INF221 – Algoritmos y Complejidad

Clase #13
Recursividad

Aldo Berrios Valenzuela Hernán Herreros Niño Vicente Lizana Estivill

Martes 13 de septiembre de 2016

Agradecimientos

Agradezco a las personas que me mandaron sus apuntes para poder elaborar este debido a mi ausencia.

1. Recursión

Nuestra herramienta principal es *reducir* problemas a problemas más "simples". Por ejemplo: Expresión regular → programa eficiente para reconocer el problema descrito usamos:

- Expresión Regular a NFA (por ejemplo Thompson)
- NFA a DFA (algoritmo de subconjuntos)
- DFA a DFA mínimo (varias opciones)
- Interpretar el DFA y traducirlo a código.

Al resolver un problema, lo dividimos en subproblemas y combinamos resultados. Notar que al hacer esto (por ejemplo, invocar `printf(3)` en C) confiamos en que la solución al subproblema hace su trabajo correctamente. *Confiamos* en terceros. De la misma manera, al invocar una función que nosotros escribimos, *confiamos* en que hace su trabajo correctamente. Usar recursión es lo mismo... solo que la función se invoca a sí misma (directa o indirectamente). Recursión es inducción en programa.

1.1. Torres de Hanoi

Descripción del problema: Hay 64 placas redondas ubicadas de mayor a menor (Figura 1). Solo se puede mover una placa a la vez y nunca placa mayor sobre una placa menor.

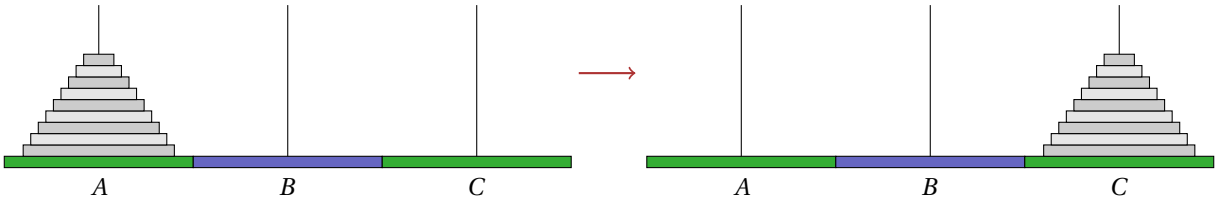

Figura 1: Queremos mover las placas ubicadas en la plataforma A y dejarlas en la plataforma C.

1.1.1. Solución (recursiva)

Una solución (recursiva) es como se muestra en la Figura 2. Esto es solución porque traduce el problema de mover n piezas a mover $n-1$ recursivamente, luego mover 1 (trivial, Figura 2), luego mover $n-1$ recursivamente.

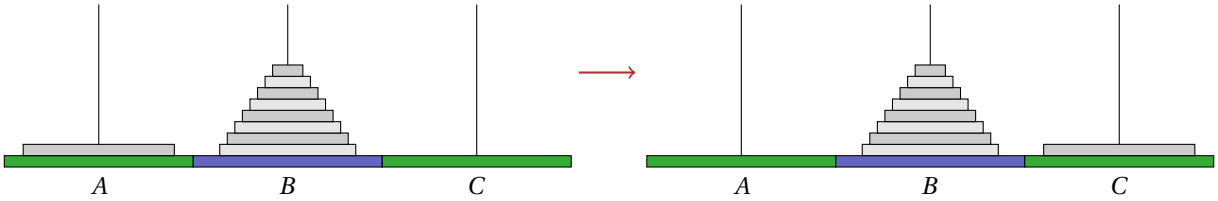

Figura 2: Luego de mover todos los discos entre las tres plataformas y dejarlos ordenados como se muestran en B, sólo nos queda mover el último disco (más grande) de A a C directamente.

Problema: Mover n piezas de A a C.

Base: Si $n = 0$, no hay que hacer nada. □

Inducción: Supongamos que sabemos mover k piezas de i a j (con $i \neq j$). Para mover $k+1$ piezas de A a B (con C de "apoyo"):

- Movemos las k piezas superiores de A a C.
- Movemos la pieza inferior de A a B.
- Movemos las k piezas de C a B.

□

Pseudocódigo:

```
hanoi (n, src, dst, tmp):
  if n > 0:
    hanoi (n-1, src, tmp, dst)
    move src -> dst
    hanoi (n-1, tmp, dst, src)
```

Aún falta convencerse de que *nunca* movemos uno grande sobre uno menor (→ luego de `hanoi (n, src, dst, tmp)`, `tmp` queda libre).

¿Cuántas movidas se requieren?. Sea $T(n)$ el número de movidas para transferir n platos.

$$\begin{aligned} T(0) &= 0 \\ T(n+1) &= 2T(n)+1, \qquad n \geq 0 \end{aligned}$$

Sea:

$$h(z) = \sum_{n \geq 0} T(n)\, z^n \tag{1.1}$$

Por propiedades:

$$\frac{h(z) - T(0)}{z} = 2h(z) + \frac{1}{1-z}$$

Entonces, despejando $h(z)$ de lo anterior se obtiene:

$$\begin{aligned} h(z) &= \frac{z}{(1-z)(1-2z)} \qquad (1.2) \\ h(z) &= \frac{A}{1-z} + \frac{B}{1-2z} \\ h(z)(1-z) &= A + \frac{(1-z)B}{1-2z} \end{aligned}$$

En seguida, aplicamos límite cuando $z \to 1$:

$$h(z)(1-z) = A + \frac{(1-z)B}{1-2z} \qquad \Big/ \lim_{z\to 1}()$$

$$\lim_{z\to 1} h(z)(1-z) = \lim_{z\to 1}\left(A + \frac{(1-z)B}{1-2z}^{\to 0}\right)$$

$$\lim_{z\to 1} h(z)(1-z) = A \tag{1.3}$$

Luego, reemplazamos $h(z)$ de (1.3) por lo que está en (1.2):

$$A = \lim_{z\to 1}\left(\frac{z}{(1-z)(1-2z)}(1-z)\right)$$

$$A = \lim_{z\to 1}\frac{z}{1-2z}$$

$$\therefore A = -1 \tag{1.4}$$

Análogamente:

$$B = 1 \tag{1.5}$$

Entonces:

$$h(z) = \frac{1}{1-2z} - \frac{1}{1-z} \tag{1.6}$$

Usando la fórmula de series geométricas, es sabido que:

$$\frac{1}{1-2z} = \sum_{n\geq 0} 2^n z^n \qquad \wedge \qquad \frac{1}{1-z} = \sum_{n\geq 0} z^n \tag{1.7}$$

Luego, si reemplazamos $h(z)$ por (1.1) en (1.6) y las fracciones de (1.6) por las obtenidas en (1.7) se obtiene:

$$\sum_{n\geq 0} T(n) z^n = \sum_{n\geq 0} 2^n z^n - \sum_{n\geq 0} z^n$$

$$\Rightarrow T(n) = 2^n - 1$$

1.2. Mergesort

Queremos ordenar $A[1,\ldots,n]$. Nuestro pseudocódigo para mergesort es:

```
mergeSort(A[1,...,n]):
    if n > 1:
        mergeSort(A[1,...,⌊n/2⌋])
        mergeSort(A[⌊n/2⌋ + 1,...,n])
        merge(A[1,...,⌊n/2⌋], A[⌊n/2⌋ + 1,...,n])
```

De la misma forma que lo hicimos con las torres de Hanoi: ¿Cuántas iteraciones se requieren?. Sea $M(n)$ el número de iteraciones para completar el merge sort. Es claro que:

$$M(0) = M(1)$$

$$M(n) = M\left(\left\lfloor \frac{n}{2}\right\rfloor\right) + M\left(\left\lceil \frac{n}{2}\right\rceil\right) + n \tag{1.8}$$

Supongamos $n = 2^k$. Reemplazando sobre (1.8) vamos obteniendo:

$$M\left(2^k\right) = M\left(\left\lfloor \frac{2^k}{2}\right\rfloor\right) + M\left(\left\lceil \frac{2^k}{2}\right\rceil\right) + 2^k$$

$$= M\left(\left\lfloor 2^{k-1}\right\rfloor\right) + M\left(\left\lceil 2^{k-1}\right\rceil\right) + 2^k \tag{1.9}$$

Es claro que $2^{k-1} \in \mathbb{N}$, por lo tanto:

$$M\left(\left\lfloor 2^{k-1} \right\rfloor\right) = M\left(\left\lceil 2^{k-1} \right\rceil\right) = M\left(2^{k-1}\right) \tag{1.10}$$

Luego, reemplazando (1.10) en (1.9) obtenemos:

$$M\left(2^{k}\right) = 2M\left(2^{k-1}\right) + 2^{k} \tag{1.11}$$

Sea $m(k) = M\left(2^{k-1}\right)$ `/* DUDA: en los apuntes entregados dice` $m(k) = M\left(2^{k}\right)$`...¿realmente es así? */`. Entonces, reemplazando sobre (1.11) se tiene:

$$m(k+1) = 2m(k) + 2^{k} \tag{1.12}$$

`/* DUDA: hay cosas que no calzan en los apuntes... */`