

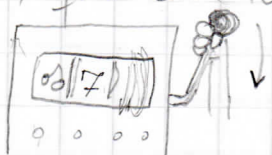
2015年11月17日 (火)

ALGORITMOS ALEATORIZADOS Y PROBABILÍSTICOS

(No determinísticos)

- pueden NO siempre terminar } probabilístico (Las Vegas)
- pueden equivocarse } probabilístico (Montecarlo)
- pueden hacer distintas cosas } aleatorizado.
frente al mismo input.

Existen algoritmos que son probabilísticos y aleatorizados a la vez. Los que son solo probabilísticos, al equivocarse con cierto input, lo hacen consistentemente siempre. Es mejor en ese caso, que sea también aleatorizado.



¡puedo estar todo el día en un casino en Las Vegas!

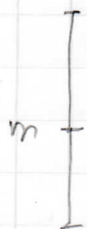
Ej: Sacar un pez "grande" del lago
"grande" $\equiv \gg$ mediana

$\frac{1}{2}$ grande
 $\frac{1}{2}$ chico

$$1 - \frac{1}{2^k}$$

max(k peces)

No es grande
con prob. $\frac{1}{2^k}$



en tiempo
fijo

¿Cómo sé que me equivoco?

Puedo implementar un algoritmo
que responde correctamente con

alta probabilidad, pero no puedo siempre
saber si efectivamente me equivoqué

o no

→ MonteCarlo

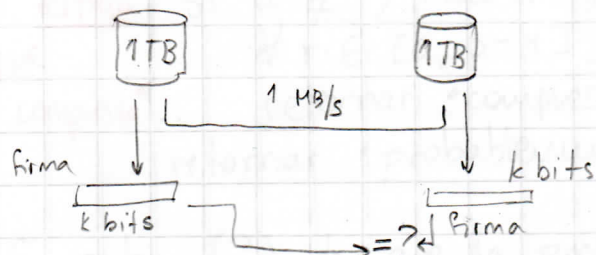
Ahora, qué pasa si sabemos que la mediana mide 1 metro?
Sacamos peces hasta sacar uno grande. Problema! puedo sacar
más peces de lo que aguanta mi bote hasta que salga el
pez que quiero. Puedo demorarme demasiado...

→ Las Vegas.

- Monte Carlo: tiempo fijo ○
 prob. error conocida ○
 1 sided / 2 sided errors X prob equivocarse en decir YES siempre
prob equivocarse en YES o No

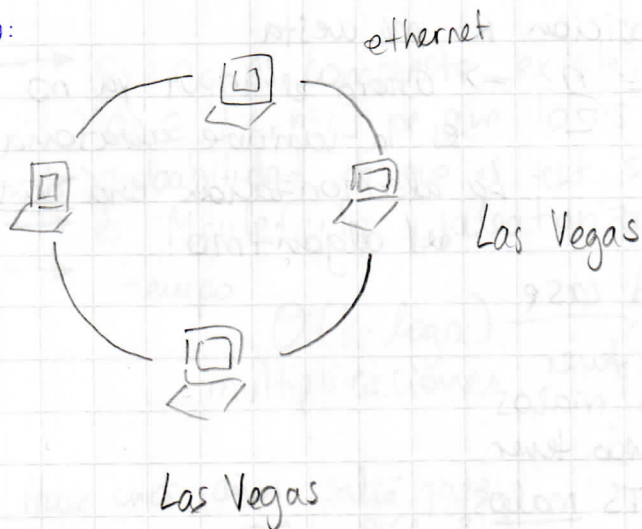
- Las Vegas: No hay error ○
 tiempo esperado se puede "acotar" ○
 peor caso NO se puede acotar X

Ej: consistencia de índices de BDs.

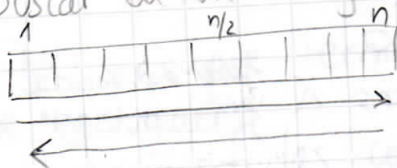


$P = \frac{1}{2^k}$ de que las firmas sean iguales (tomando 2 BDs al azar) ← SIEMPRE!

Otro:



Otro: Buscar en un arreglo desordenado



del primero al último ¿y si son noticias?
del último al primero ¿y si es un Paper?

+ importante al último
+ importante primero

Dependo de la distribución de la entrada si mi algoritmo es determinístico.
⇒ El costo promedio varía

- Opción aleatorizada:
 - tirar una moneda
 - si sale cara
buscar $1 \rightarrow n$
 - si sale sello
buscar $n \rightarrow 1$

Si lo que busco está en la posición k me cuesta

$$\frac{1}{2} \cdot k + \frac{1}{2} (n - k) = \frac{n}{2}$$

⇒ ahora el input ya no es la variable aleatoria.

La aleatorización está dentro del algoritmo.

average case \neq expected case

no me importa el input.

El costo promedio después de suficientes ejecuciones tenderá a lo mismo siempre.

puedo tener casos malos
pues puedo tener INPUTS malos

Otros: primalidad, árboles binarios aleatorizados, hashing universal y perfecto, skiplists.

Primalidad Miller-Rabin

primo(n)

Sean s, d t.q.

$$n-1 \equiv 2^s \cdot d, \quad d \text{ impar.}$$

repetir k veces {

elegir $a \in [1, n-1]$ al azar

"a es testigo
de que
n es compuesto"

si $a^d \not\equiv 1 \pmod n$ y
 $\nexists r \in [0, s-1], a^{2^r \cdot d} \not\equiv -1 \pmod n$ (*)
retornar "compuesto" (**)

retornar "probablemente compuesto"

(Se sabe (?) de que la probabilidad de que este algoritmo se equivoque es de $1/4$)

→ si n es compuesto, existen al menos $\frac{3}{4}n$ testigos $a \in [1, n-1]$ de que lo es

→ probabilidad de que el test se equivoque es $1/4^k$

→ es Monte Carlo y aleatorizado, es 1 sided

→ tiempo

$\Theta(k \cdot \log n)$
multiplicaciones

$$(*) \quad \Theta(\log n) \rightarrow \begin{matrix} a \\ a^2 \\ a^4 \\ a^8 \\ a^{16} \\ \vdots \end{matrix} \quad r=0$$

◆ Hace unos años salió paper

"Primes is in P", $\Theta(\log^3 n)$
multiplicaciones

$$\Rightarrow k \sim \log^7 n$$

pero n es gigantesco

$$(**) \quad \Theta(\log n) \left\{ \begin{matrix} \rightarrow (a^d)^2 \\ r=2 \\ \rightarrow ((a^d)^2)^2 \end{matrix} \right. \quad r=1, 2$$

Signe siendo preferible y menos costoso Miller-Rabin. Para k no muy grande, P de error es muy baja!

Para mismo input input pedo tener árboles distintos secuencia.

Árboles Aleatorizados Binarios

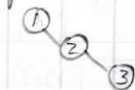
2015年11月19日(木)

$S_1 = 6$, depende del orden en que llegan las claves

1 2, 3

1 3 2

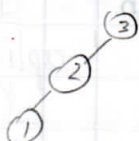
$\begin{cases} 2 3 1 \\ 2 1 3 \end{cases}$



3 1 2



3 2 1



$Cut(T, k)$ devuelve dos árboles con las claves $< k$ y $> k$

Si $T = \square$ retornar (\square, \square)

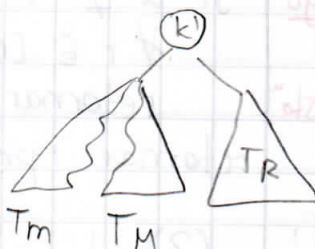
Sea $T =$



Si $k < k'$

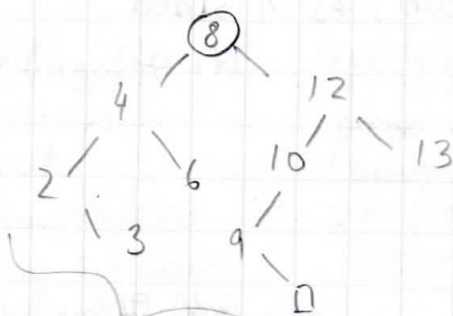
$(T_m, T_M) \leftarrow Cut(T_L, k)$

return $(T_m, \begin{matrix} k' \\ T_M \end{matrix} \begin{matrix} T_R \end{matrix})$

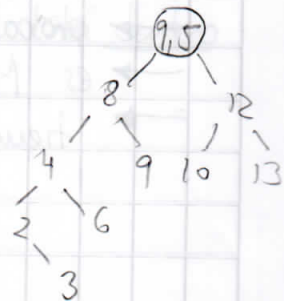
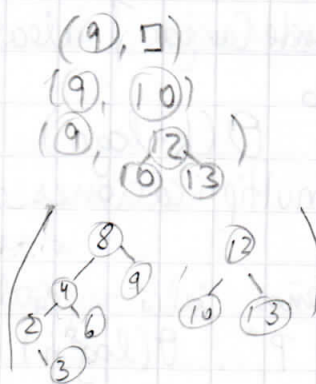


$(T_m, T_M) \leftarrow Cut(T_R, k)$

return $(\begin{matrix} k \\ T_L \end{matrix} \begin{matrix} T_R \end{matrix}, T_M)$



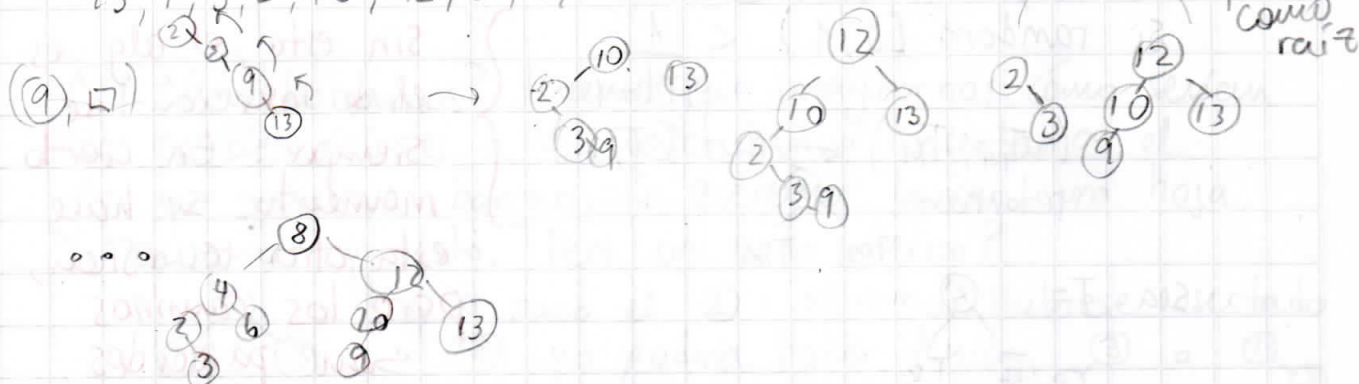
$Cut(, 9.5)$



Es como si 9.5 se hubiese ~~con~~ insertado al comienzo

Insertemos con método de cut.

13, 9, 3, 2, 10, 12, 6, 4, 8



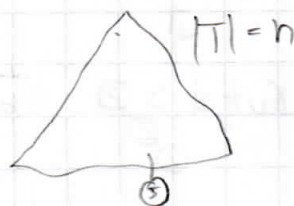
Es el mismo árbol que se obtiene al insertar de forma normal 8, 4, 6, 12, 10, 2, 3, 9, 13. !!

Ahora supongamos que todas las secuencias posibles tienen dist uniforme de aparecer.

123

Prob (nueva clave sea la primera en insertarse)

$$= \frac{1}{n+1}$$



321 y ahora insertan 4.

En todas las secuencias de 4 números, ¿cuál es la prob. de que el 4 sea la primera en insertarse, de todas las secuencias?

bajo algún algoritmo que produce árboles \neq s dependiendo del orden de inserción.

En nuestro caso ya NO depende el árbol. Estamos simulando todos los posibles casos en que la nueva clave podría haberse insertado primero según algún algoritmo que depende del orden.

Para mismo input prob. de árboles distintos

Insertar (T, k)

T = □, retornar (k)

si random [0, 1) < $\frac{1}{|T|+1}$

(T_M, T_M) ← Cut (T, k)

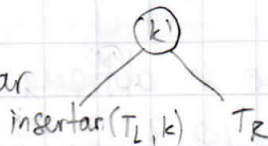
retornar



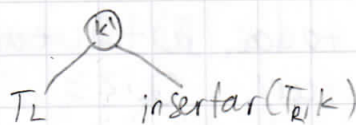
Sea T =



si k < k' retornar



else retornar



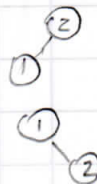
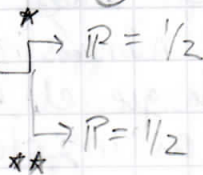
Sin esto, el alg. es el de inserción de siempre. En cierto momento, se hace esta otra recursión, pero los caminos son parecidos, de igual costo.

Ej:

Para 1 2 3 ...

insertar(1)

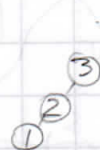
insertar(2)



esos 2 árboles salen con prob. $\frac{1}{2}$ cada 1.

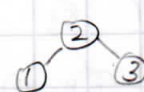
insertar(3)

→ P = 1/3 3 es raíz → hago Cut



$\frac{1}{6}$

→ P = 2/3 3 no es raíz →



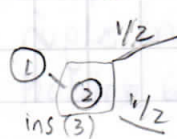
$\frac{1}{3}$

** → P = 1/3 →

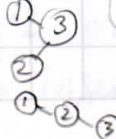


$\frac{1}{6}$

→ P = 2/3



$\frac{1}{6}$

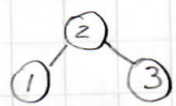




$\frac{1}{6}$

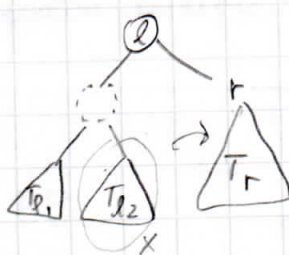
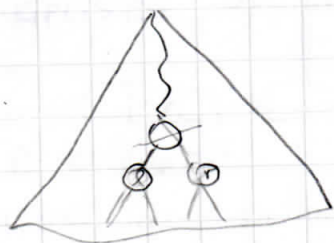
$\frac{1}{6}$

Si inserto 3 2 1 quizás el orden de casos que pasan es distinto pero obtengo la misma distribución


¿Y cómo borramos? tenemos que imaginarnos cómo serían los posibles árboles a los cuales nunca insertamos el elemento que queremos borrar. Si queremos borrar una hoja, la borramos y listo. Pero un nodo interno?


 → si saco el ②: si nunca hubiese insertado ② yo podría haber tenido  o 

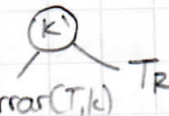
No tenemos suficiente información !! quién se primero?
→ simulamos con random




Borrar(T, k)

Sea $T =$ 

(borrar busca, eliminar "parte" con nodo vacío y retorna árbol donde puso algo en ese nodo vacío)

si $k < k'$ retornar 

Si $k > k'$ retornar 

si $|T| = 1$ retornar \square
retornar Eliminar(T_L, T_R)

