REPORTE TAREA 2 y 3
ALGORITMOS Y COMPLEJIDAD

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

Andrés Águila Montenegro

17 de noviembre de 2024

23:33

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 [1].

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 extensión máxima para esta sección es de 2 páginas.

La introducción de este tipo de informes o reportes, tiene como objetivo principal **contextualizar el problema que se va a analizar**, proporcionando al lector la información necesaria para entender la relevancia del mismo.

Es fundamental que en esta sección se presenten los antecedentes del problema, destacando investigaciones previas o principios teóricos que sirvan como base para los análisis posteriores. Además, deben explicarse los objetivos del informe, que pueden incluir la evaluación de un algoritmo, la comparación de métodos o la validación de resultados experimentales.

Aunque la estructura y el enfoque siguen principios de trabajos académicos, se debe recordar que estos informes no son publicaciones científicas formales, sino trabajos de pregrado. Por lo tanto, se busca un enfoque claro y directo, que permita al lector comprender la naturaleza del problema y los objetivos del análisis, sin entrar en detalles excesivos.

Introduction Checklist de *How to Write a Good Scientific Paper* [2], adaptada a nuestro contexto:

- Indique el **campo del trabajo** (Análisis y Diseño de algoritmos en Ciencias de la Computación), por qué este campo es importante y qué se ha hecho ya en este área, con las **citas** adecuadas de la literatura académica o fuentes relevantes.
- Identifique una **brecha** en el conocimiento, un desafío práctico, o plantee una **pregunta** relacionada con la eficiencia, complejidad o aplicabilidad de un algoritmo particular.
- Resuma el propósito del informe e introduzca el análisis o experimento, dejando claro qué se está investigando o comparando, e indique **qué es novedoso** o por qué es significativo en el contexto de un curso de pregrado.
- Evite; repetir el resumen; proporcionar información innecesaria o fuera del alcance de la materia (limítese al análisis de algoritmos o conceptos de complejidad); exagerar la importancia del trabajo (recuerde que se trata de un informe de pregrado); afirmar novedad sin una comparación adecuada con lo enseñado en clase o la bibliografía recomendada.

Recuerde que este es su trabajo, y sólo usted puede expresar con precisión lo que ha aprendido y quiere transmitir. Si lo hace bien, su introducción será más significativa y valiosa que cualquier texto automatizado. ¡Confíe en sus habilidades, y verá que puede hacer un mejor trabajo que cualquier herramienta que automatiza la generación de texto!

2. Diseño y Análisis de Algoritmos

La extensión máxima para esta sección es de 5 páginas.

2.1. Fuerza Bruta

2.1.1. Descripción de la solución

Resolver el problema de distancia de edición utilizando el enfoque de fuerza bruta no tiene gran complejidad en su diseño. Buscamos igualar s1 a s2 probando todas las combinaciones posibles que surgen a partir de las 4 operaciones que dicta el problema. Para este ejemplo se compara recursivamente y de manera ascendente los elementos de s1 y s2 con ayuda de índices (i y j). Por cada par de caracteres ubicados en s1[i] y s2[j], se evaluan en casos separados las cuatro acciones posibles a realizar con ambos elementos, agregando el costo relacionado con la operación. Cada caso representaría una *"ruta"* por la cual se buscaría llegar a la solución. Luego de calcular cada uno de los casos, se escoje el de menor costo, retornando el costo óptimo de cada subtarea y, finalmente, el costo óptimo del problema original.

2.1.2. Algoritmo utilizando fuerza bruta

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

```
1  Procedure BRUTEFORCE(s1, s2, i, j)
2      if i = longitud(s1) and j = longitud(s2) then
3          return 0                                              // Ambas cadenas están completas
4      else if i = longitud(s1) then
5          return (COSTINSERT(s2[j]) + BRUTEFORCE(s1, s2, i, j + 1))   // Insertar restante de s2
6      else if j = longitud(s2) then
7          return (COSTDELETE(s1[i]) + BRUTEFORCE(s1, s2, i + 1, j))   // Eliminar restante de s1
       // Calcula el costo mínimo entre las operaciones posibles
8      costo_del ← COSTDELETE(s1[i]) + BRUTEFORCE(s1, s2, i + 1, j)
9      costo_ins ← COSTINSERT(s2[j]) + BRUTEFORCE(s1, s2, i, j + 1)
10     costo_sub ← COSTSUBSTITUTE(s1[i], s2[j]) + BRUTEFORCE(s1, s2, i + 1, j + 1)
11     costo_trans ← ∞
12     if i + 1 < largo s1 and j + 1 < largo s2 and s1[i] = s2[j+1] and s1[i+1] = s2[j] then
13         costo_trans ← COSTTRANSPOSE(s1[i], s1[i+1]) + BRUTEFORCE(s1, s2, i + 2, j + 2)
14     return mín(costo_del, costo_ins, costo_sub, costo_trans)
15 Procedure COSTINSERT(b)
16     return costo de inserción para 'b'                        // Función de costo de inserción
17 Procedure COSTDELETE(a)
18     return costo de eliminación para 'a'                      // Función de costo de eliminación
19 Procedure COSTSUBSTITUTE(a, b)
20     return costo de sustitución entre 'a' y 'b'               // Función de costo de sustitución
21 Procedure COSTTRANSPOSE(a, b)
22     return costo de transposición entre 'a' y 'b'             // Función de costo de transposición
```

2.1.3. Ejemplo de ejecución

Vamos a simular la ejecución del algoritmo diseñado con los datos de entrada s1: “dog” y s2: “bag”.

Paso 1::

// Se checkea si es un caso base

$costo_{del} \leftarrow$ COSTDELETE(*s1[i]*) + BRUTEFORCE(*s1, s2, i + 1, j*) **Vuelve al paso 1 con otros parámetros**:

$costo_{ins} \leftarrow$ COSTINSERT(*s2[j]*) + BRUTEFORCE(*s1, s2, i, j + 1*) **Vuelve al paso 1 con otros parámetros**:

$costo_{sub} \leftarrow$ COSTSUBSTITUTE(*s1[i], s2[j]*) + BRUTEFORCE(*s1, s2, i + 1, j + 1*) **Vuelve al paso 1 con otros parámetros**:

// Solo si es que se cumple la condición

$costo_{trans} \leftarrow$ COSTTRANSPOSE(*s1[i], s1[i+1]*) + BRUTEFORCE(*s1, s2, i + 2, j + 2*) **Vuelve al paso 1 con otros parámetros**:

Paso 2:

Al finalizar todas las llamadas recursivas se evaluan los resultados con:

$$\text{mín}(costo_{del}, costo_{ins}, costo_{sub}, costo_{trans})$$

y se retorna a la llamada anterior, para que esta **repita el paso 2** o en su defecto, retorne el valor a la llamada original, resolviendo el problema.

2.1.4. Complejidad temporal y espacial

En cada instancia de este algoritmo, este realiza al menos 3 llamadas recursivas a si mismo. La cuarta llamada ocurre en casos donde sea posible llevar a cabo esta operación, por ende, tomando en cuenta esta situación como el peor caso, el algoritmo tendría 4 llamadas recursivas.

En caso de que un string sea mas largo que el otro, se llega a un caso base que, si bien tambien ejecuta una llamada recursiva, esta solo realizaría acciones de complejidad lineal, por lo que las llamadas recursivas masivas del algoritmo **se detienen al alcanzar el tamaño del string más corto.**

Si tuvieramos que combinar n elementos de 4 maneras distinas, tendríamos 4^n combinaciones, siguiendo este ejemplo y ya que este algoritmo evalúa todas las posibles combinaciones hasta el tamaño del string más pequeño, el enfoque fuerza bruta tiene una complejidad temporal de $O(4^{\text{mín}(n,m)})$, con n y m los largos de s1 y s2, respectivamente.

En cuanto a la complejidad espacial, este algoritmo no ocupa ningún tipo de memoria adicional. Al ser in-place este tiene una complejidad espacial de $O(1)$.

2.1.5. Inclusión de Transposición y costos variables

La inclusión de la acción de transposición al problema original añade un grado mayor de complejidad al problema. Esto aumenta la cantidad de llamadas recursivas, empeorando considerablemente el

rendimiento con respecto al problema sin esta opción. Los casos que presenten un mayor impacto debido a esta complejidad extra son aquellos donde la acción de "transponer" sea posible para todo el string, forzando que en cada llamada al algoritmo se ejecuten los 4 casos recursivos.

Con respecto a los costos variables, estos no afectan directamente la complejidad, ya que cada valor se recuperaría en tiempo constante a través de las funciones auxiliares.

2.2. Programación Dinámica

2.2.1. Descripción de la solución recursiva

Este enfoque se centra en hallar la solución óptima sin realizar recusión innecesaria. Para lograrlo, utlizaremos programación dinámica con el método bottom-up: precalcularemos las soluciones óptimas de los subproblemas de manera iterativa y lo almancenaremos en una matriz caché (dp). Como veremos en los siguientes puntos, el llenado de la tabla se hará según la relación de recurrencia para al finalizar poder encontrar el óptimo del problema original en la casilla con las mayores coordenadas.

2.2.2. Relación de recurrencia

$$dp_{s1,s2}(i,j) = \text{mín} \begin{cases} 0, & \text{if i = j = 0,} \\ dp_{s1,s2}(i-1,j) + costDel(s1_i) & \text{if } i > 1, \\ dp_{s1,s2}(i,j-1) + costIns(s2_j) & \text{if } j > 1, \\ dp_{s1,s2}(i-1,j-1) + costSust(s1_i, s2j) & \text{if } i,j > 1, \\ dp_{s1,s2}(i-2,j-2) + costTrsp(s1_i, s1_{i-1}) & \text{if } i,j > 1 \text{ and } s1_i = s2_{j-1} \text{ and } s1_{i-1} = s2_j. \end{cases} \tag{1}$$

2.2.3. Identificación de subproblemas

Cada subproblema se identifica por dos elementos, el último elemento de s1 y el último elemento de s2. Este enfoque busca resolver todos los posibles subproblemas de manera creciente, es decir, empezando desde s1 y s2 sin elementos, pasando por s1 con i elementos y s2 con j elementos, hasta llegar al problema de s1 y s2 completos, siendo esta la pregunta original.

2.2.4. Estructura de datos y orden de cálculo

La creación de la tabla dp implica utilizar un arreglo bidimensional, es decir, un arreglo de arreglos y para la visualización de la tabla, vamos a considerar la esquina (0,0) como superior-izquierda.

Como pudimos ver en la relacion de recurrencia, la respuesta de cada problema es el óptimo entre 4 subproblemas menores. Cada una de estas respuestas se encuentra arriba, a la izquierda o en diagonal superior-izquierda respecto a el problema actual, por lo que cada cálculo tiene como requisito que las casillas en esas direcciones ya tengan un valor asignado. Esto hace que, luego de rellenar los casos base

(fila 0 para s2 vacío y columna 0 para s1 vacío), podamos calcular cada casilla de izquierda a derecha y de arriba hacia abajo.

2.2.5. Algoritmo utilizando programación dinámica

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

```
1  Procedure DYNAMIC(s1, s2)
       // Inicializa matriz de costos
2      Crear matriz dp de dimensiones (longitud(s1)+1) x (longitud(s2)+1)
       // Rellenado de casos base
3      dp[0][0] ← 0
4      for i ← 1 to longitud(s1) do
5          dp[i][0] ← i × COSTDELETE(s1[i-1])                    // Costo de borrar lo que sobra de s1
6      for j ← 1 to longitud(s2) do
7          dp[0][j] ← j × COSTINSERT(s2[j-1])                    // Costo de insertar lo que falta de s2
8      for i ← 1 to longitud(s1) do
9          for j ← 1 to longitud(s2) do
10             costo_del ← dp[i − 1][j] + COSTDELETE(s1[i-1])
11             costo_ins ← dp[i][j − 1] + COSTINSERT(s2[j-1])
12             costo_sub ← dp[i − 1][j − 1] + COSTSUBSTITUTE(s1[i-1], s2[j-1])
13             costo_trans ← ∞
14             if i>1 and j>1 and s1[i-1] = s2[j-2] and s1[i-2] = s2[j-1] then
15                 costo_trans ← dp[i − 2][j − 2] + COSTTRANSPOSE(s1[i-2], s1[i-1])
16             dp[i][j] ← mín(costo_del, costo_ins, costo_sub, costo_trans)
17     return dp[longitud(s1)][longitud(s2)]
18 Procedure COSTINSERT(b)
19     return costo de inserción para 'b'                        // Función de costo de inserción
20 Procedure COSTDELETE(a)
21     return costo de eliminación para 'a'                      // Función de costo de eliminación
22 Procedure COSTSUBSTITUTE(a, b)
23     return costo de sustitución entre 'a' y 'b'               // Función de costo de sustitución
24 Procedure COSTTRANSPOSE(a, b)
25     return costo de transposición entre 'a' y 'b'             // Función de costo de transposición
```

2.2.6. Ejemplo de ejecución

Vamos a simular la ejecución del algoritmo diseñado con los datos de entrada s1: “dog” y s2: “bag”.

1. Rellenamos los casos base: Asumiendo que todos los costos de inserción = 1 y borrado = 2.

0	2	4	6
1			
2			
3			

2. Rellenaremos una casilla no-base: Seguiremos los pasos para rellenar (1,1)

$i = 1, j = 1$

$costo_{del} \leftarrow dp[i-1][j] + \text{COSTDELETE}(s1[i\text{-}1]) = costo_{del} \leftarrow 2+2$

$costo_{ins} \leftarrow dp[i][j-1] + \text{COSTINSERT}(s2[j\text{-}1]) = costo_{ins} \leftarrow 1+1$

$costo_{sub} \leftarrow dp[i-1][j-1] + \text{COSTSUBSTITUTE}(s1[i\text{-}1], s2[j\text{-}1]) = costo_{sub} \leftarrow 0+2$

// Costo trans no lo contamos porque no cumple las condiciones

$dp[1][1] \leftarrow \text{mín}(4,2,2) = dp[1][1] \leftarrow 2$

0	2	4	6
1	2		
2			
3			

Y así continuariamos con el rellenado de todas las tablas de izquierda a derecha y desde arriba hacia abajo. El valor óptimo se encontrará en (3,3).

2.2.7. Complejidad temporal y espacial

La complejidad temporal de este algoritmo recae en el tiempo de rellenado de la tabla caché, lo que se realiza a través de dos ciclos *for* anidados. Teniendo en cuenta que estos ciclos iteran sobre todos los subproblemas posibles, la complejidad temporal de resolver el problema a través del algoritmo propuesto es de $O(n * m)$, con $n, m =$ *tamaño s1 + 1 y tamaño s2 + 1*, respectivamente. *Nótese que para n = m el algoritmo es* $O(n^2)$*, por lo que se podría decir que es, coloquialmente, un "algoritmo cuadrático disfrazado".*

En cuanto a la complejidad espacial, la memoria extra que se ocupa para esta solución es del tamaño de la tabla dp, por ende la complejidad espacial de este algoritmo es $O(n * m)$.

2.2.8. Inclusión de Transposición y costos variables

A diferencia del método de fuerza bruta, la acción extra de transponer no implica un gran impacto en la complejidad, ya que cada una de las operaciones posibles dentro del problema tienen un costo de $O(1)$ utilizando este enfoque. Si bien es una mayor carga a que si tuvieramos solo 3 acciones, el impacto de sumar una cuarta no es considerable dentro de esta implementación, por ende, a diferencia del anterior, este algoritmo no deberia tener grandes cambios en el tiempo de ejecución con strings que permitan constantemente el uso de transposición.

Los costos variables, al igual que en el enfoque anterior, no afectan directamente la complejidad, ya que cada valor se recuperaría en tiempo constante a través de las funciones auxiliares.

3. Implementaciones

Para llevar a cabo los experimentos y las mediciones se utilizaron programas en c++ para los algoritmos, auxiliados por scripts de python para la generación de las tablas de costo, datasets, y gráficos en base a los resultados. Los programas se estructuran de la siguiente manera:

- **main.cpp** : programa principal.
- **Aux.hpp** : conjunto de funciones auxiliares, tanto para calcular los costos como para la lectura y escritura de archivos.
- **utilities.hpp** : funciones para mostrar en la terminal el tiempo trasncurrido en cada prueba.
- **BruteForce.hpp** : contiene el algoritmo de fuerza bruta, implementado en c++.
- **dynamicSolution.hpp** : contiene el algoritmo de programación dinámica, implementado en c++.
- **generateCosts.py** : script de python para generar las tablas de costo, se guardan en la carpeta *costs*
- **generateDatasets.py** : script de python para generar los datasets, se guardan en la carpeta *datasets*
- **generateImages.py** : script de python para generar los gráficos, se guardan en la carpeta *imagenes*

Además de las carpetas detalladas previamente, deben existir dos carpetas adicionales: *output* y *results*. Asegúrese de que las rutas a estas carpetas, así como a las previas, esten correctamente escritas dentro de **main.cpp** en la función main(), donde existen variables path para este propósito. Las carpetas *output* y *results* almacenan los siguientes archivos:

- **/output/outputDetailDp.txt** : archivo de texto que muestra los resultados de las pruebas del algoritmo de programación dinámica. Tiene un formato que facilita la legibilidad.
- **/output/outputDetailBf.txt** : archivo de texto que muestra los resultados de las pruebas del algoritmo de fuerza bruta. Tiene un formato que facilita la legibilidad.
- **/output/outputDp.txt** : archivo de texto que contiene los resultados de las mediciones del algoritmo de programación dinámica de manera comprimida.
- **/output/outputBf.txt** : archivo de texto que contiene los resultados de las mediciones del algoritmo de fuerza bruta de manera comprimida. Estos dos últimos archivos son utilizados para hacer los gráficos mediante **generateImages.py**.
- **/results/resultsBf.txt**
- **/results/resultsDp.txt**

Los dos últimos archivos contienen, junto con sus inputs, los resultados de cada ejecución de los algoritmos correspondientes. Con estos dos archivos es posible comprobar que ambos algoritmos llegan a un resultado correcto. Es fundamental que si se quiere utilizar estos programas para la replicación de los experimentos se respete la estructura descrita, con el fin de evitar medidas erróneas o fallos de ejecución.

4. Experimentos

La realización de los experimentos detallados en esta sección fue llevada a cabo bajo las siguientes condiciones:

- Hardware:

  CPU: Ryzen 5 1600, 3.2GHz

  RAM: 24GB DDR4 2400MHz

  Almacenamiento: SSD 512GB

- Software:

  SO: Nobara Linux 40 (KDE Plasma) x86_64

  Compilador: g++ 14.2.1

  Python: 3.12.6

  Librerías de Python:

  Pandas, Seaborn, matplotlib, Pathlib, Math, Numpy

Previo a la realización de las pruebas se cumplieron todos los requisitos para el correcto funcionamiento de los programas explicado en la sección anterior. Para ello se poblaron los archivos de costos de operación mediante **generateCosts.py**.

El problema que buscamos resolver corresponde a la *Distancia de Damerau-Levenshtein* [cita], la cual es frecuentemente usada para resolver problemas como el *funcionamiento del autocorrector de un teclado* [cita]. Por ello, para acercarnos a un escenario real las tablas de costo se poblaron sus valores asociados según que tan comunes son en el diccionario inglés y que tan probable es un error de typeo en función de la distancia entre caracteres en un teclado QWERTY. Esta forma de ajustar los costos determina la forma en la que se reconstruye la solución óptima, más no afecta los tiempos de ejecución ni el uso de memoria, los que solo dependen del algoritmo ejecutado y las características de los strings de entrada, las cuales dependen de cada dataset. Estos últimos fueron generados mediante **generateDatasets.py**. Las características de cada uno de los datasets se detallan en la siguiente sección.

Cada prueba se llevó a cabo utilizando el programa principal. Cada uno de los casos de prueba se ejecutó 4 veces, midiendo el tiempo de ejecución, el uso de memoria y guardando el resultado. Las últimas 3 fueron tomadas en cuenta a la hora de promediar para calcular los valores finales. La primera usa para realizar cualquier tipo de carga de caché que vaya a realizar el programa.

4.1. Dataset (casos de prueba)

Los conjuntos de datos para probar el rendimiento de los algoritmos fueron diseñados para evaluar el comportamiento de estos bajo circunstancias específicas, casos de recursión poco comunes y tambien sus peores casos. Los datasets escojidos fueron 5, sus nombres y características son los siguientes:

- **dataset 1 : s1 vacío**: Dataset con s2 randomizado, mientras que s1 es siempre un string vacío.
- **dataset 2 : s2 vacío**: Dataset con s1 randomizado, mientras que s2 es siempre un string vacío.
- **dataset 3 : transposed**: Dataset con s1 y s2 del mismo tamaño, s2 randomizado y s1 se construye transponiendo de a pares los elementos de s2.
- **dataset 4 : repeated chars**: Dataset con s1 y s2 del mismo tamaño, ambos strings son iguales y estan compuestos por un único caracter repetido.
- **dataset 5 : s2 fijo**: Dataset con s2 determinado y de tamaño máximo en todos los casos de prueba, s1 es un string determinado al cual se le van agregando sus caracteres progresivamente.

Todos los datasets siguen un formato: la primera linea es un entero C que representa cuantos casos de prueba contiene el archivo, seguido de C casos de prueba los cuales se componen de dos enteros n y m separados por un espacio que representan el largo de s1 y s2, respectivamente. Seguido de esto vienen los strings s1 y s2 en lineas diferentes, para luego pasar al siguiente caso de prueba.

Todo string que no esté fijado por características del dataset va a ir aumentando de tamaño progresivamente, desde 0 (vacío) hasta 22, debido a que son 23 casos de prueba en total, contando los strings vacíos. Es importante destacar que el algoritmo de fuerza bruta es incapaz de procesar ese tamaño de entrada, por lo que su ejecución en estos experimentos fue limitada hasta strings de tamaño 15 dentro de cada dataset. (ajuste que puede ser cambiado dentro de **main.cpp**).

A continuación se detallan algunos de los casos de pruebas utilizados, junto con su solución óptima:

dataset 1; caso 5: s1: “”, s2: wsmf, Costo: 12

(3) Ins w in 0 → (2) Ins s in 1 → (3) Ins m in 2 → (3) Ins f in 3

dataset 2; caso 5: s1: cpdt, s2: “”, Costo: 15

(4) Del c → (4) Del p → (4) Del d → (3) Del t

dataset 3; caso 8: s1: kbatiug, s2: bktauig, Costo 11

(4) Transp b,k → (4) Transp t,a → (3) Transp u,i

dataset 4; caso 12: s1: zzzzzzzzzzzz, s2: zzzzzzzzzzzz, Costo: 0

(0) Transp p,p → (0) Transp p,p → (0) Transp p,p → (0) Transp p,p → (0) Transp p,p

dataset 5; caso 5: s1: hlpr, s2: amqyiqfonwxuowzpnymnfz, Costo: 51

(2) Ins a in 0 → (3) Ins m in 1 → (3) Ins q in 2 → (3) Ins y in 3 → (2) Ins i in 4 → (3) Ins q in 5 → (3) Ins f in 6 → (2) Ins o in 7 → (2) Sust n,h → (3) Ins w in 9 → (3) Ins x in 10 → (3) Sust u,l → (2) Ins o in 12 → (3) Ins w in 13 → (3) Ins z in 14 → (2) Ins n in 15 → (3) Ins y in 16 → (3) Ins m in 17 → (2) Ins n in 18 → (3) Sust f,r → (3) Ins z in 20

4.2. Resultados

Las gráficas mostradas a continuación estan basadas en los archivos de salida generados por el programa principal, los cuales fueron analizados y convertidos en gráficos mediante **generateImages.py**. Los mismos datos se encuentran disponibles LINK[aquí], donde se puede ver sus versiones adaptadas a la lectura y su contraparte de datos brutos pensados para ser procesados por el script de gráficos.

Si el lector quiere consultar los resultados de cada uno de estos casos de pruebas, los archivos con esa información se encuentran LINK[aqui], donde puede comparar las respuestas entregadas por cada uno de los algoritmos.

De ahora en adelante, nos referiremos a los algoritmos como DP (Dynamic Programming) y BF (Brute Force).

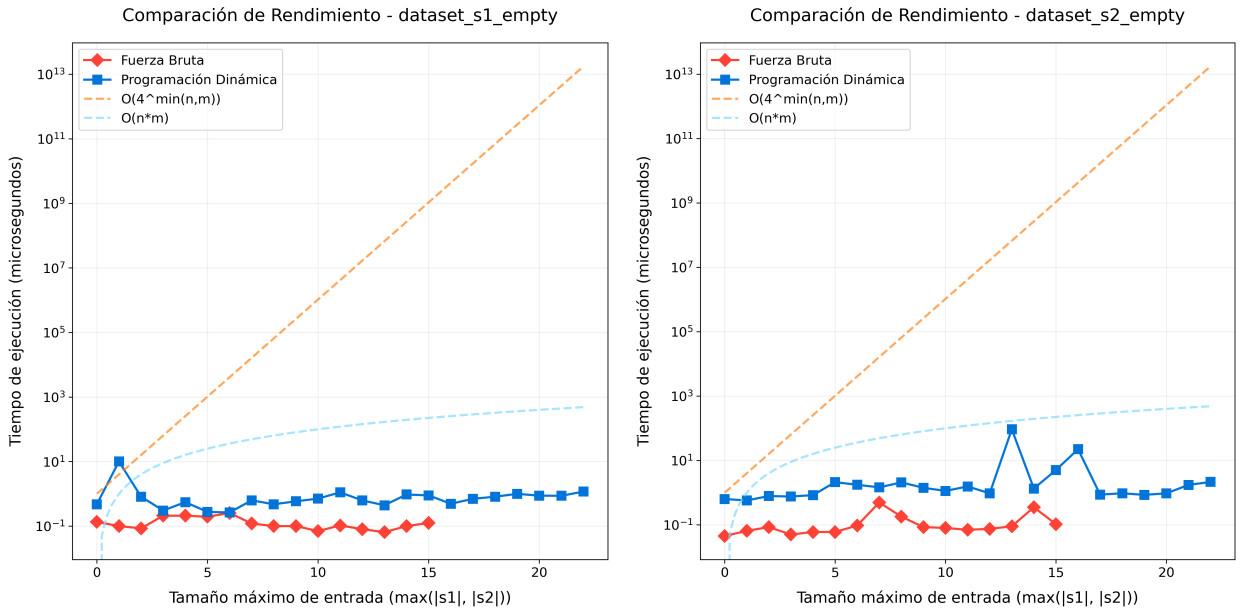

Figura 1: Comparación de rendimiento en datasets 1 y 2.

En la Figura 1 podemos ver como, si bien el algoritmo DP presenta un coportamiento esperado en base a su complejidad temporal, su contraparte de BF se aleja considerablemente de la teoría. Esto, de hecho, lo esperábamos desde la definición del algoritmo, ya que para estos datasets nuestra solución cae constantemente en un caso base, realizando exclusivamente operaciones de adición hasta recorrer el string no vacío, superando así en rendimiento incluso al enfoque DP.

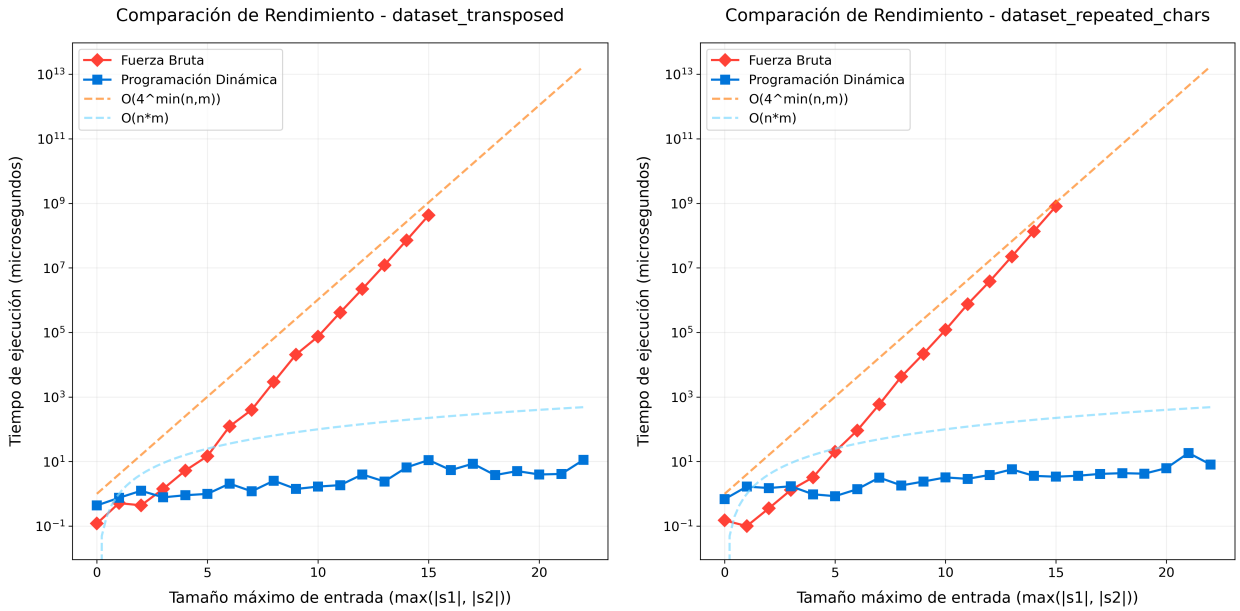

Figura 2: Comparación de rendimiento en datasets 3 y 4.

En la Figura 2, vemos que DP no cambia, pero BF sufre las consecuencias de su complejidad exponencial, esta vez siguiendo la linea de tendencia esperada según el cálculo teórico planteado previamente.

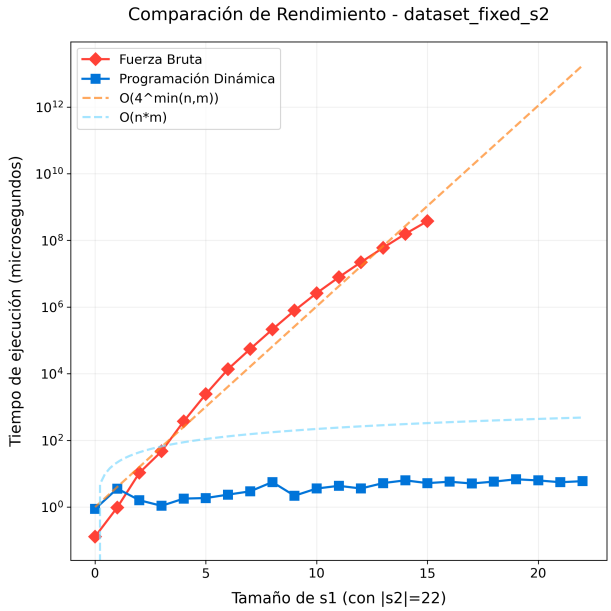

Figura 3: Rendimiento dataset 5

En la Figura 3 podemos ver nuevamente el comportamiento exponencial esperado de BF y como DP se mantiene bajo su curva de complejidad. Si nos fijamos, bajo este dataset, BF presenta un rendimiento aún peor que en los anteriores. Esto se debe a que s2, al ser constantemente de largo máximo, provoca que el tiempo de ejecución aumente con respecto a los largos variables de los datasets anteriores.

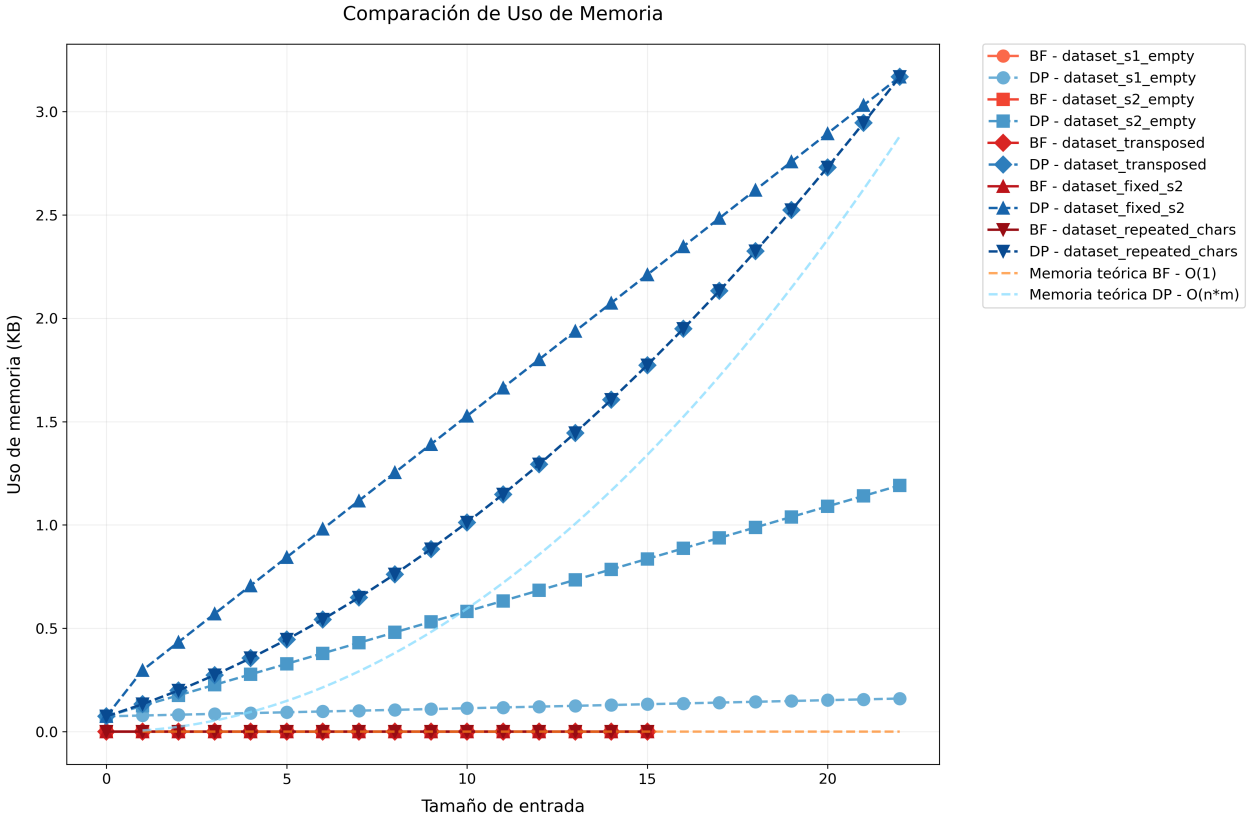

Figura 4: Comparación general de uso de memoria

En la Figura 4 vemos un gráfico general del uso de memoria de cada algoritmo dependiendo del dataset. Podemos notar que para el primer dataset, DP ocupa una mínima cantidad de memoria, ya que solo tabula un arreglo. Las mediciones superiores muestran un coportamiento acorde a el cálculo teórico, presentando un crecimiento cuadrático. La medición más extravagante es la del dataset 2, es curioso ver que no se comporta como su contraparte de s1 vacío (dataset 1). Esto se debe a que para este caso, en vez de solo tabular un arreglo, DP tabula m arreglos de tamaño 1, eso explica la considerable diferencia con el dataset 1 pero que tampoco crece de manera cuadrática como el resto de casos. Por su parte, BF se mantiene nulo en el uso de memoria, ya que es un algoritmo in-place.

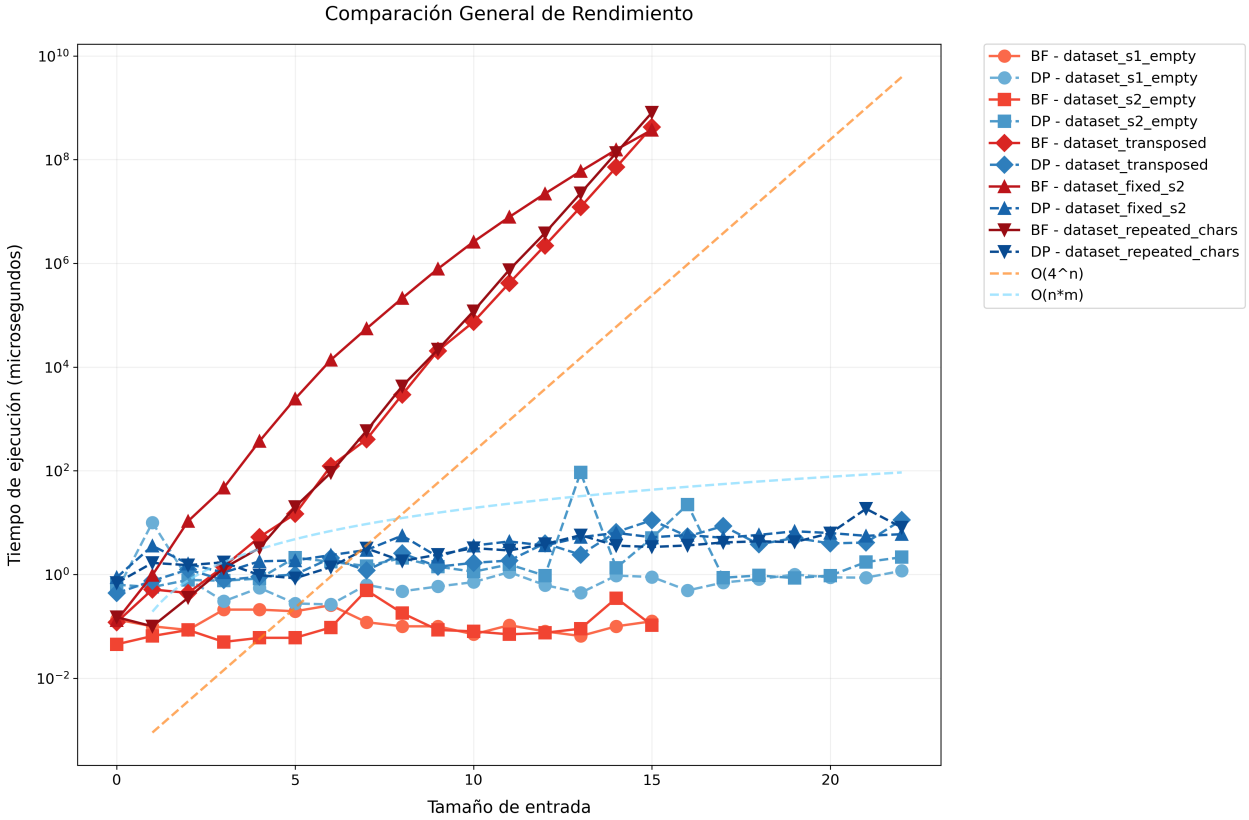

Figura 5: Comparación general de rendimiento

La Figura 5 presenta una comparación entre todas las mediciones de rendimiento vistas previamente. Podemos ver, como en los datasets 1 y 2, gracias a los casos base, BF presentan el mayor rendimiento de todos. Tambien, como en todas y cada una de las mediciones DP respeta su línea teórica, presentando un rendimiento excepcional que supera considerablemente las ejecuciones no particulares de BF. Para finalizar, se ve claramente como para los datasets 3, 4 y 5, BF tiene un crecimiento exponencial, siguiendo el ratio de la linea teórica, también podemos notar que nuestras predicciones de peor caso planteadas al diseñar el algoritmo eran correctas, ya que el peor tiempo de ejecución pertenece a BF en el **dataset 4 : repeated chars**, un dataset diseñado específicamente para permitir que en todas las llamadas del algoritmo se pueda considerar la opción de transponer, maximizando el número de llamadas recursivas, de esta forma logrando un peor rendimiento.

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] 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/.

[2] Chris Mack y Lola Muxamedova. *How to Write a Good Scientific Paper*. Nov. de 2019.