.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 0
Lineamientos sobre informes

30 de marzo de 2024

Zielonka Axel
110310

Petean Marina
110564

Romano Nicolas
110830

1. La Nacion del Fuego

1.1. Analisis del problema

1.2. Algoritmo Greedy propuesto

Aplicamos una regla sencilla que nos permita obtener el óptimo local a mi estado actual: Esto lo logramos ordenando nuestro arreglo de mayor a menor b/t Aplicamos iterativamente esa regla, esperando que nos lleve al óptimo general: Calculamos, iterando de manera ordenada el arreglo, el coeficiente de impacto para obtener la solución óptima.

1.3. Optimalidad

aca hablamos de la optimalidad del algoritmo y hacemos el analisis de si (y como) afecta la variabilidad de los valores de t_i y b_i al algoritmo planteado.

Definimos una inversión de dos elementos batalla[i] y batalla[j] dentro un orden de batallas a toda $a_i < a_j$ tal que $i < j$. Sea $a_i = \frac{b_i}{t_i}$.

Supongamos que tenemos un orden de batallas tal que la suma ponderada $\sum_{i=0}^{n} f_i \cdot b_i$ sea la óptima. Supongamos también que ese orden posee inversiones como la anteriormente definida. Ahora cuando invertimos esas inversiones, es decir, reducimos la cantidad de inversiones del orden de batalla, nosotros no podemos empeorar la suma ponderada final, podrá ser igual o mejor. Entonces al revertir todas las inversiones, nos queda un array ordenado por la relación $a = \frac{b}{t}$ de mayor a menor.

¿Por qué al revertir la inversión no podemos empeorar la suma ponderada? Supongamos que tenemos solo dos batallas (t_0, b_0) y (t_1, b_1) ordenadas tal que $a_0 \leq a_1$. Queremos demostrar que $t_0 \cdot b_0 + b_1 \cdot (t_0 + t_1)$ es siempre mayor o igual que $t_1 \cdot b_1 + b_0 \cdot (t_1 + t_0)$. En otras palabras, estamos poniendo a (t_1, b_1) como primera batalla y a (t_0, b_0) como segunda para demostrar que es igual o mejor que el orden anterior. Para esto, nos basta demostrar con método directo que la inecuación cumple.

$$t_1 \cdot b_1 + b_0 \cdot (t_1 + t_0) \leq t_0 \cdot b_0 + b_1 \cdot (t_0 + t_1)$$

Distribuimos

$$t_1 \cdot b_1 + b_0 \cdot t_1 + b_0 \cdot t_0 \leq t_0 \cdot b_0 + b_1 \cdot t_0 + b_1 \cdot t_1$$

Restamos $t_1 \cdot b_1$ y $b_0 \cdot t_0$ de ambos lados

$$b_0 \cdot t_1 \leq b_1 \cdot t_0$$

$$\frac{b_0}{t_0} \leq \frac{b_1}{t_1}$$

Es decir: $a_0 \leq a_1$. Una hipótesis de la cual partimos. Lo que quiere decir que el que tenía mejor o igual relación peso-duración era igual o más óptimo que el otro.

2. Minimizar la suma ponderada

En este trabajo de ejemplo realizaremos el análisis bla bla bla

2.1. Algoritmo

A continuación se muestra el código de solución del problema.

```
def minimizar_suma(batallas):
    batallas_indices = crear_arreglo_con_indice(batallas)
    batallas_ordenadas = sorted(batallas_indices, key=lambda batalla: -(batalla
    [0][1] / batalla[0][0]))
    f = []
    for i in range(len(batallas_ordenadas)):
        if i == 0:
            f.append(batallas_ordenadas[i][0][0])
        else:
            f.append(f[i - 1] + batallas_ordenadas[i][0][0])

    suma_ponderada = 0
    for i in range(len(batallas_ordenadas)):
        suma_ponderada += f[i] * batallas_ordenadas[i][0][1]

    return suma_ponderada, batallas_ordenadas
```

Ahora deberiamos explicar con palabras un poco que hacemos en codigo

2.2. Complejidad

aca hablamos de la complejidad del algoritmo y hacemos el analisis de si (y como) afecta la variabilidad de los valores de t_i y b_i a los tiempos del algoritmo planteado.

2.3. Mediciones

aca hablamos de las mediciones que hicimos y como se conectan con las complejidades que hablamos anteriormente. (graficos aca)

3. Testing

Aca hablamos un poco de los tests que hicimos, algunos casos borde que quizas generen duda y cualquier cosa que sea empirica.

4. Conclusiones

Acá irían las conclusiones de todo nuestro trabajo :)