Análisis de caso promedio

Parte I

Segundo semestre 2022

IIC2283

Prof. Nicolás Van Sint Jan

Antes: Terminemos con el test de primalidad

Vamos a diseñar un test de primalidad considerando los conjuntos:

$$\begin{aligned} S_n^+ &= \{a \in \mathbb{Z}_n^* \mid a^{\frac{n-1}{2}} \equiv 1 \pmod{n}\} \\ S_n^- &= \{a \in \mathbb{Z}_n^* \mid a^{\frac{n-1}{2}} \equiv -1 \pmod{n}\} \end{aligned}$$

Así, podemos definir S_n a partir de estos conjuntos:

$$S_n = S_n^+ \cup S_n^-$$

Para hacer esto necesitamos estudiar algunas propiedades de los conjuntos S_n^+, S_n^- y S_n.

- Consideramos primero el caso en que n es primo, y luego el caso en que n es compuesto

Recordatorio: caracterizando S_n, S_n^+ y S_n^-

Proposición 1

Si $n \geq 3$ es primo, entonces $S_n = \mathbb{Z}_n^*$.

Proposición 2

Si $n \geq 3$ es primo: $|S_n^+| = |S_n^-| = \frac{n-1}{2}$

Teorema

Sea $n = n_1 \cdot n_2$, donde $n_1, n_2 \geq 3$ y $\gcd(n_1, n_2) = 1$. Si existe $a \in \mathbb{Z}_n^*$ tal que $a^{\frac{n-1}{2}} \equiv -1 \pmod{n}$, entonces:

$$|S_n| \leq \frac{1}{2} \cdot |\mathbb{Z}_n^*|$$

Recordatorio: Un test de primalidad aleatorizado

TestPrimalidad(n, k)
  if $n = 2$ **then return** PRIMO
  else if n es par **then return** COMPUESTO
  else if **EsPotencia**(n) **then return** COMPUESTO
  else
    sea $a_1, \ldots, a_k$ una secuencia de números elegidos de manera uniforme e independiente desde $\{1, \ldots, n-1\}$
    for $i := 1$ **to** k **do**
      if **MCD**$(a_i, n) > 1$ **then return** COMPUESTO
      else $b_i :=$ **EXP**$(a_i, \frac{n-1}{2}, n)$
    $neg := 0$
    for $i := 1$ **to** k **do**
      if $b_i \equiv -1 \bmod n$ **then** $neg := neg + 1$
      else if $b_i \not\equiv 1 \bmod n$ **then return** COMPUESTO
    if $neg = 0$ **then return** COMPUESTO
    else return PRIMO

Outline

Test de primalidad: tercera versión (final)

Análisis de caso promedio: Quicksort

Test de primalidad: probabilidad de error

TestPrimalidad se puede equivocar de dos formas:

- Suponga que $n \geq 3$ es primo. En este caso **TestPrimalidad** da una respuesta incorrecta si $b_i \equiv 1 \bmod n$ para todo $i \in \{1, \ldots, k\}$

  Dado que $|S_n^+| = |S_n^-| = \frac{n-1}{2}$:

  - La probabilidad de que para un número a elegido con distribución uniforme desde $\{1, \ldots, n-1\}$ se tenga que $a^{\frac{n-1}{2}} \equiv 1 \bmod n$ es $\frac{1}{2}$

  Por lo tanto, la probabilidad de que **TestPrimalidad** diga COMPUESTO para $n \geq 3$ primo es $\left(\frac{1}{2}\right)^k$

Test de primalidad: probabilidad de error

- Suponga que n es compuesto, n es impar y n no es de la forma m^ℓ con $\ell \geq 2$
  - Si n es par o n es de la forma m^ℓ con $\ell \geq 2$, entonces **TestPrimalidad** da la respuesta correcta COMPUESTO

Tenemos entonces que $n = n_1 \cdot n_2$ con $n_1 \geq 3$, $n_2 \geq 3$ y $\gcd(n_1, n_2) = 1$

Además debe existir $a \in \{1, \ldots, n-1\}$ tal que $\gcd(a, n) = 1$ y $a^{\frac{n-1}{2}} \equiv -1 \bmod n$

  - Si esto no es cierto **TestPrimalidad** retorna COMPUESTO, dado que si **TestPrimalidad** logra llegar a la última instrucción **if** entonces *neg* necesariamente es igual a 0

Test de primalidad: probabilidad de error

Concluimos que $|S_n| \leq \frac{1}{2} \cdot |\mathbb{Z}_n^*|$

- Por la caracterización que dimos de S_n para n compuesto

Vamos a utilizar este resultado para acotar la probabilidad de error:

$$\Pr\left(\left(\bigwedge_{i=1}^{k} \gcd(a_i, n) = 1 \wedge (b_i \equiv 1 \bmod n \vee b_i \equiv -1 \bmod n)\right) \wedge \left(\bigvee_{j=1}^{k} b_j \equiv -1 \bmod n\right)\right)$$

Test de primalidad: probabilidad de error

Tenemos que:

$$\Pr\left(\left(\bigwedge_{i=1}^{k} \gcd(a_i, n) = 1 \wedge (b_i \equiv 1 \bmod n \vee b_i \equiv -1 \bmod n)\right) \wedge \left(\bigvee_{j=1}^{k} b_j \equiv -1 \bmod n\right)\right) \leq$$

$$\Pr\left(\bigwedge_{i=1}^{k} \gcd(a_i, n) = 1 \wedge (b_i \equiv 1 \bmod n \vee b_i \equiv -1 \bmod n)\right)$$

Por lo tanto sólo necesitamos una cota superior para la última expresión.

Test de primalidad: probabilidad de error

Tenemos que:

$$\Pr\left(\bigwedge_{i=1}^{k} \gcd(a_i, n) = 1 \wedge (b_i \equiv 1 \bmod n \vee b_i \equiv -1 \bmod n)\right)$$

$$= \prod_{i=1}^{k} \Pr(\gcd(a_i, n) = 1 \wedge (b_i \equiv 1 \bmod n \vee b_i \equiv -1 \bmod n))$$

$$= \prod_{i=1}^{k} \Pr((b_i \equiv 1 \bmod n \vee b_i \equiv -1 \bmod n) \mid \gcd(a_i, n) = 1) \cdot$$

$$\Pr(\gcd(a_i, n) = 1)$$

$$\leq \prod_{i=1}^{k} \Pr((b_i \equiv 1 \bmod n \vee b_i \equiv -1 \bmod n) \mid \gcd(a_i, n) = 1)$$

$$= \prod_{i=1}^{k} \Pr(a_i \in S_n \mid a_i \in \mathbb{Z}_n^*) \leq \prod_{i=1}^{k} \frac{1}{2} = \frac{1}{2^k}$$

Test de primalidad: probabilidad de error

Concluimos que la probabilidad de que el test diga PRIMO para el valor compuesto n está acotada por $\left(\frac{1}{2}\right)^k$

En ambos casos (si n es primo o compuesto) la probabilidad de error del algoritmo está acotada por $\left(\frac{1}{2}\right)^k$

- ¡Si $k = 100$, está probabilidad está acotada por $\left(\frac{1}{2}\right)^{100} \approx 7.9 \times 10^{-31}$!

Outline

Analizando la complejidad de un algoritmo

Hasta ahora sólo hemos analizado la complejidad de un algoritmo considerando el **peor caso**.

También tiene sentido estudiar la complejidad del algoritmo $\mathcal{A}$ en el **caso promedio**, el cual está dado por la siguiente expresión:

$$\frac{1}{k^n} \cdot \left(\sum_{w \in \Sigma^n} \text{tiempo}_{\mathcal{A}}(w) \right)$$

donde $k = |\Sigma|$ y $\Sigma^n = \{w \in \Sigma^* \mid |w| = n\}$

El caso promedio de un algoritmo

La definición anterior asume que todas las entradas posibles son **equiprobables** (i.e. distribuyen uniforme).

- Esto podría no reflejar la distribución de las entradas en la práctica

Para solucionar este problema podemos usar otras distribuciones de probabilidad.

El caso promedio de un algoritmo

Suponemos que para cada $n \in \mathbb{N}$ hay una distribución de probabilidades:

$\mathbf{Pr}_n(w)$ es la probabilidad de que $w \in \Sigma^n$ aparezca como entrada de $\mathcal{A}$

Nótese que $\sum_{w \in \Sigma^n} \mathrm{Pr}_n(w) = 1$

Ejemplo

Para la definición de caso promedio en las transparencias anteriores tenemos que $\mathrm{Pr}_n(w) = \frac{1}{k^n}$ con $k = |\Sigma|$

El caso promedio de un algoritmo

Para definir el caso promedio, para cada $n \in \mathbb{N}$ usamos una **variable aleatoria** X_n tal que para cada $w \in \Sigma^n$ se tiene que

$$X_n(w) = tiempo_{\mathcal{A}}(w)$$

Para las entradas de largo n, el número de pasos de $\mathcal{A}$ en el caso promedio es el **valor esperado** de la variable aleatoria X_n:

$$E(X_n) \quad = \quad \sum_{w \in \Sigma^n} X_n(w) \cdot \Pr_n(w)$$

Definición: Complejidad en el caso promedio

Decimos que $\mathcal{A}$ en el **caso promedio** es $O(f(n))$ si $E(X_n) \in O(f(n))$

Sobre las definiciones de peor caso y caso promedio

Notación

Las definiciones de peor caso y caso promedio pueden ser modificadas para considerar las notaciones Θ y Ω

- Simplemente reemplazando $O(f(n))$ por $\Theta(f(n))$ u $\Omega(f(n))$, respectivamente

Por ejemplo, decimos que $\mathcal{A}$ en el caso promedio es $\Theta(f(n))$ si $E(X_n) \in \Theta(f(n))$

Un ejemplo: el algoritmo de ordenación Quicksort

Quicksort es un algoritmo de ordenación muy utilizado en la práctica.

La función clave para la definición de Quicksort:

Partición(L, m, n)
    $pivote := L[m]$
    $i := m$
    for $j := m + 1$ **to** n **do**
        if $L[j] \leq pivote$ **then**
            $i := i + 1$
            intercambiar $L[i]$ con $L[j]$
    intercambiar $L[m]$ con $L[i]$
    return i

Nótese que en la definición de **Partición** la lista L es **pasado por referencia**.

Un ejemplo: el algoritmo de ordenación Quicksort

La definición de Quicksort:

Quicksort$(L,\ m,\ n)$
    if $m < n$ **then**
        $\ell :=$ **Partición**$(L,\ m,\ n)$
        Quicksort$(L,\ m,\ \ell - 1)$
        Quicksort$(L,\ \ell + 1,\ n)$

Vamos a contar el **número de comparaciones** al medir la complejidad de **Quicksort**.

¿ Es esta una **buena** medida de complejidad ?

¿Cuál es la complejidad de Quicksort?

Si **Partición** siempre divide a una lista en dos listas del mismo tamaño, entonces la complejidad de **Quicksort** estaría dada por la siguiente ecuación de recurrencia:

$$T(n) = \begin{cases} 1 & n = 0 \\ 2 \cdot T(\lfloor \frac{n}{2} \rfloor) + c \cdot n & n \geq 1 \end{cases}$$

Dado que $c \cdot n \in \Theta(n^{\log_2(2)})$, concluimos usando el Teorema Maestro que

$$T(n) \in \Theta(n \cdot \log_2(n))$$

¿Qué nos asegura que **Partición** divide a una lista en dos listas del mismo tamaño?

¿Cuál es la complejidad de Quicksort?

Ejercicio

1. Dada una lista con n elementos, ¿cuál es el peor caso para **Quicksort**?
2. Demuestre que **Quicksort** en el peor caso es $\Theta(n^2)$