

TEORÍA DE ALGORITMOS
(75.29) CURSO BUCHWALD - GENENDER

Trabajo Práctico 1

La mafia de los algoritmos greedy

2 de abril de 2025

Pablo Gaston
Choconi
106388

Ezequiel Matias
Gacitua
112234

Abel Tomas
Modesti Funes
111932

1. Análisis del problema

1.1. El problema

El problema el cual vamos a resolver en este trabajo es el siguiente:

Trabajamos para la mafia de los amigos Amarilla Pérez y el Gringo Hinz. En estos momentos hay un problema: alguien les está robando dinero. No saben bien cómo, no saben exactamente cuándo, y por supuesto que no saben quién. Evidentemente quien lo está haciendo es muy hábil (probablemente haya aprendido de sus mentores).

La única información con la que contamos son n transacciones sospechosas, de las que tenemos un timestamp aproximado. Es decir, tenemos n tiempos t_i , con un posible error e_i . Por lo tanto, sabemos que dichas transacciones fueron realizadas en el intervalo $[t_i - e_i; t_i + e_i]$.

Por medio de métodos de los cuales es mejor no estar al tanto, un interrogado dio el nombre de alguien que podría ser la rata. El Gringo nos pidió revisar las transacciones realizadas por dicha persona... en efecto, eran n transacciones. Pero falta saber si, en efecto, conciden con los timestamps aproximados que habíamos obtenido previamente.

El Gringo nos dio la orden de implementar un algoritmo que determine si, en efecto, las transacciones coinciden. Amarilla Perez nos sugirió que nos apuremos, si es que no queremos ser nosotros los siguientes sospechosos.

- Hacer un análisis del problema, y proponer un algoritmo greedy que obtenga la solución al problema planteado: Dados los n valores de los timestamps aproximados t_i y sus correspondientes errores e_i , así como los timestamps de las n operaciones s_i del sospechoso (pueden asumir que estos últimos vienen ordenados), indicar si el sospechoso es en efecto la rata y, si lo es, mostrar cuál timestamp coincide con cuál timestamp aproximado y error. Es importante notar que los intervalos de los timestamps aproximados pueden solaparse parcial o totalmente.
- Demostrar que el algoritmo planteado determina correctamente siempre si los timestamps del sospechoso corresponden a los intervalos sospechosos, o no. Es decir, si conciden, que encuentra la asignación, y si no conciden, que el algoritmo detecta esto, en todos los casos.
- Escribir el algoritmo planteado. Describir y justificar la complejidad de dicho algoritmo. Analizar si (y cómo) afecta la variabilidad de los valores de los diferentes valores a los tiempos del algoritmo planteado.
- Realizar ejemplos de ejecución para encontrar soluciones y corroborar lo encontrado. Adicionalmente, el curso proveerá con algunos casos particulares para que puedan usar de prueba.
- Hacer mediciones de tiempos para corroborar la complejidad teórica indicada. Esto, por supuesto, implica que deben generar sus sets de datos. Agregar los casos de prueba necesarios para dicha corroboración. Esta corroboración empírica debe realizarse confeccionando gráficos correspondientes, y utilizando la técnica de cuadrados mínimos.
- Agregar cualquier conclusión que les parezca relevante.

1.2. Solución propuesta

COMPLETAR...

AQUI DEBERIA ESTAR EL ANALISIS DEL PROBLEMA Y LA JUSTIFICACIÓN DE PORQUE FUNCIONA

2. Algoritmo para asignar operaciones a intervalos

2.1. Código

A continuación se muestra el código de la solución greedy del problema.

```
1     for timestamp in sorted_timestamps:
2         if operation >= timestamp["startTime"] and operation <= timestamp["
3             endTime"] and timestamp["operation"] is None:
4             if timestampContainingOperationThatEndsSooner is None or timestamp["
5                 endTime"] < timestampContainingOperationThatEndsSooner["endTime"]:
6                 # Se guarda el timestamp que contiene la operacion,
7                 # termina antes y no tenia ninguna operacion asignada
8                 timestampContainingOperationThatEndsSooner = timestamp
9             if timestampContainingOperationThatEndsSooner is None:
10                # Ningun timestamp contenia a la operacion. No es la rata
11                return False
12            else:
13                timestampContainingOperationThatEndsSooner["operation"] = operation
14
15    return True
```

2.2. Análisis de complejidad

La complejidad del algoritmo planteado posee dos partes identificables, el ordenamiento mediante MergeSort de los timestamp por tiempo de fin y la asignación de las transacciones a un timestamp

Primero, hagamos un rapido analisis del ordenamiento MergeSort, este algoritmo divide el arreglo en subarreglos de la mitad de tamaño hasta llegar a un unico elemento, el cual se considera ordenado, luego combina los mismos de manera ordenada hasta llegar a nuestro arreglo original ordenado en su totalidad. Si hablamos solo de temas de complejidad esta parte de nuestra respuesta tiene $\mathcal{O}(n \log n)$.

Por otro lado tenemos la parte principal del algoritmo, lo que busca es asignar cada transacción a un timestamp diferente. Consigue esto tomando y verificando dentro de que timestamps se encuentra y asociándolo al que termina antes, y esto se realiza por cada transacción. Por lo tanto podemos deducir que esta parte del algoritmo es de $\mathcal{O}(n^2)$.

Si vemos el algoritmo en su totalidad tiene $\mathcal{O}(n \log n) + \mathcal{O}(n^2)$, pero cuando nos vamos a n grandes, que son los casos que nos interesan para el análisis de la complejidad observamos que el algoritmo es de $\mathcal{O}(n^2)$

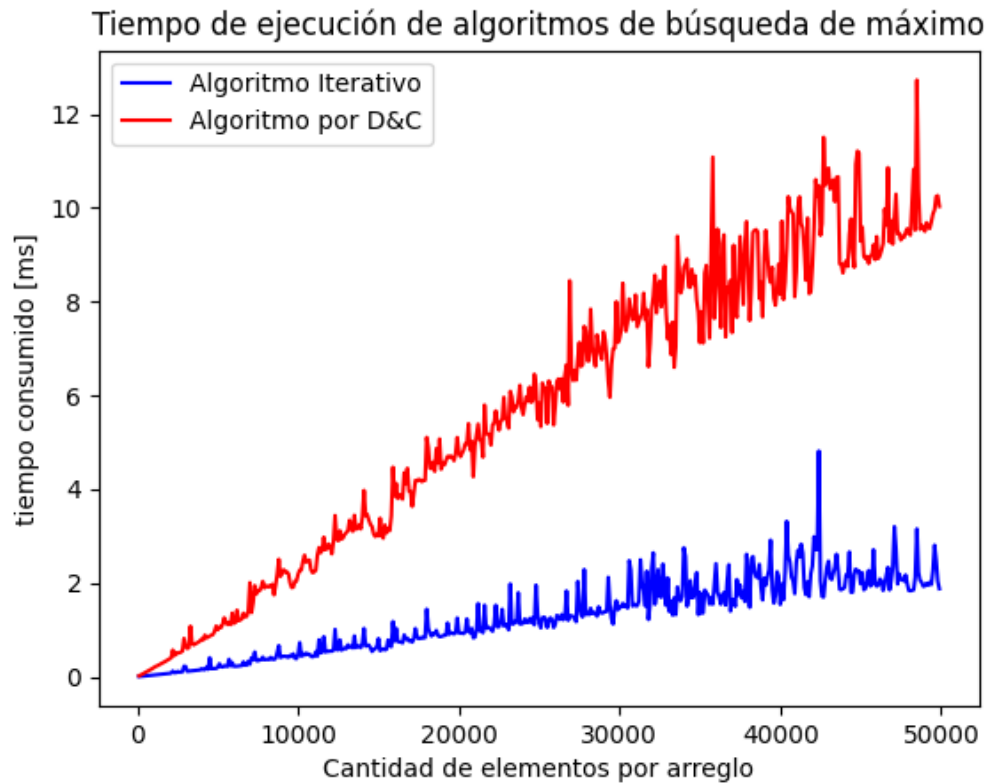
Como se va a ver en las mediciones, esto no es exacto sino que es un aproximado en los peores casos

(Lo dejo por si agregamos una ecuacion, asi esta el ejemplo)

$$\mathcal{T}(n) = 2\mathcal{T}\left(\frac{n}{2}\right) + \mathcal{O}(n)$$

3. Mediciones

Se realizaron mediciones 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`).



Como se puede apreciar, ambos algoritmos tienen una tendencia efectivamente lineal en función del tamaño de la entrada, si bien el algoritmo iterativo es más veloz en cuestión de constantes.

4. Conclusiones

Acá irían las conclusiones de todo nuestro trabajo :)