Generadores pseudoaleatorios

Elvira Mayordomo

Universidad de Zaragoza

30 de noviembre de 2015

Contenido de este tema

Contenido de este tema

Este tema está basado en:

- Denis R. Hirschfeldt, Rod G. Downey: Algorithmic Randomness and Complexity. Springer 2010.
- H.J. Karloff, P. Raghavan: Randomized algorithms and pseudorandom numbers. Journal of the Association for Computing Machinery 40 (1993) 454-476.
- E. Bach: Realistic analysis of some randomized algorithms. Journal of Computer and System Sciences 42 (1991) 30-53.

¿Números aleatorios?

- **Estadístico:** no deberían tener propiedades que se dan con probabilidad baja
- **Codificador:** no deberían ser comprimibles, no deberían ser fácilmente describibles
- **Jugador:** No deberían ser predecibles (por ejemplo independientes)
- ¿y para los algoritmistas? Un algoritmo no debería darse cuenta de si estás lanzando un dado o enchufándole otra cosa

¿Para qué?

- Algoritmos probabilistas
- Criptografía (depende de qué más o menos exigentes)
- Juego de azar
- Simulación

Contenido de este tema

Generadores pseudoaleatorios

- Es imposible que un ordenador genere números realmente aleatorios (es decir, uniformemente distribuidos).
- Un generador pseudoaleatorio (PRNG= "PseudoRandom Number Generator") es un algoritmo determinista que toma como entrada un número "pequeño" k de bits aleatorios (llamado la semilla) y da como salida un número "grande" m de bits (llamados bits pseudoaleatorios).
- La cadena de m bits es, por supuesto, una variable aleatoria (toma distintos valores con ciertas probabilidades), pero no está distribuida uniformemente sobre $\{0,1\}^m$, ya que toma como máximo 2^k valores con probabilidad distinta de cero.

Generadores pseudoaleatorios y algoritmos probabilistas

- Queremos utilizar el generador pseudoaleatorio para generar eficientemente bits que puedan sustituir a los números aleatorios necesarios en un algoritmo probabilista, sin degradar la utilidad del mismo. Por ejemplo un generador pseudoaleatorio con $k = 0$ y que funcionara para cualquier algoritmo probabilista sería "perfecto", pero no sabemos como implementarlo.
- Punto de partida algorítmico:
- ... En general, nos conformaremos con un generador pseudoaleatorio que sirva para un algoritmo probabilista (o para un conjunto de algoritmos).
- Un generador pseudoaleatorio G es **válido para un tipo de algoritmos** $\mathcal{A}$ si ningún algoritmo de los de $\mathcal{A}$ puede distinguir G de la aleatoriedad "perfecta".
- Por ejemplo nos resultarán suficiente los generadores válidos para $\mathcal{A} =$ "algoritmos probabilistas eficientes".

¿De dónde saco la semilla

- la definición de PRNG es robusta y quizás nos puede servir para algoritmos aleatorios
- Pero no está claro cómo conseguir la semilla realmente aleatoria

Contenido de este tema

Generadores hardware

- ¿Puede un fenómeno físico ser aleatorio? ¿Podemos capturarlo y utilizarlo como fuente de aleatoriedad para un ordenador?
- Ejemplos: ruido termal, efecto fotoeléctrico, otros fenómenos cuánticos.
- Un generador físico tendrá componentes para lectura, amplificación, paso de analógico a digital.
- Bastante más arriesgado es usar fenómenos observados: ritmo de tecleo, acceso a disco
- Estos generadores tienen el (pretencioso) nombre de TRNG (True Random Number Generator). Luego reutilizamos mejor este nombre

Problemas de los generadores físicos

- Falta de garantías
  - La mayoría se rompen de forma silenciosa, degradándose el resultado de forma paulatina
  - Por ejemplo la radiactividad de los detectores de humo, rápidamente decreciente.
  - Los fallos en estos dispositivos son muchos y la detección complicada, lenta y difícil.
  - Los fallos de diseño son frecuentes.
- Lentitud
- Desviaciones de la uniformidad

Lo mejor de los dos mundos

- Podemos combinar los generadores físicos y los algorítmicos
- Usaremos un algoritmo **extractor de aleatoriedad** para corregir los fallos de los generadores físicos
- Un extractor funciona si puedes garantizar una mínima entropía del generador físico
- se ocupa de las desviaciones y los fallos de aleatoriedad
- Para corregir la lentitud usaremos el generador físico (+ extractor) como semilla del PRNG
  - TRNG = físico + extractor
  - TRNG semilla de PRNG
  - Más o menos lo que hace intel desde 2012 (rdrand)

TRNG semilla de PRNG

- Ya tenemos de dónde sacar la semilla.
- ¿Podemos garantizar que el generador pseudoaleatorio no "estropea" un algoritmo probabilista?

Contenido de este tema

1. Introducción
2. Generadores pseudoaleatorios
3. Generadores aleatorios
4. **Generadores pseudoaleatorios y su interacción con algoritmos probabilistas: Quicksort con distintos generadores**

PRNG y algoritmos probabilistas

- Generador pseudoaleatorio: Alarga un número aleatorio dado
- Inconvenientes:
  - Necesitas un número aleatorio inicial
  - **El análisis del algoritmo probabilista hecho para números realmente aleatorios puede no ser válido**
  - ¿Cómo garantizar que el generador pseudoaleatorio no "estropea" un algoritmo probabilista?

Quicksort

- Bach empezó el estudio de PRNG en este sentido
- El estudio que veremos (PRNG para Quicksort) es de Karloff y Raghavan.
- Vamos a estudiar qué PRNG sirven para Quicksort con pivote aleatorio (es un algoritmo Las Vegas, con tiempo polinómico en media).

Quicksort con pivote aleatorio

Asumimos sobre Quicksort:

Q1 Cuando tenemos dos subproblemas que resolver (dos conjuntos que ordenar), resolvemos antes el más pequeño.

Q2 El procedimiento de ordenación es estable (es decir, respeta en lo posible el orden del problema original).

Q3 Para los conjuntos de un elemento también elegimos pivote (esto es para redondear el número de bits pseudoaleatorios usados: L pivotes para ordenar L números).

Generadores de congruencias lineales

- Un **generador de congruencias lineales** tiene tres parámetros m, a, c.
- Toma una **semilla aleatoria** $X_0 \leq m$ y da una secuencia aleatoria

$$X_0, \ldots, X_i, \ldots,$$

donde $X_i = (aX_{i-1} + c) \mod m$ para $i \geq 1$.

- Asumimos:

L1 $m > n$
L2 $\gcd(a, m) = 1$
L3 $c = 0$
L4 $\min\{t > 0 \mid a^t = 1 \pmod m\} > n/4$

- Estas hipótesis son para conseguir que el periodo del generador sea $\Omega(n)$ (deseable, ya que Quicksort consume n números pseudoaleatorios para ordenar n datos).

Generadores de congruencias lineales

- El periodo es como máximo m.
- Para que sea máximo, $\gcd(a, m) = 1$, $\gcd(c, m) = 1$.
- Pero el análisis es similar si tomamos $c = 0$, en ese caso el periodo máximo se consigue cuando $\gcd(a, m) = 1$.
- En L4, el mínimo t se puede interpretar como el periodo cuando $X_0 = 1$. El periodo es el mismo siempre que $\gcd(X_0, m) = 1$.

Generadores de congruencias lineales

- ¿Se pueden satisfacer L2, L3 y L4 a la vez?
- Con m primo sirve para muchos valores de a
- ¿Cómo usamos el PRNG para seleccionar pivotes?

  H1 $hash(y, L) = \lceil Ly/m \rceil$
  Para un conjunto de L números, utiliza el pivote que ocupa la posición $hash(y, L)$ (y pseudoaleatorio).
  Si $hash(y, L) = hash(y', L)$, entonces $|y - y'| \leq m/L$

Generadores de congruencias lineales

Teorema (Karloff, Raghavan)

Al implementar Quicksort con un generador de congruencias lineales satisfaciendo Q1-Q3, L1-L4 y H1, hay una entrada que requiere tiempo medio

$$\Omega(n^4/m^2 + n \log n)$$

(media sobre las semillas X_0).

- Consecuencia: Si $m = O(n)$, el tiempo medio es $\Omega(n^2)$.

Abierto sobre generadores de congruencias lineales

Abierto 1 Para $m = n^2$, Quicksort con un generador de congruencias lineales tarda tiempo medio $O(n \log n)$.

Abierto 2 Si la implementación de Quicksort no cumple Q2, también es cierto el teorema.

Quicksort con el generador de 5 semillas

- Hemos visto que Quicksort probabilista usa $O(n \log n)$ bits aleatorios y tarda $O(n \log n)$ en media.
- Vimos que usando un generador de congruencias lineales corriente ($O(\log n)$ bits aleatorios, la semilla) tarda $\Omega(n^4/m^2)$ en media.
- Karloff y Raghavan demostraron que Quicksort se puede implementar con $O(\log n)$ bits aleatorios en tiempo $O(n \log n)$ en media. Usan un nuevo generador, el generador de 5 semillas ("5-way generator").
- **Definición.** Un **generador de 5 semillas** tiene un parámetro p, primo. Toma una semilla aleatoria (a, b, c, d, e) y da una secuencia aleatoria

$$X_0, \ldots, X_i, \ldots,$$

donde $X_i = (a + bi + ci^2 + di^3 + ei^4) \mod p$ para $i \geq 0$.

Contenido de este tema

1. Introducción
2. Generadores pseudoaleatorios
3. Generadores aleatorios
4. Generadores pseudoaleatorios y su interacción con algoritmos probabilistas: Quicksort con distintos generadores