Estructuras de Datos y Algoritmos

Año 2022

Diseño de Funciones de Pseudoazar

Introducción

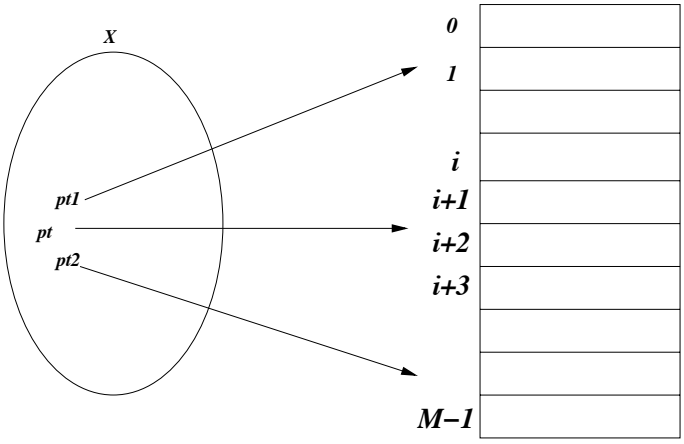
Vimos que las distintas técnicas de tratamiento de rebalse, tienen un buen comportamiento si analizamos el caso promedio y tienen complejidad $O(N)$ en el peor caso. El peor caso se da cuando la función h no distribuye adecuadamente las nuplas, por lo que el diseño de una buena función de pseudoazar es un punto importante cuando se utiliza alguna de las técnicas de tratamiento de rebalse. Así, una función de pseudo-azar ideal debería ser fácil de computar y aproximar a una función aleatoria: *para cada entrada cada salida debería ser, en algún sentido, igualmente probable*.

En una distribución pseudoaleatoria de datos, una buena función de pseudoazar debe satisfacer la propiedad de ser *un pseudoazar uniforme simple*, esto significa que todo elemento tiene igual probabilidad de caer en cualquiera de los M baldes. Más formalmente, supongamos que los elementos se toman independientemente de un conjunto X de acuerdo a una distribución de probabilidad P; es decir $P(x)$ es la probabilidad de que x sea elegido. Luego, lo que pedimos es:

$$\sum_{x \text{ tal que } h(x)=j} P(x) = \frac{1}{M},\ para\ j = 0, 1, \ldots, M-1$$

Desafortunadamente, en la mayoría de los casos no es posible verificar esta condición dado que no se conoce la distribución de probabilidad P. En la práctica se usan técnicas heurísticas para crear funciones que "probablemente" distribuyan bien.

A veces es útil en el diseño de una función pseudoaleatoria, contar con información cualitativa respecto de P. Por ejemplo: consideremos la tabla de símbolos de un compilador, en donde los elementos son secuencias arbitrarias de caracteres que representan el conjunto de identificadores de un programa. Es común que en un mismo programa se utilicen como nombres de variables *pt, pt1, pt2,* etc. Una buena función de pseudoazar debe minimizar la probabilidad de que tales identificadores colisionen; es decir, debería devolver valores con grandes diferencias para identificadores con pocas diferencias (o muy parecidos).



Así, un enfoque común es diseñar una función pseudoaleatoria que sea independiente de los patrones que existen en el conjunto de datos [4]. Es decir, cualquier función de pseudoazar provocará colisiones. Lo que se busca es evitar colisiones sistemáticas en aquellos casos donde se sabe que los elementos x exhiben un comportamiento sistemático no aleatorio.

Quizás la situación más simple sea cuando los elementos son números en punto flotante en un rango fijo. Por ejemplo, si se sabe que los elementos son mayores que 0 y menores que 1, podemos multiplicar por M y redondear al entero más cercano para obtener una dirección entre 0 y $M - 1$, como en el siguiente ejemplo, donde tomamos $M = 97$.

EJEMPLO:

x	$h(x)$
0.513870656	50
0.175725579	17
0.308633685	30
0.534531713	52
0.947630227	92
0.171727657	17
0.702230930	68
0.226416826	22
0.494766086	48
0.124698631	12
0.083895385	8
0.389629811	38
0.277230144	27

x	$h(x)$
0.368053228	36
0.983458996	95
0.535386205	52
0.765678883	74
0.646473587	63
0.767143786	74
0.780236185	76
0.822962105	80
0.151921138	15
0.625476837	61
0.314676344	31
0.346903890	34

sólo hay 3 colisiones: en 17, 52 y 74. Los dígitos más significativos de los elementos determinan los valores de la función y los menos significativos no participan para nada. Un objetivo del diseño de funciones de pseudoazar es evitar tal desbalance haciendo que cada dígito de los datos participe en el cálculo del valor de la función.

□

En general, si los elementos fuesen mayores que s y menores que t, para valores fijos s y t, podemos reescalarlos restando s y dividiendo por $t - s$, lo cual los colocaría entre 0 y 1, y luego multiplicarlos por M para calcular el valor de h.

Si los elementos fuesen valores enteros de w bits, podemos convertirlos a números en punto flotante y dividirlos por 2^w para obtener números en punto flotante entre 0 y 1, luego los multiplicamos por M como antes. Otra manera de hacerlo, si las operaciones de punto flotante son costosas y los números no son demasiado grandes como para causar overflow, sería: *multiplicar el elemento por M, luego desplazarlo a derecha w bits para dividirlo por 2^w (o si la multiplicación pudiera causar overflow, primero desplazarlo y luego multiplicarlo)*. Este tipo de funciones no se puede considerar de pseudoazar a menos que los elementos estén distribuidos uniformemente en el rango, debido a que el valor de la función sólo estará determinado por algunos de los bits de los elementos.

En nuestro ejemplo de los identificadores, el elemento pt podría ser visto como el par $(112, 116)$, dado que $p = 112$ y $t = 116$ en el conjunto de caracteres $ASCII$; luego podemos transformarlo en un

entero en base 128 de la siguiente manera : $112 \times 128 + 116 = 14452$.

En general, si x es una cadena de longitud p, podemos transformarlo a clave numérica x' de la siguiente manera:

$$x' = \sum_{i=0}^{p-1} c_i r^i$$

donde r es la base apropiada (generalmente 128) y $c_{p-1}, \ldots, c_0$ son los códigos de los correspondientes caracteres de x. Por lo tanto a continuación asumiremos, sin pérdida de generalidad, que los elementos son números naturales.

El método de la división

Este método de creación de funciones pseudoaleatorias, es sumamente sencillo y rápido, dado que sólo involucra una operación de división. Si estamos trabajando con una tabla de rebalse de M baldes, al elemento x se le asigna el balde que resulta de tomar el resto de dividir x por M; es decir, el resultado de aplicar la función módulo (mód) con M:

$$h(x) = x \mod M$$

Por ejemplo, si la tabla de rebalse tiene 16 baldes y $x = 127$, luego $h(x) = 127 \mod 16 = 15$, y al elemento $x = 127$ le corresponde el balde 15. Éste es un método simple y eficiente para enteros de w bits y posiblemente el más comúnmente usado. A tal función también se la suele llamar *modular* [4].

Podemos usar también una función modular para elementos en punto flotante. Si los elementos están en un rango muy pequeño, los podemos escalar para convertirlos en números en el rango entre 0 y 1, multiplicándolos por 2^w para obtener un entero de w bits y luego usar el módulo. Otra alternativa es sólo usar la representación binaria del elemento (si está disponible) como el operando del módulo.

Una función modular se aplica siempre y cuando tengamos acceso a los bits que componen nuestros elementos, si ellos son enteros representados en una palabra de máquina, una secuencia de caracteres empaquetados en una palabra de máquina, o algunas otras posibilidades. Una secuencia de caracteres aleatorios empaquetados en una palabra de máquina no es lo mismo que un elemento aleatorio que sea un entero, debido a que algunos de los bits son usados para propósitos de codificación, pero podemos hacer que de ambos (o de cualquier otro tipo de elemento que esté codificado en una palabra de máquina) aparezcan como índices aleatorios en una pequeña tabla.

EJEMPLO:

x	$h(x) = x \mod 97$	$h(x) = x \mod 100$	$h(x) = \lfloor (a * x) \rfloor \mod 100$ con $a = 0{,}618033$
16838	57	38	6
5758	35	58	58
10113	25	13	50
17515	55	15	24
31051	11	51	90
5627	1	27	77
23010	21	10	20
7419	47	19	85
16212	13	12	19
4086	12	86	25
2749	33	49	98
12767	60	67	90
9084	63	84	14
12060	32	60	53
32225	21	25	16
17543	83	43	42
25089	63	89	5
21183	37	83	91
25137	14	37	35
25566	25	66	0
26966	0	66	65
4978	31	78	76
20495	28	95	66
10311	29	11	72
11367	18	67	25

los valores se ven aleatorios, debido a que los elementos lo son. La función del centro $h(x) = x \mod 100$ sólo usa los dos dígitos de más a la derecha y por lo tanto es suceptible de tener un mal desempeño para elementos no aleatorios.

□

El siguiente ejemplo muestra la principal razón por la que se debe elegir el tamaño de la tabla M como un número primo para aplicar módulo. En este ejemplo, para datos de caracteres con codificación de 7-bits, tratamos al elemento como un número en base 128, un dígito para cada carácter en el elemento. La palabra `now` corresponde al número 1816567, la cual también puede verse escrita como:

$$110 \cdot 128^2 + 111 \cdot 128^1 + 119 \cdot 128^0$$

dado que el carácter ASCII de `n`, `o` y `w` son $156_8 = 110$, $157_8 = 111$ y $167_8 = 119$ respectivamente. Ahora, la elección del tamaño de tabla $M = 64$ es desafortunado para este tipo de elementos, debido a que el valor de $x \mod 64$ no se afecta por la suma de múltiplos de 64 (o 128) al x (el valor de la función de cualquier elemento es el valor de sus últimos 6 bits). Una buena función de pseudoazar seguramente

debería considerar todos los bits del elemento, particularmente en aquellos elementos de tipo alfabético. Efectos similares pueden surgir cuando M es una potencia de 2. La manera más simple de evitar tales efectos es eligiendo a M como un número primo.

EJEMPLO:

x	codif. ASCII en decimal	codif. ASCII en octal	$h(x) = x \mod 64$	$h(x) = x \mod 31$
`now`	6733767	1816567	55	29
`for`	6333762	1685490	50	20
`tip`	7232360	1914096	48	1
`ilk`	6473153	1734251	43	48
`dim`	6232355	1651949	45	21
`tag`	7230347	1913063	39	22
`jot`	6533764	1751028	52	24
`sob`	7173742	1898466	34	26
`nob`	6733742	1816546	34	8
`sky`	7172771	1897977	57	2
`hut`	6435364	1719028	52	16
`ace`	6070745	1602021	37	3
`bet`	6131364	1618676	52	11
`men`	6671356	1798894	46	26
`egg`	6271747	1668071	39	23
`few`	6331367	1684215	55	16
`jay`	6530371	1749241	57	4
`owl`	6775754	1833964	44	4
`joy`	6533771	1751033	57	29
`rap`	7130360	1880304	48	30
`gig`	6372347	1701095	39	1
`wee`	7371345	1962725	37	22
`was`	7370363	1962227	51	20
`cab`	6170342	1634530	34	24
`wad`	7370344	1962212	36	51

al considerar el tamaño de la tabla de 64 produce resultados no deseados, debido a que sólo los bits de más a la derecha del elemento contribuyen al valor de la función y los caracteres en las palabras del lenguaje natural no están uniformemente distribuidas. Por ejemplo todas las palabras terminadas en `y` producen el valor de función 57. Por el contrario, el valor primo de 31 produce menos colisiones en una tabla que poseería menos de la mitad del tamaño de la anterior.

□

Cuando usamos el método de la división debemos evitar ciertos valores para M:

- M no debe ser una potencia de dos, dado que si $M = 2^p$ entonces $h(x)$ consiste en tomar los p bits menos significativos de x. Salvo que sepamos a priori que los elementos se distribuyen de manera tal que todas las secuencias posibles de los p bits de menor orden son igualmente probables de ocurrir, la función debe diseñarse de manera tal que dependa de todos los bits de x.
- Si estamos trabajando con números decimales, M no debe ser potencia de 10, dado que la función no dependerá de todos los dígitos de x.
- Si estamos trabajando con secuencias de caracteres interpretadas en base 2^p entonces M no debe ser igual a $2^p - 1$. Si esto sucede, se puede demostrar que dos elementos que tengan los mismos caracteres pero en distinto orden producirán el mismo valor de h.

Buenos valores para M son números primos que no sean cercanos a una potencia de 2.

EJEMPLOS:

Supongamos que queremos utilizar una rebalse abierto lineal para almacenar 350 nuplas, utilizando un factor de carga $\rho = 0{,}50$ y una ranura por balde. Luego

$$M = \left\lceil \frac{N}{\rho \cdot r} \right\rceil = \left\lceil \frac{350}{0{,}5 \cdot 1} \right\rceil = 700$$

En este caso nos conviene usar $M = 701$, dado que es un número primo próximo al valor calculado, que no es cercano a una potencia de 2.

Si en cambio queremos utilizar un rebalse separado, diseñado de manera tal que una búsqueda no exitosa tenga en promedio un costo de 3 comparaciones [1], entonces:

$$M = \left\lceil \frac{N}{3} \right\rceil = \left\lceil \frac{350}{3} \right\rceil = 117$$

En este caso, el número primo más cercano es 119, pero tiene el problema de que está cerca de una potencia de 2. Luego, una elección posible sería $M = 151$.

□

El método multiplicativo

El método de la división es simple y rápido, pero no es adecuado en todas las situaciones. Por ejemplo, si estamos trabajando con un rebalse cuadrático M debería ser una potencia de 2, lo cual entra en conflicto con el método anterior.

Si x es un número real positivo, $\{x\} = x - \lfloor x \rfloor = x \mod 1$ es la parte decimal de x. Un hecho interesante es que si θ es un número irracional, entonces para n suficientemente grande las fracciones $\{\theta\}, \{2\theta\}, \{3\theta\}, \ldots, \{n\theta\}$ están distribuidas *casi* uniformemente en el intervalo $[0.,1]$. Esta propiedad se formaliza por medio del siguiente teorema:

[1] Recordar que el esfuerzo medio de localización que fracasa a priori y a posteriori, medido en baldes consultados, y bajo hipótesis de que la función h distribuye uniformemente, en un rebalse separado es ρ.

Teorema

Sea θ un número irracional. Si los puntos $\{\theta\}, \{2\theta\}, \{3\theta\}, \ldots, \{n\theta\}$ se colocan en el segmento $[0.,1]$, se forman $n+1$ segmentos que tienen a lo más tres longitudes diferentes. Además, el próximo punto $\{(n+1)\theta\}$ caerá en uno de los segmentos de mayor longitud.

Obviamente algunos valores para θ son mejores que otros; por ejemplo, si θ tiene un valor cercano a 0 o 1 se comenzará dividiendo el intervalo $[0.,1]$ en un segmento muy grande y uno muy chico. En [1], se discute la elección de θ, mostrándose que si elegimos θ como el recíproco de la razón de oro:

$$\theta = \phi^{-1} = \frac{\sqrt{5}-1}{2} \approx 0{,}61803399$$

lograremos que la distribución de las fracciones sea particularmente pareja; cada nuevo punto dividirá el intervalo más grande existente en la razón de oro (ver figura 1).

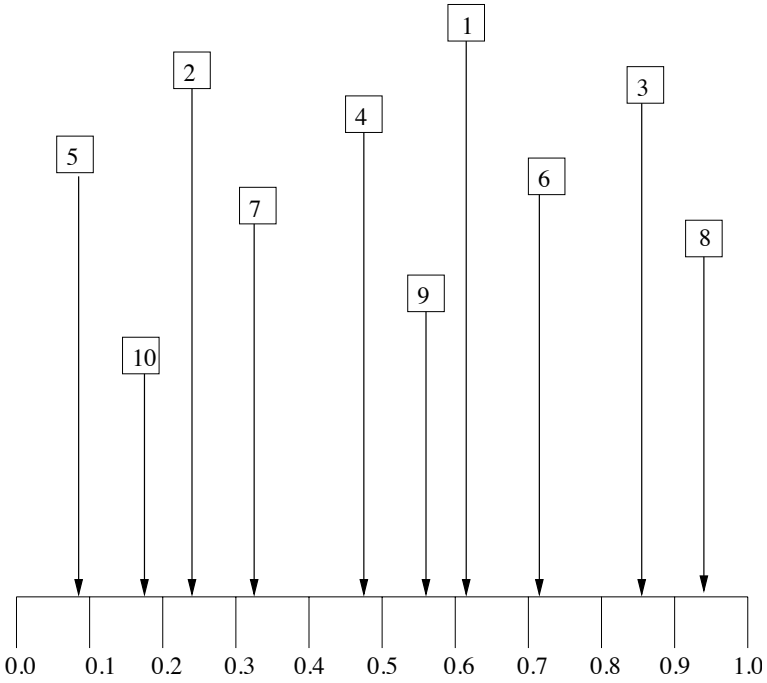


Figura 1: *Valores de $\{x\phi^{-1}\}$ para $x = 1, \ldots, 10$. Cada nuevo valor divide uno de los intervalos existentes de mayor tamaño en la razón de oro*

Habiendo elegido $\theta = \phi^{-1}$ y habiendo fijado M, definimos el método multiplicativo como :

$$h(x) = \lfloor M\{x\theta\} \rfloor$$

Desde el punto de vista matemático, este método tiene algunas propiedades interesantes. Si X es un conjunto de hileras, entonces el método probablemente distribuya los elementos uniformemente. El valor de la función h depende de todos los caracteres de la clave y sus posiciones. Aquellos elementos que están muy cercanos, por ejemplo $pt1$ y $pt2$, difieren numéricamente en un valor muy pequeño, en consecuencia el valor que les asigna h diferirá ampliamente.

EJEMPLO:

Como un ejemplo, si tenemos que $x = 123456$, $M = 10000$, y $\theta = 0{,}6180339\ldots$, entonces:

$$\begin{aligned} h(x) &= \lfloor 10000 \cdot \{123456 \cdot 0{,}6180339\ldots\}\rfloor \\ &= \lfloor 10000 \cdot \{76300{,}0041151\ldots\}\rfloor \\ &= \lfloor 10000 \cdot 0{,}0041151\ldots\rfloor \\ &= \lfloor 41{,}151\ldots\rfloor \\ &= 41 \end{aligned}$$

Se puede observar que el valor de $h(x)$ obtenido no guarda ninguna relación evidente con el valor de x, que es lo esperado de una buena función de pseudoazar.

Además, si tomamos por ejemplo $x' = 123457$, un elemento muy parecido a x, obtendríamos:

$$\begin{aligned} h(x') &= \lfloor 10000 \cdot \{123457 \cdot 0{,}6180339\ldots\}\rfloor \\ &= \lfloor 10000 \cdot \{76300{,}622149\ldots\}\rfloor \\ &= \lfloor 10000 \cdot 0{,}622149\ldots\rfloor \\ &= \lfloor 6221{,}49\ldots\rfloor \\ &= 6221 \end{aligned}$$

que es un valor muy distante al valor de $h(x)$ obtenido previamente y tampoco guarda ninguna relación evidente con el valor de x'.

□

Función de pseudoazar perfecta para conjuntos estáticos

Si conocemos el conjunto X de antemano, es posible diseñar una función que evite completamente las colisiones, es decir encontrar una función h que sea inyectiva. Una función de pseudoazar *inyectiva* h para un conjunto finito X se denomina *función perfecta*. Si además h es *sobreyectiva*, diremos que es una *función mínima perfecta* [2].

Existen varias técnicas, algunas teóricamente justificadas y algunas *ad-hoc*, que han sido desarrolladas para encontrar funciones de pseudoazar perfectas para conjuntos estáticos. Sin embargo la aplicabilidad es muy limitada, dado que en la mayoría de las aplicaciones el conjunto X no es estático.

Parecería que en este caso siempre sería posible encontrar una función de pseudoazar que cumpla con estos requisitos: dado que el conjunto es estático, lo almacenamos en una lista secuencial ordenada y usamos una búsqueda binaria para encontrar la posición donde está insertado el elemento, y retornamos esta posición como valor de h. Obviamente esto no tiene sentido: la evaluación de la función h ya sería $O(\log N)$, y si estamos calculando h para buscar un elemento ¿qué sentido tendría ir a buscarlo en otra estructura si ya lo encontramos (o no) en la lista secuencial ordenada?

[2] Notar que en este caso $|X| = M$; es decir, $h : X \longmapsto \mathbb{I}_{|X|}$ sería una función de enumeración para X, con la diferencia que la función de enumeración tiene necesariamente una función inversa.

Referencias

[1] Donald E. Knuth. *The Art of Computer Programming, Vol 3*.
Addison–Wesley; ISBN 0-201-03803-X

[2] Harry R. Lewis, Larry Denenberg. *Data Structures & Their Algorithms*.
HarperCollin Publishers; ISBN 0-673-39736-X

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. *Introduction to Algorithms*.
The MIT Press; ISBN 0-262-03141-8 y 0-262-53091-0

[4] Robert Sedgewick *Algorithms in C, Third Edition*.
Addison–Wesley; ISBN 0-201-31452-5