.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

# Trabajo Práctico 1
# Algoritmos Greedy en la Nación del Fuego

1 de abril de 2024


| Thiago Pacheco | Matias Bartellone | Iñaki Llorens |
|---|---|---|
| 111003 | 110484 | 107552 |

# 1. Objetivos del trabajo

La idea del trabajo práctico es consolidar los conocimientos adquiridos sobre el funcionamiento y el diseño de los algoritmos Greedy, para esto se deberá analizar el problema planteado y proponer un algoritmo greedy que encuentre la solución óptima al problema.

# 2. Análisis del problema

Debemos ayudar al Señor del Fuego a determinar el orden de las batallas que logre minimizar la suma $\sum b_i \cdot F_i$, donde las variables son:

- $b_i$: peso de la batalla, cuán importante es.
- $F_i$: es el momento en que termina la batalla $i$. Si la primera batalla es la $j$, entonces $F_j = t_j$, en cambio, si la batalla $j$ se realiza justo después de la batalla $i$, entonces $F_j = F_i + t_j$.
- $t_i$: es el tiempo que dura la batalla $i$.

Dado el problema, que requiere la implementación de un algoritmo greedy para minimizar la suma de operaciones entre dos variables, se ha determinado que una estrategia eficaz consiste en ordenar los elementos en función de dichas variables. Esto implica que el algoritmo debe seguir una política de selección que priorice los elementos según el valor de estas variables, asegurando así que se realicen las operaciones de manera óptima para minimizar el resultado final.

# 3. Propuesta de Solución

Para llegar a la solución óptima, exploramos varias alternativas que fuimos descartando progresivamente hasta llegar a la solución final. Consideramos 3 posibles enfoques, con el objetivo de minimizar la suma de las operaciones entre las variables $bi, Fi$, donde $Fi$ aumenta conforme ocurren más batallas.

- De esta observación, podemos concluir que un factor controlable es la gestión del peso en las batallas. Dado que $Fi$ aumenta con el tiempo, es deseable que $bi$, disminuya a medida que se producen más batallas, de modo que el producto sea el mínimo posible en cada momento y, por ende, la suma total sea mínima. Por lo tanto, una solución consiste en ordenar las batallas de mayor a menor peso.

  ---

  $Ej1 : (Bi, Ti)[(10, 2), (8, 2), (2, 2)]$

  En este ejemplo quedaría un coeficiente de 64 ordenado de mayor a menor por peso. En cambio ordenado de menor a mayor queda 96.

  ---

  $Contraejemplo : (Bi, Ti)[(2, 2), (8, 4), (100, 7]$

  En este ejemplo queda un coeficiente de 1352, cuando ordenamos de menor a mayor quedaría de 814, lo cual nos lleva al contraejemplo, por lo que esta solución no es óptima

- Otro aspecto a considerar es la duración de las batallas. Dado que Fi aumenta inevitablemente con el tiempo, es prudente seleccionar las batallas de menor duración primero, ya que esto ralentiza el crecimiento de la suma Fi. Al minimizar el ritmo de crecimiento, se busca reducir el impacto acumulativo en la suma total. Por lo tanto, otra solución propuesta es ordenar las batallas por su tiempo de duración, de menor a mayor.

---

$Ej2 : (Bi, Ti)[(2, 2), (2, 4), (2, 7)]$

en este ejemplo quedaría un coeficiente de 42 ordenado de menor a mayor por tiempo, En cambio ordenado de menor a mayor queda 96

---

$Contraejemplo : (Bi, Ti)[(2, 2), (8, 4), (100, 7)]$

En este ejemplo queda un coeficiente de 1352. Cuando ordenando de menor a mayor quedaríade 814, lo cual nos lleva al contraejemplo, por lo que esta solución no es óptima.

- Una tercera propuesta surge de combinar los aspectos clave de los dos anteriores. Consiste en ordenar las batallas según un coeficiente calculado como bi/ti donde bi representa el peso de la batalla y su duración. Este coeficiente refleja la relación entre el peso y la duración de cada batalla. Al ordenar de mayor a menor según este coeficiente, se da prioridad a las batallas que tienen un peso significativo y al mismo tiempo una duración corta. Esto equilibra la necesidad de minimizar el crecimiento de la suma Fi, con la importancia del peso de las batallas.

# 4. Demostración

Es cierto, al concentrarnos en solo un aspecto del problema en cada una de las primeras propuestas, podríamos perder una parte importante de la información. Al combinar ambos aspectos en esta tercera propuesta, logramos abordar de manera más completa la naturaleza del problema. Ordenar las batallas según el coeficiente $\frac{b_i}{t_i}$, que representa la relacion entre el peso y la duracion de cada batalla, nos permite considerar tanto la importancia del peso como la necesidad de minimizar el crecimiento de la suma Fi. Esto proporciona un equilibrio entre ambas consideraciones y puede conducir a una solución más efectiva y balanceada.

## 4.1. Demostración por *Inversiones*

Para probar que la solución propuesta de ordenar las batallas por el *Coeficiente (Ci)* $\frac{b_i}{t_i}$ es igual de buena que cualquier otra solución óptima, podemos utilizar el método de la contradicción. Supongamos que existe una solución óptima diferente a la propuesta y llamémosla *S'* Podemos realizar inversiones sobre *S'* sin alterar su optimalidad para demostrar su equivalencia con nuestra solución propuesta. Una inversión implica intercambiar el orden de dos batallas contiguas en *S'*. Si después de una inversión obtenemos una solución con un valor igual o menor, entonces la solución original también era óptima.

- **Inversión**
  Se define una inversión en una solucion tenemos dos batallas *s[i]* y *s[j]* en la secuencia tal que $i<j$ pero $C_i<C_j$

---

- Caso sin Inversiones
  Sucede que todos los ordenes posibles de las batallas sin inversiones tienen misma suma. En ausencia de inversiones, las únicas variaciones en la posición de las batallas ocurren entre

aquellas que tienen el mismo coeficiente, y estas batallas deben ser adyacentes. La última batalla en la secuencia será aquella con el mayor $F$ acumulado, pero su posición no afectará el resultado final.

---

$Ej1 : (B_i, T_i)[(100, 100), (50, 50), (2, 2)]$

Todos de coeficiente 1, vemos las combinaciones posibles:

- $100 \cdot 100 + 50 \cdot 150 + 2 \cdot 152 = 17804$
- $100 \cdot 100 + 2 \cdot 102 + 50 \cdot 152 = 17804$
- $50 \cdot 50 + 100 \cdot 150 + 2 \cdot 152 = 17804$
- $50 \cdot 50 + 2 \cdot 52 + 100 \cdot 152 = 17804$
- $2 \cdot 2 + 100 \cdot 102 + 50 \cdot 152 = 17804$
- $2 \cdot 2 + 50 \cdot 52 + 100 \cdot 152 = 17804$

---

$Ej2 : (B_i, T_i)[(200, 100), (4, 2), (50, 50), (3, 3)]$
Combinaciones posibles:

- $200 \cdot 100 + 4 \cdot 102 + 50 \cdot 152 + 3 \cdot 155 = 28473$
- $200 \cdot 100 + 4 \cdot 102 + 3 \cdot 105 + 50 \cdot 155 = 28473$
- $4 \cdot 2 + 200 \cdot 102 + 50 \cdot 152 + 3 \cdot 155 = 28473$
- $4 \cdot 2 + 200 \cdot 102 + 3 \cdot 105 + 50 \cdot 155 = 28473$

---

- Caso con Inversiones
  Proponemos un $S'$ óptimo que tiene inversiones, es decir suponemos que en el óptimo hay posiciones que están intercambiadas del orden en que nosotros proponemos la solución ($\frac{b_i}{t_i}$ de mayor a menor). Observamos que la inversion propone:
  $[(B_i, T_i)]$ donde
  $C_i < C_j \rightarrow \frac{b_i}{t_i} < \frac{b_j}{t_j} \rightarrow b_i \cdot t_j < b_j \cdot t_i$ (1)
  Si este óptimo permite inversiones, entonces puede haber un coeficiente $\frac{b_i}{t_i}$ que sea menor que su siguiente $\frac{b_j}{t_j}$.
  Tomando cualquier par de tuplas inversibles, según $S'$ se propone según la sumatoria:
  $t_i \cdot b_i + (t_i + t_j) \cdot b_j < t_j \cdot b_j + (t_j + t_j) \cdot b_i$
  $\cancel{t_i \cdot b_i} + t_i \cdot b_j + \cancel{t_j \cdot b_j} < \cancel{t_j \cdot b_j} + t_j \cdot b_i + \cancel{t_i \cdot b_i}$
  $t_i \cdot b_j < t_j \cdot b_i$ reordenando
  $b_i \cdot t_j > b_j \cdot t_i$ (2)

  Tomando (1) y (2), vemos que son contradictorias y se da el absurdo:
  $b_i \cdot t_j < b_j \cdot t_i$ (1) y $b_i \cdot t_j > b_j \cdot t_i$ (2)

Vemos que proponiendo un caso que admite inversiones este se cae solo, es decir se contradice. Y viendo un caso sin inversiones, probando todos los ordenes posibles de las batallas se obtiene la misma suma, que es óptima. Con esto probamos y queda demostrado que nuestro algoritmo para buscar el orden de las batallas es óptimo.

### 4.2. Greedy

Nuestro algoritmo es "greedy"porque en cada paso de la solución aplicamos una regla simple y localmente óptima. En este caso, la regla consiste en seleccionar siempre las variables con el coeficiente más alto según el criterio establecido y utilizarlas para operar.

## 5. Algoritmo

Para realizar el algoritmo modelamos la clase Batalla la cual tiene los atributos tiempo y peso:

```
class Batalla:
    def __init__(self, tiempo, peso):
        self.tiempo = tiempo
        self.peso = peso
```

Luego tenemos el algoritmo greedy que recibe una lista de batallas de tipo Batalla y la devuelve ordenada de mayor a menor por el coeficiente peso/tiempo. Tambien realiza la sumatoria y devuelve el coeficiente de batalla.

```
def optimizar_batallas(lista_batallas):
    lista_batallas.sort(key=lambda x: (x.peso/x.tiempo) ,reverse=True)

    suma,tiempo = 0,0
    for b in lista_batallas:
        tiempo += b.tiempo
        suma += tiempo * b.peso

    return lista_batallas, suma
```

### 5.1. Complejidad

Para calcular la complejidad de este algoritmo tenemos en cuenta dos cosas. Primero el metodo de ordenamiento de la lista de batallas, el cual asumimos que ordena en $\mathcal{O}(n \log n)$. Luego tenemos tambien el recorrido de la lista para realizar la sumatoria que tarda $\mathcal{O}(n)$. Esto nos deja con una complejidad final de $\mathcal{O}(n \log n)$

En este algoritmo domina la complejidad de $\mathcal{O}(n \log n)$ con n como la cantidad de elementos del arreglo. Esta complejidad no cambia independientemente de los valores específicos de los elementos en la lista. Incluso si los números dentro de las tuplas son extremadamente grandes o pequeños, la complejidad sigue siendo $\mathcal{O}(n \log n)$ debido a que la complejidad del algoritmo de ordenamiento no está afectada por el valor absoluto de los elementos. La complejidad del algoritmo de ordenamiento depende únicamente del número de elementos en la lista.

### 5.2. Optimalidad

La optimalidad no es afectada por la variabilidad de los valores $bi, ti$, ya que, como fue demostrado en la seccion 4, sean cuales sean sus valores, siempre es optimo.

# 6. Mediciones temporales

Se realizaron mediciones del algoritmo que optimiza las batallas en base a crear arreglos de diferentes largos, yendo de 100 en 100 elementos, donde los elementos en cada caso fueron generados por los valores pseudoaleatorios del lenguaje (el módulo `random`). Realizamos también mediciones sobre un algoritmo lineal para luego compararlos.

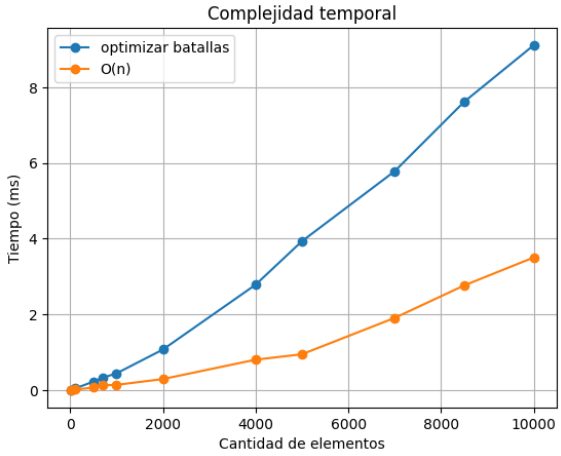


Figura 1: Comparación de Optimizar Batallas con un algoritmo O(n)


Figura 2: Comportamiento de un algoritmo $O(n \cdot \log(n))$

Ploteamos los resultados de ambas mediciones en un gráfico para mostrar como varían los tiempos de ejecucion según la cantidad de elementos. Como se puede apreciar, el algoritmo Optimizar Batallas tiene una tendencia que efectivamente es igual a la de un algoritmo $O(n \cdot \log(n))$ en función del tamaño de la entrada.

# 7. Conclusiones

En conclusión, podemos afirmar que este algoritmo es relativamente simple, dado que su lógica se basa en ordenar según un parámetro específico. Al ser un algoritmo greedy, su enfoque intuitivo y su velocidad de ejecución son notables. Sin embargo, demostrar su óptimo global fue altamente desafiante.