ALGORITMOS PROBABILISTAS

NUMÉRICOS

SHERWOOD

LAS VEGAS

MONTE CARLO

DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

INTRODUCCIÓN

- **Algoritmo probabilista**: Deja al azar la toma de algunas decisiones.
  - Cuando la decisión óptima llevaría mucho tiempo.
  - Problemas con múltiples soluciones correctas.
- **Ejemplos:**
  - Encontrar el k-ésimo menor elemento de un vector de *n* elementos
  - Problema de las ocho reinas.
  - Encontrar un factor de un número compuesto

ALGORITMOS PROBABILISTAS

- Supondremos un **generador de números aleatorios** de coste unitario
  - $a \leq$ uniforme $(a,b) < b$ con a, b de R
  - $i \leq$ uniforme (i..j) $\leq j$ con i, j de Z
  - uniforme (x) de X conjunto finito no vacío

- En la práctica generadores **seudoaleatorios** a partir de una semilla.

Dis
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

CLASIFICACIÓN DE ALGORITMOS PROBABILISTAS

- **Numéricos:** Solución aproximada a problemas numéricos para los que es imposible dar una respuesta exacta. Su precisión es mayor cuanto más tiempo se le dedique al algoritmo.
- **Monte Carlo:** Dan una respuesta concreta pero ésta no tiene por qué ser correcta.
- **Las Vegas:** Su respuesta es siempre correcta pero puede no encontrarla.
- **Sherwood:** Dan siempre respuesta y ésta es correcta.

DIS
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

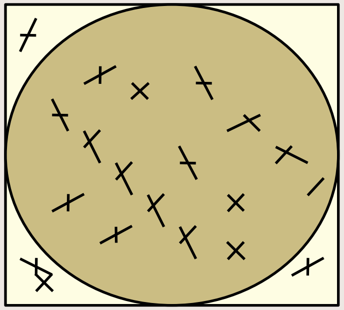
ALGORTIMOS PROBABILISTAS

NUMÉRICOS

Dis
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

ALGORTIMOS PROBABILISTAS NUMÉRICOS

- Estimación aproximada de Π:



- Tiramos n dardos sobre cuadrado y contamos el número k de los que caen en un círculo inscrito en el cuadrado.
- ¿Cuál es la proporción media de dardos en el interior del círculo?:

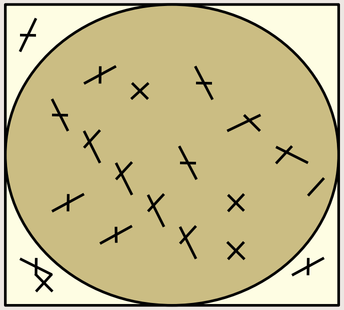
$\Pi r^2/4r^2 = \Pi/4$

- ¿Cómo se puede estimar Π?: $\Pi \cong 4k/n$

DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

ALGORTIMOS PROBABILISTAS NUMÉRICOS

Estimación de π



Función dardos(*n*)

$k \leftarrow 0$

para $i \leftarrow 1$ hasta n hacer

$x \leftarrow \mathit{uniforme}(0,1)$

$y \leftarrow \mathit{uniforme}(0,1)$

si $x^2 + y^2 <= 1$ **entonces** $k \leftarrow k+1$

devolver $4k / n$

- **¿Qué valor se estimaría si se hiciera?**

  $x \leftarrow \mathit{uniforme}(0,1)$

  $y \leftarrow x$

DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS

INTEGRACIÓN NUMÉRICA

- ***f***: **[0,1]→ [0,1]** es una función continua entonces el área de la superficie delimitada por la curva $y = f(x)$, por el eje x, por eje y, y por la derecha x=1 viene dada por:

$$\int_0^1 f(x)dx$$

- **función** curva(n)

  $k \leftarrow 0$

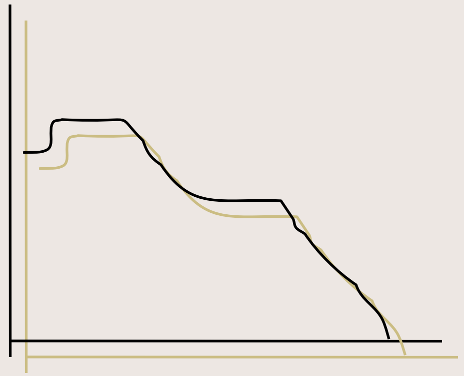
  para $i \leftarrow 1$ **hasta** n **hacer**

  $x \leftarrow$ *uniforme*(0,1)

  $y \leftarrow$ *uniforme*(0,1)

  si $y <= f(x)$ **entonces** $k \leftarrow k+1$

  devolver k/n



- Estimar **π** es equivalente a evaluar

$$4\int_0^1 (1-x^2)^{1/2}\,dx$$

DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS

INTEGRACIÓN NUMÉRICA

- **función curva(f,n,a,b)**

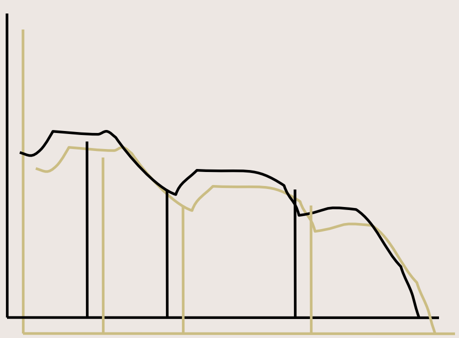
  suma $\leftarrow$ 0

  para *i* $\leftarrow$1 **hasta** *n* **hacer**

  x $\leftarrow$*uniforme*(*a*,*b*)

  suma $\leftarrow$ *suma* + *f*(*x*)

  devolver (*b*-*a*) $\times$ (*suma*/*n*)



- **función trapecio(f,n,a,b)**

  {se supone n>=2}

  delta $\leftarrow$ (*b* -*a*)/(*n*-1)

  suma $\leftarrow$ (f(a) + f(b))/2

  para *x*$\leftarrow$ *a* + *delta* **paso** *delta* **hasta** *b* - *delta* **hacer**

  suma $\leftarrow$*suma* + *f*(*x*)

  devolver suma $\times$ delta

DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

CONTEO PROBABILISTA

- **Algoritmos probabilistas** para estimar un valor entero. Ej: Calcular la cardinalidad de un conjunto *X* finito.

- **Sea *X* un conjunto** de *n* elementos en el cual muestreamos con repetición de manera uniforme e independiente. La esperanza matemática del numero de muestras antes de la primera repetición, cuando *n* es grande tiende a $\boldsymbol{k = \beta \sqrt{n}}$ **siendo** $\boldsymbol{\beta = \sqrt{\pi/2} \approx 1{,}253}$**.**

DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

CONTEO PROBABILISTA

•**Función** *contar*(X: *conjunto*)

$k \leftarrow 0$

$S \leftarrow 0$

$a \leftarrow uniforme(X)$

repetir

$k \leftarrow k+1$

$S \leftarrow S \cup \{a\}$

$a \leftarrow uniforme(X)$

hasta $a \in S$

devolver $2k^2/\pi$

- **Tiempo y espacio de orden $\sqrt{n}$** , si las operaciones sobre el conjunto son unitarias. Este espacio puede ser prohibitivo si n es grande.

Dis
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

ALGORITMOS PROBABILISTAS

Algoritmos de Sherwood

Todos
somos
iguales....

Dis
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

ALGORITMOS DE SHERWOOD

- **Sherwood:** Dan siempre respuesta y es correcta. Hace uso del azar para eliminar la diferencia entre buenos y malos ejemplares que se da en algoritmos deterministas. El caso peor depende del azar, no del ejemplar del problema.

- Análisis en media de un **algoritmo determinista(A)**, si la probabilidad de cada entrada es la misma sería:

$t_p(n) = 1/ \#X_n \sum t(x)$ (X_n conjunto de ejemplares de tamaño n)

si no

$t_p(n) = \sum p(x)\, t(x)$

puede haber un ejemplar x tal que t(x) sea mucho mayor que $t_p(n)$

DIS
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

ALGORITMOS DE SHERWOOD

- Ejemplo **Quicksort.**

- Podemos crear un **algoritmo probabilista** de forma que:

  $t_B(x) \approx t_{pA}(n) + s(n)$

  para todo ejemplar x, donde $\mathbf{t_B(x)}$ es la esperanza matemática del tiempo requerido por el algoritmo **B** sobre el ejemplar **x** y **s(n)** es el coste de uniformizar. Puede haber ejecuciones en las que el tiempo sea peor, pero no depende de la entrada.

- $\mathbf{t_{pB}(n) = 1/X_n \sum t_B(x) \approx t_{pA}(n) + s(n)}$

DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

ENCONTRAR EL K-ÉSIMO MENOR ELEMENTO

función *seleccionar* (*T*[1..*n*],*k*)

si *n* es pequeño **estonces**

ordenar *T* en orden creciente

devolver *T*[*k*]

si no p← un elemento de *T*[1..*n*]

u ← #{ $i \in [1..n]$ $T[i] < p$ }

v ← #{$i \in [1..n]$ $T[i] <= p$ }

si k < =u **entonces**

vector U[1..u]

U← los elementos de *T* menores que p

devolver *seleccionar*(U,k)

si no si k> v **entonces**

vector V[1..n-v]

V ←los elementos de T mayores que p

devolver *seleccionar* (V,k-v)

si no

devolver p

Dis
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

ENCONTRAR EL K-ÉSIMO MENOR ELEMENTO

- ¿Cuál es el mejor pivote?
- El mejor pivote sería la mediana. Si pudiéramos obtenerla con un coste constante, tendríamos un algoritmo que ejecuta una llamada recursiva y los vectores U y V tendrían como mucho $\lfloor n/2 \rfloor$.
- Suponiendo un cálculo mágico de la mediana encuentra **el k-ésimo menor elemento** en un **tiempo lineal** $t_m(n) \in O(n) + \max\{t_m(i) \mid i <= n/2\}$.
- ¿Pero, como calculamos la mediana?

DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS

ENCONTRAR EL K-ÉSIMO MENOR ELEMENTO

- Si la mediana no la podemos obtener en un tiempo constante, podemos sacrificar la eficiencia en el peor caso a cambio de una buena eficiencia en media y **escoger simplemente p=T[1].** Si hacemos esto tenemos un tiempo medio lineal, pero un **caso peor cuadrático.**

- Puedo buscar **una aproximación a la mediana**, mediante *pseudomediana,* que divide el vector en subvectores de 5 elementos y calcula la mediana exacta de estos subvectores, haciendo luego una aproximación a la mediana del vector inicial. Garantizamos así el tiempo lineal, pero debemos **gastar tiempo en la elección del pivote.**

ENCONTRAR EL K-ÉSIMO MENOR ELEMENTO

- La diferencia entre los casos peor y medio no está en el valor de los elementos, sino **en su orden**. Podemos expresar el tiempo en función de n y de una σ permutación de los n elementos.
- $t_p(n\ \sigma)$ tiempo empleado por el algoritmo que emplea la pseudomediana
- $t_s(n\ \sigma)$, tiempo empleado por el algoritmo simplificado
- Normalmente $t_p(n\ \sigma)$ es mayor que $t_s(n\ \sigma)$, pero puede haber una permutación desastrosa.
- ¿qué podríamos hacer para que el tiempo no dependiera de la permutación?
- **Solución: Algoritmo de Sherwood**

DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

ENCONTRAR EL K-ÉSIMO MENOR ELEMENTO

- **Funcion** *seleccionRB*(T[1..n], k)

  {calcula el k-esimo menor elemento de T}

  {se supone que $1<=k<=n$}

  $i \leftarrow 1$; $j \leftarrow n$

  mientras $i<j$ **hacer**

  $m \leftarrow T[uniforme(i..j)]$

  particionar(T,i,j,m,u,v)

  si $k<u$ **entonces** $j \leftarrow u-1$

  si no si k>v **entonces** $i \leftarrow v+1$

  si no $i,j \leftarrow k$

  devolver $T[i]$

DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

ENCONTRAR EL K-ÉSIMO MENOR ELEMENTO

- La esperanza matemática del tiempo de este algoritmo **es lineal independientemente del ejemplar.**
- Siempre es posible que una ejecución emplee un tiempo cuadratico pero la **probabilidad** de que ocurra es **menor cuanto mayor es n.**
- **Algoritmo de Sherwood** eficiente cualquiera que sea el ejemplar considerado.

Dis
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

CONTEO PROBABILISTA (BIS)

- **Problema:** Determinar el número de palabras diferentes en una cinta.

- **Solución a)** Ordenar las palabras de la cinta, y después recorrer secuencialmente la cinta para contar las palabras distintas. Esto sería del orden de $\theta\ (N\ lgN\ )$ siendo N el número total de palabras de la cinta.

- **Solución b)** Utilizar técnicas de direccionamiento disperso (Hashing), de esta forma recorreríamos la cinta una sola vez, tendríamos en media un orden $O(n)$, pero en el caso peor $\Omega\ (Nn)$.

Dis
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

CONTEO PROBABILISTA (BIS)

- ***M*** cota superior de n.
- ***U*** conjunto de secuencias consideradas palabras.
- ***m*** parámetro ligeramente superior a *log M*
- ***h*: *U*** $\rightarrow$ **{0,1}*m*** función Hashing capaz de transformar una cadena de ***U*** en una cadena binaria de longitud *m*
- **π(y,b)** con $b \in \{0,1\}$ es el *i* menor / *y*[*i*]=*b* o **k**+1 si ningún bit de *y* es igual a *b*

- **ALGORITMO**

{ inicialización }

y ← cadena de (*m* +1) bits a cero

{recorrido secuencial de la cinta }

para cada palabra ***x*** de la cinta **hacer**

  i ← π (*h*(*x*),1)

  y[*i*] ← 1

{primera estimación sobre *lg n*}

devolver π(y,0)

Dis
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

ALGORITMOS DE LAS VEGAS

El problema de las 8 reinas

Dis
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

Algoritmos de Las Vegas

- **A veces no dan la respuesta**
- **Se emplean** para resolver problemas para los que no se conoce ningún algoritmo determinista eficiente.
- Se corre el riesgo de tomar **decisiones que impidan llegar a la solución.**
- Permiten, a veces una eficiencia mayor para todos los ejemplares. **La esperanza matemática** del tiempo debe ser buena para todo ejemplar y la probabilidad de un tiempo excesivo despreciable.
- Se puede repetir el algoritmo hasta obtener una solución. La probabilidad de éxito es mayor cuanto de más tiempo se dispone

DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

Algoritmos de Las Vegas

- **Algoritmo** *LV*(*x*,**var** *y*,**var** *éxito*)
- **éxito** : cierto si solución, sino falso
- **x** : ejemplar a resolver
- **y** : solución al ejemplar x

función *obstinada*(*x*)

    repetir

        LV(*x,y,exito*)

    hasta *éxito*

    devolver *y*

Algoritmos de Las Vegas

- **p(x)** - probabilidad de éxito para la entrada x (> que 0)
- **s(x)** - esperanza matemática del tiempo si éxito
- **e(x) -** esperanza matemática del tiempo si fallo

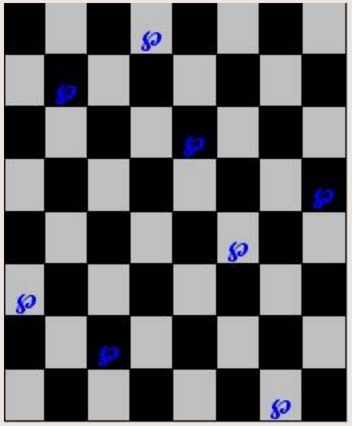
Esperanza matemática del **tiempo requerido** por *obstinado*:

$$t(x)=p(x)s(x) + (1-p(x))\,(e(x) + t(x))$$

$$t(x) = s(x) + (1-p(x)) / p(x)\;\; e(x)$$

Dis
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

Problema de las 8 reinas

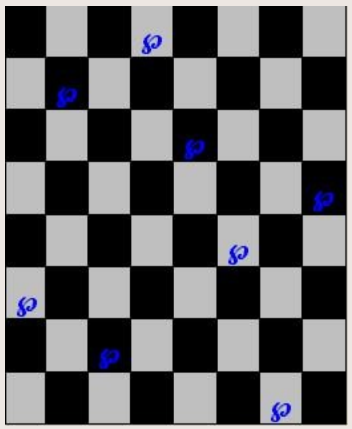


- Este problema resuelto por **backtracking** consiste en explorar sistemáticamente el árbol formado por los vectores k-prometedores. Obtenemos la primera solución después de explorar 114 de los 2057 nodos del árbol.

¿Cómo puedo aplicar un algoritmo de Vegas?

DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

Problema de las 8 reinas



•**Algoritmo voraz de las Vegas** que coloca de forma aleatoria las reinas de forma que no se ataquen mutuamente.

•Si se consiguen colocar con éxito todas las reinas el algoritmo termina con **éxito**, si no con **error.**

•La probabilidad de éxito *p* calculada con un ordenador es **12.94,** el número medio de nodos que se explora en caso de éxito *s* es **9** contando la raíz. El numero medio de nodos que se explora en caso de error es **e= 6,927**. Por tanto **la esperanza matemática del número de nodos explorados es s+(1-p)e/p=52,927.**

Dis
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

Problema de las 8 reinas

- Podemos jugar con la fórmula: Colocar aleatoriamente una cuantas reinas y después proceder por el algoritmo de vuelta atrás, sin reconsiderar la posición de las reinas colocadas de forma aleatoria.
- Si pongo más reinas al azar:

Menos tiempo, para encontrar solución o fallar

La probabilidad de fallo es mayor

Problema de las 8 reinas

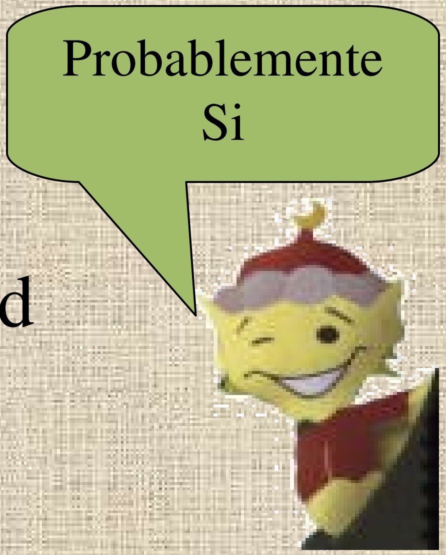
Stop Vegas	**p**	**s**	**e**	**t**
0	100,00	114,00	-----	114,001
1	100,00	39,63	-----	39,002
2	87,00	22,53	39,67	28,203
3	49,31	13,48	15,10	29,01
4	26,18	10,31	8,79	35,10
5	16,24	9,33	7,29	46,92
6	13,57	9,05	6.98	53,50
7	12,93	9,00	6,97	55,93
8	12,9	9,00	6,97	55,93

Dis
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

ALGORITMOS DE MONTE CARLO

Vector mayoritario

Comprobación de primalidad



Dis

DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

Algoritmos de Monte Carlo

- **Se emplean cuando No existe** un algoritmo eficiente determinista o de Las Vegas.
- Un **algoritmo de Monte Carlo** puede equivocarse de vez en cuando pero encuentra una solución correcta con buena probabilidad.
- En caso de error no avisa.

- **!!! NUNCA PUEDE EQUIVOCARSE SISTEMATICAMENTE SOBRE UN EJEMPLAR !!!!**

Dis
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

Algoritmos de Monte Carlo

```
funcion primo(n)
    si mcd(n,30030)=1   {alg. euclides}
        entonces devolver cierto
        si no devolver falso
```

- ¿Es un algoritmo de Monte Carlo?
- ¿Qué ocurre con la entrada n=589?

Dis
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

Algoritmos de Monte Carlo

- **¿Cuándo es útil un algoritmo de Monte Carlo?**
- Un algoritmo de Monte Carlo es ***p-correcto*** si devuelve una solución correcta con probabilidad no inferior a ***p***, con $1/2 < p < 1$.
- **Se define " La utilidad** " como p-1/2
- La probabilidad de acierto no depende del ejemplar
- Un algoritmo de Monte Carlo es **consistente** si no devuelve nunca dos soluciones correctas distintas del mismo ejemplar.

Algoritmos de Monte Carlo

- Un algoritmo de Monte Carlo es **y_0-sesgado** si existe un subconjunto ***X*** de los ejemplares y una solución conocida y_0 /
- **1)** cuando **$x \in X$** la respuesta es siempre correcta
- **2)** la respuesta correcta si **$x \notin X$**, es y_0, pero el algoritmo se puede equivocar
- Si ***MC*** es consistente y_0-sesgado y p-correcto (*y* , la respuesta del algoritmo)
  - **Si $y=y_0$**
    - si $x \in X$ por 1) la respuesta es correcta
    - si $x \notin X$ por 2) la respuesta es correcta
  - **Si $y \neq y_0$**
    - si $x \in X$ *y* es correcta
    - si $x \notin X$ el algoritmo se equivoca pues la respuesta correcta es y_0

Algoritmos de Monte Carlo

- **Si repetimos k veces** *MC*(x) con respuestas $y_1, y_2, \ldots y_k$.
  - si algún $y_i = y_0$ la solución es correcta
  - si $i \neq j$ $y_i \neq y_j$ como es consistente, $x \notin X$ y, y_0 es la solución correcta
  - si $y_i = y \neq y_0$ para todos los i es posible que la solución sea y_0 y que el algoritmo se equivoque k veces sobre $x \notin X$ pero la probabilidad es $\mathbf{(i-p)^k}$

- ¿Cómo se puede utilizar esta característica de los algoritmos de Monte Carlo?

El vector mayoritario

- Sea T[1..n], se dice que es un vector mayoritario si tiene un elemento mayoritario. Un x /{ i / T[i]=x} >n/2.

4	4	1	4	2	1	4	6	4	4	4	5	4	8

```
función  mayoritario(T[1..n])
        i← uniforme(1..n)
        x ← T[i]
        k← 0
        para j ←1 hasta n hacer
        si T[j] = x entonces k← k+1
devolver (k>n/2)
```

¿Si el vector es minoritario, existe alguna posibilidad de que el algoritmo responda lo contrario?

¿Cuál es el conjunto X?

DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

El vector mayoritario

4	4	1	4	2	1	4	6	4	4	4	5	4	8

```
función  mayoritario(T[1..n])
      i← uniforme(1..n)
      x ← T[i]
      k← 0
      para j ←1 hasta n hacer
      si T[j] = x entonces k← k+1
devolver (k>n/2)
```

¿Cuál es la Probabilidad de que siendo mayoritario, el algoritmo responda lo contrario?

El vector mayoritario

- Este algoritmo es cierto-sesgado y 1/2-correcto

función *mayoritario2*(T)

si *mayoritario*(T) **entonces devolver** cierto

si no devolver *mayoritario*(T)

- La probabilidad de que mayoritario2(T) devuelva cierto si el vector T es mayoritario es **$p+(1-p)p= 1 - (1-p)^2 > 3/4$**
- *Mayoritario2* es *cierto*-sesgado y 3/4-correcto.

Dis
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

El vector mayoritario

- Si queremos resolver el problema con una probabilidad de error inferior a ε

función *mayoritarioMC*(T, ε)

k← lg(1/ ε)

para i ← 1 **hasta** k **hacer**

si *mayoritario*(T) **entonces** **devolver** *cierto*

devolver *falso*

Dis

DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS

Comprobación de primalidad

- Tanto la generación, como la comprobación de la primalidad de números grandes, es importante en muchos problemas, por ejemplo la criptografía.
- El algoritmo determinista que se conoce es exponencial con respecto al número de dígitos
- En un problema **NP**, pero no se sabe donde esta **¿En NP-completo?** O **¿En P?**

DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

Comprobación de primalidad

- **Teorema menor de Fermat**: Sea ***n*** un numero primo. Entonces $\mathbf{a^{n-1}}$ **mod n = 1** para cualquier entero **a / 2≤ a ≤ n-1.**

n=7, a=2

$2^6 = 64$

64 mod 7 =1

n=7, a=5

$5^6 = 15625 = 2231 * 7 + 1$

5^6 mod 7 =1

- Desafortunadamente hay números compuestos que cumplen esta propiedad, por ejemplo el número compuesto **15, con a=4.**

n=15, a=4

$4^{14} = 17895697 * 15 + 1$

4^{14} mod 15 =1

4 es un falso testigo de primalidad para 15

Dis
DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS

Comprobación de primalidad

Función Fermat(n)

A := uniforme(1..n-1)

SI expomod (a,n-1,n) =1 **entonces** verdadero

SINO devolver falso

- ¿Qué podemos decir si Fermat(n) devuelve falso?
- ¿Y si devuelve verdadero?
- Los falsos testigos son escasos, pero existen números para los que la probabilidad de cometer un error es muy alta. Fermat(651693055693681) falla el 99,9965 de las veces!!!!

DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

Comprobación de primalidad

- Una modificación al teorema de Fermat disminuye drásticamente esta probabilidad
- Sea ***n*** un entero impar > 4 y ***s*** y ***t*** dos enteros positivos / ***n*-1=2**^{s}***t*** siendo ***t*** impar. Sea ***a*** un entero **2<=*a* <=*n*-2** , ***n*** es ***fuertemente pseudoprimo*** en la base ***a*** si :
  - **a^t mod n =1**
  - O, existe un entero ***i*** tal que 0<= *i*< *s*, $a^{2^i t}$ mod *n*= n-1.
- Si *n* es compuesto no puede ser fuertemente pseudoprimo en más de (n-9)/4 bases distintas.

DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS

Comprobación de primalidad

- Se puede hacer un algoritmo eficiente para comprobar si ***n*** es ***fuertemente pseudoprimo*** en la base *a*. (*Pag 385 Brassard Bratley)*

función *primo*(*n*)

a ← uniforme(2..*n*-2)

si *n* es fuertemente pseudoprimo en la base *a* **entonces devolver** *cierto*

si no devolver *falso*

- Es un algoritmo de Monte Carlo falso-sesgado y 3/4 correcto

Dis
DEPARTAMENTO DE
INFORMÁTICA Y SISTEMAS