Análisis de Algoritmos 2023

Trabajo práctico obligatorio - Recurrencias

Ignacio Alejandro Navarro Oliva
FAI-3240

Davor Kissner
FAI-3185

5 de diciembre de 2023

1 Introducción

Las recurrencias desempeñan un papel crucial en el análisis de algoritmos y la teoría de la computación, proporcionando un marco para describir la complejidad temporal de algoritmos recursivos. En este informe, exploraremos métodos formales para la resolución de recurrencias, centrándonos tanto en recurrencias de una variable como en aquellas que involucran dos variables. Los métodos que se mostrarán a lo largo de este informe son:

- Sustitución
- Árbol de recursión
- Funciones generadoras

2 Sustitución

El método de sustitución es un proceso algebráico para resolver recurrencias que está intimamente ligado con el comportamiento del algoritmo. Consiste en desarrollar los llamados recursivos y analizar su comportamiento, para luego resolver la recurrencia utilizando los términos iniciales de la recursión. Este método no es solo útil por su sencillez, sino porque también facilita la compresión del funcionamiento del algoritmo recursivo.

Resumen del proceso

Los pasos clave y la idea detrás del método de sustitucion para resolver recurrencias son:

Paso 1 Expresar la fórmula de la recurrencia a resolver.

Paso 2 Desarrollar los términos de la recurrencia.

Paso 3 Reemplazar el término anterior paso a paso de forma reiterada, analizar el comportamiento de la recurrencia y generalizarlo para cualquier paso K. Es probable que se deba utilizar alguna propiedad matemática para poder generalizar este aspecto.

Paso 4 Aplicar el caso base, reemplazando K en función de n según el caso base, y así salir de la recursividad y encontrar una solución a la recurrencia en términos de n.

2.1 Ejemplos

A continuación se detallará la aplicación del método de sustitución para recurrencias no homogéneas y de tipo divide y vencerás.

2.1.1 Ejemplo 1

Consideremos la recurrencia no homogénea $t_n = t_{n-1} + n$, con $t_0 = 1$ y su respectivo código realizado en Java:

$t_n = t_{n-1} + n$, con $t_0 = 1$

```
public static int fun(int n){
    int r;
    if (n == 0) {
        r = 1;                          //n = 0
    } else {
        r = fun(n-1) + sumatoria(n); //n > 0
    }
    return r;
}
```

Primero se desarrollan los primeros t_n de la recursión:

$t_{n-1} = t_{n-2} + n - 1$
$t_{n-2} = t_{n-3} + n - 2$
$t_{n-3} = t_{n-4} + n - 3$

Reemplazamos en t_n:

$t_n = [t_{n-2} + n - 1] + n$
$t_n = [t_{n-3} + n - 2] + (n-1) + n$
$\vdots$
$t_n = t_{n-k} + (n - (k-1)) + (n - (k-2)) + \ldots + (n-1) + n$

Sabemos que en algún punto de los llamados se llegará al caso base, por lo que tomamos $n - k = 0$, lo cual nos da que $n = k$. Reemplazamos en la fórmula desarrollada:

$t_n = t_{n-n} + (n - (n-1)) + (n - (n-2)) + \ldots + (n-1) + n$
$t_n = t_0 + (n - n + 1) + (n - n + 2) + \ldots + (n-1) + n$
$t_n = 1 + 1 + 2 + 3 + \ldots + (n-1) + n$

Reexpresando la sumatoria obtenemos la siguiente solución a la recurrencia:

$$t_n = 1 + \frac{n \cdot (n+1)}{2} \Rightarrow t_n \in O(n^2)$$

2.1.2 Ejemplo 2

Consideremos la siguiente recurrencia de tipo divide y vencerás $t_n = t_{n/2} + 1$, con $t_1 = 1$ y su respectivo código Java:

$t_n = t_{n/2} + 1$, con $t_1 = 1$

```
public static int fun2(int n) {
    int r;
    if (n == 1) {
        r = 1;
    } else {
        r = fun2(n / 2) + 1;
    }
    return r;
}
```

Primero se desarrollan los primeros términos de la recursión:

$t_{n/2} = t_{n/2^2} + 1$
$t_{n/2^2} = t_{n/2^3} + 1$

Reemplazamos en t_n:

$t_n = [t_{n/2^2} + 1] + 1 = t_{n/2^2} + 2$
$t_n = [t_{n/2^3} + 1] + 2 = t_{n/2^3} + 3$

$\vdots$

$t_n = t_{n/2^k} + k$

Sabemos que en algún punto las sucesivas divisiones permitirán alcanzar el caso base, por lo que tomamos $n/2^k = 1$, lo cual nos da que $n = 2^k$ y $k = log_2 n$. Reemplazamos en la fórmula desarrollada:

$t_n = t_{n/2^{log_2 n}} + log_2 n$
$t_n = t_{n/n^{log_2 2}} + log_2 n$
$t_n = t_{n/n} + log_2 n$
$t_n = t_1 + log_2 n = 1 + log_2 n$

$\therefore t_n = log_2 n + 1, t_n \in O(log n)$.

3 Árbol de Recursión

Un árbol de recursividad es una representación visual que muestra cómo se ejecuta una función recursiva. Ofrece una visión detallada de las llamadas recursivas, ilustrando cómo el algoritmo se ramifica hasta llegar a un caso base. Esta estructura facilita el análisis de la complejidad temporal y permite entender el proceso recursivo que se lleva a cabo.

Esctructura del árbol

Cada nodo en un árbol de recursividad simboliza una llamada recursiva específica. La llamada inicial se ubica en la cima, y las llamadas subsiguientes se desprenden debajo de ella. El árbol se expande hacia abajo, creando una estructura de jerarquía. El grado de ramificación de cada nodo se determina por el número de llamadas recursivas hechas dentro de la función. Además, la profundidad del árbol se relaciona con la cantidad de llamadas recursivas antes de alcanzar el caso base.

Caso base

El caso base actúa como el punto de finalización para una función recursiva. Establece el momento en el que se detiene la recursión y la función comienza a devolver valores. En un árbol de recursividad, los nodos que representan el caso base generalmente se muestran como nodos hoja, ya que no generan más llamadas recursivas.

Llamadas recursivas

Los nodos hijos en un árbol de recursividad simbolizan las llamadas recursivas que se realizan dentro de la función. Cada nodo hijo corresponde a una llamada recursiva distinta, lo que conduce a la creación de nuevos subproblemas. Los valores o parámetros que se pasan a estas llamadas recursivas pueden variar, lo que produce diferencias en las características de los subproblemas.

Flujo de ejecución

Navegar a través de un árbol de recursividad ofrece una visión del flujo de ejecución de una función recursiva. Desde la llamada inicial en el nodo raíz, seguimos las ramas para llegar a las llamadas subsiguientes hasta encontrar el caso base. Cuando se llega a los casos base, las llamadas recursivas empiezan a retornar y sus nodos correspondientes en el árbol se marcan con los valores devueltos. La navegación continúa hasta que se ha recorrido todo el árbol.

Resumen del proceso

Los pasos para resolver recurrencias con el método del arbol de recursión son:

Paso 1: Crear el árbol
Trazamos un árbol cuyo nodo raíz es el costo temporal para el valor n, con ramas indicando cada una de las llamadas recursivas.

Paso 2: Expandir las ramas
Las ramas correspondientes a las llamadas recursivas se expanden, ahora tienen el costo correspondiente a cada llamada y generan nuevas hojas.

Paso 3: Continuar hasta el caso base
Se sigue expandiendo el árbol hasta llegar a un caso base.

Paso 4: Sumar las operaciones por nivel

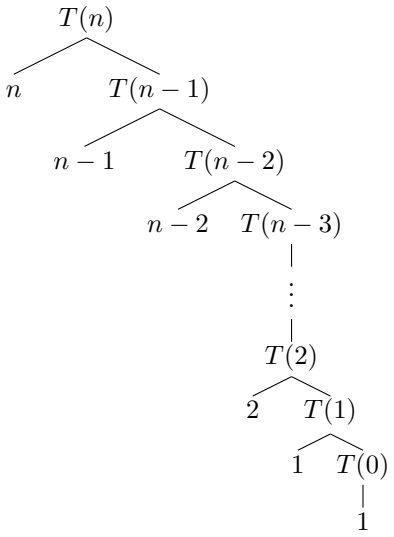
En cada nivel se suma el total de operaciones por nivel. El costo total es la suma de todos los niveles.

Paso 5: Manipular la suma algebraicamente
La suma obtenida se manipula algebraicamente para llegar al resultado final.

3.1 Ejemplos

3.1.1 Ejemplo 1

Consideremos la recurrencia homogénea $t_n = t_{n-1} + n$, con $t_0 = 1$ y armemos su respectivo árbol de recursión.



Si analizamos el tiempo que nos llevaria cada rama, podemos observar que tenemos una sumatoria de 1 hasta n y luego el valor del término inicial t_0, de modo que nos queda:

$$1 + 1 + 2 + 3 + 4 + 5 + 6 + ... + n - 2 + n - 1 + n$$

Esta expresión es equivalente a:

$$t_n = 1 + \frac{n \cdot (n+1)}{2} \Rightarrow t_n \in O(n^2)$$

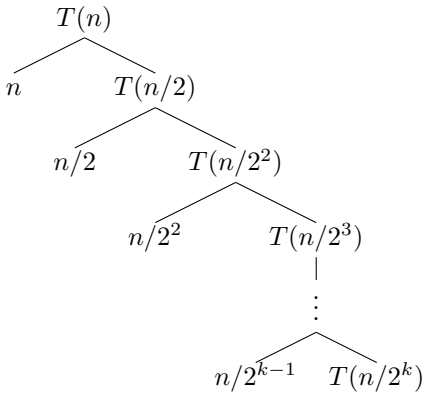
3.1.2 Ejemplo 2

Consideremos la siguiente recurrencia de tipo divide y vencerás: $t_n = t_{n/2} + n$, con $t_1 = 1$ y su respectivo código en Java:

$t_n = t_{n/2} + n$, con $t_1 = 1$

```
public static int fun2(int n) {
    int r;
    if (n == 1) {
        r = 1;
    } else {
        r = fun2(n / 2) + sumatoria(n);
    }
    return r;
}
```

Procedemos a dibujar su respectivo árbol de recursión:



Si analizamos el tiempo que nos llevaria cada rama, nos queda:

$t_n = 1 + n + n/2 + n/2^2 + n/2^3 + ... + n/2^k$

$t_n = 1 + n * [1 + 1/2 + 1/2^2 + 1/2^3 + ... + 1/2^k]$

$t_n = 1 + n * \sum_{i=0}^{k} 1/2^i$

Luego, sabiendo que la solución de la serie geometrica es $\sum_{i=0}^{k} \frac{1}{2^i} = 1$, obtenemos la siguiente solución:

$t_n = n + 1 \Rightarrow t_n \in O(n)$

4 Funciones generadoras

Supongamos que tenemos que resolver la recurrencia $t_n = t_{n-1} + n$, $t_0 = 1$. Si calculamos su secuencia, obtendremos los valores:

$$1,\ 2,\ 4,\ 7,\ 11,\ 16\ldots$$

Multiplicando cada término de la secuencia por x^n la secuencia sería:

$$1,\ 2x^1,\ 4x^2,\ 7x^3,\ 11x^4,\ 16x^5\ldots$$

Dicha secuencia puede ser representada a través de una serie y nos queda de la forma $\sum_{n=0}^{\infty} t_n \cdot x^n$. Esta serie es lo que llamamos función generadora $A(x)$ para cualquier recurrencia de una variable:

$$A(x) = \sum_{n=0}^{\infty} t_n \cdot x^n$$

Esta noción puede ser extendida a recurrencias de dos variables de la siguiente forma:

$$A(x,y) = \sum_{n,m\geq 0} t_{n,m} \cdot x^n y^m$$

4.1 Ejemplo

Una recurrencia homogénea de dos variables conocida como el Triángulo de Pascal es de la forma $t_{n,m} = t_{n-1,m} + t_{n,m-1}$, con $t_{0,r} = t_{s,0} = 1$, para cualquier s, r.

Su función generadora es:

$$A(x,y) = \sum_{n,m\geq 0} t_{n,m} \cdot x^n y^m$$

El primer paso que debemos realizar es multiplicar por $x^n y^m$ y aplicar la sumatoria $\sum_{n,m\geq 0}$ a ambos lados de la recurrencia:

$$t_{n,m} \cdot x^n y^m = (t_{n-1,m} + t_{n,m-1}) \cdot x^n y^m$$
$$t_{n,m} \cdot x^n y^m = t_{n-1,m} \cdot x^n y^m + t_{n,m-1} \cdot x^n y^m$$
$$\sum_{n,m\geq 0} t_{n,m} \cdot x^n y^m = \sum_{n,m\geq 0} (t_{n-1,m} \cdot x^n y^m + t_{n,m-1} \cdot x^n y^m)$$
$$\sum_{n,m\geq 0} t_{n,m} \cdot x^n y^m = \sum_{n,m\geq 0} t_{n-1,m} \cdot x^n y^m + \sum_{n,m\geq 0} t_{n,m-1} \cdot x^n y^m$$

Luego debemos expresarla en torno a $A(x,y)$:

$$\sum_{n,m\geq 0} t_{n,m} \cdot x^n y^m = \sum_{n,m\geq 0} t_{n-1,m} \cdot x^n y^m + \sum_{n,m\geq 0} t_{n,m-1} \cdot x^n y^m$$
$$A(x,y) = x \cdot A(x,y) + y \cdot A(x,y) + t_{inicial}$$
$$A(x,y) = x \cdot A(x,y) + y \cdot A(x,y) + 1$$

Si analizamos los términos de la recurrencia, podemos observar que uno de ellos tiene la forma $t_{n-1,m}$. Esto, en términos de la función generadora, indica que el grado de x está desplazado en una unidad. Para representar ese desplazamiento, debemos multiplicar a $A(x,y)$ por x, que es la variable cuyo grado se desplazó. Similarmente ocurre con el término $t_{n,m-1}$ en función de y. Finalmente, se toma en cuenta el término inicial, el cual es igual a 1 y de no agregarlo no aparecería en la secuencia según las sumatorias definidas. Habiendo tomado todas estas medidas, podemos realizar el despeje de la función generadora para obtener una solución:

$$A(x,y) = x \cdot A(x,y) + y \cdot A(x,y) + 1$$
$$A(x,y) - x \cdot A(x,y) - y \cdot A(x,y) = 1$$
$$A(x,y)(1 - x - y) = 1$$
$$A(x,y) = \frac{1}{1 - x - y}$$

Resumen del proceso

El método de funciones generadores puede ser resumido en simples pasos:

Paso 1 Expresar la recurrencia y su respectiva función generadora $A(x,y)$

Paso 2 Multiplicar por $x^n y^m$ y aplicar sumatoria $\sum_{n,m\geq 0}$ a ambos lados de la recurrencia

Paso 3 Reexpresar la recurrencia en función de $A(x,y)$

Paso 4 Despejar $A(x,y)$

Paso 5 Resolver $A(x,y)$ para obtener una expresión en torno a n, m capaz de calcular cada término $t_{n,m}$

La dificultad para aplicar este método radica en lo desafiante que puede resultar la reexpresión de la recurrencia y la solución de la ecuación final. Es de suma utilidad contar con identidades o propiedades matemáticas para aplicar al momento de desarrollar la recurrencia.

En en ejemplo 4.1 se habló sobre el desplazamiento de n en una unidad, y cómo se corregía para lograr representarlo en torno a la función generadora. A continuación se formaliza este ajuste, que es capaz de reindexar valores de n o m, lo cual es especialmente útil cuando se debe reexpresar en torno a $A(x,y)$:

$$\underline{\text{Reindexación/desplazamiento expresado en torno a } A(x,y)}$$
$$x^k \cdot A(x,y) = \sum_{n,m\geq 0} t_n \cdot x^{n+k} = \sum_{n\geq k, m\geq 0} t_{n-k} \cdot x^n, \forall k \geq 1$$

Cabe recalcar que este método puede ser generalizado para cualquier número de variables, incluso para recurrencias de solo una variable, y que también existen distintos tipos de funciones generadoras, como las de tipo exponencial. Sin embargo, esos conceptos quedan por fuera de este informe.

Bibliografía

[1] Michel Goemans. *Generating Functions*. 2015.

[2] David Guichard. *An Introduction to Combinatorics and Graph Theory*.

[3] Sonoo Jaiswal. *Recursion Tree Method*.

[4] Herbert S. Wilf. *generatingfunctionology*. 1992.

[5] Chung-Chih Li y Kishan Mehrotra. *Problems on Discrete Mathematics*. 2007.

[6] Luis Emilio Montenegro Salcedo y Luz Deicy Alvarado Nieto. *Técnica para solución de recurrencias, usada en el análisis de la complejidad de algoritmos recursivos*. 2015.