Algoritmos y Estructura de Datos 2

Recuperatorios

X7

Alumno: Leandro Carreira
LU: 669/18

Link a documento online (en caso que algún caracter se haya pasado mal a .pdf):
https://docs.google.com/document/d/1YTTKsL5jPRnfuZErhZuG9wHYfTRbwfUTkjYPc99aJyU/edit?usp=sharing

Ejercicio X7

7. Ejercicio X7 — Ordenamiento

En la famosa playa La Perla quieren sacar una foto área artística donde todas las sombrillas estén ordenadas formando un cuadro de colores único. Cada sombrilla podríamos considerarla como una tupla de ⟨Color, Diámetro⟩ donde un Color es un código hexadecimal de u dígitos y Diámetro un natural positivo. Como las sombrillas tienen tamaños estándar, nos indican que hay a lo sumo k posibles tamaños de sombrillas (pero no se aclara a priori cuáles k). La muchachada nos solicita un algoritmo que ordene las sombrillas por color creciente y en caso de empate por tamaño decreciente. En concreto, diseñar el siguiente algoritmo:

$$\texttt{SombrillasSort}(\textbf{in } A : \text{arreglo(Sombrilla)}, \textbf{in } B : \text{arreglo(Diámetro)}) \longrightarrow C : \text{arreglo(Sombrilla)}$$

con **complejidad** $O(n \cdot u + (k+n)\log k)$ **en peor caso**, que genere un arreglo con todas las sombrillas ordenadas como se mencionó previamente. El arreglo A de tamaño n contiene las sombrillas y el B de tamaño k los diámetros posibles. Por ejemplo:

```
SombrillasSort([<"0x6A319",21>, <"0x6A319",27>, <"0xA1719",11>, <"0xA1720",21>], [27,11,21])
==> [<"0x6A319", 27>, <"0x6A319", 21>, <"0xA1719", 11>, <"0xA1720", 21>]
```

En este ejemplo: $u = 5$ y $k = 3$.

Observaciones:

Cada caracter del color, al ser un valor hexadecimal, estará acotado por una constante: 16

Asumo que u es la cantidad de dígitos **sin** contar el "**0x**" del comienzo.

Para los for loop uso que *i in 0 to **4** ≡ i in [0,1,2,**3**]* (o sea, no inclusive el limite derecho)

Uso ***tupla.prim*** y ***tupla.segu*** para acceder a primer y segundo elemento de una tupla.

Uso operador **//** como **división entera** que **redondea hacia abajo**

Uso función **MergeSort** pasando parámetro ***"orden"*** que establece el orden de ordenamiento (no cambia la complejidad del algoritmo, solo invierte comparaciones de orden por defecto si es "decreciente").

```
SombrillasSort(in A: arreglo(Sombrilla), in B: arreglo(Diámetro))→ C: arreglo(Sombrilla)
    int n ← tam(A)                                              O(1)
    int k ← tam(B)                                              O(1)
    // Ignoro caracteres "0x" en u
    int u ← tam(A[0].prim) - 2                          O(1)

    arreglo(Sombrilla) C ← arreglo(n)                           O(n)

    // Ordeno tamaños en B con algo. estable Merge Sort
    B_ord ← MergeSort(B, orden="decreciente")                   O(k log k)

    // Hay una correspondencia directa entre los índices
```

```
// y los tamaños B_ord ordenados decrecientemente
// ej: [27, 21, 11] se corresponde con [0, 1, 2]
arreglo(lista(Sombrilla)) A_tamaños ← crearArreglo(k)      O(k)
for t in 0 to k do                                         O(k)
     A_tamaños[t] ← Vacía()                                O(1)
end for

// Ordeno sombrillas por tamaño decreciente
int idx ← 0                                                O(1)
for sombri in A do                                         O(n * log k)
     idx ← deTamañoAIdx(B_ord, sombri.segu)                O(log k)
     AgregarAtrás(A_tamaños[idx], sombri)                  O(1)
end for

// Vuelvo a convertir A en un único arreglo
A ← concatenarEnArreglo(A_tamaños, n)                      O(n * u)

// Antes de volver a ordenar, creo acceso rápido a
// los dígitos de los colores ordenados
// Gano acceso a string como arreglo de chars
arreglo(arreglo(char)) colores ← crearArreglo(n)           O(n)
i ← 0                                                      O(1)
for sombri in A do                                         O(n * u)
     colores[i] ← deStringAArreglo(sombri.prim)            O(u)
     i++                                                   O(1)
end for

// Diccionario auxiliar para mapear hexadecimal (char) a decimal (int)
dicc(char, int) deHexaADec ← diccAVL()                     O(1)
i ← 0                                                      O(1)
for c in "0123456789ABCDEF" do                             O(16 * 1) ≡ O(1)
     definir(deHexaADec, c, i)                             O(log 16) ≡ O(1)
     i++                                                   O(1)
end for

// Ordeno colores con RadixSort, recorriendo dígitos desde las unidades
// Uso dos arreglos de forma intercalada
arreglo(lista(Sombrilla)) A_colorUno ← crearArreglo(16)    O(1)
arreglo(lista(Sombrilla)) A_colorDos ← crearArreglo(16)    O(1)

for i in 0 to 16 do                                        O(1)
     A_colorUno[i] ← Vacía()                               O(1)
     A_colorDos[i] ← Vacía()                               O(1)
end
```

```
    int idx     ← 0                                                    O(1)
    int toWrite ← 0                                                O(1)
    for dígito in 0 to u do                                            O(    )
        // Recorro todos los dígitos de atrás hacia adelante...
        for i in 0 to n do
            // ...para cada una de las Sombrillas de A
            // Obtengo índice del bucket (entre 0 y 15 inclusives)
            idx ← obtener(deHexaADec, colores[i][u-1-dígito])     O(log 16) ≡ O(1)
            if dígito = u-1 then                                          O(1)
                // Primera vez copia de A
                AgregarAtrás(A_colorUno[idx], A[i])                       O(1)
                toWrite ← 2                                               O(1)
            else if toWrite = 1 then
                // Paso de ordDos a ordUno
                AgregarAtrás(A_colorUno[idx], A_colorDos[i])              O(1)
                A_colorDos[i] ← Vacía()                                   O(1)
                toWrite ← 2                                               O(1)
            else
                // Paso de ordUno a ordDos
                AgregarAtrás(A_colorDos[idx], A_colorUno[i])              O(1)
                A_colorUno[i] ← Vacía()                                   O(1)
                toWrite ← 1                                               O(1)
            end if
        end for
    end for

    // Guardo referencia para acceder al último array de sombrillas ordenadas
    if toWrite = 1 then
        ordenadas ← A_colorDos                                            O(1)
    else
        ordenadas ← A_colorUno                                            O(1)
    end if

    // Concateno todas las listas en una única (ordenada)
    C ← concatenarEnArreglo(ordenadas, n)                             O(n * u)



deTamañoAIdx(in tamañosOrd: arreglo(Diámetro), in t: Diámetro) → idx: int
    // Pre: t existe en el arreglo tamañosOrd
    int k      ← tam(tamañosOrd)                                          O(1)
    int izq    ← 0                                                        O(1)
    int der    ← k-1                                                      O(1)
    bool encontrado ← false                                               O(1)
    // Uso búsqueda binaria
```

```
    int idx    ← (der - izq) // 2                                    O(1)
    while (der-izq) > 0 and encontrado = false do                    O(log k)
        if t > tamañosOrd[idx] then                                  O(1)
            // Busco en primera mitad
            der ← idx                          O(1)
            idx ← izq + (der - izq) // 2                             O(1)
        else
            if t < tamañosOrd[idx] then                              O(1)
                // Busco en segunda mitad
                if idx = k-2 then                                    O(1)
                    // Salvo caso borde derecho
                    idx ←  k-1                              O(1)
                    encontrado ←  true                               O(1)
                else
                    izq ← idx                                        O(1)
                    idx ← izq + (der-izq) // 2              O(1)
                end if
            else
                // Lo encontré
                encontrado ← true                                    O(1)
            end if
        end if
    end while


concatenarEnArreglo(in X: arreglo(lista(Sombrilla)), in n: int)
                                                  → aplanado: arreglo(Sombrillas)
    // Función que guarda secuencialmente los elementos de las listas
    // de un arreglo de listas en un único arreglo
    arreglo(Sombrilla) aplanado ← crearArreglo(n)                    O(n)
    int i ← 0                                                        O(1)
    for lis in X do                                                  O(n * u)
        for som in lis do
            aplanado[i] ← som                                        O(u)
            i++                                                      O(1)
        end for
    end for

deStringAArreglo(in str: string) → arr: arreglo(char)
    // Ignoro los primeros dos caracteres
    u ← long(string) - 2                                O(1)
    arreglo(char) arr ← arreglo(u)                               O(u)
    int i ← 0                                                        O(1)
    for c in str do                                                  O(u)
        if i >= 2 then                                               O(1)
```

```
            arr[i-2] ← c                                    O(1)
        end if
        i++                                                 O(1)
    end for
```

---

fin :)