Trabajo Práctico

75.29 Teoría de Algoritmos I

FACULTAD
DE INGENIERIA

Universidad de Buenos Aires

Profesores:

Víctor Podberezski

1° Cuatrimestre 2020

Integrantes:

NOMBRE Y APELLIDO	PADRON
Hernán Arroyo García	91257
Ruben Jimenez	92402
Federico Rodrigez Longhi	93336
Emanuel Condo	94773

Parte 1: Problema de ausentismo.

1. Pseudocódigo

```
N = cantidad EMPLEADOS
INTERVALOS = []
EMPLEADOS = ordenar_por_Si(EMPLEADOS, 'creciente') O(nlog2n)
INTERVALO = intervalo(EMPLEADOS[0]) O(1)
Para k desde 1 hasta N O(n)
   INTERVALO_SIG_EMP= intervalo(EMPLEADOS[k]) O(1)
   Si INTERVALO_SIG_EMP se superpone con INTERVALO O(1)
      INTERVALO = interseccion(INTERVALO,
INTERVALO_SIG_EMP) O(1)
      Si k == N
         Agregar INTERVALO a INTERVALOS
   Sino
      Agregar INTERVALO a INTERVALOS
      INTERVALO = INTERVALO_SIG_EMP
Fin Para
Para cada INTERVALO en INTERVALOS O(n)
   tiempo = cualquier_punto(INTERVALO)
   UBICACION_EMPLEADOS = DTI(tiempo) O(1)esta acotado
Fin Para
Para cada UBICACION_EMP en UBICACION_EMPLEADOS O(n)
   imprimir(UBICACION_EMP)
Fin Para
```

2. Tipo de algoritmo y justificación

Objetivo

Obtener la ubicación de todos los empleados en su horario laboral, minimizando la cantidad de llamados al sistema DTI.

Estrategia

Ordenar los empleados por tiempo de inicio de actividad (Si), de tal forma que al evaluar vayamos tomando a los empleados según el menor tiempo de ingreso al trabajo.

El INTERVALO de trabajo del primer empleado puede intersecarse con el intervalo del siguiente empleado. Si se intersecan actualizamos INTERVALO con la interseción de los intervalos de ambos. Si no se intersecan, se genera un nuevo INTERVALO (con los datos del empleado que tiene la franja horaria distinta al anterior empleado). Con este nuevo INTERVALO se vuelve a evaluar las intersecciones para los siguientes empleados. Cada vez que no se intersecan los intervalos de los empleados, se guarda en una estructura la intersección de esos empleados. Asi al final de procesar todos los empleados, obtendremos una estructura de INTERVALOS, la cual contiene los intervalos optimos, para los cuales podremos elegir un punto cualquiera y hacer los llamados a la funcion DTI y armar un reporte con la ubicacion laboral de los empleados.

Justificación

Se puede observar que en cada iteración el intervalo de tiempo de consulta se hace más chico o se mantiene y va abarcando a más empleados hasta donde sea posible. Cuando el intervalo deja de intersecarse con más empleados, se elige un valor de tal intervalo para hacer la consulta. y al dejar de intesecarse es porque se necesita otro intervalo de consulta obligatoriamente del cual se tomará otro valor. Esto quiere decir que es imposible hacer la consulta en un único tiempo "t" para poder localizar a todos los empleados en su horario laboral.
Entonces, como en cada iteracion buscamos tomar a todos los empleados compatibles para hacer una unica consulta. Se puede decir que, es una elección Greedy, ya que buscamos mejorar en cada paso.

3. Complejidad

- Complejidad temporal $O(nlog_2n)$. Es la suma de las complejidades: ordenar **EMPLEADOS** por Si de forma ascendente $O(log_2n)$; recorrer los empleados, para armar los **INTERVALOS** optimos $O(n)$. Recorrer INTERVALOS para armar **UBICACION_EMPLEADOS** (las posiciones de los empleados) $O(n)$. Recorrer UBICACION_EMPLEADOS y mostrar el reporte de ubicacion de los empleados $O(n)$. Por lo tanto la complejidad total es $O(nlog_2n + n + n + n)$, esto esta acotado por $O(nlog_2n)$.

- Complejidad espacial$O(n)$. Tenemos 3 estructuras, **EMPLEADOS**, **INTERVALOS** y **UBICACION_EMPLEADOS**. Asumimos que se ordena sobre la misma estructura EMPLEADOS $O(1)$. El tamaño de INTERVALOS/UBICACION_EMPLEADOS en el peor de los casos puede ser "n", $O(n)$. Por lo tanto tenemos que la complejidad espacial total es $O(1 + n + n)$,esto esta acotado por $O(n)$.

4. Justificacion solución óptima

Supongamos que tenemos una solución óptima (Op). Si la solución propuesta (llamada MIN_DTI) no es óptima, entonces MIN_DTI realiza al menos una consulta más que Op. Entonces $\#Consultas(MIN\_DTI) > \#Consultas(Op)$. Esto quiere decir que MIN_DTI según el algoritmo tiene al menos un intervalo de consulta de más. Pero, eso no se puede dar, porque al recorrer los empleados (ordenados por tiempo de inicio de actividad) se van intersectando la franja horaria de los empleados y si deja de intersecarse, es porque es necesario una nueva consulta al sistema, porque el horario laboral de este último empleado NO coincide con todos los anteriores. El proceso se repite y genera al final k intervalos con k $<=$ n, y cada uno de esos intervalos son necesarios. Por lo tanto, podemos decir que el número de intervalos es mínimo y la cantidad de consultas al sistema DTI también.

Parte 2: Una nueva regulación industrial.

1. Pseudocódigo del proceso "A"

```
ProcesoA()
Constructor lote (O(n) lo desestimamos porque es la lectura del archivo)
Inicializo esValido en falso
Para todo item ∈ lote y no esValido
    Para todo item2 ∈ lote y no esValido
        Si son iguales item y item2, incrementamos contador
        Si contador es mayor a mitad de lote, activo esValido y fin los loops
    Fin Para
Fin Para
Retornar esValido
```

2. Pseudocódigo del proceso "B"

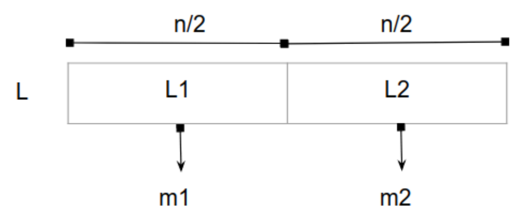
```
ProcesoB()
Constructor lote (O(n) lo desestimamos porque es la lectura del archivo)
Ordeno lote (O(nlog_2 n))
Inicializo esValido en falso
Para todo item ∈ lote y no esValido
    Si son iguales item y item siguiente, incrementamos contador
    Si son distintos inicializo contador en cero
    Si contador es mayor a mitad de lote, activo esValido y fin los loops
Fin Para
Retornar esValido
```

3. Proceso "C"

Aplicando divide y venceras, buscamos particionar el problema en en sub problemas, resolverlos y finalmente resolver el problema inicial.

Si el lote "L" tiene un item mayoritario "m", entonces m tiene que ser mayoritario del sub problema L1 ó del sub problema L2. Esto es:

$\frac{|L|}{2} < |L|_m$, entonces $\frac{|L_1|}{2} < |L_1|_m$ ó $\frac{|L_2|}{2} < |L_2|_m$



Demostración: Por el absurdo. Supongamos que "m" no es mayoritario de L1, ni de L2. Esto es: $|L_1|_m \leq \frac{|L_1|}{2}$ y $|L_2|_m \leq \frac{|L_2|}{2}$

Entonces,

$\frac{|L|}{2} < |L|_m$ // Si existiera mayoritario se tendria que cumplir
$|L|_m = |L_1|_m + |L_2|_m$
$|L_1|_m + |L_2|_m \leq \frac{|L_1|}{2} + \frac{|L_2|}{2}$ // reemplazando hipotesis
$\frac{|L_1|}{2} + \frac{|L_2|}{2} = \frac{|L_1|+|L_2|}{2} = \frac{|L|}{2}$ // Absurdo, son iguales

Sabiendo esto, podemos calcular el mayoritario en L1 y L2. Si existe m1 ó m2, estos serán candidatos para ser mayoritario del lote L, se tendrá que contar las apariciones en el lote para asignarlo como mayoritario. Sino no podemos afirmar que no existe mayoritario de L.

Pseudicódigo:

```
ProcesoC()
    Constructor lote L[1...n](O(n) lo desestimamos porque es la lectura
del archivo)
    MayoritarioRecursivo(L[1...n])
MayoritarioRecursivo(L[1 . . . n]):
    Si n = 1: Devolver L[1] (O(1))
    Sea m1 = MayoritarioRecursivo(L[1 . . . n/2])
    Sea m2 = MayoritarioRecursivo(L[n/2 + 1 . . . n])
    Si Apariciones(L, m1, 1, n) > n/2: Devolver m1 (O(n))
    Si Apariciones(L, m2, 1, n) > n/2: Devolver m2 (O(n))
    Devolver vacio (O(1))
Apariciones(L[1 . . . n], x, i, j):
    Inicializar contador
    Para cada k ∈ {i, . . . , j}:
        Si L[k] = x:
            Incrementar en uno contador
    Devolver contador
```

4. Complejidades temporales y espaciales

ProcesoA

- Complejidad temporal $O(n^2)$, es cuadratica porque por cada item del lote se compara con todos los otros elementos.
- Complejidad espacial$O(1)$, es constante porque no se tiene memoria adicional a la entrada (lote).

ProcesoB

- Complejidad temporal $O(nlog_2n)$, porque se ordena el lote, esto es de complejidad $O(nlog_2n)$, después se recorre todos los items ordenados contando los items de igual volumen $O(n)$.
- Complejidad espacial$O(n)$, es lineal porque el ordenamiento aloca un nuevo lote ordenado en memoria.

ProcesoC

- Complejidad temporal $O(nlog_2n)$. Es un merge sort modificado. Se divide en mitades hasta llegar a lotes de tamaño uno. Complejidad de caso base $O(1)$, complejidad para el calculo de apariciones $O(cn)$. El tiempo de ejecucion del procesoC, es la sumatoria de todos los tiempos de los subproblemas $(l * cn)$, siendo c=cte y l=altura de arbol, entonces $l = log_2n + 1$. Reemplazando, el tiempo de ejecucion quedaria que el tiempo de ejecucion es $cn.log_2n + cn$. Esto esta acotado por $O(nlog_2n)$.
- Complejidad espacial$O(log_2n)$, ya que a pesar de que la division y conquista no usa memoria adicional más que la del lote, se tiene una porción de memoria no constante en el stack para la recursion y esta es de $O(log_2n)$.

5. Relacion de recurrencia

Para el procesoC, se tiene la relacion de recurrencia:

$$T(n) = 2T(\tfrac{n}{2}) + 2n$$

6. Teorema Maestro

$T(n) = 2T(\frac{n}{2}) + n)$, a = 2, b = 2, f(n) = 2n
Cumple caso 2 ?
$f(n) = 0(nlog_b a)$
$f(n) = 0(nlog_2 2)$
$f(n) = 0(n)$ // f(n) esta acotada superiormente y inferior n
Entonces:
$T(n) = 0(log_2 n.(n^{log_b a}))$
$T(n) = 0(log_2 n.(n^{log_2 2}))$
$T(n) = 0(nlog_2 n.)$