Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1
Algoritmos Greedy en la Nación del Fuego

18 de septiembre de 2025

Nombre y Apellido	Padrón
Víctor Zacarías	107080
Carolina Aramayo	106260
Francisco Nahuel Tapia	107128
Joel Isaac Fernandez Fox	104424

1. Índice

Índice

2. Análisis del problema

Se tiene como objetivo analizar y resolver de forma óptima el siguiente problema:

Dado un conjunto de n batallas B, donde cada batalla i tiene un tiempo de finalización t_i y un peso b_i asociados, se debe determinar el orden en el que deben ser realizadas dichas batallas para minimizar la suma ponderada S.

$$B = [(t_0, b_0), ..., (t_n, b_n)]$$

$$S = \sum_{i=1}^{n} b_i.T_i$$

$$T_i = \sum_{i=0}^{j} t_j$$

Para cada batalla realizada i, T_i es la suma de todos los tiempos de finalización de las batallas anteriormente realizadas hasta la batalla i. Dicho de un modo más simple, es el tiempo consumido o transcurrido hasta la batalla i.

El orden en que se realicen las batallas tiene mucha importancia al momento de minimizar S. Si se desarrolla la sumatoria S es posible hacer un análisis más detallado al respecto.

$$S = b_0.t_0 + b_1.(t_0 + t_1) + ... + b_k(t_0 + t_1 + ...t_k) + ...$$

Nota 1: T_{n-1} (la sumatoria de tiempos hasta la última batalla) siempre va a valer lo mismo sin importar el orden de las batallas. Lo que siempre va a variar son los T_i anteriores.

Nota 2: Los pesos de las batallas tienen un gran impacto en el valor de S. Cada término de la expresión anterior corresponde a una batalla, a medida que se realizan más batallas se puede apreciar que la sumatoria de los tiempos T_i crece, por lo cual, para lo últimos términos se alcanzarían valores mucho más grandes al realizar el producto correspondiente, pudiendo hacer que S aumente de forma desproporcionada.

Nota 3: La duración de cada batalla puede agrandar en mayor o menor medida a los valores de los T_i, lo cual dependiendo del valor de los pesos puede hacer que S crezca más o menos.

Inevitablemente, los valores de T_i siempre va a crecer batalla a batalla, lo importante es escoger las batallas de forma que su crecimiento no sea tan marcado. Lo mismo ocurre con los pesos, cada b_i hace que el producto $b_i.T_i$ sea más grande, por lo que es importante atenuar este aumento.

La idea para que S no crezca de forma desproporcionada es escoger el orden de las batallas de forma que lo pesos b_i de gran valor estén junto a sumatorias de tiempos T_i de bajo valor, de esta forma el producto no es tan alto. Ocurre lo contrario con los pesos de b_i bajo valor, los mismos deben estar junto a sumatorias T_i de tiempos altas, el producto va a ser más alto, pero no tan alto como si se tuviese un peso más grande.

Respecto a la duración de cada batalla en particular, si se resolviesen las batallas más largas al inicio se estarían acumulando T_i muy grandes desde el principio, lo cual agrandaría el valor de S.

Para evitar que esto ocurra, las batallas con menor duración deberían ser realizadas desde el inicio, de forma de que cuando se lleguen a las últimas batallas (las de mayor duración) la suma acumulada por los T_i restantes sea la menor posible.

La forma de sintetizar todo lo comentado anteriormente es mediante el siguiente criterio de ordenamiento.

$$B = [(t_0, b_0), ..., (t_n, b_n)]$$

Ordenamiento: t_i/b_i de menor mayor.

- Para $t_i \ll$ (tiempo chico) y $b_i \gg$ (peso grande) el cociente es menor a 1.

- Para $t_i \gg$ (tiempo grande) y $b_i \ll$ (peso chico) el cociente grande.

Esto dos puntos garantizan que pueda ordenar el arreglo cumpliendo con lo comentado anteriormente. Se escogen primero a las batallas de tiempo chico y con peso grande, y por último a las batallas de tiempo grande y peso chico.

Lo propuesto anteriormente se trata de un algoritmo Greedy. El ordenamiento propuesto no es más que una optimización para organizar las batallas de modo que, a medida que se esté realizando cada una se agrande lo menos posible a la suma S.

En cada paso, el óptimo local consiste en escoger la batalla de menor t_i/b_i ya que es la que menor impacto tiene sobre S. La sucesión de óptimos locales es lo que lleva al algoritmo, en el caso particular de este problema (no todo algoritmo Greedy obtiene la solución óptima a un problema), a alcanzar al óptimo global.

En la siguiente sección se demostrará que el algoritmo propuesto efectivamente minimiza a S

3. Optimalidad de la solución propuesta

Es posible demostrar que el algoritmo propuesto obtiene la solución óptima mediante una demostración por inversiones.

- Demostración del óptimo por inversiones:

Sea B un arreglo de N batallas ordenado por t_i/b_i de menor a mayor.

B: $[(t_0, b_0),(t_1, b_1),...,(t_{n-1}, b_{n-1}),(t_n, b_n)]$

El arreglo B minimiza la sumatoria ponderada (S) de los tiempos de finalización de las batallas.

$S = \sum_{i=0}^{n} b_i.F_i$

$F_i = \sum_{j=0}^{i} t_j$

Antes de proceder con la demostración, es necesario analizar el caso en el cual el arreglo B tiene elementos con el mismo coeficiente $t_i/_{bi}$. A modo de ejemplo se va llamar $B*$ a un arreglo que cumpla con lo mencionado anteriormente.

$B* = [(t_0, b_0), ..., (t_k, b_k), (t_{k+1}, b_{k+1}), ..., (t_{k+x}, b_{k+x}), ..., (t_{k+y}, t_{k+y}), ..., (t_n, b_n)]$

Los elementos del arreglo con el mismo coeficiente son aquellos que van del elemento k hasta el elemento $k+y$. Debido a que el orden de estos elementos es indeterminado es importante demostrar que la sumatoria S es la misma sin importar el orden de dichos elementos.

Los coeficientes iguales tienen el siguiente comportamiento:

$t_k/b_k = t_{k+1}/b_{k+1} = ... = b_{k+x}/t_{k+x} = b_{k+y}/t_{k+y} = (a.t')/(a.b') = t'/b'$

De esa igualdad se puede enunciar que:

$\frac{t_{k+r}}{b_{k+r}} = \frac{t_{k+v}}{b_{k+v}}$

(1) $b_{k+v}.t_{k+r} = b_{k+r}.t_{k+v}$

Si se desarrolla la sumatoria S para $B*$ se llega a:

(2) $b_0.t_0 + ... + b_k.t_0 + ... + b_k.t_k + b_{k+1}.t_0 + ... + b_{k+1}.t_k + + b_{k+1}.t_{k+1} + ... + b_{k+x}.t_0 + ... + b_{k+x}.t_k + b_{k+x}.t_{k+1} + ... + b_{k+x}.t_{k+x} + ... + b_{k+y}.t_0 + ... + b_{k+y}.t_k + b_{k+y}.t_{k+1} + ... + b_{k+y}.t_{k+x} + ... + b_{k+y}.t_{k+y}$

La sumatoria S hasta el elemento $k + y$ es la misma sin importar el orden de los elementos con igual coeficiente.

Sea $B**$ una instancia del arreglo $B*$ con distinto orden en sus elementos de igual coeficiente:

$B** = [(t_0, b_0), ..., (t_k, b_k), (t_{k+1}, b_{k+1}), ..., (t_{k+y}, b_{k+y}), ..., (t_{k+x}, t_{k+x}), ..., (t_n, b_n)]$

La sumatoria para S para $B**$ es:

$b_0.t_0 + ... + b_k.t_0 + ... + b_k.t_k + b_{k+1}.t_0 + ... + b_{k+1}.t_k + b_{k+1}.t_{k+1} + ... + b_{k+y}.t_0 + ... + b_{k+y}.t_k + b_{k+y}.t_{k+1} + ... + b_{k+y}.t_{k+y} + ... + b_{k+x}.t_0 + ... + b_{k+x}.t_k + b_{k+x}.t_{k+1} + ... + b_{k+x}.t_{k+y} + ... + bk + x.t_{k+x}$(3)

Si la sumatoria S es la misma sin importar el orden de los elementos con el mismo coeficiente la diferencia entre las expresiones (2) y (3) debe ser nula.

Restando (2) con (3)...

Se anulan los términos de la forma $t_j.b_j$

(2)$... + b_k.t_0 + ... + b_{k+1}.t_0 + ... + b_{k+1}.t_k + ... + b_{k+x}.t_0 + ... + b_{k+x}.t_k + b_{k+x}.t_{k+1} + ... + b_{k+y}.t_0 + ... + b_{k+y}.t_k + b_{k+y}.t_{k+1} + ... + b_{k+y}.t_{k+x} + ...$

(3)$... + b_k.t_0 + ... + b_{k+1}.t_0 + ... + b_{k+1}.t_k + ... + b_{k+y}.t_0 + ... + b_{k+y}.t_k + b_{k+y}.t_{k+1} + ... + b_{k+x}.t_0 + ... + b_{k+x}.t_k + b_{k+x}.t_{k+1} + ... + b_{k+x}.t_{k+y} + ...$

Ahora se anulan los términos visiblemente iguales

(2)$... + b_{k+y}.t_{k+x} + ...$

(3)$... + b_{k+x}.t_{k+y} + ...$

Por la propiedad definida en (1) la diferencia entre (2) y (3) es nula. Como la sumatoria obtenida en (3) es un orden arbitrario del arreglo, queda demostrado que la sumatoria S es independiente del los elementos con igual coeficiente.

Una vez demostrado que un arreglo ordenado por el criterio definido anteriormente tiene el mismo S sin importar el orden de los elementos con igual coeficiente se realizará la demostración del óptimo por inversiones.

Dos elementos, (t_i, b_i) y (t_j, b_j) están invertidos si $t_j/b_j > t_i/b_i$ y si (t_i, b_i) va después que (t_j, b_j) en el arreglo.

Tomando de ejemplo el siguiente arreglo:

$B : [(t_0, b_0), ..., (t_{n-1}, b_{n-1}), (t_n, b_n)]$

El arreglo está ordenado de forma que $t_r/b_r < t_{r+v}/b_{r+v}$.

(4) $t_r.b_{r+v} < t_{r+v}.b_r$

Si se hace la sumatoria con el arreglo ordenado y con el arreglo con los elementos $n-1$ y n invertidos se tiene que:

(5) $b_0.t_0 + ... + b_{n-1}.t_0 + ... + b_{n-1}.t_{n-1} + b_n.t_0 + ... + b_n.t_{n-1} + b_n.t_n$

(6) $b_0.t_0 + ... + b_n.t_0 + ... + b_n.t_n + b_{n-1}.t_0 + ... + b_{n-1}.t_n + b_{n-1}.t_{n-1}$

Restando (5) con (6):

$b_n.t_{n-1} - b_{n-1}.t_n$

Pero el primer término es menor al segundo por (4), lo que quiere decir que (5) es menor que (6) y que por lo tanto, swapear un elemento un elemento invertido de forma que su coeficiente sea

menor al de otro no tiene otro resultado que hacer que la sumatoria S disminuya.

Por transitividad, invertir elementos N hasta que el arreglo este ordenado hace que S sea mínimo.

4. Algoritmo propuesto

```python
# O(n)
def obtener_coeficiente_de_batallas(batallas_planificadas: list[tuple[int,int]]) -> int:

    coeficiente = 0
    suma_parcial = 0

    for batalla in batallas_planificadas:
        suma_parcial += batalla[0]
        coeficiente += batalla[1]*suma_parcial

    return coeficiente


def planificar_batallas(batallas_a_planificar: list[tuple[int,int]]) -> tuple[list[tuple[int, int]], int]:

    # O(nlog(n)) por ordenar
    batallas_a_planificar_aux = sorted(batallas_a_planificar, key=lambda x: x[0]/x[1])

    #O(n)
    coef = obtener_coeficiente_de_batallas(batallas_a_planificar_aux)

    return batallas_a_planificar_aux, coef
```

Listing 1: Algoritmo de planificación de batallas

- **Complejidad de** `obtener_coeficiente_de_batallas`:

$$O(n)$$

- **Complejidad de** `planificar_batallas`:

$$O(n \log n) + O(n) = O(n \log n)$$

- **Conclusión**: El algoritmo tiene complejidad

$$O(n \log n)$$

Como bien se menciono anteriormente, y se puede entender en el código, el algoritmo se resolvió ordenando al arreglo de batallas de menor a mayor según el cociente t_i/b_i.

El resultado no fue otro que el de minimizar S.

$S = \sum_{i=1}^{n} b_i F_i$

4.1. Variabilidad de t_i, b_i

Las variables t_i, b_i afectan únicamente al orden final de las batallas, pero no afectan la cantidad de operaciones realizadas.

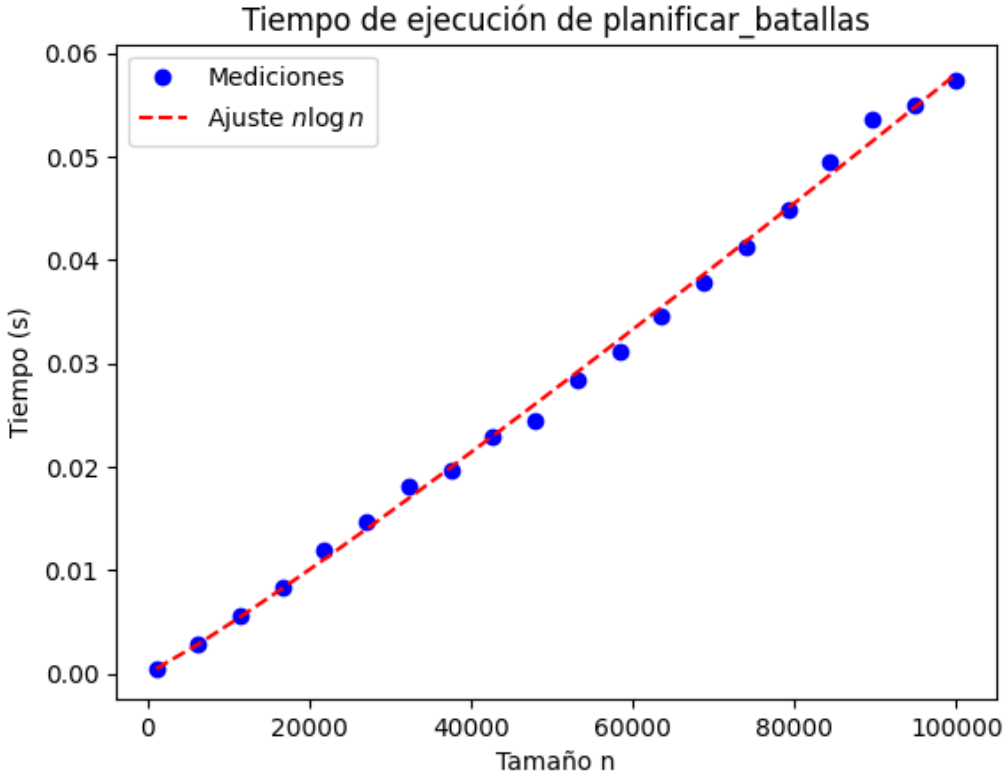
- Siempre se tienen n batallas y se realizan operaciones con cada una, ni más ni menos.
- Realizar el ordenamiento tiene complejidad $O(n \log n)$.
- La operación de calcular el cociente $\frac{t_i}{b_i}$ por cada batalla es $O(1)$.
- Calcular el coeficiente de batallas planificadas es $O(n)$ sin importar los t_i y b_i que identifican a cada batalla. Esto se debe a que el calculo del coeficiente consiste simplemente en aplicar la sumatoria S la cual está conformada por n operaciones $O(1)$.

La variabilidad de cada t_i y b_i no afecta a los tiempos de ejecución ya que la operaciones que dependen de estos valores con siempre $O(1)$. Lo que sí cambia es el resultado final del coeficiente, es decir, el orden optimo de las batallas.

5. Medición de tiempos

Para corroborar la complejidad teórica del algoritmo, se realizaron mediciones de tiempo utilizando arreglos de diferentes tamaños, generados con valores pseudoaleatorios mediante `numpy`. Para cada tamaño n, se ejecutó el algoritmo 10 veces y se promedió el tiempo de ejecución para reducir el ruido de medición.

A continuación se muestra el gráfico obtenido, junto con el ajuste por cuadrados mínimos a la función teórica $O(n \log n)$ (la complejidad teórica del algoritmo implementado):

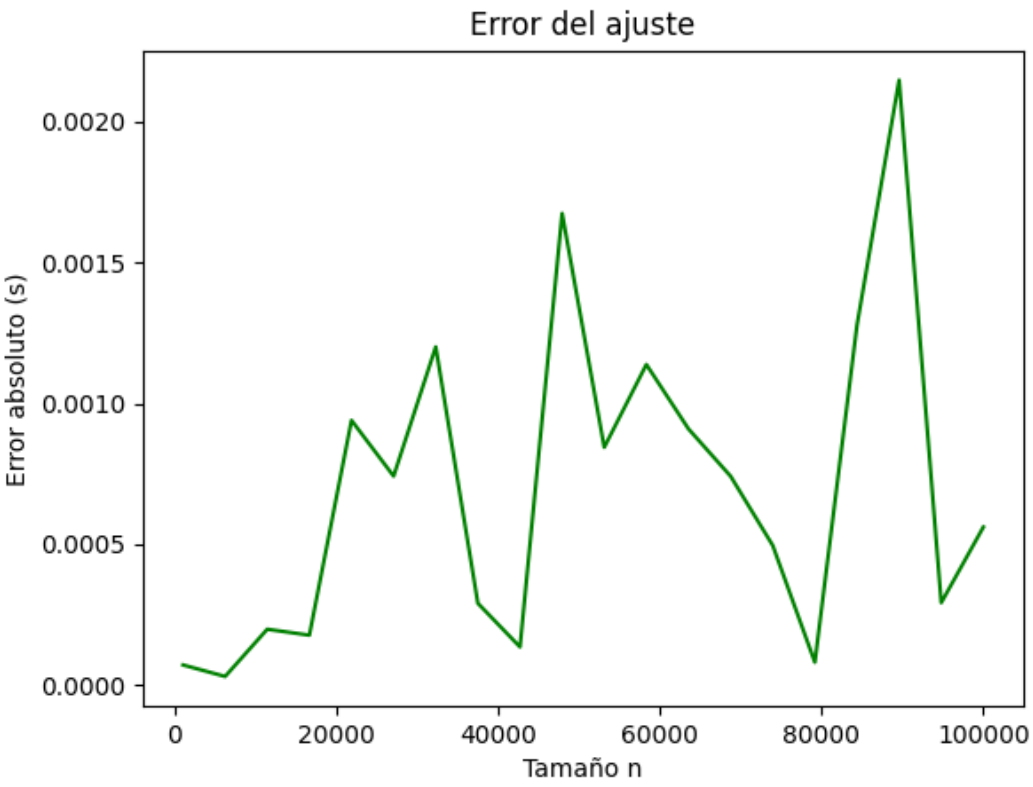


Como se observa, los datos experimentales se ajustan correctamente a la curva teórica esperada, validando la complejidad $O(n \log n)$. El ajuste se realizó utilizando la técnica de cuadrados mínimos con `curve_fit`.

- Se promediaron 10 mediciones por tamaño para cada punto.
- Los tamaños de entrada variaron de 1 000 a 100 000 elementos.

5.1. Análisis del error del ajuste

Además del gráfico de tiempos, se presenta el gráfico del error absoluto entre los valores medidos y la curva ajustada. Este gráfico permite visualizar la calidad del ajuste realizado mediante cuadrados mínimos.



Como se observa, el error absoluto es bajo en la mayoría de los casos, lo que indica un buen ajuste del modelo teórico a los datos experimentales.

6. Ejemplos de ejecucion

Para ejemplificar que el algoritmo greedy propuesto obtiene el orden óptimo, se presentan los siguientes ejemplos.

Ejemplo 1: tres batallas

Entrada:

Batalla	Tiempo t_i	Peso b_i
1	3	4
2	2	5
3	1	2

Cálculo de t_i/b_i:

Batalla	t_i/b_i
1	0.75
2	0.4
3	0.5

Orden greedy (creciente t_i/b_i): 2, 3, 1

Cálculo manual de $\sum b_i F_i$:

$$F_2 = 2, \quad F_3 = 2 + 1 = 3, \quad F_1 = 2 + 1 + 3 = 6$$

$$\sum b_i F_i = 5 \cdot 2 + 2 \cdot 3 + 4 \cdot 6 = 10 + 6 + 24 = 40$$

Ejecución del programa:

```
$ python3 main.py 1 ejemplo1.txt
--------------------------------------------------------------------------------
Salida del caso de prueba ejemplo1.txt
Coeficiente obtenido:40
Batallas planificadas:
[(2, 5), (1, 2), (3, 4)]
--------------------------------------------------------------------------------
```

—

Ejemplo 2: tres batallas iguales

Entrada:

Batalla	Tiempo t_i	Peso b_i
1	4	4
2	4	4
3	4	4

Cálculo de t_i/b_i:

Batalla	t_i/b_i
1	1
2	1
3	1

Orden greedy: cualquier orden es óptimo (todos los cocientes iguales). - Por ejemplo: 1, 2, 3

Cálculo manual de $\sum b_i F_i$:

$$F_1 = 4, \quad F_2 = 8, \quad F_3 = 12$$

$$\sum b_i F_i = 4 \cdot 4 + 4 \cdot 8 + 4 \cdot 12 = 16 + 32 + 48 = 96$$

Ejecución del programa:

```
$ python3 main.py 1 ejemplo2.txt
--------------------------------------------------------------------------------
Salida del caso de prueba ejemplo2.txt
Coeficiente obtenido:96
Batallas planificadas:
[(4, 4), (4, 4), (4, 4)]
--------------------------------------------------------------------------------
```

—

Ejemplo 3: cuatro batallas

Entrada:

Batalla	Tiempo t_i	Peso b_i
1	2	10
2	3	3
3	5	2
4	3	2

Cálculo de t_i/b_i:

Batalla	t_i/b_i
1	0.2
2	1
3	2.5
4	1.5

Orden greedy (creciente t_i/b_i): 1, 2, 4, 3

Cálculo manual de $\sum b_i F_i$:

$$F_1 = 2, \quad F_2 = 2 + 3 = 5, \quad F_4 = 5 + 3 = 8, \quad F_3 = 8 + 5 = 13$$

$$\sum b_i F_i = 10 \cdot 2 + 3 \cdot 5 + 2 \cdot 8 + 2 \cdot 13 = 20 + 15 + 16 + 26 = 77$$

Ejecución del programa:

```
$ python3 main.py 1 ejemplo3.txt
--------------------------------------------------------------------------------
Salida del caso de prueba ejemplo3.txt
Coeficiente obtenido:77
Batallas planificadas:
[(2, 10), (3, 3), (3, 2), (5, 2)]
--------------------------------------------------------------------------------
```

7. Conclusiones

El análisis y la implementación del algoritmo Greedy para la planificación de batallas muestran cómo decisiones locales óptimas pueden conducir a una solución global óptima y eficiente. Mediante el criterio de ordenamiento basado en el cociente t_i/b_i, se logró minimizar la suma ponderada de los tiempos de finalización, dando prioridad a las batallas más importantes (las de mayor peso) y rápidas.

Las mediciones experimentales de tiempo, realizadas sobre conjuntos de datos de distintos tamaños, confirmaron la complejidad $O(n \log n)$ del algoritmo. El análisis del error de ajuste evidenció un error absoluto bajo, lo que valida la eficiencia del método implementado.

Los ejemplos de ejecución presentados demuestran que el algoritmo Greedy siempre encuentra el orden óptimo, incluso en casos donde los cocientes son iguales, ya que cualquier orden resulta válido. En todos los casos, los resultados obtenidos manualmente coincidieron con los generados por el programa.

En conclusión, el algoritmo es eficiente y óptimo, y su comportamiento se ajusta perfectamente a lo esperado tanto en teoría como en la práctica, resolviendo el problema de manera robusta y confiable.