REPORTE TAREA 2 y 3
ALGORITMOS Y COMPLEJIDAD

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

Carlos Vera Quezada

15 de noviembre de 2024

00:31

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 [4].

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 [3]

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. [3]

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

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* [6] para la presentación de algoritmos. Se recomienda consultar *Algorithm2e on CTAN* [7] y *Writing Algorithms in LaTeX* [9] and *The Levenshtein Distance Algorithm* [2].

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

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, operaciones):
2      m ← tamaño de s1
3      n ← tamaño de s2
4      if i = m and j = n then
5          return 0
6      if i = m then
7          operaciones.push_back( Insertar s2[j] CostoInsertar(s2[j]) )
8          return CostoInsertar(s2[j]) + FUERZABRUTA(s1, s2, i, j + 1, operaciones)
9      if j = n then
10         operaciones.push_back( Eliminar s1[i] CostoEliminar(s1[i]) )
11         return CostoEliminar(s1[i]) + FUERZABRUTA(s1, s2, i + 1, j, operaciones)
12     Inicializar op_ins, op_del, op_sub, op_trans como vectores de string vacíos
13     Declarar ins, del, sub, trans como enteros
14     if i <m and j <n and s1[i] = s2[j] then
15         return FUERZABRUTA(s1, s2, i + 1, j + 1, operaciones)
16     else
17         ins ← CostoInsertar(s2[j]) + FUERZABRUTA(s1, s2, i, j + 1, op_ins)
18         del ← CostoEliminar(s1[i]) + FUERZABRUTA(s1, s2, i + 1, j, op_del)
19         sub ← CostoSustituir(s1[i], s2[j]) + FUERZABRUTA(s1, s2, i + 1, j + 1, op_sub)
20         trans ← INT_MAX
21         if i + 1 <m and j + 1 <n and s1[i] = s2[j + 1] and s1[i + 1] = s2[j] then
22             trans ← CostoTransponer(s1[i], s1[i + 1]) + FUERZABRUTA(s1, s2, i + 2, j + 2, op_trans)
23     costo_min ← min({ins, del, sub, trans})
24     if costo_min = ins then
25         operaciones ← op_ins
26         operaciones.push_back( Insertar s2[j] CostoInsertar(s2[j]) )
27     else if costo_min = del then
28         operaciones ← op_del
29         operaciones.push_back( Eliminar s1[i] CostoEliminar(s1[i]) )
30     else if costo_min = sub then
31         operaciones ← op_sub
32         operaciones.push_back( Sustituir s1[i] por s2[j] CostoSustituir(s1[i], s2[j]) )
33     else if costo_min = trans then
34         operaciones ← op_trans
35         operaciones.push_back( Transponer s1[i] y s1[i + 1] CostoTransponer(s1[i], s1[i + 1]) )
36     return costo_min
```

3) Ejecución

Para ejemplificar el funcionamiento del algoritmo, usaremos las cadenas abba y baba, al iniciar el

algoritmo se le entregan los siguientes parámetros

s1= abba s2= baba i=0 j=0 operaciones=op

además, para el funcionamiento de las funciones auxiliares de costos hay que definir los archivos .txt correspondientes, los valores para el ejemplo son:

Costo insercion= 1 para todas las letras del abecedario ingles en minúscula Costo eliminacion= 1 para todas las letras del abecedario ingles en minúscula Costo sustitucion= 2 para todo par de letras a sustituir del abecedario ingles en minúscula (excepto para un par idéntico (char1,char1)=0) Costo transpocion= 1 para todo par de letras a transponer del abecedario ingles en minúscula (excepto para un par idéntico (char1,char1)=0)

Pasos:

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 costo de insertar, eliminar y sustituir, en este caso ademas se calculará el costo de transponer, este calculo se compone de llamar la función de costo respectiva e 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 raiz, se calcula el minimo 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) Cabe destacar, que si el valor de transponer fuera 2, el algoritmo hubiera determinado que existen dos caminos los cuales producen el mismo costo de edición mínimo, transponer el primer y segundo carácter (coste 2) o eliminar e insertar los caracteres correspondientes (coste 2), por lo cual, las dos respuestas son validas.

6) 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.

2.2. Programación Dinámica

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

Erickson, 2019 [5]

1) Describa la solución recursiva.

2) Escriba la relación de recurrencia, incluyendo condiciones y casos base.

3) Identifique subproblemas.

4) Defina estructura de datos a utilizar y especifique el orden de calculo que realiza su programa que utiliza programación dinámica.

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 1: Este es solo un ejemplo de cómo estructurar el pseudocódigo, con retornos explícitos y llamados a funciones.

```
1  Procedure AlgorithmName(S1, S2)
2      if S1 está vacía then
3          return longitud de S2
4      else if S2 está vacía then
5          return longitud de S1
6      else if S1[0] = S2[0] then
7          return AlgorithmName(S1[1:], S2[1:])
8      else
9          costo ← AuxiliaryFunction(S1, S2)
10         return costo
11 Procedure AuxiliaryFunction(S1, S2)
12     if S1 y S2 son similares then
13         return algún valor o costo
14     else
15         return AlgorithmName(S1 modificado, S2)
```

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 [10]

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* [8].

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.

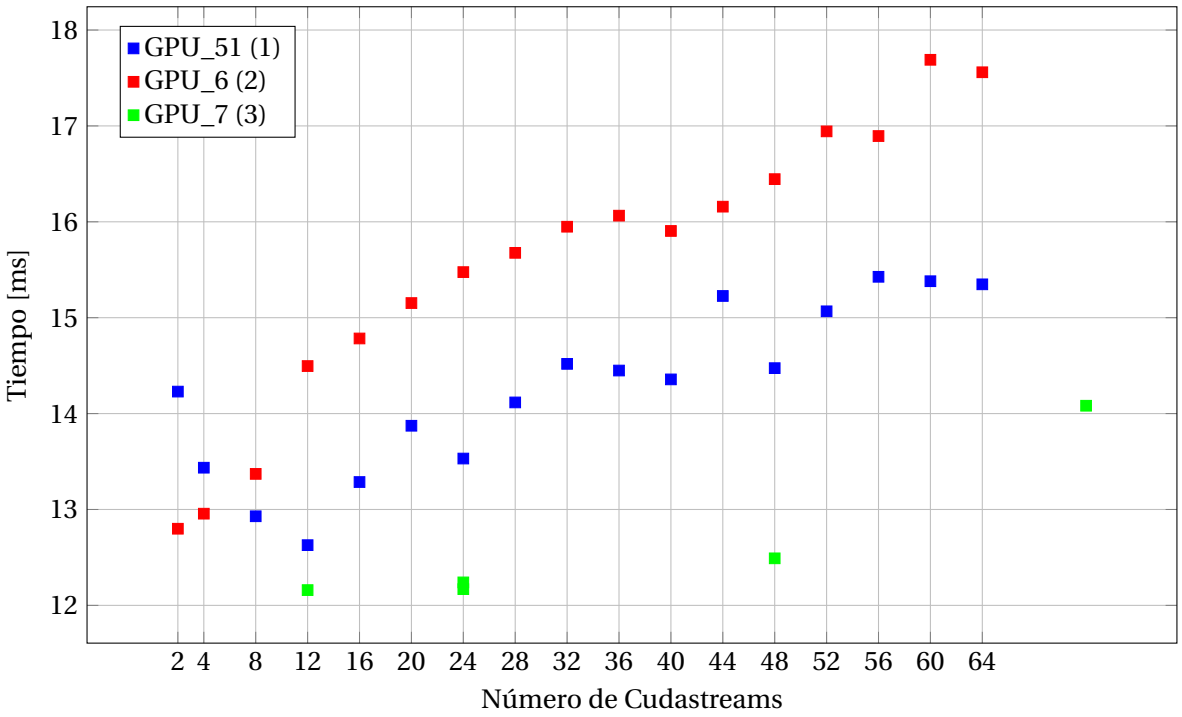

Figura 1: Ejemplo de scatterplot hecho con tikz. Tamaño ideal 1.

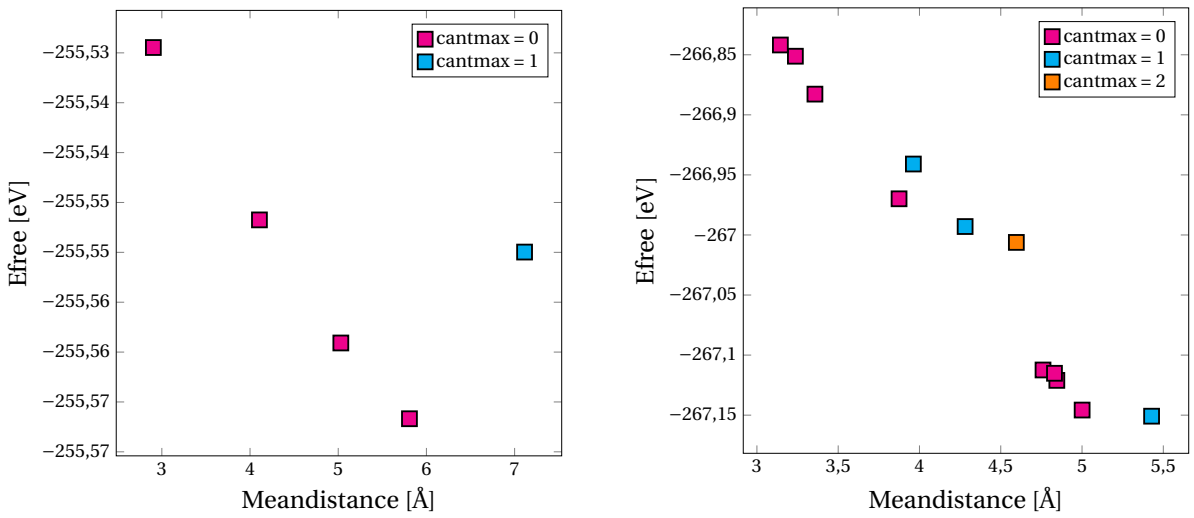

Figura 2: Ejemplo de scatterplot hecho con tikz. Tamaño ideal 2.

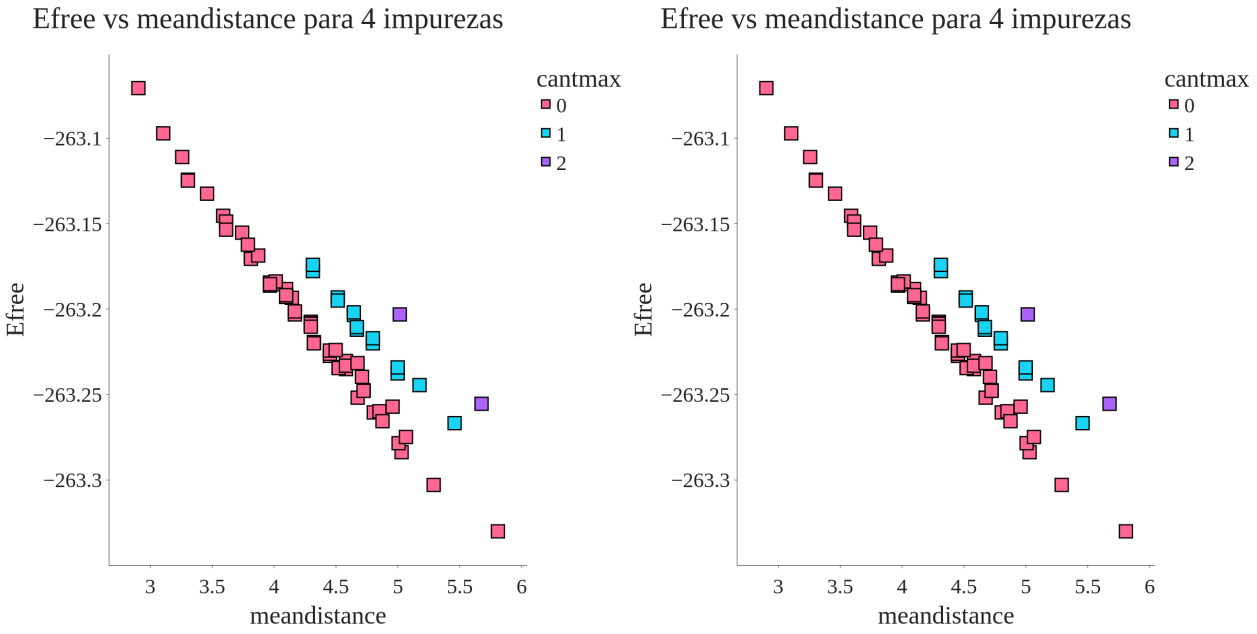

Figura 3: Ejemplo de scatterplot hecho con matplotlib.

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] Eray Araz. *The Levenshtein Distance Algorithm*. https://medium.com/@erayaraz10/the-levenshtein-distance-algorithm-cf6407d25d16. 2023.

[3] Diego Arroyuelo. *Algoritmos Discretos: Análisis y Diseño*. Mar. de 2022.

[4] 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/.

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

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

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

[8] 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.

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

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