Correctitud de algoritmos recursivos

Un algoritmo recursivo es aquél que, para calcular la respuesta de una instancia dada, primero resuelve una o más instancias más fáciles del mismo problema y luego usa sus resultados para calcular la respuesta de la instancia original. Plantearemos un algoritmo recursivo para exponenciación de naturales.

Supongamos que queremos calcular 2^{20}. Por supuesto, una forma de hacerlo es multiplicar 2 veinte veces por sí mismo, pero hay una alternativa. ¿Qué pasa si, de alguna forma, ya supieramos cuánto vale $t = 2^{10}$? En este caso, el cálculo de 2^{20} se vuelve simplemente $t \times t$ ya que $2^{20} = 2^{10} \times 2^{10}$. Nótese cómo estamos resolviendo una instancia de exponenciación de naturales, *usando otra instancia de exponenciación de naturales*, aunque la otra instancia se intuye que es más fácil porque el exponente es menor. Para calcular 2^{10} que aún no sabemos cuánto vale, podemos aplicar la misma idea: ¿qué pasa si, de alguna forma, ya supieramos cuánto vale $t' = 2^5$? entonces 2^{10} es simplemente $t' \times t'$.

Si queremos volver a aplicar la idea de calcular a^b usando $a^{\frac{b}{2}}$, debemos tener cuidado ahora al calcular 2^5. Es cierto que $2^5 = 2^{2.5} \times 2^{2.5}$, pero $2^{2.5}$ no es una instancia del problema de exponenciación de naturales y termina empeorando la situación. En general, es mejor auxiliarnos de calcular $a^{\lfloor \frac{b}{2} \rfloor}$. Cuando el exponente es impar como en 2^5, ahora tenemos $t'' = 2^{\lfloor 2.5 \rfloor} = 2^2$ y hay que observar que $2^5 = t'' \times t'' \times 2$. En un planteamiento recursivo correcto, eventualmente aparecen instancias que son tan fáciles de resolver que podemos (y debemos) resolverlas directamente. Por ejemplo, $2^0 = 1$ y no tiene caso aplicar recursión. A estas instancias se les denomina casos base. El algoritmo recursivo para exponenciación natural es:

---

subrutina ExponenciaciónNatural(natural a, natural b)
  si $b = 0$
    regresa 1
  sino
    $t \leftarrow$ ExponenciaciónNatural$(a, \lfloor \frac{b}{2} \rfloor)$
    si t es par
      regresa $t \times t$
    sino
      regresa $t \times t \times a$

Aunque la intuición nos diga que el algoritmo anterior está bien, lo demostraremos formalmente. La demostración de correctitud de algoritmos recursivos suele ser sencilla porque es una aplicación directa de inducción matemática. En inducción matemática, partimos de un primer caso que es correcto y luego demostramos simbólicamente que los siguientes casos también lo son. Dado que en la función anterior el único valor que cambia durante la recursión es el exponente b, haremos inducción sobre b. Por simplicidad notacional, en la demostración nos referiremos a la función de exponenciación natural como f.

Demostración. Primero probaremos que $f(a, 0)$ es correcto. Sabemos que a^0 es 1 y vemos que el algoritmo regresa 1 cuando $b = 0$, por lo que el caso base es correcto. Ahora supondremos

que $f(a, b)$ es correcto para toda $0 \leq b' < b$ y demostraremos que *f(a, b)* es correcto. Dado que el algoritmo se comporta de forma distinta cuando b es par que cuando no lo es, necesitamos manejar dos casos:

- Si b es par, entonces $b = 2k$. Dado que $b > 0$ entonces $0 \leq k < b$. El exponente que usamos en la recursión es $\lfloor \frac{b}{2} \rfloor = \lfloor \frac{2k}{2} \rfloor = k$. Por la hipótesis de inducción, $t = f(a, k) = a^k$. El algoritmo regresa $t \times t = a^k \times a^k = a^{2k} = a^b$, por lo que el algoritmo es correcto en este caso.

- Si b es impar, entonces $b = 2k + 1$. Dado que $b > 0$ entonces $0 \leq k < b$. El exponente que usamos en la recursión es $\lfloor \frac{b}{2} \rfloor = \lfloor \frac{2k+1}{2} \rfloor = k$. Por la hipótesis de inducción, $t = f(a, k) = a^k$. El algoritmo regresa $t \times t \times a = a^k \times a^k \times a = a^{2k+1} = a^b$, por lo que el algoritmo es correcto en este caso.

□

Otro ejemplo que trabajaremos es el del coeficiente binomial. El coeficiente binomial $\binom{n}{k}$ es la cantidad de combinaciones que se pueden formar escogiendo k objetos de los n disponibles. El coeficiente binomial está definido para enteros $0 \leq k \leq n$ y la fórmula más conocida es $\frac{n!}{k!(n-k)!}$. Sin embargo, nosotros presentaremos un planteamiento recursivo para calcular $\binom{n}{k}$ y demostraremos que es correcto. Al igual que el ejemplo anterior, en la demostración denotaremos como f a la función recursiva propuesta.

---

subrutina CoeficienteBinomial(natural n, natural k)
  si $k = 0$ o $k = n$
    regresa 1
  sino
    regresa CoeficienteBinomial$(n - 1, k)$ + CoeficienteBinomial$(n - 1, k - 1)$

Demostración. Primero probaremos que $f(n, 0)$ y $f(n, n)$ son correctos. Cuando $k = 0$ entonces $\frac{n!}{k!(n-k)!} = \frac{n!}{0!(n)!}$ que es igual a 1. Cuando $k = n$ entonces $\frac{n!}{k!(n-k)!} = \frac{n!}{n!(0)!}$ que es igual a 1. Como el algoritmo regresa 1 en ambos casos, los casos bases de algoritmo recursivo son correctos.

Ahora supondremos que $f(n', k')$ es correcto para toda $0 \leq k' \leq n' < n$ y demostraremos que *f(n, k)* es correcto. Los casos de $k = 0$ y $k = n$ están cubiertos por los casos bases, por lo que falta demostrar el caso donde $0 < k < n$. Si $k < n$ entonces $k \leq n'$ por la hipótesis de inducción. También por la hipótesis de inducción tenemos lo siguiente:

$$f(n-1, k) = \frac{(n-1)!}{k!((n-1)-k)!} = \frac{(n-1)!}{k!(n-k-1)!}$$

$$f(n-1, k-1) = \frac{(n-1)!}{(k-1)!((n-1)-(k-1))!} = \frac{(n-1)!}{(k-1)!(n-k)!}$$

La suma de las dos expresiones es

$$\begin{aligned} f(n-1,k)+f(n-1,k-1) &= \frac{(n-1)!}{k!(n-k-1)!}+\frac{(n-1)!}{(k-1)!(n-k)!} \\ &= \frac{(n-1)!}{(k-1)!(n-k-1)!}\left(\frac{1}{k}+\frac{1}{n-k}\right) \\ &= \frac{(n-1)!}{(k-1)!(n-k-1)!}\left(\frac{(n-k)+(k)}{k(n-k)}\right) \\ &= \frac{(n-1)!}{(k-1)!(n-k-1)!}\left(\frac{n}{k(n-k)}\right) \\ &= \frac{n!}{k!(n-k)!} \end{aligned}$$

por lo que $f(n-1,k)+f(n-1,k-1)=\binom{n}{k}$, lo que concluye la demostración. □

Usareos inducción matemática repetidamente durante el curso. Su aplicación en la demostración de correctitud de algoritmos recursivos es prácticamente directa. Sin embargo, demostrar la correctitud de algoritmos iterativos es más complicado.

Correctitud de algoritmos iterativos

Un algoritmo iterativo es aquél que usa ciclos. El gran inconveniente que existe al demostrar la correctitud de algoritmos iterativos es que las variables pueden cambiar su valor conforme se itera. Esto nos obliga a hacer las siguientes preguntas:

- ¿Cuánto valen las variables antes de examinar la condición del ciclo por primera vez?
- ¿Cómo cambian los valores de las variables cuando pasamos de una iteración a otra?
- ¿Cuánto valen las variables al terminar el ciclo?

Si intentamos responder estas preguntas, podremos proponer las propiedades algorítmicas que se cumplen durante la ejecución del ciclo, las cuales se denominan invariantes. Las invariantes que propongamos deben servirnos para demostrar la correctitud del algoritmo y las demostraremos a su vez mediante expresiones matemáticas que estén respaldadas por lo que el algoritmo hace.

Nos limitaremos a demostrar la correctitud de algoritmos con sólo un ciclo *mientras*, donde el ciclo debe acabar normalmente mediante la condición de control y donde además casi todo el trabajo del algoritmo se realiza dentro del ciclo. Si necesitáramos demostrar la correctitud de algoritmos con más de un ciclo o con ciclos anidados, lo más seguro es que necesitemos realizar múltiples demostraciones y luego usar sus resultados como partes de una demostración mayor.

Consideremos una variable x que está declarada antes del ciclo y que potencialmente cambia su valor durante el ciclo. Observemos que si el ciclo realiza n iteraciones, entonces la condición del ciclo se evalua $n+1$ veces y es falsa durante la última evaluación. Denotaremos como x_t al valor de x donde la t-ésima evaluación de la condición del ciclo, comenzando por $t = 0$. Esto quiere decir que x_0 corresponde con el valor de x al examinar la condición del ciclo la primera vez y x_n es el valor de x cuando termina el ciclo. La demostración de las invariantes que propongamos la haremos por inducción sobre t.

Primero probaremos que el siguiente algoritmo regresa $a - b$ para $a, b \geq 0$.

---

subrutina RestaNatural(natural a, natural b)
  mientras $b > 0$
    $a \leftarrow a - 1$
    $b \leftarrow b - 1$
  regresa a

La intuición nos dice que a se decrementa iteración tras iteración hasta alcanzar el valor de la respuesta. La cantidad de decrementos en a debe depender de la cantidad de iteraciones hechas. La intuición también nos dice que lo que evita que a se decremente de más es que b eventualmente llega a 0 y detiene el ciclo.

Demostración. Propondremos las invariantes $a_t = a - t$ y $b_t = b - t$. Sabemos que $a_0 = a$, $b_0 = b$ y verificamos las invariantes $a_0 = a - 0 = a$ y $b_0 = b - 0 = b$, por lo que éstas son correctas inicialmente. Ahora supondremos que las invariantes son correctas para $0 \leq t - 1$ y demostraremos que también lo son para t (esto implica que la condición del ciclo fue cierta en $t-1$). Por la hipótesis de inducción, $a_{t-1} = a-(t-1) = a-t+1$ y $b_{t-1} = b-(t-1) = b-t+1$. Los pasos realizados dentro del ciclo son $a_t = a_{t-1} - 1$ y $b_t = b_{t-1} - 1$. Desarrollando tenemos $a_t = a - t + 1 - 1 = a - t$ y $b_t = b - t + 1 - 1 = b - t$, por lo que las invariantes son correctas. El valor de b_t se decrementa de uno en uno ya que $b_t - b_{t-1} = (b - t) - (b - (t - 1)) = -1$, por lo que el ciclo termina cuando $b_{t'} = 0$. Por la invariante tenemos que $t' = b$. El algoritmo regresa $a_b = a - b$ por lo que es correcto. □

Ahora demostraremos que el siguiente algoritmo regresa $\sum_{i=1}^{n} i$ para $n \geq 0$.

---

subrutina Sumatoria(natural n)
  $r \leftarrow 0$, $k \leftarrow 1$
  mientras $k \leq n$
    $r \leftarrow r + k$
    $k \leftarrow k + 1$
  regresa r

La intuición nos dice que k se incrementa iteración tras iteración hasta que el ciclo se detiene, por lo que ésta debe jugar el rol de i en $\sum_{i=1}^{n} i$. Con respecto a r que aumenta en k

en cada iteración, debe ser cierto que el primer valor que sumamos es $k = 1$ y el último es $k = n$.

Demostración. Propondremos las invariantes $r_t = \sum_{i=1}^{t} i$ y $k_t = t + 1$. Sabemos que $r_0 = 0$, $k_0 = 1$ y verificamos las invariantes $r_0 = \sum_{i=1}^{0} i = 0$ y $k_0 = 0 + 1 = 1$, por lo que éstas son correctas inicialmente. Ahora supondremos que las invariantes son correctas para $0 \leq t - 1$ y demostraremos que también lo son para t (esto implica que la condición del ciclo fue cierta en $t - 1$). Por la hipótesis de inducción, $r_{t-1} = \sum_{i=1}^{t-1} i$ y $k_{t-1} = (t - 1) + 1 = t$. Los pasos realizados dentro del ciclo son $r_t = r_{t-1} + k_{t-1}$ y $k_t = k_{t-1} + 1$. Desarrollando tenemos $r_t = \sum_{i=1}^{t-1} i + t = \sum_{i=1}^{t} i$ y $k_t = t + 1$, por lo que las invariantes son correctas. El valor de k_t se incrementa de uno en uno ya que $k_t - k_{t-1} = t + 1 - t = 1$, por lo que el ciclo termina cuando $k_{t'} = n + 1$. Por la invariante tenemos que $t' = n$. El algoritmo regresa $r_n = \sum_{i=1}^{n} i$. □