ENUNCIADO TAREA 2
ALGORITMOS Y COMPLEJIDAD

«CppCorp»

Fecha límite de entrega: **25 de noviembre de 2025**

Equipo ALGORITMOS Y COMPLEJIDAD 2025-2

7 de octubre de 2025
11:43
Versión 0.0

Índice

1. Objetivos

El objetivo de esta tarea 2 es comparar diferentes técnicas de resolución de un mismo problema, evaluando su **tiempo de ejecución** y **uso de memoria**.

Deberán implementar tres enfoques: un **algoritmo de fuerza bruta**, dos **heurísticas greedy subóptimas** (algoritmos que no garantizan la solución óptima) y un **algoritmo de programación dinámica**. Además, se analizará el rendimiento y la calidad de las soluciones, especialmente en los métodos greedy, para determinar qué tan cercanas están al resultado óptimo.

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 1

```
5
10 5 cpp
-8 9 python
-20 4 cpp
-7 2 java
30 -5 cpp
```

Ejemplo de salida 1

```
31
```

Explicación 1

Se pueden formar los siguientes equipos para maximizar la productividad:

- **Equipo 1:** Empleados 1
  - Lenguaje más frecuente: `cpp` (empleados 1)
  - Productividad: $A_1 = 10$
- **Equipo 2:** Empleados 2, 3 y 4
  - Frecuencia máxima de lenguajes: `cpp`, `python` y `java` aparecen una vez cada uno
  - Para desempatar, se elige el lenguaje que genere la mínima productividad del equipo
  - Productividad:
    - Lenguaje `cpp`: $B_2 + A_3 + B_4 = 9 + (-20) + 2 = -9$
    - Lenguaje `python`: $A_2 + B_3 + B_4 = -8 + 4 + 2 = -2$
    - Lenguaje `java`: $B_2 + B_3 + A_4 = 9 + 4 + (-7) = 6$
  - Elegimos el mínimo: -9
- **Equipo 3:** Empleado 5
  - Lenguaje más frecuente: `cpp` (empleado 5)
  - Productividad: $A_5 = 30$

La productividad total máxima se obtiene sumando los equipos: $10 + (-9) + 30 = 31$.

Ejemplo de entrada 2

```
6
10 5 cpp
-5 20 c
30 -5 cpp
20 -1 java
-10 10 python
10 5 java
```

Ejemplo de salida 2

```
100
```

Explicación 2

Se pueden formar los siguientes equipos para maximizar la productividad total:

- **Equipo 1:** Empleados 1, 2 y 3
  - Lenguaje más frecuente: `cpp` (empleados 1 y 3)
  - Productividad:
$$p_1(\texttt{cpp}) + p_2(\texttt{cpp}) + p_3(\texttt{cpp}) = A_1 + B_2 + A_3 = 10 + 20 + 30 = 60$$
- **Equipo 2:** Empleados 4, 5 y 6
  - Lenguaje más frecuente: `java` (empleados 4 y 6)
  - Productividad:
$$p_4(\texttt{java}) + p_5(\texttt{java}) + p_6(\texttt{java}) = A_4 + B_5 + A_6 = 20 + 10 + 10 = 40$$

La productividad total máxima es: $60 + 40 = 100$.

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/6R6pKEp7`

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/greedy{1,2}.cpp`.
- **Programación Dinámica:** Implementar la solución óptima del problema mediante programación dinámica.
  Archivo: `code/implementation/algorithms/dynamic-programming.cpp`.

(2) Mediciones de tiempo y memoria: Implementar el programa principal en C++ `code/implementation/general.cpp` y su respectivo `makefile` que ejecute los algoritmos y genere archivos de salida en los directorios:

- `code/implementation/data/measurements/`
- `code/implementation/data/outputs/`

(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/testcases_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/6R6pKEp7`

- 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.
- El informe final compilado (`reporte.pdf`) debe generarse dentro de la carpeta `report/`.

3. Condiciones de entrega

(1) La tarea es individual, sin excepciones.

(2) La entrega debe realizarse a través de http://aula.usm.cl, indicando su **usuario de GitHub** y 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/6R6pKEp7 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 **25 de noviembre de 2025**.

A. Casos de prueba

A.0.1. Generación de casos de prueba

Los estudiantes deberán implementar un generador de casos de prueba en Python llamado `testcases_generator.py`, ubicado en:

`code/implementation/scripts/testcases_generator.py`

Requisitos del generador:

- Debe generar archivos de entrada válidos para el problema de productividad de desarrolladores.
- Cada archivo de entrada debe llamarse:

  `testcases_{n}_{i}.txt`

  donde:

  - `{n}` representa la cantidad de empleados del caso de prueba.
  - `{i}` es un identificador único para diferenciar múltiples casos con la misma cantidad de empleados.
- Formato del archivo de entrada:
  - La primera línea contiene un entero n, la cantidad de empleados, con $1 \leq n \leq 10^4$.
  - Las siguientes n líneas contienen tres elementos por empleado:

    $$A_i \; B_i \; C_i$$

    donde:
    - A_i es la productividad del empleado i cuando programa con su lenguaje favorito (entero entre -10^9 y 10^9).
    - B_i es la productividad del empleado i cuando programa con un lenguaje distinto al favorito (entero entre -10^9 y 10^9).
    - C_i es el lenguaje favorito del empleado i, representado por un string de letras minúsculas (sin espacios).
- El generador debe permitir crear múltiples casos de prueba de manera automática, variando la cantidad de empleados, los valores de productividad y los lenguajes favoritos.

Los archivos generados serán utilizados por los algoritmos implementados en C++ para probar la corrección y eficiencia de las soluciones de fuerza bruta, greedy y programación dinámica.