Introducción

En la entrada anterior vimos que en ocasiones los algoritmos pueden fallar. Vimos que incluso aunque un algoritmo resuelva correctamente muchas instancias de un problema, es posible que todavía falle en otras. Ahora hablaremos de qué podemos hacer para que esto no suceda y podamos estar seguros de que nuestro algoritmo funciona en todas las instancias.

Demostraciones de correctitud

Nuestros algoritmos deben estar acompañados de una explicación que justifique que son correctos. Como los modelos que planteamos son matemáticos, dichas explicaciones son usualmente matemáticas.

Ejemplo. Queremos resolver el siguiente problema algorítmico.

Problema. Iremos sumando números impares positivos del más chico al más grande, hasta que nos pasemos por primera vez de un entero dado n. Luego, responderemos cuántos enteros tuvimos que sumar. Por ejemplo, si nos dan 10, entonces debemos responder 4 pues se necesita sumar 4 números impares para pasarse de 10: $1+3+5=9$ todavía no se pasa de 10, pero $1+3+5+7=16$ ya se pasa de 10.

Entrada. Un entero positivo n.

Salida. Un entero que diga cuántos enteros impares positivos del más chico al más grande debemos sumar para poder pasarnos de n.

Podemos resolver el problema mediante el siguiente algoritmo, que explicaremos en pseudocódigo:

```
definimos cuantos_impares para n:
   empezamos una suma en cero
   empezamos al impar en uno
   empezamos un contador en cero
   mientras la suma sea menor o igual que n:
      a la suma le sumamos el impar
      al impar le sumamos dos
      al contador le sumamos uno
   damos el contador
```

En código de Python de la definición, y su ejecución en algunas instancias se vería así:

```
def cuantos_impares(n):
    suma=0
    impar=1
    cont=0
    while suma<=n:
        suma+=impar
        impar+=2
        cont+=1
    print(cont)

cuantos_impares(10)
cuantos_impares(15)
cuantos_impares(4)
cuantos_impares(17)
cuantos_impares(3)
```

```
4
4
3
5
2
```

Hasta aquí parece ser que todo va bien. En efecto, para las instancias que checamos, necesitamos hacer la suma con la cantidad de

impares que se indican para pasarnos del número:

$$\begin{align*} 10 &< 16 = 1 + 3 + 5 + 7\\ 15 &< 16 = 1 + 3 + 5 + 7\\ 4 &< 9 = 1 + 3 + 5\\ 17 &< 25 = 1 + 3 + 5 + 7 + 9\\ 3 &< 4 = 1 + 3, \end{align*}$$

y con menos impares no se pasa, como puedes verificar.

¿Cómo nos aseguramos de que en efecto el algoritmo propuesto da la respuesta correcta siempre? Debemos de dar un argumento, que en nuestro caso podría ser algo como lo siguiente.

«El bucle `while` hará que sigamos sumando números hasta que `suma` sea mayor que n. La variable `impar` guarda el impar que tenemos que sumar, y cada que pasa una ejecución del bucle aumenta en 2, así que se convierte en el siguiente impar. De este modo, en cada ejecución del bucle la variable `suma` aumenta un impar, luego el siguiente, luego el siguiente y así sucesivamente. El contador `cont` empieza en 0 y aumenta en 1 cada vez que se suma un impar, así que al salir del bucle en efecto nos dice cuántos impares sumamos en total.» $\square$

Demostraciones por inducción

En general, podemos usar casi cualquier herramienta matemática para mostrar que un algoritmo es correcto. Sin embargo, una herramienta muy útil para mostrar la correctitud de los algoritmos es la inducción matemática. Veamos un ejemplo.

Problema. Aumentar un entero no negativo en n

Entrada. Un entero no negativo n.

Salida. El entero $n+1$.

Proponemos como algoritmo hacer lo siguiente:

```
MasUno(y):
    si y=0, respondemos 1, si no:
        si y es impar, entonces
            respondemos 2*MasUno(Parte-entera(y/2))
        si y es par, respondemos(y+1)
```

Este algoritmo parece ser algo rebuscado para simplemente sumar uno. Pero la idea por ahora no es dar un algoritmo simple, sino uno que sea correcto. Veamos inductivamente que en efecto siempre estamos respondiendo la respuesta correcta.

Proposición. Para todo valor de n, la función `MasUno` definida arriba cumple que `MasUno(n)=n+1` .

Demostración. Hagamos inducción fuerte en y. Si $y=0$, el algoritmo tiene un caso específico para esto, que es responder 1, lo cual es correcto. Esto prueba el caso base.

Supongamos que el algoritmo funciona para cualquier entero hasta y, y veamos qué sucede con $y+1$. Esto nos lleva a dos casos.

- Caso 1 ($y+1$ es impar). En este caso, tenemos que $y+1$ es impar, digamos que $y+1=2k+1$. Cuando hacemos esto, el algoritmo cae en el caso de entrada impar, de modo que responde

$$\begin{align*} 2\text{MasUno}(\lfloor (y+1)/2\rfloor) &= 2\text{MasUno}(\lfloor(2k+1)/2\rfloor)\\ &= 2\text{MasUno}(k)\\ &=2(k+1)\\ &=2k+2\\ &=y+2. \end{align*}$$

En la tercer igualdad estamos usando la hipótesis inductiva. La respuesta final que obtenemos es correcta.

- Caso 2 ($y+1$ es par). En este caso, el algoritmo cae en el caso de que su entrada es par, y respondería $(y+1)+1=y+2$, que es correcto.

Esto termina la demostración por inducción. $\square$

Demostración por contradicción

Otra buena herramienta para mostrar que los algoritmos son correctos son las pruebas por contradicción. En algunas ocasiones suponer que cierto algoritmo da una respuesta incorrecta nos lleva a una contradicción. Veamos un ejemplo de esto.

Ejemplo. Queremos resolver el siguiente problema algorítmico.

Problema. Estamos en un cine y hay varias películas que queremos ver. Sin embargo, como hay varias salas, las películas pueden empalmarse. Sólo podemos ver una película simultáneamente. ¿Cuál es la mayor cantidad de películas que podemos ver?

Entrada. Una lista de intervalos acotados

\[[a_1,b_1], [a_2,b_2],\ldots,[a_n,b_n],\]
en donde el intervalo \([a_i,b_i]\) corresponde a la película \(i\) y quiere decir que comienza en el tiempo \(a_i\) y termina en el tiempo \(b_i\).

Salida. La máxima cantidad de películas que podemos ver completas, es decir, la máxima cantidad posible de intervalos de entre los que nos dieron, de modo que no haya dos que se traslapen.

El algoritmo que proponemos es el siguiente. Lo que haremos es ver la película que termine primero (con el valor de \(b_i\) mínimo). Luego, de entre las que sobran (las que no se intersectan con la que vimos) veremos la que termine primero. Y así sucesivamente, en cada paso, de entre las películas que no se intersecten con las ya vistas, veremos la que termine primero.

¿Cómo demostramos que este algoritmo en efecto siempre elige la mayor cantidad de películas que podemos ver? Podemos hacer una prueba por contradicción. Supongamos que esta forma \(F\) de ver las películas nos hace ver ciertas películas \(P_1,P_2,\ldots,P_m\) (en ese orden). Supongamos que existe otra forma \(F'\) de ver películas \(Q_1, Q_2,\ldots, Q_M\) (en ese orden). y que son estrictamente más, es decir, con \(M>m\).

La forma \(F\) y la forma \(F'\) podrían empezar viendo las mismas películas (digamos \(P_1=Q_1\), \(P_2=Q_2\), etc.), pero como son formas distintas, en algún momento difieren por primera vez. Digamos que sucede en la \(k\)-ésima película vista.

Así,

\[P_1=Q_1, P_2=Q_2, \ldots, P_{k-1}=Q_{k-1},\]
pero \(P_k\neq Q_k\).

Como la estrategia de \(P\) consiste en ver la película que termine primero en cada paso, sucede que \(Q_k\) termina al mismo tiempo o después que \(P_k\). Inductivamente, se puede ver que \(P_j\) termina antes de toda \(Q_j\) para \(j=k,k+1,\ldots,n\). Así, la estrategia \(F'\) tiene menos chance (o igual) de ver películas que la estrategia \(F\). Esto es una contradicción, pues habíamos dicho que se podían ver estrictamente más películas con \(F'\). Esto muestra que no puede haber una estrategia mejor que la de \(F\) y entonces esta debe ser una estrategia para ver la máxima cantidad posible de películas. \(\square\)

Tarea moral

Los siguientes problemas te ayudarán a practicar lo visto en esta entrada. Para resolverlos, necesitarás usar herramientas matemáticas, computacionales o ambas.

1. Ejecuta manualmente el algoritmo `cuantos_impares` para \(n=18\). Asegúrate de entender cómo van cambiando las variables en cada uno de los pasos.
2. Ejecuta manualmente el algoritmo `max_pelis` para la siguiente instancia con 15 intervalos.
   \[\begin{align*} &[5,20],[2,13],[12,21],[3,11],[9,21],\\ &[8,10],[20,25],[1,30],[3,12],[7,10],\\ &[14,18],[25,28],[30,31],[21,31], [0,12]. \end{align*}\]
   Asegúrate de cómo van cambiando las variables en cada uno de los pasos. Luego, implementa el algoritmo en Python.

3. En la tarea moral anterior diste un algoritmo para dar el índice en donde ocurre el valor mínimo de una lista de enteros \(a_0,a_1,\ldots,a_n\). Da un argumento de que tu algoritmo funciona.
4. Dado un entero positivo \(n\), queremos un algoritmo que nos diga cuánto es la suma de los cubos de los primeros \(n\) enteros positivos. El algoritmo propuesto es responder

   \[\left(\frac{n(n+1)}{2}\right)^2.\]

   ¿Es este algoritmo correcto? En caso de que sí, da una demostración. En caso de que no, da un contraejemplo.

5. Encuentra contraejemplos para las siguientes formas de resolver el problema de las películas:
   - Ver la primera película posible (la que empiece primero), luego la siguiente que podamos, luego la siguiente que podamos y así sucesivamente.
   - Tomar la película más corta de toda la cartelera, luego la siguiente más corta que no se intersecte con ella, luego la siguiente más corta que no se intersecte con estas dos y así sucesivamente. Ver esas películas.

Anterior
Algoritmos incorrectos

Siguiente
Modelo RAM y pensamiento asintótico

Por Leonardo Ignacio Martínez Sandoval
© Copyright 2022.