Universidad Nacional Autónoma de México
Facultad de Ciencias
Estructuras de Datos 2022-2
Tarea 04: Análisis de Algoritmos recursivos

Pedro Ulises Cervantes González
confundeme@ciencias.unam.mx

Yessica Janeth Pablo Martínez,
yessica_j_pablo@ciencias.unam.mx

Jorge Macías Gómez
jorgemacias@ciencias.unam.mx

Fecha de entrega: 22 de Abril del 2022
Hora límite de entrega: 23:59

1. Ejercicio 1 (10 puntos)

.-Dado los siguientes algoritmos analiza su complejidad (muestra el paso a paso del análisis):

- **Problema 1:**

```
public static void torreHanoi(int n , char origen, char destino, char aux){
         if(n == 1){
             System.out.println("Mueve el disco 1 de la torre " + origen + " hacia la
                  torre " + destino);
             return;
         }
         torreHanoi(n-1, origen, aux, destino);
         System.out.println("Mueve el disco " + n + " desde la torre "
                 + origen + " hacia la torre " + destino);
         torreHanoi(n-1, aux, destino, origen);
     }

}
```

- **Problema 2:**

```
//Metodo que te devuelve el mcd de dos numeros
public static int mcd(int a, int b) {
       if(b==0){
           return a;
       }

           return mcd(b, a % b);
   }
```

- **Problema 3:**

```
//Metodo que devuelve una palabra invertidad tomando en cuenta su longitud
//Ejemplo: "hola", 2 ; tenemos como resultado "loh"

    public static String palabraInvertida(String palabra, int longitud){
        if(longitud==0){
            return palabra.charAt(longitud)+"";
        }else{
        return palabra.charAt(longitud) + (palabraInvertida(palabra, longitud-1));
        }

    }
```

- **Problema 4:**

```
 //Contamos las veces que se encuentra un caracter en una cadena
 public static int cuentaCaracter(String cadena, char c){
        if(cadena.length() == 0){
            return 0;
        }else{
          return (cadena.charAt(0)== c? 1:0) + cuentaCaracter(cadena.substring(1),c);
        }
    }
```

Respuestas

Equipo:
Bonilla Reyes Dafne
García Ponce José Camilo

- ## Ejercicio 1

  - ### Problema 1

    1. Decidir acerca del parámetro n que indica el tamaño de la entrada del algoritmo:
       n será el número de discos.
    2. Identificar la operación básica del algoritmo:
       Mover discos y resolver un torre de Hanoi será $T(n)$
    3. Determinamos si el número de veces que la operación básica es ejecutada puede variar para diferentes entradas del mismo tamaño n:
       No hay variación.
    4. Expresar como una relación de recurrencia:
       Sea $T(n) = T(n-1) + T(n-1) + C$ para $n > 1$
       $T(1) = C$ una constante, en este caso 1, entonces

       $$T(n) = \begin{cases} 1 \text{ si } n = 1 \\ T(n-1) + T(n-1) + C \text{ para } n > 1 \end{cases}$$

    5. Resolver la relación de recurrencia:
       probar
       - Para $n > 1$
         $T(n) = T(n-1) + T(n-1) + C = T(n-1) + T(n-1) + 1$

- Para $n > 2$, sustituimos $T(n-1) = T(n-2) + T(n-2) + 1$
  $T(n) = (T(n-2) + T(n-2) + 1) + (T(n-2) + T(n-2) + 1) + 1 = T(n-2) + T(n-2) + T(n-2) + T(n-2) + 3$
- Para $n > 3$, sustituimos $T(n-2) = T(n-3) + T(n-3) + 1$
  $T(n) = (T(n-3) + T(n-3) + 1) + (T(n-3) + T(n-3) + 1) + (T(n-3) + T(n-3) + 1) + (T(n-3) + T(n-3) + 1) + 3 = T(n-3) + T(n-3) + T(n-3) + T(n-3) + T(n-3) + T(n-3) + T(n-3) + T(n-3) + 7$
- Podemos determinar un patrón: M
  $T(n) = T(n-(i-1)) + (2^i) - 1$
- Tomando en cuenta la condición inicial, tenemos que se sustituye $i = n$ en la fórmula anterior y se obtiene:
  $T(n) = T(n-n+1) + (2^n) - 1 = T(1) + (2^n) - 1 = 1 + (2^n) - 1 = (2^n)$

Por lo tanto, la complejidad es $\boldsymbol{O(2^n)}$.

- ## Problema 2

Antes, hagamos una demostración auxiliar:

Supongamos que $a \geq b$. PD $a\%b < \frac{a}{2}$

Caso 1: $b \leq \frac{a}{2}$
Entonces, $a\%b < b \leq \frac{a}{2}$, y así $a\%b < \frac{a}{2}$

Caso 2: $b > \frac{a}{2}$
Entonces, $a\%b = a - b < \frac{a}{2}$, y así $a\%b < \frac{a}{2}$

1. Decidir acerca del parámetro n que indica el tamaño de la entrada del algoritmo:
   Serán b y a.
2. Identificar la operación básica del algoritmo:
   Módulo de $M(a, b) = a\%b$.
3. Determinamos si el número de veces que la operación básica es ejecutada puede variar para diferentes entradas del mismo tamaño:
   No hay variación.
4. Expresar como una relación de recurrencia:
   Sea $M(a, b) = M(b, a\%b) + C$ para $b > 0$.
   $M(a, 0) = C$ una constante, en este caso 1, entonces

$$M(a,b) = \begin{cases} 1 \text{ si } b = 0 \\ M(b, a\%b) + C \text{ para } b > 1 \text{ y } a \geq b \\ M(b, a\%b) + 1 + C \text{ para } b > 1 \text{ y } b > a \end{cases}$$

5. Resolver la relación de recurrencia:
   probar
   Supongamos que $a > b > 0$. En caso contrario, solo se agrega una operación extra que cambia sus lugares.
   Veamos como se comporta la función:
   $M(a_0, b_0) = M(b_0, a_0\%b_0)$ con $b_0 = a_1$ y $a_0\%b_0 = b_1$
   $M(a_1, b_1) = M(b_1, a_1\%b_1)$ con $b_1 = a_2$ y $a_1\%b_1 = b_2$
   $M(a_2, b_2) = M(b_2, a_2\%b_2)$ con $b_2 = a_3$ y $a_2\%b_2 = b_3$

   Observemos que $b_{i+2} = a_{i+1}\%b_{i+1} < \frac{a_{i+1}}{2} = \frac{b_i}{2}$.
   Podemos notar que cada dos pasos partimos a b en al menos la mitad hasta llegar a que $b = 0$, entonces el máximo número de pasos que necesitamos son $2log_2(b) + C$ si $a \geq b$, pero

si $a < b$ necesitamos $2log_2(b) + 1 + C$, que sería lo mismo que arriba, pero con un paso extra de reacomodo.

Por lo tanto, la complejidad es $\boldsymbol{O(log_2(n))}$.

- **Problema 3**
  1. Decidir acerca del parámetro n que indica el tamaño de la entrada del algoritmo:
     n será la longitud.
  2. Identificar la operación básica del algoritmo:
     Sacar un carácter y concatenar, será $P(n)$.
  3. Determinamos si el número de veces que la operación básica es ejecutada puede variar para diferentes entradas del mismo tamaño n:
     No hay variación.
  4. Expresar como una relación de recurrencia:
     Sea $P(n) = C_1 + P(n-1)$ para $n > 0$.
     $P(0) = C_2$ una constante, en este caso 3, asumiendo que $chartArt()$ es constante, entonces

     $$P(n) = \begin{cases} C_2 = 3 \text{ si } n = 0 \\ C_1 + P(n-1) \text{ para } n > 0 \end{cases}$$

  5. Resolver la relación de recurrencia:
     probar
     - Para $n > 0$
       $P(n) = C + P(n-1)$
     - Para $n > 1$, sustituimos $P(n-1) = C + P(n-2)$
       $P(n) = C + (C + P(n-2)) = 2C + P(n-2)$
     - Para $n > 2$, sustituimos $P(n-2) = C + P(n-3)$
       $P(n) = 2C + (C + P(n-3)) = 3C + P(n-3)$
     - Podemos determinar un patrón: M
       $P(n) = iC + P(n-i)$
     - Tomando en cuenta la condición inicial, tenemos que se sustituye $i = n$ en la fórmula anterior y se obtiene:
       $P(n) = nC + P(n-n) = nC + P(0) = nC + 3$

  Por lo tanto, la complejidad es $\boldsymbol{O(n)}$.

- **Problema 4**
  1. Decidir acerca del parámetro n que indica el tamaño de la entrada del algoritmo:
     n será la longitud de la cadena.
  2. Identificar la operación básica del algoritmo:
     Revisar caracteres, será $S(n)$.
  3. Determinamos si el número de veces que la operación básica es ejecutada puede variar para diferentes entradas del mismo tamaño n:
     No hay variación.
  4. Expresar como una relación de recurrencia:
     Sea $S(n) = C + S(n-1)$ para $n > 0$
     $S(0) = C$ una constante, en este caso 1, entonces

     $$S(n) = \begin{cases} 1 \text{ si } n = 0 \\ C + S(n-1) \text{ para } n > 0 \end{cases}$$

  5. Resolver la relación de recurrencia:
     probar

- Para $n > 0$
  $S(n) = C + S(n-1)$
- Para $n > 1$, sustituimos $S(n-1) = C + S(n-2)$
  $S(n) = C + (C + S(n-2)) = 2C + S(n-2)$
- Para $n > 2$, sustituimos $S(n-2) = C + S(n-3)$
  $S(n) = 2C + (C + S(n-3)) = 3C + S(n-3)$
- Podemos determinar un patrón: M
  $S(n) = iC + S(n-i)$
- Tomando en cuenta la condición inicial, tenemos que se sustituye $i = n$ en la fórmula anterior y se obtiene:
  $S(n) = nC + S(n-n) = nC + S(0) = nC + 1$

Por lo tanto, la complejidad es $\boldsymbol{O(n)}$.