Función de hash

Queríamos convertir la clave $k \in D$ en un valor numérico

Para eso definimos una **función de hash** como sigue:

$$h : D \rightarrow \mathbb{N}_0$$

¿Pero como realmente calculamos h? Depende del dominio…

¿Cuáles son los posibles dominios que podríamos tener?


HASH
FUNCTION
LITERALLY ANYTHING
IS THIS A VALID DOMAIN?
imgflip.com

Primero lo primero

¿Qué propiedades nos interesan para la función de Hash?

Propiedades deseables

Idealmente la función de hash debería:

- Ser rápida de calcular
- Distribuir uniforme
- Tener pocas colisiones
- Ser compacta
- Usar toda y solo la información relevante del dato

Hasheando números

Digamos que nuestro dominio son RUTs de los alumnos de la universidad

Tratemos de usar el código verificador como hash

¿Cómo se calcula?

Hasheando números

18.936.676-?

$6 \times 2 = 12, 7 \times 3 = 21, 6 \times 4 = 24, 6 \times 5 = 30$

$3 \times 6 = 18, 9 \times 7 = 63, 8 \times 2 = 16, 1 \times 3 = 3$

Sumamos y nos da 187, luego calculamos el módulo 11

187 % 11 = 0

Y lo restamos a 11

11 – 0 = 11

Finalmente si el resultado fue 10, lo cambiamos por k y si fue 11 lo cambiamos por 0

Hasheando números

Asumamos que el valor del hash es el obtenido antes de hacer el modulo

¿Qué propiedades de las mencionadas cumple el hash?

Hasheando números

- ✓ Ser rápida de calcular
- Distribuir uniforme
- Tener pocas colisiones
- ✓ Ser compacta
- ✓ Usar toda y solo la información relevante del dato

Hasheando números

Aprovechemos la información que tenemos del dominio para una mejor función

¿Qué sabemos del dominio?

Hasheando números

Los primeros 2 dígitos tienen mucha relación con la edad del alumno

El resto de los dígitos son prácticamente aleatorios

Usemos simplemente los dígitos debajo de los millones como función de hash

Hasheando números

- ✓ Ser rápida de calcular
- ✓ Distribuir uniforme
- ✓ Tener pocas colisiones
- ✓ Ser compacta
- ✓ Usar toda y solo la información relevante del dato

Hasheando strings

Digamos que nuestro dominio ahora son secuencias de ADN, por ejemplo $AGTAGTCCGTACATCGAT$

Las secuencias son de largo arbitrario y son básicamente aleatorias

¿Cómo transformamos una secuencia a un número?

Interpretar letras como números

ASCII?

Por que no del 0 al 3 y luego lo interpretamos como un número en base 4?

Hasheando strings

Interpretamos la secuencia de largo m como un número en base b:

$$h(X) = x_1 b^{m-1} + x_2 b^{m-2} + \cdots + x_{m-1} b^1 + x_m\, b^0$$

¿Sobre qué rango caen estos datos?

Hasheando strings

✓ Ser rápida de calcular

✓ Distribuir uniforme

✓ Tener pocas colisiones

* Ser compacta

✓ Usar toda y solo la información relevante del dato

Ahora que pasa si el string tiene más posibles letras

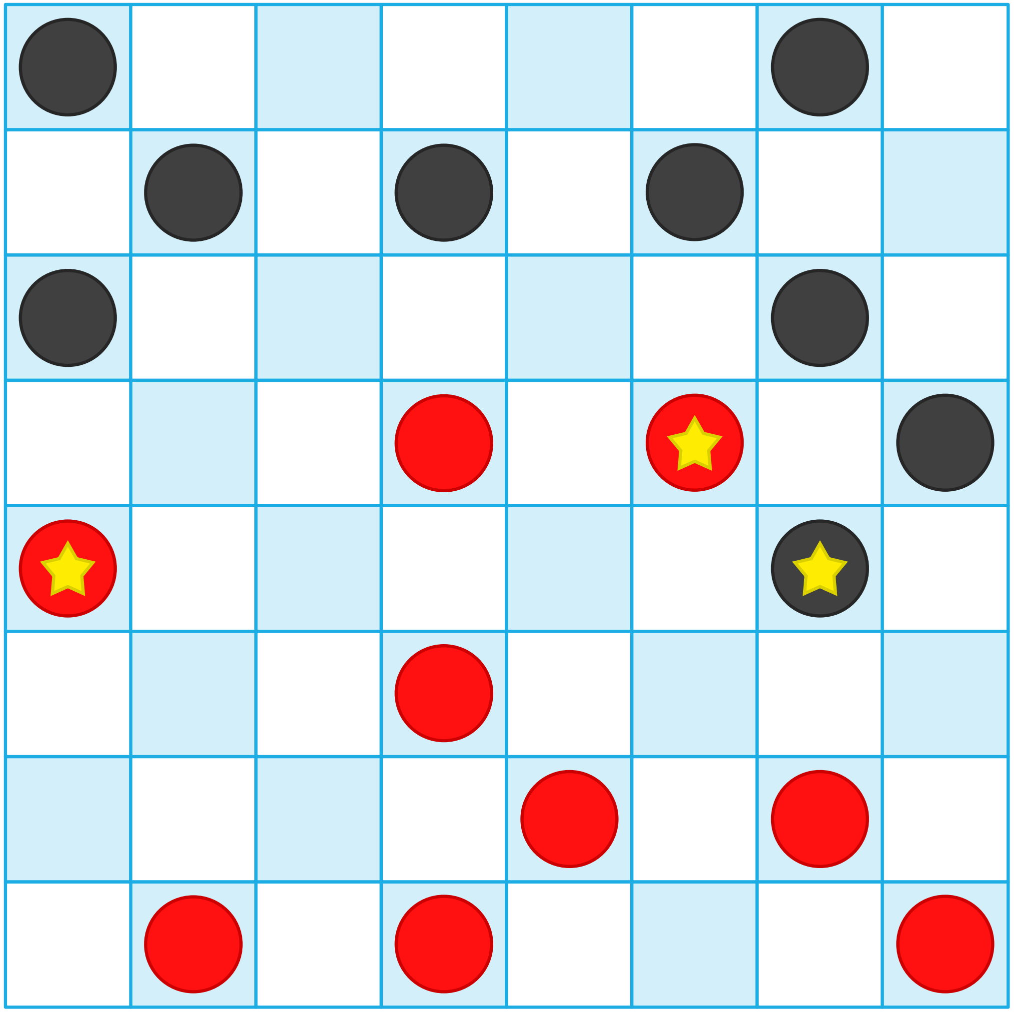
ASCII? Base 256?

En la práctica se usa base 33… por que funciona bien

(Todo calza pollo)

https://theasciicode.com.ar/

Hasheando tableros: Damas



Cada posición tiene una ficha o está vacía

Las fichas pueden ser normales o reinas

Las fichas pueden ser rojas o negras

Por lo tanto en cada posición hay 4 posibles fichas

Hasheando tableros

Nuestro dominio es gigantesco, pero nuestro recorrido tiene que caber en un *int* de 64 bits

En este tipo de problemas se hashean en el orden de cientos millones de tableros

Considerando esto, ¿qué propiedades nos interesa priorizar?

Hasheando tableros

1. Ser rápida de calcular
2. Distribuir uniforme
3. Usar toda y solo la información relevante del dato

Y se hace menos importante

4. Tener pocas colisiones (el dominio es muchísimo más grande que el recorrido por lo que no es realista no tener colisiones)
5. Ser compacta (tiene que ser muy malo nuestro hash como para no ser compacto con un dominio tan grande)

Hasheando tableros

1. Ser rápida de calcular
2. Distribuir uniforme
3. Usar toda y solo la información relevante del dato

Para conservar la primera y la tercera propiedad nuestro hash debe iterar por todo el tablero pero su complejidad no puede ser más que el tamaño del tablero

Para asegurar uniformidad podemos aprovechar números aleatorios y la función XOR

XOR

El operador XOR (eXclusive OR) es un operador lógico definido como:

p	q	$p \oplus q$	$q \oplus p$	$q \oplus p \oplus q = p$
0	0	0	0	0
0	1	1	1	0
1	0	1	1	1
1	1	0	0	1

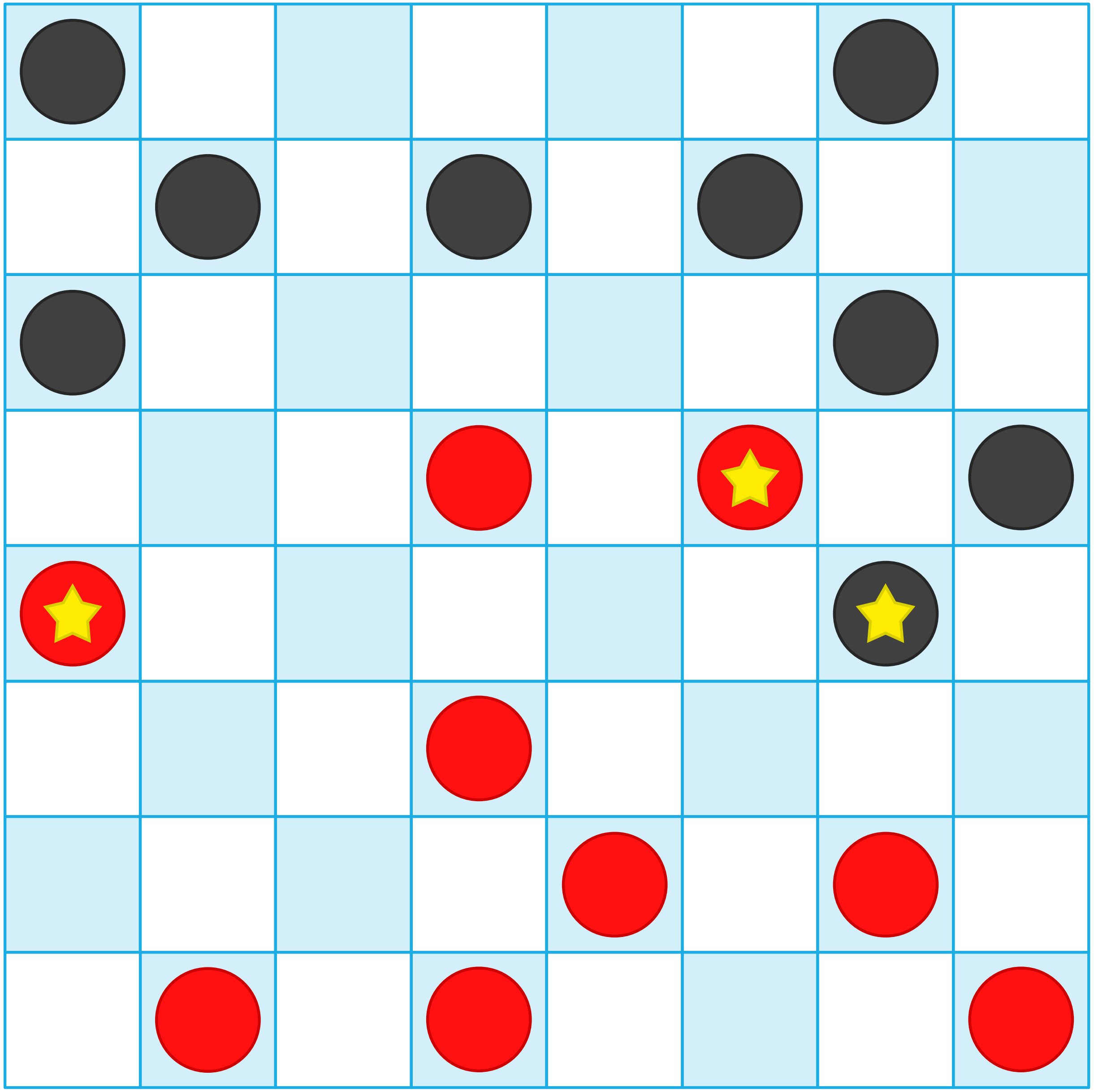
La gracia es que depende de ambos números (a diferencia del AND o el OR donde en algunos casos no hay que revisar ambos número) y es reversible

Números aleatorios y XOR

Si A es un número cualquiera, y B es un número aleatorio que distribuye **uniforme**.

Entonces $A \oplus B$ **distribuye uniforme.**

Aprovechemos esta propiedad para construir un hash



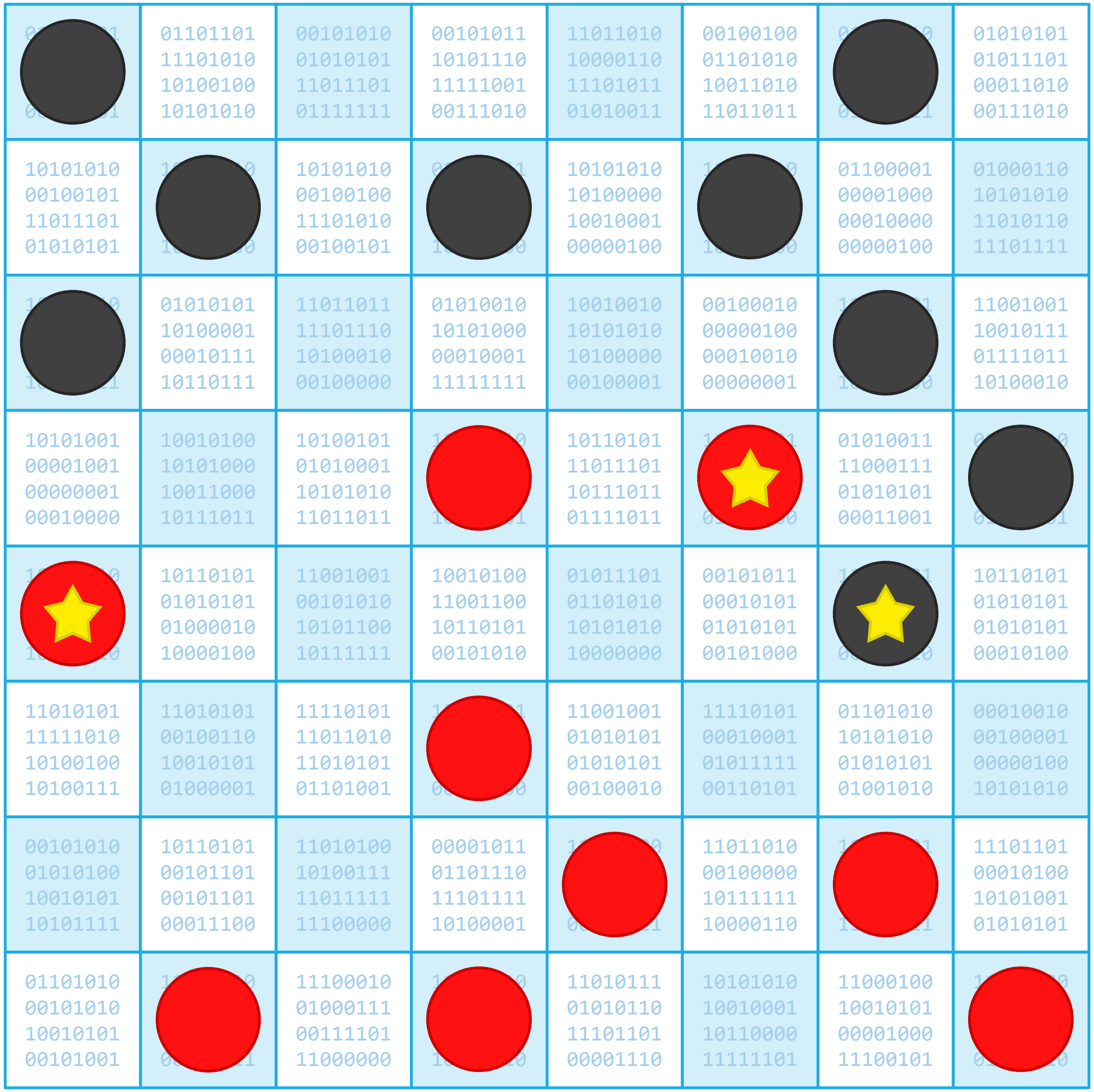
Dado un tablero nos interesa utilizar solo las casillas con piezas

Además nos interesa distinguir que tipo de pieza está en la casilla

01101101 01101010 11001010 00111001	01101101 11101010 10100100 10101010	00101010 01010101 11011101 01111111	00101011 10101110 11111001 00111010	11011010 10000110 11101011 01010011	00100100 01101010 10011010 11011011	01001010 10101010 00101010 01110111	01010101 01011101 00011010 00111010
10101010 00100101 11011101 01010101	10101110 10100010 01010101 10000100	10101010 00100100 11101010 00100101	01010011 10110110 10101010 10101000	10101010 10100000 10010001 00000100	11011010 10101010 10001000 10001000	01100001 00001000 00010000 00000100	01000110 10101010 11010110 11101111
10101010 10100101 00101010 10111111	01010101 10100001 00010111 10110111	11011011 11101110 10100010 00100000	01010010 10101000 00010001 11111111	10010010 10101010 10100000 00100001	00100010 00000100 00010010 00000001	11010101 01000010 00100000 10000000	11001001 10010111 01111011 10100010
10101001 00001001 00000001 00010000	10010100 10101000 10011000 10111011	10100101 01010001 10101010 11011011	11111010 11010101 01011010 10010101	10110101 11011101 10111011 01111011	11010101 00010010 10101010 01010100	01010011 11000111 01010101 00011001	01001010 10110101 01100101 01010101
10101010 10101010 10101010 10100110	10110101 01010101 01000010 10000100	11001001 00101010 10101100 10111111	10010100 11001100 10110101 00101010	01011101 01101010 10101010 10000000	00101011 00010101 01010101 00101000	11010101 01101110 10101001 00101010	10110101 01010101 01010101 00010100
11010101 11111010 10100100 10100111	11010101 00100110 10010101 01000001	11110101 11011010 11010101 01101001	10111001 01101010 01010010 00100100	11001001 01010101 01010101 00100010	11110101 00010001 01011111 00110101	01101010 10101010 01010101 01001010	00010010 00100001 00000100 10101010
00101010 01010100 10010101 10101111	10110101 00101101 00101101 00011100	11010100 10100111 11011111 11100000	00001011 01101110 11101111 10100001	11010000 11110111 01010010 00111111	11011010 00100000 10111111 10000110	11110101 00000011 10101111 11111111	11101101 00010100 10101001 01010101
01101010 00101010 10010101 00101001	11110010 10100010 10010101 00000011	11100010 01000111 00111101 11000000	11010010 10000101 10001010 10101010	11010111 01010110 11101101 00001110	10101010 10010001 10110000 11111101	11000100 10010101 00001000 11100101	11010100 10101010 10100101 01000010

Guardemos en el tablero distintos números aleatorios

En cada casilla guardamos 4 números distintos que corresponden a cada tipo de pieza



Al calcular el hash usaremos solo los números donde hay piezas

En cada casilla con una pieza usaremos el número correspondiente a la pieza

01101101 01101010 11001010 00111001	01101101 11101010 10100100 10101010	00101010 01010101 11011101 01111111	00101011 10101110 11111001 00111010	11011010 10000110 11101011 01010011	00100100 01101010 10011010 11011011	01001010 10101010 00101010 01110111	01010101 01011101 00011010 00111010
10101010 00100101 11011101 01010101	10101110 10100010 01010101 10000100	10101010 00100100 11101010 00100101	01010011 10110110 10101010 10101000	10101010 10100000 10010001 00000100	11011010 10101010 10001000 10001000	01100001 00001000 00010000 00000100	01000110 10101010 11010110 11101111
10101010 10100101 00101010 10111111	01010101 10100001 00010111 10110111	11011011 11101110 10100010 00100000	01010010 10101000 00010001 11111111	10010010 10101010 10100000 00100001	00100010 00000100 00010010 00000001	11010101 01000010 00100000 10000000	11001001 10010111 01111011 10100010
10101001 00001001 00000001 00010000	10010100 10101000 10011000 10111011	10100101 01010001 10101010 11011011	11111010 11010101 01011010 10010101	10110101 11011101 10111011 01111011	11010101 00010010 10101010 01010100	01010011 11000111 01010101 00011001	01001010 10110101 01100101 01010101
10101010 10101010 10101010 10100110	10110101 01010101 01000010 10000100	11001001 00101010 10101100 10111111	10010100 11001100 10110101 00101010	01011101 01101010 10101010 10000000	00101011 00010101 01010101 00101000	11010101 01101110 10101001 00101010	10110101 01010101 01010101 00010100
11010101 11111010 10100100 10100111	11010101 00100110 10010101 01000001	11110101 11011010 11010101 01101001	10111001 01101010 01010010 00100100	11001001 01010101 01010101 00100010	11110101 00010001 01011111 00110101	01101010 10101010 01010101 01001010	00010010 00100001 00000100 10101010
00101010 01010100 10010101 10101111	10110101 00101101 00101101 00011100	11010100 10100111 11011111 11100000	00001011 01101110 11101111 10100001	11010000 11110111 01010010 00111111	11011010 00100000 10111111 10000110	11110101 00000011 10101111 11111111	11101101 00010100 10101001 01010101
01101010 00101010 10010101 00101001	11110010 10100010 10010101 00000011	11100010 01000111 00111101 11000000	11010010 10000101 10001010 10101010	11010111 01010110 11101101 00001110	10101010 10010001 10110000 11111101	11000100 10010101 00001000 11100101	11010100 10101010 10100101 01000010

Finalmente calcularemos el hash del tablero haciendo XOR de todos los números

El XOR entre números se hace bit a bit

01101101 01101010 11001010 00111001	01101101 11101010 10100100 10101010	00101010 01010101 11011101 01111111	00101011 10101110 11111001 00111010	11011010 10000110 11101011 01010011	00100100 01101010 10011010 11011011	01001010 10101010 00101010 01110111	01010101 01011101 00011010 00111010
10101010 00100101 11011101 01010101	10101110 10100010 01010101 10000100	10101010 00100100 11101010 00100101	01010011 10110110 10101010 10101000	10101010 10100000 10010001 00000100	11011010 10101010 10001000 10001000	01100001 00001000 00010000 00000100	01000110 10101010 11010110 11101111
10101010 10100101 00101010 10111111	01010101 10100001 00010111 10110111	11011011 11101110 10100010 00100000	01010010 10101000 00010001 11111111	10010010 10101010 10100000 00100001	00100010 00000100 00010010 00000001	11010101 01000010 00100000 10000000	11001001 10010111 01111011 10100010
10101001 00001001 00000001 00010000	10010100 10101000 10011000 10111011	10100101 01010001 10101010 11011011	11111010 11010101 01011010 10010101	10110101 11011101 10111011 01111011	11010101 00010010 10101010 01010100	01010011 11000111 01010101 00011001	01001010 10110101 01100101 01010101
10101010 10101010 10101010 10100110	10110101 01010101 01000010 10000100	11001001 00101010 10101100 10111111	10010100 11001100 10110101 00101010	01011101 01101010 10101010 10000000	00101011 00010101 01010101 00101000	11010101 01101110 10101001 00101010	10110101 01010101 01010101 00010100
11010101 11111010 10100100 10100111	11010101 00100110 10010101 01000001	11110101 11011010 11010101 01101001	10111001 01101010 01010010 00100100	11001001 01010101 01010101 00100010	11110101 00010001 01011111 00110101	01101010 10101010 01010101 01001010	00010010 00100001 00000100 10101010
00101010 01010100 10010101 10101111	10110101 00101101 00101101 00011100	11010100 10100111 11011111 11100000	00001011 01101110 11101111 10100001	11010000 11110111 01010010 00111111	11011010 00100000 10111111 10000110	11110101 00000011 10101111 11111111	11101101 00010100 10101001 01010101
01101010 00101010 10010101 00101001	11110010 10100010 10010101 00000011	11100010 01000111 00111101 11000000	11010010 10000101 10001010 10101010	11010111 01010110 11101101 00001110	10101010 10010001 10110000 11111101	11000100 10010101 00001000 11100101	11010100 10101010 10100101 01000010

01101101
01001010
10101110
01010011
11011010
10101010
11010101
01001010
11010101
01101010
11110111
00000011
10100010
10000101
10101010
10101001
10100110
⊕ 01010100

10101000

n es el número de piezas distintas y están numeradas de 0 a $n - 1$

$\boldsymbol{zobristInit}(\boldsymbol{w}, \boldsymbol{h}, \boldsymbol{n})$:

$\boldsymbol{M}$ = matriz de dimensiones $\boldsymbol{w} \times \boldsymbol{h} \times \boldsymbol{n}$

$\boldsymbol{foreach}$ celda $\boldsymbol{x}$ $\boldsymbol{in}$ $\boldsymbol{M}$:

$\boldsymbol{x}$ = número aleatorio uniforme

$\boldsymbol{return}$ $\boldsymbol{M}$

Esta función se llama al principio del programa y se usa $\boldsymbol{M}$ para calcular el hash de los tableros

$zobristHash(M, B, w, h)$:

$v = 0$

$for\ row = 0..h-1$

$for\ col = 0..w-1$

$b = B[row][col]$

$if\ b \neq \emptyset$:

$v = v \oplus M[row][col][b.value]$

$return\ v$

Hasheando tableros

Este método se conoce como *Zobrist hashing* ya que fue inventado por Albert Linsey Zobrist

El resultado final del hash distribuye de manera uniforme entre 0 y $2^{64} - 1$ siempre y cuando los números iniciales sean uniformes