ENUNCIADO TAREA 2
ALGORITMOS Y COMPLEJIDAD

«Encontrando Diferencias entre dos Secuencias»

Fecha límite de entrega:

Profesor/es: Raquel Pezoa, Juan Pablo Castillo, Maximiliano Ojeda
Equipo ALGORITMOS Y COMPLEJIDAD 2025-1

16 de mayo de 2025
17:54
Versión 0.0

Índice

1. Objetivos

El objetivo de esta tarea 2 es introducirnos en el Diseño de Algoritmos. Para esto, utilizaremos dos paradigmas de programación: fuerza bruta y programación dinámica. El primer paradigma es un enfoque fundamental en el diseño de algoritmos y es especialmente útil para resolver problemas donde todas las posibles soluciones deben ser evaluadas exhaustivamente y lo compararemos con el paradigma de programación dinámica.

En este contexto, ustedes deberán implementar 2 algoritmos que **encuentra las diferencias entre dos secuencias** utilizando los 2 enfoques antes mencionados.

2. Especificaciones

En esta sección se describen los pasos a seguir para completar la tarea, la cual consiste en desarrollar código en C++ «Implementaciones» y en realizar un informe «Informe». Se espera que cada uno de los pasos se realice de manera ordenada y siguiendo las instrucciones dadas.

> Abra este documento en algún lector de pdf que permita hipervínculos, ya que en este documento el texto en color azul suele indicar un hipervínculo.

(1) En caso de cualquier duda, contactarse directamente con **Pablo Álvarez**. Para esto pueden hacerlo por mensaje directo en discord (`pabloealvarez`) o correo electrónico (`pablo.alvarezs@sansano.usm.cl`). En caso de cualquier de modificaciones, todas se informarán tanto por aula como por discord.

(2) Todo lo necesario para realizar la tarea se encuentra en la **rama master** del repositorio de github:

https://github.com/pabloealvarez/INF221-2025-1-TAREA-1/tree/master/code

(3) En el repositorio del punto (2) pueden existir otras ramas, pero master siempre será donde se encuentra la información oficial.

2.1. Descripción del Problema: Encontrando diferencias entre 2 secuencias

2.1.1. Encontrando Diferencias entre dos Secuencias

Se quiere implementar una herramienta que permita mostrar todas las diferencias que existen entre dos secuencias de texto. En particular, dadas dos secuencias de caracteres $s[1..n]$ y $t[1..m]$, se deben mostrar pares de substrings en que difieren s y t (cada par debe tener un substring de s y otro de t)[1]. En particular, para que esto sea útil para el usuario, se debe mostrar **la mínima cantidad de pares de substrings** que indiquen todas las diferencias entre s y t.

Hint: para definir los pares de substrings a mostrar, se debe encontrar una serie de caracteres que aparezcan en ambas secuencias de entrada (s y t). Esos caracteres particionan a s y t en substrings correspondientes que difieren, los cuales deben mostrarse. El problema es encontrar dichos caracteres de tal manera de minimizar la cantidad de pares a mostrar.

2.1.2. Formato de Entrada

La entrada de datos será por la entrada estándar (`stdin`), y contendrá varios casos de prueba. La primera línea de la entrada contiene un entero K indicando la cantidad de casos a probar ($1 \leq K \leq 10{,}000{,}000$). Luego, le siguen los K casos, cada uno con el siguiente formato. La primera línea comienza con un entero n, y es seguido por una secuencia s de n caracteres. Ambos datos están separados por un único espacio en la entrada. Puede asumir $1 \leq n \leq 1{,}000{,}000$. La segunda línea del caso comienza con un entero m, y es seguido por una secuencia t de m caracteres. Ambos datos están separados por un único espacio en la entrada. Puede asumir $1 \leq m \leq 1{,}000{,}000$.

Un ejemplo de entrada es el siguiente:

```
3
6 ABCLGH
6 AELFHR
6 AGGJAB
7 GZJZAMB
16 Este es un texto
18 Este es otro texto
```

Hint: para probar su programa de una mejor manera, ingrese los datos de entrada con el formato indicado en un archivo de texto (por ejemplo, el archivo `input-1.dat`). Luego, ejecute su programa desde la terminal, redirigiendo la entrada estándar como a continuación:

```
./problema1 < input-1.dat
```

De esta manera, evitará tener que entrar los datos manualmente cada vez que prueba su programa.

2.1.3. Formato de Salida

La salida se hará por la salida estándar (`stdout`). La salida comienza con el valor K, indicando la cantidad de casos de prueba (es el mismo valor K de la entrada). Luego, para cada uno de los casos de prueba de la entrada, deben imprimirse todos los pares de substrings que difieren en las secuencias de entrada. El formato a usar es el siguiente. Si la cantidad de pares a mostrar para un caso es L, se debe imprimir el valor L, y a continuación se deben mostrar L líneas, cada una conteniendo el correspondiente par de strings, separados por un único espacio. Sólo se deben mostrar los substrings que difieren en ambas secuencias. Los substrings mostrados pueden ser vacíos, aunque sólo tiene sentido mostrar los pares en los que al menos unos de los substrings no es vacío.

La salida correspondiente al ejemplo mostrado anteriormente es:

[1] Note que el string vacío ϵ también es válido como substring

```
3
3
BC E
G F
 R
4
A
G Z
 Z
 M
1
un otro
```

Note que en la tercera línea del primer caso, el substring correspondiente a la primera secuencia es vacío. Algo similar ocurre en la primera línea del segundo caso, en donde el substring correspondiente a la segunda secuencia es vacía.

2.2. Implementaciones

(1) Implementar cada uno los algoritmos en C++:

- Para el paradigma de fuerza bruta, se deben implementar en C++, en el archivo correspondiente, el algoritmo para encontrar diferencias entre dos secuencias.
  - code/brute_force/algorithm/sequence_difference.cpp
- Para el paradigma de pogramación dinámica, se deben implementar en C++, en los archivos correspondientes, el algoritmo para encontrar diferencias entre dos secuencias.
  - code/dynamic_programming/algorithm/sequence_difference.cpp

(2) Implementar el programa que realizará las mediciones de tiempo y memoria en C++ (programas principales) y su respectivo `makefile`, que ejecutará los algoritmos y generará los archivos de salida en cada uno de los directorios measurements brute force y measurements dynamic programming con los resultados de cada uno de los algoritmos.

- code/brute_force/brute_force.cpp
- code/dynamic_programming/dynamic_programming.cpp

(3) Implementar el programa que generará los gráficos en PYTHON y que se encargará de leer los archivos generados por los programas principales guardados en measurements brute force y measurements dynamic programming, para luego graficar los resultados obtenidos y guardarlos en formato PNG en plots brute force y plots dynamic programming.

- code/brute_force/scripts/plot_generator.py
- code/dynamic_programming/scripts/plot_generator.py

(4) Documentar Cada uno de los pasos anteriores

- Completar el archivo `README.md` del directorio `code`
- Documentar en cada uno de sus programas, al inicio de cada archivo, fuentes de información, referencias y/o bibliografía utilizada para la implementación de cada uno de los algoritmos.

2.3. Informe

Luego de realizar las implementaciones y experimentos, se debe generar un informe en LaTeX que contenga los resultados obtenidos y una discusión sobre ellos. En el siguiente repositorio podrá encontrar el Template que **deben utilizar**, en esta entrega:

https://github.com/pabloealvarez/INF221-2025-1-TAREA-1/tree/master/report

- No se debe modificar la estructura del informe.
- Las indicaciones se encuentran en el archivo `README.md` del repositorio y en la plantilla de LaTeX.

3. Condiciones de entrega

(1) La tarea se realizará individualmente (esto es grupos de una persona), sin excepciones.

(2) La entrega debe realizarse vía `http://aula.usm.cl`, entregando la url del repositorio privado de GitHub donde se encuentra el código fuente de su tarea. Debe clonar el siguiente repositorio y crear uno nuevo privado en su cuenta de GitHub con el mismo nombre que el repositorio original (`INF221-2025-1-TAREA-1`).

`https://github.com/pabloealvarez/INF221-2025-1-TAREA-1/tree/master/code`

- El repositorio de github **DEBE SER PRIVADO**, ya que de lo contrario cualquier persona podrá acceder a su código y cometer plagio, siendo usted responsable de ello.
- **DEBERÁ DAR ACCESO A LOS AYUDANTES DE SU RESPECTIVO CAMPUS**, antes de la fecha de entrega. Para ello, se informará antes de la fecha de entrega el nombre de usuario de los ayudantes de su respectivo campus y deberá agregar a los ayudantes como colaboradores del repositorio que contiene su entrega.

(3) Si se utiliza algún código, idea, o contenido extraído de otra fuente, este **debe** ser citado en el lugar exacto donde se utilice.

(4) 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.

(5) 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).

(6) No modifique `preamble.tex`, `report.tex`, `rules.tex`, estructura de directorios, nombres de archivos, configuración del documento, etc. Sólo agregue texto, imágenes, tablas, código, etc. En el códigos fuente de su informe/reporte, no agregue paquetes, ni archivos .tex.

(7) La fecha límite de entrega es el día .

NO SE ACEPTARÁN TAREAS FUERA DE PLAZO.

(8) 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.