Algoritmos y Estructuras de Datos II Clase 19

Carreras:

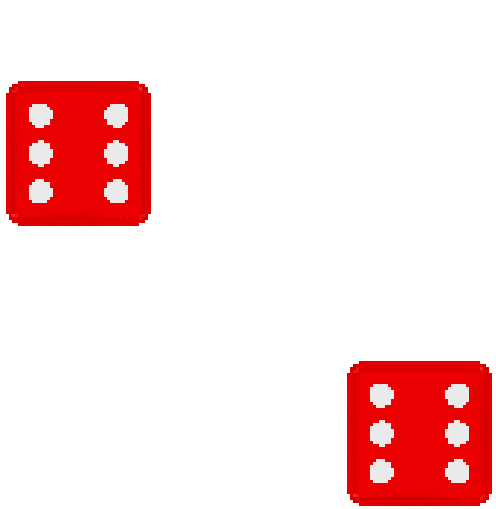
Licenciatura en Informática

Ingeniería en Informática

2024

Unidad III

Técnicas de diseño de algoritmos



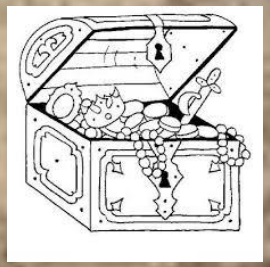
Algoritmos Probabilistas(1)

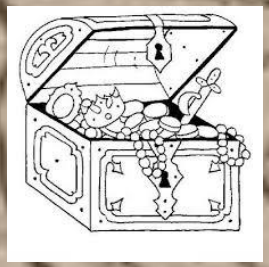
Introducción

En el libro "***Fundamentos de Algoritmia***", Brassard & Bratley (pag 365) se presenta el siguiente reto:

- Se quiere encontrar un tesoro escondido en un lugar descrito por un mapa que no se puede terminar de descifrar.

- Se ha logrado limitar la búsqueda del tesoro a dos lugares muy distantes entre sí.
- En cuanto se llega a uno de esos dos lugares se sabe de inmediato si está el tesoro o no allí.
- Hacen falta 5 días para llegar a cualquiera de los dos lugares o para ir de uno a otro.





5 días

5 días

5 días

Introducción

- Se necesitan 4 días para terminar de descifrar el mapa.

Pero existe un inconveniente…

- Hay un dragón que visita el tesoro cada noche y se lleva una parte del mismo.

- Aparece un genio se ofrece “gentilmente” a descifrar en el acto el mapa a cambio del botín del dragón de 3 noches.

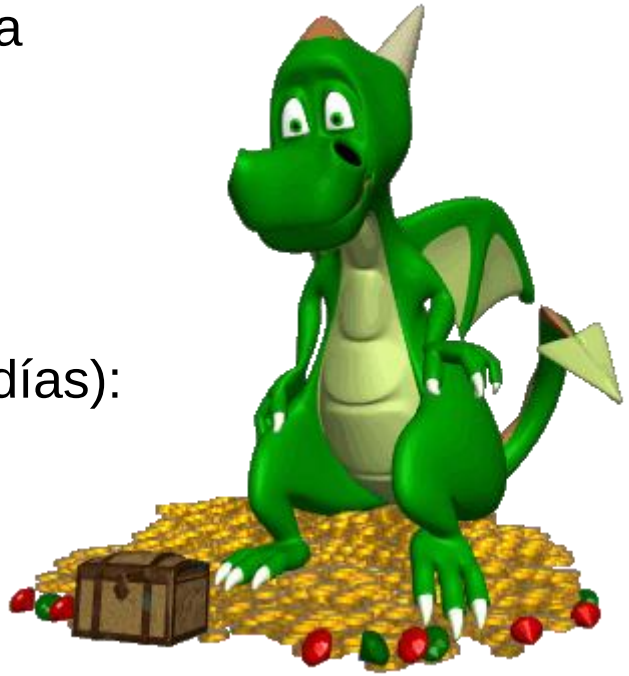
¿Conviene aceptar la oferta del genio?

Introducción

Sean:

x: Valor del tesoro que queda hoy

y: Valor del tesoro que se lleva el dragón cada día
(se puede suponer que x>9y)



Hay 3 opciones:

a) Descifrar el mapa (4 días) e ir por el tesoro (5 días):

Beneficio = x-9y

b) Pagarle 3y al genio e ir por el tesoro (5 días):

Beneficio = x-8y

c) Tirar una moneda al aire e ir por el tesoro:

Beneficio esperado = 1/2(x-5y)+1/2(x-10y) = x-7.5y

Algoritmos Probabilistas

Cuando un algoritmo tiene que hacer una elección a veces es preferible elegir al azar en vez de gastar mucho tiempo tratando de ver cual es la mejor elección.

Los ***algoritmos probabilistas*** son aquellos que basan su resultado en decisiones aleatorias, de tal forma que, en promedio se obtiene una buena solución al problema planteado, para cualquier distribución de datos de entrada.

Algoritmos Probabilistas

Característica fundamental:

Comportamiento NO DETERMINISTA

- El mismo algoritmo puede comportarse de distinta forma aplicado a los mismos datos.

- A partir de ***los mismos datos*** se puede:

  - tener distintos tiempos de ejecución,
  - obtener distintas soluciones,
  - en algunos casos se puede llegar a soluciones erróneas.

Algoritmos Probabilistas

Más diferencias con los algoritmos deterministas:

- Ventaja frente a un algoritmo determinista: si hay más de una respuesta correcta, *se pueden obtener varias respuestas diferentes* ejecutando varias veces el algoritmo probabilista, en cambio un algoritmo determinista encontrará siempre la misma respuesta.

- Un algoritmo probabilista *puede tener una situación de lazo infinito* que ocurra con baja probabilidad, lo que no está permitido en un algoritmo determinista. En caso que esto ocurra, se puede reiniciar el algoritmo con el mismo caso y esperar que este caso no se presente.

- Hasta *se puede permitir que los algoritmos probabilistas produzcan resultados incorrectos*, basta que sea con una probabilidad muy baja.

Algoritmos Probabilistas

Y más diferencias con los algoritmos deterministas:

- El *análisis de algoritmos probabilistas suele ser muy complicado* y requiere muchos conocimientos de probabilidad y estadística.

- Hay muchos algoritmos probabilistas que *dan respuestas que no necesariamente son correctas*, pero puede ser que esa respuesta incierta sea más precisa y seguramente más rápida que una respuesta obtenida en forma determinística.

- Hay problemas para los cuales no se conoce ningún algoritmo (determinístico) que pueda dar una respuesta con certeza en una cantidad razonable de tiempo y sin embargo *hay algoritmos probabilistas que pueden resolver rápidamente el problema* si se permite una pequeña probabilidad de error.

- Por ejemplo, si se quiere determinar si un número de 1000 dígitos es primo o compuesto.

Algoritmos Probabilistas

Destacando diferencias con los algoritmos deterministas:

- Si existe más de una solución para un dado conjunto de datos, un algoritmo determinista siempre encuentra la **misma solución** (a no ser que se programe para encontrar varias o todas).

- Un algoritmo probabilista ***puede encontrar soluciones diferentes*** ejecutándose varias veces con los mismos datos.

- A un algoritmo determinista no se le permite que calcule una solución incorrecta para ningún dato.

- Un ***algoritmo probabilista puede equivocarse*** siempre que esto ocurra con una probabilidad pequeña para cada dato de entrada.

Algoritmos Probabilistas

Destacando diferencias con los algoritmos deterministas:

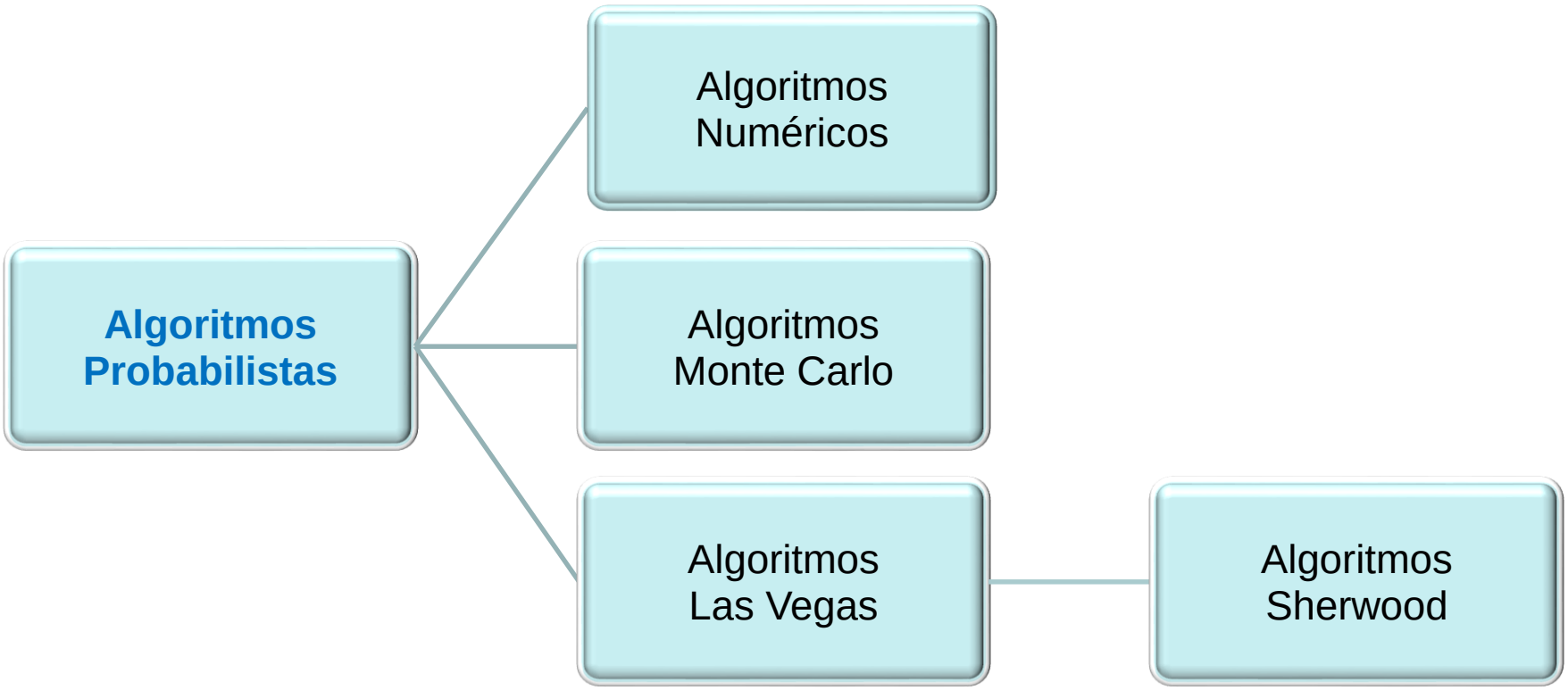
- A un algoritmo determinista ***nunca*** se le permite que no termine, hacer una división por 0, entrar en un bucle infinito, etc.

- A un algoritmo probabilista se le puede permitir siempre que eso ocurra con una ***probabilidad muy pequeña*** para datos cualesquiera. Si ocurre, se aborta el algoritmo y se repite su ejecución con los mismos datos.

- El análisis de la eficiencia de un algoritmo determinista es, en algunos casos, difícil.

- El análisis de eficiencia de los algoritmos probabilistas es, generalmente, **muy difícil.**

Clasificación de Algoritmos Probabilistas



Clasificación de Algoritmos Probabilistas

- **Algoritmos para problemas numéricos**

La respuesta es siempre aproximada (simulación, calculo numérico).

- **Algoritmos de Monte Carlo**

La respuesta es siempre exacta pero puede no ser correcta.

- **Algoritmos Las Vegas**

Nunca da una respuesta incorrecta pero puede no dar ninguna respuesta.

- **Algoritmos Sherwood**

Siempre da una respuesta y la respuesta es siempre correcta.

Algoritmos Probabilistas

Ejemplo: A la pregunta:

¿Cuándo descubrió Colón América?

Las respuestas si los algoritmos se invocan 5 veces serían:

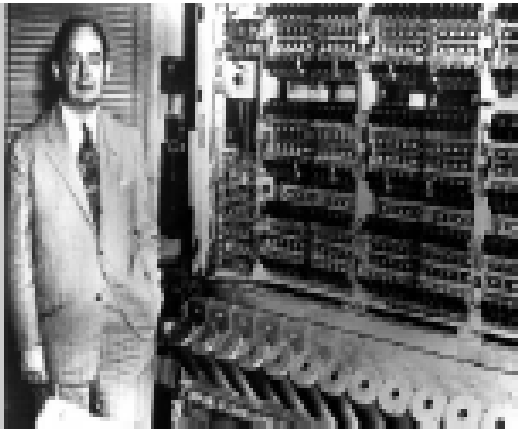
- Un algoritmo numérico:
  - Entre 1400 y 1500
  - Entre 1485 y 1499
  - Entre 1470 y 1510
  - Entre 1491 y 1499
  - Entre 1490 y 1496
- Un algoritmo de Monte Carlo:
  - 1492, 1492, 789, 1492, 1492
- Un algoritmo de Las Vegas:
  - 1492, …, 1492, 1492, 1492

Generador de números al azar

" Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin. "

– *Jon von Neumann (left)*

– *ENIAC (right)*



- Un número aleatorio es un resultado de una variable al azar según una función de distribución. Si no se especifica ninguna distribución, se presupone que se usa la distribución uniforme continua en el intervalo [0,1).

- Es fácil simular la generación de números aleatorios, mediante algoritmos de generación de números pseudoaleatorios.

- Todos los lenguajes de programación proveen un **generador de números aleatorios** (RNG por sus siglas en inglés).

Generador de números al azar

- Se supone que se tiene disponible un generador de números pseudoaleatorios de costo computacional unitario.

- Dados dos reales a y b con a < b:

uniforme (a,b) : real x real → real

devuelve un número real x elegido al azar en el intervalo $a \leq x < b$.

- La distribución de x es uniforme en el intervalo y llamadas sucesivas del generados dan valores independientes de x.

- Para generar números enteros al azar:

uniforme (i…j) : entero x entero → entero

devuelve un entero k elegido al azar uniformemente en el intervalo:

$i \leq k \leq j$

Tiempo Esperado – Tiempo Promedio

- El *tiempo promedio* de un *algoritmo determinista* se refiere al tiempo promedio requerido por el algoritmo cuando se consideran igualmente probables todas las instancias posibles de un tamaño dado.

- El *tiempo esperado* de un *algoritmo probabilista* se define para cada instancia individual, se refiere al tiempo promedio que se necesitaría para resolver ese mismo ejemplar una y otra vez. Esta es una noción robusta, que no depende de la distribución de probabilidades de las instancias que haya que resolver, porque las posibilidades implicadas quedan bajo control directo del algoritmo.

- En el caso de un algoritmo probabilista resulta significativo hablar del *tiempo esperado promedio* y del *tiempo esperado que requiere el peor caso posible* de un tamaño dado.

Algoritmos numéricos

- Se usan algoritmos numéricos desde tiempos de la segunda guerra mundial.
- Dan una ***solución aproximada.***
- Dan un intervalo de confianza. (Por ejemplo: “con probabilidad del 95% la respuesta es: 35 ± 0.2”)
- A mayor tiempo de ejecución, mejor es la aproximación.
- Son útiles cuando la solución exacta es demasiado costosa (o hasta imposible de calcular) por razones de precisión o de tiempo de ejecución.

Algoritmos numéricos

Un ejemplo:

Simulación de un sistema de espera (fila):

Estimar el tiempo medio de espera en el sistema.

En muchos casos la solución exacta no es posible.

Otro ejemplo:

La aguja de Buffon

Es el primer algoritmo probabilista de la historia.

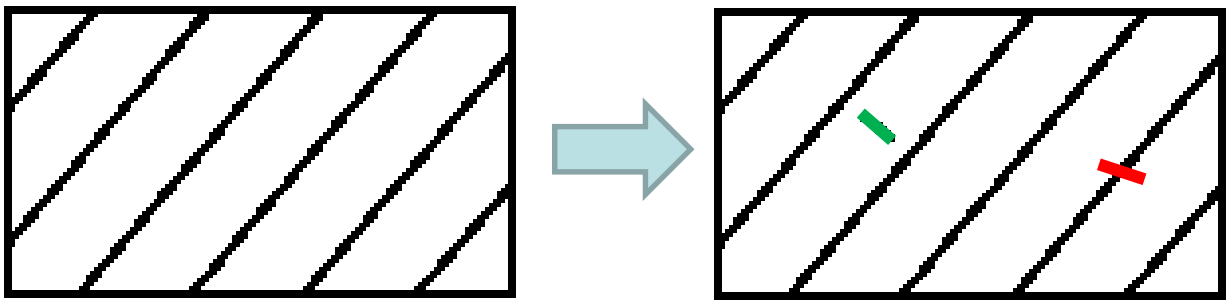
***Georges Louis Leclerc**, conde de Buffon (1707 - 1788) fue un naturalista, botánico, matemático, biólogo, cosmólogo y escritor francés. Autor: G.L. Leclerc, Conde de Buffon: "Essai d'arithmétique morale", 1777.*

Aguja de Buffon

Experimento: Suponer un piso que esta construido con listones de madera paralelos de ancho constante. Considerar que los listones están pegados entre sí y por lo tanto la separación entre los mismos es 0. Se tiene además agujas cuya longitud es la mitad de la ancho del listón

Se tira una aguja al piso, la aguja puede caer:

- Sobre un solo listón, como la aguja verde.
- Sobre una rendija y por lo tanto sobre 2 listones, como el aguja roja.

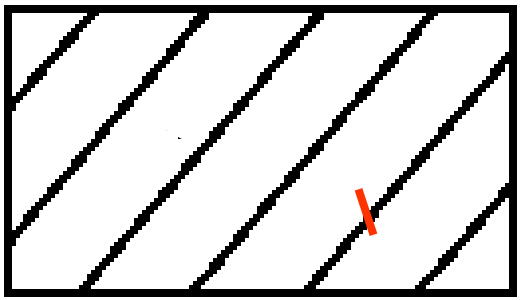


Aguja de Buffon

En el siglo XVIII, el conde de Buffon demostró que:

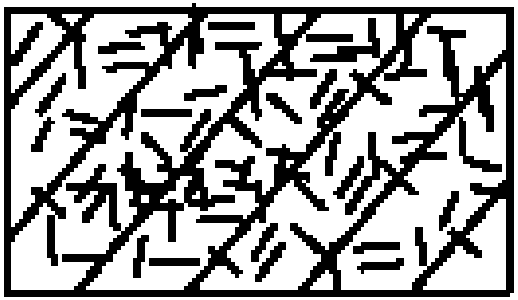
La probabilidad de que la aguja caiga encima de una rendija (quedando sobre 2 listones) es:

$$1/\pi$$



Si se dejan caer *n* agujas: sirve para predecir de forma aproximada cuantas agujas caerán entre dos planchas:

$$n/\pi$$



Aguja de Buffon

- Mas interesante algorítmicamente:

  sirve para calcular el valor de π.

- Se dejan caer n agujas y se cuentan el número de agujas k que caen entre dos listones:

  $k = n/\pi$

- Entonces se puede utilizar n/k como aproximación de π.

- Cuanto más grande sea n (más agujas se tire) más precisa será la estimación de π

Aguja de Buffon

Será útil este método?

- Teóricamente: cuanto *más agujas* de deben dejar caer, *más precisa* resulta la aproximación.

- La *convergencia es muy lenta*: para obtener un valor de $\pi \pm 0.01$ con probabilidad 0.99 se necesitan tirar 1.500.000 agujas.

(ver detalles en el libro: "***Fundamentos de Algoritmia***". Brassard & Bratley. Prentice Hall, 1998)

- Prácticamente: esta técnica para calcular el valor aproximado de π *no es útil*, usando algoritmos deterministas se pueden obtener aproximaciones de π mucho mejores.

- Una simulación:

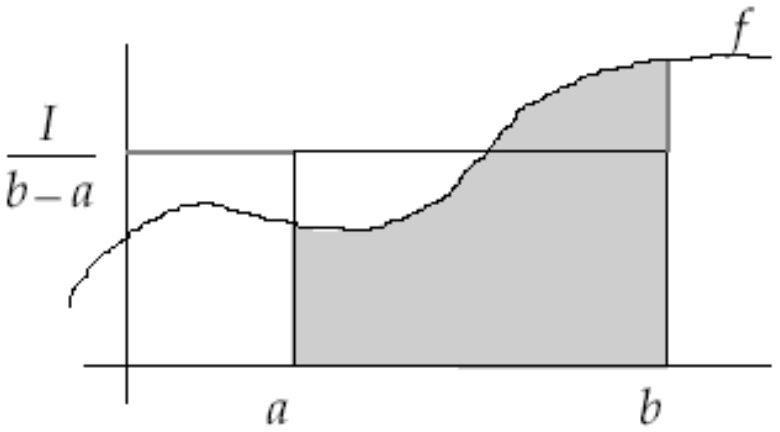
https://mste.illinois.edu/activity/buffon/#simulation

Integración numérica

Dada una función $f : R \rightarrow R^+$ continua en un intervalo $[a, b]$

- El área de la superficie limitada por la curva $y = f(x)$, el eje x y las líneas verticales: $x=a$ y $x=b$ es:

$$I = \int_a^b f(x)dx$$



Considere el rectángulo de base: $(b\text{-}a)$ y de altura: $I/(b\text{-}a)$

- a) El área del rectángulo es I
- b) El área de la superficie bajo la curva también es I

Con a) y b) se puede concluir que el rectángulo y la curva tienen la misma altura media.

Integración numérica

Diseñar de un algoritmo probabilista para calcular I

- Se puede hacer una estimación de la altura media A_m de la curva por muestreo aleatorio.
- Una aproximación de la integral será en este caso:

$$I = A_m \cdot (b - a)$$

```
Funcion intMC(f, n, a, b) : función x entero x real x real → real
suma ← 0
Para i = 1 hasta n hacer
        x ← uniforme(a,b)
        suma ← suma + f(x)
retorna (b - a) * (suma / n)
Fin
```

Integración numérica

- Puede demostrarse que la varianza del estimador calculado por la función anterior es inversamente proporcional al número *n* de muestras generadas y que la distribución del estimador es aproximadamente *normal*, cuando *n* es grande.

- Por lo tanto, el error esperado es inversamente proporcional a $\sqrt{n}$

- Esto significa 100 veces más de trabajo para obtener una cifra más de precisión

Integración numérica

- La versión determinista: es similar pero estima la altura media a partir de puntos equidistantes.
- El algoritmo determinista calcula el valor de f(x) a intervalos regulares:

```
Funcion intDET(f, n, a, b) : función x entero x real x real → real
suma ← 0
delta ← (b-a)/n
x ← a+delta/2
Para i = 1 hasta n hacer
        suma ← suma + f(x)
        x ← x + delta
retorna suma * delta
Fin
```

- Este algoritmo no funciona en todos los casos.

Integración numérica

En general, la versión determinista es *más eficiente* (menos iteraciones para obtener precisión similar).

Para todo algoritmo determinista de integración puede construirse una función que “lo vuelve loco” (no así para la versión probabilista).

Por ejemplo, para la función: $f(x) = \sin^2((100!)\pi x)$

toda llamada a intDET(f,n,0,1) con $1 \leq n \leq 100$ devuelve 0,
aunque el valor exacto es 0.5.

- Otra ventaja: cálculo de integrales múltiples.

Algoritmos deterministas: para mantener la precisión, el costo crece exponencialmente con la dimensión del espacio.

En la práctica, se usan algoritmos probabilistas para dimensión 4 o mayor.

Existen técnicas híbridas (parcialmente sistemáticas y parcialmente probabilistas): *integración cuasi-probabilista*.

Algoritmos Monte Carlo

- Dan la respuesta exacta con una alta probabilidad.
- En algunas ocasiones dan una respuesta incorrecta.
- No se puede saber si la respuesta es la correcta.
- Se reduce la probabilidad de error cuanto más se alarga el tiempo de la ejecución.
- Cuantas más veces se ejecute, más seguro se estará de la corrección de la solución.

Algoritmos Monte Carlo

- Sea p un número real tal que $0<p<1$.
- Un algoritmo Monte Carlo es ***p*–correcto** si: devuelve una solución correcta con probabilidad mayor o igual que p, cualesquiera que sean los datos de entrada.
- A veces, el valor de p dependerá del tamaño de la entrada, pero nunca de los datos de la entrada en sí.
- Esto significa que un Algoritmo Monte Carlo:
  - A veces da una solución incorrecta.
  - Con una ***alta probabilidad*** encuentra una solución correcta sea cual sea la entrada.

Informalmente se puede decir que el algoritmo funciona bien la mayoría de las veces.

Algoritmos Monte Carlo

¿Cuándo es útil un algoritmo de Monte Carlo?

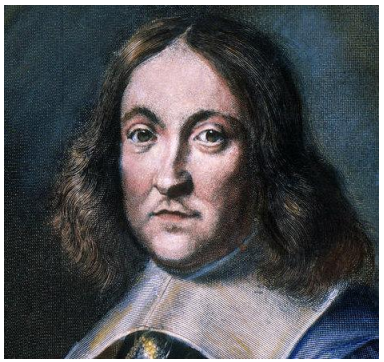
- Un algoritmo de Monte Carlo es ***p-correcto*** si devuelve una solución correcta con probabilidad no inferior a ***p;*** para que sea útil es importante que *p* cumpla con: $1/2<p<1$.
- **Se define "utilidad"** como: ***p* -1/2**
- La probabilidad de acierto no depende del ejemplar.
- Un algoritmo de Monte Carlo es **consistente** si no devuelve nunca dos soluciones correctas distintas del mismo ejemplar.

Algoritmos Monte Carlo

- El algoritmo de Monte Carlo más conocido es el de *determinar si un número es primo.*

- La comprobación de primalidad se remonta al año 1640

con los estudios de un matemático: Fermat.



Pierre de Fermat (1601-1665), fue un destacado jurista y matemático francés. Se interesó por la teoría de números y realizó varios descubrimientos en este campo.

- Este problema es especialmente útil para las técnicas criptográficas, que se basan en la factorización de números muy grandes. La seguridad de estos algoritmos radica en el problema de la factorización de números enteros en factores primos.
- No se conoce ningún algoritmo determinista que resuelva este problema en un tiempo razonable para números muy grandes (cientos de dígitos decimales).

Algoritmos Monte Carlo

- **RSA** (Rivsest, Shamir, Adleman) , desarrollado en 1977. Es el primer y más utilizado algoritmo criptográfico de clave publica para cifrar y para firmar digitalmente
- Se basa en la obtención de la clave publica mediante la multiplicación de dos números grandes que sean primos
- Los mensajes enviados se representan mediante números, y el funcionamiento se basa en el producto, conocido, de dos números primos grandes elegidos al azar y mantenidos en secreto.
- Estos primos son del orden de 10^{200} , y cada día son de mayor tamaño con el aumento de la velocidad de cálculo de las computadoras.
- La seguridad de este algoritmo radica en que no se conocen maneras rápidas de factorizar un número grande en sus factores primos utilizando computadoras tradicionales.

Determinación de números primos

Números primos del 1 al 100

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Ver a Adrian Paenza:

https://www.youtube.com/watch?v=ZxNXoPANlc0

http://www.youtube.com/watch?v=2a52rKpHGT8&NR=1

Un algoritmo (método del listillo):

https://www.youtube.com/watch?v=_jhPh7K6cNo