Instituto Politécnico Nacional
Escuela Superior de Cómputo
Análisis de Algoritmos

Tarea 05

Analisis de algoritmos recursivos

Alumno:

López Manríquez Ángel (2017630941)

Grupo: 3CM3

M. en C. Edgardo Adrián Franco Martínez

Fecha: 22 de junio de 2019

Índice

1. Función recursiva

Calcular la complejidad para el siguiente algoritmo recursivo:

```
int FuncionRecursiva(int num) {
    if (num == 0) return 1;
    else if (num < 2) {
      int resultado = 0;
      for (int i = 0; i < num * num; i++)
         resultado *= num;
      return resultado;
    }
    return FuncionRecursiva( num - 1 ) * FuncionRecursiva(num - 2);
}
```

Como se nos pide la complejidad O para los algoritmos no nos preocuparemos tanto por las operaciones, sino que consideraremos a todas estas con un costo de 1, asi, tenemos que para $n = 0$ la operacion cuesta 1, si $n = 1$ tambien pues se hace un for de una sola iteracion, finalmente si $n > 2$ tenemos la llamada de la funcion recursiva, por tanto la función complejidad $T(n)$ queda como:

$$T(n) = \begin{cases} 1 & \text{si } n = 0, 1 \\ T(n-1) + T(n-2) & \text{si } n >= 2 \end{cases}$$

Se trata de una recurrencia lineal homogénea.

Resolviendo la recurrencia homogénea

La función $T_h(n)$ cumple que $T_h(n) = T_h(n-1)+T_h(n-2)$. Haciendo el cambio $T_h(n) = x^n$ obtenemos $x^n = x^{n-1} + x^{n-2}$, simplificamos y obtenemos la ecuación característica de la recurrencia: $x^2 - x - 1 = 0$ cuyas raices son $x_1 = \frac{1+\sqrt{5}}{2}, x_2 = \frac{1-\sqrt{5}}{2}$ dando lugar a

$$T_n = c_1\Big(\frac{1+\sqrt{5}}{2}\Big)^n + c_2\Big(\frac{1-\sqrt{5}}{2}\Big)^n \in O\Big(\Big(\frac{1+\sqrt{5}}{2}\Big)^n\Big)$$

2. Producto

```
int Producto(int a, int b) {
    if (b == 0) return 0;
    return a + Producto(a, b - 1);
}
```

Para este algoritmo no hay necesidad alguna de usar ecuaciones de recurrencia pues, como su nombre lo indica, la funcion da, en efecto, el producto ab ya que hacemos:

$$\sum_{k=1}^{b} a = ab$$

donde la complejidad es determinada por b dando lugar a una complejidad lineal $O(n)$.

3. Recorrido en inorden de un arbol

```
void inorderTraversal(TreeNode root)
{
    if(root != nullptr)
    {
        inorderTraversal(root.getLeft());
        process(root.getValue());
        inorderTraversal(root.getRight());
    }
}
```

Una vez mas, la obtencion de la funcion complejidad es bastante trivial pues para hacer un recorrido en inorden tenemos que pasar por el nodo del arbol una sola vez por lo que su complejidad es $O(n)$ donde n representa el numero de nodos del arbol.

4. Busqueda ternaria

```
#include <math.h>

// f stands for a function pointer. This function returns the value x such that when it
// is evaluated at the unimodal function f produces a maximum between l and r where l < r
double ternarySearch(double (*f)(double) , int l , int r, double absolutePrecision) {
    if (abs(r - l) <= absolutePrecision) return ( l + r ) / 2.0;
    int mLeft = ( 2 * l + r ) / 3;
    int mRight = ( l + 2 * r ) / 3;
    return ( f(mLeft) < f(mRight) ) ?
        ternarySearch(f, mLeft, r, absolutePrecision) :
        ternarySearch (f, l, mRight, absolutePrecision);
}
```

Los analisis de estos dos ultimos algoritmos, a diferencia de los anteriores, ya no son tan obvios, aqui, tenemos que analizar un algoritmos en el cual, como dice la descripcion, regresa un x_0 de una funcion unimodal (es decir, que solo tiene un unico maximo) tal que $f(x_0) = max\{f\}$ para $l < x < r$ donde $x \in \mathbf{R}$, con una precision variable (un error definido). Para analizar este algoritmo, sea $n = r - l$ la dimension del problema a definir, vemos que el peor caso se da cuando `absolutePrecision` $\to 0$, es decir, cuando $n \leq 0$. Tenemos que, para cualquier valor de n tenemos un costo asociado a la funcion, supongamos que tal costo es 1. Si no se ejecuta el caso base vemos que la dimension del problema se reduce a $2n/3$ por lo que nuestra funcion T_n se define como:

$T(n) = T(2n/3) + 1 = \Theta(\log n)$ para $n > 0$, haciendo uso del teorema maestro

5. Quicksort

```
/** This function will put all the elements less than v[r] to the left of this
 * and the remaininig elements to the right.
 * @return position of the middle element which we used to compare. */
template<typename t>
int rearrangeVector(std::vector<t> v, int l, int r) {
    t pivot = v[r];
    int pivotIndex = l;
    for (int i = l; i < r; ++i) {
        if (v[i] <= pivot) {
            swap(v[i], a[pivotIndex]);
            ++pivotIndex;
        }
    }
    swap(v[pivotIndex], v[r]);
    return pivotIndex;
}

template<typename t>
void quicksort(std::vector<t> v, int l, int r) {
    if (l < r) {
        int pivotIndex = rearrangeVector(v, l, r);
        quicksort<t> (v, l, pivotIndex - 1);
        quicksort<t> (v, pivotIndex + 1, r);
    }
}
```

La implementacion de el algoritmo fue reescrito de manera distinta a como esta en la presentación con el fin de ser mas claro con la descripción de la función dado que es mas facil ver la complejidades de esta manera, asi, vemos que la complejidad de la subrutina de `rearrangeVector` es $O(n)$ pues el bucle increment la variable `i` en una unidad.

Dividimos el problema en dos partes ambas no iguales, pero para el analisis supongamos que lo son, por tanto tenemos que nuestra funcion complejidad es:

$$T(n) = \begin{cases} 1 & \text{si } n = 1 \\ 2T(n/2) + n & \text{en otro caso} \end{cases}$$

Por esta ocasion, no podemos hacer uso del teorema maestro, pero podemos hacer uso de el teorema de Akra bazzi, hablemos de este teorema.

5.1. Teorema de Akra bazzi

El metodo de Akra-Bazzi aplica a las formulas de recurrencia de la forma:

$$T(x) = g(x) + \sum_{i=1}^{k} a_i T(b_i x + h_i(x)) \qquad \text{para } x \geq x_0.$$

Las condiciones para su uso son:

- suficientes casos bases dados
- a_i y b_i son constantes para todo i
- $0 < b_i < 1 \ \forall i$
- $|g(x)| \in O(x^c)$ donde c es constante.
- $|h_i(x)| \in O(x \cdot \log^{-2}(x)) \ \forall i$
- x_0 es constante.

El comportamiento asintotico de $T(x)$ es encontrado al determinar p tal que $\sum_{i=1}^{k} a_i b_i^p = 1$ y poner tal valor en la ecuacion:

$$T(x) \in \Theta\left(x^p \left(1 + \int_1^x \frac{g(u)}{u^{p+1}} du\right)\right) \tag{1}$$

Usando este teorema para el quicksort tenemos $a_1 = 2, b_1 = 1/2$ y $f(n) = n$, hallando p tenemos $2(1/2)^p = 1 \implies p = 1$ y

$$\begin{aligned} T(n) &= \Theta\left(n^p + n^p \int_1^n \frac{f(t)}{t^{p+1}} dt\right) \\ &= \Theta\left(n\Big(1 + \int_1^n \frac{t}{t^{1+1}} dt\Big)\right) \\ &= \Theta\left(n\Big(1 + \ln n\Big)\right) \\ &= \Theta(n \ln n) \end{aligned}$$

Sin embargo, esto se cumple cuando los valores del vector estan desordenados, curiosamente, a diferencia de otros metodos de ordenamiento el peor caso se da cuando el vector esta ordenado pues la variable `pivorIndex` es decrementada solo en una unidad dando lugar a

$$T(n) = \begin{cases} 1 & \text{si } n = 1 \\ T(n-1) + n & \text{en otro caso} \end{cases}$$

Al resolver esta recurrencia tenemos $\Theta(n^2)$ para el peor caso. Este caso puede ser evadido si tomamos un pivote de manera aleatoria y lo intercambiamos en la ultima posicion del "subvector"llamado recursivamente.

6. Solución de modelos recurrentes

Resolver las siguientes ecuaciones y dar su orden de complejidad:

6.1. Recurrencia 1

$$T(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ 3T(n-1) + 4T(n-2) & \text{si } n \geq 2 \end{cases}$$

Tenemos una recurrencia lineal homogénea. Haciendo el cambio $T(n) = x^n$, obtenemos la ecuación característica: $x^2 - 3x - 4 = 0$ cuyas raíces son $x_1 = 4$ y $x_2 = -1$. por tanto:

$$T(n) = C_1 4^n + C_2(-1)^n$$

Usando las condiciones iniciales:

$$T(0) = 0 \implies C_1 + C_2 = 0$$
$$T(1) = 1 \implies 4C_1 - C_2 = 1$$

Dando lugar a $C_1 = \frac{1}{5}, C_2 = -\frac{1}{5}$). Asi:

$$T(n) = \frac{1}{5}\left(4^n - (-1)^n\right)$$
$$T(n) \in O(4^n)$$

6.2. Recurrencia 2

$$t(n) = \begin{cases} 5 & \text{si } n = 0 \\ 27 & \text{si } n = 1 \\ 3t(n-1) + 4t(n-2) + (n+5)2^n & \text{si } n \geq 2 \end{cases}$$

tenemos una recurrencia lineal no homogénea.

resolviendo la parte homogénea

la ecuación característica de la parte homogénea es $x^2 - 3x - 4 = 0$, y tiene como raíces a $x_1 = 4$ y $x_2 = -1$. la solución a la parte homogénea es entonces:

$$T_h(n) = c_1(4)^n + c_2(-1)^n$$

resolviendo la parte no homogénea

propongamos $T_p(n) = (an + b)2^n$. sustituimos en la recurrencia original y obtenemos:

$$\begin{aligned}
(an + b)2^n &= 3(a(n-1) + b)2^{n-1} + 4(a(n-2) + b)2^{n-2} + (n + 5)2^n \\
4(an + b)2^n &= 6(a(n-1) + b)2^n + 4(a(n-2) + b)2^n + 4(n + 5)2^n \\
4an + 4b &= 6an - 6a + 6b + 4an - 8a + 4b + 4n + 20 \\
4an + 4b &= (10a + 4)n - 14a + 10b + 20
\end{aligned}$$

comparamos coeficientes y obtenemos el sistema:

$$\begin{aligned}
4a = 10a + 4 &\implies a = -\frac{2}{3} \\
4b = -14a + 10b + 20 &\implies b = -\frac{44}{9}
\end{aligned}$$

por lo tanto, $T_p(n) = \left(-\frac{2}{3}n - \frac{44}{9}\right)2^n$

solución final

la recurrencia es la suma de las dos soluciones anteriores:

$$T(n) = c_1(4)^n + c_2(-1)^n + \left(-\frac{2}{3}n - \frac{44}{9}\right)2^n$$

considerando las condiciones iniciales, obtenemos el sistema:

$$\begin{aligned}
t(0) = 5 &\implies c_1 + c_2 - \frac{44}{9} = 5 \\
t(1) = 27 &\implies 4c_1 - c_2 - \frac{100}{9} = 27
\end{aligned}$$

cuya solución es $(c_1, c_2) = (\frac{48}{5}, \frac{13}{45})$. por lo tanto, la solución final es:

$$\begin{aligned}
T(n) &= \frac{48}{5}(4)^n + \frac{13}{45}(-1)^n + \left(-\frac{2}{3}n - \frac{44}{9}\right)2^n \\
T(n) &\in O(4^n)
\end{aligned}$$

6.3. Recurrencia 3

$$T(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ 2T(n-1) + 3^n & \text{si } n \geq 2 \end{cases}$$

Tenemos una recurrencia lineal no homogénea.

Resolviendo la parte homogénea

La ecuación característica de la parte homogénea es $x - 2 = 0$, cuya única raíz es $x_1 = 2$. Entonces:

$$T_h(n) = C_1(2)^n$$

Resolviendo la parte no homogénea

Para la parte no homogénea propongamos $T_p(n) = A(3)^n$. Sustituimos en la recurrencia original:

$$\begin{aligned} A(3)^n &= 2A(3)^{n-1} + 3^n \\ 3A(3)^n &= 2A(3)^n + 3(3)^n \\ 3A &= 2A + 3 \\ A &= 3 \end{aligned}$$

Por lo tanto, la solución no homogénea es $T_p(n) = 3(3)^n$.

Solución final

La recurrencia original es la suma de las dos soluciones anteriores:

$$T(n) = C_1(2)^n + 3(3)^n$$

Pero como tenemos la condición inicial $T(0) = 0$, se tiene que cumplir que $C_1 + 3 = 0$, es decir, $C_1 = -3$. Por lo tanto:

$$\begin{aligned} T(n) &= 3(3^n - 2^n) \\ T(n) &\in O(3^n) \end{aligned}$$

7. Cotas de algoritmos recurrentes

Calcular la cota de complejidad que tendrían los algoritmos con los siguientes modelos recurrentes.

7.1. Modelo 1

$$T(n) = 3T(n/3) + 4T(n/2) + 2n^2 + n$$

Por el teorema de Akra-Bazzi, tenemos que $a_1 = 3$, $b_1 = \frac{1}{3}$, $a_2 = 4$, $b_2 = \frac{1}{2}$ y $f(n) = 2n^2 + n \in O(n^2)$. Hay que hallar el valor de p tal que:

$$3\left(\frac{1}{3}\right)^p + 4\left(\frac{1}{2}\right)^p = 1$$

Donde $p \approx 2.3646$, entonces la solución de la recurrencia es:

$$\begin{aligned}
T(n) &\in \Theta\left(n^p + n^p \int_1^n \frac{f(t)}{t^{p+1}} dt\right) \\
&= \Theta\left(n^p + n^p \int_1^n \frac{t^2}{t^{p+1}} dt\right) \\
&= \Theta\left(n^p + n^p \int_1^n t^{1-p} dt\right) \\
&= \Theta\left(n^p + n^p \cdot \frac{n^{2-p} - 1}{2 - p}\right) \\
&= \Theta\left(n^p + \frac{n^2 - n^p}{2 - p}\right) \\
&= \Theta\left(\frac{(p-1)n^p - n^2}{p - 2}\right) \\
&= \Theta(n^{2.3646})
\end{aligned}$$

7.2. Modelo 2

$$T(n) = T(n-1) + T(n-2) + T(n/2)$$

El término $T(n/2)$ solo le agrega una complejidad de $O(\log n)$ a la recurrencia, por lo tanto, podemos ignorarlo pues no tenemos un modelo para este problems por lo que solo nos preocupamos por resolver $T(n) = T(n-1) + T(n-2)$.

La ecuación característica es $x^2 - x - 1 = 0$, cuyas raíces son:

$$x_1 = \frac{1+\sqrt{5}}{2}$$
$$x_2 = \frac{1-\sqrt{5}}{2}$$

Por lo tanto:

$$T(n) = C_1\Big(\frac{1+\sqrt{5}}{2}\Big)^n + C_2\Big(\frac{1-\sqrt{5}}{2}\Big)^n$$

Y si $C_1 \neq 0$:

$$T(n) \in \Theta\Big(\Big(\frac{1+\sqrt{5}}{2}\Big)^n\Big)$$

7.3. Modelo 3

$$T(n) = T(n/2) + 2T(n/4) + 2$$

Por el teorema de Akra-Bazzi, tenemos que $a_1 = 1$, $b_1 = \frac{1}{2}$, $a_2 = 2$, $b_2 = \frac{1}{4}$ y $f(n) = 2 \in O(1)$. Hay que hallar el valor de p tal que:

$$1\left(\frac{1}{2}\right)^p + 2\left(\frac{1}{4}\right)^p = 1$$

La solución es $p = 1$. Por lo tanto, la solución de la recurrencia es:

$$\begin{aligned}
T(n) &\in \Theta\left(n^p + n^p \int_1^n \frac{f(t)}{t^{p+1}} dt\right) \\
&= \Theta\left(n^p + n^p \int_1^n \frac{1}{t^{p+1}} dt\right) \\
&= \Theta\left(n^p + n^p \int_1^n t^{-p-1} dt\right) \\
&= \Theta\left(n^p + n^p \cdot \frac{n^{-p} - 1}{-p}\right) \\
&= \Theta\left(n^p - \frac{1 - n^p}{p}\right) \\
&= \Theta\left(\frac{(p+1)n^p - 1}{p}\right) \\
&= \Theta(2n - 1) \\
&= \Theta(n)
\end{aligned}$$

7.4. Modelo 4

$$T(n) = 2T(n/2) + 4T(n/4) + 10n^2 + 5n$$

Por el teorema de Akra-Bazzi, tenemos $a_1 = 2$, $b_1 = \frac{1}{2}$, $a_2 = 4$, $b_2 = \frac{1}{4}$ y $f(n) = 10n^2 + 5n \in O(n^2)$. Hay que hallar el valor de p tal que:

$$2\left(\frac{1}{2}\right)^p + 4\left(\frac{1}{4}\right)^p = 1$$

Donde $p = 2 - \lg(-1 \pm \sqrt{5})$, tomando una de las soluciones tenemos $p \approx 1.69424$. Por tanto, la solución es:

$$\begin{aligned} T(n) &\in \Theta\left(n^p + n^p \int_1^n \frac{f(t)}{t^{p+1}} dt\right) \\ &= \Theta\left(n^p + n^p \int_1^n \frac{t^2}{t^{p+1}} dt\right) \\ &= \Theta\Big(\frac{1}{2-p}\Big((1-p)n^p + n^2\Big)\Big) \\ &= \Theta(n^2) \text{ Pues } p < 2 \end{aligned}$$