Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC1253 - Matemáticas Discretas

Tarea 5

20 de octubre de 2020

2º semestre 2020 - Profesores G. Diéguez - F. Suárez

Requisitos

- La tarea es individual. Los casos de copia serán sancionados con la reprobación del curso con nota 1,1.
- **Entrega**: Hasta las 23:59:59 del 2 de noviembre a través del buzón habilitado en el sitio del curso (Canvas).
  - Esta tarea debe ser hecha completamente en LaTeX. Tareas hechas a mano o en otro procesador de texto **no serán corregidas**.
  - Debe usar el template LaTeX publicado en la página del curso.
  - Cada problema debe entregarse en un archivo independiente de las demas preguntas.
  - Los archivos que debe entregar son un archivo `PDF` por cada pregunta con su solución con nombre `numalumno-P1.pdf` y `numalumno-P2.pdf` , junto con un `zip` con nombre `numalumno.zip`, conteniendo los archivos `numalumno-P1.tex` y `numalumno-P2.tex` que compilan su tarea. Si su código hace referencia a otros archivos, debe incluirlos también.
- El no cumplimiento de alguna de las reglas se penalizará con un descuento de 0.5 en la nota final (acumulables).
- No se aceptarán tareas atrasadas.
- Si tiene alguna duda, el foro de Canvas es el lugar oficial para realizarla.

Problemas

Problema 1 - Correctitud y Complejidad

a) Escriba un algoritmo iterativo que resuelva el problema del Mínimo Común Múltiplo.

   Su algoritmo debe recibir como input dos números y devolver como output el número natural que corresponda al mínimo común múltiplo del input.

b) Demuestre que su algoritmo es correcto.

c) Calcule la complejidad de su algorimo, en el mejor y peor caso, en función de la cantidad de dígitos del input.

Solución

a) Input: $a, b \in \mathbb{N} \setminus \{0\}$

   Output: $\text{mcm}(a, b)$

   Código:

```
def mcm(a,b):
    init_a = a
    init_b = b
    k = 0
    while a != b:
        if a < b:
            a = a + init_a
        else:
            b = b + init_b
        k = k + 1
    return a
```

b) Como invariante del algoritmo elegimos:

$$\frac{a}{\text{init\_a}} + \frac{b}{\text{init\_b}} = k + 2$$

   Notemos que k representa el número de iteración del while.

   BI: Para $k = 0$ tenemos que $\text{init\_a} = a$ y que $\text{init\_b} = b$, por ende:

$$\frac{a}{\text{init\_a}} + \frac{b}{\text{init\_b}} = 2 = k + 2$$

HI: Supongamos que para $k = n$ se cumple que:

$$\frac{a_n}{\text{init\_a}} + \frac{b_n}{\text{init\_b}} = n + 2$$

TI: En la iteración $k = n + 1$ entonces, existen 2 situaciónes, $a_{n+1} = a_n + \text{init\_a}$ o (excluyente) $b_{n+1} = b_n\text{init\_b}$. Sin pérdida de generalidad, suponemos que $a_{n+1} = a_n + \text{init\_a}$.

$$\begin{aligned}
\frac{a_{n+1}}{\text{init_a}} + \frac{b_{n+1}}{\text{init_b}} &= \frac{a_n + \text{init_a}}{\text{init_a}} + \frac{b_{n+1}}{\text{init_b}} && (1)\\
&= \frac{a_n}{\text{init_a}} + \frac{b_n}{\text{init_b}} + 1 && (2)\\
&= n + 2 + 1 && (3)\\
&= (n + 1) + 2 && (4)
\end{aligned}$$

Por lo tanto, se cumple que es invariante. Ahora, si el algoritmo termina, sabemos que $a = b$. Por lo tanto, como trivialmente init_a$|a$ y init_b$|b$, significa que esas divisiones deben ser enteros. Pero, como $a = b$, entonces ambos son divisibles por tanto init_a como init_b. Es decir, el valor final de a y b es múltiplo de los valores iniciales. Falta demostrar que es el mínimo.

Digamos, por contradicción, que existe un múltiplo de los valores iniciales que es menor que el que retorna nuestro algoritmo. Entonces, si nuestro algoritmo returna un $v = k_a\text{init\_a} = k_b\text{init\_b}$, y existe un $w = j_a\text{init\_a} = j_b b$ tal que $w < v$, entonces necesariamente $k_a > j_a$ o $k_b > j_b$. Sin pérdida de generalidad, $k_a > j_a$. Sin embargo, nuestro algoritmo necesariamente pasó por algún a tal que $a = j_a\text{init\_a}$ y tuvo que haber existido una primera iteración en la que se llegó a $a = j_a\text{init\_a}$. Por lo tanto, como el algoritmo no terminó en esa iteración, existen dos situaciones: $a > b$ o $b > a$. Si $a > b$, entonces implica que $b < j_b\text{init\_b}$, por lo que se incrementaría b hasta llegar a j_b, lo cual es una contradiccón ya que implica que w se alcanzó en nuestro algoritmo. Si $b > a$, entonces implica que $b > j_b\text{init\_b}$, por lo que nuestro algoritmo tuvo en alguna iteración $b = j_b\text{init\_b}$, lo que lleva de nuevo a la misma situación. Sin embargo, ahora sólo existe el caso de que $b > a$, lo que genera la misma contradicción que antes.

Para demostrar que el algoritmo termina, sabemos que el peor caso es cuando el mínimo común múltiplo es $a \cdot b$, lo que significa que a a debemos sumarle b veces a y a b debemos sumarle a veces b, esto quiere decir que como dentro del loop siempre se van sumando los valores inciales a las variables x e y, en algún momento debemos llegar a que x y y serán igual a $a \cdot b$.

c) Es claro que el mejor caso se alcanza cuando el algoritmo no entra al while, es decir, cuando a es igual a b, de modo que se ejecuta una cantidad fija y constante de pasos $\mathcal{O}(1)$, por lo tanto en el mejor caso el algoritmo tiene complejidad $\mathcal{O}(1)$.

Por otro lado, el peor caso ocurre cuando el mínimo común múltiplo de a, b es $a \cdot b$. Esto se debe a que para que a llege a $a \cdot b$, se debe sumar b veces a y para que b llegue al resultado $a \cdot b$ se le debe sumar a veces b, de modo que la complejidad en términos numéricos sería $\mathcal{O}(a + b)$. Luego, esto se expresa en términos de los dígitos de cada número, x e y respectivanente, sabiendo que para un número f con d dígitos, se tiene $f \leq 10^d$, y por lo tanto:

$$\mathcal{O}(a + b) \leq \mathcal{O}(10^x + 10^y)$$

Finalmente, la complejidad en el peor caso es $\mathcal{O}(10^x + 10^y)$

Pauta (6 pts.)

a) 1 pto

b) 2 ptos

c) 3 ptos

Puntajes parciales y soluciones alternativas a criterio del corrector.

Problema 2 - Notación asintótica

Son ciertas las siguientes afirmaciones? Demuestre

a) $\sqrt{n!} \in O(n\sqrt{n})$

b) $n^{\sqrt{n}} \in O((\sqrt{n})^n)$

Solución

a) La afirmación es falsa, lo demostraremos por contradicción.

Supongamos que $\sqrt{n!} \in O(n\sqrt{n})$. Por definición sabemos que existe un $c \in \mathbb{R}$ y un $n_0 \in \mathbb{N}$ tal que para todo $n \geq n_0$ se tiene que

$$\begin{aligned}\sqrt{n!} &\leq c \cdot n\sqrt{n} \\ &= \sqrt{c^2 \cdot n^2 \cdot n} \\ &= \sqrt{c^2 \cdot n^3}\end{aligned}$$

Cancelando las raíces:

$$\begin{aligned}n! &\leq c^2 \cdot n^3 \\ &\leq c_1 \cdot n^3\end{aligned}$$

De donde obtenemos que $n! \in O(n^3)$, esto es una contradicción ya que sabemos el factorial crece más rápido que las funciones polinomiales en términos asintóticos[1]. Concluímos que $\sqrt{n!} \notin O(n\sqrt{n})$.

Pauta (3 pts.)

- 3.0 ptos por despejar y concluir.

Puntajes parciales y soluciones alternativas a criterio del corrector.

b) La afirmación es verdadera, para demostrarlo utilizaremos el siguiente reemplazo:

$$u = \sqrt{n} \Rightarrow u^2 = n$$

Si reemplazamos en la expresión original obtenemos que

$$n^{\sqrt{n}} = (u^2)^u = u^{2u} \tag{1}$$

[1]Como ejercicio, puede mostrarlo por inducción.

Por otro lado sabemos que

$$2u \leq u^2 \tag{2}$$

con $u \geq 2$, combinando (1) y (2):

$$(u^2)^u = u^{2u} \leq u^{u^2} \tag{3}$$

Ahora podemos reemplazar los valores originales en (3):

$$(u^2)^u \leq u^{u^2} \Rightarrow n^{\sqrt{n}} \leq (\sqrt{n})^n$$

con $u \geq 2 \Rightarrow u^2 \geq 4 \Rightarrow n \geq 4$. Concluímos que $n^{\sqrt{n}} \in O((\sqrt{n})^n)$ con $c = 1$ y $n_0 = 4$.

Pauta (3 pts.)

- 1.5 ptos por acotar.
- 0.75 ptos por encontrar n_0.
- 0.75 ptos por encontrar c.

Puntajes parciales y soluciones alternativas a criterio del corrector.