1. Fuerza Bruta

1) Solución diseñada:

Como solución diseñada, se optó por implementar un algoritmo de fuerza bruta recursiva, el cual recibe como parámetros dos cadenas S1 y S2, dos contadores i y j que representan un indice de las cadenas S1 y S2, respectivamente, los cuales serán comparados. Ademas, la función recibe un lista de operaciones, la cual se usa para almacenar las operaciones optimas que producen la distancia minima de edición.

Para calcular la distancia minima de edición, el algoritmo calculara todas las posibles combinaciones de operaciones las cuales transforman la cadena S1 en S2, donde se quedara con la combinación la cual produzca el menor costo, estos costos están asociados a un matriz de costos definidas para cada tipo de operación.

La solución diseñada fue una iteración sobre al algoritmo de fuerza bruta recursiva propuesto en el blog AfterAcademy [1] , donde se modifico para agregar la operación de transposición y llevar el registro de operaciones optimas.

2) Pseudocódigo:

```
1  Function FuerzaBruta(s1, s2, i, j):
2      m ← |s1|, n ← |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 ← CostoInsertar(s2[j]) + FuerzaBruta(s1, s2, i, j + 1)
13         del ← CostoEliminar(s1[i]) + FuerzaBruta(s1, s2, i + 1, j)
14         sub ← CostoSustituir(s1[i], s2[j]) + FuerzaBruta(s1, s2, i + 1, j + 1)
15         trans ← 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 ← CostoTransponer(s1[i], s1[i + 1]) + FuerzaBruta(s1, s2, i + 2, j + 2)
18     costo_min ← 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, ademas, 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, ademas se comprueba si alguno de los indices 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 .^{a}bba.^{en} "baba.^{es} de 1, lo cual corresponde a realizar una transición del primer y segundo carácter, este valor es retornado a donde fue llamada la función en un inicio.
5) 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 $O\left(4^{\text{mín}(n,m)}\right)$, 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, ademas, 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 ademas solo almacena variables de manera lineal, la complejidad espacial del algoritmo pertenece a $O(\text{mín}(n,m))$ donde este mínimo corresponde a la altura del árbol de recursion.

Ademas, se podría considerar la memoria que se usa para almacenar las operaciones que producen la distancia de edición minima, 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 $O\left(3^{\text{mín}(n,m)}\right)$

a $O\left(4^{\min(n,m)}\right)$, por otra parte, los costos variables modifican las operaciones que producen la distancia de edición minima, 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 opto 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, ademas se implemento 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 ultima posición.

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

2) Pseudocódigo

```
1  Function PROGRAMACIONDINAMICA(S1, S2):
2      m ← |S1|, n ← |S2|
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] + costo_eliminacion(S1[i-1])
6      for j = 1 to n do
7          dp[0][j] = dp[0][j-1] +costo_insercion(S2[j-1])
8      for i = 1 to m do
9          for j = 1 to n do
10             costoIns, costoDel, costoSub = costo_insercion(S2[j-1]) , costo_eliminacion(S1[i-1])
                costo_sustitucion(S1[i-1], S2[j-1])
11             dp[i][j] = dp[i-1][j-1] + costoSub
12             if dp[i][j-1] + costoIns <dp[i][j] then
13                 dp[i][j] = dp[i][j-1] + costoIns
14             if dp[i-1][j] + costoDel <dp[i][j] then
15                 dp[i][j] = dp[i-1][j] + costoDel
16             if i + 1 <m and j + 1 <n and S1[i] = S2[j + 1] and S1[i + 1] = S2[j] then
17                 if dp[i-2][j-2] + costo_transposicion(S1[i-2], S1[i-1]) <dp[i][j] then
18                     dp[i][j] = dp[i-2][j-2] + costo_transposicion(S1[i-2], S1[i-1])
19     return dp[m][n]
```

3) Ejecución del algoritmo

Para ejemplificar el funcionamiento del algoritmo, usaremos las cadenas abba y baba, ademas, 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 calcularan todas las operaciones llamando a las funciones de costos, para luego tomar como base la operación se sustitución con la cual se realizaran las comparaciones para determinar cual 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 retornara la distancia de edición minima, la cual corresponde a transponer los primeros dos caracteres y su costo es 1.

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 $O(n^2)$, esto se debe a que el algoritmo realiza m x n operaciones con costo lineal, lo que corresponde a cada combinación de caracteres de las cadenas de entrada, ademas, 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 $O(n^2)$ ya que tiene que almacenar m+1 x n+1 resultados, esto ocurre en todos los casos ya que siempre se llenara la matriz al completo.

Ademas, se podría considerar la memoria que se usa para almacenar las operaciones que producen la distancia de edición minima, la cual funciona igual a la matriz dp, por lo tanto se necesaria $O(n^2)$ espacio adicional, por lo cual la complejidad espacial del algoritmo se mantiene igual.

5) Transposiciones y costos variables

La inclusion 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 lineal.

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 optimo local, lo cual al

volver a componer la solución general nos entrega una solución la cual también es optima, En especifico, si tenemos S_1[0:i] y S_2[0:j], para resolver el problema necesitamos conocer como se transforma S_1[0:i-1] a S_2[0:j-1], asi sucesivamente.

Esta solución se demuestra mediante el principio de optimalidad de Bellman [5], 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 DistanciaExtendida(i, j), para (i,j) hasta (m,n), se define como:

$$\text{DistanciaExtendida}(i,j) = \begin{cases} 0 & \text{si } i = 0 \text{ y } j = 0, \\ j \cdot \text{costo\_insercion}(S_2[j-1]) & \text{si } i = 0, \\ i \cdot \text{costo\_eliminacion}(S_1[i-1]) & \text{si } j = 0, \\ \min\{\text{DistanciaExtendida}(i-1,j-1) + \text{costo\_replace}(S_1[i-1], S_2[j-1]), & \\ \quad \text{DistanciaExtendida}(i,j-1) + \text{costo\_insert}(S_2[j-1]), & \\ \quad \text{DistanciaExtendida}(i-1,j) + \text{costo\_delete}(S_1[i-1]), & \\ \quad \text{DistanciaExtendida}(i-2,j-2) + \text{costo\_trans}(S_1[i-2], S_1[i-1])\} & \text{si } i > 0 \text{ y } j > 0. \end{cases}$$

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 recalcular ciertos resultados.

2.2.4. Estructura de datos y orden de cálculo

Como estructura de datos se utilizo una matriz para almacenar los resultados parciales óptimos, los cuales compondrán el resultado final, ademas 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 indice para componer la solución general.

3. Implementaciones

La extensión máxima para esta sección es de 1 página.

Aquí deben explicar la estructura de sus programas haciendo referencias a los archivos y funciones de su entrega. No adjunte código en esta sección.

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.

4. Experimentos

La extensión máxima para esta sección es de 6 página.

> *"Non-reproducible single occurrences are of no significance to science."*
>
> —Popper,2005 [6]

En la sección de Experimentos, es fundamental detallar la infraestructura utilizada para asegurar la reproducibilidad de los resultados, un principio clave en cualquier experimento científico. Esto implica especificar tanto el hardware (por ejemplo, procesador Intel Core i7-9700K, 3.6 GHz, 16 GB RAM DDR4, almacenamiento SSD NVMe) como el entorno software (sistema operativo Ubuntu 20.04 LTS, compilador g++ 9.3.0, y cualquier librería relevante). Además, se debe incluir una descripción clara de las condiciones de entrada, los parámetros utilizados y los resultados obtenidos, tales como tiempos de ejecución y consumo de memoria, que permitan a otros replicar los experimentos en entornos similares. *La replicabilidad es un aspecto crítico para validar los resultados en la investigación científica computacional* [4].

4.1. Dataset (casos de prueba)

La extensión máxima para esta sección es de 2 páginas.

Es importante generar varias muestras con características similares para una misma entrada, por ejemplo, variando tamaño del input dentro de lo que les permita la infraestructura utilizada en ests informe, con el fin de capturar una mayor diversidad de casos y obtener un análisis más completo del rendimiento de los algoritmos.

Aunque la implementación de los algoritmos debe ser realizada en C++, se recomienda aprovechar otros lenguajes como Python para automatizar la generación de casos de prueba, ya que es más amigable para crear gráficos y realizar análisis de los resultados. Python, con sus bibliotecas como `matplotlib` o `pandas`, facilita la visualización de los datos obtenidos de las ejecuciones de los distintos algoritmo bajo diferentes escenarios.

Debido a la naturaleza de las pruebas en un entorno computacional, los tiempos de ejecución pueden variar significativamente dependiendo de factores externos, como la carga del sistema en el momento de la ejecución. Por lo tanto, para obtener una medida más representativa, siempre es recomendable ejecutar múltiples pruebas con las mismas características de entrada y calcular el promedio de los resultados.

4.2. Resultados

La extensión máxima para esta sección es de 4 páginas.

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 especficiar como obtener esos archivos desde una ejecución en otro computador (otra infraestructura para hacer lso 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. 1 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.

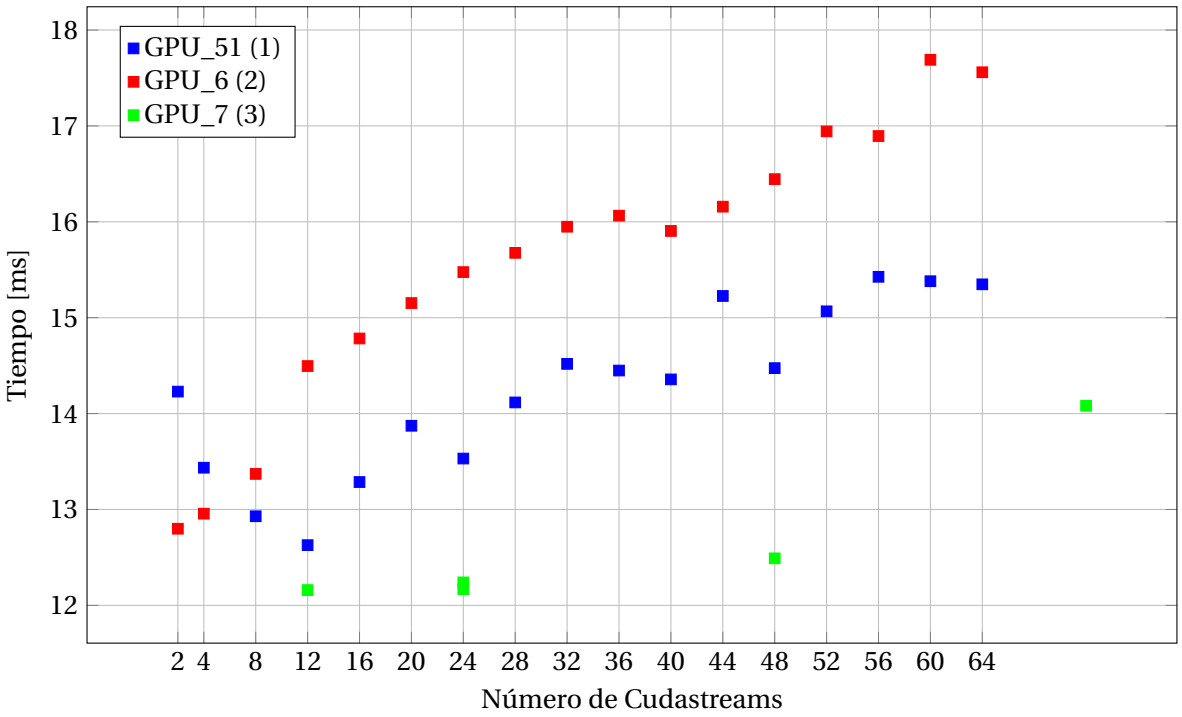

Figura 1: Ejemplo de scatterplot hecho con tikz. Tamaño ideal 1.

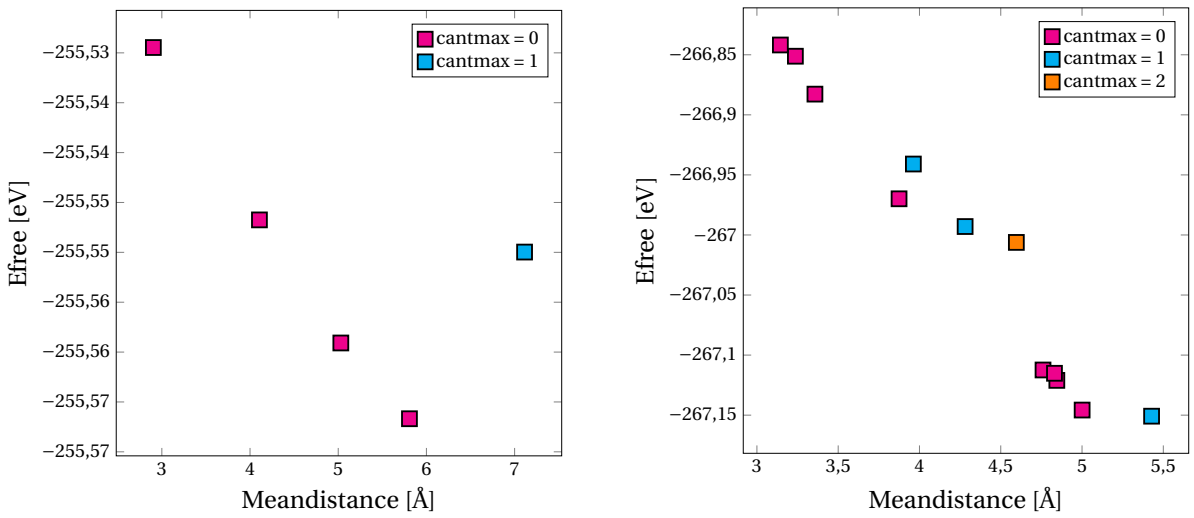

Figura 2: Ejemplo de scatterplot hecho con tikz. Tamaño ideal 2.

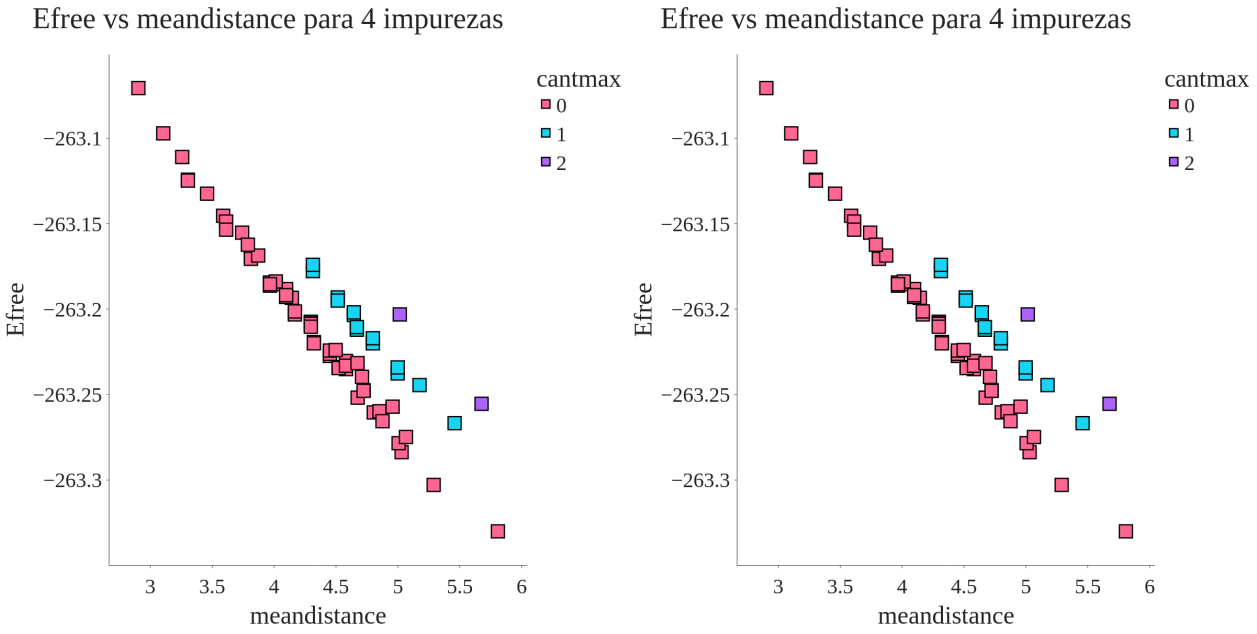

Figura 3: Ejemplo de scatterplot hecho con matplotlib.

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 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

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] Jorge Fonseca y Kazem Taghva. «The State of Reproducible Research in Computer Science». En: ene. de 2020, págs. 519-524. ISBN: 978-3-030-43019-1. DOI: 10.1007/978-3-030-43020-7_68.

[5] Rossel Keila. *Optimización con programación dinámica*. Ene. de 2022.

[6] K. Popper. *The Logic of Scientific Discovery*. Routledge Classics. Taylor & Francis, 2005. ISBN: 9781134470020. URL: https://books.google.cl/books?id=LWSBAgAAQBAJ.