.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos

(75.29) Curso Buchwald - Genender

Trabajo Práctico 1

Algoritmos Greedy en la Nación del Fuego

8 de abril de 2024

Integrantes	Padrón
Lanfranco, Tomás	110883
Medone Sabatini, Juan Ignacio	103878
Rebollo, Matías Gabriel	109671

Índice

Objetivo

El objetivo del presente trabajo es idear un algoritmo para el Señor del Fuego que logre determinar el orden óptimo en el que debe llevar a cabo un conjunto de batallas. Dicho algoritmo debe implementarse de forma Greedy. Además, se brindará un análisis completo del problema y del algoritmo en cuestión.

Análisis del problema

Se nos pide ordenar un total de n batallas, donde cada batalla i consta de dos atributos:

- b_i : la importancia de la batalla, es un número real no negativo.
- t_i : el tiempo necesario para ganar la batalla, un real positivo.

Además, se define la felicidad producida por la victoria j como $F_j = F_i + t_j$, donde F_i corresponde a la batalla anterior ($F_j = t_j$ para la primera batalla). Se define como el orden óptimo de batallas a aquel orden tal que minimiza la siguiente sumatoria:

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

Podemos desarrollar esta sumatoria para observar cómo afectan las magnitudes de b_i y t_i:

$$\sum_{i=1}^{n} b_i \cdot F_i = b_1 \cdot F_1 + b_2 \cdot F_2 + b_3 \cdot F_3 + ... + b_n \cdot F_n$$

$$= b_1 \cdot t_1 + b_2 \cdot (t_1 + t_2) + b_3 \cdot (t_1 + t_2 + t_3) + \cdots + b_n \cdot \sum_{i=1}^{n} t_i$$

En este punto se puede ver que el término que acompaña a cada b_i va *creciendo*, siendo b_n (i.e. la importancia de la última batalla), el término más afectado. Ahora, sigamos desarrollando la suma, en esta ocasión distribuyendo b_i:

$$\sum_{i=1}^{n} b_i \cdot F_i = b_1 \cdot t_1 + b_2 \cdot (t_1 + t_2) + b_3 \cdot (t_1 + t_2 + t_3) + \cdots + b_n \cdot \sum_{i=1}^{n} t_i$$

$$= t_1 \sum_{i=1}^{n} b_i + t_2 \sum_{i=2}^{n} b_i + t_3 \sum_{i=3}^{n} b_i + \cdots + t_n \cdot b_n$$

En este caso notamos que los términos que acompañan a cada t_i van *disminuyendo*, por ende el t_i más afectado será el t_1 (i.e. el tiempo de la primera batalla).

Luego, llegamos a dos conclusiones:

- Si obviamos la importancia de las batallas, éstas se deben ordenar en forma ascendiente según el tiempo de duración (*las más cortas primero*).
- Si obviamos la duración de las batallas, debemos ordenarlas de manera descendiente según la importancia (*las más importantes primero*).

Si bien esto nos da un indicio de hacia donde debemos encarar el problema, no es suficiente, pues no podemos simplemente ignorar una parte entera del mismo. Entonces, debemos buscar una forma que cumpla esta relación lo mejor posible. Se propone el siguiente algoritmo:

1. Ordenamos las batallas de mayor a menor según la relación b_i/t_i. Este será el orden óptimo en el que se deberán llevar a cabo las batallas.
2. Calculamos el *coeficiente de impacto*, para eso iteramos sobre las batallas y aplicamos una *regla sencilla*: por cada batalla calculamos el término $b_i \cdot F_i$, y los vamos acumulando hasta terminar la iteración. Esta será la parte greedy del algoritmo.

Demostración mediante inversiones

Sea $a_i = b_i/t_i$. Considerando la solución A obtenida mediante nuestro algoritmo, diremos que una solución A' tiene una *inversión* si hay un par de batallas i y j tal que i se realiza antes que j, pero $a_i < a_j$. Por su funcionamiento, el A producido por nuestro algoritmo no puede tener inversiones. Por lo tanto, para demostrar que nuestro algoritmo es óptimo, debemos demostrar lo siguiente:

1. Dos soluciones distintas sin inversiones tienen el mismo coeficiente de impacto.
2. Existe una solucion óptima sin inversiones.

Dos soluciones distintas sin inversiones tienen el mismo coeficiente de impacto

Si dos soluciones ofrecen un orden de batallas distinto y no tienen inversiones, entonces solo puede diferir el orden en el que se realizan batallas de igual a_i. Sean S y S' dos soluciones que difieren por el orden de un elemento (el elemento i precede a j en S, pero invierten su orden en S') y que no tienen inversiones. A continuación se calcula la diferencia entre los coeficientes de impacto de ambas soluciones:

Aclaración: llamaremos k a la batalla realizada justo antes de i y j.

$$S = \sum_{x=1}^{n} b_x \cdot F_x = \cdots + b_i \cdot F_i + b_j \cdot F_j + \cdots = \cdots + b_i \left(F_k + t_i\right) + b_j \left(F_k + t_i + t_j\right) + \cdots =$$

$$\cdots + b_i \cdot F_k + b_i \cdot t_i + b_j \cdot F_k + b_j \cdot t_i + b_j \cdot t_j + \cdots$$

$$S' = \sum_{x=1}^{n} b_x \cdot F_x = \cdots + b_j \cdot F_j + b_i \cdot F_i + \cdots = \cdots + b_j \left(F_k + t_j\right) + b_i \left(F_k + t_i + t_j\right) + \cdots =$$

$$\cdots + b_j \cdot F_k + b_j \cdot t_j + b_i \cdot F_k + b_i \cdot t_i + b_i \cdot t_j + \cdots$$

$$S - S' = b_j t_i - b_i t_j$$

Como S y S' no tienen inversiones solo difieren elementos de igual a_i, entonces definimos $a = b_i/t_i = b_j/t_j$. Además, reemplazando $b_i = t_i a$ y $b_j = t_j a$ y reescribiendo la diferencia queda:

$$S - S' = t_j a t_i - t_i a t_j = 0 \implies S = S'$$

Por lo tanto, sus coeficientes de impacto son iguales.

Existe una solución óptima sin inversiones

Considerando la solución O, diremos que tiene al menos una inversión, ergo existe un par de batallas consecutivas i y j tal que i precede a j pero $a_i < a_j$. Si invertimos el orden de i y j, obtenemos una nueva solución con una inversión menos, la cual llamaremos O'. Luego, debemos demostrar que esta nueva solución tiene un coeficiente de impacto no mayor al de O.

Desarrollando la diferencia entre los coeficientes de impacto de O y O' de la misma manera que el ítem anterior, se llega al siguiente resultado.

$$O - O' = b_j t_i - b_i t_j$$

Reemplazando $b_i = t_i a_i$ y $b_j = t_j a_j$ la ecuación queda:

$$O - O' = t_j a_j t_i - t_i a_i t_j$$

$$a_j > a_i \implies t_j a_j > a_i t_j \implies t_j a_j t_i > t_i a_i t_j \implies O - O' > 0 \implies O > O'$$

Por lo tanto, la inversión no aumentó el coeficiente de impacto.

Finalmente, queda demostrado que la solución A producida por nuestro algoritmo produce un coeficiente de impacto óptimo.

Algoritmo y complejidad

A continuación expondremos el código de nuestro algoritmo junto con el respectivo análisis de complejidad.

Implementación

```python
# Código en main.py

def get_orden_optimo(batallas):
    return sorted(batallas, reverse=True,
        key=lambda batalla: batalla[IMPORTANCIA]/batalla[TIEMPO])

def calcular_coeficiente_de_impacto(batallas):
    felicidad = 0
    suma = 0
    for batalla in batallas:
        felicidad += batalla[TIEMPO]
        suma += batalla[IMPORTANCIA]*felicidad
    return suma
```

Análisis complejidad

Inicialmente se lleva a cabo un preprocesamiento que consta de cargar el dataset de batallas (archivo *.txt*) en una lista de tuplas, lo cual se realiza en tiempo lineal $\mathcal{O}(n)$.

Luego, el algoritmo en sí consta de dos partes:

1. Se ordenan las batallas con el criterio planteado. Considerando el uso del *sorted* de Python, esto se hace en $\mathcal{O}(n \log n)$.
2. Se iteran las batallas y se realiza la suma ponderada descrita en el problema, tardando $\mathcal{O}(n)$.

Por lo tanto, la complejidad total resulta en $\mathcal{O}(n) + \mathcal{O}(n \log n) + \mathcal{O}(n) = \mathcal{O}(n \log n)$.

Análisis variabilidad de b_i y t_i

En esta sección analizaremos cómo afecta la variabilidad de los valores de b_i y t_i al funcionamiento de nuestro algoritmo. Para ello se propuso analizar los casos en donde las batallas tienen la misma duración, en donde tienen igual importancia, y cuando la relación b_i/t_i es la misma para todas las batallas.

Batallas de igual importancia

$b_i = b \forall i \in batallas$

$$\sum_{i=1}^{n} b \cdot F_i = b \sum_{i=1}^{n} F_i = b \left(t_1 + (t_1 + t_2) + \cdots + \sum_{i=1}^{n} t_i \right) =$$

$$b \left(n \cdot t_1 + (n-1) \cdot t_2 + \cdots + t_n \right)$$

Se puede observar que el tiempo que más veces aparece en la suma es el de la primera batalla, por lo tanto el orden óptimo es aquel en donde las batallas más cortas se realizan antes.

Se puede reescribir la relación b_i/t_i de la siguiente manera: $\frac{b_i}{t_i} = \frac{b}{t_i}$

Sean j y k dos batallas tal que $t_j < t_k$ ("la batalla j es más corta que la batalla k"), entonces se observa lo siguiente:

$$t_j < t_k \implies \frac{1}{t_j} > \frac{1}{t_k} \implies \frac{b}{t_j} > \frac{b}{t_k}$$

Luego, al ordenar según nuestro criterio, la batalla j irá antes que la batalla k y en general, tal como se había observado, las batallas más cortas se pelearán antes y entonces nuestro algoritmo conserva su optimalidad.

Este caso no afecta a la complejidad de nuestro algoritmo, pues el ordenamiento debe realizarse igualmente para obtener el orden óptimo.

Batallas de igual duración

$$t_i = t \forall i \in batallas, F_j = \sum_{i=1}^{j} t_i \implies F_j = \sum_{i=1}^{j} t = j \cdot t$$

$$\sum_{i=1}^{n} b_i \cdot F_i = b_1 \cdot F_1 + b_2 \cdot F_2 + \cdots + b_n \cdot F_n = b_1 \cdot t + b_2 \cdot (2t) + \cdots + b_n \cdot (nt) =$$

$$(b_1 + 2b_2 + \cdots + nb_n) \cdot t$$

Se puede ver que el b_i con más apariciones en la suma es el b_n, por lo que buscamos que este sea el más chico. Por lo tanto, la solución óptima implica realizar las batallas con mayor b_i (las más importantes) primero.

Es posible reescribir la relación b_i/t_i de la siguiente manera: $\frac{b_i}{t_i} = \frac{b_i}{t}$

Sean j y k dos batallas tal que $b_j > b_k$ ("la batalla j es más importante que la batalla k"), entonces se observa lo siguiente:

$$b_j > b_k \implies \frac{b_j}{t} > \frac{b_k}{t}$$

Por ende, según nuestro algoritmo, se peleará la batalla j antes que la k, y en general, las batallas más importantes se lucharán primero, lo cual concuerda con lo planteado anteriormente y por consiguiente nuestra solución sigue siendo óptima.

Al igual que el caso anterior, el ordenamiento se sigue llevando a cabo de la misma manera, por lo que la complejidad de nuestro algoritmo no varía.

Misma relación b_i/t_i en todas las batallas

Se trata del caso en el que todas las soluciones posibles *no tienen inversiones* y tal como se demostró en la sección correspondiente, el coeficiente de impacto será el mismo para cualquier orden, ergo nuestro algoritmo encontrará el óptimo.

Además, para este caso particular no es necesario ordenar las batallas, por lo que estamos ante el mejor caso de nuestro algoritmo, y debido a la manera en la que se implementa el algoritmo de ordenamiento de Python, la complejidad del ordenamiento se ve reducida a $\mathcal{O}(n)$. Por lo tanto, la nueva complejidad total queda $\mathcal{O}(n) + \mathcal{O}(n) + \mathcal{O}(n) = \mathcal{O}(n)$ en función de los datos de entrada.

Casos de prueba

Se realizaron algunos ejemplos de ejecución con el propósito de validar la eficacia y optimalidad de nuestra implementación del algoritmo. Además de los casos proporcionados por la cátedra, se agregaron ejemplos adicionales para corroborar la cobertura del mismo. Estos se encuentran dentro de la carpeta *ejemplos* del repositorio.

Se prestó especial atención a la posible detección de fallos en el código con casos no contemplados inicialmente. Entre los ejemplos de ejecución se encuentran casos con: números decimales, números muy grandes y batallas con misma duración. Se observó que nuestro algoritmo respondió satisfactoriamente a todos ellos.

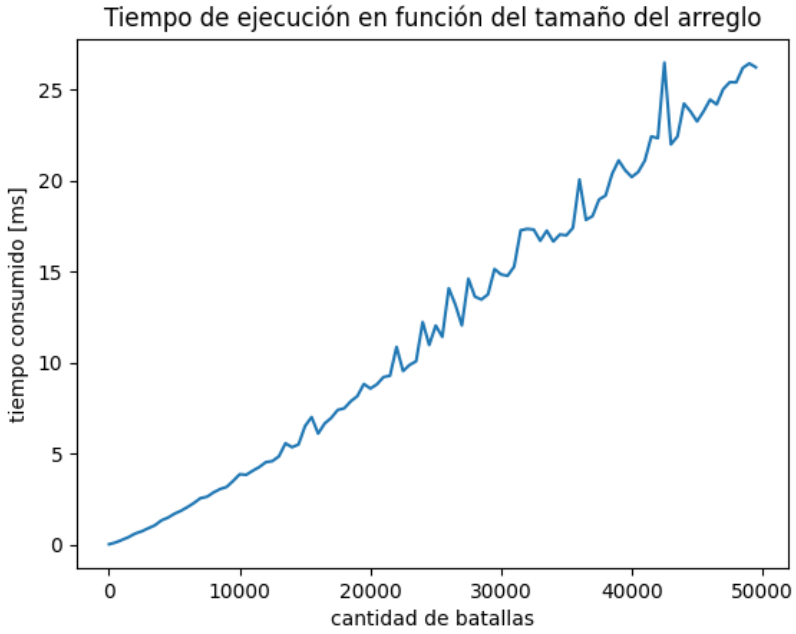
Además de la validación de los casos, se agregó una comparación de los resultados del coeficiente de impacto con el ordenamiento de nuestro algoritmo frente a otros ordenamientos menos eficientes para este problema.

	Nuestro Algoritmo	Importancia mayor a menor	Tiempo menor a mayor
Importancia decimal1	128.89000000000001	128.89000000000001	128.89000000000001
Importancia decimal2	8390.910000000002	10368.26	12164.99
Mismo tiempo	165000	165000	252000
Numeros muy grandes	1.6435902372095263e+20	1.6454821957261767e+20	1.7362205005103884e+20

Mediciones

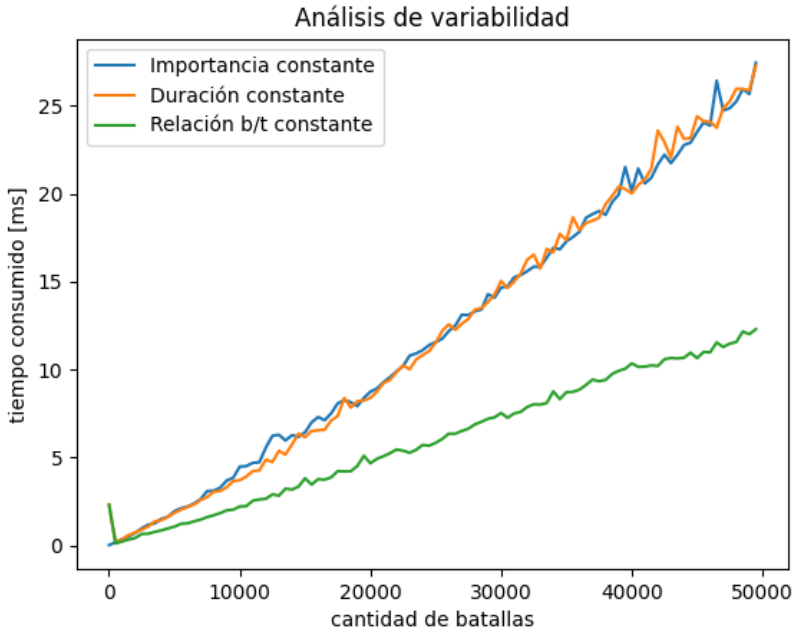
Se realizaron una serie de mediciones para visualizar la complejidad de nuestro algoritmo.

Para la primer medición se fueron generando muestras aleatorias de tamaño n, con n yendo de 10 a 50000 elementos, añadiendo 500 elementos en cada iteración y tomando el tiempo de cada muestra.



Si bien es no es tan sencillo de notar, se puede observar que el gráfico no tiene una tendencia lineal, en especial al probar con muestras más grandes.

Para el siguiente gráfico se repitió la experiencia anterior para los casos mencionados en el análisis de variabilidad de b_i y t_i.



Como se puede ver en este gráfico, la complejidad del algoritmo no tiene ningún cambio notable al tratar casos donde se mantiene constante una variable del problema. Sin embargo, se puede notar a simple vista la mejora en los tiempos del algoritmo al tratarse de casos en donde no hay inversiones, apreciando una tendencia lineal.

Conclusiones

Tras haber realizado todos los análisis, ejemplos y mediciones correspondientes, podemos concluir que:

- El algoritmo propuesto obtiene siempre la solución óptima al problema en cuestión para todos los casos posibles. La variabilidad de los valores de t_i y b_i no afecta la optimalidad del mismo.
- La complejidad del algoritmo en general es $\mathcal{O}(n \log n)$.
- Existe un caso particular (misma relación b_i/t_i en todas las batallas) en el cual se reduce la complejidad algorítmica a $\mathcal{O}(n)$.