Resumen C3

Alg. determinísticos: hacen siempre lo mismo

Alg. aleatorizado. ~. Incorporan componente al azar

↳ no existen malos inputs, sino malas ejecuciones.

MONTECARLO:

- se equivoca con cierta probabilidad
- no siempre sabe si falló
- se puede iterar para reducir prob. de error.
- one/two side error

// pescar pez grande (mayor que la mediana) / consistencia base datos (con hash)

LAS VEGAS

- responde correctamente
- no tiene tpo. de terminación garantizado
- se habla de tiempo esperado

// acceso a Ethernet

* Caso promedio: se supone alguna condición sobre el input

* Tiempo esperado: para el peor input posible...

Complejidad Computacional

ZPP: tpo esperado polinomial sin error (Las Vegas)

RP y coRP: MonteCarlo one-sided (tpo polinomial)

BPP: Montecarlo two-sided

* Y se cree que $P = BPP$

Alg. Primalidad: Miller Rabin

MonteCarlo One-sided en decir que es primo, con prob. error $= 1/4^k$ (haciendo $O(k \cdot \log n)$ multiplicaciones).

IDEA: intentar k veces que el nro. es compuesto. k veces intentar encontrar "a" un testigo de que n es compuesto.

* $Pr(a \text{ es testigo}) = 3/4$

$a \equiv a \pmod n$

$a \equiv b \pmod n \Rightarrow b \equiv a \pmod n$

$\lambda a \equiv \lambda b \pmod n$

Fermat: $a \in \mathbb{N}$ y p primo que no $\div$ a

$\Rightarrow p$ debe ser un factor de $a^{p-1} - 1$

Primo(n): $O(k \log n)$ multiplicaciones

- sean s, d t.q. $n - 1 = 2^s \cdot d$, d impar $O(\log n)$
- repetir k veces:
  - escoger $a \in [1, \ldots, n-1]$ al azar
  - si $a^d \not\equiv 1 \bmod n$ y $\forall r \in [0 \ldots s-1]$ $a^{2^r \cdot d} \not\equiv -1 \bmod n$: $O(\log n)$

    return "compuesto"

  return "primo"

$Pr(\text{error}) = 1/4^k$

↳ **Generar nro primo (de l bits):**

$p \leftarrow$ al azar entre $[2^{l-1}, \ldots, 2^l - 1]$

si primo(p): return p * Las Vegas, pero se puede equivocar.

Árboles Aleatorizados: ABB

Se distribuye como si sus elementos se hubiesen insertado en algún orden aleatorio con distribución uniforme.

$\exists$ árbol binario T con los elementos $x_1, \ldots, x_n$ (en ese orden).

1) **Inserción**: se puede insertar en c/posición con prob. $\frac{1}{n+1}$.

insertar(x, T):

$|T| = 0$, return (x)

elegir r al azar entre $[1, \ldots, |T| + 1]$

si $r = 1$, convertir x en raíz T (* operación cut)

si no

↳ $x < x_1$

$T' \leftarrow$ insertar(x, T_L)

return

↳ $x \geq x_1$

$T' \leftarrow$ insertar(x, T_R)

return

Scanned with CamScanner

cut(x,T): # elementos menores y mayores a x.

si $|T|=0$, return (NULL, NULL)

T.raiz < x:

$(T_<, T_>) \leftarrow$ cut(T.derecho, x)

$T_L \leftarrow$ <T.raiz, T.izquierdo, $T_<$>

$T_R \leftarrow T_>$

si no:

$(T_<, T_>) \leftarrow$ cut(T.izquierdo, x)

$T_R \leftarrow$ <T.raiz, T.derecho, $T_>$>

$T_L \leftarrow T_<$

return (T_L, T_R)

2) **Borrado**. buscar x y luego borrarlo

¿Qué se hace con la raíz vacía?

Se elige con prob. $\frac{|T_L|}{|T_L+T_R|}$ que la raíz de T_L llegó primero

borrar(x,T):

si $|T|=1$, return NULL

elegir r al azar en $[1 \dots |T|-1]$

si $r \leq |T.izq|$:

$I \leftarrow$ T.izq

T.raiz $\leftarrow$ I.raiz

T.izq $\leftarrow$ I.izq

T.derecho $\leftarrow$ merge(<–, I.derecho, T.derecho>)

si no:

$D \leftarrow$ T.der

T.raiz $\leftarrow$ D.raiz

T.der $\leftarrow$ D.der

T.izq $\leftarrow$ merge(<–, T.izq, D.izq>)

Búsqueda, inserción y borrado en tiempo $O(\log n)$ esperado (independiente del orden de inserción)

* Hay más permutaciones que nos dejan en árbol balanceado.

Hashing universal

S: conjunto de datos, $|S|=n$

$h: \mathcal{X} \longrightarrow [0 \dots m-1]$

Idea: el costo sea $O(1)$

Problema: colisiones $\Leftrightarrow h(x)=h(y)$

↳ **familia universal**: $h \in \mathcal{H}$ sii:

$Pr(h(x)=h(y)) \leq \frac{1}{m}$, $x \neq y$

$C_{x,y} = \begin{cases} 1 & h(x)=h(y) \\ 0 & \sim \end{cases}$

$C_{x,S} = \sum_{y \in S} C_{x,y}$: # elementos que colisionan con x.

$\{E(X) = \sum x_i \cdot P(x_i)\}$

$E(C_{x,S})$ = costo esperado de inserción, búsq, etc

$O(\frac{n}{m}) \Rightarrow$ si $n=m$, $O(1)$.

* $\mathcal{H} = \{h_{a,b},\ a \in [1 \dots p-1],\ b \in [0 \dots p-1]\}$?

$h_{a,b}(x) = ((ax+b) \bmod p) \bmod m$

p primo $\Rightarrow \exists!$ inverso multiplicativo

$\exists\ |\mathcal{H}| = p(p-1)$ posibles funciones $h_{a,b}$.

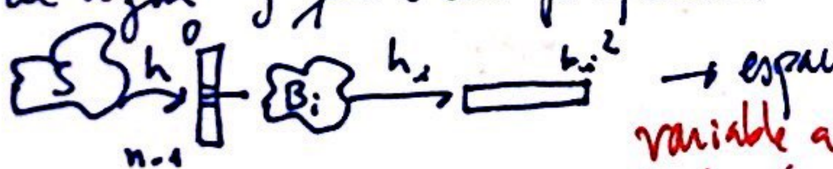
¿$Pr(C_{x,y}=1)$? $Pr((ax+b) \equiv r \bmod p \ \wedge\ (ay+b \equiv s \bmod p)) = \frac{1}{p(p-1)}$

$\exists$ p formas de elegir r y $\lceil \frac{p}{m} \rceil - 1$ para s.

$Pr(C_{x,y}=1] \leq \frac{p(p-1)}{m} \cdot \frac{1}{p(p-1)} = \frac{1}{m}$

Hashing Perfecto

Conocer conjunto S fijo de antemano.

- Si $m \geq n^2$, $\exists$ Pr. $= \frac{1}{2}$ de escoger una $h \in \mathcal{H}$ al azar y que sea perfecta. → mucho espacio $O(n^2)$
- 

→ espacio: $X = \sum b_i^2$ variable aleatoria que depende de la función h.

$\sum b_i^2 = \sum_{x,y \in S} C_{x,y}$

$E(X) = E(\sum C_{x,y}) = \sum_{x \in S} E(C_{x,x}) + \sum_{x \neq y} E(C_{x,y}) \leq n + (n-1)\frac{n}{n} < 2n.$

$\Rightarrow E(X) < 2n \rightarrow$ espacio $O(n)$.