REPORTE TAREA 2-3
ALGORITMOS Y COMPLEJIDAD

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

Alejandro Vergara

17 de noviembre de 2024

20:08

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* [4], 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

Para los dos paradigmas de programación utilizados, se utilizo de estrategia la idea de calcular todos los costos posibles recursivamente, para finalmente elegir el minimo, es decir, se tendra una llamada recursiva cuando la funcion sea sustituir, insertar, eliminar o transponer, donde cada una obtendra su respectivo coste, eligiendo asi el minimo entre estos.

Las diferencia con el problema de minima distancia entre cadenas es que ahora se podra ejecutar con distintos costos, y tambien se le agrega la funcion de transponer, donde a esta solo se le considero intercambiar con el caracter que tiene adelante solo si los dos caracteres resultantes quedan en la posicion que deberian segun la cadena 2,

por ejemplo:

palabra: ab

objetivo: ba

en este casi si se podra ejecutar una transposición,

sin embargo si se tiene algo del tipo:

palabra: acb

objetivo: ba

no se podra resolver con transposiciones, ya que primero necesitamos eliminar 'c' para que luego sea factible la transposición, sin embargo esto no estara soportado por los algoritmos

Se tomaron otros supuestos, los cuales son, si los dos caracteres a comparar son iguales, no se comparara con ninguna funcion (sustituir, insertar, eliminar o transponer), ya que como son iguales se considero que el costo sera 0.

La entrada consiste en dos cadenas de caracteres, la primera sera la que buscaremos transformar a la segunda, estas dos las etiquetaremos, la primera sera simplemente **palabra**, y la segunda sera **objetivo**, estas dos serán guardadas como variables globales para las dos implementaciones, también los costos de cada funcion estaran designados de forma matricial/vectorial que especificara cada costo especifico dependiendo de cual/es caracteres esten implicados.

Por Ultimo tenemos que la palabra en realidad no se vera modificada, puesto que no es necesario para calcular el costo minimo, solo es necesario saber que si se quiere eliminar un caracter el tamaño de la palabra se reducira 1, si queremos insertar un caracter, se asumira que se insertará el correcto, por ende revisaremos el siguiente caracter de objetivo con el actual de palabra, es decir se reducira 1 a objetivo, si se quiere sustituir, hay que reducir palabra y objetivo en 1 puesto que ya cumplimos con ese caracter, y por ultimo si transponemos (se cumplen las condiciones) se reduciran los dos en 2.

2.1. Fuerza Bruta

> *"Indeed, brute force is a perfectly good technique in many cases; the real question is, can we use brute force in such a way that we avoid the worst-case behavior?"*
>
> — Knuth, 1998 [3]

Para el enfoque de fuerza bruta tenemos dos variables globales (strings), una será la palabra y la otra el objetivo. Esta funcion recibe dos parametros, **i**, tamaño de la palabra a resolver - 1, y **j**, que es el tamaño del objetivo a resolver - 1, es importante recalcar que la palabra se va subdividiendo, generando asi todos los subproblemas posibles y resolviendolos con fuerza bruta.

Algoritmo 1: Distancia Minima de Edición - Fuerza Bruta

```
1  String palabra, objetivo
2  Procedure DistanciaEdicionFB(i, j)
3      if palabra vacia (i <0) and objetivo vacio (j <0) then
4          return 0
5      if palabra vacia (i <0) then
6          return (costo insertar objetivo_j) + DistanciaEdicionFB(i, j-1)
7      if objetivo vacio (j <0) then
8          return (costo eliminar palabra_i) + DistanciaEdicionFB(i-1, j)
9      if palabra[i] == objetivo[j] then
10         return DistanciaEdicionFB(i-1, j-1)
11     eliminar ← DistanciaEdicionFB(i-1, j) + (costo eliminar palabra_i)
12     insertar ← DistanciaEdicionFB(i, j-1) + (costo insertar objetivo_j)
13     sustituir ← DistanciaEdicionFB(i-1, j-1) + (costo sustituir palabra_i por objetivo_j)
14     transponer ← ∞
15     if i y j son >0 and palabra_i == objetivo_{j-1} and palabra_{i-1} == objetivo_j then
16         transponer ← DistanciaEdicionFB(i-2, j-2) + (costo transponer palabra_i por palabra_{i-1})
17     return (minimo entre eliminar, insertar, sustituir, transponer)
```

Analizando este algoritmo tenemos que en el peor caso se generan 4 llamadas recursivas, las cuales son:

1 Eliminar: DistanciaEdicionFB(i-1, j)

2 Insertar: DistanciaEdicionFB(i, j-1)

3 Sustituir: DistanciaEdicionFB(i-1, j-1)

4 transponer: DistanciaEdicionFB(i-2, j-2)

por ende podemos generar una relacion de recurrencia que se aproxima al problema , esta es:

$$T(i, j) = 4 * T(i-1, j-1)$$

ademas tenemos que T(-1,-1) = 1, esto ya que en este caso solo retornara 0, dejando que el problema trivial de dos palabras vacias $\in$ O(1), ahora si tenemos una palabra de tamaño n, y un objetivo de tamaño

m, tendremos que T(n,m) ira aumentando en potencia de 4, es decir:

$$T(n,m) = 4 * T(n-1, m-1)$$

$$T(n,m) = 4 * 4 * T(n-2, m-2)$$

$$T(n,m) = 4 * 4 * ... * T(-1, -1)$$

Hay que tener en cuenta de que la relacion de recurrencia no se cumple siempre, por ejemplo si i<0, tenemos que ya no habran 4 llamadas recursivas sino 1, por ende como estamos considerando el peor caso tendremos una cota superior para la complejidad temporal, que será:

$$T(n,m) \in O(4^{max(n,m)})$$

Para la complejidad espacial tenemos que, ignorando la utilizacion de matrices/vectores de costos, el algoritmo utilizara un espacio extra O(1)

Ejemplo de ejecucion:
considerando los siquientes costos:

- $costo\_sub(a,b) = 3$ si $a \neq b$ y 0 si $a = b$
- $costo\_ins(b) = 1$ para cualquier caracter b
- $costo\_del(a) = 1$ para cualquier caracter a
- $costo\_trans(a,b) = 1$ para cualquier par de caracteres adyacentes a, b

y los strings:

- palabra = 'zcoal'
- objetivo = 'hola'

Con el proposito de resumir, solo se considerara el camino mas optimo para conseguir que la palabra se transforme en objetivo, comenzaremos llamando a la funcion con **i = 4** y **j = 3**.

- DistanciaEdicionFB(4, 3)
  Se ejecutaran las llamadas recursivas de eliminar, insertar y sustituir, pero al momento de entrar en el condicional para transponer nos damos cuenta de que si se respeta, logrando transponer la palabra y llamandose recursivamente con **i = 2** y **j = 1**.
- DistanciaEdicionFB(2, 1)
  Se comparan los dos caracteres, y notamos que son los mismos, por lo tanto solo se revisaran los siguientes caracteres **i = 1** y **j = 0**.

- DistanciaEdicionFB(1, 0)
  En este momento en palabra tendremos c y en objetivo h, por lo cual se llama recursivamente a cada una de las funciones, pero no a transponer porque no cumple con las condiciones, aca se generan varios caminos pero se optara por que lo mejor sera eliminar, esto es porque ahora estamos buscando que 'zc' se transforme en 'h', con lo cual se podria simplemente sustituir c por h pero esto tiene un costo de 3, sale mas barato eliminar caracteres y luego insertarlos, lo cual tendria un costo de 2, por ende la llamada recursiva sera con **i = 0** y **j = 0**.

- DistanciaEdicionFB(0, 0)
  siguiendo lo expuesto anteriormente tambien eliminaremos este caracter, por ende la llamada recursiva sera con **i = -1** y **j = 0**.

- DistanciaEdicionFB(-1, 0)
  Ahora i pasa a ser negativo, esto indica que el subproblema se limita a solo agregar los caracteres que faltan (if de la linea 5 pseudocodigo), como solo nos queda esta opcion la llamada recursiva sera con **i = -1** y **j = -1**.

- DistanciaEdicionFB(-1, -1)
  Como los dos indices son negativos retorna 0, indicando que ha terminado.

Recopilando los costos efectuados tenemos un total de 4, que corresponden a:
terminar(0) + insertar(1) + eliminar(1) + eliminar(1) + transponer(1)

2.2. Programación Dinámica

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

Erickson, 2019 [2]

Para este enfoque se utilizo una tabla que indica los costos de cada subproblema, haciendo que el programa realice menos llamadas recursivas, logrando un mejor rendimiento.
En esta implementacion a palabra y a objetivo se les agrega un caracter al inicio, el cual es '-', esto es para simplificar el acceso a la matriz de subproblemas, tambien tendremos la matriz que estara implementada con un vector de 2 dimensiones, esta estara inicializada con valores -1 en todos los puntos menos en el (0,0), en este punto valdra 0, que es el caso base, la solucion al problema trivial de palabra=" y objetivo = ".

Algoritmo 2: Distancia Minima de Edición - Programación Dinamica

```
1  String palabra, objetivo
2  Matriz tabla
3  Procedure DistanciaEdicionPD(i, j)
4      if tabla_ij != -1 then
5          return tabla_ij
6      if palabra vacia (i == 0) then
7          tabla_ij ← (costo insertar objetivo_j) + DistanciaEdicionPD(i, j-1)
8          return tabla_ij
9      if objetivo vacio (j == 0) then
10         tabla_ij ← (costo eliminar palabra_i) + DistanciaEdicionPD(i-1, j)
11         return tabla_ij
12     if palabra[i] == objetivo[j] then
13         tabla_ij ← DistanciaEdicionPD(i-1, j-1) return tabla_ij
14     eliminar ← DistanciaEdicionPD(i-1, j) + (costo eliminar palabra_i)
15     insertar ← DistanciaEdicionPD(i, j-1) + (costo insertar objetivo_j)
16     sustituir ← DistanciaEdicionPD(i-1, j-1) + (costo sustituir palabra_i por objetivo_j)
17     transponer ← ∞
18     if i y j son >0 and palabra_i == objetivo_{j-1} and palabra_{i-1} == objetivo_j then
19         transponer ← DistanciaEdicionPD(i-2, j-2) + (costo transponer palabra_i por palabra_{i-1})
20     tabla_ij ← (minimo entre eliminar, insertar, sustituir, transponer)
21     return tabla_ij
22 Procedure set()
23     ...
24     palabra ← " – " + leertxt
25     objetivo ← " – " + leertxt
26     setear tabla con valores -1
27     tabla[0][0] ← 0
28     ...
```

El algoritmo con el enfoque de programacion dinámica, especificamente utilizando el enfoque top down, es muy similar a fuerza bruta, de hecho de igual forma se calculan todos los caminos posibles, pero en este con este enfoque no se volvera a recalcular, sino que los resultados parciales (resultados del subproblema) se iran guardando, ahorrando asi llamadas recursivas y mejorando el rendimiento.
Este algoritmo divide el problema en varios subproblemas, dependiendo de la ultima funcion que se el ejecuto a la palabra, la manera mas facil de verlo es que se fija el ultimo caracter y se decide hacer una operacion, en donde si se agrega, o se sustituye un caracter, siempre se considerara que es el que beneficia al problema, por ende tendremos una fraccion del problema resuelto, faltaria el resto, ese seria el primer subproblema.
Para la complejidad temporal tenemos que lo que buscaremos realizar es llenar la matriz con los subproblemas, esta tendra un tamaño de $n * m$ donde n es el tamaño de palabra y m es el tamaño de objetivo, en conclusión su complejidad será:

$$T(n,m) \in O(n * m)$$

Por otro lado para la complejidad espacial tendremos que de espacio extra (ignorando espacio utilizado

para los costos) tendremos la matriz de subproblemas, la cual consta de n filas y m columnas, por ende el espacio estara definido por:

$$E(n,m) \in O(n * m)$$

Ejemplo de ejecucion: considerando los siquientes costos:

- $costo\_sub(a,b) = 3$ si $a \neq b$ y 0 si $a = b$
- $costo\_ins(b) = 1$ para cualquier caracter b
- $costo\_del(a) = 1$ para cualquier caracter a
- $costo\_trans(a,b) = 1$ para cualquier par de caracteres adyacentes a, b

y los strings:

- palabra = 'abc'
- objetivo = 'bad'

como el tamaño de palabra y de objetivo es 3, sin embargo le agregaremos el '-' antes, que como ya fue explicado anteriormente es para acceder de manera mas sencilla a la matriz, sabiendo esto los dos strings quedaran de largo 4, por ende comenzaremos llamando a la funcion con **i = 3** y **j = 3**.
Para este ejemplo el algoritmo se comportara igual que el de fuerza bruta e ira llenando la matriz de a poco, en este problema es visible que la menor distancia es eliminar 'c', agregar 'd', transponer 'ab', con un costo total de 3, el algoritmo generara la siguiente matriz:

$$\begin{matrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 1 & 2 \\ 2 & 1 & 1 & 2 \\ 3 & 2 & 2 & 3 \end{matrix} \tag{1}$$

de la cual podemos apreciar que el algoritmo efectivamente revisa todos los i y los j posibles, y que el resultado será 3 (DistanciaEdicionPD(2,3) + 1), como ya sabemos el resultado, podemos seguir el camino de regreso, como la primera operación fue 'eliminar' tenemos que ir a (2, 3), que guarda el valor 2, esto nos indica que DistanciaEdicionPD(2,3) = 2, luego la siguiente operacion fue 'agregar' en consecuencia visitaremos la matriz en (2,2), que es 1, entonces DistanciaEdicionPD(2,2) = 1, por ultimo la operacion que viene sera transponer, la cual tiene un costo de 1, que viene de DistanciaEdicionPD(0,0) + 1, pero como i=0 y j=0, el resultado es 0

3. Implementaciones

Ahora se especificara mas acerca de las implementaciones de estos programas, estas se encuentran en el siguiente repositorio, en este se encuentra la información de todo lo relacionado con el informe.
La implementación tiene 4 archivos de gran importancia, estos son:

- `costos.cpp:`
  En este subprograma se tendran las implementaciones de los costos de las funciones.

- `entrada.txt:`
  Aqui se ingresara en la primera linea la palabra, y en la segunda el objetivo para resolver el problema.

- `FuerzaBruta/distanciaDeEdicion.cpp`
  En este archivo se guarda la implementacion con fuerza bruta del problema, esta posee dos funciones, las cuales son distanciaEdicion(i,j), y set(), la primera resuelve el problema con fuerza bruta revisando todos los casos posibles, la segunda lee los strings del archivo entrada.txt.

- `prg_dinamica/distanciaDeEdicion.cpp`
  En este archivo se guarda la implementacion con programacion dinamica del problema, esta posee dos funciones, las cuales son distanciaEdicion(i,j), y set(), la primera resuelve el problema con programacion dinamica revisando todos los casos posibles pero sin recalcularlos, y la segunda lee los strings del archivo entrada.txt y a estas se les agrega un '-' al inicio, esto es solo para facilitar el acceso a la matriz, no perjudica a la entrada.

También se necesitan 4 archivos txt para los costos, estos tendran un tamaño donde n = 26 (alfabeto inglés, solo con letras minusculas) estos son; cost_insert.txt, cost_delete.txt, cost_replace.txt, cost_transpose.txt, estos estaran guardados en la carpeta costos, y cada uno guardara n (insert, delete) o $n*n$ (replace, transpose) numeros enteros, de igual manera en el repositorio hay un generador de costos aleatorio.

Para correr los programas, solo hay que modificar entrada.txt, ingresando la palabra y el objetivo deseados, luego desde la carpeta **tarea2-3** compilar con:
g++ -o out fuerzabruta/distanciadeedicion.cpp -Wall
si se quiere por fuerza bruta, o
g++ -o out prg_dinamica/distanciadeedicion.cpp -Wall
si se quiere por programacion dinamica, luego para ejecutar tan solo ingresar el comando:
.\out

4. Experimentos

> "*Non-reproducible single occurrences are of no significance to science.*"
>
> —Popper,2005 [5]

Para los experimentos realizados, se utilizaron costos fijos, es decir, para todos los casos se utilizaron las mismas matrices y vectores. Estos fueron generados aleatoriamente con el archivo 'costos.cpp' y están guardados en el repositorio.
Si se quiere utilizar 'costos.cpp', hay que tener en cuenta que este funciona con rangos de valores: se tiene que ingresar el valor mínimo y el valor máximo. Además, su salida se guarda en una carpeta llamada **costos**, que contiene los archivos de texto con los valores.
También, en algunos casos, se crearon palabras aleatorias con el archivo 'entrada_generator.py'. En este programa se necesita ingresar el string objetivo y generará el string palabra, el cual contiene los caracteres generados según los parámetros dados.
hardware con las siguientes especificaciones:

- Procesador Intel Core i3-12100F
- RAM 16 GB DDR4
- Almacenamiento ssd SATA
- SO windows 10 pro, pero los algoritmos fueron ejecutados en la maquina virtual de ubuntu 18.04.6
- Librerias utilizadas: iostream, string, fstream, random, vector, algorithm, climits, chrono, sstream
- Los algoritmos fueron compilados utilizando g++ 7.5.0

4.1. Dataset (casos de prueba)

Los experimentos realizados fueron muy variados, cada uno tiene un enfoque particular, esto debido a que el problema tiene muchas variables que son importantes de analizar, los enfoques utilizados fueron:

1 objetivo fijo, este fue "paralelepipedo"de tamaño 14, y la palabra varió con el proposito de que se ejecutara solo una función, es decir eliminar, insertar, sustituir o transponer, las palabras de entrada fueron "paralelepipedoa", "paralelepiped", "paralelepipewo", "paralelepipeod".

2 objetivo fijo, este fue "paralelepipedo"de tamaño 14, y la palabra se definio aleatoriamente (con el programa de python), de un tamaño $\pm$ 2 con el proposito de analizar como se comporta en casos mas aleatorios.

3 objetivo fijo, este fue .ªbcabcabca"de tamaño 10, y ahora se fueron eligiendo palabras aleatorias (con el programa de python), y se comenzo con que estas tuvieran un tamaño de 5 y luego se fue

aumentando de 5 en 5 hasta llegar a tamaño 20, esto para ver como se comporta el tiempo de ejecucion en base al largo de la entrada.

4 este enfoque es muy parecido al anterior, sin embargo se utiliza un string objetivo que es .ªbcdefghij", esto genera mas variedad en los caracteres que se utilizaran en la palabra, y se tiene el proposito de ver como aumenta el tiempo de ejecucion con strings con caracteres mas variados.

5 este enfoque busca hacer crecer la palabra y el objetivo al mismo tiempo, y para estos strings se utilizaron los caracteres 'a', 'b', 'c', esto para ver que tanto afecta que los dos strings aumenten.

6 este enfoque busca hacer hacer lo mismo que el anterior pero con mas varianza en los caracteres de los strings, objetivo va aumentando en orden alfabetico.

Por ultimo, en el repositorio hay un archivo exel que contiene todos los datos medidos, la palabra utilizada, el tiempo medido, el resultado de distancia de edicion, el espacio utilizado y los graficos hechos.

4.2. Resultados

En los resultados tenemos que, para el primer enfoque, los tiempos con programación dinámica son muy bajos comparados con fuerza bruta, y del gráfico podemos ver que la eliminación fue el que mas se tardo, esto se atribuye a que para que solo sea una eliminación, hay que tener 1 carácter de mas en palabra, lo cual hace que su longitud sea mas larga, teniendo asi mas casos de prueba, por otro lado tenemos que sustituir es el que menos tarda, y esto se debe a que como las palabras son muy similares, después de sustituir, todo sera igual, no haciendo asi, mas llamadas recursivas.

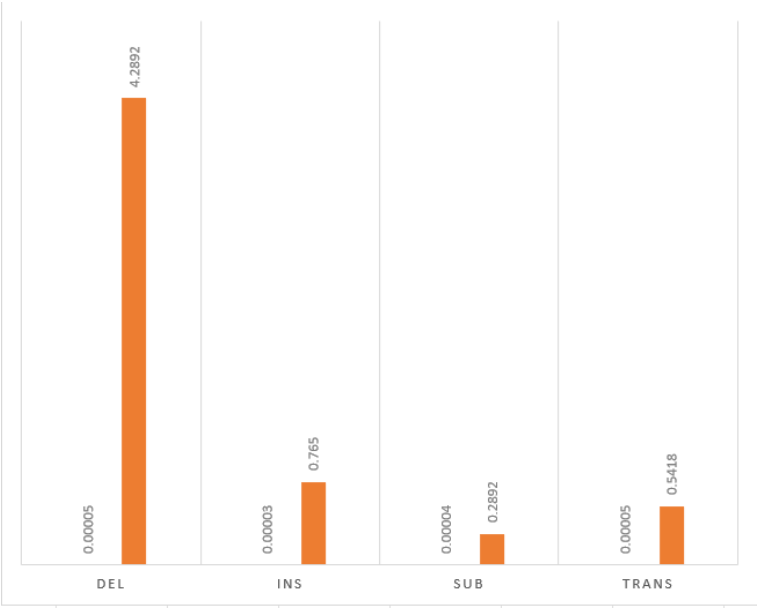

Figura 1: Gráfico para funciones especificas

Con el segundo enfoque se obtiene que al igual que en enfoque anterior, pero en este se puede presenciar de mejor manera como crece exponencialmente el enfoque de fuerza bruta, y como programa-

ción dinámica crece mínimamente, por otro lado el espacio se mantiene constante, en general fuerza bruta utilizo menos espacio que programación dinámica, que es lo que se esperaba en un inicio, sin embargo a aveces resultaba dar distintos espacios para una misma entrada, esto se debe a que el sistema operativo busca la mejor manera de manejar la memoria dinámica (Heap)

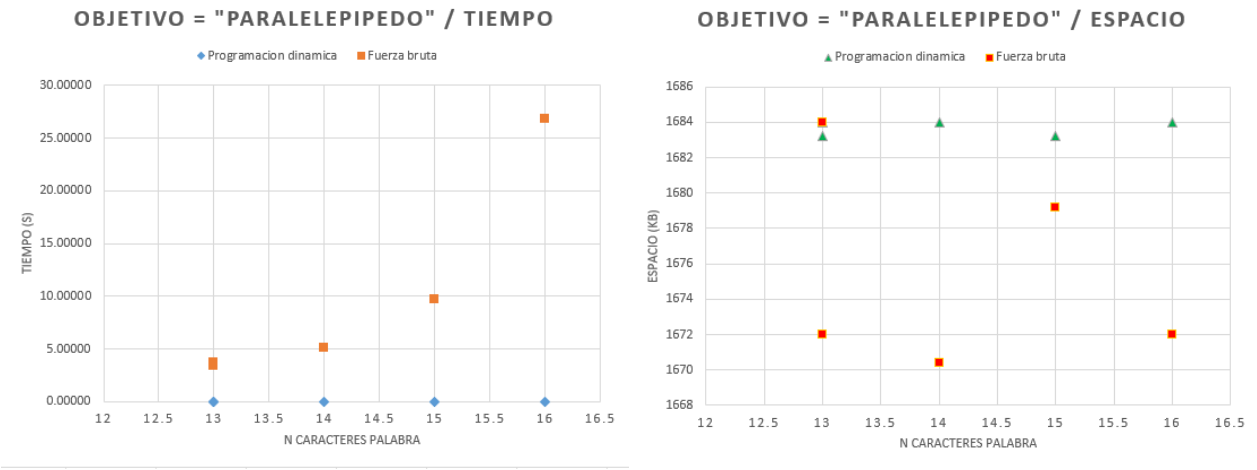

Figura 2: Gráficos con palabras aleatorias para objetivo paralelepipedo

Ahora para los siguientes gráficos se combinaran los enfoques para comparar de la forma mas rigurosa posible, con los enfoques donde la palabra crece se genero el siguiente gráfico para programación dinámica y fuerza bruta:

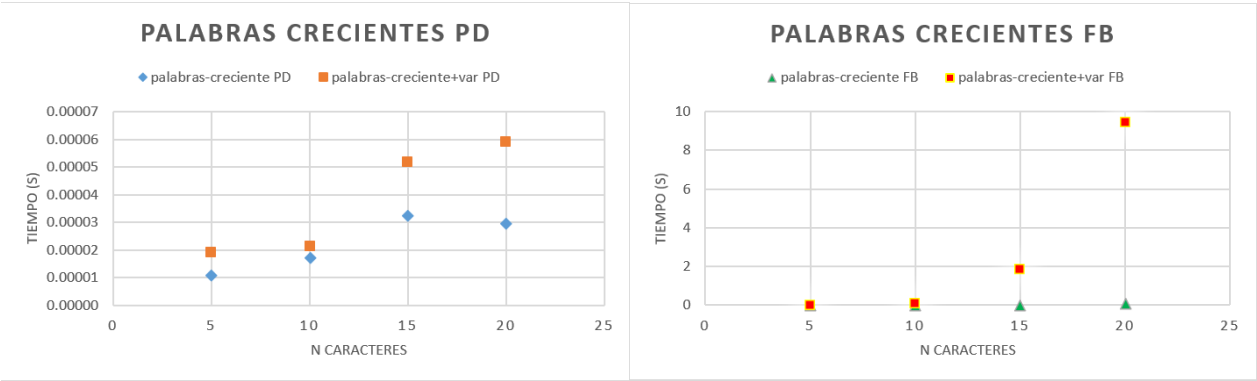

Figura 3: Gráficos para palabras de largo creciente con objetivo fijo

Estos gráficos se separaron para que sea mas facil ver como crece programacion dinamica, que antes parecia ser constante, aqui se revela que no lo es, se comporta de forma mas lineal, aunque el grafico no es muy exacto porque las palabras para ir aumentando tienen que cambiar.
también se observa que si los caracteres son poco variados, el programa funcionara mejor con cualquiera de los enfoques, de hecho para fuerza bruta, si los caracteres son variados se comporta mucho mas exponencial que si no lo son, esto se debe a que es mas probable que en algún momento se encuentre algún carácter igual, reduciendo las llamadas recursivas, y por tanto reduciendo los tiempos.

Por otro lado se ve la gran diferencia que hay entre fuerza bruta y programación dinámica, esta esta a una escala mucho menor, sus mayor valor no alcanzan a ser 0.001 segundos, mientras que con fuerza bruta, para palabras de 15 caracteres ya tarda 2 segundos.

Los siguientes gráficos muestran como se comportan los tiempos cuando palabra y objetivo tienen el mismo largo y este va variando, aqui ya se pierde mucho la exactitud porque palabra y objetivo tienen que ir cambiando para aumentar su tamaño, lo cual hace que los problemas sean distintos.

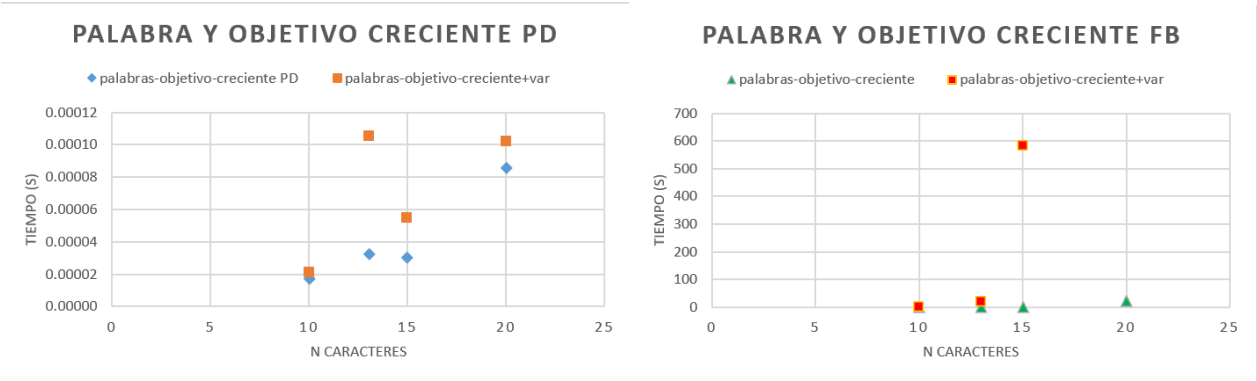

Figura 4: Gráficos para palabras y objetivos de largo creciente

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-2-3-rol.tar.gz` (`rol` con dígito verificador y sin guión).

  Dicho tarball debe contener las fuentes en LaTeX 2_ε (al menos `tarea-2-3.tex`) de la parte escrita de su entrega, además de un archivo `tarea-2-3.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] Jeff Erickson. *Algorithms*. Jun. de 2019. ISBN: 978-1-792-64483-2.

[3] Donald E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. USA: Addison Wesley Longman Publishing Co., Inc., 1998. ISBN: 0201896850.

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

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