

ENUNCIADO TAREA 2

ALGORITMOS Y COMPLEJIDAD

«*[TODO: Añadir título]*»

Fecha límite de entrega: **21 de abril de 2025**

Equipo ALGORITMOS Y COMPLEJIDAD 2025-2

6 de octubre de 2025

01:27

Versión 0.0

Índice

1. Objetivos	2
1.1. Problema	2
1.2. Entrada y salida	3
2. Especificaciones	4
2.1. Implementaciones	4
2.2. Informe	4
3. Condiciones de entrega	5
A. Casos de prueba	6

1. Objetivos

El objetivo de esta tarea 2 es comparar diferentes técnicas para resolver un problema específico, evaluando tanto el **tiempo de ejecución** como el **uso de memoria**.

En particular, los estudiantes deberán implementar tres enfoques distintos: un **algoritmo de fuerza bruta**, dos **algoritmos greedy** (subóptimos o no necesariamente correctos) y un **algoritmo de programación dinámica**. La idea es analizar y comparar el rendimiento de cada técnica, y además, en el caso de los algoritmos greedy, evaluar **qué tan correctas o cercanas a la óptima son las soluciones obtenidas**.

1.1. Problema

En **CppCorp**, una empresa de desarrollo de software fundada en 2001, se busca optimizar la productividad de los desarrolladores. A lo largo de los años, la empresa ha crecido y cuenta con un equipo diverso de programadores con distintos lenguajes favoritos y niveles de experiencia. En el año 2025, Carlos fue nombrado jefe de la empresa y se propuso mejorar la eficiencia y productividad de todos los equipos de desarrollo.

Los empleados están organizados en fila según el orden de ingreso a la empresa, es decir, el empleado i fue el i -ésimo en entrar. Cada desarrollador i tiene un **lenguaje de programación favorito** C_i y si programa en su lenguaje favorito su productividad es A_i , mientras que si programa en otro lenguaje su productividad es B_i .

Carlos desea formar equipos que maximicen la **productividad total de la empresa**, definida como la suma de las productividades de todos los equipos. Cada desarrollador **pertenece exactamente a un equipo**, y no hay empleados fuera de los equipos. Para ello, los equipos se forman como **segmentos contiguos no vacíos** de la fila de empleados.

La productividad individual de un empleado i dado que programa con el lenguaje L se define como

$$p_i(L) = \begin{cases} A_i & \text{si } C_i = L \\ B_i & \text{si } C_i \neq L \end{cases}$$

La productividad de un equipo se obtiene sumando las productividades individuales de todos los miembros del equipo usando el lenguaje que más miembros prefieren, y en caso de empate se toma la opción que genere la menor productividad posible. Formalmente, para un segmento $[l, r]$, la productividad del equipo se define como:

$$\text{productividad_equipo}(l, r) = \min_{L \in \arg\max_{\ell} |\{i \in [l, r] : C_i = \ell\}|} \sum_{i=l}^r p_i(L)$$

Aquí, $\arg\max_x f(x)$ representa el **conjunto de valores x que maximizan $f(x)$** . En nuestra fórmula, $|\{i \in [l, r] : C_i = \ell\}|$ representa la cantidad de miembros del segmento cuyo lenguaje favorito es ℓ , y por lo tanto

$$\arg\max_{\ell} |\{i \in [l, r] : C_i = \ell\}|$$

es el **conjunto de lenguajes más frecuentes** en el segmento.

La **productividad total de la empresa** se obtiene sumando las productividades de todos los segmentos que forman los equipos:

$$\text{ProdTotal} = \sum_{\text{segmentos } [l, r]} \text{productividad_equipo}(l, r)$$

Cada segmento $[l, r]$ contribuye con su propia productividad según las reglas descritas, y el objetivo es encontrar la **partición óptima** de la fila que maximice ProdTotal, asegurando que todos los desarrolladores estén incluidos en algún equipo.

Dominio de los valores y casos de prueba

Para esta tarea, los valores de los parámetros cumplen las siguientes restricciones:

- n , la cantidad de empleados, es un entero tal que $1 \leq n \leq 10^4$.
- A_i y B_i , las productividades de cada empleado, son enteros en el rango $[-10^9, 10^9]$.
- C_i es un string formado únicamente por letras minúsculas del alfabeto inglés, sin espacios, que indica el lenguaje favorito del empleado i .

1.2. Entrada y salida

Formato de entrada

El input se presenta de la siguiente manera:

```
n
A_1 B_1 C_1
A_2 B_2 C_2
...
A_n B_n C_n
```

Formato de salida

ProdTotal

donde ProdTotal es la productividad máxima que se puede obtener al formar los equipos de acuerdo a las reglas descritas anteriormente.

Ejemplo de entrada

```
5
10 5 cpp
8 3 python
6 4 cpp
7 2 java
5 5 cpp
```

Ejemplo de salida

26

Explicación

Se pueden formar los siguientes equipos para maximizar la productividad:

■ **Equipo 1:** Empleados 1, 2, 3

- Lenguaje más frecuente: cpp (empleados 1 y 3)
- Productividad:

$$(10 + 6) + B_2 = 16 + 3 = 19$$

■ **Equipo 2:** Empleados 4 y 5

- Frecuencia máxima de lenguajes: java y cpp aparecen una vez cada uno
- Para desempatar, se elige el lenguaje que genere la mínima productividad del equipo (regla del problema)
- Productividad:
 - Lenguaje java: $(7) + B_5 = 7 + 5 = 12$
 - Lenguaje cpp: $(5) + B_4 = 5 + 2 = 7$
- Elegimos el mínimo: 7

La productividad total máxima se obtiene sumando los equipos: $19 + 7 = 26$.

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 dudas, enviar preguntas al foro de la Tarea 2 o a los ayudantes del curso. Cualquier modificación o aclaración se informará tanto por aula como por los canales oficiales de GitHub Classroom.
- (2) Todo lo necesario para realizar la tarea se encuentra en el repositorio de GitHub Classroom:

<https://classroom.github.com/a/ONcKwA6A>

Nota: La estructura de archivos y el template del proyecto se encuentran disponibles en el repositorio generado. El repositorio será automáticamente privado y contendrá toda la estructura necesaria para completar la tarea.

- (3) El repositorio generado contiene toda la información oficial y estructura necesaria para completar la tarea.

2.1. Implementaciones

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

- **Algoritmo Fuerza Bruta:** Resolver el problema de forma exhaustiva utilizando fuerza bruta.
Archivo: `code/implementation/algorithms/brute-force.cpp`.

- **Algoritmos Greedy:** Implementar dos heurísticas greedy (*subóptimas*) para el problema.

Archivos: `code/implementation/algorithms/greedy1,2.cpp`. Archivo: `code/implementation/algorithms/dynar`

- (2) Mediciones de tiempo y memoria: Implementar los programas principales en C++ y su respectivo `makefile` que ejecute los algoritmos y genere archivos de salida en los directorios:

- `code/implementation/data/measurements/`

- (3) Generación de gráficos: Implementar scripts en Python que lean los archivos de medición y generen gráficos en formato PNG en:

- `code/implementation/data/plots/`

- Scripts de ejemplo: `code/implementation/scripts/array_generator.py` y `plot_generator.py`.

- (4) Documentación: 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 bibliografía utilizada.

2.2. Informe

Generar un informe en \LaTeX que contenga los resultados obtenidos y una discusión sobre ellos. El template oficial se encuentra en el repositorio de GitHub Classroom en el directorio `report/` y debe utilizarse en esta entrega:

<https://classroom.github.com/a/ONcKwA6A>

- 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 es individual, sin excepciones.
- (2) La entrega debe realizarse vía <http://aula.usm.cl>, entregando la URL del repositorio de GitHub Classroom donde se encuentra el código fuente de su tarea. Debe aceptar la invitación de GitHub Classroom para obtener su repositorio de trabajo: <https://classroom.github.com/a/ONcKwA6A> El repositorio generado por GitHub Classroom será automáticamente privado y contendrá toda la estructura y template necesarios para completar la tarea.
- (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 de que todas sus entregas tengan sus datos completos: número de la tarea, ramo, semestre, nombre y rol. Puede incluirlos como comentarios en sus fuentes \LaTeX (en \TeX los 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 los códigos fuente de su informe/reporte, no agregue paquetes ni archivos `.tex`.
- (7) La fecha límite de entrega es el día **08 de octubre de 2025**.

A. Casos de prueba

A.0.1. Ordenamiento de un arreglo unidimensional de números enteros

Entrada:

- Leer un arreglo unidimensional A desde el archivo $\{n\}_{\{t\}}_{\{d\}}_{\{m\}}.txt$.
 - n hace referencia a la cantidad de elementos (o largo del arreglo) y pertenece al conjunto $\mathcal{N} = \{10^1, 10^3, 10^5, 10^7\}$.
 - t hace referencia al tipo de matriz, y pertenece al conjunto $\mathcal{T} = \{\text{ascendente}, \text{descendente}, \text{aleatorio}\}$.
 - d hace referencia al conjunto dominio de cada elemento del arreglo. $d = \{D1, D7\}$, donde $D7$ implica que el dominio es $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ y $D1$ que el dominio es $\{0, 1, 2, 3, \dots, 10^7\}$.
 - m hace referencia a la muestra aleatoria (o caso de prueba) y pertenece al conjunto $\mathcal{M} = \{a, b, c\}$.

Salida:

- Escribir la matriz resultante $M_1 \times M_2$ en el archivo $\{n\}_{\{t\}}_{\{d\}}_{\{m\}}_{\text{out}}.txt$.

Los programas que generan estos arreglos se encuentran en

[code/sorting/scripts/array_generator.py](#)

A.0.2. Multiplicación de matrices cuadradas de números enteros

Entrada:

- Leer dos matrices cuadradas de entrada M_1 y M_2 desde los archivos $\{n\}_{\{t\}}_{\{d\}}_{\{m\}}_1.txt$ y $\{n\}_{\{t\}}_{\{d\}}_{\{m\}}_2.txt$, respectivamente.
 - n hace referencia a la dimensión de la matriz (n filas y n columnas) y pertenece al conjunto $\mathcal{N} = \{2^4, 2^6, 2^8, 2^{10}\}$.
 - t hace referencia al tipo de matriz, y pertenece al conjunto $\mathcal{T} = \{\text{dispersa}, \text{diagonal}, \text{densa}\}$.
 - d hace referencia al dominio de cada coeficiente de la matriz $d = \{D0, D10\}$, donde $D0$ implica que el dominio es $\{0, 1\}$ y $D10$ que el dominio es $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
 - m hace referencia a la muestra aleatoria (o caso de prueba) y pertenece al conjunto $\mathcal{M} = \{a, b, c\}$.

Salida:

- Escribir la matriz resultante $M_1 \times M_2$ en el archivo $\{n\}_{\{t\}}_{\{d\}}_{\{m\}}_{\text{out}}.txt$.

Los programas que generan estas matrices se encuentran en

[code/matrix_multiplication/scripts/matrix_generator.py](#)