.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1
Algoritmos Greedy

7 de abril de 2024

Tomás Hevia	Manuel Campoliete	Andrés Colina
110934	110479	110680

1. Análisis del problema

Se tiene una serie de batallas, y cada batalla tiene asignado un tiempo de duración (t_i) y una ponderación (b_i). Cada batalla tiene también un tiempo de finalización ($F_i = F_{i-1} + t_i = t_1 + t_2 + ... + t_{i-1} + t_i$ siendo $F_1 = t_1$). Por supuesto que los tiempos de finalización de las batallas dependen del orden en el que éstas se realicen.

Como queremos maximizar la felicidad, obteniéndose ésta a partir de comunicar una victoria obtenida lo antes posible para generar un mayor impacto, podemos determinar que de alguna manera queremos que una batalla finalice lo antes posible. Y si la batalla en cuestión tiene una alta ponderación, aún mejor. De realizarse una batalla con bajo tiempo de finalización y alta ponderación, estaríamos, aparentemente, maximizando la felicidad y también realizando aquellas batallas que se encuentran entre las batallas más importantes.

En definitiva, queremos realizar primero aquellas batallas con menor tiempo de finalización y mayor ponderación. Pero mencionamos inicialmente que los tiempos de finalización de las batallas dependen del orden en el que éstas se realicen. ¿Cómo determinamos este orden?

Lo que haremos es considerar los t_i. Si una batalla tiene un t_i elevado, tiene mayor probabilidad de tener en consecuencia un F_i elevado, y viceversa.

De esta manera proponemos ordenar las batallas en base al coeficiente C $= t_i/b_i$ de forma ascendente. Para batallas con tiempo de duración pequeño y ponderación alta, C será pequeño y por lo tanto dichas batallas estarán entre las primeras que queremos realizar. En caso contrario, para batallas con tiempo de duración alto (y por lo tanto F_i elevado también) y baja ponderación, C será elevado y dichas batallas estarán entre las últimas que queremos realizar. La felicidad que produce obtener la victoria de estas batallas de por sí no es grande, ya que t_i elevado $\rightarrow$ F_i elevado $\rightarrow$ menos impacto generado $\rightarrow$ menos felicidad obtenida.

Nota: es completamente equivalente ordenar por D $= b_i/t_i$ de manera descendente. Aplica el mismo razonamiento pero a la inversa.

Entonces, proponemos como solución general ordenar por t_i/b_i de manera ascendente, esperando obtener el orden óptimo tal que se minimiza la sumatoria ponderada de los tiempos de finalización de las batallas.

2. Posibles soluciones previas descartadas

Previo a la idea de ordenar en base a t_i/b_i, establecimos otras alternativas más simples e intuitivas, que luego fueron descartadas en base al sencillo descubrimiento de contraejemplos que refutaron la idea de que éstos ordenamientos llevan a la obtención de un orden óptimo que minimiza la sumatoria propuesta. Aquí los mencionamos:

2.1. Ordenamiento por Mayor Peso b_i

Realizar las batallas en base a su mayor importancia parece una buena aproximación. No tenemos en cuenta la duración de la batalla, pero esperamos que priorizando su importancia se consiga un óptimo. Podemos encontrar un contrejemplo de forma genérica de la siguiente manera:

Contraejemplo: $[(t_1, 10), (t_2, 1)]$

Para que el ejemplo sirva de contraejemplo, se debería cumplir lo siguiente:

$$(10 \times t_1) + (1 \times (t_1 + t_2)) > (1 \times t_2) + (10 \times (t_2 + t_1))$$
$$10t_1 + t_1 + t_2 > 11t_2 + 10t_1$$
$$t_1 > 10t_2$$

Tomando $t_1 = 20$ y $t_2 = 1$, tenemos:

$$\begin{aligned} \text{Ordenamiento propuesto:} &\quad [(20, 10), (1, 1)] \\ \text{Solución:} &\quad 10 \times 20 + 1 \times 21 = 200 + 21 = 221 \\ \text{Óptima:} &\quad 1 \times 1 + 10 \times 21 = 1 + 210 = 211 \end{aligned}$$

2.2. Ordenamiento por Menor Tiempo t_i

Realizar las batallas en base a su menor tiempo de duración también parece una buena aproximación. No tenemos en cuenta la ponderación de la batalla, pero esperamos que priorizando su corta duración, se consigan inicialmente F_i pequeños que maximizen la felicidad obtenida. Análogamente, podemos encontrar un contrejemplo de forma genérica de la siguiente manera:

Contraejemplo: $[(1, b_1), (10, b_2)]$

Para que el ejemplo sirva de contraejemplo, se debería cumplir lo siguiente:

$$\begin{gathered} (b_1 \times 1) + (b_2 \times 11) > (b_2 \times 10) + (b_1 \times 11) \\ b_1 + 11b_2 > 10b_2 + 11b_1 \\ b_2 > 10b_1 \end{gathered}$$

Tomando $b_1 = 1$ y $b_2 = 20$, tenemos:

$$\begin{aligned} \text{Ordenamiento propuesto:} &\quad [(1, 1), (10, 20)] \\ \text{Solución:} &\quad 1 \times 1 + 20 \times 11 = 1 + 220 = 221 \\ \text{Óptima:} &\quad 20 \times 10 + 1 \times 11 = 200 + 11 = 211 \end{aligned}$$

3. Proposición de un algoritmo Greedy: Ordenamiento por Coeficiente t_i/b_i

Proponemos el siguiente algoritmo Greedy: Dado un arreglo de tuplas de la forma (t_i, b_i, i), se ordena el mismo en base al coeficiente pensado, lo que consiste en una **optimización**. Lo que se busca es realizar la batalla con menor relación t_i/b_i (o mayor relación b_i/t_i) que todavía no hayamos realizado. Para no tener que buscar ese mínimo (o máximo) a cada paso del algoritmo, ordenamos inicialmente el arreglo.

Luego, aplicaremos una regla sencilla para obtener el óptimo local al estado actual: Parados sobre el arreglo en la batalla con menor relación tiempo/peso que todavía no haya sido considerada, se la considerara siguiente en la sumatoria. Aplicamos dicha regla iterativamente esperando obtener un óptimo global (el orden óptimo de las batallas que minimiza la sumatoria propuesta).

El algoritmo propuesto es el siguiente, el cual devuelve la sumatoria mínima y el orden óptimo en el que ésta se minimiza:

```
1  def ordenar_por_menor_relacion_tiempo_peso(arr):
2      return sorted(arr, key= lambda batalla : batalla[TIEMPO]/batalla[PESO])
3
4  def cal_sumatoria(arr):
5      arr = ordenar_por_menor_relacion_tiempo_peso(arr)
6
7      sumatoria = 0
8      tiempo_acumulado = 0
9      orden_batallas = []
10
11     for batalla in arr:
12         tiempo_acumulado += batalla[TIEMPO]
13         sumatoria += (batalla[PESO] * tiempo_acumulado)
14         orden_batallas.append(batalla[IDX])
15
16     return sumatoria, orden_batallas
```

La complejidad del algoritmo propuesto es $\mathcal{O}(nlogn)$, y esta radica enteramente en el ordenamiento (sorted() implementa el método de ordenamiento Timsort, ordenamiento comparativo derivado de Merge Sort e Insertion Sort, y cuya complejidad es $\mathcal{O}(nlogn)$. El resto de las operaciones (asignaciones, operaciones aritméticas, retornos, etc) son $\mathcal{O}(1)$. Por supuesto que las operaciones comprendidas entre las lineas 12 y 14 están dentro de un loop, quedando ese bloque en complejidad $\mathcal{O}(n)$. Por propiedades de notación Big O, la complejidad final del algortimo queda en $\mathcal{O}(nlogn)$.

3.1. Demostración de que el Algoritmo es Óptimo por Inversiones

Se busca demostrar que ordenar los elementos por t_i/b_i de forma ascendente minimiza la sumatoria desde 1 hasta n de $b_i * F_i$, donde F_i es la suma de todos los t anteriores incluido el actual ($F_i = t_1 + t_2 + ... + t_i$, $F_1 = t_1$).

Aclaración: los i = (1, 2, ..., n) dados por el orden inicial de las batallas, luego de ordenar estas últimas según el criterio propuesto, quedan desordenados (representan el orden óptimo en el que deberían realizarse las batallas, o al menos eso suponemos). Para lo que sigue de la demostración y para simplificar, tomamos nuevos súbindices i = (1, 2, ..., n) pero que ahora representan los índices de las batallas ya ordenadas según nuestro criterio.

Dicho esto, supongamos que hay una inversión en el orden óptimo de los elementos. Es decir, hay dos elementos contiguos (t_i, b_i) y (t_j, b_j) tales que $i < j$ pero $t_i/b_i > t_j/b_j$. Ahora, comparamos los términos que involucran a estos dos elementos en la sumatoria:

- Para el elemento (t_i, b_i), su contribución a la sumatoria es: b_i * ($t_1 + t_2 + ... + t_i$)
- Para el elemento (t_j, b_j), su contribución a la sumatoria es: b_j * ($t_1 + t_2 + ... + t_i + t_j$)
- La contribución de ambos a la sumatoria resulta: b_i*t_1 + b_i*t_2 + ... + b_i*t_i + b_j*t_1 + b_j*t_2 + ... + b_j*t_i + b_j*t_j

Dado que el orden es óptimo, al intercambiar los elementos inversibles, la sumatoria resultante del nuevo orden obtenido no puede ser mejor (en nuestro caso, mas pequeña) que la resultante del orden óptimo, porque justamente es el óptimo, aquel orden que garantiza la menor sumatoria posible. Ésta no debería poder mejorarse (minimizarse aún más). Veamos que ocurre si intercambiamos la batalla i y la batalla j:

- Para el elemento (t_j, b_j), su contribución a la sumatoria es: $b_j * (t_1 + t_2 + ... + t_j)$
- Para el elemento (t_i, b_i), su contribución a la sumatoria es: $b_i * (t_1 + t_2 + ... + t_j + t_i)$
- La contribución de ambos a la sumatoria resulta: $b_j * t_1 + b_j * t_2 + ... + b_j * t_j + b_i * t_1 + b_i * t_2 + ... + b_i * t_j + b_i * t_i$

Todos los términos de ambas contribuciones son iguales salvo por $b_j * t_i$ (en la primera contribución, con el batalla i antes que la j) y $b_i * t_j$ (en la segunda contribución, con la batalla j antes que la i). Pero supusimos inicialmente que $t_i/b_i > t_j/b_j$, entonces $b_j * t_i > b_i * t_j$. Esto implica que al intercambiar estas dos batallas en el orden óptimo, obtendremos una sumatoria menor, lo que contradice la suposición de que el orden original era el óptimo.

Queda demostrado entonces por el absurdo, que no es posible un órden óptimo con alguna inversión. Eso demuestra que nuestro orden, aquel que ordena las elementos por t_i/b_i de forma ascendente es de hecho el orden óptimo, y aquel que minimiza la sumatoria ponderada de los tiempos de finalización de las batallas.

Segunda aclaración: por supuesto que una inversión puede darse de forma no consecutiva. A lo largo de esta demostración, supusimos en todo momento que los elementos i y j eran consecutivos, pero esto podría no ser así.

En caso de que no sea así, existirán otras inversiones entre éstos y los elementos del medio. Es decir, si existe una inversión (entre elementos no consecutivos) existirán siempre inversiones entre elementos consecutivos (*). La idea subyacente es ir intercambiando éstas hasta llegar a un orden sin inversiones: el óptimo.

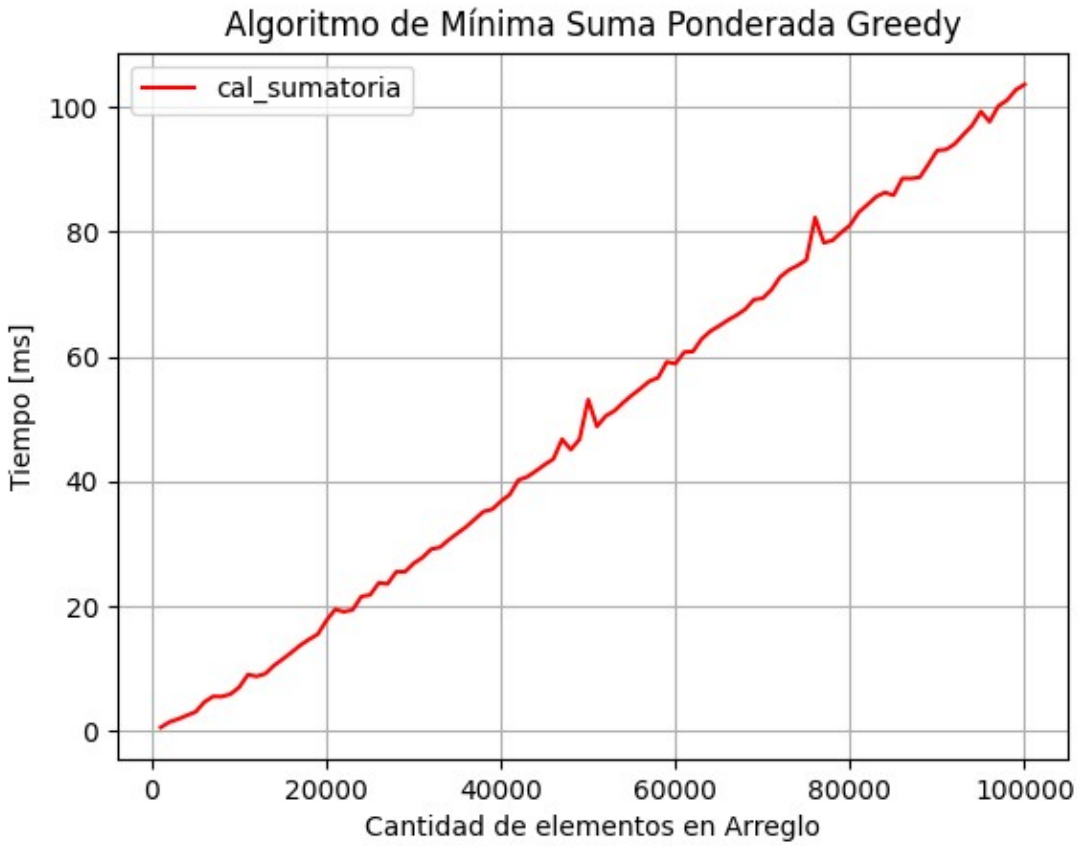
(*) Dado el arreglo [1,4,3,2,5] no sólo hay inversión entre el 4 y el 2 sino que también entre el 4 y el 3 (consecutivos) y entre el 3 y el 2 (consecutivos).

3.2. Consideraciones Extra

- La variabilidad de los valores de t_i y b_i no afecta la optimalidad del algoritmo planteado. Sin importar los valores de t_i y b_i, el algoritmo siempre sigue los mismos pasos: ordena por menor relación t_i/b_i, y luego calcula la sumatoria en base a este orden resultante. Hemos demostrado que el algoritmo es óptimo, lo que significa que dado cualquier set de datos de t_i y b_i, siempre obtendremos el orden óptimo y por lo tanto la sumatoria mínima, independientemente de los valores de t_i y b_i.
- La variabilidad de los valores de t_i y b_i no afecta los tiempos de ejecución del algoritmo, siempre y cuando $M < N$ y $P < N$, siendo N la cantidad de batallas, M el valor que puede tomar t y P el valor que puede tomar b. Esta aclaración es de suma importancia ya que si M y P son mucho más grandes que N para cada t_i, b_i entonces la complejidad del algoritmo necesariamente termina siendo $\mathcal{O}(N * P * M)$ dado que en cada iteración del algoritmo se multiplican estos números grandes. No obstante para este caso, no tiene mucho sentido hablar de t_i extremadamente grandes dado que una batalla no puede durar por tanto tiempo (además de que es improbable que cada batalla tenga la misma duración), por otra parte los pesos en principio no deberían tener valores similares puesto que en un escenario real, una batalla decisiva siempre va a tener más peso que una batalla la cual se consideraba casi ganada desde un principio.

4. Mediciones

Con el fin de comprobar si la complejidad del algoritmo era consistente con el tiempo de ejecución del programa, se crearon sets de datos (usando el modulo `random` que provee el lenguaje) de un largo entre 1000 y 100000 (los elementos (t_i/b_i) varían en un rango de 1 a 999). Se ejecutó el algoritmo para cada set de datos y se le tomó su tiempo de ejecución un total de 5 veces para poder tener un resultado promedio un poco más aproximado y real. Los resultados se presentan en el siguiente gráfico:



Se puede observar que el algoritmo tiene una tendencia un poco peor que lineal en función de la cantidad de elementos de cada set de datos. Resulta complicado dar por hecho que la complejidad sea exactamente la propuesta teóricamente pues pareciera ser hasta mejor que la misma. No obstante sigue siendo un rendimiento mucho mas rápido que un algoritmo cuadrático. Se puede argumentar que la tendencia mostrada en el gráfico esta vinculada a el rendimiento que tenga ***Timsort***

5. Conclusión

Mediante el análisis del problema, la propuesta de diferentes opciones de resolver el problema (y su posterior refutación mediante contraejemplos), el desarrollo del algoritmo, los tests desarrollados y la demostración por inversiones, llegamos a la conclusión de que la forma óptima de resolver este problema es mediante el algoritmo Greedy planteado, el cual ordena las batallas en base al coeficiente C = t_i/b_i de manera ascendente y propone desarrollar las batallas en el orden dado, logrando así optimizar la sumatoria ponderada.

Cabe destacar que también es posible llegar a la solución óptima ordenando en base al coeficiente D = b_i/t_i y ordenar el arreglo de manera descendente.

Como conclusión final de este informe, luego de todo lo desarrollado en el mismo, podemos afirmar con seguridad que este algoritmo va a devolver siempre la resolución óptima del problema y que el Señor del Fuego va a estar satisfecho.