Universidad Nacional Autónoma de México

Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas

Axel Daniel Malváez Flores

Examen 2

Matemáticas Discretas

---

11 de Octubre de 2022

1. Problemas

1. (2.5 pts) Estima de manera asintótica la cantidad de pasos del modelo RAM que usa el siguiente algoritmo de Python, donde n es un número entero. Tu respuesta debe dar la cantidad de pasos $f(n)$ en términos de la notación Θ de otra función sencilla.

   Análisis en el peor caso:

```
1       min = 1/3   # 1 OE
2       i_buena, j_buena, k_buena=1,1,1 # 3 OE
3       for j in range(n):  #  n  OE
4           for k in range(j): # n-1 OE (cuando j = n-1)
5               for l in range(j+k): # 2n - 3 OE (si j = n-1 y k = n-2)
6                   if 1/(i+k+l) < min: # 8 OE (4 accesos,2 sumas, 1 div, 1 <)
7                       min=1/i+k+l # 7 OE (3 accesos, 2 sumas, 1 div, 1 asignacion)
8                       i_buena = i # 2 OE
9                       j_buena = j # 2 OE
10                      k_buena = k # 2 OE
11      print(i_buena, j_buena, k_buena, min) # 5 OE (suponiendo, print 1 OE)
```

   Tenemos entonces que las operaciones elementales del algoritmo son:

$$\begin{aligned} 1+3+(n((n-1)((2n-3)(8+7+2+2+2)))) + 5 &= n(n-1)(2n-3)(21)+9 \\ &= (n^2-n)(2n-3)(21)+9 \\ &= (2n^3-2n^2-3n^2+3n)(21)+9 \\ &= (42n^3-105n^2+63n)+9 \\ &= 42n^3-105n^2+63n+9 \end{aligned}$$

   Por tanto a lo más los pasos de nuestro algoritmo con n un entero, será dado por $f(n) = 42n^3 - 105n^2 + 63n + 9$. Ahora notemos que como:

$$\begin{aligned} 42n^3 &\leq 42n^3 \quad \forall n \geq 1 \\ -105n^2 &\leq n^3 \quad \forall n \geq 1 \\ 63n &\leq 63n^3 \quad \forall n \geq 1 \\ 9 &\leq 9n^3 \quad \forall n \geq 1 \end{aligned}$$

   así:

$$\begin{aligned} 42n^3 - 105n^2 + 63n + 9 &\leq 42n^3 + n^3 + 63n^3 + 9n^3 \quad \forall n \geq 1 \\ 42n^3 - 105n^2 + 63n + 9 &\leq 115n^3 \quad \forall n \geq 1 \end{aligned}$$

Entonces si $C = 115$, $n_0 = 1$ y $g(n) = n^3$, entonces:

$$f(n) \leq Cg(n) \quad \forall n \geq n_0$$

$\therefore f(n) = O(n^3)$, o bien el algoritmo tiene complejidad $O(n^3)$

Ahora verifiquemos si $g(n) = O(f(n))$, esto quiere decir que tenemos que demostrar que n^3 es $O(42n^3 - 105n^2 + 63n + 9)$, entonces usando el *criterio del límite*, vemos que:

$$\begin{aligned}
\lim_{n\to\infty} \frac{n^3}{42n^3 - 105n^2 + 63n + 9} &= \lim_{n\to\infty} \frac{\frac{n^3}{n^3}}{\frac{42n^3}{n^3} - \frac{105n^2}{n^3} + \frac{63n}{n^3} + \frac{9}{n^3}} \\
&= \lim_{n\to\infty} \frac{1}{42 - \frac{\cancel{105}}{\cancel{n}} + \frac{\cancel{63}}{\cancel{n^2}} + \frac{\cancel{9}}{\cancel{n^3}}} \\
&= \frac{1}{42 - 0 + 0 + 0} \\
&= \frac{1}{42}
\end{aligned}$$

Por lo tanto como el límite existe y es finito entonces $g(n) = O(f(n))$.

$\therefore$ Como tenemos que $f(n) = O(g(n))$ y $g(n) = O(f(n))$, entonces tenemos que $f(n) = \Theta(g(n))$ o bien $f(n) = \theta(n^3)$. Así tenemos que la complejidad en notación Θ es cúbica.

2. Ordena asintóticamente las siguientes funciones de acuerdo a cuáles son dominadas por cuáles otras. Si hay dos con el mismo crecimiento asintótico, no importa cuál pongas primero:

$$(\log n)^2, \log(\log n), \log(n^2), \log(n!), n^2$$

Para cada una de tus comparaciones, verifica que en efecto es válida usando el criterio del límite superior.

Recordemos que el criterio del límite superior nos dice que se tiene que $f(n) = O(g(n))$ si y sólo si

$$lim_{n\to\infty} sup \frac{f(n)}{g(n)} < \infty$$

.

Por lo que entonces haremos por casos y evaluaremos en cada caso el límite entre dos funciones distintas para compararlas y verificar cuál domina a cuál.

- Si $f(n) = \log(\log(n))$ y $g(n) = \log(n^2)$, entonces viendo el límite tenemos que

  Usando la regla de L'Hopital

$$\begin{aligned}
\lim_{n\to\infty} \frac{\log(\log(n))}{\log(n^2)} &= \lim_{n\to\infty} \frac{\frac{1}{n\log(n)}}{\frac{2}{n}} \\
&= \lim_{n\to\infty} \frac{\cancel{n}}{2\cancel{n}\log(n)} \\
&= \lim_{n\to\infty} \frac{1}{2\log(n)} \\
&= \frac{1}{2}\left(\lim_{n\to\infty} \frac{1}{\log(n)}\right) \\
&= \frac{1}{2}\left(\lim_{n\to\infty} \frac{1}{\log(n)} = 0\right) \\
&= \frac{1}{2}\cdot 0 \\
&= 0
\end{aligned}$$

  Por lo tanto como el límite existe y es finito entonces $f(n) = O(g(n))$.

  Ahora veamos si $g(n) = O(f(n))$:

  Usando la regla de L'Hopital

$$\begin{aligned}
\lim_{n\to\infty} \frac{\log(n^2)}{\log(\log(n))} &= \lim_{n\to\infty} \frac{\frac{2}{n}}{\frac{1}{n\log(n)}} \\
&= \lim_{n\to\infty} \frac{2\cancel{n}\log(n)}{\cancel{n}} \\
&= \lim_{n\to\infty} 2\log(n) \\
&= 2\lim_{n\to\infty} \log(n) \\
&= 2\cdot(\lim_{n\to\infty} \log(n) = \infty) \\
&= \infty
\end{aligned}$$

  Entonces notamos que $g(n) \neq O(f(n))$.

  $\therefore$ Como $\log(\log(n)) = O(\log(n^2))$, pero no es cierto que $\log(n^2) = O(\log(\log(n)))$, entonces $\log(\log(n))$ es dominada por $\log(n^2)$.

- Si $f(n) = \log(n^2)$ y $g(n) = (\log n)^2$, entonces viendo el límite tenemos que

  Usando la regla de L'Hopital

$$\begin{aligned}\lim_{n\to\infty} \frac{\log(n^2)}{(\log n)^2} &= \lim_{n\to\infty} \frac{\frac{2}{n}}{\frac{2log(n)}{n}} \\ &= \lim_{n\to\infty} \frac{\cancel{2n}}{\cancel{2n}\log(n)} \\ &= \lim_{n\to\infty} \frac{1}{\log(n)} \\ &= 0\end{aligned}$$

  Por lo tanto como el límite existe y es finito entonces $f(n) = O(g(n))$.

  Ahora veamos si $g(n) = O(f(n))$:

  Usando la regla de L'Hopital

$$\begin{aligned}\lim_{n\to\infty} \frac{(\log n)^2}{\log(n^2)} &= \lim_{n\to\infty} \frac{\frac{2log(n)}{n}}{\frac{2}{n}} \\ &= \lim_{n\to\infty} \frac{\cancel{2n}\log(n)}{\cancel{2n}} \\ &= \lim_{n\to\infty} \log(n) \\ &= \infty\end{aligned}$$

  Entonces notamos que $g(n) \neq O(f(n))$.

  $\therefore$ Como $\log(n^2) = O((\log n)^2)$, pero no es cierto que $(\log n)^2 = O(\log(n^2))$, entonces $\log(n^2)$ es dominada por $(\log n)^2$.

Para las últimas dos comparaciones, primero veamos que $\log(n!) = \Theta(n\log(n))$.

Dem
Sabemos que

$$\log(n!) = \log(1) + \log(2) + \ldots + \log(n-1) + \log(n)$$

Entonces notemos que en particular

$$\log(1) + \log(2) + \ldots + \log(n-1) + \log(n) \leq \log(n) + \log(n) + \ldots + \log(n) + \log(n)$$

$$\log(1) + \log(2) + \ldots + \log(n-1) + \log(n) \leq n\log(n)$$

Sea $f(n) = \log(n!)$, $g(n) = n\log(n)$, $C = 1$ y $n_0 = 1$, entonces tenemos que

$$f(n) \leq Cg(n) \quad \forall n \geq n_0$$

$\therefore$ Por lo que entonces $\log(n!) = O(n\log(n))$.

Por otra parte tenemos que como

$$\log(n!) = \log(1) + \ldots + \log(\frac{n}{2}) + \ldots + \log(n)$$

entonces es fácil notar que si tomamos desde la mitad hasta el final de esta suma, esto será menor que $\log(n!)$, es decir

$$\begin{aligned}
\log(1) + \ldots + \log(\frac{n}{2}) + \ldots + \log(n) &\geq \log(\frac{n}{2}) + \ldots + \log(n) \\
&= \log(\frac{n}{2}) + \log(\frac{n}{2}+1) + \log(\frac{n}{2}+2)\ldots + \log(n-1) + \log(n) \\
&\geq \underbrace{\log(\frac{n}{2}) + \log(\frac{n}{2}) + \ldots + \log(\frac{n}{2})}_{\frac{n}{2} \text{ sumandos}} \\
&= \frac{n}{2} \cdot \log(\frac{n}{2})
\end{aligned}$$

Entonces sea $f(n) = \log(n!)$, $g(n) = n\log(\frac{n}{2})$, $C = \frac{1}{2}$ y $n_0 = 1$, así

$$f(n) \geq Cg(n) \quad \forall n \geq n_0$$

$\therefore$ Por lo que entonces $\log(n!) = \Omega(n\log(n))$.

$\therefore$ Como tenemos que $\log(n!) = \Omega(n\log(n))$ y $\log(n!) = O(n\log(n))$, entonces $\log(n!) = \Theta(n\log(n))$

Por lo tanto en lugar de trabajar con $\log(n!)$, podemos trabajar con $n\log(n)$, así entonces

- Si $f(n) = (\log n)^2$ y $g(n) = n\log(n)$. Entonces viendo el límite tenemos que

$$\lim_{n\to\infty} \frac{(\log n)^2}{n\log(n)} = \lim_{n\to\infty} \frac{\log(n)}{n}$$

Usando la regla de L'Hopital

$$\begin{aligned} &= \lim_{n\to\infty} \frac{1}{n} \\ &= 0 \end{aligned}$$

Por lo tanto como el límite existe y es finito entonces $f(n) = O(g(n))$.

Ahora veamos si $g(n) = O(f(n))$:

$$\lim_{n\to\infty} \frac{n\cancel{\log(n)}}{(\log n)\cancel{^2}} = \lim_{n\to\infty} \frac{n}{\log(n)}$$

Usando la regla de L'Hopital

$$\begin{aligned} &= \lim_{n\to\infty} \frac{1}{\frac{1}{n}} \\ &= \lim_{n\to\infty} n \\ &= \infty \end{aligned}$$

Entonces notamos que $g(n) \neq O(f(n))$.

$\therefore$ Como $(\log n)^2 = O(n\log(n))$, pero no es cierto que $n\log(n) = O((\log n)^2)$, entonces $(\log n)^2$ es dominada por $n\log(n)$.

- Si $f(n) = n\log(n)$ y $g(n) = n^2$. Entonces viendo el límite tenemos que

$$\lim_{n\to\infty} \frac{\cancel{n}\log(n)}{n^{\cancel{2}}} = \lim_{n\to\infty} \frac{\log(n)}{n}$$

Usando la regla de L'Hopital

$$\begin{aligned} &= \lim_{n\to\infty} \frac{1}{n} \\ &= 0 \end{aligned}$$

Por lo tanto como el límite existe y es finito entonces $f(n) = O(g(n))$.

Ahora veamos si $g(n) = O(f(n))$:

$$\lim_{n\to\infty} \frac{n^{\cancel{2}}}{\cancel{n}\log(n)} = \lim_{n\to\infty} \frac{n}{\log(n)}$$

Usando la regla de L'Hopital

$$\begin{aligned} &= \lim_{n\to\infty} \frac{1}{\frac{1}{n}} \\ &= \lim_{n\to\infty} n \\ &= \infty \end{aligned}$$

Entonces notamos que $g(n) \neq O(f(n))$.

$\therefore$ Como $n\log n = O(n^2)$, pero no es cierto que $n^2 = O(n\log(n))$, entonces $n\log(n)$ es dominada por n^2.

Por lo tanto vemos que como:

- $\log(\log(n))$ es dominada por $\log(n^2)$
- $\log(n^2)$ es dominada por $(\log n)^2$

  Ahora como $\log(n!) = \Theta(n\log(n))$
- $(\log n)^2$ es dominada por $n\log(n)$ o bien por $\log(n!)$
- $n\log(n)$ o bien $\log(n!)$ es dominada por n^2

Así, nuestro orden final desde el más pequeño al más grande queda como:

$$\log(\log(n)), \log(n^2), (\log n)^2, \log(n!), n\log(n), n^2$$

3. Tienes un arreglo A que tiene n elementos, cada uno de ellos verde, blanco o rojo. Se quiere ordenar los elementos del arreglo de modo que aparezcan primero todos los verdes, luego todos los blancos y luego todos los rojos. Sólo se permiten operaciones del siguiente estilo:

   - Para cualquier i, examinar de qué color es el i-ésimo elemento de A.
   - Para cualesquiera i y j, intercambiar los elementos en la posición i y j de A.

   Encuentra un algoritmo correcto que ordene a los elementos. Tu solución debe correr en tiempo O(n).

```
1   n = longitud de A # 2 OE suponiento len(A) es O(1)
2
3   menor = 0  # 1 OE
4   mayor = n - 1  # 3 OE
5   medio = 0  #1 OE
6
7   while medio <= mayor: # n OE (recorremos todo el arreglo)
8       if arr[medio] == 'verde' : # 3 OE sup. comparacion constante
9           # Aqui intercambiamos
10          arr[menor], arr[medio] = arr[medio], arr[menor] #4 OE
11          menor += 1 # 2 OE
12          medio += 1 # 2 OE
13      elif arr[medio] == 'blanco': # 3 OE sup. comparacion constante
14          medio += 1 # 2 OE
15      else:
16          # Aqui intercambiamos
17          arr[medio], arr[mayor] = arr[mayor], arr[medio] # 4 OE
18          mayor -= 1 # 2 OE
```

Tenemos que entonces a lo más el número de pasos se dará si únicamente entra a puros `if`, o bien solo hay bolas verdes, entonces la función que nos da los pasos será

$$\begin{aligned} f(n) &= 2 + 1 + 3 + 1 + n(11) \\ &= 11n + 7 \end{aligned}$$

Para ver que $f(n) = O(n)$, usamos el *criterio del límite*, entonces veamos:

Utilizando la regla de L'Hopital

$$\begin{aligned} \lim_{n\to\infty} \frac{11n+7}{n} &= \lim_{n\to\infty} \frac{11}{1} \\ &= \lim_{n\to\infty} 11 \\ &= 11 \end{aligned}$$

$\therefore f(n) = O(n)$.

Ahora para ver que es **correcto**, notemos que en cada iteración iremos recorriendo los elementos del arreglo A. Inicialmente tendremos tres apuntadores, *menor* para saber en qué posición van los elementos **verdes**, *medio* para saber en cuál van los **blancos** y finalmente *mayor* para saber en qué posición están los **rojos**. Como los apuntadores *menor* y *medio* se inicializan en el primer elemento del arreglo y el *mayor* en el último elemento, en la primera iteración verificamos, si es verde los apuntadores *medio* y *menor* aumentan en 1, si es blanco, únicamente avanza el apuntador *medio* que es el correspondiente al lugar donde inician los blancos, por otra parte si es rojo, el apuntador *mayor* disminuye (intuitivamente en uno avanzamos en el arreglo y en otro retrocedemos en el arreglo), si avanzamos, nos quedan $n - 1$ elementos por recorrer y si retrocedemos también, por tanto cada que avancemos o retrocedamos disminuirá el arreglo a verificar, así irémos verificando con el apuntador *medio* que es el que se va moviendo conforme a las iteraciones. Una vez que el apuntador *medio* choque con el apuntador *mayor* correspondiente a la casilla donde inician los elementos rojo, habremos terminado, esto nos regresa una lista donde los primeros elementos son de color verde, después están los blancos y al final los rojos.

4. Se tienen n islas. Para cada j en $\{1, 2, ..., n\}$, en la isla j hay un tesoro con A_j monedas de oro, pero hay B_j tigres que se tienen que enfrentar para obtener el tesoro. Un capitán pirata desea visitar algunas de las islas, pero su tripulación accederá a ello sólo si tras cada isla visitada, la cantidad de monedas de oro obtenidas en total hasta el momento es mayor al número de tigres enfrentados en total hasta el momento. El capitán pirata quiere determinar cuál es la mayor cantidad de monedas de oro que puede obtener.

Escribe en pseudocódigo un algoritmo que resuelva este problema y demuestra que es correcto. Estima de manera asintótica cuántos pasos tomará en el modelo RAM. Explica las suposiciones que estás haciendo. No es necesario que tu algoritmo sea óptimo.

Explicación del pseudocódigo

Iniciamos con un número n de islas. Posteriormente iniciamos la lista A ($[A_0, A_1, \ldots, A_n]$) que es una lista que en la entrada i guarda la cantidad de monedas que hay en la isla i, además también inicializamos la lista B ($[B_0, B_1, \ldots, B_n]$) que es una lista que en la entrada i guarda la cantidad de tigres que cuidan el tesoro en la isla i.

Posteriormente creamos la lista L es una lista de tuplas, donde cada tupla representa una isla, así la isla i es la tupla (A_i, B_i), esta lista L tiene n tuplas.

Luego creamos la lista CL que es la lista de las combinaciones en las que se pueden recorrer las n tuplas, de las cuales hay $n!$ posibles formas.

Finalmente declaramos una variable *gananciaMayor* que nos guardará la mayor ganancia posible al momento de recorrer nuestras islas.

Entonces el algoritmo sigue:

a) Para cada posible forma de recorrido j en CL, vamos iterando sobre este recorrido j las islas y mientras la cantidad de monedas obtenidas no sea menor a la cantidad de tigres seguimos visitando, si resulta que la cantidad de monedas totales es menor a la de tigres totales rompemos el ciclo (pues es la restricción de la tripulación).

b) Verificamos si el número de monedas totales obtenidas en ese recorrido es mayor a la variable *gananciaMayor* y de ser así actualizamos esta variable.

c) Finalmente después de recorrer todas las posibles formas de recorrer las islas, regresamos la variable *gananciaMayor*

Notemos que este algoritmo es bastante ineficiente, en primera porque utilizamos las posibles formas de recorrer las islas y esto se hace en $O(n!)$, además puede que existan recorridos i y j que difieran a partir de la mitad, pero que el algoritmo en el segundo paso termine en una entrada menor a la mitad, de ser así tanto el recorrido i como el j dan el mismo valor de monedas totales, por lo que es innecesario volver a hacer el cálculo.

Propuesta de algoritmo:

```
1   n <- numero de islas
2
3   A <- Lista con la cantidad de monedas en la isla i para la entrada i
4   B <- Lista con la cantidad de tigres en la isla i para la entrada i
5   L <- Lista de tuplas (A_i, B_i) que representan a una isla i (hay n
        tuplas de la forma (A_i, B_i))
6   CL <- Lista de formas distinta de visitar las n islas (hay n! formas)
        donde la isla 0 es (A_0, B_0), la isla 1 es (A_1, B_1) y asi
        sucesivamente.
7
8   gananciaMayor <- 0
9
10  for cada posible recorrido de islas j in CL then
11      monedas <- 0
12      tigres <- 0
13      visitada <- 0
14      while el no. monedas sea mayor que el no. tigres then
15          cant_monedas, cant_tigres <- j[visitada]
16          monedas <- monedas + cant_moneda
17          tigres <- tigres + cant_tigres
18          visitada <- visitada + 1
19
20      if monedas es mayor a gananciaMayor then
21          gananciaMayor <- monedas
22
23  regresa gananciaMayor
```

El algoritmo es correcto pues verifica para todas las formas posibles de recorrer nuestras n islas y guarda la *gananciaMaxima*, así eventualmente tendremos que un recorrido generará la *gananciaMaxima* y por tanto este será el valor esperado de la función.

Análisis del algoritmo: Suponiendo que todo este algoritmo está dentro de una función, tenemos que

- En la línea 1, suponiendo que sabemos el número de islas, la complejidad es $O(1)$.
- En la línea 3, suponiendo que ya nos dan la lista A, entonces esto es $O(1)$
- En la línea 4, suponiendo que ya nos dan la lista B, entonces esto es $O(1)$
- En la línea 5, suponiendo que nosotros hacemos la lista de tuplas, tenemos que recorrer A y B, sin embargo puede ser en el mismo ciclo y por tanto esto es $O(n)$
- En la línea 6, dado que tenemos n islas, la complejidad de listar las $n!$ posibles formas de recorrer las islas es de complejidad $O(n!)$.
- En la línea 8, dado que asignamos un valor a una variable, esto es $O(1)$.
- Ahora primero analizamos lo del ciclo interno, las líneas $11, 12, 13$ se hacen en $O(1)$, el ciclo *while* se repite a lo más n veces pues recorremos las n islas del recorrido j. Por tanto la línea 14 tiene $O(n)$, finalmente las lineas $20, 21$ se hacen en $O(1)$.
- Dado que la parte interna del ciclo tiene complejidad $O(n)$ y nuestro ciclo se repite $n!$ veces pues iteramos CL, entonces la complejidad desde la línea 10 hasta la 21 es de $O(n \cdot n!)$.
- Finalmente la línea 23, corre en tiempo $O(1)$.

Por lo que entonces deducimos que la complejidad de nuestro algoritmo es de $O(n \cdot n!)$. Claramente es muuuuy ineficiente.