Análisis y desarrollo de Algoritmos (ADA)
Ejercicios adicionales

Fabrizio David Franco Amayo - 201710466

9 de octubre de 2020

1. Ejercicio - Método de sustitución

Se explicará el uso del método de sustitución para la expresión

$$T(n) = T(\lceil \tfrac{n}{2} \rceil) + T(\lfloor \tfrac{n}{2} \rfloor) + \Theta(n)$$

El método por sustitución se vale de una suposición para realizar una demostración por inducción. En este caso la suposición es brindada por la guía y es:

$$T(k) \leq c \cdot (k-2) lg(k-2)$$

La cual satisfacería que $T(n) = O(nlg(n))$

1.1. Manejo de Ceiling y Floor

Adicionalmente, se pide considerar la definición de *ceil* y f*floor* en el análisis asintótico, es importante recordar que una aproximación cercana podría realizarse considerando:

$$T(n) = T(\left\lceil \frac{n}{2} \right\rceil) + T(\left\lfloor \frac{n}{2} \right\rfloor)$$

$$T(n) \approx 2T(\frac{n}{2})$$

Sin embargo, la solución más precisa se obtiene al utilizar la definición de *ceil* y *floor* tal como se solicita.

$$T(n) = T(\left\lceil \frac{n}{2} \right\rceil) + T(\left\lfloor \frac{n}{2} \right\rfloor) + \Theta(n)$$

Por otra parte, lidiar con *ceil* y *floor* es sencillo, mas no trivial, por ello va a ser detallado como se trata con ellos a partir de sus deficiones formales:

$$\lceil n \rceil \geq n$$

$$\lfloor n \rfloor \leq n$$

Delimitando ambas inecuaciones por una unidad se obtiene:

$$n - 1 < \lfloor n \rfloor \leq n \leq \lceil n \rceil < n + 1$$

Por otra parte, se tienen las siguiente propiedades como menciona Donald Knuth en Concrete Mathematics:

$$\begin{array}{lcl} \lfloor x \rfloor = n & \iff & n \leqslant x < n+1\,, \\ \lfloor x \rfloor = n & \iff & x-1 < n \leqslant x\,, \\ \lceil x \rceil = n & \iff & n-1 < x \leqslant n\,, \\ \lceil x \rceil = n & \iff & x \leqslant n < x+1\,. \end{array}$$

Figura 1: Propiedades de ceiling y floor - C. Mathematics

Ya que el método de sustitución hace uso de inducción donde se comprueba para un n+1, esta es planteada a través de una inecuación, a partir de ello, es posible reemplazar *ceil* y *floor* en dicha inecuación haciendo uso de las propiedades citadas anteriormente que también tienen la forma de una inecuación, por ello la inecuación inicial no altere su comportamiento y los *ceil* y *floor* ya fueron resueltos, lo cual nos permite continuar con el resto de las operaciones. La estructura básica que se utilizará para la resolución es la siguiente:

$$T(n) \leq \lfloor n \rfloor$$

$$T(n) \leq n$$

Asimismo:

$$T(n) \leq \lceil n \rceil$$

$$T(n) \leq n + 1$$

1.2. Desarrollo

Reemplazando la suposición planteada

$$T(n) \leq c_1 \cdot (\left\lceil \frac{n}{2} \right\rceil - 2) lg(\left\lceil \frac{n}{2} \right\rceil - 2) + c_1 \cdot (\left\lfloor \frac{n}{2} \right\rfloor - 2) lg(\left\lfloor \frac{n}{2} \right\rfloor - 2) + \Theta(n)$$

Se requiere transformar el $\Theta(n)$, se hace de la siguiente manera apoyado en la definición formal:

Dada una función $g(n)$ se denota $\Theta(g(n))$ al set:

$\Theta(g(n))$ = { f(n) : tal que existen constantes positivas c_1, c_2 y n_0 que satisfacen $\leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ para $n \geq n_0$ } En este caso, $g(n) = n$ y

$f(n) = c \cdot n$. Por, lo tanto reemplazando en la ecuación:

$$T(n) \leq c_1 \cdot (\left\lceil \frac{n}{2} \right\rceil - 2) lg(\left\lceil \frac{n}{2} \right\rceil - 2) + c_1 \cdot (\left\lfloor \frac{n}{2} \right\rfloor - 2) lg(\left\lfloor \frac{n}{2} \right\rfloor - 2) + c_2 n$$

Resolviendo *floor* como $\lfloor n \rfloor \leq n + 1$ y *ceil* como $\lfloor n \rfloor \leq n$ resulta:

$$T(n) \leq c_1 \cdot (\frac{n}{2} + 1 - 2) lg(\frac{n}{2} + 1 - 2) + c_1 \cdot (\frac{n}{2} - 2) lg(\frac{n}{2} - 2) + c_2 n$$

$$T(n) \leq c_1 \cdot (\frac{n}{2} - 1) lg(\frac{n}{2} - 1) + c_1 \cdot (\frac{n}{2} - 2) lg(\frac{n}{2} - 2) + c_2 n$$

Es necesario dar la forma del asumption, por ello se remplaza los siguientes términos respetando la inecuación:

$\Rightarrow \frac{n}{2} - 2 \leq \frac{n}{2} - 1$

$\Rightarrow lg(\frac{n}{2} - 2) \leq lg(\frac{n}{2} - 1)$

En ambos casos se reemplazará por los términos del lado derecho para dar la forma que complete a la inducción. Resultando:

$$T(n) \leq c_1 \cdot (\frac{n}{2} - 1) lg(\frac{n}{2} - 1) + c_1 \cdot (\frac{n}{2} - 1) lg(\frac{n}{2} - 1) + c_2 n$$

$$T(n) \leq c_1 \cdot (\frac{n-2}{2}) lg(\frac{n-2}{2}) + c_1 \cdot (\frac{n-2}{2}) lg(\frac{n-2}{2}) + c_2 n$$

$$T(n) \leq 2 \cdot [c_1 \cdot (\frac{n-2}{2}) lg(\frac{n-2}{2})] + c_2 n$$

$$T(n) \leq [c_1 \cdot (n-2) lg(\frac{n-2}{2})] + c_2 n$$

$$T(n) \leq c_1 \cdot (n-2) \cdot [lg(n-2) - lg(2)] + c_2 n$$

$$T(n) \leq c_1 \cdot (n-2) \cdot [lg(n-2) - 1] + c_2 n$$

$$T(n) \leq c_1 \cdot (n-2) \cdot lg(n-2) - c_1 \cdot (n-2) \cdot + c_2 n$$

$$T(n) \leq c_1 \cdot (n-2) \cdot lg(n-2) - [c_1 \cdot (n-2) - c_2 n]$$

Para cerrar la inducción, la expresión entre paréntesis debe ser mayor o igual a 0:

$$0 \leq c_1 \cdot (n-2) - c_2 n$$

$$c_2 n \leq c_1 \cdot (n-2)$$

$$\frac{c_2 n}{n-2} \leq c_1$$

Se cierra la inducción con $\frac{c_2 n}{n-2} \leq c_1$ y $n > 2$. Por lo tanto queda demostrado:

$$T(n) = O(nlg(n))$$

2. Ejercicio - Sumatoria

2.1. Análisis previo - Quicksort

Se muestra una implementación en C++ elaborada por mí del Quicksort, en base a ella se realizará el análisis:

```
1    void quicksort( int primer_id, int ultimo_id){
2
3        if (primer_id < ultimo_id){
4            int pivote = ordenar_particion(primer_id, ultimo_id);
5            quicksort(primer_id,pivote - 1);
6            quicksort( pivote + 1, ultimo_id);
7        }
8    }
```

Listing 1: Ordenar Particion de QuickSort - Recursivo

```
1   int ordenar_particion(int primer_id,int pivote_id){
2
3        int pivote = elements[generateRandomBetween(primer_id,
     pivote_id)];
4        int actual_id = primer_id;
5        for (int comparador = primer_id; comparador <pivote_id;
     comparador++){
6
7            if (elements[comparador] <pivote){
8                swap(&elements[actual_id], &elements[comparador]);
9                actual_id++;
10           }
11       }
12       swap(&elements[actual_id], &elements[pivote_id]);
13       return actual_id;
14   }
```

Listing 2: Implementación de QuickSort - Recursivo

Este algoritmo va a funcionar mejor o peor dependiendo de que tan ordenado se encuentre el array antes de realizar el ordenamiento. El caso promedio es cuando parcialmente ordenado. Considerando que a P(n) como el tiempo requerido para ordenar_particion se tiene $P(n) = \Theta(n)$. El algoritmo se llama a sí mismo de forma recursiva con subarrays que tienen como patrón de tamaño pivote y (n-pivote). En esta versión del programa, el pivote es aleatorizado, todos elementos tienen la misma capacidad de ser seleccionados como pivote con probabilidad $\frac{1}{n}$, por ende el tiempo de ejecución puede se expresado como:

$$T(n) = \sum_{q=1}^{n-1}(T(q) + T(n-q)) \cdot \frac{1}{n} + \Theta(n)$$

Reasignando variable k por q, dado que T(k) incurre en la sumatoria una vez como $T(q)$ y otra como $T(n-q)$, resultando:

$$T(n) = \sum_{k=1}^{n-1} T(k) \cdot \frac{2}{n} + \Theta(n)$$

Usando el método de sustitución con el siguiente asumption:

$$T(n) \leq a \cdot n \cdot lg(n) + b$$

Reemplazando:

$$T(n) \leq \sum_{k=1}^{n-1}(a \cdot k \cdot lg(k) + b) \cdot \frac{2}{n} + \Theta(n)$$

$$T(n) = \sum_{k=1}^{n-1}(klg(k)) \cdot \frac{2a}{n} + \frac{2b}{n} \cdot (n-1) + \Theta(n)$$

Como guess se tiene que la sumatoria esta superiormente acotada por la siguiente expresión:

$$\sum_{k=1}^{n-1} k \cdot lg(k) \leq \frac{1}{2}n^2 lg(n) - \frac{1}{8}n^2$$

Dicho guess, produce lo siguiente en la sustitución original:

$$T(n) \leq (\frac{1}{2}n^2 lg(n) - \frac{1}{8}n^2) \cdot \frac{2a}{n} + \frac{2b}{n} \cdot (n-1) + \Theta(n)$$

$$T(n) \leq a \cdot nlg(n) - \frac{a \cdot n}{4} + 2b + \Theta(n)$$

$$T(n) \leq a \cdot nlg(n) - \frac{a \cdot n}{4} + b - [\frac{a \cdot n}{4} - b - dn]$$

$$T(n) \leq a \cdot n \cdot lg(n) + b$$

Se cierra la inducción con:

$$\frac{a \cdot n}{4} - b - dn \geq 0$$

$$n(\frac{a}{4} - d) - b \geq 0$$

$$a > 4d$$

Por ende, faltaría probar que la sumatoria efectivamente está acotada, por el bound propuesto y dicho procedimiento se detalla a continuación.

2.2. Desarrollo

$$\sum_{k=1}^{n-1} k \cdot lg(k) \leq \frac{1}{2} n^2 lg(n) - \frac{1}{8} n^2$$

Primero resulta conveniente expandir la sumatoria del lado izquierdo:

$$\sum_{k=1}^{n-1} k \cdot lg(k) = \sum_{k=1}^{\lceil \frac{n}{2} \rceil - 1} k \cdot lg(k) + \sum_{k=\lceil \frac{n}{2} \rceil}^{n-1} k \cdot lg(k)$$

Ya que se busca probar un upper bound, para cada nueva sumatoria se puede obtener un upper bound, extrayendo el lg(k) de la sumatoria, y multiplicando dicha sumatoria por el logaritmo más cercano al límite alcanzado por k en cada sumatoria, resultando lo siguiente:

1. Para la primera sumatoria, un upper bound (el más cercano) estaría en lg($\frac{n}{2}$). Por propiedades de logaritmos se puede expresar como lg(n) - lg(2), es decir, lg(n) -1.

2. Para la segunda sumatoria, el bound con las mismas propiedades que para la anterior sumatoria sería con lg(n)

Al aplicar los upper bounds, se obtiene:

$$\sum_{k=1}^{n-1} k \cdot lg(k) \leq (lg(n) - 1) \cdot \sum_{k=1}^{\lceil \frac{n}{2} \rceil - 1} k + lg(n) \cdot \sum_{k=\lceil \frac{n}{2} \rceil}^{n-1} k$$

Al operar se obtiene:

$$\sum_{k=1}^{n-1} k \cdot lg(k) \leq lg(n) \cdot \sum_{k=1}^{\lceil \frac{n}{2} \rceil - 1} k - \sum_{k=1}^{\lceil \frac{n}{2} \rceil - 1} k + lg(n) \cdot \sum_{k=\lceil \frac{n}{2} \rceil}^{n-1} k$$

Al agrupar la primera y tercera sumatoria del lado derecho:

$$\sum_{k=1}^{n-1} k \cdot lg(k) \leq lg(n) \cdot \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil \frac{n}{2} \rceil - 1} k$$

Resolviendo las sumatorias con la forma:

$$\sum_{k=1}^{n} k = \frac{n \cdot (n+1)}{2}$$

Se obtiene:

1. Para la primera sumatoria:

$$lg(n) \cdot \sum_{k=1}^{n} k = lg(n) \cdot (\frac{n \cdot (n-1)}{2})$$

2. Para la segunda sumatoria:

$$- \sum_{k=1}^{\lceil \frac{n}{2} \rceil - 1} k = -\frac{(\frac{n}{2} - 1) \cdot \frac{n}{2}}{2}$$

Reagrupando términos se obtiene:

$$\sum_{k=1}^{n-1} k \cdot lg(k) \leq lg(n) \cdot (\frac{n \cdot (n-1)}{2}) - \frac{(\frac{n}{2} - 1) \cdot \frac{n}{2}}{2}$$

Al operar se obtiene:

$$\sum_{k=1}^{n-1} k \cdot lg(k) \leq lg(n) \cdot \frac{n^2 - n}{2} - \frac{\frac{n^2}{4} - \frac{n}{2}}{2}$$

$$\sum_{k=1}^{n-1} k \cdot lg(k) \leq lg(n) \cdot \frac{n^2 - n}{2} - \frac{\frac{n^2}{4} - \frac{n}{2}}{2}$$

$$\sum_{k=1}^{n-1} k \cdot lg(k) \leq lg(n) \cdot \frac{n^2}{2} - \frac{n \cdot lg(n)}{2} - \frac{n^2}{8} + \frac{n}{4}$$

Dando la forma de la expresión original.

$$\sum_{k=1}^{n-1} k \cdot lg(k) \leq \cdot \frac{1}{2} n^2 lg(n) - \frac{1}{8} n^2 - [\frac{n \cdot lg(n)}{2} - \frac{n}{4}]$$

La expresión entre corchetes es mayor o igual a 0 para $n \geq 0$. En consecuencia queda demostrado que:

$$\sum_{k=1}^{n-1} k \cdot lg(k) \leq \frac{1}{2} n^2 lg(n) - \frac{1}{8} n^2$$

3. Hanoi - Correctitud

El problema de Hanoi es un histórico problema matemático. Este *puzzle* cuenta con 3 bases y n discos colocados en una de las bases de forma descendente. El objetivo a completarse es trasladar la torre de discos en el mismo orden a una base diferente. Como restricciones, un disco no puede ser colocado sobre otro disco más pequeño y solo puede ser movido un disco a la vez. Este problema matemático es un referente clásico de los problemas de recursividad.

Para resolver este problema, el mismo puede ser expresado de la siguiente manera: Si se desea resolver el problema de la torre de Hanoi para *n discos*, primero es necesario mover *n-1 discos* a la base intermedia (aquella que no es la de destino ni la de origen), se mueve el disco de mayor a la base de destino. Ahora se tiene un torre con *n -1* , dicha torre tiene como origen la base que antes era la intermedia; por otro lado, la base de origen anterior se encuentra libre, por lo que ahora esa será la nueva base intermedia para mover la torre de *n-1 discos* siguiendo el mismo razonamiento recursivo que acaba de ser explicado.
La correctitud se establece sobre una solucion(algoritmo), no sobre un problema, por lo que el approach explicado anteriormente será transformado a un pseudocódigo para posteriormente hacer la prueba de correctitud. (Este algoritmo fue desarrollado por mí en la Práctica Calificada 1:

```
1 Hanoi(cantidad, origen,parada,destino):
2     if(cantidad==0):
3         No se realizan movimientos
4     else:
5         Hanoi(cantidad-1, origen, destino, parada)
6         Llevar el disco de origen a destino sin parada
7         Hanoi(cantidad-1,parada,origen,destino)
```

Listing 3: Algoritmo Hanoi - Recursivo

En relación a la correctitud del algoritmo anterior, el caso base ocurre cuando no hay discos, en dicha situación es necesario realizar ningún movimiento de discos entre las bases. Cuando no ocurre el caso base, se aplica el algoritmo de Hanoi de forma recursiva con *n-1 discos* teniendo como destino la base intermedia, es decir, usando como parada el destino final de *n discos* (Nótese que

esto es no es problema porque dicha base se encuentra sin un disco menor encima y en los siguientes pasos se seguirá moviendo discos). En esta situación, se tiene el disco mayor libre para ser movido a la torre de destino sin romper la regla de orden descendente. Ahora los *n-1 discos* ubicados en la torre de parada son movilizados a la torre de destino aplicando Hanoi a los *n-1 discos*, en este caso, será la base que inicialmente era origen la que será usada como parada.
Hasta el momento la recursión tiene la forma:

$$Hanoi(n) \leq 2 \cdot Hanoi(n-1) + 1$$

Es decir, se sabe que el problema puede ser resuelto en como máximo $2 \cdot Hanoi(n-1) + 1$. Sin embargo, aún no se ha probado que este algoritmo resuelve el problema en la mínima cantidad de movimientos. Analizando el problema, se puede buscar alguna forma mejor de resolverlo, pero no existe dicha solución mejor. Esto debido a que en algún punto se tiene que mover el disco mayor y en dicho momento, los otros *n-1 discos* ya se encuentran en la base de parada, para ello se necesitaron al menos Hanoi(n-1) movimientos. Ahora nuevamente se deben mover los *n-1 discos* más pequeños sobre el disco mayor, dicho proceso también requiere Hanoi(n-1) movimientos. Lo cual resulta en:

$$Hanoi(n) \geq 2 \cdot Hanoi(n-1) + 1$$

Al juntar ambas inecuaciones con el caso base de este algoritmo se obtiene:

$$Hanoi(0) = 0$$

Para $n > 0$ se tiene:

$$Hanoi(n) = 2 \cdot Hanoi(n-1) + 1$$

En este punto, ya se ha probado la correctitud de este algoritmo como un método de solución recursivo para el problema de las torres de Hanoi que trabaja realizando la mínima cantidad de movimientos.

Referencias

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. (2009). Introduction to Algorithms, Third Edition. The MIT Press.

[2] Cull, P., & Ecklund Jr., E. F. (1985). Towers of Hanoi and Analysis of Algorithms. The American Mathematical Monthly, 407–420. https://doi.org/10.2307/2322448

[3] Donald Knuth, Oren Patashnik y Ronald Graham. (1994).Chapter 1: Recurrent problems. Concrete Mathematics. Addison-Wesley.